数据结构与算法设计

教材:

[1][殷] 殷人昆, 数据结构, 清华大学.

[2][王] 王晓东,计算机算法设计与分析,电子工业.

[3][S] 唐常杰等译, Sipser著, 计算理论导引, 机械工业.

参考资料:

[4][严]严蔚敏等,数据结构, 清华大学.

[5][C] 潘金贵等译, Cormen等著, 算法导论, 机械工业.

[6][M] 黄林鹏等译, Manber著, 算法引论-一种创造性方法, 电子.

[7][刘] 刘汝佳等, 算法艺术与信息学竞赛, 清华大学.

数据结构与算法设计习题课-III

算法策略

内容

1. 回溯法
2. 贪心算法
3. 分治策略
4. 动态规划

1. 回溯

运动员最佳配对问题

问题描述: 羽毛球队有男女运动员各n人. 给定2个n×n矩阵P和Q.

P[i][j]是男运动员i与女运动员j配混合双打的男运动员竞赛优势;

Q[i][j]是女运动员i与男运动员j配混合双打的女运动员竞赛优势.

由于技术配合和心理状态等各种因素影响, P[i][j]不一定等于Q[j][i].

男运动员i和女运动员j配对的竞赛优势是P[i][j]*Q[j][i].

设计一个算法, 计算男女运动员最佳配对法, 使得各组男女双方竞赛优势的总和达到最大.

数据输入: input.txt, 第1行有一个正整数n(1≤n≤20),接下来2n行是P和Q

结果输出: 最佳配对的各组男女双方竞赛优势总和

说明:回溯算法问题解答需要剪枝函数和伪代码.

输入:

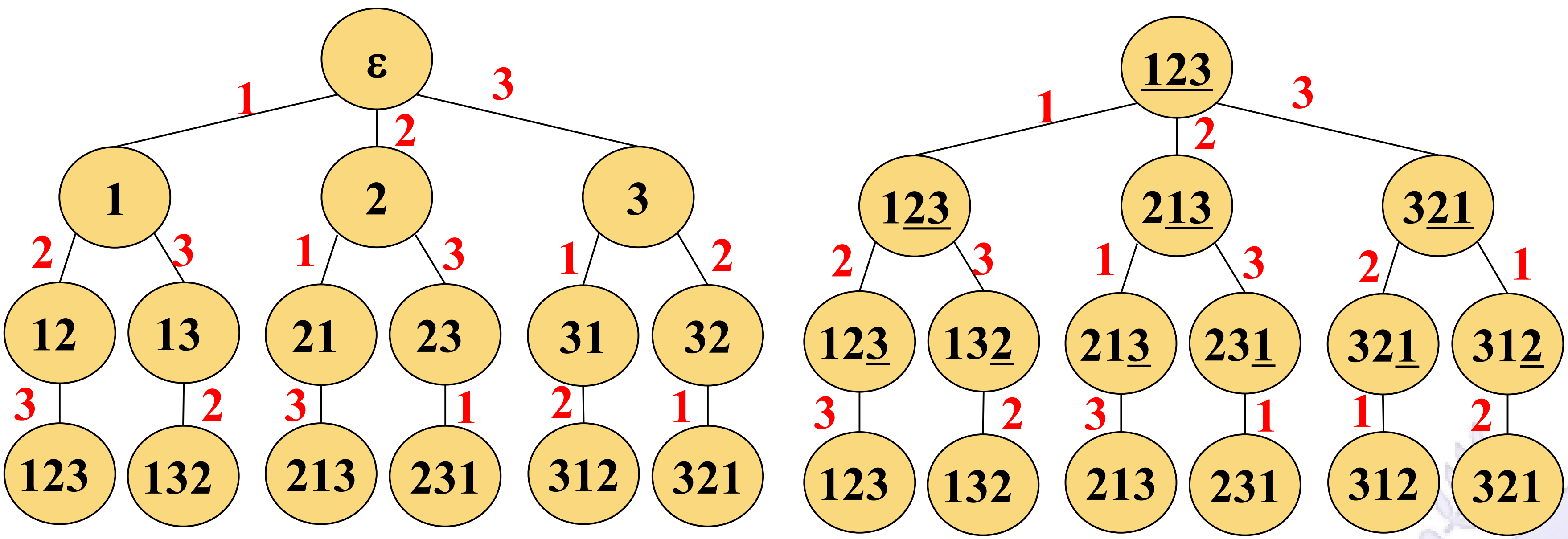
3
10 2 3
2 3 4
3 4 5
2 2 2
3 5 3
4 5 1

输出
52

1. 回溯

解: 男运动员位置不动, 女运动员全排列, 回溯搜索最优值

解空间： 是n的全排列, 所以选择排列树作为解空间结构.



