HIT
CS&E

第八章
树的搜索策略

张炜
计算机科学与工程系

提要

8.1 为什么引入搜索策略
8.2 基本的搜索策略
8.3 优化的搜索策略
8.4 人事安排问题
8.5 旅行商售货问题
8.6 0-1 背包问题
8.7 A*算法

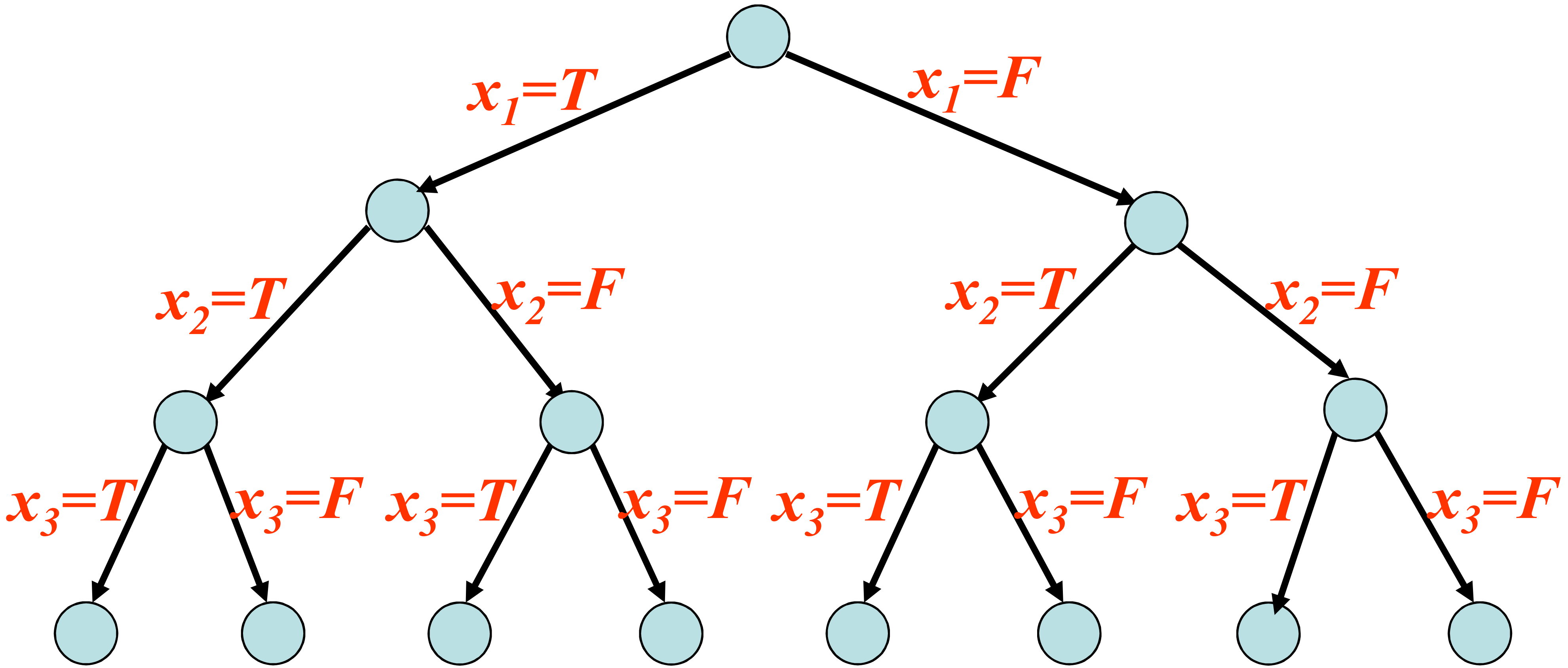
8.1 *Motivation of Tree Searching*

很多问题可以表示成为树,
于是,这些问题可以使用树
搜索算法来求解

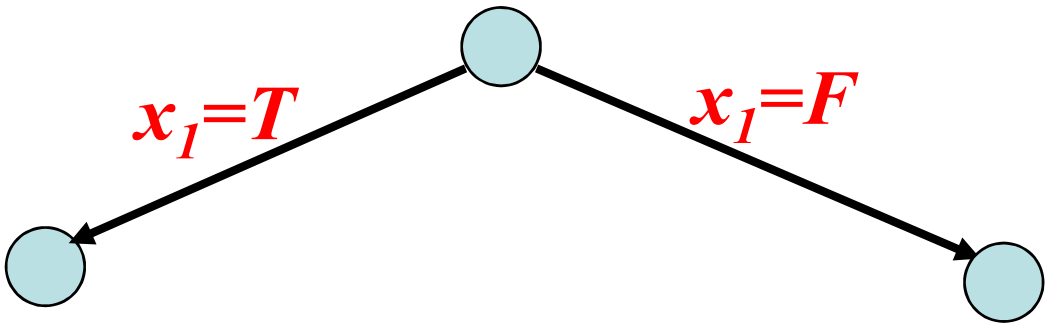
布尔表达式可满足性问题

- 问题的定义
  - 输入：n个布尔变量$x_1, x_2, \ldots, x_n$
    关于$x_1, x_2, \ldots, x_n$的k个析取布尔式
  - 输出：是否存在一个$x_1, x_2, \ldots, x_n$的一种赋值
    使得所有k个布尔析取式皆为真
  - 显示约束和隐式约束

- 把问题表示为树
  - 通过不断地为赋值集合分类来建立树
    (以三个变量(x_1, x_2, x_3)为例)



- 求解问题
  - 设有布尔式：$-x_1, x_1, x_2 \vee x_5, x_3, -x_2$



8-Puzzle问题

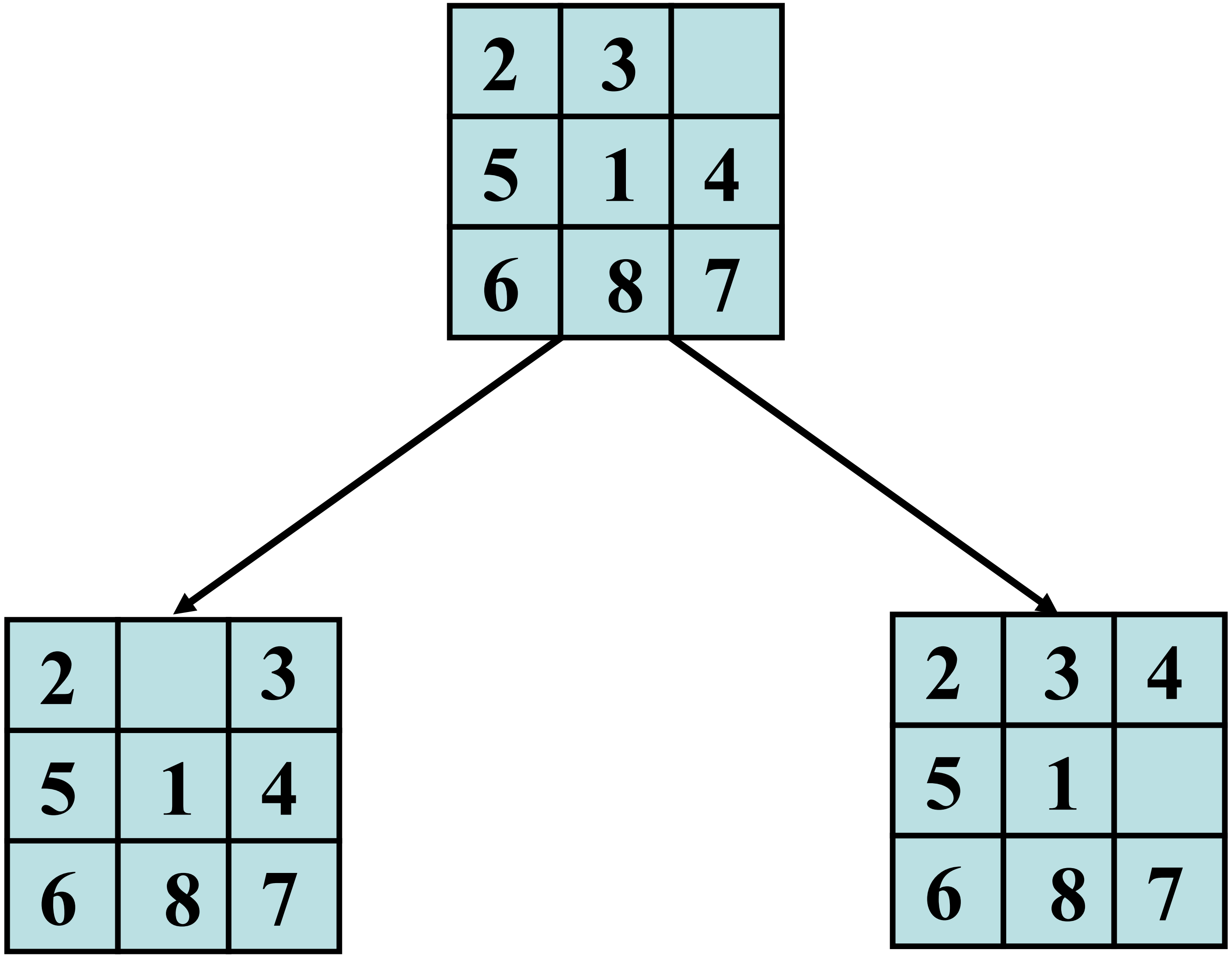
- 问题的定义
  - 输入：具有8个编号小方块的魔方

2	3	
5	1	4
6	8	7

  - 输出：移动系列，经过这些移动，魔方达如下状态

1	2	3
8		4
7	6	5

- 转换为树搜索问题

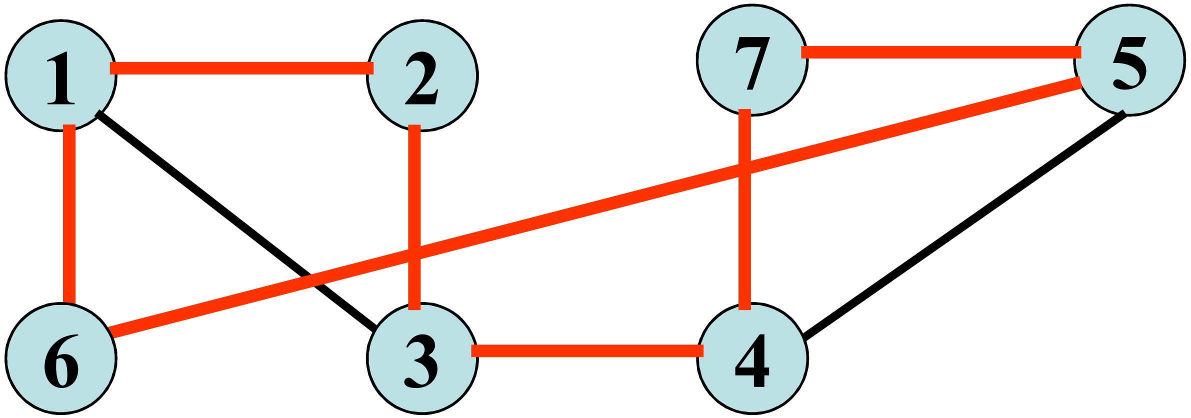


Hamiltonian环问题

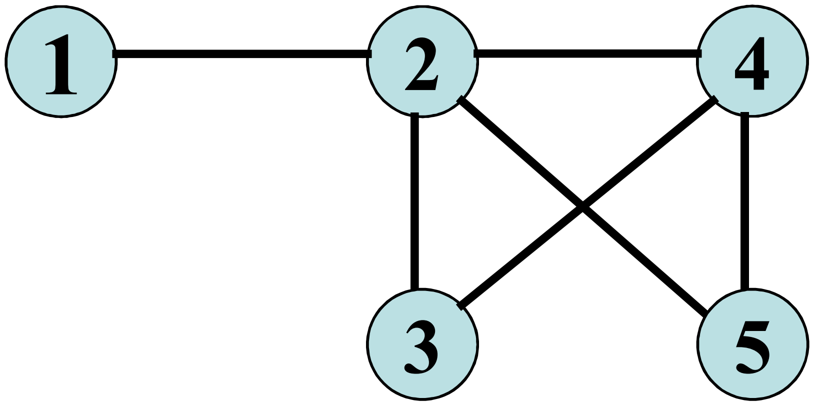
- 问题定义
  - 输入：具有n个节点的连通图$G=(V, E)$
  - 输出：G中是否具有Hamiltonian环

沿着G的n条边经过每个节点一次，并回到起始节点的环称为G的一个Hamiltonian环。

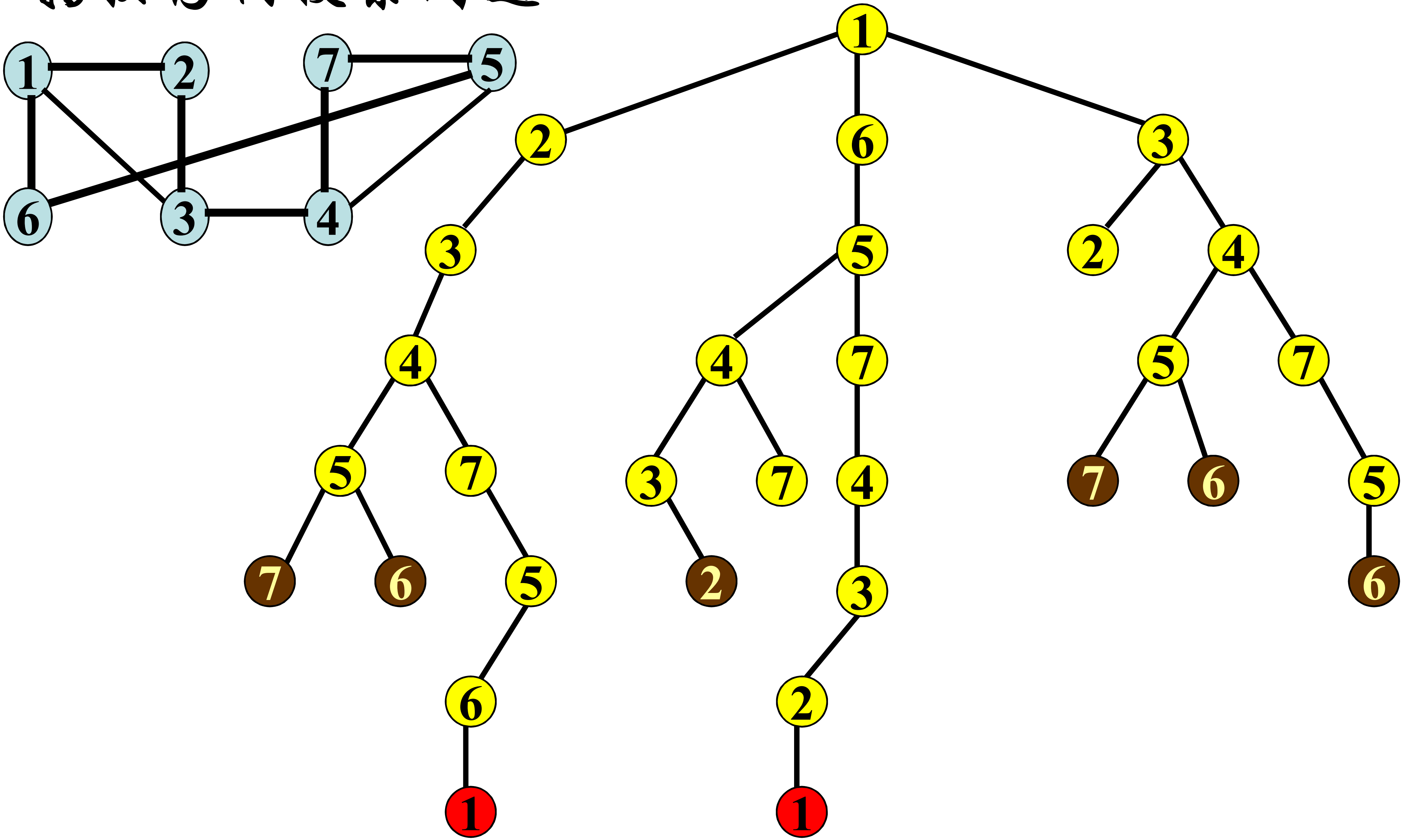
有Hamiltonian环图:

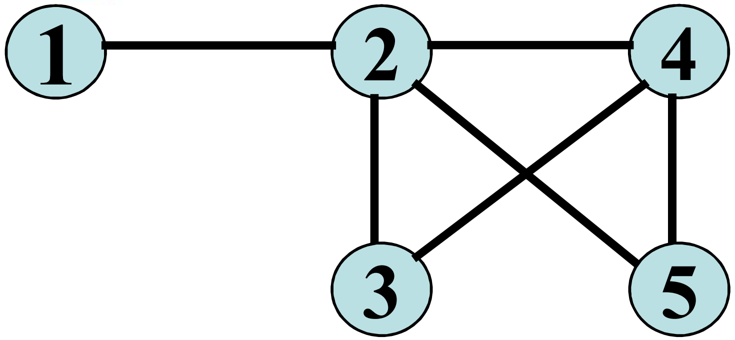


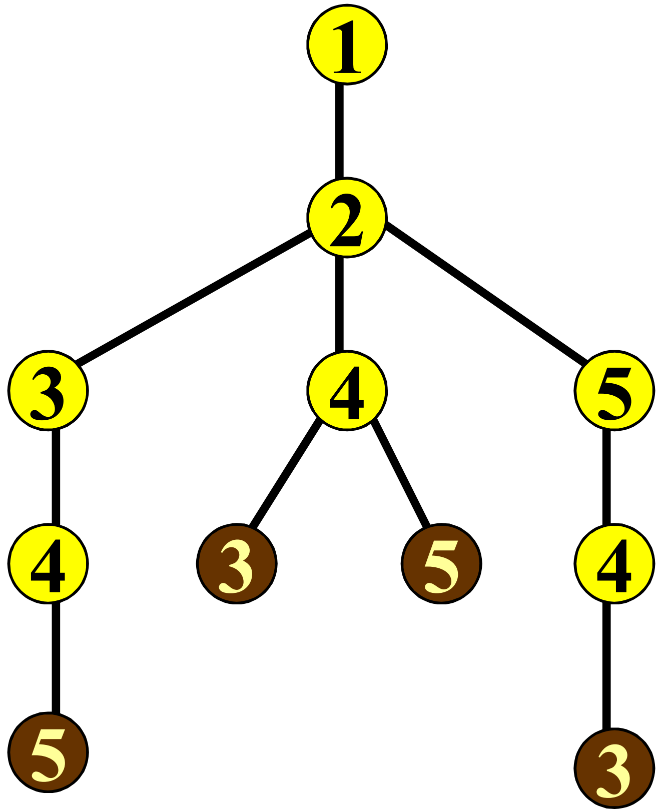
无Hamiltonian环图:



- 转换为树搜索问题




1
2
4
3
5


1
2
3
4
5
4
3
5
4
5
3

8.2 *Basic Tree Searching Strategies*

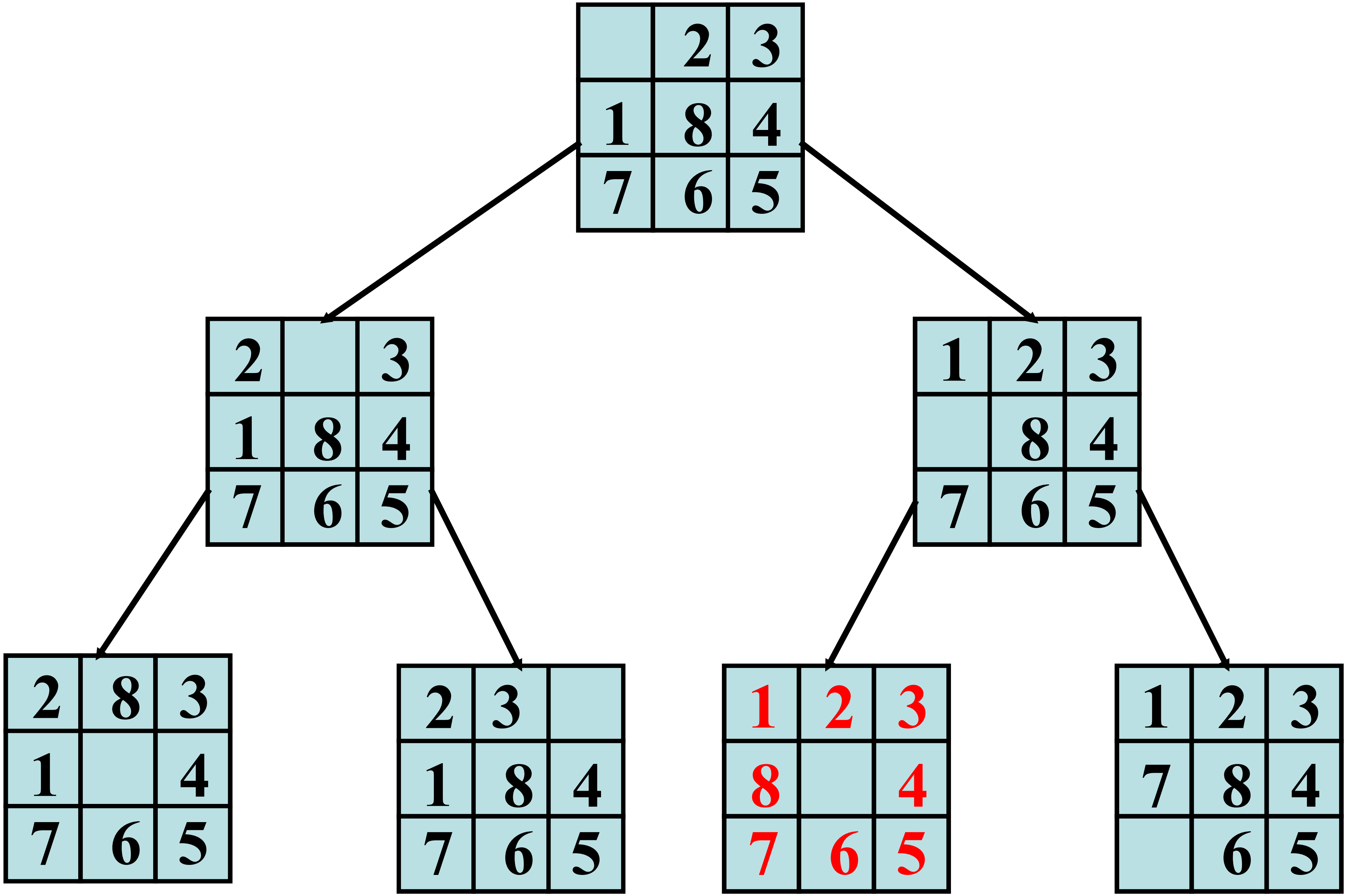
- **Breadth-First Search**
- **Depth-First Search**

Breadth-First Search

- 算法

1. 构造由根组成的队列*Q*;
2. If *Q*的第一个元素*x*是目标节点 Then 停止;
3. 从*Q*中删除*x*，把*x*的所有子节点加入Q的末尾;
4. If *Q*空 Then 失败 Else *goto* 2.

