HIT
CS&E

第六章

树搜索策略原理

程思瑶
计算机科学与技术学院

参考资料

R.C.T.Lee, S.S.Tseng, R.C.Chang, and Y.T.Tsai,

Introduction to the Design and Analysis of Algorithms

Chapter 5

Page 157~219

提要

6.1 树搜索方法的研究动机

6.2 简单树搜索策略

6.3 最优树搜索策略

6.4 人员分配问题

6.5 旅行商问题

6.6 A* 算法

6.1 树搜索方法的研究动机

很多问题的解可以表示成为树.

解为树的节点或路径.

求解这些问题可以转化为树搜索问题

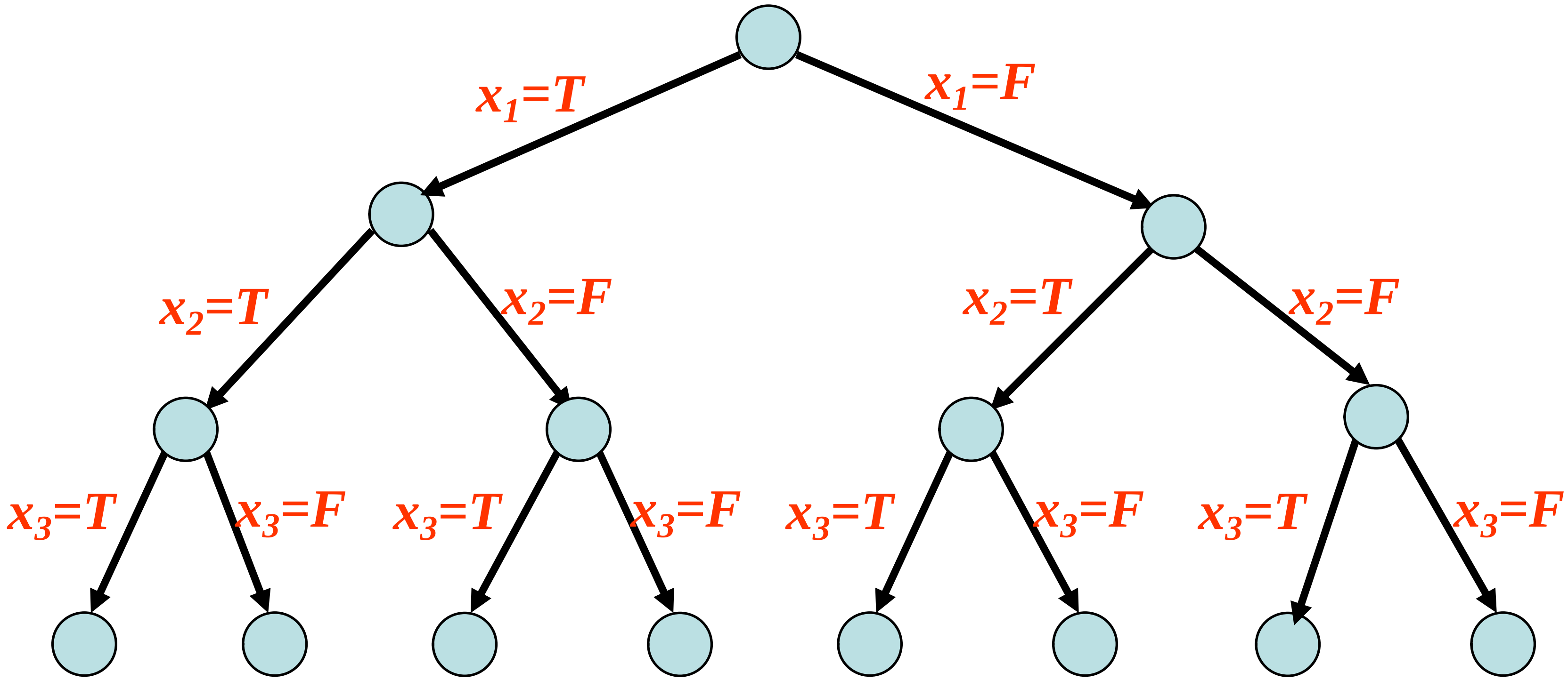
布尔表达式可满足性问题

- 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个析取式
  - 输出: 是否存在$x_1, x_2, \ldots, x_n$的一种赋值
    使得所有k个析取式皆为真

HIT

- 把问题的解表示为树
  - 通过不断地为赋值集合分类来建立树

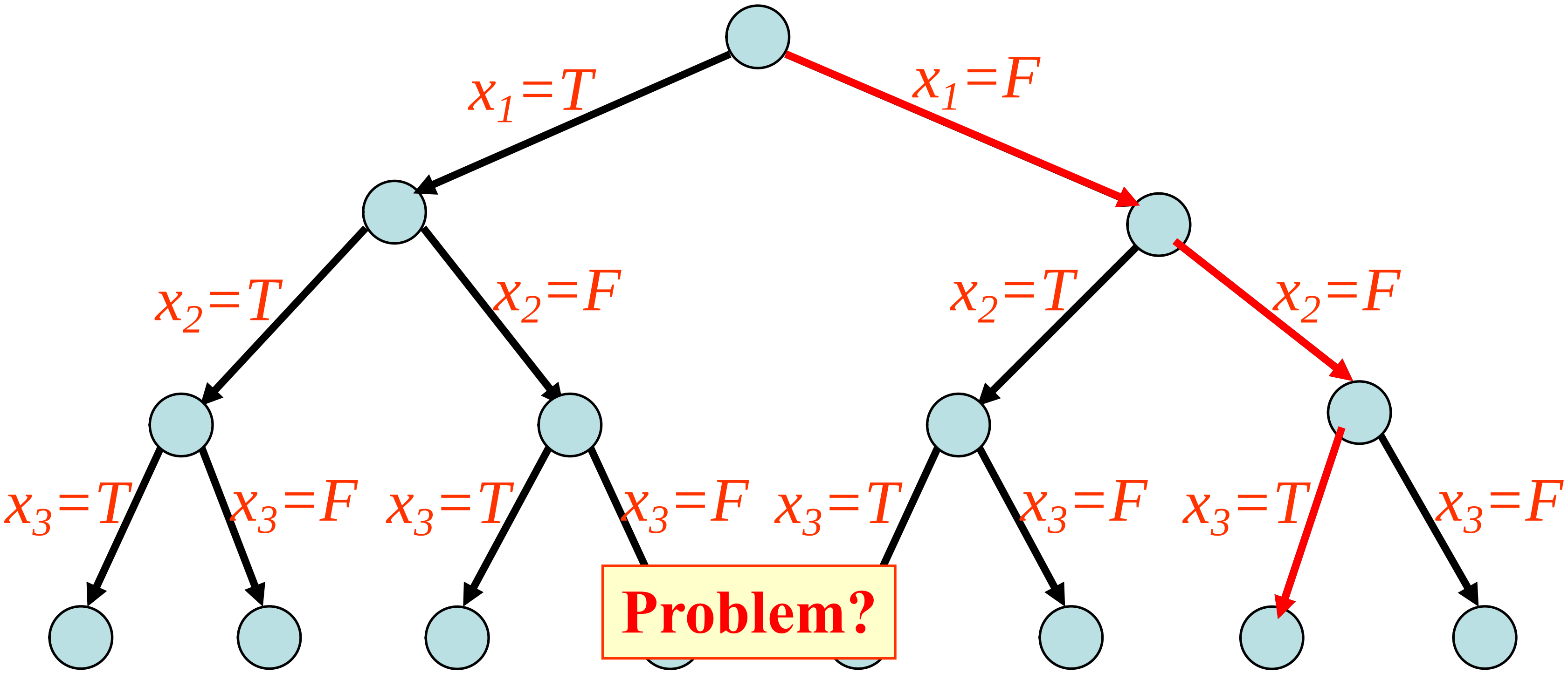
    (以三个变量(x_1, x_2, x_3)为例)



HIT
CS&E

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

问题的解是一条路径



8-Puzzle问题

- 问题的定义
  - 输入：具有8个编号小方块的魔方

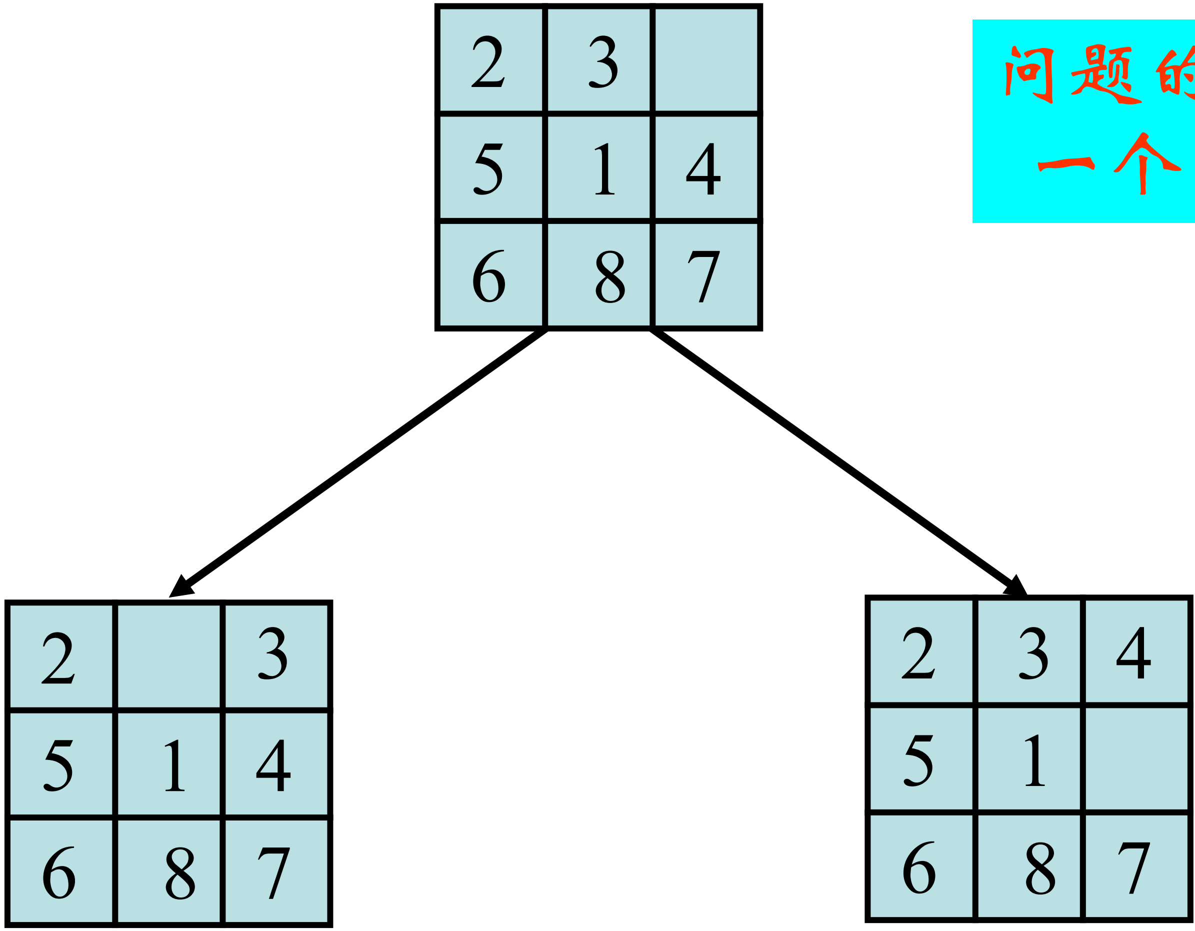
2	3	
5	1	4
6	8	7

  - 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

如何构造树？

- 转换为树搜索问题



问题的解是一个节点

HIT
CS&E

6.2 简单树搜索策略

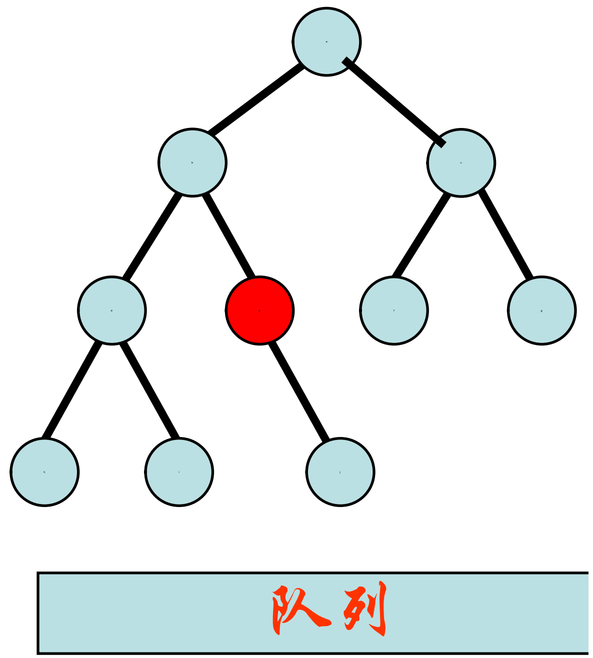
- 先广搜索(Breadth-First Search)
- 先深搜索(Depth-First Search)
- 回溯搜索(Backtracking Search)

