HIT
CS&E

Design and Analysis of Computer Algorithms

李建中
数据与知识工程研究中心
计算机科学与技术学院

HIT
CS&E

HIT
CS&E

Who is Who

- Instructor: 李建中
- Room: A41
- Office: 科技大厦，6007房间
- Telephone: 86415827
- Email: lijzh@hit.edu.cn
- Office hours: Wed. 4:00-6:00

HIT
CS&E

- TA: 王宏志
- Office: 科技大厦, 6011房间
- Telephone: 86415872-19
- Office hours: Wed. 4:00-6:00

HIT
CS&E

Conduct in the Classroom

1. Please try to be on-time to class.

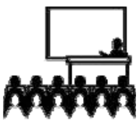
- When you come in late, you disrupt your class. As a general rule, if you are more than 10 minutes late, you should not enter the classroom.

HIT
CS&E

2. Please do not talk while in class except to raise questions.
3. You should not do things during class that disrupt the class or distract your classmates. If you have a pager or cellular phone, turn if off when you are in class.
4. Please pay attention to the signs that tell you not to eat or drink in the classrooms.

HIT
CS&E

3. Cheating will be punished.
   - You should not copy assignments, projects, or allow your work to be copied by others. Cheating in exams are forbidden. Each must do his/her own work.



- You will get zero mark for the work if caught cheating for the first time. For the second time, you will FAIL the course and be reported to the University.

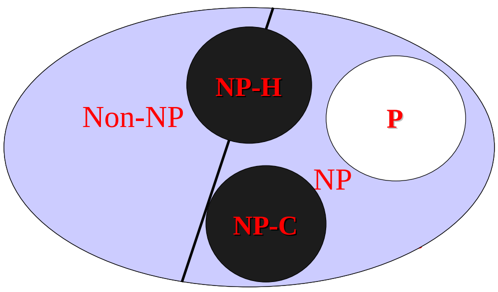
HIT
CS&E

Information about Class

- Class News
  - Will be posted by Emails
- Class Materials
  - Reading in Design and Analysis of Algorithms
- Homework
  - Must be submitted by emails on time
- Grading
  - Final Exam (Written Test): 100%
  - Homework will be considered

HIT
CS&E

What do we Learn?



HIT
CS&E

Outline of the course

Chapter 1. Introduction
- **1.1 Role of Algorithms in Computer Science**
- **1.2 Algorithms**
- **1.3 Analyzing Algorithms**
- **1.4 Designing Algorithms**

Chapter 2. Mathematical Foundations
- **2.1 Growth of Functions**
- **2.2 Recurrences**

HIT
CS&E

Chapter 3. Sorting and Loop Invariant Proof
- **3.1 Heap Sort**
- **3.2 Quick Sort**
- **3.3 Sort in Linear Time**
- **3.4 Medins and Order Statistics**

HIT
CS&E

Chapter 4. Divide-and-conquer Algorithms
- **4.1 Elements of divide-and-conquer**
- **4.2 Multiplication of Integers**
- **4.3 Multiplication of Matrices**
- **4.4 Finding the Closest Pair of Points**
- **4.5 Finding the Convex Hull**

HIT
CS&E

Chapter 5. Dynamic Programming

5.1 Elements of Dynamic Programming
5.2 Matrix-chain multiplication
5.3 Longest Common Susequence
5.4 凸多边形的三角剖分
5.5 0/1 Knapsack Problem
5.6 The Optimal binary search trees

HIT
CS&E

Chapter 6. Greedy Algorithms

6.1 Elements of Greedy Algorithms
6.2 An activity-selection problem
6.3 Huffman codes
6.4 Theoretical foundations of Greedy Algorithms
6.5 A task-scheduling problem
6.6 Minimal spanning tree problem
6.7 Single-source shortest path problem

HIT
CS&E

Chapter 7. Amortized Analysis

7.1 The aggregate method
7.2 The accounting method
7.3 The potential method
7.4 Dynamic tables

HIT
CS&E

Chapter 8. Searching Strategies

8.1 Motivation of Tree Searching
8.2 Basic Tree Searching Strategies
8.3 Optimal Tree Searching Strategies
8.4 Personnel Assignment Problem
8.5 Traveling Salesperson Problem
8.6 The A* Algorithm

HIT
CS&E

Chapter 9. Theory of NP-Completeness

9.1 NP Problems
9.2 Cook's Theorem
9.3 NP-Complete Problems
9.4 NPO Problems
9.5 Random Turing Machine

HIT
CS&E

Chapter 10. Approximation Algorithms

10.1 Introduction
10.2 The Vertex-cover Problem
10.3 The Set-covering Problem
10.4 The Traveling-salesman Problem
10.5 Randomization and Linear Programming
10.6 The Subset-sum Problem

HIT CS&E

Chapter 11. Random Algorithms

HIT CS&E

Chapter 12. On-Line Algorithms

HIT CS&E

References

1. Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms,* The MIT Press, Second Edition, 2002.
2. Vijay V. Vazirani, *Approximation Algorithms*, Springer-Verlag, 2001.
3. Motwan R. and Raghaven P., *Randomized Algorithms*, Cambridge University Press, 1995.
4. D. E. Knuth, *Art of the Computer Programming*, *Vol. 3, Addison-Wesley, 1973*.
5. A.V.Aho, J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

HIT CS&E

Important Journals

1. Journal of Algorithms
2. Acta Informatica
3. SIAM Journal on Computing
4. Journal of Computer and System Sciences
5. Journal of the ACM
6. BIT

HIT CS&E

7. Information and Control
8. ACM Computing Surveys
9. Mathematics of Computation
10. Information Processing Letters
11. Theoretical Computer Science
12. Algorithmica

HIT CS&E

Important Conferences

1. STOC: ACM Symp on Theory of Computing
2. FOCS: IEEE Symp on Foundations of Computer Science
3. COLT: Computational Learning Theory
4. LICS: IEEE Symp on Logic in Computer Science
5. SCG: ACM Symp on Computational Geometry
6. SODA: ACM/SIAM Symp on Discrete Algorithms
7. SPAA: ACM Symp on Parallel Algorithms and Architectures
8. PODC: ACM Symp on Principles of Distributed Computing
9. ISSAC: Intl. Symp on Symbolic and Algebraic Computation
10. CRYPTO: Advances in Cryptology
11. EUROCRYPT: European Conf on Cryptography

第一章
算法设计与分析overview

李建中
数据与知识工程研究中心
计算机科学与技术学院

提要

1.1 Role of Algorithms in Computer Science

- 算法是计算机科学的主题
- 计算机科学体系
- 算法设计分析的地位
- 算法设计与分析的意义

算法是计算机科学的重要主题

- **70年代前**
  - 计算机科学基础的主题没有被清楚地认清
- **70年代**
  - Knuth出版了《The Art of Computer Programming》
  - 以算法研究为主线确立了算法为计算机科学基础的重要主题
  - 1974年获得图灵奖
- **70年代后**
  - 算法作为计算机科学核心推动了计算机科学技术飞速发展

计算机科学技术的体系

- 解决一个计算问题的过程

可计算否 → 可计算理论
能行可计算否 → 计算复杂性理论
设计与分析算法 → 算法设计与分析
用计算机语言实现算法 → 数据结构与程序设计
软件系统 → 编译、OS、…

- 可计算理论
  - 计算模型
  - 可计算问题/不可计算问题
  - 计算模型的等价性--图灵/Church命题
- 计算复杂性理论
  - 在给定的计算模型下研究问题的复杂性
    - 固有复杂性
    - 复杂性下界
    - 平均复杂性
    - 复杂性问题的分类: P=NP?
    - 抽象复杂性研究

- 算法设计和分析
  - 设计算法的理论、方法和技术
  - 分析算法的理论、方法和技术
  - 具体问题的算法设计与分析
- 计算机软件
  - 系统软件
  - 工具软件
  - 应用软件

1.2 Concepts of Algorithms

- 算法的定义
- 问题的定义
- 算法的实例

算法的定义

定义1 (计算) 可由一个给定计算模型机械地执行的规则或计算步骤序列称为该计算模型的一个计算.

- 注意
  - 一个计算机程序是一个计算（计算模型是计算机）
  - 计算可能永远不停止——不是算法

定义2(算法)算法是一个满足下列条件的计算:

终止性: 有限步内必须停止,
确定性: 每步都是严格定义和确定的动作,
能行性: 每个动作都能被精确地机械执行,
输入: 具有满足给定约束条件的输入,
输出: 产生满足给定约束条件的结果.

算法的目的是求解问题。
什么是问题?

定义3 (问题) 设Input和Output是两个集合. 一个问题是一个关系$P \subseteq Input \times Output$, Input称为问题P的输入集合, Input的每个元素称为P的一个输入, Output称为问题P的输出或结果集合, Output的每个元素称为P的一个结果.

- 注意
  - 问题定义了输入和输出的关系

例. SORT问题定义如下

输入: $Input=\{<a_1, \ldots, a_n> \mid a_i\text{是整数}\}$

输出: $Output=\{<b_1, \ldots, b_n> \mid b_i\text{是整数}, b_1 \leq \ldots \leq b_n\}$

问题: $SORT=\{(<a_1, \ldots, a_n>, <b_1, \ldots, b_n>) \mid$
$<a_1, \ldots, a_n> \in Input,$
$<b_1, \ldots, b_n> \in Output,$
$\{a_1, \ldots, a_n\}=\{b_1, \ldots, b_n\}\}$

定义4(问题实例). 问题P 的一个实例是P 的一个二元组.

- 注意
  - 问题是一个二元组集合，问题实例是一个二元组.
  - 一个算法求解完整的问题，而不是仅求解一个问题的一个或几个实例.

算法示例

- 问题定义
  - Input={$<a_1,\ldots,a_n>$ | a_i是整数}
  - output={$<b_1,\ldots,b_n>$ | b_i是整数, 且$b_1 \le \ldots \le b_n$}
  - P={($<a_1,\ldots,a_n>,<b_1,\ldots,b_n>$) | $<a_1,\ldots,a_n>\in$Input, $<b_1,\ldots,b_n>\in$output, $\{a_1,\ldots,a_n\}=\{b_1,\ldots,b_n\}$}
- 算法的思想
  - 扑克牌游戏

A[1,......,n] = 5,2,4,6,1,3
A[1,......,n] = 5
A[1,......,n] = 2,5
A[1,......,n] = 2,4,5
A[1,......,n] = 2,4,5,6
A[1,......,n] = 1,2,4,5,6
A[1,......,n] = 1,2,3,4,5,6

- 算法描述
  - Insertion-sort(A)
  - Input: A[1,.....,n]=n个数
  - output: A[1,.....,n]=n个sorted数
  - FOR j=2 To n Do
  - key←A[j];
  - i←j-1
  - WHILE i>0 AND A[i]>key Do
  - A[i+1]←A[i];
  - i←i-1;
  - A[i+1]←key;

			j			
2	3	4	1	7	9	8
		i				

Quistion:
If input is n sorted number,
how many comparisons do we need to do?

1.3 Analyzing Algorithms

- 算法的正确性分析
- 算法的复杂性分析

算法的正确性分析

定义5 (算法正确性). 一个算法是正确的，如果它对于每一个输入都最终停止，而且产生正确的输出.

- 什么算法是不正确算法?
  - ①在一个或多个输入上不停止
  - ②对所有输入都停止，但对一个或多个输入产生不正确结果
- 什么近似算法/随机算法的正确性?
  - ①对所有输入都停止
  - ②产生近似正确的解/产生不多的不正确解

- 为什么要进行正确性证明?
  - 调试程序=程序正确性证明?
    程序调试只能证明程序有错!
    不能证明程序无错误!!
- 如何证明算法的正确性?
  - 证明算法对所有输入都停止
  - 证明对每个输入都产生正确结果
- 近似算法的正确性分析?
  - 证明算法对所有输入都停止
  - 分析算法的误差/获得正确解的概率

算法的复杂性分析

- 目的
  - 分析算法对不同输入所需资源量
- 复杂性测度:
  - 时间、空间、I/O等
  - 是输入大小的函数
- 用途:
  - 为求解一个问题选择最佳算法、最佳设备
- 需要的数学基础
  - 离散数学、组合数学、概率论、代数等
- 需要的数学能力
  - 建立算法复杂性的数学模型
  - 数学模型化简

定义6 (输入的大小). 设*Input*是问题*P*的输入集合，*P*的输入大小是一个函数 *F*：*Input*→N，N是正整数集合.

- 示例:
  - 排序问题的输入大小?
  - 矩阵问题的输入大小?
  - 图论问题的输入大小?

定义7 (时间复杂性). 一个算法对特定输入的时间复杂性是该算法对该输入产生结果需要的原子操作或“步”数

- 注意
  - 时间复杂性是输入大小的函数
  - 我们假设每一步的执行需要常数时间，实际上每步需要的时间量可能不同.

定义8 (空间复杂性). 一个算法对特定输入的空间复杂性是该算法对该输入产生结果所需要的存储空间的大小.

定义9 (最坏复杂性). 设*Input*是问题*P*的输入集合，*Complexity(Y)*是求解*P*的实例*X*的算法*A*的复杂性函数，*Y=Size(X)*是确定*P*的实例*X*的大小的函数，*A*的最坏复杂性是

$$Max\{Complexity(size(y)) \mid y \in Input\}$$

定义10 (最小复杂性).

$$Min\{Complexity(size(y)) \mid y \in Input\}$$

定义11 (平均复杂性). 设$y \in Input$, y作为算法*A*的输入出现的概率是p_y, *A*的平均复杂性为

$$\sum_{y \in Input} p_y \times \mathrm{complexity}(\mathrm{size}(y))$$

1.4 Designing Algorithms

- 算法的设计方法
- 算法的分析方法

算法设计方法

- Divide-and-Conquer
- Dynamic Programming
- Greedy Algorithms
- Approximation Algorithms
- Randomlized Algorithms
- Tree Searching Strategies
- On-Line Algorithms
- Genetic Algorithms
- Parellel Algorithms

算法的分析方法

- 不同的设计方法有不同的分析方法

HIT CS&E

第二章

算法分析的数学基础

李建中
计算机科学与工程系

HIT CS&E

阅读资料

《Introduction to Algorithms》
- 第三章
- 第四章

《Concrete Mathematics》

HIT CS&E

提要

HIT CS&E

2.1 计算复杂性函数的阶

HIT CS&E

2.1.1 同阶函数

定义2.1.1. 设$f(n)$和$g(n)$是正值函数。如果$\exists c_1, c_2>0$, n_0, $\forall n>n_0$, $c_1g(n)\le f(n)\le c_2g(n)$，则称$f(n)$与$g(n)$同阶，记作$f(n)=\theta(g(n))$。

$\theta(g(n))$可以视为如下集合：

$\{f(n) \mid \exists c_1, c_2>0, n_0, \forall n>n_0, c_1g(n)\le f(n)\le c_2g(n)\}$

即所有与$g(n)$同阶函数的集合

HIT CS&E

思考题

用定义证明下列结论：

1. $f(n)=an^2+bn+c=\theta(n^2)$
2. $6n^3\neq\theta(n^2)$
3. $p(n)=\sum_{i=0}^{d} a_i n^i = \theta(n^d)$

HIT CS&E

2.1.2 低阶函数集合

定义2.1.2. 设$f(n)$和$g(n)$是正值函数。如果$\exists c>0$, n_0, $\forall n>n_0$, $f(n)\le cg(n)$，则称$f(n)$比$g(n)$低阶或$g(n)$是$f(n)$的上界，记作$f(n)=O(g(n))$。

$O(g(n))$可以视为如下集合:

$\{f(n) \mid \exists c,\ n_0,\ \forall n>n_0,\ f(n)\le cg(n)\}$

称为所有比$g(n)$低阶的函数

© DB-LAB (2003)

HIT CS&E

思考题

用定义证明下列结论:

1. $n=O(n^2)$
2. 如果$f(n)=\theta(g(n))$，则$f(n)=O(g(n))$

© DB-LAB (2003)

HIT CS&E

2.1.3 高阶函数集合

定义2.1.3. 设$f(n)$和$g(n)$是正值函数。如果$\exists c>0$, n_0, $\forall n>n_0$, $f(n)\ge cg(n)$，则称$f(n)$比$g(n)$高阶或$g(n)$是$f(n)$的下界，记作$f(n)=\Omega(g(n))$。

$\Omega(g(n))$可以视为如下集合:

$\{f(n) \mid \exists c,\ n_0,\ \forall n>n_0,\ f(n)\ge cg(n)\}$

称为所有比$g(n)$高阶的函数

© DB-LAB (2003)

HIT CS&E

思考题

用定义证明下列结论:

$f(n)=\theta(g(n))$ iff $f(n)=O(g(n))$且$f(n)=\Omega(g(n))$

© DB-LAB (2003)

HIT CS&E

2.1.4 严格低阶函数集合

定义2.1.4. 设$f(n)$和$g(n)$是正值函数。如果$\forall c>0$, n_0, $\forall n>n_0$, $f(n)<cg(n)$，则称$f(n)$严格比$g(n)$低阶或$g(n)$是$f(n)$的严格上界，记作$f(n)=o(g(n))$。

$o(g(n))$可以视为如下集合:

$\{f(n) \mid \forall c,\ n_0,\ \forall n>n_0,\ f(n)<cg(n)\}$

称为所有比$g(n)$严格低阶的函数

© DB-LAB (2003)

HIT CS&E

思考题

用定义证明下列结论:

1. $2n=o(n^2)$
2. $2n^2\ne o(n^2)$?
3. $f(n)=o(g(n)) \Rightarrow \lim_{n\to\infty}\frac{f(n)}{g(n)}=0$

© DB-LAB (2003)

HIT CS&E

2.1.5 严格高阶函数集合

定义2.1.4. 设$f(n)$和$g(n)$是正值函数。如果$\forall c>0$, n_0, $\forall n>n_0$, $f(n)>cg(n)$，则称$f(n)$严格比$g(n)$高阶或$g(n)$是$f(n)$的严格下界，记作$f(n)=w(g(n))$。

$w(g(n))$可以视为如下集合：

$\{f(n) \mid \forall c,\ n_0,\ \forall n>n_0,\ f(n)>cg(n)\}$

称为所有比$g(n)$严格高阶的函数

HIT CS&E

思考题

用定义证明下列结论：

1. $f(n)=w(g(n))$ iff $g(n)=o(f(n))$
2. $n^2/2=w(n)$
3. $n^2/2\neq w(n^2)$?
4. $f(n)=w(g(n)) \Rightarrow \lim_{n\to\infty}\frac{f(n)}{g(n)}=\infty$

HIT CS&E

2.1.6 函数阶的性质

A 传递性：

(a) $f(n)=\theta(g(n)) \wedge g(n)=\theta(h(n)) \Rightarrow f(n)=\theta(h(n))$

(b) $f(n)=O(g(n)) \wedge g(n)=O(h(n)) \Rightarrow f(n)=O(h(n))$

(c) $f(n)=\Omega(g(n)) \wedge g(n)=\Omega(h(n)) \Rightarrow f(n)=\Omega(h(n))$

(d) $f(n)=o(g(n)) \wedge g(n)=o(h(n)) \Rightarrow f(n)=o(h(n))$

(e) $f(n)=w(g(n)) \wedge g(n)=w(h(n)) \Rightarrow f(n)=w(h(n))$.

HIT CS&E

2.1.6 函数阶的性质(续)

B 自反性：

(a) $f(n)=\theta(f(n))$,

(b) $f(n)=O(f(n))$,

(c) $f(n)=\Omega(f(n))$.

C 对称性

$f(n)=\theta(g(n))$ iff $g(n)=\theta(f(n))$.

D 反对称性：

$f(n)=O(g(n))$ iff $g(n)=\Omega(f(n))$

$f(n)=o(g(n))$ iff $g(n)=w(f(n))$

HIT CS&E

注意

所有函数都是可比的吗？

$f(n)=n$ 与 $g(n)=n^{1+\sin(n)}$ 可比吗？

HIT CS&E

2.2 标准符号和通用函数

- Floor 和 Ceiling
- 多项式

HIT CS&E

2.2.1 Floor和ceiling

定义2.2.1(Flours和ceiling). $\lfloor x\rfloor$表示小于或等于x的最大整数. $\lceil x\rceil$表示大于等于x的最小整数.

HIT CS&E

命题 2.2.1 $x-1<\lfloor x\rfloor\le x\le\lceil x\rceil<x+1$

命题 2.2.2 对于任意整数n, $\lceil n/2\rceil+\lfloor n/2\rfloor=n$

命题 2.2.3 设 *n*、*a*、*b* 是任意整数，$a\ne 0,b\ne 0$，则

(1) $\lceil\lceil n/a\rceil/b\rceil=\lceil n/ab\rceil$。

(2) $\lfloor\lfloor n/a\rfloor/b\rfloor=\lfloor n/ab\rfloor$

HIT CS&E

2.3 和式的估计与界限

1. 线性和

命题 2.4.5 $\sum_{k=1}^{n}(ca_k+b_k)=c\sum_{k=1}^{n}a_k+\sum_{k=1}^{n}b_k$

命题 2.4.6 $\sum_{k=1}^{n}\theta(f(k))=\theta\left(\sum_{k=1}^{n}f(k)\right)$

HIT CS&E

2. 级数

命题 2.4.7 $\sum_{i=1}^{n}i=\frac{n(n+1)}{2}=\theta(n^2)$

命题 2.4.8 $\sum_{k=0}^{n}x^k=1+x+x^2+.....+x^n=\frac{x^{n+1}-1}{x-1}\qquad(x\ne 1)$

$\sum_{k=0}^{\infty}x^k=\frac{1}{1-x}\qquad |x|<1$

命题 2.4.9 $H_n=\sum_{k=1}^{n}\frac{1}{k}=\ln n+O(1)$

HIT CS&E

命题 2.4.10 $\sum_{k=1}^{n}(a_k-a_{k-1})=a_n-a_0$.

命题 2.4.11 $\sum_{k=0}^{n-1}(a_k-a_{k+1})=a_0-a_n$

命题 2.4.12 $\sum_{k=1}^{n-1}\frac{1}{k(k+1)}=\sum_{k=1}^{n-1}\left(\frac{1}{k}-\frac{1}{k+1}\right)=1-\frac{1}{n}$

命题 2.4.13 $\lg(\prod_{k=1}^{n}a_k)=\sum_{k=1}^{n}\lg a_k$

HIT CS&E

3. 和的界限

例 1. 证明 $\sum_{k=0}^{n}3^k=O(3^n)$

证 证明对于 c≥ 3/2 存在一个 n_0，当$n\ge n_0$时$\sum_{k=0}^{n}3^k\le c3^n$.

当 *n=0* 时，$\sum_{k=0}^{n}3^k=1\le c=c3^n$.

设 *n≤m* 时成立．令 $n=m+1$，则

$\sum_{k=0}^{m+1}3^k=\sum_{k=0}^{m}3^k+3^{m+1}\le c3^m+3^{m+1}=c3^{m+1}\left(\frac{1}{3}+\frac{1}{c}\right)\le c3^{m+1}$.

HIT CS&E

例1. $\sum_{k=1}^{n} k \le \sum_{k=1}^{n} n = n^2$

例 2. $\sum_{k=1}^{n} a_i \le n \times \max\{a_k\}$.

例 3. 设对于所有 $k \ge 0$，$a_{k+1}/a_k \le r < 1$，求 $\sum_{k=0}^{n} a_k$ 的上界.

解：$a_1/a_0 \le r \Rightarrow a_1 \le a_0 r$，

$a_2/a_1 \le r \Rightarrow a_2 \le a_1 r \le a_0 r^2$，

$a_3/a_2 \le r \Rightarrow a_3 \le a_2 r \le a_0 r^3$ ……

$a_k/a_{k-1} \le r \Rightarrow a_k \le a_{k-1} r \le a_0 r^k$

于是，$\sum_{k=0}^{n} a_k \le \sum_{k=0}^{\infty} a_0 r^k = a_0 \sum_{k=0}^{\infty} r^k = \frac{a_0}{1-r}$.

HIT CS&E

例 4. 求 $\sum_{k=1}^{\infty} (k/3^k)$ 的界

解. 使用例 3 的方法. $\frac{k+1}{3^{k+1}} \Big/ \frac{k}{3^k} = \frac{1}{3} \cdot \frac{k+1}{k} = \frac{1}{3}\left(1+\frac{1}{k}\right) \le \frac{2}{3} = r$. 于是

$$\sum_{k=1}^{\infty} \frac{k}{3^k} \le \sum_{k=1}^{\infty} a_1 r^k = \sum_{k=1}^{\infty} \frac{1}{3}\left(\frac{2}{3}\right)^k = \frac{1}{3} \cdot \frac{1}{1-\frac{2}{3}} = 1 .$$

HIT CS&E

例5. 如果 $f(k)$ 单调递增，则 $\int_{m-1}^{n} f(x)dx \le \sum_{k=m}^{n} f(k) \le \int_{m}^{n+1} f(x)dx$.

例6. 当 $f(x)$ 单调递减时，$\int_{m}^{n+1} f(x)dx \le \sum_{k=m}^{n} f(k) \le \int_{m-1}^{n} f(x)dx$.

HIT CS&E

2.4 递归方程

- 递归方程：递归方程是使用具有小输入值的相同方程来描述一个方程.
- 递归方程例：Merge-sort排序算法的复杂性方程
  - $T(n)=\theta(1)$ if $n=1$
  - $T(n)=2T(n/2)+\theta(n)$ if $n>1$.
  - $T(n)$的解是$\theta(n\log n)$

HIT CS&E

求解递归方程的三个主要方法

- Substitution方法：
  - Guess first,
  - 然后用数学归纳法证明.
- Iteration方法：
  - 把方程转化为一个和式
  - 然后用估计和的方法来求解.
- Master方法：
  - 求解型为$T(n)=aT(n/b)+f(n)$的递归方程

HIT CS&E

2.4.1 Substitution方法

Substitution方法 Ⅰ：联想已知的$T(n)$

例1. 求解$2T(n/2 + 17) + n$

解：猜测：$T(n) = 2T\left(\frac{n}{2}+17\right)+n$ 与 $T(n) = 2T\left(\frac{n}{2}\right)+n$ 只相差一个 17.

当 n 充分大时 $T\left(\frac{n}{2}+17\right)$ 与 $T\left(\frac{n}{2}\right)$ 的差别并不大，因为

$\frac{n}{2}+17$ 与 $\frac{n}{2}$ 相差小. 我们可以猜 $T(n) = O(n\lg n)$.

证明：用数学归纳法

HIT CS&E

Substitution方法II：上挤下压

例 3. 求解 $T(n)=2T\left(\frac{n}{2}\right)+n$.

解：首先证明 $T(n)=\Omega(n)$，$T(n)=O(n^2)$

然后逐阶地降低上界、提高下界。

$\Omega(n)$的上一个阶是$\Omega(n\log n)$，

$O(n^2)$的下一个阶是$O(n\log n)$。

HIT CS&E

Substitution方法III：变量替换

经变量替换把递归方程变换为熟悉的方程.

例 6. 求解 $T(n)=2T(\sqrt{n})+\lg n$

解：令 $m=\lg n$，则 $n=2^m$，$T(2^m)=2T\left(2^{\frac{m}{2}}\right)+m$.

令 $S(m)=T(2^m)$ 则 $T\left(2^{\frac{m}{2}}\right)=S\left(\frac{m}{2}\right)$. 于是，$S(m)=2S\left(\frac{m}{2}\right)+m$

显然，$S(m)=O(m\lg m)$，即 $T(2^m)=\theta(m\lg m)$.

由于 $2^m=n$，$m=\lg n$，$T(n)=\theta(\lg n\times\lg(\lg n))$.

HIT CS&E

2.4.2 Iteration方法

方法：

循环地展开递归方程，

把递归方程转化为和式，

然后可使用求和技术解之。

例 1. $T(n)=n+3T(\lfloor n/4\rfloor)$，$T(1)=1$

$$=n+3(\lfloor n/4\rfloor+3T(\lfloor n/16\rfloor))$$

$$=n+3(\lfloor n/4\rfloor+3(\lfloor \tfrac{n}{16}\rfloor+3T(\lfloor n/64\rfloor)))$$

$$=n+3\lfloor n/4\rfloor+9\lfloor \tfrac{n}{16}\rfloor+27T(\lfloor n/64\rfloor)$$

$$=n+3\lfloor n/4\rfloor+3^2\left\lfloor \tfrac{n}{4^2}\right\rfloor+3^3\left(\left\lfloor \tfrac{n}{4^3}\right\rfloor\right)+\ldots+3^iT(\lfloor n/4^i\rfloor)$$

$\boxed{令 \frac{n}{4^i}=1\Rightarrow 4^i=n\Rightarrow i=\log_4 n}$

$$=n+3\lfloor n/4\rfloor+3^2\left\lfloor \tfrac{n}{4^2}\right\rfloor+3^3\left(\left\lfloor \tfrac{n}{4^3}\right\rfloor\right)+\ldots+3^{\log_4 n}T(\lfloor 1\rfloor)$$

$$\le\sum_{i=0}^{\log_4 n}3^i\,n/4^i \qquad \le n\sum_{i=0}^{\infty}\left(\frac{3}{4}\right)^i=n\times\frac{1}{1-\frac{3}{4}}=4n=O(n$$

HIT CS&E

2.4.3 Master method

目的：求解 $T(n)=aT(n/b)+f(n)$ 型方程，$a\ge 1,b>0$是常数，

$f(n)$是正函数

方法：记住三种情况，则不用笔纸即可求解上述方程.

HIT CS&E

Master 定理

定理 2.4.1 设$a\ge1$和$b>1$是常数，$f(n)$是一个函数，$T(n)$是定义在非负整数集上的函数$T(n)=aT(n/b)+f(n)$. $T(n)$可以如下求解：

(1). 若 $f(n)=O(n^{\log_b a-\varepsilon})$，$\varepsilon>0$是常数，则 $T(n)=\theta(n^{\log_b a})$.

(2). 若 $f(n)=\theta(n^{\log_b a})$，则 $T(n)=\theta(n^{\log_b a}\lg n)$.

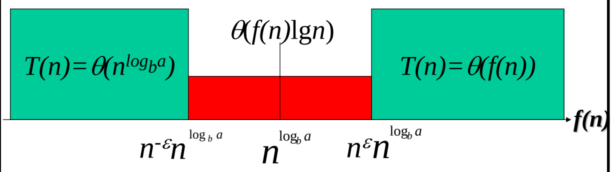
(3). 若 $f(n)=\Omega(n^{\log_b a+\varepsilon})$，$\varepsilon>0$是常数，且对于所有充分大的n $af(n/b)\le cf(n)$，C<1是常数，则 $T(n)=\theta(f(n))$.

*直观地：我们用 $f(n)$ 与 $n^{\log_b a}$ 比较

(1). 若 $n^{\log_b a}$ 大，则 $T(n)=\theta(n^{\log_b a})$

(2). 若 $f(n)$ 大，则 $T(n)=\theta(f(n))$

(3). 若 $f(n)$ 与 $n^{\log_b a}$ 同阶，则 $T(n)=\theta(n^{\log_b a}\lg n)=\theta(f(n)\lg n)$.



对于红色部分，Master定理无能为力

HIT CS&E

更进一步:

(1). 在第一种情况, $f(n)$不仅小于$n^{\log_b a}$，必须多项式地小于,即对于一个常数$\varepsilon>0$，$f(n)=O(\frac{n^{\log_b a}}{n^{\varepsilon}})$.

(2). 在第三种情况, $f(n)$不仅大于$n^{\log_b a}$，必须多项式地大于，即对一个常数$\varepsilon>0$，$f(n)=\Omega(n^{\log_b a}\cdot n^{\varepsilon})$.

HIT CS&E

Master定理的使用

例 1.求解 $T(n)=9T(n/3)+n$.

解: $a=9$，$b=3$，$f(n)=n$，$n^{\log_b a}=\theta(n^2)$

$\because f(n)=n=O(n^{\log_b a-\varepsilon})$，$\varepsilon=1$

$\therefore T(n)=\theta(n^{\log_b a})=\theta(n^2)$

例 2. 求解 $T(n)=T(2n/3)+1$.

解: $a=1$，$b=(3/2)$，$f(n)=1$，$n^{\log_b a}=n^{\log_{3/2}1}=n^0=1$，

$f(n)=1=\theta(1)=\theta(n^{\log_b a})$，$T(n)=\theta(n^{\log_b a}\lg n)=\theta(\lg n)$

HIT CS&E

例 3. 求解 $T(n)=3T(n/4)+n\lg n$

解: $a=3$, $b=4$, $f(n)=n\lg n$, $n^{\log_b a}=n^{\log_4 3}=O(n^{0.793})$

(1) $f(n)=n\lg n\ge n=n^{\log_b a+\varepsilon}$，$\varepsilon\approx 0.2$

(2) 对所有 n, $af(n/b)=3\times\frac{n}{4}\lg\frac{n}{4}=\frac{3}{4}n\lg\frac{n}{4}\le\frac{3}{4}n\lg n=cf(n)$，$c=\frac{3}{4}$.

于是，$T(n)=\theta(f(n))=\theta(n\lg n)$

例 4. 求解 $T(n)=2T(n/2)+n\lg n$.

解: $a=2$, $b=2$, $f(n)=n\lg n$, $n^{\log_b a}=n$. $f(n)=n\lg n$ 大于 $n^{\log_b a}=n$，但不是多项式地大于，Master 定理不适用于该 $T(n)$.

HIT CS&E

Master定理的证明

引理 1: 设 a≥1, b>1, n=b^k, k 是正整数， 则方程

$$T(n)=\begin{cases}\theta(1) & if\quad n=1\\ aT(n/b)+f(n) & if\quad n=b^i\end{cases}$$

的解为:

$$T(n)=\theta(n^{\log_b a})+\sum_{j=0}^{\log_b n-1}a^j f(n/b^j)$$

证明: $T(n)=f(n)+aT(n/b)$

$=f(n)+af(n/b)+a^2T(n/b^2)$

$=f(n)+af(n/b)+a^2f(n/b^2)+a^3T(n/b^3)+\ldots$

$+a^{\log_b n-1}f(n/b^{\log_b n-1})+a^{\log_b n}T(n/b^{\log_b n})$

由于 $a^{\log_b n}=n^{\log_b a}$，$a^{\log_b n}T(n/b^{\log_b n})=a^{\log_b n}T(1)=\theta(n^{\log_b a})$. 于是

$$T(n)=\theta(n^{\log_b a})+\sum_{j=0}^{\log_b n-1}a^j f(n/b^j).$$

HIT CS&E

Master定理的证明(续)

引理 2: 设 a≥1, b>1, n=b^k, k 是正整数, $g(n)=\sum_{j=0}^{\log_b n-1}a^j f(n/b^j)$, 则

(1) if $f(n)=O(n^{\log_b a-\varepsilon})$ for $\varepsilon>0$, 则 $g(n)=O(n^{\log_b a})$

(2) if $f(n)=\theta(n^{\log_b a})$, then $g(n)=\theta(n^{\log_b a}\lg n)$

(3) if $af(n/b)\le cf(n)$ for some 0<c<1 and all n≥b, then $g(n)=\theta(f(n))$.

HIT CS&E

引理2的证明

(1) if $f(n)=O(n^{\log_b a-\varepsilon})$ for $\varepsilon>0$，则 $g(n)=O(n^{\log_b a})$

证明： (1) $g(n)=O(\sum_{j=0}^{\log_b n-1} a^j (\frac{n}{b^j})^{\log_b a-\varepsilon})$

$$= n^{\log_b a-\varepsilon}\sum_{j=0}^{\log_b n-1}(\frac{ab^{\varepsilon}}{b^{\log_b a}})^j$$

$$= n^{\log_b a-\varepsilon}\sum_{j=0}^{\log_b n-1}(b^{\varepsilon})^j$$

$$= n^{\log_b a-\varepsilon}(\frac{b^{\varepsilon\log_b n}-1}{b^{\varepsilon}-1})$$

$$= n^{\log_b a-\varepsilon}(\frac{n^{\varepsilon}-1}{b^{\varepsilon}-1})$$

$$= n^{\log_b a-\varepsilon}O(n^{\varepsilon})=O(n^{\log_b a})$$

HIT CS&E

引理2的证明（续）

(2) if $f(n)=\theta(n^{\log_b a})$, then $g(n)=\theta(n^{\log_b a}\lg n)$

证明:

(2) 由于 $f(n/b^j)=\theta((n/b^j)^{\log_b a})$， $g(n)=\theta(\sum_{j=0}^{\log_b n-1} a^j(\frac{n}{b^j})^{\log_b a})$.

$$\sum_{j=0}^{\log_b n-1} a^j(\frac{n}{b^j})^{\log_b a}=n^{\log_b a}\sum_{j=0}^{\log_b n-1}(\frac{a}{b^{\log_b a}})^j=n^{\log_b a}\sum_{j=0}^{\log_b n-1}1=n^{\log_b a}\log_b n$$

$$\Rightarrow g(n)=\theta(n^{\log_b a}\log_b n)=\theta(n^{\log_b a}\lg n).$$

(3) if $af(n/b)\le cf(n)$ for some 0<c<1 and all n≥b, then $g(n)=\theta(f(n))$.

