武漢大學 WUHAN UNIVERSITY

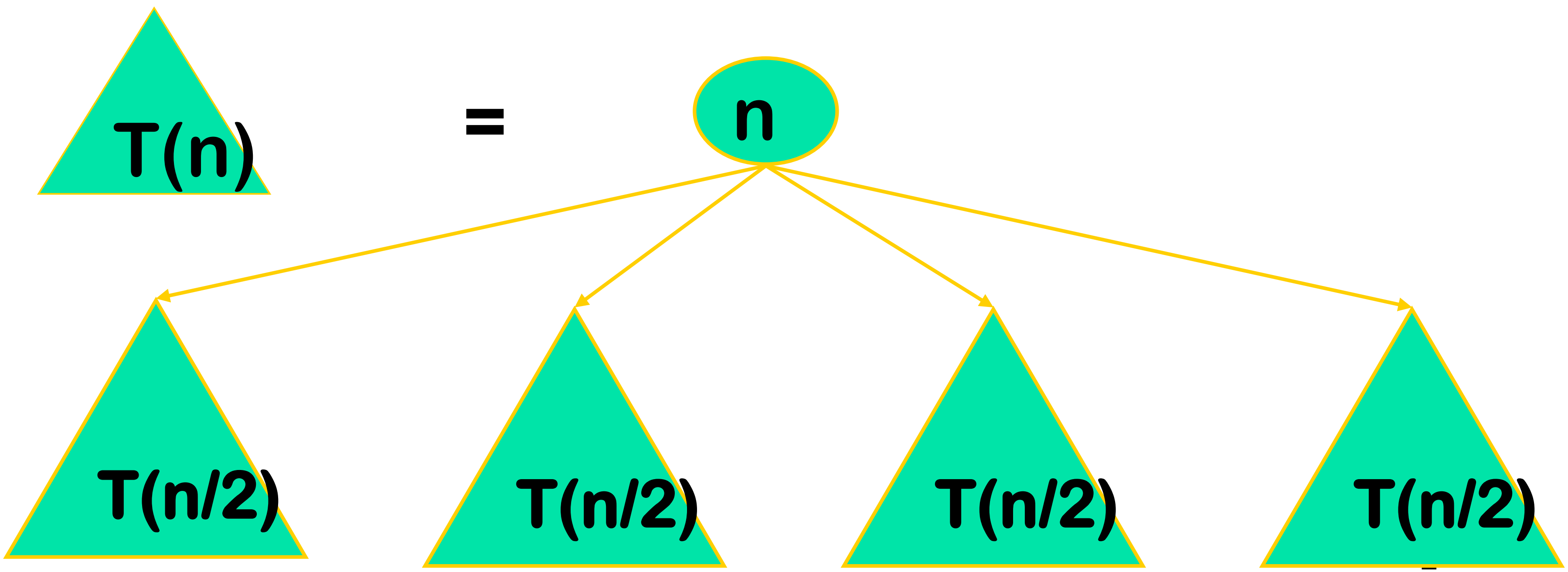
动态规划

算法设计与分析

武汉大学
国家网络安全学院
李雨晴

动态规划的基本思想

- 动态规划算法与分治法类似，其基本思想也是将待求解问题分解成若干个子问题，先求解子问题，再从子问题的解得到原问题的解

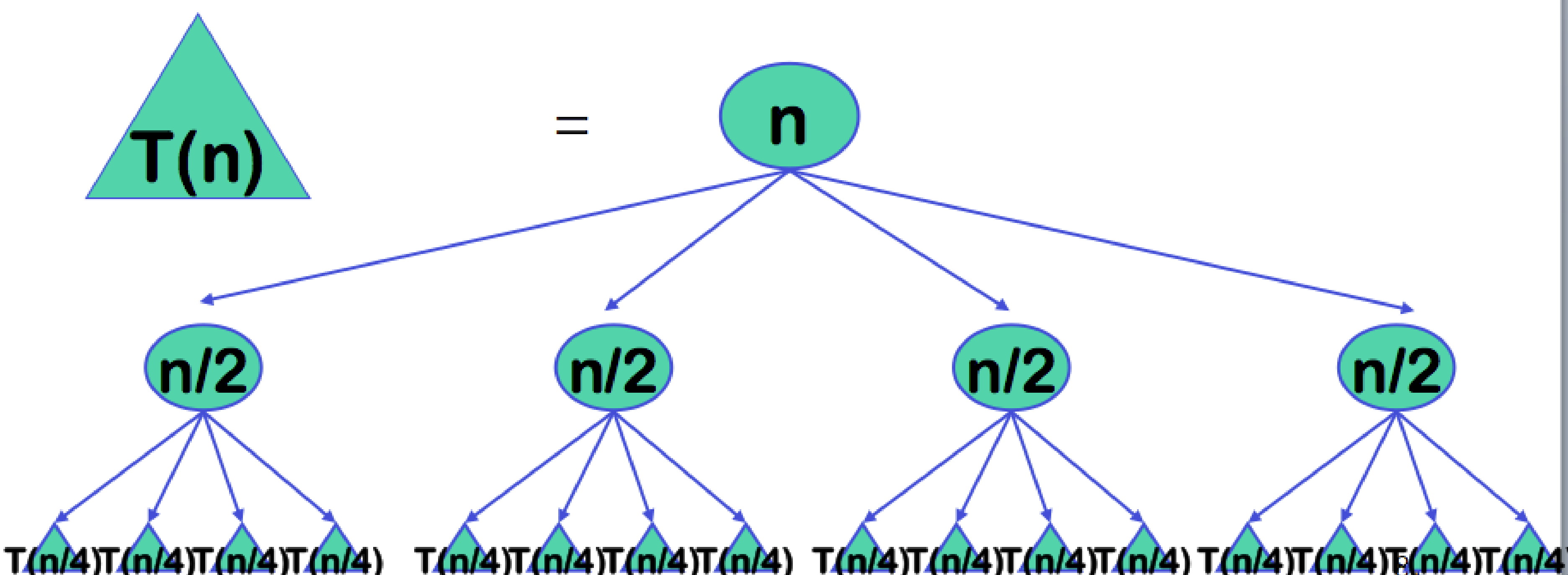


动态规划的基本思想

- 和分治法的区别
  - 主要用于优化问题（求最优解）
  - 子问题并不独立，即子问题是可能重复的
    - 重复的子问题，不需要重复计算

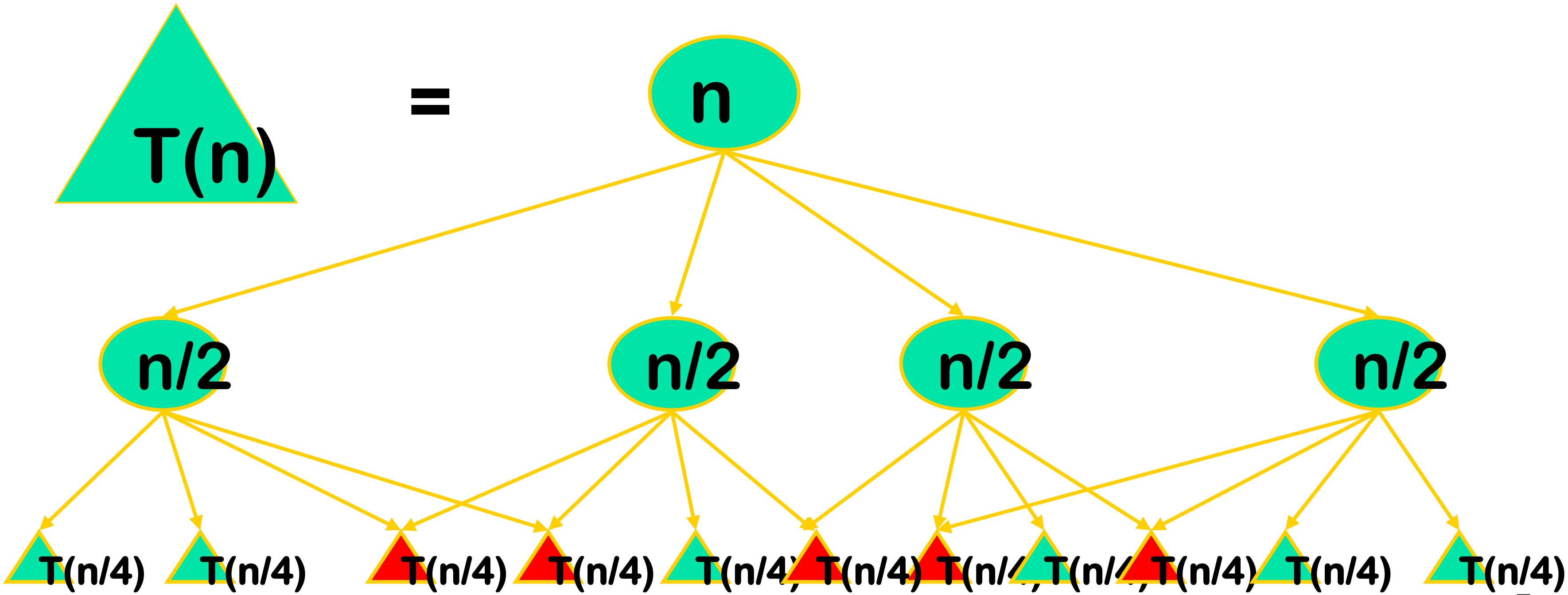
动态规划的基本思想

- 但是经分解得到的子问题往往不是互相独立的。不同子问题的数目常常只有多项式量级。在用分治法求解时，有些子问题被重复计算了许多次



动态规划的基本思想

- 如果能够保存已解决的子问题的答案，而在需要时再找出已求得的答案，就可以避免大量重复计算，从而得到多项式时间算法



动态规划的基本思想

- 动态规划的实质是分治和消除冗余
  - 将问题实例分解为一系列更小的、相似的子问题
  - 求解每个子问题仅一次，并存储子问题的解以避免计算重复的子问题
  - 自底向上地计算
  - 适用范围：一类可分解为多个相关子问题的优化问题，且子问题的解被重复使用

动态规划原理

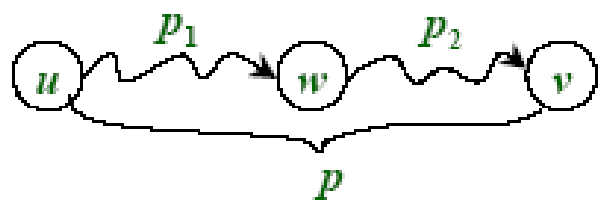
- 具备两个要素
  - 最优子结构
  - 子问题重叠
- 最优子结构
  - 最优解一定包含子问题的最优解
  - 缩小子问题集合，只需那些优化问题中包含的子问题，降低实现复杂性
- 子问题重叠
  - 很多子问题的解将被多次使用
  - 子问题空间必须足够小

动态规划原理：最优子结构

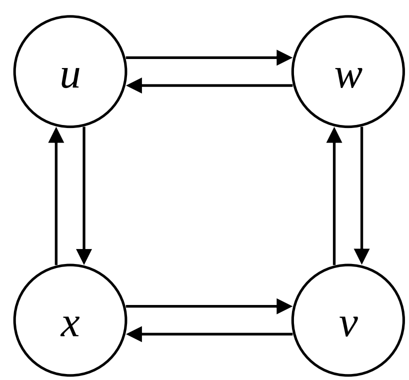
- 在使用动态规划算法来求解问题时，如果问题不具有最优子结构性质，而想当然的认为有，就会适得其反，达不到求解的效果。
- 给定一个有向图$G=(V,E)$和顶点$u,v\in V$，考虑下面两个问题
  - **不带权的最短路径问题:**寻找一条从u到v含有最少边数的路径。这样的路径必须是简单路径即序列中顶点不重复出现的路径，要不然从这个路径中删去一个环会产生一条含有更少边数的路径。
  - **不带权的最长简单路径:**寻找一条从u到v含有最多边数的路径。这条路径必须是简单路径，要不然可以多次地绕着一个环遍历，从而得到一条含有任意多边数的路径。

动态规划原理：最优子结构

- 对于不带权的最短路径问题，问题具有最优子结构性质。解一个子问题跟解另一个子问题是独立的



- 对于不带权的最长路径问题，不带权的最长路径问题则不具有最优子结构性质。解一个子问题跟解另一个子问题是不独立的



动态规划的基本步骤

- 找出最优解的性质，并刻画其结构特征
- 递归地定义最优值