HIT
CS&E

Breadth-First Search

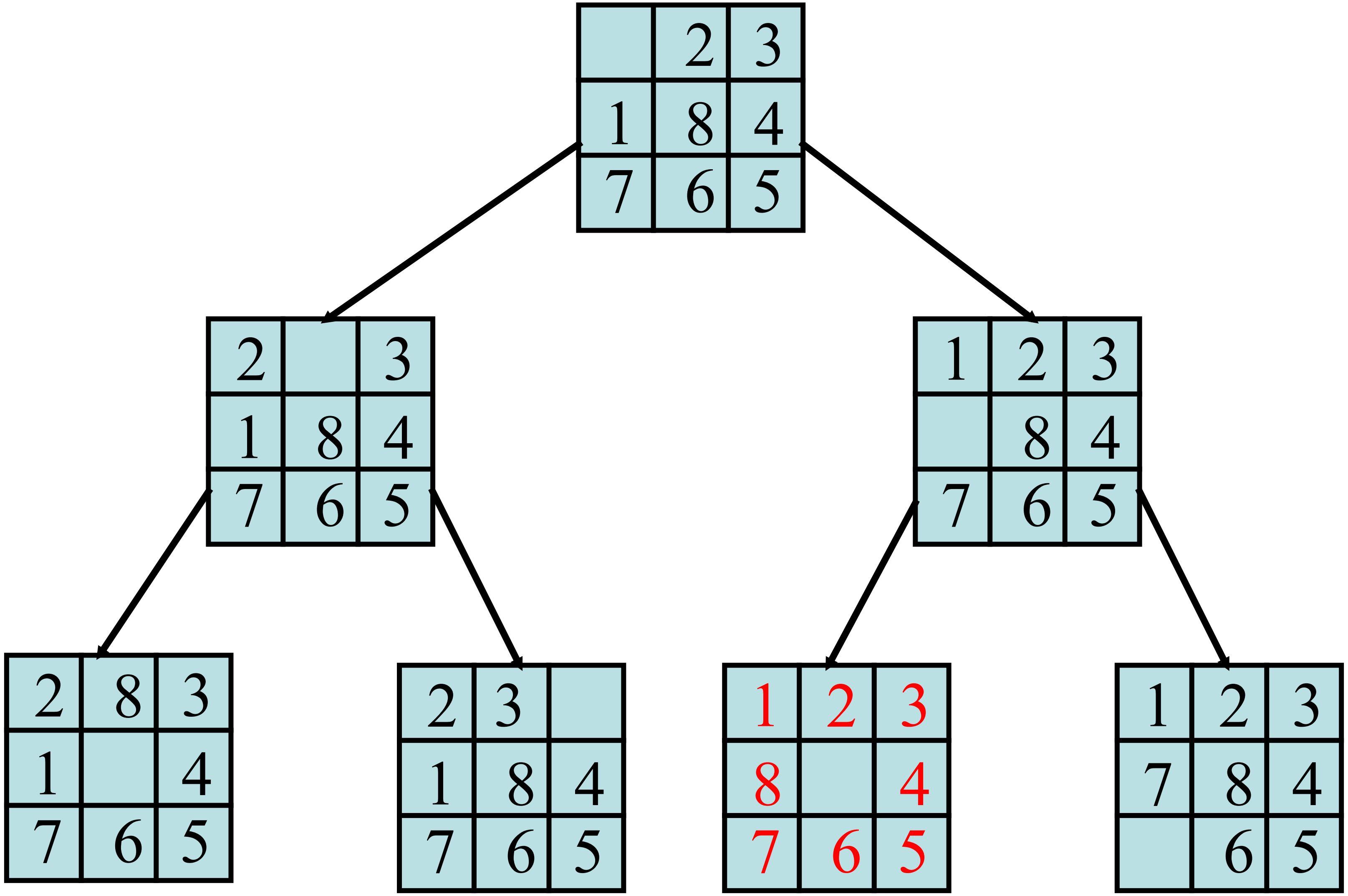
- 算法

  1. 构造仅由根组成的队列Q;
  2. IF Q的第一个元素x是目标节点
  3. Then 输出解,停止;
  4. ELSE 从Q中删除x，把x的所有子节点加入Q的末尾;
  5. IF Q空 Then 无解
  6. Else goto 2.



- 计算复杂性（搜索树节点为n）
  - 最好复杂性(目标节点为根)：$O(1)$
  - 最坏复杂性：$O(n)$
  - 平均复杂性：$\sum_{1\leq i\leq n} p_i \mathrm{Cost}(\alpha_i)$,
    α_i是第i个节点，p_i是α_I为目标节点的概率。

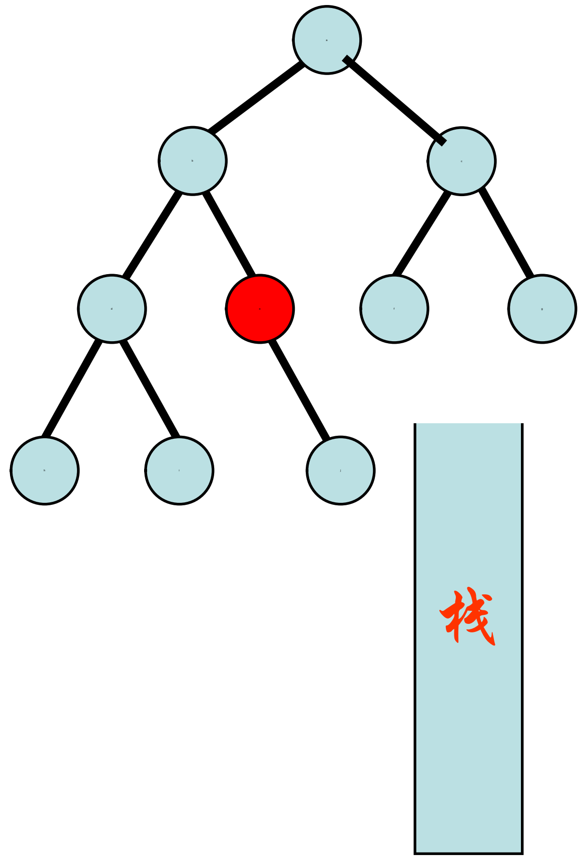
- **例：求解8-Puzzle问题(注意队列如何保证宽度优先)**



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
3. Then 输出解，停止;
4. ELSE $T = Top(S)$, $Pop(S)$;
5. 把T的所有子节点压入栈顶;
6. If S空 Then 无解;
7. Else goto 2.



- 计算复杂性（搜索树节点为n）

最好复杂性(目标节点为根)：$O(1)$

最坏复杂性：$O(n)$

平均复杂性：$\sum_{1\leq i\leq n} p_i \mathrm{Cost}(\alpha_i)$,

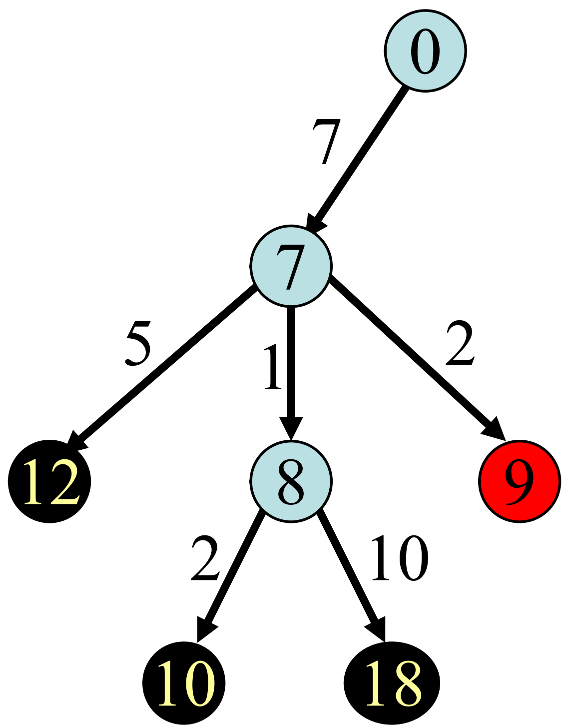
α_i是第i个节点，p_i是α_i为目标节点的概率。

HIT

- 例1. 求解子集合和问题(注意栈如何保证深度优先)

输入: S={7, 5, 1, 2, 10}

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

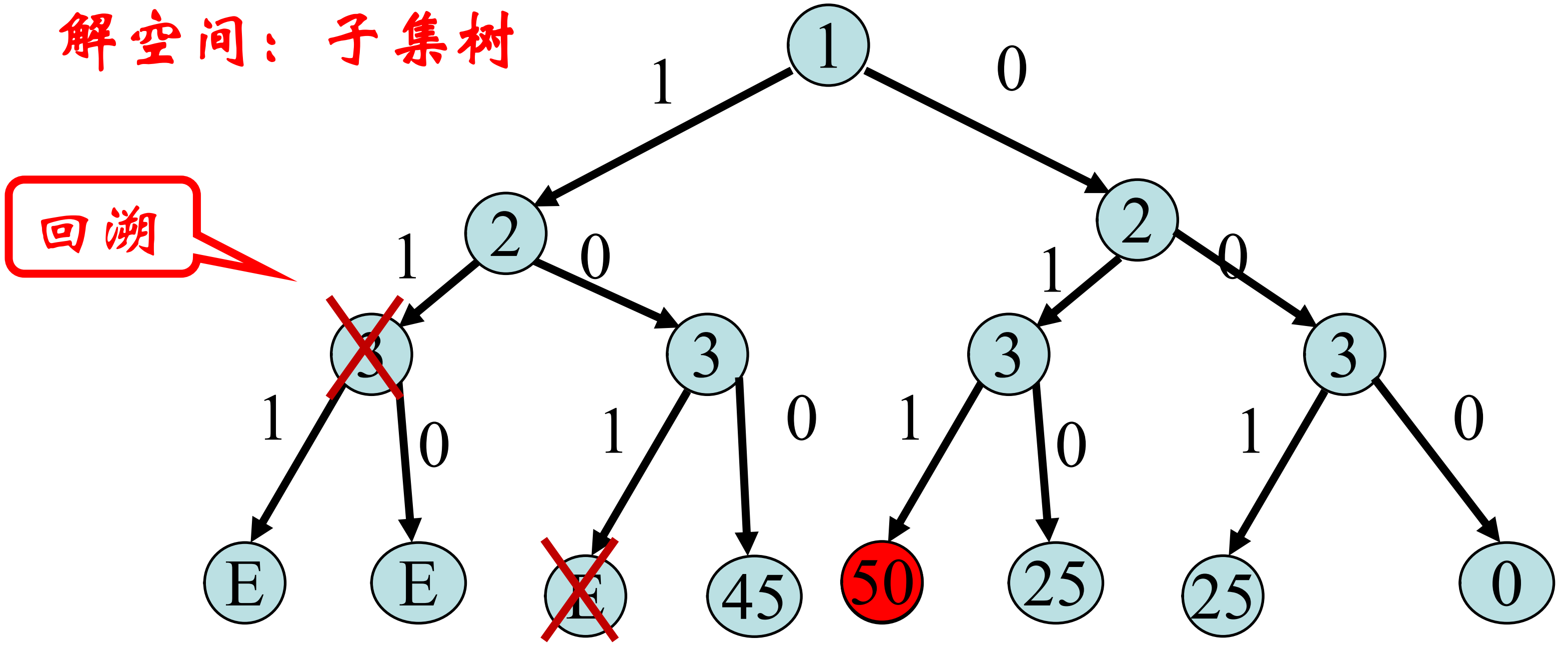


Backtracking Search

- 算法思想
  - 是穷举搜索的第一次改进
    - 整个解空间太大，某些无意义的搜索可以通过剪枝规避
      - 用约束函数在扩展节点处剪去不满足约束条件的子树
      - 或用界限函数剪去得不到最优解的子树
  - 按深度优先策略从根出发搜索解空间树
  - 搜索至任意节点时，先判断该节点是否包含问题的解
    - 若肯定不包含，则跳过以该节点为根的子树，逐层向祖先节点回溯
    - 否则，进入子树，继续深度优先搜索
  - 常见两类解空间树
    - 子集树：计算"从n个元素的集合S中找满足某种性质的子集"问题时。具有2^n个叶节点，遍历树需$\Omega(2^n)$时间
    - 排列树：计算"确定n个元素满足某种性质的排列"问题时。具有$n!$个叶节点，遍历树需$\Omega(n!)$的时间

- 例1. n=3时的0-1背包问题(背包容量30)

解空间：子集树



问题实例： w_1=16, v_1= 45

w_2=15, v_2= 25

w_3=15， v_3= 25

6.3 最优树搜索策略

- 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
    - 使用什么启发式测度与问题本身相关

- 用8-Puzzle问题来说明爬山策略的思想

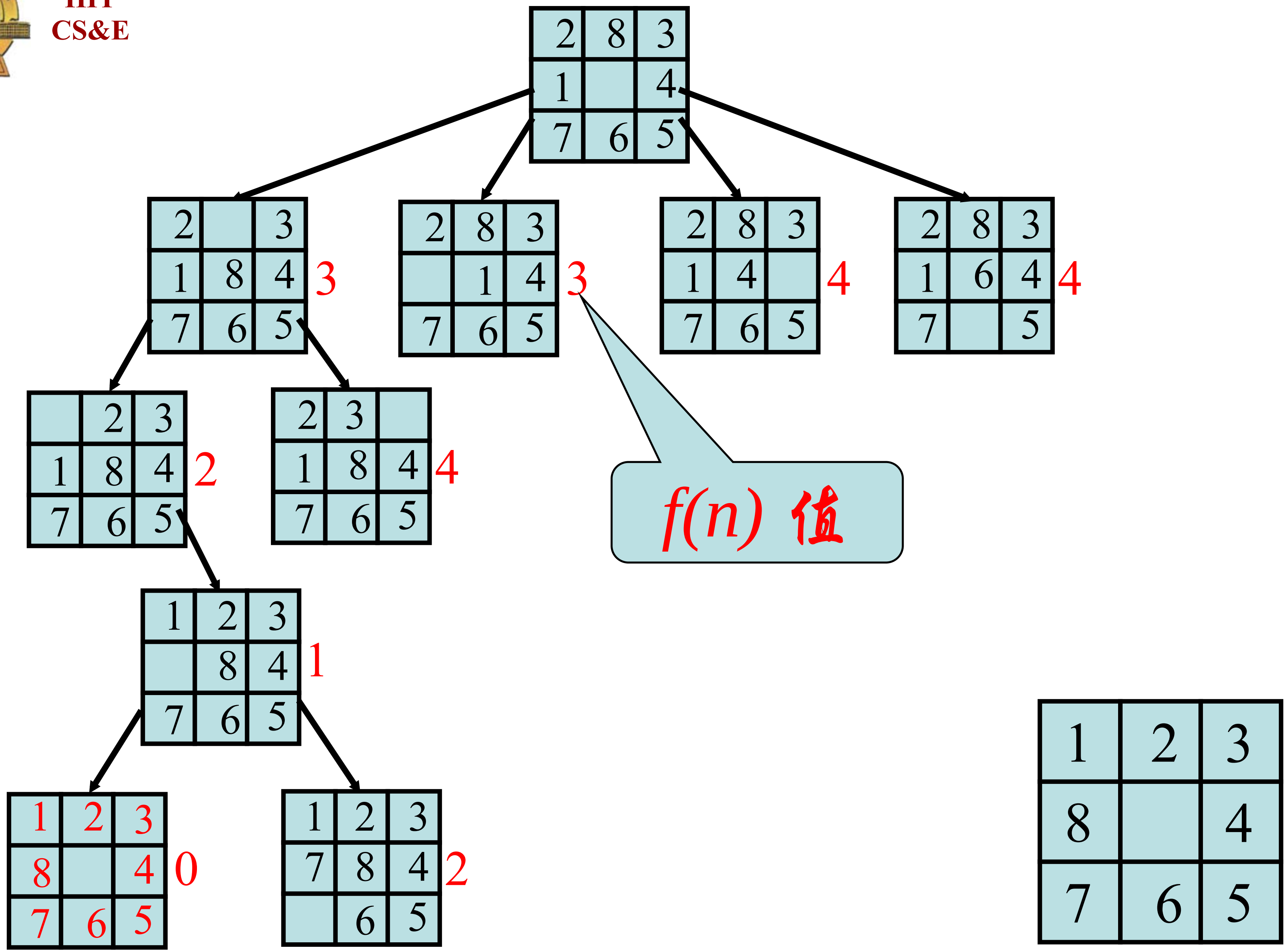
1	2	3
8		4
7	6	5

  - 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：

2	8	3
1		4
7	6	5

    - 则$f(n)=3$, 因为方块1、2、8处于错误位置.

