

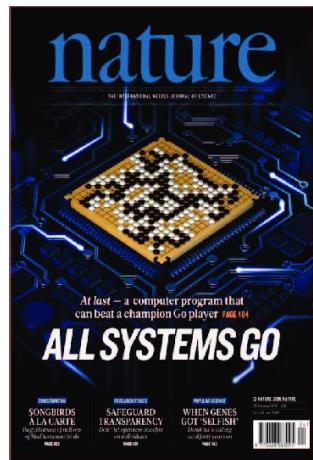
算法设计与分析

第六章 搜索策略

哈尔滨工业大学
李穆



搜索算法的应用



提纲

- 6.1 暴力美学：搜索漫谈**
- 6.2 深度优先与广度优先**
- 6.3 搜索的优化**
- 6.4 剪枝方法论与人员安排问题**
- 6.5 旅行商问题**
- 6.6 A*算法**



布尔表达式可满足性问题

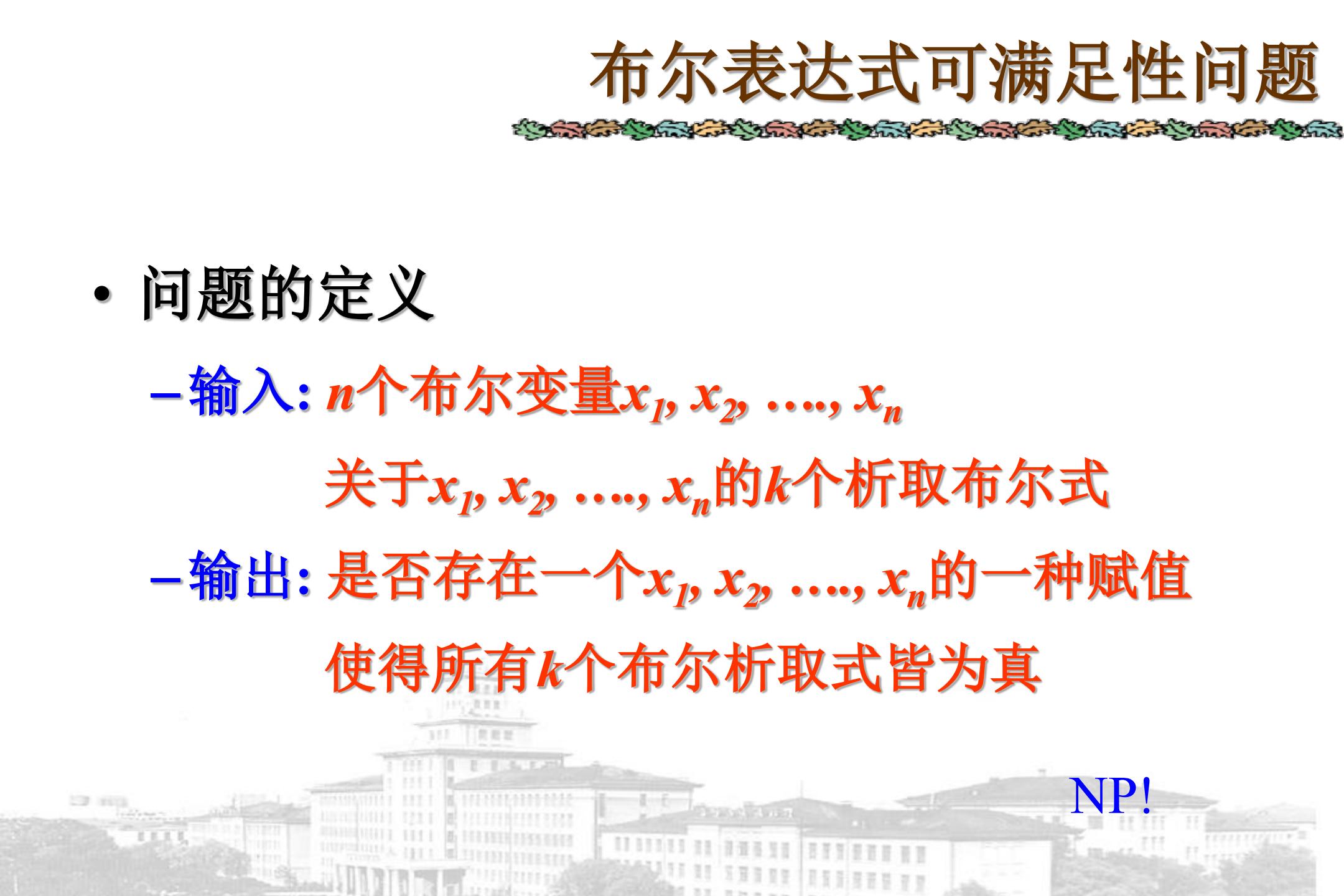
- 问题的定义

- 输入: n 个布尔变量 x_1, x_2, \dots, x_n