- 例：求解8-Puzzle问题



Depth-First Search

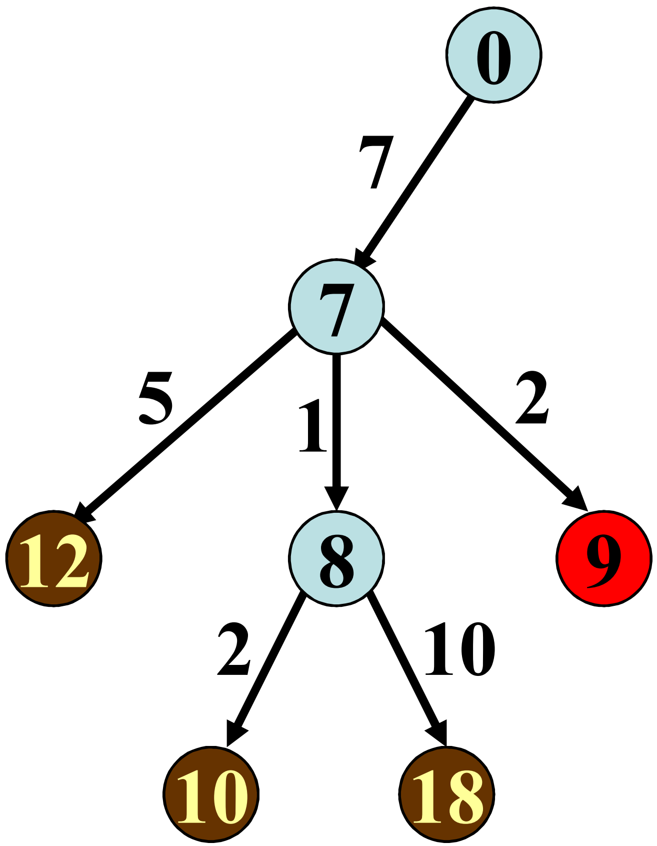
- 算法

1. 构造一个由根构成的单元素栈*S*;
2. If *Top*(*S*)是目标节点 Then 停止;
3. *Pop*(*S*), 把*Top*(*S*)的所有子节点压入栈顶;
4. If S空 Then 失败 Else *goto* 2.

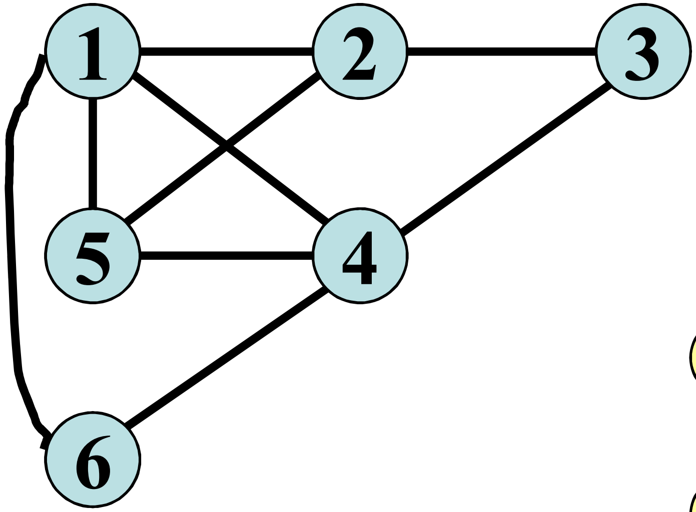
- 例1. 求解子集合和问题

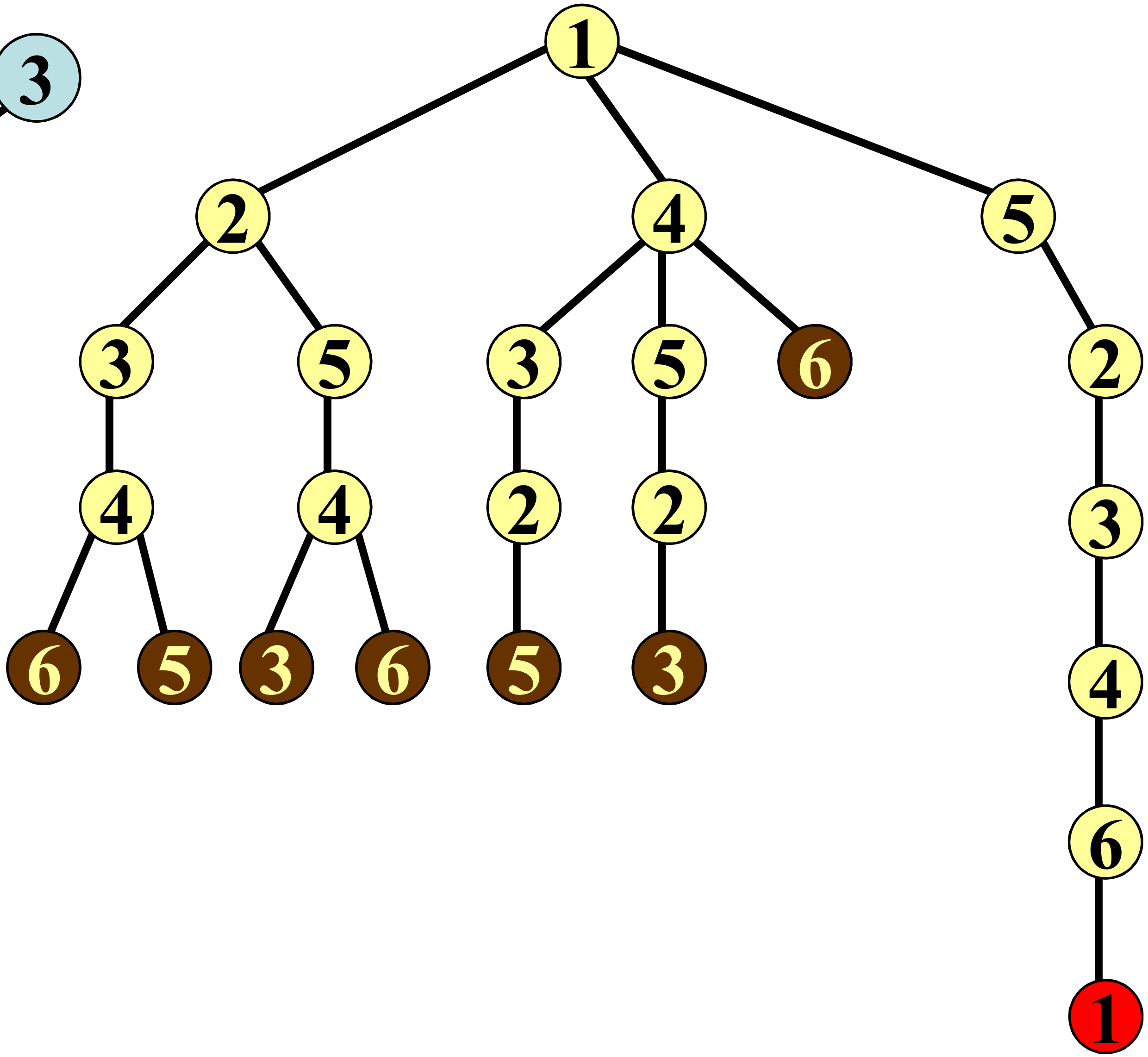
  输入: S={7, 5, 1, 2, 10}

  输出: 是否存在$S' \subseteq S$，使得$Sum(S')=9$



- 例2. 求解Hamiltonian环问题





8.3 *Optimal Tree Searching Strategies*

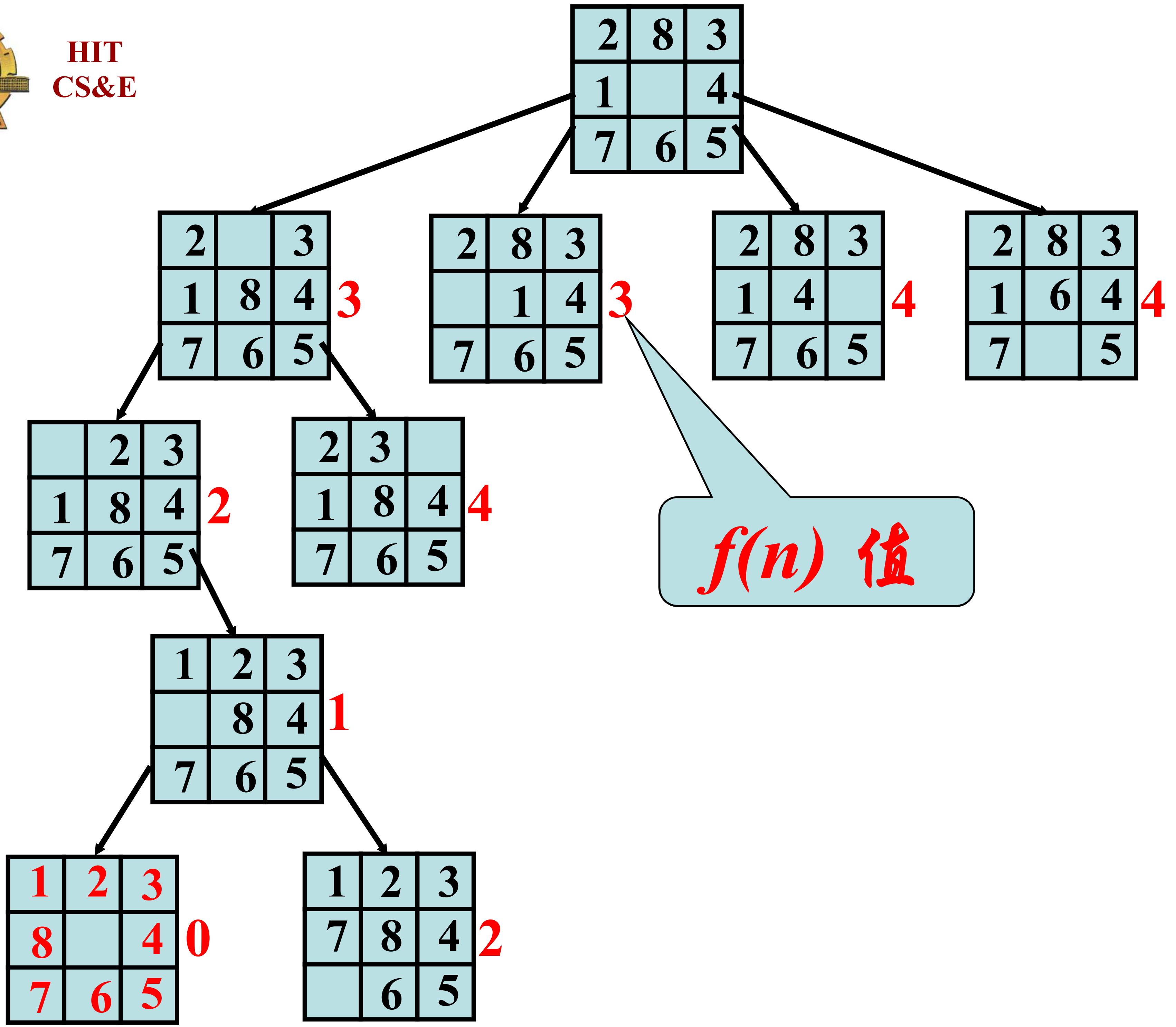
- **Hill Climbing**
- **Best-First Search Strategy**
- **Branch-and-Bound Strategy**

Hill Climbing

- 基本思想
  - 在深度优先搜索过程中，我们经常遇到多个节点可以扩展的情况，首先扩展哪个?
  - 爬山策略使用贪心方法确定搜索的方向，是优化的深度优先搜索策略
  - 爬山策略使用启发式测度来排序节点扩展的顺序

- 用8-Puzzle问题来说明爬山策略的思想
  - 启发式测度函数: $f(n)=W(n)$, $W(n)$是节点n中处于错误位置的方块数.
  - 例如, 如果节点n如下, 则$f(n)=3$, 因为方块1、2、8处于错误位置.

2	8	3
1		4
7	6	5


f(n) 值

- **Hill Climbing算法**

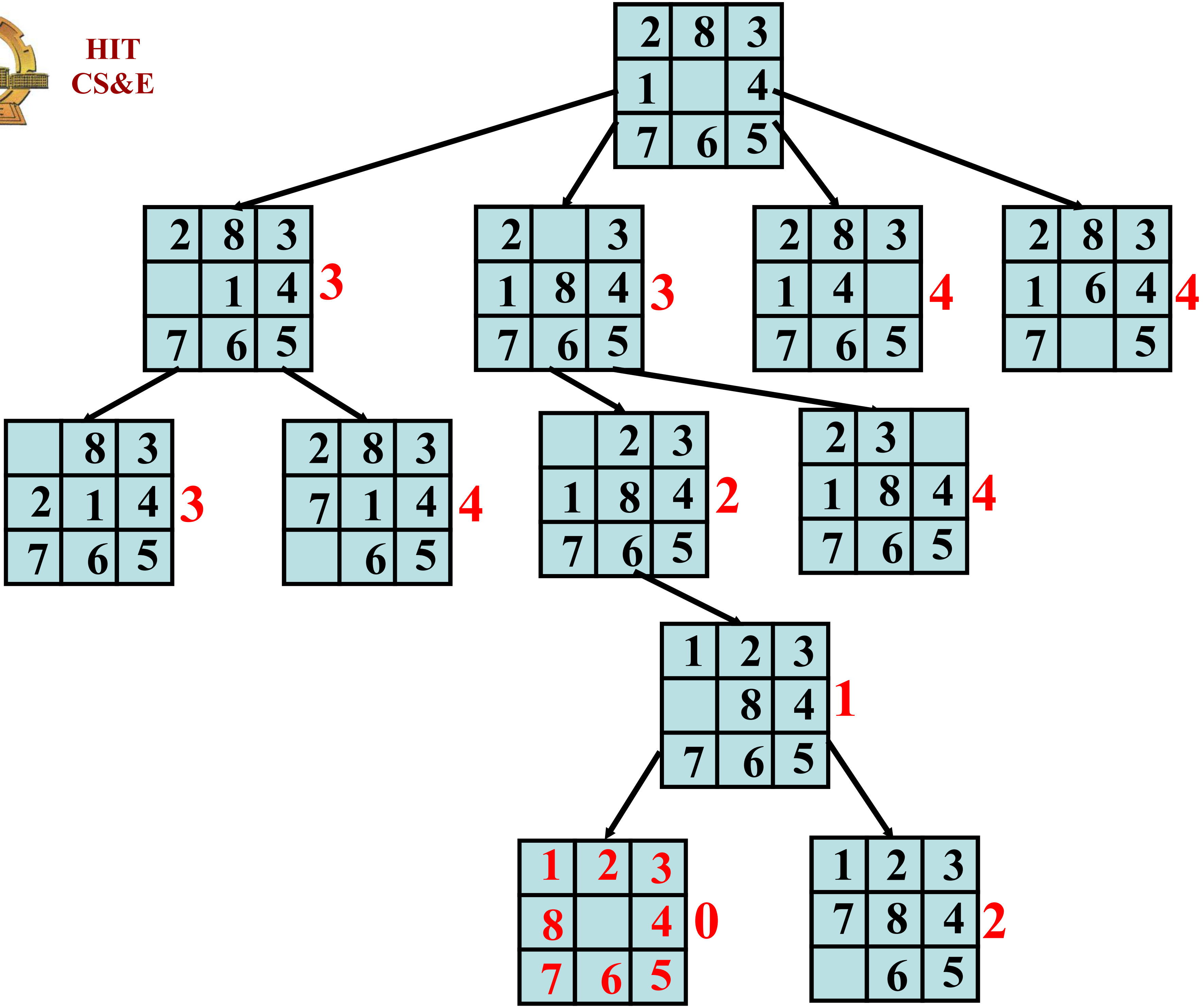
1. 构造由根组成的单元素栈*S*;
2. If *Top(S)*是目标节点 Then 停止;
3. Pop(*S*);
4. *S*的子节点按照其启发测度由大到小的顺序压入*S*;
5. If *S*空 Then 失败 Else *goto* 2.

Best-First Search Sttrategy

- 基本思想
  - 结合深度优先和广度优先的优点
  - 根据一个评价函数，在目前产生的所有节点中选择具有最小评价函数值的节点进行扩展.
  - 具有全局优化观念，而爬山策略仅具有局部优化观念.

- **Best-First Search算法**
  1. 使用评价函数构造一个堆*H*, 首先构造由根组成的单元素堆;
  2. If *H*的根*r*是目标节点 Then 停止;
  3. 从*H*中删除*r*, 把*r*的子节点插入*H*;
  4. If *H*空 Then 失败 Else *goto* 2.
- **8-Puzzle问题实例**

HIT
CS&E



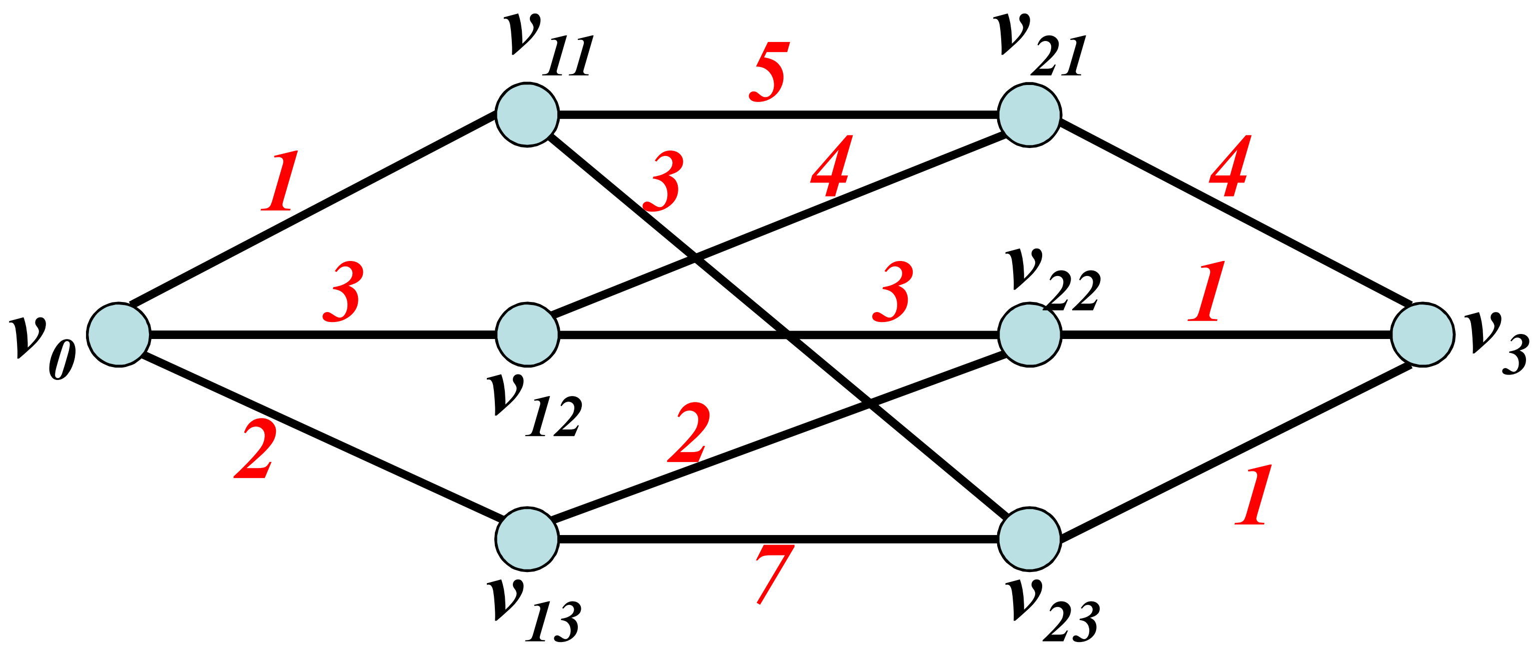
2 8 3
1 4
7 6 5
2 8 3
1 4
7 6 5
3
2 3
1 8 4
7 6 5
3
2 8 3
1 4
7 6 5
4
2 8 3
1 6 4
7 5
4
8 3
2 1 4
7 6 5
3
2 8 3
7 1 4
6 5
4
2 3
1 8 4
7 6 5
2
2 3
1 8 4
7 6 5
4
1 2 3
8 4
7 6 5
1
1 2 3
8 4
7 6 5
0
1 2 3
7 8 4
6 5
2