HIT
CS&E



- **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 Strategy

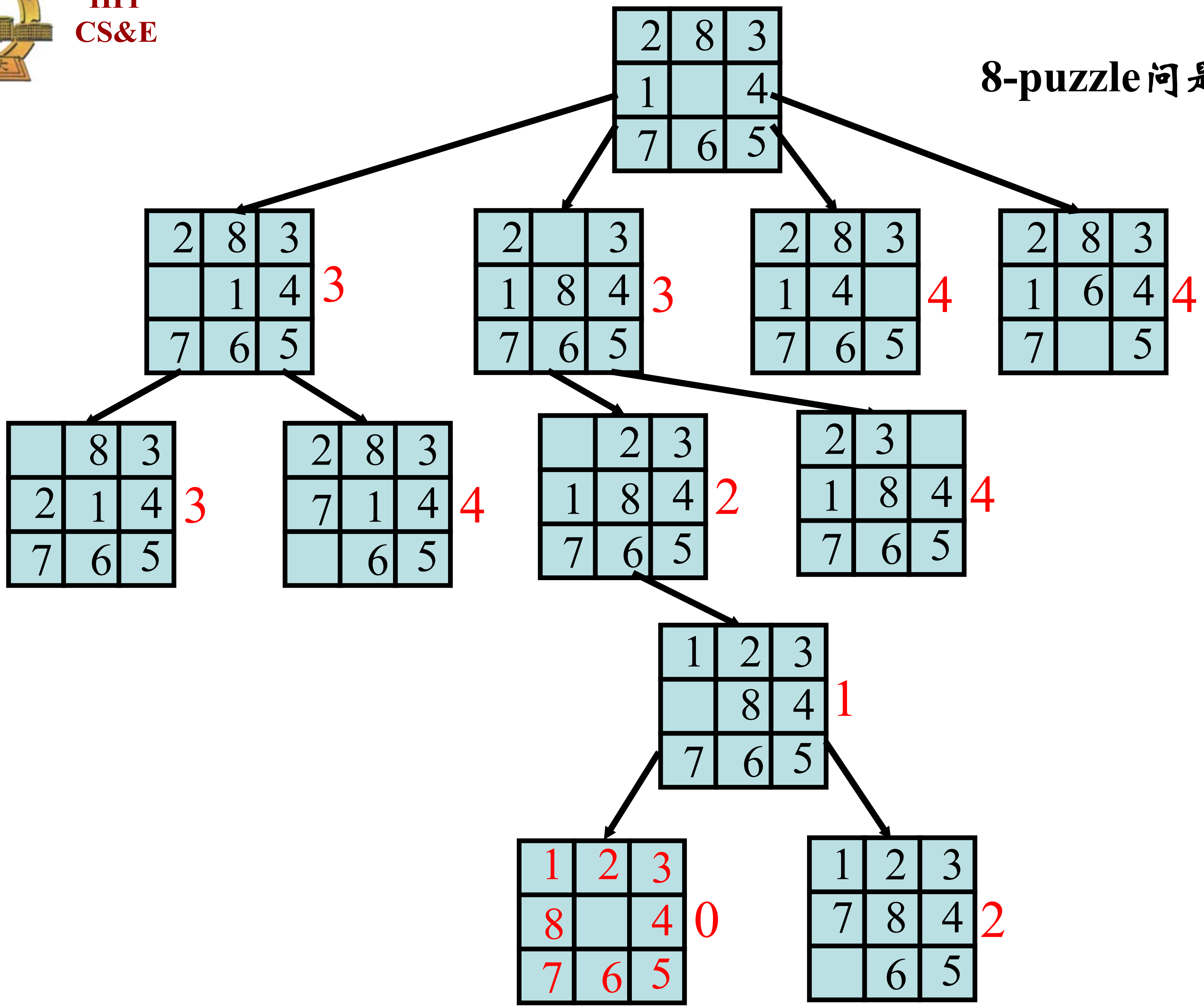
- 基本思想
  - 结合深度优先和广度优先的优点于一个方法
  - 根据一个评价函数,在目前产生的所有节点中选择具有最小评价函数值的节点进行扩展.
  - 具有全局优化观念,而爬山策略仅具有局部优化观念.

- **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.

Heapsort方法见Intro. to Algo. 第II部分 第6章

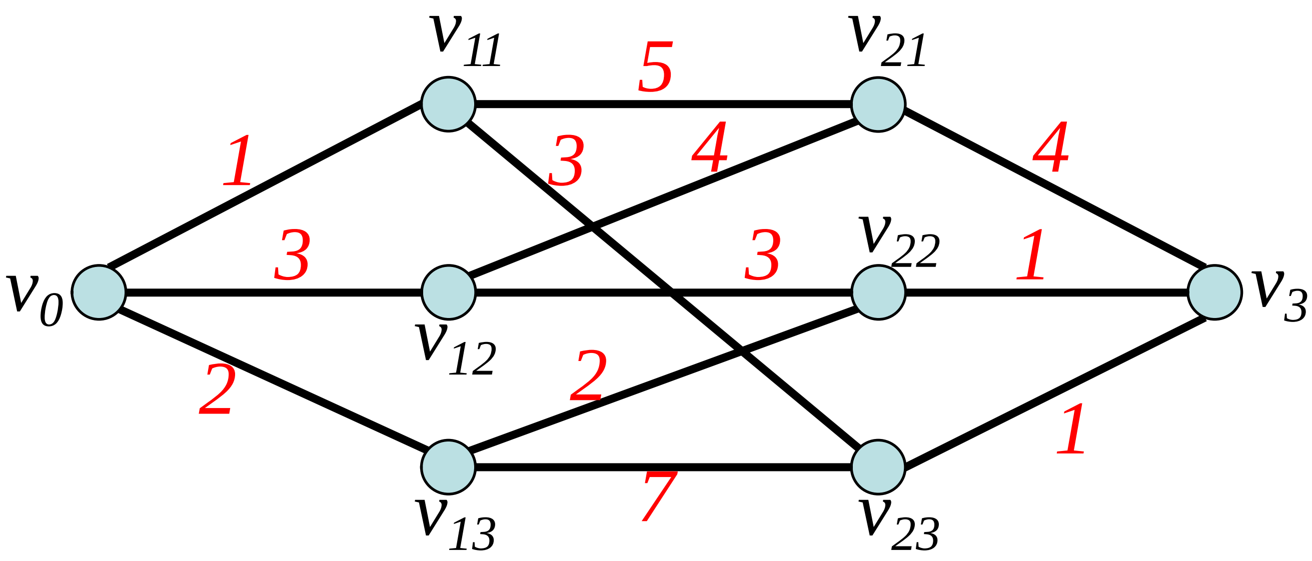
8-puzzle问题实例



Branch-and-Bound Strategy

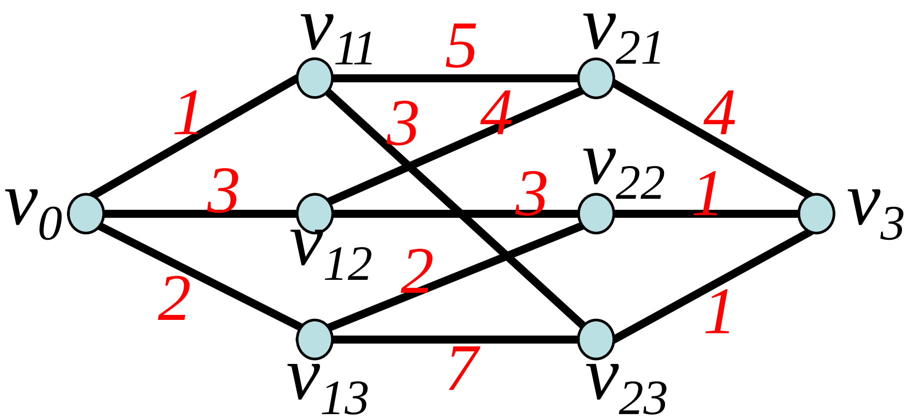
- 基本思想
  - 上述方法很难用于求解优化问题
  - 分支界限策略可以有效地求解组合优化问题
  - 发现优化解的一个界限
  - 缩小解空间，提高求解的效率

