- 有一座高度是**10**级台阶的楼梯，从下往上走，每跨一步只能向上**1**级或者**2**级台阶。要求用程序来求出一共有多少种走法。

计算理论与
算法分析设计

李玉岗

教材:

[1][王] 王晓东,计算机算法设计与分析(第4版),电子工业.

[2][S] 唐常杰等译, Sipser著, 计算理论导引, 机械工业.

参考资料:

[3][C] 潘金贵等译, Cormen等著, 算法导论, 机械工业.

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

[5][刘] 刘汝佳等, 算法艺术与信息学竞赛, 清华大学.

[6][L] Lewis等著, 计算理论基础, 清华大学.

第三章 动态规划
dynamic programming

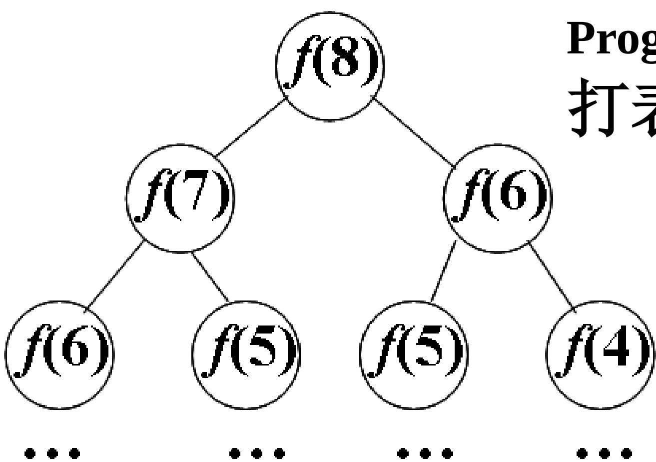
1. 动态规划一般原理(与分治对比), Bellman, OSP
2. 动态规划设计步骤: 矩阵连乘
3. 如何设计动态规划算法: 子结构和策略
   最长公共子序列, 最大子段和, 最长递增子序列
4. 背包问题 (动态规划与贪心对比)
5. 最短路问题 (全路径, Bellman-Ford, Dijkstra)

动态规划与分治

- 分治的过程:
  分解—递归解子问题—合并
- 若有大量重复子问题, 则不宜分治
- 举例: Fibonacci数的计算

$f(n) = f(n\text{-}1) + f(n\text{-}2)$, $f(1)=1$, $f(0)=0$.

输入n, 输出$f(n)$.



Programming的含义:
打表记录中间结果

输入规模?
$\log_2 n$

递归:

```
int f(int n)
{ if(n<2) return(1);
  return(f(n-1)+f(n-2);}
```

$2^{O(n)}$时间, O(n)空间

动态规划:

```
f[1]=1;f[0]=0;
for(i=2,i<n,i++)
    f[i]=f[i-1]+f[i-2];
```

$O(n)$时间, $O(n)$空间

```
f=1; b=0;i=1;
while(i++<n){
  temp=f;f=f+b;b=temp;}
```

$O(n)$时间, $O(1)$空间

Fibonacci数矩阵算法与DP

由 $f(n) = f(n\text{-}1) + f(n\text{-}2)$, 知

$$\begin{bmatrix} f(n+1) \\ f(n) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} f(n) \\ f(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} f(1) \\ f(0) \end{bmatrix}$$

求幂问题: 输入: x, n; 输出: x^n.
设二进制表示$n=(b_k\ldots b_1b_0)_2$, 则

1. 对$i=0$到$k-2$, 计算$x^{2^{i+1}} = x^{2^i} \times x^{2^i}$
2. 计算$x^n = x^{\sum_{i=0}^{k-1} b_i \times 2^i} = \prod_{i=0}^{k-1}\left(x^{2^i}\right)^{b_i}$

- 乘法次数O(log$n$)
- 记录个数O(log$n$)
- 输入规模O(log$n$)

Fibonacci数$f(n)$ mod M矩阵算法

- 问题: 输入n, 输出 $f(n)$ mod M.
- 输入规模: Θ(log n). 算法:

0. A=(b_k,…,b_1,b_0), B[0]=((1,1),(1,0)), ans = ((1,0),(0,1))
1. 对 i = 0..k-1,
2. 计算B[i+1] = B[i]*B[i] mod M
3. 对 i = 0..k,
4. 若A[i]==1, 则 ans = ans * B[i] mod M
5. 输出ans

- 时间O(log n), 空间O(log n), 打表记录中间结果

DP适用条件和设计步骤

- 最优子结构性质, optimal substructure property

  OSP: 最优策略的子策略也是最优.

- 重叠子结构性质

  Programming是指使用表格化的算法.

- Bellman, 1955, 奠定DP数学基础.

设计步骤 [王]

1) 描述最优解的结构
2) 递归定义最优解
3) 自底向上计算最优值
4) 由计算结果构造最优解

第三章 动态规划
dynamic programming

1. 动态规划一般原理(与分治对比), **Bellman, OSP**
2. 动态规划设计步骤: 矩阵连乘
3. 如何设计动态规划算法: 子结构和策略
   最长公共子序列, 最大子段和, 最长递增子序列
4. 背包问题 (动态规划与贪心对比)
5. 最短路问题 (全路径, **Bellman-Ford, Dijkstra**)

矩阵连乘问题([王])

- 输入: 给定矩阵$A_1, A_2, \ldots, A_n$, A_i与A_{i+1}可乘
- 输出: 计算量最小的乘法次序
- 输入样例: A_1(10×100阶), A_2(100×5阶), A_3(5×50阶),
- 两种计算次序: $((A_1 \times A_2) \times A_3)$, $(A_1 \times (A_2 \times A_3))$

  $A_1 \times A_2$的计算量: 10×100×5 (乘法次数)

  $((A_1 \times A_2) \times A_3)$: 10×100×5 + 10×5×50 = 7500

  $(A_1 \times (A_2 \times A_3))$: 100×5×50 + 10×100×50 = 75000
- 样例输出: $((A_1 \times A_2) \times A_3)$
- 取整数序列$q_0, q_1, \ldots, q_n$, 设A_i是 $q_{i-1} \times q_i$ 阶矩阵
- n个矩阵连乘不同次序个数:

$$\frac{1}{n}\binom{2n-2}{n-1} = \Omega(2^n)$$

Catalan数

分析最优解结构、建立递推关系

假设定好了$A_1 \ldots A_n$一个乘法次序P

- 用$A[i:j]$记连乘积 $A_i \ldots A_j$, 相应计算量T[i,j]
- 设P最后乘法在A_k后断开, 即 A[1:k]×A[k+1:n]
  那么P的计算量为 T[1,k] + T[k+1,n] + $q_0 \times q_k \times q_n$.
- 若P最优, 则P在A[1:k]和A[k+1:n]上也最优
  最优子结构性质：最优策略的子策略也是最优.

设$A[i:j]$的最小计算量为m[i,j], 那么

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

最优值与最优解的区别

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

- 输入: 给定矩阵$A_1, A_2,\ldots, A_n$的维数序列:

  $q_0,q_1,\ldots,q_n$, 即 A_i 是 $q_{i\text{-}1} \times q_i$ 阶矩阵
- 输出: 计算量最小的乘法次序
- 最优解是要输出的次序

  最优值是最优解的计算量

DP适用条件和设计步骤

- **OSP:** 最优策略的子策略也是最优.
- 重叠子问题性质: 记录中间结果.

设计步骤 1) 描述最优解的结构

2) 递归定义最优解

3) 自底向上计算最优值

4) 由计算结果构造最优解

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

观察最优值计算

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

- 需要计算的是m[1,n], 但没法直接计算
- m[1,1]=m[2,2]=…=m[n,n]=0
- m[1,2] = ?

  m[1,2] = $q_0 \times q_1 \times q_2$, m[2,3] = $q_1 \times q_2 \times q_3$, m[3,4], …
- m[1,3] = ?

  min{ m[1,1] + m[2,3] + $q_0 \times q_1 \times q_3$, m[1,2] + m[3,3] + $q_0 \times q_2 \times q_3$ }
- 自底向上计算；表格化方法

计算最优值图示

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

设维数序列为(30, 35, 15, 5, 10, 20, 25)

m		i					
		1	2	3	4	5	6
j	1						
	2						
	3						
	4						
	5						
	6						

m		i					
		1	2	3	4	5	6
j	1						
	2						
	3						
	4						
	5						
	6						

计算最优值图示

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

(30, 35, 15, 5, 10, 20, 25)

$m[1,2] = q_0 \times q_1 \times q_2 =$
$30 \times 35 \times 15 = 15750$

m		*i*					
		1	2	3	4	5	6
j	1						
	2						
	3						
	4						
	5						
	6						

m		*i*					
		1	2	3	4	5	6
j	1	0	15750				
	2		0	2625			
	3			0	750		
	4				0	1000	
	5					0	5000
	6						0