- 以自底向上的方式计算出最优值
- 根据计算最优值时得到的信息，构造最优解

- 步骤①-③是动态规划的基本步骤。如果只需要求出最优值的情形，步骤④可以省略。
- 若需要求出问题的一个最优解，则必须执行步骤④，步骤③中记录的信息是构造最优解的基础；

动态规划实例

- 最长公共子序列
- 矩阵链相乘
- 所有点对最短路径问题
- 背包问题
- 最优二叉搜索树

矩阵链相乘

- 给定n个连乘的矩阵$A_1 \cdot A_2 \ldots A_{n-1} \cdot A_n$，问：所需要的最小乘法次数(最优值)是多少次？对应此最小乘法次数，矩阵是按照什么结合方式相乘(最优解)的？

$$(A)_{p\times q} \cdot (B)_{q\times r}$$　　所需要的乘法次数为：$p\times q\times r$

观察结论：多个矩阵连乘时，相乘的结合方式不同，所需要的乘法次数大不相同。

矩阵链相乘

- 穷举（蛮力）法
- 动态规划*

将矩阵连乘积 $A_i A_{i+1} \ldots A_j$ 简记为$A[i:j]$，这里$i \le j$

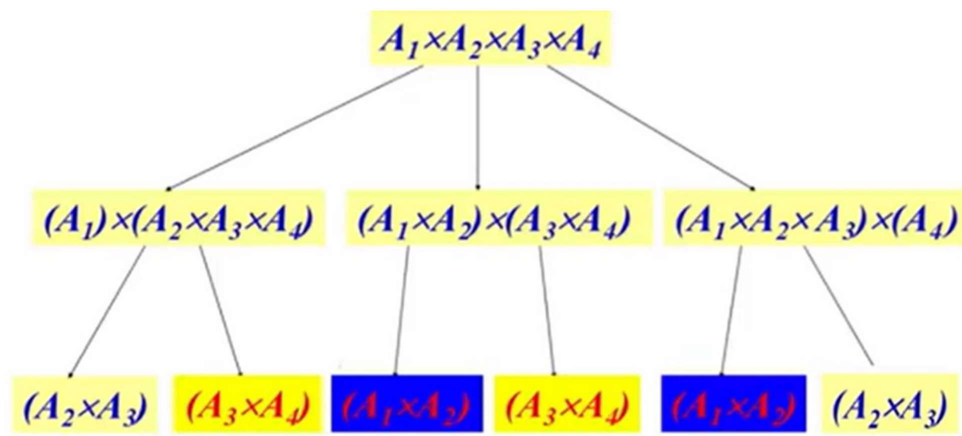
数组$r[1,2,\ldots,n+1]$中存储每个矩阵的行数和列数

1.分析最优解的结构

- 最优子结构
  - 计算$A[i:j]$的最优次序所包含的计算矩阵子链 $A[i:k-1]$和$A[k:j]$的次序也是最优的
  - 证明：反证法，假设子问题$A[i:k-1]$和$A[k:j]$的计算次序不是最优的，则说明对问题$A[i:k-1]$和$A[k:j]$分别有一种更优的计算次序，分别使得计算$A[i:k-1]$和$A[k:j]$的乘法次数更少，将该种计算次序分别替换原来对子问题$A[i:k-1]$和$A[k:j]$的计算次序，则$A[i:j]$可得到一种新的计算次序，它比原来最优计算次序的乘法次数更少，显然矛盾。
  - 具有最优子结构：问题的最优解包含子问题最优解

1.分析最优解的结构

- 重叠子问题
  - 具有子问题重叠性：很多子问题的解将被多次使用



2.建立递归关系

- 最优值的递归方程
  - 计算$A[i:j]$，$1\le i\le j\le n$，所需要的最少数乘次数$C[i,j]$，则原问题的最优值为$C[1,n]$
  - 当$i=j$时，$A[i:j]=A_i$，因此，$C[i,i]=0$，$i=1,2,\ldots,n$
  - 当$i<j$时，设k为最优断开点

$$C[i,j] = \underbrace{C[i,k-1]}_{\text{计算}A[i:k-1]\text{需要的最少乘法次数}} + \underbrace{C[k,j]}_{\text{计算}A[k:j]\text{需要的最少乘法次数}} + \underbrace{r_i r_k r_{j+1}}_{\text{计算}A[i:k-1]\text{和}A[k:j]\text{相乘需要的最少乘法次数}}$$

