复盘与反思

陈长建

计算机科学系

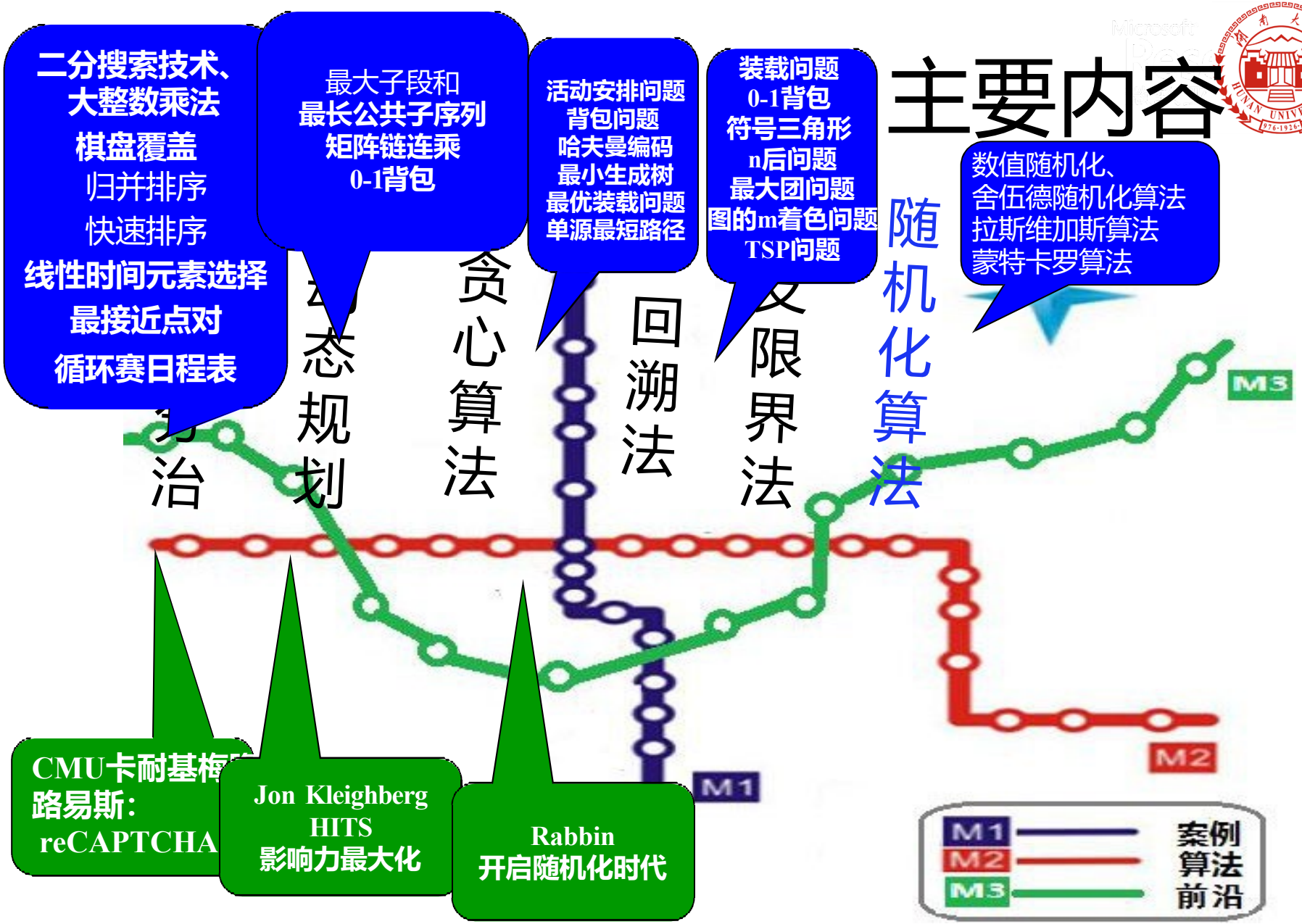
主要内容

- 分治法复习及练习
- 动态规划算法复习及练习
- 贪心算法复习及练习
- 回溯算法复习及练习
- 分支限界法复习及练习
- 随机化算法复习及练习

试题范围

1. 教材
2. 课件
3. 小班讨论主题
4. 实验案例
5. 其他相关拓展

主要内容


二分搜索技术、
大整数乘法
棋盘覆盖
归并排序
快速排序
线性时间元素选择
最接近点对
循环赛日程表
最大子段和
最长公共子序列
矩阵链连乘
0-1背包
活动安排问题
背包问题
哈夫曼编码
最小生成树
最优装载问题
单源最短路径
装载问题
0-1背包
符号三角形
n后问题
最大团问题
图的m着色问题
TSP问题
数值随机化、
舍伍德随机化算法
拉斯维加斯算法
蒙特卡罗算法
分治
动态规划
贪心算法
回溯法
分支限界法
随机化算法
M1
M2
M3
CMU卡耐基梅
路易斯:
reCAPTCHA
Jon Kleighberg
HITS
影响力最大化
Rabbin
开启随机化时代
M1
M2
M3
案例
算法
前沿

分治法复习

分治法的适用条件

-分治法所能解决的问题之特征

- 分而可解
  - 该问题的规模缩小到一定的程度就可以容易地解决
- 分而相同
  - 该问题可以分解为若干个规模较小的相同问题
- 分而可合
  - 利用该问题分解出的子问题的解可以合并为该问题的解
- 分而独立
  - 该问题所分解出的各个子问题是相互独立的。

分治法的基本步骤

```
divide-and-conquer(P)
  {
   if ( | P | <= n0) adhoc(P);    //解决小规模的问题
   divide P into smaller subinstances P1,P2,...,Pk；  //分解问题
   for (i=1,i<=k,i++)
     yi=divide-and-conquer(Pi);  //递归的解各子问题
   return merge(y1,...,yk);  //将各子问题的解合并为原问题的解
  }
```

分、解、合、得

平衡(balancing)子问题的思想，它几乎总是比子问题规模不等的做法要好。

分治法的复杂性分析

- 一个分治法将规模为**n**的问题分成**k**个规模为**n / m**的子问题去解。
- 设分解阀值n0=1，且adhoc解规模为1的问题耗费1个单位时间
- 再设将原问题分解为k个子问题以及用merge将k个子问题的解合并为原问题的解需用f(n)个单位时间
- 用T(n)表示该分治法解规模为|P|=n的问题所需的计算时间，
- 则有：

$$T(n)=\begin{cases} O(1) & n=1 \\ kT(n/m)+f(n) & n>1 \end{cases}$$

分治法-总回顾

分治思想很明了，

分而治之，治而合之。

递归分治8案例，

二分搜索为经典，

大整数分解做乘法；

合并、快速为排序，

棋盘覆盖为一例。

线性时间选择、最接近点对、

循环赛日程表

减少乘法次数

仿快排，
递归处理子数组

利用中位数，
构造平衡子问题

先左后右，对角复制
左上对右下，左下对右上

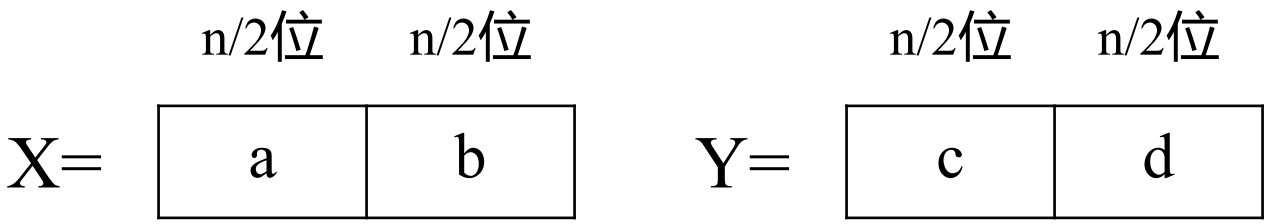
案例分析1——二分搜索技术

- 给定已按升序排好序的n个元素a[0:n-1]，现要在这n个元素中找出一特定元素x。

思考：
三分搜索、四分搜索?

案例分析2——大整数乘法

- 请设计一个有效的算法，可以进行两个n位大整数的乘法运算
  - 小学的方法：$O(n^2)$
  - 分治法：



    - X = ab
    - Y = cd
    - $X = a \cdot 10^{n/2} + b$
    - $Y = c \cdot 10^{n/2} + d$
    - $XY = (a*c) \cdot 10^{n} + (a*d+b*c) \cdot 10^{n/2} + b*d$

时间复杂度分析

- XY =(a*c)•10^n + (a*d+b*c) •$10^{n/2}$ + b*d

$$T(n)=\begin{cases}O(1) & n=1\\ 4T(n/2)+O(n) & n>1\end{cases}$$

- $T(n) = 4T(n/2) + O(n)$
- $= 4[4T(n/2^2)+O(n/2)] + O(n) = 4^2T(n/2^2)+O(n)$
- $=\ldots=O(4^{\log_2 n})+O(n)$
- 对数运算规则 $a^{\log_b c} = c^{\log_b a}$
- $T(n) = O(4^{\log_2 n}) = O(n^{\log_2 4}) = O(n^2)$

×没有改进☹

乘法变换

- $XY = (a*c) \cdot 10^n + (a*d+b*c) \cdot 10^{n/2} + b*d$
- 变换：$a*d+b*c = (a+b)*(c+d) - (a*c) - (b*d)$
- $XY = (a*c) \cdot (10^n - 10^{n/2}) + (a+b)*(c+d) \cdot 10^{n/2} + (b*d) \cdot (1 - 10^{n/2})$
- 4次乘法减为3次乘法

两种变换方法

- $XY = a*c \bullet 10^{n} + ((a-c)*(b-d)+a*c+b*d) \bullet 10^{n/2} + b*d$
- $XY = a*c \bullet 10^{n} + ((a+c)*(b+d)-a*c-b*d) \bullet 10^{n/2} + b*d$

细节问题：两个XY的复杂度都是O(nlog3)，但考虑到a+c,b+d可能得到m+1位的结果，使问题的规模变大，故不选择第2种方案。

时间复杂度分析

$$T(n)=\begin{cases}O(1) & n=1\\ 3T(n/2)+O(n) & n>1\end{cases}$$

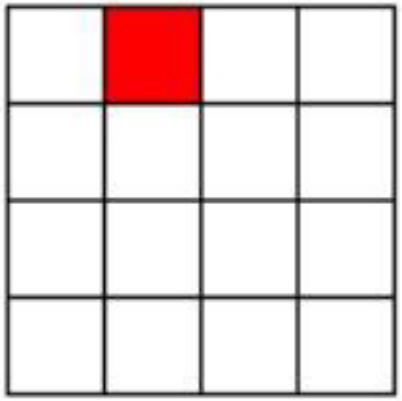
$T(n)=O(n^{\log 3})=O(n^{1.585})$ ✓较大的改进☺