计算最优值图示

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

$m[1,3] = \min\{ m[1,1]+m[2,3]+q_0\times q_1\times q_3,\ m[1,2]+m[3,3]+q_0\times q_2\times q_3 \}$

$= \min\{ 0+2625+30\times35\times5,\ 15750+0+30\times15\times5 \} = \min\{18000, 7875\}$

m		*i*					
		1	2	3	4	5	6
j	1						
	2						
	3						
	4						
	5						
	6						

m		(30, 35, 15, 5, 10, 20, 25)					
		1	2	3	4	5	6
j	1	0	15750	7875			
	2		0	2625	4375		
	3			0	750	2500	
	4				0	1000	3500
	5					0	5000
	6						0

计算最优值图示

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

$$
\begin{aligned}
m[1,4] = \min\{ & m[1,1]+m[2,4]+q_0\times q_1\times q_4, \\
& m[1,2]+m[3,4]+q_0\times q_2\times q_4, \\
& m[1,3]+m[4,4]+q_0\times q_3\times q_4\} \\
= \min\{ & 0 + 4375 + 30\times35\times10, \\
& 15750 + 750 + 30\times15\times10, \\
& 7875 + 0 + 30\times5\times10\} \\
= \min\{ & 9375, 21000, 14875 \}
\end{aligned}
$$

m		(30, 35, 15, 5, 10, 20, 25)					
		1	2	3	4	5	6
j	1	0	15750	7875	9375		
	2		0	2625	4375	7125	
	3			0	750	2500	5375
	4				0	1000	3500
	5					0	5000
	6						0

计算最优值图示

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

m[1,5] = min { 9375 + 0 + 30×10×20, 7875 + 1000 + 30×5×20,

15750 + 2500 + 30 × 15 × 20, 0 + 7125 + 30 × 35 × 20 }

= min { 15750, 11875, 27250, 28125 }

m		i					
		1	2	3	4	5	6
j	1						
	2						
	3						
	4						
	5						
	6						

m		(30, 35, 15, 5, 10, 20, 25)					
		1	2	3	4	5	6
j	1	0	15750	7875	9375	11875	15125
	2		0	2625	4375	7125	10500
	3			0	750	2500	5375
	4				0	1000	3500
	5					0	5000
	6						0

计算最优值算法

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

输入: n和维数序列 $(q_0,q_1,\ldots,q_n)$

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; m[i,j] = INF;
5. 　　　　对 k = i 到 j-1
6. 　　　　　　t =m[i,k]+m[k+1,j]+ $q_{i-1}\times q_k\times q_j$,
7. 　　　　　　若m[i,j]>t, 则m[i,j]=t
8. 输出: m[1,n]

INF如何处理? 如何构造最优解?

计算最优值同时标记分断点

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

输入: n和维数序列 $(q_0,q_1,\ldots,q_n)$

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]+ $q_{i-1}\times q_i\times q_j$;
6. 　　　　对 k = i + 1 到 j-1
7. 　　　　　　t =m[i,k]+m[k+1,j]+ $q_{i-1}\times q_k\times q_j$,
8. 　　　　　　若m[i,j]>t, 则m[i,j]=t; s[i,j]=k;

输出: s //s[i,j]是计算[i:j]段的分断点

构造最优解

Traceback(1, n, s) //输出最优解, s[i,j]是[i:j]的最优分断点

Traceback(i, j, s)   //[C]

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

10, 100, 5, 50

s[1,3]=2

输出样例:

((A1 A2) A3)

Traceback2(i,j,s)   //[王], 书中解释不匹配

1. 若 i == j 返回
2. Traceback2(i, s[i,j],s)
3. Traceback2(s[i,j]+1,j,s)
4. 打印 "A", i, ",", s[i,j], "× A", s[i,j]+1, ",", j

输出样例

A 1,1 × A 2,2

A 1,2 × A 3,3

DP适用条件和设计步骤

- **OSP: 最优策略的子策略也是最优.**
- **重叠子问题性质：记录中间结果.**

设计步骤 1) 描述最优解的结构

2) 递归定义最优解

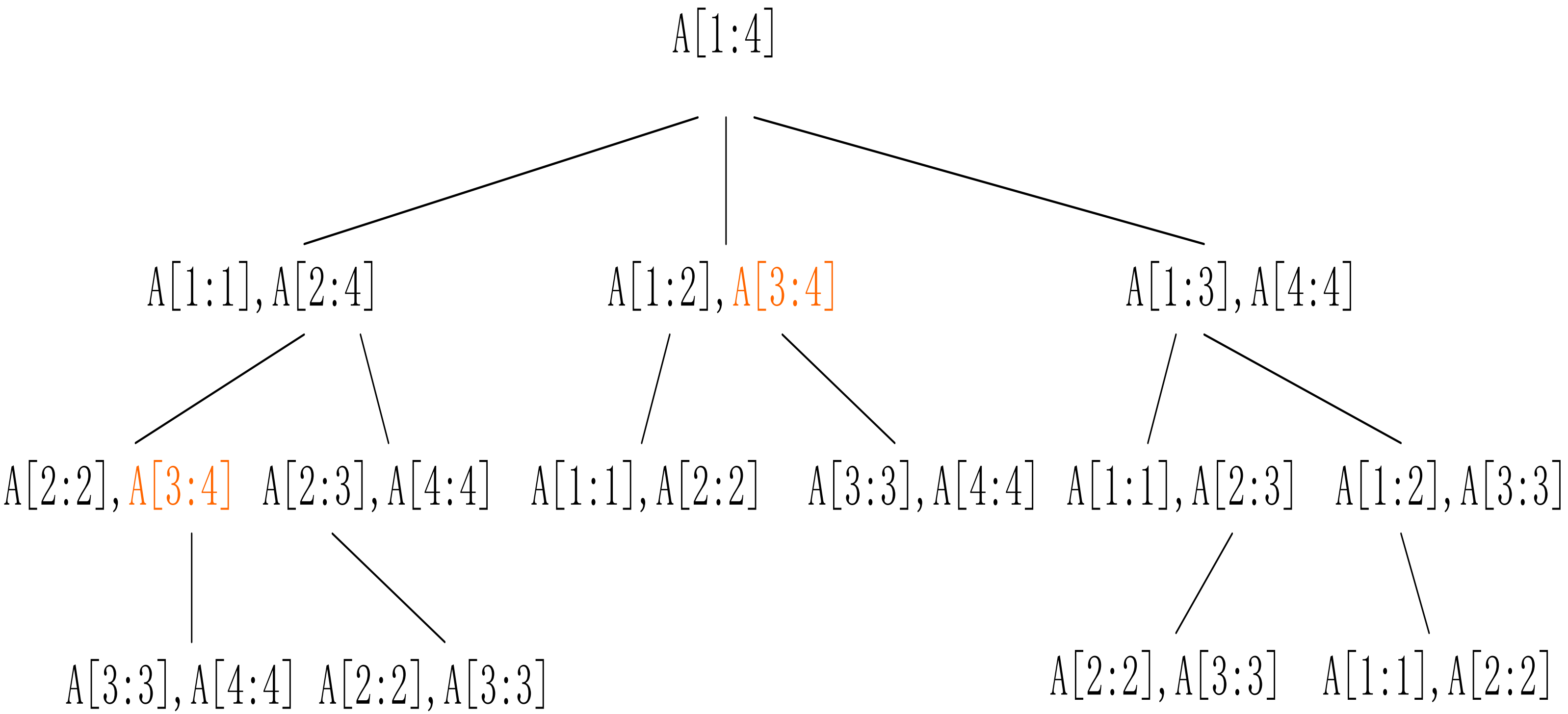
3) 自底向上计算最优值 → 自顶向下

4) 由计算结果构造最优解

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

重叠子问题

自顶向下计算m[1][4]过程如下:



设置备忘录

备忘录方法: 自顶向下

- **MEMOIZED-MATRIX-CHAIN**

  1. 对所有i,j, m[i,j]=0
  2. LU(1,n)   //LookUp

- **LU(i,j)**

  1. 若 m[i,j]>0, 返回 m[i,j]
  2. 若 i = = j, 返回0
  3. s[i,j] = i; m[i,j] = LU(i,i) + LU(i+1,j) + $q_{i-1}\times q_i\times q_j$;
  4. 对 k = i + 1 到 j-1
  5.   t = LU(i, k) + LU(k+1,j) + $q_{i-1}\times q_k\times q_j$,
  6.   若m[i,j]>t, 则m[i,j]=t; s[i,j]=k;
  7. 返回m[i][j]

自底向上与自顶向下的比较

- 动态规划方法采用自底向上
- 备忘录方法采用自顶向下
- 都能解决重叠子问题
- 当所有子问题都至少要求解一次时，用动态规划方法比较好
- 当部分子子问题不用求解时，用备忘录方法比较好
- 矩阵连乘问题宜用动态规划

第三章 动态规划
dynamic programming

1. 动态规划一般原理(与分治对比), **Bellman, OSP**
2. 动态规划设计步骤: 矩阵连乘
3. 如何设计动态规划算法: 子结构和策略
   最长公共子序列, 最大子段和, 最长递增子序列
4. 背包问题 (动态规划与贪心对比)
5. 最短路问题 (全路径, **Bellman-Ford, Dijkstra**)

如何使用DP解决问题

- DP的适用条件是OSP和重叠子问题
- 动态规划的关键是子结构和决策(量)
- 如何设计?
- 递归思考: 由小到大, 由简到繁
- 尝试、调整子结构和决策量.
- 矩阵连乘 :子结构, 决策, 决策量?

