第15章动态规划

引论

- 动态规划法在本课程介绍的算法设计方法中是最难的
- 应用：

（1）0/1背包问题

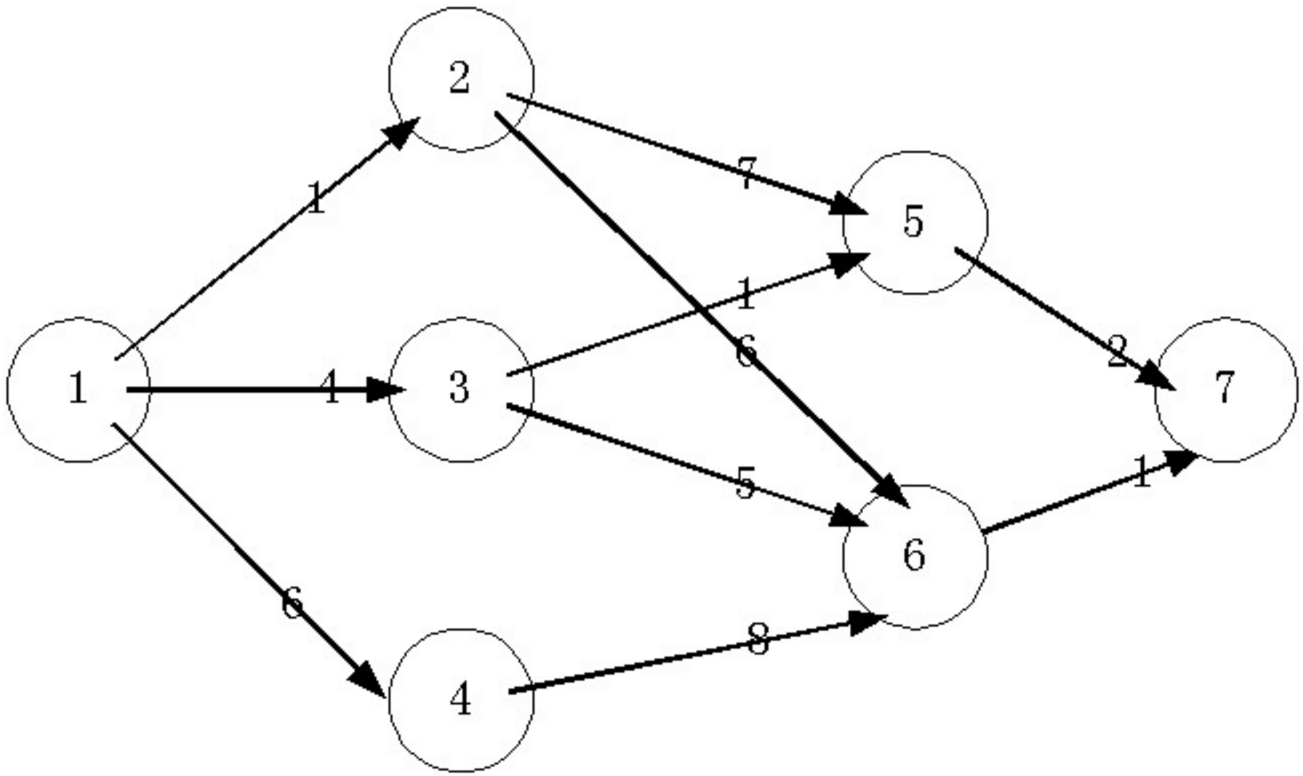
（2）矩阵乘法链

（4）最短路径

15.1动态规划原理

- 从算法设计的角度看，动态规划是一种在各个不同大小(size)的子问题的优化值之间建立递归关系并求解的过程.
- 能用动态规划求解的问题必须满足优化原理:优化解包含的子问题的解也是优化的.
- 利用优化原理,使用枚举法建立不同长度子问题的优化值之间的递归关系—动态规划方程.
- 动态规划得到的是精确解.
- 子问题的数目决定算法的复杂性.
- 实现时要尽可能消去递归.

例15-1 [多段图]



例15-1 [最短路径]（结论）

- 多段图问题满足优化原理:

  最短路(1->3->5->7)上的子路径(3->5->7)

  是3到目的节点7在子图上的最短路.

- 无论最短路的下一跳是{2, 3, 4}中的那个节点, 其后的路径也应是最短路.
- 节点1到目的节点的最短路长度c(1)可从2, 3, 4到目的节点的最短路长度c(i)＋节点1到这些节点的边成本cost(1, i)经枚举得到: $c(1)=\min_{i \in \{2,3,4\}}\{c(i)+cost(1,i)\}$

多段图的动态规划算法

- 但2,3,4到目的节点的最短路长度c(2), c(3), c(4) 还不知道!
- 我们须计算c(2),c(3),c(4);仍使用优化原理.
- 一般情形:

  设c(i)为i到目的节点的最短路长度, A(i)为与i相邻的结点集合,有:

  $c(i)=\min_{j\varepsilon A(i)}\{c(j)+cost(i, j)\}$
- 但c(i)由i 到目的节点的子图来决定,和节点1怎样走到i 没关系(Markov 性质).
- 我们有c(7)=0

从c(7)开始向前计算

- 初始c(7)=0
- 依次计算c(6),…,c(1):
- C(6)=1,c(5)=2,
- c(4)=8+c(6)
- C(3)=min{1+c(5),5+c(6)}
- C(2)=min{7+c(5),6+c(6)}
- C(1)=min{1+c(2),4+c(3),6+c(4)}
- 递归还可从前向后：c(i)=节点1到节点i的最短路的长度；递归从c(1)=0开始。

例15-2 [0/1背包问题]

- 0/1背包问题的解指物品1,…,n的一种放法(x_1, ⋯,x_n的0/1赋值), 使得效益值最大.
- 假定背包容量不足以装入所有物品:面临选择.
- 因为目标函数是非负数之和=>
- 优化原理:无论优化解是否放物品1,相对剩余背包容量,优化解对物品2,…,n的放法,也是优化解.

