程序设计方法与实践

——动态规划

动态规划

- 动态规划（dynamic programming）是一种算法设计技术，是一种**多阶段决策过程**最优的通用方法。
- 如果问题是由交叠的子问题所构成的，就可以用动态规划技术。
  - 这样的子问题出现在对给定问题求解的递推关系中，这个递推关系中包含了相同类型的更小子问题的解
- 思想：与其对交叠的子问题一次又一次的求解，不如对每个较小的子问题**只求解一次**并把结果记录在表中。这样就可以从表中得出原始问题的解。

动态规划

动态规划的解决该类问题的思想：

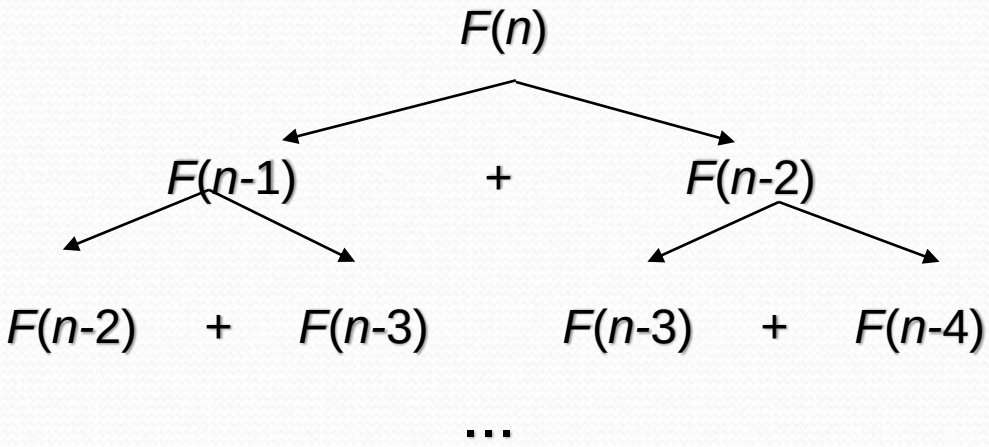
- 对较小的子问题进行一次求解，并把结果记录下来，然后利用较小问题的解，求解出较大问题的解，直到求解出最大问题的解。

- 斐波那契数列Fibonacci sequence:

  0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , …

  - $F(n) = F(n\text{-}1) + F(n\text{-}2)$
  - $F(0) = 0$
  - $F(1) = 1$

动态规划



- 递归式中，对函数相同值计算了多遍。使用表格记录方法

$F(0) = 0$
$F(1) = 1$
$F(2) = 1+0 = 1$
…
$F(n-2) =$
$F(n-1) =$
$F(n) = F(n-1) + F(n-2)$

0	1	1	. . .	$F(n-2)$	$F(n-1)$	$F(n)$

进一步简化，表格可以仅存储最后两个值

8.1三个基本例子

例1，币值最大化问题：

- 给定一排n个硬币，面值为正整数c1,c2…cn,这些整数并不一定两两不同。如何选择硬币，使得在其原始位置互不相邻的条件下，所选硬币总币值最大。
- 思路：较小的问题解决后记录下来，帮助求解较大的问题。
- 设F(*n*) 总币值
- 考虑 如果选择了最后一个硬币,最大总币值?
  如果没有选择最后一个硬币，最大总币值?

- $F(n) = \max\{c_n + F(n-2),\ F(n-1)\}$ for $n > 1$,
- $F(0) = 0,\ F(1)=c_1$

8.1三个基本例子

例1，币值最大化问题：

- $F(n) = \max\{c_n + F(n-2),\ F(n-1)\}$ for $n > 1$,
- $F(0) = 0,\ F(1)=c_1$
- 5, 1, 2, 10, 6, 2

index	0	1	2	3	4	5	6
coins	--	5	1	2	10	6	2
F( )	0	5					

F(0)=0,F(1)=5

8.1三个基本例子

例1，币值最大化问题：

- $F(n) = \max\{c_n + F(n-2),\ F(n-1)\}$ for $n > 1$,
- $F(0) = 0,\ F(1)=c_1$
- 5, 1, 2, 10, 6, 2

index	0	1	2	3	4	5	6
coins	--	5	1	2	10	6	2
F( )	0	5	5				

F(2)=max(1+0,5)=5

8.1三个基本例子

例1，币值最大化问题：

- F(n) = max{c_n + F(n-2), F(n-1)} for n > 1,
- F(0) = 0, F(1)=c_1
- 5, 1, 2, 10, 6, 2

index	0	1	2	3	4	5	6
coins	--	5	1	2	10	6	2
F( )	0	5	5	7			

F(3)=max(2+5,5)=7

8.1三个基本例子

例1，币值最大化问题：

- $F(n) = \max\{c_n + F(n-2),\ F(n-1)\}$ for $n > 1$,
- $F(0) = 0,\ F(1)=c_1$
- 5, 1, 2, 10, 6, 2

index	0	1	2	3	4	5	6
coins	--	5	1	2	10	6	2
F( )	0	5	5	7	15		

F(4)=max(10+5,7)=15

8.1三个基本例子

例1，币值最大化问题：

- $F(n) = \max\{c_n + F(n-2),\ F(n-1)\}$ for $n > 1$,
- $F(0) = 0,\ F(1)=c_1$
- 5, 1, 2, 10, 6, 2

index	0	1	2	3	4	5	6
coins	--	5	1	2	10	6	2
F( )	0	5	5	7	15	15	

F(5)=max(6+7,15)=15

8.1三个基本例子

例1，币值最大化问题：

- $F(n) = \max\{c_n + F(n-2),\ F(n-1)\}$ for $n > 1$,
- $F(0) = 0,\ F(1)=c_1$
- 5, 1, 2, 10, 6, 2

index	0	1	2	3	4	5	6
coins	--	5	1	2	10	6	2
F( )	0	5	5	7	15	15	17

F(6)=max(2+15,15)=17

8.1三个基本例子

例2，找零问题：

- 设找零金额为n，最少用到多少面额为d1<d2<…<dm的硬币？假设有m种面额,且d1=1。
- 较小的问题解决后记录下来，帮助求解较大的问题。
- 定义F(n)为金额n的最少硬币数目，F(0)=0。
- 找零方法：在总额n-dj的硬币上加入一个面值dj的硬币
- $F(n) = \min_{j:n \geq d_j} \{F(n-d_j)\} + 1 \qquad F(0) = 0$

8.1三个基本例子

例2，找零问题：

- $F(n) = \min_{j:n \geq d_j} \{F(n-d_j)\} + 1$ $\qquad F(0) = 0$
- n=6,币值为d1=1,d2=3,d3=4

n	0	1	2	3	4	5	6
F	0						

F(0)=0

8.1三个基本例子

例2，找零问题：

- $F(n) = \min_{j:n \geq d_j} \{F(n-d_j)\} + 1$ $\qquad F(0) = 0$
- n=6,币值为d1=1,d2=3,d3=4

n	0	1	2	3	4	5	6
F	0	1					

F(1)=min(F(1-1)}+1=1

8.1三个基本例子

例2，找零问题：

- $F(n) = \min_{j:n \geq d_j} \{F(n-d_j)\} + 1$ $\qquad F(0) = 0$
- n=6,币值为d1=1,d2=3,d3=4

n	0	1	2	3	4	5	6
F	0	1	2				

F(2)=min(F(2-1)}+1=2

8.1三个基本例子

例2，找零问题：

- $F(n) = \min_{j:n \geq d_j} \{F(n-d_j)\} + 1$ $\qquad F(0) = 0$
- n=6,币值为d1=1,d2=3,d3=4

n	0	1	2	3	4	5	6
F	0	1	2	1			

F(3)=min(F(3-1),F(3-3)}+1=1

8.1三个基本例子

例2，找零问题：

- $F(n) = \min_{j:n \geq d_j} \{F(n-d_j)\} + 1$ $\qquad F(0) = 0$
- n=6,币值为d1=1,d2=3,d3=4

n	0	1	2	3	4	5	6
F	0	1	2	1	1		

F(4)=min(F(4-1),F(4-3),F(4-4)}+1=1

8.1三个基本例子

例2，找零问题：

- $F(n) = \min_{j:n \geq d_j} \{F(n-d_j)\} + 1$ $\qquad F(0) = 0$
- n=6,币值为d1=1,d2=3,d3=4

n	0	1	2	3	4	5	6
F	0	1	2	1	1	2	

F(5)=min(F(5-1),F(5-3),F(5-4)}+1=2

8.1三个基本例子

例2，找零问题：

- $F(n) = \min_{j:n \geq d_j} \{F(n-d_j)\} + 1 \qquad F(0) = 0$
- n=6,币值为d1=1,d2=3,d3=4