  子结构--[i:j], 决策--分段点, 决策量: 计算量
- 这里的决策(量)是自然的, 有时需要调整
- 怎么会想到用这样的子结构呢?

  多试几步就能发现: [1:n]→[1:k], [k+1,n]→…

最长公共子序列(子结构与决策量)

- 输入: 字符串 $X=x_1x_2\ldots x_n$, $Y=y_1y_2\ldots y_m$,
- 输出: X和Y最长的公共子序列(LCS)
- 样例: X=ABCBDAB, Y=BDCABA, 输出: BCAB
- 自然的子结构: $X_i = x_1x_2\ldots x_i$, $Y_j = y_1y_2\ldots y_j$.
- 自然的决策(量): LCS(长度)
- 如何寻找最优决策? (回顾矩阵连乘问题)
- 递归思考: 如果 $x_i = y_\mathrm{j}$, 那么
  (X_i,Y_j)的LCS一定可以含有(x_i, y_j)配对;
- 如果 $x_i \neq y_\mathrm{j}$, 那么
  (X_i,Y_j)的LCS是(X_{i-1},Y_j)或(X_i,Y_{j-1})的LCS.

LCS: 递归定义最优解

- 输入: X=$x_1x_2\ldots x_n$, Y=$y_1y_2\ldots y_m$, 输出: X和Y的LCS
- 定义: c[i][j] = X_i,Y_j的LCS长度(OSP)

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

LCSlength(n,m,x,y,c)

1. 初值 c[0][1:m]=0, c[1:n][0]=0
2. 对i=1:n, j=1:m
3. 若x[i]==y[j], 则 c[i][j]=c[i-1][j-1]+1
4. 否则 若c[i][j-1]>c[i-1][j], 则 c[i][j]=c[i][j-1]
5. 否则 c[i][j]=c[i-1][j]

输出c[n][m]. //构造最优解略. 如何减少存储空间?见附录

最大子段和(ms,[王,M])

- 输入: 实数序列($x_1,x_2,\ldots,x_n$)
- 输出: 有最大和的子段 (x_i, $x_{i+1},\ldots,x_j$)
- 输入样例: (−2, 1, −3, 4, −1, 2, 1, −5, 4)
- 样例输出: (4, -1, 2, 1)
- 注: ms ≥ 0.

直接、分治(见附录)、动规都试一下

直接法(直接法是基础):

对i=1:n, j=i:n, 求[i:j]段的和. $O(n^3)$

改进的直接法:

1. 对i, 求[1:i]和. 2. 对i,j, 求[i:j]和.

$O(n)+O(n^2)=O(n^2)$

最大子段和(ms)的递归思考

- 输入: (x_1,x_2,…,x_n), 输出: ms. 例: (2,-3,1.5,-1,3,-2,-3,3)
- 使用什么子结构,决策量? [i:j], [1:i]? 试[1:i], ms.
- 递归尝试: 已知[1:i-1]的ms[i-1], 求[1:i]的ms[i]?
- ms[i-1] : max{ ∅,[1:1],[1:2],…,[1:i-1],…,[i-1,i-1] }
- ms[i] : max{ ms[i-1], [1:i], [2:i], …, [i:i] }.
- T(i) = T(i-1) + O(i)? 复杂度 O(n^2)? 怎么改进?
- tms[i]: ∅, [1:i], [2:i], …, [i:i]. ms[i]: ms[i-1], tms[i]
- 递归调整: 已知[1:i-1]的ms, tms, 求[1:i]的ms, tms
- tms[i-1]: [1:i-1], [2:i-1], …, [i-1:i-1],∅.
- tms[i]: tms[i-1]+[i:i], ∅.
- tms的最优子结构性质OSP?

最大子段和(ms)的递归思考

- 输入: $(x_1,x_2,\ldots,x_n)$, 输出: ms. 例: (2,-3,1.5,-1,3,-2,-3,3)
- 使用什么子结构、决策量?
- 子结构: [1:i], 决策量: ms, tms.
- ms[i] := [1:i]的最大子段和
- tms[i] := [1:i]的最大尾部子段和
- 递归: 已知[1:i-1]的ms, tms,
  求[1:i]的ms, tms

tms[0] = ms[0] =0

tms[i] = max{ 0, x_i + tms[i-1] }, i ≥ 1

ms[i] = max{ ms[i-1], tms[i] }, i ≥ 1

算法

1. ms=0 //ms
2. tms=0 //tms
3. 对i=1:n
4. tms+=x[i]
5. 若tms<0, tms=0
6. 若tms>ms, ms=tms

- 时间O(n)
- 添递归量
- OSP: tms
- 减少了存储

最大子段和(MS)

序号	序列	tms	ms	借用/*(ms更新)
0		0	0	
1	2	2	2	0/*
2	-3	0	2	0
3	1.5	1.5	2	0
4	-1	0.5	2	3
5	3	3.5	3.5	4/*
6	-2	1.5	3.5	5
7	-3	0	3.5	0
8	3	3	3.5	0

算法

1. ms=0 //ms
2. tms=0 //tms
3. 对i=1:n
4. tms+=x[i]
5. 若tms<0, tms=0
6. 若tms>ms, ms=tms

最长递增子序列(LIS[M])

- 输入: 实数序列($x_1,x_2,\ldots,x_n$)
- 输出: $x_{i_1}\le x_{i_2}\le\ldots\le x_{i_k}$, 其中k最大且$i_1< i_2<\ldots<i_k$.
- 输入样例: (2,8,9,4,6,1,3,7,5,10), OSP?
- 输出样例: (2,4,6,7,10)
- 穷搜(指数时间?) DP? 子结构? 决策量?

归纳尝试一: 已知[1:i]的1个LIS, 求[1:i+1]的LIS?

- 假设已知 (2,8,9,4,6,1,3) 的1个LIS (2,8,9), 加入7
- 7不能使(2,8,9)变长, 但是可以使(2,4,6)变长
- 领悟: x_{i+1}可能可以使得其它LIS变长
- 调整: 记录尾巴最小的LIS: MTLIS

添加修改递归量

- 输入: 实数序列$(x_1,x_2,\ldots,x_n)$
- 输出: $x_{i_1}\le x_{i_2}\le\ldots\le x_{i_k}$, 其中k最大且$i_1< i_2<\ldots<i_k$.
- 尾巴最小的LIS: MTLIS

归纳尝试二: 知[1:i]的MTLIS, 求[1:i+1]的MTLIS?

- 假设已知(2,8,9,4) 的MTLIS (2,8,9), 加入6
- 6不能使(2,8,9)变长, 但(2,4,6)是新的MTLIS
- 领悟: x_{i+1}可能可以构成新的MTLIS
- 调整: 需要记录最长和次长的MTIS
- 调整: 需要记录长为k的MTIS: MTIS(k), k=1,2,…

归纳尝试三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

MTIS计算举例

- 输入: 实数序列($x_1,x_2,\ldots,x_n$)
- 输出: $x_{i_1} \le x_{i_2} \le \ldots \le x_{i_k}$, 其中k最大且$i_1 < i_2 < \ldots < i_k$.
- 长为k, 尾巴最小的IS: MTIS(k), k=1,2,…

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
MTIS(1)	2									
MTIS(2)	2	8								
MTIS(3)	2	8	9							

加入4

能否得到MTIS(4)

能否改变MTIS(3)

能否改变MTIS(2)

能否改变MTIS(1)

MTIS计算举例

- 输入: 实数序列$(x_1,x_2,\ldots,x_n)$
- 输出: $x_{i_1} \le x_{i_2} \le \ldots \le x_{i_k}$, 其中k最大且$i_1 < i_2 < \ldots < i_k$.
- 长为k, 尾巴最小的IS: MTIS(k), k=1,2,…

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
MTIS(1)	2									
MTIS(2)	2			4						
MTIS(3)	2	8	9							

加入6

能否得到MTIS(4)
能否改变MTIS(3)
能否改变MTIS(2)
能否改变MTIS(1)

MTIS计算举例

- 输入: 实数序列$(x_1,x_2,\ldots,x_n)$
- 输出: $x_{i_1}\leq x_{i_2}\leq\ldots\leq x_{i_k}$, 其中k最大且$i_1< i_2<\ldots<i_k$.
- 长为k, 尾巴最小的IS: MTIS(k), k=1,2,…

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
MTIS(1)	2									
MTIS(2)	2			4						
MTIS(3)	2			4	6					

