HITWH
SE

第七章

Tree Searching Strategies

HITWH
SE

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

7.1 Motivation of Tree Searching

7.2 Optimal Tree Searching Strategies

7.3 Personnel Assignment Problem

7.4 Traveling Salesperson Problem

7.5 0/1 Knapsack Problem

7.6 The A* Algorithm

7.1 Motivation of Tree Searching

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

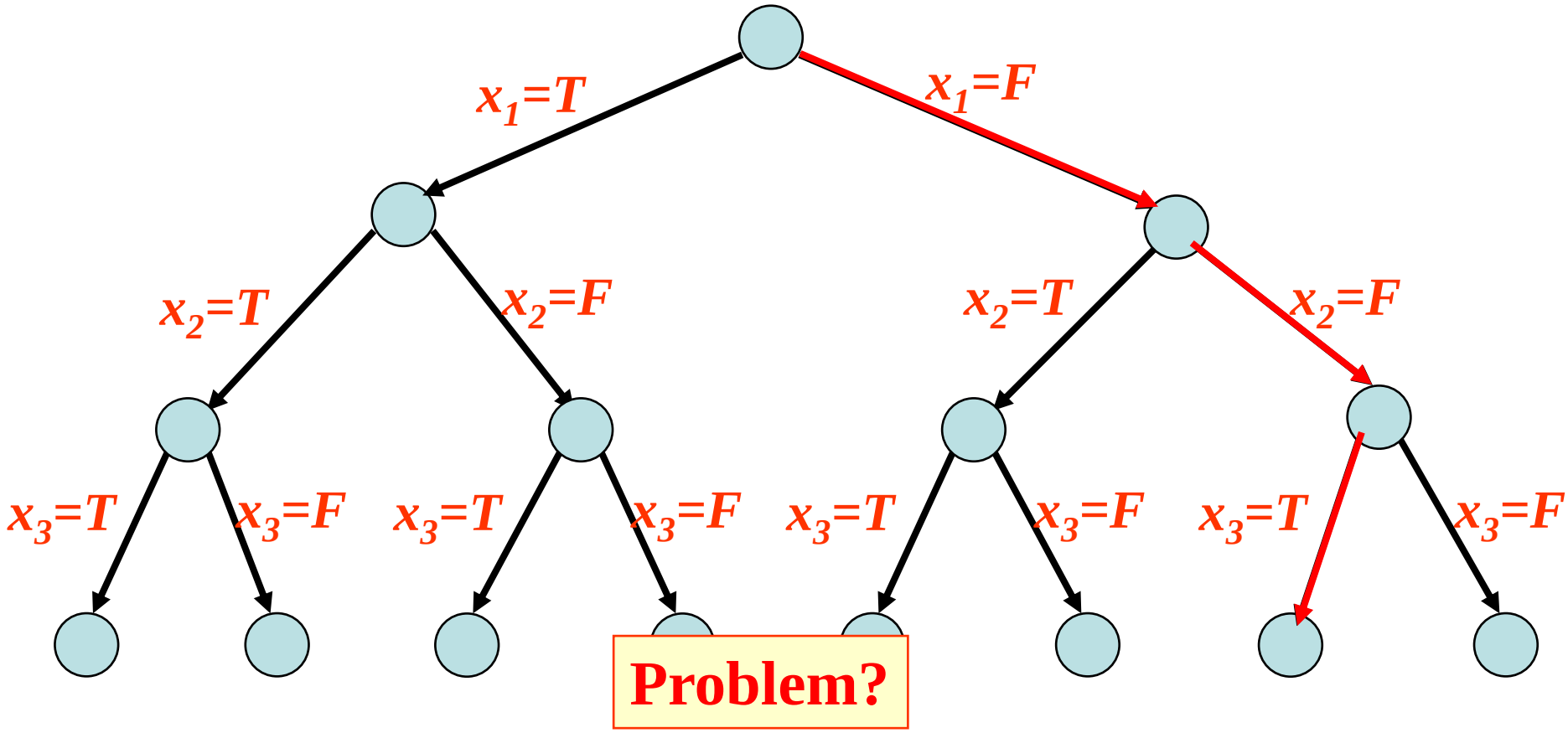
- 问题的定义
  - 输入：n个布尔变量$x_1, x_2, \ldots, x_n$
    关于$x_1, x_2, \ldots, x_n$的k个析取式
  - 输出：是否存在$x_1, x_2, \ldots, x_n$的一种赋值
    使得所有k个析取式皆为真

例如：$\neg x_1$，$x_2 \vee x_3$，x_3，$\neg x_2$

- 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x_1, x_2, x_3为例)

对于：$-x_1$，$x_2 \vee x_3$，x_3，$-x_2$



问题的解是由根到叶的一条路径

8-Puzzle问题

- 问题的定义
  - 输入：具有8个编号小方块的魔方

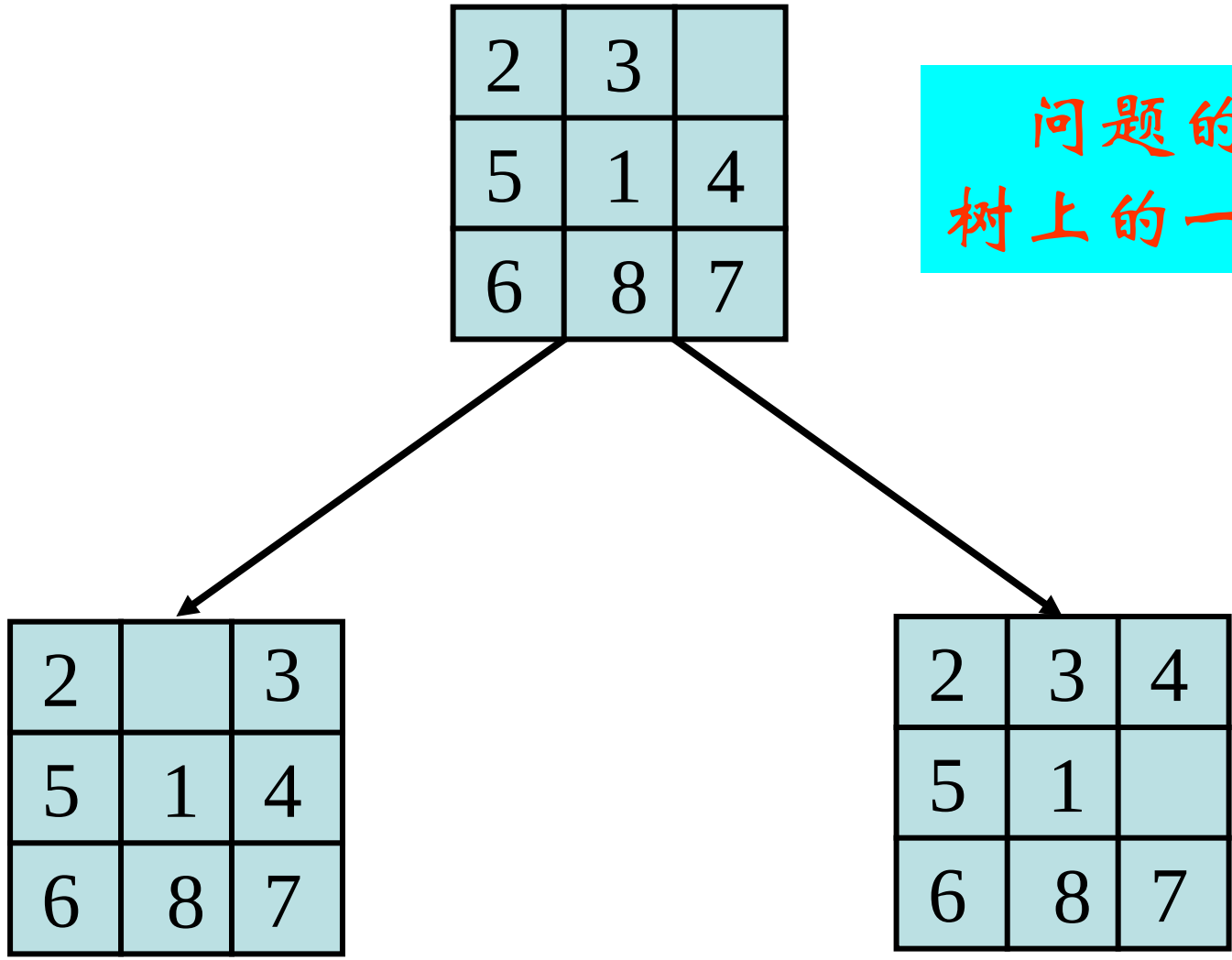
2	3	
5	1	4
6	8	7

  - 输出：移动系列，经过这些移动，魔方到达如下状态

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
树上的一个节点

HITWHSE

7.2 Optimal Tree Searching Strategies

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

Hill Climbing

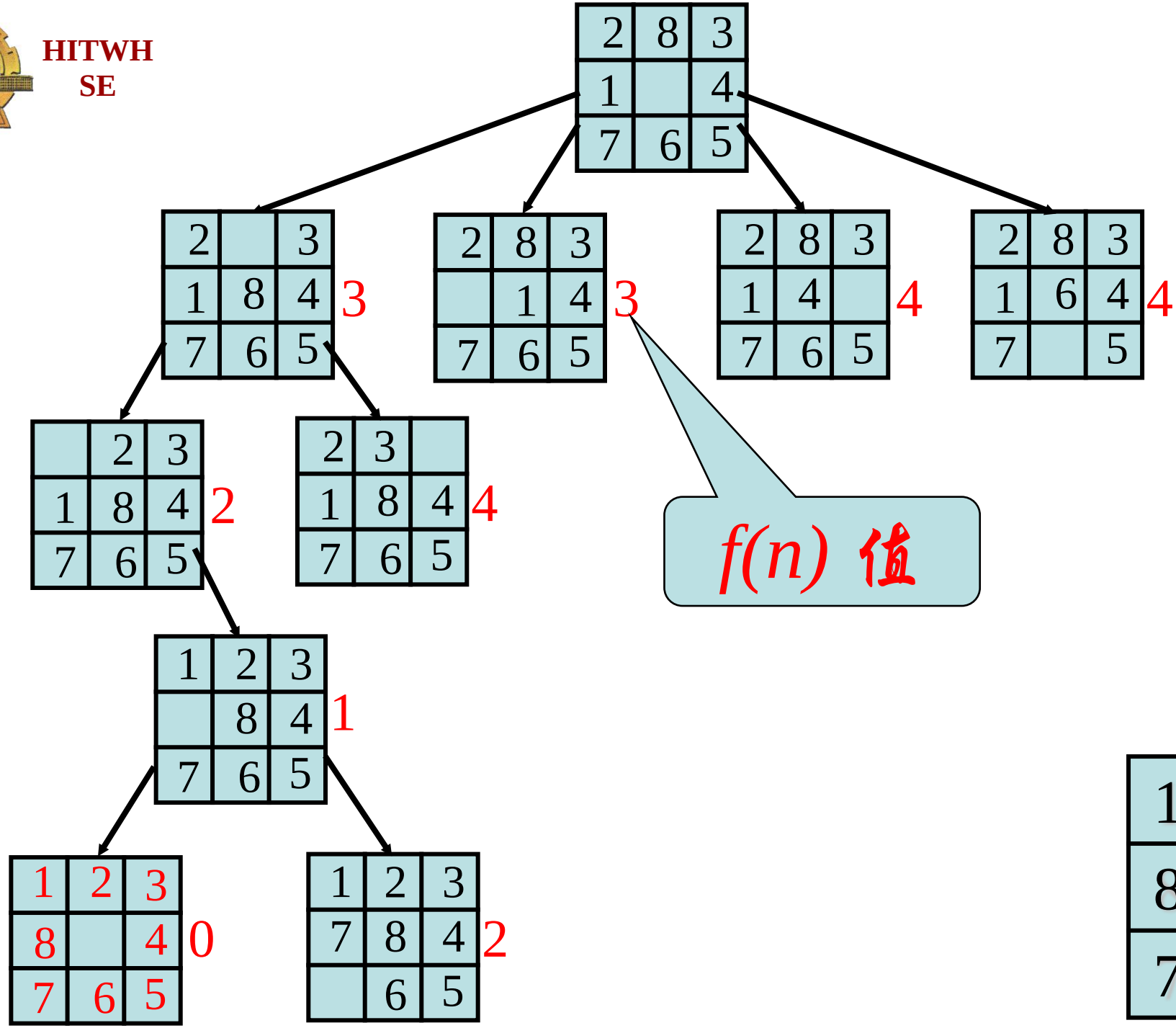
- 基本思想
  - 在深度优先搜索过程中，我们经常遇到多个节点可以扩展的情况，首先扩展哪个?
  - 爬山策略使用贪心方法确定搜索的方向，是优化的深度优先搜索策略
  - 爬山策略使用启发式测度来排序节点扩展的顺序
    - 使用什么启发式测度与问题本身相关

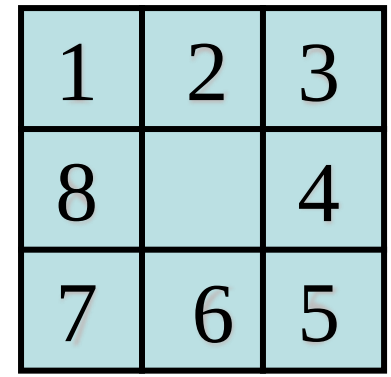
1	2	3
8		4
7	6	5

- 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:

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.


3
3
4
4
2
4
1
0
2
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 Strategy

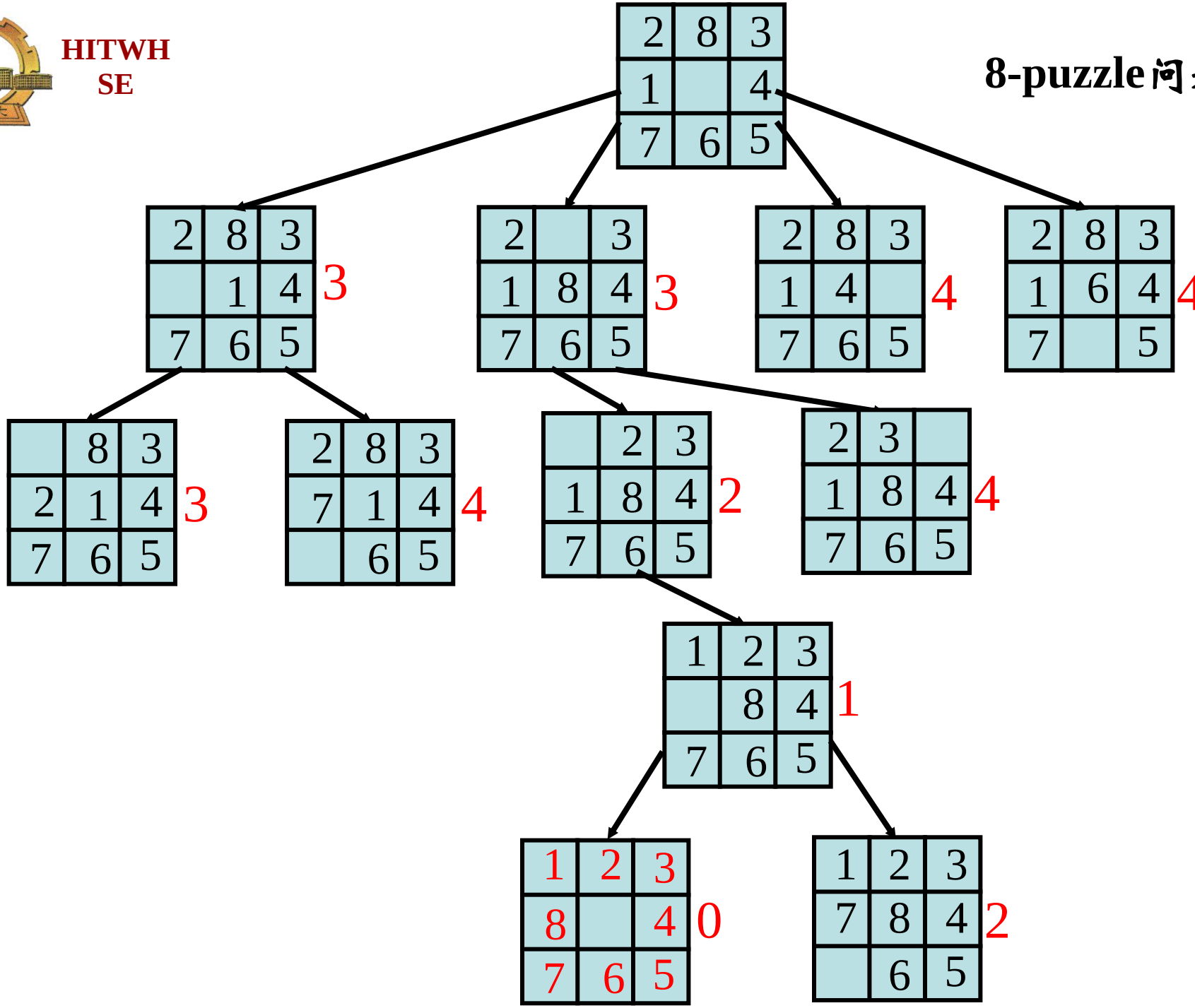
- 基本思想(也称最少代价优先，Least-cost-first)
  - 结合深度优先和广度优先的优点于一个方法
  - 根据一个评价函数，在目前产生的所有节点中选择具有最小评价函数值的节点进行扩展.
  - 具有全局优化观念，而爬山策略仅具有局部优化观念.