n	0	1	2	3	4	5	6
F	0	1	2	1	1	2	2

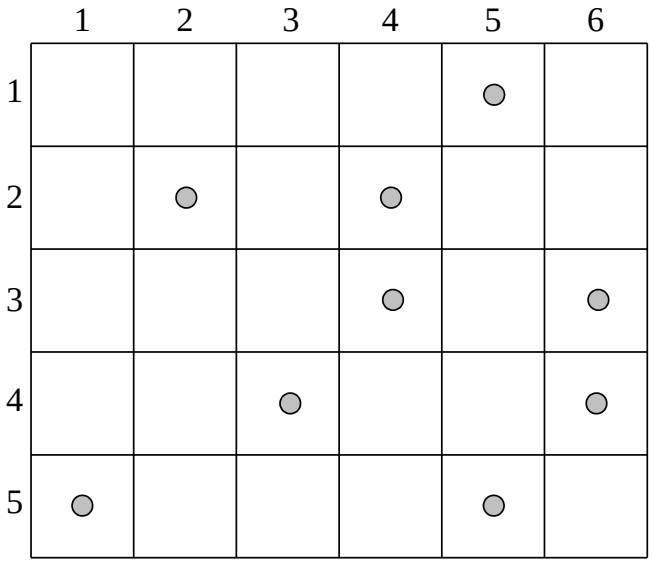
F(6)=min(F(6-1),F(6-3),F(6-4)}+1=2

8.1三个基本例子

例3，硬币收集问题：

- N×m格木板放有一些硬币，每格最多一个硬币。木板左上方一个机器人收集尽可能多的硬币并把它们带到右下方的单元格。每一步机器人可以从当前位置右移一格或者下移一格。如果遇到硬币就收集起来。求机器人收集最大硬币数给出移动路径。

- 设F(i,j)为走到i行j列收集最大硬币数
- (i,j)可以由(i-1,j)或者(i,j-1)到达,对应已收集硬币F(i-1,j)和F(i,j-1)。

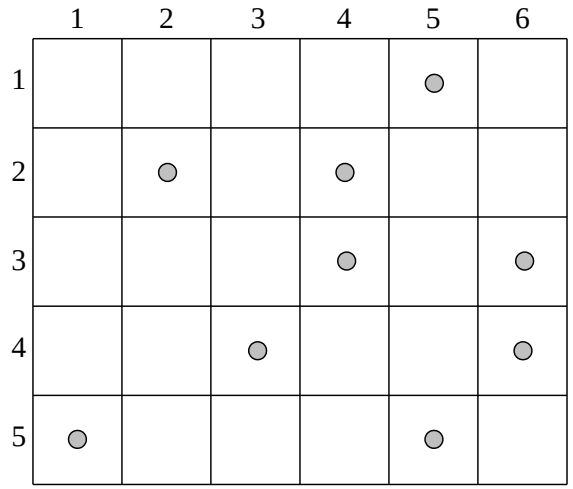


$$F(i,j) = max\{F(i-1,j), F(i,j-1)\} + c_{i,j}$$
$$F(0,j) = 0; F(i,0) = 0;$$

8.1三个基本例子

例3，硬币收集问题：

- $F(i,j) = max\{F(i-1,j), F(i,j-1)\} + c_{i,j}$，　$1 \le i \le n, 1 \le j \le m$,
- $F(0,j) = 0$，　$1 \le j \le m$; $F(i,0) = 0$，　$1 \le i \le n$;



	1	2	3	4	5	6
1	0					
2						
3						
4						
5						

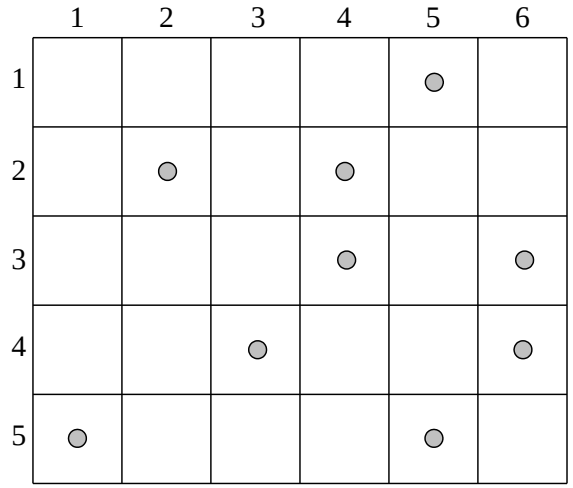
F(1,1)=C[1,1]=0

计算左上角

8.1三个基本例子

例3，硬币收集问题：

- $F(i,j) = max\{F(i-1,j), F(i,j-1)\} + c_{i,j}$，　$1 \le i \le n, 1 \le j \le m$,
- $F(0,j) = 0$，　$1 \le j \le m$; $F(i,0) = 0$，　$1 \le i \le n$;



	1	2	3	4	5	6
1	0	0	0	0	1	1
2						
3						
4						
5						

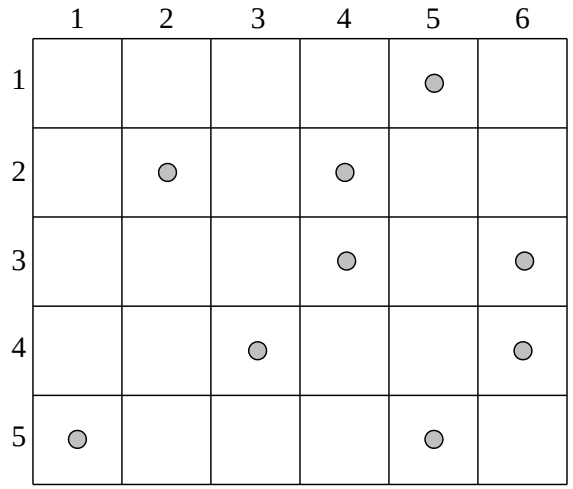
F(1,2)=max(F(0,2),F(1,1)}+C[1,2]=0

计算第一行

8.1三个基本例子

例3，硬币收集问题：

- $F(i,j) = max\{F(i-1,j), F(i,j-1)\} + c_{i,j}$，　$1 \le i \le n, 1 \le j \le m$,
- $F(0,j) = 0$，　$1 \le j \le m$; $F(i,0) = 0$，　$1 \le i \le n$;



	1	2	3	4	5	6
1	0	0	0	0	1	1
2	0					
3	0					
4	0					
5	1					

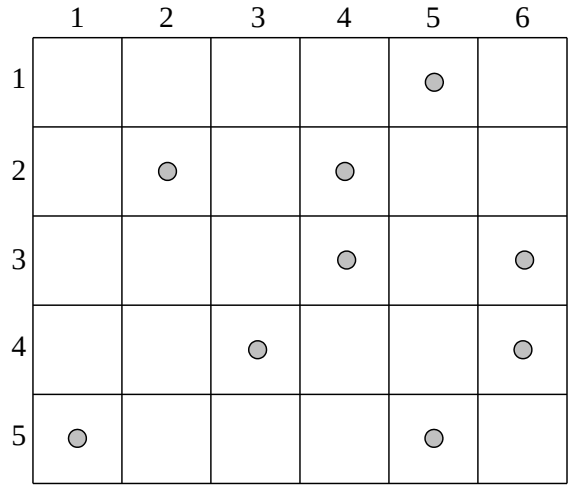
F(2,1)=max(F(1,1),F(2,0)}+C[2,1]=0

计算第一列

8.1三个基本例子

例3，硬币收集问题：

- $F(i,j) = max\{F(i-1,j), F(i,j-1)\} + c_{i,j}$，　$1 \le i \le n, 1 \le j \le m$,
- $F(0,j) = 0$，　$1 \le j \le m$; $F(i,0) = 0$，　$1 \le i \le n$;