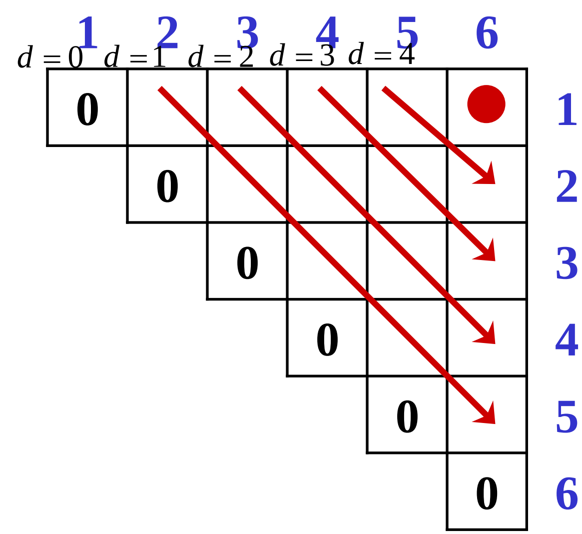
  - 可以递归地定义$C[i,j]$为：k的位置只有 $j-i$ 种可能

$$C[i,j] = \begin{cases} 0 & \text{if } i = j, \\ \min\limits_{i<k\le j}\{C[i,k-1] + C[k,j] + r_i r_k r_{j+1}\} & \text{if } i < j. \end{cases}$$

3.自底向上计算最优值

- 对于1≤*i*≤*j*≤*n*不同的有序对*(i,j)*对应于不同的子问题，不同子问题的个数最多只有 $\binom{n}{2} + n = \Theta(n^2)$

- 计算所有的 ***C[i,j]***:
  --Start by setting ***C[i,i]***=0 for ***i*** = 1,…,n.
  --Then compute ***C***[1,2], ***C***[2,3],…,***C***[*n-1*,*n*].
  --Then ***C***[1,3], ***C***[2,4],…,***C***[*n-2*,*n*],…
  --… so on till we can compute ***C***[1,*n*].



$$C[i,j] = \begin{cases} 0 & \text{if } i = j, \\ \min\limits_{i<k\le j}\{C[i,k-1] + C[k,j] + r_i r_k r_{j+1}\} & \text{if } i < j. \end{cases}$$

矩阵链相乘

输入： r[1..n+1]，表示n个矩阵规模的n+1个整数.
输出： n个矩阵连乘的最小乘法次数.

1. for i←1 to n {填充对角线d_0}
2. C[i,i] ←0
3. end for
4. for d←1 to n-1 {填充对角线d_1到d_{n-1}}
5. for i←1 to n-d {填充对角线d_i的每个项目}
6. j←i+d {该对角线上j,i满足的关系}
7. C[i,j] ←∞
8. for k←i +1 to j
9. C[i,j] ←min{ C[i,j], C[i,k-1]+ C[k,j]+ $r_i \times r_k \times r_{j+1}$}
10. end for
11. end for
12. end for
13. return C[1,n]

4. 构造最优解

- 保存对每个子问题的最佳划分点
- $s[i, j]$ = 通过把乘积$A[i{:}j]$从A_{k-1}和A_k之间分开得到最优计算次序的划分点k的值

$$s[1,n]=3 \Rightarrow A[1{:}6] = A[1{:}2]\ A[3{:}6]$$

$$s[1,3]=2 \Rightarrow A[1{:}3] = A[1{:}1]\ A[2{:}3]$$

$$s[4,6]=6 \Rightarrow A[4{:}6] = A[4{:}5]\ A[6{:}6]$$

$$C[i,j] = \begin{cases} 0 & \text{if } i = j, \\ \min_{i<k\le j}\{C[i,k-1] + C[k,j] + r_i r_k r_{j+1}\} & \text{if } i < j. \end{cases}$$

$$(M_1)_{5\times10}\cdot(M_2)_{10\times4}\cdot(M_3)_{4\times6}\cdot(M_4)_{6\times10}\cdot(M_5)_{10\times2} \qquad r_1=5, r_2=10, r_3=4, r_4=6, r_5=10, r_6=2$$

$d=0$ $d=1$ $d=2$ $d=3$ $d=4$

C[1,1]=0 (M_1)	C[1,2]=200 $(M_1 M_2)$	C[1,3]=320	C[1,4]=620	C[1,5]=348
	C[2,2]=0 (M_2)	C[2,3]=240 $(M_2 M_3)$	C[2,4]=640 $(M_2)(M_3M_4)$	C[2,5]=248
		C[3,3]=0 (M_3)	C[3,4]=240 (M_3M_4)	C[3,5]=168
			C[4,4]=0 (M_4)	C[4,5]=120 $(M_4 M_5)$
				C[5,5]=0 (M_5)

$$C[2,4] = \min_{2<k\le4}\{C[2,k-1] + C[k,4] + r_2\cdot r_k\cdot r_{4+1}\}$$

$$\left.\begin{array}{l} k=3 \rightarrow C[2,4]=C[2,2]+C[3,4]+r_2\cdot r_3\cdot r_{4+1} = 0+240+10\cdot4\cdot10 = 640 \rightarrow (M_2)\cdot(M_3\cdot M_4) \\ k=4 \rightarrow C[2,4]=C[2,3]+C[4,4]+r_2\cdot r_4\cdot r_{4+1} = 240+0+10\cdot6\cdot10 = 840 \rightarrow (M_2\cdot M_3)\cdot(M_4) \end{array}\right\} \min$$

算法复杂度

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

$$=c\left(\sum_{i=1}^{n-1}1+\sum_{i=1}^{n-2}2+\sum_{i=1}^{n-3}3+\ldots+\sum_{i=1}^{n-(n-1)}(n-1)\right)$$

$$=c\left((n-1)\cdot 1+(n-2)\cdot 2+(n-3)\cdot 3+\ldots+(n-(n-1))\cdot(n-1)\right)$$

$$=c\left(n\cdot 1+n\cdot 2+n\cdot 3+\ldots+n\cdot(n-1)-1\cdot 1-2\cdot 2-(n-1)\cdot(n-1)\right)$$

$$=c\left(n(1+2+3+\ldots+(n-1))-\sum_{k=1}^{n-1}k^2\right)$$

$$=c\left(n\cdot\frac{n(n-1)}{2}-\frac{1}{6}(n-1)n(2n-1)\right)$$

$$=\frac{1}{6}\left(cn^3-cn\right)\qquad=\Theta\left(n^3\right)$$

蛮力方法:

$$T(n)=\Omega\left(\frac{4^n}{n^{1.5}}\right)$$

所有点对最短路径问题

- 问题
  - 设$G=(V,E)$是一个有向图，每条边(i,j)有一个非负长度$l[i,j]$, 如果顶点i到顶点j没有边，则$l[i,j]=\infty$
  - 求所有点对的最短路径（距离）

所有点对最短路径问题

- 分析最优解的结构
- 建立递归关系
  - V 包含了n个顶点，记为 $\{1,2,\ldots,n\}$
  - 假设已经知道(i,j)在不经过$\{k+1,k+2,\ldots,n\}$个顶点时的距离，那么可否推出在不经过$\{k, k+1,\ldots,n\}$个顶点时,顶点$\{i,j\}$之间的距离
  - 设$d^k_{i,j}$为$\{i,j\}$在不经过$\{k+1,k+2,\ldots,n\}$中任何顶点时的距离

所有点对最短路径问题

- 建立递归关系
  - $d_{i,j}^{k}$与$d_{i,j}^{k-1}$的区别在于？是否允许经过顶点k
  - 得出最优值的递归方程

$$d_{i,j}^{k} = \begin{cases} l[i,j] & 若\ k=0 \\ \min\{d_{i,j}^{k-1}, d_{i,k}^{k-1} + d_{k,j}^{k-1}\} & 若\ 1 \leqslant k \leqslant n \end{cases}$$