- **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

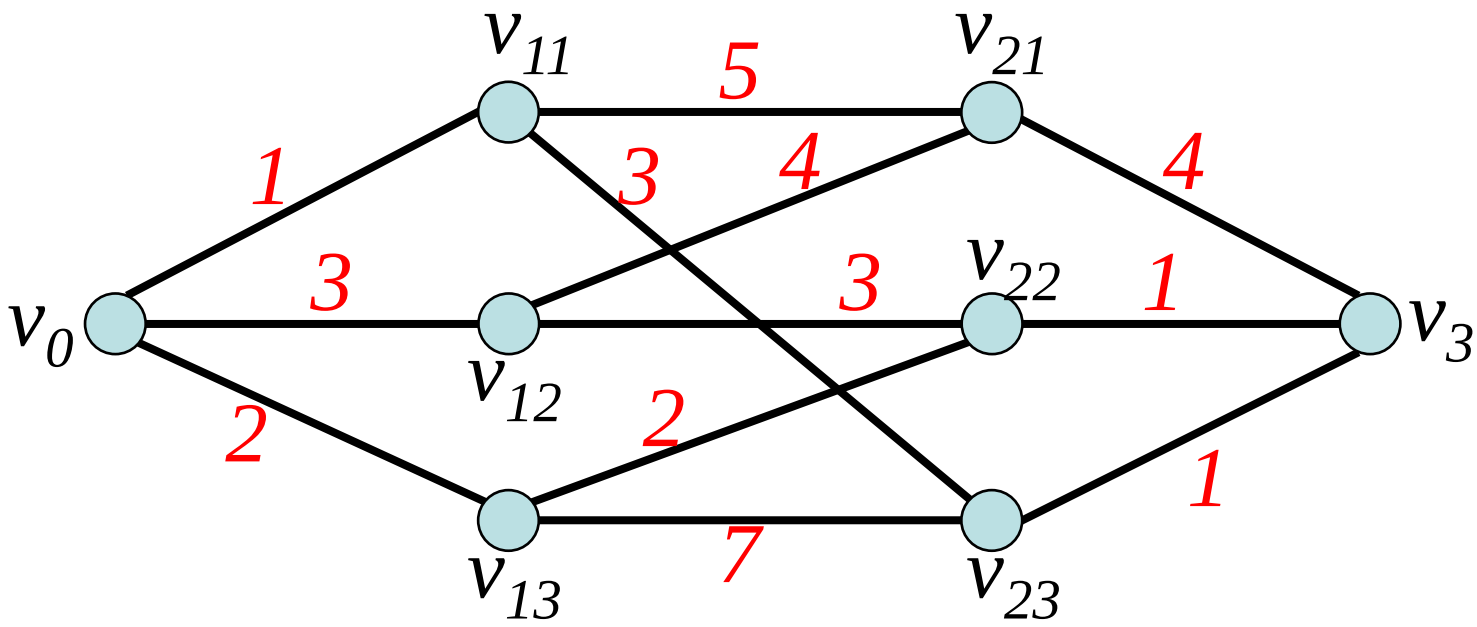
8-puzzle问题实例



Branch-and-Bound Strategy

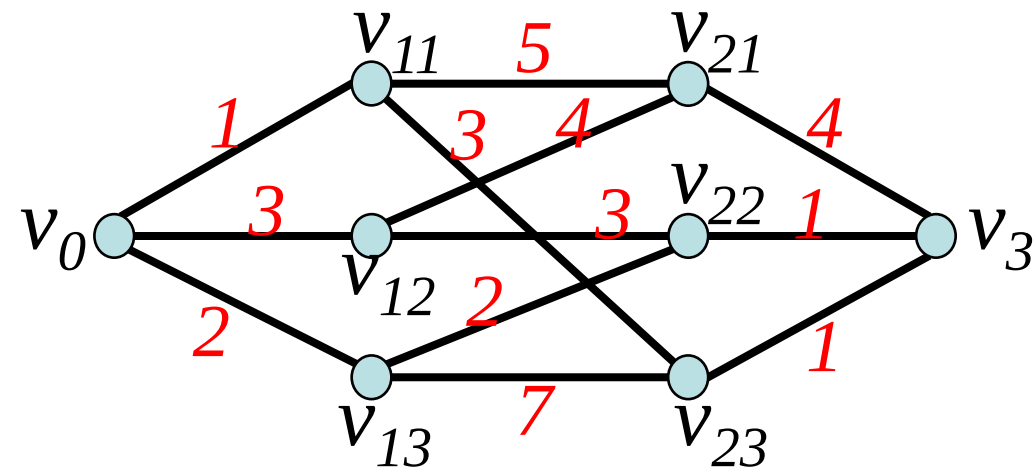
- 上述方法很难用于求解优化问题
- 分支界限策略可以有效地求解组合优化问题
- 分支界限基本思想
  - 迅速找到一个可行解
  - 将该可行解作为优化解的一个界限
  - 利用界限裁剪解空间，提高求解的效率

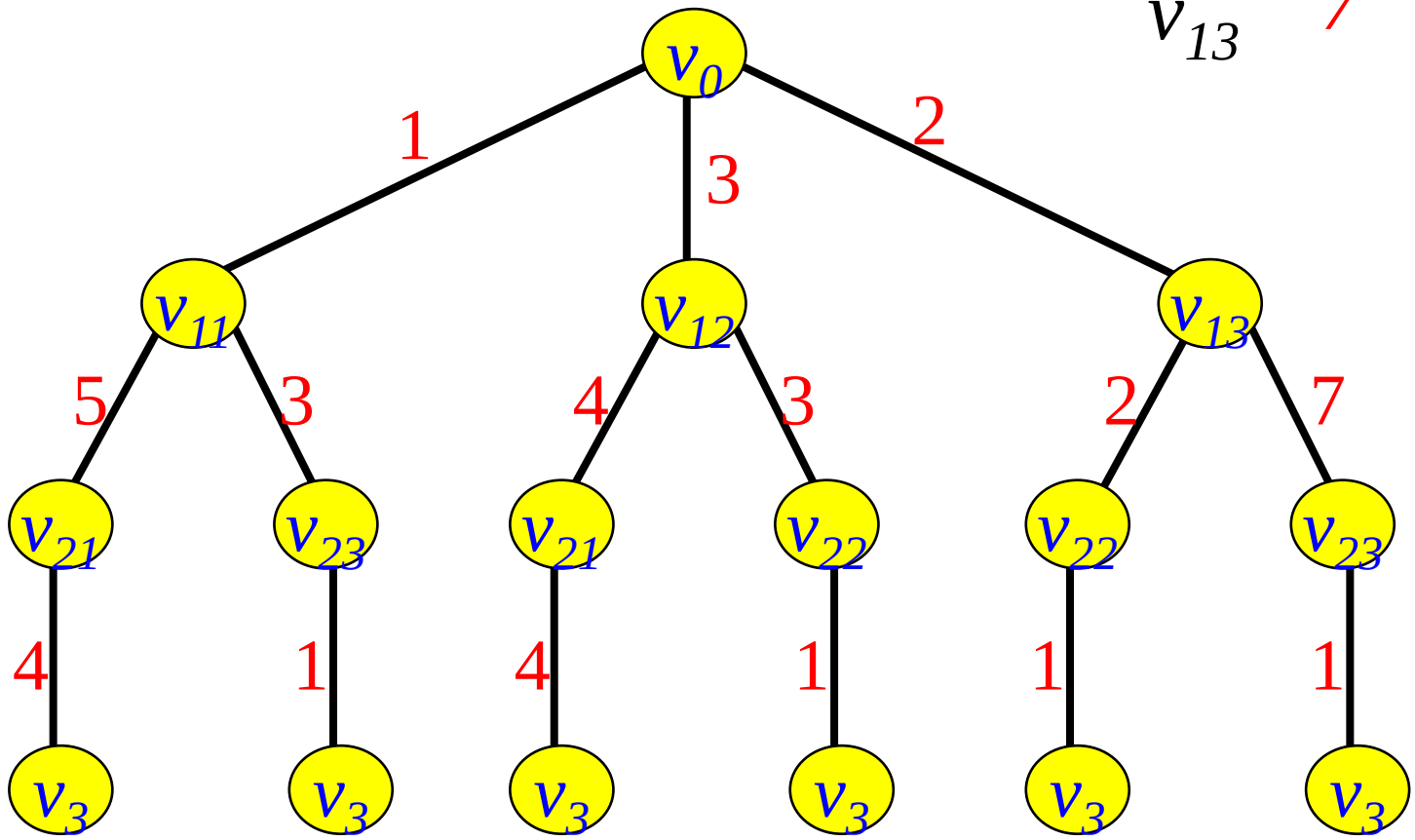
- 多阶段图搜索问题
  - 输入：多阶段图



  - 输出：从v_0到v_3的最短路径

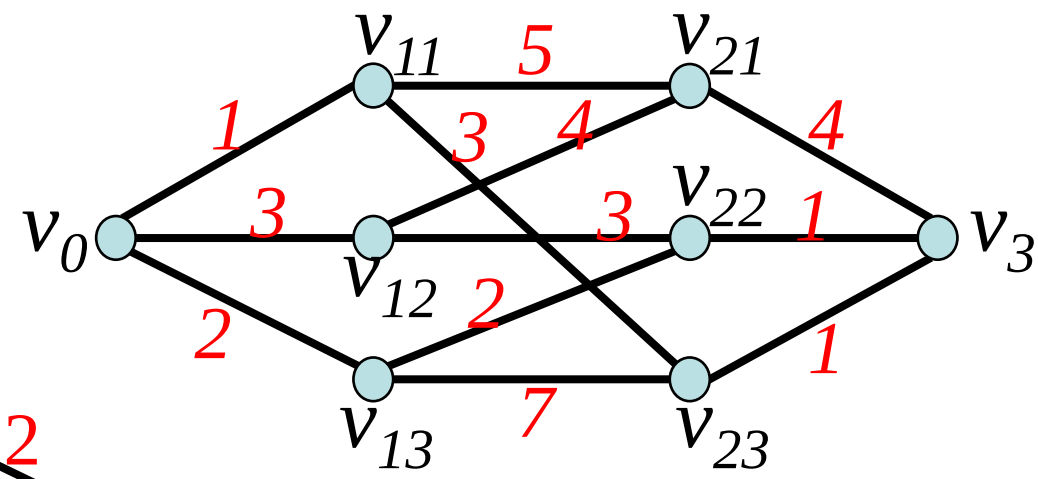
- 问题的树表示

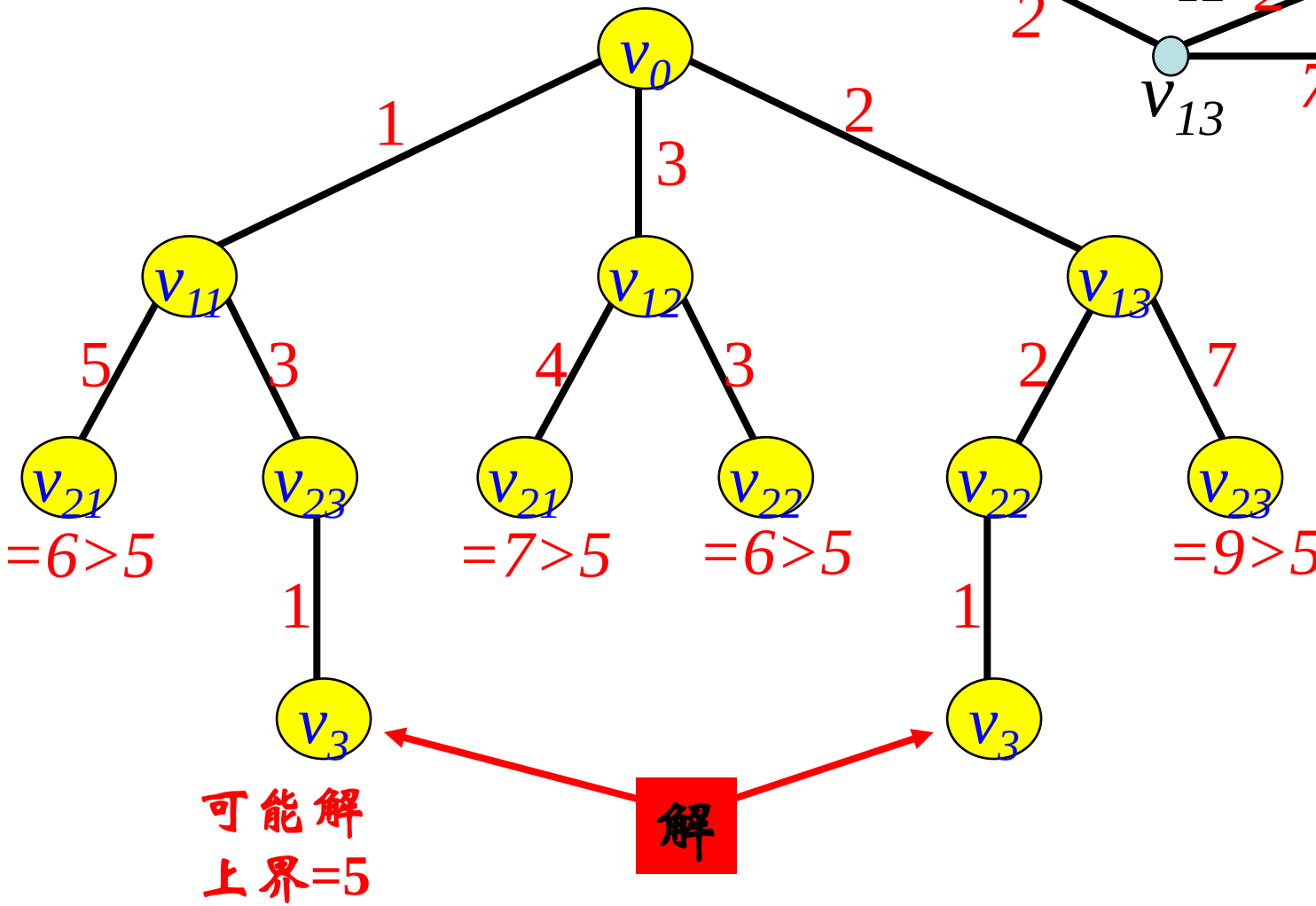

v_{11}
5
v_{21}
1
3
4
4
v_{22}
v_0
3
3
1
v_3
v_{12}
2
2
1
v_{13}
7
v_{23}


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)
    - 爬山法
    - Best-First
  - 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.

7.3 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
  - 人的有序集合$P=\{P_1, P_2, \ldots, P_n\}$, $P_1<P_2<\ldots<P_n$
  - 工作的集合$J=\{J_1, J_2, \ldots, J_n\}$, J是偏序集合
  - 矩阵$[C_{ij}]$, C_{ij}是P_i被分配工作J_j的代价
- 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$i\neq j$，$f(P_i)\neq f(P_j)$.

问题的定义

例. 给定P=$\{P_1, P_2, P_3\}$，$J=\{J_1, J_2, J_3\}$，$J_1 \le J_3$, $J_2 \le J_3$.