	1	2	3	4	5	6
1	0	0	0	0	1	1
2	0	1	1	2	2	2
3	0					
4	0					
5	1					

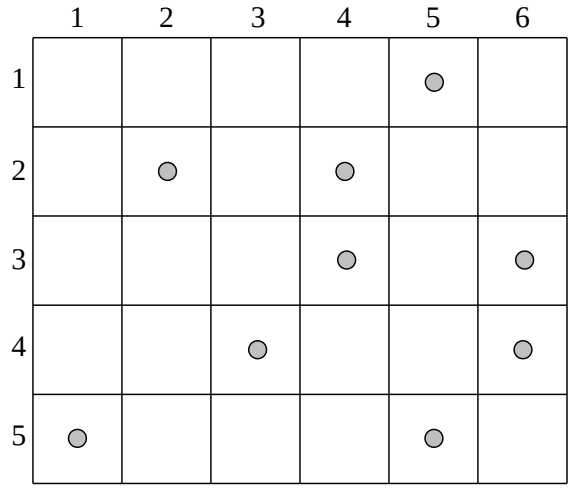
F(2,2)=max(F(1,2),F(2,1)}+C[2,2]=1

计算第二行

8.1三个基本例子

例3，硬币收集问题：

- $F(i,j)=max\{F(i-1,j),F(i,j-1)\}+c_{i,j}$，　$1\le i\le$ n, $1\le$ j $\le$ m,
- $F(0,j)=0$，　$1\le$ j $\le$ m; $F(i,0)=0$，　$1\le i\le$ n;



	1	2	3	4	5	6
1	0	0	0	0	1	1
2	0	1	1	2	2	2
3	0	1	1	3	3	4
4	0	1	2	3	3	5
5	1	1	2	3	4	5

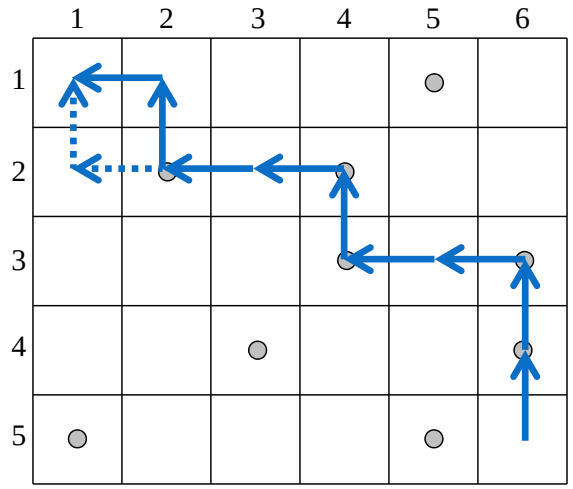
F(3,2)=max(F(1,2),F(2,1)}+C[2,2]=1

计算第三行、四行、五行

8.1三个基本例子

例3，硬币收集问题：

- $F(i,j) = max\{F(i-1,j), F(i,j-1)\} + c_{i,j}$，　$1 \le i \le n, 1 \le j \le m$,
- $F(0,j) = 0$，　$1 \le j \le m$; $F(i,0) = 0$，　$1 \le i \le n$;
- **通过回溯找路径**



	1	2	3	4	5	6
1	0	0	0	0	1	1
2	0	1	1	2	2	2
3	0	1	1	3	3	4
4	0	1	2	3	3	5
5	1	1	2	3	4	5

8.2背包问题和记忆功能

- 给定n个重量为w1,…,wn的价值为v1,…,vn的物品和一个承重为W的背包，求这些物品中最有价值的一个子集。

- 找递推公式，用较小实例的解的形式表示背包问题解。
- 由前i个物品定义的实例，背包承重j。F(i,j)是最优解。
  - 如果不包括第i个物品的子集中，最优子集价值为F(i-1,j)
  - 包含第i个物品子集中，最优子集由F(i-1,j-wi)+vi

- $$F(\boldsymbol{i},\boldsymbol{j}) = \begin{cases} \boldsymbol{max}\{\boldsymbol{F}(\boldsymbol{i}-\boldsymbol{1},\boldsymbol{j}), v_i + \boldsymbol{F}(\boldsymbol{i}-1,\boldsymbol{j}-w_i)\}, & j - w_i \geq 0 \\ F(i-1,j) & ,j - w_i < 0 \end{cases}$$

容量可放第i个物体

放不下第i个物体了

8.2背包问题和记忆功能

- $$F(i,j)=\begin{cases} max\{F(i-1,j),v_i+F(i-1,j-w_i)\}, & j-w_i\geq 0 \\ F(i-1,j), & j-w_i<0 \end{cases}$$
- 例：承重W=5

物品	重量	价值
1	2	12
2	1	10
3	3	20
4	2	15

- 1、填入第一行第一列

背包承重

放i物体

i,w	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

8.2背包问题和记忆功能

- $F(i,j)=\begin{cases} max\{F(i-1,j),v_i+F(i-1,j-w_i)\}, & j-w_i\geq 0 \\ F(i-1,j), & j-w_i<0 \end{cases}$
- 例：承重W=5

物品	重量	价值
1	2	12
2	1	10
3	3	20
4	2	15

- 2、w1=2,v1=12

  F(0,j),F(0,j-2)+12

i,w	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0					
3	0					
4	0					

8.2背包问题和记忆功能

- $$F(i,j)=\begin{cases}max\{F(i-1,j),v_i+F(i-1,j-w_i)\}, & j-w_i\geq 0\\ F(i-1,j), & j-w_i<0\end{cases}$$
- 例：承重W=5

物品	重量	价值
1	2	12
2	1	10
3	3	20
4	2	15

- 3、w2=1,v2=10

  F(1,j),F(1,j-1)+10

i,w	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0					
4	0					

8.2背包问题和记忆功能

- $$F(i,j)=\begin{cases}max\{F(i-1,j),v_i+F(i-1,j-w_i)\}, & j-w_i\geq 0\\ F(i-1,j), & j-w_i<0\end{cases}$$
- 例：承重W=5

物品	重量	价值
1	2	12
2	1	10
3	3	20
4	2	15

- 4、w3=3,v3=20

  F(2,j),F(2,j-3)+20

i,w	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0					

8.2背包问题和记忆功能

- $$F(i,j)=\begin{cases} max\{F(i-1,j),v_i+F(i-1,j-w_i)\}, & j-w_i\geq 0 \\ F(i-1,j), & j-w_i<0 \end{cases}$$
- 例：承重W=5

物品	重量	价值
1	2	12
2	1	10
3	3	20
4	2	15

- 5、w4=2,v4=15

  F(3,j),F(3,j-2)+15

i,w	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	37

8.2背包问题和记忆功能

背包问题的自底向上的动态规划伪代码。

a. **Algorithm** *DPKnapsack*$(w[1..n], v[1..n], W)$
//Solves the knapsack problem by dynamic programming (bottom up)
//Input: Arrays $w[1..n]$ and $v[1..n]$ of weights and values of n items,
// knapsack capacity W
//Output: Table $V[0..n, 0..W]$ that contains the value of an optimal
// subset in $V[n, W]$ and from which the items of an optimal
// subset can be found
for $i \leftarrow 0$ **to** n **do** $V[i, 0] \leftarrow 0$
for $j \leftarrow 1$ **to** W **do** $V[0, j] \leftarrow 0$
for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow 1$ **to** W **do**
    if $j - w[i] \geq 0$
      $V[i, j] \leftarrow \max\{V[i-1, j], v[i] + V[i-1, j-w[i]]\}$
    else $V[i, j] \leftarrow V[i-1, j]$
return $V[n, W], V$

8.2背包问题和记忆功能

- 记忆化
  - 前述问题用动态规划求解，满足一个用交叠的子问题来表示递推关系。如果用自顶而下的递推关系求解导致多次求解公共子问题，效率低下。如果自底向上求解填充表格，每个子问题求一次。但某些具体问题的解并不一定需要所有子问题求解。使用自顶向下和自底向上结合，只对必要的子问题求解且仅求解一次。用记忆化方法
  - 自顶向下求解，同时维护一个自底向上的动态规划表格，同时标记每个单元格是否被计算过。之后一旦需要计算一个值，先检查是否计算过，如果已经计算过则直接取值，否则使用递归调用进行计算，并记录返回值到表格中。

8.2背包问题和记忆功能

- 记忆化
  - 背包问题改进

```
MFK（i,j）算法
if F[i,j]<0                    //表示还没有计算过，需要计算
    if j<weights[i]
        value = MFK(i-1,j)
    else
        value = max(MFK(i-1,j),values[i]+MFK(i-1,j-weight[i]))
    F[i,j]=value      //记录计算结果，以后复用
Return F[i,j]
```

8.2背包问题和记忆功能

- 记忆化
- $$F(\boldsymbol{i},\boldsymbol{j})=\begin{cases}\boldsymbol{max}\{\boldsymbol{F}(\boldsymbol{i}-\boldsymbol{1},\boldsymbol{j}),v_i+\boldsymbol{F}(\boldsymbol{i}-1,\boldsymbol{j}-w_i)\},j-w_i\geq 0\\ F(i-1,j) \qquad\qquad ,j-w_i<0\end{cases}$$
- 例：承重W=5