第二个式子表示$\{i,j\}$在不经过$\{k+1,k+2,\ldots,n\}$中任何顶点时的距离要么不经过顶点k，要么经过顶点k

所有点对最短路径问题

$$D_0 \rightarrow D_1 \rightarrow D_2 \rightarrow \ldots\ldots \rightarrow D_n$$

- 自底向上计算最优值
  - 最终计算结果的存储： $n+1$个$n \times n$数组D
  - 最低层（递归式中边界条件）的d^0（用D_0表示）

$$D_0[i,i]=0,\ D_0[i,j]=l[i,j] \qquad \text{如果}(i,j)\text{有边}$$

$$D_0[i,j]=\infty \qquad \text{如果}(i,j)\text{无边}$$

  - 按照递归式得出第k层的D

$$D_k[i,j]=\min\{D_{k-1}[i,j],D_{k-1}[i,k]+D_{k-1}[k,j]\};$$

所有点对最短路径问题

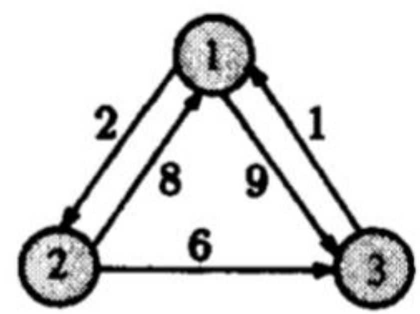
$$D_k[i,j]=\min\{D_{k-1}[i,j],D_{k-1}[i,k]+D_{k-1}[k,j]\}$$

此递归式在计算的过程中需要保留2个$n\times n$矩阵（D_{k-1}和D_k）

考察：对如右图所示进行最短距离计算

$$D_0=\begin{bmatrix}0&2&9\\8&0&6\\1&\infty&0\end{bmatrix}\quad D_1=\begin{bmatrix}0&2&9\\8&0&6\\1&3&0\end{bmatrix}$$

$$D_2=\begin{bmatrix}0&2&8\\8&0&6\\1&3&0\end{bmatrix}\quad D_3=\begin{bmatrix}0&2&8\\7&0&6\\1&3&0\end{bmatrix}$$



一个重要的发现是第k次迭代中第k行和第k列都是不变的，即上述递归式中的$D_{k-1}[i,k]$和$D_{k-1}[k,j]$和$D_k[i,k]$和$D_k[k,j]$是一样的。所以可以在一个矩阵中完成

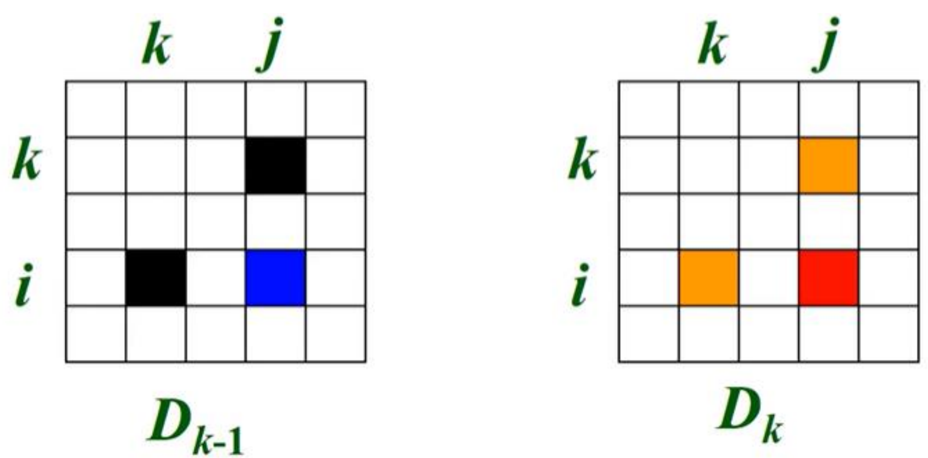
所有点对最短路径问题

一个重要的发现是第k次迭代中第k行和第k列都是不变的，即上述递归式中的$D_{k-1}[i,k]$和$D_{k-1}[k,j]$和$D_k[i,k]$和$D_k[k,j]$是一样的。所以可以在一个矩阵中完成

$$d_{i,k}^{k}=\min\{d_{i,k}^{k-1},\quad d_{i,k}^{k-1}+d_{k,k}^{k-1}\}=d_{i,k}^{k-1}$$

$$d_{k,j}^{k}=\min\{d_{k,j}^{k-1},\quad d_{k,k}^{k-1}+d_{k,j}^{k-1}\}=d_{k,j}^{k-1}$$

$$d_{i,j}^{k}=\min\{d_{i,j}^{k-1},\quad d_{i,k}^{k-1}+d_{k,j}^{k-1}\}，\text{若}\quad 1\le k\le n$$



所有点对最短路径问题

- 算法：FLOYD

输入：$n \times n$ 维矩阵 $l[1\cdots n, 1\cdots n]$，以便对于有向图 $G = (\{1,2,\cdots,n\}, E)$ 中的边 (i,j) 的长度为 $l[i,j]$。

输出：矩阵 D，使得 $D[i,j]$ 等于 i 到 j 的距离。

1. $D \leftarrow l$ {将输入矩阵 l 复制到 D}
2. **for** $k \leftarrow 1$ *to* n
3. **for** $i \leftarrow 1$ *to* n
4. **for** $j \leftarrow 1$ *to* n
5. $D[i,j] = \min\{D[i,j], D[i,k] + D[k,j]\}$
6. **end for**
7. **end for**
8. **end for**

运行时间是 $\Theta(n^3)$

空间复杂性是 $\Theta(n^2)$

0-1背包问题

- 早期的作品可追溯到**<u>1897</u>**年，数学家Tobias Dantzig提出
- **<u>1978</u>**年由Merkle和Hellman提出的
- <u>广泛应用</u>
  - 资源分配
  - 选择投资和投资组合
  - 密码学（生成密钥为Merkle-Hellman）

0-1背包问题

0-1背包问题

1公斤
$6000

2公斤
$10000

3公斤
$9000

5公斤

0-1背包问题

1公斤
$6000

2公斤
$10000

3公斤
$9000

5公斤

Value=6000+10000=16000

0-1背包问题

1公斤
$6000

2公斤
$10000

3公斤
$9000

5公斤

Value=10000+9000=19000

0-1背包问题定义

- 给定n个物品$\{u_1, u_2, \ldots, u_n\}$和一个背包，物品$i$ 的重量为w_i，价值为v_i，已知背包的承重量为C
- 问：在不撑破背包的条件下，选择哪些物品装入背包，得到的**总价值**最大？

0-1背包问题定义

- 0-1背包问题的形式化描述：

  给定$C > 0$，$w_i > 0$, $v_i > 0, 1 \leq i \leq n$, 找出一个n元的0-1 向量$(x_1, x_2, \ldots, x_n)$， $x_i \in \{0,1\}, 1 \leq i \leq n$, 求如下优化问题：

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

$$s.t. \quad \sum_{i=1}^{n} w_i x_i \leq C, \quad x_i \in \{0,1\}, 1 \leq i \leq n$$

1.分析最优解的结构

- 最优子结构
  - 设$(x_1, x_2, \dots, x_n)$是原问题的一个最优解，则$(x_2, \dots, x_n)$是下面相应子问题的一个最优解：

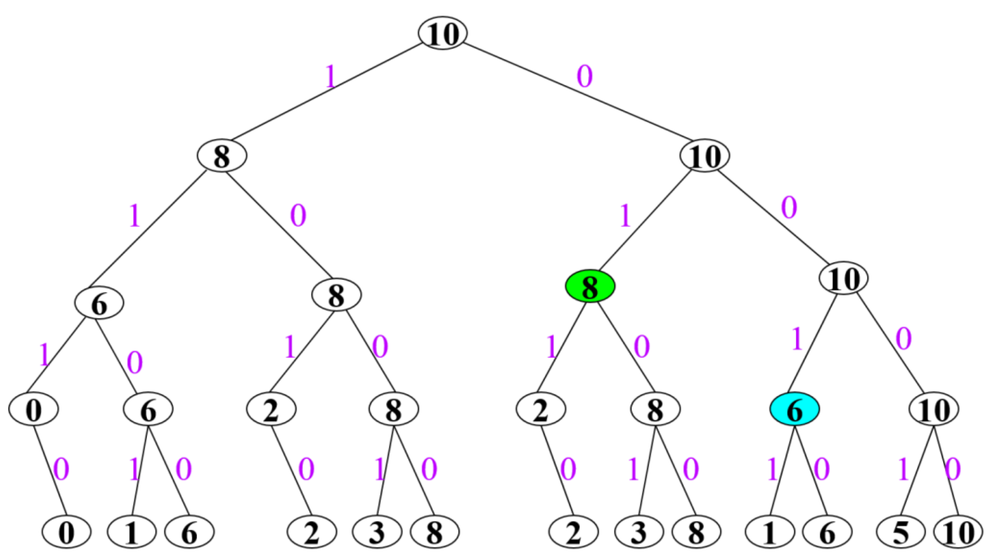
$$\max \sum_{i=2}^{n} v_i x_i$$

$$s.t. \quad \sum_{i=2}^{n} w_i x_i \le c - w_1, x_i \in \{0,1\}, \; 2 \le i \le n$$