$P_1 \to J_1$、$P_2 \to J_2$、$P_3 \to J_3$ 是否为可能的解？

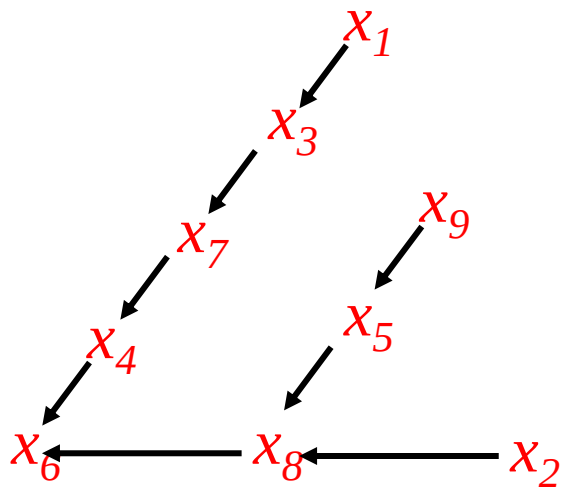
$P_1 \to J_1$、$P_2 \to J_3$、$P_3 \to J_2$ 是否为可能的解？

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



$x \rightarrow y$表示$x \leq y$

拓扑排序：

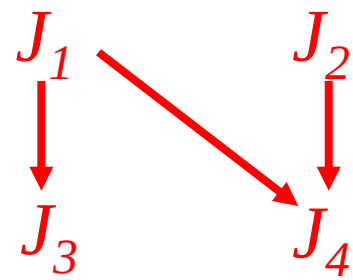
$x_1\ x_3\ x_7\ x_4\ x_9\ x_5\ x_2\ x_8\ x_6$

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

- 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.

例. $P=\{P_1, P_2, P_3, P_4\}$, $J=\{J_1, J_2, J_3, J_4\}$, J的偏序如下



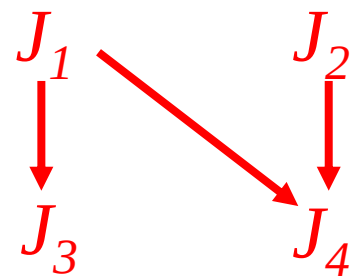
则J所有的拓扑排序序列有：(J_1, J_2, J_3, J_4)、(J_1, J_2, J_4, J_3)、(J_1, J_3, J_2, J_4)、(J_2, J_1, J_3, J_4)、(J_2, J_1, J_4, J_3).

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

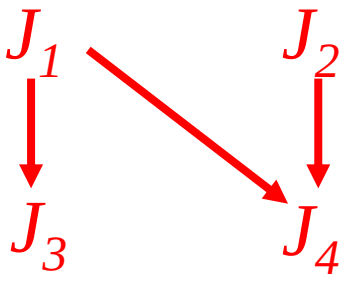
- 问题的解空间

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.

例. $P=\{P_1, P_2, P_3, P_4\}$, $J=\{J_1, J_2, J_3, J_4\}$, J的偏序如下

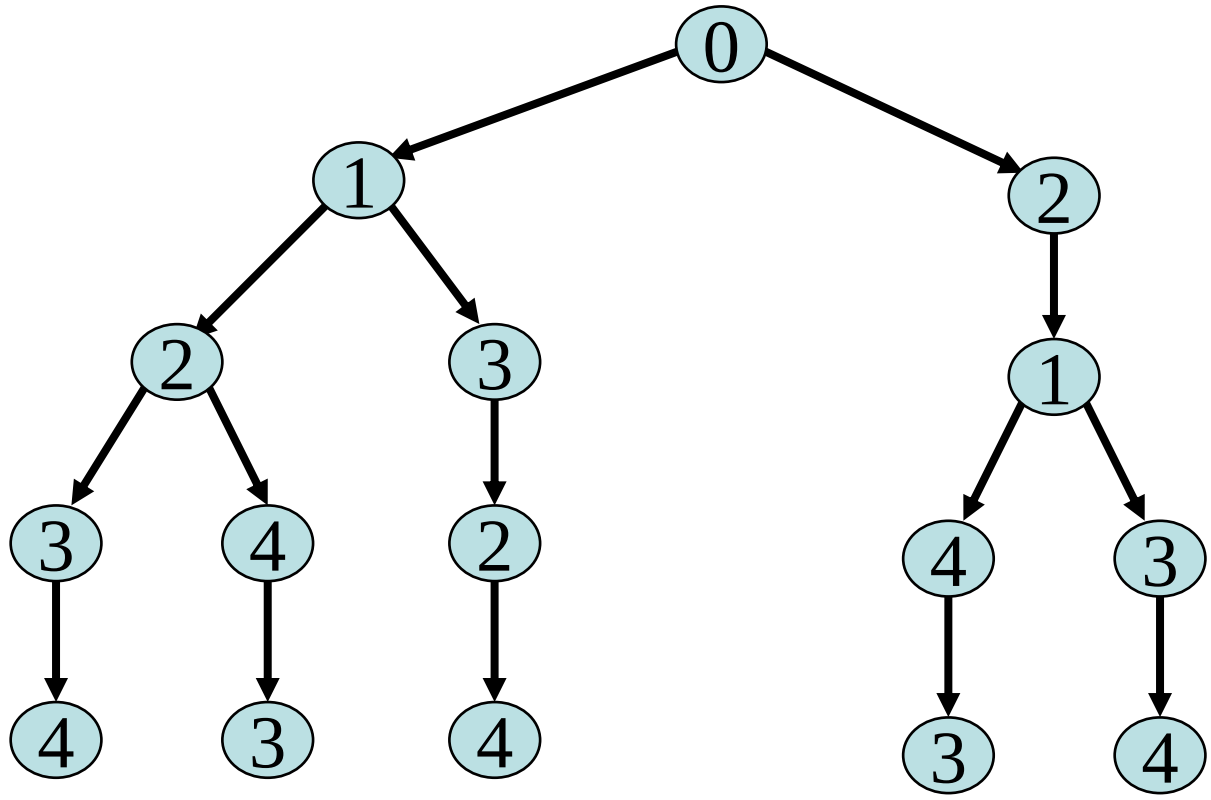


解空间是工作集合所有拓扑排序的序列集合，每个序列对应于一个可能的解



(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)



被分配的人

P_1

P_2

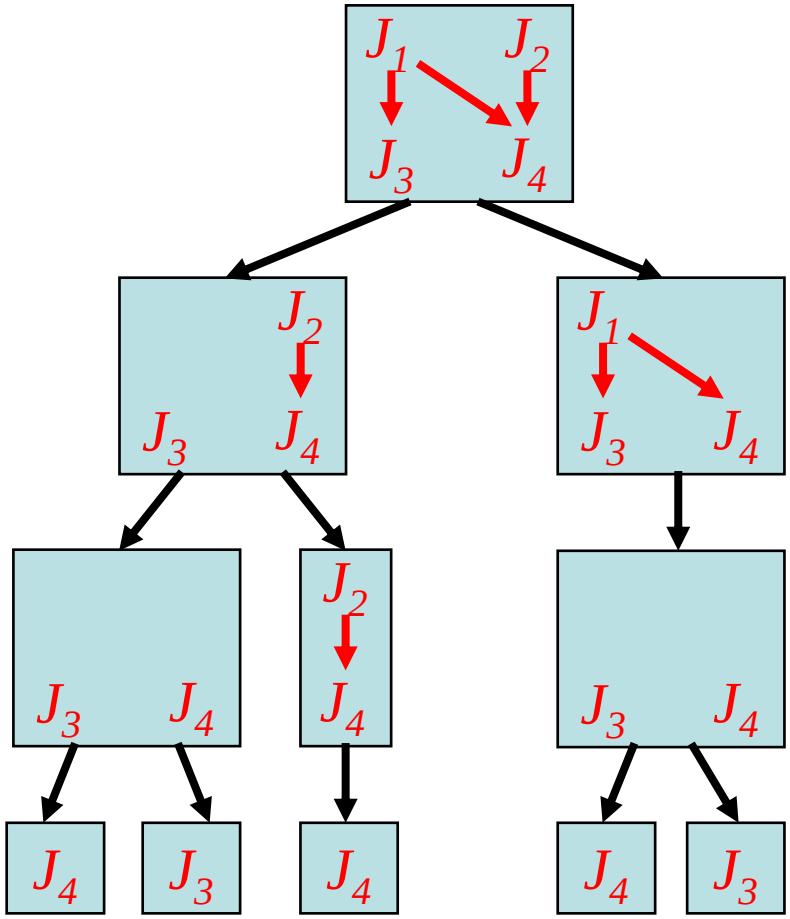
P_3

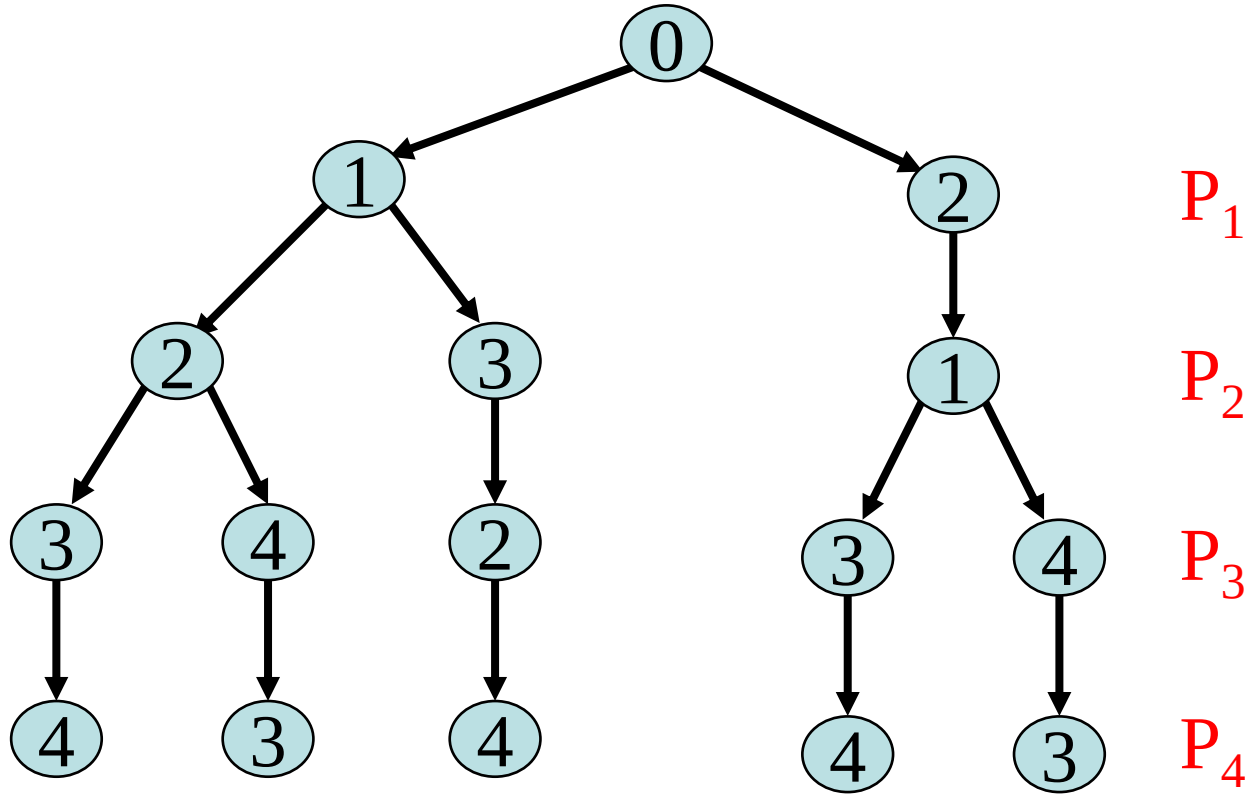
P_4

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

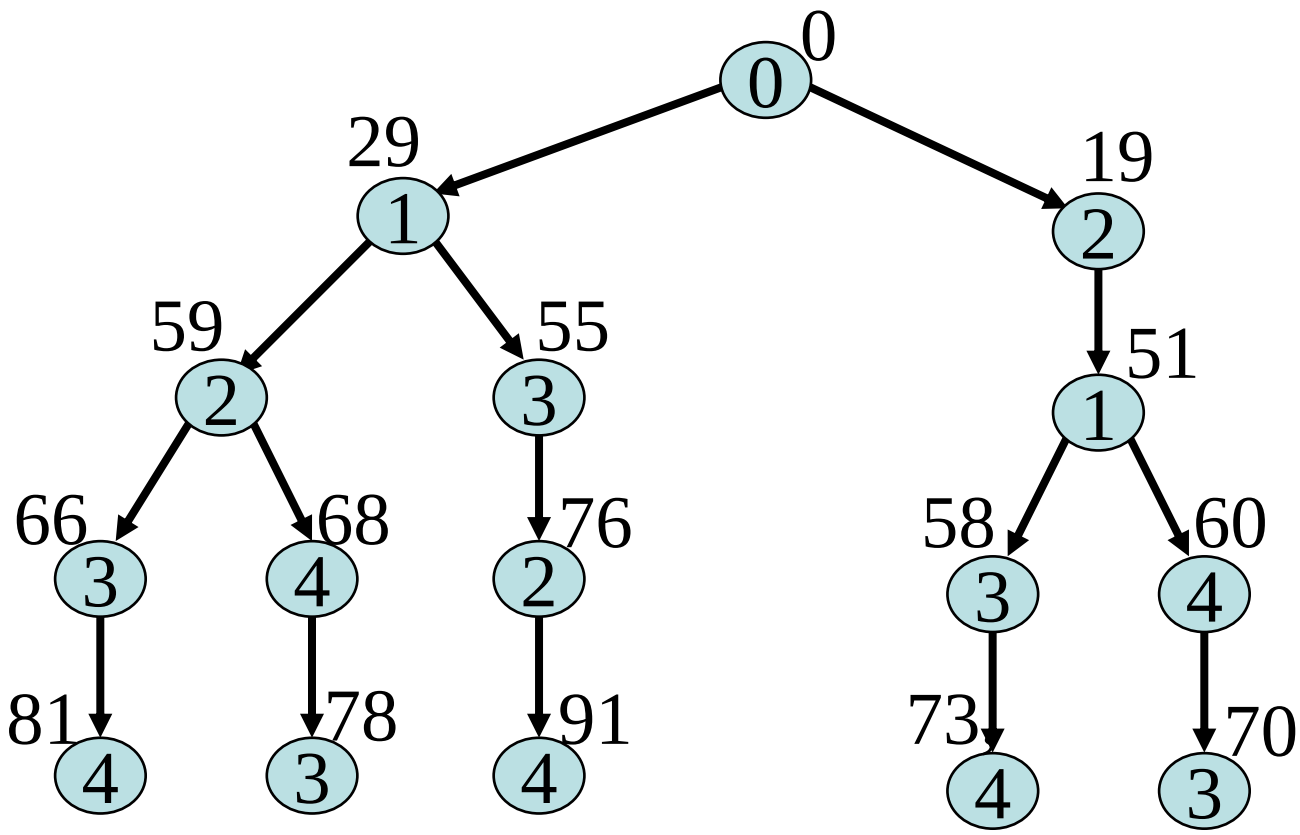
输入：偏序集合*S*, 树根*root*.

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*.