- 关于 x_1, x_2, \dots, x_n 的 k 个析取布尔式

- 输出: 是否存在一个 x_1, x_2, \dots, x_n 的一种赋值

- 使得所有 k 个布尔析取式皆为真

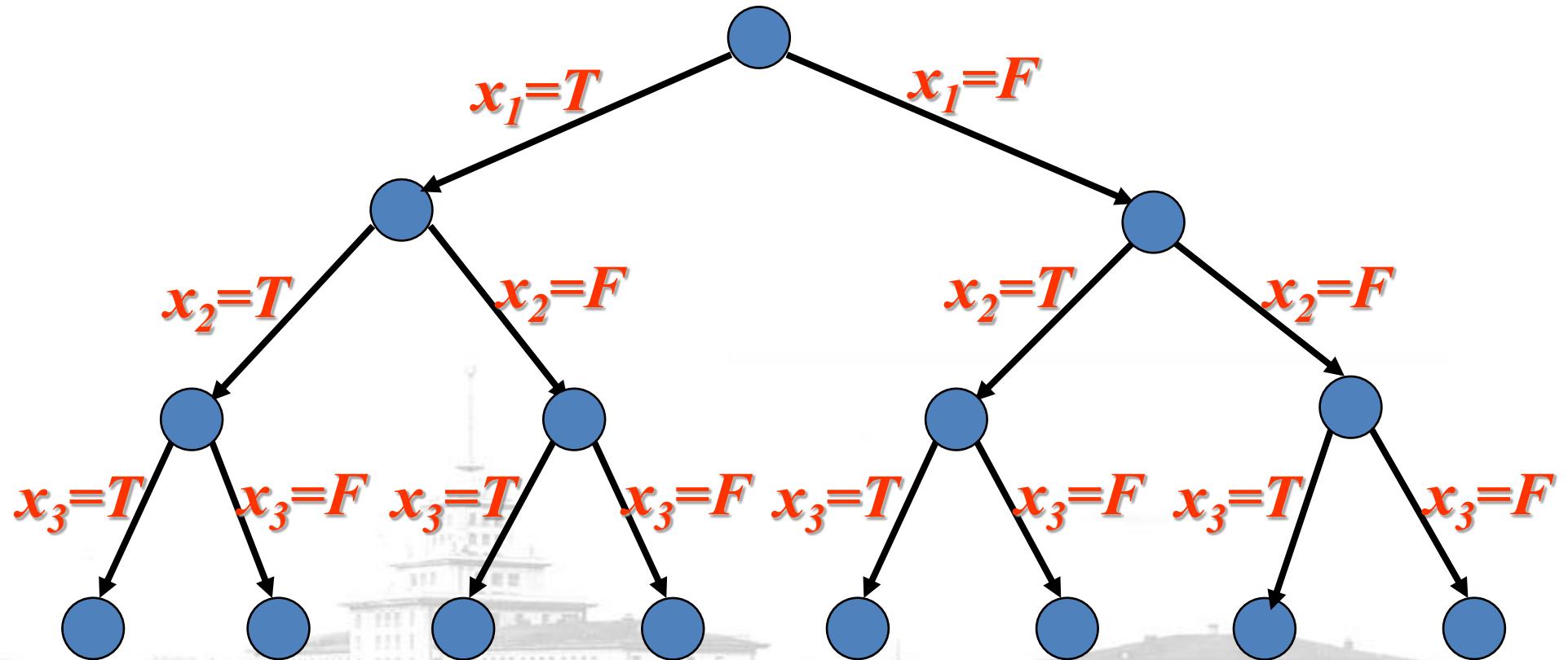
A faint, grayscale watermark-style image of a classical building with multiple levels and columns is visible in the background.

NP!