1.分析最优解的结构

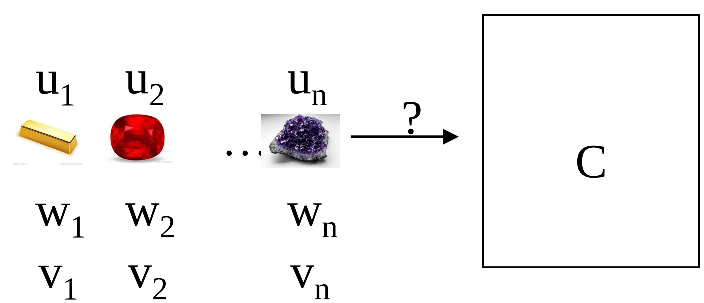
- 重叠子问题

例子: n=5, v=[6,3,5,4,6], w=[2,2,6,5,4], C=10



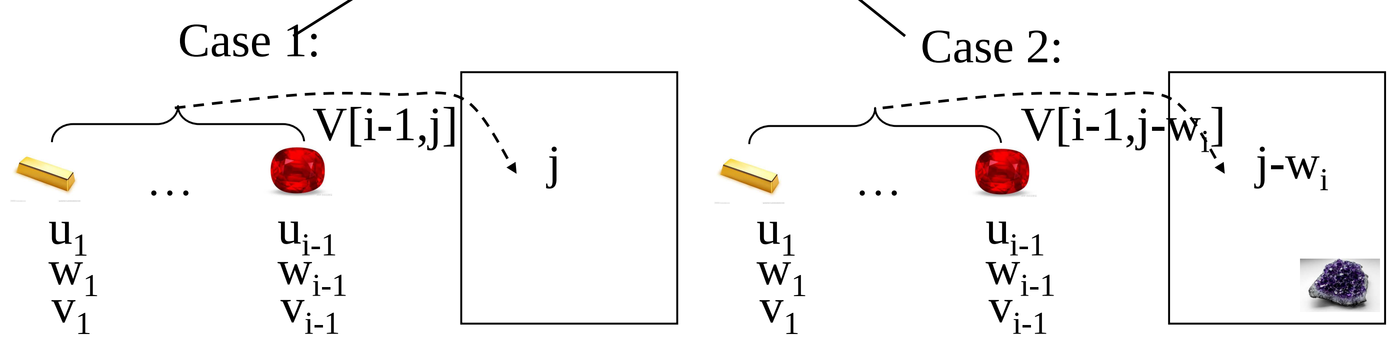
2.建立递归关系

- 设$V[i,j]$表示从前i个物品$\{u_1, u_2, \ldots, u_i\}$中取出一部分装入承重量为$j$的背包所能取得的最大价值
- 当i=0或j=0时，$V[i,j]$=0
- 当i=n,j=C时， $V[n,C]$就是原问题的解



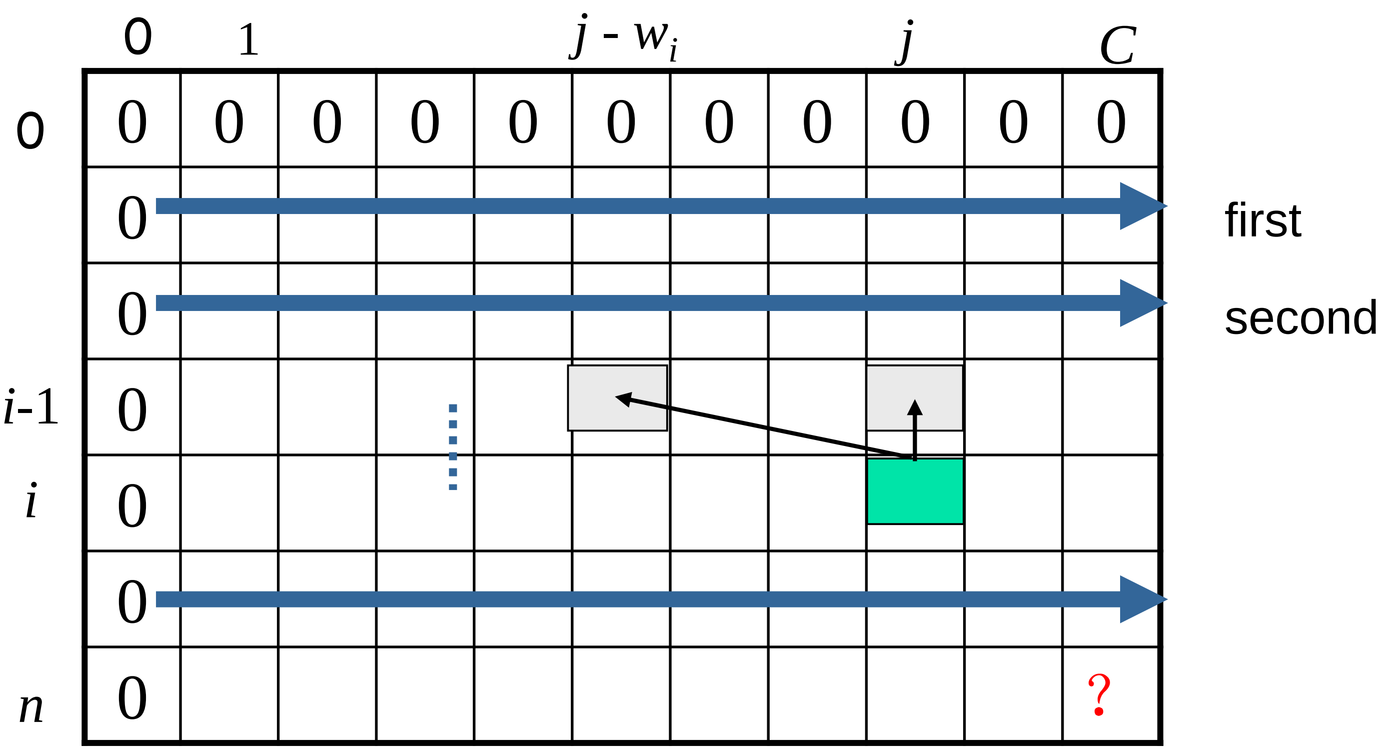
2.建立递归关系

$$V[i,j]=\begin{cases}0 & if\ \ i=0\ \ or\ \ j=0\\ V[i-1,j] & if\ \ j<w_i\\ \max\{V[i-1,j],V[i-1,j-w_i]+v_i\} & if\ \ i>0\ and\ \ j\ge w_i\end{cases}$$



3.自底向上计算最优值

$$V[i,j]=\begin{cases}0 & if\ \ i=0\ \ or\ \ j=0\\ V[i-1,j] & if\ \ j<w_i\\ \max\{V[i-1,j],V[i-1,j-w_i]+v_i\} & if\ \ i>0\ and\ \ j\geq w_i\end{cases}$$



动态规划算法

输入：物品集合$\{u_1,u_2,\ldots,u_n\}$，重量分别为$w_1, w_2,\ldots,w_n$，价值分别为$v_1,v_2,\ldots,v_n$, 承重量为C的背包

输出：背包所能装物品的最大价值

```
1. for i←0 to n
2.   V[i,0] ←0
3. end for
4. for j←0 to C
5.   V[0,j] ←0
6. end for
7. for i←1 to n //前i个物品
8.   for j←1 to C //承重量C与物品重量wi均为整数，故j为整数
9.       V[i,j] ← V[i-1,j]
10.      if wi≤ j then V[i,j] ←max{ V[i,j], V[i-1,j-wi]+vi}
11.      end if
12.  end for
13. end for
14. return V[n,C]
```