(3) g(n)中的所有项皆为正. 由于对于 0<c<1 和 all n≥b,

$af(n/b)\le cf(n)$,

$af(\frac{n}{b^2})\le cf(\frac{n}{b})$,

$af(\frac{n}{b^3})\le cf(\frac{n}{b^2})$,

…

$af(\frac{n}{b^j})\le cf(\frac{n}{b^{j-1}})$

我们有 $a^j f(n/b)\cdots f(n/b^{j-1})f(n/b^j)\le c^j f(n)f(n/b)\cdots f(n/b^{j-1})$

$$\Rightarrow a^j f(\frac{n}{b^j})\le c^j f(n)$$

于是,

$$g(n)=\sum_{j=0}^{\log_b n-1} a^j f(\frac{n}{b^j})\le\sum_{j=0}^{\log_b n-1} c^j f(n)\le f(n)\sum_{j=0}^{\infty}c^j=f(n)(\frac{1}{1-c})=\theta(f(n))$$

HIT CS&E

Master定理的证明(续)

引理 3: a≥1, b>1, n=b^k, k 为正整数，则

$$T(n)=\begin{cases}\theta(1) & if \quad n=1\\ aT(n/b)+f(n) & if \quad n=b^k\end{cases}$$

的解为:

(1) if $f(n)=O(n^{\log_b a-\varepsilon})$ for some $\varepsilon>0$, then $T(n)=\theta(n^{\log_b a})$

(2) if $f(n)=\theta(n^{\log_b a})$, then $T(n)=\theta(n^{\log_b a}\lg n)$

(3) if $f(n)=\Omega(n^{\log_b a+\varepsilon})$ for some $\varepsilon>0$, and if

$af(n/b)\le cf(n)$ for some 0<c<1 and all 充分大的 n, then $T(n)=\theta(f(n))$

HIT CS&E

引理3的证明

证明: (1) 由引理 1 和引理 2:

$T(n)=\theta(n^{\log_b a})+O(n^{\log_b a})=\theta(n^{\log_b a})$

(2) $T(n)=\theta(n^{\log_b a})+\theta(n^{\log_b a}\lg n)=\theta(n^{\log_b a}\lg n)$

(3) $T(n)=\theta(n^{\log_b a})+\theta(f(n))=\theta(f(n))$

HIT CS&E

Master定理的证明(续)

当 n 不是 b 的幂时

思想: 当 $T(n)$单调递增（单调递减类似）

- $aT(\lfloor\frac{n}{b}\rfloor)+f(n)\le aT(\frac{n}{b})+f(n)\le aT(\lceil\frac{n}{b}\rceil)+f(n)$
- 求 $T(n)=aT(\lceil\frac{n}{b}\rceil)+f(n)$的上界、$T(n)=aT(\lfloor\frac{n}{b}\rfloor)+f(n)$的下界可得到 $T(n)$的界限。
- 求 $T(n)=aT(\lfloor\frac{n}{b}\rfloor)+f(n)$的下界类似于求 $T(n)=aT(\lceil\frac{n}{b}\rceil)+f(n)$的上界，所以我们只求 $T(n)=aT(\lceil\frac{n}{b}\rceil)+f(n)$的上界

HIT CS&E

Master定理的证明(续)

- 方法仍然是循环展开 $T(n)=aT(\lceil \frac{n}{b} \rceil)+f(n)$
- 需要处理序列：

$$n$$
$$\left\lceil \frac{n}{b} \right\rceil$$
$$\left\lceil \left\lceil \frac{n}{b} \right\rceil \Big/ b \right\rceil$$
$$\left\lceil \left\lceil \left\lceil \frac{n}{b} \right\rceil \Big/ b \right\rceil \Big/ b \right\rceil$$
$$\cdots$$

HIT CS&E

Master定理的证明(续)

定义：$n_i=\begin{cases} n & if \quad i=0 \\ \lceil n_{i-1}/b \rceil & if \quad i>0 \end{cases}$

引理 4. $n_0 \le n$，$n_1 \le \frac{n}{b}+1$，$n_2 \le \frac{n}{b^2}+\frac{1}{b}+1$，$n_3 \le \frac{n}{b^3}+\frac{1}{b^2}+\frac{1}{b}+1$，$\ldots$，$n_i \le \frac{n}{b^i}+\sum_{j=0}^{i-1}\frac{1}{b^j} \le \frac{n}{b^i}+\frac{b}{b-1}$。

证：由 $\lceil x \rceil \le x+1$ 可证。

HIT CS&E

Master定理的证明(续)

引理 5：当 $i=\lfloor \log_b n \rfloor$ 时，$n_i \le b+\frac{b}{b-1}=O(1)$

证：由于 $n_i \le \frac{n}{b^i}+\frac{b}{b-1}$，我们有

$$n_{\lfloor \log_b n \rfloor} \le \frac{n}{b^{\lfloor \log_b n \rfloor}}+\frac{b}{b-1} \le \frac{n}{b^{(\log_b n)-1}}+\frac{b}{b-1}=b+\frac{b}{b-1}=O(1)。$$

HIT CS&E

Master定理的证明(续)

引理 6: $T(n)=aT(\lceil \frac{n}{b} \rceil)+f(n) \le \theta(n^{\log_b a})+\sum_{j=0}^{\lfloor \log_b n \rfloor -1} a^j f(n_j)$

证：
$$\begin{aligned} T(n) &= f(n_0)+aT(n_1)=f(n_0)+af(n_1)+a^2 f(n_2) \\ &\le f(n_0)+af(n_1)+a^2 f(n_2)+\ldots+a^{\lfloor \log_b n \rfloor -1} f(n_{\lfloor \log_b n \rfloor -1}) \\ &\quad + a^{\lfloor \log_b n \rfloor} T(n_{\lfloor \log_b n \rfloor}) \\ &= \theta(n^{\log_b a})+\sum_{j=0}^{\lfloor \log_b n \rfloor -1} a^j f(n_j) \end{aligned}$$

HIT CS&E

Master定理的证明(续)

引理 7: $g(n)=\sum_{j=0}^{\lfloor \log_b n \rfloor -1} a^j f(n_j)$ 可以界限如下：

（1） If $f(n)=O(n^{\log_b a-\varepsilon})$ for $\varepsilon>0$，$g(n)=O(n^{\log_b a})$.

（2） If $f(n)=\theta(n^{\log_b a})$, then $g(n)=\theta(n^{\log_b a}\lg n)$.

（3） If $af(\lceil n/b \rceil) \le cf(n)$ for 0<c<1 and all 充分大的 n, then $g(n)=\theta(f(n))$.

HIT CS&E

引理7的证明

证明：（3）由 $af(\lceil n/b \rceil) \le cf(n)$ 有:

$$af(\left\lceil \frac{n}{b} \right\rceil) \le cf(n) \Leftrightarrow af(n_1) \le cf(n_0)$$

$$af(\left\lceil \left\lceil \frac{n}{b} \right\rceil \Big/ b \right\rceil) \le cf(\left\lceil \frac{n}{b} \right\rceil) \Leftrightarrow af(n_2) \le cf(n_1)$$

$$\cdots$$

$$af(\left\lceil \ldots \left\lceil \frac{n}{b} \right\rceil \Big/ b \ \ldots \right\rceil) \le cf(\left\lceil \ldots \left\lceil \frac{n}{b} \right\rceil \ \ldots \right\rceil) \Leftrightarrow af(n_j) \le cf(n_{j-1})$$

$$\Rightarrow a^j f(n_1)\cdots f(n_{j-1})f(n_j) \le c^j f(n_0)f(n_1)\cdots f(n_{j-1})$$

$$\Rightarrow a^j f(n_j) \le c^j f(n_0)=c^j f(n)$$

证明的其余部分与引理 2 的（3）的证明类似。

HIT CS&E

引理7的证明

（2）只要证明 $f(n_j)=O(n^{\log_b a}/a^j)=O((\frac{n}{b^j})^{\log_b a})$，即可用引理 2 的（2）的证明完成本证明。

$$j \le \lfloor \log_b n \rfloor \Rightarrow b^j \le b^{\lfloor \log_b n \rfloor} = b^{\log_b n-\delta} = n \cdot \frac{1}{b^\delta} \quad (0 \le \delta < 1)$$

$$\Rightarrow \frac{b^j}{n} \le \frac{1}{b^\delta} < 1\ (\because b>1) \Rightarrow b^j/n < 1。$$

由于 $f(n)=\theta(n^{\log_b a})$，$\exists c>0$，使对于充分大的 n_j，

$$\begin{aligned} f(n_j) &\le cn_j^{\log_b a} \le c(\frac{n}{b^j}+\frac{b}{b-1})^{\log_b a} \\ &= c(\frac{n}{b^j})^{\log_b a}(1+\frac{b^j}{n}\cdot\frac{b}{b-1})^{\log_b a} \\ &\le c(\frac{n^{\log_b a}}{a^j})(1+\frac{b}{b+1})^{\log_b a} \qquad (\because \frac{b^j}{n}<1) \\ &\le O(\frac{n^{\log_b a}}{a^j}) \qquad (\because c(1+\frac{b}{b+1})^{\log_b a} \text{是常数}) \end{aligned}$$

于是（2）被证明。

HIT CS&E

引理7的证明

(1) 只要证明 $f(n_j)=O((\frac{n}{b^j})^{\log_b a-\varepsilon})$，则本证明的其余部分与引理 2 的（1）相同。类似（2）可证明 $f(n_j)=O((\frac{n}{b^j})^{\log_b a-\varepsilon})$。

至此，我们完成了 **Master** 定理的证明。

第三章

Sorting and correctness proof with Loop Invariant

李建中
数据与知识工程研究中心
哈工大计算机科学技术学院

提要

参考资料

《Introduction to Algorithm》

Chapter 6, 7, 8

3.1 Heapsort

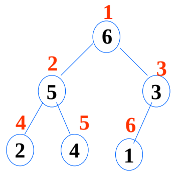
- **Heaps**
- **Maintaining heap property**
- **Building a heap**
- **Heapsort Algorithm**

Heap

- A heap is an array object that can be viewed as a nearly complete binary tree.

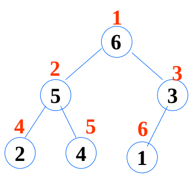
A

6	5	3	2	4	1



A

6	5	3	2	4	1



- *length*[A]=size of the array A.
- *Heap-size*[A]=number of elements in the heap
- *Heap-size*[A] $\le$ *length*[A]
- *Parent*(*i*)=$\lfloor i/2 \rfloor$
- *Left*(*i*)=*2i*
- *Right*(*i*)=*2i+1*

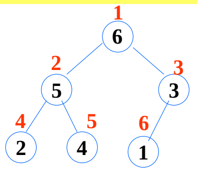
HIT CS&E

Two kinds of Heap

- **Max-Heap**

In a max-heap, every node i other than the root satisfies the following property:

$A[\text{Parent}(i)] \geq A[i]$.



A

6	5	3	2	4	1

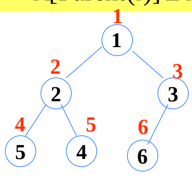
The largest element is at the root.

HIT CS&E

- **Min-Heap**

In a min-heap, every node i other than the root satisfies the following property:

$A[\text{Parent}(i)] \leq A[i]$.



A

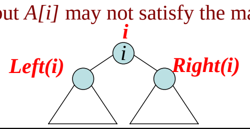
1	2	3	5	4	6

The smallest element is at the root.

HIT CS&E

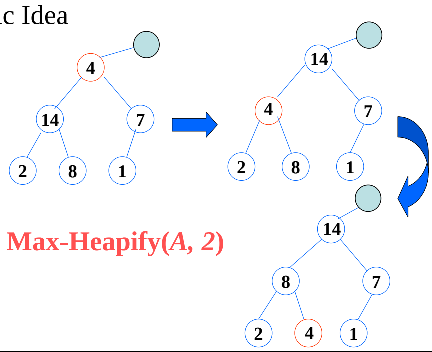
Maintaining the heap property

- Max-Heapify(*A*, *i*)
  - Input: array *A* and index *i* into *A*
  - Output: makes the subtree rooted at *A[i]* become a max-heap
  - Assume:
    - two subtrees rooted at *Left(i)* and *Right(i)* are max-heaps, but *A[i]* may not satisfy the max-heap property.



HIT CS&E

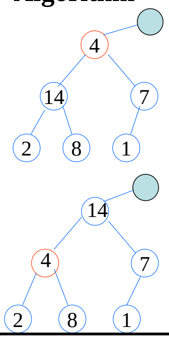
- Basic Idea



Max-Heapify(*A*, 2)

HIT CS&E

- **Algorithm**



Max-Heapify(A, i)
$l \leftarrow \text{Left}(i)$;
$r \leftarrow \text{Right}(i)$;
if $l \leq \text{heap-size}[A]$ and $A[l] > A[i]$
  then $\text{largest} \leftarrow l$
  else $\text{largest} \leftarrow i$;
if $r \leq heap-size[A]$ and $A[r] > A[largest]$
  then $\text{largest} \leftarrow r$;
if $\text{largest} \neq i$
  then begin exchange $A[i] \leftrightarrow A[\text{largest}]$;
    Max-Heapify(A, largest);

HIT CS&E

- Time Complexity

Let $h = height$. Then h is the largest integer satisfying $1+2+\cdots+2^{h-1} \leq n$, that is, $2^h \leq n$. Hence, $h = \lfloor \lg n \rfloor$.

$height = \lfloor \lg n \rfloor$ where $n = heap\text{-}size[A]$

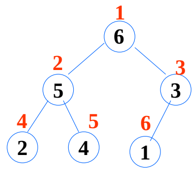
Hence, Max-Heapify runs in time $O(\lg n)$.

HIT CS&E

Building a heap

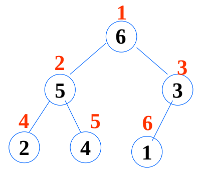
- Lemma. Let n=*length*[A], then the elements in the subarray A[($\lfloor n/2 \rfloor$+1) .. n] are all leaves of the three.

  Proof: Homework



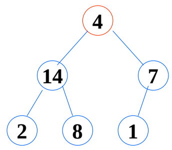
HIT CS&E

- Idea of the algorithm
  - The elements in A[($\lfloor n/2 \rfloor$+1) .. n] are 1-element heap
  - The algorithm goes through the rest nodes of the tree and runs Max-Heapify on each one.



HIT CS&E

- **Algorithm**



Build-Max-Heap(A)

heap-size[A] $\leftarrow$ *length*[A];

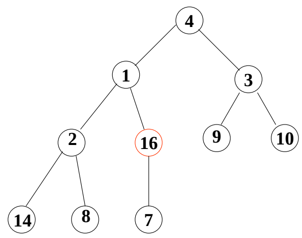
for $i \leftarrow \lfloor \textit{length}[A]/2 \rfloor$ downto 1

do Max-Heapify(A, i);

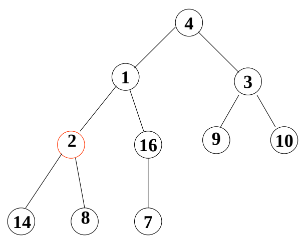
HIT CS&E

- **实例:**

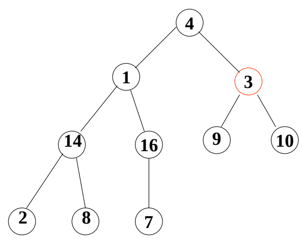
4	1	3	2	16	9	10	14	8	7

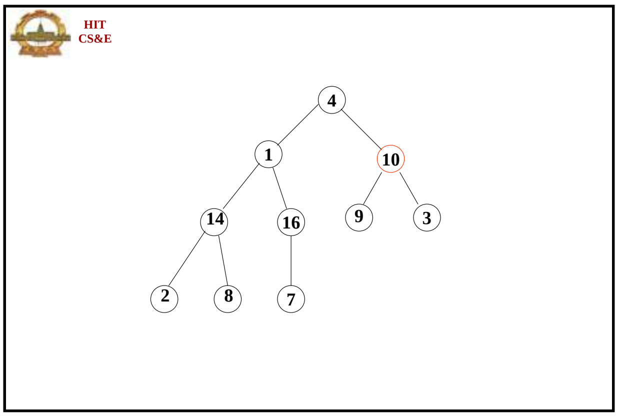


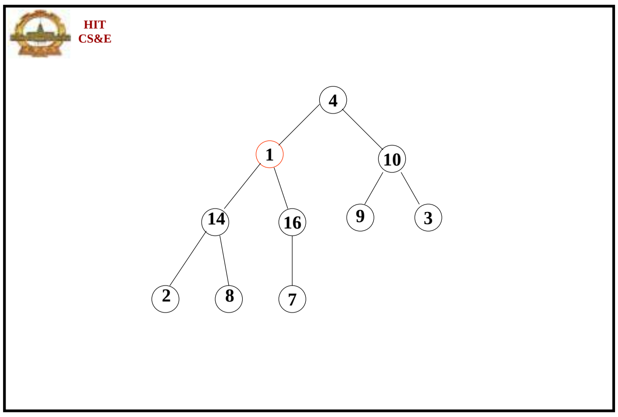
HIT CS&E

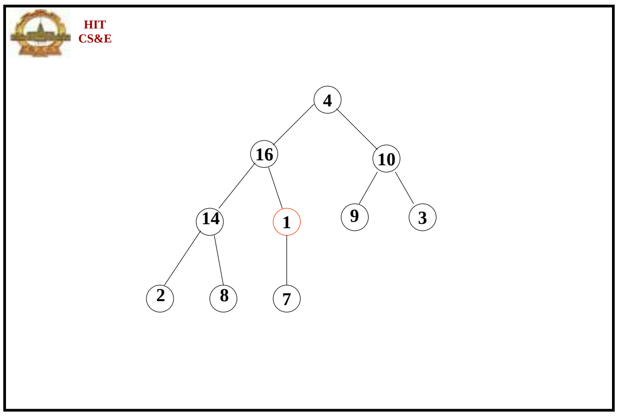


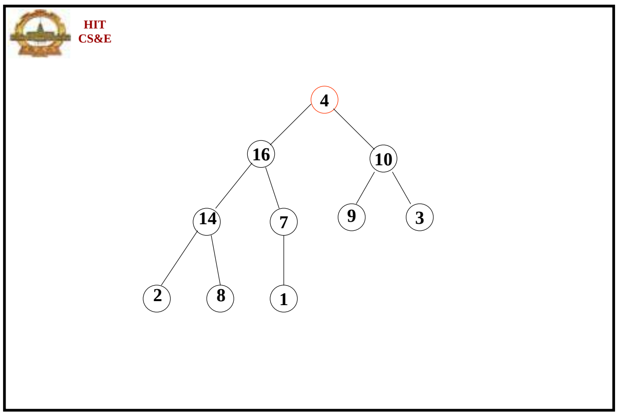
HIT CS&E

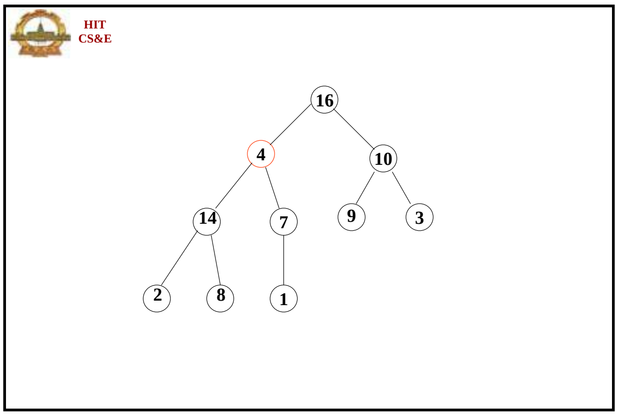


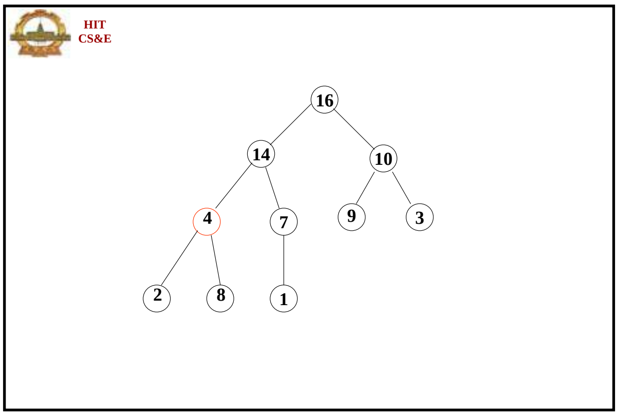

HIT
CS&E
4
1
10
14
16
9
3
2
8
7


HIT
CS&E
4
1
10
14
16
9
3
2
8
7

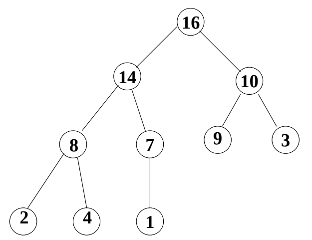

HIT
CS&E
4
16
10
14
1
9
3
2
8
7


HIT
CS&E
4
16
10
14
7
9
3
2
8
1


HIT
CS&E
16
4
10
14
7
9
3
2
8
1


HIT
CS&E
16
14
10
4
7
9
3
2
8
1

HIT CS&E

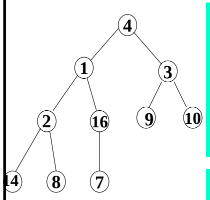


HIT CS&E

- Correctness proof by loop invariant
  - Loop invariant:
    在第i次循环开始之前, 节点$i+1, i+2, ..., n$皆为heap的根.
- Proof by induction
  - Initialization:
    第1次循环开始之前, $i=\lfloor n/2 \rfloor$. 节点$\lfloor n/2 \rfloor+1, \lfloor n/2 \rfloor+2, \ldots, n$是叶, 于是皆为1-element heap的根. 循环不变量为真.
  - Maintenance:
    设第i次循环开始时循环不变量为真, 即$i+1, \ldots, n$皆为heap的根. 算法Max-Heapify(A, i)使得i为heap的根. 于是, 在第i-1次循环开始之前, $i, i+1, \ldots, n$皆为heap的根. 循环不变量为真.
- Termination: $i=0$, 节点1, 2,…, n皆为heap根, 算法正确.

HIT CS&E

- Time complexity



running time $= \sum_{h=0}^{\lfloor height \rfloor} \#node(h) \cdot O(h)$

where $\#node(h)$ is the number of nodes rooted at which the subtree has height h.

$height = \lfloor \lg n \rfloor$ where $n = heap\text{-}size(A)$

$\#node(h) = \left\lceil \frac{n}{2^{h+1}} \right\rceil$ **Proof is Homework**

HIT CS&E

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h} \right) = O(n)$$

because

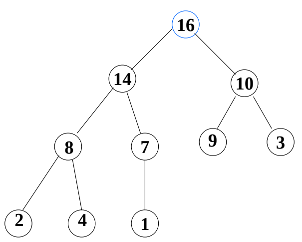
$$\sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{1/2}{(1-1/2)^2} = 2$$

HIT CS&E

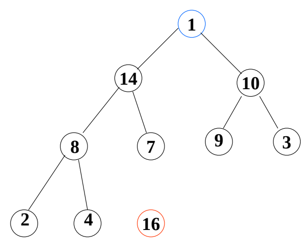
Heapsort Algorithm

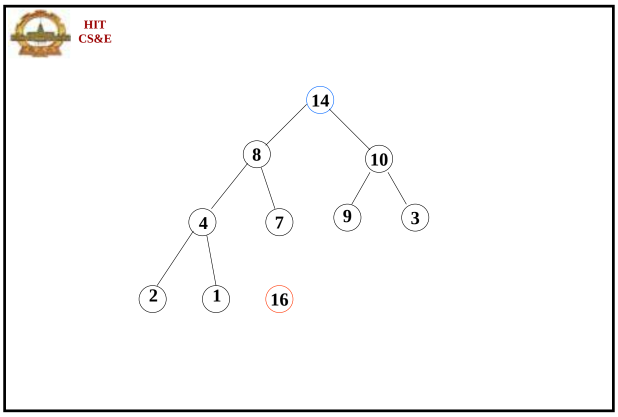
- Basic idea

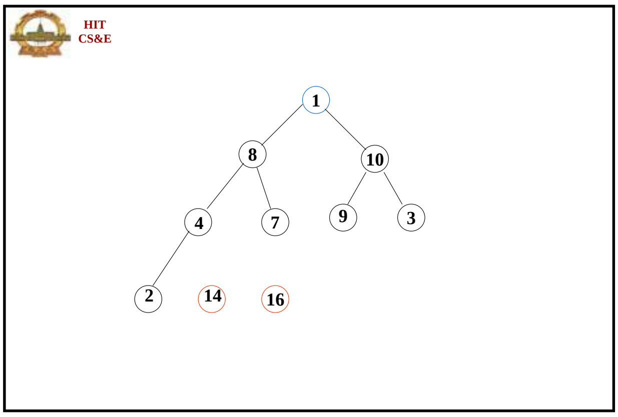
16	14	10	8	7	9	3	2	4	1

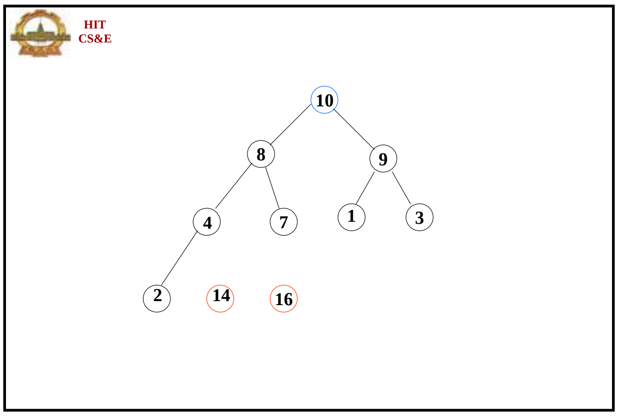


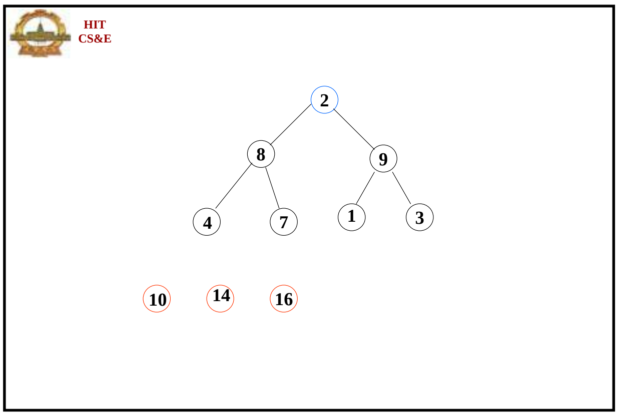
HIT CS&E

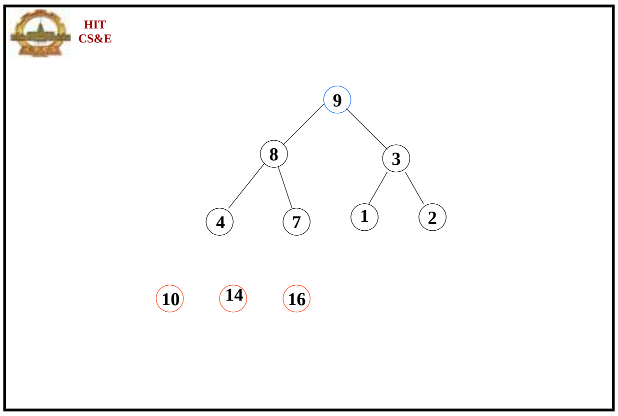


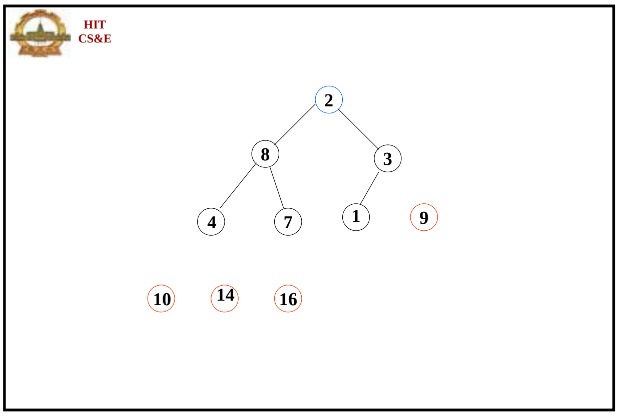

HIT
CS&E
14
8 10
4 7 9 3
2 1 16

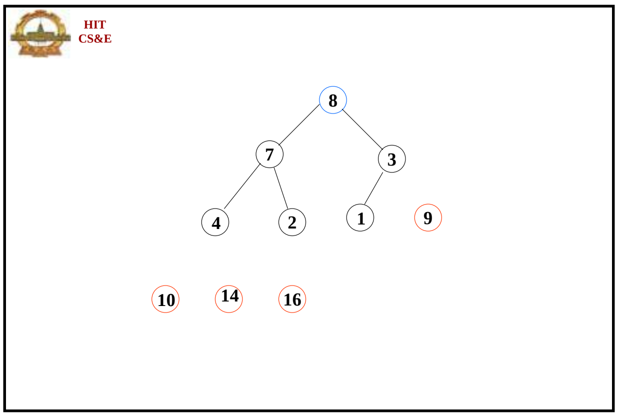

HIT
CS&E
1
8 10
4 7 9 3
2 14 16

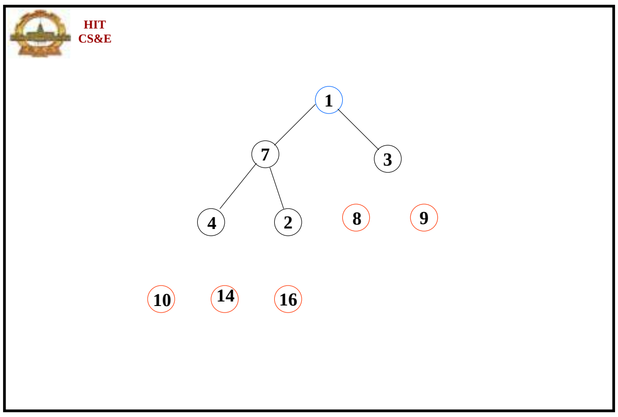

HIT
CS&E
10
8 9
4 7 1 3
2 14 16

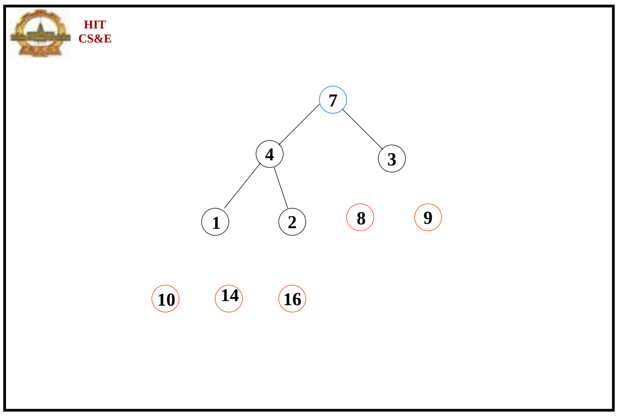

HIT
CS&E
2
8 9
4 7 1 3
10 14 16

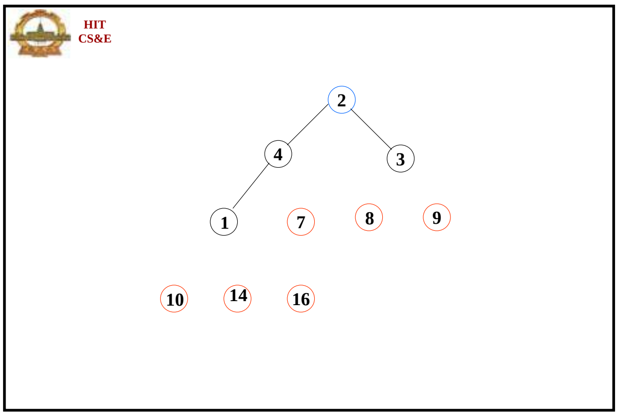

HIT
CS&E
9
8 3
4 7 1 2
10 14 16

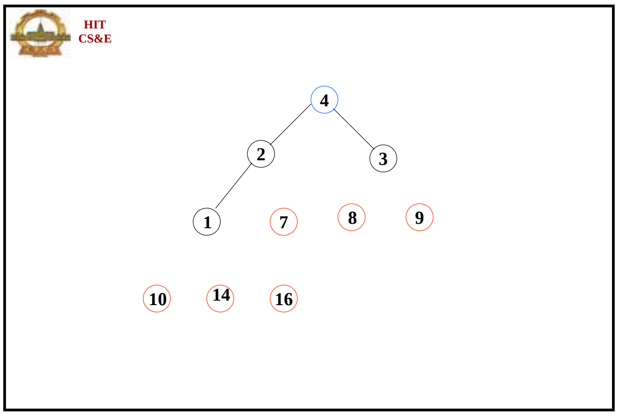

HIT
CS&E
2
8 3
4 7 1 9
10 14 16

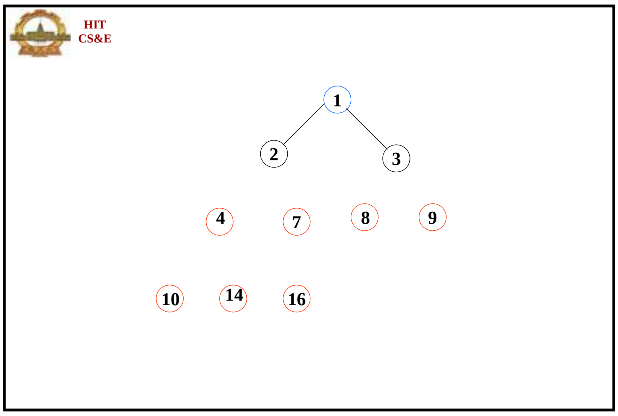

HIT
CS&E
8
7 3
4 2 1 9
10 14 16


HIT
CS&E
1
7 3
4 2 8 9
10 14 16

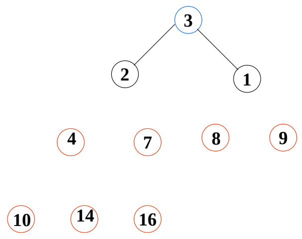

HIT
CS&E
7
4 3
1 2 8 9
10 14 16


HIT
CS&E
2
4 3
1 7 8 9
10 14 16

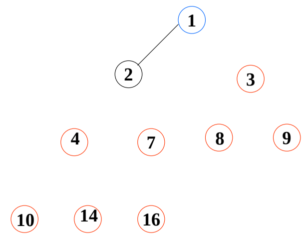

HIT
CS&E
4
2 3
1 7 8 9
10 14 16


HIT
CS&E
1
2 3
4 7 8 9
10 14 16

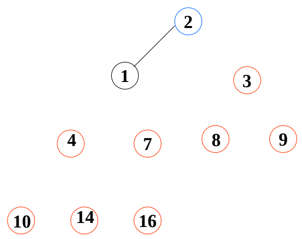
HIT CS&E



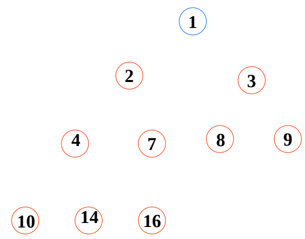
HIT CS&E



HIT CS&E



HIT CS&E



HIT CS&E

- Summary of the Basic idea
  - Using Build-Max-Heap to build a max-heap on the input array *A[1 .. n]*.
  - Since the maximum element is at the root *A[1]*, it can be put into its correct final position by exchanging it with *A[n]*
  - Run *Max-Heapify(A, 1)* on *A[1 .. n-1]*
  - Repeat the process until all elements are processed

HIT CS&E

- Algorithm

Heapsort(A)
  Buid - Max - Heap(A);
  for $i \leftarrow length[A]$ downto 2
    do begin
      exchange $A[1] \leftrightarrow A[i]$;
      $heap\text{-}size[A] \leftarrow heap\text{-}size[A] - 1$;
      Max - Heapify(A,1);
    end - for

HIT CS&E

- **Time complexity**

Heapsort(A)

Buid - Max - Heap(A); O(n)

for $i \leftarrow length[A]$ downto 2 O(n)次循环

do begin

exchange $A[1] \leftrightarrow A[i]$;

$heap\text{-}size[A] \leftarrow heap\text{-}size[A]-1$;

Max - Heapify(A,1); 每次O(lg n)

end - for

Time complexity is O($n$lg n)

HIT CS&E

3.2 Quicksort

- Idea of Quicksort
- Quicksort Algorithm
- Correctness Proof
- Performance Analysis

HIT CS&E

Idea of Quicksort

- Divide-and-Conquer
  - Divide:
    - Partition $A[p..r]$ into $A[p..q]$ and $A[q+1..r]$.
    - $\forall x \in A[p...q]$, $\forall y \in A[q+1...r]$, $x \leq y$.
    - q is generated by partition algorithm.
  - Conquer:
    - Sort $A[p...q]$ and $A[p+1...r]$ using quicksort recursively
  - Combine:
    - Since $A[p...q]$ and $A[p+1...r]$ have been sorted, nothing to do

HIT CS&E

Quicksort Algorithm

QUICKSORT(A,p,r)
If $p<r$
Then q=Partition(A, p, r);
QUICKSORT(A, p, q);
QUICKSORT(A, $q+1$, r);

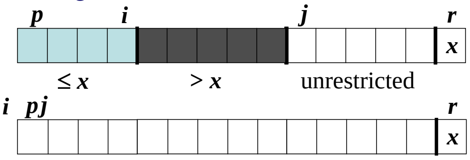
HIT CS&E

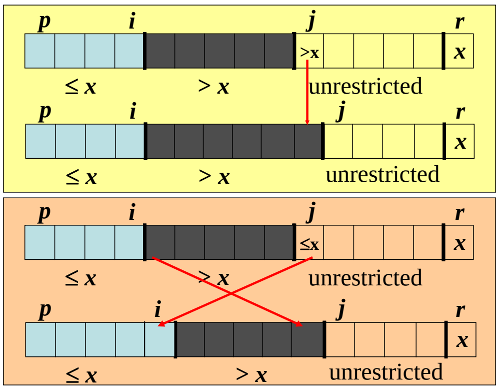
- **Example**

3, 4, 2, 1, $\hat{5}$	8, 9, 7, 6
1, 2, 3, 4, $\hat{5}$	6, 7, 8, 9

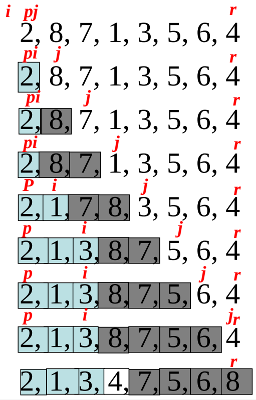
HIT CS&E

- Partition Algorithm
  - Take x=$A[r]$ as a pivot
  - Classify other elements by comparing with x.
  - During the running of the algorithm, A is partitioned into 4 regions:





HIT CS&E



Partition(A, p, r)

$x \leftarrow A[r]$;

$i \leftarrow p-1$;

for $j \leftarrow p$ to $r-1$

do if $A[j] \le x$

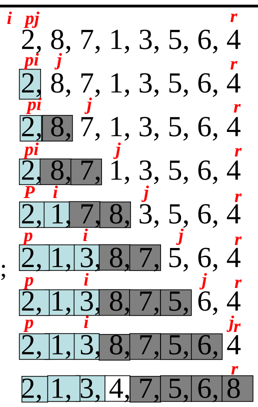
$i \leftarrow i+1$;

exchange $A[i] \leftrightarrow A[j]$;

exchange $A[i+1] \leftrightarrow A[r]$;

return $i+1$;

Running time: $\Theta(n)$



HIT CS&E

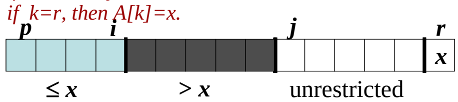
- Correctness Proof
  - Loop Invariant

At the start of the loop of lines 3-6 for any k

1. if $p \le k \le i$, then $A[k] \le x$.

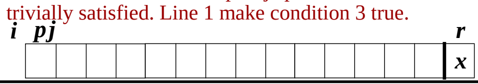
2. if $i+1 \le k \le j-1$, then $A[k] > x$.

3. if $k=r$, then $A[k]=x$.



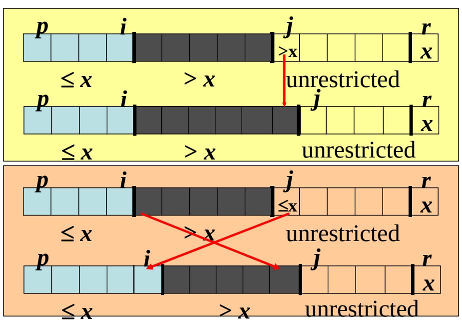
  - Initialization:

Prior the first iteration: $i=p-1$, $j=p$, condition 1 and 2 are trivially satisfied. Line 1 make condition 3 true.



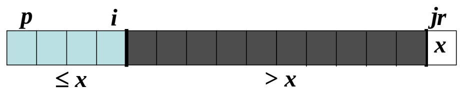
HIT CS&E

- Maintenance:



HIT CS&E

- Termination



At termination, $j=r$. We have three sets:

1. those less than or equal to x.
2. those greater than x.
3. a singleton set containing x.

- After finishing the algorithm

The final two steps move the pivot into its position in the middle of the array by swapping it with the leftmost element that is greater than x.

HIT
CS&E

- Performance analysis
  - Time complexity of PARTITION: $\theta(n)$
  - Best case time complexity of Quicksort
    - Array in partition into 2 equal sets
    - $T(n)=2T(n/2)+\theta(n)$
    - $T(n)=\theta(nlogn)$

HIT
CS&E

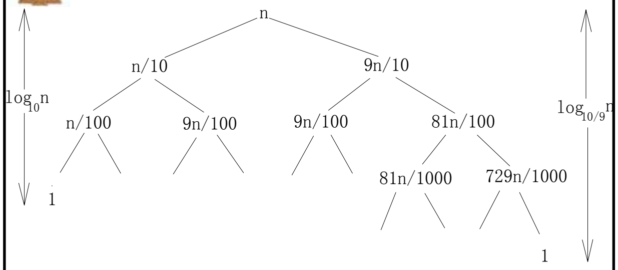
- Worst case time complexity of Quicksort
  - Worst Case
    - |A[p..q]|=1，|A[q+1..r]|=n-1
    - The worst case happens in call to Partition Algorithm
  - Time complexity
    - $T(1)=\theta(1)$
    - $T(n)=T(n-1)+\theta(n)=\theta(n^2)$

HIT
CS&E

- Average time complexity of Quicksort
  - Average time complexity is near $\theta(nlgn)$ rather than $\theta(n^2)$
  - Why? See an example
    - Assume PARTITION always generates 9 :1 partition
    - Run time of QUICKSORT is

      $T(n)=T(9n/10)+T(n/10)+\theta(n^2) = \boldsymbol{\theta(nlgn)}$
    - The process of Partitioning is as followoing

HIT
CS&E



- 从第一曾到$\log_{10}n=\theta(\log n)$层，每层花费$n$时间
- 从第$\log_{10}n$到第$\log_{10/9}n=\theta(\log n)$层，每层最多花费n时间
- 在9:1的不平衡划分下，运行时间仍是$\boldsymbol{\theta(n\log n)}$

HIT
CS&E

3.3 Sorting in Linear Time

- **Lower bounds for sorting**
- **Counting Sort Algorithm**
- **Radix Sort Algorithm**
- **Bucket Sort Algorithm**

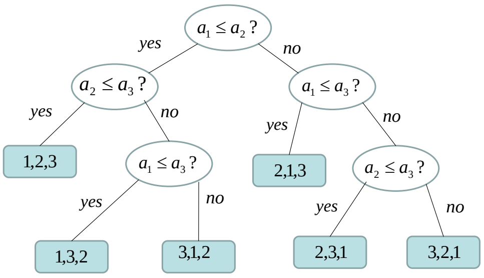
HIT
CS&E

Lower bounds for sorting

- Decision-tree model
  - Comparison sort
    - Use only comparisons between elements to gain order information about input sequence
    - Comparison sort can be viewed as decision-tree
  - Decision-tree model
    - Is a full binary tree to express the comparisons of a sort algorithm by ignoring the other aspects of the algorithm
    - All permutations of the input elements must be the leaves
    - A inter node of the tree is a comparison $a_i \leqslant a_j$

HIT CS&E

- A decision-tree for insert sort on 3 input elements



HIT CS&E

- Lower bound for the worst case

In a comparison sort, the (worst case) running time is the depth (or height) of the decision tree=$T(n)$.