物品	重量	价值
1	2	12
2	1	10
3	3	20
4	2	15

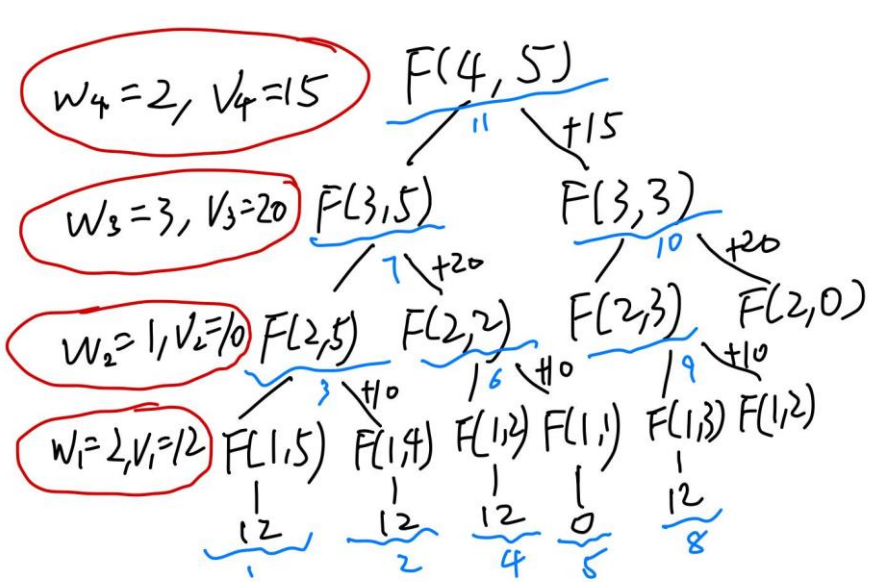
- **填入第一行第一列**
- 其他位置赋值-1表示未计算过

i,w	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	-1	-1	-1	-1	-1
2	0	-1	-1	-1	-1	-1
3	0	-1	-1	-1	-1	-1
4	0	-1	-1	-1	-1	-1

8.2背包问题和记忆功能

- 记忆化
- $F(i,j)=\begin{cases} max\{F(i-1,j), v_i+F(i-1,j-w_i)\}, j-w_i\geq 0 \\ F(i-1,j) \end{cases}, j-w_i<0$
- 例：承重W=5

物品	重量	价值
1	2	12
2	1	10
3	3	20
4	2	15



i,w	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	-1	12	22	-1	22
3	0	-1	-1	22	-1	32
4	0	-1	-1	-1	-1	37

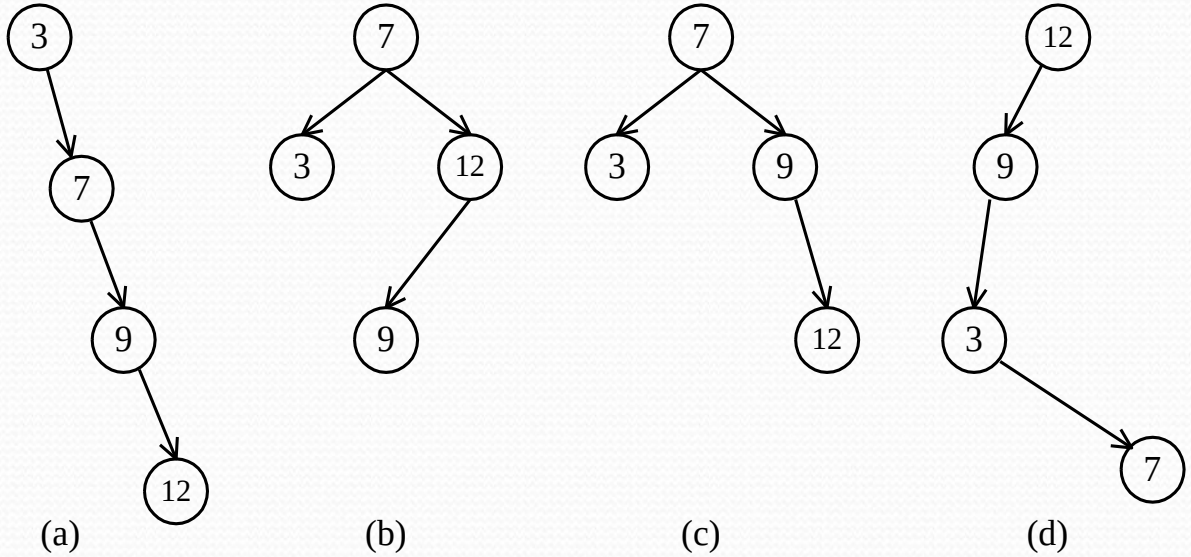
8.3 最优二叉查找树

- 最优二叉查找树就是在查找中的平均键值比较次数最低的。
- 设$a_1,\ldots,a_n$是从小到大排列的互不相等的键。p_1，…，p_n是它们的查找概率。T_i^j是由键$a_i,\ldots a_j$构成的二叉树，C[i,j]是在这棵树中成功查找的最小的平均查找次数。
- 为了推导出动态规划算法中隐含的递推关系，需要考虑从键$a_i,\ldots a_j$中选择一个**根的所有可能的方法**。即
  - 它的根为a_k，
  - 它的左子树T_i^{k-1}中的键$a_i,\ldots a_{k-1}$是最优排列的，
  - 它的右子树T^j_{k+1}中的键$a_{k+1},\ldots a_j$也是最优排列的。

8.3 最优二叉查找树

- 二叉查找树一种重要的数据结构，主要应用于字典的实现。
- 例如：**设3，7，9，12四个键分别以概率0.1,0.2,0.4,0.3来查找。**

3，7，9，12的二叉查找树有：



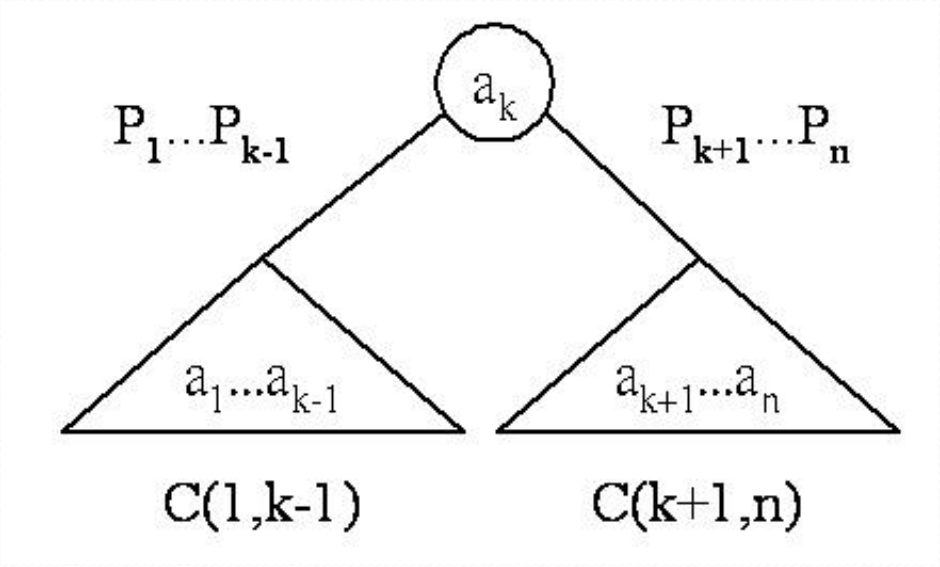
哪一棵树在成功查找时，键的平均比较次数最少?