背包容量为0

没有物品

自底向上计算

$T(n) = \Theta(nC)$

时间复杂度

- 算法运算量对于输入规模不是多项式的
  - C的值是根据输入规模指数变化的
  - C不可能通过$\log_2 C$的多项式函数控制
- 这种和参数的取值而不是参数的规模成多项式关系的算法，叫做伪多项式时间算法

$$V[i,j]=\begin{cases}0 & if\ \ i=0\ \ or\ \ j=0\\ V[i-1,j] & if\ \ j<w_i\\ \max\{V[i-1,j],V[i-1,j-w_i]+v_i\} & if\ \ i>0\ and\ \ j\geq w_i\end{cases}$$

例子：背包的承重量为C=5；给定3个物品，重量(w)分别为1，2，3；价值(v)依次为6，10，9。背包中最多能装的物品的总价值是多少？

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	6	6	6	6	6
2	0	6	10	16	16	16
3	0	6	10	16	16	19

1公斤
\$6000

2公斤
\$10000

5公斤

3公斤
\$9000

因为$w_3=3<=j=5$;
所以$V[3,5] = \max\{V[3-1,5], V[3-1,5-w_3]+v_3\}$
$= \max\{V[2,5], V[2,2]+9\}= \max\{16,10+9\} =19$

$$V[i,j]=\begin{cases}0 & if\ \ i=0\ \ or\ \ j=0\\ V[i-1,j] & if\ \ j<w_i\\ \max\{V[i-1,j],V[i-1,j-w_i]+v_i\} & if\ \ i>0\ and\ \ j\geq w_i\end{cases}$$

例子：背包的承重量为C=9；给定4个物品，重量(w)分别为2，3，4，5；价值(v)依次为3，4，5，7。问：背包中最多能装的物品的总价值是多少?

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	3	3	3	3	3	3	3	3
2	0	0	3	4	4	7	7	7	7	7
3	0	0	3	4	5	7	8	9	9	12
4	0	0	3	4	5	7	8	10	11	12

因为w_3=4<=j=7;

所以V[3, 7] = max{V[3-1,7], V[3-1,7-w_3]+v_3}

= max{V[2,7], V[2,3]+5}= max{7,4+5} = 9

0-1背包问题

- 有了最优值V[n,C]，如何求最优解呢？

输入：物品集合$\{u_1,u_2,\ldots,u_n\}$，重量分别为$w_1, w_2,\ldots,w_n$，价值分别为$v_1,v_2,\ldots,v_n$, 承重量为C的背包, V[n,C]

输出：装入背包物品的标号和各自价值

```
1. i←n
2. j←C
3. while (i>0&& j>0)
4.     if V[i,j]=V[i-1][j- w_i ]+ v_i
5.          print i, v_i
6.          j←C- w_i
7.      end if
8.      i←i-1
9. end while
```

空间复杂度

$$V[i,j]=\begin{cases}0 & \text{if } i=0 \text{ or } j=0\\ V[i-1,j] & \text{if } j<w_i\\ \max\{V[i-1,j], V[i-1,j-w_i]+v_i\} & \text{if } i>0 \text{ and } j\ge w_i\end{cases}$$

- $s(n)=\Theta(nC)$
- 能只使用一维数组吗？$s(n)=\Theta\ (2C)$

	0	1				j - w_i			j		C
0	0	0	0	0	0	0	0	0	0	0	0
	0										
	0										
i-1	0										
i	0										
	0										
n	0										?

```
KNAPSACK( )
{  W[0,1,2,…., C] = 0;
   Q[0,1,2,…., C] = 0;
   for(i=1; i≤n; i++)
   {    for( j=1; j≤C; j++)
        {  Q[ j] = W[ j];
           if(s_i ≤ j)   Q[ j]= max{Q[ j], W[ j-s_i]+v_i};
        }
        W[1,2,…, C] = Q[1,2,…, C];
   }
   return W[C];
}
```

空间复杂度

$$V[i,j]=\begin{cases}0 & \text{if } i=0 \text{ or } j=0\\ V[i-1,j] & \text{if } j<w_i\\ \max\{V[i-1,j], V[i-1,j-w_i]+v_i\} & \text{if } i>0 \text{ and } j\ge w_i\end{cases}$$

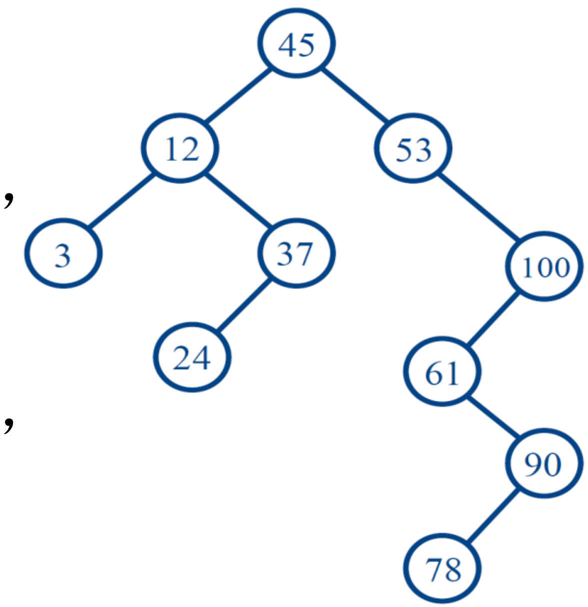
- $s(n)=\Theta(2C)$
- 还能进一步降低空间开销吗？ $s(n)=\Theta(C)$,倒着更新

	0	1				$j-w_i$			j		C
0	0	0	0	0	0	0	0	0	0	0	0
	0										
	0										
i-1	0										
i	0										
	0										
n	0										?

```
KNAPSACK( )
{  W[0,1,2,…., C] = 0;
   for(i = 1; i≤n; i++)
   {   for( j = C; j≥1; j--){
          if(s_i ≤ j)  W[ j]= max{W[ j], W[ j-s_i]+v_i};
       }
   }
   return W[C];
}
```