- 解空间的加权树表示

	J_1	J_2	J_3	J_4
P_1	29	19	17	12
P_2	32	30	26	28
P_3	3	21	7	9
P_4	18	13	10	15



被分配的人员

1

2

3

4

裁剪效果不好，搜索代价高！！

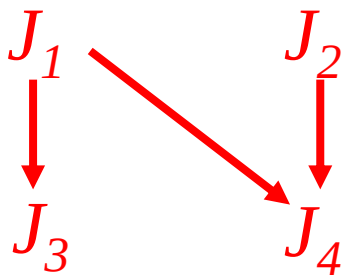
如何改进？

求解问题的分支界限搜索算法

- 计算解的代价的下界

例：

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



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

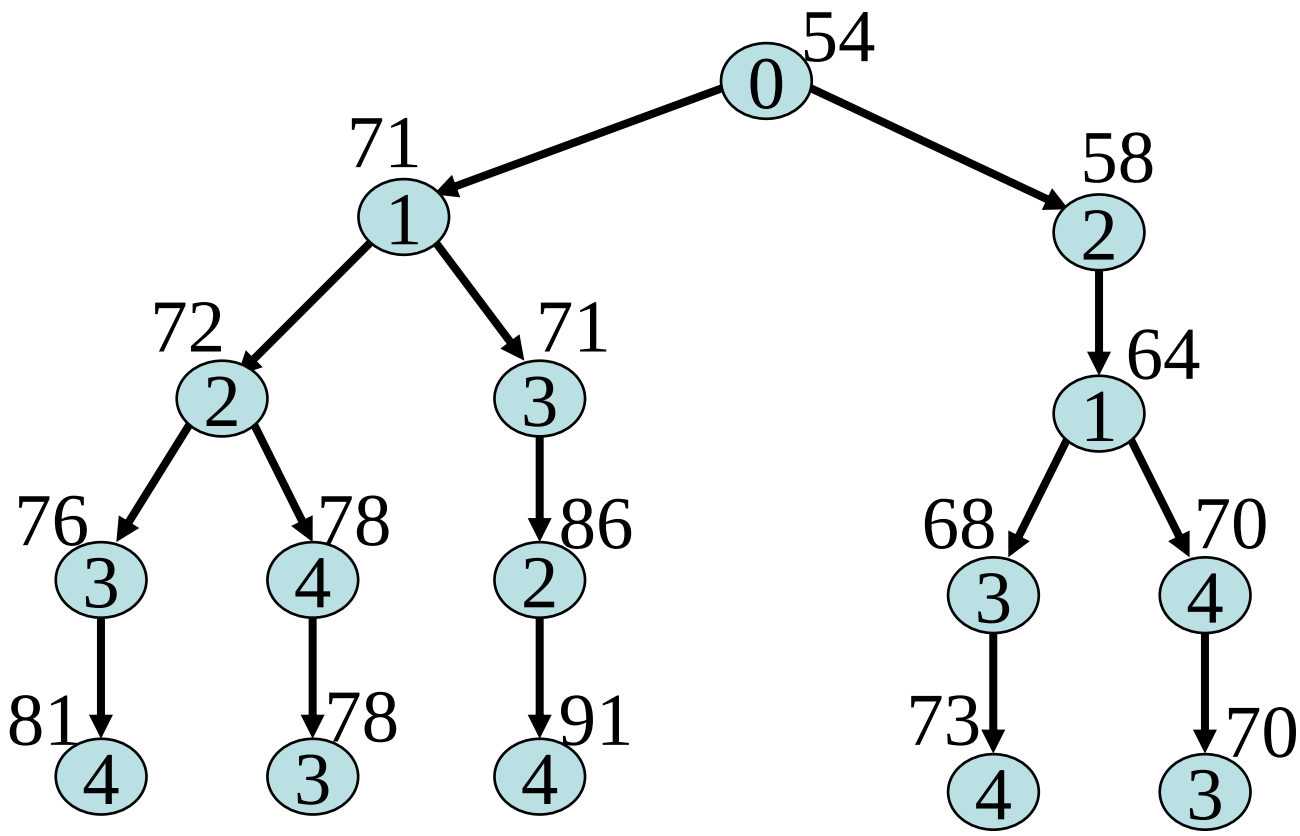
命题2：把代价矩阵每行(列)各元素减去同一个数，不影响优化解的求解.

- 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.

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

- 改进后解空间的加权树表示



被分配的人员

1

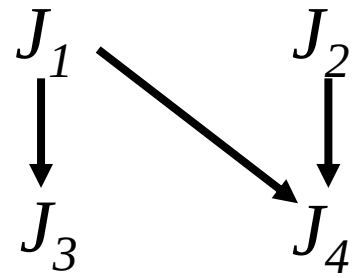
2

3

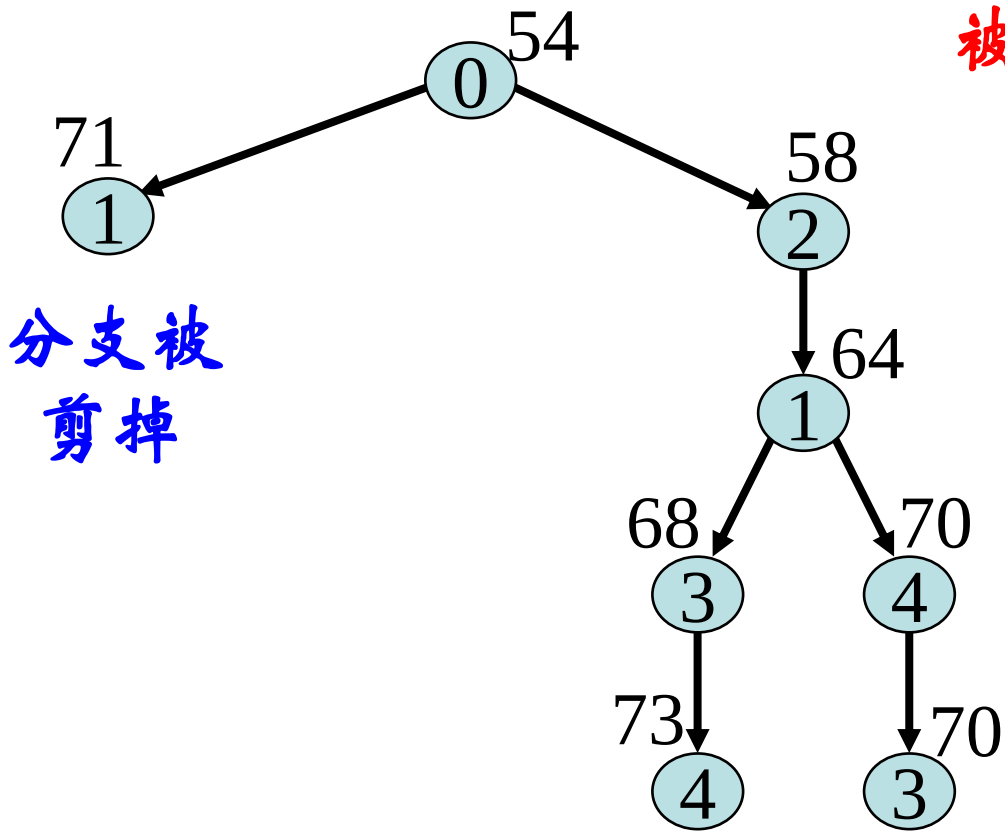
4

算法的思想(用爬山法)

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



	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



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

7.4 Traveling Salesperson 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：
  - 选择图上的边(i, j)，满足
    - 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
    - 使左子树代价下界不变，
    - 使右子树代价下界增加最大
  - 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大于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

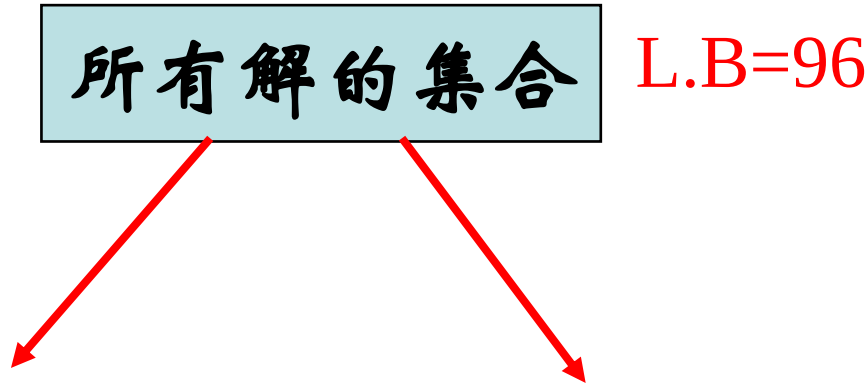
	$j=1$	2	3	4	5	6	7	
$i=1$	∞	0	83	9	30	6	50	-3
2	0	∞	66	37	17	12	26	-4
3	29	1	∞	19	0	12	5	-16
4	32	83	66	∞	49	0	80	-7
5	3	21	56	7	∞	0	28	-25
6	0	85	8	42	89	∞	0	-3
7	18	0	0	0	58	13	∞	-26
			-7	-1			-4	

- 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

为什么？

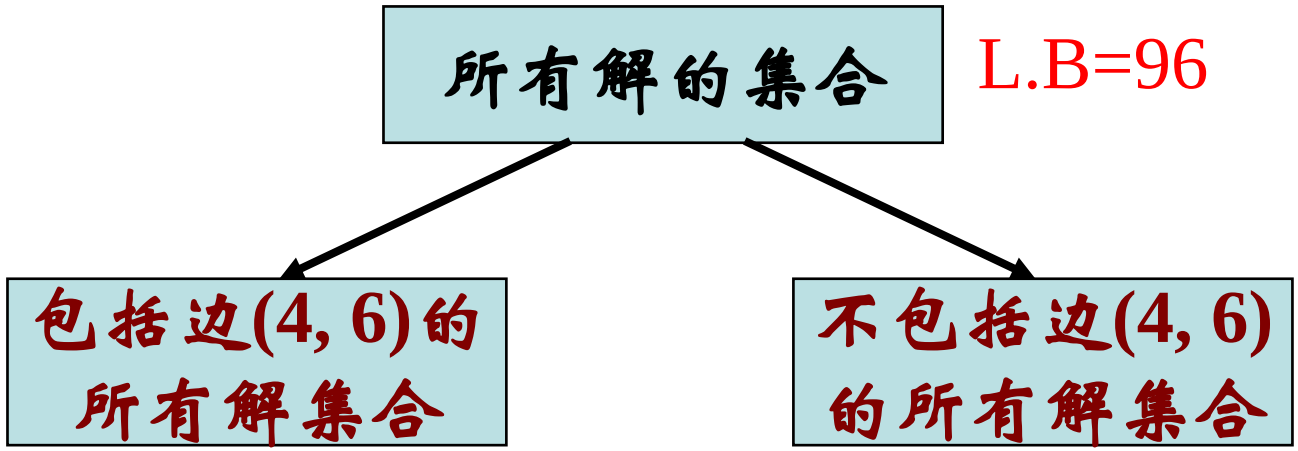
- 得到如下根结点及其代价下界



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



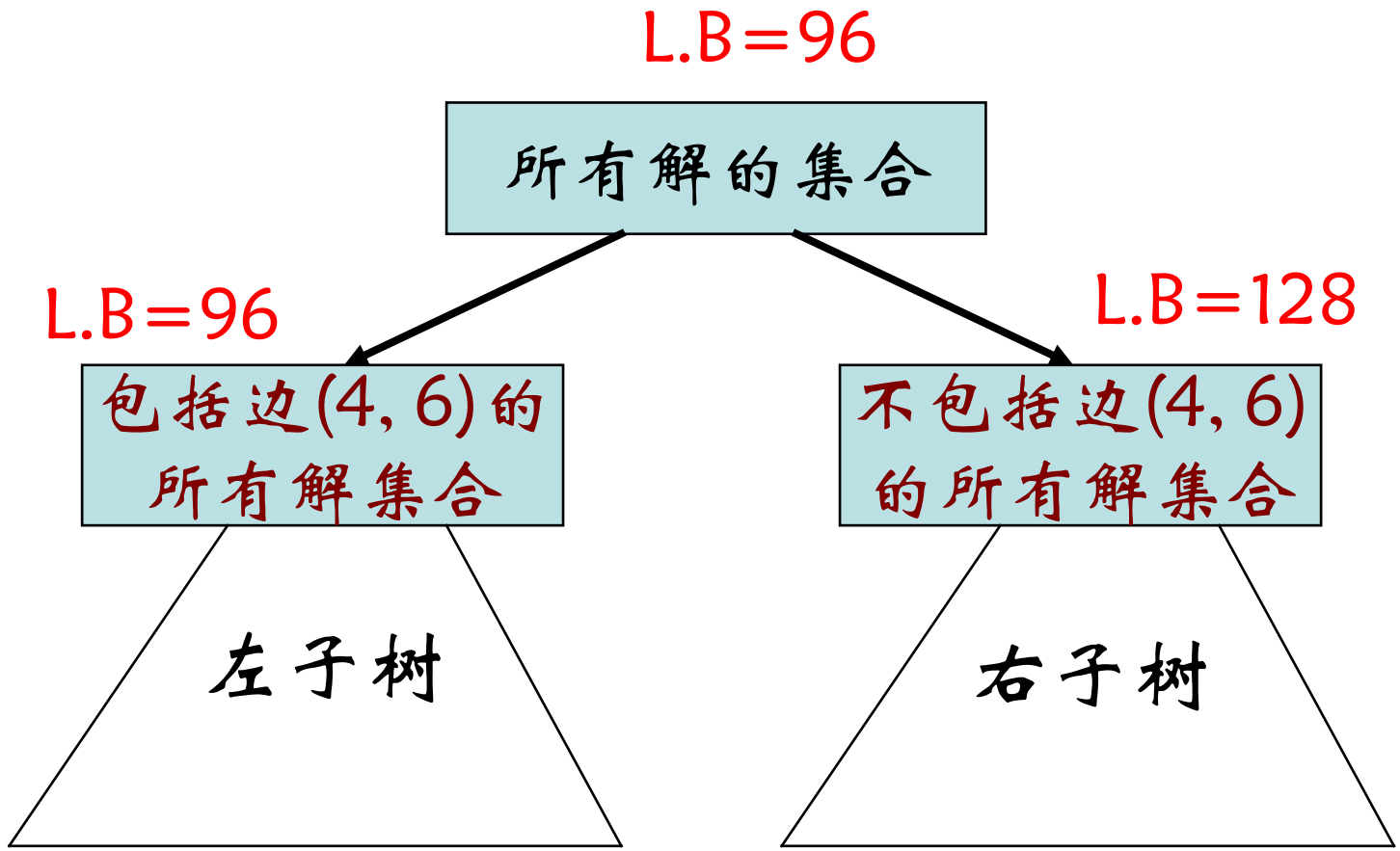
Cost-R(1,2)=6+0=6
Cost-R(2,1)=12+0=12
Cost-R(3,5)=1+17=18
Cost-R(4,6)=32+0=32
Cost-R(5,6)=3+0=3
Cost-R(6,1)=0+0=0
Cost-R(6,7)=0+5=5
Cost-R(7,2)=0+0=0
Cost-R(7,3)=0+8=8
Cost-R(7,4)=0+7=7

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

➢ 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∞.

	$j=1$	2	3	4	5		7
$i=1$	∞	0	83	9	30		50
2	0	∞	66	37	17		26
3	29	1	∞	19	0		5
4							
5	3	21	56	7	∞		28
6	0	85	8	∞	89		0
7	18	0	0	0	58		∞

- **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

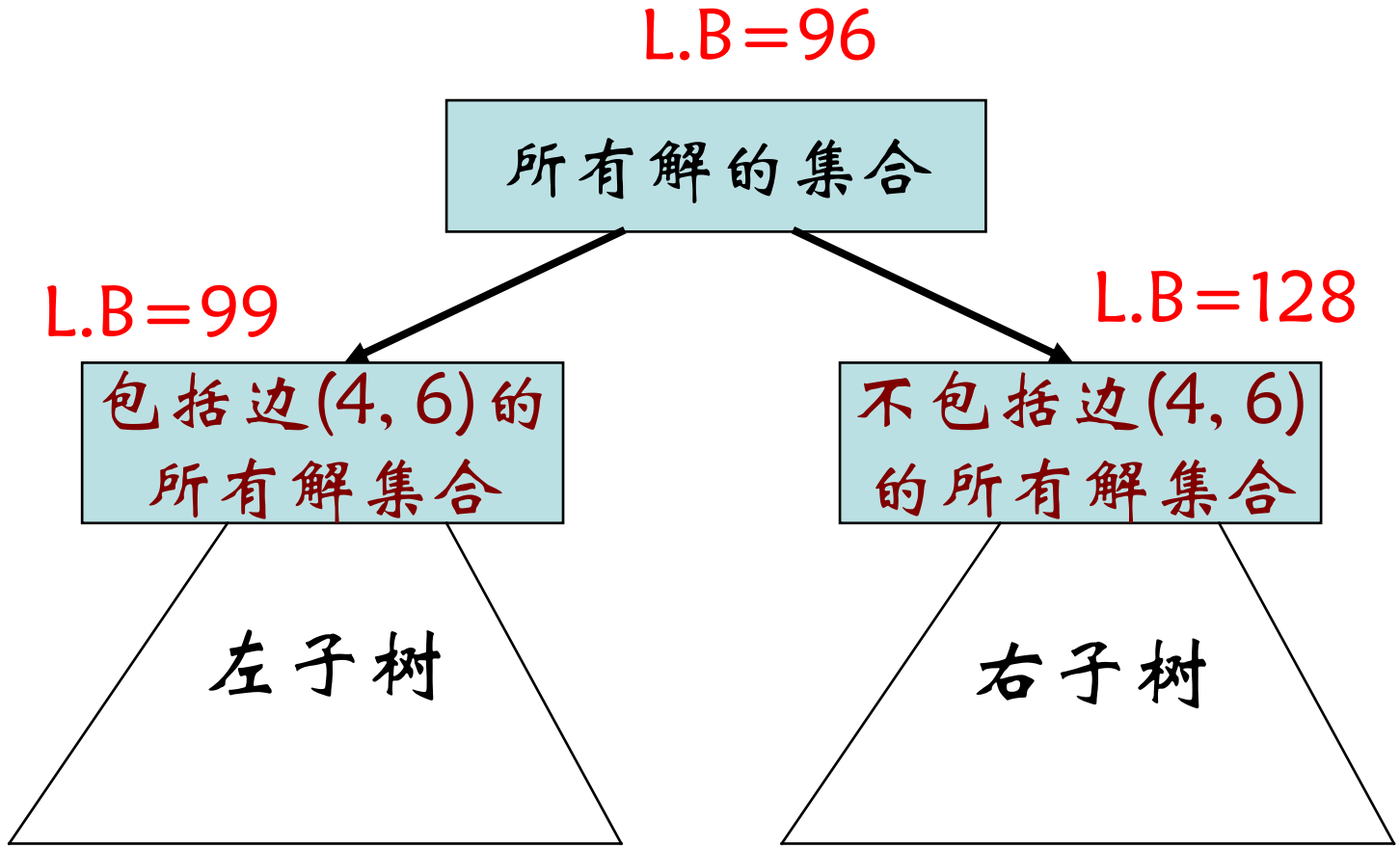
	$j=1$	2	3	4	5		7	
$i=1$	∞	0	83	9	30		50	
2	0	∞	66	37	17		26	
3	29	1	∞	19	0		5	
4								
5	0	18	53	4	∞		25	- 3
6	0	85	8	∞	89		0	
7	18	0	0	0	58		∞	