Since the decision tree has $n!$ leaves, its depth $T(n)$ satisfies

$$2^{T(n)} \ge n!$$

Thus, $T(n) = \Omega(n \lg n)$.

Are there sort algorithm with complexity *O(n)?*

HIT CS&E

Counting Sort

- Input: *A[1..n]* , 0⩽*A[i]*⩽*k* for 1⩽*i*⩽*n*
- Output: *B[1..n]*=sorted *A[1..n]*
- Idea
  - Use *C[0..k]* to compute the position of each *A[i]*
  - Put each *A[i]* for *i=n to 1* into *B[1..n]*

HIT CS&E

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	2	0	2	3	0	1

	1	2	3	4	5	6	7	8
B								

	0	1	2	3	4	5
C	2	2	4	7	7	8

	1	2	3	4	5	6	7	8
B							3	

	0	1	2	3	4	5
C	2	2	4	6	7	8

	1	2	3	4	5	6	7	8
B		0					3	

	0	1	2	3	4	5
C	1	2	4	6	7	8

	1	2	3	4	5	6	7	8
B		0				3	3	

	0	1	2	3	4	5
C	1	2	4	5	7	8

	1	2	3	4	5	6	7	8
B	0	0	2	2	3	3	3	5

HIT CS&E

- Algorithm

for $i \leftarrow 1$ to k — $A: 3, 6, 4, 1, 3, 4, 1, 4$

  do $C[i] \leftarrow 0;$

for $j \leftarrow 1$ to $length[A]$ — $C: 0, 0,0, 0,0, 0$

  do $C[A[j]] \leftarrow C[A[j]]+1;$ — $C: 2, 0, 2, 3, 0, 1$

for $i \leftarrow 2$ to k

  do $C[i] \leftarrow C[i]+C[i-1];$ — $C: 2, 2, 4, 7, 7, 8$

for $j \leftarrow length[A]$ downto 1

  do begin — $A: 3, 6, 4, 1, 3, 4, 1, \hat{4}$

    $B[C[A[j]]] \leftarrow A[j];$ — $B: \ , \ , \ , \ , \ , \ , 4,$

    $C[A[j]] \leftarrow C[A[j]]-1;$ — $C: 2, 2, 4, 6, 7, 8$

HIT CS&E

- Time complexity

for $i \leftarrow 1$ to k — $O(k)$

  do $C[i] \leftarrow 0;$

for $j \leftarrow 1$ to $length[A]$ — $O(n)$

  do $C[A[j]] \leftarrow C[A[j]]+1;$

for $i \leftarrow 2$ to k — $O(k)$

  do $C[i] \leftarrow C[i]+C[i-1];$

for $j \leftarrow length[A]$ downto 1 — $O(n)$

  do begin

    $B[C[A[j]]] \leftarrow A[j];$

    $C[A[j]] \leftarrow C[A[j]]-1;$

Time Complexity=O(n+k)

HIT CS&E

- Property of Counting Sort
  - Counting sort doesn't sort in place
  - Counting sort is stable
    - That is, the same value appear in the output array in the same order as they do in the input array.

HIT CS&E

Radix Sort Algorithm

- Idea of Radix sort algorithm
  - Use stable sort algorithm
  - Sort the *n* *d*-digit elements from the lowest digit to the highest digit

329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

HIT CS&E

- Radix sort algorithm

Ininput : Array A, each element is a number of d digit.
Radix - Sort(A, d)
for $i \leftarrow 1$ to d do
    use a stable sort to sort array A on digit i;

- Time complexity of Radix sort algorithm
  - Using Counting sort algorithm, $0 \leqslant A[i] \leqslant k-1$
  - The time complexity is $O(d(n+k))$

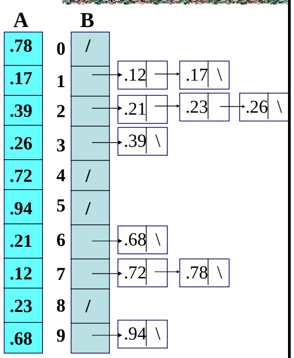
HIT CS&E

- Extension of Radix sort
  - Input: *n* *b*-digit number, any $r \leqslant b$
  - Radix sort can sort these numbers in $\Theta((b/r)(n+2^r))$
  - Why
    - View each number as $d=\lceil b/r \rceil$ digits of r bits each.
    - Each digit is an integer in the range 0 to 2^r-1
    - Use counting sort with $k= 2^r-1$

HIT CS&E

Bucket Sort

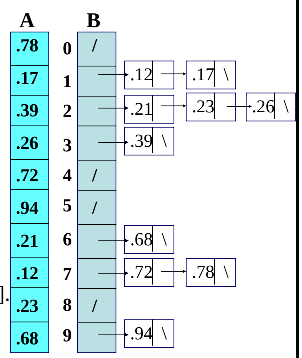
- **Assumption of Bucket Soot**
  - Input is elements uniformly distributed in [0, 1)
- **Idea of Bucket Soot**
  - Divide [0, 1) into *n* equal-sized bucket
  - Distribute the input into the *n* bucket
  - Sort the numbers in each bucket
  - List all the sorted numbers in each bucket in order



HIT CS&E

- **Bucket Sort Algorithm**

Bucket-Sort(A)
1. n=$length[A]$;
2. For i=1 To n Do
3.   Insert $A[i]$ into list $B[\lfloor nA[i] \rfloor]$;
4. For i=0 To n-1 Do
5.   Sort list $B[i]$ with insert sort;
6. concatenate lists $B[0]$, …, $B[n-1]$.



HIT
CS&E

- **Time complexity**
  - Let the random variable n_i=|$B[i]$|
  - The time complexity:

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^{\ 2})$$

  - Since E[n_i^2]=*2-1/n* (Homewok)

$$E[T(n)] = \Theta(n) + \sum_{i=0}^{n-1} E[O(n_i^{\ 2})]$$
$$= \Theta(n) + O(n(2-1/n)) = \Theta(n)$$

HIT CS&E

第四章

Divide-and-Conquer 技术

李建中
数据与知识工程研究中心

HIT CS&E

提要

HIT CS&E

参考资料

Chapter 3

Page 80 - 96

HIT CS&E

4.1 Divide-and-Conquer原理

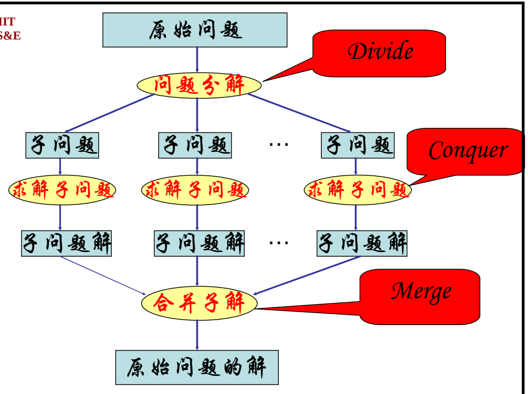
- Divide-and-Conquer算法的设计
- Divide-and-Conquer算法的分析

HIT CS&E

Divide-and-Conquer算法的设计

- 设计过程分为三个阶段
  - Divide: 整个问题划分为多个子问题
  - Conquer: 求解各子问题(递归调用正设计的算法)
  - Combine: 合并子问题的解, 形成原始问题的解

HIT CS&E



HIT CS&E

Divide-and-Conquer算法的分析

- 分析过程
  - 建立递归方程
  - 求解
- 递归方程的建立方法
  - 设输入大小为n, $T(n)$为时间复杂性
  - 当$n<c$, $T(n)=\theta(1)$

HIT CS&E

- Divide阶段的时间复杂性
  - 划分问题为a个子问题。
  - 每个子问题大小为n/b。
  - 划分时间可直接得到$=D(n)$
- Conquer阶段的时间复杂性
  - 递归调用
  - Conquer时间$=aT(n/b)$
- Combine阶段的时间复杂性
  - 时间可以直接得到$=C(n)$

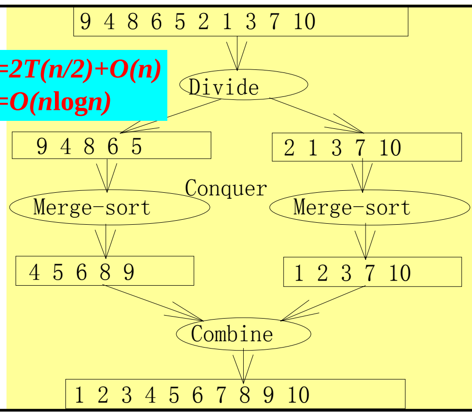
HIT CS&E

- 总之
  - $T(n)=\theta(1)$ if $n<c$
  - $T(n)=aT(n/b)+D(n)+C(n)$ otherwise
- 求解递归方程$T(n)$
  - 使用第二章的方法

例1.

$T(n)=2T(n/2)+O(n)$

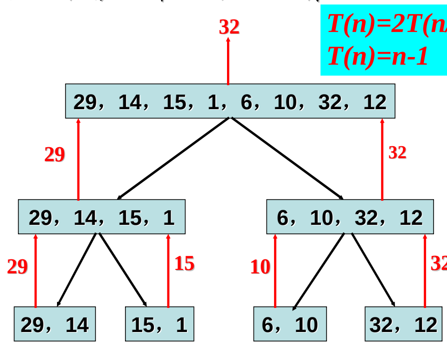
$T(n)=O(n\log n)$



例2. 求一个集合中的最大数算法

$T(n)=2T(n/2)+1$

$T(n)=n-1$



HIT CS&E

4.2 整数乘法

HIT CS&E

问题定义

输入：n位二进制整数X和Y
输出：X和Y的乘积

通常，计算X*Y时间复杂性为O(n^2)，
我们给出一个复杂性为O($n^{1.59}$)的算法。

HIT CS&E

算法的数学基础

X= | A (n/2位) | B (n/2位) | Y= | C (n/2位) | D (n/2位) |

$XY = (A2^{n/2} + B)(C2^{n/2} + D)$
$= AC2^n + AD2^{n/2} + BC2^{n/2} + BD$
$= AC2^n + ((A-B)(D-C)+AC+BD)2^{n/2} + BD$

如此计算需要
$T(n)=4T(n/2) + O(n)=O(n^2)$

算法

1. 计算A-B和D-C;
2. 计算n/2位乘法AC、BD、(A-B)(D-C);
3. 计算M=(A-B)(D-C)+AC+BD;
4. N=AC左移n位，M=M左移n/2位；
5. 计算XY=N+M+BD。

HIT CS&E

算法的分析

- 建立递归方程

$T(n)=\theta(1)$ if $n=1$

$T(n)=3T(n/2)+O(n)$ if $n>1$

- 使用Master定理

$T(n)=O(n^{\log 3})=O(n^{1.59})$

HIT CS&E

4.3 矩阵乘法

HIT CS&E

问题定义

输入：两个n× n矩阵A和B
输出：A和B的积

通常，计算AB的时间复杂性为O(n^3)，
我们给出一个复杂性为O($n^{2.81}$)的算法。

HIT CS&E

算法的数学基础

- 把$C=AB$中每个矩阵分成大小相同的4个子矩阵

每个子矩阵都是一个$n/2 \times n/2$矩阵

- 于是

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & \\ A_{21} & \end{bmatrix}$$

如此计算需要
$T(n)=8T(n/2) +O(n^2)=O(n^3)$

- 展开并整理等式的右边：

$C_{11} = A_{11}B_{11} + A_{12}B_{21}$， $C_{12} = A_{11}B_{12} + A_{12}B_{12}$
$C_{21} = A_{21}B_{11} + A_{22}B_{21}$， $C_{12} = A_{21}B_{12} + A_{22}B_{12}$

HIT CS&E

算法

- 近一步简化得到$n/2\times n/2$矩阵的10个加减和7个乘法

$$M_1 = A_{11}(B_{12} - B_{22})$$
$$M_2 = (A_{11} + A_{12})B_{22}$$
$$M_3 = (A_{21} + A_{22})B_{11}$$
$$M_4 = A_{22}(B_{21} - B_{11})$$
$$M_5 = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$M_6 = (A_{12} - A_{22})(B_{21} + B_{22})$$
$$M_7 = (A_{11} - A_{12})(B_{11} + B_{12})$$

HIT CS&E

- 计算$C=AB$需要另外8个$n/2\times n/2$矩阵加减

$$C_{11} = M_5 + M_4 - M_2 + M_6$$
$$C_{12} = M_1 + M_2$$
$$C_{21} = M_3 + M_4$$
$$C_{22} = M_5 + M_1 - M_3 - M_7$$

HIT CS&E

算法复杂性分析

- 18个$n/2\times n/2$矩阵加减法，每个需$O(n^2)$
- 7个$n/2\times n/2$矩阵乘法
- 建立递归方程

$T(n)=O(1)$ $n=2$

$T(n)=7T(n/2) + O(n^2)$ $n>2$

- 使用Master定理求解T(n)

$T(n) = O(n^{\log 7}) \approx O(n^{2.81})$

HIT CS&E

4.4 Finding the closest pair of points

Page 92 – 96

HIT CS&E

问题定义

输入：Euclidean空间上的n个点的集合Q

输出：$P_1, P_2 \in Q$,

$Dis(P_1, P_2)=Min\{Dis(X, Y) \mid X, Y \in Q\}$

$Dis(X, Y)$是Euclidean距离：

如果$X=(x_1, y_1)$, $Y=(x_2, y_2)$，则

$$Dis(X,Y)=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$$

HIT CS&E

一维空间算法

- 利用排序的算法
  - 算法
    - 把Q中的点排序

    - 通过排序集合的线性扫描找出最近点对
  - 时间复杂性
    - $T(n)=O(n\log n)$

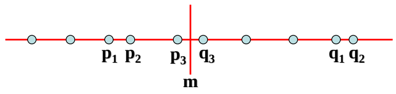
HIT CS&E

一维空间算法(续)

- Divide-and-conquer算法

Preprocessing:

1. 如果Q中仅包含2个点，则返回这个点对；
2. 否则求Q中点的中位数m。



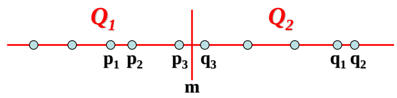
©DB-LAB(2003)

HIT CS&E

Divide:

3. 用Q中点坐标中位数m把Q划分为两个大小相等的子集合

$Q_1 = \{x \in Q \mid x \le m\}$, $Q_2 = \{x \in Q \mid x > m\}$



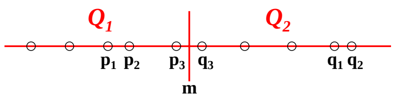
©DB-LAB(2003)

HIT CS&E

Conquer:

4. 递归地在Q_1和Q_2中找出最接近点对 (p_1, p_2)和(q_1, q_2)

Merge:



5. 在(p_1, p_2)、(q_1, q_2)和某个(p_3, q_3)之间选择最接近点对(x, y)，其中p_3是Q_1中最大点，q_3是Q_2中最小点，(x, y)是Q中最接近点。

HIT CS&E

- 时间复杂性
  - Divide阶段需要$O(n)$时间
  - Conquer阶段需要$2T(n/2)$时间
  - Merge阶段需要$O(n)$时间
  - 递归方程

    $T(n) = O(1)$ $n = 2$

    $T(n) = 2T(n/2) + O(n)$ $n \ge 3$
  - 用Master定理求解$T(n)$

    $T(n) = O(n\log n)$

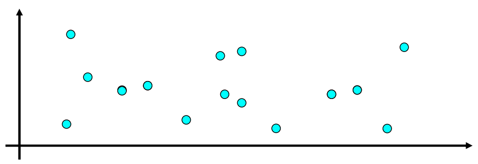
HIT CS&E

二维空间算法

- Divide-and-conquer算法

Preprocessing:

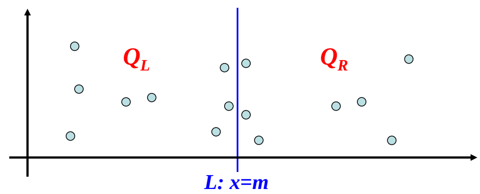
1. 如果Q中仅包含一个点，则算法结束；
2. 否则把Q中点分别按x-坐标值和y-坐标值排序.

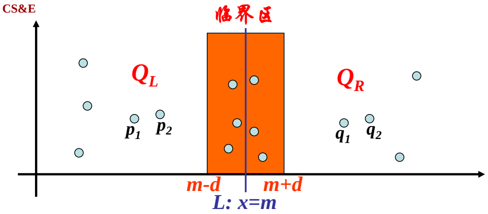


HIT CS&E

Divide:

3. 计算Q中各点x-坐标的中位数m；
4. 用垂线L:$x=m$把Q划分成两个大小相等的子集合Q_L和Q_R，Q_L中点在L左边，Q_R中点在L右边.

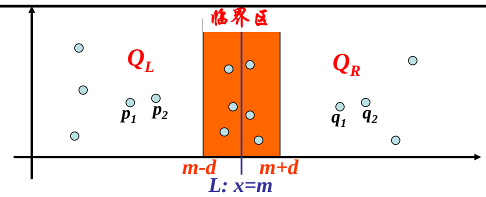




Conquer:

5. 递归地在Q_L、Q_R中找出最接近点对: $(p_1, p_2)\in Q_L$, $(q_1, q_2)\in Q_R$

6. $d=\min\{Dis(p_1, p_2), Dis(q_1, q_2)\}$;



Merge:

1. 在临界区查找距离小于d的点对(p_l, q_r), $p_l\in Q_L$, $q_r\in Q_R$;
2. 若找到, 则(p_l, q_r)是临界区中最接近点对, 否则(p_1, p_2)和(q_1, q_2) 中距离最小者为Q中最接点对.

关键是(p_l, q_r)的搜索方法及其搜索时间

HIT CS&E

- (p_l, q_r)的搜索算法
  1. 把临界区中所有点集合投影到分割线L上;
  2. 对于左临界区的每个点p, 考察p-右临界区的每个q (这样的点至多6个), 如果$Dis(p, q)<d$, 则令 $d=Dis(p, q)$;
  3. 如果d发生过变化, 与最后的d对应的点对即为(p_l, q_r), 否则不存在(p_l, q_r).

HIT CS&E

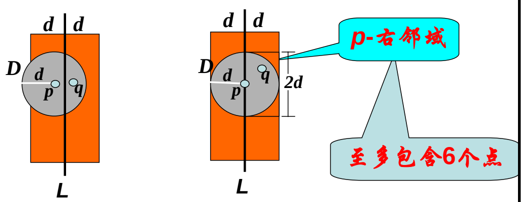
- 时间复杂性
  - Divide阶段需要$O(n)$时间
  - Conquer阶段需要$2T(n/2)$时间
  - Merge阶段需要$O(n)$时间
  - 递归方程

    $T(n)= O(1)$ $\quad n = 2$

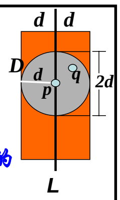
    $T(n) = 2T(n/2) + O(n)$ $\quad n\geq 3$
  - 用Master定理求解$T(n)$

    $T(n) = O(n\log n)$

(p_l, q_r)的搜索时间:
- 若(p, q)是最接近点对而且$p\in Q_L$, $q\in Q_R$, 则$dis(p, q)<d$, (p, q)只能在下图的区域D.
- 若p在分割线L上, 包含(p, q)的区域D最大, 嵌于$d\times 2d$的矩形(p-右邻域)中, 如下图所示.

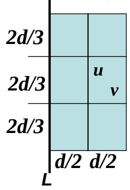


HIT CS&E



定理1. 对于左临界区中的每个点p, p-右邻域中仅包含6个点。

证明: 把p-右邻域划分为6个$(d/2)\times(2d/3)$的矩形。



若p-右邻域中点数大于6, 由鸽巢原理, 至少有一个矩形中有两个点. 设为u、v, 则

$(x_u-x_v)^2+ (y_u-y_v)^2\leq (d/2)^2+(2d/3)^2=25d^2/36$

即$Dis(u, v)\leq 5d/6<d$, 与d的定义矛盾。

HIT
CS&E

4.5 Finding the convex hull

HIT
CS&E

问题定义

输入：平面上的*n*个点的集合*Q*
输出： CH(*Q*): *Q*的convex hull

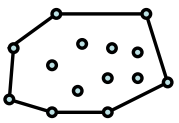
*Q*的convex hull是一个最小凸多边形*P*，*Q*的点或者在*P*上或者在*P*内

凸多边形*P*是具有如下性质多边形：连接*P*内任意两点的边都在*P*内

HIT
CS&E

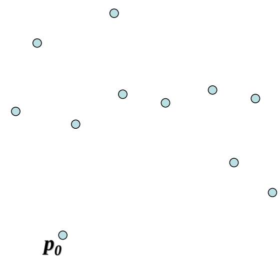
Graham-Scan算法

- 基本思想
  - 当沿着Convex hull逆时针漫游时，总是向左转
  - 在极坐标系下按照极角大小排列，然后逆时针方向漫游点集，去除非Convex hull顶点(非左转点)。



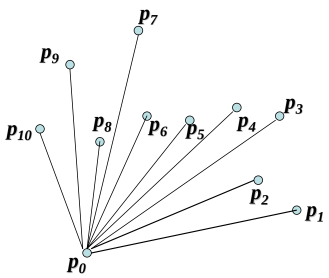
HIT
CS&E

第1步



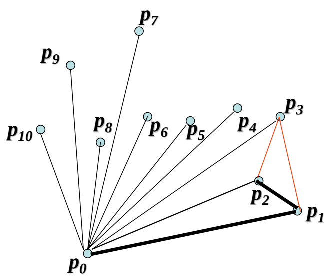
HIT
CS&E

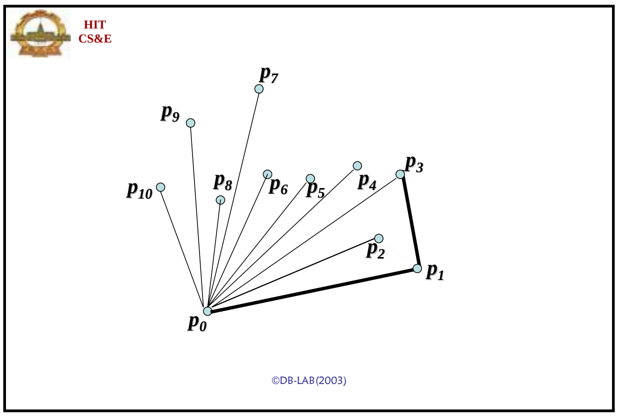
第2步

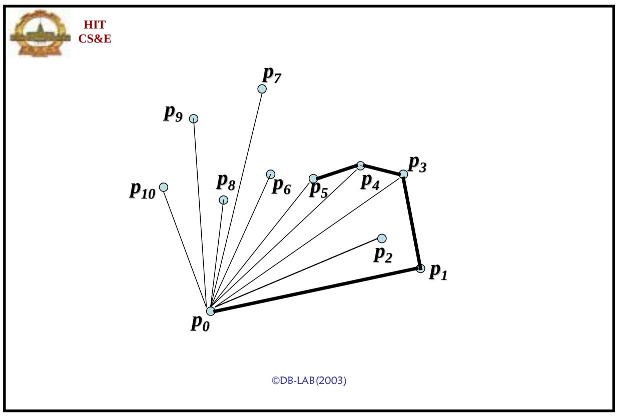


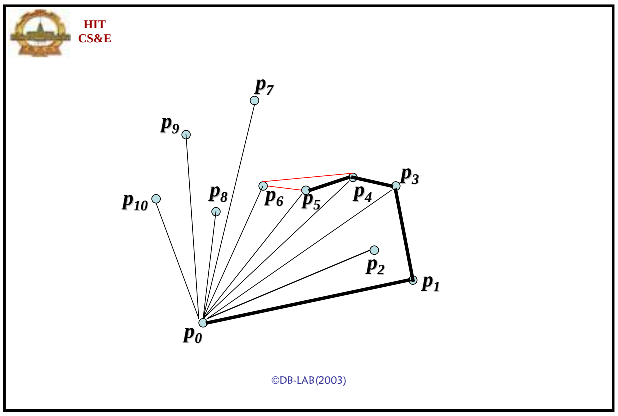
HIT
CS&E

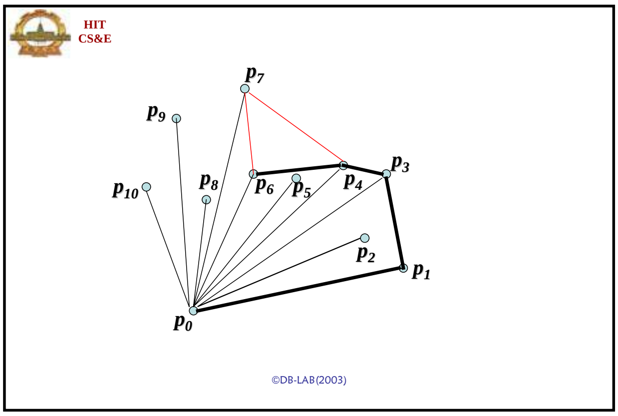
第3步

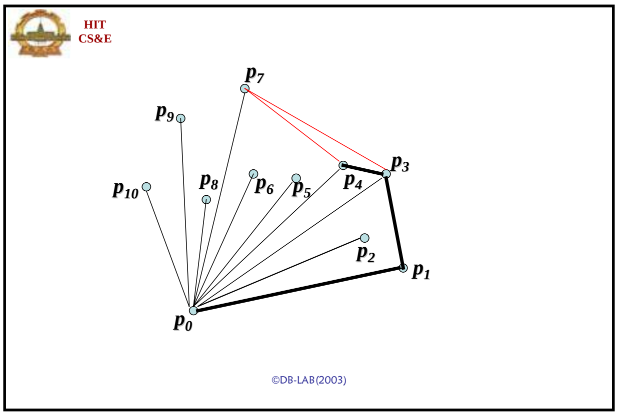


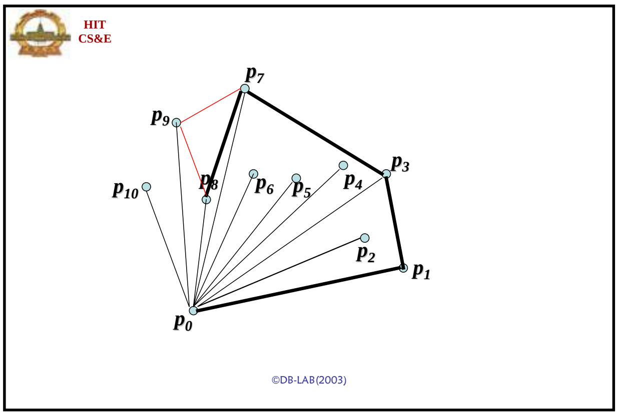

HIT
CS&E
p_7
p_9
p_{10}
p_8
p_6
p_5
p_4
p_3
p_2
p_1
p_0
©DB-LAB(2003)

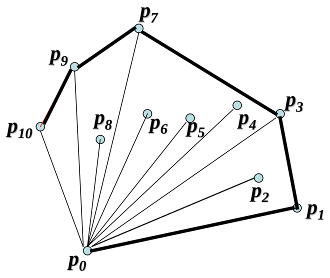

HIT
CS&E
p_7
p_9
p_{10}
p_8
p_6
p_5
p_4
p_3
p_2
p_1
p_0
©DB-LAB(2003)


HIT
CS&E
p_7
p_9
p_{10}
p_8
p_6
p_5
p_4
p_3
p_2
p_1
p_0
©DB-LAB(2003)


HIT
CS&E
p_7
p_9
p_{10}
p_8
p_6
p_5
p_4
p_3
p_2
p_1
p_0
©DB-LAB(2003)


HIT
CS&E
p_7
p_9
p_8
p_6
p_5
p_4
p_3
p_2
p_1
p_0
©DB-LAB(2003)


HIT
CS&E
p_7
p_9
p_{10}
p_8
p_6
p_5
p_4
p_3
p_2
p_1
p_0
©DB-LAB(2003)



算法Graham-Scan(Q)

/* 栈S从底到顶存储按逆时针方向排列的CH(Q)顶点 */

1. 求Q中y-坐标值最小的点p_0;
2. 按照与p_0极角(逆时针方向)大小排序Q中其余点，结果为<$p_1, p_2, \cdots, p_n$>;
3. Push(p_0, S); Push(p_1, S); Push(p_2, S);
4. FOR i=3 TO n DO
5. While Next-to-top(S)、Top(S)和p_i形成非左移动 Do
6. Pop(S);
7. Push(p_i, S);
8. Rerurn S.

- 时间复杂性
  - 第1步需要$O(n)$时间
  - 第2步需要$O(n\log n)$时间
  - 第3步需要$O(1)$时间
  - 第4-7步需要$O(n)$时间
  - 总时间复杂性$T(n)=O(n\log n)$

- 正确性分析

定理. 设n个二维点的集合Q是Graham-Scan算法的输入，|Q|≥3，算法结束时，栈S中自底到顶存储CH(Q)的顶点（按照逆时针顺序）.

证明： See pp. 88-90.

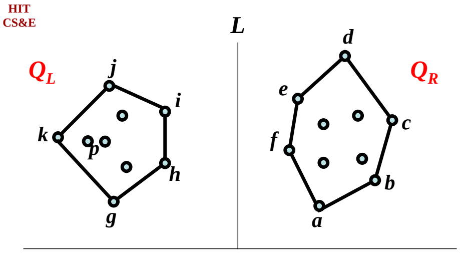
Divide-and-conquer算法

Preprocess: (时间复杂性为$O(1)$)

1. 如果|Q|<3, 算法停止;
2. 如果|Q|=3, 按照逆时针方向输出CH(Q)的顶点;

Divide:(使用$O(n)$算法求中值)

1. 选择一个垂直于x-轴的直线把Q划分为基本相等的两个集合Q_L和Q_R, Q_L在Q_R的左边;



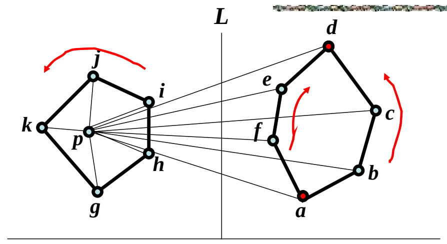
Conquer: (时间复杂性为$2T(n/2)$)

1. 递归地为Q_L和Q_R构造CH(Q_L)和CH(Q_R);

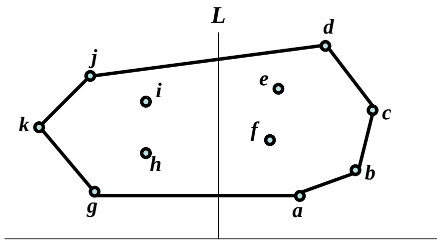
Merge:

我们先通过一个例子来看Merge的思想

Merge实例



3个序列：<g, h, i, j, k>, <a, b, c, d>, <f, e>
合并以后：<g, h, a, b, f, c, e, d, i, j, k>



Graham-Scan on *<g, h, a, b, f, c, e, d, i, j, k>*

Merge:(时间复杂性为$O(n)$)

1. 找一个Q_L的内点p;
2. 在CH(Q_R)中找与p的极角最大和最小顶点u和v;
3. 构造如下三个点序列:
   (1) 按逆时针方向排列的CH(Q_L)的所有顶点,
   (2) 按逆时针方向排列的CH(Q_R)从u到v的顶点,
   (3) 按顺时针方向排列的CH(Q_R)从u到v的顶点;
4. 合并上述三个序列;
5. 在合并的序列上应用Graham-Scan.

时间复杂性

- Preprocessing阶段
  - $O(1)$
- Divide阶段(使用$O(n)$算法求中值)
  - $O(n)$
- Conquer阶段
  - $2T(n/2)$
- Merge阶段
  - $O(n)$

时间复杂性

- 总的时间复杂性
  $T(n)=2T(n/2)+O(n)$
- 使用Master定理
  $T(n)= O(n\log n)$

HIT CS&E

第五章

*Dynamic Programming*技术

李建中
计算机科学与工程系

HIT CS&E

提要

HIT CS&E

参考资料

Chapter 4

Pages 97 －137

HIT CS&E

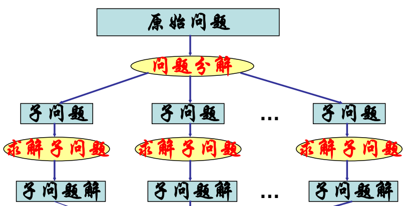
5.1 Elements of Dynamic Programming

Why?
What?
How?

HIT CS&E

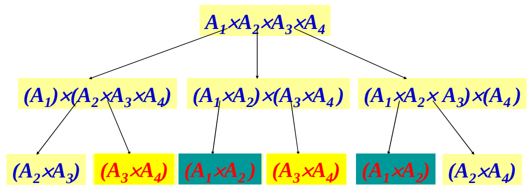
Why?

- **Divide-and-conquer技术的问题**



问题：如果子问题不是相互独立的，分治方法将重复计算公共子问题，效率很低

- **优化问题的特点**
  - 优化问题：给定一个代价函数，在解空间中搜索具有最小或最大代价的优化解



  - 优化问题的特点
    - 可划分为多个子问题，优化解包含子问题的优化解
    - 子问题的解被重复使用

HIT CS&E

What?

- Dynamic Programming特点
  - 把原始问题划分成一系列子问题
  - 求解每个子问题仅一次，并将其结果保存在一个表中，以后用到时直接存取，不重复计算，节省计算时间
  - 自底向上地计算
- 适用范围
  - 一类优化问题：可分为多个相关子问题，子问题的解被重复使用

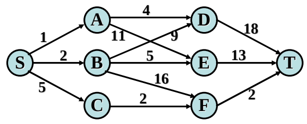
HIT CS&E

How?

- 使用Dynamic Programming的条件
  - Optimal substructure（优化子结构）
    - 当一个问题的优化解包含了子问题的优化解时，我们说这个问题具有优化子结构。
  - Subteties（重叠子问题）
    - 在问题的求解过程中，很多子问题的解将被多次使用

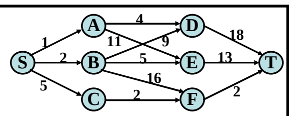
HIT CS&E

- Dynamic Programming的实例
  - 最短路径问题

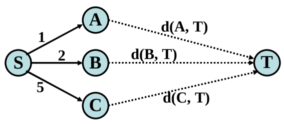


  - Dynamic Programming求解过程

HIT CS&E

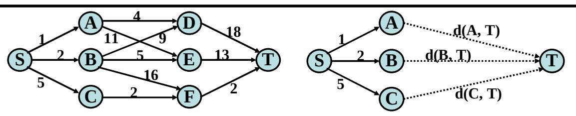


1：划分求d(S,T)问题为三个子问题

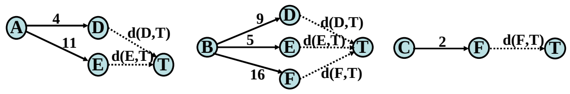


从S到T的最短路径长度为：

d(S,T)=min{1+d(A,T), 2+d(B,T), 5+d(C,T)}



2：划分d(A,T)、d(B,T)、d(C,T)为6个子问题

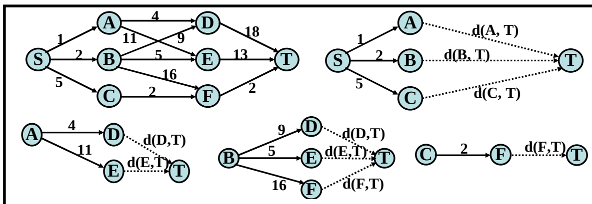


3：求解最小子问题

d(A,T)=min{4+d(D,T), 11+d(E,T)}=min{22, 24}=22 <A,D,T>

d(B,T)=min{9+d(D,T), 5+d(E,T), 16+d(F,T)}=min{27, 18, 18}=18 <B,E,T>

d(C,T)=min{2+d(F,T)}=4 <C,F,T>



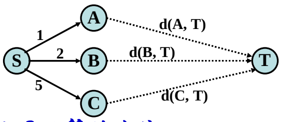
d(A,T)=22 <A,D,T>, d(B,T)=18 <B,E,T>, d(C,T)=4 <C,F,T>

4：最后确定从S到T的最短路径

d(S,T)=min{1+d(A,T), 2+d(B,T), 5+d(C,T)}=min{23, 20, 9}=9

<S, C, F, T>

- Dynamic Programming算法的设计步骤
  - 分析优化解的结构：划分子问题



  - 递归地定义最优解的代价
    d(S,T)=min{1+d(A,T), 2+d(B,T), 5+d(C,T)}
  - 递归地划分问题，直至不可划分
  - 自底向上求解各个子问题：
    - 计算优化解代价并保存之
    - 获取构造最优解的信息
  - 根据构造最优解的信息构造优化解

HIT CS&E

5.2 Matrix-chain Multiplication

HIT CS&E

问题的定义

- 输入：$<A_1, A_2, ..., A_n>$, A_i是$p_{i-1}\times p_i$矩阵
- 输出：计算$A_1\times A_2\times ...\times A_n$的最小代价方法

矩阵乘法的代价/复杂性：乘法的次数

若A是$p\times q$矩阵，B是$q\times r$矩阵，则$A\times B$的代价是$O(pqr)$

HIT CS&E

Motivation

- 矩阵链乘法的实现
  - 矩阵乘法满足结合率。
  - 计算一个矩阵链的乘法可有多种方法：

例如， $(A_1\times A_2\times A_3\times A_4)$
$= (A_1\times (A_2\times (A_3\times A_4)))$
$= ((A_1\times A_2)\times (A_3\times A_4))$
....
$= (((A_1\times A_2)\times A_3)\times A_4)$

HIT CS&E

- 矩阵链乘法的代价与计算顺序的关系
  - 设A_1=10×100矩阵，A_2=100×5矩阵，A_3=5×50矩阵
    $T((A_1\times A_2)\times A_3)=10\times 100\times 5+10\times 5\times 50=7500$
    $T(A_1\times (A_2\times A_3))=100\times 5\times 50+10\times 100\times 50=75000$

结论：不同计算顺序有不同的代价

HIT CS&E

- 矩阵链乘法优化问题的解空间
  - 设$p(n)$=计算n个矩阵乘积的方法数
  - $p(n)$的递归方程

$$(A_1 \times \ldots \times A_k)\times(A_{k+1}\times \ldots \times A_{n-1}\times A_n)$$

如此之大的解空间是无法用枚举方法求出最优解的！

$p(n)=C(n-1)$=Catalan数$=\frac{1}{n}\binom{2(n-1)}{n-1} = \Omega(4^n/n^{3/2})$

HIT CS&E

下边开始设计求解矩阵链乘法问题的 Dynamic Programming算法

- 分析优化解的结构
- 递归地定义最优解的代价
- 递归地划分子问题，直至不可分
- 自底向上地求解各个子问题
  - 计算优化解的代价并保存之
  - 获取构造最优解的信息
- 根据构造最优解的信息构造优化解

HIT CS&E

分析优化解的结构

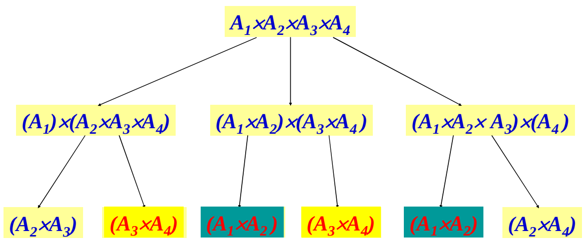
- 两个记号
  - $A_{i\sim j}=A_i\times A_{i+1}\times\ldots\times A_j$
  - $cost(A_{i\text{-}j})$=计算$A_{i\text{-}j}$的代价
- 优化解的结构

定理. 若计算$A_{1\sim n}$的优化顺序在k处断开矩阵链，即

具有优化子结构：
问题的优化解包括子问题优化解

解，对应于子问题$A_{k+1\sim n}$的解必须是$A_{k+1\sim n}$的优化解.