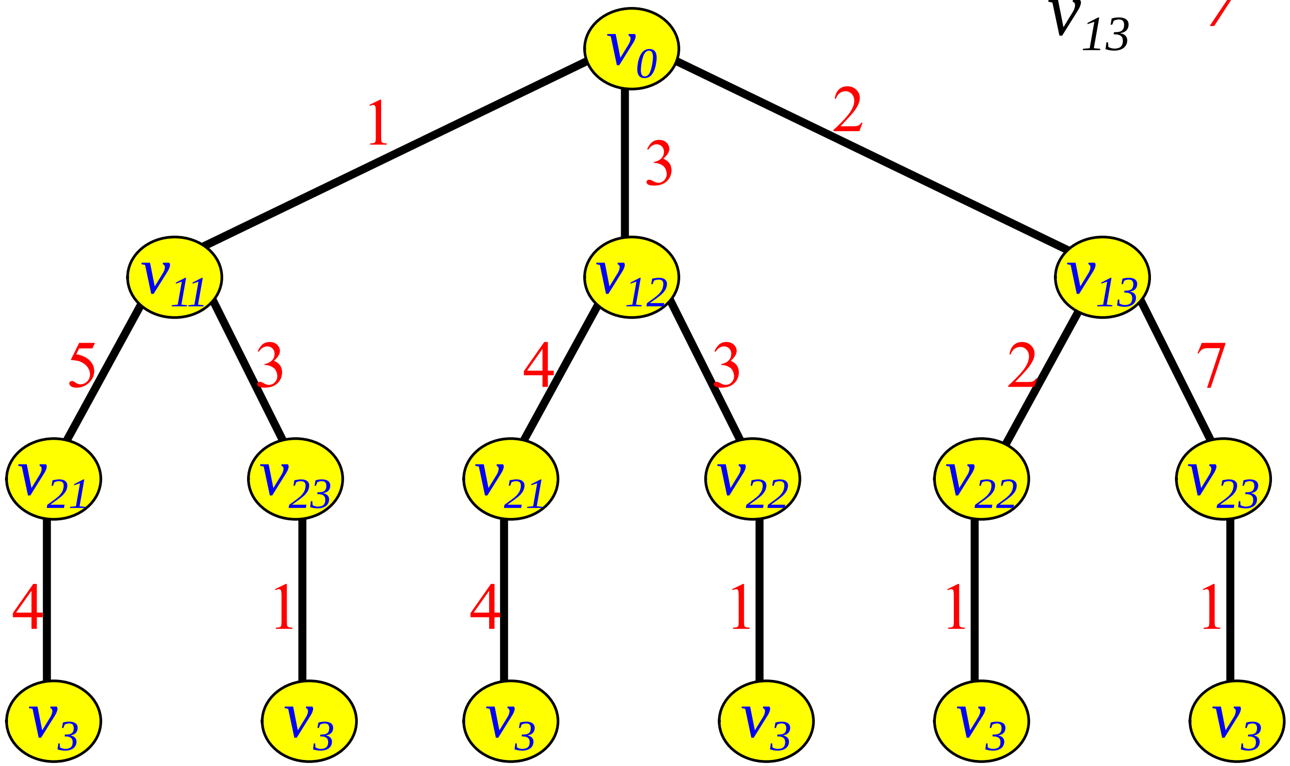
- 多阶段图搜索问题
  - 输入：多阶段图



  - 输出：从v_0到v_3的最短路径

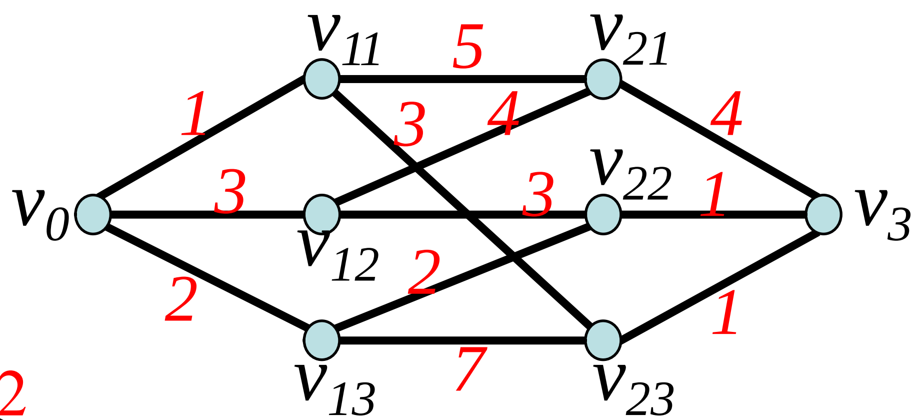
- 问题的树表示

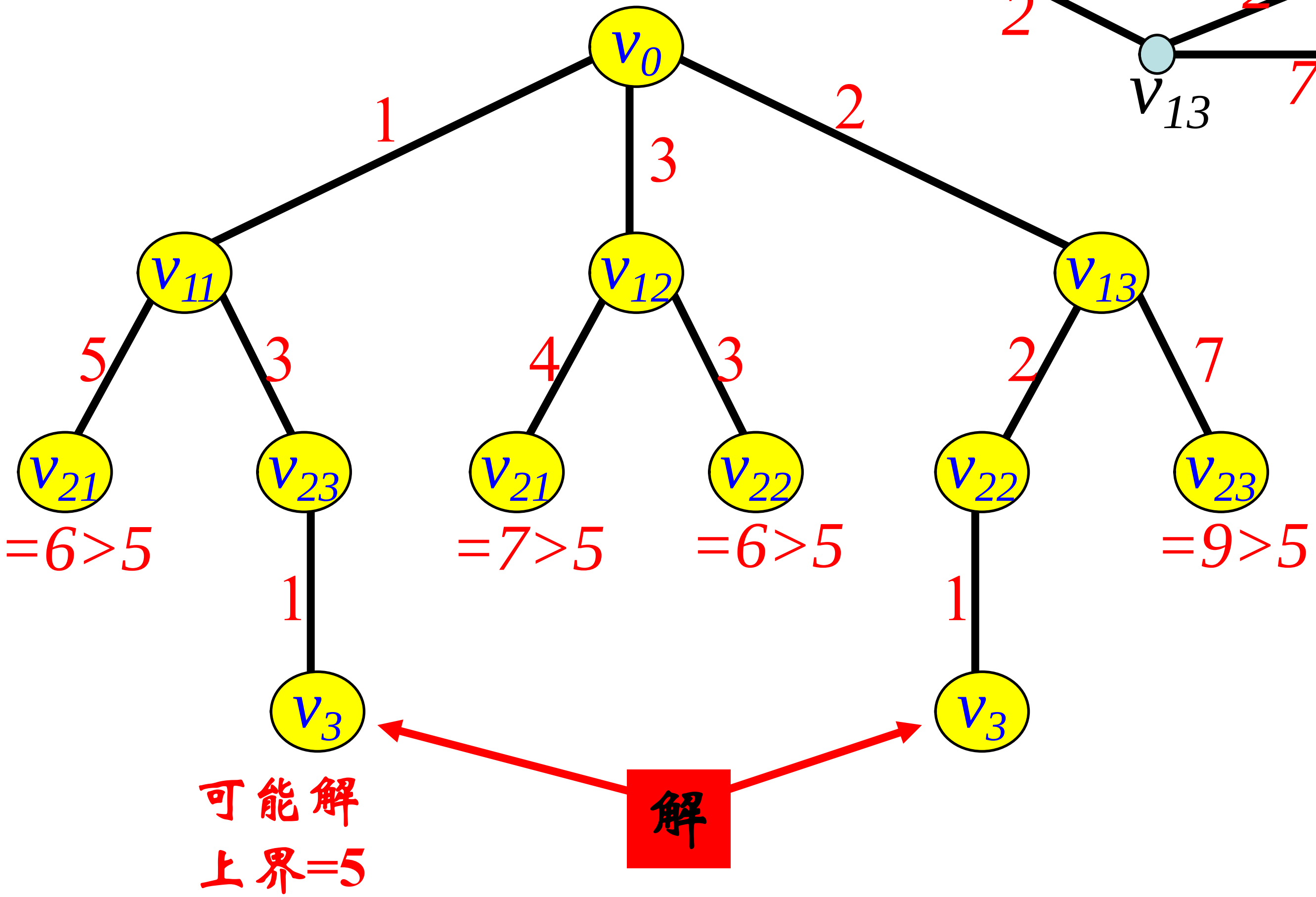

v_{11}
5
v_{21}
1
3
4
4
v_0
3
v_{12}
3
v_{22}
1
v_3
2
2
1
v_{13}
7
v_{23}


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

评价函数：路径和





- 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.

6.4 人员分配问题

- 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

> **例.** 给定P={P_1, P_2, P_3}，J={J_1, J_2, J_3}，$J_1 \leq J_3$, $J_2 \leq J_3$.
> $P_1 \rightarrow J_1$、$P_2 \rightarrow J_3$、$P_3 \rightarrow J_2$ 是否为可能的解？

- 输出
  - 矩阵$[X_{ij}]$, $X_{ij}=1$表示P_i被分配J_j，$\sum_{i,j} C_{ij}X_{ij}$最小
    - 不同人分配不同工作
    - 每个人被分配一种工作,
    - f: $P \rightarrow J$是工作分配函数, 满足:

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

      如果$i \neq j$， $f(P_i) \neq f(P_j)$.

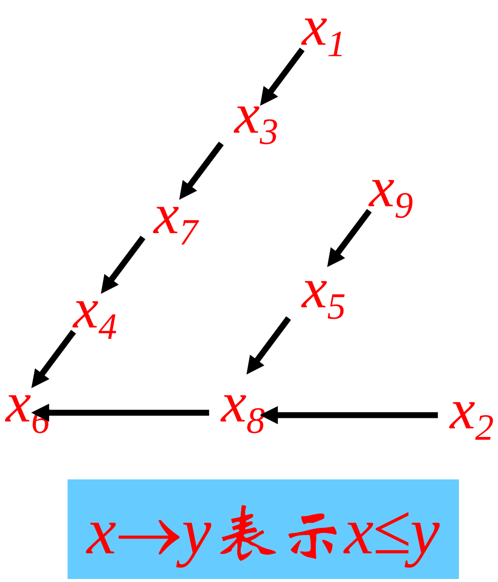
求解问题的思想

- 问题转化为树搜索问题
  - $J=\{J_1, J_2, \ldots, J_n\}$的每个拓扑序列对应一个解
  - 用一个拓扑序列树把所有拓扑序列安排在一个树中，每个路径对应一个拓扑序列
  - 问题成为在树中搜索最小路径问题
- 构造拓扑序列树
- 使用分支界限法搜索拓扑序列树求解问题
- 改拓扑序列树构造算法为分支界限求解算法

转换为树搜索问题

- 拓扑排序
  - 输入：偏序集合$(S, \leq)$
  - 输出：S的拓扑序列是$<s_1, s_2, \ldots, s_n>$，

    满足：如果$s_i \leq s_j$，则s_i排在s_j的前面.
  - 例



拓扑排序：

$x_1\ x_3\ x_7\ x_4\ x_9\ x_5\ x_2\ x_8\ x_6$

- 问题的解空间

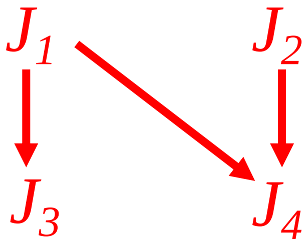
命题1. $P_1 \rightarrow J_{k1}$、$P_2 \rightarrow J_{k2}$、…、$P_n \rightarrow J_{kn}$是一个可能解，当且仅当J_{k1}、J_{k2}、…、J_{kn}是一个拓扑排序的序列.

证明：Homework

解空间是工作集合所有拓扑排序的序列集合，
每个序列对应于一个可能的解

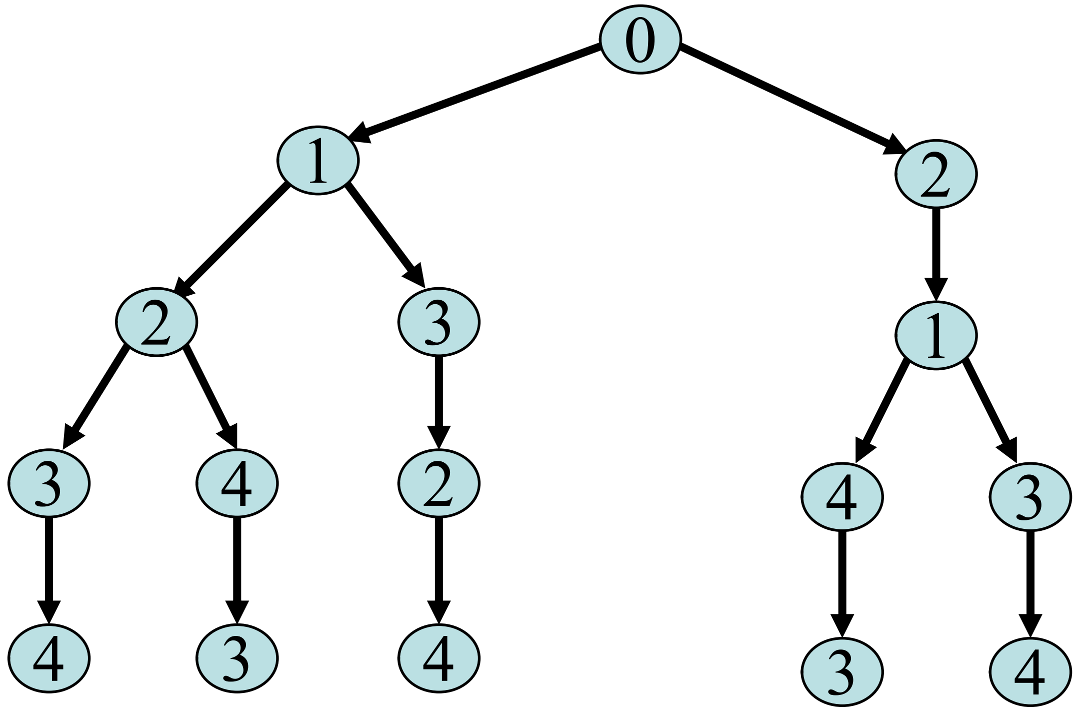
则J所有的拓扑排序序列有：(J_1, J_2, J_3, J_4)、(J_1, J_2, J_4, J_3)、(J_1, J_3, J_2, J_4)、(J_2, J_1, J_3, J_4)、(J_2, J_1, J_4, J_3).

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



(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)

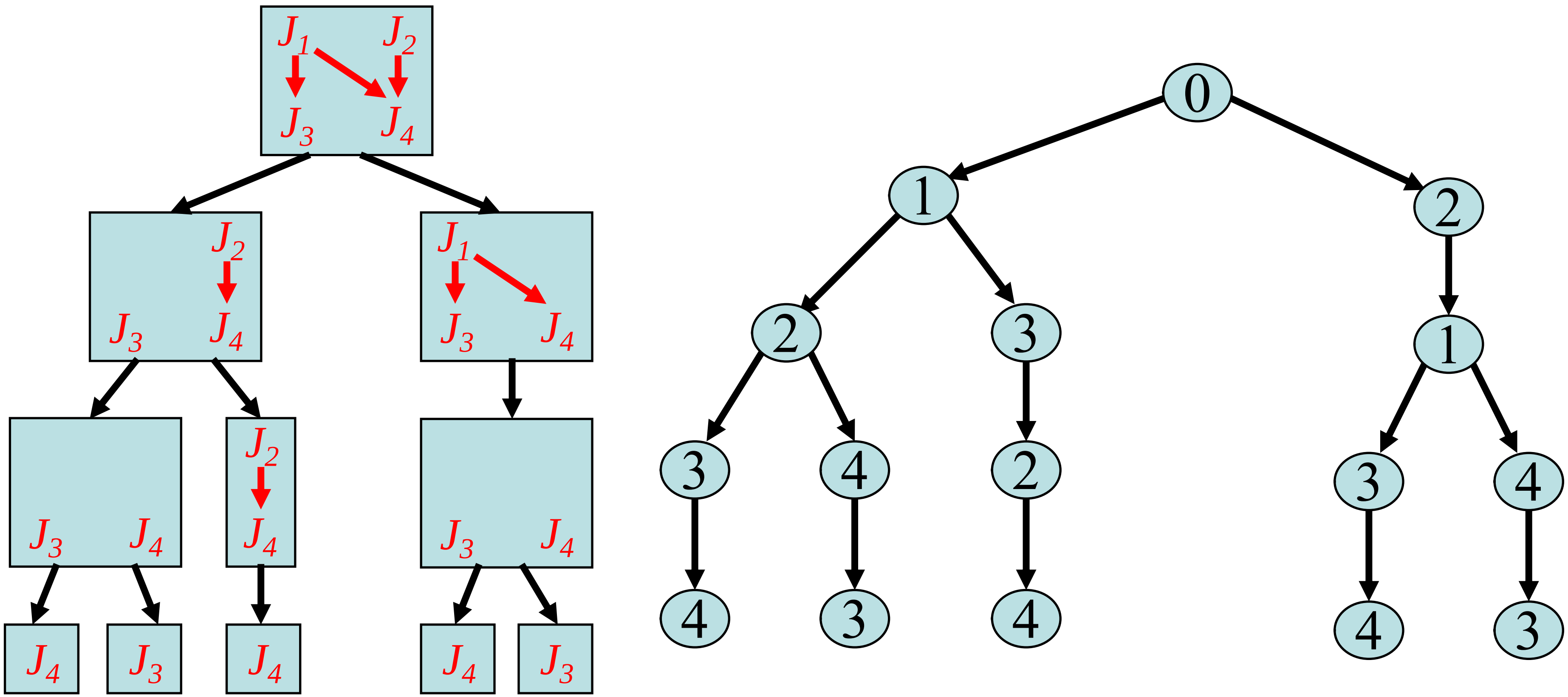
- 问题的树表示(即用树表示所有拓扑排序序列)



如何由偏序关系直接生成拓扑序列树呢

(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)

- 拓扑序列树的生成算法的基本思想



- **拓扑序列树的生成算法**

  输入：偏序集合*S*, 树根*root*.

  输出：由*S*的所有拓扑排序序列构成的树.

  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*的子结点;
  3. For *root*的每个子结点*v* Do
  4. $S=S-\{v\}$;
  5. 把*v*作为根，递归地处理*S*.

- **计算解的代价的下界**

例.