- 变量设计: 当前得分cs, 最佳得分bests, x[1:n]女运动员的排列
- 定义函数 $f(i,m,x) = \max_{j=m+1}^{n} P[i][x[j]]*Q[x[j]][i]$, 其中i>m,
  是在前m位男运动员已配对的情况下, 男运动员i配对其她女运动员的上界
- 定义函数 Upb(m, x) = $f(m+1,m,x)+f(m+2,m,x)+\ldots+f(n,m,x)$.
  当前m位男运动员已配对的情况下, cs+Upb(m,x)是余下情况配对的上界,
- 由此可以设计剪枝(限制函数)条件 cs+Upb(m, x) > bests
- 注意:
  - 注1: 有的同学没有设计剪枝条件, 这不能体现回溯的优势.
  - 注2: 有同学使用 cs < bests作为剪枝条件, 这是错误的.
  - 因为可能当前还有很多没有配对, 当所有配对完成后会有更优值.
  - 注3: 也可以设计其它的剪枝条件.
  - 注4: 函数f, Upb与排列x有关, 在每个节点都要重新计算, 不能统一计算
  - 注5: 有同学先计算矩阵F[i][j]=P[i][j]*Q[j][i], 这是更好的方法.

1. 回溯

初始: 当前得分cs=0, 最佳得分bests=0,

对i=1:n, x[i]=i, 是女运动员的初始排列

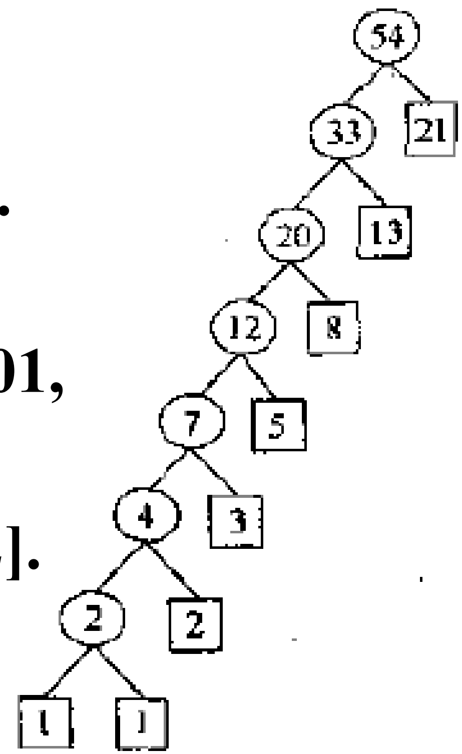
backtrack(t) //t是层号

1. 若 t > n, 返回
2. 对 j = t : n
3. | 交换x[t],x[j],
4. | cs+=P[t][x[t]]*Q[x[t]][t],
5. | 若 cs+Upb(m, x) > bests, //剪枝
6. | | 若cs>bests, 则 bests=cs, //更新最优解
7. | | backtrace(t+1)
8. | cs-=P[t][x[t]]*Q[x[t]][t]
9. | 交换x[t],x[j],

主程序执行backtrack(1)即可

2. 贪心

1. 字符a~h出现的频率恰好是前8个Fibonacci数, 它们的Huffman编码是什么? 将结果推广到 n个字符的频率恰好是前n个Fibonacci数的情形.



解:根据a~h的频率, 画出Huffman编码树如右图

所以各字符编码为: h:1, g:01, f:001, e:0001, d:00001, c:000001, b:0000001, a:0000000,

对n符号情形. 记第i符号为i, 则频率f[i]=f[i-1]+f[i-2].

- 记1:k为前k节点合并, 频率$f[1:k]=\text{sum}_{i=1}^{k} f[i]$.
- 由数学归纳法易证k≥2时$f[k+1]\le f[1:k]<f[k+2]$.
- 所以对所有k≥2, 都有1:k与k+1是兄弟.
- 继续该过程得类似右图的偏二叉树为其Huffman编码树
- 于是对i=2:n, i的编码为$0^{n-i}1$, 1的编码是0^{n-1}.

2. 贪心

2. 若在0-1背包问题中, 各物品依重量递增排列时,
其价值恰好降序排列, 对这个特殊的0-1背包问题, 设计一个有效算法找出最优解, 并说明算法的正确性.

解: 设物品1:n按照重量w[1:n]依次递增, c为容量

(1)贪心选择性质: 未选物品中最轻物品必定在某最优解中

证明: 反证法, 若不包含最轻物品, 则可用最轻物品替换任一物品得到更优解.