- 子问题重叠性



具有子问题重叠性

HIT CS&E

递归地定义最优解的代价

- 假设
  - $m[i, j]$ = 计算$A_{i\sim j}$的最小乘法数
  - $m[1, n]$ = 计算$A_{1\sim n}$的最小乘法数
  - $A_1 \ldots A_k A_{k+1} \ldots A_n$ 是优化解(k实际上是不可预知)
- 优化解$(A_i \ldots A_k)(A_{k+1} \ldots A_j)$的代价方程

$m[i, i]$ = 计算$A_{i\sim i}$的最小乘法数=0

$m[i, j] = m[i, k] + m[k+1, j] + p_{i-1}p_kp_j$

其中，$p_{i-1}p_kp_j$是计算$A_{i\sim k}\times A_{k+1\sim j}$所需乘法数，

$A_{i\sim k}$和$A_{k+1\sim j}$分别是$p_{i-1}\times p_k$和$p_k\times p_j$矩阵.

A_i是$p_{i-1}\times p_i$矩阵.

HIT CS&E

考虑到所有的k，优化解的代价方程为

$m[i, j]= 0$ if $i=j$

$m[i, j]= \min_{i\le k<j}\{ m[i, k]+m[k+1, j]+p_{i-1}p_kp_j\}$ if $i<j$

HIT CS&E

递归地划分子问题

$m[i, j]= \min_{i\le k<j}\{ m[i, k] + m[k+1, j] + p_{i-1}p_kp_j \}$

m[1,1]	m[1,2]	m[1,3]	m[1,4]	m[1,5]
	m[2,2]	m[2,3]	m[2,4]	m[2,5]
		m[3,3]	m[3,4]	m[3,5]
			m[4,4]	m[4,5]
				m[5,5]

HIT CS&E

自底向上计算优化解的代价

$m[i, j]= \min_{i\le k<j} \{ m[i, k] + m[k+1, j] + p_{i-1} p_k p_j \}$

m[1,1]	m[1,2]	m[1,3]	m[1,4]	m[1,5]
	m[2,2]	m[2,3]	m[2,4]	m[2,5]
		m[3,3]	m[3,4]	m[3,5]
			m[4,4]	m[4,5]
				m[5,5]


```
Matrix-Chain-Order(n)
FOR  i=1  TO  n  DO
     m[i, i]=0;
FOR  l=2  TO  n  DO /* 计算l对角线 */
     FOR  i=1  TO  n-l+1  DO
          j=i+l-1;
          m[i, j]= ∞;
          FOR  k←i  To  j-1  DO /* 计算m[i,j] */
               q=m[i, k]+m[k+1, j]+p_{i-1}p_k p_j
               IF  q<m[i, j]  THEN  m[i,j]=q;
Return m.
```


HIT CS&E

获取构造最优解的信息

S[i,j]记录$A_iA_{i+1}\ldots A_j$的最优划分处在A_k与A_{k+1}之间

```
Matrix-Chain-Order(p)
n=length(p)-1;
FOR  i=1  TO  n  DO
     m[i, i]=0;
FOR  l=2  TO  n  DO
     FOR  i=1  TO  n-l+1  DO
          j=i+l-1;
          m[i, j]= ∞;
          FOR  k←i  To  j-1  DO
               q = m[i, k]+m[k+1, j]+p_{i-1} p_k p_j
               IF q<m[i, j] THEN m[i,j]=q,  s[i,j]=k;
Return m and s.
```

HIT CS&E

构造最优解

```
Print-Optimal-Parens(s, i, j)
IF j=i
THEN  Print "A"i;
ELSE  Print "("
      Print-Optimal-Parens(s, i, s[i, j])
      Print-Optimal-Parens(s, s[i, j]+1, j)
      Print ")"
```

S[i, j]记录$A_i\ldots A_j$的最优划分处；
S[i, S[i,j]]记录$A_i\ldots A_{s[i,j]}$的最优划分处；
S[S[i,j]+1, j]记录$A_{s[i,j]+1}\ldots A_j$的最优划分处.

调用Print-Optimal-Parens(s, 1, n)即可输出$A_{1\sim n}$的优化计算顺序

HIT CS&E

算法复杂性

- 时间复杂性
  - 计算代价的时间
    - (l, i, k)三层循环，每层至多n-1步
    - $O(n^3)$
  - 构造最优解的时间：$O(n)$
  - 总时间复杂性为：$O(n^3)$
- 空降复杂性
  - 使用数组m和S
  - 需要空间$O(n^2)$

HIT CS&E

5.3 Longest Common Susequence

- 问题的定义
- 最长公共子序列 (LCS) 结构分析
- 建立求解LCS长度的递归方程
- 递归地划分子问题
- 自底向上计算LCS长度,记录解构造信息
- 构造优化解

HIT CS&E

问题的定义

- 子序列
  - $X=(A, B, C, B, D, B)$
  - $Z=(B, C, D, B)$是X的子序例
  - $W=(B, D, A)$不是X的子序例
- 公共子序列
  - Z是序列X与Y的公共子序列如果Z是X的子序也是Y的子序列。

HIT CS&E

最长公共子序列（LCS）问题

输入：$X = (x_1, x_2, \ldots, x_n)$，$Y = (y_1, y_2, \ldots y_m)$

输出：X与Y的最长公共子序列

$Z=(z_1, z_2, \ldots, z_n)$

HIT CS&E

最长公共子序列结构分析

- 第i前缀
  - 设$X=(x_1, x_2, \ldots, x_n)$是一个序列
    则$X_i=(x_1, \ldots, x_i)$是X的第i前缀

例，$X=(A, B, D, C, A)$, $X_1=(A)$, $X_2=(A, B)$, $X_3=(A, B, D)$

HIT CS&E

- 优化子结构

X和Y的LCS的优化解结构为

$LCS_{XY}=LCS_{X_{m-1}Y_{n-1}}+\langle x_m=y_n\rangle$ if $x_m=y_n$

$LCS_{XY}=LCS_{X_{m-1}Y}$ if $x_m\neq y_n$, $z_k\neq x_m$

$LCS_{XY}=LCS_{XY_{n-1}}$ if $x_m\neq y_n$, $z_k\neq y_n$

HIT CS&E

- 优化子结构

定理1（优化子结构）设$X=(x_1, \ldots, x_m)$、$Y=(y_1, \ldots, y_n)$是两个序列，$LCS_{XY}=(z_1, \ldots, z_k)$是$X$与$Y$的LCS，我们有：

(1) 如果$x_m=y_n$, 则$z_k=x_m=y_n$, $LCS_{XY} = LCS_{X_{m-1}Y_{n-1}} + \langle x_m=y_n\rangle$,
$LCS_{X_{m-1}Y_{n-1}}$是X_{m-1}和Y_{n-1}的LCS.

(2) 如果$x_m\neq y_n$，且$z_k\neq x_m$，则LCS_{XY}是X_{m-1}和Y的LCS，即
$LCS_{XY} = LCS_{X_{m-1}Y}$

(3) 如果$x_m\neq y_n$,且$z_k\neq y_n$,则LCS_{XY}是X与Y_{n-1}的LCS，即
$LCS_{XY} = LCS_{XY_{n-1}}$

HIT CS&E

证明：

(1). $X=\langle x_1, \ldots, x_{m-1}, x_m\rangle$, $Y=\langle y_1, \ldots, y_{n-1}, x_m\rangle$，则

$LCS_{XY} = LCS_{X_{m-1}Y_{n-1}} + \langle x_m=y_n\rangle$.

设$z_k\neq x_m$，则可加$x_m=y_n$到Z，得到一个长为$k+1$的X与Y的公共序列，与Z是X和Y的LCS矛盾。于是$z_k=x_m=y_n$。

设存在X_{m-1}与Y_{n-1}的非最长公共子序列Z_{k-1}，使得

$LCS_{XY} = Z_{k-1} + \langle x_m=y_n\rangle$，

则由于$|Z_{k-1}|<|LCS_{X_{m-1}Y_{n-1}}|$,

$|LCS_{XY} = Z_{k-1} + \langle x_m=y_n\rangle| < |LCS_{XY} = LCS_{X_{m-1}Y_{n-1}} + \langle x_m=y_n\rangle|$,

与LCS_{XY}是LCS矛盾。

HIT CS&E

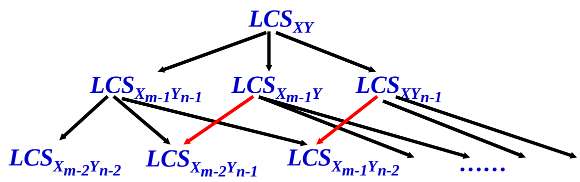
(2) $X=<x_1, \ldots, x_{m-1}, x_m>$, $Y=<y_1, \ldots, y_{n-1}, y_n>$，

$x_m \neq y_n$，$z_k \neq x_m$，则 $LCS_{XY}= LCS_{X_{m-1}Y}$

由于$z_k \neq x_m$，$Z=LCS_{XY}$是X_{m-1}与Y的公共子序列。我们来证Z是X_{m-1}与Y的LCS。设X_{m-1}与Y有一个公共子序列W，W的长大于k，则W也是X与Y的公共子序列，与Z是LCS矛盾。

(3) 证明同(2)。

- 子问题重叠性



LCS问题具有子问题重叠性

HIT CS&E

建立LCS长度的递归方程

- $C[i, j]$ = X_i与Y_j 的LCS的长度
- LCS长度的递归方程

$C[i, j] = 0$ if $i=0$ 或 $j=0$

$C[i, j] = C[i-1, j-1] + 1$ if $i, j>0$ and $x_i = y_j$

$C[i, j] = Max(C[i, j-1], C[i-1, j])$ if $i, j>0$ and $x_i \neq y_j$

HIT CS&E

递归划分与自底向上求解

- 基本思想

$C[i, j] = 0$

if $i=0$ 或 $j=0$

$C[i, j] = C[i-1, j-1] + 1$

if $i, j>0$ and $xi = yj$

$C[i, j] = Max(C[i, j-1], C[i-1, j])$

if $i, j>0$ and $xi \neq yj$

	C[i-1, j-1]	C[i-1,j]	
	C[i, j-1]	C[i, j]	

HIT CS&E

- 递归划分子问题与自底向上求解过程

C[0,0]	C[0,1]	C[0,2]	C[0,3]	C[0,4]
C[1,0]	C[1,1]	C[1,2]	C[1,3]	C[1,4]
C[2,0]	C[2,1]	C[2,2]	C[2,3]	C[2,4]
C[3,0]	C[3,1]	C[3,2]	C[3,3]	C[3,4]

HIT CS&E

- 计算LCS长度的算法
  - 数据结构

    $C[0:m,0:n]$: $C[i,j]$是X_i与Y_j的LCS的长度

    $B[1:m,1:n]$: $B[i,j]$是指针，指向计算$C[i,j]$时所选择的子问题的优化解所对应的C表的表项

算法实例

- $C[i, j] = 0$, $i=0$ 或 $j=0$
- $C[i, j] = C[i-1, j-1] + 1$, $x_i = y_j$
- $C[i, j] = Max(C[i, j-1], C[i-1, j])$, $x_i \neq y_j$

	y_j	B	D	C	A	B	A
x_i	0	0	0	0	0	0	0
A	0	↑0	↑0	↑0	↖1	←1	↖1
B	0	↖1	←1	←1	↑1	↖2	←2
C	0	↑1	↑1	↖2	←2	↑2	↑2
B	0	↖1	↑1	↑2	↑2	↖3	←3
D	0	↑1	↖2	↑2	↑2	↑3	↑3
A	0	↑1	↑2	↑2	↖3	↑3	↖4
B	0	↖1	↑2	↑2	↑3	↖4	↑4

LCS-length(X, Y)

m←length(X); n←length(Y);
For i←1 To m Do $C[i,0]$←0;
For j←1 To n Do $C[0,j]$←0;
For i←1 To m Do
  For j←1 To n Do
    If $x_i = y_j$
    Then $C[i,j]$←$C[i-1,j-1]+1$; $B[i,j]$←"↖";
    Else If $C[i-1,j] \geq C[i,j-1]$
      Then $C[i,j]$←$C[i-1,j]$; $B[i,j]$←"↑";
      Else $C[i,j]$←$C[i,j-1]$; $B[i,j]$←"←";
Return C and B.

HIT CS&E

构造优化解

- **基本思想**
  - 从$B[m, n]$开始按指针搜索
  - 若$B[i, j]$="↖", 则$x_i=y_j$是LCS的一个元素
  - 如此找到的"LCS"是X与Y的LCS的Inverse

©DB-LAB(2003)

HIT CS&E

Print-LCS(B, X, i, j)
IF $i=0$ or $j=0$ THEN Return;
IF $B[i, j]$="↖"
THEN Print-LCS(B, X, $i-1$, $j-1$); Print x_i;
ELSE If $B[i, j]$="↑"
    THEN Print-LCS(B, X, $i-1$, j);
    ELSE Print-LCS(B, X, i, $j-1$).

Print-LCS(B, X, length(X), length(Y))
可打印出X与Y的LCS。

HIT CS&E

算法复杂性

- **时间复杂性**
  - 计算代价的时间
    - (i, j)两层循环, i循环m步, j循环n步
    - $O(mn)$
  - 构造最优解的时间: $O(m+n)$
  - 总时间复杂性为: $O(mn)$
- **空间复杂性**
  - 使用数组C和B
  - 需要空间$O(mn)$

HIT CS&E

5.4 Optimal Polygon Triangulation

©DB-LAB(2003)

HIT CS&E

问题的定义

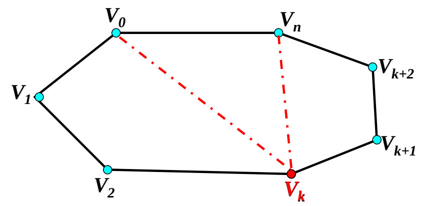
- **多边形**
  多边形表示为顶点坐标集 $P=(v_0,v_1,\ldots v_n)$ 或顶点序列 $v_0v_1\ldots v_{n-1}v_n$
- **简单多边形**
  除了顶点以外没有任何边交叉点的多边形
- **多边形的内部、边界与外部**
  – 平面上由多边形封闭的点集合称为多边形**内部**，
  – 多边形上的点集合称为多边形的**边界**
  – 平面上除多边形内部和边界以外的点集合称为多边形的**外部**

- **弦**
  多边形P上的任意两个不相邻结点 v_i、v_{i+1} 所对应的线段 v_iv_{i+1} 称为弦
- **三角剖分**
  一个多边形P的三角剖分是将P划分为不相交三角形的弦的集合
- **优化三角剖分问题**
  – 输入：多边形P和代价函数W
  – 输出：求P的三角剖分T，使得代价 $\sum_{s\in S_T}W(s)$ 最小，其中 S_T 是T所对应的三角形集合

HIT CS&E

优化解结构的分析

- 设
  – $P=(v_0,v_1,\ldots v_n)$ 是n+1个顶点的多边形
  – T_P 是P的优化三角剖分，包含三角形 $v_0v_kv_n$



  – $T_P=T(v_0, \ldots, v_k)\cup T(v_k, \ldots, v_n)\cup\{v_0v_k,\ v_kv_n\}$

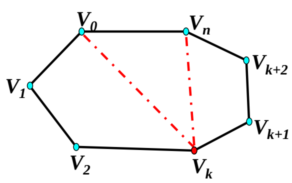
HIT CS&E

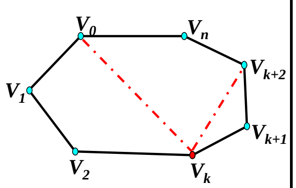
- 三角剖分问题具有优化子结构

  定理. 设 $P=(v_0,v_1,\ldots v_n)$ 是n+1个顶点的多边形. 如果 T_P 是P的包含三角形 $v_0v_kv_n$ 的优化三角剖分，即
  $T_P=T(v_0, \ldots, v_k)\cup T(v_k, \ldots, v_n)\cup\{v_0v_k,\ v_kv_n\}$,
  则
  (1). $T(v_0, \ldots, v_k)$ 是 $P_1=(v_0,v_1, \ldots,v_k)$ 的优化三角剖分，
  (2). $T(v_k, \ldots, v_n)$ 是 $P_2=(v_k,v_{k+1},\ldots,v_n)$ 的优化三角剖分。

HIT CS&E

- 三角剖分问题具有子问题重叠性





HIT CS&E

优化三角剖分的代价函数

- 设 $t[i,j] = < v_{i-1,}v_{i,\ldots,}v_j >$ 的优化三角剖分代价

$t[i, i] = t[j, j] = 0$

$t[i, j] = \min_{i\le k<j}\{t[i, k] + t[k+1, j] + w(\Delta_{v_{i-1}v_kv_j})\}$

注意：
$t[i, k] = <v_{i-1}, v_i, \ldots, v_k>$ 的优化三角剖分代价
$t[k+1, j] = <v_k, v_i, \ldots, v_j>$ 的优化三角剖分代价

HIT CS&E

优化三角剖分动态编程算法

- 优化三角剖分与矩阵链乘法问题一致.
- Homework 1:

  修改算法

  Matrix-chain-Order

  Print-Optimal-Parens

  使其计算$t[i,j]$并构造优化三角剖分解

HIT CS&E

5.5 0/1 Knapsack Problem

HIT CS&E

问题的定义

给定n种物品和一个背包，物品i的重量是w_i，价值v_i，背包承重为C，问如何选择装入背包的物品，使装入背包中的物品的总价值最大？

对于每种物品只能选择完全装入或不装入，一个物品至多装入一次。

HIT CS&E

- 输入：$C>0$, $w_i>0$, $v_i>0$, $1\le i\le n$
- 输出：$(x_1, x_2, \dots, x_n)$, $x_i\in\{0, 1\}$, 满足

  $\Sigma_{1\le i\le n} w_i x_i \le C$, $\Sigma_{1\le i\le n} v_i x_i$ 最大

等价的整数规划问题

$$\max \Sigma_{1\le i\le n} v_i x_i$$
$$\Sigma_{1\le i\le n} w_i x_i \le C$$
$$x_i\in\{0, 1\},\ 1\le i\le n$$

HIT CS&E

优化解结构的分析

定理（优化子结构）如果$(y_1, y_2, \dots, y_n)$是0-1背包问题的优化解，则$(y_2, \dots, y_n)$是如下子问题的优化解：

$$\max \Sigma_{2\le i\le n} v_i x_i$$
$$\Sigma_{2\le i\le n} w_i x_i \le C - w_1 y_1$$
$$x_i\in\{0, 1\},\ 2\le i\le n$$

证明：如果$(y_2, \dots, y_n)$不是子问题优化解，则存在$(z_2, \dots, z_n)$是子问题更优的解。于是，$(y_1, z_2, \dots, z_n)$是原问题比$(y_1, y_2, \dots, y_n)$更优解，矛盾。

HIT CS&E

建立优化解代价的递归方程

- 定义$m(i, j)$：

  背包容量为j，可选物品为$x_i, x_{i+1}, \dots, x_n$时，问题的最优解的代价是$m(i, j)$.
- 形式地

  问题 $\max \Sigma_{i\le k\le n} v_k x_k$

  $\Sigma_{i\le k\le n} w_k x_k \le j$

  $x_k\in\{0, 1\},\ i\le k\le n$

  的最优解代价为$m(i, j)$.

HIT CS&E

• 递归方程

$m(i, j) = m(i+1, j) \qquad 0 \le j < w_i$
$m(i, j) = \max\{m(i+1, j), m(i+1, j-w_i)+v_i\} \qquad j \ge w_i$

$m(n, j) = 0 \qquad 0 \le j < w_n$
$m(n, j) = v_n \qquad j \ge w_n$

HIT CS&E

自底向上计算优化解的代价

$m(i, j) = m(i+1, j),\ 0 \le j < w_i$
$m(i, j) = \max\{m(i+1, j), m(i+1, j-w_i)+v_i\},\ j \ge w_i$
$m(n, j) = 0,\ \ 0 \le j < w_n$
$m(n, j) = v_n,\ \ j \ge w_n$

	$m(i, j)$
$m(i+1, j-w_i)$	$m(i+1, j)$

令w_i=整数, $n=4$

$m(1,C)$

$m(2,C-w_1)$ … $m(2,C)$

$m(3, C-w_1-w_2)$ … $m(3,C-w_1)$ … $m(3,C)$

… $m(4, C-w_1-w_2-w_3)$ … $m(4, C-w_1-w_2)$ … $m(4,C-w_1)$ … $m(4, C)$

HIT CS&E

$m(i, j) = m(i+1, j),\ 0 \le j < w_i$
$m(i, j) = \max\{m(i+1, j), m(i+1, j-w_i)+v_i\},\ j \ge w_i$
$m(n, j) = 0,\ \ 0 \le j < w_n$
$m(n, j) = v_n,\ \ j \ge w_n$

	$m(i, j)$
$m(i+1, j-w_i)$	$m(i+1, j)$

令w_i=整数, $n=4$

$m(1,C)$

$m(2, 0)$ … $m(2,w_2-1)$ $m(2, w_2)$ … $m(2,C-1)$ $m(2,C)$

$m(3, 0)$ … $m(3,w_3-1)$ $m(3, w_3)$ … $m(3,C-1)$ $m(3,C)$

$m(4, 0)$ … $m(4,w_4-1)$ $m(4, w_4)$ … $m(4,C-1)$ $m(4,C)$

• 算法

```
For j=0 To min(w_n-1, C) Do
    m[n, j] = 0;
For j=w_n To C Do
    m[n, j] = v_n;
For i=n-1 To 2 Do
    For j=0 To min(w_i-1, C) Do
        m[i, j] = m[i+1, j];
    For j=w_i To C Do
        m[i, j]=max{m[i+1, j], m[i+1, j-w_i]+v_i};
If C<w_1
Then m[1, C]=m[2, C];
Else m[1, C]=max{m[2, C], m[2, C-w_1]+v_1};
```

HIT CS&E

构造优化解

1. $m(1, C)$是最优解代价值，相应解计算如下:
   If $m(1, C) = m(2, C)$
   Then $x_1 = 0$;
   Else $x_1 = 1$;
2. 如果 $x_1=0$, 由$m(2, C)$继续构造最优解;
3. 如果 $x_1=1$, 由$m(2, C-w_1)$继续构造最优解.

HIT CS&E

算法复杂性

- 时间复杂性
  - 计算代价的时间
    - $O(Cn)$
  - 构造最优解的时间: $O(Cn)$
  - 总时间复杂性为: $O(n)$
- 空降复杂性
  - 使用数组m
  - 需要空间$O(Cn)$

HIT CS&E

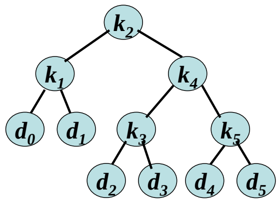
5.6 The Optimal Binary Search Trees

HIT CS&E

问题的定义

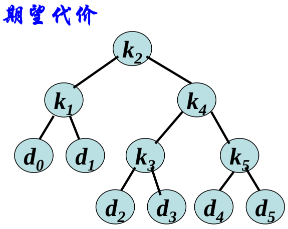
- 二叉搜索树T
  - 结点
    - $K=\{k_1, k_2, \ldots, k_n\}$
    - $D=\{d_0, d_1, \ldots, d_n\}$
    - d_i对应区间(k_i, k_{i+1})
      d_0对应区间$(-\infty, k_1)$
      d_n对应区间$(k_n, +\infty)$
  - 附加信息
    - 搜索k_i的概率为p_i
    - 搜索d_i的概率为q_i

$$\sum_{i=1}^{n} p_i + \sum_{j=0}^{n} q_j = 1$$



HIT CS&E

- 搜索树的期望代价



$$E(T)=\sum_{i=1}^{n}(DEP_T(k_i)+1)p_i+\sum_{j=0}^{n}(DEP_T(d_j)+1)q_j$$

HIT CS&E

- 问题的定义

输入: $K=\{k_1, k_2, \ldots, k_n\}$, $k_1< k_2< \ldots < k_n$,
$P=\{p_1, p_2, \ldots, p_n\}$, p_i为搜索k_i的概率
$Q=\{q_0, q_1, \ldots, q_n\}$, q_i为搜索值d_i的概率

输出: 构造K的二叉搜索树T, 使得

$$E(T)=\sum_{i=1}^{n}(DEP_T(k_i)+1)p_i+\sum_{j=0}^{n}(DEP_T(d_j)+1)q_j$$

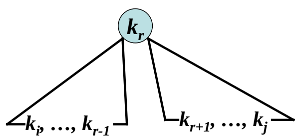
最小

HIT CS&E

优化二叉搜索树结构的分析

- 划分子问题

$K=\{k_i, k_{i+1}, \ldots, k_j\}$的优化解的根必为$K$中某个$k_r$



如果$r=i$, 左子树$\{k_i, \ldots, k_{i-1}\}$仅包含d_{i-1}
如果$r=j$, 右子树$\{k_{r+1}, \ldots, k_j\}$仅包含d_j

HIT CS&E

- 优化子结构

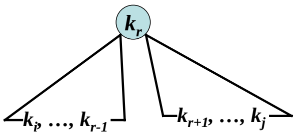
定理. 如果优化二叉搜索树T具有包含关键字集合$\{k_i, k_{i+1}, \ldots, k_j\}$的子树$T'$, 则$T'$是关于关键字集合$\{k_i, k_{i+1}, \ldots, k_j\}$的子问题的优化解.

证明: 若不然, 必有关键字集$\{k_i, k_{i+1}, \ldots, k_j\}$子树$T''$, T''的期望搜索代价低于T'.
用T''替换T中的T', 可以得到一个期望搜索代价比T小的原始问题的二叉搜索树。
与T是最优解矛盾.

HIT CS&E

- 用优化子结构从子问题优化解构造优化解

$K=\{k_i, k_{i+1}, \ldots, k_j\}$的优化解的根必为$K$中某个$k_r$



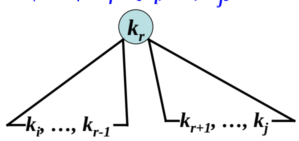
只要对于每个$k_r \in K$，确定$\{k_i, \ldots, k_{r-1}\}$和$\{k_{r+1}, \ldots, k_j\}$的优化解，我们就可以求出K的优化解。

如果$r=i$，左子树$\{k_i, \ldots, k_{i-1}\}$仅包含d_{i-1}
如果$r=j$，右子树$\{k_{r+1}, \ldots, k_j\}$仅包含d_j

HIT CS&E

建立优化解的搜索代价递归方程

- 令$E(i, j)$为$\{k_i, \ldots, k_j\}$的优化解T_{ij}的期望搜索代价
  - 当$j=i-1$时，T_{ij}中只有叶结点d_{i-1}，$E(i, i-1)=q_{i-1}$
  - 当$j \geq i$时，选择一个$k_r \in \{k_i, \ldots, k_j\}$：



当把左右优化子树放进T_{ij}时，每个结点的深度增加1

$E(i, j)=P_r + E(左子树) + W(i, r-1) + E(右子树) + W(r+1, j)$

- 计算$W(i, r-1)$和$W(r+1, j)$

由 $$E(LT+1)=\sum_{l=i}^{r-1}(DEP_{左}(k_l)+2)p_l+\sum_{l=i-1}^{r-1}(DEP_{左}(d_l)+2)q_l$$

$$E(LT)=\sum_{l=i}^{r-1}(DEP_{左}(k_l)+1)p_l+\sum_{l=i-1}^{r-1}(DEP_{左}(d_l)+1)q_l$$

知 $$W(i,r-1)=E(LT+1)-E(LT)=\sum_{l=i}^{r-1}p_l+\sum_{l=i-1}^{r-1}q_l$$

同理，$$W(r+1,j)=\sum_{l=r+1}^{j}p_l+\sum_{l=r}^{j}q_l$$

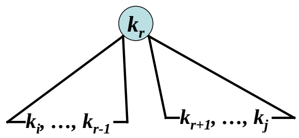
令 $$W(i, j)=W(i, r-1) + W(r+1, j) + p_r=\sum_{l=i}^{j}p_l+\sum_{l=i-1}^{j}q_l$$

$$=W(i, j-1)+p_j+q_j$$

$W(i, i-1)=q_{i-1}$

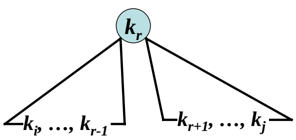
$W(i, j)=W(i, r-1) + W(r+1, j) + p_r =W(i, j-1)+p_j+q_j$

$E(i, j)=P_r + E(左子树) + W(i, r-1) + E(右子树) + W(r+1, j)$



$E(i, j) = E(i, r-1) + E(r+1, j) + W(i, j)$

HIT CS&E



总之

$E(i, j)=q_{i-1}$ If $j=i-1$
$E(i, j)=\min_{i \leq r \leq j}\{E(i,r-1)+E(r+1,j)+W(i, j)\}$ If $j \geq i$

$W(i, i-1) = q_{i-1}$, $W(i, j) = W(i, j-1) + p_j + q_j$

HIT CS&E

自下而上计算优化解的搜索代价

$E(i, j)=q_{i-1}$ If $j=i-1$
$E(i, j)=\min_{i \leq r \leq j}\{E(i,r-1)+E(r+1,j)+W(i, j)\}$ If $j \geq i$

q_0 =	E(1,0)	E(1,1)	E(1,2)	E(1,3)	E(1,4)
	q_1 =	E(2,1)	E(2,2)	E(2,3)	E(2,4)
		q_2 =	E(3,2)	E(3,3)	E(3,4)
			q_3 =	E(4,3)	E(4,4)
				q_4 =	E(5,4)

HIT CS&E

- $W(i, i\text{-}1) = q_{i\text{-}1},\ \ W(i, j) = W(i, j\text{-}1) + p_j + q_j$

$q_0 =$	W(1,0)	W(1,1)	W(1,2)	W(1,3)	W(1,4)
	$q_1 =$	W(2,1)	W(2,2)	W(2,3)	W(2,4)
		$q_2 =$	W(3,2)	W(3,3)	W(3,4)
			$q_3 =$	W(4,3)	W(4,4)
				$q_4 =$	W(5,4)

HIT CS&E

- 算法
  - 数据结构
    - $E[1{:}n{+}1;\ 0{:}n]$: 存储优化解搜索代价
    - $W[1{:}\ n{+}1;\ 0{:}n]$: 存储代价增量
    - $Root[1{:}n;\ 1{:}n]$: $Root(i, j)$记录子问题 $\{k_i, \ldots, k_j\}$优化解的根

HIT CS&E

Optimal-BST(*p*, *q*, *n*)

```
For i=1 To n+1 Do
    E(i, i-1) = q_{i-1};
    W(i, i-1) = q_{i-1};
For l=1 To n Do
    For i=1 To n-l+1 Do
        j=i+l-1;
        E(i, j)=∞;
        W(i, j)=W(i, j-1)+p_j+q_j;
        For r=i To j Do
            t=E(i, r-1)+E(r+1, j)+W(i, j);
            If t<E(i, j)
            Then E(i, j)=t; Root(i, j)=r;
Return E and Root
```

HIT CS&E

算法的复杂性

- 时间复杂性
  - (l, i, r)三层循环，每层循环至多n步
  - 时间复杂性为$O(n^3)$
- 空间复杂性
  - 二个$(n+1)\times(n+1)$数组，一个$n\times n$数组
  - $O(n^2)$

HIT CS&E

Homework 2: 优化解的构造算法

HIT CS&E

第六章
Greedy Algorithm

李建中
计算机科学与工程系

HIT CS&E

提要

HIT CS&E

参考资料

Chapter 5

Pages 138-184

HIT CS&E

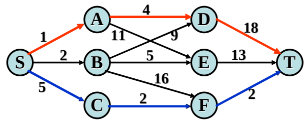
6.1 *Elements of Greedy Algorithms*

- *Greedy*算法的基本概念
- *Greedy*算法与动态规划方法的比较
- *Greedy*算法的设计方法

HIT CS&E

*Greedy*算法的基本概念

- **Greedy算法的实例**
  - 最短路径问题



  - Greedy求解过程

Greedy算法不一定产生正确解

HIT CS&E

- **Greedy算法的基本思想**
  - 求解最优化问题的算法包含一系列步骤
  - 每一步都有一组选择
  - 作出在当前看来最好的选择
  - 希望通过作出局部优化选择达到全局优化选择
  - Greedy算法不一定总产生优化解
- **Greedy算法产生优化解的条件**
  - Greedy-choice-property
  - Optimal substructure

HIT CS&E

- Greedy选择性
  若一个优化问题的全局优化解可以通过局部优化选择得到，则该问题称为具有Greedy选择性.
- 一个问题是否具有Greedy选择性需证明

HIT CS&E

- 优化子结构
  若一个优化问题的优化解包含它的子问题的优化解，则称其具有优化子结构

HIT CS&E

与动态规划方法的比较

- 动态规划方法可用的条件
  - 优化子结构
  - 子问题重叠性
- Greedy方法可用的条件
  - 优化子结构
  - Greedy选择性
- 可用Greedy方法时，动态规划方法可能不适用
- 可用动态规划方法时，Greedy方法可能不适用

HIT CS&E

准确Greedy算法的设计方法

1. 设计贪心选择方法：
   - 贪心选择方法
   - 剩余子问题
2. 证明：对于1中贪心选择来说，所求解的问题具有优化子结构
3. 证明：对于1中贪心选择算法来说，所求解的问题具有Greedy选择性
4. 按照1中设计的贪心选择方法设计算法

HIT CS&E

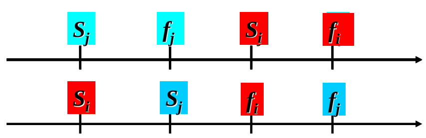
6.2 *An activity-selection problem*

- 问题的定义
- 设计贪心选择方法
- 优化解的结构分析
- Greedy选择性
- 算法设计
- 算法复杂性分析

HIT CS&E

问题的定义

- 活动
  - 设$S=\{1,2,\ldots,n\}$是n个活动的集合，所有活动共享一个资源，该资源同时只能为一个活动使用
  - 每个活动i有起始时间s_i，终止时间f_i，$s_i \le f_i$
- 相容活动
  - 活动i和j是相容的，若$s_j \ge f_i$或$s_i \ge f_j$，即



HIT CS&E

- 问题定义
  - 输入：$S=\{1, 2, \ldots, n\}$，
    $F=\{ [s_i,\ f_i] \}$，$n \geq i \geq 1$
  - 输出：S的最大相容集合

HIT CS&E

设计贪心选择方法

- 贪心思想

  为了选择最多的相容活动，每次选f_i最小的活动，使我们能够选更多的活动

- 贪心选择方法
  - 选择：
    - 每次选择具有最小结束时间的活动f_i
  - 剩余子问题的结构：
    - $S_i=\{ j \in S \mid s_j \geq f_i \}$

HIT CS&E

优化解结构分析

引理1 设$S=\{1,2,\ldots,n\}$是n个活动集合，$[s_i,\ f_i]$是活动的起始终止时间，且$f_1 \leq f_2 \leq \ldots \leq f_n$，S的活动选择问题的某个优化解包括活动1.

证 设A是一个优化解，按结束时间排序A中活动，设其第一个活动为k，第二个活动为j.

如果$k=1$，引理成立.

如果$k \neq 1$，令$B=A-\{k\} \cup \{1\}$,

由于A中活动相容，$f_1 \leq f_k \leq s_j$, B中活动相容.

因为$|B|=|A|$, 所以B是一个优化解，且包括活动1.

定理1. 设$S=\{1, 2, \ldots, n\}$是n个活动集合，$[s_i,\ f_i]$是活动i的起始终止时间，且$f_1 \leq f_2 \leq \ldots \leq f_n$，设A是S的调度问题的一个优化解且包括活动1，则$A'=A-\{1\}$是$S'=\{i \in S | s_i \geq f_1\}$的调度问题的优化解.

证. 显然，A'中的活动是相容的.

我们仅需要证明A'是最大的.

定理1说明活动选择问题具有优化子结构

由于$|A|=|A'|+1$, $|B|=|B'|+1>|A'|+1=|A|$，与A最大矛盾.

HIT CS&E

Greedy选择性

定理2. 设 $S=\{1, 2, \ldots, n\}$ 是 n 个活动集合，$f_{l_0}=0$, l_i 是 $S_i=\{ j \in S \mid s_j \geq f_{l_{i-1}}\}$ 中具有最小结束时间 f_{l_i} 的活动. 设A是S的包含活动1的优化解, 其中 $f_1 \leq f_2 \leq \ldots \leq f_n$，则$A=\bigcup_{i=l_1}^{l_k}\{i\}$

$S_1=\{ j \in S \mid s_j \geq f_{l_0}=0\}$
$S_2=\{ j \in S \mid s_j \geq f_{l_1}=f_1\}$
$S_3=\{ j \in S \mid s_j \geq f_{l_2}\}$
……
$S_k=\{ j \in S \mid s_j \geq f_{l_{k-1}}\}$
$S_{k+1}=\{ j \in S \mid s_j \geq f_{l_k}\}=\theta$

证. 对$|A|$作归纳法.

当$|A|=1$时, 由引理1, 命题成立.

设$|A| \leq k-1$时，命题成立.

当$|A|=k$时, 由定理1，$A=\{l_1=1\} \cup A_1$.

A_1是 $S_2=\{ j \in S \mid s_j \geq f_{l_1}= f_1 \}$ 的优化解.

由归纳假设，$A_1=\bigcup_{i=l_2}^{l_k}\{i\}$. 于是，$A=\bigcup_{i=l_1}^{l_k}\{i\}$.

HIT CS&E

算法的设计

- 贪心选择方法
  - 选择：
    - 每次选择具有最小结束时间的活动f_i
  - 剩余子问题的结构：
    - $S_i=\{ j \in S \mid s_j \geq f_i \}$

HIT CS&E

- 算法

  （设 $f_1 \le f_2 \le \ldots \le f_n$ 已排序）

  Greedy-Activity-Selector(*S*, *F*)

  n←**lenyth(S);**

  A←**{1}**

  j←**1**

  For ***i***←***2*** **To** ***n*** **Do**

  If $s_i \ge f_j$

  Then $A \leftarrow A \cup \{i\}$; $j \leftarrow i$;

  Return ***A***

HIT CS&E

复杂性设计

- 如果结束时间已排序

  $T(n)=\theta(n)$

- 如果结束时间未排序

  $T(n)=\theta(n)+\theta(nlogn)=\theta(nlogn)$

HIT CS&E

算法正确性证明

定理3. Greedy-Activity-Selector算法能够产生最优解.

证. Greedy-Activity-Selector算法按照引理3的Greedy选择性进行局部优化选择.