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

$$
\begin{tikzcd}
J_1 \arrow[d] \arrow[dr] & J_2 \arrow[d] \\
J_3 & J_4
\end{tikzcd}
$$

得到一个具有最小代价**12+26+3+10+3=54**的任务分配方案：

$P_1 \rightarrow J_4$、$P_2 \rightarrow J_3$、$P_3 \rightarrow J_1$、$P_4 \rightarrow J_2$

它不满足偏序约束，故不是可行解

由此得到可行解的代价下界=54

求解问题的分支界限搜索算法

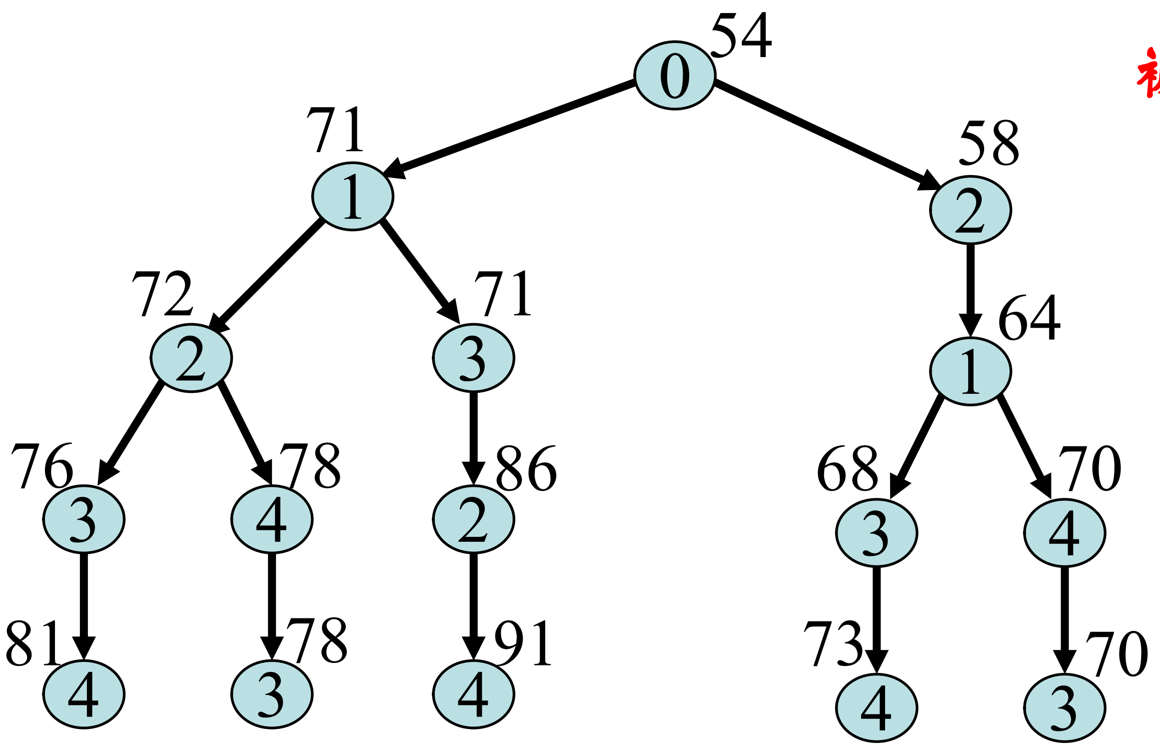
- 计算解的代价的下界

命题2：把代价矩阵每行(列)各元素减去同一个数，不影响优化解的求解.

  - 代价矩阵每行(列)减去该行(列)的最小数，使得每行和每列至少有一个零，其余各元素非零
  - 如此简化代价矩阵对解无影响
    (因为每个解代价都减去了一个相同的数)
  - 每行和列减去的数的总和是解的下界.

- 解空间的加权树表示

$$\begin{array}{c|cccc|} & J_1 & J_2 & J_3 & J_4 \\ P_1 & 17 & 4 & 5 & 0 \\ P_2 & 6 & 1 & 0 & 2 \\ P_3 & 0 & 15 & 4 & 6 \\ P_4 & 8 & 0 & 0 & 5 \end{array}$$



被分配的人

1

2

3

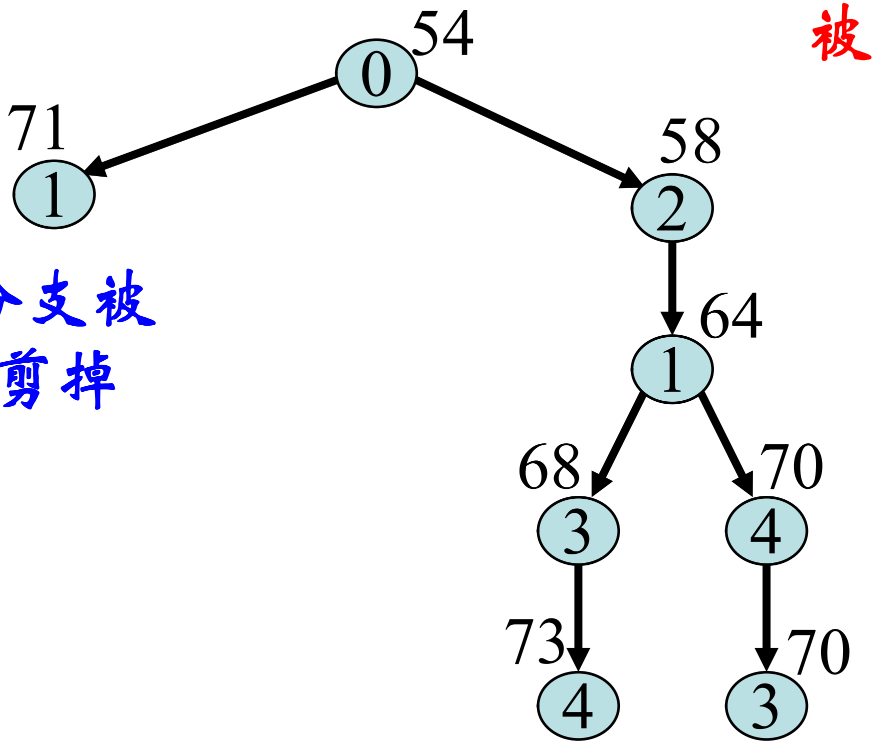
4

算法的思想(用爬山法)

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

$$\begin{tikzcd} J_1 \arrow[d] \arrow[dr] & J_2 \arrow[d] \\ J_3 & J_4 \end{tikzcd}$$

$$\begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \\ P_4 \end{array}\begin{array}{c} \begin{array}{cccc} J_1 & J_2 & J_3 & J_4 \end{array} \\ \begin{bmatrix} 17 & 4 & 5 & 0 \\ 6 & 1 & 0 & 2 \\ 0 & 15 & 4 & 6 \\ 8 & 0 & 0 & 5 \end{bmatrix} \end{array}$$



分支被剪掉

被分配到的人

1

2

3

4

- 分支界限搜索(使用爬山法)算法
  1. 建立根结点，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，每产生一个结点，其权值为加工后的代价矩阵对应元素加其父结点权值;
  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.

修改拓扑排序算法，可以写出严格的分支界限搜索算法

6.5 旅行商问题

- 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$G=(V, E)$,

- 每个结点都没有到自身的边,
- 每对结点间都有一条非负加权边(由代价矩阵表示)

输出：一条由任意一个结点开始,

经过每个结点一次,

最后返回开始结点的路径,

该路径的代价(即权值之和)最小.

转换为树搜索问题

- 所有解集合作为树根，由代价矩阵计算所有解的代价的下界；
- 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)
    使右子树代价下界增加最大，
    使左子树代价下界不变
  - 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

- 设代价矩阵如下

$$
\begin{array}{ccc}
 & \begin{array}{ccccccc} j=1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} & \\
\begin{array}{c} i=1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} &
\begin{bmatrix}
\infty & 0 & 83 & 9 & 30 & 6 & 50 \\
0 & \infty & 66 & 37 & 17 & 12 & 26 \\
29 & 1 & \infty & 19 & 0 & 12 & 5 \\
32 & 83 & 66 & \infty & 49 & 0 & 80 \\
3 & 21 & 56 & 7 & \infty & 0 & 28 \\
0 & 85 & 8 & 42 & 89 & \infty & 0 \\
18 & 0 & 0 & 0 & 58 & 13 & \infty
\end{bmatrix} &
\begin{array}{c} -3 \\ -4 \\ -16 \\ -7 \\ -25 \\ -3 \\ -26 \end{array} \\
 & \begin{array}{ccccccc} & & -7 & -1 & & & -4 \end{array} &
\end{array}
$$

- 构造根节点
  - 根结点为所有解的集合
  - 计算根结点的代价下界

    =3+4+16+7+25+3+26+7+1+4=96

- 得到如下根结点及其代价下界

所有解的集合 L.B=96

划分边（i, j）满足：

- $Cost(i, j) = 0$
- 右子树代价下界增加最大

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

- 构造根结点的两个子结点
  - 选择使右子树代价下界增加最大的划分边(4, 6)
  - 建立根结点的子结点：
    - ✓ 左子结点为包括边(4, 6)的所有解集合
    - ✓ 右子结点为不包括边(4, 6)的所有解集合

所有解的集合　L.B=96

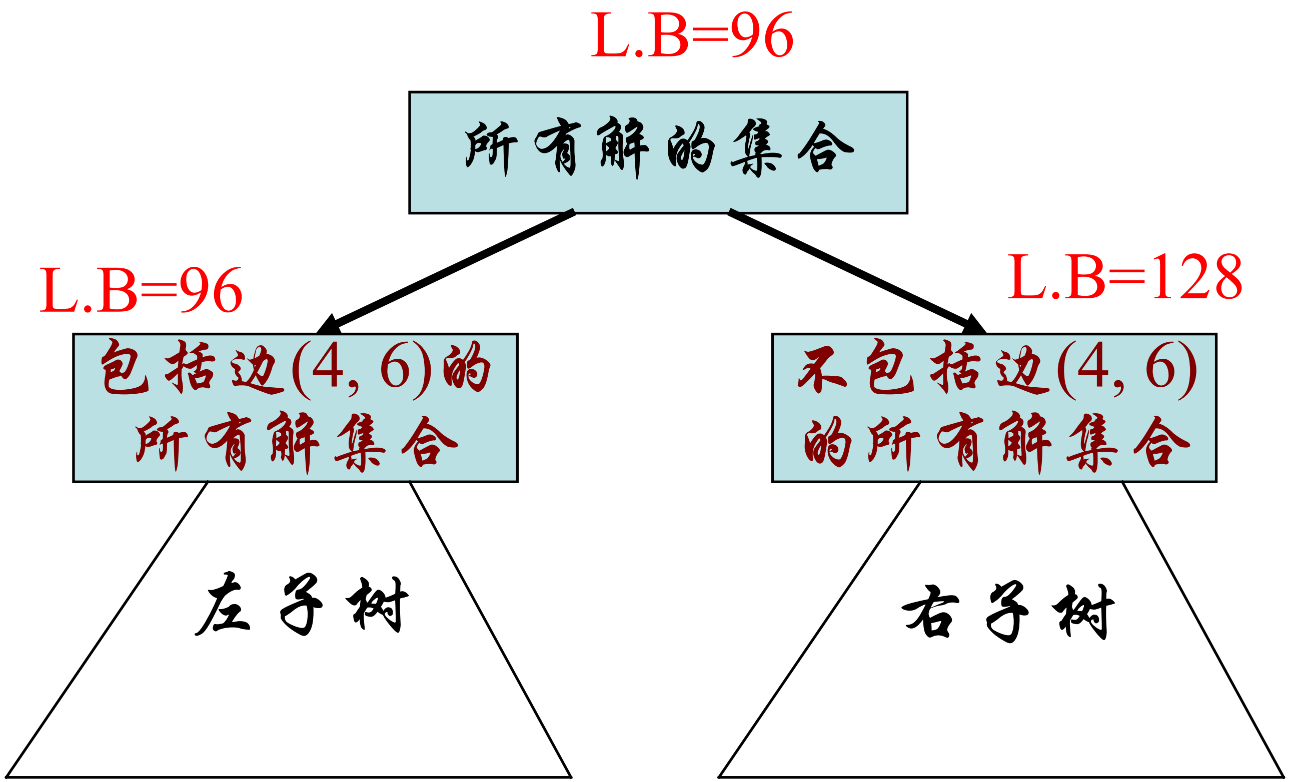
包括边(4, 6)的所有解集合

不包括边(4, 6)的所有解集合

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

- 计算左右子结点的代价下界
  - ✓ (4, 6)的代价为0, 所以左结点代价下界仍为96.
  - ✓ 我们来计算右结点的代价下界:
    - 如果一个解不包含(4, 6), 它必包含一条从4出发的边和 进入结点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, 右结点代价下界为: 96+32+0=128.

- 目前的树为



- 递归地构造左右子树根的代价矩阵
  - 构造左子树根对应的代价矩阵
    - 左子结点为包括边(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∞.