(2)最优子结构性质: 选择物品1后的子问题为[2, n]

从最优解中去掉物品1,

它仍是物品2:n和容量c-w[1]的最优解

证明: 反证法, 否则可以替换2:n的选择得到更优解.

算法: 按重量递增排序(O(nlogn)), 依次放入背包, 直到超重(O(n))

2 贪心

- 输入: n物品重W[1:n], 背包容量C
- 输出: 装包使得件数最多.

3. 将上述最优装载问题的贪心算法推广到2艘船的情形，
   贪心算法还能产生最优解吗?

- 解: 直接用贪心算法不行. （说明：答案需要举反例）

  最优装载要求装载件数最多.

  其贪心算法是每次选择最轻的物品.

  设有物品分别重1,2,3,4,5, 船1容量7, 船2容量8.

  若按照最优装载的贪心算法,

  船1装1,2,3, 船2装4, 只能装4件物品.

  最优解是船1装1,2,4, 船2装3,5.

2. 贪心

3. 将最优装载问题的贪心算法推广到2艘船的情形

贪心算法还能产生最优解吗?

参考: 假设两艘船容量分别为c_1,c_2.

1) 先对一艘船容量c_1+c_2做最优装载.

   即优先放最轻, 设选出了物品1到k.

2) 对物品1到k, 第一艘船容量c_1,

   做背包问题, 装包重量最大.

3) 对1到k中剩余的物品, 按最优装载放入第二艘船.

2. 贪心

4. 最优分解问题.

问题描述:设n是一个正整数,将n分解为若干互不相同的自然数之和,且使这些自然数的乘积最大.

算法设计:对于给定的正整数n,计算最优分解方案.

数据输入:由文件input.txt提供输入数据.

文件只有一行,是正整数n.

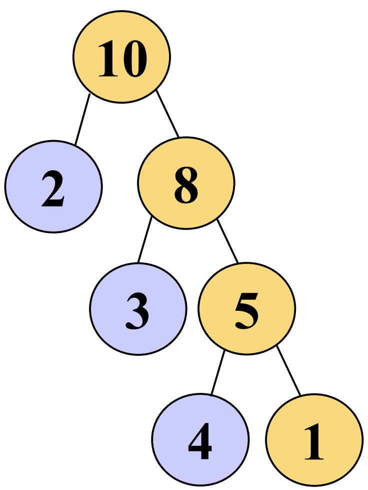
结果输出:将计算的最大乘积输出到文件output.txt

例如若n=10, 则最优分解为2+3+5, 最大乘积为30.

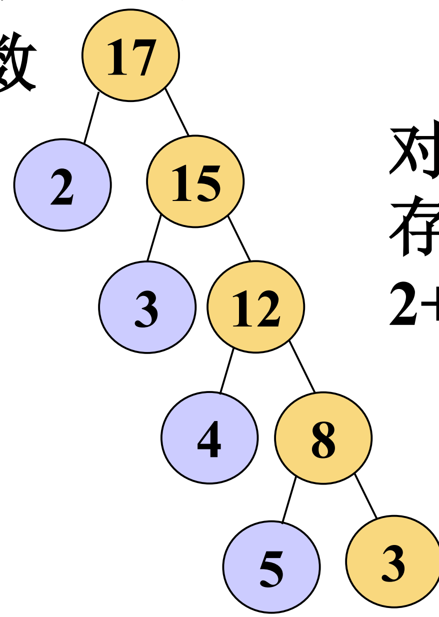
2. 贪心

- n<=4时不分解a*b最大
- n=a+b
- 当|a-b| 最小时，a*b最大
- 题设要求分解的数字互不相同，a和b要越接近越好，所以
- 分解为连续的自然数

对任意自然数n≥2,
存在唯一k使得
$2+3+\ldots+k \le n < 2+3+\ldots+(k+1)$.



n=10 = 2+3+5



n=17 = 2+4+5+6

4. 最优分解问题.

对任意自然数$n\geq 2$, 存在唯一k使得

$$2+3+\ldots+k \leq n < 2+3+\ldots+(k+1).$$

令$m=n-2-3-\ldots-k$, 则$0\leq m \leq k$.