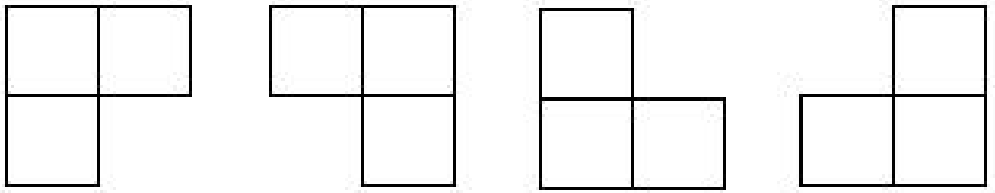
大整数乘法总结

- 小学的方法：$O(n^2)$ ✗效率太低
- 分治法: $O(n^{1.585})$ ✓较大的改进
- 更快的方法：
  - 如果将大整数分成更多段，用更复杂的方式把它们组合起来，将有可能得到更优的算法。
  - 最终，这个思想导致了快速傅利叶变换(Fast Fourier Transform)的产生。该方法也可以看作是一个复杂的分治算法，对于大整数乘法，它能在O(nlogn)时间内解决。
- 是否能找到线性时间算法？目前为止还没有结果。

案例分析3——棋盘覆盖

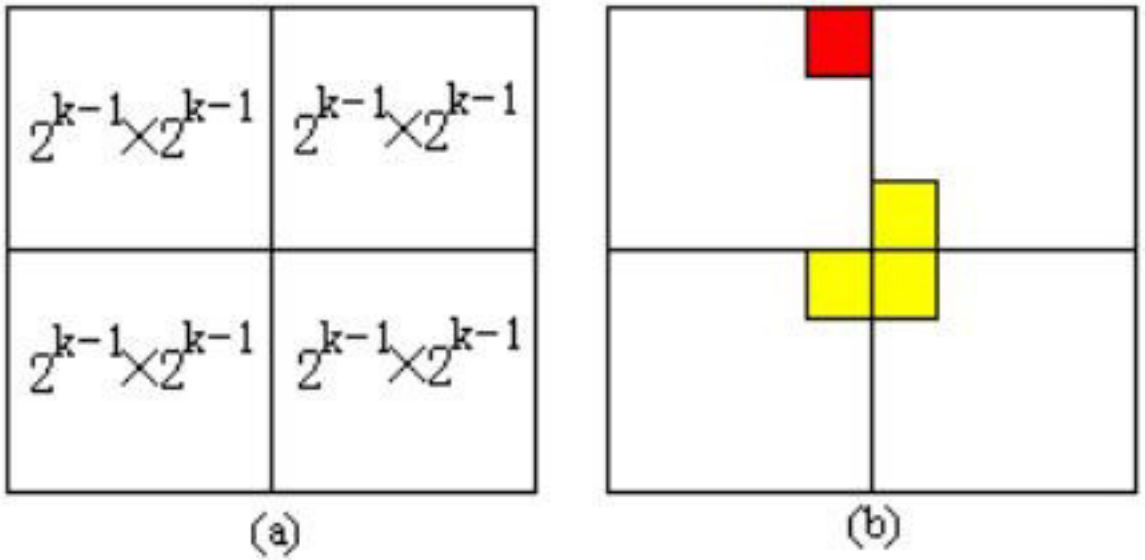
- 在一个$2^k \times 2^k$个方格组成的棋盘中，恰有一个方格与其它方格不同，称该方格为一特殊方格，且称该棋盘为一特殊棋盘。在棋盘覆盖问题中，要用图示的4种不同形态的L型骨牌覆盖给定的特殊棋盘上除特殊方格以外的所有方格，且任何2个L型骨牌不得重叠覆盖。





案例分析3——棋盘覆盖

- 当k>0时，将$2^k \times 2^k$棋盘分割为4个$2^{k-1} \times 2^{k-1}$子棋盘(a)所示。特殊方格必位于4个较小子棋盘之一中，其余3个子棋盘中无特殊方格。为了将这3个无特殊方格的子棋盘转化为特殊棋盘，可以用一个L型骨牌覆盖这3个较小棋盘的会合处，如 (b)所示，从而将原问题转化为4个较小规模的棋盘覆盖问题。递归地使用这种分割，直至棋盘简化为棋盘1×1。



案例分析4——合并排序

- 基本思想：将待排序元素分成大小大致相同的2个子集合，分别对[illegible]并成为所要

复杂度分析：

$$T(n)=\begin{cases} O(1) & n\le 1 \\ 2T(n/2)+O(n) & n>1 \end{cases}$$

T(n)=O(nlogn)渐进意义下的最优算法

```
public stat
  {
  if (left<ri
    {//至少
     int i=(l
     mergeSort(a, left, i); //在第一个子集合上分类(递归)
     mergeSort(a, i+1, right); //在第二个子集合上分类(递归)
     merge(a, b, left, i, right);  //合并到数组b
     copy(a, b, left, right);    //复制回数组a
     }
  }
```

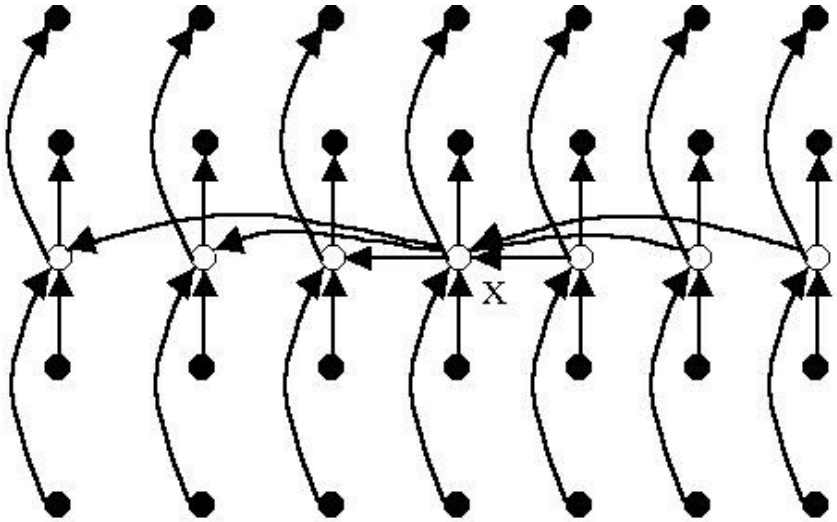
案例分析5——快速排序

- 在快速排序中，记录的比较和交换是从两端向中间进行的，关键字较大的记录一次就能交换到后面单元，关键字较小的记录一次就能交换到前面单元，记录每次移动的距离较大，因而总的比较和移动次数较少。

```
template<class Type>
void QuickSort    (Type a[], int p, int r)
{
        if (p<r) {
            int q=Partition(a,p,r);
            QuickSort (a,p,q-1); //对左半段排序
            QuickSort (a,q+1,r); //对右半段排序
            }
}
```

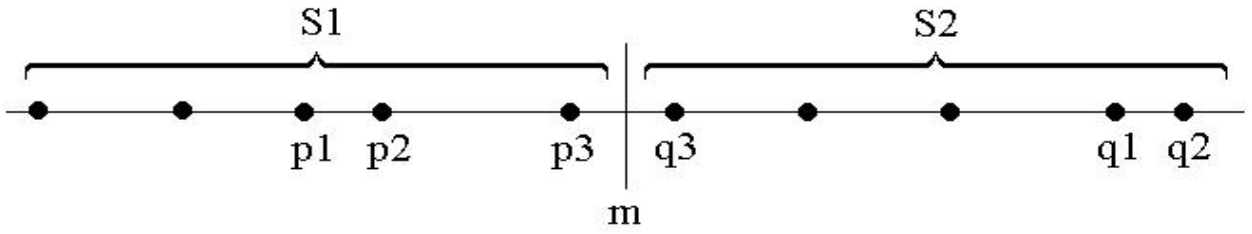
案例分析6——线性时间元素选择

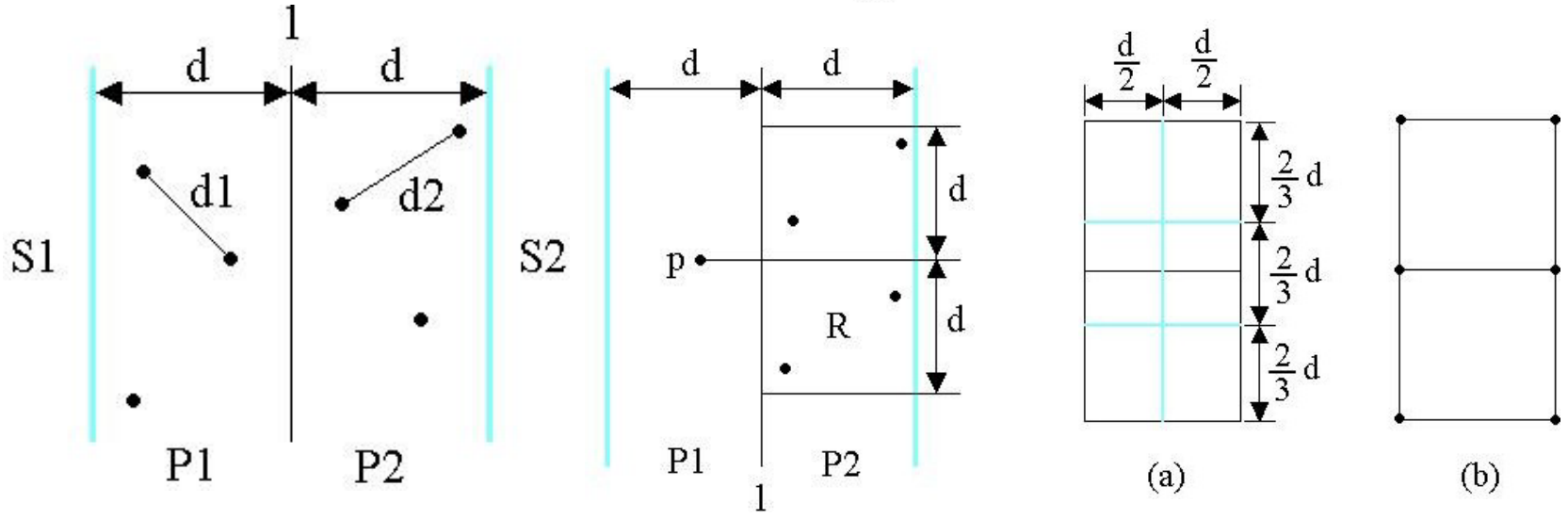
- 给定线性序集中n个元素和一个整数k，$1 \le k \le n$，要求找出这n个元素中第k小的元素
- **基本思路**
  - 分解问题、缩小规模
  - 利用快速排序思想、每次递归处理部分子数组
  - 利用中位数，构造线性时间复杂度的算法