加入1

能否得到MTIS(4)
能否改变MTIS(3)
能否改变MTIS(2)
能否改变MTIS(1)

MTIS计算举例

- 输入: 实数序列($x_1,x_2,\ldots,x_n$)
- 输出: $x_{i_1} \le x_{i_2} \le \ldots \le x_{i_k}$, 其中k最大且$i_1 < i_2 < \ldots < i_k$.
- 长为k, 尾巴最小的IS: MTIS(k), k=1,2,…

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
MTIS(1)						1				
MTIS(2)	2			4						
MTIS(3)	2			4	6					

MTIS计算举例

- 输入: 实数序列($x_1,x_2,\ldots,x_n$)
- 输出: $x_{i_1} \le x_{i_2} \le \ldots \le x_{i_k}$, 其中k最大且$i_1 < i_2 < \ldots < i_k$.
- 长为k, 尾巴最小的IS: MTIS(k), k=1,2,…

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
MTIS(1)						1				
MTIS(2)						1	3			
MTIS(3)	2			4	6					

MTIS计算举例

- 输入: 实数序列($x_1,x_2,\ldots,x_n$)
- 输出: $x_{i_1}\le x_{i_2}\le\ldots\le x_{i_k}$, 其中k最大且$i_1< i_2<\ldots<i_k$.
- 长为k, 尾巴最小的IS: MTIS(k), k=1,2,…

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
MTIS(1)						1				
MTIS(2)						1	3			
MTIS(3)	2			4	6					
MTIS(4)	2			4	6			7		

MTIS计算举例

- 输入: 实数序列($x_1,x_2,\ldots,x_n$)
- 输出: $x_{i_1} \le x_{i_2} \le \ldots \le x_{i_k}$, 其中k最大且$i_1 < i_2 < \ldots < i_k$.
- 长为k, 尾巴最小的IS: MTIS(k), k=1,2,…

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
MTIS(1)						1				
MTIS(2)						1	3			
MTIS(3)						1	3		5	
MTIS(4)	2			4	6			7		

MTIS计算举例

- 输入: 实数序列$(x_1,x_2,\ldots,x_n)$
- 输出: $x_{i_1}\le x_{i_2}\le\ldots\le x_{i_k}$, 其中k最大且$i_1< i_2<\ldots<i_k$.
- 长为k, 尾巴最小的IS: MTIS(k), k=1,2,…

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
MTIS(1)						1				
MTIS(2)						1	3			
MTIS(3)						1	3		5	
MTIS(4)	2			4	6			7		
MTIS(5)	2			4	6			7		10

时间复杂度？
$O(n^3)$?
只记录尾巴?
$O(n^2)$?
OSP?

MTIS计算举例

- 输入: 实数序列($x_1,x_2,\ldots,x_n$)
- 输出: $x_{i_1}\le x_{i_2}\le\ldots\le x_{i_k}$, 其中k最大且$i_1< i_2<\ldots<i_k$.
- 长为k, 尾巴最小的IS: MTIS(k), k=1,2,…

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
MTIS(1)						1				
MTIS(2)						1	3			
MTIS(3)						1	3		5	
MTIS(4)	2			4	6			7		

尾巴递增?

只有一个位置改变?

MTIS(k).last

len: 最大长度

- x_{i+1} < MTIS(1).last
- $x_{i+1}\ge$ MTIS(len).last
- $x_{i+1}\ge$ MTIS(s-1).last

  x_{i+1} < MTIS(s).last

MTIS计算举例

- 输入: 实数序列($x_1,x_2,\ldots,x_n$)
- 输出: $x_{i_1}\le x_{i_2}\le\ldots\le x_{i_k}$, 其中k最大且$i_1< i_2<\ldots<i_k$.
- 长为k, 尾巴最小的IS: MTIS(k), k=1,2,…

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
MTIS(1)	2									
MTIS(2)	2			4						
MTIS(3)	2			4	6					

尾巴递增?

只有一个位置改变?

MTIS(k).last

len: 最大长度

- x_{i+1} < MTIS(1).last
- $x_{i+1}\ge$ MTIS(len).last
- $x_{i+1}\ge$ MTIS(s-1).last

  x_{i+1} < MTIS(s).last

MTIS计算举例

- 输入: 实数序列($x_1,x_2,\ldots,x_n$)
- 输出: $x_{i_1}\le x_{i_2}\le\ldots\le x_{i_k}$, 其中k最大且$i_1< i_2<\ldots<i_k$.
- 长为k, 尾巴最小的IS: MTIS(k), k=1,2,…

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
MTIS(1)						1				
MTIS(2)						1	3			
MTIS(3)	2			4	6					
MTIS(4)	2			4	6			7		

尾巴递增?

只有一个位置改变?

MTIS(k).last

len: 最大长度

- x_{i+1} < MTIS(1).last
- $x_{i+1}\ge$ MTIS(len).last
- $x_{i+1}\ge$ MTIS(s-1).last

  x_{i+1} < MTIS(s).last

只记录尾巴，增加父亲标记

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
父亲	0									
MTIS(1).last	2									
MTIS(2).last										
MTIS(3).last										
MTIS(4).last										
MTIS(5).last										

只记录尾巴，增加父亲标记

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
父亲	0	1								
MTIS(1).last	2									
MTIS(2).last		8								
MTIS(3).last										
MTIS(4).last										
MTIS(5).last										

只记录尾巴，增加父亲标记

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
父亲	0	1	2							
MTIS(1).last	2									
MTIS(2).last		8								
MTIS(3).last			9							
MTIS(4).last										
MTIS(5).last										

只记录尾巴，增加父亲标记

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
父亲	0	1	2	1						
MTIS(1).last	2									
MTIS(2).last				4						
MTIS(3).last			9							
MTIS(4).last										
MTIS(5).last										

只记录尾巴，增加父亲标记

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
父亲	0	1	2	1	4					
MTIS(1).last	2									
MTIS(2).last				4						
MTIS(3).last					6					
MTIS(4).last										
MTIS(5).last										

只记录尾巴，增加父亲标记

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
父亲	0	1	2	1	4	0				
MTIS(1).last						1				
MTIS(2).last				4						
MTIS(3).last					6					
MTIS(4).last										
MTIS(5).last										

只记录尾巴，增加父亲标记

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
父亲	0	1	2	1	4	0	6			
MTIS(1).last						1				
MTIS(2).last							3			
MTIS(3).last					6					
MTIS(4).last										
MTIS(5).last										

只记录尾巴，增加父亲标记

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
父亲	0	1	2	1	4	0	6	5		
MTIS(1).last						1				
MTIS(2).last							3			
MTIS(3).last					6					
MTIS(4).last								7		
MTIS(5).last										

只记录尾巴，增加父亲标记

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
父亲	0	1	2	1	4	0	6	5	7	
MTIS(1).last						1				
MTIS(2).last							3			
MTIS(3).last									5	
MTIS(4).last								7		
MTIS(5).last										

只记录尾巴，增加父亲标记

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
父亲	0	1	2	1	4	0	6	5	7	8
MTIS(1).last						1				
MTIS(2).last							3			
MTIS(3).last									5	
MTIS(4).last								7		
MTIS(5).last										10

MTIS计算举例

- 输入: 实数序列$(x_1,x_2,\ldots,x_n)$
- 输出: $x_{i_1} \le x_{i_2} \le \ldots \le x_{i_k}$, 其中k最大且$i_1 < i_2 < \ldots < i_k$.

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

- 为什么每添加一数只有一个MTIS(k)会改变?
- 性质: MTIS(1).last ≤ MTIS(2).last ≤ …
- 证明: 若MTIS(i).last > MTIS(i+1).last, 矛盾.
- x_{i+1}改变MTIS(s): MTIS(s-1).last ≤ x_{i+1} < MTIS(s).last
- 怎么找修改位置?
- 采用二分搜索找s, 时间O(logn)×n.

[王]第三章习题1,2

LIS算法

- 输入: 实数序列($x_1, x_2, \ldots, x_n$)
- 输出: $x_{i_1} \le x_{i_2} \le \ldots \le x_{i_k}$, 其中k最大且$i_1 < i_2 < \ldots < i_k$.

1. 数组X, L, F //X是输入, F[i]: 记录X[i]的父亲
   //L[i]: X[L[i]]=MTIS(i).last, 即MTIS(i).last的位置
2. L[i] = 0, F[i] = 0; L[1] = 1, F[1] = 0, len = 1 //len:最大长度
3. 对于 i = 2 : n, // X[L[1]] ≤ X[L[2]] ≤ … ≤ X[L[len]]; X[i]?
4. 若X[L[1]] > X[i], 则 F[i] = 0; L[1] = i.
5. 若X[L[len]] ≤ X[i], 则 F[i] = L[len]; len++; L[len] = i.
6. 否则 二分搜索L[1:len], 求s使得X[L[s-1]] ≤ X[i] < X[L[s]]
7. F[i]=L[s-1]; L[s]=i.
8. pt = L[len];
9. 对 i = len:1, D[i] = X[pt]; pt = F[pt]. 输出D.