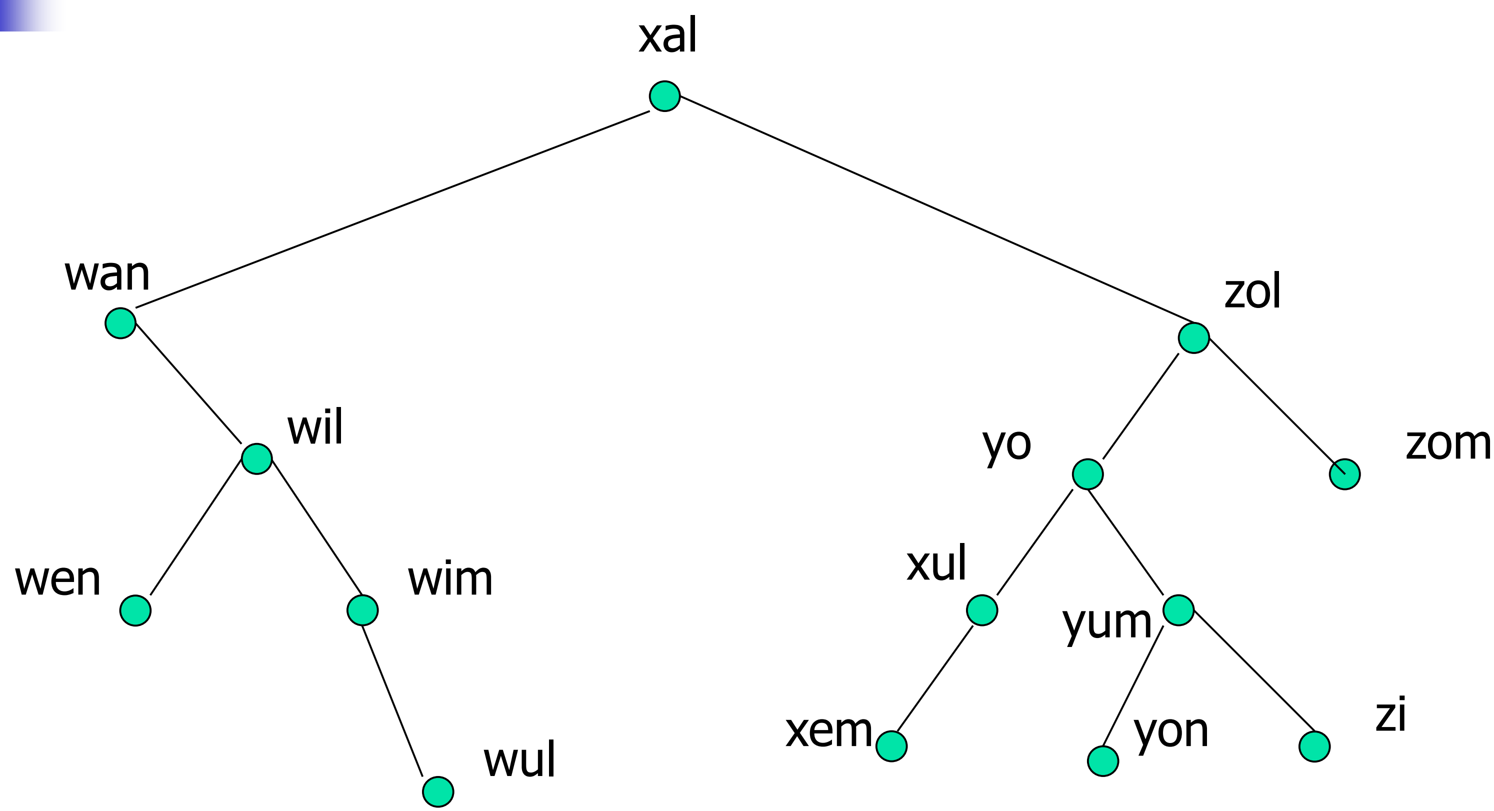
二叉搜索树

- 定义
  - 每个结点作为搜索对象，它的关键字是互不相同的。
  - 对于树上的所有结点，如果它有左子树，那么左子树上所有结点的关键字都小于该结点的关键字。
  - 对于树上的所有结点，如果它有右子树，那么右子树上所有结点的关键字都大于该结点的关键字
  - 它的左、右子树也分别为二叉搜索树



在随机的情况下，二叉搜索树的平均查找长度和logn等数量级的

二叉搜索树



搜索过程：

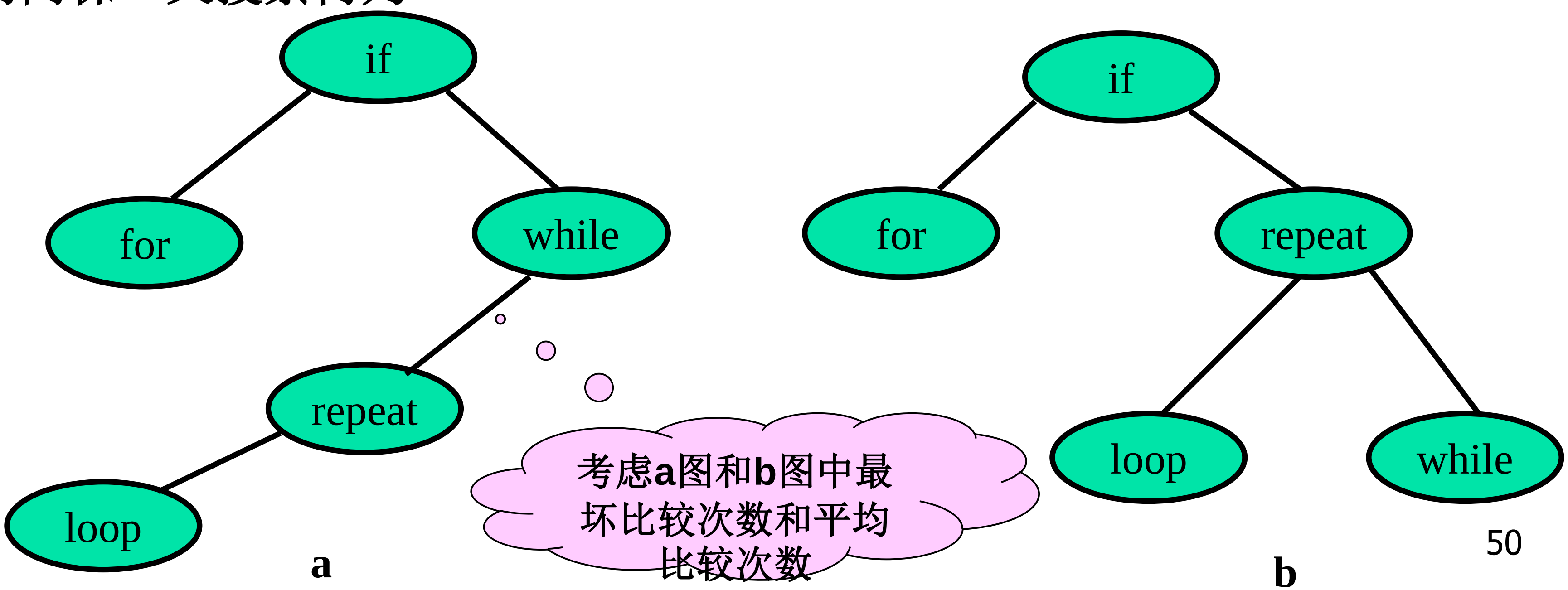
从根结点开始，如果根为空，则搜索不成功；否则使用待搜索值与根结点比较，如果待搜索值等于根结点关键字，则搜索成功返回，如果小于根结点，则向左子树搜索；如果大于根结点，则向右子树搜索。

二叉搜索树

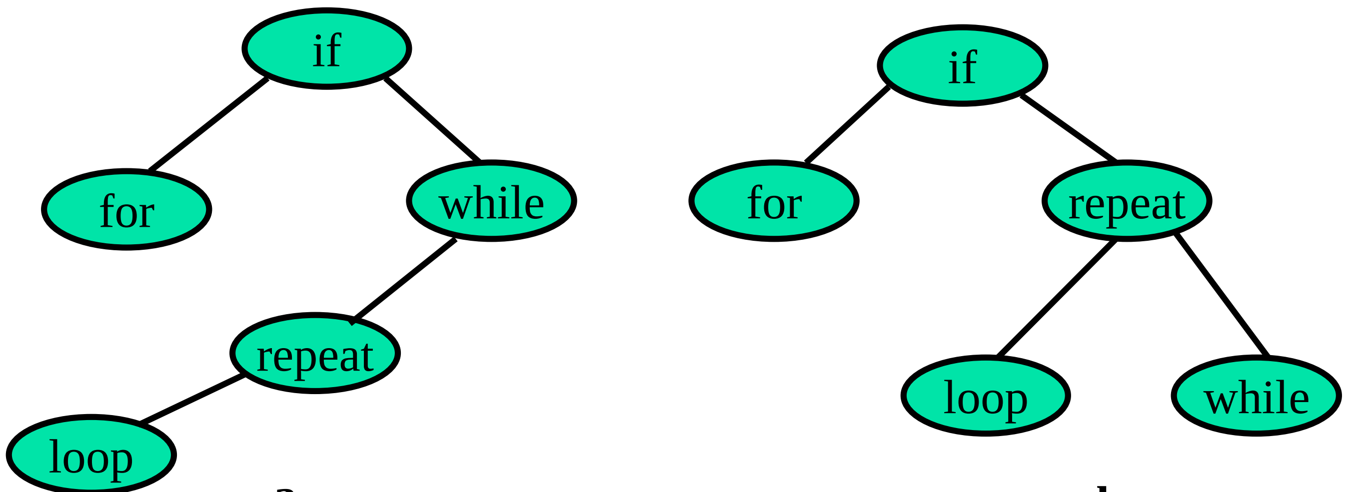
- 对于一个给定的关键字集合，可能有若干不同的二叉搜索树
- 如对保留字的子集

Name:	1	2	3	4	5
	for	if	loop	repeat	while

的两棵二叉搜索树为



二叉搜索树



a　　b

- 构造不同的二叉搜索树就有不同的性能特征。
- 二叉搜索树**a**在最坏情况下找一个标识符需要**4**次比较，而**b**表示的二叉搜索树最坏情况下只需**3**次比较。
- 假设只作成功的检索并且检索每个标识符的概率相同，则两棵二叉搜索树在平均情况下各需要**12/5**和**11/5**次比较。