- 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)



8-Puzzle问题



- 问题的定义

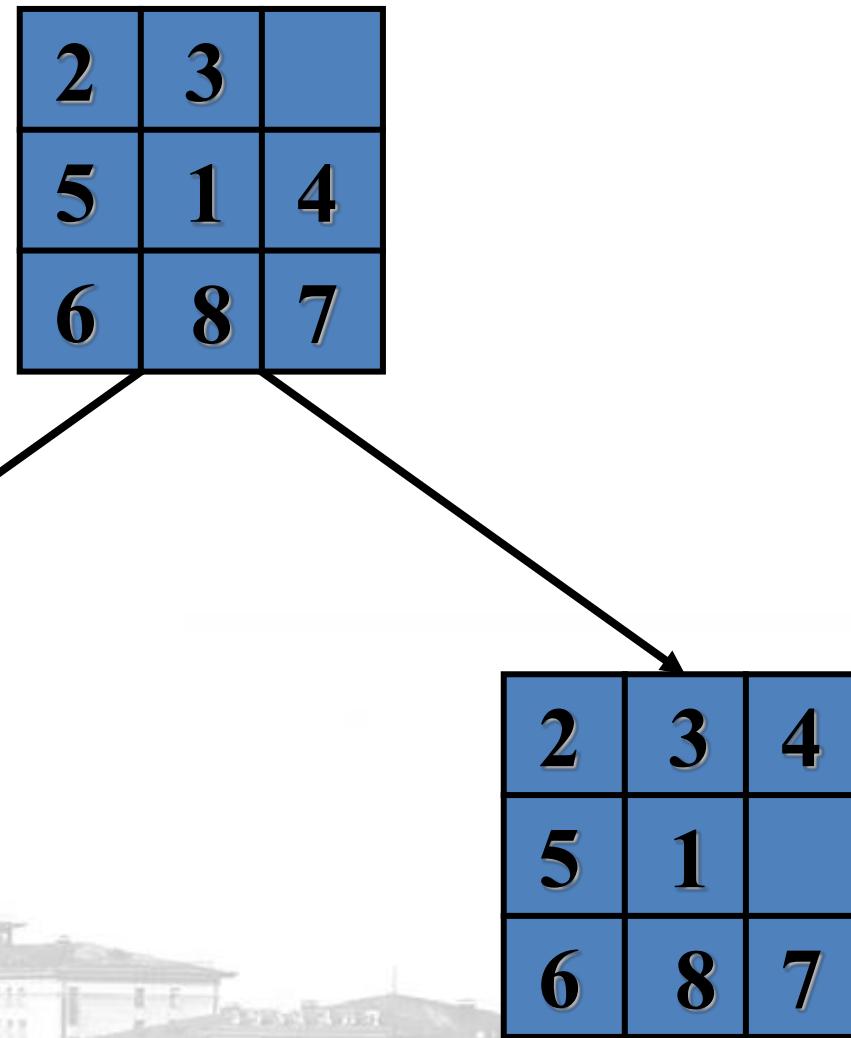
- 输入: 具有8个编号小方块的魔方

| | | |
|---|---|---|
| 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