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

➢ 修改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

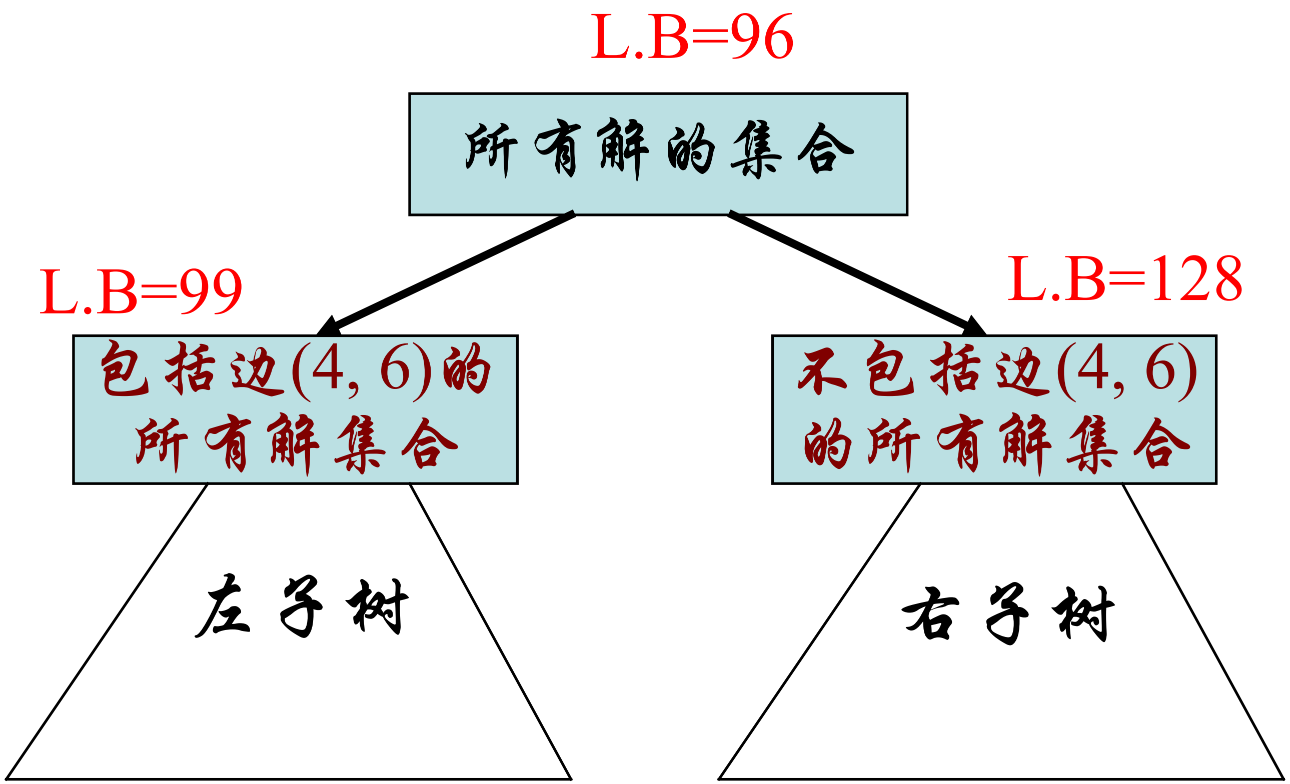
$$
\begin{array}{c|cccccc|c}
 & j=1 & 2 & 3 & 4 & 5 & 7 & \\
\hline
i=1 & \infty & 0 & 83 & 9 & 30 & 50 & \\
2 & 0 & \infty & 66 & 37 & 17 & 26 & \\
3 & 29 & 1 & \infty & 19 & 0 & 5 & \\
5 & 0 & 18 & 53 & 4 & \infty & 25 & -3 \\
6 & 0 & 85 & 8 & \infty & 89 & 0 & \\
7 & 18 & 0 & 0 & 2 & 58 & \infty &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

➢ 构造右子树根对应的代价矩阵

- ✓ 右子结点为不包括边(4, 6)的所有解集合，只需要把C[4, 6]设置为∞
- ✓ 结果矩阵如下

$$
\begin{array}{ccc}
 & \begin{array}{ccccccc} j=1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} & \\
\begin{array}{c} i=1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} &
\begin{bmatrix}
\infty & 0 & 83 & 9 & 30 & 6 & 50 \\
0 & \infty & 66 & 37 & 17 & 12 & 26 \\
29 & 1 & \infty & 19 & 0 & 12 & 5 \\
0 & 51 & 34 & \infty & 17 & \infty & 48 \\
3 & 21 & 56 & 7 & \infty & 0 & 28 \\
0 & 85 & 8 & 42 & 89 & \infty & 0 \\
18 & 0 & 0 & 0 & 58 & 13 & \infty
\end{bmatrix} &
\begin{array}{c} \\ \\ \\ -32 \\ \\ \\ \\ \end{array}
\end{array}
$$

- 目前的树为



$$\begin{bmatrix} \infty & 0 & 83 & 9 & 30 & 50 \\ 0 & \infty & 66 & 37 & 17 & 26 \\ 29 & 1 & \infty & 19 & 0 & 5 \\ 0 & 18 & 53 & 4 & \infty & 25 \\ 0 & 85 & 8 & \infty & 89 & 0 \\ 18 & 0 & 0 & 0 & 58 & \infty \end{bmatrix}$$

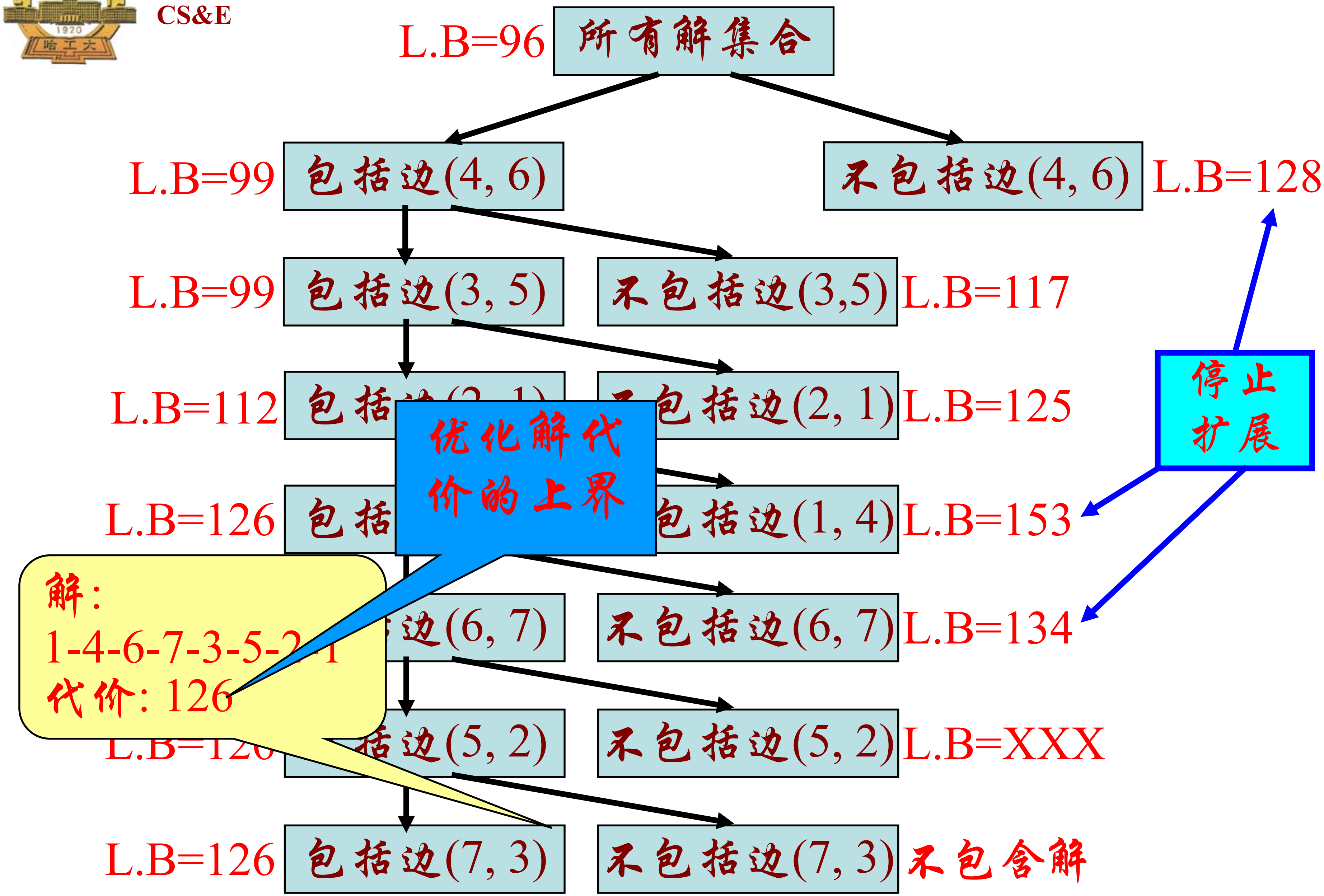
- 使用爬山策略扩展左子树根(因为其代价下界小)
  - ✓ 选择边使右子结点代价下界增加最大的划分边(3, 5)
  - ✓ 左子结点为包括边(3, 5)的所有解集合
  - ✓ 右子结点为不包括边(3, 5)的所有解集合
  - ✓ 计算左、右子结点的代价下界: 99和117
- 目前树扩展为:

HIT
CS&E

$$\begin{bmatrix} \infty & 0 & 83 & 9 & 30 & 50 \\ 0 & \infty & 66 & 37 & 17 & 26 \\ 29 & 1 & \infty & 19 & 0 & 5 \\ 0 & 18 & 53 & 4 & \infty & 25 \\ 0 & 85 & 8 & \infty & 89 & 0 \\ 18 & 0 & 0 & 0 & 58 & \infty \end{bmatrix}$$

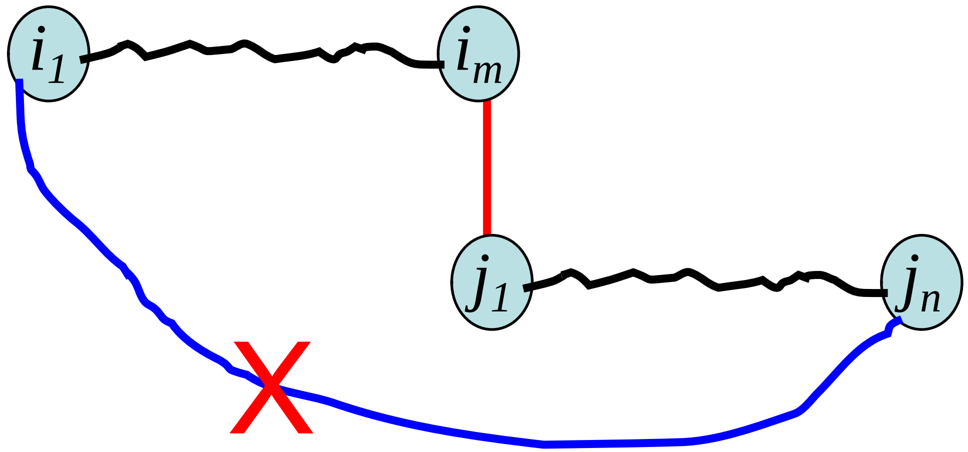
$$\begin{bmatrix} \infty & 0 & 83 & 9 & 50 \\ 0 & \infty & 66 & 37 & 26 \\ 0 & 18 & \infty & 4 & 25 \\ 0 & 85 & 8 & \infty & 0 \\ 18 & 0 & 0 & 0 & \infty \end{bmatrix}$$

$$\begin{bmatrix} \infty & 0 & 83 & 9 & 30 & 50 \\ 0 & \infty & 66 & 37 & 17 & 26 \\ 29 & 1 & \infty & 19 & \infty & 5 \\ 0 & 18 & 53 & 4 & \infty & 25 \\ 0 & 85 & 8 & \infty & 89 & 0 \\ 18 & 0 & 0 & 0 & 58 & \infty \end{bmatrix}$$


L.B=96
所有解集合
L.B=99
包括边(4, 6)
不包括边(4, 6)
L.B=128
L.B=99
包括边(3, 5)
不包括边(3,5)
L.B=117
L.B=112
L.B=125
L.B=126
包括边(1, 4)
L.B=153
不包括边(6, 7)
L.B=134
不包括边(5, 2)
L.B=XXX
L.B=126
包括边(7, 3)
不包括边(7, 3)
不包含解
停止扩展
优化解代价的上界
解:
1-4-6-7-3-5-2-1
代价: 126

注 意

如果i_1-i_2-…-i_m和j_1-j_2-…-j_n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。



- 算法概要

1. 根节点为所有解集合;
2. 使用代价矩阵，计算根节点表示的解集合的代价下界;
3. 使用爬山策略扩展当前的树节点α直至找到一个解:
   - 选择满足下列条件的边(x,y):
     – 使右子树代价下界增加最大,
     – 使左子树代价下界不变.
   - 构造α的左右子树:
     – 左子树为包括边(x,y)的所有解集合;
     – 右子树为不包括边(x,y)的所有解集合;
     – 计算左右子树解代价下界;
     – 构造左右子树的代价矩阵, 修改其解代价下界;
4. 用找到解的代价进行剪枝爬山搜索, 直到找到最优解.