Branch-and-Bound Strategy

- 基本思想
  - 上述方法很难用于求解优化问题
  - 分支界限策略可以有效地求解组合优化问题
  - 发现优化解的一个界限
  - 缩小解空间，提高求解的效率
- 举例说明分支界限策略的原理

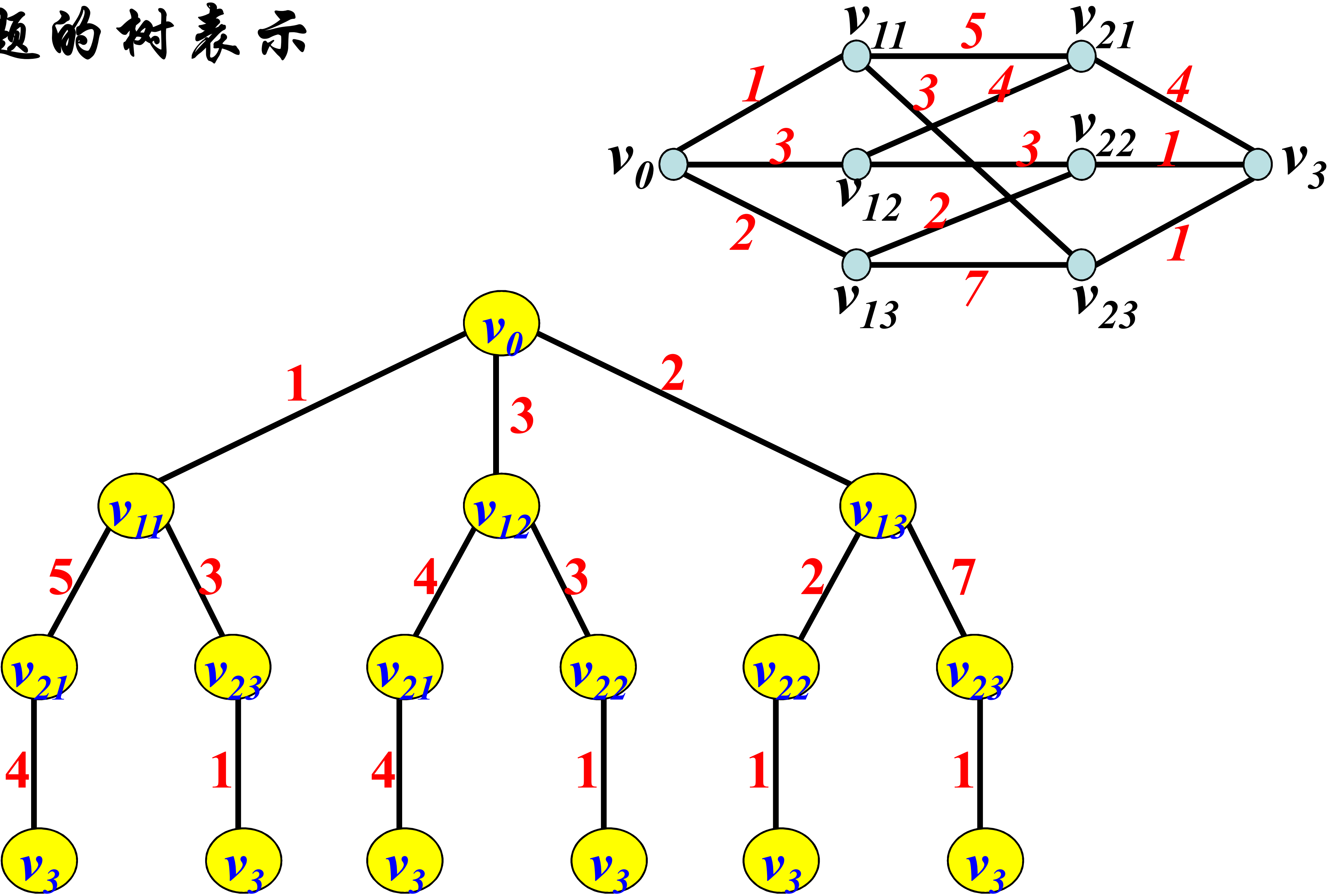
HIT
CS&E

- 多阶段图搜索问题
  - 输入：多阶段图



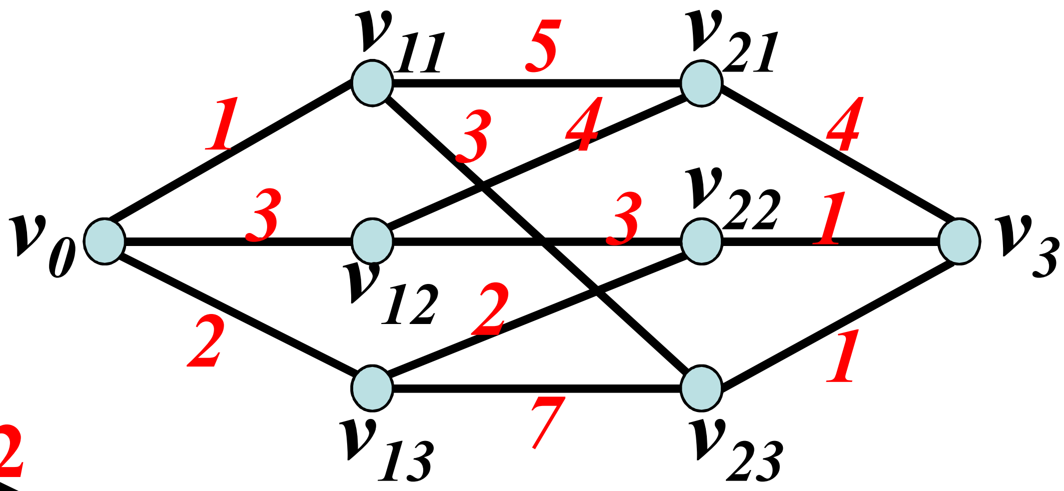
  - 输出：从v_0到v_3的最短路径

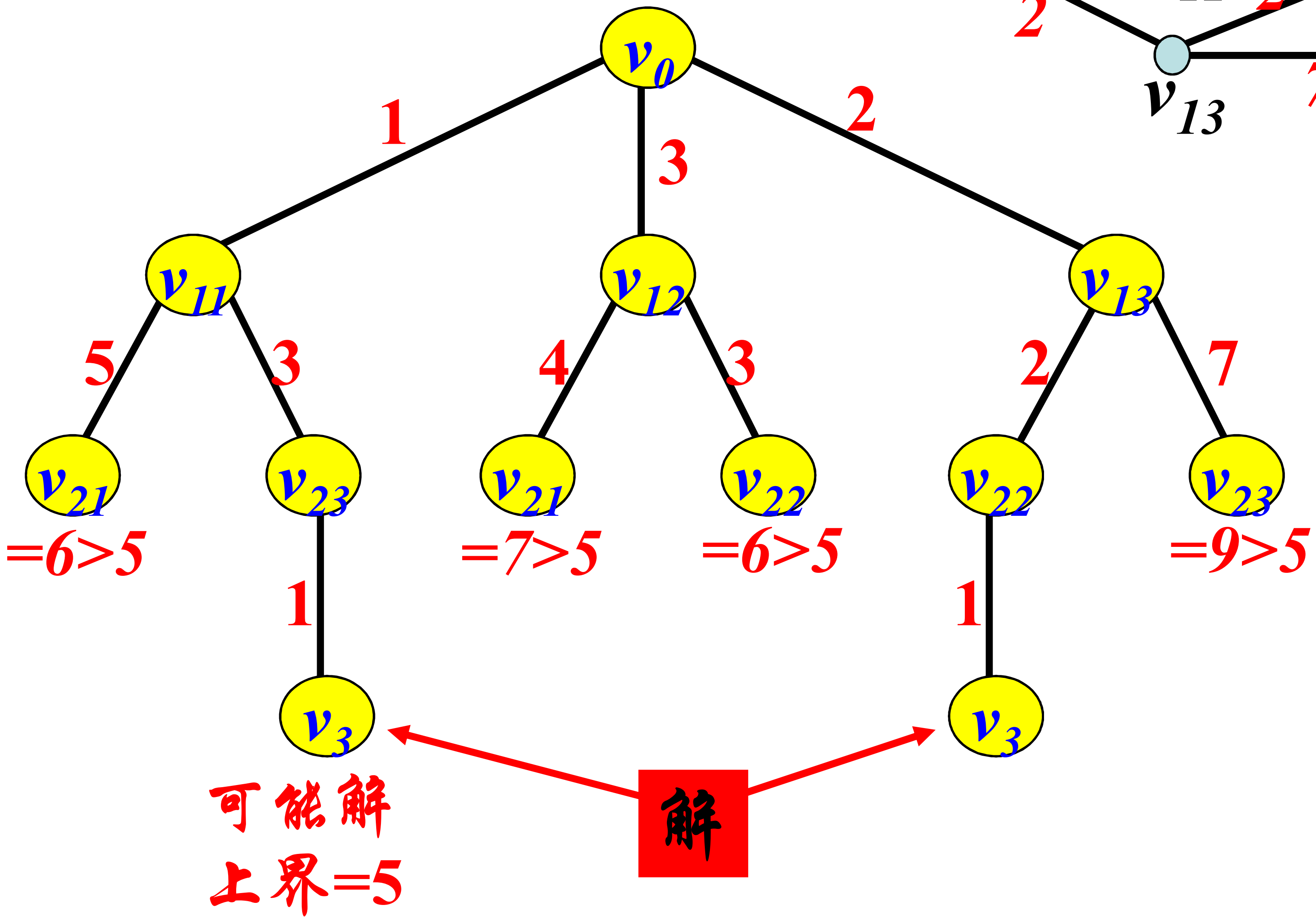
- 问题的树表示



v_{11}
v_{21}
v_0
v_{12}
v_{22}
v_3
v_{13}
v_{23}
5
1
3
4
4
3
3
1
2
2
1
7
v_0
1
3
2
v_{11}
v_{12}
v_{13}
5
3
4
3
2
7
v_{21}
v_{23}
v_{21}
v_{22}
v_{22}
v_{23}
4
1
4
1
1
1
v_3
v_3
v_3
v_3
v_3
v_3

- 使用爬山策略进行分支界限搜索





- 分支界限策略的原理
  - 产生分支的机制(使用前面的任意一种策略)
  - 产生一个界限(可以通过发现可能解)
  - 进行分支界限搜索,即剪除不可能产生优化解的分支.

8.4 *Personnel Assignment Problem*

- 问题的定义
- 转换为树搜索问题
- 求解问题的分支界限搜索算法

问题的定义

- 输入
  - 人的集合$P=\{P_1, P_2, \ldots, P_n\}$，$P_1<P_2<\ldots<P_n$

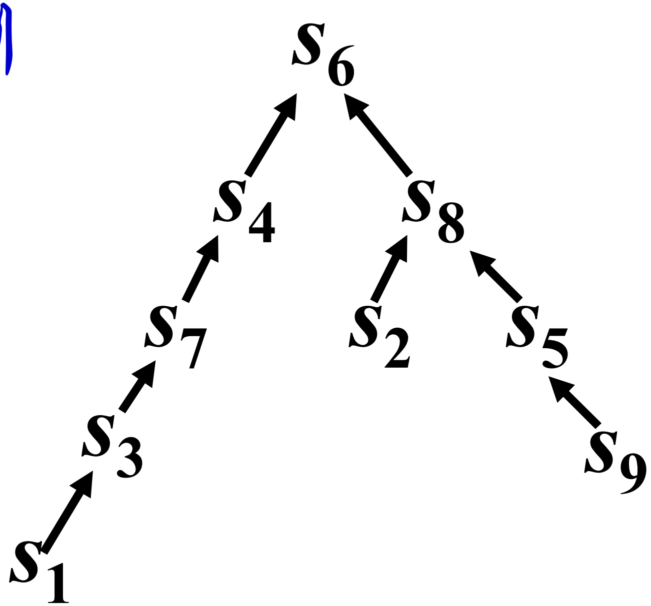
例. 给定$P=\{P_1, P_2, P_3\}$, $J=\{J_1, J_2, J_3\}$, $J_1 \leq J_3$, $J_2 \leq J_3$.
$P_1 \rightarrow J_1$、$P_2 \rightarrow J_2$、$P_3 \rightarrow J_3$是可能的解.
$P_1 \rightarrow J_1$、$P_2 \rightarrow J_3$、$P_3 \rightarrow J_2$不可能是解.

  - 如果$f(P_i) \leq f(P_j)$，则$P_i \leq P_j$

HIT
CS&E

转换为树搜索问题

- **拓朴排序**
  - 输入: 偏序集合$(S, \leq)$
  - 输出: S的拓朴序列是$<s_1, s_2, \ldots, s_n>$,
    满足: 如果$s_i \leq s_j$, 则s_i排在s_j的前面.
  - 例



拓朴排序:

s_1 s_3 s_7 s_4 s_9 s_5 s_2 s_8 s_6

- 问题的解空间

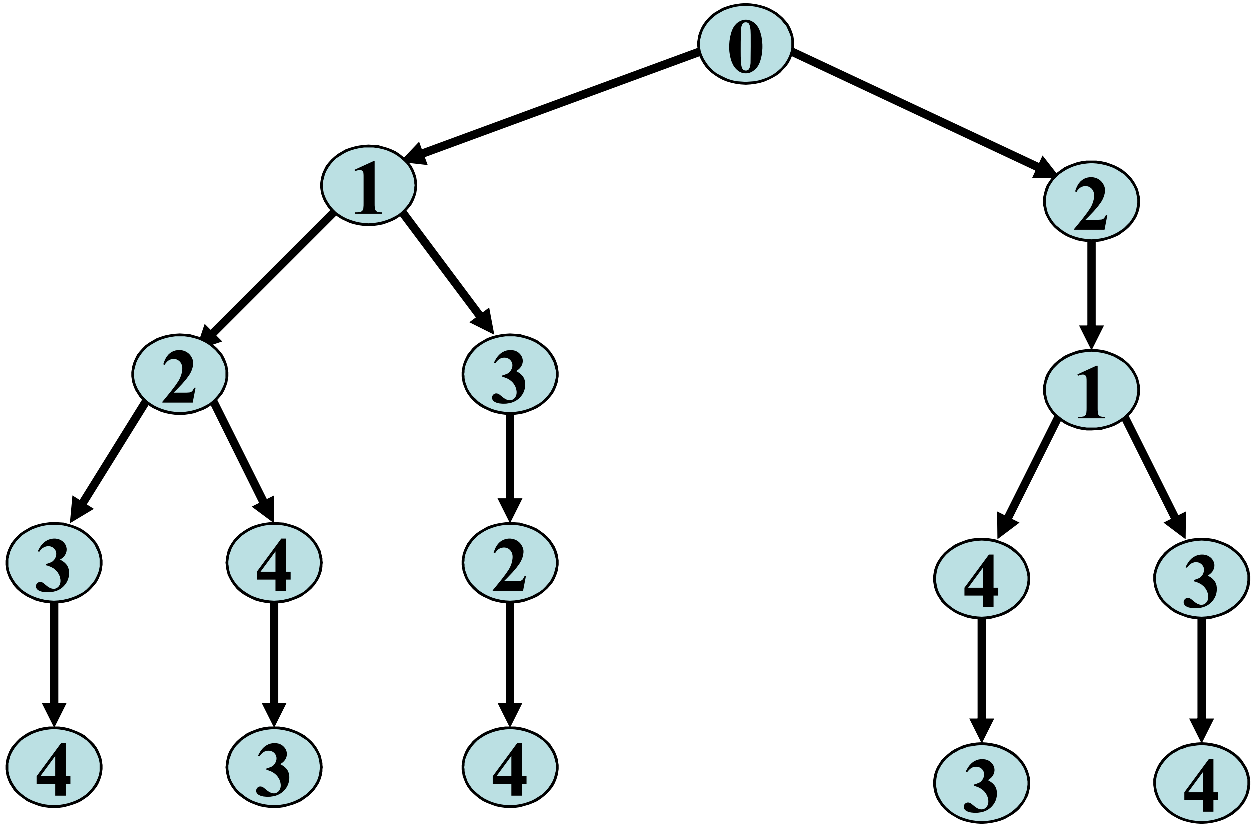
命题1. $P_1 \rightarrow J_{k1}$、$P_2 \rightarrow J_{k2}$、…、$P_n \rightarrow J_{kn}$是一个可能解，当且仅当J_{k1}、J_{k2}、…、J_{kn}必是一个拓朴排序的序列.

问题的解空间是所有拓朴排序的序列集合，每个序列对于一个可能的解

对 (J_1, J_2, J_3, J_4)、(J_1, J_2, J_4, J_3)、(J_1, J_3, J_2, J_4)、(J_2, J_1, J_3, J_4)、(J_2, J_1, J_4, J_3)是拓朴排序序列

(J_1, J_2, J_4, J_3)对应于$P_1 \rightarrow J_1$、$P_2 \rightarrow J_2$、$P_3 \rightarrow J_4$、$P_4 \rightarrow J_3$

- 问题的树表示(即用树表示所有拓扑排序序列)



● 拓朴序列树的生成算法

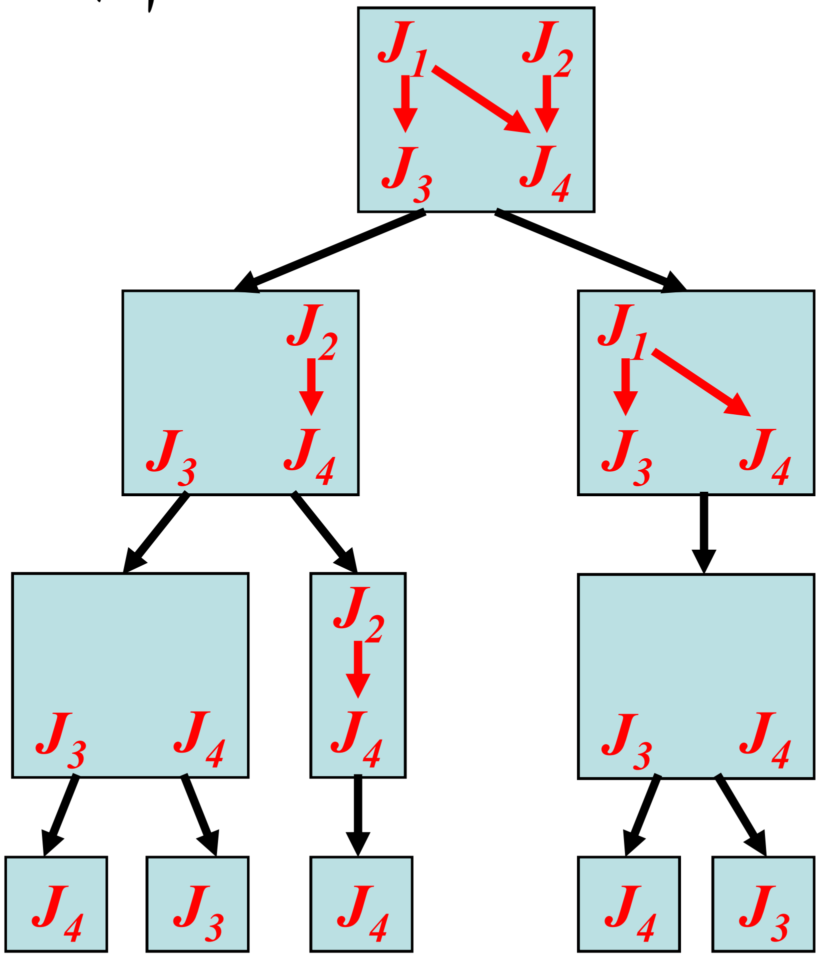
输入：偏序集合S, 树根*root*.