案例分析7——最接近点对问题

- 给定平面上n个点的集合S，找其中的一对点，使得在n个点组成的所有点对中，该点对间的距离最小。





案例分析8——循环赛日程表

- 设计一个满足以下要求的比赛日程表：
- (1)每个选手必须与其他n-1个选手各赛一次；
- (2)每个选手一天只能赛一次；
- (3)循环赛一共进行n-1天

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

动态规划复习及练习

动态规划——一图一故事


```
A * "1+1+1+1+1+1+1+1 =?" *

A : "上面等式的值是多少"
B : *计算* "8!"

A *在上面等式的左边写上 "1+" *
A : "此时等式的值为多少"
B : *quickly* "9!"
A : "你怎么这么快就知道答案了"
A : "只要在8的基础上加1就行了"
A : "所以你不用重新计算因为你记住了第一个等式的值为8!动态规划算法也可以说是 '记住求过的解来节省时间'"
```

动态规划算法的起源

- 20世纪50年代，“最优化原则”，最优化问题的一种新的算法——动态规划算法。
- 最优化原则：
  - 无论过程的初始状态和初始决策是什么，其余的决策都必须相对于初始决策所产生的状态构成一个最优决策序列。（最优子结构性质）

Richard Bellman
美国数学家，美国工程院、科学院院士
动态规划创始人

动态规划的难点

- 子问题
- 递推变量
- 递推式

动态规划案例1——最大子段和

- 问题描述：给定长度为n的整数序列，a[1...n], 求[1,n]某个子区间[i,j]使得a[i]+…+a[j]和最大。或者求出最大的这个和。
- 注：所有整数均为负整数时，定义其最大字段和为0.
  - 例如:(-2,11,-4,13,-5,2)
  - 最大子段和为20,所求子区间为[2,4].

应用实例：收支记录表，收入最丰的一段记录。

动态规划案例1——最大子段和

- 穷举法：穷举所有的[1,n]之间的区间. $O(n^3)$
- 穷举法改进 $\sum_{k=i}^{j} a_k = a_j + \sum_{k=i}^{j-1} a_k$ $O(n^2)$
- 分治法： $O(nlogn)$
  - 在[1, n/2]区域内
  - 在[n/2+1,n]区域内
  - 起点位于[1,n/2],终点位于[n/2+1,n]内
- 动态规划法： $O(n)$
  - b[j]=max{b[j-1]+a[j],a[j]}

可能的疑点（1）

- 最大子段和
  - 是否会出现跳跃
  - a[j]为负怎么办
- 子段{a[s]，a[s+1]，…，a[j-1]，a[j]}是以a[j]为尾元素的最大子段，定义b[j]=max{a[i]+...+a[j]}
  - 如果b[j-1]>0，则b[j]=b[j-1]+a[j]；
  - 如果b[j-1]<=0，则b[j]=a[j]。
  - b[j]=max{b[j-1]+a[j],a[j]}

案例分析2——最长公共子序列（LCS）

- 问题描述：给定2个序列$X=\{x_1,x_2,\ldots,x_m\}$和$Y=\{y_1,y_2,\ldots,y_n\}$，找出X和Y的最长公共子序列。
- 提炼问题三要素：
  - 输入：(x1,x2,...,xm)，Y = (y1,y2,...yn)
  - 输出：Z = X与Y的最长公共子序列
  - 约束：无

案例分析2——最长公共子序列

- 子问题的递归结构:
- 用c[i][j]记录序列和的最长公共子序列的长度。其中，Xi={x1,x2,…,xi}； Yj={y1,y2,…,yj}。
- 当i=0或j=0时，空序列是Xi和Yj的最长公共子序列。故此时C[i][j]=0。
- 其它情况下，由最优子结构性质可建立递推关系，如下：

$$c[i][j]=\begin{cases} 0 & i=0,j=0 \\ c[i-1][j-1]+1 & i,j>0;x_i=y_j \\ \max\{c[i][j-1],c[i-1][j]\} & i,j>0;x_i\neq y_j \end{cases}$$

案例分析3——矩阵链乘法

- 问题描述
- 给定n个矩阵{$A_1,A_2,\ldots,A_n$}，其中A_i与A_{i+1}是可乘的，i=1，2…，n-1。如何确定计算矩阵连乘积的计算次序，使得依此次序计算矩阵连乘积需要的数乘次数最少。
- 提炼问题三要素
  - 输入：<$A_1, A_2, ..., A_n$>，　　A_i是矩阵
  - 输出：计算$A_1 \times A_2 \times ... \times A_n$的最小代价方法
  - 约束条件：A_i与A_{i+1}是可乘的，i=1，2…，n-1

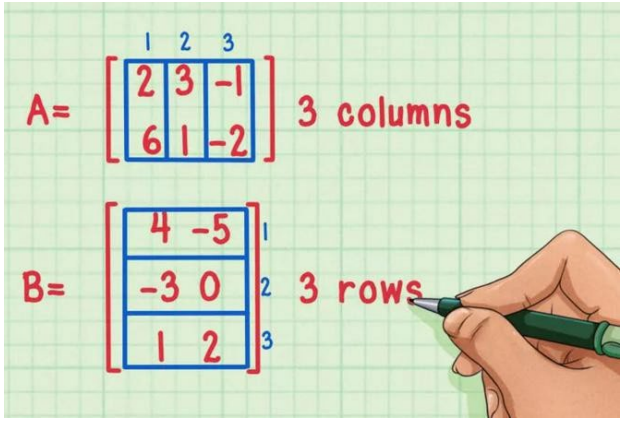
动态规划步骤2：建立递归关系

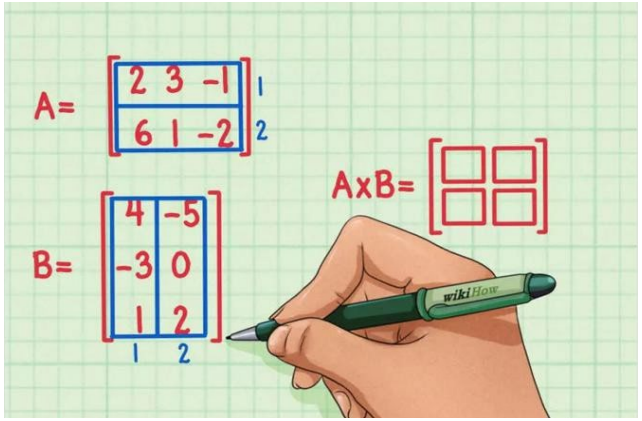
$$m[i,j]=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\} & i<j\end{cases}$$

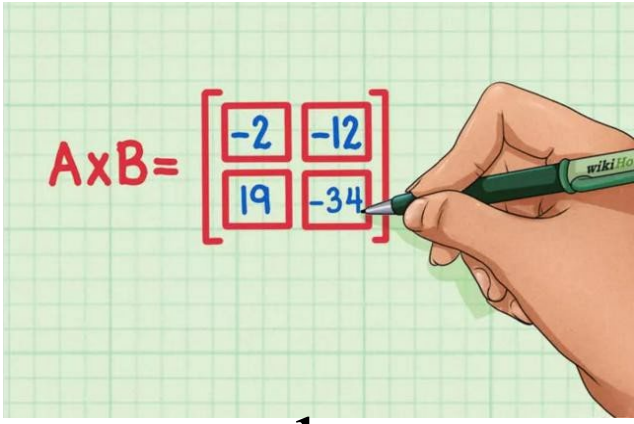
k 的位置只有 $j-i$ 种可能

回顾与反思——矩阵链乘法

- 矩阵乘法的代价/复杂性:乘法的次数
- 若A是p×q矩阵，B是q × r矩阵，则A × B的代价是O(pqr)





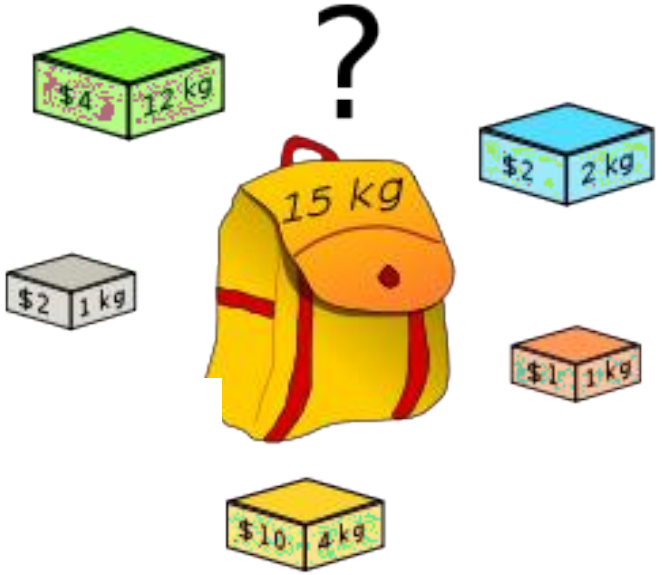


如果A是p*q的矩阵 B是q*r的矩阵　那么乘积C是p*r的矩阵　$c_{ij}=c_{ij}+a_{ik}*b_{kj}$

```
for(i=1;i<=p;i++)
    for(j=1;j<=r;j++)
    {
        c[i][j]=0;
        for(k=1;k<=q;k++)
            c[i][j]+=a[i][k]*b[k][j];
    }
```