背包问题满足的优化原理

- 例如 $n=5,c=10,w=[2,2,6,5,4],p=[6,3,5,4,6]$.
- 其优化解为(1,1,0,0,1),即优化的物品装入背包的方法为 1,2,5.
- 物品1占背包容量2,剩下容量为8.
- 优化解中包含的子问题:$n=4,c'=c-2$(物品1的重量),物品为2,3,4,5
- (1,0,0,1),即放物品2和5,是上述子问题的优化解.
- 背包问题满足的优化原理.

优化值间的递归式

- 虽然我们不知道优化解是否放物品1,但我们可以利用优化原理,从枚举“放”和“不放”两种情形建立优化值之间的递归式:
- 设f(i, y)为以背包容量y,放物品i,…,n,得到的优化效益值,以下递归关系成立:
  - $f(1,c)=\max\{f(2,c), f(2,c-w_1)+p_1\}$
  - 先求子问题的优化值(递归),再从2种可能性中选出最优的.
- 须求解:任意给定容量y, 任意i,…,n 种物品的子问题.

例15-2 [0/1背包问题]（解）

- $n=3$, $w=[100,14,10]$, $p=[20,18,15]$, $c=116$
  - 放进物品1($x_1 = 1$),背包容量还剩r=16.
    $[x_2,x_3]= [1,0]$ 为子问题的优化解,值为18.
  - 不放物品1($x_1= 0$)则对于剩下的两种物品而言,容量限制条件为116，[1,1]为子问题优化解,值为33
- 前者效益值为38, 后者为33; 所以优化解为[1,1,0], 优化值为38.

例15-4 [0/1背包]

- 令f(i,y) 表示容量为y,物品i,i+1,…,n 的优化效益值,按优化原理可列递归关系如下：

$$f(i,y)=\begin{cases}\max\{f(i+1,y),f(i+1,y-w_i)+p_i\} & y\geqslant w_i\\ f(i+1,y) & 0\leqslant y<w_i\end{cases}\qquad(15\text{-}1)$$

- 第2行表示背包容量y不足以放下物品i.

$$f(n,y)=\begin{cases}p_n & y\geqslant w_n\\ 0 & 0\leqslant y<w_n\end{cases}\qquad(15\text{-}2)$$

例15-4 0/1背包的DP算法

- 问题要求计算f(1,c), 所以计算过程中不必计算

  f(i, y), y>c

- 计算从f(n, *)开始

$$f(n,y)=\begin{cases} p_n & y \geqslant w_n \\ 0 & 0 \leqslant y < w_n \end{cases}$$

- 然后应用(15-1)式递归计算

  f(i, y)　　　i=n-1,n-2，…,2，

- 最按 $f(1,c)=\max\{f(2,c), f(2,c-w_1)+p_1\}$计算f(1,c).
- 例题15.2：n=3, w=[100,14,10], p=[20,18,15], c=116. 求解如下：

$$f(3,y)=\begin{cases} 0, 0 \le y < 10 \\ 15, y \ge 10 \end{cases}$$

例15-2 [0/1背包]

利用递归式（15-1）,可得

$$f(2,y)=\begin{cases}0, 0\le y<10\\ 15, 10\le y<14\\ 18, 14\le y<24\\ 33, y\ge 24\end{cases}$$

- 因此最优解f(1,116)

$=\max\{f(2,116),f(2,116-w_1)+p_1\}$

$=\max\{f(2,116),f(2,16)+20\}$

$=\max\{33,38\}=38$

例15-2 [0/1背包]

- 使用traceback方法从优化值得到优化解:
- 现在计算x_1,---,x_n值, traceback如下:
  - $f(1,c) = f(2,c) => x_1=0$；否则$x_1=1$.
  - 设y为确定了$x_1,\ldots,x_{i-1}$后背包的剩余容量,确定x_i如下:

    如果$w_i>y$, 则$x_i=0$; 如果$w_i \le y$,

    $f(i,y)=f(i+1,y) => x_i=0$,否则$x_i=1$.
  - 依次类推.
- 该例中,$f(2,116)=33 \ne f(1,116)$,所以$x_1=1$.
- 剩余容量=116-100=16,
  $f(2,16)=18 \ne f(3,16)=14$ 因此$x_2=1$,
- 此时r=16-14=2,不足以放物品3,所以x_3= 0。

例题15.21旅行商问题

- 求图G=(V, E)的最小成本周游路线(见208页)
- 动态规划解:令S为V的不含节点1的子集, 设g(i,S)表示从节点i 出发,经过S中的所有节点各一次,到达节点1 的最短路长度(子问题的优化值),根据优化原理有以下递归式

$$\boxed{g(i,S)=\min_{j\in S}\{c_{i,j}+g(j,S-\{j\})\}}$$

- 原问题为g(1,V-{1}),
- 初始: $g(i, \varnothing)=c_{i,1}$;
- 从S=Ø开始，依次对|S|=1,2,---, n−1, 计算