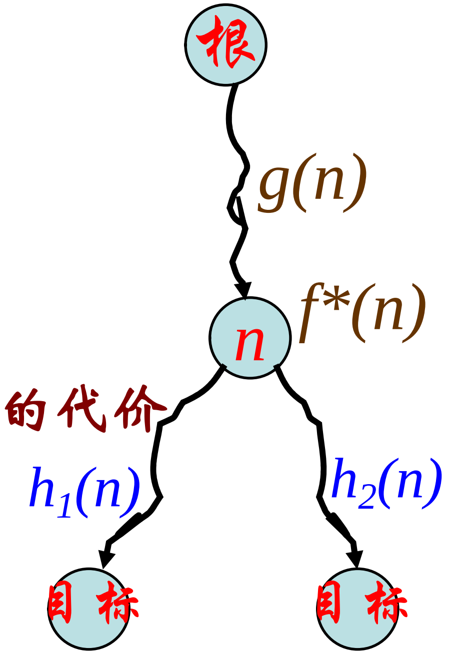
6.6 A* 算法

- 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*算法的核心是在某些情况下，我们一旦得到了解，就得到了优化解，于是算法可以停止
  - A*算法使用Best-first策略求解优化问题

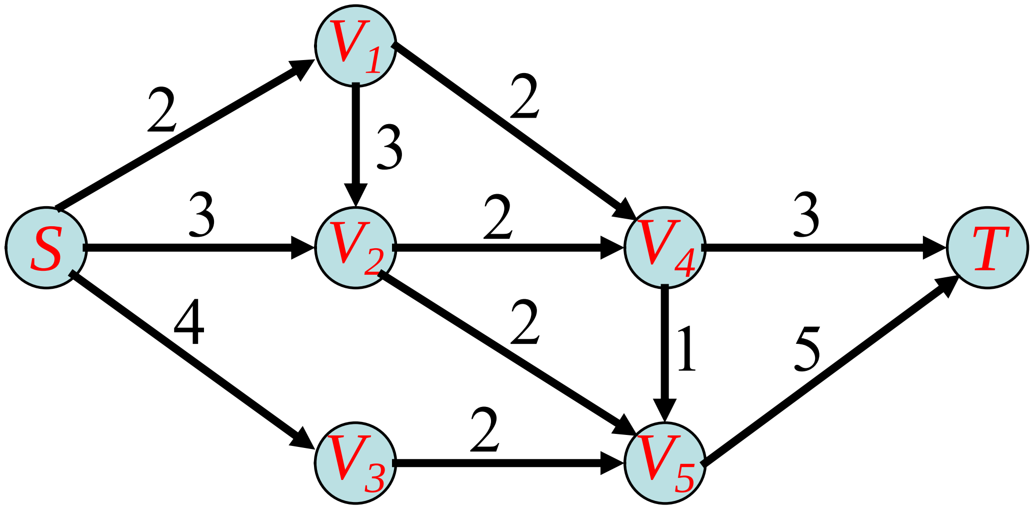
- A*算法关键—代价函数
  - 对于任意节点n
    - $g(n)$ = 从树根到n的代价
    - $h^*(n)$ = 从n到目标结点的优化路径的代价
    - $f^*(n) = g(n) + h^*(n)$是经过结点n到达目标结点的代价
  - $h^*(n) = ?$
    - 不知道!
    - 于是, $f^*(n)$也不知道
  - 估计$h^*(n)$
    - 使用任何方法去估计$h^*(n)$, 用$h(n)$表示$h^*(n)$的估计
    - $h(n) \le h^*(n)$总为真
    - $f(n)=g(n)+h(n) \le g(n)+h^*(n)=f^*(n)$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



$$h^*(n)=\min\{h_1(n), h_2(n)\}$$

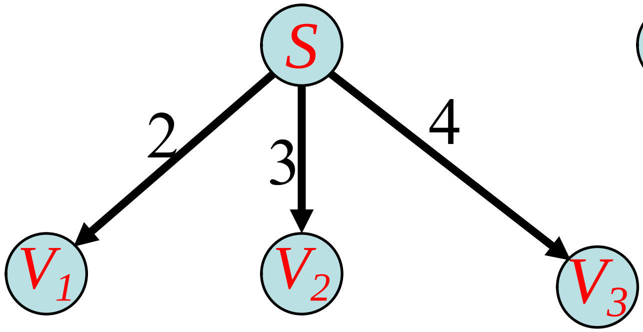
例1. 最短路径问题:

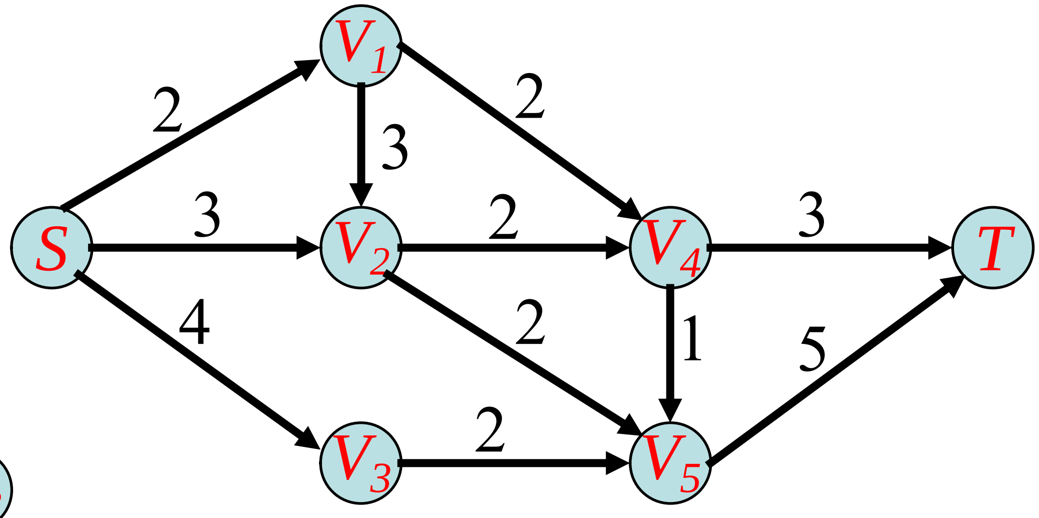
– 输入:



– 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$h^*(V_1)$

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

A*算法规则

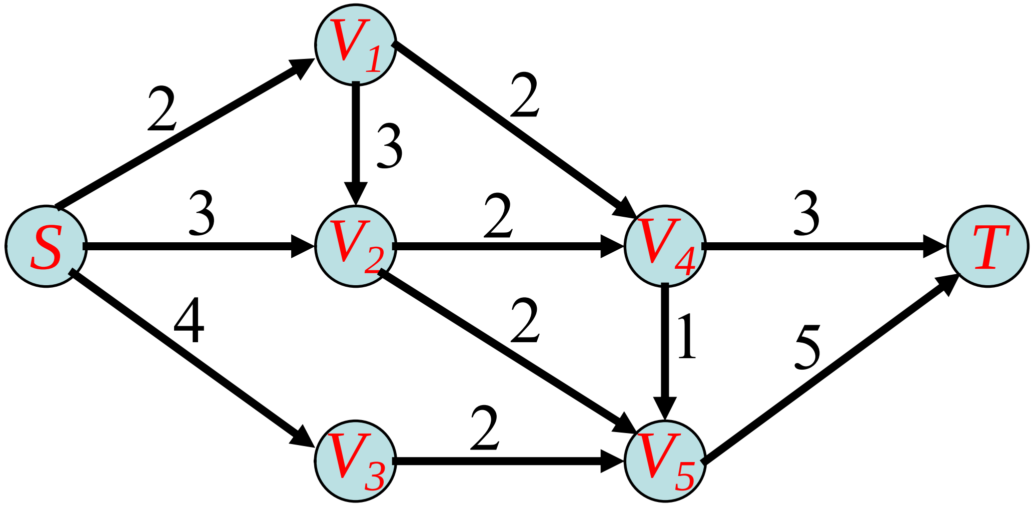
(1). 使用Best-first规则;

(2). 算法中代价函数$f(n)$定义为$g(n)+h(n)$, $g(n)$是从根到n的路径代价, $h(n)$是$h^*(n)$的估计值, 且对于所有n, $h(n)\leq h^*(n)$;

(3). 当选择到的结点是目标结点时, 算法停止, 该目标结点即为优化解.

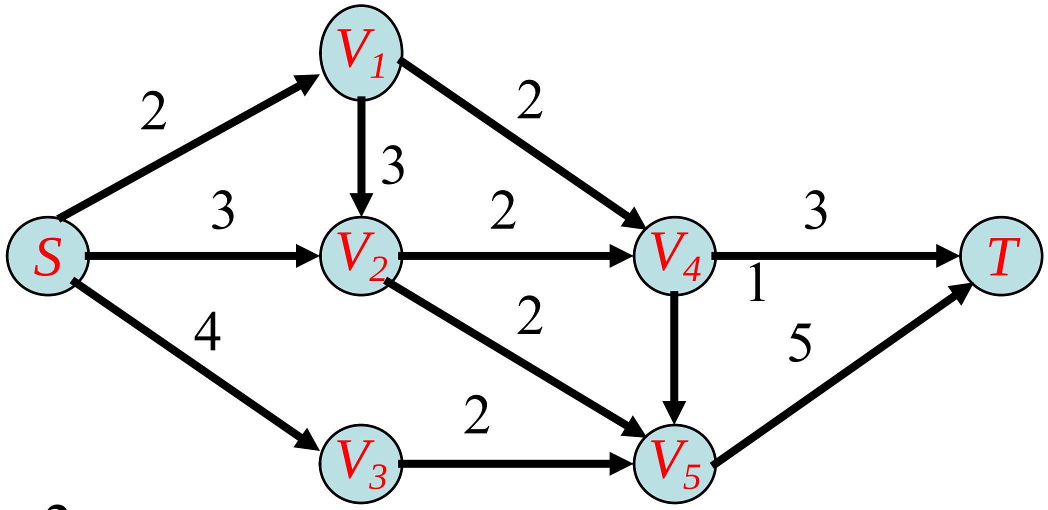
应用A*算法求解最短路径问题

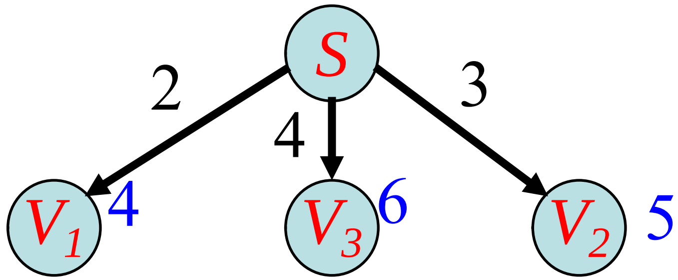
- 问题的输入：



HIT

Step 1



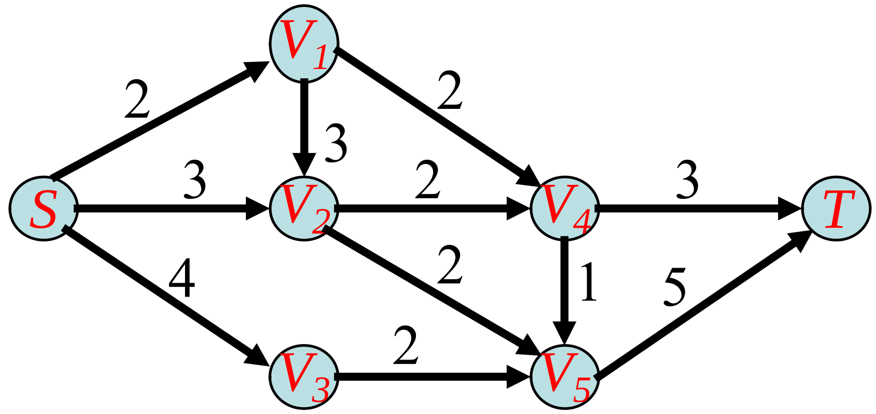


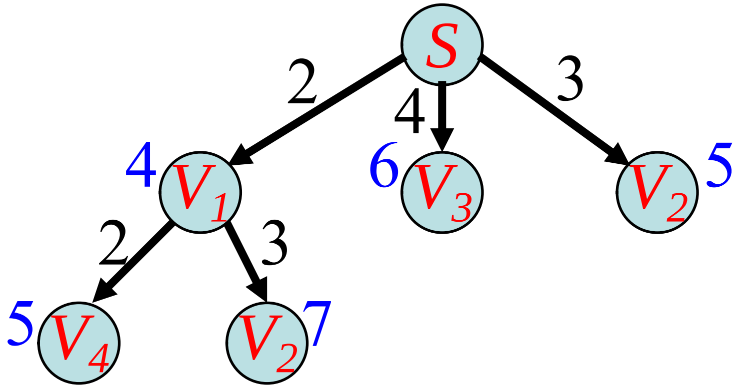
$g(V_1)=2$ $\quad h(V_1)=min\{2,3\}=2$ $\quad f(V_1)=2+2=4$