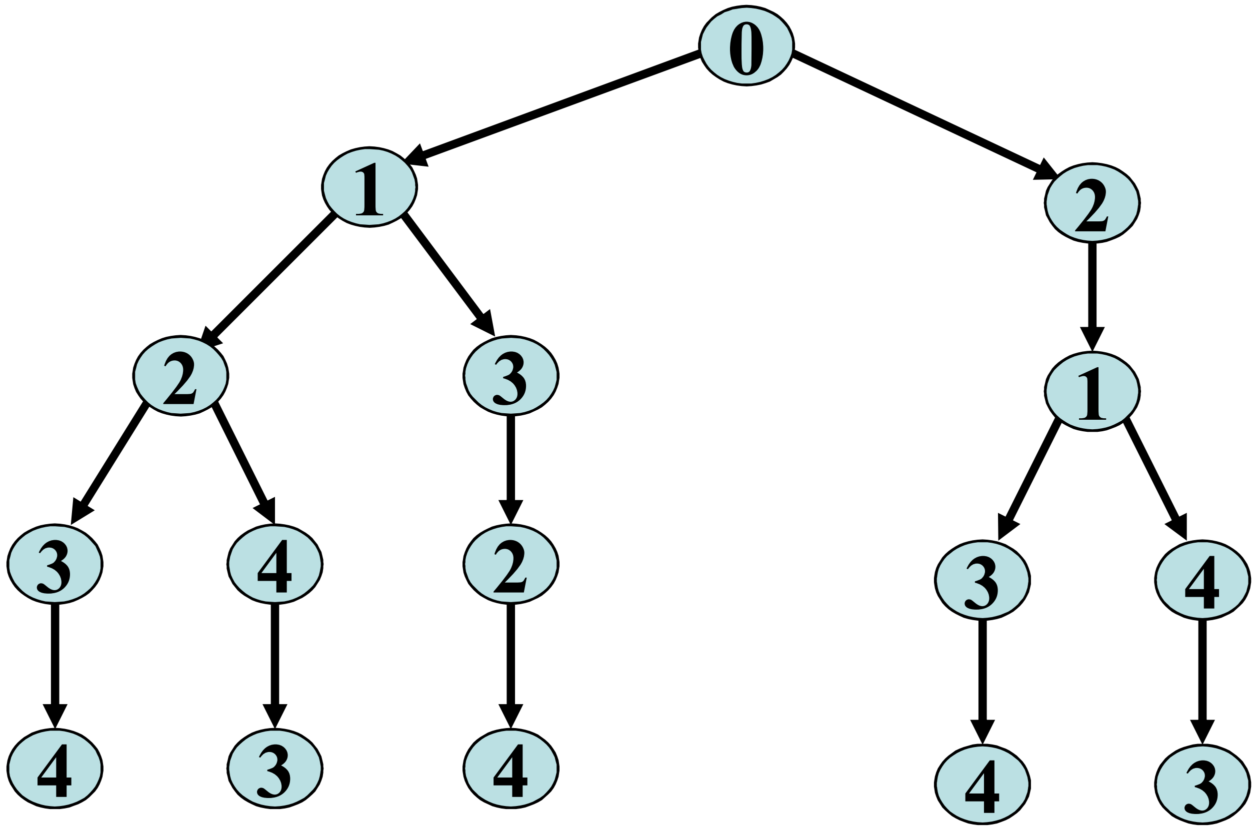
输出：由S的所有拓朴排序序列构成的树.

1. 生成树根*root*;
2. 选择偏序集中没有前序元素的所有元素，作为*root*的子节点;
3. For *root*的每个字节点v Do
4. $S=S-\{v\}$;
5. 把v作为根，递归地处理S.

HIT
CS&E

- 例





$$
\begin{array}{c}
 \\ P_1 \\ P_2 \\ P_3 \\ P_4
\end{array}
\begin{array}{c}
\begin{array}{cccc} J_1 & J_2 & J_3 & J_4 \end{array} \\
\begin{bmatrix}
29 & 19 & 17 & 12 \\
32 & 30 & 26 & 28 \\
3 & 21 & 7 & 9 \\
18 & 13 & 10 & 15
\end{bmatrix}
\end{array}
$$

- 计算解的代价的下界

命题2. 把代价矩阵某行(列)的各元素减去同一个数，不影响优化解的求解.

- 代价矩阵的每行(列)减去同一个数(该行或列的最小数)，使得每行和每列至少有一个零，其余各元素非负.
- 每行(列)减去的数的和即为解的下界.

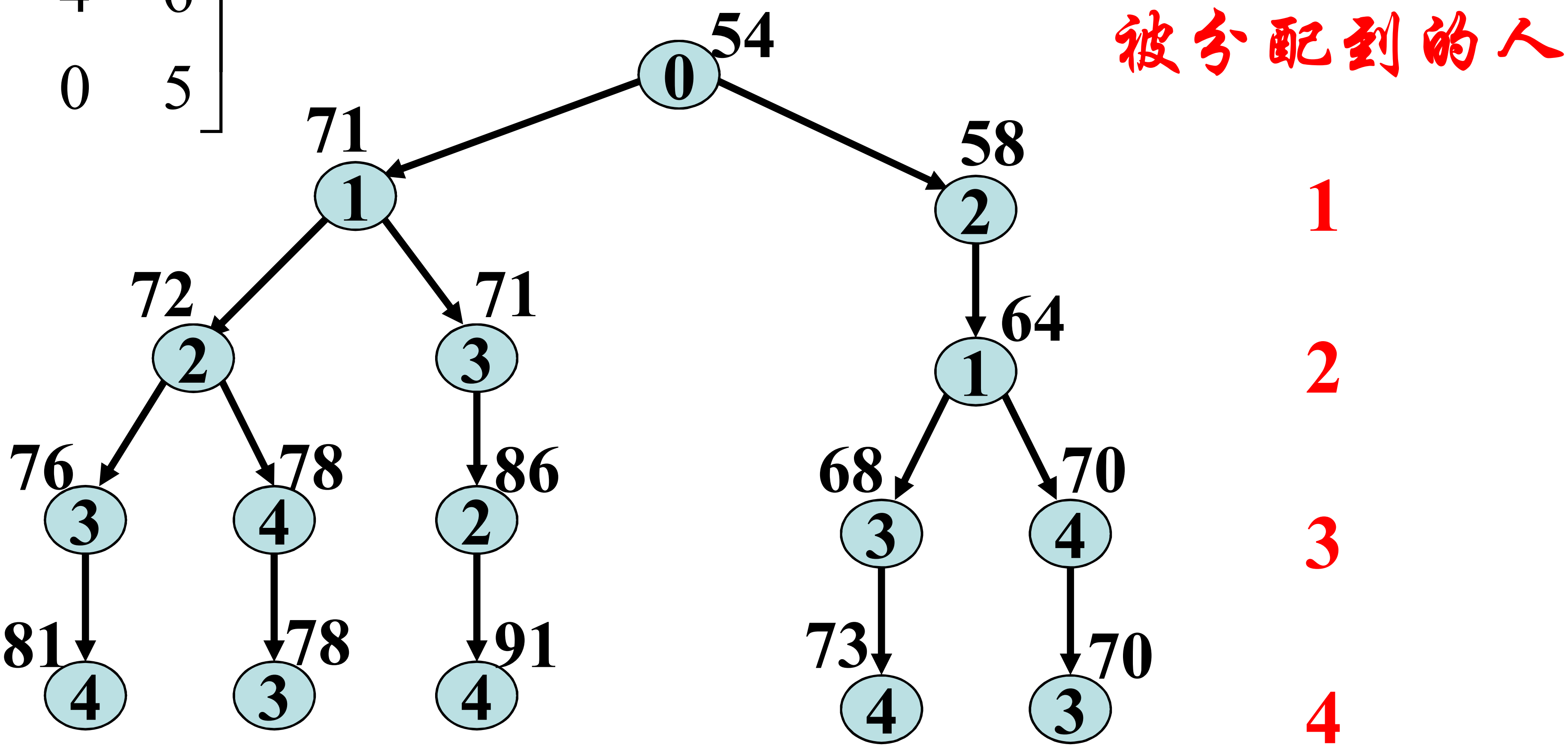
例.

$$\begin{array}{ccc} & \begin{array}{cccc} J_1 & J_2 & J_3 & J_4 \end{array} & \\ \begin{array}{c} P_1 \\ P_2 \\ P_3 \\ P_4 \end{array} & \begin{bmatrix} 29 & 19 & 17 & 12 \\ 32 & 30 & 26 & 28 \\ 3 & 21 & 7 & 9 \\ 18 & 13 & 10 & 15 \end{bmatrix} & \begin{array}{c} -12 \\ -26 \\ -3 \\ -10 \end{array} \\ & \begin{array}{cccc} & -3 & & \end{array} & \end{array} \Rightarrow \begin{bmatrix} 17 & 4 & 5 & 0 \\ 6 & 1 & 0 & 2 \\ 0 & 15 & 4 & 6 \\ 8 & 0 & 0 & 5 \end{bmatrix}$$

解代价下界=12+26+3+10+3=54

解空间的加权树表示

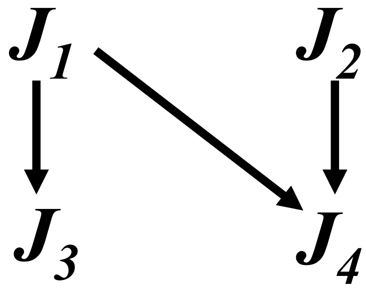
$$\begin{bmatrix} 17 & 4 & 5 & 0 \\ 6 & 1 & 0 & 2 \\ 0 & 15 & 4 & 6 \\ 8 & 0 & 0 & 5 \end{bmatrix}$$



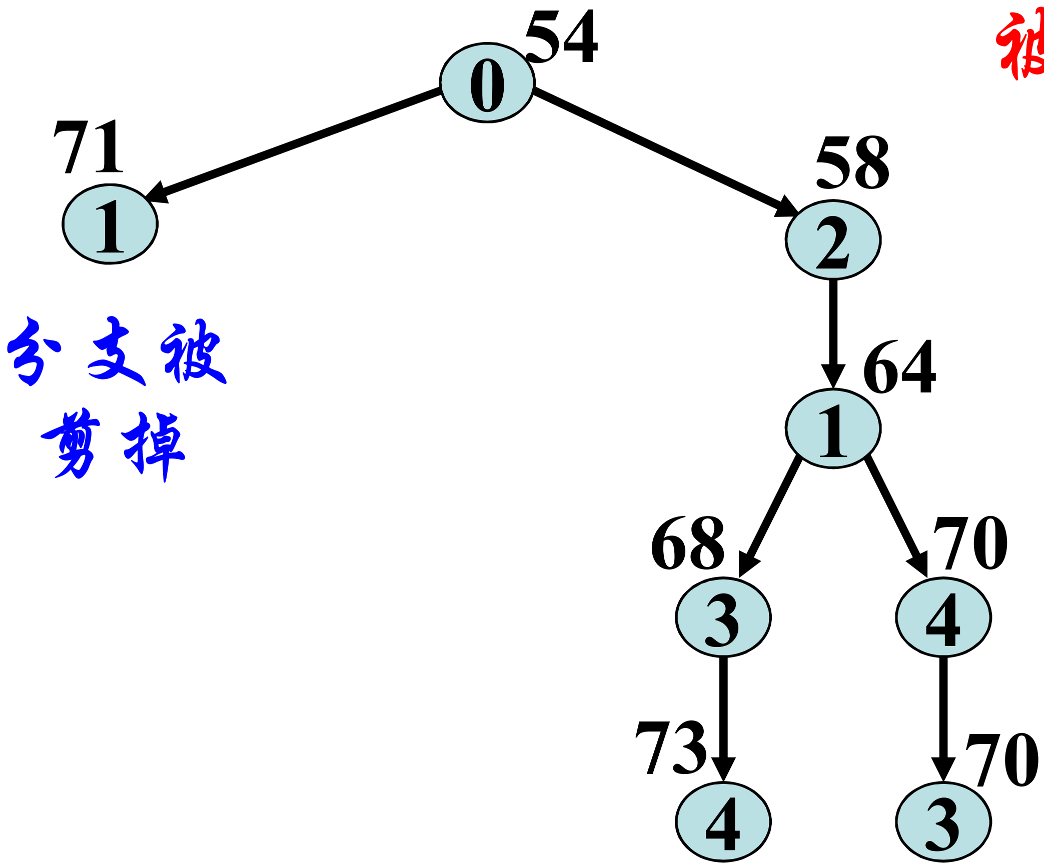
- 分支界限搜索(使用爬山法)算法
  1. 建立根节点，其权值为解代价下界；
  2. 使用爬山法，类似于拓扑排序序列树生成算法求解问题，每产生一个节点，其权值为加工后的代价矩阵对应元素加其父节点权值；
  3. 一旦发现一个可能解，将其代价作为界限，循环地进行分支界限搜索：剪掉不能导致优化解的子解，使用爬山法继续扩展新增节点，直至发现优化解.

- 例

$\{P_1, P_2, P_3, P_4\}$



$$\begin{bmatrix} 17 & 4 & 5 & 0 \\ 6 & 1 & 0 & 2 \\ 0 & 15 & 4 & 6 \\ 8 & 0 & 0 & 5 \end{bmatrix}$$



被分配到的人

1

2

3

4

8.5 Traveling Salesperson Optimization Problem

- 问题的定义
- 转换为树搜索问题
- 分支界限搜索算法

问题的定义

输入：无向连通图$G=(V, E)$,

每个节点都没有到自身的边,

每对节点之间都有一条非负加权边.

输出：一条由任意一个节点开始

经过每个节点一次

最后返回开始节点的路径,

该路径的代价(即权值只和)最小.

转换为树搜索问题

- 所有解集合作为树根，其权值由代价矩阵使用上节方法计算；
- 用爬山法递归地划分解空间，得到二叉树
- 划分过程：
  - 如下选择图上满足下列条件的边(i, j)
    - $\text{Cost}(i, l)=\max\{\text{Cost}(k,l) \mid \forall k \in V\}$
    - $\text{Cost}(i, j)=0$

    使右子树代价下界增加最大
  - 所有包含(i, j)的解集合作为左子树
  - 所有不包含(i, j)的解集合作为右子树
  - 计算出左右子树的代价下界

分支界限搜索算法

- 在上述二叉树建立算法中增加如下策略：
  - 发现优化解的上界α;
  - 如果一个子节点的代价下界超过α, 则终止该节点的扩展.

- 下边我们用一个例子来说明算法

- 构造根节点，设代价矩阵如下

$$\begin{array}{ccc}
 & \begin{array}{ccccccc} j=1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} & \\
\begin{array}{c} i=1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} & \begin{bmatrix} \infty & 3 & 93 & 13 & 33 & 9 & 57 \\ 4 & \infty & 77 & 42 & 21 & 16 & 34 \\ 45 & 17 & \infty & 36 & 16 & 28 & 25 \\ 39 & 90 & 80 & \infty & 56 & 7 & 91 \\ 28 & 46 & 88 & 33 & \infty & 25 & 57 \\ 3 & 88 & 18 & 46 & 92 & \infty & 7 \\ 44 & 26 & 33 & 27 & 84 & 39 & \infty \end{bmatrix} & \begin{array}{c} -3 \\ -4 \\ -16 \\ -7 \\ -25 \\ -3 \\ -26 \end{array} \\
 & \begin{array}{ccccccc} & & -7 & -1 & & & -4 \end{array} &
\end{array}$$

- 根节点为所有解的集合
- 计算根节点的代价下界

➢ 得到如下根节点及其代价下界

所有解的集合

L.B=96

➢ 变换后的代价矩阵为

$$\begin{array}{c|ccccccc} & j=1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline i=1 & \infty & 0 & 83 & 9 & 30 & 6 & 50 \\ 2 & 0 & \infty & 66 & 37 & 17 & 12 & 26 \\ 3 & 29 & 1 & \infty & 19 & 0 & 12 & 5 \\ 4 & 32 & 83 & 66 & \infty & 49 & 0 & 80 \\ 5 & 3 & 21 & 56 & 7 & \infty & 0 & 28 \\ 6 & 0 & 85 & 8 & 42 & 89 & \infty & 0 \\ 7 & 18 & 0 & 0 & 0 & 58 & 13 & \infty \end{array}$$

$f(1,2)=6+0=6$

$f(2,1)=12+0=12$

$f(3,5)=1+17=18$

$f(4,6)=32+0=32$

$f(5,6)=3+0=3$

$f(6,1)=0+0=0$

$f(6,7)=0+5=5$

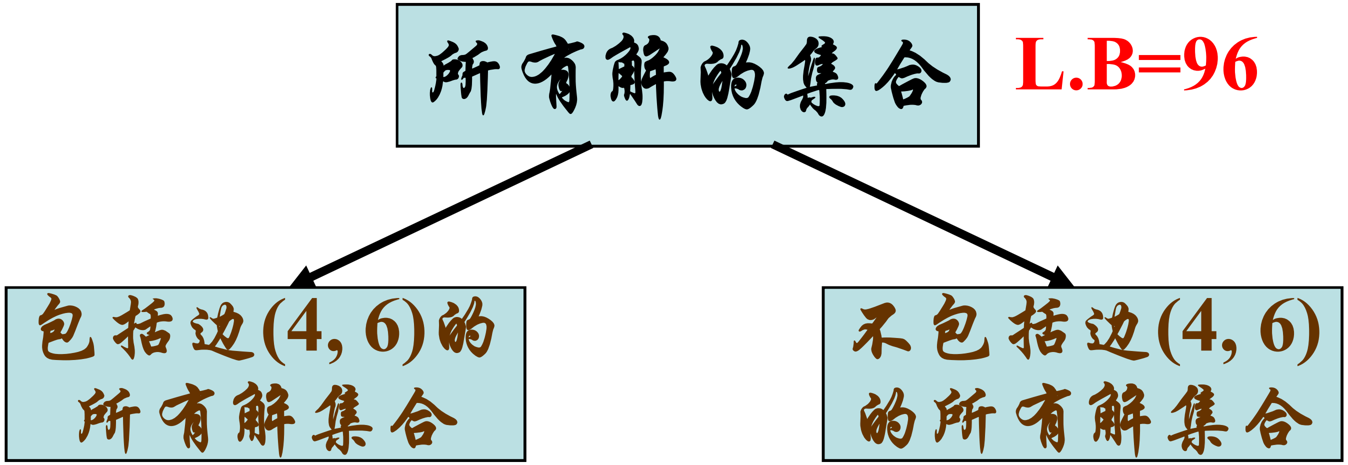
$f(7,2)=0+0=0$

$f(7,3)=0+8=8$

$f(7,4)=0+7=7$

$$\begin{bmatrix} \infty & 0 & 83 & 9 & 30 & 6 & 50 \\ 0 & \infty & 66 & 37 & 17 & 12 & 26 \\ 29 & 1 & \infty & 19 & 0 & 12 & 5 \\ 32 & 83 & 66 & \infty & 49 & 0 & 80 \\ 3 & 21 & 56 & 7 & \infty & 0 & 28 \\ 0 & 85 & 8 & 42 & 89 & \infty & 0 \\ 18 & 0 & 0 & 0 & 58 & 13 & \infty \end{bmatrix}$$

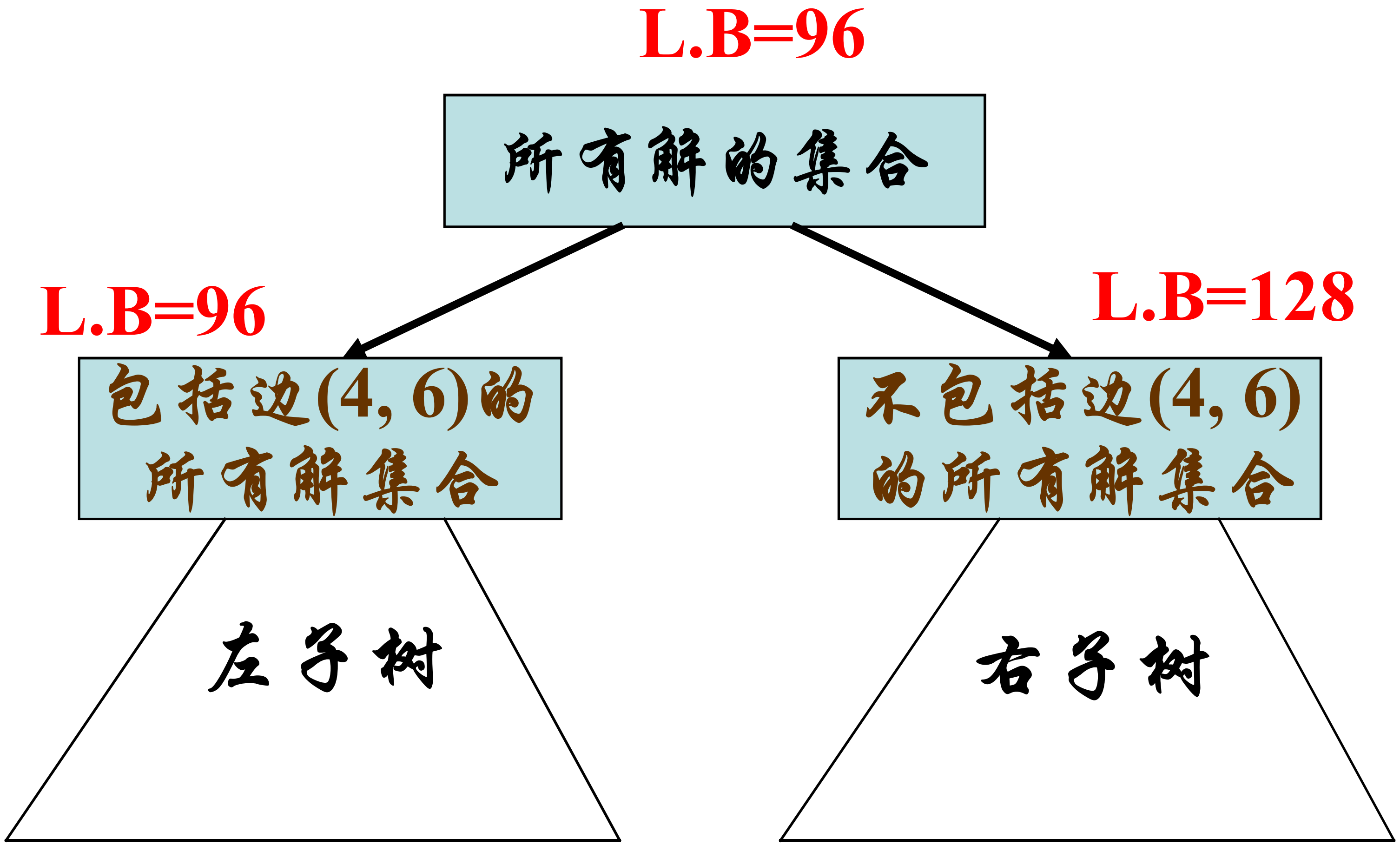
- **构造根节点的两个子节点**
  - ➢ 选择使子节点代价下界增加最大的划分边(4, 6)
  - ➢ 建立根节点的子节点：
    - ✓ 左子节点为包括边(4, 6)的所有解集合
    - ✓ 左子节点为不包括边(4, 6)的所有解集合



➢ 计算左右子节点的代价下界

- ✓ (4, 6)的代价为0, 所以左节点代价下界仍为96.
- ✓ 我们来计算右节点的代价下界:
  - ◆ 如果一个解不包含(4, 6), 它必包含一条从4出发的边和 进入节点6的边.
  - ◆ 由变换后的代价矩阵可知, 具有最小代价由4出发的边为(4, 1), 代价为32.
  - ◆ 由变换后的代价矩阵可知, 具有最小代价进入6的边为(5, 6). 代价为0.
  - ◆ 于是, 右节点代价下界为: 96+32+0=128.