案例分析4——0-1背包问题

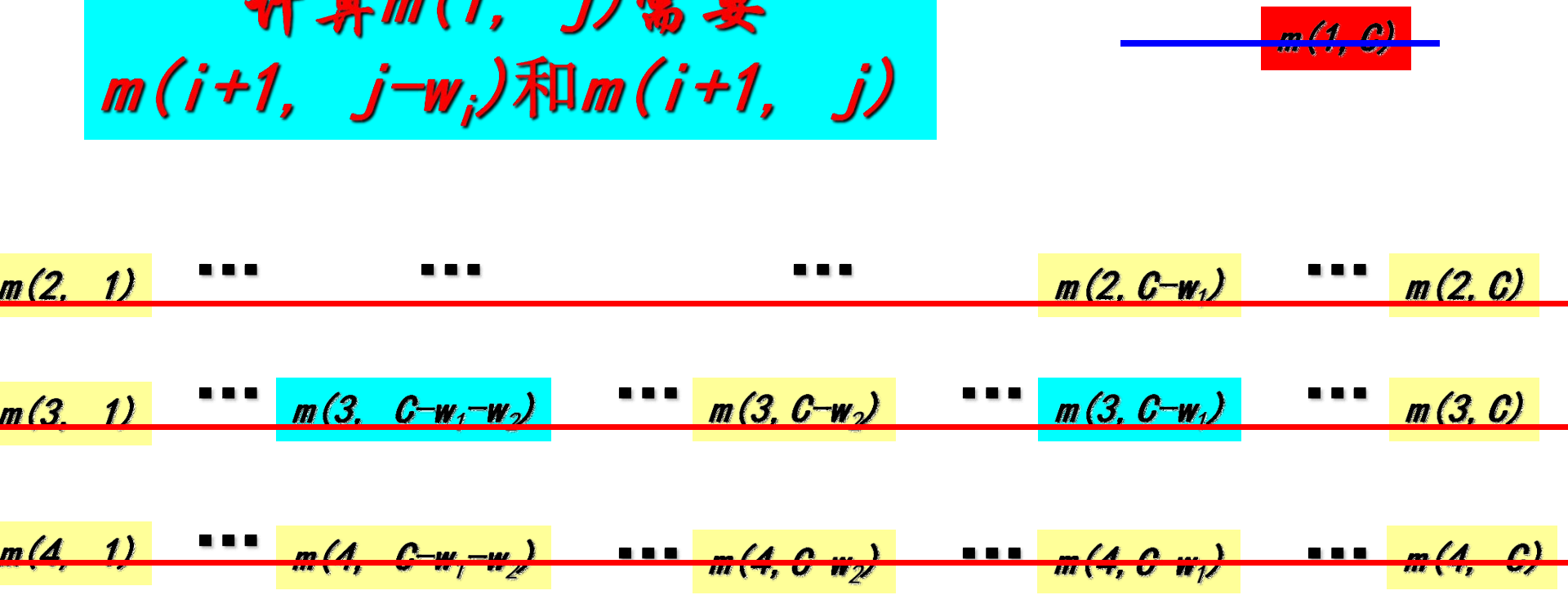
- 问题描述：给定n种物品和一背包。物品i 的重量是wi，其价值为vi，背包的容量为C。问应如何选择装入背包的物品，使得装入背包中物品的总价值最大?



- 提炼问题三要素
  - 输入：$C>0$，$w_i>0$，$v_i>0$，$1\le i\le n$
  - 输出：$(x_1, x_2, \cdots, x_n)$，$x_i\in\{0, 1\}$，
  - 约束：$\Sigma_{1\le i\le n} w_i x_i \le C$，$\Sigma_{1\le i\le n} v_i x_i$ 最大

0-1背包问题-算法实质

计算$m(i, j)$需要
$m(i+1, j-w_i)$和$m(i+1, j)$



回顾与反思

- 最大子段和 b[j]=max{b[j-1]+a[j],a[j]} O(n)
- LCS

$$c[i][j]=\begin{cases}0 & i=0,j=0\\ c[i-1][j-1]+1 & i,j>0;x_i=y_j\\ \max\{c[i][j-1],c[i-1][j]\} & i,j>0;x_i\neq y_j\end{cases}$$

O(mn)

- 矩阵链连乘

$$m[i,j]=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\} & i<j\end{cases}$$

$O(n^3)$

- 0-1背包

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

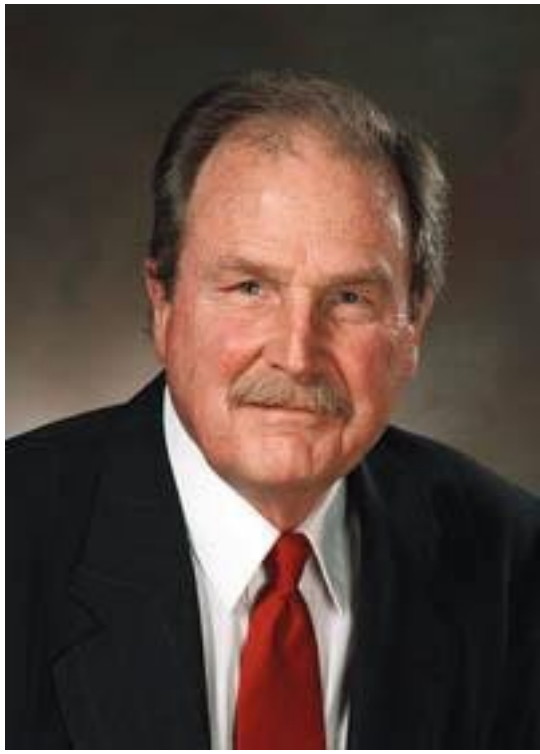
$O(n2^n)$

贪心算法的故事1、2

贪心算法案例——最优编码树问题

- 'Huffman Codes' used in fax machines, modems, other applications involving the compression of data

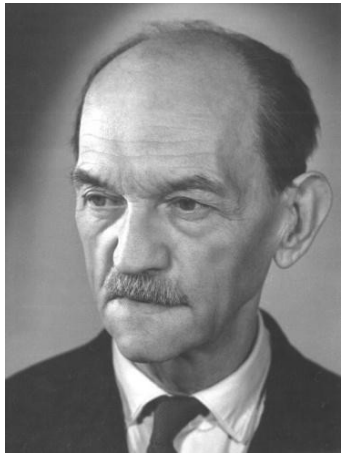
  In 1951, David A. Huffman and his MIT information theory classmates were given the choice of a term paper or a final exam. The professor, Robert M. Fano, assigned a term paper on the problem of finding the most efficient binary code.
  Huffman, unable to prove any codes were the most efficient, was about to give up and start studying for the final when he hit upon the idea of using a frequency-sorted binary tree and quickly proved this method the most efficient.



David Huffman： 1925-1999

贪心算法案例——最小生成树之Prim算法

- Prim算法



Vojtěch Jarník

Robert Clay Prim

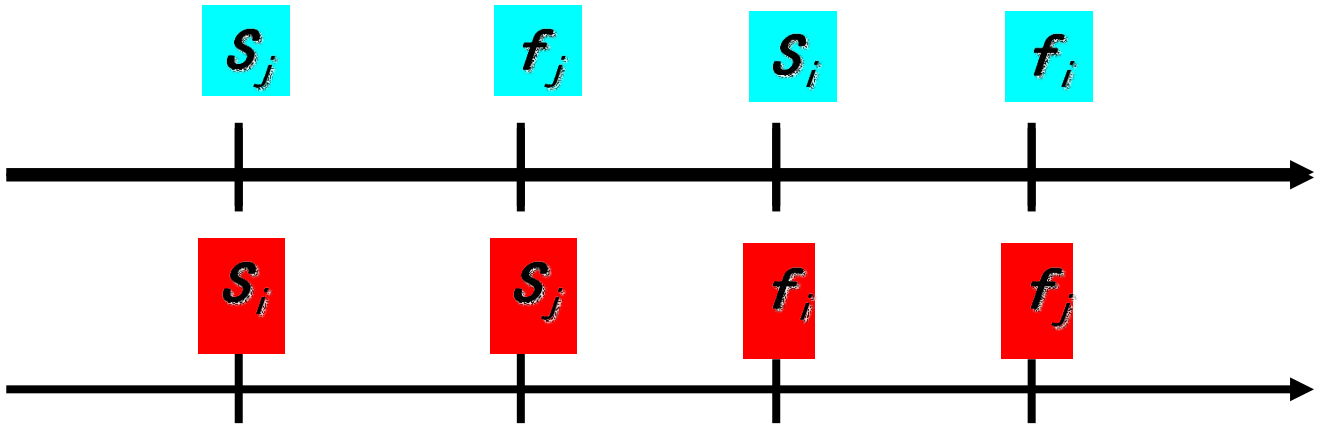
Edsger W.Dijkstra

1. Vojtěch Jarník (1930)
2. Robert Prim (1957): Ph. D. in Princeton University； Bell Laboratories
3. Edsger Dijkstra (1959)

The DJP algorithm – more famously known as the Prim's Algorithm.

案例分析1——活动安排问题

- 活动
  - 设S={1,2,…,n}是n个活动的集合，各个活动使用同一个资源，资源在同一时间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



案例分析1——活动安排问题

- [算法思路]将n个活动按结束时间非减序排列,依次考虑活动i , 若i 与已选择的活动相容,则添加此活动到相容活动子集.

[例] 设待安排的11个活动起止时间按结束时间的非减序排列

i	1	2	3	4	5	6	7	8	9	10	11
s[i]	1	3	0	5	3	5	6	8	8	2	12
f[i]	4	5	6	7	8	9	10	11	12	13	14

最大相容活动子集(**1,4,8,11**),

也可表示为等长n元数组:(1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1)

案例分析3——最优前缀编码树问题

- 提取问题要素（问题形式化定义）
  - 输入: 字母表 C = {c1, c2, ..., cn }，

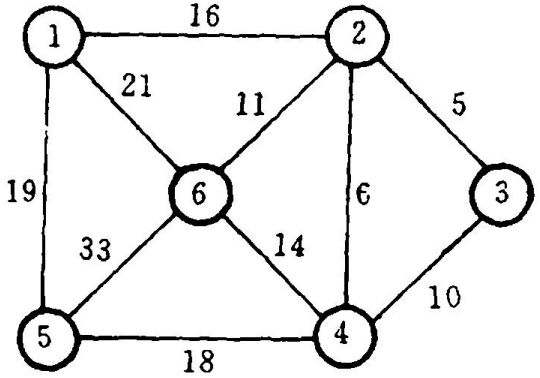
    频率表 F = {f(c1), f(c2), ..., f(cn)}
  - 输出: 具有最小B(T)的C前缀编码树

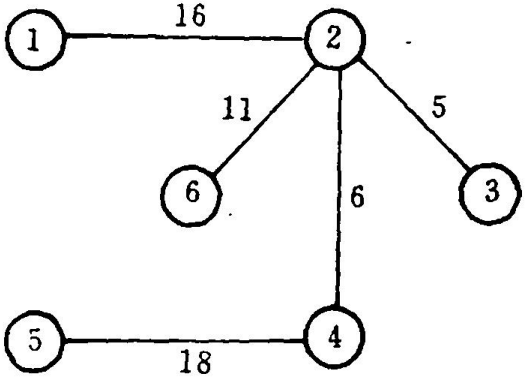
贪心思想：
循环地选择具有最低频率的两个结点，生成一棵子树，直至形成树

案例分析4——最小生成树

- 问题描述:
- 设G(V,E)是一个无向连通带权图。 E中每条边(v, w)的权为c[v][w],若 G的一个子图G’是一棵包含G的所有顶点的树，则称G’为G的生成树
- 生成树各边权的总和称为该生成树的耗费。耗费最小的生成树称为G的最小生成树

- 形式化描述:
- 输入:无向连通带权图可行解:图的生成树
- 优化函数:生成树的各边权值之和最优解:使优化函数达到最小的生成树.