- 考虑有如下邻接矩阵的图:

$$\begin{pmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{pmatrix}$$

- 计算过程如下: $g(2,\Phi)=c_{21}=5,\ldots$

  $g(2,\{3\})=15$, $g(2,\{4\})=18$, $g(3,\{2\})=18,\ldots$

  $g(2,\{3,4\})=\min\{c_{23}+g(3,\{4\}),c_{24}+g(4,\{3\})$
  $=25$

  $g(1,\{2,3,4\})=\min\{c_{12}+g(2,\{3,4\}),c_{13}+g(3,\{2,4\}),$
  $c_{14}+g(4,\{2,3\})\}$

例题15.21货郎担问题

- 算法的时间复杂度为:
- |S|=k 的子问题个数为 C_{n-2}^{k}
- 在|S|=k时，求最小值需做k-1次比较
- 算法的时间复杂度(比较次数)为

$$\boxed{\sum_{k=1}^{n-2}(n-1)(k-1)C_{n-2}^{k}=(n-1)\sum_{k=1}^{n-2}(k-1)C_{n-2}^{k}=\Theta(n^2 2^n)}$$

动态规划法步骤:

- 在应用动态规划法时,须先验证欲求解的问题是否满足优化原理.
- 应用优化原理建立子问题优化解的值(优化值)之间的递归式.
- 解优化值满足的递归式.
- 回溯(traceback)从优化值构造优化解.
- 在例题1中求优化值的过程中需记录下达到最小值的邻接节点编号, 以便进行traceback.

算法复杂性

- 直接用递归实现动态规划递归方程往往会引发大量重复计算,算法的计算量变得非常可观.最好使用迭代法实现动态规划算法;
- 迭代实现需要存贮所有子问题的优化解的值f(i,y), 以便避免重复计算,所以算法往往需要较大的存储空间.
- 算法的复杂性来自子问题的数目,通常子问题的数目很大.

15.2.1　0/1背包问题DP算法的实现

- 1. 递归实现
- 2. 权为整数的迭代实现
- 3. 元组法实现

1. 递归实现

- 前面已建立了背包问题的动态规划递归方程，程序15-1是该递归方程的直接实现.

$$f(n,y)=\begin{cases} p_n & y \geqslant w_n \\ 0 & 0 \leqslant y < w_n \end{cases}$$

$$f(i,y)=\begin{cases} \max\{f(i+1,y), f(i+1,y-w_i)+p_i\} & y \geqslant w_i \\ f(i+1,y) & 0 \leqslant y < w_i \end{cases}$$

程序15-1 背包问题的递归实现

```
int F(int i, int y)
{// 返回 f(i,y).
   if (i == n) return (y < w[n]) ? 0 : p[n];
   if (y < w[i]) return F(i+1,y);
   return max(F(i+1,y), F(i+1,y-w[i]) + p[i]);
}
```

- 执行调用F(1,c) 返回f(1,*c*)值.
- 程序15-1的最坏时间复杂性t(n):
- t(1)=a；t(n)=2t(n-1)+b（n>1),

其中a,b为常数,求解可得$t(n)=\Theta(2^n)$

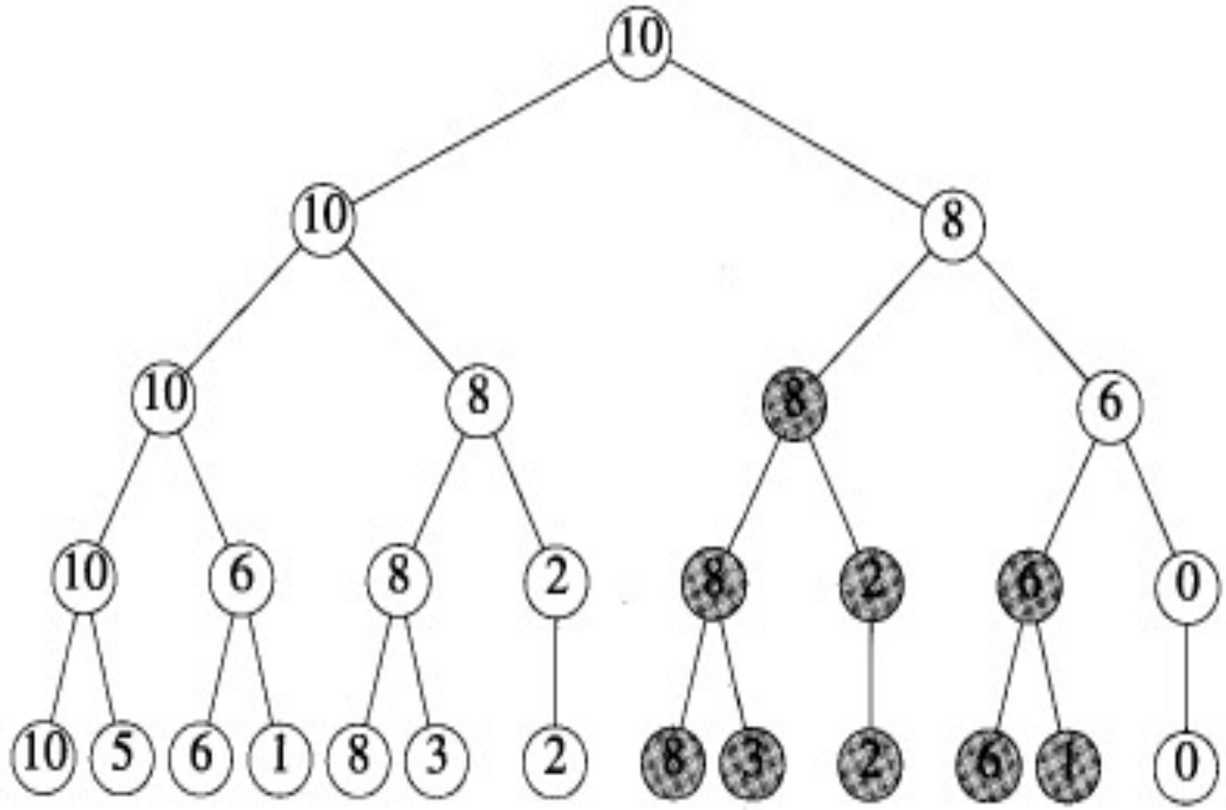
例15-5

- 设n=5,p=[6,3,5,4,6],w=[2,2,6,5,4] 且 c=10 ,求f (1,10).
- 为了确定f(1,10),调用函数F(1,10).
- 递归调用关系如图15-1的树型结构所示:

  其中每个节点用y值来标记;第j层的节点对应F(j,*);因此根节点表示F(1,10),而它有左、右子节点,分别对应F(2,10)和F(2,8).

图15-1 递归调用树

- f(1,10)
- f(2,*)
- f(3,*)



- 节点内标出的数是背包剩余容量y; 未标出的分枝是剩余容量不足以放任何物品.

例15-5（续）

- 从递归调用树可以看到程序15-1总共执行了28次递归调用.
- 我们注意到,其中重复计算的节点,如f(3,8)计算过两次，相同情况的还有

  f(4,8), f(4,6), f(4,2), f(5,8), f(5,6), f(5,3), f(5,2) 和f(5,1).
- 如果保留以前的计算结果,则可将节点数减至19(省略图中的阴影节点的计算).

2. W取整数时的迭代实现

- 当物品重量为整数时,可设计一相当简单的算法(见程序15-2)来求解f(1,*c*).
- 该实现用二维数组f [i][y]来保存每个f(i,y)的值,并且只计算一次.
- 二维数组需Θ(nc)空间.
- 函数Traceback从f[i][y]产生优化的x_i值.
- Knapsack的复杂性Θ(nc),似乎是多项式算法.但因c的二进制输入长度为$\log_2 c$, 所以nc仍是输入长度的指数函数.
- Traceback的复杂性为Θ(n).

程序15-2 f 和x 的迭代计算

```
template<class T>
void Knapsack(T p[], int w[], int c, int n, T** f)
{// 对于所有i和y计算f[i][y]

  // 初始化 f[n][]
  for (int y = 0; y <= yMax; y++)
     f[n][y] = 0;
  for (int y = w[n]; y <= c; y++)
     f[n][y] = p[n];
```

程序15-2 f 和x 的迭代计算(续1)

```
// 计算剩下的f
for (int i = n - 1; i > 1; i--) {
   for (int y = 0; y <= yMax; y++)
      f[i][y] = f[i+1][y];
   for (int y = w[i]; y <= c; y++)
      f[i][y] = max(f[i+1][y], f[i+1][y-w[i]] + p[i]);
   }
 f[1][c] = f[2][c];
 if (c >= w[1])
   f[1][c] = max(f[1][c], f[2][c-w[1]] + p[1]);
}
```

- 注: $yMax=w_i$

程序15-2 f 和x 的迭代计算(续2)

```
template<class T>
void Traceback(T **f, int w[], int c, int n, int x[])
{// 计算x
    for (int i = 1; i < n; i++)
      if (f[i][c] == f[i+1][c]) x[i] = 0;
      else {x[i] = 1;
            c -= w[i];}
    x[n] = (f[n][c]) ? 1 : 0;
}
```

例题5.6

- n=5,p=[6,3,5,4,6],w=[2,2,6,5,4] 且 c=10 ,求f (1,10).

	y										
i	0	1	2	3	4	5	6	7	8	9	10
5	0	0	0	0	6	6	6	6	6	6	6
4	0	0	0	0	6	6	6	6	6	10	10
3	0	0	0	0	6	6	6	6	6	10	11
2	0	0	3	3	6	6	9	9	9	10	11

Figure 15.2 *f* function for Example15.6

3. 元组法实现

- 程序15-2有两个缺点:
  - 1)要求物品重量为整数;
  - 2)当背包容量c 很大时,例如$c>2^n$,程序15-2的要求的存储空间为$\Omega(n2^n)$.下述元组法克服了上述缺点.
- 元组法将函数f(i, y)的跳跃点以元组(y, f(i, y))形式存储于一个线性表P(i)中.
- 表P(i)中的元组(y,f(i, y)) 按y的增序排列.
- P(i)中的元组(a,b)表示:存在一种装物品{i,---,n}的方案,能以容量y, $a\le y<a'$, a'为下一元组的横坐标,得到效益值b.
- 下面给出从f(i+1,y)的线性表P(i+1)得出f(i,y)的线性表P(i)的算法.

元组法

- 按f(i,y)的定义: $f(i, y)=\max\{f(i+1,y),f(i+1,y-w_i)+p_i\}$,首先需要从P(i+1)得到函数$f^*(i+1,y)=f(i+1,y-w_i)+p_i$的元组集合Q.
- 设$(a,b)\in Q$, 则$(a-w_i, b-p_i)$必为P(i+1)的元组,反之亦然. 所以,P(i+1)的每个元组(w,p)对应Q的一个元组$(w+w_i, p+p_i)$.
- Q的元组(u,v)代表装物品{i,---,n}的一种方案:以背包容量u,能得到效益值v.
- 需设计一算法从P(i+1)和Q得到f(i,y)的元组(即P(i)的元组)

元组法

- 从P(i+1)和Q得到P(i)的元组:

  因P(i+1)和Q内元组均按w和p的增序排列,所以可用以前学过的merge算法进行合并.