➢ 目前的树为



- 递归地构造左右子树
  - ➢ 构造左子树根对应的代价矩阵
    - ✓ 左子节点为包括边(4, 6)的所有解集合，所以矩阵的第4行和第6列应该被删除
    - ✓ 由于边(4, 6)被使用，边(6, 4)不能再使用，所以代价矩阵的元素 $C[6, 4]$ 应该设置为∞.
    - ✓ 结果矩阵如下

$$
\begin{array}{cc}
 & \begin{array}{cccccc} j=1 & 2 & 3 & 4 & 5 & 7 \end{array} \\
\begin{array}{c} i=1 \\ 2 \\ 3 \\ 5 \\ 6 \\ 7 \end{array} &
\begin{bmatrix}
\infty & 0 & 83 & 9 & 30 & 50 \\
0 & \infty & 66 & 37 & 17 & 26 \\
29 & 1 & \infty & 19 & 0 & 5 \\
3 & 21 & 56 & 7 & \infty & 28 \\
0 & 85 & 8 & \infty & 89 & 0 \\
18 & 0 & 0 & 0 & 58 & \infty
\end{bmatrix}
\end{array}
$$

- 计算左子树根的代价下界
  - ✓ 矩阵的第5行不包含0
  - ✓ 第5行元素减3，左子树根代价下界为：**96+3=99**
  - ✓ 结果矩阵如下

$$
\begin{array}{c|cccccc|}
 & j=1 & 2 & 3 & 4 & 5 & 7 \\
\hline
i=1 & \infty & 0 & 83 & 9 & 30 & 50 \\
2 & 0 & \infty & 66 & 37 & 17 & 26 \\
3 & 29 & 1 & \infty & 19 & 0 & 5 \\
5 & 0 & 18 & 53 & 4 & \infty & 25 \\
6 & 0 & 85 & 8 & \infty & 89 & 0 \\
7 & 18 & 0 & 0 & 0 & 58 & \infty \\
\hline
\end{array}
$$

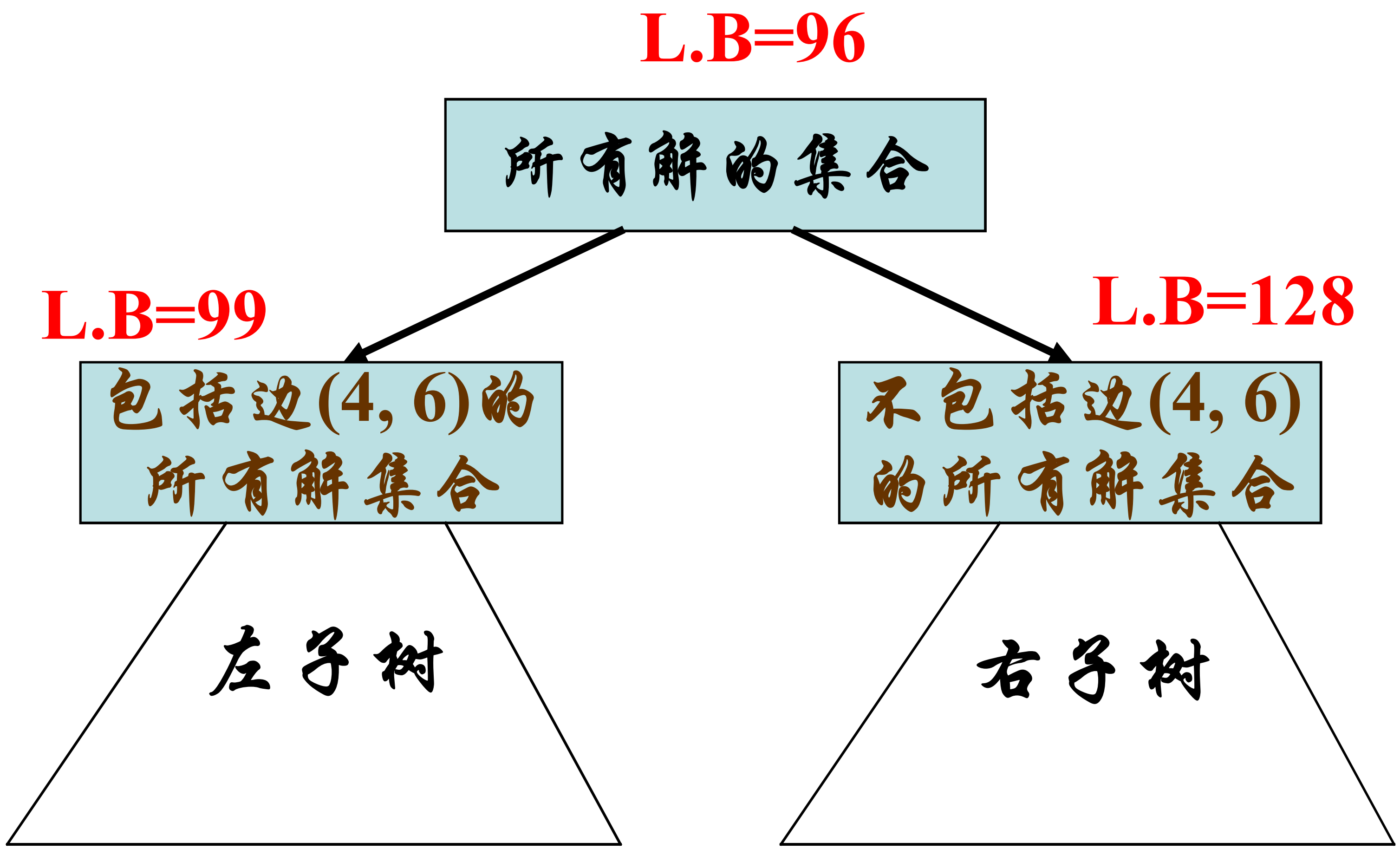
- 构造右子树根对应的代价矩阵
  - ✓ 右子节点为不包括边(4, 6)的所有解集合，只需要把 $C[4, 6]$ 设置为∞
  - ✓ 结果矩阵如下

$$
\begin{array}{cc}
 & \begin{array}{ccccccc} j=1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\
\begin{array}{c} i=1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} &
\begin{bmatrix}
\infty & 0 & 83 & 9 & 30 & 6 & 50 \\
0 & \infty & 66 & 37 & 17 & 12 & 26 \\
29 & 1 & \infty & 19 & 0 & 12 & 5 \\
32 & 83 & 66 & \infty & 49 & \infty & 80 \\
3 & 21 & 56 & 7 & \infty & 0 & 28 \\
0 & 85 & 8 & 42 & 89 & \infty & 0 \\
18 & 0 & 0 & 0 & 58 & 13 & \infty
\end{bmatrix}
\end{array}
$$

- ➢ 计算右子树根的代价下界
  - ✓ 矩阵的第4行不包含0
  - ✓ 第4行元素减32，右子树根代价下界为：**128**
  - ✓ 结果矩阵如下

$$
\begin{array}{c}
 \\ i=1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7
\end{array}
\begin{array}{c}
\begin{array}{ccccccc} j=1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\
\begin{bmatrix}
\infty & 0 & 83 & 9 & 30 & 6 & 50 \\
0 & \infty & 66 & 37 & 17 & 12 & 26 \\
29 & 1 & \infty & 19 & 0 & 12 & 5 \\
0 & 51 & 34 & \infty & 17 & \infty & 48 \\
3 & 21 & 56 & 7 & \infty & 0 & 28 \\
0 & 85 & 8 & 42 & 89 & \infty & 0 \\
18 & 0 & 0 & 0 & 58 & 13 & \infty
\end{bmatrix}
\end{array}
$$

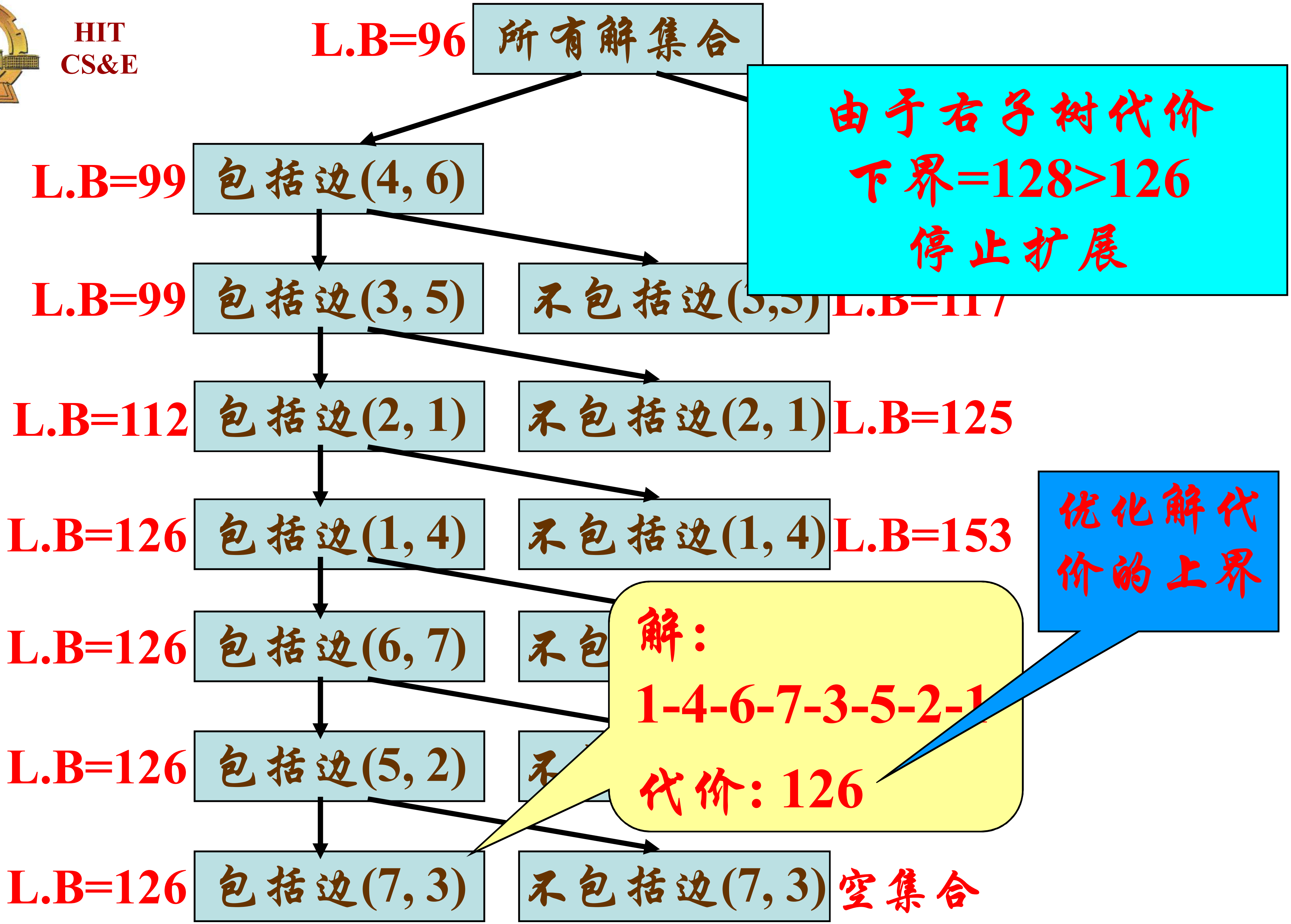
目前的树为



- 使用爬山策略扩展左子树根
  - ✓ 选择边使子节点代价下界增加最大的划分边(3, 5)
  - ✓ 左子节点为包括边(3, 5)的所有解集合
  - ✓ 右子节点为不包括边(3, 5)的所有解集合
  - ✓ 计算左、右子节点的代价下界: 99和117

- 目前树扩展为:

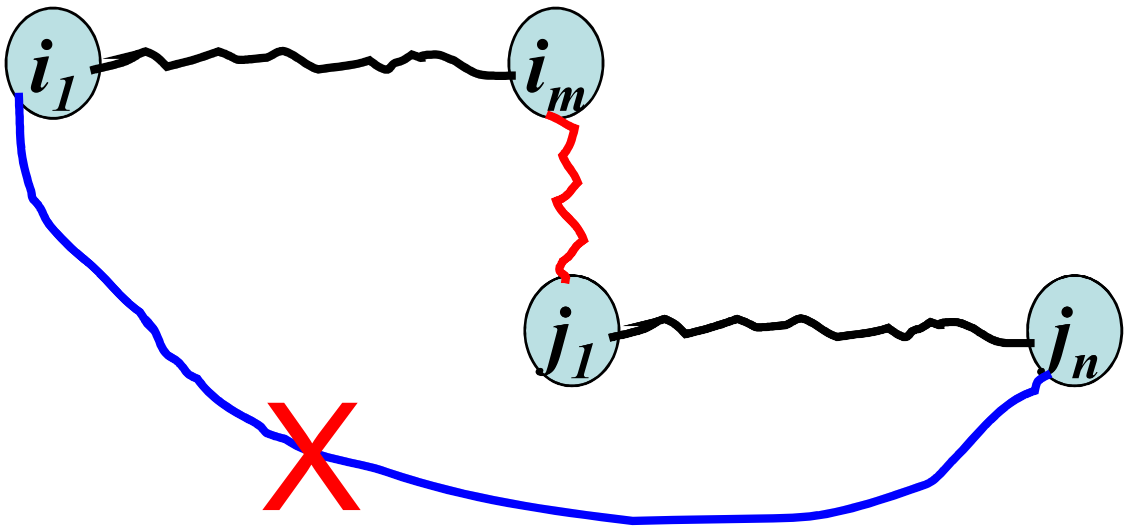
HIT
CS&E



L.B=96
所有解集合
由于右子树代价
下界=128>126
停止扩展
L.B=99
包括边(4, 6)
L.B=99
包括边(3, 5)
不包括边(3,5)
L.B=117
L.B=112
包括边(2, 1)
不包括边(2, 1)
L.B=125
L.B=126
包括边(1, 4)
不包括边(1, 4)
L.B=153
优化解代价的上界
L.B=126
包括边(6, 7)
解：
1-4-6-7-3-5-2-1
代价：126
L.B=126
包括边(5, 2)
L.B=126
包括边(7, 3)
不包括边(7, 3)
空集合

注 意

如果i_1-i_2-…-i_m和j_1-j_2-…-j_m已被包含在一个正在构造的路径中，(i_m, j_1)被加入，则必须避免j_n到 i_1的路径被加入。否则出现环。



8.6 0-1 *backpacking problem*

- 问题的定义
- 转换为树搜索问题
- 分支界限搜索算法

问题定义

给定*n*种物品和一个背包，物品*i*的重量是w_i，价值v_i，背包承重为*C*，问如何选择装入背包的物品，使装入背包中的物品的总价值最大？

对于每种物品只能选择完全装入或不装入，一个物品至多装入一次。

- 输入：$C>0,\ w_i>0,\ v_i>0,\ 1\leq i\leq n$
- 输出：$(x_1, x_2, \ldots, x_n),\ x_i \in \{0, 1\}$，满足

$$\sum_{1\leq i\leq n} w_i x_i \leq C,\quad \sum_{1\leq i\leq n} v_i x_i \text{ 最大}$$

转换为树搜索问题

- 空包为树根，代价下界*LB*,代价的上界*UB*;
  - 贪心算法可行解得*LB*
  - 分数背包问题的优化解代价*UB*
- 用爬山法依次考虑每个物品的取舍

递归地划分解空间，得到二叉树

- 划分过程：$(x_1,\ldots,x_k)\rightarrow(x_1,\ldots,x_k,x_{k+1})$

✓左子树，将第k+1个物品放入背包$(x_1,\ldots,x_k, 1)$

计算节点代价的下界LB,上界UB

✓右子树，将第k+1个物品舍弃,$(x_1,\ldots,x_k, 0)$

计算节点代价的下界LB,上界UB

计算节点的下、上界

- 计算结点的代价下界*LB*和上界*UB*;

  已经发现的可行解的代价opt

  $V=v_1x_1+\ldots+v_kx_k$

  待求解的子问题C-$(w_1x_1+\ldots+w_kx_k)$

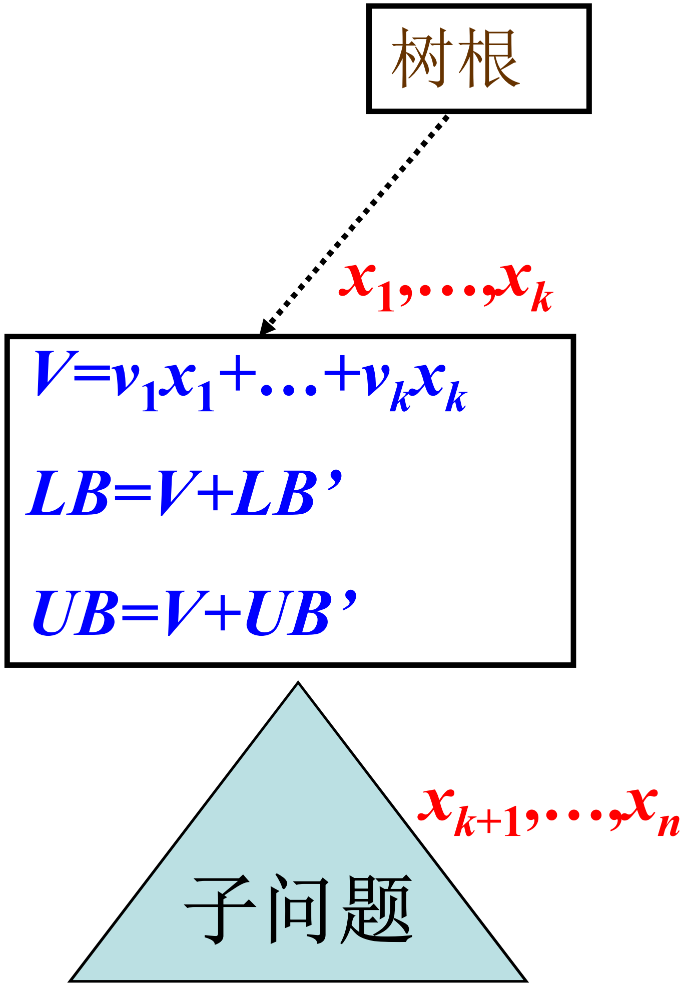
  $w_{k+1},\ldots,w_n$

  $v_{k+1},\ldots,v_n$

  贪心算法在子问题上的解*LB'*

  分数背包算法在子问题上的解*UB'*

- $LB=V+LB'$
- $UB=V+UB'$



LB':贪心算法可行解

UB':分数背包算法可行解

- 空包为树根，代价下界*LB*,代价的上界*UB*;
  - 贪心算法可行解得*LB*
  - 分数背包问题的优化解代价*UB*
  - *opt*=0,用于记录当前发现最优可行解的代价
- 用爬山法取舍第k+1个物品,递归地划分解空间

划分过程: $(x_1,\ldots,x_k) \rightarrow (x_1,\ldots,x_k,x_{k+1})$

- (1) $C<w_1x_1+\ldots+w_{k+1}x_{k+1}$,$(x_1,\ldots,x_k,x_{k+1})$不可行，舍弃
- (2)*UB*=*LB*—记录opt=UB,终止扩展$(x_1,\ldots,x_k,x_{k+1})$
  - 在剩下的子问题C-w_1x_1-...-$w_{k+1}x_{k+1}$上，贪心策略将得到最优解
  - 贪心算法的解$(x_{k+2},\ldots,x_n)$,与$(x_1,\ldots,x_k,x_{k+1})$构成优化解
- (3)UB<opt, 扩展$(x_1,\ldots,x_k,x_{k+1})$得不到优于opt的解，舍弃
- (4)其他情况，继续扩展