HIT CS&E

6.3 Huffman codes

- 问题的定义
- 设计贪心选择方法
- 优化解的结构分析
- 算法设计
- 算法复杂性分析

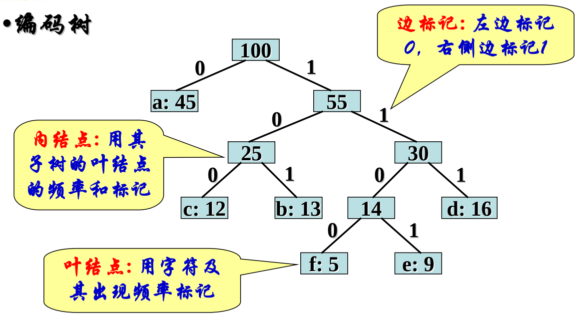
HIT CS&E

问题的定义

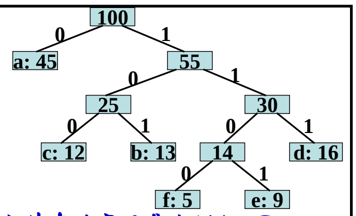
- 二进制字符编码
  - 每个字符用一个二进制0、1串来表示.
- 固定长编码
  - 每个字符都用相同长的0、1串表示.
- 可变长编码
  - 经常出现的字符用短码，不经常出现的用长码
- 前缀编码
  - 无任何字符的编码是另一个字符编码的前缀

HIT CS&E

- 编码树



HIT CS&E



- 编码树T的代价
  - 设C是字母表(给定文件中的字母集合), $\forall c \in C$
  - $f(c)$是c在文件中出现的频数
  - $d_T(c)$是叶子c在树T中的深度，即c的编码长度
  - T的代价是编码一个文件的所有字符的代码位数:

$$B(T) = \sum_{c \in C} f(c) d_T(c)$$

HIT CS&E

- 优化编码树问题

输入: 字母表 $C = \{c_1, c_2, \ldots, c_n\}$,

频数表 $F = \{f(c_1), f(c_2), \ldots, f(c_n)\}$

输出: 具有最小$B(T)$的C前缀编码树

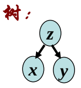
贪心思想:

循环地选择具有最低频数的两个结点，生成一棵子树，直至形成树

HIT CS&E

设计贪心选择方法

- 贪心选择方法
  - 选择方法:
    - 每次选择具有最低频数的两个节点x和y，构造一个子树:

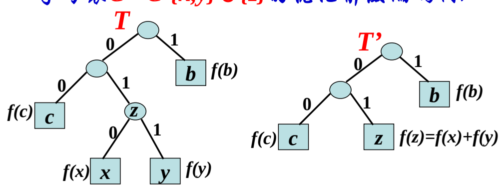


  - 剩余子问题的结构:
    - $C' = C - \{x, y\} \cup \{z\}$

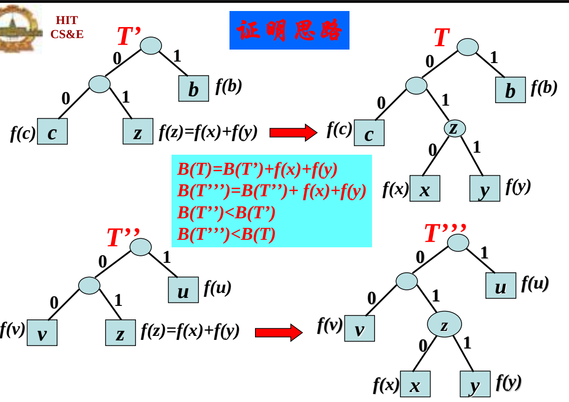
HIT CS&E

优化解的结构分析

定理1. 设T是字母表C的优化前缀树，$\forall c \in C$，$f(c)$是c在文件中出现的频数. 设x、y是T中任意两个相邻叶结点，z是它们的父结点，则z作为频数是$f(z)=f(x)+f(y)$的字符，$T'=T-\{x,y\}$是字母表$C'=C-\{x,y\}\cup\{z\}$的优化前缀编码树.



HIT CS&E



证. 往证$B(T)=B(T')+f(x)+f(y)$.

$B(T)= \Sigma_{c\in C} f(c)d_T(c)=f(x)d_T(x)+f(y)d_T(y)+\Sigma_{c\in C'-\{z\}} f(c)d_T(c)$

由于 $d_T(x)=d_T(y)=d_{T'}(z)+1$

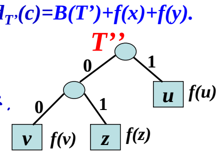
$= f(x)(d_{T'}(z)+1)+f(y)(d_{T'}(z)+1)+\Sigma_{c\in C'-\{z\}} f(c)d_{T'}(c)$

$= f(x)+f(y)+f(x)d_{T'}(z)+f(y)d_{T'}(z)+\Sigma_{c\in C'-\{z\}} f(c)d_{T'}(c)$

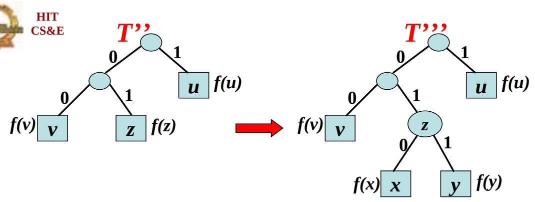
由于$f(x)+f(y)=f(z)$

$= f(x)+f(y)+f(z)d_{T'}(z) +\Sigma_{c\in C'-\{z\}} f(c)d_{T'}(c)=B(T')+f(x)+f(y)$.

若T'不是C'的优化前缀编码树，
则必存在T''，使$B(T'')<B(T')$.
因为z是C'中字符，它必为T''中的叶子.
把结点x与y加入T''，作为z的子结点，
则得到C的一个如下前缀编码树T''':



HIT CS&E



如上可证：

$B(T''') = B(T'')+f(x)+f(y)$。

由于$B(T'')<B(T')$，

$B(T''') = B(T'')+f(x)+f(y) < B(T')+f(x)+f(y)=B(T)$

与T是优化的矛盾，故T'是C'的优化编码树.

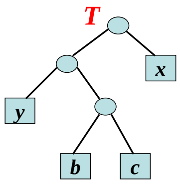
HIT CS&E

Greedy选择性

定理2. 设C是字母表，$\forall c \in C$，c具有频数$f(c)$, x、y是C中具有最小频数的两个字符，则存在一个C的优化前缀树，x与y的编码具有相同最大长度，且仅在最末一位不同.

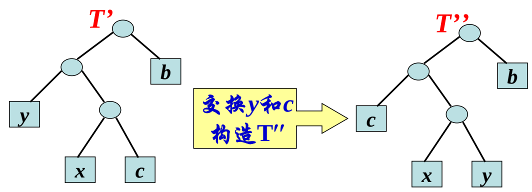
证：若T是C的优化前缀树，如果x和y是具有最大深度的两个兄弟字符，则命题得证。

若不然，设b和c是具有最大深度的两个兄弟字符：

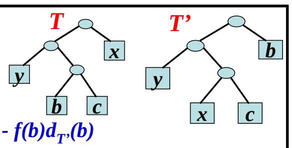


不失一般性，设$f(b) \leq f(c)$, $f(x) \leq f(y)$. 因x与y是具有最低频数的字符，$f(b) \geq f(x)$, $f(c) \geq f(y)$. 交换T的b和x，从T构造T'：

HIT CS&E



往证T''是最优化前缀树.



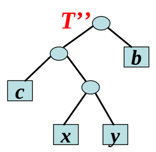
$B(T)-B(T')$

$= \sum_{c \in C} f(c)d_T(c) - \sum_{c \in C} f(c)d_{T'}(c)$

$= f(x)d_T(x) + f(b)d_T(b) - f(x)d_{T'}(x) - f(b)d_{T'}(b)$

$= f(x)d_T(x) + f(b)d_T(b) - f(x)d_T(b) - f(b)d_T(x)$

$= (f(b)-f(x))(d_T(b)-d_T(x))$.



$\because f(b) \geq f(x)$, $d_T(b) \geq d_T(x)$ (因为b的深度最大)

$\therefore B(T)-B(T') \geq 0$, $B(T) \geq B(T')$

同理可证$B(T') \geq B(T'')$. 于是$B(T) \geq B(T'')$.

由于T是最优化的，所以$B(T) \leq B(T'')$.

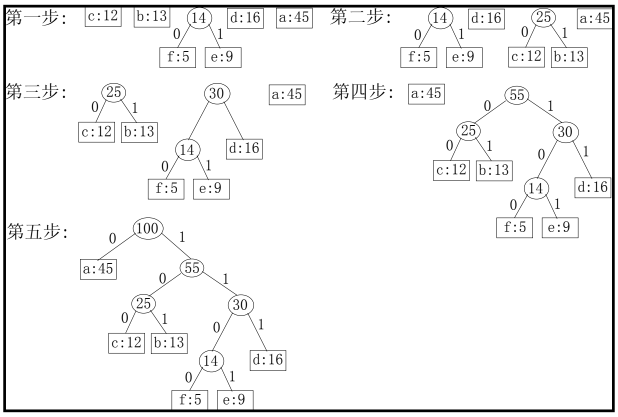
于是，$B(T)=B(T'')$，T''是C的最优化前缀编码树.

在T''中，x和y具有相同最大长度编码，且仅最后位不同.

HIT CS&E

算法的设计

- 基本思想
  - 循环地选择具有最低频数的两个结点，生成一棵子树，直至形成树
  - 例子：f:5, e:9, c:12, b:13, d:16, a:45



- Greedy算法 (Q是min-heap，见第六章)

```
Huffman(C, F)
n←|C|; Q←根据F排序C; T为一个空树;
FOR i←1 To n-1 Do
    z←Allocate-Node( );
    left[z]← x ←Extract-min(Q) /*从Q删除x*/;
    right[z]← y ←Extract-min(Q) /*从Q删除y */;
    f(z)←f(x)+f(y);
    Insert(Q, z, f(z));
Return Extract-min(Q)  /* 返回树的根 */
```

HIT CS&E

复杂性分析

- 第1步：$O(n\log n)$
- 每次循环：$O(\log n)$，循环n-1次：$O(n\log n)$
- $T(n)=O(n\log n)$

HIT CS&E

正确性证明

定理3. Huffman算法产生一个优化前缀编码树

证. 由于定理1和定理2成立，而且Huffman算法按照定理2的Greedy选择性确定的规则进行局部优化选择，所以Huffman算法产生一个优化前缀编码树。

HIT CS&E

6.4 Minimal spanning tree problem

- 问题的定义
- 设计贪心选择法
- 优化解结构分析
- Greedy选择性
- Kruskal算法
- 算法复杂性

HIT CS&E

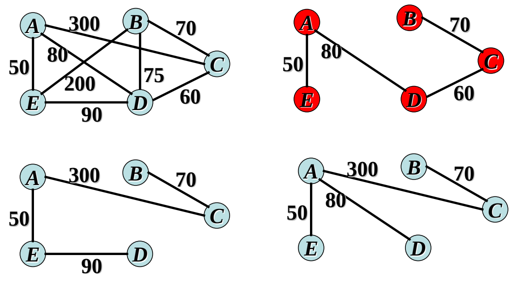
问题的定义

- 生成树
  - 设$G=(V, E)$是一个边加权无向连通图。G的生成树是无向树$T=(V, E_T)$, $E_T\subseteq E$.
  - 如果 $W: E\rightarrow\{实数\}$ 是G的权函数，T的权值定义为$W(T)=\sum_{(u,v)\in T}W(u,v)$.
- 最小生成树
  - G的最小生成树是W(T)最小的G之生成树.
- 问题的定义

  输入：无向连通图$G=(V, E)$，权函数W

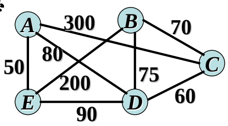
  输出：G的最小生成树

• 实例



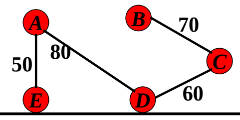
设计贪心选择方法

• 基本思想



设A是一个最小生成树的边子集合，如果A∪{(u, v)}也是一个最小生成树的边子集合，则(u, v)称为对A是安全的.

- 初始：A=空；构造森林G_A=(V, A);
- 循环：选择连接G_A中两棵树的最小安全边加入A，直至所有边都考察完.



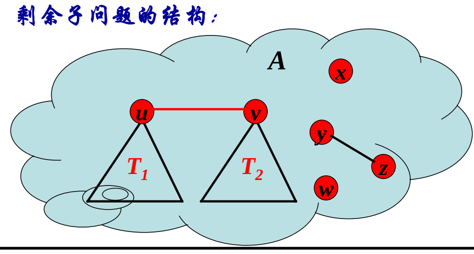
HIT CS&E

- 一般算法的定义

Generic-MST(G, W)
1. A=∅;
2. While A 不是生成树 Do
3. 寻找一个最小安全边(u, v);
2. A=A∪{ (u, v) };
3. Return A

HIT CS&E

- 贪心选择方法
  - 贪心选择：从森林中选择一条最小安全边连接两棵子树为一棵子树，直至森林成为一棵树
  - 剩余子问题的结构：

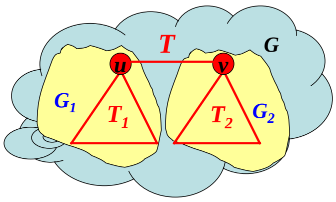


HIT CS&E

优化解的结构分析

定理1. 设T是G的最小生成树. 如果T包含子树T_1和T_2, T_1是G的子连通图G_1的生成树，T_2是G的子连通图G_2的生成树，则T_1是G_1的最小生成树，T_2是G_2的最小生成树.

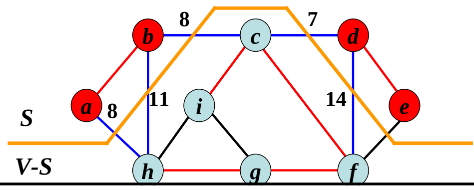
证.



HIT CS&E

Greedy选择性

定义1. 无向图G=(V, E)的一个划分是V的划分(S, V-S).
定义2. 若u∈S, v∈V-S, 则(u, v)称为划分(S, V-S)的交叉边.
定义3. 如果边集合A中没有边是划分(S, V-S)的交叉边，则称划分(S, V-S)尊重A.
定义4. 划分(S, V-S)的交叉边(u, v)称为轻边，如果在所有(S, V-S)的交叉边中，(u, v)的权值最小.



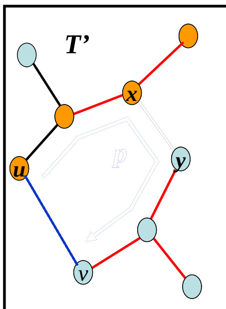
HIT CS&E

定理2. 设$G=(V, E)$是具有边加权函数W的无向连通图，$A\subseteq E$是包含在G的某个最小生成树中的边集合，$(S,V-S)$是G的尊重A的任意划分，(u, v)是$(S,V-S)$的交叉轻边，则(u, v)对A是安全的.

证. 令T是包含A的最小生成树.

如果(u, v)属于T，则(u, v)对A是安全的.

设(u, v)不属于T. 我们构造一个G的最小生成树T'，使其包含$A\cup\{(u, v)\}$，从而证明(u, v)安全.



- S=黄结点集合
- $V-S$=浅蓝结点集合
- A=红边集合

由于u和v在划分$(S, V-S)$的两边，T至少存在一条交叉边在从u到v的路径p中，设为(x, y).

由于划分尊重A, (x, y)不在A中. 删除p中的(x, y), 增加(u, v), 得到

$T'=T - \{(x, y)\} \cup\{(u, v)\}$.

往证T'是最小生成树.

因为(u, v)是轻交叉边，(x, y)是交叉边，$W(u, v)\leq W(x, y)$.

$W(T')=W(T)-W(x,y)+W(u,v)\leq W(T)$

由于T是最小生成树, $W(T')=W(T)$.

T'是最小生成树, $A\cup\{(u,v)\}\subseteq T'$, (u, v)对于A是安全的.

HIT CS&E

推论1. 设$G=(V, E)$是具有边加权函数W的无向连通图，$A\subseteq E$是包含在G的某个最小生成树中的边集合，$C=(V_C, E_C)$是森林$G_A=(V, A)$中的树. 如果(u, v)是连接C和G_A中另一个树的交叉轻边，则(u, v)对A是安全的.

证. 划分$(V_C, V-V_C)$尊重A，因为A的边要么在C中，要么在$G_A=(V, A)$的另一个树中.

(u, v)是关于这个划分的交叉轻边.

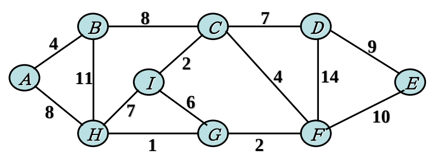
于是, (u, v)对A是安全的.

HIT CS&E

Kruskal算法

- 基本思想

  找到连接森林$G(V, A)$中两棵树的安全轻边加入A，直至$G(V, A)$成为一棵树。
- 实例



HIT CS&E

Kruskal算法

MST-Kruskal(G, W)
1. $A=\vartheta$;
2. For $\forall v\in V[G]$ Do
3. Make-Set(v); /* 建立只有v的集合 */
4. 按照W值的递增顺序排序$E[G]$;
5. For $\forall(u, v) \in E[G]$ (按W值的递增顺序) Do
6. If Find-Set(u)$\neq$Find-Set(v)
7. Then $A=A\cup\{(u, v)\}$; Union(u, v);
8. Return A

HIT CS&E

算法复杂性

- 令$n=|V|$, $m=|E|$
- 第4步需要时间: $O(m\log m)$
- 第2-3步执行$O(n)$个Make-Set操作

  第5-8步执行$O(m)$个Find-Set和Union操作

  需要时间: $O((n+m)\alpha(n))$
- $m\geq n-1$(因为G连通), 由$\alpha(n)<\log n<\log m$
- 总时间复杂性: $O(m\log m)$

集合操作的复杂性见Intro. To Algo. 第21章 (498-509)

HIT CS&E

算法正确性

定理2. MST-Kruskal(G, W)算法能够产生图G的最小生成树.

证. 因为算法按照Greedy选择性进行局部优化选择，并且每次选择的都是最小边.

HIT CS&E

6.5 Theoretical foundations of Greedy Algorithms

- Matroid (拟阵)
- Matroid 实例
- Matroid的性质
- 加权Matroid上的Greedy算法
- 算法的正确性

HIT CS&E

Matroid（拟阵）

- Matroid定义

Matroid是一个序对$M=(S, I)$，满足:

(1) S是一个有限非空集合.

(2) I是非空的S子集的集族，I中的子集称为S的独立子集.

(3) 遗传性: 如果$A\in I$，$B\subseteq A$，则$B\in I$

(4) 交换性: 如果$A\in I$，$B\in I$，$|A|<|B|$，则$\exists x\in B-A$使得$A\cup\{x\}\in I$.

HIT CS&E

- 实例(Graphic Matroid)

定义1. 设 $G = (V, E)$ 是一个无向图，由 G 确定的$M_G=(S_G, I_G)$定义如下: S_G是G的边集合E，$I_G=\{A\,|\,A\subseteq E, G_A(V, A)$是森林$\}$.

定理1. 如果G是一个无向图，则$M_G=(S_G, I_G)$是一个Matroid.

证. ① $S_G=E$是一个有限集合.

② $\forall e\in E$, $G_e(V, \{e\})$是一个森林，$\{e\}\in I_G$ 于是，I_G是S_G的非空集族.

③ M_G满足遗传性:

如果$A\in I_G$，$B\subseteq A$，则$G_A(V, B)$是一个森林. 于是，$A\in I_G$，M_G满足遗传性.

④ M_G满足交换性:

设$A\in I_G$, $B\in I_G$, $|A|<|B|$.

图论定理：具有k条边的森林具有$|V|-k$棵树.

$G_A=(V, A)$和$G_B=(V, B)$分别具有$|V|-|A|$和$|V|-|B|$棵树.

由于$|V|-|A|>|V|-|B|$, G_B中必存在树T, T的节点在G_A的不同树中.

由于T是连通的, T必包含边(u, v)：在G_A的不同树中.

于是, $(V, A\cup\{(u,v)\})$是森林, $A\cup\{(u,v)\}\in I_G$, $(u,v)\in B-A$, M_G满足交换性.

- Matroid的性质

定义2. 设$M=(S, I)$是一个Matroid, $A\in I$. 如果$A\cup\{x\}\in I$, $x\notin A$, x称为A的一个extension.

定义3. 设$M=(S, I)$是Matroid, $A\in I$. 若A没有extension, 则称A为最大独立子集合.

定理2. 一个Matroid的所有最大独立子集合都具有相同大小.

证. 设A是Matroid M的最大独立子集合，而且存在M的另一个独立子集合B, $|A|<|B|$.

根据M的交换性, $\exists x\in B-A$使$A\cup\{x\}\in I$, 与A最大矛盾.

HIT CS&E

加权Matroid上的Greedy算法

- Matroid的优化子集

实际背景:
很多可用Greedy算法获得最优解的问题可以归结为在加权Matroid中寻找优化子集的问题，即给定$M=(S, I)$和权函数W，找到独立子集$A\in I$，使得$W(A)$最大。

HIT CS&E

- 实例1: 最大生成森林问题
  - 问题定义
    输入：无向图$G=(V, E)$，权函数$W: E\rightarrow$正整数集
    输出：边子集合$A\subseteq E$, $G_A(V, A)$是森林，$W(A)$最大
  - 转换为加权Matroid上寻找优化子集问题
    - 构造：
      $M_G=(S_G,I_G)$, $S_G=E$, $I_G=\{A\,|\,A\subseteq E, G_A(V,A)$是森林$\}$, M_G是拟阵
    - M_G的最优子集A满足：
      - (V, A)是森林
      - $W(A)$最大

HIT CS&E

- 实例2: 最小生成森林问题
  - 问题定义
    输入：无向图$G=(V, E)$，权函数$W: E\rightarrow$正整数集
    输出：边子集合$A\subseteq E$, $G_A(V, A)$是森林，$W(A)$最小
  - 最大转换为加权Matroid上寻找优化子集问题
    - 构造：
      $M_G=(S_G,I_G)$, $S_G=E$, $I_G=\{A\,|\,A\subseteq E, G_A(V,A)$是森林$\}$，$M_G$是拟阵
    - 构造权函数W'
      - $W'(e)=W_0-W(e)$，$W_0>\max\{W(e)\}$
      - $\forall e\in E$，$W'(e)>0$.
      - W'扩展为$W'(A)=|V|W_0-W(A)$，$\forall A\subseteq E$
    - M_G的最优子集A满足：
      - (V, A)是森林
      - $W'(A)$最大，即$W(A)$最小.

HIT CS&E

- 加权Matroid优化子集问题的定义
  输入：Matroid $M=(S, I)$, 函数$W: S\rightarrow$正数集
  输出：M的独立子集$A\in I$, 使得$W(A)$最大

- 算法

```
Greedy(M, W)
1  A=Φ;
2  按权W值递减排序S;
3  For ∀x∈S（按W(x)递减顺序）DO
4      If  A∪{x}∈I    /* 选择目前W(x)最大的x */
5      Then A=A∪{x};
6  Return A.
```

- 时间复杂性
  step 2: $O(|S|\log|S|)$
  step 4: $O(f(|S|))$
  $T(|S|) = O(\,|S|\log|S|+|S|\,f(|S|)\,)$

- 优化子结构

定理1. 设x是第一个被Greedy算法选中的元素. 包含x的优化子集A包含子问题$M'=(S', I')$的优化子集$A'=A-\{x\}$, $M'=(S', I')$定义如下:
$S'=S-\{y|\ W(y)<W(x)\}-\{x\}$
$I'=\{B\subseteq S'\ |\ B\cup\{x\}\in I\}$
而且M'的权函数与M的权函数相同.

证. 因为$A=A'\cup\{x\}\in I$, 所以$A'=A-\{x\}\in I'$.
若A'不是M'的优化子集, 则存在M'的一个优化子集B使得$W(B)>W(A')$.
由于$B\cup\{x\}\in I$, $W(A)=W(A')+W(x)$,
$W(B\cup\{x\})=W(B)+W(x)>W(A')+W(x)=W(A)$,
与A优化矛盾.

- **Greedy选择性**

定理2. 设$M=(S, I)$是一个加权Matroid, W是M的权函数, S按W值递减排序. 设x是S中第一个满足$\{x\}\in I$的元素, 则存在一个M的优化子集A, $x\in A$.

证. 若存在优化子集A包含x, 则引理得证.
否则, 设B是任意非空优化子集, $x\notin B$.
S中x之前的元素y必不在B中, 否则由遗传性, $\{y\}\in I$,
与"x是S中第一个满足$\{x\}\in I$的元素"矛盾.
显然, $\forall z\in B$, $W(x)\geq W(z)$. 如下构造含x的优化子集A:
初始: $A=\{x\}\in I$;
用交换性: $\forall z\in B-A$, 若$A\cup\{z\}\in I$, $A=A\cup\{y\}$, 直至$|A|=|B|$.
显然, $\exists w\in B$, $A=(B-\{w\})\cup\{x\}$.
于是, $W(A)=W(B)-W(w)+W(x)\geq W(B)$.
因为B是优化子集, 所以$W(A)\leq W(B)$, $W(A)=W(B)$.
A是优化子集, 且$x\in A$.

HIT CS&E

- **算法正确性**

引理1. Greedy算法返回一个独立子集合.

证. 设Greedy返回集合A, 对$|A|$做数学归纳法.
当$|A|=0$时, A是空集, 由遗传性, A是独立子集合.
设$|A|\leq k$时, A是独立子集.
当$|A|=k+1$时, A由第4-5步得到, 即$A=A\cup\{x\}$.
第4步已判定$A\cup\{x\}\in I$, $A=A\cup\{x\}$是独立子集.

HIT CS&E

引理2. 设$M=(S, I)$是一个Matroid. 如果$x\in S$不是空集Φ的extension, 则x不是任何独立子集的extension.

证. 反证. 设$x\in S$是独立子集A的extension但不是Φ的extension.
由于x是A的extension, $A\cup\{x\}\in I$. 由M的遗传性, $\{x\}\in I$, 即$\{x\}$是Φ的 extension, 矛盾.

推论1. 任何元素一旦不能被初始选中, 则永远不会被选中.

推论2. Greedy算法不会由于不再考虑未被初始选中的元素而产生错误.

HIT CS&E

定理3. 设$M=(S, I)$是一个Matroid, W是M的权函数, Greedy(M,W)返回一个M的优化子集.

证. ①. 引理2说明, 任何没有被Greedy选中的S元素, 以后不会被选中, 可以不再考虑.
②. 算法每次循环都按照定理给出方法进行贪心选择最大元素x.
③. 算法每次循环按照定理2的优化子结构求解子问题M'的包含x的最优子集的问题.
于是, Greedy(M, W)返回一个M的优化子集.

HIT CS&E

6.6 A task-scheduling problem

HIT CS&E

问题定义

- 单位时间任务
  需要一个单位时间就能够完成的任务
- 单位时间任务的调度问题
  输入:
  单位时间任务集$S=\{1, 2, \ldots, n\}$
  正整数任务期限$D=\{d_1, d_2, \ldots, d_n\}$, 任务$i$须在$d_i$前完成
  非负权集$W=\{w_1,w_2,\ldots,w_n\}$, 任务i不在d_i前完成罚款w_i
  输出:
  S的一个调度(S的一个执行顺序), 具有最小总罚款

HIT
CS&E

- 转换为加权Matroid的优化子集问题

定义1. 设S是一个任务调度, 任务i在S中是迟的如果它在d_i之后完成; 否则是早的.

定义2. 如果在一个调度中, 早任务总是先于迟任务, 则称该调度具有早任务优先形式.

定义3. 如果一个调度具有早任务优先形式而且按期限单调递增顺序执行各个早任务, 则称该调度具有规范化形式.

HIT
CS&E

- 任务调度的规范化

第一步: 将调度安排成早任务优先形式, 即如果早

寻找最优调度问题成为寻找最优早任务集合A的问题. 一旦A被确定后, 就可以按期限单调递增序列出A中的所有元素, 然后按任意顺序列出迟任务(即$S-A$)

- 调度优先形式不改变任何任务的早或迟状态
- 调度规范形式不改变任何任务的早或迟状态

HIT
CS&E

定义4. 任务集合A称为独立的如果存在一个关于A的调度, 使得A中的任务皆非迟任务.

例. 一个优化调度的早任务集合是独立任务集合.

以下:
用I表示所有独立任务集合的集族
用$N_t(A)$表示A中期限小于等于t的任务数

HIT
CS&E

$N_t(A)$ =A中期限小于等于t的任务数

引理1. 对于任何任务集合A, 下边的命题等价:
1. A是独立集合,
2. 对于$t=1, 2, \ldots, n$, $N_t(A)\le t$,
3. 如果按照期限递增顺序调度A中任务, 则A中无迟任务.

证. 1→2. 如果$N_t(A)>t$, 则有多于t个任务需要在t时间内完成, 不存在使得A中无迟任务的调度.
2→3. 若A中任务依其期限递增排列, 则2意味着排序后, 在时间t之前必须完成的A中任务数至多为t. 于是, 按期限递增顺序调度A中任务, A无迟任务.
3→1. 显然.

定理1. 若S是一个带期限的单位时间任务的集合, 且I为所有独立任务集构成的集族, 则$M=(S, I)$是一个Matroid.

证明. 1. S是非空有限集合.
2. I是S的子集的非空集族, 因为单个任务集属于I.
3. 遗传性: 若$A\in I$, $B\subseteq A$, 则$B\in I$.
4. 交换性: 设$A, B\in I$, $|B|>|A|$, $k=max_{1\le t\le n}\{t \mid N_t(B)\le N_t(A)\}$.

$N_t(B)\le N_t(A)$ | k | $N_j(B)>N_j(A)$
$t\le k$ | | $j>k$

于是, B中包含了比A中更多的具有期限$k+1$的任务. 设$x\in B-A$, x具有期限$k+1$. 令$A'=A\cup\{x\}$. 往证A'独立.
对于$1\le t\le k$, $N_t(A')=N_t(A)\le t$, 因为A是独立的.
对于$k<t\le n$, $N_t(A')\le N_t(B)\le t$, 因为B是独立的.
于是, A'是独立的.

$N_t(A)$ =A中期限小于等于t的任务数

HIT
CS&E

最后, 任务调度问题转换为$M=(S, I)$上寻找最优子集问题, M的加权函数为W(罚款)

HIT
CS&E

作业：

背包问题定义如下：

输入：正数 P_1，P_2，…, P_n, W_1，W_2, …, W_m, M

输出：X_1，X_2，…, X_n, $0 \le X_i \le 1$, 使得

$\sum_{1 \le i \le n} P_i X_i$ 最大

$\sum_{1 \le i \le n} W_i X_i \le M$

给出一个求解背包问题的贪心算法，并证明算法的正确性。

HIT
CS&E

第七章
Amortized Analysis

李建中
计算机科学与工程系

HIT
CS&E

提要

HIT
CS&E

参考资料

Chapter 6 Amortized Analysid

Pages 185 - 194

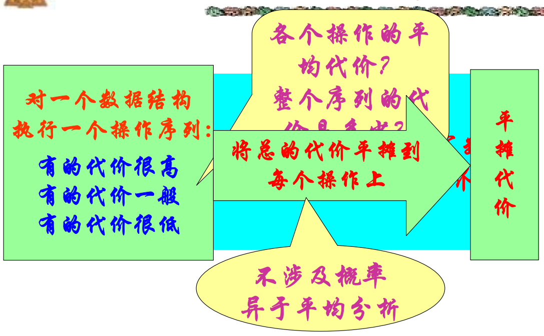
HIT
CS&E

7.1 *Elements of Amortized Analysis*

- *Amortized Analysis*的基本思想
- *Amortized Analysis*方法

HIT
CS&E

Amortized Analysis的基本思想



HIT
CS&E

Amortized Analysis的基本思想

- Aggregate Analysis方法（每个操作的代价）
  - 确定n个操作的上界$T(n)$, 每个操作平摊$T(n)/n$
- Accounting方法（整个操作序列的代价）
  - 不同类型操作赋予不同的平摊代价
  - 某些操作在数据结构的特殊对象上"预付"代价
- Potential方法（整个操作序列的代价）
  - 不同类型操作赋予不同的平摊代价
  - "预付"的代价作为整个数据结构的"能量"

HIT CS&E

7.2 Aggregate Analysis

- 聚集方法的原理
- 聚集方法的实例之一
- 聚集方法的实例之二

HIT CS&E

聚集方法的原理

数据结构上共有n个操作，最坏情况下：

操作1: Cost=t_1
操作2: Cost=t_2
.
.
.
操作n: Cost=t_n

$T(n)=\sum_{i=1}^{n} t_i$

平摊代价：$T(n)/n$

每个操作被赋予相同代价，不管操作类型

HIT CS&E

聚集方法实例之一：栈操作系列

- 普通栈操作及其时间代价
  - Push(S, x): 将对象压x入栈S
  - Pop(S)：弹出并返回S的顶端元素
  - 两个操作的运行时间都是$O(1)$
  - 可把每个操作的实际代价视为1
  - n个Push和Pop操作系列的总代价是n
  - n个操作的实际运行时间为$\theta(n)$

HIT CS&E

- 新的普通栈操作及其时间代价
  - 操作Multipop(S, k):

    去掉S的k个项对象，当$|S|<k$时弹出整个栈
  - 实现算法

    Multipop(S, k)
    1 While not STACK-EMPTY(S) and $k\neq0$ Do
    2 Pop(S);
    3 $k\leftarrow k-1$.
  - Multipop(S, k)的实际代价 (设Pop的代价为1)
    - Multipop的代价为$min(|S|, k)$

HIT CS&E

- 初始栈为空的n个栈操作序列的分析
  - n个栈操作序列由Push、Pop和Multipop组成
  - 粗略分析
    - 最坏情况下，每个操作都是Multipop
    - 每个Multipop的代价最坏是$O(n)$
    - 操作系列的最坏代价为$T(n)=O(n^2)$
    - 平摊代价为$T(n)/n=O(n)$
  - 精细分析
    - 一个对象在每次被压入栈后至多被弹出一次
    - 在非空栈上调用Pop的次数(包括在Multipop内的调用)至多为Push执行的次数，即至多为n
    - 最坏情况下操作序列的代价为$T(n)\leq 2n=O(n)$
    - 平摊代价=$T(n)/n=O(1)$

分析太粗糙！！！

n-1个push
1个multipop

HIT CS&E

聚集方法实例之二：二进计数器

- 问题定义：由0开始计数的k位二进计数器

  输入：k位二进制变量x，初始值为0

  输出：$x+1 \bmod 2^k$

  数据结构：

  $A[0..k-1]$作为计数器，存储x

  x的最低位在$A[0]$中，最高位在$A[k-1]$中

  $x=\sum_{i=0}^{k-1} A[i]\cdot 2^i$

HIT CS&E

- 计数器加1算法

输入：$A[0..k-1]$，存储二进制数x

输出：$A[0..k-1]$，存储二进制数$x+1 \bmod 2^k$

```
INCREMENT(A)
i←0
while  i<k and A[i]=1  Do
      A[i]←0;
      i←i+1;
If  i<length[A]  Then  A[i]←1
```

- 初始为零的计数器上n个INCREMENT操作分析

Counter Value N	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	每个操作 Cost
0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	2
3	0	0	0	0	0	0	1	1	1
4	0	0	0	0	0	1	0	0	3
5	0	0	0	0	0	1	0	1	1
6	0	0	0	0	0	1	1	0	2
7	0	0	0	0	0	1	1	1	1
8	0	0	0	0	1	0	0	0	4
9	0	0	0	0	1	0	0	1	1
10	0	0	0	0	1	0	1	0	2
11	0	0	0	0	1	0	1	1	1
12	0	0	0	0	1	1	0	0	3
13	0	0	0	0	1	1	0	1	1
14	0	0	0	0	1	1	1	0	2
15	0	0	0	0	1	1	1	1	1
16	0	0	0	1	0	0	0	0	5

HIT CS&E

- 粗略分析
  - 每个Increment的时间代价最多$O(k)$
  - n个Increment序列的时间代价最多$O(kn)$
  - n个Increment平摊代价为$O(k)$
  - 例如上例中：$k*n=8*16=128$
- 精细分析

HIT CS&E

操作Cost＝O(发生改变的位数)

Counter Value N	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

HIT CS&E

- 精细分析
  - $A[0]$每次操作发生一次改变，总次数为n
  - $A[1]$每两次操作发生一次改变，总次数为$n/2$
  - $A[2]$每四次操作发生一次改变，总次数为$n/4$
  - 一般地
    - 对于$i=0, 1, \ldots, \lg n$, $A[i]$改变次数为$n/2^i$
    - 对于$i>\lg n$, $A[i]$不发生改变
    （因为n个操作结果为n，仅涉及$A[0]$至$A[\lg n]$位）
  - $T(n)=\sum_{0\leq i\leq \lg n} n/2^i < n\sum_{0\leq i\leq \infty} 1/2^i = O(n)$
  - 每个Increment操作的平摊代价为$O(1)$

HIT CS&E

7.3 The Accounting Method

- Accounting方法的原理
- Accounting方法的实例之一
- Accoutning方法的实例之二

HIT CS&E

Accounting方法的原理

- Accounting方法
  - 目的是分析n个操作系列的复杂性上界
  - 一个操作序列中有不同类型的操作
  - 不同类型的操作的操作代价各不相同
  - 于是我们为每种操作分配不同的平摊代价
    - 平摊代价可能比实际代价大，也可能比实际代价小

数据结构中存储的Credit在任何时候都必须非负，即$\Sigma_{1\le i\le n}\alpha_i - \Sigma_{1\le i\le n}c_i \ge 0$永远成立

  - 平摊代价的选择规则：
    - 设c_i和α_i是操作i的实际代价和平摊代价
    - $\Sigma_{1\le i\le n}\alpha_i \ge \Sigma_{1\le i\le n}c_i$ 必须对于任意n个操作序列都成立

HIT CS&E

栈操作序列的分析

- 各栈操作的实际代价
  - Cost(PUSH)=*1*
  - Cost(POP)=*1*
  - Cost(MULTIPOP)=*min{k, s}*
- 各栈操作的平摊代价
  - Cost(PUSH)=2
    - 一个1用来支付PUSH的开销,
    - 另一个1存储在压入栈的元素上, 预支POP的开销
  - Cost(POP)=*0*
  - Cost(MULTIPOP)=*0*
- 平摊代价满足
  - $\Sigma_{1\le i\le n}\alpha_i - \Sigma_{1\le i\le n}c_i \ge 0$ 对于任意n个操作序列都成立
    - 因为栈中的对象数≥0
- n个栈操作序列的总平摊代价
  - $O(n)$

HIT CS&E

二进制计数器Increment操作序列分析

- Increment操作的平摊代价
  - 每次一位被置1时，付2美元
    - 1美元用于置1的开销
    - 1美元存储在该"1"位上，用于支付其被置0时的开销
    - 置0操作无需再付款
  - Cost(Increment)=2
- 平摊代价满足
  - $\Sigma_{1\le i\le n}\alpha_i \ge \Sigma_{1\le i\le n}c_i$ 对于任意n个操作序列都成立，即任何时刻，计数器上的Credit非负
- n个Increment操作序列的总平摊代价
  - $O(n)$

HIT CS&E

7.4 The Potential Method

- Potential方法的原理
- Potential方法的实例之一
- Potential方法的实例之二

HIT CS&E

Potential方法的原理

- Potential方法
  - 目的是分析n个操作系列的复杂性上界
  - 在会计方法中，如果操作的平摊代价比实际代价大，我们将余额与数据结构的数据对象相关联
  - Potential方法把余额与整个数据结构关联，所有的这样的余额之和，构成**数据结构的势能**
    - 如果操作的平摊代价大于操作的实际代价，**势能增加**
    - 如果操作的平摊代价小于操作的实际代价，要用数据结构的势能来支付实际代价，**势能减少**

- 数据结构势能的定义
  - 考虑在初始数据结构D_0上执行n个操作
  - 对于操作i
    - 操作i的实际代价为c_i
    - 操作i将数据结构D_{i-1}变为D_i
    - 数据结构D_i的势能是一个实数$\phi(D_i)$，ϕ是一个正函数
    - 操作i的平摊代价：$\alpha_i = c_i + \phi(D_i) - \phi(D_{i-1})$
  - n个操作的总平摊代价（必须是实际代价的上界）

$$\sum_{i=1}^{n}\alpha_i = \sum_{i=1}^{n}(c_i + \phi(D_i) - \phi(D_{i-1}))$$
$$= \sum_{i=1}^{n}c_i + \phi(D_n) - \phi(D_0)$$