- 合并时使用以下支配(选优)规则:
  - 设(a,b)和(u,v)是来自P(i+1)和Q的元组,若 $a \geq u$ 且 $b < v$,则称(a,b)受(u,v) 支配. 因为
  - (a,b)代表以容量a得到效益值b的方案,
  - 而(u,v)代表以较少的容量u得到较大效益值v的装包方案.
- 按(15.1),合并时舍弃被支配的元组(选优).

元组法

- P(i+1)与Q合并,并按支配规则舍弃被支配的元组即可得到P(i).
- 在产生P(i)时丢弃w>c的元组(w,v).
- 得到P(2)后不再产生P(1):
  - P(2)的最后一个元组是f(2,c)对应的元组.
  - 设线性表P(2)中满足$w+w_1\le c$的最后一个元组为(w,v).
  - 将$v+p_1$ 与P(2)的最后一个元组对应的效益值p做比较,效益值大的即为优化效益值f(1,c).

例15-6

- 设n=5，p=[6,3,5,4,6]，w=[2,2,6,5,4]且c=10 ,求f(1,10)。
- P(5)=[(0,0), (4,6)]，Q=[(5,4),(9,10)]
- (5,4)代表一种方案,其效益值不如(4,6)代表的方案好,舍弃(5,4),得P(4)
- P(4)=[(0,0),(4,6),(9,10)], 加(6,5)
- Q=[(6,5),(10,11)]合并得P(3),舍弃了(15,15),因为15超过了背包容量.
- 合并P(4)和Q得P(3)

例题15-6(续):设n=5，p=[6,3,5,4,6]，w=[2,2,6,5,4] 且c=10 ,求f(1,10)。

- P(3)=[(0,0),(4,6),(9,10),(10,11)]
- Q=[(2,3),(6,9)]合并舍弃得到
- P(2)=[(0,0),(2,3),(4,6),(6,9),(9,10),(10,11)]
- (2,6)+(6,9)=(8,15)优于(10,11),最优效益值为15.
- 回溯求解为[1,1,0,0,1]

- P(i) 中的元组个数至多为P(i+1)中元组个数的2倍.初始P(n)=2,所以:

  P(i)中的元组个数不超过$2^{(n-i+1)}$

- 计算Q需O(|P(i+1)|)的时间,合并P(i+1)和Q需要O(|P(i＋1)|+|Q|)=O(2|P(i+1)|)的时间.所以计算P(i)需O(2^{n-i+1})时间.
- 计算所有P(i) 时所需要的总时间O(2^n).
- 存在输入使算法最坏情形为2^n量级.

15.2.3 矩阵乘法链

- m×n矩阵A与n×p矩阵B相乘需要做mnp个元素乘法.
- 计算三个矩阵A,B和C的乘积ABC有两种方式:(A*B)*C和A* (B*C).
- 尽管这两种不同的计算顺序所得的结果相同,但所需元素乘法数却不同.例如,

例题

- 假定A为100 × 1矩阵,B为1 × 100矩阵,C为100 × 1矩阵,(A*B) *C需乘法数为 $100\times1\times100+100\times100\times1=20000$
- 而 A* (B*C) 所需乘法数为 $1\times100\times1+100\times1\times1=200$
- 问题:对任意给定长度q的矩阵乘法链

  $$M_1\times\ldots,\times M_q\ ,$$

  求优化的乘法顺序使得计算该乘法链所用的乘法数最少.
- 长度q的矩阵乘法链有指数量级 $\Omega(2^q)$ 的可能的乘法顺序(有q个叶节点的二叉数的数目).

动态规划解

- 用M(i, j)表示链$M_i \times \ldots, \times M_j$ (i≤j)的乘积.假设优化的矩阵链乘法顺序最后计算乘积

$$M(i, k) \times M(k+1, j).$$

- 则计算M(i,j)的优化乘法顺序在计算子链M(i,k)和M(k+1,j)时的也是优化的.
- 考虑5个矩阵的乘法链,其行列数为r=(10, 5, 1, 10, 2, 10),即M_1为10×5的矩阵,等等.
- 优化的乘法顺序为 (M1×M2)×((M3×M4)×M5))
- (M3×M4)×M5对子链M(3,5)=M3×M4×M5也是优化的.

动态规划解(续)

- 设c(i,j)为计算M(i,j)的优化乘法数(优化值),根据优化原理,优化值之间满足:
  - $c(i,j)=\mathrm{MIN}_{i\le k<j}\{c(i,k)+c(k+1,j)+r_i r_{k+1} r_{j+1}\}$
  - 令kay(i,j)为达到最小值的k.
- 我们可用上述递归计算c(1,q)
- 用kay(i, j)回溯找到优化的乘法顺序.

例15-13

- 设$q = 5$和$r =(10 , 5 , 1 , 10 , 2 , 10)$,由动态规划的递归式得：

$$c(1,5) = \min\{c(1,1) + c(2,5) + 500, c(1,2) + c(3,5) + 100, c(1,3) + c(4,5) + 1000, c(1,4) + c(5,5) + 200\} \quad (15\text{-}4)$$

$$c(2,5)=\min\{c(2,2)+c(3,5)+50, c(2,3)+c(4,5)+500, c(2,4)+c(5,5)+100\} \quad (15\text{-}5)$$

例15.13

- c(3,5)=40,kay(3,5)=4

  c(2,4)=30,kay(2,4)=2

  c(1,5)=190,kay(1,5)=2
- Traceback(1,5):从kay(1,5)=2得到

  M(1,5)=M(1,2)×M(3,5)

  M(3,5)对应kay(3,5)=4,因此

  M(3,5)=M(3,4)×M(5,5)

- 计算c(i, j) 和kay(i, j) 的递归代码见程序15-6.
- 在函数C中,r 为全局一维数组,kay是全局二维数组.
- 函数C返回c(i j) 之值且置kay[i] [j]=kay (i,j).
- 函数Traceback 利用函数C中已算出的kay值来推导出最优乘法算法的步骤.