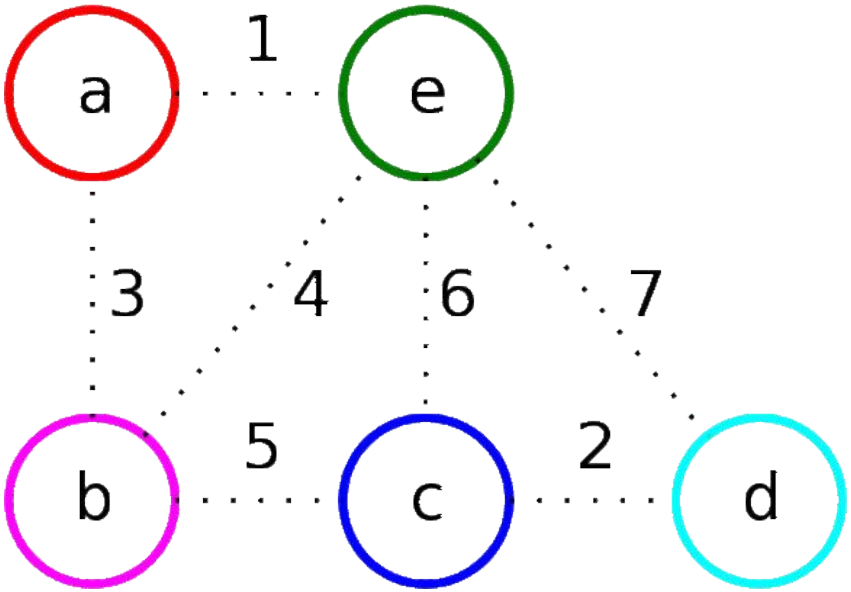


案例分析4——最小生成树之Kruskal算法

- Kruskal算法
- G=(V, E), V=<1, 2, …, n}。

Joseph Kruskal

Edge	ab	ae	bc	be	cd	ed	ec
Weight	3	1	5	4	2	7	6



案例分析5——最小生成树之prim

- Prim算法
- 设G=(V,E)是一个连通带权图, y={1, 2, …, n}。
- 算法思路: 首先置S={1}, T= Ø.若S ⊂ V, 就作如下的贪心选择：选取满足条件i ∈ S, j ∈ V-S,且c[i][j]最小的边(i, j),将顶点j添加到S中边(i, j)添加到T中.这个过程一直进行到S=V时为止. T中的所有边构成G的一棵最小生成树。

算法描述

```
void Prim(int n, Type    * * c)
    { T= Ø;
      S ={1};
       while (S!= V) {
              (i, j) = i ∈ S且j ∈ V- S的最小权边;
              T=TU{(i,j)};
               S= S U{j}; }
            }
```

案例分析6——最优装载

- 问题描述：
- 有一批集装箱要装上一艘载重量为c的轮船。其中集装箱i的重量为Wi。最优装载问题要求确定在装载体积不受限制的情况下，将尽可能多的集装箱装上轮船。

[数学模型] 输入:(x1,x2,...xn),xi=0,货箱i不装船; xi=1,货箱i装船。

可行解: 满足约束条件 $\sum_{i=1}^{n} W_i X_i$ ≤c的输入

优化函数: $\sum_{i=1}^{n} W_i X_i$

最优解:使优化函数达到最大值的一种输入.

回溯算法复习

回溯法-总回顾

回溯方法重搜索，

深度优先约束限界。

递归、迭代、子集树、排列树，

四种算法框架需掌握。

地图着色完全m叉树、

0-1背包子集树，

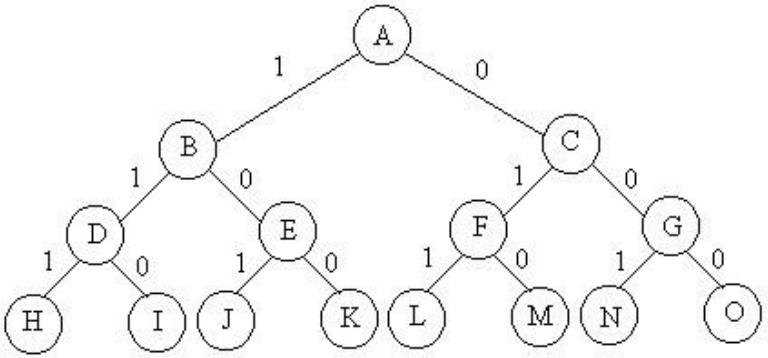
TSP旅行商排列树，

最大团问题子集树，

n后问题和装载问题。

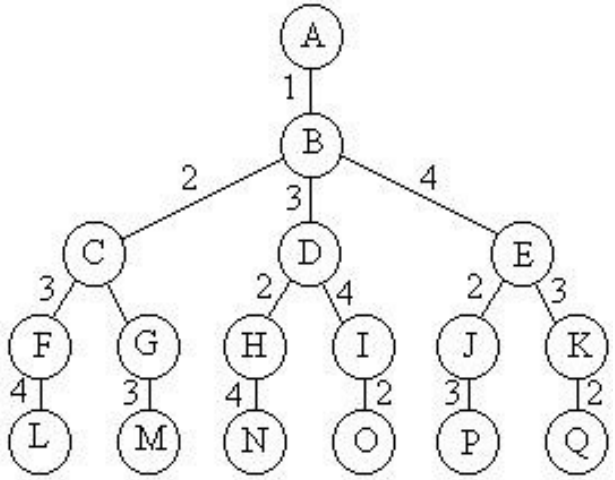
全是NP问题。

回溯法解题的算法框架-子集树与排列树



遍历子集树需$O(2^n)$计算时间

```
void backtrack (int t)
{
  if (t>n) output(x);
    else
      for (int i=0;i<=1;i++) {
        x[t]=i;
        if (legal(t)) backtrack(t+1);
      }
}
```

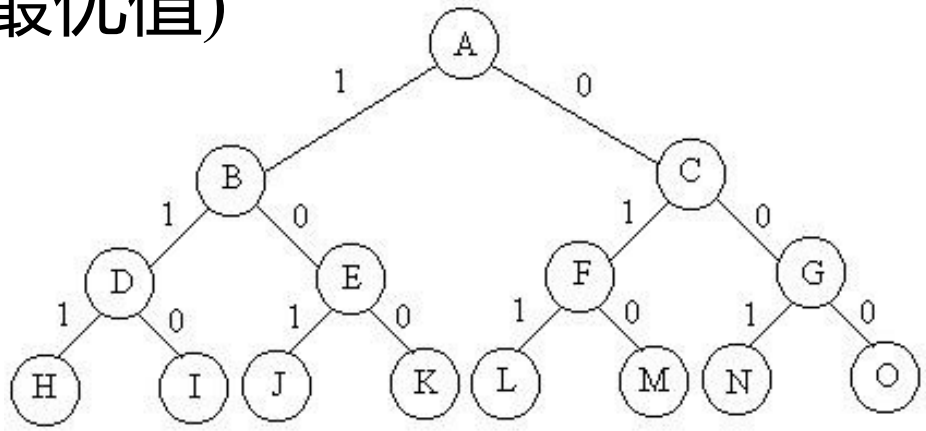


遍历排列树需要O(n!)计算时间

```
void backtrack (int t)
{
  if (t>n) output(x);
    else
      for (int i=t;i<=n;i++) {
        swap(x[t], x[i]);
        if (legal(t)) backtrack(t+1);
        swap(x[t], x[i]);
      }
}
```

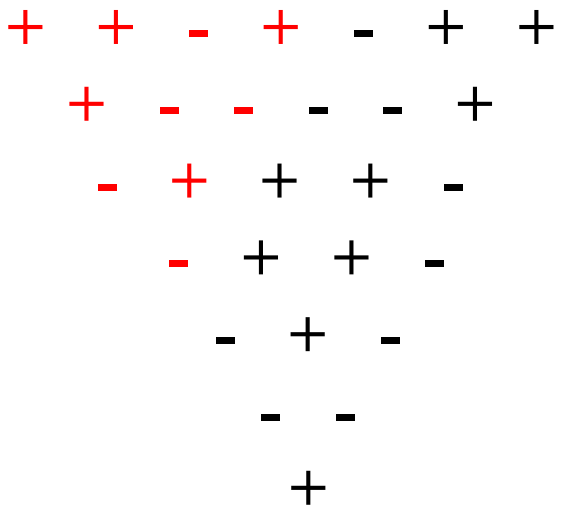
案例分析2——0-1背包问题

- 解空间：子集树
- 可行性约束函数：$\sum_{i=1}^{n} W_i X_i \le c$
- 限界函数（剪去右子树）：cp+r≤bestp (当前价
- 值+右子树价值≤当前最优值)



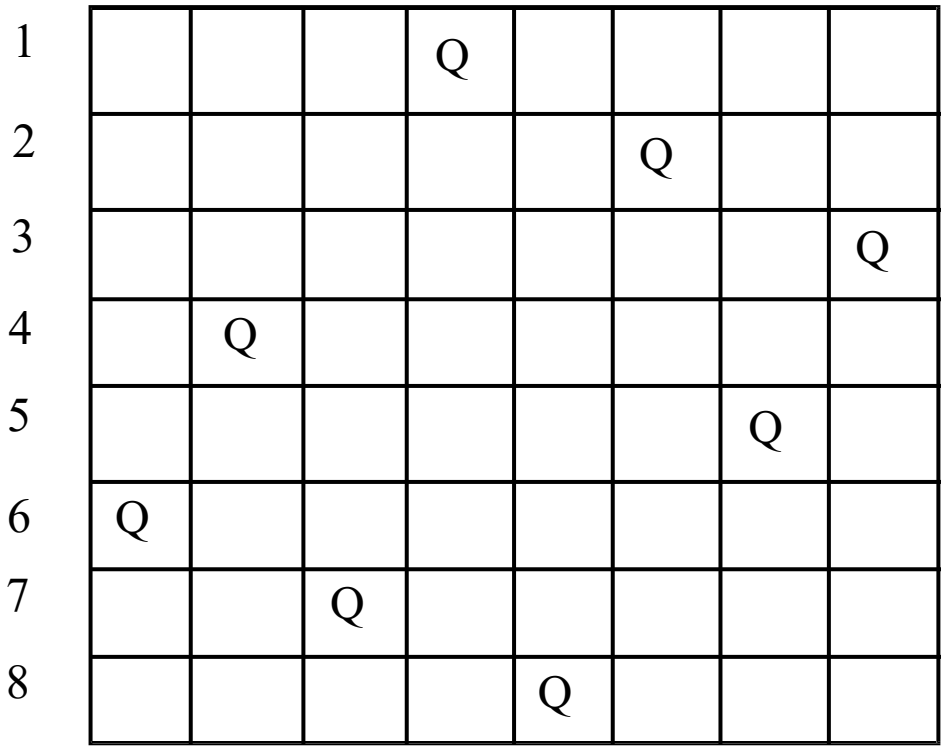
案例分析3——符号三角形问题

- 解向量：用n元组x[1:n]表示符号三角形的第一行。
- 可行性约束函数：当前符号三角形所包含的“+”个数与“-”个数均不超过n*(n+1)/4
- 无解的判断：n*(n+1)/2为奇数