最优二叉搜索树

- 如何构建一棵二叉搜索树，使得期望搜索代价最少
  - 给定序列 $K = <k_1, k_2, ..., k_n>$， 其中n个关键字互不相同，且都已排好序 $(k_1 < k_2 \quad \cdots < k_n)$, 并且有 $n+1$ 个“虚拟” 关键字 $d_0, d_1, d_2, ..., d_n$,其中d_i表示所有在k_i 和k_{i+1}之间的值
  - 希望从这些关键字中建立一棵二叉搜索树。对于每个关键字 k_i, 搜索概率为 p_i ; 对于每个 d_i, 搜索概率为 q_i 。假设一个搜索的代价为二叉树的节点数, i.e., 在 T 中找到的节点的深度 +1。从而在 T 中所做的一次搜索所花费的预期代价为:

$$E(T) = \sum_{i=1}^{n}(l_{k_i}+1)p_i + \sum_{i=0}^{n}(l_{d_i}+1)q_i$$

  - 给定一个概率集合, 目标是构造一棵二叉搜索树T，使得 $E(T)$ 最小，称这样的树为 最优二叉搜索树

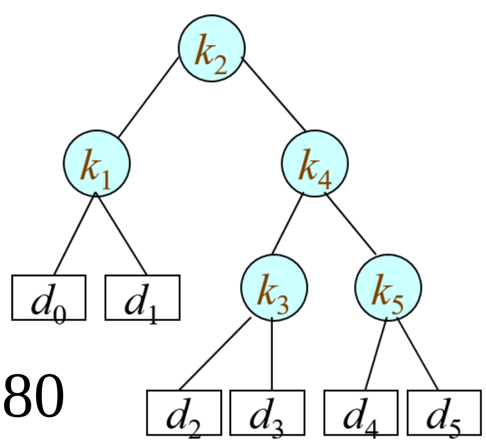
最优二叉搜索树

- 如对下面关键字集合扩充后的两棵二叉搜索树为

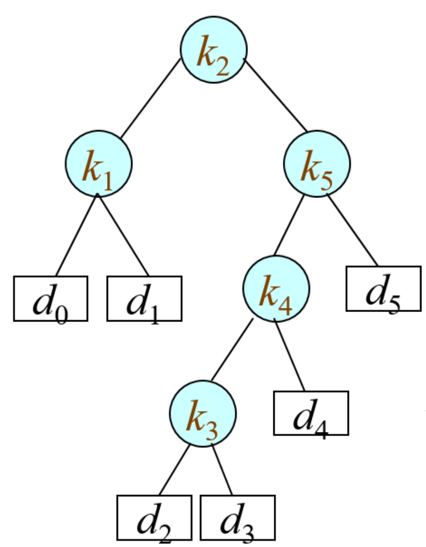
Name: 1 2 3 4 5

for if loop repeat while

i	0	1	2	3	4	5
p_i		0.15	0.1	0.05	0.1	0.2
q_i	0.05	0.1	0.05	0.05	0.05	0.1



$E(T)= 2.80$



$E(T)= 2.75$(最优)

最优二叉搜索树

- 期望搜索代价

$$\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1$$

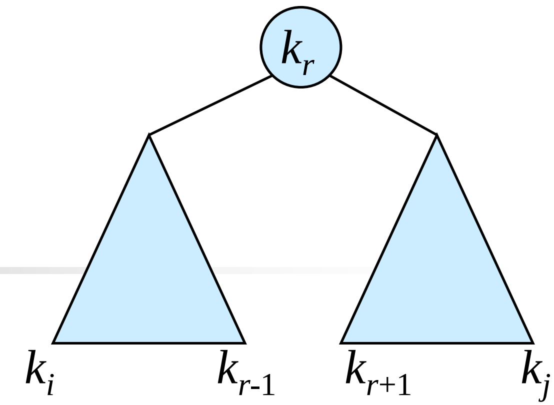
$$E[\text{search cost in } T]$$

$$= \sum_{i=1}^{n} (\mathrm{d}_T(k_i)+1) \cdot p_i + \sum_{i=0}^{n} (\mathrm{d}_T(d_i)+1) \cdot q_i$$

$$= \sum_{i=1}^{n} \mathrm{d}_T(k_i) \cdot p_i + \sum_{i=1}^{n} p_i + \sum_{i=0}^{n} \mathrm{d}_T(d_i) \cdot q_i + \sum_{i=0}^{n} q_i$$

$$= 1 + \sum_{i=1}^{n} \mathrm{d}_T(k_i) \cdot p_i + \sum_{i=0}^{n} \mathrm{d}_T(d_i) \cdot q_i$$

最优二叉搜索树



- 分析最优解的结构
  - 最优子结构性质：假设由关键字 $k_i, \ldots, k_j$构成的最优二叉搜索树的根节点为k_r，则k_r的左右子树也是最优二叉搜索树
  - 证明：如果最优二叉搜索树 T 的左子树 T' 包含关键字 $k_i, \ldots, k_{r\text{-}1}$, 那么子树 T'必须也是最优的，对于带关键字 $k_i, \ldots, k_{r\text{-}1}$ 和虚拟关键字 $d_{i\text{-}1}, \ldots, d_r$的子问题而言，如果有一棵子树$T''$的期望搜索代价低于子树 T',那么可以从T中剪下T'并连到 T''中，从而存在一棵二叉搜索树的期望搜索代价小于T,这与 T是最优的矛盾。

最优二叉搜索树

- 建立递归方程
  - 定义 $e[i, j]$ =对于 $k_i,\ldots,k_j$ 和虚拟节点 $d_{i-1}, \ldots, d_j$, 最优 BST 的期望搜索成本
  - If $j = i-1$, then $e[i, j] = q_{i-1}$.
  - If $j \geq i$, 选出树根 k_r, 对于某个r, $i \leq r \leq j$, 递归地构造一棵最优 BSTs,
    - 对$k_i,\ldots,k_{r-1}$ 构造左子树
    - 对$k_{r+1},\ldots,k_j$ 构造右子树

最优二叉搜索树

- 建立递归方程
  - 当最优的子树成为一个结点的子树时,每个原来在最优子树中结点的深度加1，期望搜索代价增加

$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

  - 如果 k_r 是一棵由$k_i,..,k_j$ 组成的最优BST的根：

$$e[i, j] = p_r + (e[i, r-1] + w(i, r-1)) + (e[r+1, j] + w(r+1, j))$$

$$= e[i, r-1] + e[r+1, j] + w(i, j).$$ (because $w(i, j)=w(i,r-1) + p_r + w(r + 1, j)$)

  - 但是，我们还不知道 k_r. 因此,

$$e[i, j] = \begin{cases} q_{i-1} & j = i-1, \\ \min_{i \le r \le j}\{e[i, r-1] + e[r+1, j] + w[i, j]\} & i \le j. \end{cases}$$

最优二叉搜索树

- 构造最优解，对于每个子问题 (i,j), 存储:
  - 期望搜索成本的表格 $e[1..n+1, 0..n]$，只使用入口 $e[i, j]$, 其中 $j \geq i-1$.
  - $\text{root}[i, j]$ = 由 $k_i,..,k_j$组成的子树的根，其中 $1 \leq i \leq j \leq n$.
  - $w[1..n+1, 0..n]$ = 所有节点的概率和

    $w[i, i-1] = q_{i-1}$ for $1 \leq i \leq n$.

    $w[i, j] = w[i, j-1] + p_j + q_j$ for $1 \leq i \leq j \leq n$.

最优二叉搜索树

- 给出标识符集{1, 2, 3}＝{do, if, stop}存取概率
- 若 $P_1=0.5$，$P_2=0.1$，$P_3=0.05$，$q_0=0.15$，$q_1=0.1$，$q_2=0.05$，$q_3=0.05$，构造一棵最优二叉搜索树

q0=0.15, **P1=0.5**, q1=0.1, **P2=0.1**, q2=0.05, **P3=0.05**, q3=0.05

	0	1	2	3
1	0.15	0.75	0.9	1
2		0.1	0.25	0.35
3			0.05	0.15
4				0.05

W(i, j)

	0	1	2	3
1	0.15	1	1.45	1.85
2		0.1	0.4	0. 7
3			0.05	0.25
4				0.05

e(i, j)

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

root(i, j)

最优二叉搜索树

```
OPTIMAL-BST(p,q,n)
let e[1..n+1,0..n],w[1..n+1,0..n],and root[1..n,1..n]be new tables
for i=1 to n+1
    e[i,i-1]=q_{i-1}
    w[i,i-1]=q_{i-1}
for l=1 to n
    for i = 1 to n-l+1
        j = i+l-1
        e[i,j]=∞
        w[i,j]=w[i,j-1]+p_j+q_j
        for r =i to j
            t =e[i,r-1]+e[r+1,j]+w[i,j]
            if t<e[i,j]
                e[i,j]=t
                root[i,j]=r
return e and root
```

作业

- P141-143:
  - 7.9
  - 7.16(a)
  - 7.22 (画出表格即可)
  - 7.30