0-1背包问题搜索实例

构造树根

C=50

编号i	1	2	3	4	5	6
价值v_i	60	100	120	140	30	40
重量w_i	10	20	30	35	10	20
v_i/w_i	6	5	4	4	3	2

Opt=0

$V=0$, LB=190, UB=240

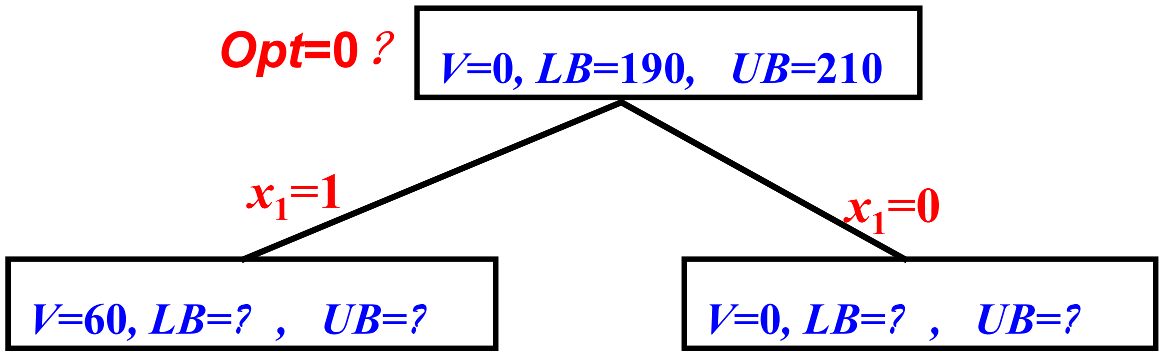
$V=0$

$LB=0+(60\times1+100\times1+120\times0+140\times0+30\times1+40\times0)$
$=190$

$UB=0+(60\times1+100\times1+120\times(50-10-20)/30$
$=240$

HIT
CS&E

第一个物品的取舍



C=50

编号i	1	2	3	4	5	6
价值v_i	60	100	120	140	30	40
重量w_i	10	20	30	35	10	20
v_i/w_i	6	5	4	4	3	2

左子树 V=60 C'=50-10=40

LB=60+(100 × 1+120 × 0+140× 0+30× 1+40× 0)

=190

UB=60+(100 × 1+120 ×(40-20)/30

=240

右子树 V=0 C'=50-0=50

LB=0+(100 × 1+120 × 1+140× 0+30× 0+40× 0)

=220

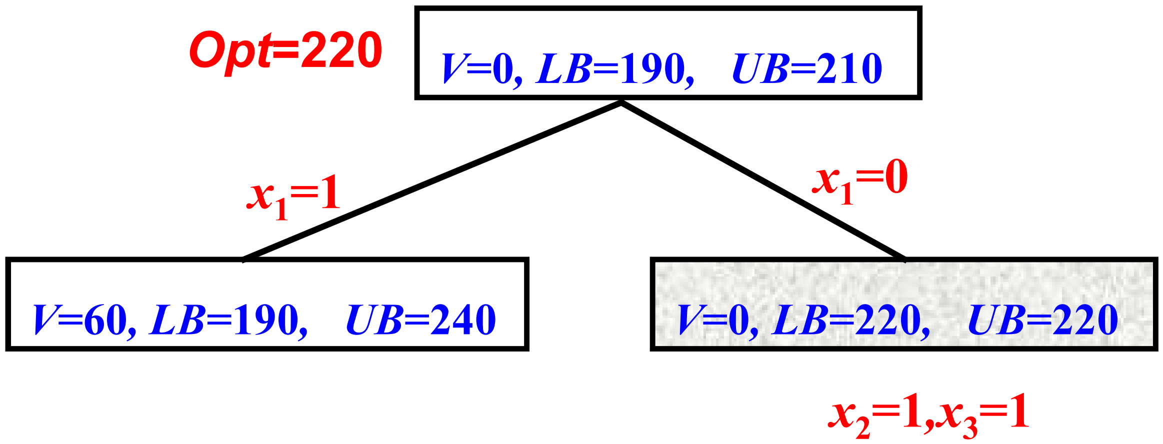
UB=0+(100 × 1+120 ×1

=220

0-1背包问题搜索实例

第一个物品的取舍

Opt=220



$C=50$

编号i	1	2	3	4	5	6
价值v_i	60	100	120	140	30	40
重量w_i	10	20	30	35	10	20
v_i/w_i	6	5	4	4	3	2

左子树 $V=60$ $C'=50-10=40$

$LB=60+(100\times 1+120\times 0+140\times 0+30\times 1+40\times 0)$

$=190$

$UB=60+(100\times 1+120\times(40-20)/30$

$=240$

右子树 $V=0$ $C'=50-0=50$

$LB=0+(100\times 1+120\times 1+140\times 0+30\times 0+40\times 0)$

$=220$

$UB=0+(100\times 1+120\times 1$

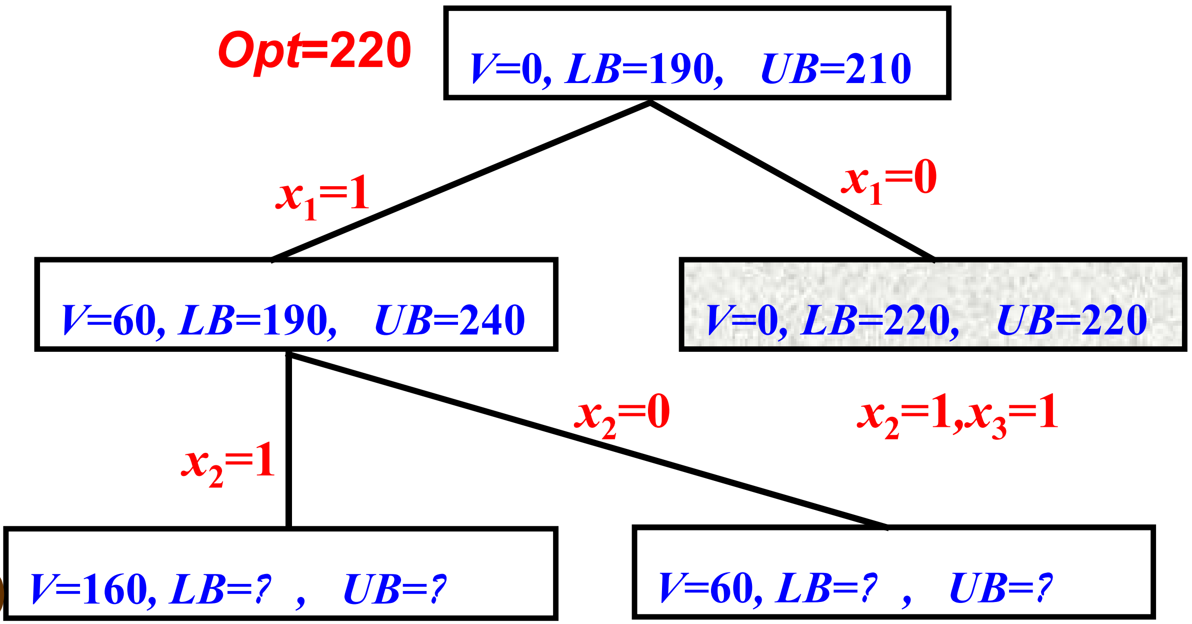
$=220$

爬山法

第二个物品的取舍

C=50

编号i	1	2	3	4	5	6
价值v_i	60	100	120	140	30	40
重量w_i	10	20	30	35	10	20
v_i/w_i	6	5	4	4	3	2



左子树 V=160 C'=50-10-20=20

LB=160+(120 × 0+140× 0+30× 1+40× 0)

=190

UB=160+(120 ×20/30)

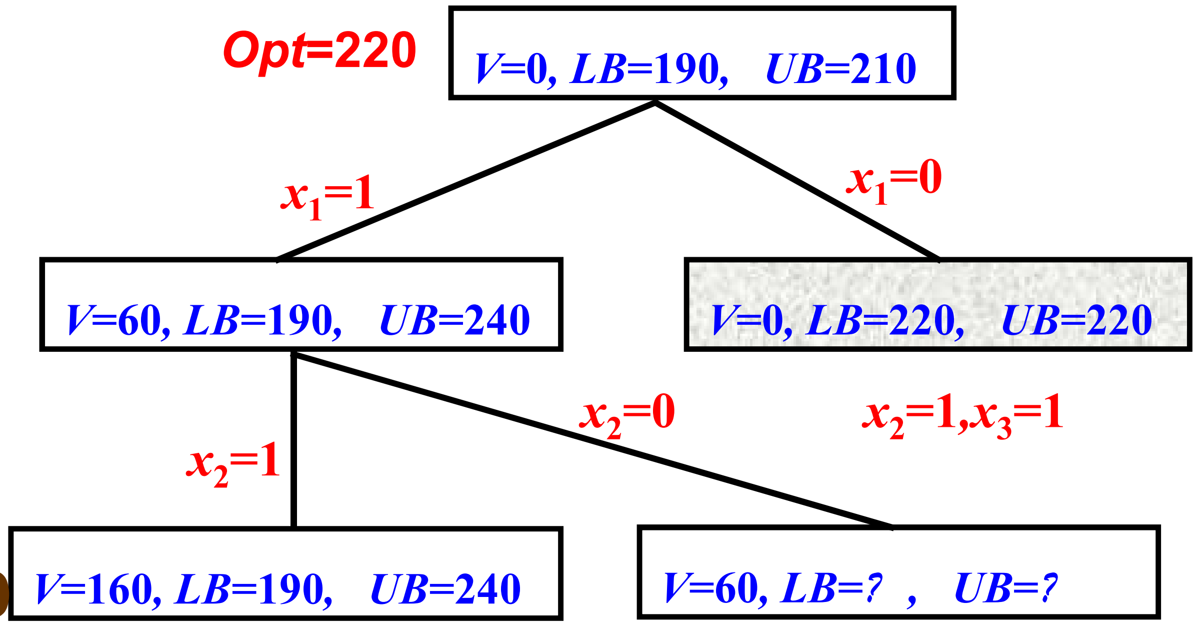
=240

爬山法

第二个物品的取舍

C=50

编号i	1	2	3	4	5	6
价值v_i	60	100	120	140	30	40
重量w_i	10	20	30	35	10	20
v_i/w_i	6	5	4	4	3	2



左子树 V=160 C'=50-10-20=20

LB=160+(120 × 0+140× 0+30× 1+40× 0)

=190

UB=160+(120 ×20/30)

=240

右子树 V=60 C'=50-10=40

LB=60+(120 × 1+140× 0+30× 1+40× 0)

=210

UB=160+(120 ×1+140×(40-30)/35)

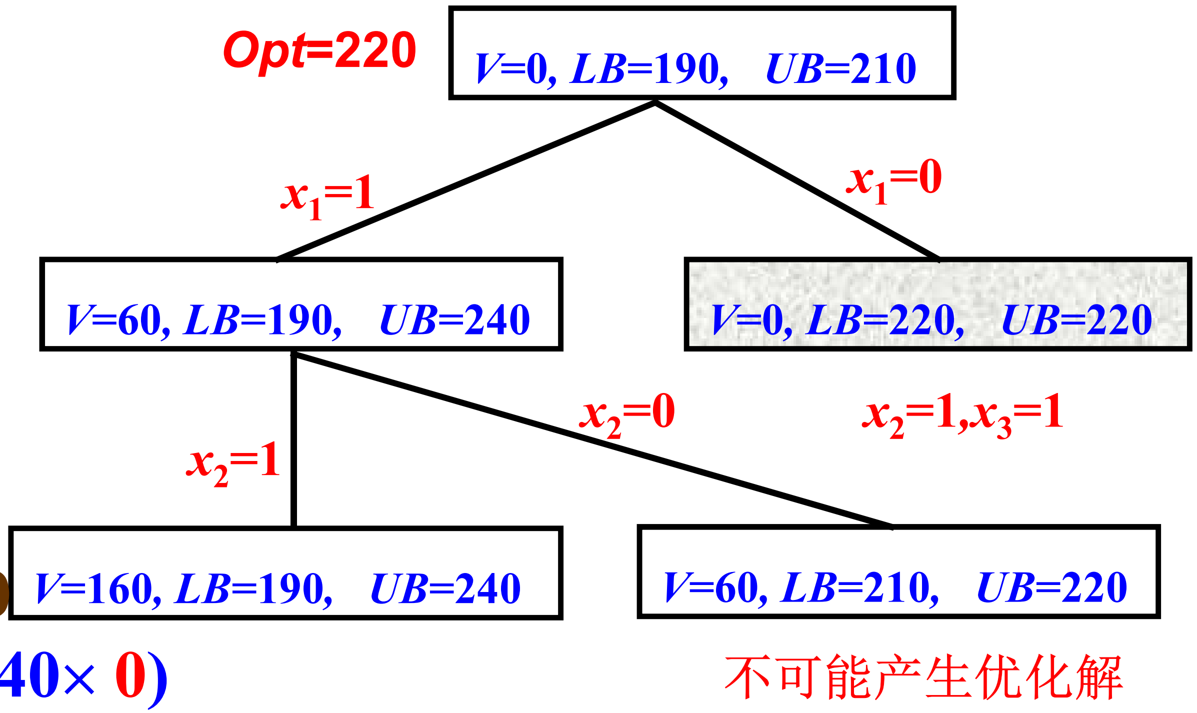
=220

爬山法

第二个物品的取舍

C=50

编号i	1	2	3	4	5	6
价值v_i	60	100	120	140	30	40
重量w_i	10	20	30	35	10	20
v_i/w_i	6	5	4	4	3	2



左子树 $V=160$ C'=50-10-20=20

$LB=160+(120\times 0+140\times 0+30\times 1+40\times 0)$

$=190$

$UB=160+(120\times 20/30)$

$=240$

右子树 $V=60$ C'=50-10=40

$LB=60+(120\times 1+140\times 0+30\times 1+40\times 0)$

$=210$

$UB=160+(120\times 1+140\times (40-30)/35)$

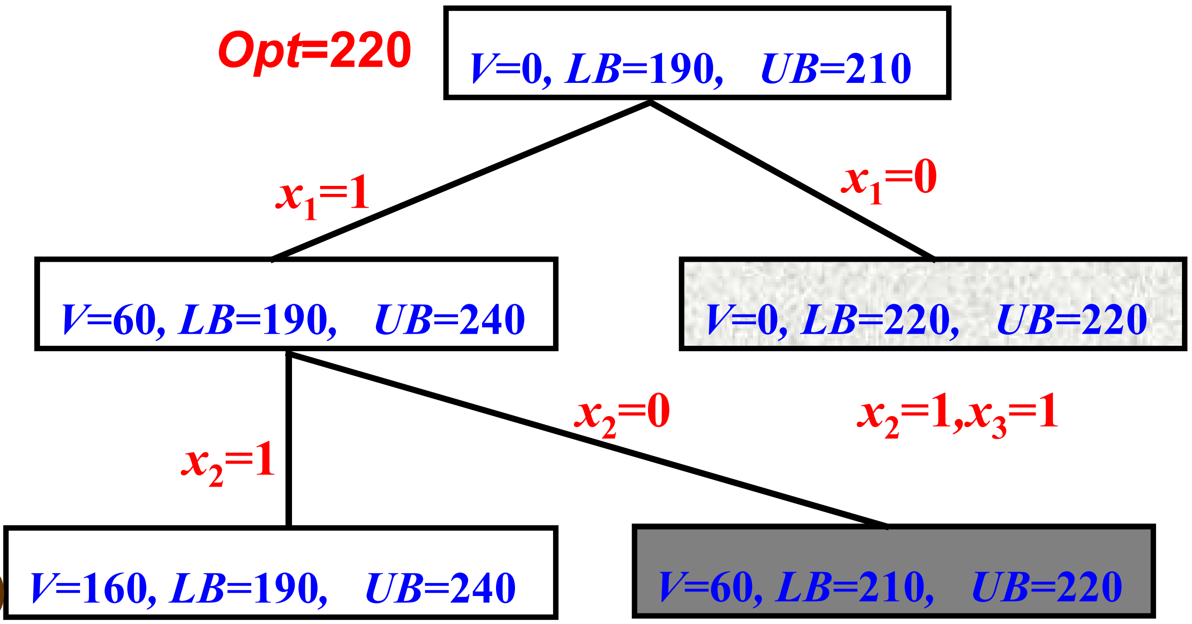
$=220$

爬山法

第二个物品的取舍

C=50

编号i	1	2	3	4	5	6
价值v_i	60	100	120	140	30	40
重量w_i	10	20	30	35	10	20
v_i/w_i	6	5	4	4	3	2



左子树 V=160 C'=50-10-20=20

LB=160+(120 × 0+140× 0+30× 1+40× 0)

=190

UB=160+(120 ×20/30)

=240

右子树 V=60 C'=50-10=40

LB=60+(120 × 1+140× 0+30× 1+40× 0)

=210

UB=160+(120 ×1+140×(40-30)/35)

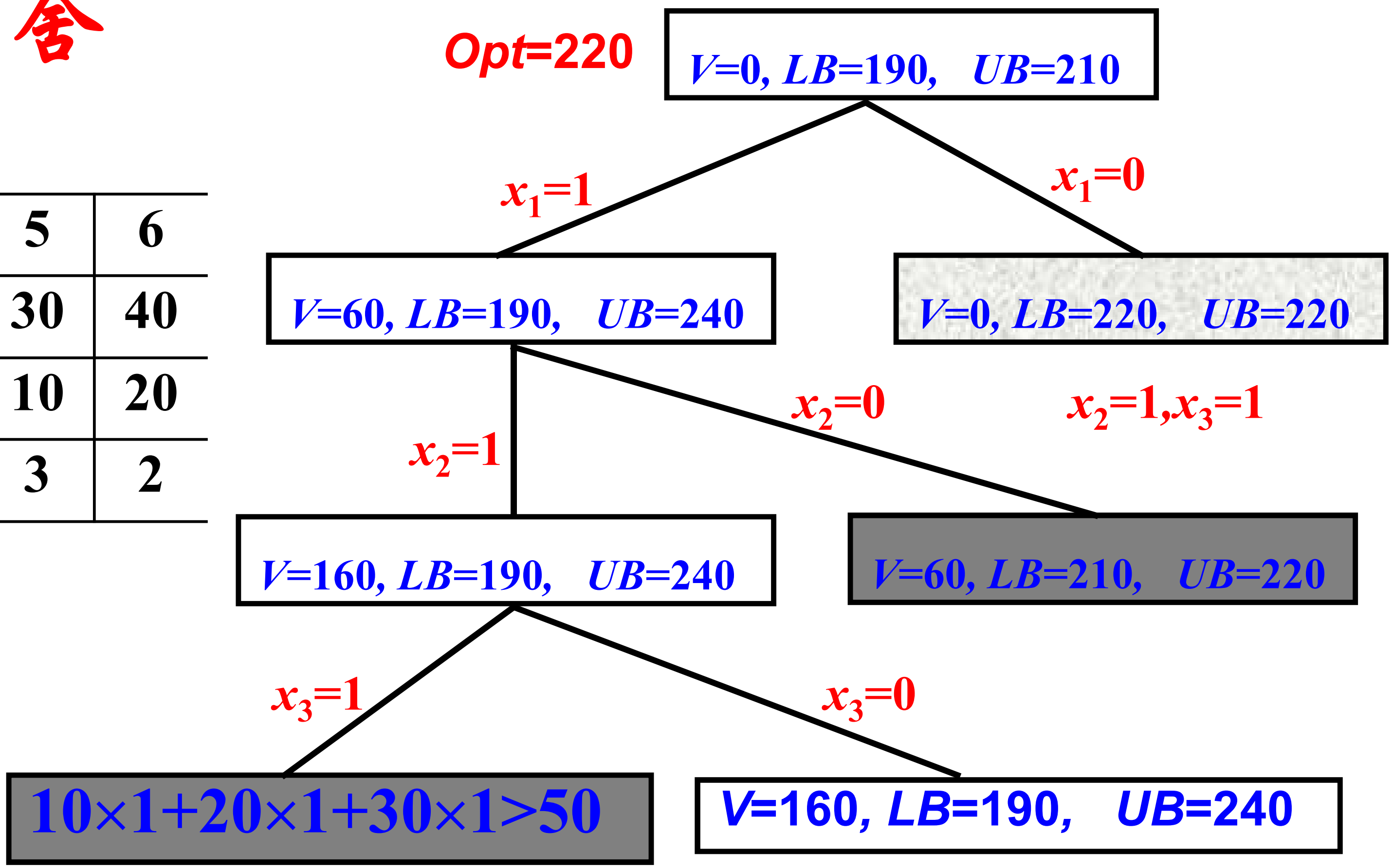
=220

爬山法

第二个物品的取舍

C=50

编号i	1	2	3	4	5	6
价值v_i	60	100	120	140	30	40
重量w_i	10	20	30	35	10	20
v_i/w_i	6	5	4	4	3	2