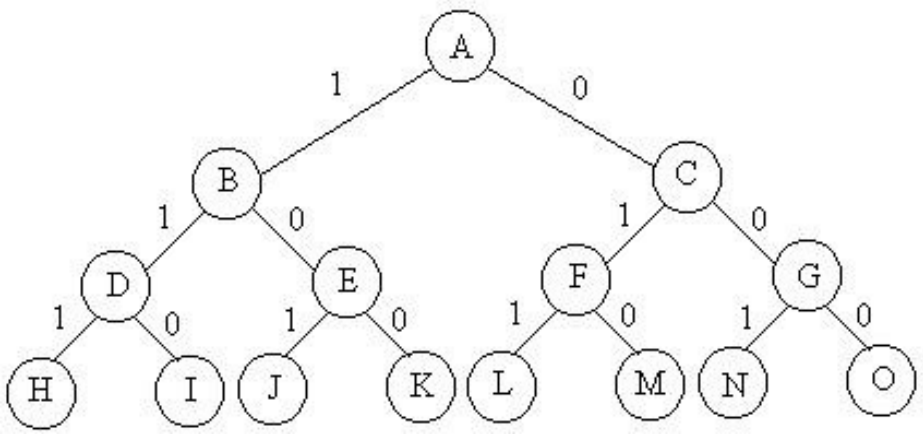
案例分析4- n后问题

- 在n×n格的棋盘上放置彼此不受攻击的n个皇后。n后问题等价于在n×n格的棋盘上放置n个皇后，任何2个皇后不放在同一行或同一列或同一斜线上。



案例分析5——最大团问题

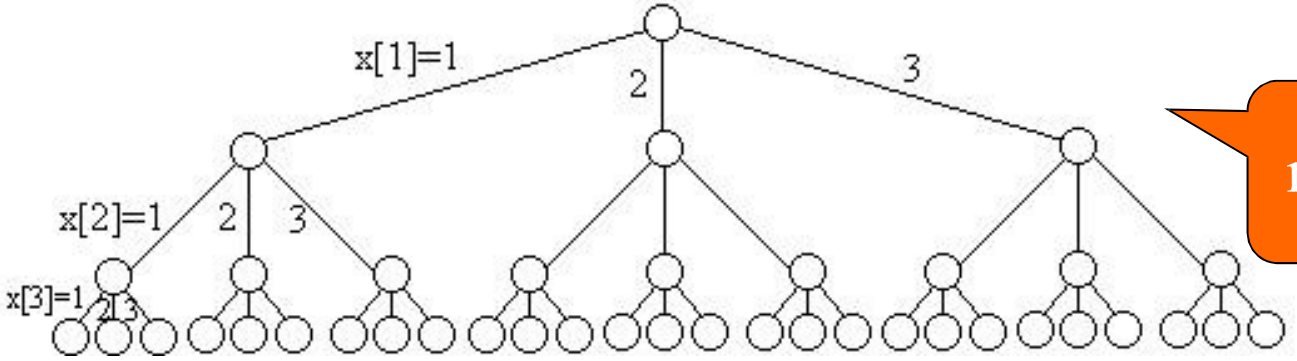
- 解空间：子集树
- 可行性约束函数：顶点i到已选入的顶点集中每一个顶点都有边相连。
- 上界函数：有足够多的可选择顶点使得算法有可能在右子树中找到更大的团。



案例分析6- 图的m着色问题

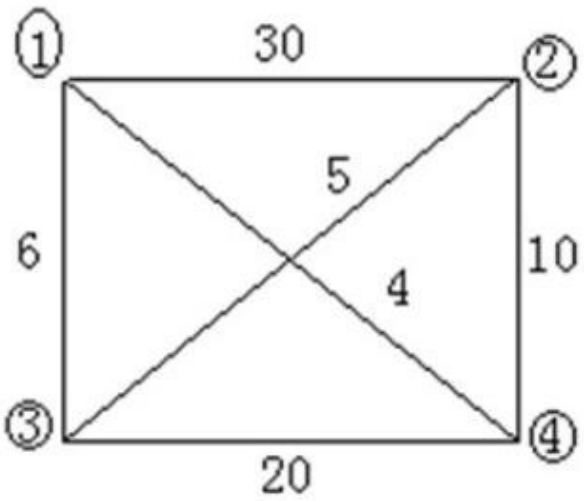
- 解向量：(x1,x2,…,xn)表示顶点i所着颜色x[i]解空间树：n+1高度的完全m叉树
- 可行性约束函数：顶点i与已着色的相邻顶点颜色不重复。

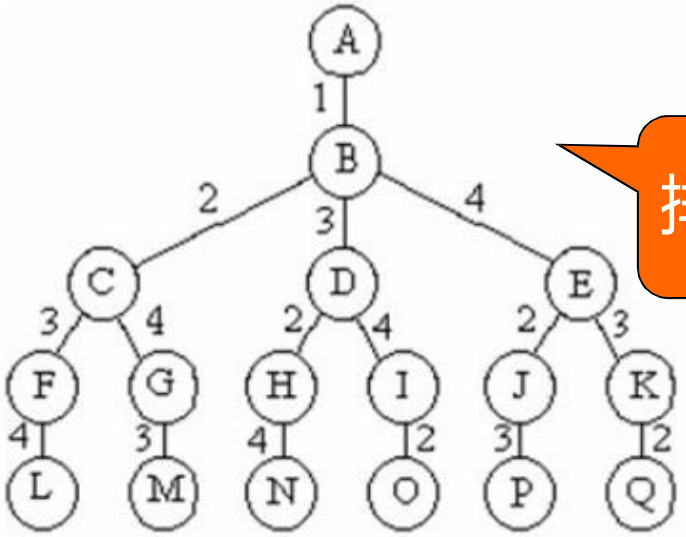
例： 5元组(1， 2， 2， 3， 1)表示对具有5个顶点的无向图的一种着色，顶点A着颜色1，顶点B着颜色2，顶点C着颜色2.....



案例分析7- 旅行售货员问题 (Travelling Salesman Problem:TSP)

- 问题描述
- 某售货员要到若干城市去推销商品，已知各城市之间的路程（旅费），他要选定一条从驻地出发，经过每个城市一遍，最后回到驻地的路线，使总的路程（总旅费）最小。





回溯法效率分析

- 回溯算法的效率在很大程度上依赖于以下因素：
  - (1)产生x[k]的时间；
  - (2)满足显约束的x[k]值的个数；
  - (3)计算约束函数constraint的时间；
  - (4)计算上界函数bound的时间；
  - (5)满足约束函数和上界函数约束的所有x[k]的个数。
- 好的约束函数能显著地减少所生成的结点数。但这样的约束函数往往计算量较大。因此，在选择约束函数时通常存在生成结点数与约束函数计算量之间的折中。