0.1*1+0.2*2 +0.4*3+0.3* 4=2.9	0.1*2+0.2*1 +0.4*3+0.3* 2=2.2	0.1*2+0.2*1 +0.4*2+0.3* 3=2.1	0.1*3+0.2*4 +0.4*2+0.3* 1=2.2

8.3 最优二叉查找树

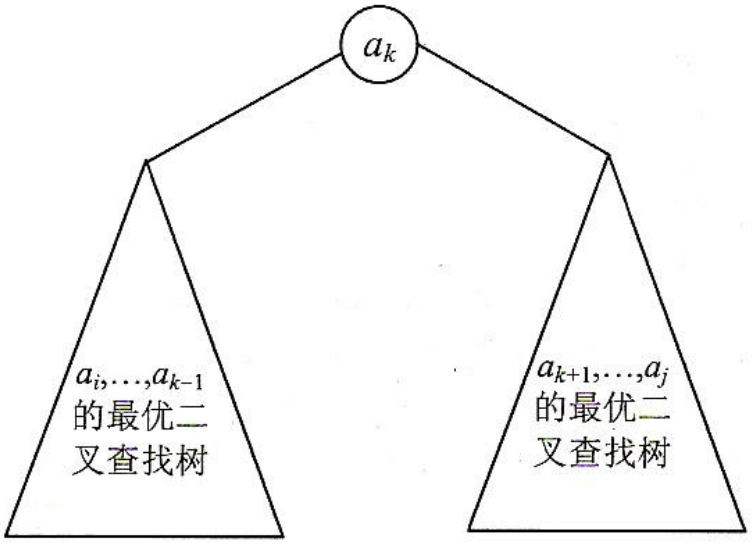
- 对于n个$a_1<a_2<...<a_n$互不相等的键，设$p_1,p_2,...,p_n$分别是它们的查找概率。
- 假设T_1^n是由这n个键构成的一棵最优二叉查找树，如何计算成功查找的最小平均键比较次数$C[1,n]$，以及如何构造这棵树？
- 动态规划分析

  划分阶段(子问题)：为了得到C[1,n]，须从1到n中选取一个节点k作为根节点，则有这样一棵树：



8.3 最优二叉查找树

- 对于$a_i...a_j$构成的一棵最优二叉树，有：



$$
\begin{aligned}
C[i,j] &= \min_{i\le k\le j}\{p_k\times 1+\sum_{s=i}^{k-1}p_s\times(a_s\text{在}T_i^{k-1}\text{中的层数}+1)\\
&\quad+\sum_{s=k+1}^{j}p_s\times(a_s\text{在}T_{k+1}^{j}\text{中的层数}+1)\}\\
&=\min_{i\le k\le j}\{p_k+\sum_{s=i}^{k-1}p_s\times a_s\text{在}T_i^{k-1}\text{中的层数}+\sum_{s=i}^{k-1}p_s\\
&\quad+\sum_{s=k+1}^{j}p_s\times a_s\text{在}T_{k+1}^{j}\text{中的层数}+\sum_{s=k+1}^{j}p_s\}\\
&=\min_{i\le k\le j}\{\sum_{s=i}^{k-1}p_s\times a_s\text{在}T_i^{k-1}\text{中的层数}\\
&\quad+\sum_{s=k+1}^{j}p_s\times a_s\text{在}T_{k+1}^{j}\text{中的层数}+\sum_{s=i}^{j}p_s\}\\
&=\min_{i\le k\le j}\{C[i,k-1]+C[k+1,j]\}+\sum_{s=i}^{j}p_s
\end{aligned}
$$

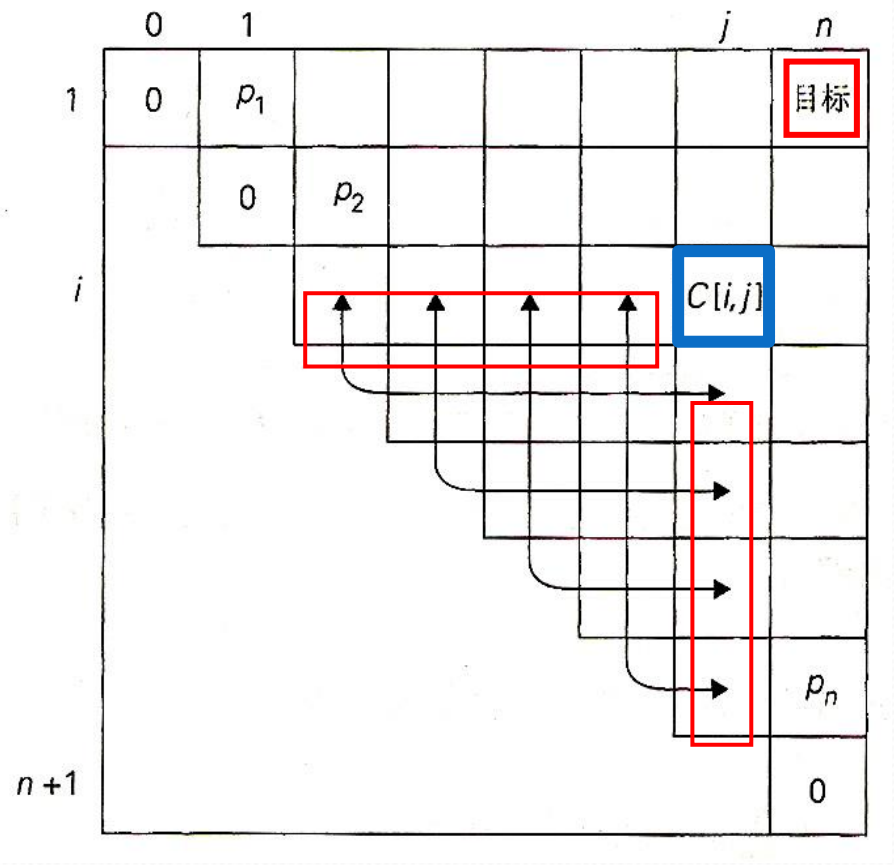
当$1\le i\le j\le n$时，$C[i,j]=\min_{i\le k\le j}\{C[i,k-1]+C[k+1,j]\}+\sum_{s=i}^{j}p_s$

当$1\le i\le n+1$时，$C[i,i-1]=0$　　当$1\le i\le n$时，$C[i,i]=p_i$

8.3 最优二叉查找树

$$C[i,j] = \min_{i \le k \le j} \{C[i,k-1] + C[k+1,j]\} + \sum_{s=i}^{j} p_s$$

当$1 \le i \le n+1$时，$C[i,i-1]=0$　当$1 \le i \le n$时，$C[i,i]=p_i$



这里只是得到最优二叉树中成功平均比较次数。

如果想得到二叉树本身，该怎么办？

8.3 最优二叉查找树

- 最优二叉树举例

键	3	7	9	12
概率	0.1	0.2	0.4	0.3

初始表格

主表

	0	1	2	3	4
1	0	0.1			
2		0	0.2		
3			0	0.4	
4				0	0.3
5					0

根表

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

当$1 \le i \le n+1$时，$C[i,i-1]=0$　当$1 \le i \le n$时，$C[i,i]=p_i$

$$\text{当} 1 \le i \le j \le n \text{ 时，} \quad C[i,j] = \min_{i \le k \le j} \{C[i,k-1] + C[k+1,j]\} + \sum_{s=i}^{j} p_s$$

- 最优二叉树举例

键	3	7	9	12
概率	0.1	0.2	0.4	0.3

主表

	0	1	2	3	4
1	0	0.1	0.4		
2		0	0.2		
3			0	0.4	
4				0	0.3
5					0

根表

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

$$C[1,2] = \min \begin{cases} k=1: C[1,0] + C[2,2] + \sum_{s=1}^{2} p_s = 0 + 0.2 + 0.3 = 0.5 \\ k=2: C[1,1] + C[3,2] + \sum_{s=1}^{2} p_s = 0.1 + 0 + 0.3 = 0.4 \end{cases} = 0.4$$

$$\text{当} 1 \le i \le j \le n \text{ 时，} \quad C[i,j] = \min_{i \le k \le j} \{C[i,k-1] + C[k+1,j]\} + \sum_{s=i}^{j} p_s$$

- 最优二叉树举例

键	3	7	9	12
概率	0.1	0.2	0.4	0.3

主表

	0	1	2	3	4
1	0	0.1	**0.4**		
2		0	0.2	**0.8**	
3			0	0.4	
4				0	0.3
5					0

根表

	0	1	2	3	4
1		1	**2**		
2			2	**3**	
3				3	
4					4
5					

$$C[2,3] = \min \begin{array}{ll} k=2: & C[2,1] + C[3,3] + \sum_{s=2}^{3} p_s = 0 + 0.4 + 0.6 = 1.0 \\ k=3: & C[2,2] + C[4,3] + \sum_{s=2}^{3} p_s = 0.2 + 0 + 0.6 = \mathbf{0.8} \end{array} = 0.8$$

$$\text{当} 1 \le i \le j \le n \text{ 时，} \quad C[i,j] = \min_{i \le k \le j} \{C[i,k-1] + C[k+1,j]\} + \sum_{s=i}^{j} p_s$$

- 最优二叉树举例

键	3	7	9	12
概率	0.1	0.2	0.4	0.3

主表

	0	1	2	3	4
1	0	0.1	**0.4**		
2		0	0.2	**0.8**	
3			0	0.4	**1.0**
4				0	0.3
5					0