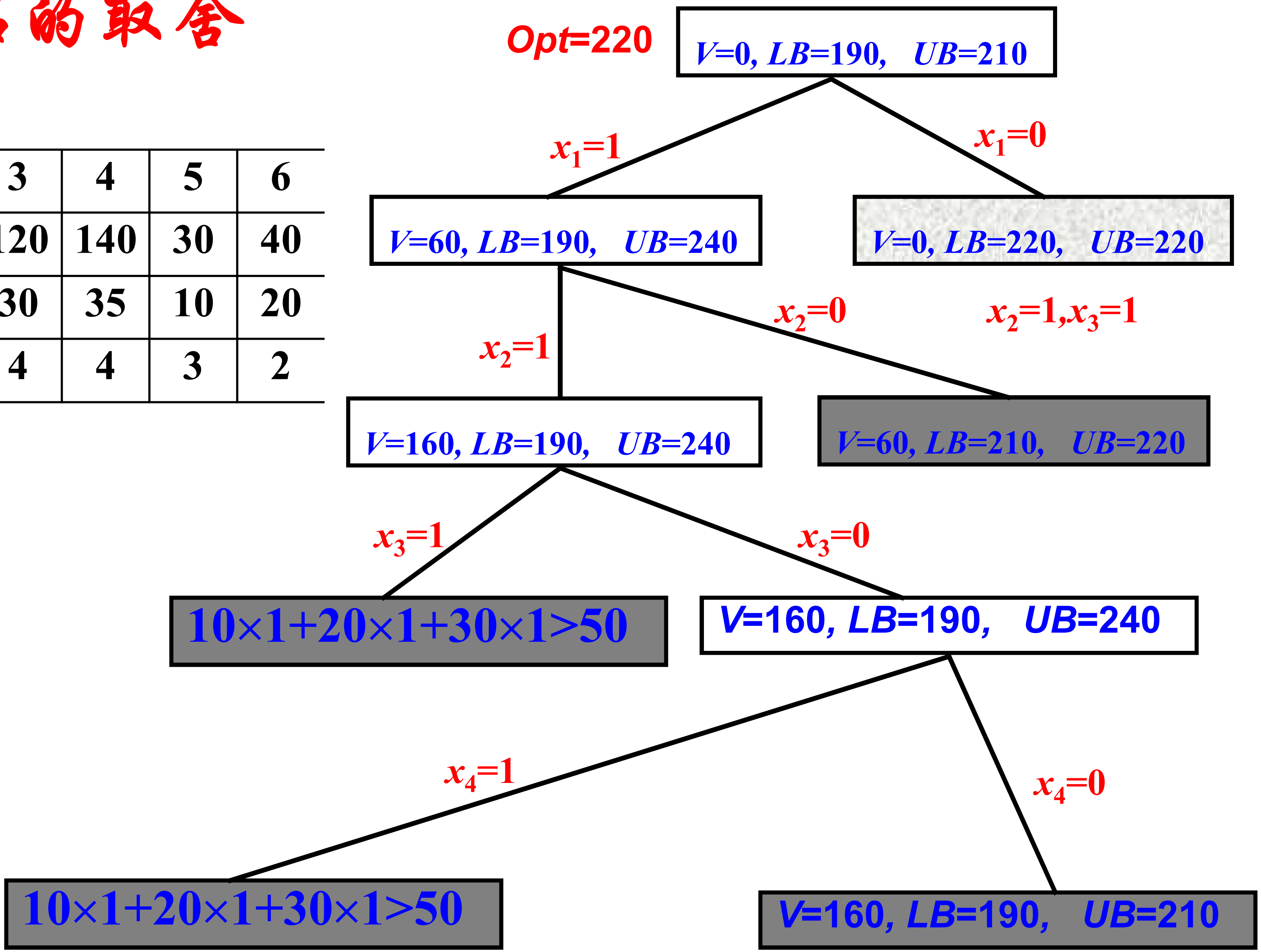


爬山法

第二个物品的取舍

C=50

编号i	1	2	3	4	5	6
价值v_i	60	100	120	140	30	40
重量w_i	10	20	30	35	10	20
v_i/w_i	6	5	4	4	3	2



8.7 The A* Algorithm

- A*算法的基本思想
- A*算法的规则
- 应用A*算法求解最短路径问题

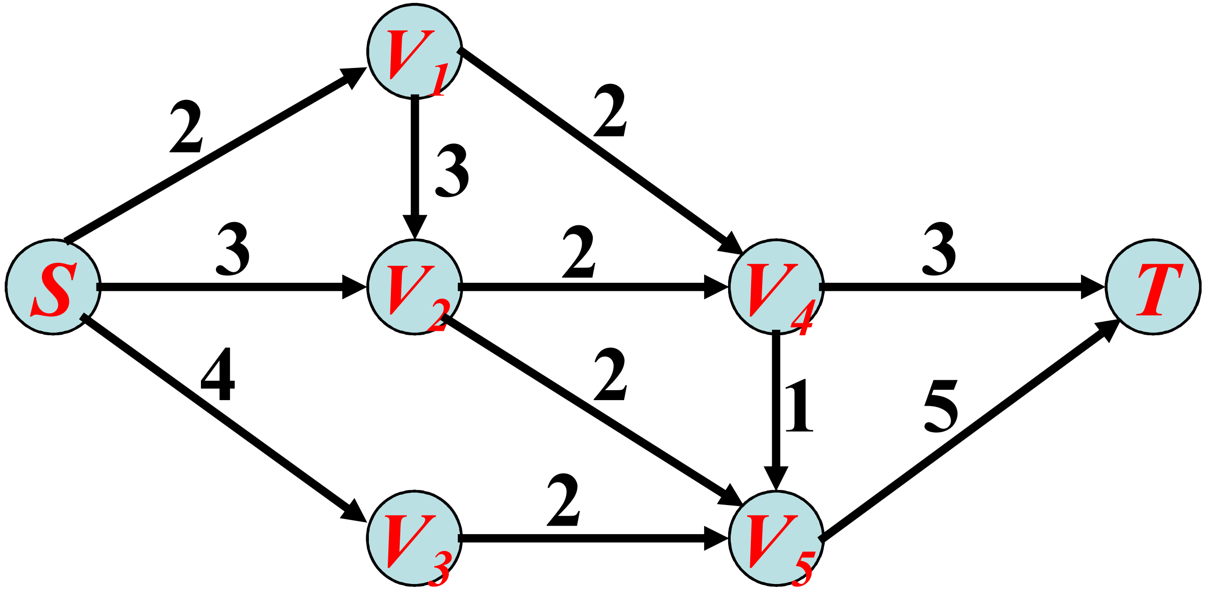
A*算法的基本思想

- A*算法与分支界限策略的比较
  - 分支界限策略是为了剪掉不能达到优化解的分支
  - 分支界限策略的关键是“界限”
  - A*算法的核心是告诉我们在某些情况下，我们得到的解一定是优化解，于是算法可以停止
  - A*算法试图尽早地发现优化解
  - A*算法经常使用Best-first策略求解优化问题
  - P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths in graphs. IEEE Trans. Syst. Sci. and Cybernetics, SSC-4(2):100-107, 1968

- A*算法关键——代价函数
  - 对于任意节点n
    - $g(n)$＝从树根到n的代价
    - $h^*(n)$＝从n到目标节点的优化路径的代价
    - $f^*(n)＝g(n)＋h^*(n)$是节点n的代价
  - What is the value of $h^*(n)$?
    - 不知道！
    - 于是，$f^*(n)$也不知道
  - 估计$h^*(n)$
    - 使用任何方法去估计$h^*(n)$，用$h(n)$表示$h^*(n)$的估计
    - $h(n) \le h^*(n)$总为真
    - $f(n)=g(n)+h(n) \le g(n)+h^*(n)=f^*(n)$定义为$n$的代价

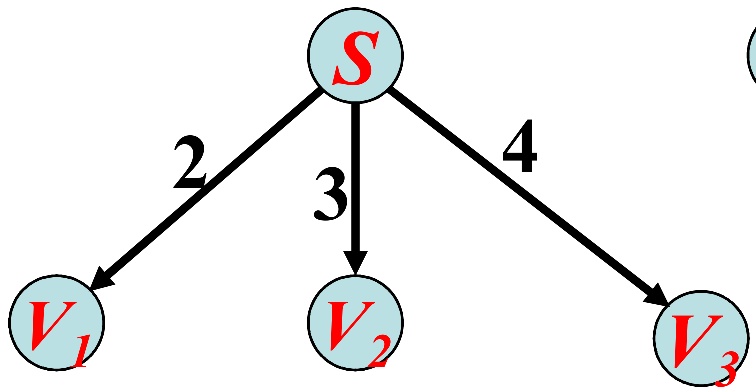
例1. 最短路径问题：

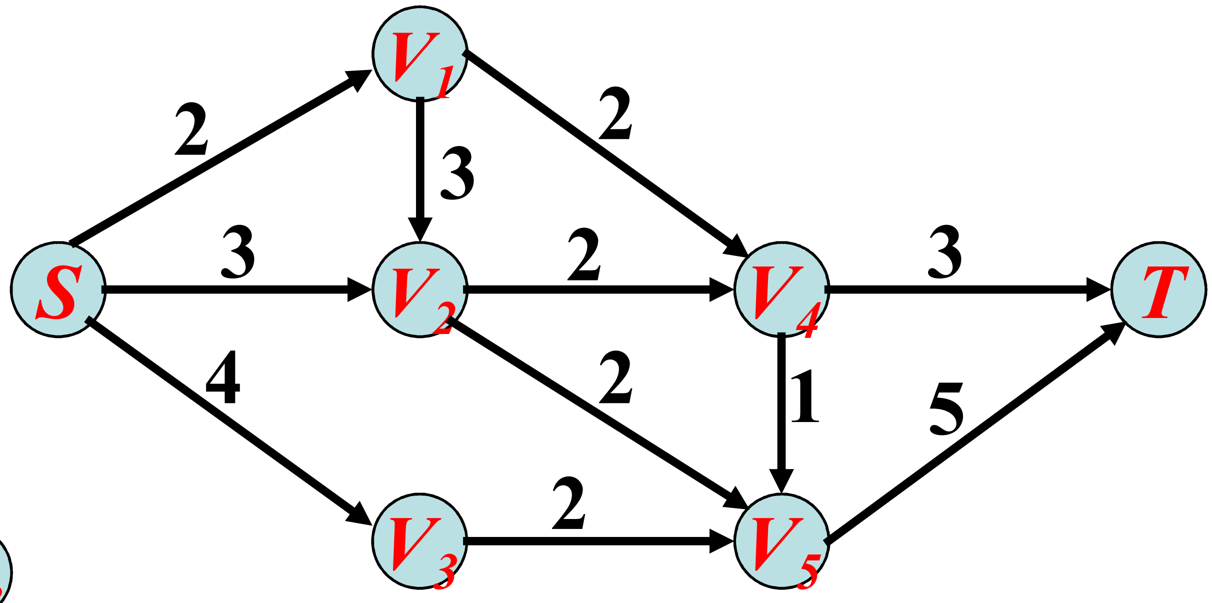
- 输入：



- 输出：发现一个从*S*到*T*的最短路径

– 求解树的第一阶段





$g(V_1)=2,\ g(V_2)=3,\ g(V_3)=4$

$\cancel{h^*(V_1)=5,\ f^*(V_1)=g(V_1)+h^*(V_1)=7}$

– 估计 $h^*(n)$

- 从 V_1 出发有两种可能：代价为2，代价为3，最小者为2
- $h^*(V_1)\geq 2$，选择 $h(n)=2$ 为 $h^*(V_1)$ 的估计值
- $f(V_1)=g(v_1)+h(V_1)=4$ 为 V_1 的代价

- A*算法本质—已经发现的解是优化解

定理1. 使用Best-first策略搜索树，如果A*选择的节点是目标节点，则该节点表示的解是优化解.

证明.

令n是任意扩展到的节点，t是选中目标节点.

往证$f(t)=g(t)$是优化解代价.

(1). A*算法使用Best-first策略，$f(t)\le f(n)$.

(2). A*算法使用$h(n)\le h^*(n)$估计规则，$f(t)\le f(n)\le f^*(n)$.

(3). $\{f^*(n)\}$中必有一个为优化解的代价，令其为$f^*(s)$.

我们有$f(t)\le f^*(s)$.

(4). t是目标节点$h(t)=0$，所以$f(t)=g(t)+h(t)=g(t)\le f^*(s)$.

(5). $f(t)=g(t)$是一个可能解，$g(t)\ge f^*(s)$，$f(t)=g(t)=f^*(s)$.

A*算法的规则

(1). 使用Best-first策略搜索树;

(2). 节点n的代价函数为$f(n)=g(n)+h(n)$, $g(n)$是从根到n的路径代价, $h(n)$是从n到某个目标节点的优化路径代价;

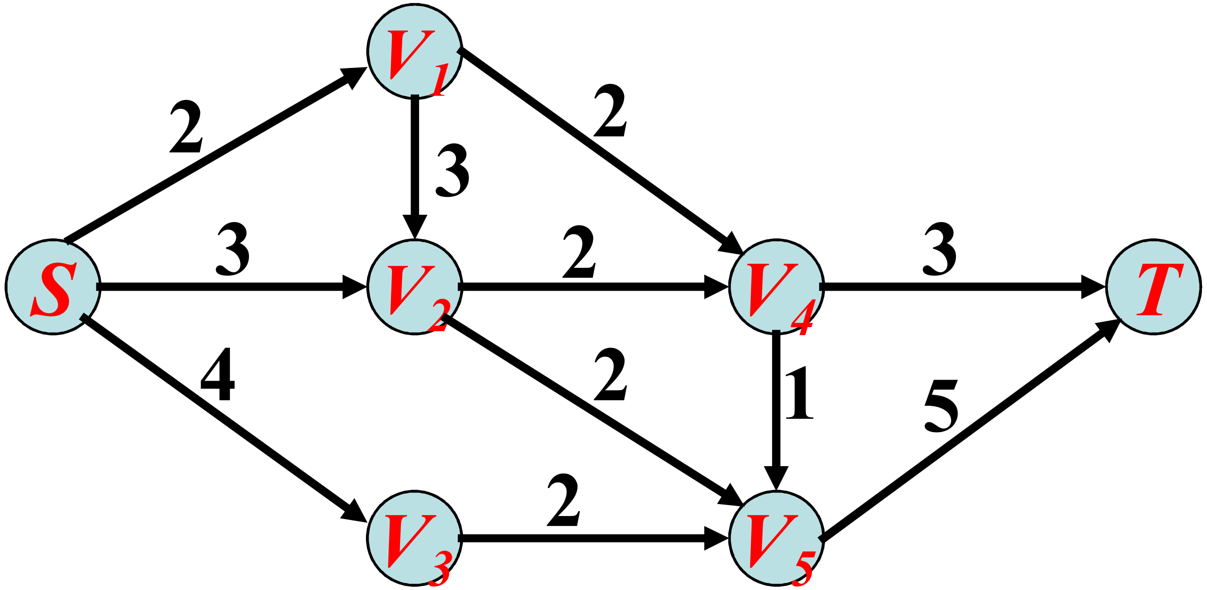
(3). 对于所有n, $h(n)\leq h^*(n)$;

(4). 当选择到的节点是目标节点时, 算法停止, 返回一个优化解.

HIT
CS&E

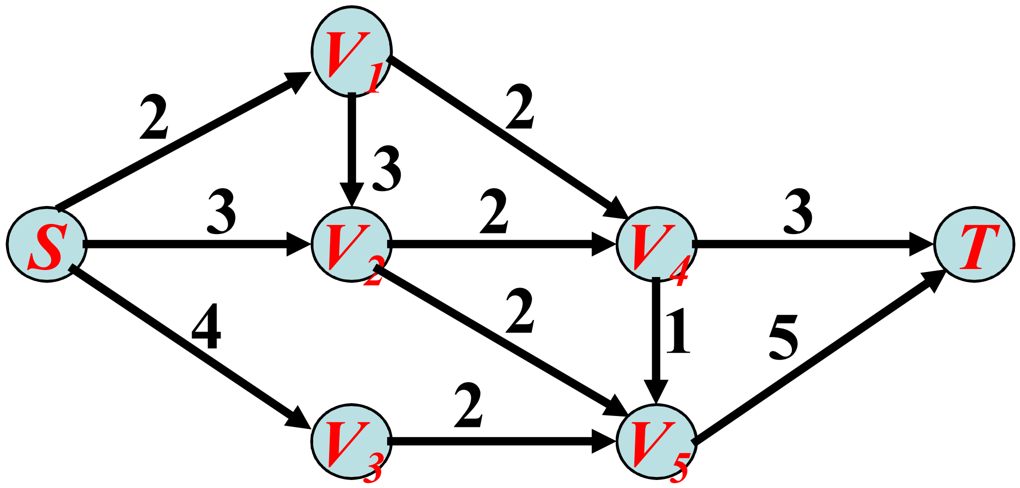
应用A*算法求解最短路径问题

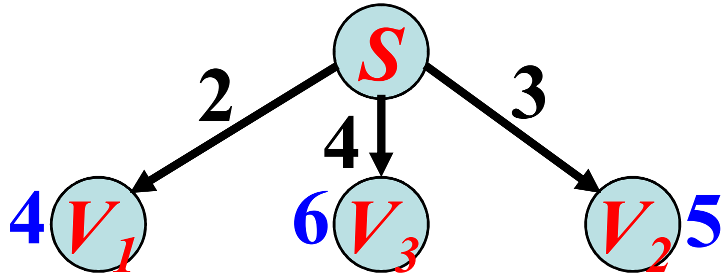
- 问题的输入:



- A*算法的执行全过程

Step 1



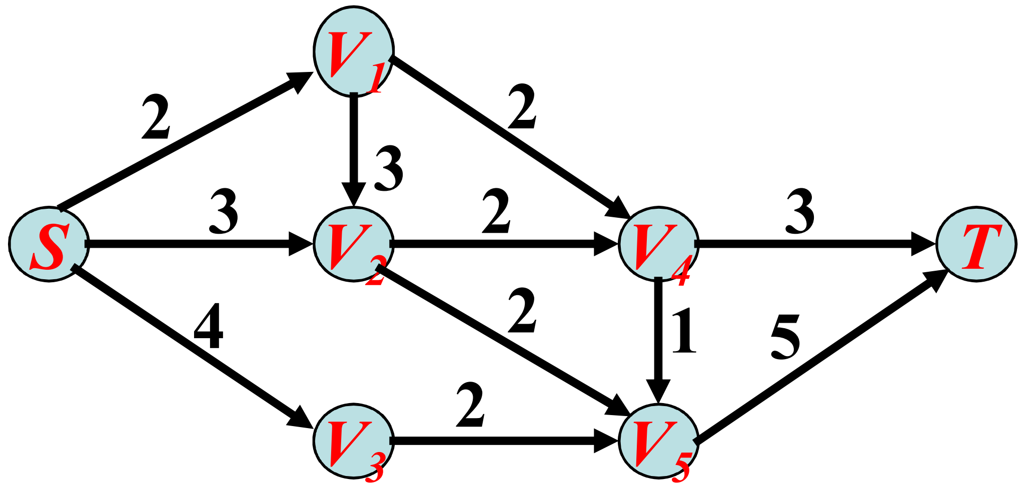


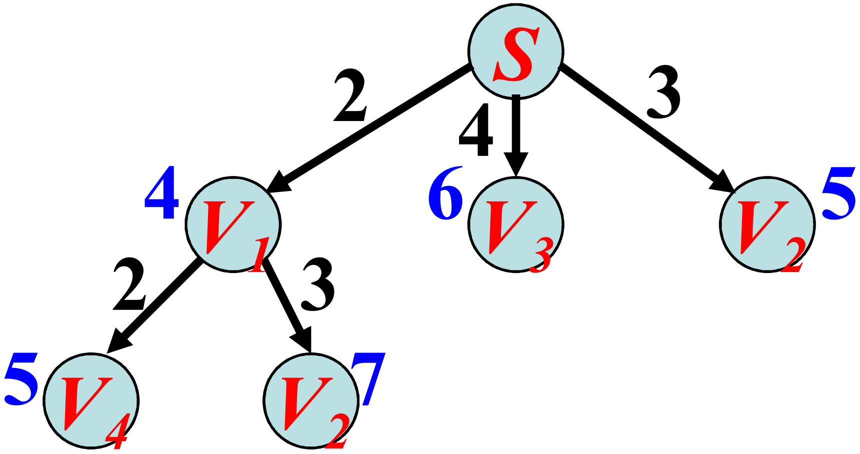
$g(V_1)=2$ $h(V_1)=min\{2,3\}=2$ $f(V_1)=2+2=4$