  - 关键是ϕ的定义
    - 保证$\phi(D_n)\ge\phi(D_0)$，使总平摊代价是总实际代价的上界
    - 如果对于所有i，$\phi(D_i)\ge\phi(D_0)$，可以保证$\phi(D_n)\ge\phi(D_0)$
    - 实际可以定义$\phi(D_0)=0$，$\phi(D_i)\ge 0$

HIT
CS&E

栈操作序列的分析

- 栈的势能定义
  - $\phi(D_m)$定义为栈D_m中对象的个数，于是
    - $\phi(D_0)=0$，D_0是空栈
    - $\phi(D_i)\geq 0=\phi(D_0)$，因为栈中对象个数不会小于0
    - n个操作的总平摊代价是实际代价的上界
  - 栈操作的平摊代价（设栈D_{i-1}中具有s个对象）
    - PUSH: $\alpha_i=c_i+\phi(D_i)-\phi(D_{i-1})=1+(s+1)-s=2$
    - POP: $\alpha_i=c_i+\phi(D_i)-\phi(D_{i-1})=1+(s-1)-s=0$
    - MULTIPOP(S, k): 设$k'=min(k,s)$

      $\alpha_i=c_i+\phi(D_i)-\phi(D_{i-1})=k'+(s-k')-s=k'-k'=0$
  - n个栈操作序列的平摊代价是$O(n)$

HIT
CS&E

二进制计数器操作序列分析

- 计数器的势能定义
  - $\phi(D_m)$定义为第m个操作后计数器中1的个数b_m
    - $\phi(D_0)=0$，D_0是空栈
    - $\phi(D_i)\geq 0=\phi(D_0)$，因为计数器中$1$的个数不会小于0
    - 于是，n个操作的总平摊代价是实际代价的上界
  - INCREMENT操作的平摊代价
    - 第i个INCREMENT操作resets t_i bits，实际代价为t_i+1
    - 计算操作i的平摊代价$\alpha_i=c_i+\phi(D_i)-\phi(D_{i-1})$
      - If $b_i=0$，操作i resets所有k位，所以$b_{i-1}=t_i=k$
      - If $b_i>0$，则$b_i=b_{i-1}-t_i+1$
      - 于是 $b_i\leq b_{i-1}-t_i+1$

        $\phi(D_i)-\phi(D_{i-1})=b_i-b_{i-1}\leq b_{i-1}-t_i+1-b_{i-1}=1-t_i$
      - 平摊代价$\alpha_i=c_i+\phi(D_i)-\phi(D_{i-1})\leq(t_i+1)+(1-t_i)=2$
  - n个操作序列的总平摊代价是$O(n)$

HIT
CS&E

第七章

Tree Searching Strategies

骆吉洲
计算机科学与工程系

HIT
CS&E

提要

HIT
CS&E

参考资料

《计算机算法设计与分析》
- 第5章，第6章

《网站资料》
- 第七章

HIT
CS&E

7.1 Motivation of Tree Searching

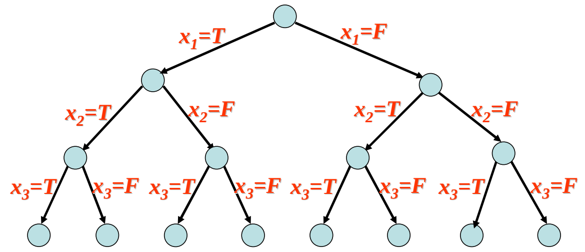
很多问题可以表示成为树.
于是, 这些问题可以使用树搜索算法来求解

HIT
CS&E

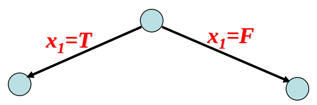
布尔表达式可满足性问题

- 问题的定义
  - 输入: n个布尔变量$x_1, x_2, \ldots, x_n$
    关于$x_1, x_2, \ldots, x_n$的k个析取布尔式
  - 输出: 是否存在一个$x_1, x_2, \ldots, x_n$的一种赋值
    使得所有k个布尔析取式皆为真

- 把问题表示为树
  - 通过不断地为赋值集合分类来建立树
    (以三个变量(x_1, x_2, x_3)为例)



- 求解问题
  - 设有布尔式：$-x_1$, x_1, $x_2 \vee x_5$, x_3, $-x_2$



8-Puzzle问题

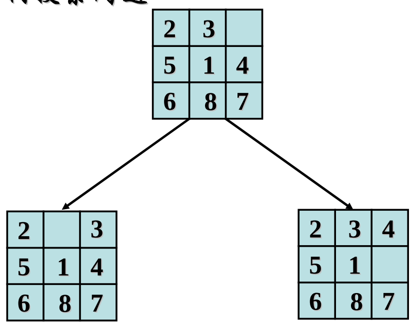
- 问题的定义
  - 输入：具有8个编号小方块的魔方

2	3	
5	1	4
6	8	7

  - 输出：移动系列，经过这些移动，魔方达如下状态

1	2	3
8		4
7	6	5

- 转换为树搜索问题

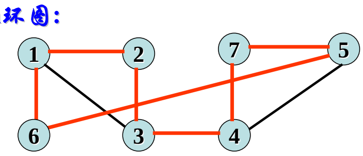


Hamiltonian环问题

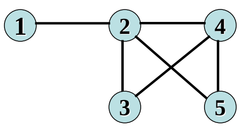
- 问题定义
  - 输入：具有n个节点的连通图$G=(V, E)$
  - 输出：G中是否具有Hamiltonian环

沿着G的n条边经过每个节点一次，并回到起始节点的环称为G的一个Hamiltonian环。

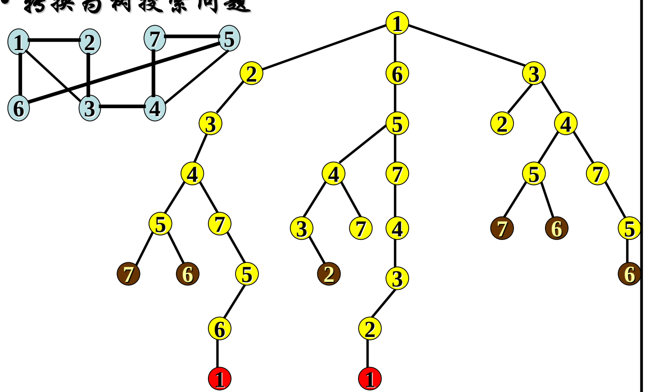
有Hamiltonian环图：



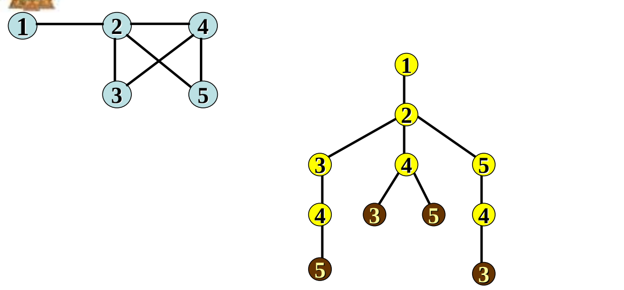
无Hamiltonian环图：



- 转换为树搜索问题



HIT CS&E



HIT CS&E

7.2 *Basic Tree Searching Strategies*

- **Breadth-First Search**
- **Depth-First Search**

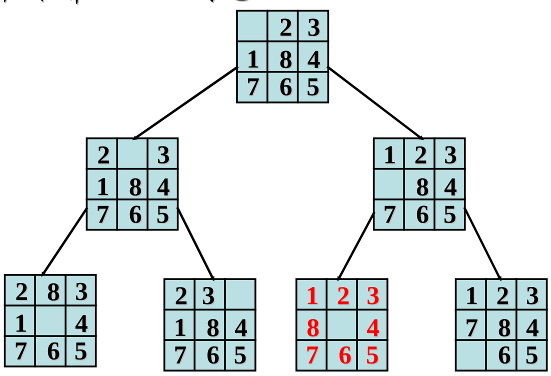
HIT CS&E

Breadth-First Search

- **算法**

1. 构造由根组成的队列Q;
2. If Q的第一个元素x是目标节点 Then 停止;
3. 从Q中删除x，把x的所有子节点加入Q的末尾;
4. If Q空 Then 失败 Else *goto* 2.

- **例：求解8-Puzzle问题**



HIT CS&E

Depth-First Search

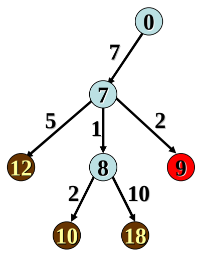
- **算法**

1. 构造一个由根构成的单元素栈S;
2. If $Top(S)$是目标节点 Then 停止;
3. $Pop(S)$, 把$Top(S)$的所有子节点压入栈顶;
4. If S空 Then 失败 Else *goto* 2.

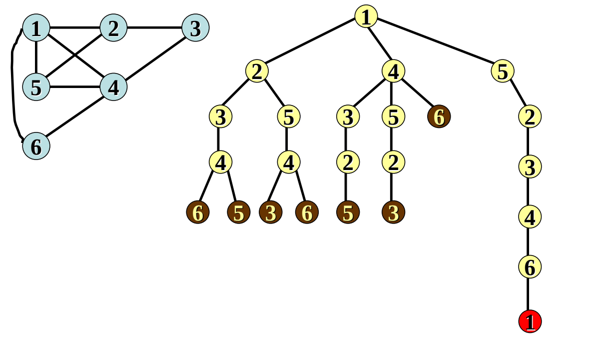
- **例1. 求解子集合和问题**

输入: S={7, 5, 1, 2, 10}

输出: 是否存在$S' \subseteq S$, 使得$Sum(S')=9$



• 例2. 求解Hamiltonian环问题



HIT CS&E

7.3 Optimal Tree Searching Strategies

- Hill Climbing
- Best-First Search Strategy
- Branch-and-Bound Strategy

HIT CS&E

Hill Climbing

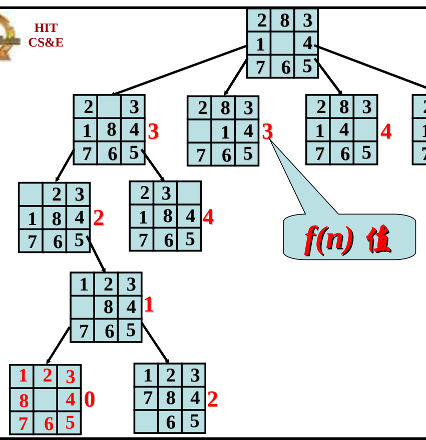
- 基本思想
  - 在深度优先搜索过程中，我们经常遇到多个节点可以扩展的情况，首先扩展哪个?
  - 爬山策略使用贪心方法确定搜索的方向，是优化的深度优先搜索策略
  - 爬山策略使用启发式测度来排序节点扩展的顺序

HIT CS&E

- 用8-Puzzle问题来说明爬山策略的思想
  - 启发式测度函数: $f(n)=W(n)$, $W(n)$是节点n中处于错误位置的方块数.
  - 例如，如果节点n如下，则$f(n)=3$，因为方块1、2、8处于错误位置.

2	8	3
1		4
7	6	5

HIT CS&E



HIT CS&E

- Hill Climbing算法
  1. 构造由根组成的单元素栈S;
  2. If $Top(S)$是目标节点 Then 停止;
  3. Pop(S);
  4. S的子节点按照其启发测度由大到小的顺序压入S;
  5. If S空 Then 失败 Else *goto* 2.

HIT CS&E

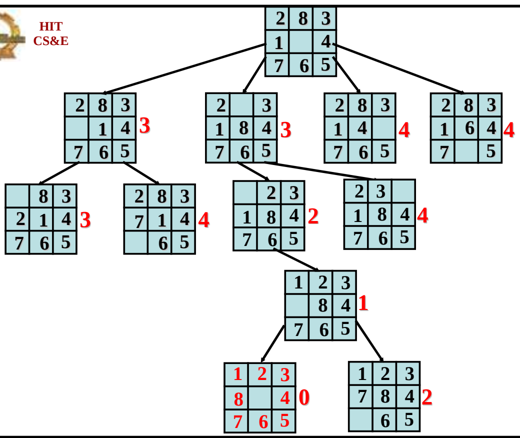
Best-First Search Sttrategy

- 基本思想
  - 结合深度优先和广度优先的优点
  - 根据一个评价函数, 在目前产生的所有节点中选择具有最小评价函数值的节点进行扩展.
  - 具有全局优化观念, 而爬山策略仅具有局部优化观念.

HIT CS&E

- BesT-First Search算法
  1. 使用评价函数构造一个堆H, 首先构造由根组成的单元素堆;
  2. If H的根r是目标节点 Then 停止;
  3. 从H中删除r, 把r的子节点插入H;
  4. If H空 Then 失败 Else *goto* 2.
- 8-Puzzle问题实例

HIT CS&E



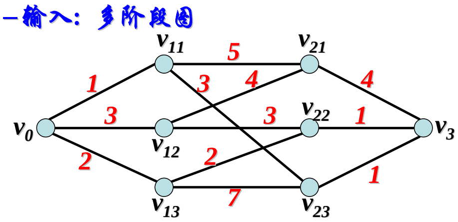
HIT CS&E

Branch-and-Bound Strategy

- 基本思想
  - 上述方法很难用于求解优化问题
  - 分支界限策略可以有效地求解组合优化问题
  - 发现优化解的一个界限
  - 缩小解空间, 提高求解的效率
- 举例说明分支界限策略的原理

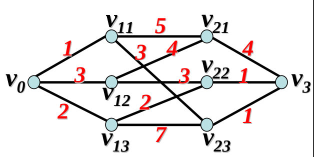
HIT CS&E

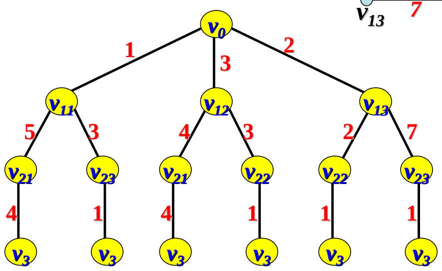
- 多阶段图搜索问题
  - 输入: 多阶段图



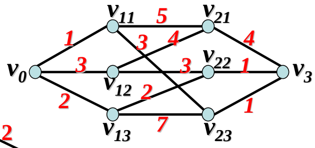
  - 输出: 从v_0到v_3的最短路径

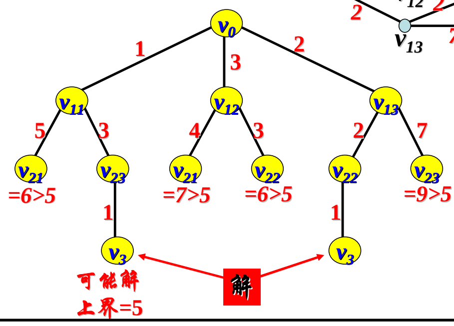
- 问题的树表示





- 使用爬山策略进行分支界限搜索





HIT CS&E

- 分支界限策略的原理
  - 产生分支的机制(使用前面的任意一种策略)
  - 产生一个界限(可以通过发现可能解)
  - 进行分支界限搜索，即剪除不可能产生优化解的分支.

HIT CS&E

7.4 Personnel Assignment Problem

- 问题的定义
- 转换为树搜索问题
- 求解问题的分支界限搜索算法

HIT CS&E

问题的定义

- 输入
  - 人的集合$P=\{P_1, P_2, \ldots, P_n\}$，$P_1<P_2< \ldots <P_n$

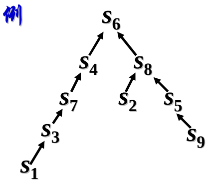
例. 给定P=$\{P_1, P_2, P_3\}$, J=$\{J_1, J_2, J_3\}$, $J_1 \le J_3$, $J_2 \le J_3$.
$P_1 \to J_1$、$P_2 \to J_2$、$P_3 \to J_3$是可能的解.
$P_1 \to J_1$、$P_2 \to J_3$、$P_3 \to J_2$不可能是解.

  - 如果$f(P_i) \le f(P_j)$，则$P_i \le P_j$

HIT CS&E

转换为树搜索问题

- 拓扑排序
  - 输入：偏序集合$(S, \le)$
  - 输出：S的拓扑序列是$<s_1, s_2, \ldots, s_n>$，
    满足：如果$s_i \le s_j$，则s_i排在s_j的前面.
  - 例



拓扑排序：

s_1 s_3 s_7 s_4 s_9 s_5 s_2 s_8 s_6

- 问题的解空间

命题1. $P_1 \to J_{k1}$、$P_2 \to J_{k2}$、…、$P_n \to J_{kn}$是一个可能解，当且仅当J_{k1}、J_{k2}、…、J_{kn}必是一个拓扑排序的序列.

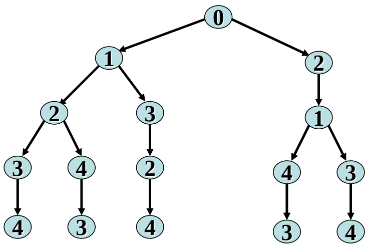
问题的解空间是所有拓扑排序的序列集合，
每个序列对于一个可能的解

(J_2, J_1, J_3, J_4)、(J_2, J_1, J_4, J_3)是拓扑排序序列

(J_1, J_2, J_4, J_3)对应于$P_1 \to J_1$、$P_2 \to J_2$、$P_3 \to J_4$、$P_4 \to J_3$

HIT CS&E

• 问题的树表示(即用树表示所有拓扑排序序列)



HIT CS&E

● 拓扑序列树的生成算法

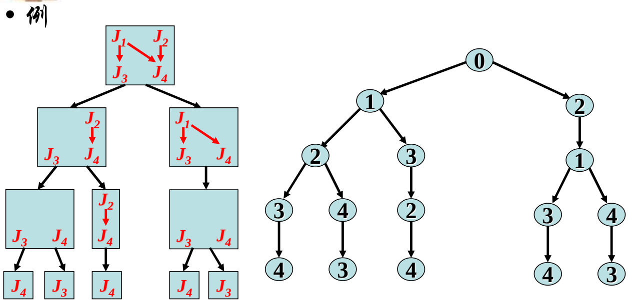
输入：偏序集合S, 树根$root$.

输出：由S的所有拓扑排序序列构成的树.

1. 生成树根$root$;
2. 选择偏序集中没有前序元素的所有元素，作为 $root$的子节点;
3. For $root$的每个字节点v Do
4. $S=S-\{v\}$;
5. 把v作为根，递归地处理S.

HIT CS&E

• 例



HIT CS&E

求解问题的分支界限搜索算法

• 计算解的代价的下界

命题2. 把代价矩阵某行(列)的各元素减去同一个数，不影响优化解的求解.

– 代价矩阵的每行(列)减去同一个数(该行或列的最小数), 使得每行和每列至少有一个零, 其余各元素非负.

– 每行(列)减去的数的和即为解的下界.

HIT CS&E

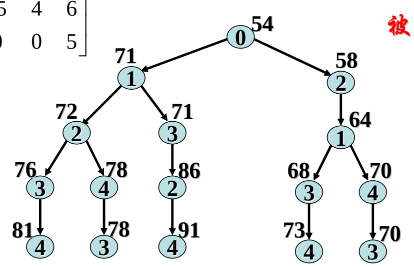
例.

$$\begin{array}{c|cccc|c} & J_1 & J_2 & J_3 & J_4 & \\ \hline P_1 & 29 & 19 & 17 & 12 & -12 \\ P_2 & 32 & 30 & 26 & 28 & -26 \\ P_3 & 3 & 21 & 7 & 9 & -3 \\ P_4 & 18 & 13 & 10 & 15 & -10 \\ \hline & & -3 & & & \end{array} \Rightarrow \begin{bmatrix} 17 & 4 & 5 & 0 \\ 6 & 1 & 0 & 2 \\ 0 & 15 & 4 & 6 \\ 8 & 0 & 0 & 5 \end{bmatrix}$$

解代价下界=12+26+3+10+3=54

解空间的加权树表示

$$\begin{bmatrix} 17 & 4 & 5 & 0 \\ 6 & 1 & 0 & 2 \\ 0 & 15 & 4 & 6 \\ 8 & 0 & 0 & 5 \end{bmatrix}$$



被分配到的人

1

2

3

4

HIT CS&E

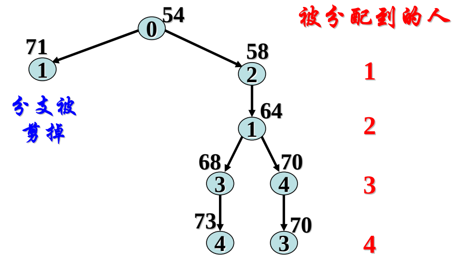
- 分支界限搜索(使用爬山法)算法
  1. 建立根节点,其权值为解代价下界;
  2. 使用爬山法,类似于拓朴排序序列树生成算法求解问题,每产生一个节点,其权值为加工后的代价矩阵对应元素加其父节点权值;
  3. 一旦发现一个可能解,将其代价作为界限,循环地进行分支界限搜索:剪掉不能导致优化解的子解,使用爬山法继续扩展新增节点,直至发现优化解.

- 例

$\{P_1, P_2, P_3, P_4\}$

$$\begin{tikzcd} J_1 \arrow[d] \arrow[dr] & J_2 \arrow[d] \\ J_3 & J_4 \end{tikzcd}$$

$$\begin{bmatrix} 17 & 4 & 5 & 0 \\ 6 & 1 & 0 & 2 \\ 0 & 15 & 4 & 6 \\ 8 & 0 & 0 & 5 \end{bmatrix}$$



HIT CS&E

7.5 Traveling Salesperson Optimization Problem

- 问题的定义
- 转换为树搜索问题
- 分支界限搜索算法

HIT CS&E

问题的定义

输入:无向连通图$G=(V, E)$,
每个节点都没有到自身的边,
每对节点之间都有一条非负加权边.

输出:一条由任意一个节点开始
经过每个节点一次
最后返回开始节点的路径,
该路径的代价(即权值只和)最小.

HIT CS&E

转换为树搜索问题

- 所有解集合作为树根,其权值由代价矩阵使用上节方法计算;
- 用爬山法递归地划分解空间,得到二叉树
- 划分过程:
  - 如下选择图上满足下列条件的边(i, j)
    - $\text{Cost}(i, 1)=\max\{\text{Cost}(k,1) \mid \forall k\in V\}$
    - $\text{Cost}(i, j)=0$

    使右子树代价下界增加最大
  - 所有包含(i, j)的解集合作为左子树
  - 所有不包含(i, j)的解集合作为右子树
  - 计算出左右子树的代价下界

HIT CS&E

分支界限搜索算法

- 在上述二叉树建立算法中增加如下策略:
  - 发现优化解的上界α;
  - 如果一个子节点的代价下界超过α,则终止该节点的扩展.
- 下边我们用一个例子来说明算法

- 构造根节点,设代价矩阵如下

$$
\begin{array}{c|c|c}
 & \begin{array}{ccccccc} j=1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} & \\
\begin{array}{c} i=1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} &
\begin{bmatrix}
\infty & 3 & 93 & 13 & 33 & 9 & 57 \\
4 & \infty & 77 & 42 & 21 & 16 & 34 \\
45 & 17 & \infty & 36 & 16 & 28 & 25 \\
39 & 90 & 80 & \infty & 56 & 7 & 91 \\
28 & 46 & 88 & 33 & \infty & 25 & 57 \\
3 & 88 & 18 & 46 & 92 & \infty & 7 \\
44 & 26 & 33 & 27 & 84 & 39 & \infty
\end{bmatrix} &
\begin{array}{c} -3 \\ -4 \\ -16 \\ -7 \\ -25 \\ -3 \\ -26 \end{array} \\
 & \begin{array}{ccccccc} & & -7 & -1 & & & -4 \end{array} &
\end{array}
$$

- ➢ 根节点为所有解的集合
- ➢ 计算根节点的代价下界

- ➢ 得到如下根节点及其代价下界

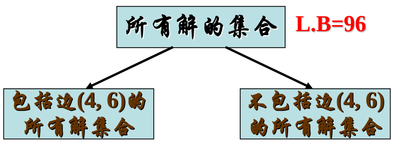
所有解的集合 L.B=96

- ➢ 变换后的代价矩阵为

$$
\begin{array}{c|c}
 & \begin{array}{ccccccc} j=1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\
\begin{array}{c} i=1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} &
\begin{bmatrix}
\infty & 0 & 83 & 9 & 30 & 6 & 50 \\
0 & \infty & 66 & 37 & 17 & 12 & 26 \\
29 & 1 & \infty & 19 & 0 & 12 & 5 \\
32 & 83 & 66 & \infty & 49 & 0 & 80 \\
3 & 21 & 56 & 7 & \infty & 0 & 28 \\
0 & 85 & 8 & 42 & 89 & \infty & 0 \\
18 & 0 & 0 & 0 & 58 & 13 & \infty
\end{bmatrix}
\end{array}
$$

$$
\begin{bmatrix}
\infty & 0 & 83 & 9 & 30 & 6 & 50 \\
0 & \infty & 66 & 37 & 17 & 12 & 26 \\
29 & 1 & \infty & 19 & 0 & 12 & 5 \\
32 & 83 & 66 & \infty & 49 & 0 & 80 \\
3 & 21 & 56 & 7 & \infty & 0 & 28 \\
0 & 85 & 8 & 42 & 89 & \infty & 0 \\
18 & 0 & 0 & 0 & 58 & 13 & \infty
\end{bmatrix}
$$

- 构造根节点的两个子节点
  - ➢ 选择使子节点代价下界增加最大的划分边(4, 6)
  - ➢ 建立根节点的子节点:
    - ✓ 左子节点为包括边(4, 6)的所有解集合
    - ✓ 左子节点为不包括边(4, 6)的所有解集合

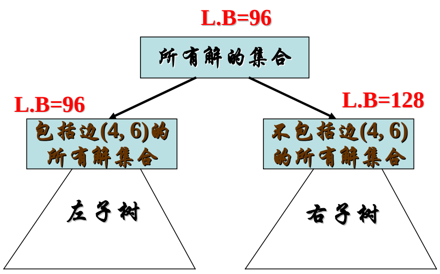


HIT CS&E

- ➢ 计算左右子节点的代价下界
  - ✓ (4, 6)的代价为0, 所以左节点代价下界仍为96.
  - ✓ 我们来计算右节点的代价下界:
    - ◆ 如果一个解不包含(4, 6), 它必包含一条从4出发的边和进入节点6的边.
    - ◆ 由变换后的代价矩阵可知, 具有最小代价由4出发的边为(4, 1), 代价为32.
    - ◆ 由变换后的代价矩阵可知, 具有最小代价进入6的边为(5, 6), 代价为0.
    - ◆ 于是, 右节点代价下界为: 96+32+0=128.

HIT CS&E

- ➢ 目前的树为



- 递归地构造左右子树
  - ➢ 构造左子树根对应的代价矩阵
    - ✓ 左子节点为包括边(4, 6)的所有解集合, 所以矩阵的第4行和第6列应该被删除
    - ✓ 由于边(4, 6)被使用, 边(6, 4)不能再使用, 所以代价矩阵的元素C[6, 4]应该设置为∞.
    - ✓ 结果矩阵如下

$$
\begin{array}{c|c}
 & \begin{array}{cccccc} j=1 & 2 & 3 & 4 & 5 & 7 \end{array} \\
\begin{array}{c} i=1 \\ 2 \\ 3 \\ 5 \\ 6 \\ 7 \end{array} &
\begin{bmatrix}
\infty & 0 & 83 & 9 & 30 & 50 \\
0 & \infty & 66 & 37 & 17 & 26 \\
29 & 1 & \infty & 19 & 0 & 5 \\
3 & 21 & 56 & 7 & \infty & 28 \\
0 & 85 & 8 & \infty & 89 & 0 \\
18 & 0 & 0 & 0 & 58 & \infty
\end{bmatrix}
\end{array}
$$

- **计算左子树根的代价下界**
  - ✓ 矩阵的第5行不包含0
  - ✓ 第5行元素减3，左子树根代价下界为：96+3=99
  - ✓ 结果矩阵如下

i \ j	j=1	2	3	4	5	7
i=1	∞	0	83	9	30	50
2	0	∞	66	37	17	26
3	29	1	∞	19	0	5
5	0	18	53	4	∞	25
6	0	85	8	∞	89	0
7	18	0	0	0	58	∞

- **构造右子树根对应的代价矩阵**
  - ✓ 右子节点为不包括边(4, 6)的所有解集合，只需要把C[4, 6]设置为∞
  - ✓ 结果矩阵如下

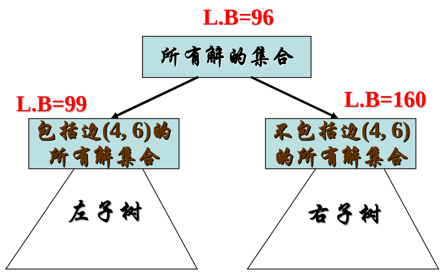
i \ j	j=1	2	3	4	5	6	7
i=1	∞	0	83	9	30	6	50
2	0	∞	66	37	17	12	26
3	29	1	∞	19	0	12	5
4	32	83	66	∞	49	∞	80
5	3	21	56	7	∞	0	28
6	0	85	8	42	89	∞	0
7	18	0	0	0	58	13	∞

- **计算右子树根的代价下界**
  - ✓ 矩阵的第4行不包含0
  - ✓ 第4行元素减32，右子树根代价下界为：128+32=160
  - ✓ 结果矩阵如下

i \ j	j=1	2	3	4	5	6	7
i=1	∞	0	83	9	30	6	50
2	0	∞	66	37	17	12	26
3	29	1	∞	19	0	12	5
4	0	51	34	∞	17	∞	48
5	3	21	56	7	∞	0	28
6	0	85	8	42	89	∞	0
7	18	0	0	0	58	13	∞

HIT CS&E

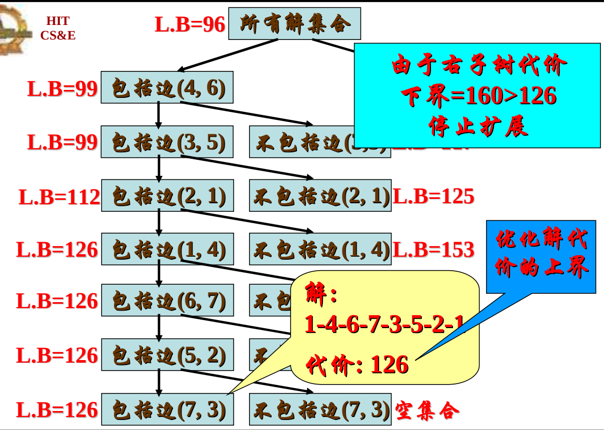
- **目前的树为**



HIT CS&E

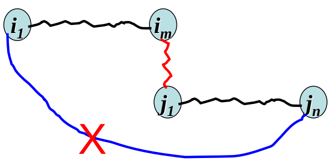
- **使用爬山策略扩展左子树根**
  - ✓ 选择边使子节点代价下界增加最大的划分边(3, 5)
  - ✓ 左子节点为包括边(3, 5)的所有解集合
  - ✓ 右子节点为不包括边(3, 5)的所有解集合
  - ✓ 计算左、右子节点的代价下界：99和117
- **目前树扩展为：**

HIT CS&E



注 意

如果i_1-i_2-…-i_m和j_1-j_2-…-j_m已被包含在一个正在构造的路径中，(i_m, j_1)被加入，则必须避免j_n到 i_1的路径被加入。否则出现环。



HIT CS&E

7.6 The A* Algorithm

- A*算法的基本思想
- A*算法的规则
- 应用A*算法求解最短路径问题

HIT CS&E

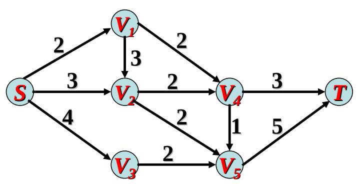
A*算法的基本思想

- A*算法与分支界限策略的比较
  - 分支界限策略是为了剪掉不能达到优化解的分支
  - 分支界限策略的关键是"界限"
  - A*算法的核心是告诉我们在某些情况下，我们得到的解一定是优化解，于是算法可以停止
  - A*算法试图尽早地发现优化解
  - A*算法经常使用Best-first策略求解优化问题

- A*算法关键—代价函数
  - 对于任意节点n
    - $g(n)$＝从树根到n的代价
    - $h^*(n)$＝从n到目标节点的优化路径的代价
    - $f^*(n)=g(n) + h^*(n)$是节点n的代价
  - What is the value of $h^*(n)$?
    - 不知道！
    - 于是，$f^*(n)$也不知道
  - 估计$h^*(n)$
    - 使用任何方法去估计$h^*(n)$，用$h(n)$表示$h^*(n)$的估计
    - $h(n) \leq h^*(n)$总为真
    - $f(n)=g(n)+h(n) \leq g(n)+h^*(n)=f^*(n)$定义为n的代价

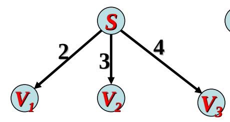
例1. 最短路径问题：

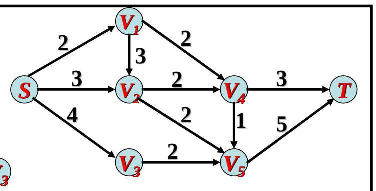
– 输入：



– 输出：发现一个从S到T的最短路径

– 求解树的第一阶段





$g(V_1)=2$, $g(V_2)=3$, $g(V_3)=4$

~~$h^*(V_1)=5$, $f^*(V_1)=g(V_1)+h^*(V_1)=7$~~

– 估计$h^*(n)$
- 从V_1出发有两种可能：代价为2，代价为3，最小者为2
- $h^*(V_1) \geq 2$，选择$h(n)=2$为$h^*(V_1)$的估计值
- $f(V_1)=g(v_1)+h(V_1)=4$为V_1的代价

- A*算法本质—已经发现的解是优化解

定理1. 使用Best-first策略搜索树，如果A*选择的节点是目标节点，则该节点表示的解是优化解.

证明.

令n是任意扩展到的节点，t是选中目标节点.

往证$f(t)=g(t)$是优化解代价.

(1). A*算法使用Best-first策略，$f(t) \le f(n)$.

(2). A*算法使用$h(n) \le h^*(n)$估计规则，$f(t) \le f(n) \le f^*(n)$.

(3). $\{f^*(n)\}$中必有一个为优化解的代价，令其为$f^*(s)$. 我们有$f(t) \le f^*(s)$.

(4). t是目标节点$h(t)=0$，所以$f(t)=g(t)+h(t)=g(t) \le f^*(s)$.

(5). $f(t)=g(t)$是一个可能解，$g(t) \ge f^*(s)$，$f(t)=g(t)=f^*(s)$.

HIT CS&E

A*算法的规则

(1). 使用Best-first策略搜索树;

(2). 节点n的代价函数为$f(n)=g(n)+h(n)$，$g(n)$是从根到n的路径代价，$h(n)$是从n到某个目标节点的优化路径代价;

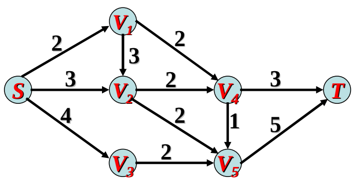
(3). 对于所有n，$h(n) \le h^*(n)$;

(4). 当选择到的节点是目标节点时，算法停止，返回一个优化解.