3. 迭代方法

- 函数c 的动态规划递归式可用以下迭代方法来实现.
- 按s = 2, 3, …, q-1 的顺序计算 c(i,i+s),

$$c(i,i)=0,\ 1\leqslant i\leqslant q$$

$$c(i,i+1)=r_i r_{i+1} r_{i+2};\ kay(i,i+1)=i,\ 1\leqslant i<q$$

$$c(i,i+s)=\min_{i\leqslant k<i+s}\{c(i,k)+c(k+1,i+s)+r_i r_{k+1} r_{i+s+1}\};$$

$$kay(i,i+s)=\text{获得上述最小值的}\,k$$

$$1\leqslant i\leqslant q-s,\ 1<s<q$$

- 保存计算的每个c 和kay值, 可避免大量重复计算.但需$O(q^2)$的存储空间。

例15-14

- 考察例15-13中五个矩阵的情况。
- s=2,3,4
- s=2,计算c(1,3),c(2,4),c(3,5)
- s=3,计算c(1,4),c(2,5)
- s=4,计算c(1,5)

时间复杂度

- 计算c 和kay 的迭代程序见函数MatrixChain(见程序15-8),该函数的复杂性为$O(q^3)$.计算出kay 后同样可用程序15-6中的Traceback 函数算出相应的最优乘法顺序.
- 计算C(i, i+s)需 $\Theta(s)$时间.
- 对s＝2,…q-1,要 计算q-s个C(i,i+s),时间复杂度为$\Theta((q-s)s)$.
- 所以时间复杂度为$\Theta(q^3)$

程序15-8 c 和kay 的迭代计算

```
void MatrixChain(int r[], int q, int **c, int **kay)

{// 为所有的Mij 计算耗费和 kay
  // 初始化c[i][i], c[i][i+1]和 kay[i][i+1]

  for (int i = 1; i < q; i++) {
    c[i][i] = 0;
    c[i][i+1] = r[i]*r[i+1]*r[i+2];
    kay[i][i+1] = i;
    }
  c[q][q] = 0;
```

```
//计算余下的 c和kay
for (int s = 2; s < q; s++)
  for (int i = 1; i <= q - s; i++) {
    // k = i时的最小项
    c[i][i+s] = c[i][i] + c[i+1][i+s] + r[i]*r[i+1]*r[i+s+1];
    kay[i][i+s] = i;

    // 余下的最小项
    for (int k = i+1; k < i + s; k++) {
      int t = c[i][k] + c[k+1][i+s] + r[i]*r[k+1]*r[i+s+1];
      if (t < c[i][i+s]) {// 更小的最小项
        c[i][i+s] = t;
        kay[i][i+s] = k;}
      }
    }
}
```

15.2.4 All-Pair最短路问题

- 最短路径:假设G为有向图,其中每条边都有一个成本(cost),图中每条有向路径的长度(或成本)定义为该路径上各边的成本之和.
- 对于每对顶点(i, j), 定义从i 到j 的所有路径中,具有最小长度的路径为从i 到j 的最短路.
- All-Pair最短路问题:求每对点间的最短路.
- 假定图上无负成本的环路, 这时只需考虑简单路径: 加上环路只会增加路径成本.

动态规划解

- 将节点按1到n编号(任意编号).
- 定义c(i,j,k)=i到j的中间节点编号不超过k的最短路长度,即, 包含节点i和j及节点1,…,k的子图上的最短路.
- c(i, j, n)是在原来的图上i到j的最短路长度, 也即我们要求的最短路长度.
- 因为只考虑简单路径, 所以

  c(i,k,k)=c(i,k,k-1)和

  c(k,j,k)=(k,j,k-1)

  c(i,i,k)=0 for all k
- 特别c(i,j,0)=cost(i,j)或∞.

例15-15 符号c(i,j,k)

- 如图15-4所示,从顶点1到顶点3的路径有

(1) 1,2,5,3(10)
(2) 1,4,3(28)
(3) 1,2,5,8,6,3(9)
(4) 1,4,6,3(27)
括号内数字为路径长度,路径(3)最短.

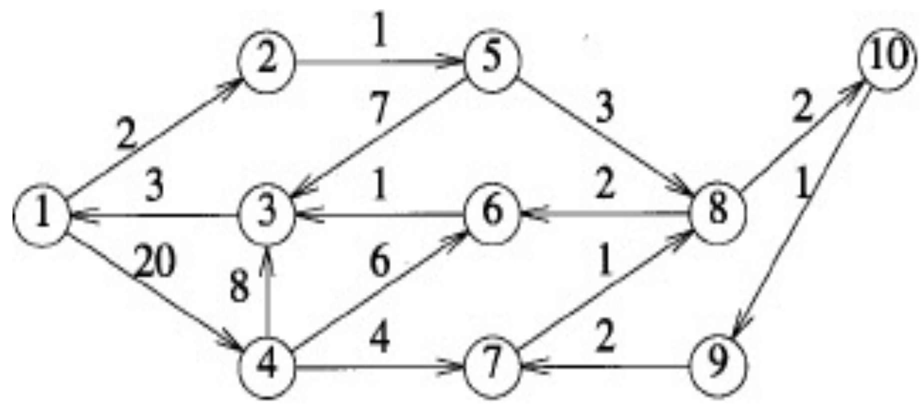


图15-4 有向图

- c(1,3,0)=∞, c(1,3,1)=∞, c(1,3,2)=∞
  c(1,3,3)=∞, c(1,3,4)=28, c(1,3,5)=10, c(1,3,6)=10
  c(1,3,7)=10, c(1,3,8)=9