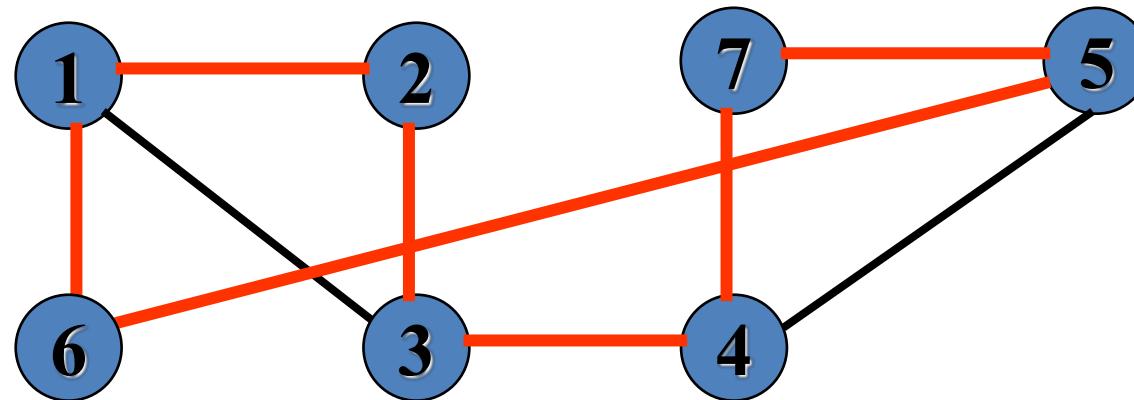
- 问题定义

- 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

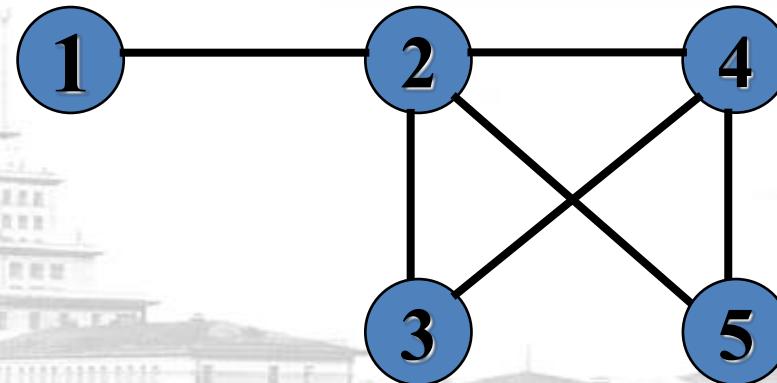
沿着 G 的*n*条边经过每个节点一次，并回到起始节点的环称为G的一个Hamiltonian环。

NP!

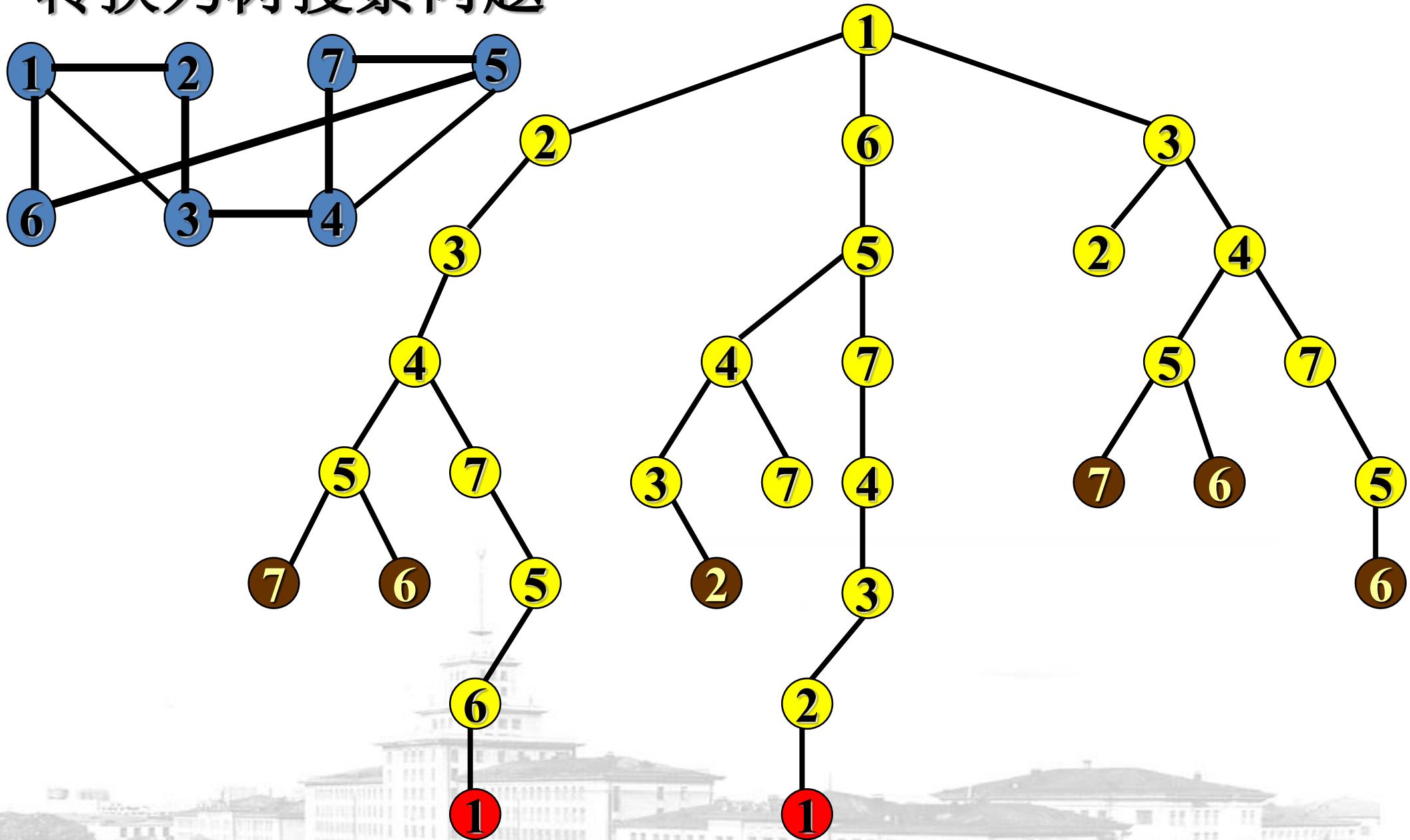
有Hamiltonian环图:

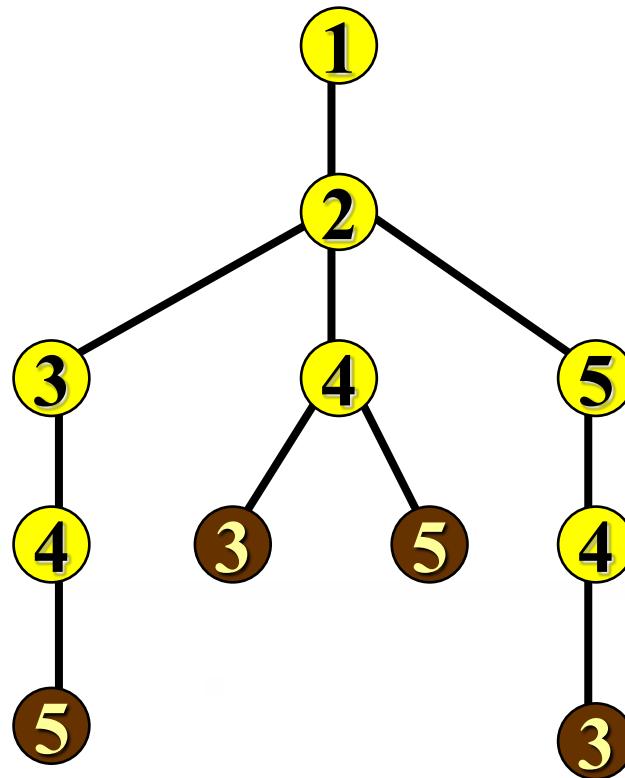
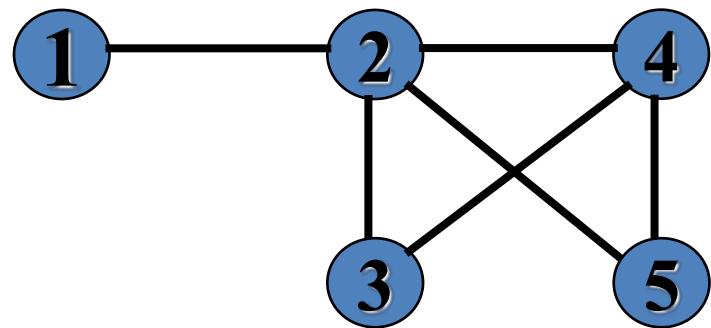


无Hamiltonian环图:



- 转换为树搜索问题

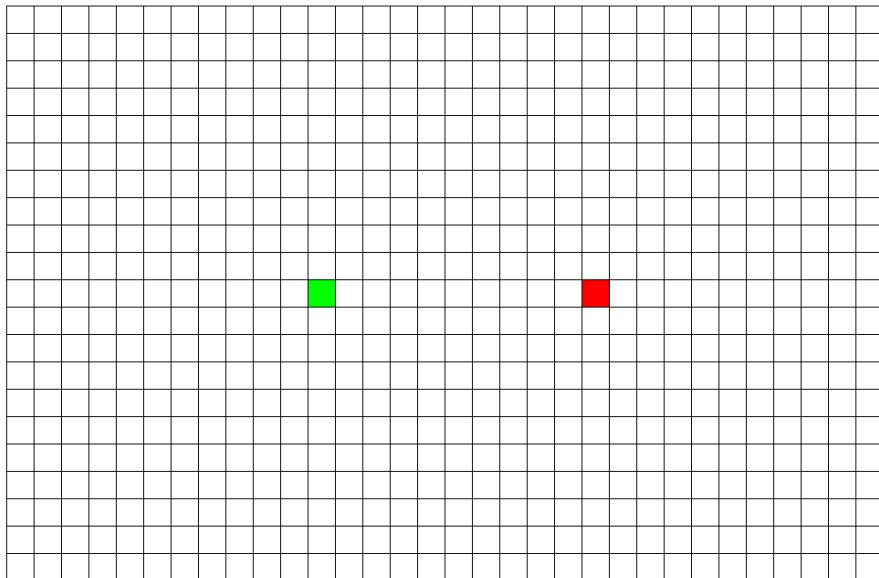




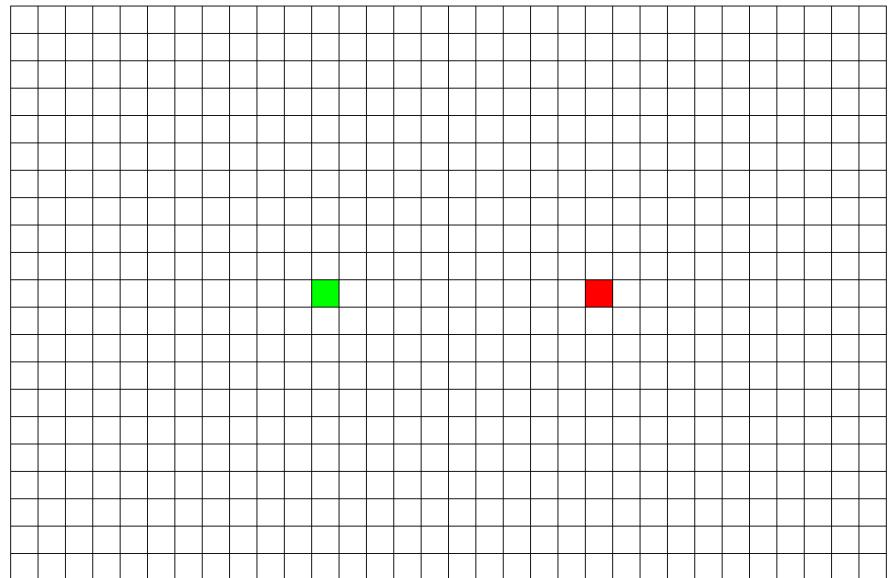
6.2 深度优先与广度优先

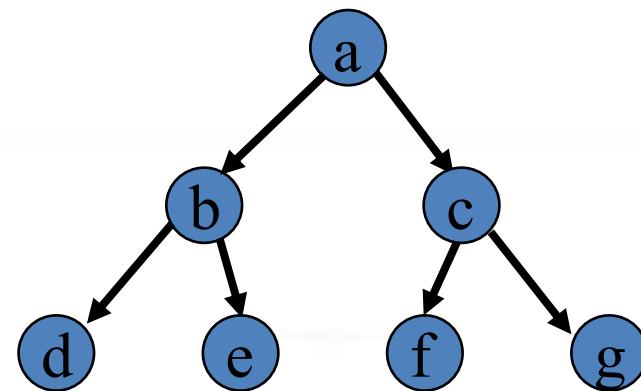
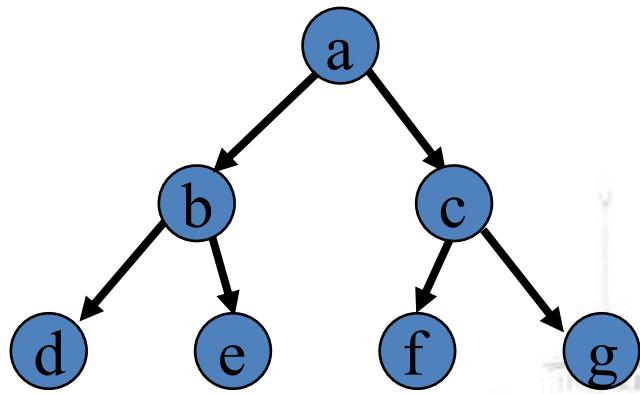
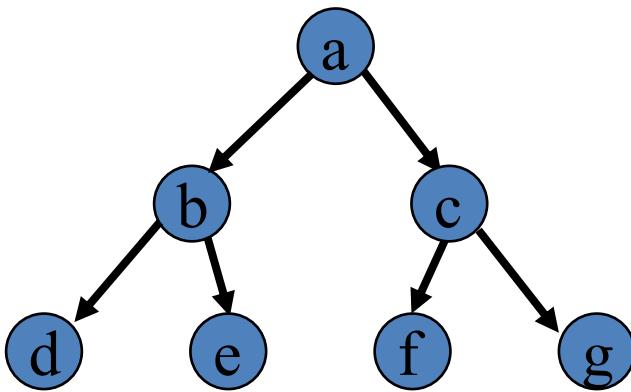