HIT CS&E

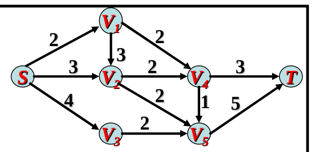
应用A*算法求解最短路径问题

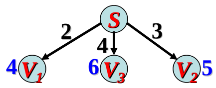
- 问题的输入:



- A*算法的执行全过程

Step 1



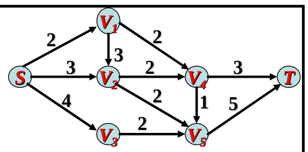


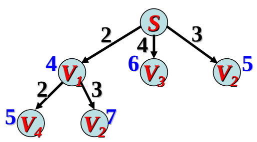
$g(V_1)=2$　$h(V_1)=min\{2,3\}=2$　$f(V_1)=2+2=4$

$g(V_3)=4$　$h(V_3)=min\{2\}=2$　$f(V_3)=4+2=6$

$g(V_2)=3$　$h(V_2)=min\{2,2\}=2$　$f(V_2)=2+2=5$

Step 2. 扩展V_1

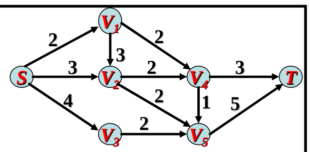


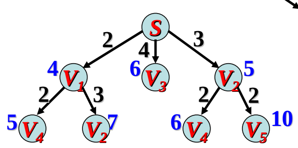


$g(V_4)=2+2=4$　$h(V_4)=min\{3,1\}=1$　$f(V_4)=4+1=5$

$g(V_2)=2+3=5$　$h(V_2)=min\{2, 2\}=2$　$f(V_2)=5+2=7$

Step 3. 扩展V_2

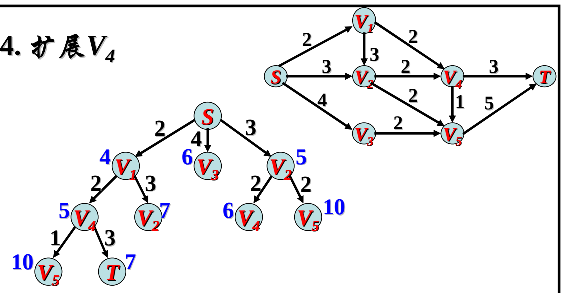




$g(V_4)=3+2=5$　$h(V_4)=min\{3,1\}=1$　$f(V_4)=5+1=6$

$g(V_5)=3+2=5$　$h(V_5)=min\{5\}=5$　$f(V_2)=5+5=10$

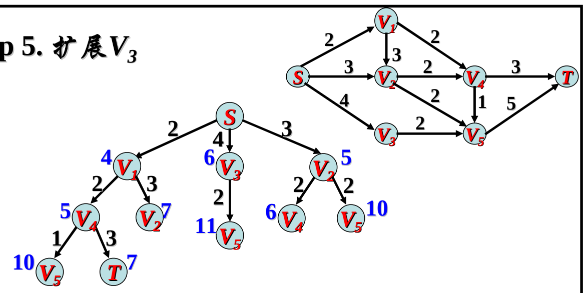
Step 4. 扩展 V_4



$g(V_5)=2+2+1=5$ $h(V_5)=min\{5\}=5$ $f(V_5)=5+5=10$

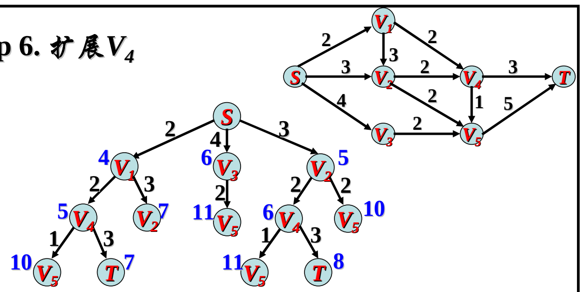
$g(T)=2+2+3=7$ $h(T)=0$ $f(V_2)=7+0=7$

Step 5. 扩展 V_3



$g(V_5)=4+2=6$ $h(V_5)=min\{5\}=5$ $f(V_5)=6+5=11$

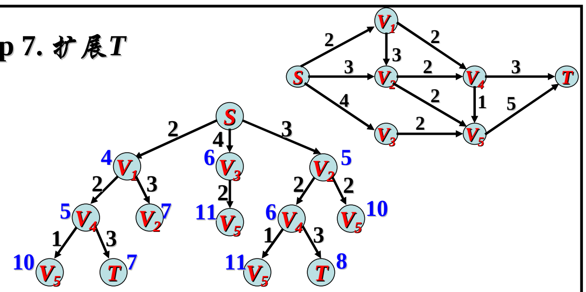
Step 6. 扩展 V_4



$g(V_5)=3+2+1=6$ $h(V_5)=min\{5\}=5$ $f(V_5)=6+5=11$

$g(T)=3+2+3=8$ $h(T)=0$ $f(T)=8+0=8$

Step 7. 扩展 T



因为T是目标节点，所以我们得到解：

$S \rightarrow V_1 \rightarrow V_4 \rightarrow T$

HIT CS&E

第八章

Randomized Algorithms

李建中
计算机科学与工程系

HIT CS&E

提要

HIT CS&E

参考文献

- 《计算机算法设计与分析》
  - 第7章
- 《Randomized Algorithms》

  Rajeev Motwani and Prabhakar, Raghavan, Cambridge University Press
- 《网站资料》
  - 第8章

HIT CS&E

8.1 *Introduction to Randomized Algorithms*

- 随机算法的基本概念
- 随机算法的分类
- 随机算法的性能分析方法

HIT CS&E

随机算法的基本概念

- 什么是随机算法
  - 随机算法是一种使用概率和统计方法在其执行过程中对于下一计算步骤作出随机选择的算法
- 随机算法的优越性
  - 对于有些问题: 算法简单
  - 对于有些问题: 时间复杂性低
  - 对于有些问题: 同时兼有简单和时间复杂性低
- 随机算法的随机性
  - 对于同一实例的多次执行, 效果可能完全不同
  - 时间复杂性的一个随机变量
  - 解的正确性和准确性也是随机的

HIT CS&E

随机算法的分类

- 随机数值算法
  - 主要用于数值问题求解
  - 算法的输出往往是近似解
  - 近似解的精确度与算法执行时间成正比
- Monte Carlo算法
  - 主要用于求解需要准确解的问题
  - 算法可能给出错误解
  - 获得精确解概率与算法执行时间成正比

HIT CS&E

- Las Vegas算法
  - 一旦找到一个解, 该解一定是正确的
  - 找到解的概率与算法执行时间成正比
  - 增加对问题反复求解次数, 可是求解无效的概率任意小
- Sherwood算法
  - 一定能够求得一个正确解
  - 确定算法的最坏与平均复杂性差别大时, 加入随机性, 即得到Sherwood算法
  - 消除最坏行为与特定实例的联系

HIT CS&E

随机算法的性能分析

- 随机算法分析的特征
  - 仅依赖于随机选择, 不依赖于输入的分布
  - 确定算法的平均复杂性分析:
    依赖于输入的分布
  - 对于每个输入都要考虑算法的概率统计性能
- 随机算法分析的目标
  - 平均时间复杂性: 时间复杂性随机变量的均值
  - 获得正确解的概率
  - 获得优化解的概率
  - 解的精确度估计

HIT CS&E

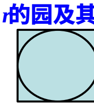
8.2 *Randomized Numerical Algorithms*

- 计算p值
- 计算定积分

HIT CS&E

计算p值

- 数学基础
  - 设有一个半径为*r*的园及其外切四边形



  - 向正方形随机地投掷*n*个点, 设*k*个点落入园内
  - 投掷点落入园内的概率为 $(pr^2)/(4r^2)=p/4$.
  - 用*k/n*逼近p/4, 即*k/n*»p/4, 于是p»(4k)/n.
  - 我们可以令*r=1*用投掷*n*个点的方法计算p

- 算法

  K=0;
  For *i=1* To *n*
  随机地产生四边形中的一点(*x*, *y*);
  If $x^2+y^2£1$ Then *k=k+1*;
  Endfor
  Return (4k)/n
- 时间复杂性=*O(n)*
  - 不是输入的大小, 而是随机样本的大小
- 解的精确度
  - 随着随机样本大小*n*增加而增加

HIT CS&E

计算定积分

- 问题
  - 计算积分 $\int_a^b g(x)dx$
- 数学基础
  - 令*f(x)*是区间[*a*, *b*]上的一组独立、同分布的随机变量$\{x_i\}$的任意密度函数
  - 令$g^*(x)=g(x)/f(x)$, 则$\{g^*(x)\}$是密度为*f(x)*的随机变量集合, 而且

$$E(g^*(x_i))=\int_a^b g^*(x)f(x)dx=\int_a^b g(x)dx=I$$

$$E(g^*(x_i)) = \int_a^b g^*(x) f(x) dx = \int_a^b g(x) dx = I$$

– 由强大数定律 $\Pr\left(\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} g^*(x_i) = I \right) = 1$

– 我们可以用 $\left(\frac{1}{n}\sum_{i=1}^{n} g^*(x_i) \right)$ 来近似计算 I

– 令 $f(x)=1/(b-a)$ $a \le x \le b$

– 索求积分可以由如下 I' 来近似计算 I

$$I' = \frac{1}{n}\sum_{i=1}^{n} g^*(x_i) = \frac{1}{n}\sum_{i=1}^{n} g(x_i)/f(x_i) = \frac{1}{n}\sum_{i=1}^{n}(b-a)g(x_i)$$

HIT CS&E

- **算法**

```
I=0;
For  i=1  To  n
    随机产生[a, b]中点x;
    I=I+g(x);
Endfor
Return  (b-a)*I/n
```

- 时间复杂性 $=O(n)$
  - 不是输入的大小, 而是随机样本的大小
- 解的精确度
  - 随着随机样本大小 n 增加而增加

HIT CS&E

8.3 Randomized Selection Algorithms

- 问题的定义
- 随机算法
- 算法的性能分析

HIT CS&E

问题的定义

- 输入: $S=\{x_1, x_2, \ldots, x_n\}$,
  整数 k, $1 \le k \le n$.
- 输出: S 中第 k 个最小元素.

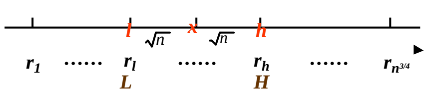
记号

$Rank(Q, t)$ = 集合 Q 中的元素 t 的rank
(第 k 小元素的rank是 k)
$\min(Q, i)$ = 集合 Q 中第 i 个最小元素.

HIT CS&E

随机算法

- 基本思想
  - 从 S 中随机地抽取 $n^{3/4}$ 个样本存入 R, 排序 R
  - S 中第 k 最小元素可能成为 R 中 $x=kn^{3/4}/n$ 最小元素
  - 为了解决误差问题, 我们考察区间 $[x-n^{1/2}, x+n^{1/2}]$



  - 把 S 中属于 $[L, H]$ 数据存入 P
  - 在 P 中查找 $\min(S, k)$

LAZYSELECT(S, k)

1. R=独立、均匀、可放回地从 S 随机选取的 $n^{3/4}$ 元素;
2. 在 $O(n\log n)$ 时间内排序 R;
3. $x=(k/n)n^{3/4}$; /* $(k/n)n^{3/4}=kn^{-1/4}$ */
4. $l=\max\{\lfloor x-\sqrt{n} \rfloor, 0\}$; $h=\min\{\lceil x+\sqrt{n} \rceil, n^{3/4}\}$;
5. $L=\min(R, l)$; $H=\min(R, h)$;
6. $L_p=Rank(S,L)$, $H_p=Rank(S,H)$; /*L 和 H 与 S 元素比较*/
7. $P=\{y \in S \mid L \le y \le H\}$;
8. If $\min(S, k) \in P$ and $|P| \le 4n^{3/4}+1$
   /* $\max(S, k) \in P$ 可由 $L_p \le k \le H_p$ 确定 */
9. Then 排序 P, $\min(S, k)=\min(P, (k-L_p))$, 算法结束;
10. ELSE goto 1.

HIT CS&E

算法的性能分析

• 数学基础
  - 数学期望
    - 离散随机变量X的数学期望$E[X]=\sum_i i\times P(X=i)$
    - 若$f(x)$是定义在整数集上的实数值函数，则

      $E[f(X)]=\sum_i f(i)\times P(X=i)$.
  - Markov不等式
    - $P(Y\ge t)\le E[Y]/t$, 其中Y为非负随机变量, $t>0$.

HIT CS&E

- 方差的性质与Chebyshev不等式
  - 方差$\sigma_x^2=E[(X-\mu_x)^2]$, μ_x为随机变量X的数学期望
  - σ_x称为标准差
  - Chebyshev不等式：$P(|X-\mu_x|>t\sigma_x)\le 1/t^2$
  - 如果随机变量X满足$P(X=1)=p$, $P(X=0)=1-p$, 则

    $\sigma_x^2=p(1-p)$.
  - 若$X=\sum_{1\le i\le n}X_i$, $\sigma_x^2=\sum_{1\le i\le n}\sigma_{x_i}^2$, X_i是独立随机变量
  - 若随机变量X_i满足$P(X_i=1)=p$, $P(X_i=0)=1-p$, 则

    $\sigma_x^2=np(1-p)$.

HIT CS&E

•算法的性能分析

定理. 算法执行1-9步一遍就可求出$min(S,k)$的概率是$1-O(n^{-1/4})$, 即算法需要$2n+O(n\log n)$次比较就可求出$min(S,k)$的概率是$1-O(n^{-1/4})$.

证明. 若算法执行1-9一遍可求出$min(S,k)$, 则第6步需$2n$次比较, 其他步需$O(n\log n)$次比较, 总需$2n+O(n\log n)$次比较.

往证算法执行1-9一遍可求出$min(S,k)$的概率是$1-O(n^{-1/4})$.

算法执行1-9一遍可求出$min(S,k)$的条件是:

(1). $min(S,k)$在L和H之间即P包含$min(S,k)$,

(2). $|P|\le 4n^{3/4}+1$.

我们首先来计算上述两个条件失败的概率.

A. 计算条件(1)不成立的概率

条件(1)不成立当且仅当$L>min(S,k)$或$H<min(S,k)$.

令$X_i=1$如果第i个随机样本$\le min(S,k)$, 否则$X_i=0$.

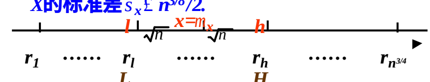
于是, $P(X_i=1)=k/n$, $P(X_i=0)=1-k/n$.

令 $X=\sum_{1\le i\le n^{3/4}}X_i$是$R$中小于等于$min(S,k)$的样本数. 我们有

X的数学期望$\mu_x=n^{3/4}k/n=kn^{-1/4}$,

X的方差$\sigma_x^2=n^{3/4}(k/n)(1-k/n)\le n^{3/4}/4$,

X的标准差$\sigma_x\le n^{3/8}/2$.



如果$L>min(S,k)$, $X<l$. 如果$H<min(S,k)$, $X\ge h$. 于是

$P(L>min(S,k))=P(X<l)=P(X<\mu_x-n^{1/2})=P(|X-\mu_x|>n^{1/2})$,

$P(H<min(S,k))=P(X\ge h)=P(X>h)+P(X=h)=P(|X-\mu_x|\ge n^{1/2})+(n^{3/4}+1)^{-1}$.

应用Chebyshev不等式, 又由$2n^{1/8}\sigma_x\le n^{1/2}$, 我们有

$P(|X-\mu_x|>n^{1/2})\le P(|X-\mu_x|>2n^{1/8}\sigma_x)\le 1/(2n^{1/8})^2=O(n^{-1/4})$. 于是

$P(L>min(S,k))=P(H<min(S,k))=O(n^{-1/4})$.

B. 计算P包含$min(S,k)$但$|P|\le 4n^{3/4}+2$不成立的概率

令$k_l=\max\{0,k-2n^{3/4}\}$, $k_h=\min\{k+2n^{3/4},n\}$.

"P包含$min(S,k)$但$|P|\le 4n^{3/4}+1$不成立"发生当且仅当

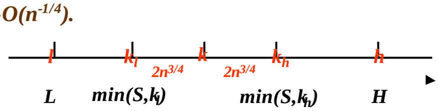
$L<min(S,k_l)$或$H>min(S,k_h)$.

类似与上面A中的分析，

$P(L<min(S,k_l))=P(H>min(S,k_h))=O(n^{-1/4})$.

由A和B, "算法执行1-9一遍就可以求出$min(S,k)$"不成立的概率是$O(n^{-1/4})$.

即，"算法执行1-9一遍就可以求出$min(S,k)$"的概率是$1-O(n^{-1/4})$.



HIT CS&E

8.4 Randomized Algorithm to Test Whether Number is Prime

- 问题的定义
- 随机算法设计
- 算法的性能分析

HIT CS&E

问题的定义

- 输入
  - 一个正整数N
- 输出
  - N是否素数

HIT CS&E

随机算法的设计

- 基本思想
  - 对N进行m次测试
  - 如果有一次测试成功, 则回答N是合数
  - 如果m次测试均失败, 则回答N是素数
  - 回答N是合数时, 答案百分之百正确
  - 回答N是素数时, 答案正确的概率是$1\text{-}2^{-m}$

- 随机算法

1. 随机地选择m个数$\{b_1, b_2, \ldots, b_m\}$, 满足

   1£ $b_1, b_2, \ldots, b_m$ £ N;

2. For $i=1$ To m Do
3. If $W(b_i)$成立 Then Return (N是合数);
4. Return (N是素数)

$W(b_i)$定义如下:

(1) $b_i^{N\text{-}1}$ ¹ $1 \bmod N$, 或

(2) $j[$(N-1)/2^j=k$是整数, $1 <$ (b_i^k与N的最大公因子) $< N$].

HIT CS&E

例1. 给定$N=12$. 选择测试数集$\{2, 3, 7\}$

测试 2: $2^{12\text{-}1} = 2048$ ¹ $1 \bmod 12$, $W(2)$成立.

N是合数.

HIT CS&E

例2. 给定$N=11$,选择测试数集$\{2, 5, 7\}$

测试 2: $2^{11\text{-}1} = 1024 = 1 \bmod 11$,

测试 5: $5^{11\text{-}1} = 9765625 = 1 \bmod 11$,

测试 7: $7^{11\text{-}1} = 282475249 = 1 \bmod 11$,

结论: 11可能是素数

答案正确的概率为$1\text{-}2^{-3}$

HIT CS&E

算法性能的分析

定理1. (1) 如果对于任意1£$b<N$, $W(b)$成立, 则N是合数.

(2) 如果N是合数, 则$(N\text{-}1)/2$£|{b | 1£$b<N$, $W(b)$}|

*(1)说明算法是正确的.

*(2)说明, 如果N是合数, 则至少一半$b(b<N)$使$W(b)$成立

定理2. 算法的回答"N是素数"正确的概率是$1\text{-}2^{-m}$.

HIT CS&E

8.5 *Randomized Sorting Algorithm*

- 问题的定义
- 随机算法
- 算法性能的分析

HIT CS&E

问题的定义

输入
- $S=\{x_1, x_2, \ldots, x_n\}$

•输出
- 排序的S

HIT CS&E

随机算法

- **基本思想**
  - 采用随机抽样的方法确定集合的划分点
  - 把集合划分为两个子集合
  - 分别递归地在每个子集合上使用随机排序算法

HIT CS&E

•算法

1. 均匀等可能地在S中随机抽取一个样本y;
2. 比较S中每个元素，把S划分为如下两个集合：

   $S_1=\{ x \mid x Î S, x<y \}$, $S_2=\{ x \mid x Î S, x>y \}$;
3. 递归地排序S_1和S_2;
4. 顺序输出排序的S_1, y, S_2;

HIT CS&E

算法性能的分析

•基本概念

- $S_{(i)}$表示S中阶为i的元素

  例如，$S_{(1)}$和$S_{(n)}$分别是最小和最大元素
- 随机变量X_{ij}定义如下：

  $X_{ij}=1$如果$S_{(i)}$和$S_{(j)}$在运行中被比较，否则为0
- X_{ij}是$S_{(i)}$和$S_{(j)}$的比较次数
- 算法的比较次数为$\sum_{i=1}^{n}\sum_{j>i} X_{ij}$
- 算法的平均复杂性为$E[\sum_{i=1}^{n}\sum_{j>i} X_{ij}]=\sum_{i=1}^{n}\sum_{j>i} E[X_{ij}]$

HIT CS&E

•计算$E[X_{ij}]$

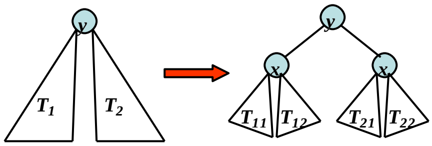
- 设p_{ij}为$S_{(i)}$和$S_{(j)}$在运行中比较的概率，则

  $E[X_{ij}]=p_{ij}´1+(1-p_{ij})´0=p_{ij}$

关键问题成为求解p_{ij}

•求解P_{ij}

我们可以用树表示算法的计算过程



•我们可以观察到如下事实:

- 一个子树的根必须与其子树的所有节点比较
- 不同子树中的节点不可能比较
- 任意两个节点至多比较一次

•当$S_{(i)}$, $S_{(i+1)}$, …, $S_{(j)}$在同一子树时, $S_{(i)}$和$S_{(j)}$才可能比较

- 由随机算法的特点, $S_{(i)}$, $S_{(i+1)}$, …, $S_{(j)}$在同一子树的概率为1
- 只有$S_{(i)}$或$S_{(j)}$被选为划分点时, $S_{(i)}$和$S_{(j)}$才可能比较
- $S_{(i)}$, $S_{(i+1)}$, …, $S_{(j)}$等可能地被选为划分点, 所以$S_{(i)}$和$S_{(j)}$进行比较的概率是: $2/(j-i+1)$, 即

$$p_{ij}=2/(j-i+1)$$

•现在我们有

$$\sum_{i=1}^{n}\sum_{j>i}E[X_{ij}]=\sum_{i=1}^{n}\sum_{j>i}p_{ij}=\sum_{i=1}^{n}\sum_{j>i}\frac{2}{j-i+1}$$

$$\le\sum_{i=1}^{n}\sum_{k=1}^{n-i+1}\frac{2}{k}\le 2\sum_{i=1}^{n}\sum_{k=1}^{n}\frac{1}{k}=2nH_n=O(n\log n)$$

定理. 随机排序算法的期望时间复杂性为O(nlogn)

8.6 A Min-Cut Algorithm

- 问题定义
- 随机算法
- 算法性能的分析

问题定义

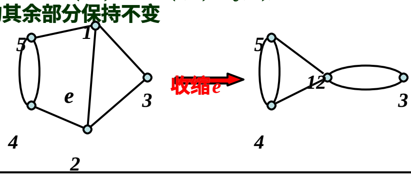
- 输入:
  - 无向多重连通图*G*
- 输出
  - 一个Min-Cut

- 图*G*的一个*Cut*是一组边, 从*G*中删除这个*Cut*将导致两个或多个连通分量
- *Cut*的大小是其边数, 多重边重复计算
- 最小*Cut*是具有最少边的*Cut*

随机算法

- 基本概念
  - *Cut*可以视为节点集的划分*V=(C, V-C)*, *Cut*是所有*G*中连接*C*和*V-C*的边集合.
  - 图*G*的边*(x, y)*的*contraction*:
    - 用新节点代替节点*x*和*y*或边*(x, y)*,
    - " vÎ V, 用边*(v, z)*代替边*(x, v)*或*(y, v)*,
    - *G*的其余部分保持不变

例如:

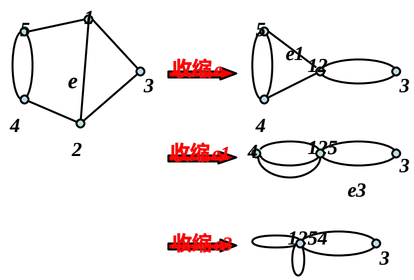


- 我们用$G/(x, y)$表示G的边(x, y)的收缩
- 边集合F Í G收缩记作G/F
- 图G/F的节点集合表示为V/F
- 图G/F的节点集合表示为E/F

- 随机算法

1. $H=G$;
2. While $|H(V)| > 2$ Do
3. 随机地从$H(E)$中选择一条边(x, y);
4. $F=F$È $\{(x, y)\}$;
5. $H=H/(x, y)$;
6. Cut=连接H中两个元节点的G的所有边.

例



$Cut=\{(1,3), (2, 3)\}$

算法的性能分析

定理1. 如果算法的输入是具有n个节点的多重图, 则算法的时间复杂性为$O(n^2)$.

证明. 一次边收缩需要$O(n)$时间.
至多进行O(n)次收缩.
于是, 算法时间复杂性为$O(n^2)$.

注意:

我们仅证明了在$O(n^2)$时间内算法能够求出一个Cut, 但是这个Cut不一定是优化的.

引理1. 如果k是min-cut的大小, 则G至少有$kn/2$条边.

证. 如果$|G(E)|<kn/2$, 则存在一个度小于k的节点p. 删除与p相关连的k条, 把G划分为两个连通分量, 其一是仅包含p.
于是, 与p相关连的边集合是一个cut.
但是这个cut的大小$<k$, 与min-cut大小为k矛盾.

引理2. 算法输出的cut是连接两个剩余节点的没有被收缩过的边.

证. 从算法定义可以看到, 算法输出的cut是连接两个剩余节点的没有被收缩的边的集合.

引理1. 如果k是图G的一个min-cut的大小, 则G至少有$kn/2$条边.

证. 如果$|G(E)|<kn/2$, 则存在一个度小于k的节点p. 删除与p相关连的k条, 把G划分为两个连通分量, 其一是仅包含p的连通分量.
于是, 与p相关连的边集合是一个cut.
但是这个cut的大小$<k$, 与min-cut大小为k矛盾.

定理2. 设C是一个min-cut, 其大小为k. 在算法结束时, C中无边被收缩过的概率大于$2/n^2$.

证. A_i表示第i步没有选中C的边, $1 \le i \le n-2$.

在第1步, 选中的边在C中的概率至多为$k/(kn/2)=2/n$, 即

$$Pr(A_1) \ge 1-2/n.$$

在第2步, 若A_1发生, 则至少有$k(n-1)/2$条边(每次收缩减少一个节点), 选中C中边的概率为$2/(n-1)$, 即

$$Pr(A_2 | A_1) \ge 1-2/(n-1).$$

在第i步, 若A_1至A_{i-1}发生, 则有$n-i+1$个节点, 即至少有$k(n-i+1)/2$条边, 于是

$$Pr(A_i | \cap_{1\le j\le i-1} A_j) \ge 1-2/(n-i+1)$$

最后我们有

$$Pr(\cap_{1\le i\le n-2} A_i) \ge \prod_{1\le i\le n-2}(1-2/(n-i+1))=2/n(n-1)>2/n^2$$

HIT
CS&E

推论1. 如果重复运行算法$n^2/2$次, 每次独立随机地选择收缩边, 不能发现一个min-cut的概率为

$$\left(1-\frac{2}{n^2}\right)^{n^2/2} < \frac{1}{e}$$

1. 考虑如下问题随机算法：

输入：$S=\{s_1,s_2,\ldots,s_n \mid s_i\in\boldsymbol{R}\}$

输出：$\min(S, k)$—S 中第 k 小的元素

Random_Select(S,k)

1. 从 S 中随机选择一个元素 s；
2. $S_1=\{s_i|\ s_i\in S,\ s_i<s\}$, $S_2=\{s_i|\ s_i\in S,\ s_i>s\}$;
3. IF $|S_1|=k-1$ THEN 返回 s;
4. ELSE IF $|S_1|>k$ THEN 返回 Random_Select(S_1, k);
5. ELSE 返回 Random_Select(S_2, $k-|S_1|$);

问题：

（1）该算法属于哪一类随机算法？

（2）证明：存在常数 $b<1$，使得算法递归过程中所考虑集合的大小的数学期望为 bn。

（3）证明：算法时间复杂度的数学期望为 $O(n)$。

2. 考虑最小割问题的随机算法：

输入：一个多重无向连通图 $G=(V,E)$；

输出：G 的一个最小边割。

Random_Mincut:

1. 为图 G 的任意边赋予一个随机独立的正权值；
2. 找出 G 的最小生成树 T；
3. 删除 T 中权值最大的一条边得到两棵树 T_1,T_2；
4. 令 T_1 的顶点集为 C，则 T_2 的顶点集为 $V-C$；
5. $cut=\{uv|\ uv\in E, u\in C, v\in V-C\}$
6. 输出 cut.

证明：算法输出一个最小割的概率为$\Omega(1/n^2)$.

提示：将上述算法与课堂上讲的算法关联起来。

3. 考虑简单连通图$G = (V;E)$上的最大独立子集问题的如下随机算法。

算法：IndependentSet()

输入：$G = (V;E)$

输出：$I\subseteq V$使得$\forall\ uv\in E$均有：$u\in I$ 或 $v\in I$

过程：1. 为V中每个顶点随机分配｛1,2,…,$|V|$｝中唯一标签，不同顶点具有不同标签；

2. $I\rightarrow\varnothing$, $S\leftarrow V$;

3. while $S\neq\varnothing$ do

4. $u\leftarrow S$中标签最小的顶点

5. $I\leftarrow I\cup\{u\}$

6. 从 S 中删除 u 和 u 的相邻顶点；

7. return I

将 IndependentSet 算法输出的集合记为 I。证明：

（1）I 是 $G = (V;E)$的一个独立集；

（2）对$\forall\ u\in V$, u 在 I 中的概率等于 $1/d_u$, 其中 d_u 表示 u 在 G 中的度。

4.设 $a_1,a_2,\ldots,a_n$ 是 n 个不同数构成的列表。如果 $i<j$ 且 $a_i>a_j$ 则称 a_i 和 a_j 是倒置的。冒泡排序算法的实质是不断交换列表中相邻的倒置元素，直到列表中没有倒置元素为止。假设冒泡排序算法的输入是一个随机排列，等可能地是 n!个排列中的任意一个。确定冒泡排序算法需要交换的倒置元素个数的数学期望。

5.有一个函数 $F:\{0,1,\ldots,n-1\}\to\{0,1,\ldots,m-1\}$，且 $F((x+y) \bmod n) = F(x)+F(y) \bmod m$ 对 $\forall x,y\in\{0,1,\ldots,n-1\}$成立。设 $F(x)$存储在一个数组中，数组下标表示自变量的值，数组元素的值表示函数值；由于某种意外，数组中 1/5 的函数值被恶意串改。试设计一个随机算法使其对$\forall z\in\{0,1,\ldots,n-1\}$算法能够以大于 1/2 的概率计算出正确的 $F(z)$。如果运行算法 3 次，你应该返回什么样的值，此时算法得到正确 $F(z)$的概率有什么变化？

HIT CS&E

第九章
Approximation Algorithm

骆吉洲
计算机科学与工程系

HIT CS&E

提要

HIT CS&E

参考资料

《Introduction to Algorithms》
- 第35章

《网站资料》
- 第9章

HIT CS&E

9.1 *Introdution*

- 近似算法的基本概念
- 近似算法的性能分析

HIT CS&E

近似算法的基本概念

- 近似算法的基本思想
  - 很多实际应用中问题都是NP-完全问题
  - NP-完全问题的多项式算法是难以得到的
  - 求解NP-完全问题的方法：
    - 如果问题的输入很小，可以使用指数级算法圆满地解决该问题
    - 否则使用多项式算法求解问题的近似优化解
  - 什么是近似算法
    - 能够给出一个优化问题的近似优化解的算法
    - 近似算法主要解决优化问题

HIT CS&E

近似算法的性能分析

- 近似算法的时间复杂性
  - 分析目标和方法与传统算法相同
- 近似算法解的近似度
  - 本节讨论的问题是优化问题
    - 问题的每一个可能的解都具有一个正的代价
    - 问题的优化解可能具有最大或最小代价
    - 我们希望寻找问题的一个近似优化解
  - 我们需要分析近似解代价与优化解代价的差距
    - Ratio Bound
    - 相对误差
    - $(1+\varepsilon)$-近似

- **Ratio Bound**

定义1(Ratio Bound) 设A是一个优化问题的近似算法, A具有ratio bound $p(n)$, 如果

$$\max\left\{\frac{C}{C^*},\frac{C^*}{C}\right\}\le p(n)$$

其中n是输入大小, C是A产生的解的代价, C^*是优化解的代价.

- 如果问题是最大化问题, max{C/C*, C*/C}=C*/C
- 如果问题是最小化问题, max{C/C*, C*/C}=C/C*
- 由于C/C*<1当且仅当C*/C >1, Ratio Bound不会小于1
- Ratio Bound越大, 近似解越坏

- 相对误差

定义2(相对误差) 对于任意输入, 近似算法的相对误差定义为|C-C*|/C*, 其中C是近似解的代价, C*是优化解的代价.

定义3(相对误差界) 一个近似算法的相对误差界为$\varepsilon(n)$, 如果|C-C*|/C*≤ $\varepsilon(n)$.

结论1. $\varepsilon(n) \le p(n)-1$.

证. 对于最小化问题

$\varepsilon(n)$=|C-C*|/C*=(C-C*)/C*=C/C* -1=p(n)-1.

对于最大化问题

$\varepsilon(n)$=|C-C*|/C*=(C*-C)/C*= (C*/C -1)/(C*/C)
= (p(n)-1)/p(n) ≤ p(n)-1.

- 对于某些问题, $\varepsilon(n)$和p(n)独立于n, 用p和ε表示之.
- 某些NP-完全问题的近似算法满足: 当运行时间增加时, Ratio Bound和相对误差将减少.
- 结论1表示, 只要求出了Ratio Bound就求出了$\varepsilon(n)$

- 近似模式

定义4 (近似模式) 一个优化问题的近似模式是一个以问题实例I和ε>0为输入的算法. 对于任意固定ε, 近似模式是一个(1+ε)-近似算法.

定义5 一个近似模式A(I, ε)称为一个多项式时间近似模式, 如果对于任意ε>0, A(I, ε)的运行时间是|I|的多项式.

定义 6 一个近似模式称为完全多项式时间近似模式, 如果它的运行时间是关于1/ε和输入实例大小n的多项式.

HIT CS&E

9.2 The Vetex-cover Problem

- 问题的定义
- 近似算法的设计
- 算法的性能分析

HIT CS&E

问题的定义

输入: 无向图G=(V, E)

输出: C⊆V, 满足

(1). ∀(u, v)∈E, u∈C或者v∈C

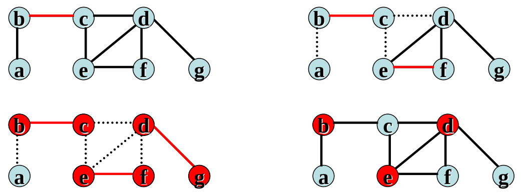
(2). C是满足条件(1)的最小集合。

理论上已经证明优化结点覆盖问题是NP-完全问题.

HIT CS&E

近似算法的设计

- 算法的基本思想



算法解：{b, c, e, f, d, g}　　优化解：{b, e, d}

HIT CS&E

- 算法

APPROX-Vertex-Cover (G)

1. $C=0$
2. $E'=E[G]$;
3. While $E'\neq 0$ DO
4. 　任取$(u, v)\in E'$;
5. 　$C=C\cup\{u, v\}$;
6. 　从E'中删除所有与u或v相连的边;
7. Roturn C

HIT CS&E

算法的性能分析

- 时间复杂性
  $T(G)=O(|E|)$
- Ratio Bound

定理. Approx-Vertex-Cover的Ratio Bound为2.

证. 令$A=\{(u, v) \mid (u, v)$是算法第4步选中的边$\}$.
若$(u,v)\in A$, 则与(u, v)邻接的边皆从E'中删除.
于是, A中无相邻接边.
第5步的每次运行增加两个结点到C, $|C|=2|A|$.
设C^*是优化解, C^*必须覆盖A.
由于A中无邻接边, C^*至少包含A中每条边的一个结点. 于是,
$|A|\leq|C^*|$, $|C|=2|A|\leq 2|C^*|$, 即$|C|/|C^*|\leq 2$.

HIT CS&E

9.3 The Traveling-salesman Problem

- ➢问题的定义
- ➢近似算法设计
- ➢算法的性能分析

HIT CS&E

问题的定义

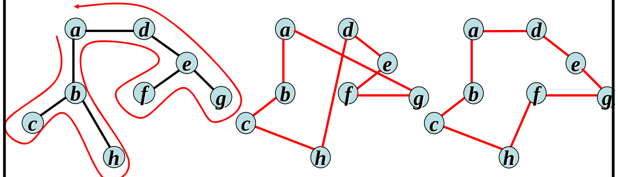
- 输入
  完全无向图$G=(V,E)$;
  代价函数$C: E\rightarrow$非负整数集合
  C满足三角不等式:
  $C(u,w)\leq C(u,v)+C(v,w)$.
- 输出
  具有最小代价的Hamilton环

- Hamilton环是一个包含V中每个结点一次的简单环.
- 代价函数的扩展: 设$A\subseteq E$, $C(A)=\sum_{(u,v)\in E} C(u, v)$.
- 不满足三角不等式的TSP问题无具有常数Ration Bound的近似算法, 除非NP=P.

HIT CS&E

近似算法的设计

- 基本思想
  - 首先构造最小生成树(可以使用第五章的算法)
  - 先序遍历最小生成树, 构造TSP的解



先序遍历: abchdefga　　先序遍历解　　优化解

HIT CS&E

- 近似算法

APPROX-TSP-TOUR(G,C)

1. 选择一个$r\in V[G]$作为生成树的根;
2. 调用MST-Prim(G, C, r)生成一个最小生成树T;
3. 先序遍历T, 形成有序结点表L;
4. 按照L中的顺序访问各结点, 形成哈密顿环.

HIT CS&E

算法的性能分析

- 时间复杂性

第2步: $O(|E|+|V|\log|V|)=O(|V|^2+|V|\log|V|)=O(|V|^2)$

第3步: $O(|E|)=O(|V|^2)$, 因为G是完全图,

第4步: $O(|V|)$

$T(G)=O(|V|^2)$

HIT CS&E

- 解的精确度

定理1. APPROX-TSP-TOUR具有Ratio Bound 2.

证.

设H^*是TSP问题的优化解, H是算法产生的近似解.我们需要证明$C(H)\leq 2C(H^*)$.

从H^*中删除任意一条边, 可以得到G的一个生成树T'.

设T是算法第2步产生的导致H的最小生成树, 则

$C(T)\leq C(T')\leq C(H^*)$.

T的一个full walk W列出了所有结点(第一次访问的和以后从一个子树返回时再访问的). 前面例子的full walk给出顺序: a,b,c,b,h,b,a,d,e,f,e,g,e,d,a

HIT CS&E

由于W通过每条边两次, $C(W)=2C(T)$, 进而$C(W)\leq 2C(H^*)$.

W不是哈密顿环, 因为它通过某些结点多于一次.

根据三角不等式, 我们可以从W中删除对一个结点的任何访问, 而不增加代价. (例如: 从$u\rightarrow v\rightarrow w$ 删除v得$u\rightarrow w$)

反复地应用上述操作, 我们可以从W中删除所有对任何结点的非第一次访问, 得到一个算法中的preoder walk.

在我们的例子中, 操作结果是: a, b, c, h, d, e, f, g.

由于T的preoder walk导致H, 我们有$C(H)\leq C(W)$, 即

$C(H)\leq 2C(H^*)$,

明所欲证.

HIT CS&E

近似求解一般TSP问题的难度

定理2. 如果P≠NP, $p>1$, 则不存在Ratio Bound为p的求解一般TSP问题的多项式时间近似算法。

证. 反证法

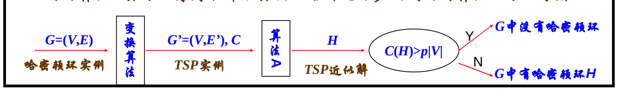
设P≠NP且对某$p>1$, 存在近似比为p的一般TSP问题的多项式近似算法A.

令$G=(V,E)$是哈密顿回路问题的任意一个实例, 则可以如下构造TSP的一个实例$G'=(V,E')$, 其中$E'=\{(u,v)|u,v\in V, u\neq v\}$, $C(u,v)=1$若$uv\in E$, 否则$C(u,v)=p|V|+1$.显然, 可以在$|V|$和$|E|$的多项式时间内由G建立出G'和C.

一方面, 如果G中具有一个哈密顿环H, 则C分配给H的每条边的代价为1, 因此G'包含一个代价为$|V|$的TSP回路。

另一方面, 如果G中不存在哈密顿环, 则G'中的TSP回路至少有一条边不在E中。故, G'的任意TSP回路的代价至少为$(p|V|+1)+(|V|-1)>p|V|$.

利用算法A去可以得到求解哈密顿回路问题的多项式时间算法如下。矛盾



HIT CS&E

9.4 The Set-covering Problem

- 问题的定义
- 近似算法的设计
- 算法的性能分析

HIT
CS&E

问题的定义

- **输入:**

  有限集X, X的所有子集族F, $X=\cup_{S\in F} S$

- **输出:**

  $C\subseteq F$，满足

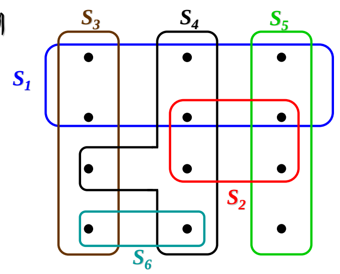
  (1). $X=\cup_{S\in C} S$,

  (2). C是满足条件(1)的最小集族, 即$|C|$最小.

*最小集合覆盖问题是很多实际问题的抽象.

*最小集合覆盖问题是NP-完全问题.

- 问题的实例



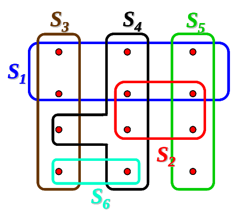
X=12个黑点， $F=\{S_1, S_2, S_3, S_4, S_5, S_6\}$

优化解$C=\{S_3, S_4, S_5\}$

HIT
CS&E

近似算法的设计

- **基本思想**
  - 贪心选择:选择能覆盖最多未被覆盖元素的子集



$C=\{S_1, S_4, S_5, S_3\}$

HIT
CS&E

- **算法**

Greedy-Set-Cover(X, F)

1. $U\leftarrow X$; /* U是X中尚未被覆盖的元素集 */
2. $C\leftarrow\theta$;
3. While $U\neq\theta$ Do
4. Select $S\in F$ 使得$|S\cap U|$最大;

   /* Greedy选择—选择能覆盖最多U元素的子集S */
5. $U\leftarrow U-S$;
6. $C\leftarrow C\cup\{S\}$; /* 构造X的覆盖 */
7. Return C.

HIT
CS&E

算法性能的分析

- **时间复杂性**
  - 3-6的循环次数至多为$min(|X|, |F|)$
  - 计算$|S\cap U|$需要时间$O(|X|)$
  - 第4步需要时间$O(|F||X|)$
  - $T(X,F)=O(|F||X|min(|x|,|F|))$

- **Ration Bound**

定理1. 令$H(d)=\sum_{1\leq i\leq d}1/I$. Greedy-Set-Covers是多项式$p(n)$-近似算法, $p(n)=H(max\{|S|\mid S\in F\})$.

证. 我们已经算法是多项式算法, 仅需计算Ratio Bound.
设C^*是优化集合覆盖, C^*的代价是$|C^*|$.
令S_i是由Greedy-Set-Cover选中的第i个子集.
当把S_i加入C时, C的代价加1. 我们把选择S_i增加的代价均匀分配到由S_i首次覆盖的所有结点.
$\forall x\in X$, 令c_x是分配到x的代价. 若x被S_i首次覆盖, 则

$$c_x=\frac{1}{|S_i-(S_1\cup S_2\cup\ldots\cup S_{i-1})|}$$

HIT
CS&E

显然，算法给出的解C的代价为$|C|$, $|C|$平均地分布到X的所有点. 由于C^*也覆盖X, 我们有

$$|C| = \sum_{x \in X} c_x \le \sum_{S \in C^*} \sum_{x \in S} c_x$$

注意：上式的小于成立是因为C^*中各子集可能相交，某些c_x被加了多次，而左式每个c_x只加一次.

如果$\forall S \in F$, $\sum_{x \in S} c_x \le H(|S|)$成立，则

$$|C| \le \sum_{S \in C^*} H(|S|) \le |C^*| \cdot H(max\{|S| \mid S \in F\}),$$

即$|C|/|C^*| \le H(max\{|S| \mid S \in F\})$, 定理成立.

下边我们来证明：对于$\forall S \in F$, $\sum_{x \in S} c_x \le H(|S|)$.

HIT
CS&E

对于$\forall S \in F$和$i=1,2,\ldots|C|$, 令$u_i=|S-(S_1 \cup S_2 \cup \ldots \cup S_i)|$是$S_1$、$S_2$、…、$S_i$被选中后，$S$中未被覆盖的点数. S_i先于S被选中.