➢ 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　右子树根代价下界为：**96+32=128**

	$j=1$	2	3	4	5	6	7	
$i=1$	∞	0	83	9	30	6	50	
2	0	∞	66	37	17	12	26	
3	29	1	∞	19	0	12	5	
4	0	51	34	∞	17	∞	48	-32
5	3	21	56	7	∞	0	28	
6	0	85	8	42	89	∞	0	
7	18	0	0	0	58	13	∞	

- 目前的树为

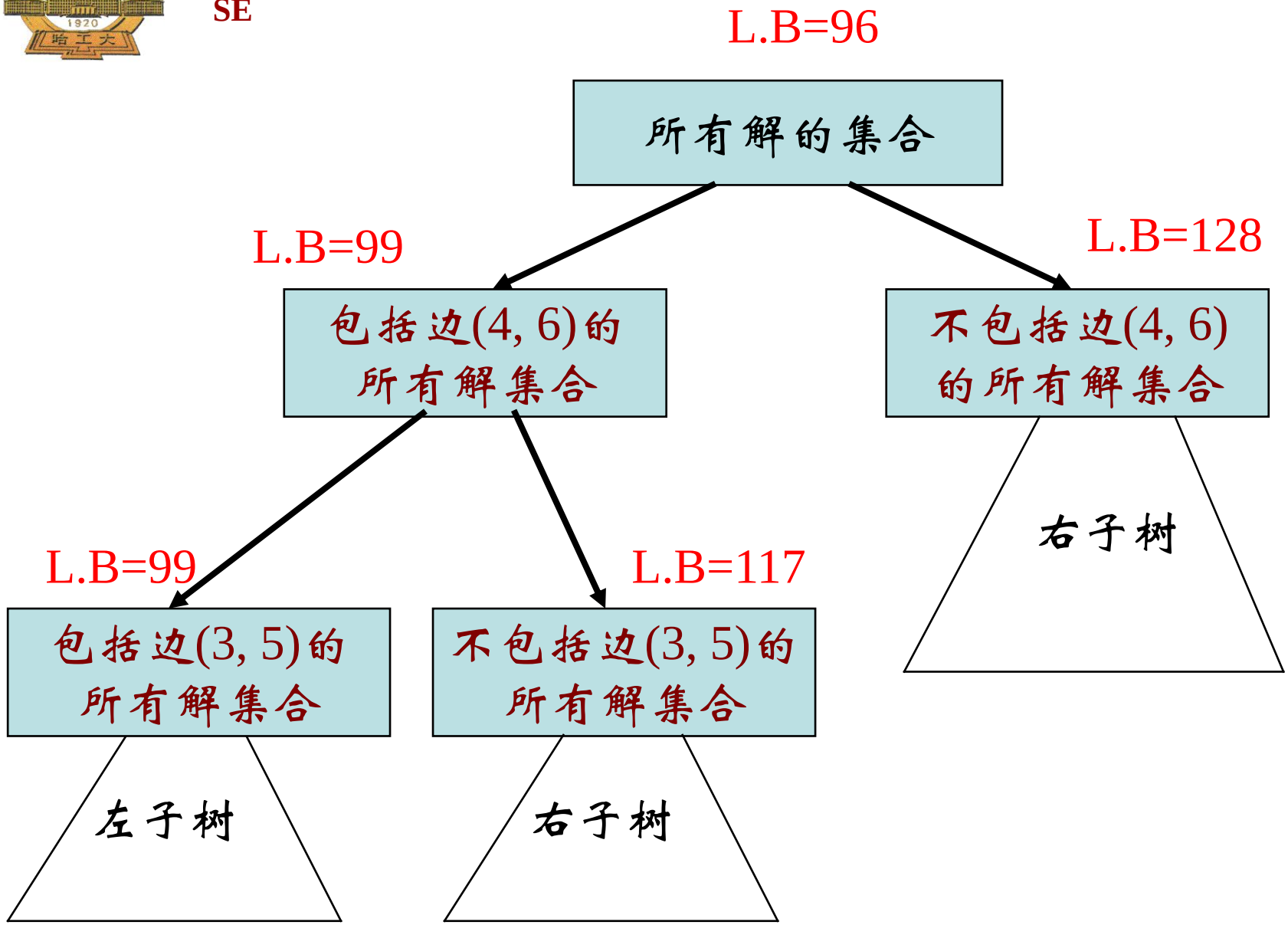


左子树根的代价矩阵

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

选哪条边?

- 使用爬山策略扩展左子树根(因为其代价下界小)
  - ✓ 选择(3, 5)作为划分边，因其右子结点代价的下界增加最大
  - ✓ 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(99+1+17)
- 目前树扩展为:


L.B=96
所有解的集合
L.B=99
包括边(4, 6)的
所有解集合
L.B=128
不包括边(4, 6)
的所有解集合
右子树
L.B=99
包括边(3, 5)的
所有解集合
L.B=117
不包括边(3, 5)的
所有解集合
左子树
右子树

HITWH
SE

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

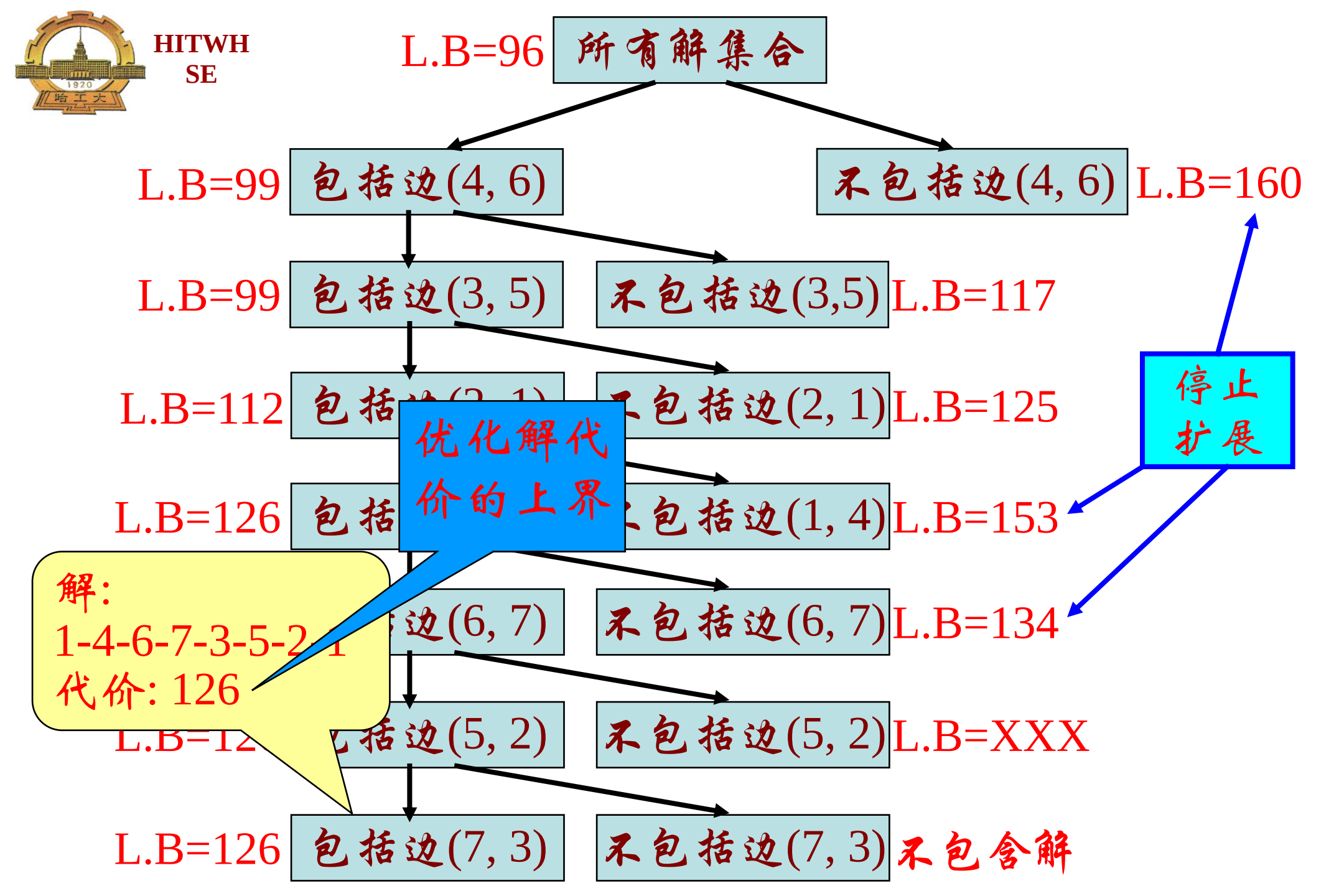
左子结点代价矩阵

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

右子结点代价矩阵

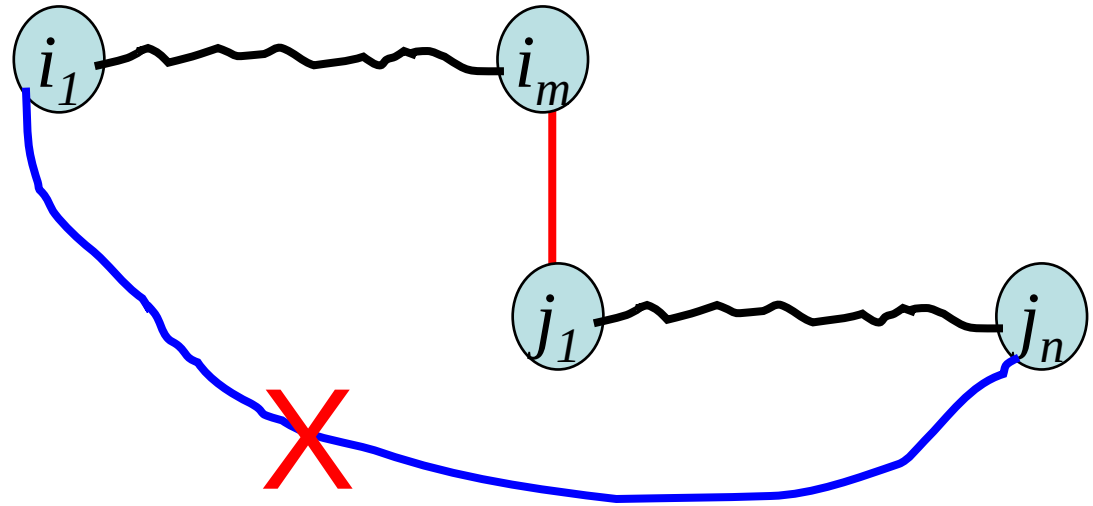
$$\begin{bmatrix} \infty & 0 & 83 & 9 & 13 & 50 \\ 0 & \infty & 66 & 37 & 0 & 26 \\ 28 & 0 & \infty & 18 & \infty & 4 \\ 0 & 18 & 53 & 4 & \infty & 25 \\ 0 & 85 & 8 & \infty & 72 & 0 \\ 18 & 0 & 0 & 0 & 41 & \infty \end{bmatrix}$$

L.B=99+1+17=117


HITWH
SE
L.B=96
所有解集合
L.B=99
包括边(4, 6)
不包括边(4, 6)
L.B=160
L.B=99
包括边(3, 5)
不包括边(3,5)
L.B=117
L.B=112
不包括边(2, 1)
L.B=125
L.B=126
不包括边(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
优化解代价的上界
停止扩展
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.

HITWH
SE

7.5 0/1 Knapsack

- 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

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：$C>0, w_i>0, v_i>0, 1\le i\le n$

输出：$(x_1, x_2, \ldots, x_n), x_i\in\{0, 1\}$，满足：

(1) $\Sigma_{1\le i\le n} w_i x_i \le C$,

(2) $\Sigma_{1\le i\le n} v_i x_i$ 最大.

转换为树搜索问题

- 按照“价值重量比”降序排列n个物品
- 空包为根；
- 用爬山法或Best-First递归地划分解空间，得到二叉树
  - 树中第i层的每个节点都代表了n个物品中所有符合以下特征的子集：
    - 每个子集对应于序列中前i个物品的一个特定选择
    - 这个特定选择是根据从根到该节点的一条路径确定的
      - 向左的分支表示包含下一个物品
      - 向右的分支表示不包含下一个物品
  - 每个节点中记录如下信息
    - 当前装入背包物品的总重量w及总价值v
    - 此时背包能够容纳的物品价值上界UB

分支界限搜索算法

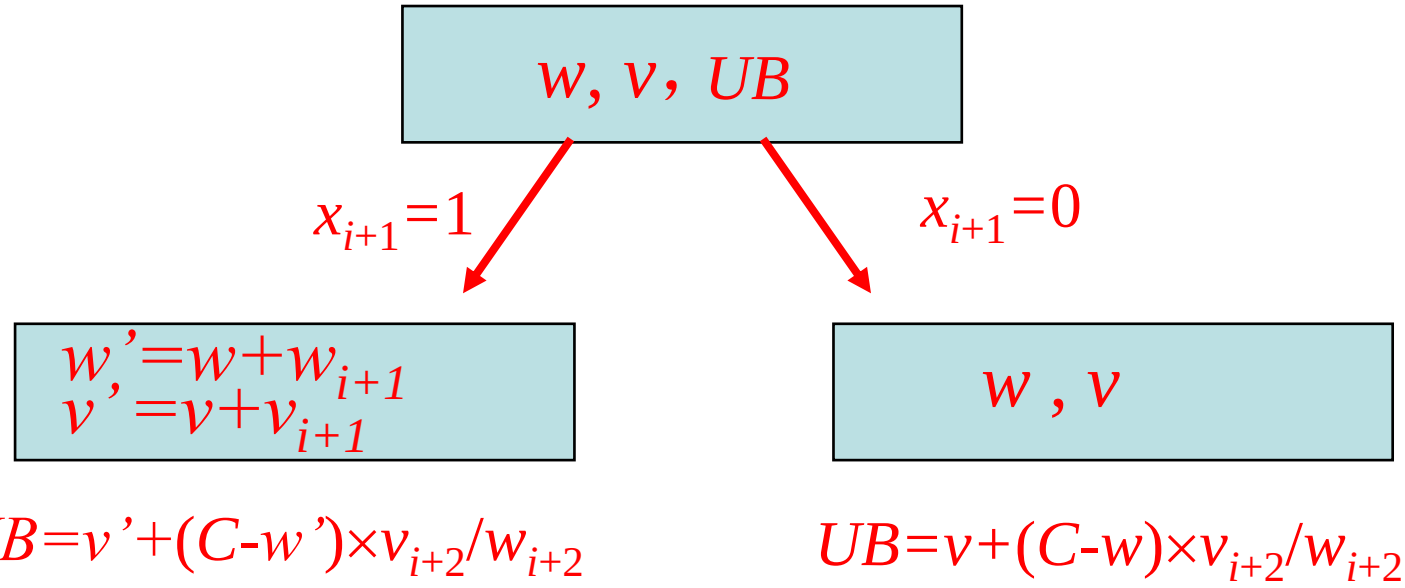
- 计算节点代价的上界*UB*：
  - $UB=v+(C-w)\times(v_{i+1}/w_{i+1})$，或者
  - $UB=v+UB'$
    - *UB'*是部分背包算法在子问题($\{i+1, i+2,\ldots,n\}$, $C-w$)上的最优解代价
- 0/1背包问题的优化解与部分背包问题优化解之间的关系？
  - 0/1背包问题的优化解是部分背包问题的可行解
  - 部分背包问题的优化解是0/1背包问题优化解上界