第三章 动态规划
dynamic programming

1. 动态规划一般原理(与分治对比), **Bellman, OSP**
2. 动态规划设计步骤: 矩阵连乘
3. 如何设计动态规划算法: 子结构和策略
   最长公共子序列, 最大子段和, 最长递增子序列
4. 背包问题 (动态规划与贪心对比)
5. 最短路问题 (全路径, **Bellman-Ford, Dijkstra**)

0-1背包问题([王]p71)

- 输入: n物品重W[1:n], 价值V[1:n], 背包容量C
- 输出: 装包使得价值最大 (物品重量为整数).
- 例: W:{1,2,3,4},V:{2,3,4,9}, C=5
- 子结构? 决策量? 递推关系? 最优值?
- [1:i], g[i,w] = 由[1:i]组合出重量≤w的最大价值
- g[i,w] = max{ g[i-1,w], g[i-1,w-W[i]] + V[i] } (OSP)
- 最优值: g[n,C]

g(i,k)	W,V	k=0	1	2	3	4
g(0,k)		0	0	0	0	0
g(1,k)	(1,2)	0	2	2	2	2
g(2,k)	(2,3)	0	2	3	5	5
g(3,k)	(3,4)	0	2	3	5	6
g(4,k)	(4,9)	0	2	3	5	9

0-1背包: 编程

- 输入: n物品重W[1:n], 价值V[1:n], 背包容量C
- 输出: 装包使得价值最大 (物品重量为整数).
- [1:i], g[i,w] = 由[1:i]组合出重量≤w的最大价值
- g[i,w] = max{ g[i-1,w], g[i-1,w-W[i]] + V[i] } (OSP)
- 最优值: g[n,C]

1. 初始 g[0,0:C] = 0,
2. 对 i = 1:n, 对 w = 0:C,
3. g[i,w] = g[i-1,w]
4. 若w ≥ W[i], pt=g[i-1,w-W[i]]+V[i];
5. 若pt > g[i,w], g[i,w]=pt
6. 输出g[n,C] //时间O(nC). 输入规模?

输入规模: max{ n, $\log_2 C$ }

1. 初始g[0:C]=0
2. 对 i = 1:n, 对 w = C:1,
3. 若w≥W[i],
4. pt=g[w-W[i]]+V[i];
5. 若pt>g[w], g[w]=pt;
6. 输出 g[C]

最优装载([王]p95)

0-1背包问题

- 输入: n物品重W[1:n],
  价值V[1:n],
  背包容量C
- 物品重量为整数.
- 输出: 装包使得价值最大.
- 每件物品只能取或不取

$$\max \sum_{i=1}^{n} V_i x_i$$

$$\sum_{i=1}^{n} W_i x_i \le C$$

$$x_i \in \{0,1\}, 1 \le i \le n$$

最优装载

- 输入: n物品重W[1:n],
  背包容量C
- 输出: 装包使得件数最多.
- 每件物品只能取或不取

$$\max \sum_{i=1}^{n} x_i$$

$$\sum_{i=1}^{n} W_i x_i \le C$$

$$x_i \in \{0,1\}, 1 \le i \le n$$

最优装载

最优装载

- 输入: n物品重W[1:n],
  背包容量C
- 输出: 装包使得件数最多.
- 每件物品只能取或不取

$$\max \sum_{i=1}^{n} x_i$$

$$\sum_{i=1}^{n} W_i x_i \le C$$

$$x_i \in \{0,1\}, 1 \le i \le n$$

贪心算法(O(nlogn)):

1. 对重量 W[1:n] 升序排列

2. 优先添加重量小的物品, 直到不能再添加

贪心选择性质: 最优解可以包含重量最小的

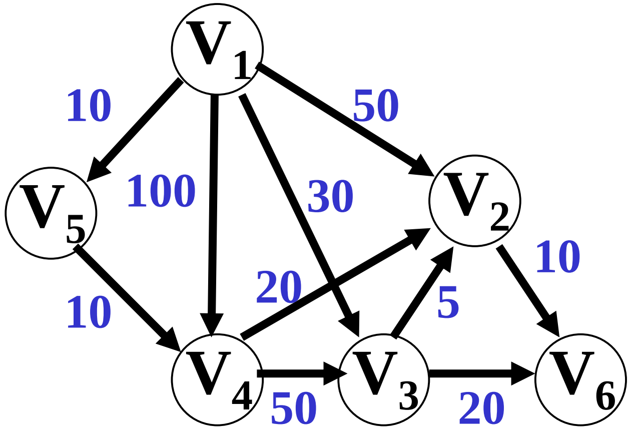
OSP: 最优解([1:n],C)去掉重量最小([2:n],C-W[1])仍是最优解

第三章 动态规划
dynamic programming

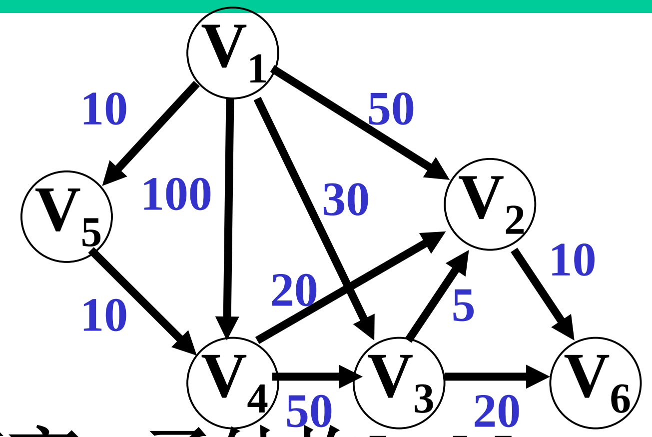
1. 动态规划一般原理(与分治对比), **Bellman, OSP**
2. 动态规划设计步骤: 矩阵连乘
3. 如何设计动态规划算法: 子结构和策略
   最长公共子序列, 最大子段和, 最长递增子序列
4. 背包问题 (动态规划与贪心对比)
5. 最短路问题 (全路径, **Bellman-Ford, Dijkstra**)

最短路问题

- 输入: 带权有向图或无向图 $G = (V,E,w)$, 权代表距离
- 输出: 节点之间的最短距离(路径)
- 例: 找下图中V_1到V_2的最短距离, 最短路径. OSP?
- OSP: 一条最短路径, 其任意子段都是最短路径
- 全点对(all-pair)最短路,Floyd-Warshall,$O(|V|^3)$,
- 单源(single source)最短路, Dijkstra, $O(|E|\log|V|)$,
- 带负权, Bellman-Ford, $O(|V||E|)$
- $\delta(u,v)$: u,v之间最短路径的距离
- 三角不等式: $\delta(u,v) \le \delta(u,z) + \delta(z,v)$
- OSP对应 动态规划 或 贪心 ?



全点对最短路(APSP): DP



- 输入: G=(V,E,w), w权(非负);
- 输出: 所有点对间最短距离
- 节点集V[1:n]. 子结构? 决策量?
- 决策量D[i,j]:顶点i与j之间的最短距离? 子结构[1:k]?
- D[i,j][k]: 从V[i]到V[j],中间只经过V[1:k]的最短距离

D[1,6][1]? D[1,6][2]? D[1,6][3]? 怎么想到? 如何递归?

- D[i,j][k]　或者 不经过k,　或者 经过k(仅1次)

D[i,j][k] = min{ D[i,j][k-1], D[i,k][k-1]+D[k,j][k-1] }

$$D[i,j][k]=\begin{cases} w[i,j] & k=0 \\ \min\{D[i,j][k-1], D[i,k][k-1]+D[k,j][k-1]\} & k>0 \end{cases}$$

APSP: Floyd-Warshall算法

$$D[i,j][k]=\begin{cases} w[i,j] & k=0 \\ \min\{D[i,j][k-1], D[i,k][k-1]+D[k,j][k-1]\} & k>0 \end{cases}$$

1. D[i,j][0] = w[i,j], 不存在的边值取无穷大
2. 对k=1:n
3.   对i=1:n, 对j=1:n
4.   若D[i,k][k-1]+D[k,j][k-1] < D[i,j][k-1]
6.   则 D[i,j][k] =D[i,k][k-1]+D[k,j][k-1];
7.   否则 D[i,j][k] = D[i,j][k-1]

如何减少存储?

注: D[i,k][k] = D[i,k][k-1]?