- 我们建立$c(i,j,k)$和$c(i,j,k-1)$之间的递归关系.
- 对于任意$k>0$, i到j的中间节点编号不超过k的最短路上, 或包含节点k或不包括节点k. 所以有递归如下:

$$c(i,j,k)=\min\{c(i, j, k-1), c(i, k, k-1)+ c(k, j, k-1)\}$$

- 如果直接用递归程序求解上式,则计算$c(i,j,n)$的复杂度极高.利用迭代方法可将计算c 值的时间减少到$O(n^3)$.

- 迭代算法的伪代码如下:
- 令$C^{(k)}$代表矩阵$(c(i,j,k))_{i,j=1,\dots,n}$,因$c(i,i,k)=0$ for all k,所以矩阵$C^{(k)}$的对角线元素为0.
- 算法迭代计算$C^{(k)}$,$k=0,\dots,n$
- 初始$C^{(0)}=(c(i, j))$,即图的邻接矩阵,无边相连的i和j 令$c(i, j)=\infty$.
- 因$c(i,k,k)=c(i,k,k-1)$,$c(k,j,k)=c(k,j,k-1)$,所以,矩阵$C^{(k)}$的k行、k列上的元素不变:

  $C^{(k)}(i,k)=C^{(k-1)}(i,k)$, $C^{(k)}(k,j)=C^{(k-1)}(k,j)$.

- 矩阵$C^{(k)}$非k行、k列上的元素,按下式计算

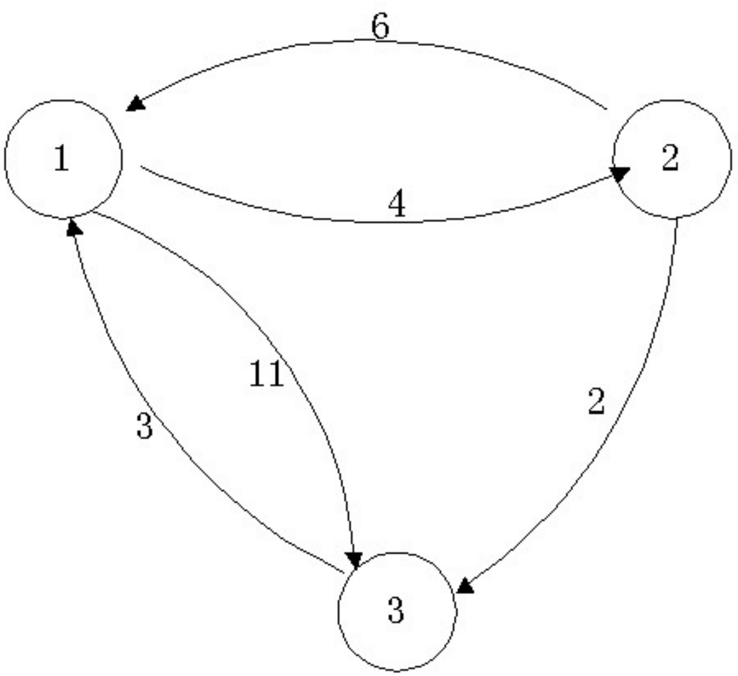
$$C^{(k)}(i,j) \leftarrow \min\{C^{(k-1)}(i, j),\ C^{(k-1)}(i, k) + C^{(k-1)}(k,j)\},$$

即

$$C^{(k)}(i,j) \leftarrow \min\{C^{(k-1)}(i, j),\ C^{(k)}(i, k) + C^{(k)}(k,j)\},$$

- 所以算法只需使用一个矩阵,每次迭代时,用第k列的i行元素和第k行的j列元素之和去更新元素$C^{(k-1)}(i, j)$.
- 算法迭代至多n次，每次迭代需$O(n^2)$时间，所以算法的时间复杂度为$O(n^3)$.

图15-6 最短路径的例子



$$A = \begin{pmatrix} \infty & 4 & 11 \\ 6 & \infty & 2 \\ 3 & \infty & \infty \end{pmatrix}$$