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

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

Step 2. 扩展V_1

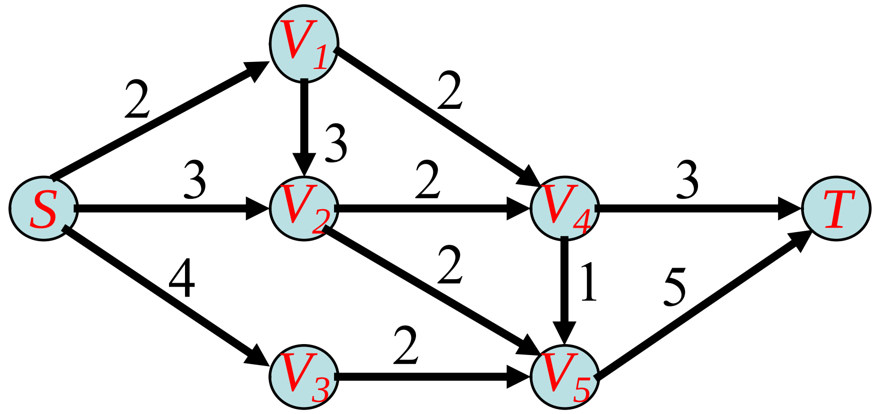


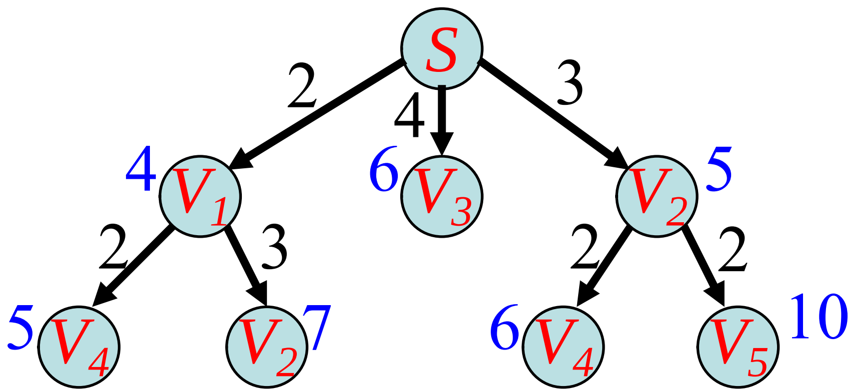


$g(V_4)=2+2=4 \quad h(V_4)=min\{3,1\}=1 \quad f(V_4)=4+1=5$

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

Step 3. 扩展V_2



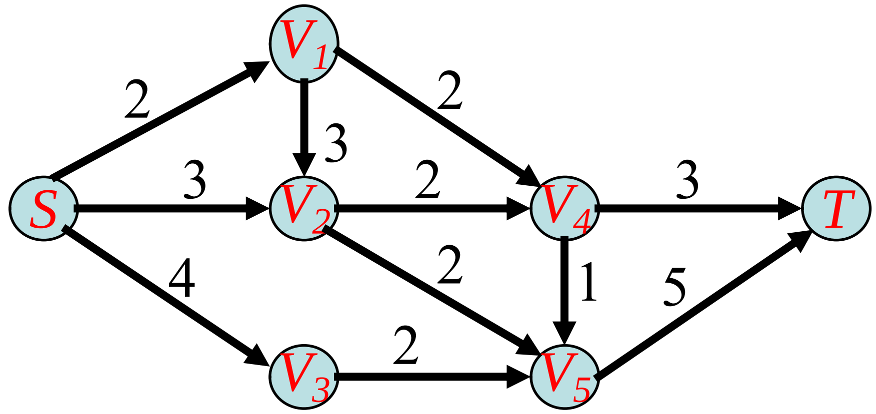


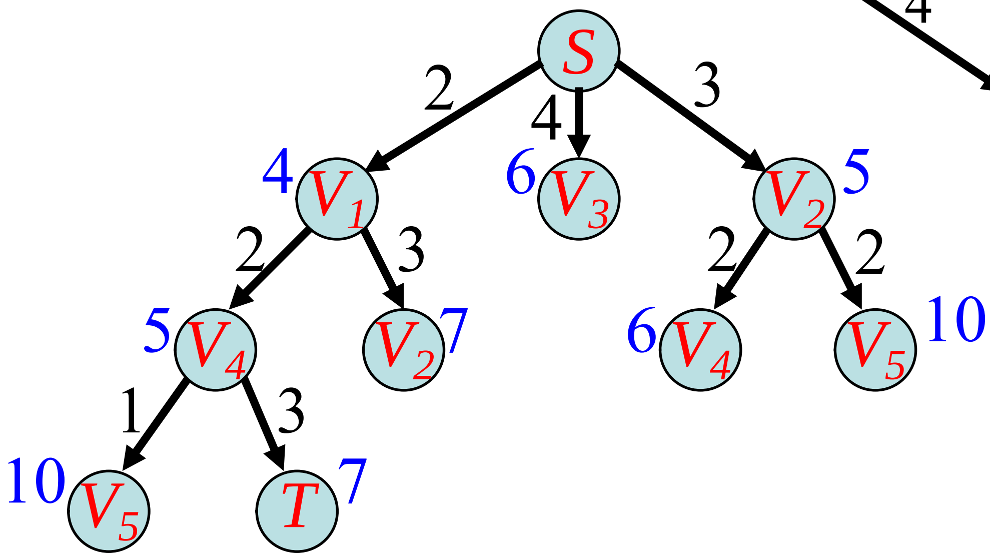
$$g(V_4)=3+2=5 \quad h(V_4)=min\{3,1\}=1 \quad f(V_4)=5+1=6$$

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

HIT

Step 4. 扩展V_4



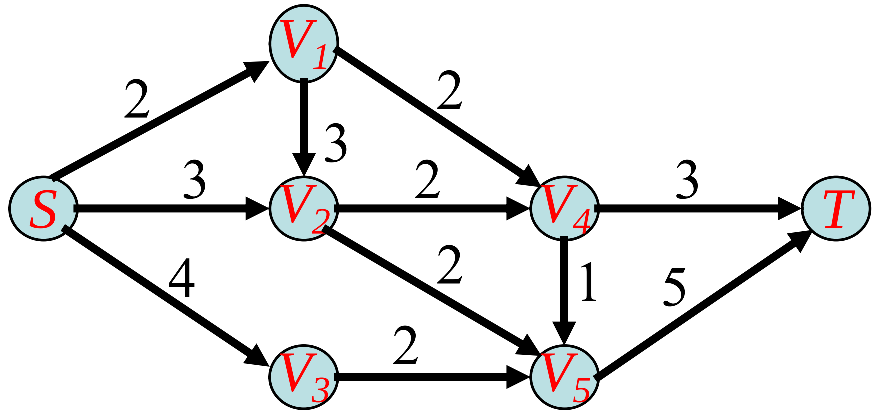


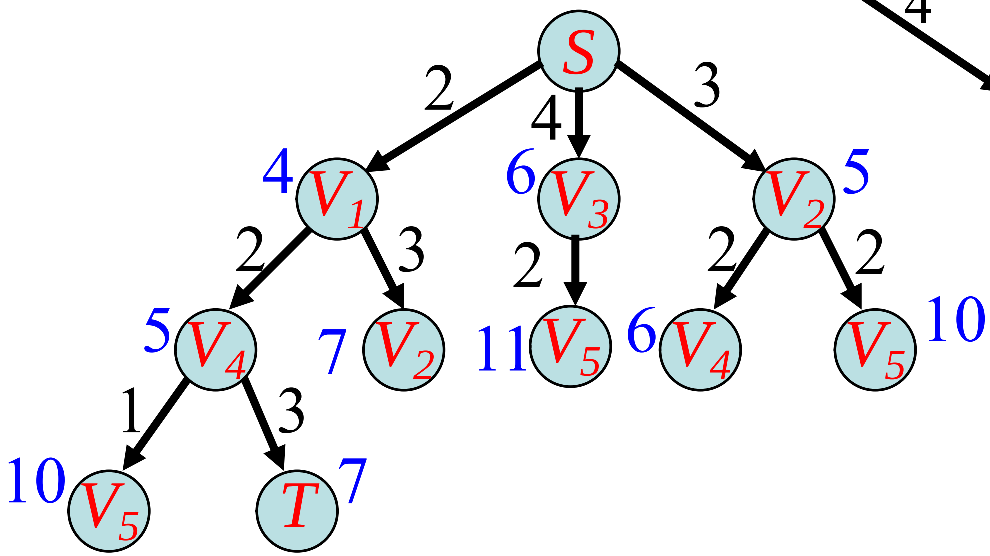
$g(V_5)=2+2+1=5$ $\quad$ $h(V_5)=min\{5\}=5$ $\quad$ $f(V_5)=5+5=10$

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

HIT

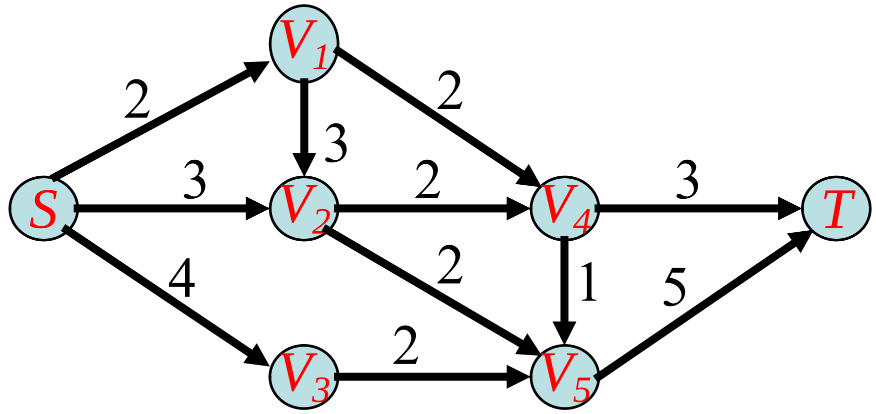
Step 5. 扩展V_3

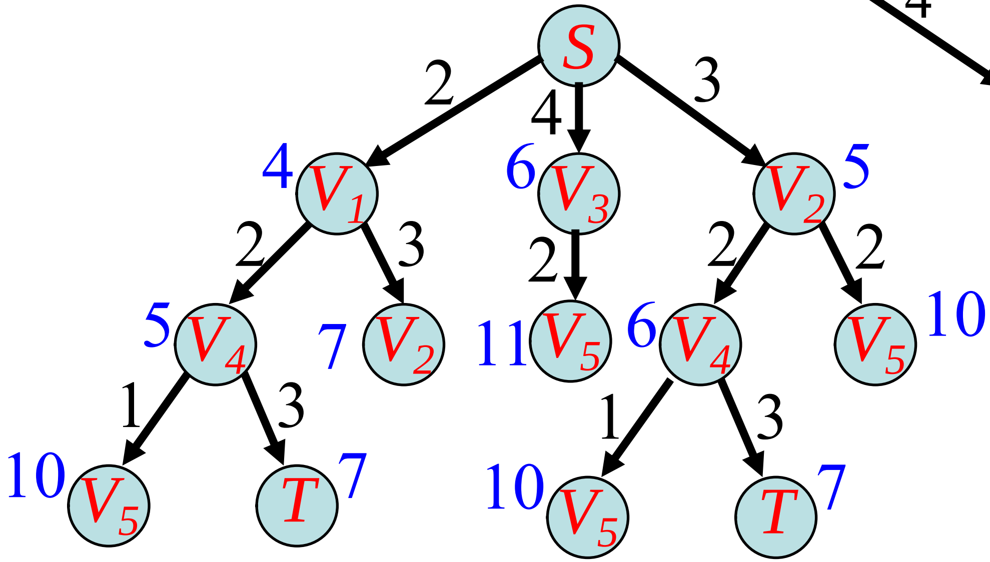




$g(V_5)=4+2=6$ $h(V_5)=min\{5\}=5$ $f(V_5)=6+5=11$

Step 6. 扩展V_4



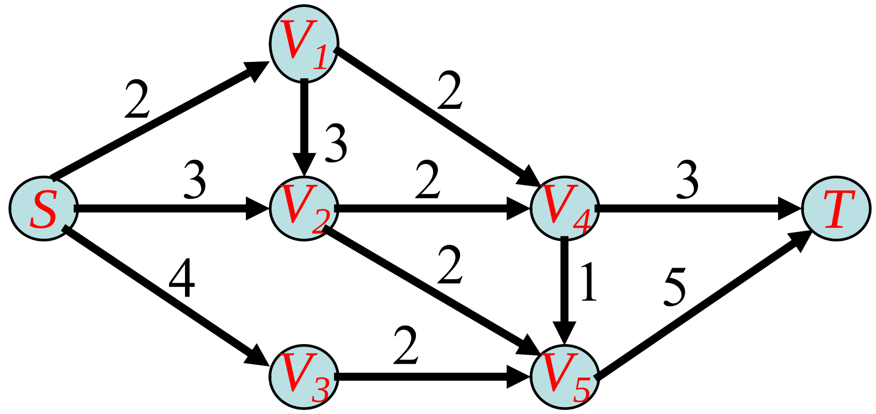


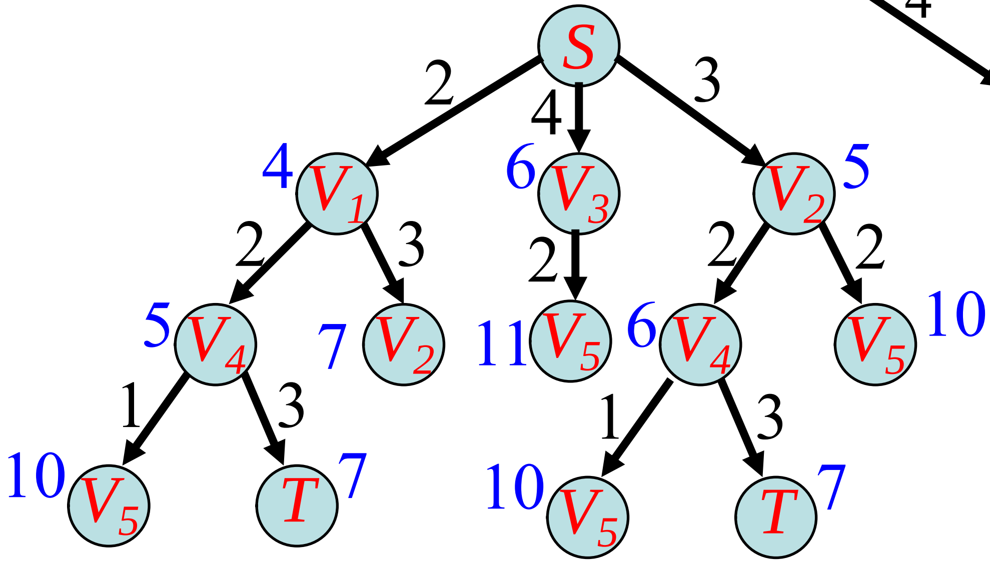
$g(V_5)=3+2+1=6$ $\quad$ $h(V_5)=min\{5\}=5$ $\quad$ $f(V_5)=6+5=10$

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

HIT

Step 7. 扩展T



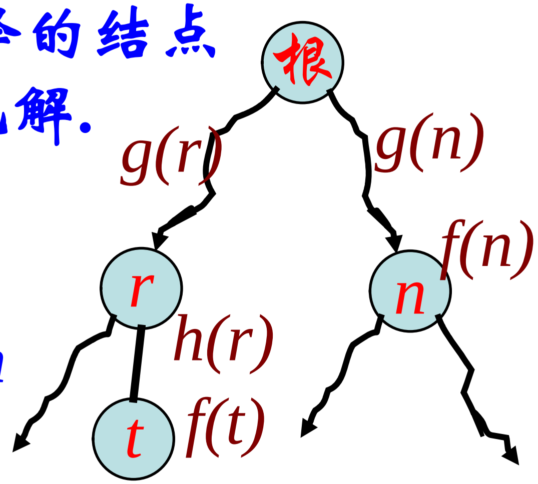


因为T是目标结点, 所以我们得到解:

$S \rightarrow V_1 \rightarrow V_4 \rightarrow T$

• A*算法的正确性

定理1. 使用Best-first策略搜索树，如果A*选择的结点是目标结点，则该结点表示的解是优化解.



证明.

令t是选中的目标结点，r是t父结点

假设存在一个优化解s，需要途经节点n

而n为晚于t而扩展的结点。

由于A*算法使用Best-first策略，$f(t) \leq f(n)$。

并且由于A*算法使用$h(n) \leq h^*(n)$估计规则，$f(t) \leq f(n) \leq f^*(n)$.

考虑到$f^*(n)=g(n)+h^*(n)$是经过n的可能解的最小代价

因而，我们有$f(t) \leq f^*(s)=g(n)+h^*(n)$.

同时，因为s为优化解，而$f(t)=g(t)+h(t)=f^*(t)$是t对应可能解的代价，故$f(t) \geq f^*(s)$，

所以，$f(t)=f^*(s)$，即Root经r到t亦为优化解。