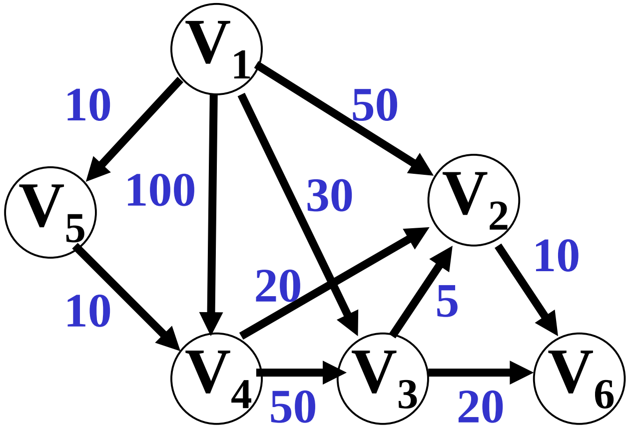
任何最短路径至多经过k一次

1. D[i,j] = w[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]
6.   则 D[i,j] = D[i,k]+D[k,j];
7.   p[i,j] = p[k,j]; //解标记

原因. 构造解. O(n^3).

p[i,j]: 在i到j的最短路上j的前驱

函数d和松弛操作



- 输入: G=(V,E,w,s), w权, s起点
- 输出: s到其它点最短距离(路径)
- δ(u,v): u,v之间最短距离
- 初始d[s]=0; 其它u, d[u]=∞.
- 松弛操作relax(u,v): //其中(u,v)是图G的一条边

  若d[v]>d[u]+w(u,v), 则d[v]=d[u]+w(u,v); p[v]=u;

  作用:减小d[·]. 松弛后 d[v] ≤ d[u]+w(u,v)
- 设计一列松弛操作. (1,2) (1,5) (1,3) (3,6) (1,4) (3,2) (2,6)
- d始终是δ(s,·)的上界. (初始? 松弛(u,v)后?)

  $\delta(s,v) \le \delta(s,u) + w(u,v) \le d[u] + w(u,v) = d[v]$
- 若一列松弛中有子列沿s-u最短路, 则d[u]=δ(s,u). (?)

Bellman-Ford处理有负权情况

- 输入: G=(V,E,w,s), w权, s起点; 输出: δ(s,·)
- 通过松弛减小d[·]; d始终是δ(s,·)的上界.
- 若一列松弛中有子列沿s-u最短路, 则d[u]=δ(s,u).
- 若没有负回路, 则存在最短路.

初始d[s]=0, 其它点d[u]=INF,

1. 对i =1: |V|-1

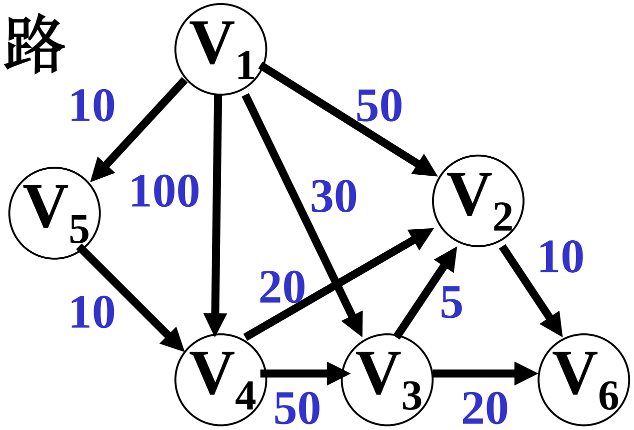
2.   对每条边(u,v), 松弛(u,v)   //包含所有松弛子列

3. 对每条边(u,v),   //检查负回路

4.   若d[v]>d[u]+w(u,v), 则返回假 //?

5. 返回真

O(|V| |E|) 动态规划? 贪心?



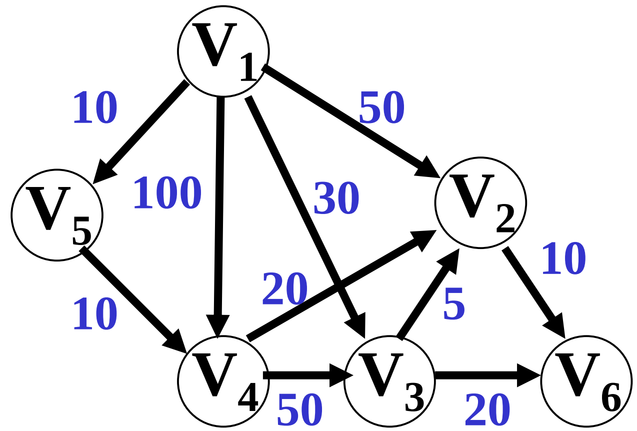
Dijkstra处理无负权情况

1. d[s]=0, d[u]=INF, S空, Q=V
2. 当Q非空
3. 取Q中d[u]最小u, 加入S
4. ∀v∈N(u), 松弛(u,v). //u的邻居

- Q用数组: O(|V|²+ |E|);
- Q用最小堆:

1. d[s]=0, d[u]=INF, S=空.
2. 对所有v∈V按d[v]值建堆Q.
3. 当Q非空, 取Q堆顶u,
4. 若u∈S, 则break; 否则加入S.
5. ∀v∈N(u), 松弛(u,v),
6. 若d[v]更新, 则加入Q

O(|E|log|E|) = O(|E|log|V|)



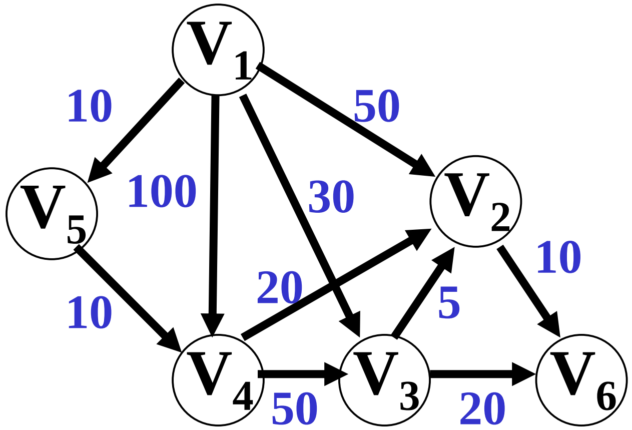
父	v_1	v_2	v_3	v_4	v_5	v_6
	0	∞	∞	∞	∞	∞
v_1		50	30	100	10	
v_5				20		
v_4		40	30			
v_3		35				50
v_2						45

Dijkstra处理无负权情况([王]p100)

- 输入: G=(V,E,w,s), w权, s起点; 输出: δ(s,·)
- 通过松弛减小d[·]; d始终是δ(s,·)的上界.
- 若一列松弛中有子列沿s-u最短路, 则d[u]=δ(s,u).

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

记u的邻居为N(u): ∀v∈N(u)



- 贪心选择性质:

  从Q中取出u时, d[u] = δ(s,u).

  直观解释: 没有负权边, 后面的松弛不会再改变d[u].

Dijkstra贪心选择性质正确性证明

性质: 从Q中取u放入S时, $d[u]=\delta(s,u)$.

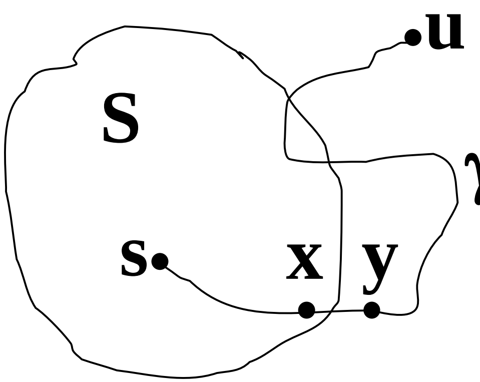
归纳基础: 初始$S=\{s\}$, $d[s]=\delta(s,s)=0$.

归纳假设: 在某一步, $\forall v\in S$, $d[v]=\delta(s,v)$.

归纳证明: 从Q中取出u (Q中d[u]最小)时$d[u]=\delta(s,u)$

反证法: 假设$d[u] > \delta(s,u)$

设γ是s到u的最短路径



$$d[y] \le_1 d[x] + w[x,y] =_2 \delta(s,x) + w[x,y]$$

$$=_3 \delta(s,y) \le_4 \delta(s,u) <_5 d[u] \le_6 d[y].$$ 矛盾

1,2: $x\in S$. 3,4: γ是最短路径. 5: 假设. 6: Q中d[u]最小

注: [王]中证明逻辑不容易理清

编程题Layout

- 牛n头排成直线, 坐标 $x_1 \le x_2 \le \ldots \le x_n$,
- 有l对牛互相喜欢, d对牛互相讨厌
- 互相喜欢的牛 $i \le j$, 必须 $x_j - x_i \le l_{ij}$,
- 互相讨厌的牛 $s \le t$, 必须 $x_t - x_s \ge d_{st}$.
- 求x_1和x_n最大距离.
- 输入:

第1行: n, l, d,

l行: (i,j,l_{ij}) 牛i,j的互相喜欢, 距离上界l_{ij},

d行: (s,t,d_{st}) 牛s,t互相讨厌, 距离下界d_{st},

- 输出: -1(若不存在排队); -2(若可以无限大); 最大距离

例1. $d_{12} = 10$. 例2. $d_{23}=100$, $l_{14} = 10$. 例3. $d_{23}=10$, $l_{14} = 20$.