分支限界法复习

分支限界回顾

分支限界广度优先，

搜索解空间为一解。

普通队列or优先级队列。

0-1背包、 TSP旅行商、

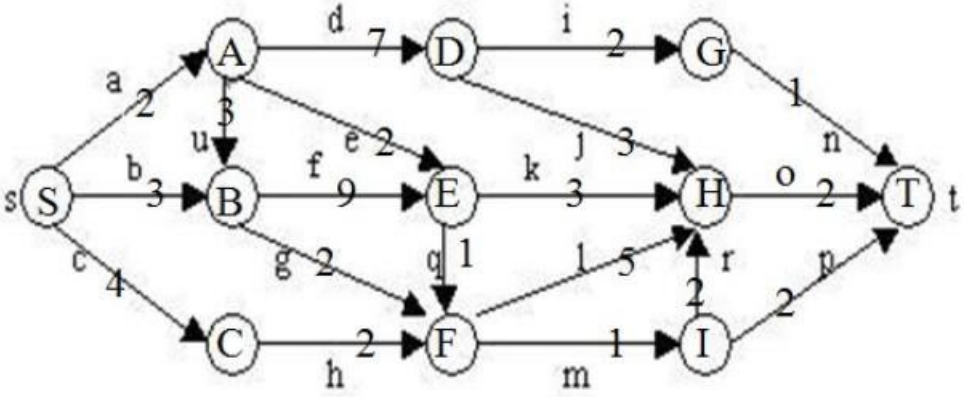
最大团问题。

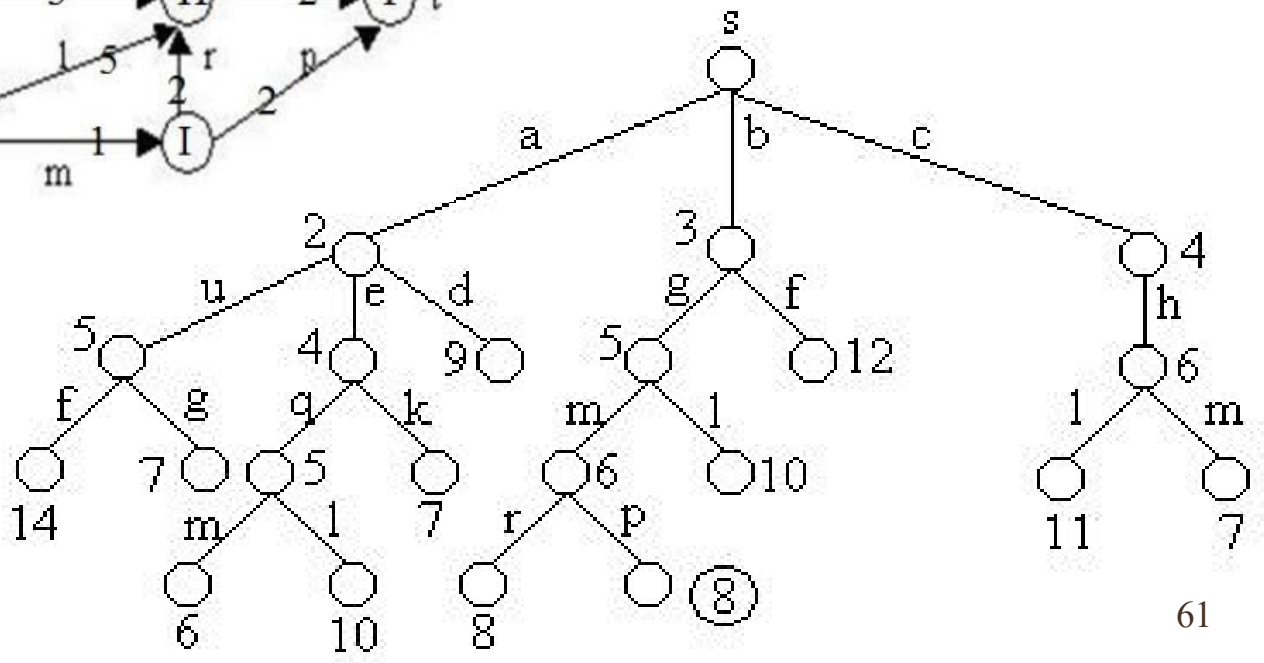
本章案例皆亲切，

旧题新解多总结。

分支限界案例1- 单源最短路径问题

- 解空间树。其中，每一个结点旁边的数字表示该结点所对应的当前路长。





案例分析2- 装载问题

- 算法改进

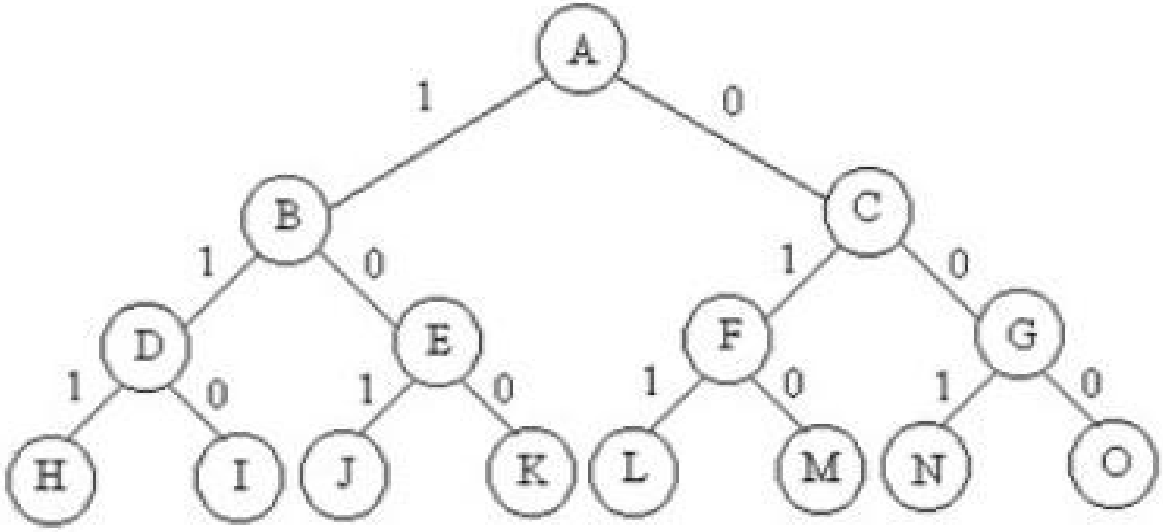
提前更新 bestw

右儿子剪枝

```
// 检查左儿子结点
  Type wt = Ew + w[i];     // 左儿子结点的重量
      if (wt <= c) {       // 可行结点
        if (wt > bestw) bestw = wt;
        // 加入活结点队列
        if (i < n) Q.Add(wt);
}
// 检查右儿子结点
    if (Ew + r > bestw && i < n)
      Q.Add(Ew);         // 可能含最优解
    Q.Delete(Ew);        // 取下一扩展结点
```

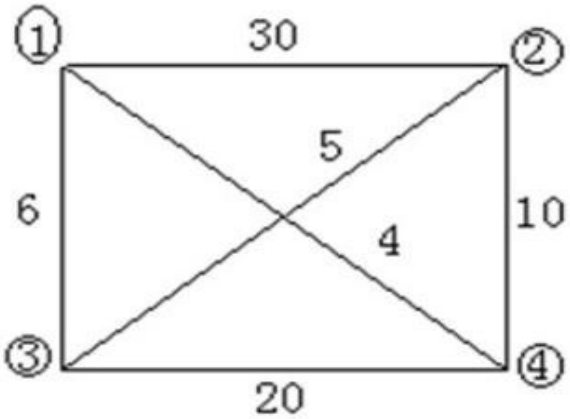
案例分析3-0-1背包问题

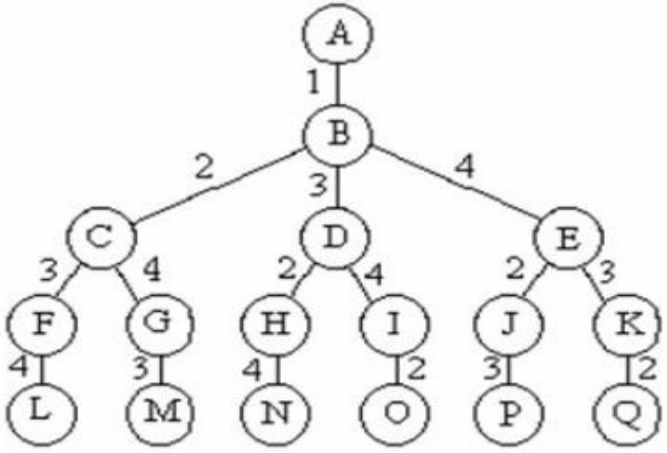
- 例: 给定n=3，w={16,15,15}, p={45,25,25}, c=30。
- 价值大者优先
- {}—>{A}—>{B,C}—>{C，D，E}—>{C，E}—>{C，J，K}—>{C}—>{F，G}—>{G，L，M}—>{G，M}—>{G}—>{N，O}—>{O}—>{}



案例分析4- 旅行售货员问题（TSP）

- 问题描述
- 某售货员要到若干城市去推销商品，已知各城市之间的路程（旅费），他要选定一条从驻地出发，经过每个城市一遍，最后回到驻地的路线，使总的路程（总旅费）最小。

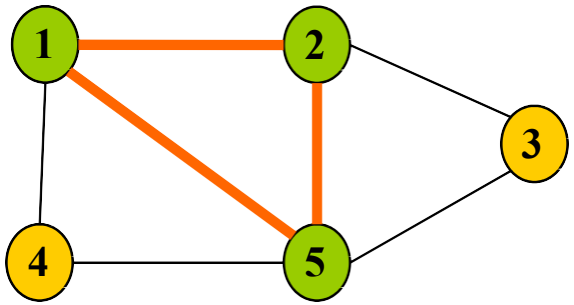




{ }—{B}—{C，D，E}—{C，D，J，K}—{C，J，K，H，I}
—{C，J，K，I，N}—{C，K，I，N，P}—{C，I，N，P，Q}
—{C，N，P，Q，O}—{C，P，Q，O}—{C，Q，O}—{Q，O，F，G}
—{Q，O，G，L}—{Q，O，L，M}—{O，L，M}—{O，M}—{M}—{ }

案例分析5- 最大团问题

- 团（clique）：社交概念，社会团体，团体中的个体互相认识。
- 最大团问题（Maximum Clique Problem,MCP）是图论中一个经典的组合优化问题，也是一类NP完全问题。



例：子集{1，2}是G的大小为2的完全子图。这个完全子图不是团，因为它被G的更大的完全子图{1，2，5}包含。{1，2，5}是G的最大团。{1，4，5}和{2，3，5}也是G的最大团。

随机化算法复习

开启随机化算法的时代-Michael Rabin

- The start of the modern era of random algorithms
- Turing Award winner (with Dana Scott in 1976)因他们的合著论文“有限自动机与其判定性问题”(1959，非确定自动机)

Michael Rabin

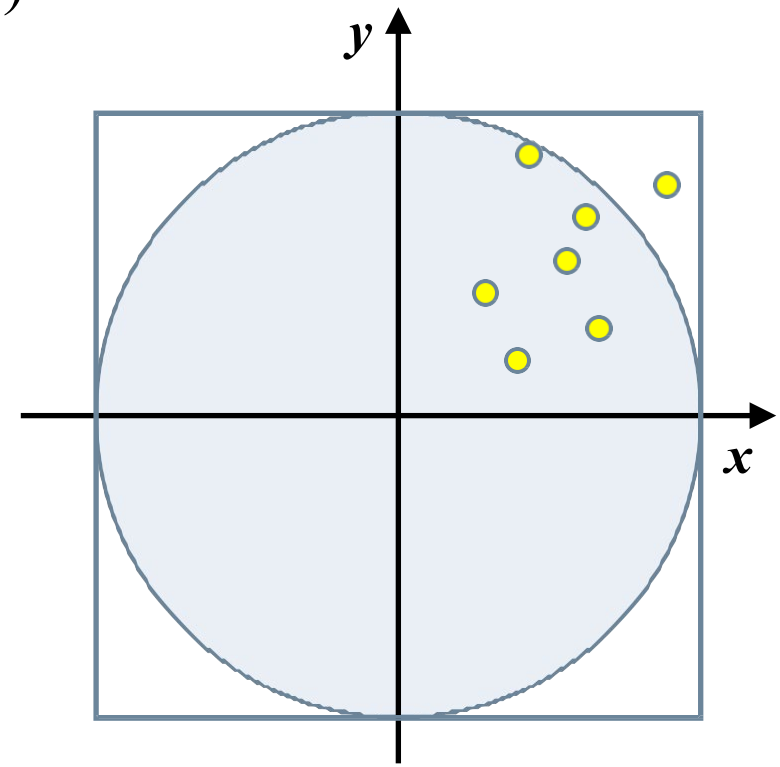
What is so impressive about Rabin’s work is the unique combination of depth and breadth; the unique combination of solving hard open problems and the creation of entire new fields.