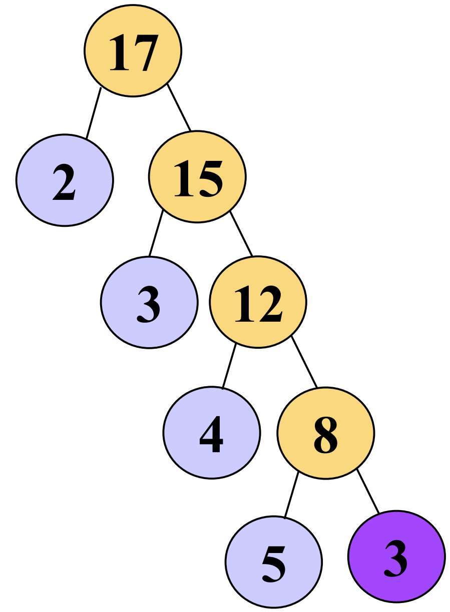
若$m=0$, 则分解为$\{2,3,\ldots,k\}$;

若$1\leq m\leq k-1$, 则分解为$\{2,3,\ldots,k+1\}-\{k-m+1\}$;

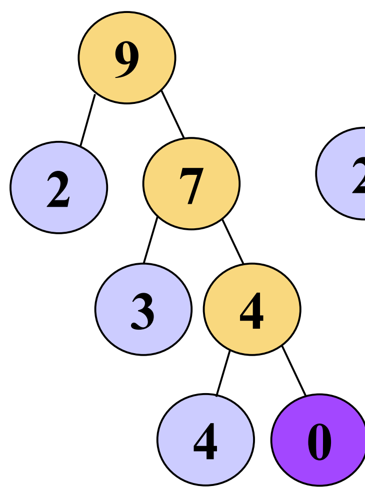
若$m=k$, 则分解为$\{3,4,\ldots,k+2\}-\{k+1\}$.

贪心选择: 取最小不同数的和.

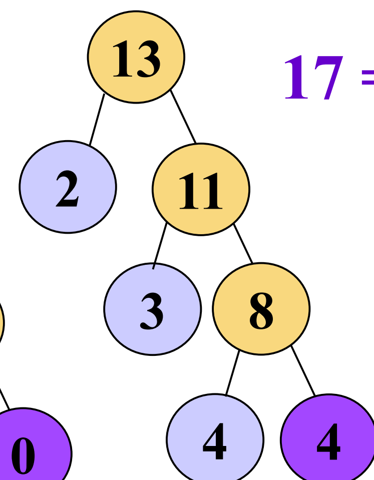
算法正确性证明:



$17 = 2+4+5+6$



$9 = 2+3+4$



$13 = 3+4+6$

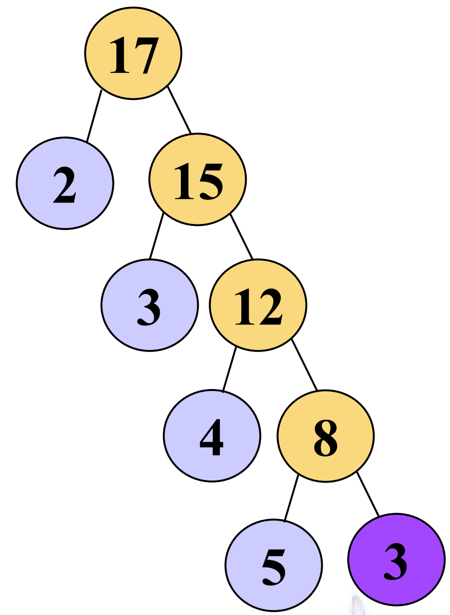
2. 贪心

最优分解算法 //a[1],…,a[k]是n的最优分解

1. k=1; a[1]=2; n-=2;
2. 当 n > a[k],
3. 　　　k++; a[k]=a[k-1]+1; n-=a[k]
4. 若 n == a[k],
5. 　　　a[k]++; n--
6. 对 i = 0:n-1,
7. 　　　a[k-i]++

或者若不要求a[i]递增改为

4. 若 n == a[k], //此时a[k]=k+1
5. 则 a[1] += n
6. 否则 a[k+1-n] +=n



17 = 2+4+5+6

3. 分治

1	2	3	4	5	6	7	8	9	10
-10	-8	-7	0	2	5	7	9	15	20

- 设n个不同的整数按照递增的顺序排好序后存于T[1:n]中.
- 若存在一个下标i，1≤ i ≤n, 使得T[i]=i. 设计一个有效算法找到这个下标. 要求算法在最坏情况下的计算时间O(log n).
- 解: 若T[1:n]严格增, 由解存在知:
  - T[i]>i蕴含∀j>i(T[j]>j),
  - T[i]<i蕴含∀j<i(T[j]<j).
  - 满足二分法条件, 可用二分搜索, 时间O(log n):

```
1. left=1; right=n;
2. while(left<=right)
3. {  mid=(left+right)/2
4.     if(T[mid]=mid) return(mid)
5.     if( T[mid] < mid ) left=mid+1
6.     else right=mid-1
7. }
```

错误:
对递增或递减, 比较中点T[i]和i后左右区间取错了

3. 分治

2.9 设T[0:n-1]是n个元素的数组. 对任一元素x, 设S(x)={ i | T[i]=x}. 当|S(x)|>n/2时, 称x为主元素. 设计一个线性时间算法, 确定T[0:n-1]是否有一个主元素.