本章小结

最优子结构性质OSP

动态规划算法的设计步骤

矩阵连乘

最长公共子序列

最大子段和

最长递增子序列

0-1背包, 分数背包, 最优装载

全路径最短路 单源最短路   带负权最短路

习题

本章作业

算法分析题

3. 考虑下面的整数线性规划问题 .

即给定 序列$a_1, a_2, \ldots, a_n$, 求

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

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

算法实现题:

石子合并问题, 数字三角形问题, 游艇出租问题

本章作业

算法实现题:

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≤n≤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	输出文件示例
input.txt	**output.txt**
4	**43**
4 4 5 9	**54**

本章作业

算法实现题:

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
```

本章作业

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

附录

注记1: 最优子结构性质

矩阵连乘, 最长公共子序列, 最大子段和

全路径最短路 单源最短路   带负权最短路

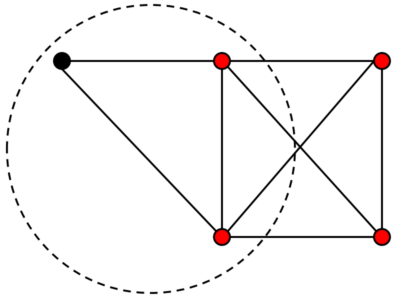
最长递增子序列:

输入: (2,8,9,4,6,1,3,7,5,10) 输出: (2,4,6,7,10)

前4个最优子结构MTIS(1):2, MTIS(2):24, MTIS(3):289

0-1背包, 分数背包, 最优装载

最大团(完全子图)问题不满足OSP



注记2

动态规划 R. Bellman 1955

矩阵连乘 $O(n^3)$, 现O(n log n)

最长公共子序列O(mn), Knuth问是否最优

最大子段和 [王, M]

最长递增子序列 [王, M]Gries 1981

0-1背包, NP完全

全路径最短路 $O(n^3)$ …/Floyd/Warshall1962, 有改进

单源最短路 Dijkstra1959无优先队列, 有改进

带负权最短路Bellman1958, Ford1959, 有改进

(分数)背包问题(knapsack Prob)

0-1背包问题

- 输入: n物品重W[1:n],
  价值V[1:n],
  背包容量C
- 物品重量为整数.
- 输出: 装包使得价值最大.
- 每件物品只能取或不取

$$\max \sum_{i=1}^{n} V_i x_i$$

$$\sum_{i=1}^{n} W_i x_i \le C$$

$$x_i \in \{0,1\}, 1 \le i \le n$$

(分数)背包问题

- 输入:物品重W[1:n],价值V[1:n]
- 输出: 装包使得价值最大.
- 物品i可以取重量0~W[i]

$$\max \sum_{i=1}^{n} V_i x_i$$

$$\sum_{i=1}^{n} W_i x_i \le C$$

$$0 \le x_i \le 1, 1 \le i \le n$$

(分数)背包问题

(分数)背包问题

- 输入:物品重W[1:n],价值V[1:n]
- 输出: 装包使得价值最大.
- 物品i可以取重量0~W[i]

$$\max \sum_{i=1}^{n} V_i x_i$$

$$\sum_{i=1}^{n} W_i x_i \le C$$

$$0 \le x_i \le 1, 1 \le i \le n$$

贪心算法:

1. 对单位重量价值 $\{ V[i]/W[i] \}_{i=1}^{n}$ 排序

2. 优先放入单位重量价值大的物品, 直到填满

O(nlogn),

0-1背包能贪心吗?

0-1背包: W={1,2,3}, V={6,10,12}, C=5

递推关系对比:如何节省空间

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

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

LCSlength(n,m,x,y,c)

1. 初值 c[0:1][0:m]=0

2. 对i=1:n, j=1:m

3. 若x[i]==y[j], 则 c[i%2][j]=c[(i-1)%2][j-1]+1

4. 否则 若c[i%2][j-1]>c[(i-1)%2][j], 则 c[i%2][j]=c[i%2][j-1]

5. 否则 c[i%2][j]=c[(i-1)%2][j]

输出c[n%2][m]. //能否进一步减少存储空间? O(min{m,n})

最大子段和(ms)-分治

- 输入:S=($x_1,x_2,\ldots,x_n$)　输出:最大和子段 ($x_i,x_{i+1},\ldots,x_j$)

1. 分成两段S_L和S_R.
2. 递归求两段的ms: M_L, M_R.
3. 求跨两段的ms与M_L, M_R的最大.

合并法一: 在S_L中遍历尾和在S_R中遍历头

$f(n) = \Theta(n^2)$,

$T(n) = \Theta(n^2)$.

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

合并法二: S_L的最大尾ms + S_R的最大头ms

$f(n) = \Theta(n)$,

$T(n) = \Theta(n\log n)$.

跨界序列和的最优子结构(OSP)

拆分方案数1

- 输入: 正整数集A[1:k], 正整数n
- 输出: 将n拆分为A中不同数的和的方案数
- 使用什么子结构, 决策(量)?
- [1:i], f[i,s] = s用A[1:i]中不同数拆分的方案数
- 输出什么?
- f[k,n]. f[i,s] = ? 递推关系

f(i,k)	A[i]	k=0	1	2	3	4	5	6	7	8	9	10	11
f(1,k)	1	1	1	0	0	0	0	0	0	0	0	0	0