最短路径

$$C^{(0)}=\begin{pmatrix}0 & 4 & 11\\ 6 & 0 & 2\\ 3 & \infty & 0\end{pmatrix}$$

$C^{(0)}(i, j)=c(i, j)$

$$C^{(1)}=\begin{pmatrix}0 & 4 & 11\\ 6 & 0 & 2\\ 3 & 7 & 0\end{pmatrix}$$

$C^{(1)}(3,2)=\min\{C^{(0)}(3,2),C^{(0)}(3,1)+C^{(0)}(1,2)\}=7$

$$C^{(2)}=\begin{pmatrix}0 & 4 & 6\\ 6 & 0 & 2\\ 3 & 7 & 0\end{pmatrix}$$

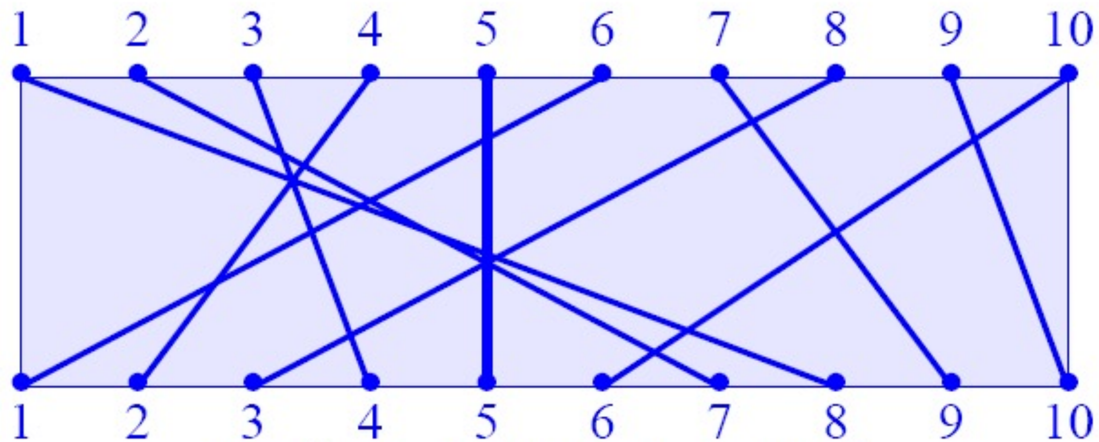
$C^{(2)}(1,3)=\min\{C^{(1)}(1,3), C^{(1)}(1,2)+C^{(1)}(2,3)\}$
$=\min\{11,4+2\}=6$

$$C^{(3)}=\begin{pmatrix}0 & 4 & 6\\ 5 & 0 & 2\\ 3 & 7 & 0\end{pmatrix}$$

$C^{(3)}(1,2)=\min\{4,6+7\}=4$

$C^{(3)}(2,1)=\min\{6,2+3\}=5$

15.2.5Noncrossing subset of nets



$C = [8, 7, 4, 2, 5, 1, 9, 3, 10, 6]$

Figure 15.7 A wiring instance

15.2.5不交叉网的子集

- 图15.7中每个i 有唯一一个网(i,C_i)
- 例如图15.7满足$i\leq 5$ 且$C_i\leq 7$的不交叉网的子集有：

  (1)满足上述条件的单个网构成的子集

  (2)子集{(4,2),(5,5)}和{(3,4),(5,5)}
- 最大子集为{(4,2),(5,5)}

15.2.5不交叉网的子集

- MNS(最大不交叉网的子集)
- 设MNS(i,j)为所有满足：

  $u \le i$ 且$C_u \le j$

  的最大不交叉网构成的子集
- size(i,j)为MNS(i,j)内的nets数目

  根据优化解中是否包含网(i,C_i)我们有以下递归关系：

  (15.6)、(15.7)和(15.8)

15.2.5不交叉网的子集

$$size(1,j)=0 \qquad if\ \ j<C_1 \qquad (15.6)$$

$$size(1,j)=1 \qquad j\ge C_1$$

$$size(i,j)=size(i-1,j), \quad j<C_i \qquad (15.7)$$

$$size(i,j)=\max\{size(i-1,j), size(i-1,C_i-1)+1\}, \quad j\ge C_i$$

(15.8)

例15.20

- 对图15.8，有:

size(1,j)=0, i=1,2,3,4,5,6,7

size(1,j)=1,i=8,9,10

因 c_2=7,所以size(2,j)=0 for j=1,…,6

size(2,7)=size(1,6)+1=0

size(2,8)=max{size(1,8),size(1,6)+1}=1

- 其它见图15.9

例15.20

i	j									
	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	1	1	1
2	0	0	0	0	0	0	1	1	1	1
3	0	0	0	1	1	1	1	1	1	1
4	0	1	1	1	1	1	1	1	1	1
5	0	1	1	1	2	2	2	2	2	2
6	1	1	1	1	2	2	2	2	2	2
7	1	1	1	1	2	2	2	2	3	3
8	1	1	2	2	2	2	2	2	3	3
9	1	1	2	2	2	2	2	2	3	4
10	1	1	2	2	2	3	3	3	3	4

Figure 15.9 $size(i,j)$s for the instance of Figure 15.7

例15.20(taceback)

- Nets 可按i 编号
- 如果size(i,j)!=size(i-1,j) 则MNS(i,j)包含net i

  令$j = C_i - 1$；

  重复上述过程；

复习要求

- 根据优化原理列递归式
- 设计实现递归式的迭代算法(列表)
- 0/1背包问题

  矩阵乘法链

  多段图

  求各对点之间的最短路
- 要求会做实例；分析算法的复杂度

习题

1. 设c(i)为多段图上节点1到目的节点的最短路长度，试列出动态规划的递归式.并就课堂上的例子给出求解过程.

2. 0/1背包问题:

$n=4, c=20, w=(10,15,6,9)$

$p=(2,5,8,1)$

(1) 产生元组集合P(1),P(2),P(3)和该背包问实例的解

(2)证明当重量和效益值均为整数时动态规划算法的时间复杂度为

$$O(\min\{2^n, n\Sigma_{1\le i\le n}p_i, nc\})$$

提示:$|P(i)|\le 1+\Sigma_{i\le j\le n}p_j$

习题

3. 子集和数问题:设$S=\{s_1,s_2,\ldots,s_n\}$ 为n个正数的集合,试找出和数不超过M且最大的S的子集, 该问题是NP-难度问题,试用动态规划法设计一算法.

习题

4. 设一个矩阵乘法链的行列数为 r=(10,20,50,1,100),用动态规划算法给出优化的乘法顺序和优化的乘法数.
5. 补充例题15.17的计算过程
6. 本章习题19