算法1: 性质: 若数列有主元素, 则中位数必为主元素.

1. 使用线性时间选择找中位数p, 即第$\lfloor (n+1)/2 \rfloor$大的数,
2. 再计数p出现次数k.
3. 若k>n/2, 则a为主元素; 否则无主元素.

找中位数时间O(n), 计数a出现次数时间O(n).

2.9 设T[0:n-1]是n个元素的数组. 对任一元素x, 设S(x)={ i | T[i]=x}. 当|S(x)|>n/2时, 称x为主元素. 设计一个线性时间算法, 确定T[0:n-1]是否有一个主元素.

算法2:

性质: 若数列有主元素, 则去掉两个不同数, 主元素不变.

	4	5	4	4	4	3	2	4	6	4
当前主元素	4	5	4							
ct	1	1	1	2	3	2	1	2	1	2

	4	5	4	4	4	3	2	2	2	2
当前主元素	4	5	4					2		
ct	1	1	1	2	3	2	1	1	2	3

2.9 设T[0:n-1]是n个元素的数组. 对任一元素x, 设S(x)={ i | T[i]=x}. 当|S(x)|>n/2时, 称x为主元素. 设计一个线性时间算法, 确定T[0:n-1]是否有一个主元素.

算法2: 性质: 若数列有主元素, 则去掉两个不同数, 主元素不变.

1. p=T[0], ct=1, i=1, //p记可能主元素, ct为计数器,

2. 当i<n,

若T[i]==p, 则(ct++,i++);

否则(ct- -,i++;若ct==0, p=T[i], i++, ct++)

3. 计数p出现次数k, 若k>n/2则p是主元素, 否则无主元素.

注1: 有人用计数排序的方法, 当知道数组T的取值范围时是可行的.

注2: 使用分治算法，检测每段主元素是否合并后主元素，时间O(nlogn).

2.25 在线性时间选择算法中, 输入元素被划分为5个一组, 如果将它们划分为7个一组, 该算法仍然是线性时间算法吗? 划分成3个一组又怎样?

解: 以T(n)记输入n个数序列时的算法的最坏时间复杂度.

(1)若划分为7个一组, 则存在常数 n_0, c, d > 0使得

$T(n) \le T(n/7) + T(3n/4) + d\,n$, $n > n_0$; $T(n) \le c$, $n \le n_0$.

以下归纳证明对任意 $n > 0$, $T(n) \le s\,n$, 其中 $s = \max\{c, 28d/3\}$. 这里$3/28=1-(1/7+3/4)$

首先对于 $n \le n_0$, 有 $T(n) \le c \le s\,n$.

其次归纳假设对于 $k > n_0$, 任意 $n < k$ 有 $T(n) \le s\,n$.

于是 $T(k) \le T(k/7) + T(3k/4) + d\,k$, //迭代1次

$\le s\,k / 7 + 3\,s\,k / 4 + d\,k$, //归纳假设

$\le s\,k + (d - 3\,s / 28)\,k$

$\le s\,k$

综上所述, 对任意 $n > 0$, $T(n) \le s\,n$, 即 $T(n) = O(n)$, 所以仍然是线性时间算法.

3. 分治

2.25 在线性时间选择算法中, 输入元素被划分为5个一组, 如果将它们划分为7个一组, 该算法仍然是线性时间算法吗? 划分成3个一组又怎样?

解:(2)若分成3个一组, 则由于1/3+3/4=13/12>1而得不到$T(n) = O(n)$.

事实上可以用数学归纳证明 $T(n) = O(n^t)$, 其中t是满足$(1/3)^t+(3/4)^t=1$ 的实数($t\approx 1.152$).

若划分为3个一组, 则存在常数 n_0, c, d > 0使得

$$T(n) \le T(n/3) + T(3n/4) + d\,n,\ n > n_0;\ T(n) \le c,\ n \le n_0.$$

以下归纳证明存在s > 0, 对任意 n > 0, $T(n) \le s\,n^t - w\,n$, 其中 $w = 12d$.

首先存在s>0, 对于 $n \le n_0$, 有 $T(n) \le c \le s\,n^t - w\,n$.