深度优先



广度优先





深度优先

广度优先

Breadth-First Search



- 算法

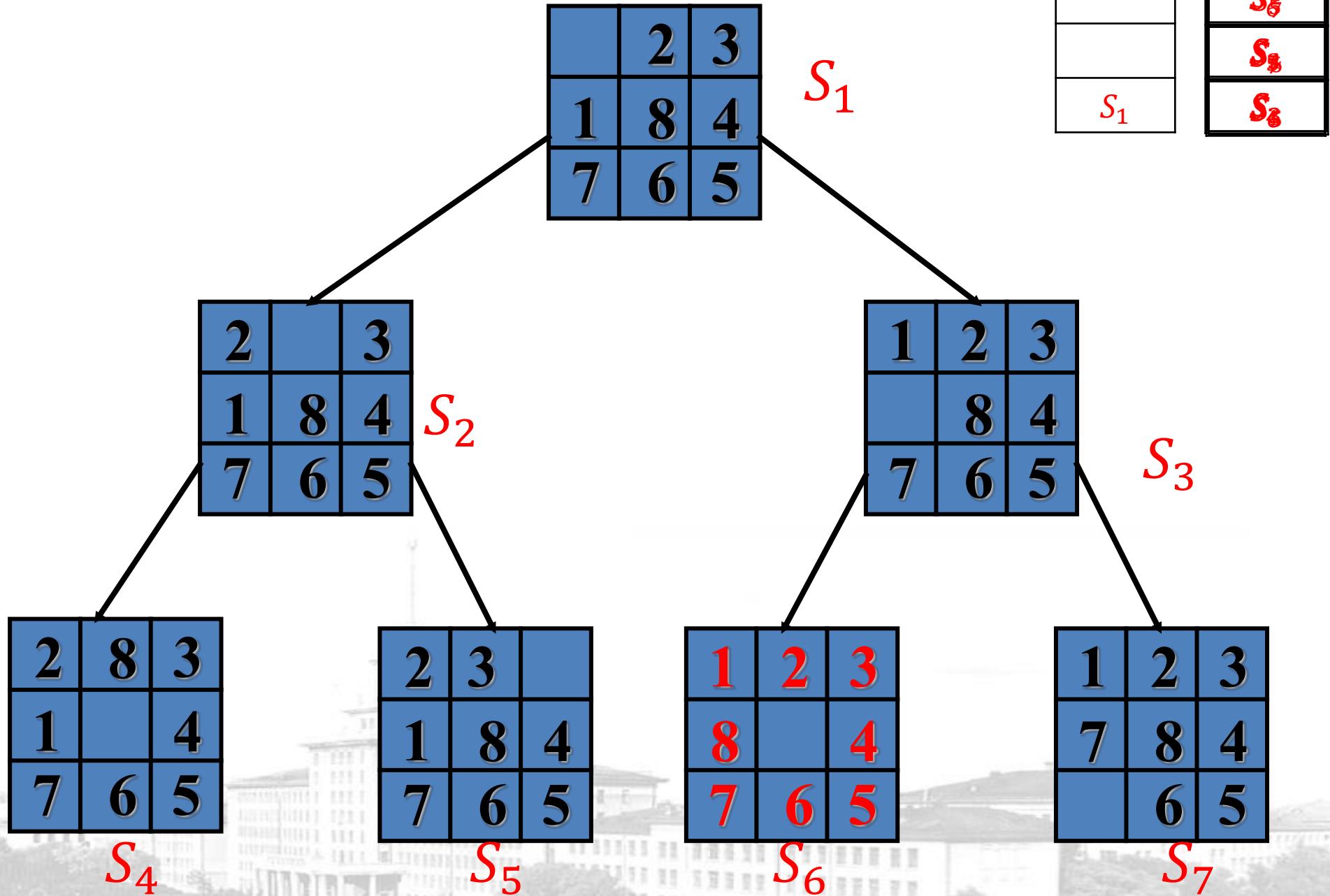
Breadth-First-Search

Input: 问题输入

Output: 问题输出

1. $Q \leftarrow \{root\}$ // 队列
2. While not $Q.empty()$ Do
3. $x \leftarrow Q.Dequeue()$
4. If x 是目标节点 Then Return x
5. For $v \in x$ 可以扩展的节点 Do
6. $Q.Enqueue(v)$
7. Return 无解