- 根节点及其代价上界

$w=0,\ v=0$

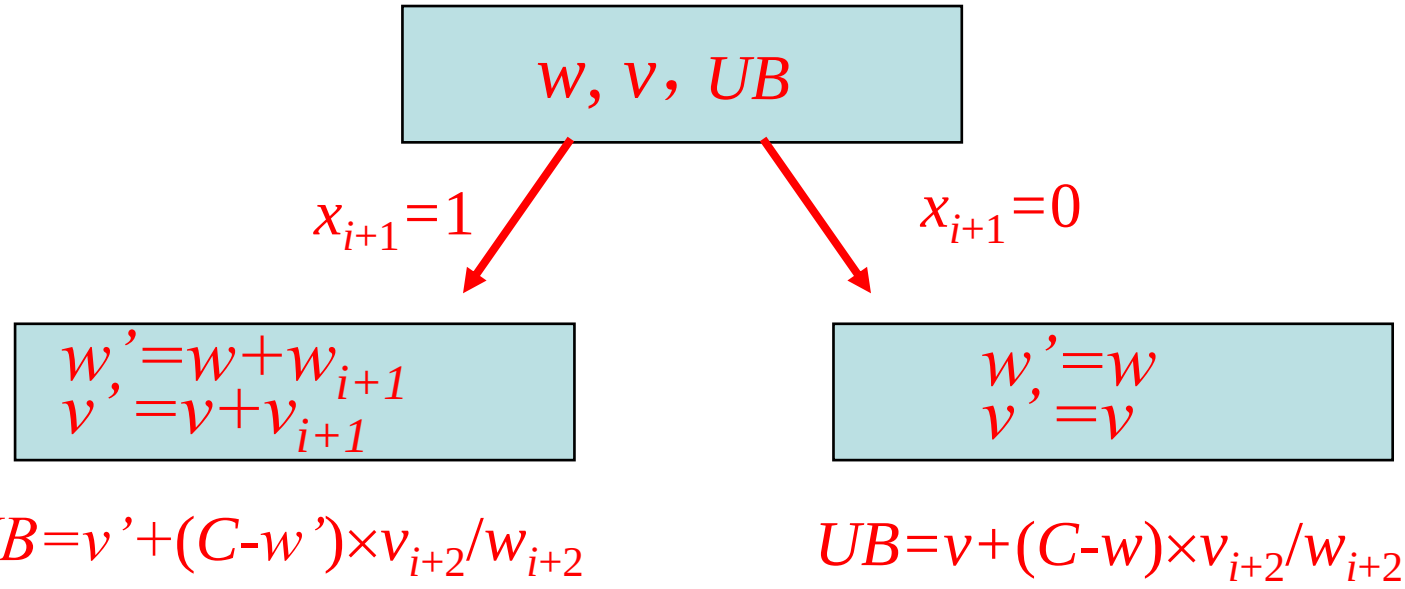
$$UB=v+(C-w)\times v_1/w_1$$
$$= C\times v_1/w_1$$

- 第i层节点的搜索及其代价上界



根据左右儿子节点的代价的上界，确定下一次待搜索的节点

- 第i层节点的搜索及其代价上界

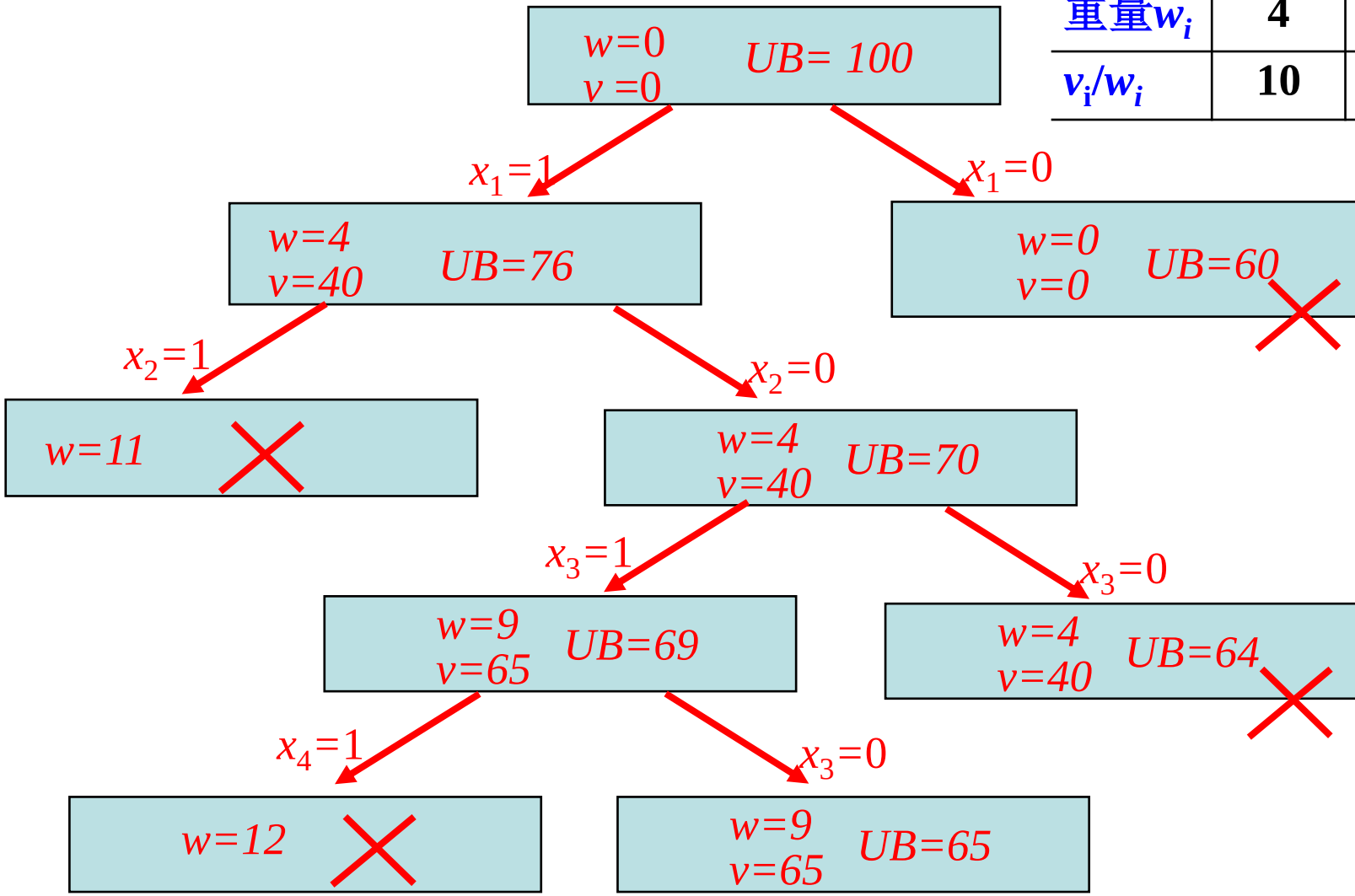


- 对于i+1层节点：
  - 若$w'>C$,停止搜索，由根到该节点对应路径不是可行解一部分
  - 若其UB小于当前优化解下界，则由根到该节点对应的路径一定不会产生最优解，停止搜索
  - 否则，优先选择UB较大的节点，继续爬山法搜索

• 举例

C=10

编号i	1	2	3	4
价值v_i	40	42	25	12
重量w_i	4	7	5	3
v_i/w_i	10	6	5	4



最优解！

HITWH
SE

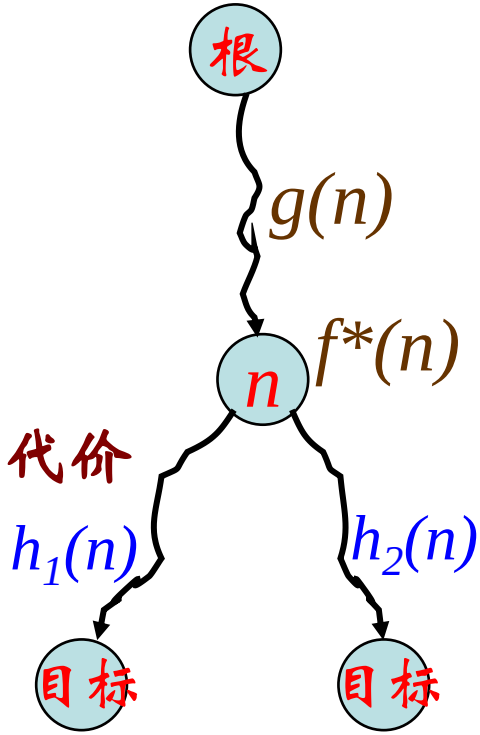
7.6 The A* Algorithm

- 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

- 基本思想
  - A*算法使用Best-first策略进行搜索
  - 在某些情况下，我们一旦得到了一个解，它一定是优化解，于是算法可以停止
  - 无需搜索整个解空间
- 与分支界限策略的不同
  - 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”

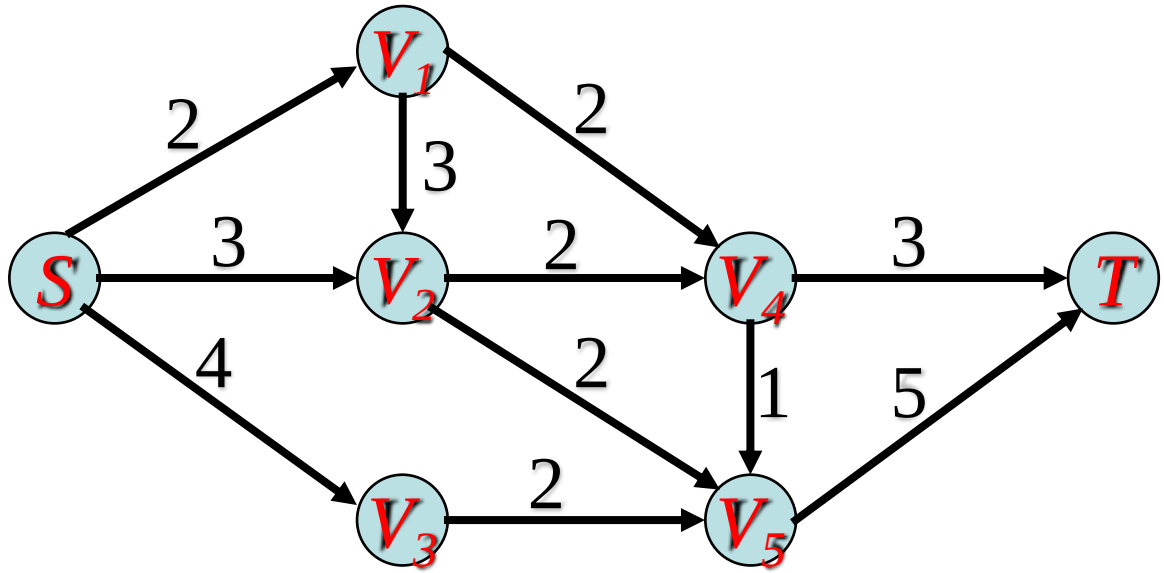
- A*算法关键—代价函数
  - 对于任意节点n
    - $g(n)$ = 从树根到n的代价
    - $h^*(n)$ = 从n到目标结点的优化路径的代价
    - $f^*(n) = g(n) + h^*(n)$是经过结点n到达目标结点的代价
  - $h^*(n)$ = ?
    - 不知道!
    - 于是, $f^*(n)$也不知道
  - 估计$h^*(n)$
    - 使用任何方法去估计$h^*(n)$, 用$h(n)$表示$h^*(n)$的估计
    - $h(n) \le h^*(n)$总为真
    - $f(n)=g(n)+h(n) \le g(n)+h^*(n)=f^*(n)$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



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

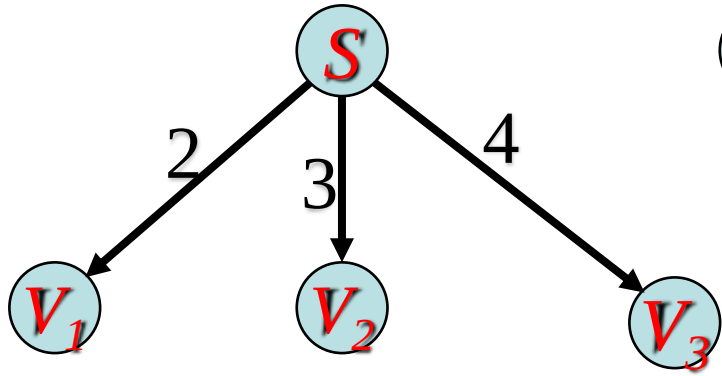
例1. 最短路径问题:

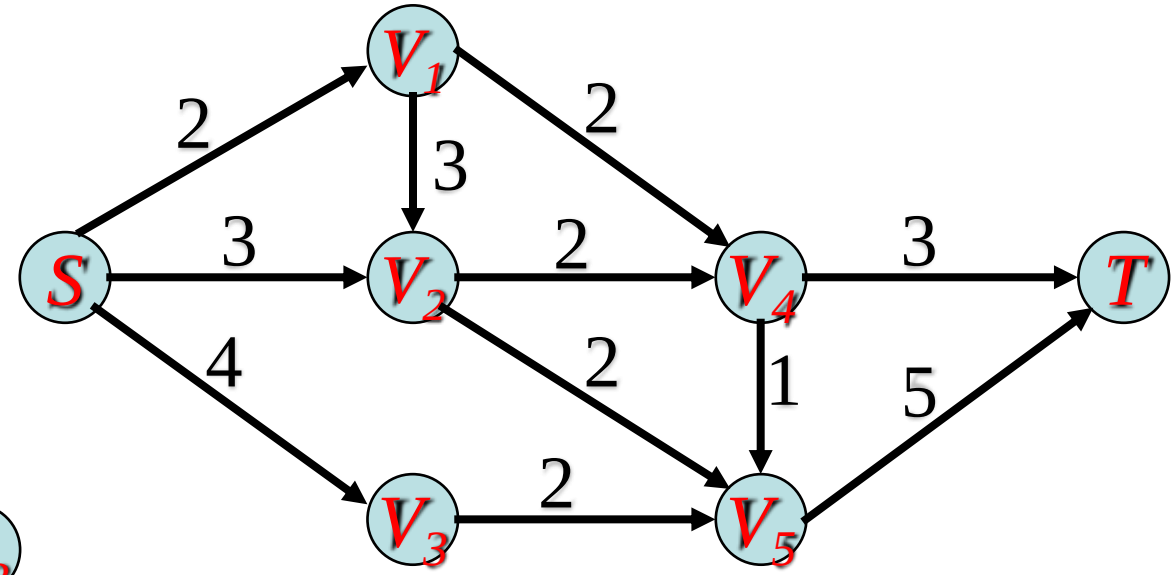
- 输入:



- 输出: 发现一个从S到T的最短路径

– 求解树的第一阶段





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

~~$h^*(V_1)=5,\ f^*(V_1)=g(V_1)+h^*(V_1)=7$~~

– 估计$h^*(V_1)$

- 从V_1出发有两种可能：代价分别为2和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

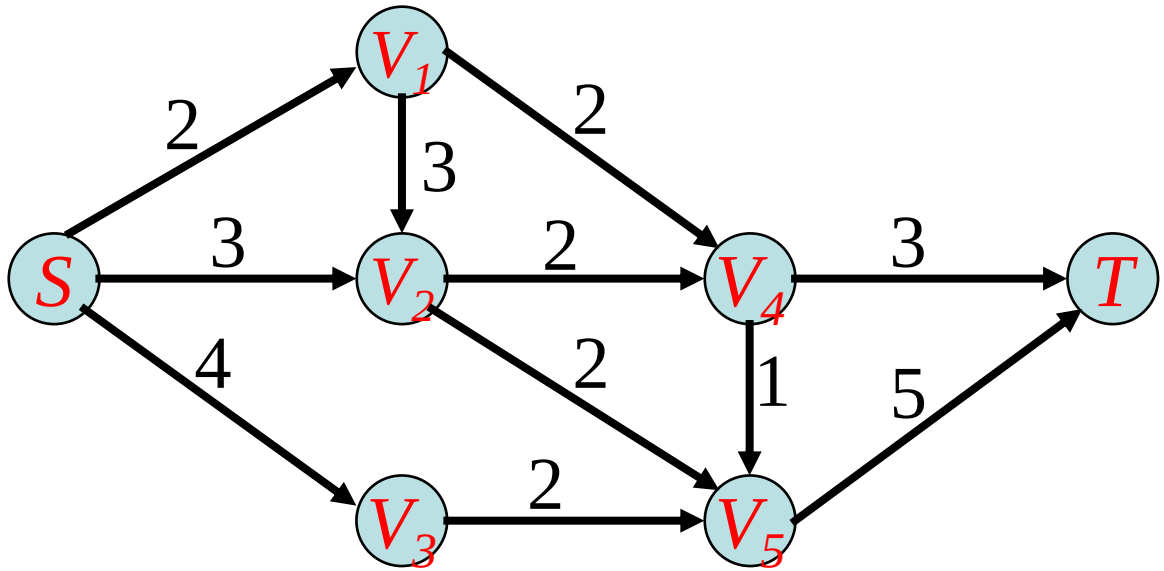
(1). 使用Best-first搜索策略;

(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, $h(n)\leq h^*(n)$;

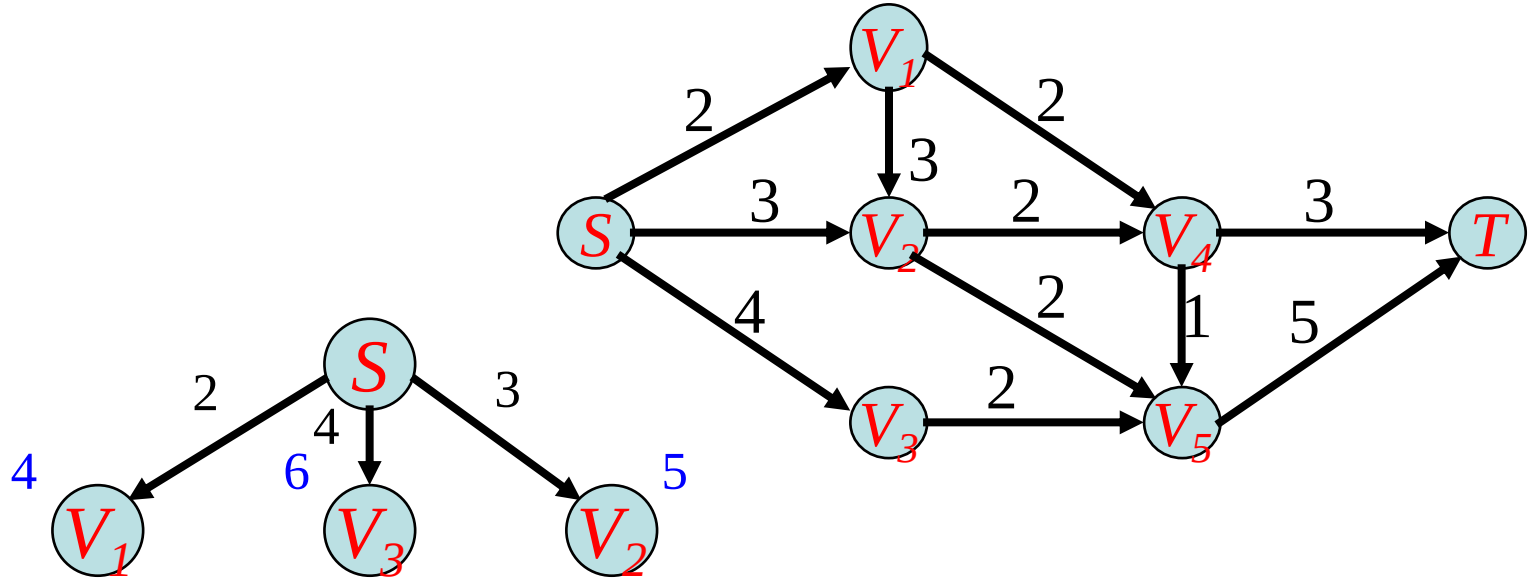
(3). 当选择到的结点是目标结点时, 算法停止, 该目标结点即为优化解.

应用A*算法求解最短路径问题

- 问题的输入:



Step 1

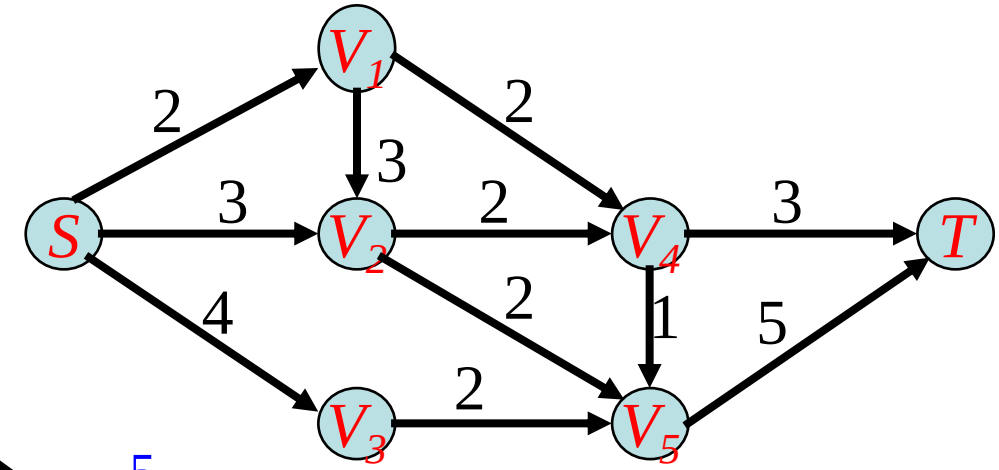


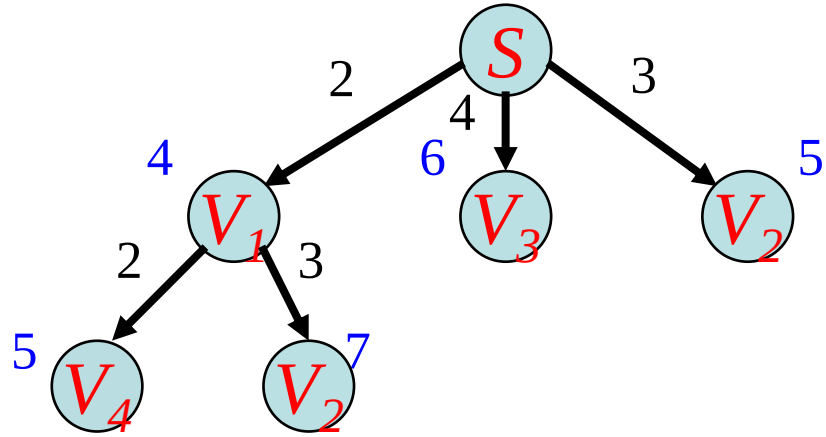
$g(V_1)=2$ $h(V_1)=min\{2,3\}=2$ $f(V_1)=2+2=4$