其次归纳假设对于 $k > n_0$, 任意 $n < k$ 有 $T(n) \le s\,n^t - w\,n$.

于是 $T(k) \le T(k/3) + T(3k/4) + d\,k$, //迭代1次

$\le s\,(k/3)^t - w(k/3) + s\,(3k/4)^t - w(3k/4) + d\,k$, //归纳假设

$\le s\,k^t - w\,k - (w/12 + d)\,k$

$= s\,k^t - w\,k$

综上所述, 对任意 n > 0, $T(n) \le s\,n$, 即 $T(n) \le s\,k^t - w\,k = O(n^t)$. 最坏情况超过线性时间

3. 分治

2.25 在线性时间选择算法中，输入元素被划分为5个一组，如果将它们划分为7个一组，该算法仍然是线性时间算法吗？划分成3个一组又怎样？

注: 参考递推关系存在常数 n_0, c, d > 0使得

$T(n) \le T(n/3) + T(2n/3) + d\,n,\ n > n_0;\ T(n) \le c,\ n \le n_0.$

以下归纳证明存在s > 0, 对任意 n > 0, $T(n) \le s\,n\log_2 n$, 其中 $s>\max\{c,3d\}$.

首先存在s>0, 对于 $1< n \le n_0$, 有 $T(n) \le c \le s\,n\log_2 n$.

其次归纳假设对于 $k > n_0$, 任意 $1< n < k$ 有 $T(n) \le s\,n\log_2 n$.

于是 $T(k) \le T(k/3) + T(2k/3) + d\,k$, //迭代1次

$\le s\,(k/3)\log_2 k/3 + s\,(2k/3)\log_2(2k/3) + d\,k$, //归纳假设

$\le s\,k\log_2 k - k(\ s(\log_2 27/4)/3 - d\ )$

$< s\,k\log_2 k$

综上所述，对任意 $n > 0$, $T(n) \le s\,n\log_2 n$, 最坏情况超过线性时间.

3. 分治

2.25 在线性时间选择算法中, 输入元素被划分为5个一组, 如果将它们划分为7个一组, 该算法仍然是线性时间算法吗? 划分成3个一组又怎样?

注1:有同学得到更精确的递推公式:

$$T(n) \le T(n/7) + T(5n/7+8) + d\,n,$$

$$T(n) \le T(n/3) + T(2n/3+4) + d\,n$$

注2: 有同学对于$T(n) = T(n/7) + T(3n/4) + d\,n$ 给出了下面的计算方法

$$T(n) \approx dn + dn(1/7+3/4) + dn(1/7+3/4)^2 + dn(1/7+3/4)^3 + \dots$$

$$= dn\ 1/\ (1-1/7-3/4) = 28dn/3$$

4. 动态规划

1. 考虑下面的整数线性规划问题 .

即给定 序列$a_1, a_2,\dots,a_n$, 求

$$\max\ c_1x_1+c_2x_2+\dots+c_nx_n,$$

满足 $a_1x_1+a_2x_2+\dots+a_nx_n \le b$, x_i为非负整数

解: 动态规划, 子结构[1:i], OSP

设f[i][k]为用X[1:i]组合出重量k的最大价值

则 f[i][k]=max{f[i-1][k],f[i][k-x[i]]+c[i]}

去掉第1维坐标,

修改拆分方案数2即可得到算法.

时间复杂度O(nb)

注意递推关系二维, 编程可以一维.

1. 初始f[1:n]=0, f[0]=0
2. 对i=1:n, 对s=X[i]:b,
3. 若 c[i]+f[s-X[i]]>f[s],
4. 则f[s]+=f[s-X[i]]+c[i]
5. 输出f[b]

4. 动态规划

2. 石子合并问题

问题描述: 在一个圆形操场的四周摆放着n堆石子. 现在要将石子有次序地合并成一堆. 规定每次只能选相邻的2堆石子合并成一堆, 并将新的一堆石子数记为该次合并的得分. 试设计一个算法, 计算出将n堆石子合并成一堆的最小得分和最大得分.

算法设计: 对于给定n堆石子, 计算合并成一堆的最小得分和最大得分.

数据输入: 由文件input.txt提供输入数据. 文件的第1行是正整数n, $1 \le n \le 100$, 表示有n堆石子. 第2行有n个数, 分别表示n堆石子的个数.

结果输出: 将计算结果输出到文件output.txt, 文件第1行是最小得分, 第2行是最大得分.

输入文件示例
input.txt
4
4 4 5 9

输出文件示例
output.txt
43
54

4. 动态规划

解: 圆周上石子合并, 子结构[i:j], 当j>n时指跨过第n堆到第j%n堆.