令$u_0=|S|$, k是满足下列条件的最小数：$u_k=0$, 即S中每个元素被S_1、S_2、…、S_k中至少一个覆盖.

显然，$u_{i-1} \ge u_i$, $u_{i-1}-u_i$是S中由S_i第一次覆盖的元素数.于是，

$$\sum_{x \in S} c_x = \sum_{i=1}^{k} (u_{i-1} - u_i) \bullet \frac{1}{|S_i - (S_1 \cup S_2 \cup \ldots \cup S_{i-1}|}$$

注意：$|S_i-(S_1 \cup S_2 \cup \ldots \cup S_{i-1})| \ge |S-(S_1 \cup S_2 \cup \ldots \cup S_{i-1}| = u_{i-1}$，因为Greed算法保证：$S$不能覆盖多于$S_i$覆盖的新结点数，否则$S$将在$S_i$之前被选中. 于是，

$$\sum_{x \in S} c_x \le \sum_{i=1}^{k} (u_{i-1} - u_i) \bullet \frac{1}{u_{i-1}}$$

HIT
CS&E

$$\begin{aligned}
\sum_{x \in S} c_x &\le \sum_{i=1}^{k} (u_{i-1} - u_i) \bullet \frac{1}{u_{i-1}} \\
&= \sum_{i=1}^{k} \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{u_{i-1}} \\
&\le \sum_{i=1}^{k} \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{j} \qquad (\because j \le u_{i-1}) \\
&= \sum_{i=1}^{k} \left(\sum_{j=1}^{u_{i-1}} \frac{1}{j} - \sum_{j=1}^{u_i} \frac{1}{j} \right) \\
&= \sum_{i=1}^{k} H(u_{i-1}) - H(u_i) \\
&= H(u_0) - H(u_k) \\
&= H(u_0) - H(0) \qquad (\because u_k = 0) \\
&= H(u_0) = H(|S|) \qquad (\because H(0) = 0,\ u_0 = S)
\end{aligned}$$

HIT
CS&E

复杂性分析

推论1. Greedy-Set-Cover是一个多项式$\ln(|x|+1)$-近似算法.

证. 由不等式$H(n) \le \ln(n+1)$可知

$H(max\{|S| \mid S \in F\}) \le H(|X|) \le \ln|X|+1$.

HIT
CS&E

9.5 Randomization and Linear Programming

- 求解Max-3-CNF问题随机近似算法
- 求解最小节点覆盖问题的线性规划算法

HIT
CS&E

求解Max-3-CNF问题随机近似算法

- 基本概念

定义1. 设C是随机近似算法RAS产生的问题P的近似解的代价，C^*是问题P的准确解的代价，n是P的大小. 若$max(C/C^*, C^*/C) \le p(n)$, 则称$RAS$具有近似比$p(n)$. 我们也称$RAS$是一个随机$p(n)$-近似算法.

HIT CS&E

- Max-3-CNF问题的定义

输入：合取范式*CNF*，

每个析取式具有三个变量，

没有任何变量和它的非在同一个析取式中

输出：一个变量赋值，最大化值为1的析取式个数

- 随机算法

Random-Max-3-CNF(*CNF*)

1. For 对于*CNF*中的每个变量*x* Do
2. 随机地为x赋值：*x* =0的概率为1/2, *x* =1的概率为1/2;
3. Return.

- 性能分析

定理. Random-Max-3-CNF是一个随机8/7-近似算法.

证.

假定输入*CNF*中具有*n*个变量，*m*个析取式，第i个析取式的形式为 $x_{i1} \vee x_{i2} \vee x_{i3}$.

对$i=1, 2, \ldots, m$，定义随机变量：

$Y_i=1$ 如果第i个析取式为1，否则$Y_i=0$.

Pr(第i个析取式为0)$=Pr(x_{i1}=0)Pr(x_{i2}=0)Pr(x_{i3}=0)=(1/2)^3=1/8$.

Pr(第i个析取式为1)$=1- 1/8 = 7/8$.

$E[Y_i]=7/8$.

令$Y=Y_1+Y_2+\ldots+Y_m$. Y是*CNF*中值为1的析取式的个数.

$E[Y]=\sum_{1\leq i\leq m}E[Y_i]=\sum_{1\leq i\leq m}7/8=m7/8$.

显然，优化解的代价为m. 于是近似比$=m/(m7/8)=8/7$.

HIT CS&E

求解节最小点覆盖问题的线性规划算法

- 问题的定义
  - 输入：无向图$G=(V, E)$，每个节点具有权$w(v)$.
  - 输出：$C \subseteq V$，满足

    (1). $\forall (u, v)\in E$, $u\in C$或者$v\in C$

    (2). $w(C)$最小, $w(C)=\sum_{c\in C}w(c)$.

以前的节点覆盖算法不再适用!

HIT CS&E

- 问题转化为0-1线性规划问题$P_{0\text{-}1}$
  - 对于 $\forall v\in V$, 定义 $x(v)\in\{0, 1\}$如下：
    - 若v在节点覆盖中，则$x(v)=1$，否则$x(v)=0$.
    - $\forall (u, v)\in E$，若u、v或两者在覆盖中，则$x(u)+x(v)\geq 1$.
  - 对应的0-1整数规划问题$P_{0\text{-}1}$
    - 优化目标：最小化 $\sum_{v\in V}w(v)x(v)$
    - 约束条件：$x(u)+x(v)\geq 1$ for $\forall v\in V$

      $x(v)\in\{0, 1\}$ for $\forall v\in V$
  - 0-1整数规划问题是NP-完全问题
  - 我们需要设计近似算法

HIT CS&E

- 用线性规划问题的解近似0-1规划问题的解
  - 对于 $\forall v\in V$, 定义 $x(v)\in[0, 1]$
  - $P_{0\text{-}1}$对应的线性规划问题*LP*
    - 优化目标：最小化 $\sum_{v\in V}w(v)x(v)$
    - 约束条件：$x(u)+x(v)\geq 1$ for $\forall v\in V$

      $x(v)\in[0, 1]$ for $\forall v\in V$
  - 线性规划问题具有多项式时间算法
  - $P_{0\text{-}1}$的可能解是*LP*问题的可能解
  - $P_{0\text{-}1}$解的代价≥*LP*的解的代价

HIT CS&E

- 近似算法

Approx-Min-VC(*G*, *w*)

1. $C=0$;
2. 计算*LP*问题的优化解*y*;
3. For each $v\in V$ Do
4. If $x(v)\geq 1/2$ Then $C=C\cup\{v\}$;

   /* 用四舍五入法把LP的解近似为$P_{0\text{-}1}$的解 */
5. Return *C*.

HIT CS&E

- 算法的性能

定理. Approx-Min-VC是一个多项式时间2-近似算法

证.

由于求解LP需多项式时间，Approx-Min-VC的For循环需要多项式时间，所以算法需要多项式时间.

下边证明Approx-Min-VC的近似比是2.

往证算法产生的C是一个节点覆盖.

$\forall(u, v)\in E$，由约束条件可知 $x(u)+x(v)\geq 1$. 于是，$x(u)$和$x(v)$至少一个大于等于1/2，即u、v或两者在C中. C是一个覆盖.

HIT CS&E

往证$w(C)/w(C^*)\leq 2$.

令C^*是P_{0-1}的优化解，z^*是LP优化解的代价. 因为C^*是LP的可能解，$w(C^*)\geq z^*$.

$$\begin{aligned} z^* = \textstyle\sum_{v\in V} w(v)x(v) &\geq \textstyle\sum_{v\in V:\ x(v)\geq 1/2} w(v)x(v) \\ &\geq \textstyle\sum_{v\in V:\ x(v)\geq 1/2} w(v)1/2 \\ &= \textstyle\sum_{v\in C} w(v)1/2 \\ &= (1/2)\textstyle\sum_{v\in C} w(v) \\ &= (1/2)w(C). \end{aligned}$$

由$w(C^*)\geq z^*$, $w(C^*)\geq(1/2)w(C)$, 即 $w(C)/w(c^*)\leq 2$.

HIT CS&E

9.6 The Subset-sum Problem

- 问题的定义
- 指数时间算法
- 完全多项式时间近似模式

HIT CS&E

问题定义

- 输入:

  (S, t), $S=\{x_1, x_2, ..., x_n\}$,

  x_i和t均是正整数

- 输出:

  $\sum_{x\in A} x$, 满足:

  $A\subseteq S$, $\sum_{x\in A} x\leq t$

  $\sum_{x\in A} x = max\{\sum_{x\in B} x \mid B\subseteq S\}$

HIT CS&E

指数时间算法

- 算法

(设S是集合, x是正整数, 定义$S+x=\{s+x \mid s\in S\}$)

Exact-Subset-Sum($S=\{x_1, x_2, ..., x_n\}$, t)

1. $n\leftarrow|S|$;
2. $L_0\leftarrow<0>$;
3. For $i\leftarrow 1$ To n Do
4. $\quad L_i\leftarrow$Merge-List(L_{i-1}, $L_{i-1}+x_i$);
5. $\quad$删除L_i中所有大于t的元素;
6. Return L_n中最大元素.

HIT CS&E

- 计算过程:
  - $L_0=<0>$
  - $L_1=<0, x_1>$ /* 前一个元素所有子集的和(不大于t) */
  - $L_2=<0, x_1, x_2, x_1+x_2>$ /* 前二个元素所有子集的和(不大于t) */
  - $L_3=<0, x_1, x_2, x_1+x_2, x_3, x_1+x_3, x_2+x_3, x_1+x_2+x_3>$ /* 前三个元素所有子集的和(不大于t) */
  - L_i=前i个元素所有子集的和(不大于t)

对n作数学归纳法可以证明:
L_n=前n个元素所有子集的和(不大于t)

HIT CS&E

- 时间复杂性

第4步：$|L_i|=2|L_{i-1}|=2^2|L_{i-2}|=\ldots=2^i|L_0|=2^i$

$T(n)=O(2^n)$ 如果t比较大

1. $n \leftarrow |S|$;
2. $L_0 \leftarrow <0>$;
3. For $i \leftarrow 1$ To n Do
4. $L_i \leftarrow$ Merge-List(L_{i-1}, $L_{i-1}+x_i$);
5. 删除L_i中所有大于t的元素；
6. Return L_n中最大元素.

HIT CS&E

完全多项式时间近似模式

- 基本思想：

修剪L，对于多个相近元素，只留一个代表，尽量缩小每个L的长度

– 设$\delta(0<\delta<1)$是修剪参数，根据δ修剪L:

(1). 从L中删除尽可能多的元素，

(2). 如果L'是L修剪后的结果，则对每个从L中删除的元素y，L'中存在一个元素$z \le y$，使得

$$(1-\delta)y \le z \le y$$

– 如果y被修剪掉，则存在一个代表y的z在L中，而且z相对于y的相对误差小于δ.

- 修剪算法

Trim($L=\{y_1, y_2, \ldots, y_m\}$, δ) /* $y_i \le y_{i+1}$, $0<\delta<1$, 输出缩小的表L'. */

$m \leftarrow |L|$;

$L' \leftarrow <y_1>$;

$last \leftarrow y_1$;

For $i \leftarrow 2$ To m Do

If $last<(1-\delta)y_i$

/* 即$y_{i-1}<(1-\delta)y_i$, 由L和L'有序，对$\forall y \in L'$，不满足$(1-\delta)y_i \le y \le y_i$ */

Then y_i加入到L'末尾； /* 因L'中目前没有能够表示y_i的元素 */

$last \leftarrow y_i$;

Return L'.

- 复杂性：$O(|L|)=O(m)$

- 完全多项式近似模式

输入：$S=\{x_1, x_2, \ldots, x_n\}$, $t \ge 0$, $0<\varepsilon<1$

输出：近似解z

Approx-Subset-Sum(S, t, ε)

1. $n \leftarrow |S|$;
2. $L_0 \leftarrow <0>$
3. For $i \leftarrow 1$ To n Do
4. $L_i \leftarrow$ Merge-List(L_{i-1}, $L_{i-1}+x_i$);
5. $L_i \leftarrow$ Trim(L_i, ε/n) /* 修剪参数 $\delta=\varepsilon/n$ */
6. 从L_i中删除大于t的元素；
7. 令z是L_n中最大值；
8. Return z.

- 性能分析

定理1. Approx-Subset-Sum是子集求和问题的一个完全多项式时间近似模式.

证. 令$P_0=\{0\}$, $P_i=\{x \mid x=\sum_{y \in A} y, A \subseteq \{x_1, x_2, \ldots, x_i\}\}$. 例如，

令$S=\{1, 4, 5\}$, 则

$P_1=\{0, 1\}$,

$P_2=\{0, 1, 4, 5\}$,

$P_3=\{0, 1, 4, 5, 6, 9, 10\}$.

使用数学归纳法可以证明：$P_i=P_{i-1} \cup (P_{i-1}+x_i)$.

使用数学归纳法可以证明L_i是P_i中所有不大于t的元素的有序表.

L_i经第5步修剪以及第6步的大于t元素的删除，仍然有$L_i \subseteq P_i$. 于是，第8步返回的z是S的某个子集的和. 我们需证明

(1). $C^*(1-\varepsilon) \le z$, 即$(C^*-z)/C^* \le \varepsilon$, C^*是优化解，z是近似解.

注意，由于子集合求和问题是最大化问题，$(C^*-z)/C^*$是算法的相对误差.

(2). 算法是关于$|S|$和$1/\varepsilon$的多项式时间算法.

(1). 往证$C^*(1-\varepsilon)\le z$

对i作归纳法证明: $\forall y\in P_i, y\le t$, 存在一个$z'\in L_i$使$(1-\varepsilon/n)^i y\le z'\le y$.

当$i=0$时$P_i=\{0\}$, $L_i=\{0\}$, 命题成立.

设当$i\le k$时命题成立. $P_{k+1}=P_k\cup\{P_k+x_{k+1}\}$.

由归纳假设, $\forall y\in P_{k+1}\cap P_k, y\le t$, 存在$z'\in L_k\subseteq L_{k+1}$使

$$(1-\varepsilon/n)^k y\le z'\le y.$$

于是, $(1-\varepsilon/n)^{k+1}y\le z'\le y$.

对于$\forall y'\in P_{k+1}-P_k$, $y'=y+x_{k+1}\le t$, $y\in P_k$. 由归纳假设, 存在$z'\in L_k\subseteq L_{k+1}$使$(1-\varepsilon/n)^k y\le z'\le y$. 于是,

$$(1-\varepsilon/n)^k y+x_{k+1}\le z'+x_{k+1}\le y+x_{k+1}.$$

由于$z'\in L_k$, $z'+x_{k+1}\in L_{k+1}$, 而且

$$((1-\varepsilon/n)^k y+x_{k+1})-((1-\varepsilon/n)^{k+1}(y+x_{k+1}))$$
$$=(1-\varepsilon/n)^k(y-(1-\varepsilon/n)y)+(x_{k+1}-(1-\varepsilon/n)^{k+1}x_{k+1})>0,$$

即$(1-\varepsilon/n)^{k+1}(y+x_{k+1})\le(1-\varepsilon/n)^k y+x_{k+1}\le z'+x_{k+1}\le y+x_{k+1}$.

最后, 若$C^*\in P_n$是子集合求和问题的优化解, 则存在一个$z'\in L_n$, 使$(1-\varepsilon/n)^n C^*\le z'\le C^*$. 因算法解$z=\max(L_n)$, $(1-\varepsilon/n)^n C^*\le z'\le z\le C^*$.

由于$(1-\varepsilon/n)^n$的一阶导数大于0, $(1-\varepsilon/n)^n$是关于n递增的函数. 因为$n>1$, $(1-\varepsilon)<(1-\varepsilon/n)^n$.

于是, $(1-\varepsilon)C^*\le z$, 即近似解z与优化解的相对误差不大于ε.

(2). 往证算法的时间复杂性是n与$1/\varepsilon$的多项式

先计算$|L_i|$的上界. 修剪后, L_i中的相邻元素z和z'满足:

$$z'<(1-\varepsilon/n)z,\ 即\ z/z'>1/(1-\varepsilon/n).$$

如果L_i中具有$k+2$个元素, 则必有$y_0=0$, $y_1=z_0$, $y_2>z_0\cdot 1/(1-\varepsilon/n)$, $y_3>z_0\cdot 1/(1-\varepsilon/n)^2,\ldots, y_{k+1}>z_0\cdot 1/(1-\varepsilon/n)^k$, 而且$z_0\cdot 1/(1-\varepsilon/n)^k\le t$.

由$z_0\cdot 1/(1-\varepsilon/n)^k\le t$, $k\le\log_{1/(1-\varepsilon/n)}t$, 对$\log_{1/(1-\varepsilon/n)}t$, 台劳展开$\ln(1-\varepsilon/n)$,

$$|L_i|=k+2\le 2+\log_{1/(1-\varepsilon/n)}t=2+(\ln t/-\ln(1-\varepsilon/n))\le 2+n\ln t/\varepsilon.$$

算法的运行时间是$|L_i|$的多项式, 即n和$1/\varepsilon$的多项式.

HIT CS&E

9.7 近似算法中的线性规划方法

- 线性规划与对偶原理
- Min-max关系
- 用线性规划方法设计近似算法的两类方法
- 应用实例: 集合覆盖问题的近似算法

HIT CS&E

线性规划

线性规划问题: 在约束条件为线性表达式的前提下对一个线性目标函数进行优化

例1 minimize $7x_1+x_2+5x_3$

subject to $x_1-x_2+3x_3\ge 10$

$5x_1+2x_2-x_3\ge 6$

$x_1,x_2,x_3\ge 0$

maximize $10y_1+6y_2$

subject to $y_1+5y_2\le 7$

$-y_1+2y_2\le 1$

$3y_1-y_2\le 5$

$-y_1,-y_2\le 0$

线性规划的一般形式

最小化问题	最大化问题
min cx	max by
st. $Ax\ge b$	st. $By\le c$

HIT CS&E

满足所有约束条件的一组变量称为线性规划问题的**可行解**

使得目标函数达到最优取值的可行解称为线性规划问题的**最优解**

minimize $7x_1+x_2+5x_3$

subject to $x_1-x_2+3x_3\ge 10$

$5x_1+2x_2-x_3\ge 6$

$x_1,x_2,x_3\ge 0$

$x=(2,1,3)$是上述线性规划问题的一个可行解; $x=(7/4,0,11/4)$是上述线性规划问题的最优解, 目标函数的最优值为26.

线性规划问题可以在多项式时间内求解: Karmarkar算法

HIT CS&E

线性规划问题的对偶问题

minimize $7x_1+x_2+5x_3$

subject to $x_1-x_2+3x_3\ge 10$

$5x_1+2x_2-x_3\ge 6$

$x_1,x_2,x_3\ge 0$

maximize $10y_1+6y_2$

subject to $y_1+5y_2\le 7$

$-y_1+2y_2\le 1$

$3y_1-y_2\le 5$

$-y_1,-y_2\le 0$

对偶问题

$7x_1+x_2+5x_3\ge x_1-x_2+3x_3\ge 10$

$7x_1+x_2+5x_3\ge 5x_1+2x_2-x_3\ge 6$

$7x_1+x_2+5x_3\ge y_1(x_1-x_2+3x_3)+y_2(5x_1+2x_2-x_3)\ge 10y_1+6y_2$

$(y_1+5y_2)x_1+(-y_1+2y_2)x_2+(3y_1-y_2)x_3$

对偶问题 26 原问题

对于一般的线性规划问题

$$\text{Min } cx$$

$$\text{St } \quad Ax \geq b \qquad \text{原问题}$$

$$x \geq 0$$

其对偶问题为

$$\text{Max } b^T y$$

$$\text{St } \quad A^T y \leq c^T \qquad \text{对偶问题}$$

$$y \geq 0$$

HIT CS&E

对偶定理

定理1. 在线性规划问题中，原问题的最优值有限当且仅当对偶问题的最优值有限。并且，如果 $x^*=(x_1^*,\ldots,x_n^*)$ 和 $y^*=(y_1^*,\ldots,y_m^*)$ 分别是原问题和对偶问题的最优解，则 $cx^*=b^T y^*$。

定理2. 在线性规划问题中，如果 $x=(x_1,\ldots,x_n)$ 和 $y=(y_1,\ldots,y_m)$ 分别是原问题和对偶问题的可行解，则 $cx \geq by$。

证明：

HIT CS&E

由于y是对偶问题的可行解，且x_j非负

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{j=1}^{n} \Big(\sum_{i=1}^{m} a_{ij} y_i\Big) x_j$$

由于x是对偶问题的可行解，且y_i非负

$$\sum_{i=1}^{m} \Big(\sum_{j=1}^{n} a_{ij} x_j\Big) y_i \geq \sum_{i=1}^{m} b_i y_i$$

注意到

$$\sum_{j=1}^{n} \Big(\sum_{i=1}^{m} a_{ij} y_i\Big) x_j = \sum_{i=1}^{m} \Big(\sum_{j=1}^{n} a_{ij} x_j\Big) y_i$$

证毕

HIT CS&E

定理3. 如果 $x=(x_1,\ldots,x_n)$ 和 $y=(y_1,\ldots,y_m)$ 分别是原问题和对偶问题的可行解，则x和y分别是原问题和对偶问题的最优解当且仅当下面的条件同时成立：

原问题的补松弛条件：

$$\text{对于} 1 \leq j \leq n\text{：} \; x_j = 0 \text{或者} \sum_{i=1}^{m} a_{ij} y_i = c_j$$

对偶问题的补松弛条件：

$$\text{对于} 1 \leq i \leq m\text{：} \; y_i = 0 \text{或者} \sum_{j=1}^{n} a_{ij} x_j = b_i$$

定理4. 如果 $x=(x_1,\ldots,x_n)$ 和 $y=(y_1,\ldots,y_m)$ 分别是原问题和对偶问题的可行解，且满足

原问题的补松弛条件：$\alpha \geq 1$

$$\text{对于 } 1 \leqslant j \leqslant n\text{：} \; x_j = 0 \text{或者 } \frac{c_j}{\alpha} \leqslant \sum_{i=1}^{m} a_{ij} y_i \leqslant c_j$$

对偶问题的补松弛条件：$\beta \geq 1$

$$\text{对于 } 1 \leqslant i \leqslant m\text{：} \; y_i = 0 \text{或者 } b_i \leqslant \sum_{j=1}^{n} a_{ij} x_j \leqslant \beta \bullet b_i$$

则

$$\sum_{j=1}^{n} c_j x_j \leqslant \alpha \bullet \beta \bullet \sum_{i=1}^{m} b_i y_i$$

证明：$\displaystyle\sum_{j=1}^{n} c_j x_j \leqslant \alpha \sum_{j=1}^{n} \Big(\sum_{i=1}^{m} a_{ij} y_i\Big) x_j = \alpha \sum_{i=1}^{m} \Big(\sum_{j=1}^{n} a_{ij} x_j\Big) y_i \leqslant \alpha \bullet \beta \bullet \sum_{i=1}^{m} b_i y_i$

许多(基于Primal-dual schema设计的)近似算法以定理4为理论基础

HIT CS&E

Min-max关系

最大流问题

输入：有向图$G=(V,E)$，源顶点$s \in V$，接收顶点$t \in V$，每条边e的流量限制$c(e)>0$.

输出：从s到t的最大流。

即，对每条边e赋值$f(e)$使得$\sum_{ut \in E} f(ut)$最大且满足

流量约束：$f(e)<c(e)$

守恒约束：$\sum_{uv \in E} f(uv) = \sum_{uv' \in E} f(uv') \qquad u \in V \;\; u \neq s, u \neq t$

HIT CS&E

最小割问题

输入：有向图$G=(V,E)$，源顶点$s\in V$, 接收顶点$t\in V$,每条边e的流量限制$c(e)>0$.

输出：s和t之间的最小割。

即，$s\in X\subseteq V$，$t\in V-X$使得$\sum_{u\in X, v\in V-X} c(uv)$最小

最小割问题与最大流问题间的min-max关系

任意s,t-割的容量给出了任意s,t-可行流的流量的上界

因此：如果一个s,t-割的容量等于一个s,t-可行流的流量，则该割是一个最小割且该流是一个最大流

HIT CS&E

最大流问题的线性规划表示

$$\begin{aligned} \max\quad & f_{ts} & \\ \text{s.t.}\quad & f_{ij} \le c_{ij} & ij\in\mathcal{E} \\ & \sum_{j:ji\in E} f_{ji} - \sum_{j:ij\in E} f_{ij} \le 0 & i\in\mathcal{V} \\ & f_{ij}\ge 0 & ij\in\mathcal{E} \end{aligned}$$

$$\begin{aligned} \min\quad & \sum_{ij\in\mathcal{E}} c_{ij} d_{ij} & \\ \text{s.t.}\quad & d_{ij} - p_i + p_j \ge 0 & ij\in\mathcal{E} \\ & p_s - p_t \ge 1 & \\ & d_{ij} \in \{0,1\} & ij\in\mathcal{E} \\ & p_i \in \{0,1\} & i\in\mathcal{V} \end{aligned}$$

$$\begin{aligned} \min\quad & \sum_{ij\in\mathcal{E}} c_{ij} d_{ij} & \\ \text{s.t.}\quad & d_{ij} - p_i + p_j \ge 0 & ij\in\mathcal{E} \\ & p_s - p_t \ge 1 & \\ & d_{ij} \ge 0 & ij\in\mathcal{E} \\ & p_i \ge 0 & i\in\mathcal{V} \end{aligned}$$

考虑整数规划的解

$p_s^*=1,\ p_t^*=0$

$X=\{i \mid p_i^*=1\}$ $V-X=\{i \mid p_i^*=0\}$

X,V-X是一个最小s,t-割

HIT CS&E

两类基于LP方法的近似算法

很多组合优化问题可以表达成整数线性规划问题

将整数约束条件放宽，即得到一个线性规划问题　　LP-松弛问题

线性规划问题可以用Karmarkar算法在多项式时间内求解

如何将线性规划问题的解，变成整数得到原问题的一个近似解?

方法1：舍入法　　保证舍入得到的近似解代价不会大幅度增加

方法2：primal-dual schema

构造LP-松弛问题的一个整数可行解x作为输出

构造LP-松弛问题的对偶问题的可行解z

比较上述两个解的代价可以得到近似比的界限

两种方法的主要区别在于运行时间，第一种方法需要精确求解线性规划，而第二种不需要。此外，由第二种方法得到的算法可能能够转换成组合优化算法。

HIT CS&E

LP方法的应用实例

- 集合覆盖问题
- 集合覆盖问题的线性规划表示
- 对偶过滤方法
- 舍入法
- 随机舍入方法
- Primal-dual schema

HIT CS&E

问题的定义

- 输入:

  有限集U, U的一个子集族$\mathcal{S}$, $X=\cup_{S\in\mathcal{S}} S$，每个集合$S$的代价$c(S)$

- 输出:

  $C\subseteq\mathcal{S}$，满足

  (1). $U=\cup_{S\in C} S$,

  (2). C是满足条件(1)的代价最小的集族, 即$\sum_{S\in C} c(S)$最小.

HIT CS&E

集合覆盖问题的线性规划表示

对F中的每个集合S，引入一个变量x_S

$x_S=0$ 表示$S\notin C$　　　　$x_S=1$ 表示$S\in C$

集合覆盖问题的线性规划表示

$$\begin{aligned} \text{minimize}\quad & \sum_{S\in\mathcal{S}} c(S) x_S & \\ \text{subject to}\quad & \sum_{S: e\in S} x_S \ge 1, & e\in U \\ & x_S\in\{0,1\}, & S\in\mathcal{S} \end{aligned}$$

HIT CS&E

LP-松弛问题—原问题

$$\text{minimize} \quad \sum_{S \in \mathcal{S}} c(S) x_S$$
$$\text{subject to} \quad \sum_{S: e \in S} x_S \geq 1, \quad e \in U$$
$$x_S \geq 0, \quad S \in \mathcal{S}$$

对偶问题

$$\text{maximize} \quad \sum_{e \in U} y_e$$
$$\text{subject to} \quad \sum_{e \in S} y_e \leq c(S), \quad S \in \mathcal{S}$$
$$y_e \geq 0, \quad e \in U$$

HIT CS&E

贪心算法（对偶过滤方法）

贪心算法：

1. $F=\phi$; $C=\phi$;
2. while $F \neq U$ do
   从$\mathcal{S}$中挑选一个集合S使得$c(S)/|S-F|$最小;
   $\alpha = c(S)/|S-F|$;
   对任意$e \in S-F$，令$price(e) = \alpha$;
   $C=C \cup \{S\}$，$F=F \cup S$, $\mathcal{S}=\mathcal{S}-\{S\}$;
3. 输出C

C的代价为$\sum_{e \in U} price(e)$

引理1. 对任意$e \in U$令$y_e = price(e)/H_n$, 则由所有y_e构成的向量y是对偶问题的一个可行解。

证明：显然$y_e \geq 0$对任意$e \in U$成立，只需对任意$S \in \mathcal{S}$验证$\sum_{e \in S} y_e \leq c(S)$成立.

假设将S中所有元素依它们在算法中被覆盖的先后次序列出为$e_1, \ldots, e_k$

考察e_i第一次被覆盖的时刻，由于S中此时还有$k-i+1$个元素未被覆盖，将$c(S)$平摊到这些元素上，每个元素分得的代价为$c(S)/(k-i+1)$。

由于在覆盖e_i时，S本身也是优选集合，且选择集合S'时要求$c(S')/|S'-F|$达到最小，故$price(e_i) \leq c(S)/(k-i+1)$。进而

$$y_{e_i} \leqslant \frac{1}{H_n} \frac{c(S)}{k-i+1}$$

故

$$\sum_{e \in S} y_e = \sum_{i=1}^{k} y_{e_i} = \frac{c(S)}{H_n}\left(\frac{1}{k} + \frac{1}{k-1} + \ldots + \frac{1}{1}\right) = \frac{H_k}{H_n} c(S) \leqslant c(S)$$

证毕

定理5. 贪心算法的近似比为H_n。

证明：

OPT_f OPT

对偶问题 | LP-松弛问题(原问题)

对偶问题的可行解的值不超过集合覆盖问题最优解的值OPT

故

$$\sum_{e \in U} y_e = \sum_{e \in U} \frac{price(e)}{H_n} \leqslant OPT$$

即 $\sum_{e \in U} price(e) \leq H_n OPT$

注意：不等式的左端恰好是近似解的代价。

证毕

HIT CS&E

舍入法

频率：对于$e \in U$，e的频率指的是$\mathcal{S}$中包含e的集合的个数

f: U中元素的最大频率

集合覆盖问题的LP-舍入算法

1. 用单纯形法求解LP-松弛问题
2. For $S \in \mathcal{S}$ Do
   IF $x_S \geq 1/f$ THEN $C = C \cup \{S\}$
3. 输出C

定理6. 对于集合覆盖问题，LP-舍入算法的近似比为f

证明：

对于任意$e \in U$, 由于e至多属于f个集合中，为了确保

$$\sum_{\substack{S \in \mathcal{S} \\ e \in S}} x_S \geqslant 1$$

必有某个x_S使得$x_S \geq 1/f$。因此，算法输出的集族中必有一个集合包含了e；进而，算法的输出覆盖了U。

在舍入过程中，对任意$S \in C$，x_S被舍入为1，至多被放大f倍。因此

$$OPT_f = \sum_{S \in \mathcal{S}} c(S) x_S = \sum_{S \in C} c(S) x_S + \sum_{\text{其他}S} c(S) x_S \geqslant \sum_{S \in C} c(S) \frac{1}{f} + \sum_{\text{其他}S} c(S) x_S \geqslant \frac{1}{f} \sum_{S \in C} c(S)$$

从而，$\sum_{S \in C} c(S) \leq f\, OPT_f \leq f\, OPT$.

证毕

HIT CS&E

集合覆盖问题的LP-随机舍入算法

LP-随机舍入算法

1. 用单纯形方法求解LP-松弛问题得到最优解$x=<x_S:S\in\mathcal{S}>$
2. $C=\phi$
3. For $\forall S\in\mathcal{S}$ Do

   独立地产生一个随机数 *rand*

   IF *rand* > $1-x_S$ THEN $C=C\cup\{S\}$;

   /* S被选入C的概率为x_S*/
4. 输出C

定理7. 对于集合覆盖问题的LP-随机舍入算法，C的代价的数学期望为OPT_f，其中OPT_f是LP-松弛问题的最优解的值。

证明：$E(cost(C))=\sum_{S\in\mathcal{S}} p_r[S\text{被选入}C].c(S)$

$= \sum_{S\in\mathcal{S}} x_S \cdot c(S)$

$=OPT_f$ 证毕

定理8. 对于集合覆盖问题的LP-随机舍入算法，$\forall a\in U$被C覆盖的概率大于$1-1/e$。

证明：

设a属于$\mathcal{S}$的k个集合中，将LP-松弛问题中这些集合对应的变量记为$x_1,\ldots,x_k$.

在LP-松弛问题的优化解中，$x_1+\ldots+x_k\geq 1$。

$P_r[a\text{未被}C\text{覆盖}] = (1-x_1)(1-x_2)\ldots(1-x_k)$

$\leq (1-(x_1+\ldots+x_k)/k)^k$

$\leq (1-1/k)^k$

$Pr[a\text{被}C\text{覆盖}] = 1- Pr[a\text{未被}C\text{覆盖}]$

$\geq 1-(1-1/k)^k$

$\geq 1-1/e$ 证毕

改进策略：为了得到完整的集合覆盖，独立运行LP-随机舍入算法$c\log n$次，其中c满足$(\frac{1}{e})^{c\log n}\leqslant\frac{1}{4n}$，将所有输出集合求并得到C'，然后输出C'。

$P_r[C'\text{未覆盖}U] \leq \sum_{a\in U} P_r[C'\text{未覆盖}a] \leq n\cdot[(1/e)^{c\log n}] = 1/4$

$E(cost(C'))=OPT_f\cdot c\cdot\log n$

$P_r[cost(C') \geq OPT_f\cdot 4c\log n] \leq 1/4$　　Markov 不等式：$P_r(X>t)\leqslant\frac{E(X)}{t}$

$P_r[C'\text{覆盖}U \text{ 且 } cost(C')\leq OPT_f.4c\log n] = 1- P_r[C'\text{未覆盖}U \text{ 或 } cost(C')\geq OPT_f.4c\log n]$

$\leq 1-(1/4+1/4)=1/2$

HIT CS&E

Primal-dual schema

基于Primal-dual Schema的集合覆盖近似算法

1. $x\leftarrow 0$;　/*向量，$\mathcal{S}$中的每个集合S对应一个分量x_s*/
2. $y\leftarrow 0$;　/*向量，U中的每个元素e对应一个分量y_e*/
3. $F\leftarrow\phi$;　/*记录被覆盖的子集*/
4. while $F\neq U$ Do
5. 取$e_0\in U-F$;
6. 增大y_{e_0}直到 $\sum_{e:\,e\in S} y_e= c(S)$ 对某个$S\in\mathcal{S}$成立;
7. 对第6步中满足$\sum_{e:\,e\in S} y_e= c(S)$ 的任意$S\in\mathcal{S}$，令$x_s=1$, $F=F\cup S$;
8. 输出x中$x_s=1$的所有集合构成的子集族;

引理2. 在上述算法中，while循环结束后，x和y分别是原问题和对偶问题的可行解。

证明：

1. While循环结束后，U中的所有元素均被覆盖。
2. 算法初始时，$0 = \sum_{e:\,e\in S} y_e\leq c(S)$对任意$S\in\mathcal{S}$成立。

   算法运行过程中，当$\sum_{e:\,e\in S} y_e= c(S)$对某个$S\in\mathcal{S}$成立后，S中的所有元素均被加入到F中，因此在算法以后运行的各个阶段内第5步不会再选中S中的任何元素，即$\sum_{e:\,e\in S} y_e$不会再增加。

   基于以上两条原因，算法结束后，$\sum_{e:\,e\in S} y_e\leq c(S)$对任意$S\in\mathcal{S}$成立。

引理3. 在上述算法中，while循环结束时x和y满足以下两个性质：

(1) 对于$\forall S\in\mathcal{S}$，$x_s\neq 0 \Rightarrow \sum_{e:\,e\in S} y_e = c(S)$;

(2) 对于$\forall e\in U$，$y_e\neq 0 \Rightarrow \sum_{S:\,e\in S} x_s \leq f$（U中元素的最大频率）

证明：

1. 根据算法的第7步即可证得1。
2. 根据f的定义，对于$\forall e\in U$, e至多属于f个集合；且，对于$\forall S\in\mathcal{S}$，$x_s=1$或0；从而结论(2)成立.

定理9. 基于primal-dual schema的集合覆盖近似算法的近似比为f

证明：由引理2和引理3，我们知道，算法结束时x和y分别是原问题和对偶问题的可行解，且

(1) 对于$\forall S\in\mathcal{S}$， $x_s=0$ 或 $c(S)/1 \le \sum_{e:\ e\in S} y_e \le c(S)$；

(2) 对于$\forall e\in U$， $y_e=0$ 或 $1 \le \sum_{S:\ e\in S} x_s \le f\cdot 1$;

这恰好是定理4中的条件（$\alpha=1$，$\beta=f$）。

由定理4，$cost(C) = \sum_{S:\ S\in C} c(S) = \sum_{S:\ S\in\mathcal{S}} x_s c(S) \le 1\cdot f\cdot \sum_{e\in U} y_e$

由于y是对偶问题的可行解，故$\sum_{e\in U} y_e \le cost(C^*)$.

在算法设计过程中，我们实际上要先寻找恰当的α和β使得定理4中的条件得到满足，然后用算法确保该条件成立。通常，我们往往固定α或β为1，仅让另一个参数变化。算法近似比的好坏，往往取决于所确定的参数的优劣。

1. SONET 电话负载问题

输入：含有 n 个顶点的环（其中所有顶点按顺时针方向列出得到 0, 2, 3,…, n-1）。一个电话呼叫集合 C，其中$(i,\ j)\in C$ 表示节点 i 呼叫节点 j。任意电话的路由既可以按顺时针方向进行也可以按逆时针方向进行。对于 C 中电话的路由策略，边$(i, i+1 \bmod n)$的负载为通过该边的电话个数 L_i。环的总负载为 $\max_{1\le i\le n} L_i$。

输出：C 中电话的一个路由策略，使得环的总负载最小。

问题：设计一个 2-近似算法求解 SONET 电话负载问题。

2. 匹配问题

Consider the maximum weight matching problem in a (non-bipartite) graph $G = (V;E)$. More precisely, given a non-negative weight w_{ij} for every edge $(i,j)\in E$, the problem is to find a matching of maximum total weight.

Consider the following greedy algorithm: start from an empty matching and repeatedly add an edge of maximum weight among all edges which do not meet any of the edges chosen previously. Stop as soon as the matching is maximal (i.e., no other edge can be added). Let M_G denote the greedy matching and Z_G its cost. In this problem you are asked to show that the greedy algorithm is a 2-approximation algorithm.

(a) Show that the following linear program gives an upper bound Z_{LP} on the optimal value Z_{opt} of the maximum weight matching problem.

$$\begin{array}{lll} min & \sum_{x_i\in V} x_i & \\ \text{s.t.} & x_i + x_j \ge w_{ij} & \text{for all } (i;j) \in E \\ & x_i \ge 0 & \text{for all } i \in V \end{array}$$

(b) From the greedy matching M_G, construct a feasible solution x to the above linear program and show that its value is $2Z_G$. Conclude that $2Z_G \ge Z_{opt}$.

3.装箱问题

输入：长度为 C 的箱子（数量不限），长度分别为 $0\le w_1,w_2,\ldots,w_n\le C$ 的 n 个不可分割的物品。

输出：物品的装箱策略，使得所用箱子的个数最少。

First-Fit 算法:

依次考察每个物品。对于当前物品 i，在以前用过的箱子 $B_1,\ldots,B_k$ 中找到下标最小的能容纳物品 i 的箱子，将物品 i 放入找到的箱子中；如果找不到这样的箱子，则新开一个箱子 B_{k+1} 将 i 放入。

问题：证明 first-fit 算法的近似比为 2。