阅读资料:
Rabin Flips a Coin | Gödel's Lost Letter and P=NP
https://rjlipton.wordpress.com/2009/03/01/rabin-flips-a-coin/

随机算法的分类(4类-1)

- 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

- 近似计算圆周率
  - 向方框内随机掷点x=r(0,1), y=r(0,1)
  - 落在圆内概率$\pi/4$
  - 近似值≈ 圆内点数c / 总点数n

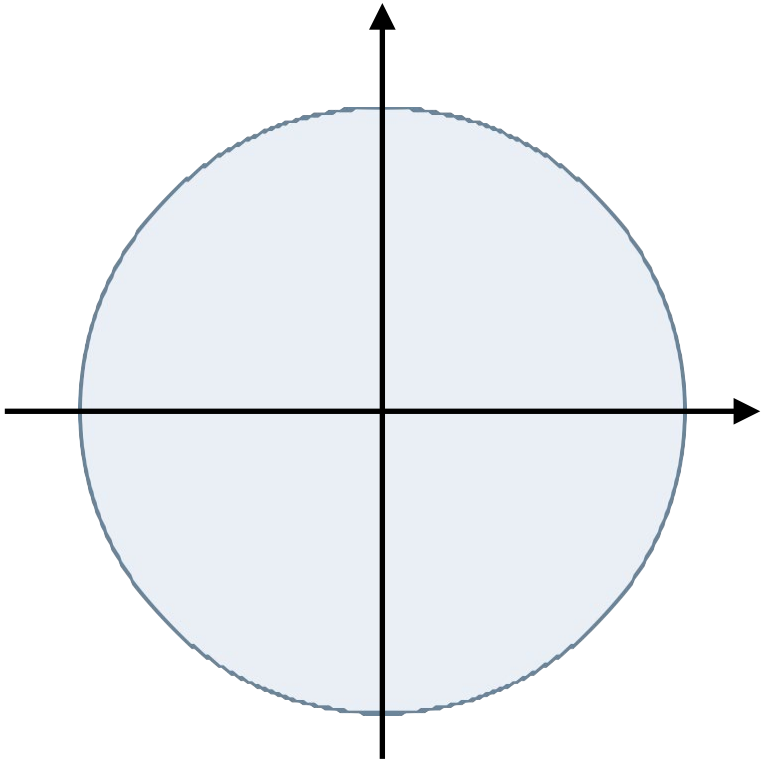


*n*越大，近似度越高

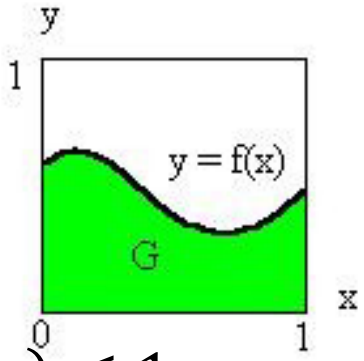
案例分析1-用随机投点法计算π值

- 近似计算圆周率
  - 向方框内随机掷点x=r(0,1), y=r(0,1)
  - 落在圆内概率π/4
  - 近似值 ≈ 圆内点数c / 总点数n

*n*越大，近似度越高



案例分析2-计算定积分



- 设$f(x)$是$[0,1]$上的连续函数，且$0 \le f(x) \le 1$
- 需要计算的积分为$I = \int_0^1 f(x)\mathrm{d}x$，积分$I$等于图中的绿色面积$G$
- 在图所示单位正方形内均匀地作投点试验，则随机点落在曲线$y = f(x)$下面的概率为:

$$P_r\{y \le f(x)\} = \int_0^1 \int_0^{f(x)} \mathrm{d}y\mathrm{d}x = \int_0^1 f(x)\mathrm{d}x$$

- 假设向单位正方形内随机地投入 n 个点(x_i, y_i)。如果有 m 个点落入 G 内，则随机点落入 G 内的概率:

$$I \approx \frac{m}{n}$$

随机算法的分类(4类-2)

- 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

采样越多
越接近最优解

例子：筐里有100个苹果，让我每次闭眼拿1个，挑出最大的。于是我随机拿1个，再随机拿1个跟它比，留下大的，再随机拿1个……我每拿一次，留下的苹果都至少不比上次的小。拿的次数越多，挑出的苹果就越大，但除非拿100次，否则无法肯定挑出了最大的。

这个挑苹果的算法，就属于蒙特卡罗算法——尽量找好的，但不保证是最好的。

蒙特卡罗(Monte Carlo)算法

- 对于一个解所给问题的蒙特卡罗算法MC(x)，如果存在问题实例的子集X使得：

1. 当x属于X时，MC(x)返回的解是正确的；
2. 当x属于X时，正确解是y0，但MC(x)返回的解未必是y0。称上述算法MC(x)是偏y0的算法。

重复调用一个一致的、p正确、偏y_0的蒙特卡罗算法k次，可得到一个$(1-(1-p)^k)$正确的蒙特卡罗算法，且所得算法仍是一个一致的偏y_0蒙特卡罗算法。

蒙特卡罗(Monte Carlo)算法-主元素问题

- 设T[1:n]是一个含有n个元素的数组。当 $|\{i|T[i]=x\}|>n/2$ 时，称元素x是数组T的主元素。

```
template<class Type>
bool Majority(Type *T, int n)
{// 判定主元素的蒙特卡罗算法
   int i=rnd.Random(n)+1;
   Type x=T[i];// 随机选择数组元素
   int k=0;
   for (int j=1;j<=n;j++)
   if (T[j]==x) k++;
   return (k>n/2);
  // k>n/2 时T含有主元素
}
```

```
template<class Type>
bool MajorityMC(Type *T, int n, double e)
{// 重复调用k次Majority算法
   int k=ceil(log(1/e)/log(2));
   for (int i=1;i<=k;i++)
     if (Majority(T,n)) return true;
   return false;
}
```

蒙特卡罗(Monte Carlo)算法-主元素问题

```
template<class Type>
bool MajorityMC(Type *T, int n, double e)
{// 重复调用k次Majority算法
  int k=ceil(log(1/e)/log(2));
  for (int i=1;i<=k;i++)
        if (Majority(T,n))
              return true;
        return false;
}
```

对于任何给定的$\varepsilon > 0$，算法MajorityMC重复调用$\lceil \log(1/\varepsilon) \rceil$次算法Majority。它是一个偏真蒙特卡罗算法，且其错误概率小于ε。算法MajorityMC所需的计算时间显然是$O(n\log(1/\varepsilon))$。

随机算法的分类(4类-3)

采样越多
越有机会找到最优解

- 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, 可使求解无效的概率任意小

例子：有一把锁，给我100把钥匙，只有1把是对的。于是我每次随机拿1把钥匙去试，打不开就再换1把。我试的次数越多，打开（最优解）的机会就越大，但在打开之前，那些错的钥匙都是没有用的。
这个试钥匙的算法，就是拉斯维加斯的——尽量找最好的但不保证能找到。

拉斯维加斯(Las Vegas)算法-n后问题

- 改进：将上述随机放置策略与回溯法相结合。
- 可先在棋盘的若干行中随机地放置皇后，然后在后继行中用回溯法继续放置，直至找到一个解或宣告失败。
- 随机放置的皇后越多，后继回溯搜索所需的时间就越少，但失败的概率也就越大。

stopVegas	p	s	e	t
0	1.0000	262.00	--	262.00
5	0.5039	33.88	47.23	80.39
12	0.0465	13.00	10.20	222.11

随机算法的分类(4类-4)

- 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

舍伍德(Sherwood)算法

- 设A是一个确定性算法，当它的输入实例为 x 时所需的计算时间记为 $t_A(x)$。设 X_n 是算法 A 的输入规模为 n 的实例的全体，则当问题的输入规模为 n 时，算法 A 所需的平均时间为:

$$\bar{t}_A(n) = \sum_{x \in X_n} \frac{t_A(x)}{|X_n|}$$

- 有些$x \in X_n$ $t_A(x) \gg \bar{t}_A(n)$
- 希望获得一个随机化算法 B，使得对问题的输入规模为 n的每一个实例均有:

$$t_B(x) = \bar{t}_A(n) + s(n)$$

- 这就是舍伍德算法设计的基本思想。当 $s(n)$ 与 $t_A(n)$ 相比可忽略时，舍伍德算法可获得很好的平均性能。

分治案例-线性时间元素选择

- 问题
  - 给定线性序集中n个元素和一个整数k，$1 \le k \le n$，要求找出这 n个元素中第k小的元素.

- 基本思路：
  - 分解问题、缩小规模
  - 利用快速排序思想、每次递归处理部分子数组
  - 利用中位数，构造线性时间复杂度的算法

舍伍德(Sherwood)算法-随机选基准

static RandomNumber rnd;

Pages 214-215

```
int i = l,
j = l + rnd.Random(r - l + 1);//随机选择划分基准
Swap(a[i], a[j]);
j = r + 1;
Type pivot = a[l];
```

随机化算法-总回顾

随机算法四大类：

数值随机化算法求近似解、

Sherwood算法消除最坏与平均之差异、

Monte Carlo算法高概率求正确解

（无法判断正确）、

Las Vegas算法改进算法性能

（可能找不到解）

What can we do?

1. Big data offer new challenges for academic researchers;
2. Traditional algorithmic techniques may no longer be proper;
3. A new computation model is proposed for formal study of big data algorithms;
4. Many computational problems can be solved based on the new model and new techniques;
5. Many open problems are left for further and deeper research.

这是一个最好的时代，也是一个最坏的时代。

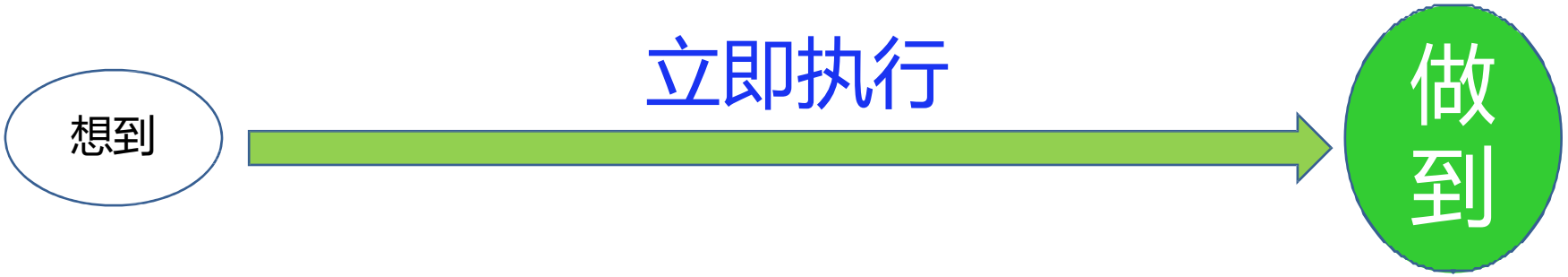
设计和实施第n+1个算法

- 有效复习
  - 明确任务（各科）
  - 明确时间（考试时间、自己的空闲时间片）
  - 合理安排
- 面向未来
  - 明确兴趣目标
  - 明确规划
  - 合理安排

设计一个算法并不难，难的是优化。

Can it be better?

从思想到行动-立即执行



难走的都是上坡路


往者不可谏
来者犹可追
论语

悦纳自己
拥抱世界

学习留痕，稳步前进！

重知识提升

更重方法积累