- 定义m[i][len]为合并第i堆到第i+len-1堆石子能得到的最少分数

  x[i][len]为合并第i堆到第i+len-1堆石子能得到的最多分数
- $m[i][len] = \min\{ m[i][k]+m[i+k][len-k]+ sum[i:i+len-1] \mid 0 \le k < len\}$
- $x[i][len] = \max\{ x[i][k]+x[i+k][len-k]+ sum[i:i+len-1] \mid 0 \le k < len\}$

```
1. 对 i = 1 到 n,  m[i][1]=0, x[i][1]=0
2. 对 len = 2 到 n,  对i = 1 到 n
3.      j=i+len-1;  m[i][len] = m[i][1]+m[i+1][len-1]+ sum[i:j];
4.      x[i][len] = x[i][1]+x[i+1][len-1]+ sum[i:j];
5.      对 k = 2 到 len-1
6.         t=m[i][k]+m[i+k][len-k]+ sum[i:j],
7.         若m[i][len]>t, 则m[i][len]=t;
8.         t=x[i][k]+x[i+k][len-k]+ sum[i:j],
9.         若x[i][len]<t, 则x[i][len]=t;
10. 输出min{m[i][n]}
11. 输出max{x[i][n]}
```

sum[i:j]是第i堆到第j堆石子总数

时间复杂度$O(n^3)$

此外还可以

- 打印合并次序
- 加速

4. 动态规划

参考分析: 讨论直线上石子合并问题的算法

- 动规, 子结构[i:j], OSP, 类似于矩阵连乘问题
- 定义m[i,j]为从第i堆到第j堆的石子合并能得到的最少分数, 那么

  $$m[i,j] = \min \{ m[i,k]+m[k+1,j]+ sum[i:j] \quad | \, i \le k < j\}$$

  其中sum[i:j]是第i堆到第j堆石子总数
- 修改矩阵连乘公式可以得到下面的算法(其中s[i,j]是最佳分断点)

```
1. 对 i = 1 到 n,  m[i,i]=0,
2. 对 r = 1 到 n-1
3.      对i = 1 到 n-r
4.          j = i + r; s[i,j] = i;
5.          m[i,j] = m[i,i]+m[i+1,j]+ sum[i:j];
6.          对 k = i + 1 到 j-1
7.              t =m[i,k]+m[k+1,j]+ sum[i:j],
8.              若m[i,j]>t, 则m[i,j]=t; s[i,j]=k;
```

输出m[1,n], 合并次序

```
Traceback(i, j, s)
1.  若i = = j, 打印 a[i]
2.  否则    打印 "("
3.        Traceback(i, s[i,j], s)
4.        打印 "+"
4.        Traceback(s[i,j]+1,j,s)
5.        打印 ")"
```

4. 动态规划

参考分析加速:

- 上面的程序计算耗费$O(n^3)$时间
- 由于本问题满足动态规划加速原理, 最佳分断点满足

$$s[i,j-1] \le s[i,j] \le s[i+1,j]$$

所以程序可以修改如下

1. 对 i = 1 到 n, m[i,i]=0, s[i,i]=0
2. 对 r = 1 到 n-1
3. 　　对i = 1 到 n-r
4. 　　　　j = i + r; div=s[i,j-1]; m[i,j] = m[i,div]+m[div+1,j]+ sum[i:j];
5. 　　　　对 k = div + 1 到 s[i+1,j]
6. 　　　　　　t =m[i,k]+m[k+1,j]+ sum[i:j],
7. 　　　　　　若m[i,j]>t, 则 m[i,j]=t; s[i,j]=k;

输出m[1,n]

4. 动态规划

3. 数字三角形问题

问题描述: 给定一个有n行数字组成的数字三角形, 如下图所示. 试设计一个算法, 计算出从三角形的顶至底的一条路径, 使该路径经过的数字和最大.

算法设计: 对于给定的n行数字组成的三角形, 计算从三角形顶至底的路径经过的数字和的最大值.

数据输入: 由文件input.txt提供输入数据. 文件的第1行数字三角形的行数n, 1≤n≤100. 接下来n行是数字三角形各行中的数字. 所有数字在0~99之间.

结果输出: 将计算结果输出到文件output.txt, 文件第1行中的数是计算出的最大值.

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

数字三角形

输入文件示例

input.txt

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

输出文件示例

output.txt

30

4. 动态规划

动规, 两种方式, 自顶向下, 自底向上

- 自顶向下, 子结构1:i(行)

定义m[i,j]为从第1行到第i行第j列能得到的最大分数, 那么

m[i,j] = a[i,j] + max { m[i-1,j], m[i-1,j-1]}, 当j≤i; =0, 当j>i或j=0.