根表

	0	1	2	3	4
1		1	**2**		
2			2	**3**	
3				3	**3**
4					4
5					

$$C[3,4] = \min \begin{array}{ll} k=3: & C[3,2] + C[4,4] + \sum_{s=3}^{4} p_s = 0 + 0.3 + 0.7 = \mathbf{1.0} \\ k=4: & C[3,3] + C[5,4] + \sum_{s=3}^{4} p_s = 0.4 + 0 + 0.7 = 1.1 \end{array} = 1.0$$

$$\text{当} 1 \le i \le j \le n \text{ 时，} \quad C[i,j] = \min_{i \le k \le j} \{C[i,k-1] + C[k+1,j]\} + \sum_{s=i}^{j} p_s$$

- 最优二叉树举例

键	3	7	9	12
概率	0.1	0.2	0.4	0.3

主表

	0	1	2	3	4
1	0	0.1	**0.4**	**1.1**	
2		0	0.2	**0.8**	
3			0	0.4	**1.0**
4				0	0.3
5					0

根表

	0	1	2	3	4
1		1	**2**	**3**	
2			2	**3**	
3				3	**3**
4					4
5					

$$C[1,3] = \min \begin{cases} k=1: & C[1,0] + C[2,3] + \sum_{s=1}^{3} p_s = 0 + 0.8 + 0.7 = 1.5 \\ k=2: & C[1,1] + C[3,3] + \sum_{s=1}^{3} p_s = 0.1 + 0.4 + 0.7 = 1.2 \\ k=3: & C[1,2] + C[4,3] + \sum_{s=1}^{3} p_s = 0.4 + 0 + 0.7 = \mathbf{1.1} \end{cases} = 1.1$$

$$\text{当} 1 \le i \le j \le n \text{ 时，} \quad C[i,j] = \min_{i \le k \le j}\{C[i,k-1] + C[k+1,j]\} + \sum_{s=i}^{j} p_s$$

- 最优二叉树举例

键	3	7	9	12
概率	0.1	0.2	0.4	0.3

主表

	0	1	2	3	4
1	0	0.1	**0.4**	**1.1**	
2		0	0.2	**0.8**	**1.4**
3			0	0.4	**1.0**
4				0	0.3
5					0

根表

	0	1	2	3	4
1		1	**2**	**3**	
2			2	**3**	**3**
3				3	**3**
4					4
5					

$$C[2,4] = \min \begin{cases} k=2: & C[2,1] + C[3,4] + \sum_{s=2}^{4} p_s = 0 + 1.0 + 0.9 = 1.9 \\ k=3: & C[2,2] + C[4,4] + \sum_{s=2}^{4} p_s = 0.2 + 0.3 + 0.9 = \mathbf{1.4} \\ k=4: & C[2,3] + C[5,4] + \sum_{s=2}^{4} p_s = 0.8 + 0 + 0.9 = 1.7 \end{cases} = 1.4$$

$$\text{当} 1 \le i \le j \le n \text{ 时，} \quad C[i,j] = \min_{i \le k \le j} \{C[i,k-1] + C[k+1,j]\} + \sum_{s=i}^{j} p_s$$

- 最优二叉树举例

键	3	7	9	12
概率	0.1	0.2	0.4	0.3

主表

	0	1	2	3	4
1	0	0.1	**0.4**	**1.1**	**1.7**
2		0	0.2	**0.8**	**1.4**
3			0	0.4	**1.0**
4				0	0.3
5					0

根表

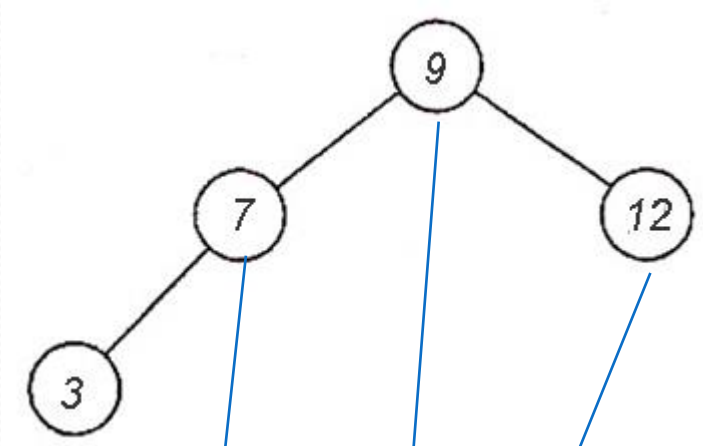
	0	1	2	3	4
1		1	**2**	**3**	**3**
2			2	**3**	**3**
3				3	**3**
4					4
5					

$$C[1,4] = \min \begin{cases} k=1: & C[1,0] + C[2,4] + \sum_{s=1}^{s=4} p_s = 0 + 1.4 + 1.0 = 2.4 \\ k=2: & C[1,1] + C[3,4] + \sum_{s=1}^{4} p_s = 0.1 + 1.0 + 1.0 = 2.1 \\ k=3: & C[1,2] + C[4,4] + \sum_{s=1}^{4} p_s = 0.4 + 0.3 + 1.0 = \mathbf{1.7} \\ k=4: & C[1,3] + C[5,4] + \sum_{s=1}^{4} p_s = 1.1 + 0 + 1.0 = 2.1 \end{cases} = 1.7$$

$$\text{当} 1 \le i \le j \le n \text{ 时，} \quad C[i,j] = \min_{i \le k \le j} \{C[i,k-1] + C[k+1,j]\} + \sum_{s=i}^{j} p_s$$

- 最优二叉树举例

键	3	7	9	12
概率	0.1	0.2	0.4	0.3



主表

	0	1	2	3	4
1	0	0.1	0.4	1.1	1.7
2		0	0.2	0.8	1.4
3			0	0.4	1.0
4				0	0.3
5					0

根表

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

K=2
左:C[1,1]
右:NULL

K=3
左:C[1,2]
右:C[4,4]

8.3 最优二叉查找树

算法 OptimalBST(P[1..n])

```
//用动态规划算法求解最优二叉树
//输入：一个n个键的有序列表的查找概率数组P[1…n]
//输出：成功查找平均比较次数，根表R
for i =1 to n do
    C[i,i-1]=0;C[i,i]=P[i];R[i,i]=i;
C[n+1,n]=0;
for d =1 to n-1 do //对角线计数，记录第n条对角线
    for i =1 to n-d do
        j=i+d; minval=∞;
        for k =i to j do
            if C[i,k-1]+C[k+1,j]<minval
                minval=C[i,k-1]+C[k+1,j];kmin=k;
        R[i,j]=kmin;    sum=P[i];
        for s=i+1 to j do sum=sum+P[s];
        C[i,j]=minval+sum;
return C[1,n],R;
```

分析：OptimalBST算法的时间效率是$\Theta(n^3)$

主表

i	0	1	j=2	j=3	j=4	
1	0	0.1	0.4	1.1	1.7	d=3
2		0	0.2	0.8	1.4	d=2
3			0	0.4	1.0	d=1
4				0	0.3	
5					0	

根表

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

$$\text{当} 1 \le i \le j \le n \text{ 时，} \quad C[i,j] = \min_{i \le k \le j} \{C[i,k-1] + C[k+1,j]\} + \sum_{s=i}^{j} p_s$$

8.3 最优二叉查找树

a. OptimalBST算法的时间效率是**Θ(n³)? Why?**

$$\sum_{d=1}^{n-1}\sum_{i=1}^{n-d}\sum_{k=i}^{i+d} 1 = \sum_{d=1}^{n-1}\sum_{i=1}^{n-d}(i+d-i+1) = \sum_{d=1}^{n-1}\sum_{i=1}^{n-d}(d+1)$$

$$= \sum_{d=1}^{n-1}(d+1)(n-d) = \sum_{d=1}^{n-1}(dn+n-d^2-d)$$

$$= \sum_{d=1}^{n-1} nd + \sum_{d=1}^{n-1} n - \sum_{d=1}^{n-1} d^2 - \sum_{d=1}^{n-1} d$$

$$= n\frac{(n-1)n}{2} + n(n-1) - \frac{(n-1)n(2n-1)}{6} - \frac{(n-1)n}{2}$$

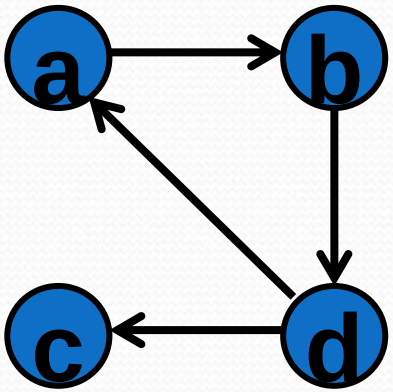
$$= \frac{1}{2}n^3 - \frac{2}{6}n^3 + O(n^2) \in \Theta(n^3).$$