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

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

Step 2. 扩展V_1

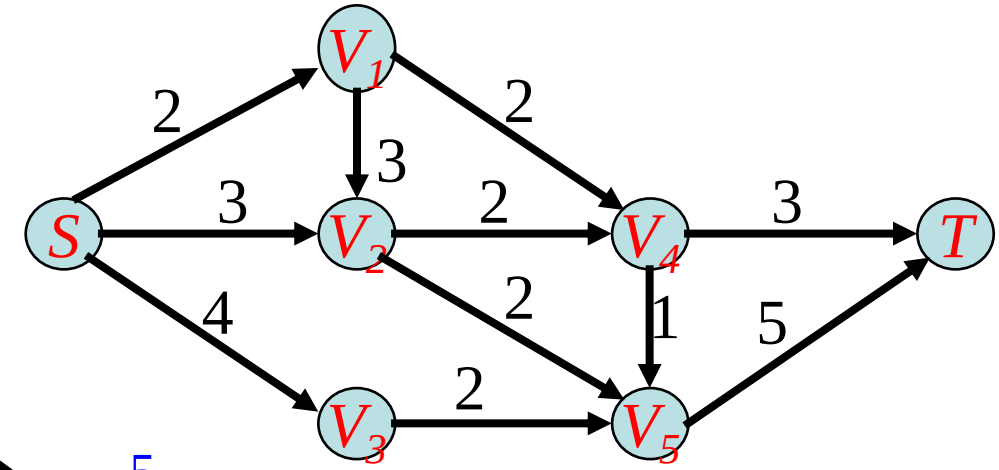


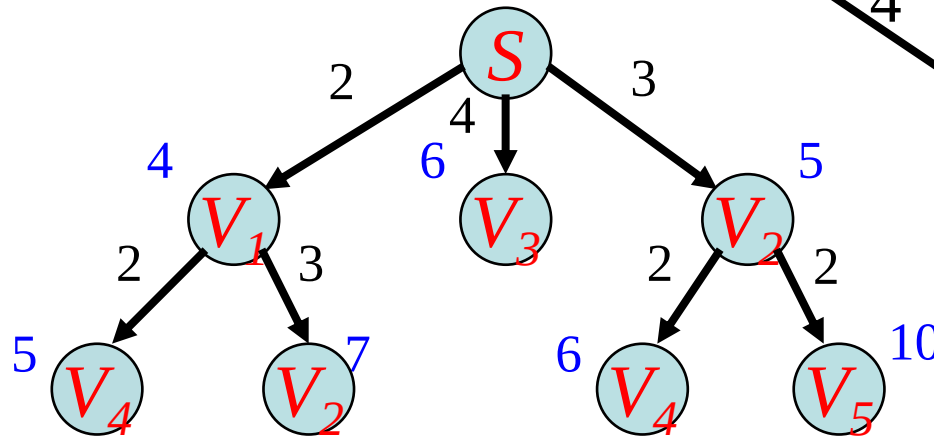


$g(V_4)=2+2=4$ $h(V_4)=min\{3,1\}=1$ $f(V_4)=4+1=5$

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

Step 3. 扩展V_2

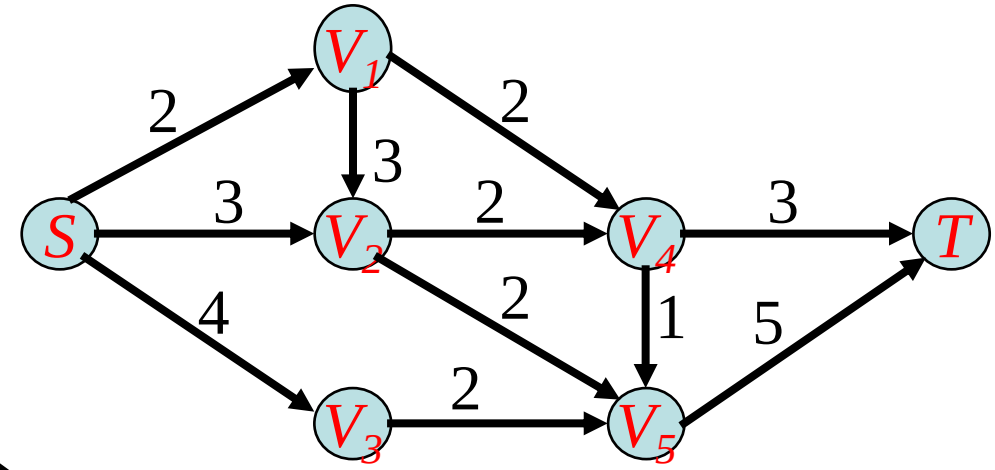


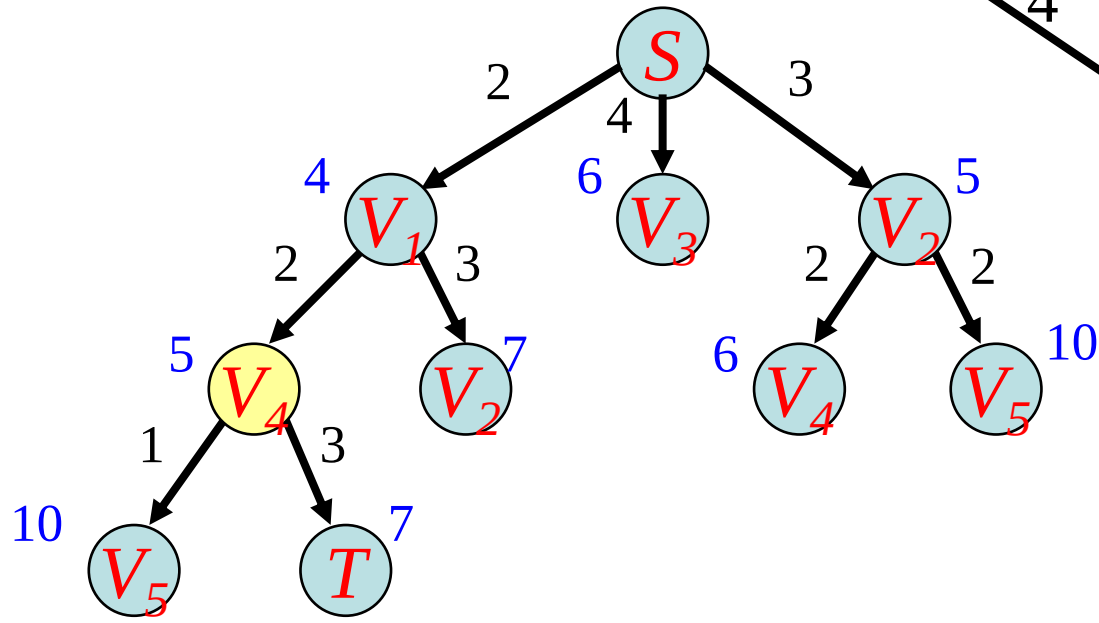


$g(V_4)=3+2=5$ $h(V_4)=min\{3,1\}=1$ $f(V_4)=5+1=6$

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

Step 4. 扩展V_4

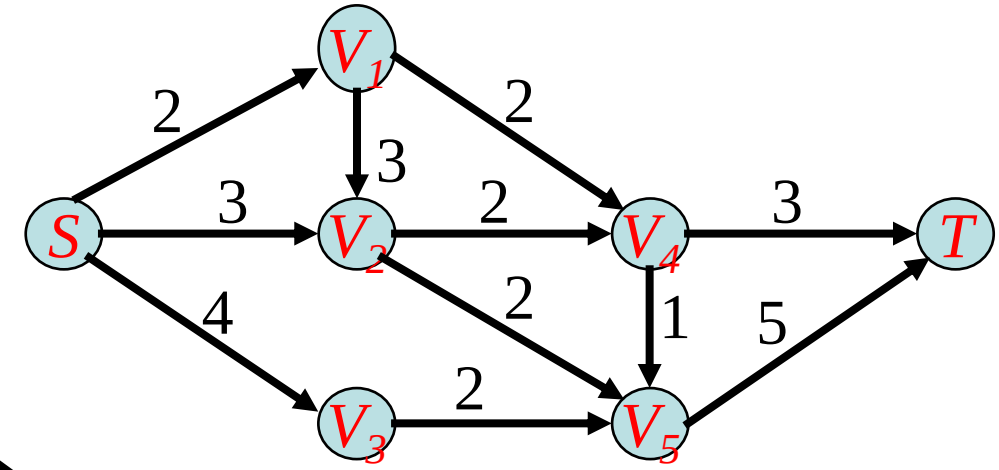


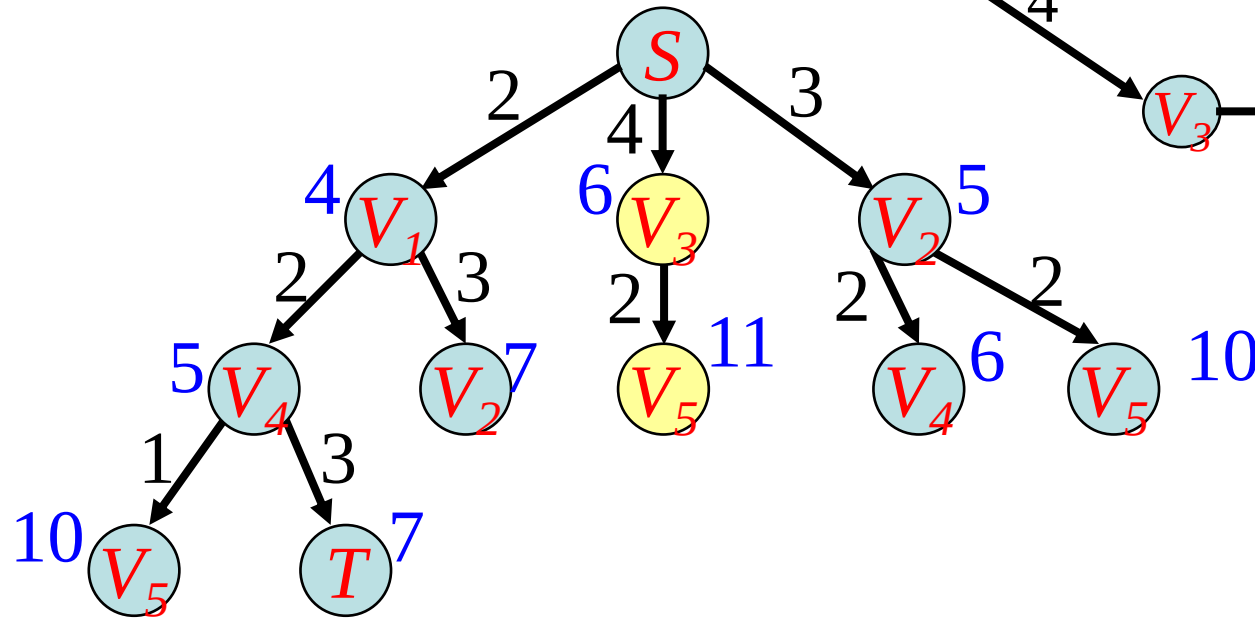


$g(V_5)=2+2+1=5$ $h(V_5)=min\{5\}=5$ $f(V_5)=5+5=10$

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

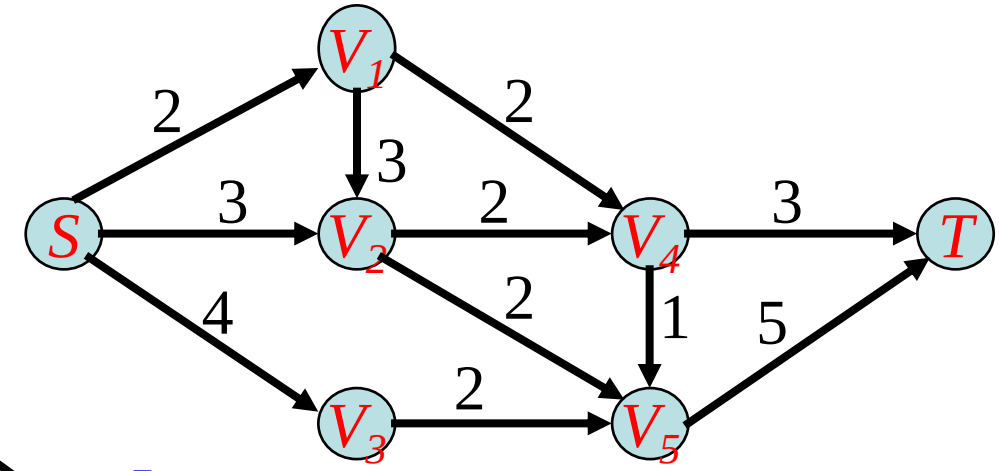
Step 5. 扩展V_3

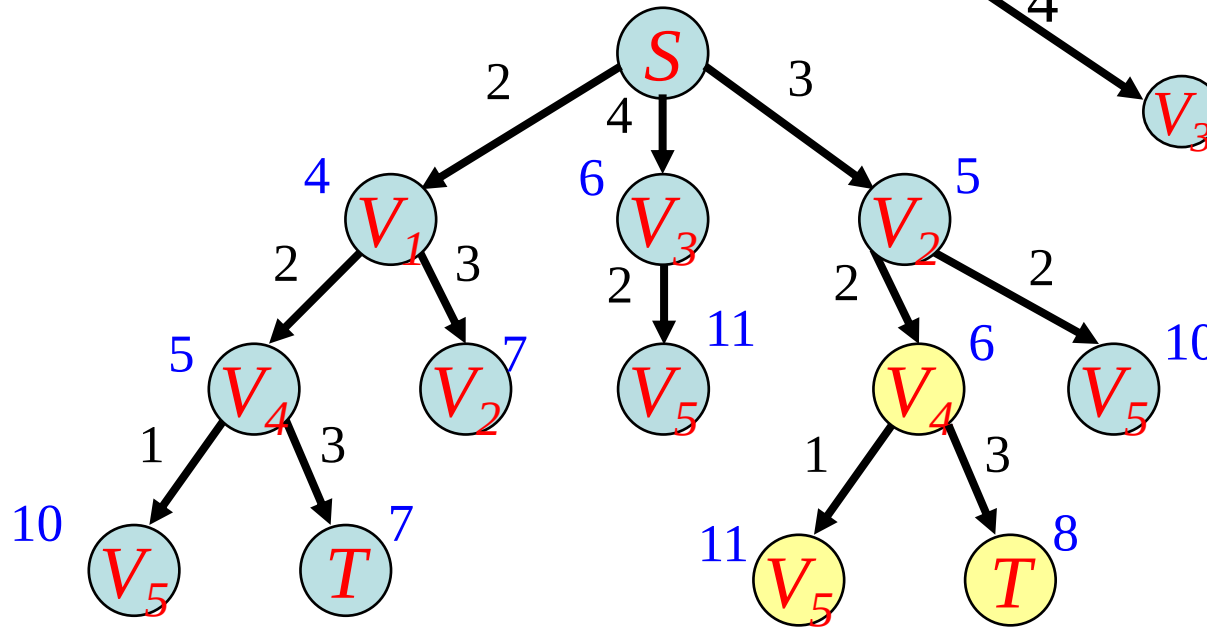




$g(V_5)=4+2=6$　　$h(V_5)=min\{5\}=5$　　$f(V_5)=6+5=11$

Step 6. 扩展V_4

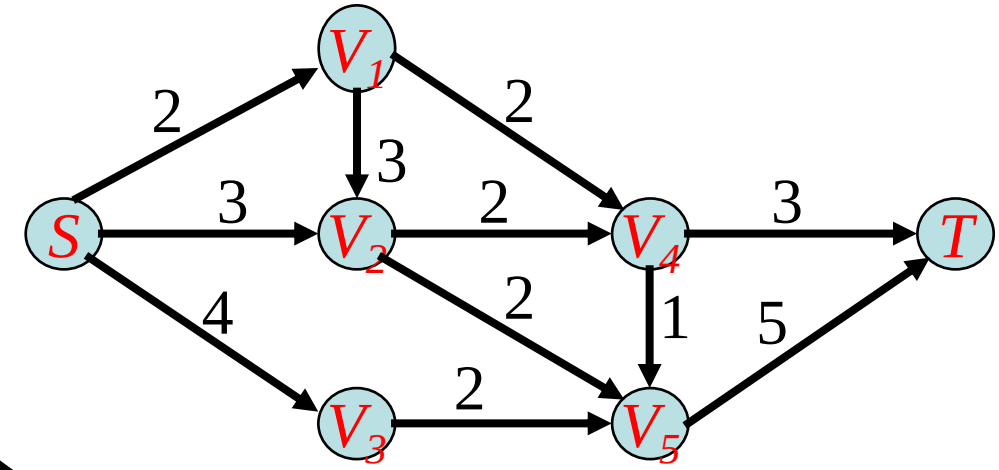


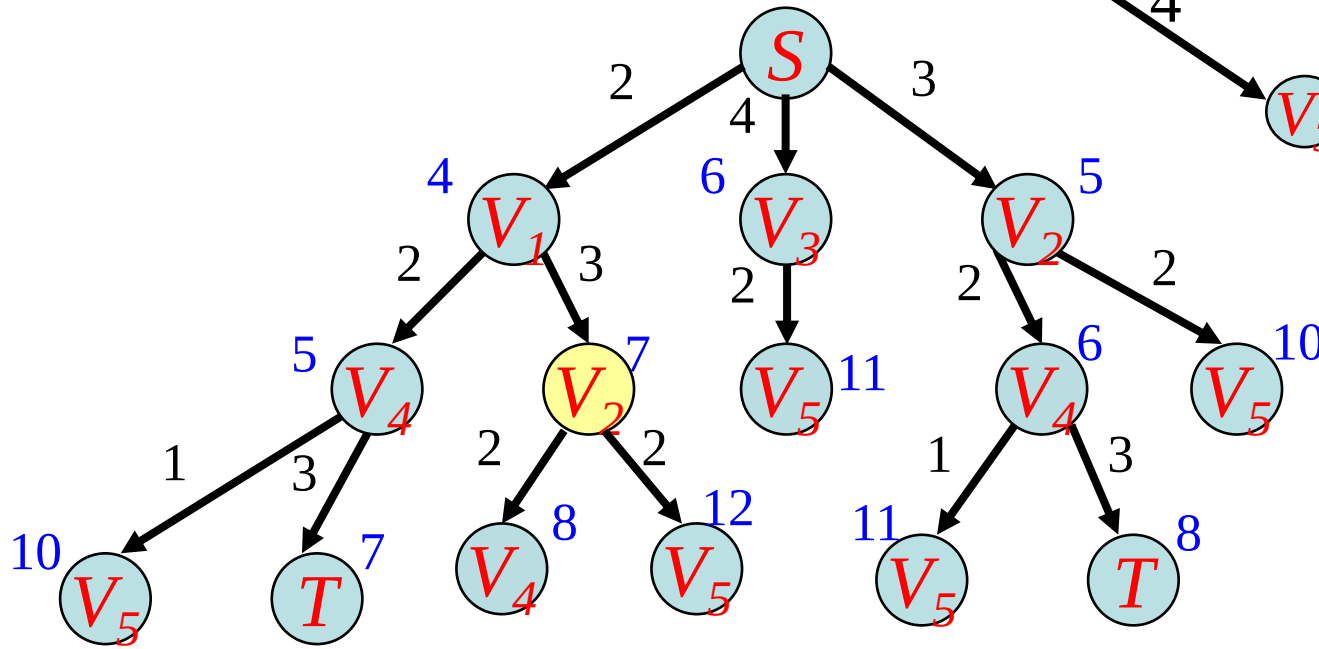


$g(V_5)=3+2+1=6$ $h(V_5)=min\{5\}=5$ $f(V_5)=6+5=11$

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

Step 7. 扩展V_2

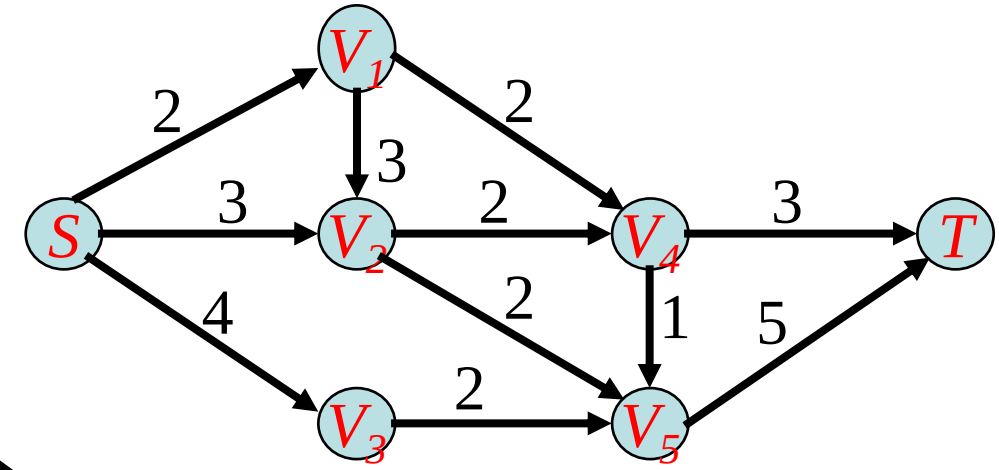


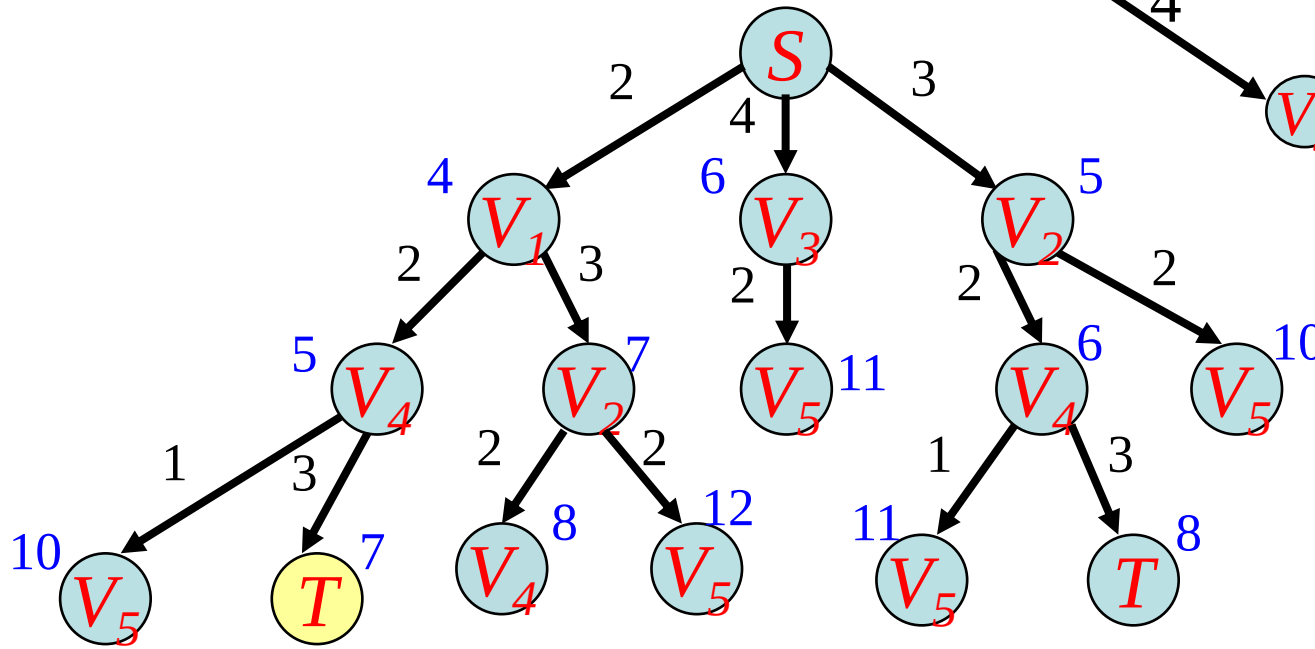


$g(V_4)=3+2+2=7$ $h(V_4)=min\{3,1\}=1$ $f(V_4)=7+1=8$

$g(V_5)=3+2+2=7$ $h(V_5)=min\{5\}=5$ $f(V_5)=7+5=12$

Step 8. 扩展T





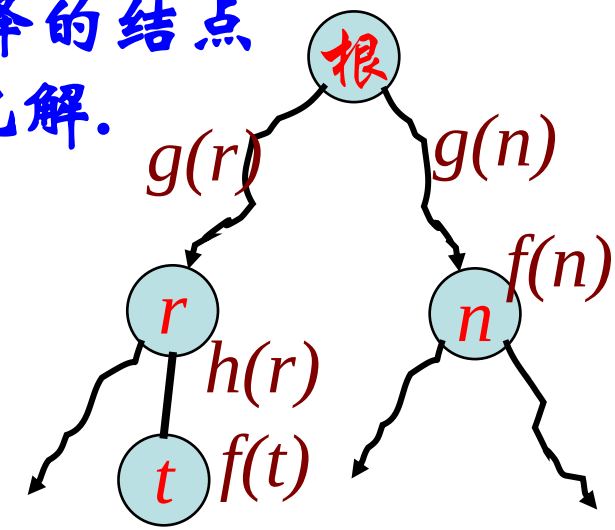
因为T是目标结点，所以我们得到解：

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

该解即为最优解

- 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

n是任意已扩展到的结点.

往证$f(t)=g(t)=g(r)+h(r)$是优化解代价.

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

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

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

($f^*(n)=g(n)+h^*(n)$是经过n的可能解的最小代价).

(4). t是目标节点，$h(t)=0$，所以$f(t)=g(t)+h(t)=f^*(t)$是t对应可能解的代价，$f(t)\ge f^*(s)$，即$f(t)=f^*(s)$.