注: 递推关系不能去掉第1维, 编程可去掉第1维

根据递推公式编程

1. m[1,1]=a[1,1], m[1,0]=0,
2. 对 i = 2 : n
3. 对j = 1 : i
4. m[i,j]=a[i,j];
5. 若m[i-1,j-1]>m[i-1,j]
6. 则m[i,j]+=m[i-1,j-1]
7. 否则 m[i,j]+=m[i-1,j]
8. 输出 max { m[n,j] | 1≤j≤n }

去掉第1维坐标编程

1. m[1]=a[1,1], m[0]=0, m[2:n]=0
2. 对 i = 2 : n
3. 对j = i : 1
4. 若m[j-1]>m[j], 则m[j]=m[j-1]
5. m[j]+=a[i,j]
6. 输出 max { m[j] | 1≤j≤n }

4. 动态规划

动规, 两种方式, 自顶向下, 自底向上

- 自底向上 , 子结构1:i(行)

定义m[i,j]为从第n行到第i行第j列能得到的最大分数, 那么

m[i,j] = a[i,j] + max { m[i+1,j], m[i+1,j+1]}, 当j≤i

根据递推公式编程

1. m[n,i]=a[n,i],
2. 对 i = n-1 : 1
3.   对j = 1 : i
4.   m[i,j]=a[i,j];
5.   若m[i+1,j+1]>m[i+1,j]
6.   则m[i,j]+=m[i+1,j+1]
7.   否则 m[i,j]+=m[i+1,j]
8. 输出 m[1,1]

去掉第一维编程

1. 对j=1:n, m[j]=a[n,j],
2. 对 i = n-1 到 1
3.   对j = 1 到 i
4.   若m[j+1]>m[j], 则m[j]=m[j+1]
5.   m[j]+=a[i,j],
6. 输出m[1]

4. 动态规划

算法实现题: 租用游艇问题
问题描述: 长江游艇俱乐部在长江上设置了n个游艇出租站1,2,…,n. 游客可在这些游艇出租站租用游艇, 并在下游的任何一个游艇出租站归还游艇. 游艇出租站i到出租站j之间的租金为r(i,j), 1≤i<j≤n. 试设计一个算法, 计算出从游艇出租站1到游艇出租站n所需的最少租金, 并分析算法的计算复杂性.
算法设计: 对于给定的游艇出租站i到游艇出租站j的租金r(i,j), 1≤i<j≤n. 计算出租站1到n所需的最少租金.
数据输入: 由文件input.txt提供输入数据. 文件的第1行有一个正整数n, n≤200, 表示有n个游艇出租站. 接下来n-1行是r(i,j), 1≤i<j≤n.
结果输出: 将计算出的游艇出租站1到n最少租金输出到文件output.txt.

输入文件示例
input.txt
3
5 15
7

输出文件示例
output.txt
12

4. 动态规划

解法一: 子结构OSP分析同全路径最短路

1. D[i,j] = r[i,j],
2. 对k=1:n
3.   对i=1:n, 对j=1:n
4.   若 D[i,k]+D[k,j] < D[i,j]
5.   则 D[i,j] = D[i,k]+D[k,j];
6. 输出D[1,n]

时间$O(n^3)$:

45两步常数时间

23三重循环$O(n^3)$

解法二: 子结构OSP分析同单源最短路

1. 初始d[1]=0, 其它点d[u]=INF, S空, Q=V
2. 当Q非空
3.   取出Q中u使得d[u]最小
4.   将u添加到S中
5.   对u的每个邻居v, 松弛(u,v).

松弛(u,v):

1. 若d[v]>d[u]+r(u,v),
2. 则d[v]=d[u]+r(u,v).

注意到边数$O(n^2)$.

Q用数组时间$O(n^2)$:

3时间$O(n)$, 23时间$O(n^2)$, 45总和时间$O(n^2)$

Q用最小堆$O(n^2 \log n)$:

23时间$O(n \log n)$, 45总和时间$O(n^2 \log n)$.

4. 动态规划

解法三: 依题意r(i,j)只有i<j的值, 有下面的算法

取子结构1:j, 定义f[j]为从1到j的最少租金

$f[j] = \min \{ f[i] + r[i,j] \mid 1 \le i < j \}$

1. f[1]=0, f[2:n]=INF
2. 对j=2:n,对i=1:j-1,
3. 若 f[j]>f[i]+r[i,j],
4. 则 f[j]=f[i]+r[i,j]
5. 输出f[n]

时间$O(n^2)$