8.4 Warshall算法和Floyd算法

- Warshall算法用于计算有向图的传递闭包。
  - 问题：求对于给定图的顶点之间是否存在着任意长度的有向路径。

邻接矩阵

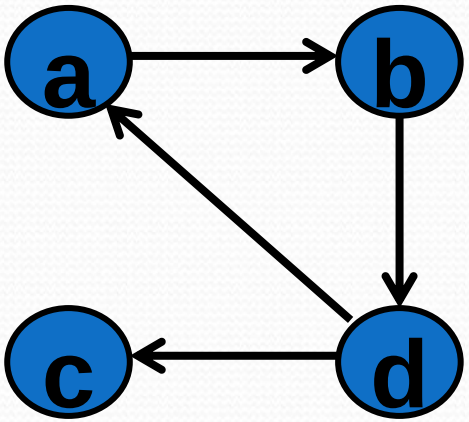
- 复习：一个有向图的邻接矩阵A={a_{ij}}是一个布尔矩阵
  - 当且仅当从第i个顶点到第j个顶点之间有一条有向边时，矩阵第i行第j列的元素为1；否则为0



$$A = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

传递闭包

- 一个n个顶点有向图的**传递闭包**可以定义为一个**n阶布尔矩阵**T={t_{ij}}：
  - 如果从第i个顶点到第j个顶点之间存在一条有效的有向路径（即长度大于0的有向路径），那么t_{ij}为1，否则为0



$$A=\begin{array}{c} \\ a \\ b \\ c \\ d \end{array}\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

$$T=\begin{array}{c} \\ a \\ b \\ c \\ d \end{array}\begin{array}{c}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}\end{array}\begin{array}{cccc} a & b & c & d \end{array}$$

生成传递闭包的算法

- 可以在深度优先查找和广度优先查找的帮助下生成有向图的传递闭包：
  - 从第i个顶点开始，无论采用哪种遍历方法，都能够得到通过第i个顶点访问到所有顶点的信息。因此传递闭包的第i行的相应位置为1。
  - 以每个顶点为起始点做一次这样的遍历就生成了整个图的传递闭包。
- **这样将对同一有向图遍历多次**，故寻找更高效的算法——Warshall算法

Warshall算法

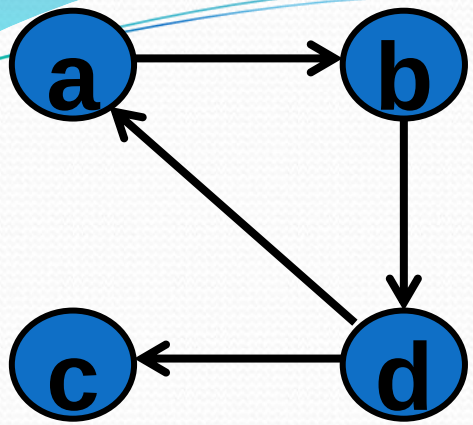
- 通过一系列n阶布尔矩阵来构造一个给定的n个顶点有向图的传递闭包。

$$R^{(0)}, \ldots, R^{(k-1)}, R^{(k)}, \ldots, R^{(n)}$$

  - 每一个这种矩阵都提供有向图中有向路径的特定信息。

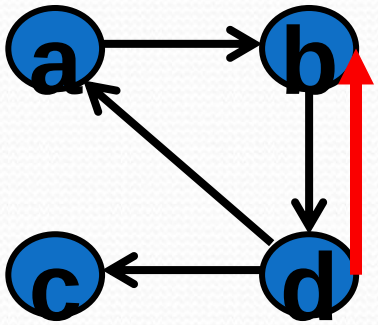
Warshall算法

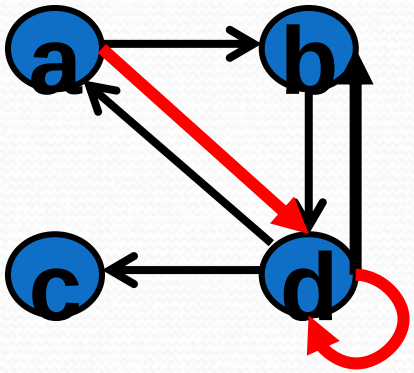
- **当且仅当**从第i个顶点到第j个顶点之间存在一条有向路径（长度大于0），并且路径的每一个中间顶点的编号不大于k时，矩阵$R^{(k)}$的第i行第j列的元素$r_{ij}^{(k)}$ 的值等于1
  - 从矩阵$R^{(0)}$开始，这个矩阵不允许它的路径包含任何中间顶点，所以$R^{(0)}$就是有向图的邻接矩阵
  - $R^{(1)}$包含允许使用第一个顶点作为中间顶点的路径信息

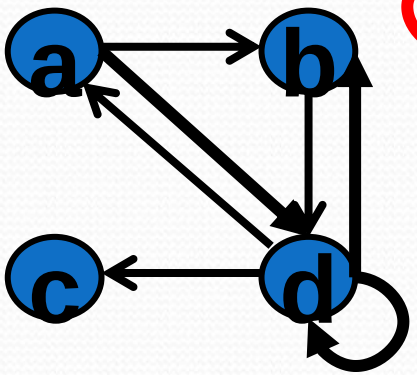


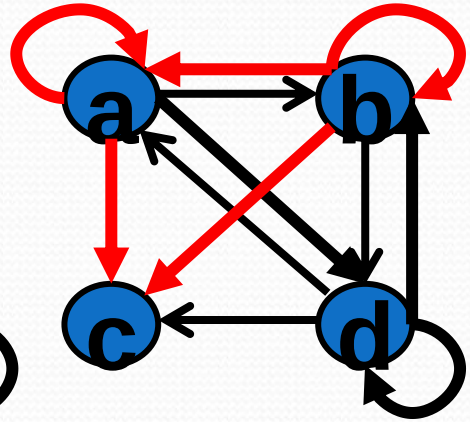
$$A = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{bmatrix} a & b & c & d \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

$$T = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{bmatrix} a & b & c & d \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$









$$R^{(1)} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix} \quad R^{(2)} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad R^{(3)} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad R^{(4)} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Warshall算法

$$R^{(0)}, \ldots, R^{(k-1)}, R^{(k)}, \ldots, R^{(n)}$$

- 具体地说，序列中的每个后继矩阵相对于它的前驱来说，都允许增加一个顶点作为其路径上的顶点。
- 序列中的最后一个矩阵，反映了能够以有向图的所有n个顶点作为中间顶点的路径，因此就是有向图的传递闭包。

➢ **算法的中心思想是：**

任何$R^{(k)}$中的所有元素都可以通过它在序列中的直接前驱$R^{(k-1)}$计算得到。

- 把矩阵$R^{(k)}$中第i行第j列的元素$r_{ij}^{(k)}$置为1。这意味着存在一条从第i个顶点v_i到第j个顶点v_j的路径，路径中每个中间顶点的编号都≤k，**这种路径可能有两种情况**：
  - 中间顶点的编号都小于k：即中间顶点列表中不包含第k个顶点，那么这条从v_i到v_j的路径中的顶点编号也不大于k-1，所以$r_{ij}^{(k-1)}=1$。
  - 路径的中间顶点包含第k个顶点v_k ，假定v_k只出现一次，那么路径可以改写成下面这种形式：