f(2,k)	2	1	1	1	1	0	0	0	0	0	0	0	0
f(3,k)	3	1	1	1	2	1	1	1	0	0	0	0	0
f(4,k)	4	1	1	1	2	2	2	2	2	1	1	1	0

拆分方案数1

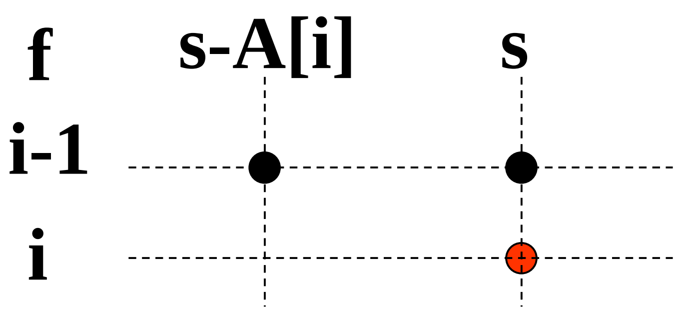
- 输入: 正整数集A[1:k], 正整数n
- 输出: 将n拆分为A中不同数的和的方案数
- [1:i], f[i,s] = s用A[1:i]中不同数拆分的方案数
- 输出f[k,n]. f[i,s] = ?
- 添加A[i]会产生什么变化?
- 添加A[i]后, 拆分中有含和不含A[i]两种情况
- f[i , s] = f[i-1 , s] + f[i-1 , s - A[i]]
- 为计算 f[k,n], 需要计算f[i,s], 0≤i≤k, 0≤s≤n

$$
f[i,s]=\begin{cases} 1 & i=0, s=0 \\ 0 & (i=0, s>0)\text{或}(s<0) \\ f[i-1,s]+f[i-1,s-A[i]] & i>0 \end{cases}
$$

拆分方案数1示例

$$f[i,s]=\begin{cases} 1 & i=0, s=0 \\ 0 & (i=0, s>0)\text{或}(s<0) \\ f[i-1,s]+f[i-1,s-A[i]] & i>0 \end{cases}$$

f(i,k)	A[i]	k=0	1	2	3	4	5	6	7	8	9	10	11
f(1,k)	1	1	1	0	0	0	0	0	0	0	0	0	0
f(2,k)	2	1	1	1	1	0	0	0	0	0	0	0	0
f(3,k)	3	1	1	1	2	1	1	1	0	0	0	0	0
f(4,k)	4	1	1	1	2	2	2	2	2	1	1	1	0



拆分方案数1算法

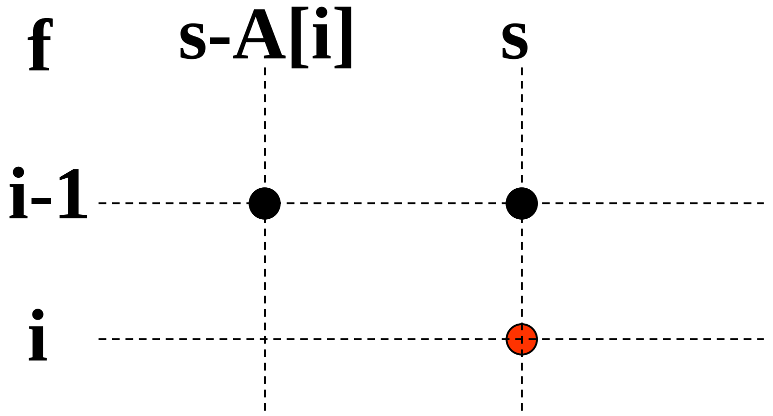
$$f[i,s]=\begin{cases} 1 & i=0, s=0 \\ 0 & (i=0, s>0)\text{或}(s<0) \\ f[i-1,s]+f[i-1,s-A[i]] & i>0 \end{cases}$$

1. 初始 f[0,1:n] = 0, f[0,0] = 1
2. 对 i = 1:k,
3. 对 s = 0:n,
4. f[i,s] = f[i-1,s];
5. 若s ≥ A[i], f[i,s] += f[i-1,s-A[i]]
6. 输出f[k,n]

时间O(kn)　对比输入规模

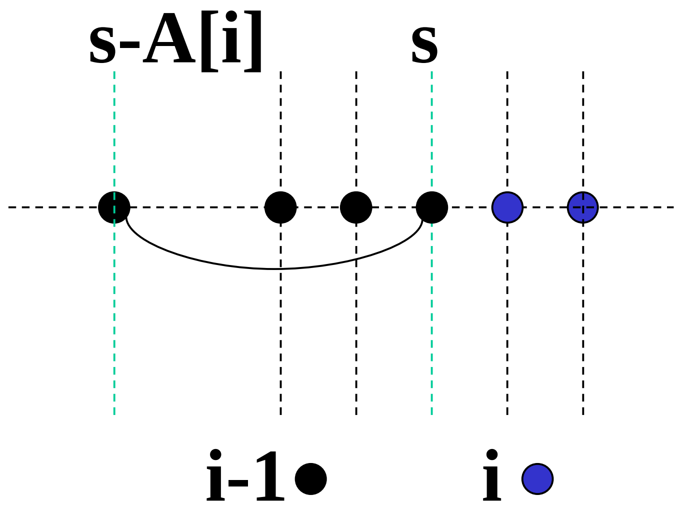
拆分方案数1二维数组改一维

1. 初始 f[0,1:n] = 0, f[0,0] = 1
2. 对 i = 1:k, 对 s = 0:n,
3.   f[i,s] = f[i-1,s];
4.   若s ≥ A[i], f[i,s] += f[i-1,s-A[i]]
5. 输出f[k,n]



1. 初始 f[1:n] = 0, f[0] = 1
2. 对 i = 1:k,
3.   对 s = n:1,
4.   若s ≥ A[i], f[s] += f[s-A[i]]
5. 输出f[n]

思考: 第3行改为 s = 1:n会如何?



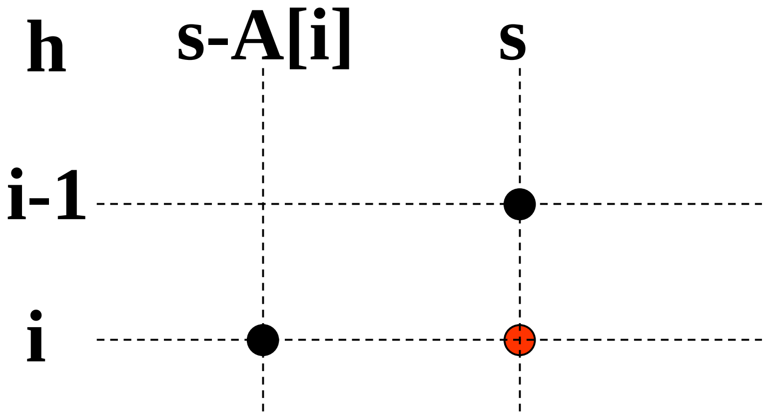
拆分方案数2

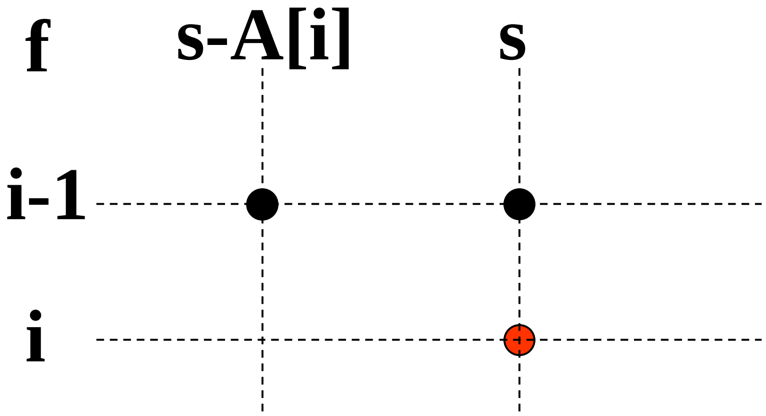
- 输入: 正整数集A[1:k], 正整数n
- 输出: 将n拆分为A中数(可用多次)的和的方案数
- 子结构? 决策量?
- [1:i], h[i,s] = s用A[1:i]中数拆分的方案数
- h[i,s] = h[i-1,s] + h[i-1,s-A[i]] + h[i-1,s-2A[i]] + …
- = h[i-1,s] + h[i,s-A[i]]

1. 初始h[1:k,0]=0, h[0,0]=1
2. 对i=1:k,
3. 对s=0:n,
4. h[i,s]=h[i-1,s];
5. 若s≥A[i], h[i,s]+=h[i,s-A[i]]
6. 输出h[k,n]

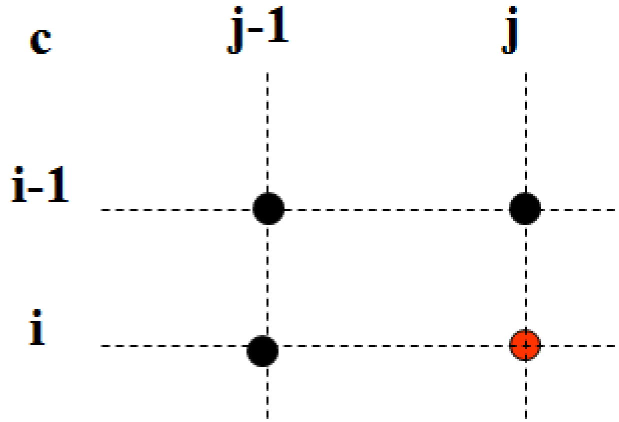
三种计算关系的比较

- h[i,s]=h[i-1,s]+h[i, s-A[i]]
- f[i,s]=f[i-1,s]+f[i-1, s-A[i]]





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



对比拆分1和拆分2的二维改一维

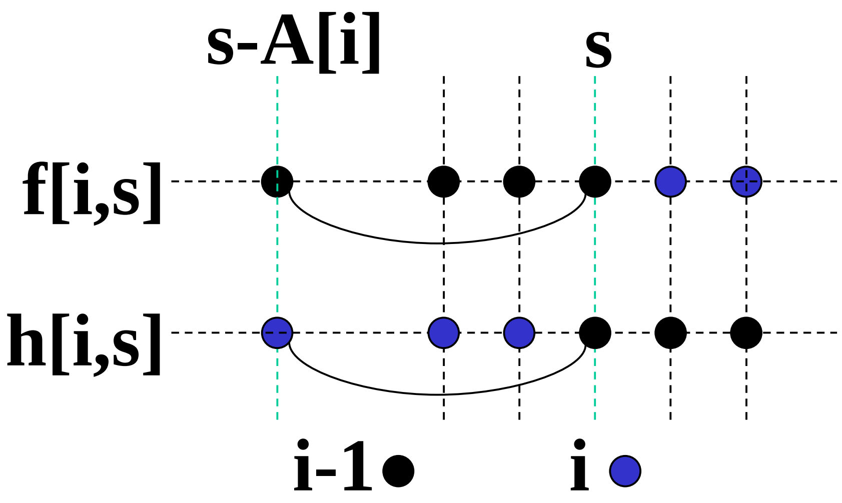
- f[i,s]=f[i-1,s]+f[i-1, s-A[i]]
- h[i,s] = h[i-1,s] + h[i, s-A[i]]

拆分1--每数至多用1

1. 初始 f[1:n] = 0, f[0] = 1
2. 对 i = 1:k,
3. 对 s = n:1,
4. 若s ≥ A[i], f[s] += f[s-A[i]]
5. 输出f[n]

拆分2--每数可用0至多次

1. 初始h[1:n]=0, h[0]=1
2. 对 i = 1:k,
3. 对 s = 1:n,
4. 若s≥A[i], h[s]+=h[s-A[i]]
5. 输出h[n]



Cable master

输入n(=4)根线缆, 各根长分别为cable[1:n]

割出等长k(=11)根, 求最大长度.

使用二分搜索

1. 读入n, k, cable[1:n], cable[i]*=100
2. L=0, R=max(cable[1:n])
3. 当R-L>1,
4. cnt=0, mid=(L+R)/2,
5. 对i=1:n, cnt+=(int)cable[i]/mid,
6. 若cnt≥k, 则L=mid, 否则 R=mid
7. L+=0.1, 输出L/100(保留两位小数)

4 11
8.02
7.43
4.57
5.39

油井

输入n个油井的坐标，求由东向西的主管道纵坐标

使得各油井向主管道输油管长度和最小

若n为奇数, 则取第(n+1)/2小的纵坐标最优

若n为偶数, 则取第n/2小第n/2+1小纵坐标之间都是最优

依题意, 取第n/2小的纵坐标

两种情况下答案都是第[(n+1)/2]小的纵坐标

使用线性时间选择算法.