$g(V_3)=4$ $h(V_3)=min\{2\}=2$ $f(V_3)=4+2=6$

$g(V_2)=3$ $h(V_2)=min\{2,2\}=2$ $f(V_2)=2+2=5$

Step 2. 扩展V_1

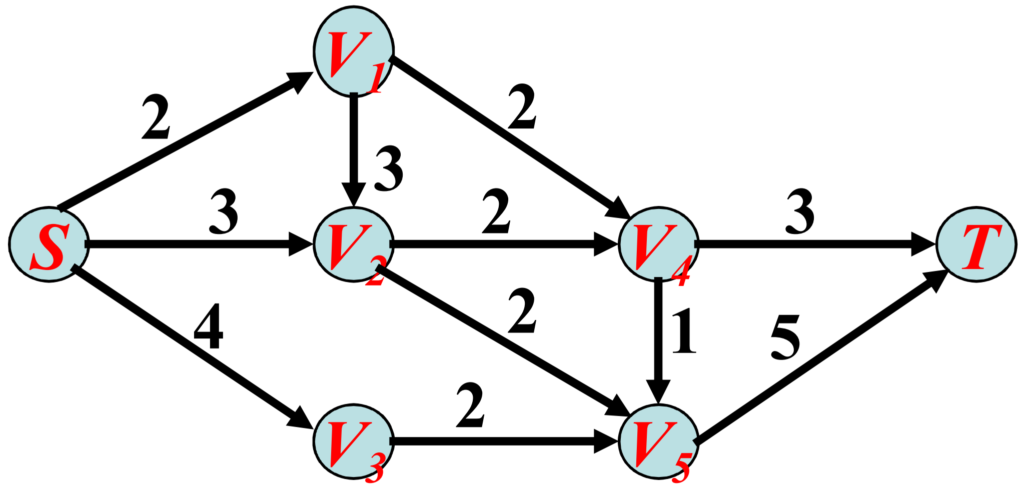


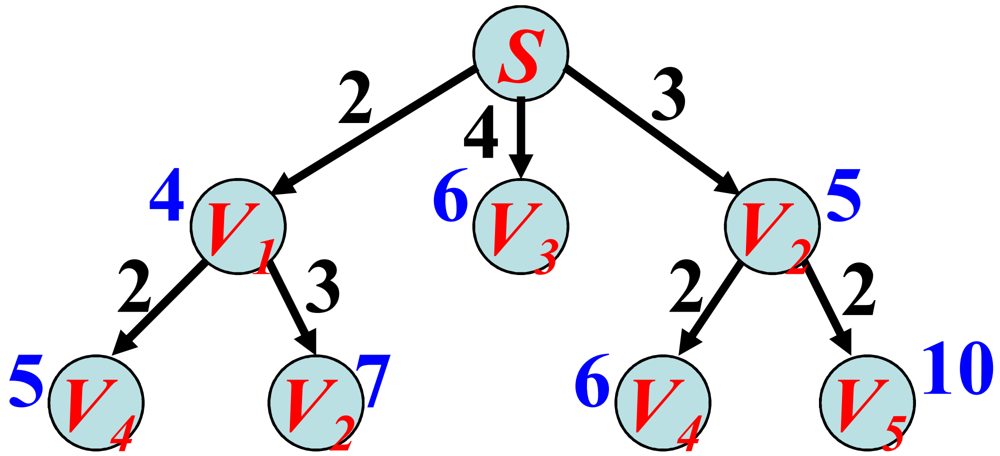


$g(V_4)=2+2=4 \quad h(V_4)=min\{3,1\}=1 \quad f(V_4)=4+1=5$

$g(V_2)=2+3=5 \quad h(V_2)=min\{2, 2\}=2 \quad f(V_2)=5+2=7$

Step 3. 扩展 V_2

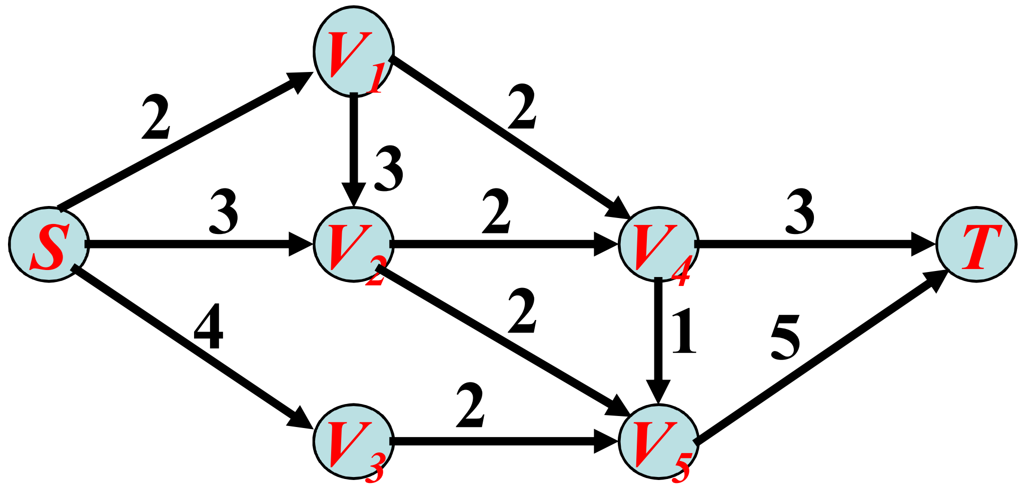


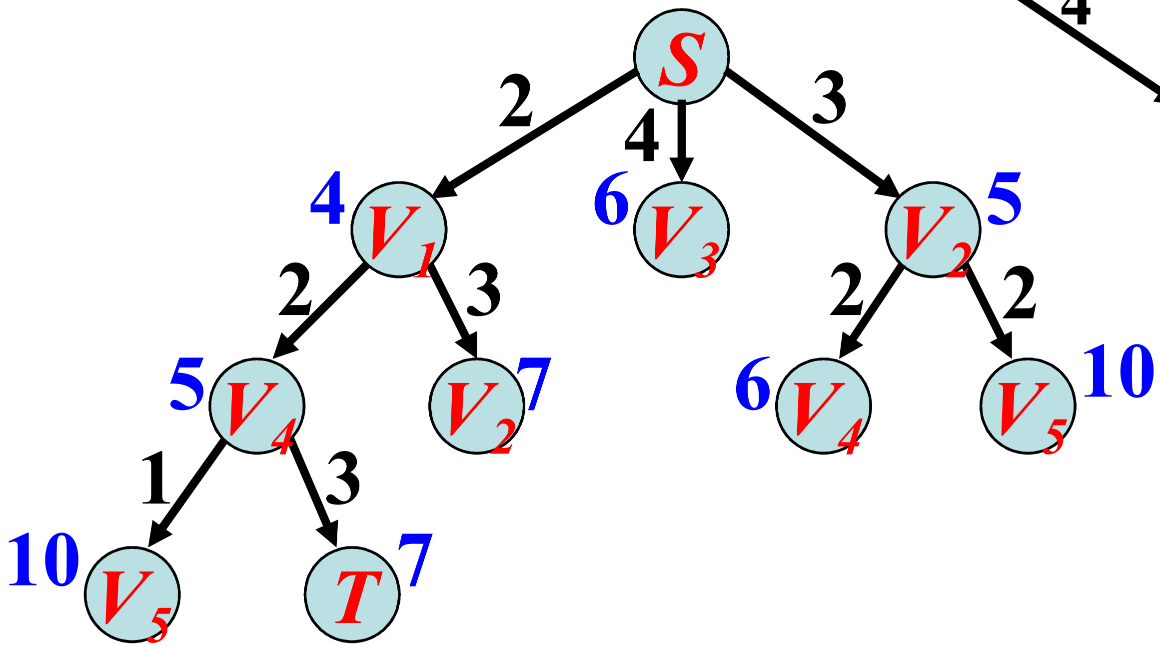


$$g(V_4)=3+2=5 \quad h(V_4)=min\{3,1\}=1 \quad f(V_4)=5+1=6$$

$$g(V_5)=3+2=5 \quad h(V_5)=min\{5\}=5 \quad f(V_2)=5+5=10$$

Step 4. 扩展V_4

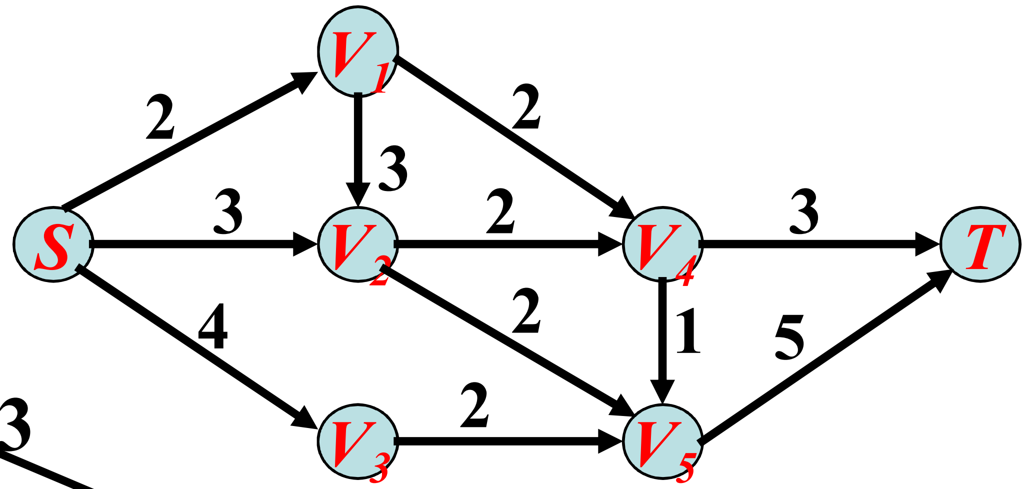


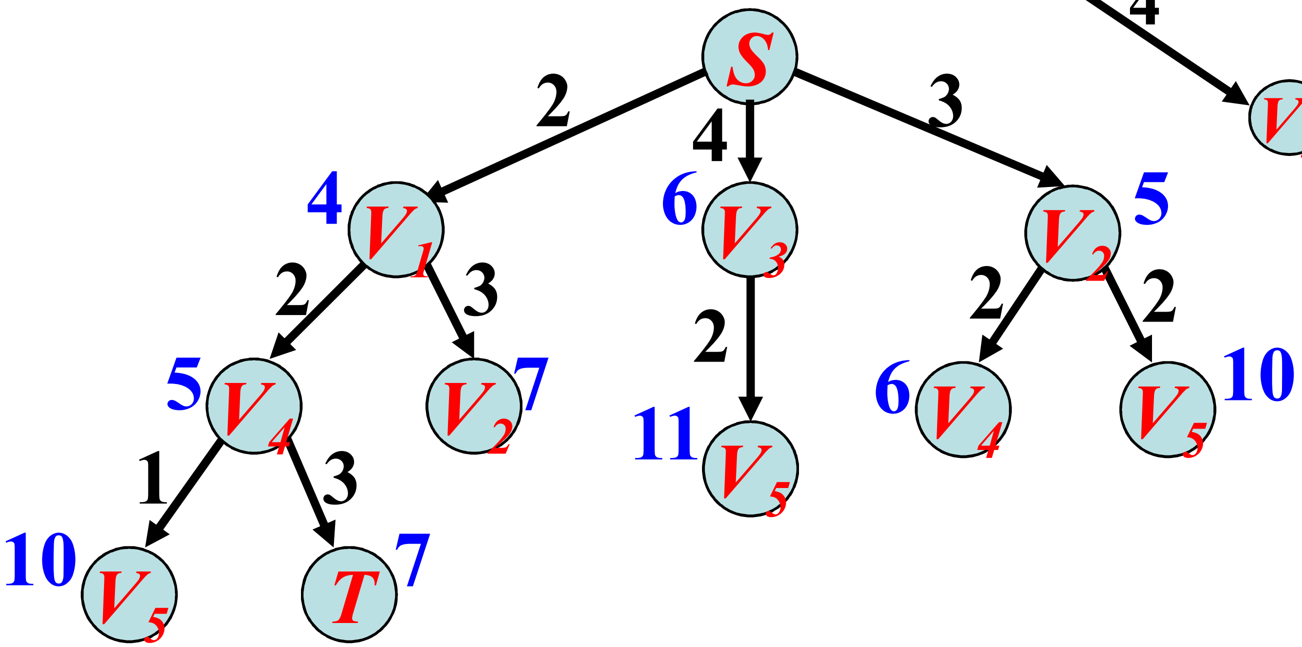


$g(V_5)=2+2+1=5$ $h(V_5)=min\{5\}=5$ $f(V_5)=5+5=10$

$g(T)=2+2+3=7$ $h(T)=0$ $f(V_2)=7+0=7$

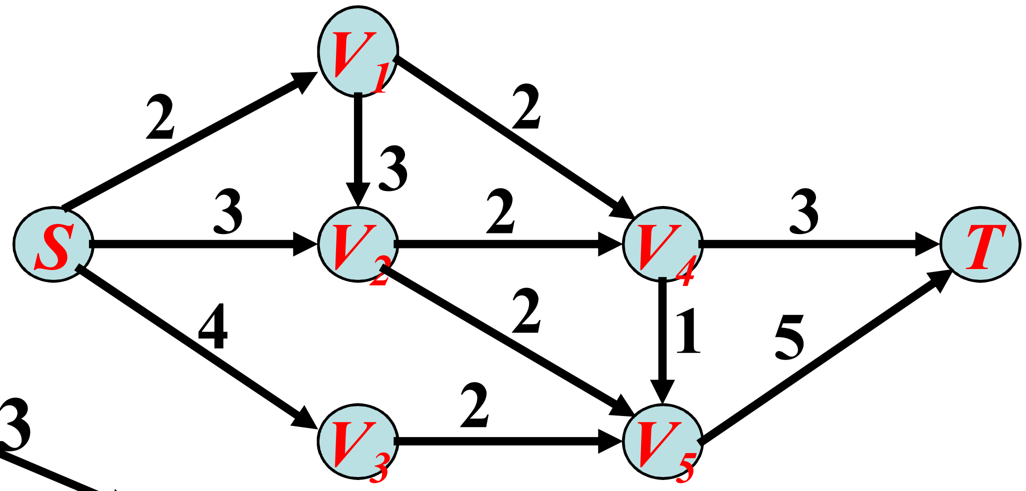
Step 5. 扩展V_3

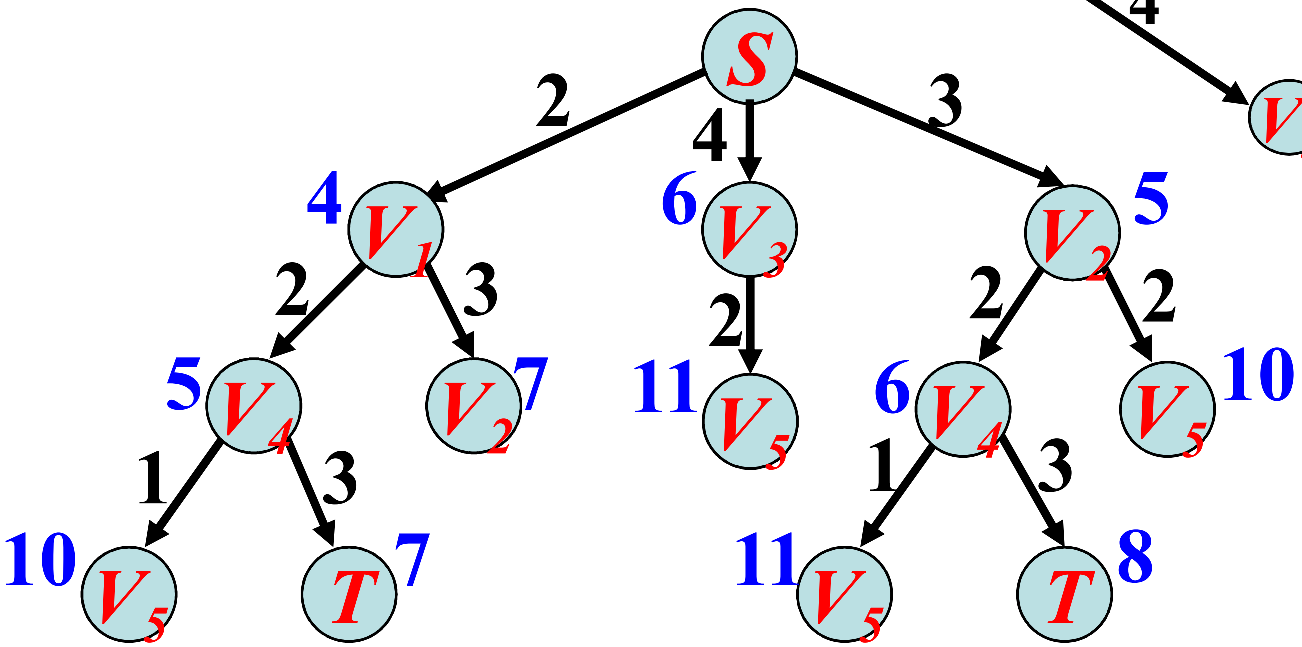




$g(V_5)=4+2=6$ $h(V_5)=min\{5\}=5$ $f(V_5)=6+5=11$

Step 6. 扩展V_4

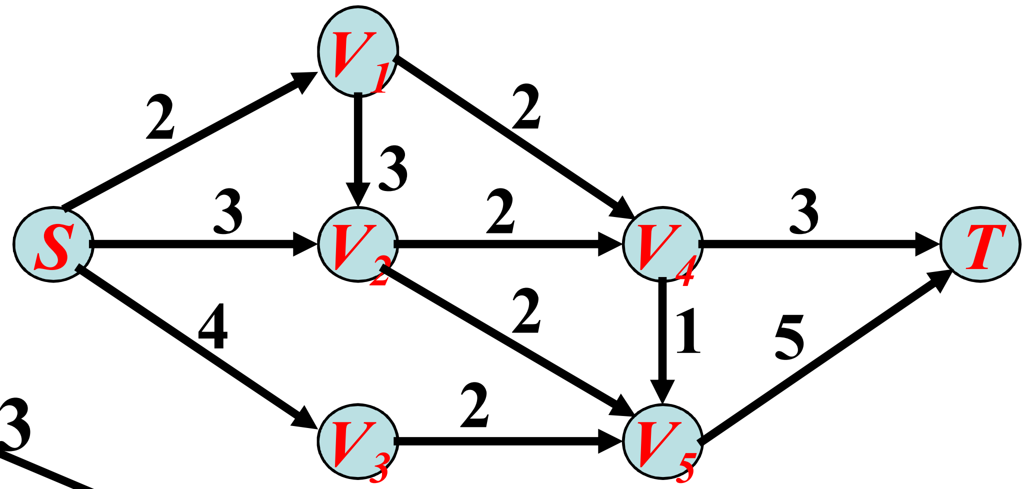


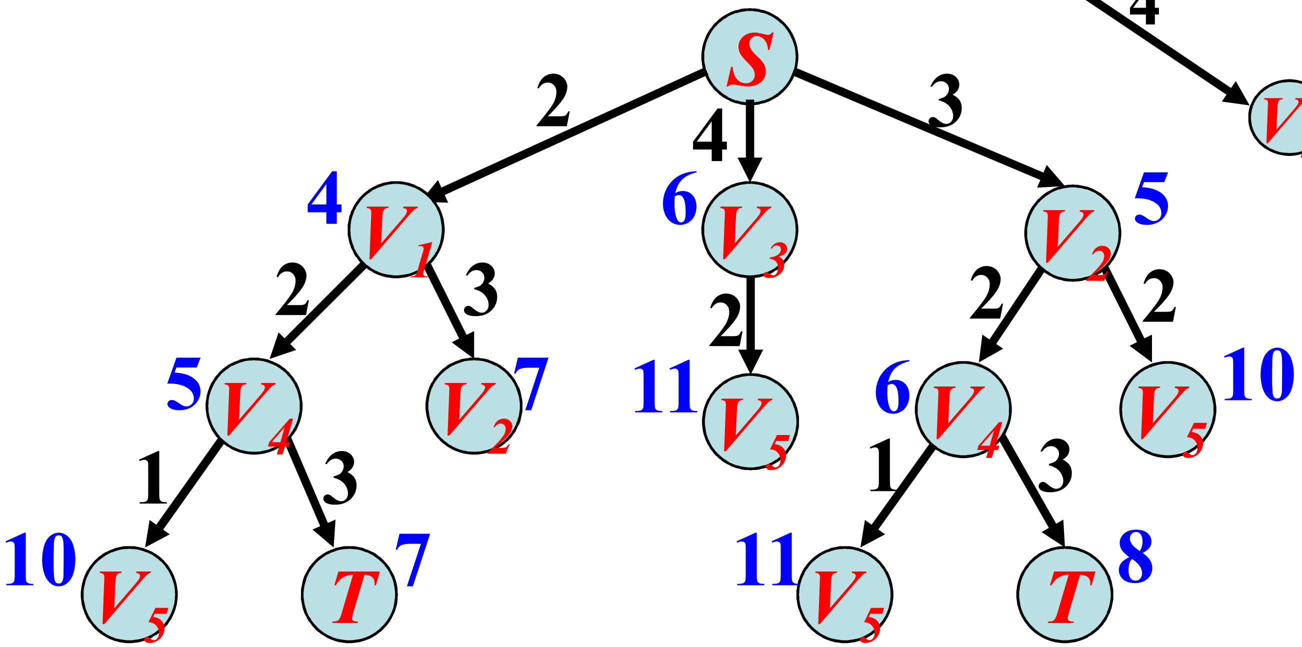


$g(V_5)=3+2+1=6$ $h(V_5)=min\{5\}=5$ $f(V_5)=6+5=11$

$g(T)=3+2+3=8$ $h(T)=0$ $f(T)=8+0=8$

Step 7. 扩展T





因为T是目标节点，所以我们得到解：

$$S\to V_1\to V_4\to T$$