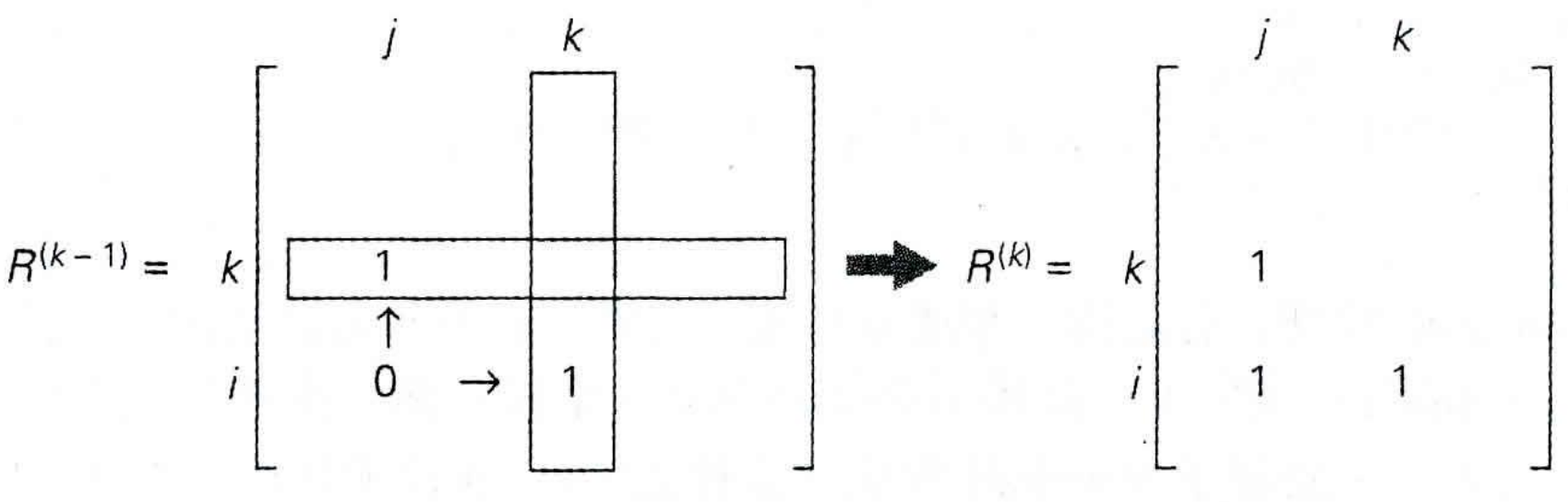
    $v_i \rightarrow$ (编号≤k-1的顶点)$\rightarrow v_k \rightarrow$(编号≤k-1的顶点) $\rightarrow v_j$
  - 这个表现形式的第一部分意味着存在一条从v_i到v_k的路径，因此$r_{ik}^{(k-1)}=1$；而第二部分意味着存在一条从v_k到v_j的路径，$r_{kj}^{(k-1)}=1$。
- 所以对于如何从$R^{(k-1)}$的元素中生成$R^{(k)}$的元素有：

$$r_{ij}^{(k)} = r_{ij}^{(k-1)} \;||\; (r_{ik}^{(k-1)} \&\& r_{kj}^{(k-1)})$$

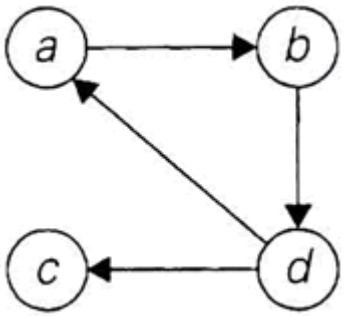
Warshall算法思想

- 递推关系式

$$r_{ij}^{(k)} = \begin{cases} r_{ij}^{(k-1)} & , k \in C \\ r_{ik}^{(k-1)} \text{ 且 } r_{kj}^{(k-1)} & , k \in C \end{cases}$$



$$r_{ij}^{(k)} = \begin{cases} r_{ij}^{(k-1)} & , k \in C \\ r_{ik}^{(k-1)} \;\&\&\; r_{kj}^{(k-1)} & , k \in C \end{cases}$$



$R^{(0)} =$

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	0	1	0

$R^{(1)} =$

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	**1**	1	0

$R^{(2)} =$

	a	b	c	d
a	0	1	0	**1**
b	0	0	0	1
c	0	0	0	0
d	1	1	1	**1**

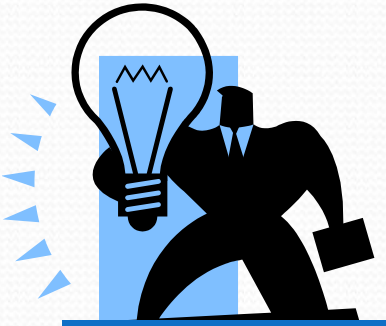
$R^{(3)} =$

	a	b	c	d
a	0	1	0	1
b	0	0	0	1
c	0	0	0	0
d	1	1	1	1

$R^{(4)} =$

	a	b	c	d
a	**1**	1	**1**	1
b	**1**	**1**	**1**	1
c	0	0	0	0
d	1	1	1	1

Warshall算法



Warshall算法的时间效率是$O(n^3)$

```
算法  Warshall(A[1..n,1..n])
//实现计算传递闭包的Warshall算法
//输入：包括n个节点有向图的邻接矩阵
//输出：该有向图的传递闭包
R(0)=A
for(k=1;k<=n;k++)
   for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
          R(k)[i,j]=R(k-1)[i,j] or (R(k-1)[i,k] and R(k-1)[k,j])
return  R(n)
```

- 应用Warshall算法求下列邻接矩阵的传递闭包。

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$R^{(0)} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad R^{(1)} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

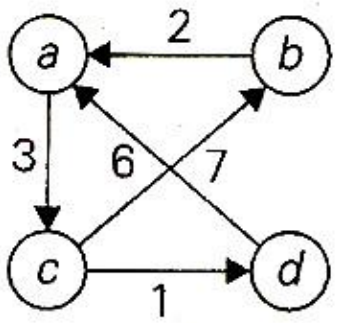
$$R^{(2)} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad R^{(3)} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$R^{(4)} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} = T$$

Floyd算法

the All-Pairs Shortest-path Problem

- 完全最短路径问题要求找出一个有n个节点的加权连通图中每个节点到其他所有节点之间的最短距离。



(a)

$$W = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

(b)

$$D = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 7 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix} \end{array}$$

(c)

- **使用类似于Washall算法的方法来生成这个最短距离矩阵，这就是Floyd算法。**

Floyd算法的思想

- Floyd算法通过一系列n阶矩阵来计算一个n节点加权图的最短距离矩阵。

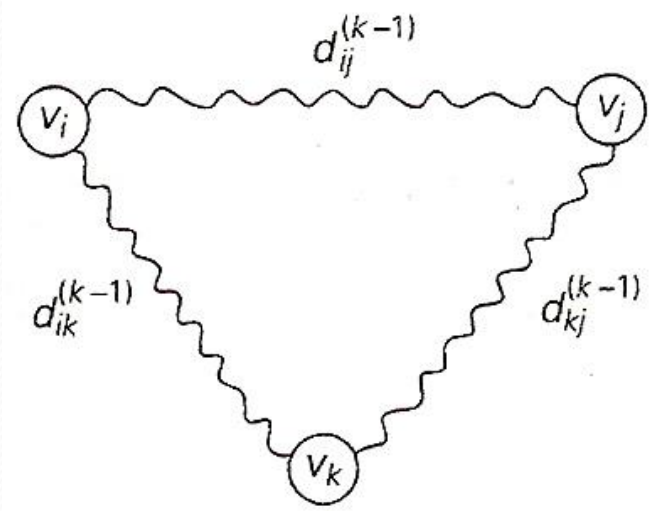
$D^{(0)},\ldots, D^{(k-1)}, D^{(k)},\ldots, D^{(n)}$

$D^{(k)}$:矩阵中任意一对节点间的最短路径值，路径上最大编号节点不大于k

两个条件：

$v_i \rightarrow$(节点编号都不大于k的中间节点集)$\rightarrow v_j$

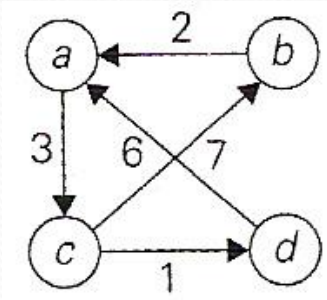
$v_i \rightarrow$(编号≤k-1节点集) $\rightarrow k \rightarrow$(编号≤$k$-1节点集) $\rightarrow v_j$



当$k\geq 1$, $d_{ij}^{(0)}=w_{ij}$时

$d_{ij}^{(k)}=\min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)}+d_{kj}^{(k-1)}\}$

Floyd算法实例



$D^{(0)}=$

	a	b	c	d
a	0	∞	3	∞
b	2	0	∞	∞
c	∞	7	0	1
d	6	∞	∞	0

$D^{(1)}=$

	a	b	c	d
a	0	∞	3	∞
b	2	0	**5**	∞
c	∞	7	0	1
d	6	∞	**9**	0

$D^{(2)}=$

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	**9**	7	0	1
d	6	∞	9	0

$D^{(3)}=$

	a	b	c	d
a	0	**10**	3	**4**
b	2	0	5	**6**
c	9	7	0	1
d	6	**16**	9	0

$D^{(4)}=$

	a	b	c	d
a	0	10	3	4
b	2	0	5	6
c	**7**	7	0	1
d	6	16	9	0

$$d_{ij}^{(k)}=\min\{d_{ij}^{(k-1)},d_{ik}^{(k-1)}+d_{kj}^{(k-1)}\}$$

Floyd算法

```
算法   Floyd(W[1..n,1..n])
//实现计算完全最短路径的Floyd算法
//输入：不包含长度为负的回路的图的权重矩阵W
//输出：包含最短路径长度的距离矩阵
D=W
for(k=1;k<=n;k++)
   for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
          D[i,j]=min{D[i,j],D[I,k]+D[k,j]}
return  D
```