• 例: 求解8-Puzzle问题



Depth-First Search



- 算法

Depth-First-Search

Input: 问题输入

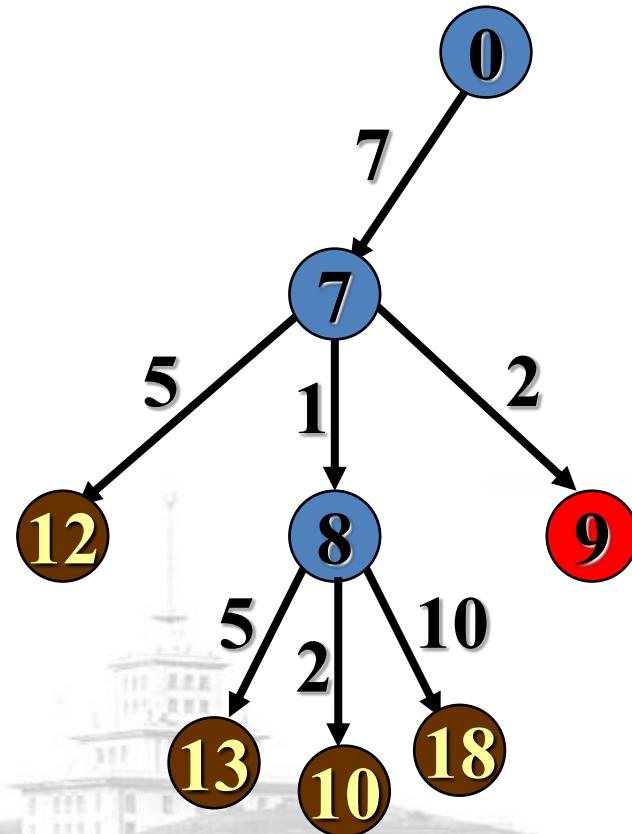
Output: 问题输出

1. $S \leftarrow \{root\}$ // 队列
2. While not $S.empty()$ Do
3. $x \leftarrow Q.pop()$
4. If x 是目标节点 Then Return x
5. For $v \in x$ 可以扩展的节点 Do
6. $Q.push(v)$
7. Return 无解

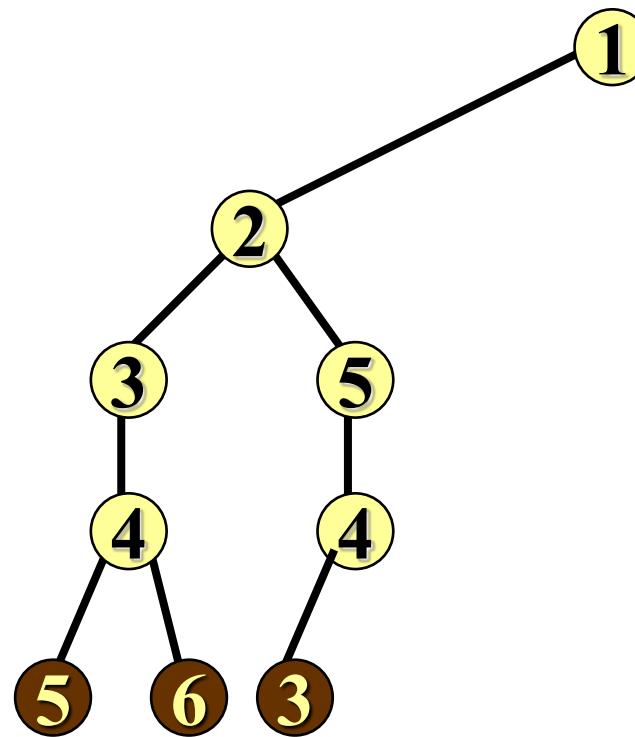
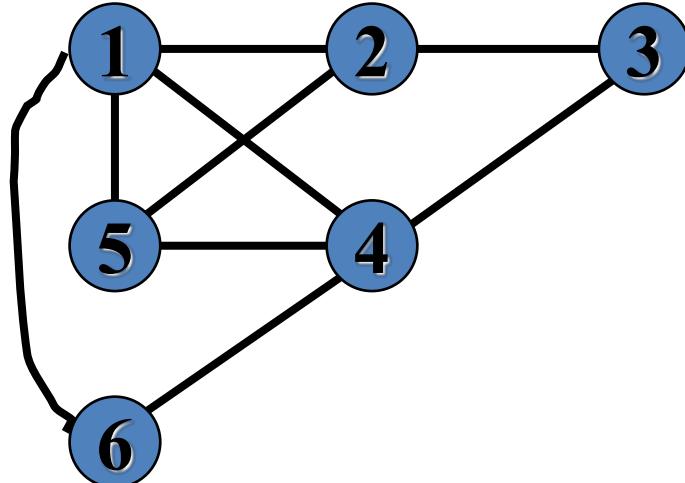
- 例1. 求解子集合和问题

输入: $S=\{7, 5, 1, 2, 10\}$

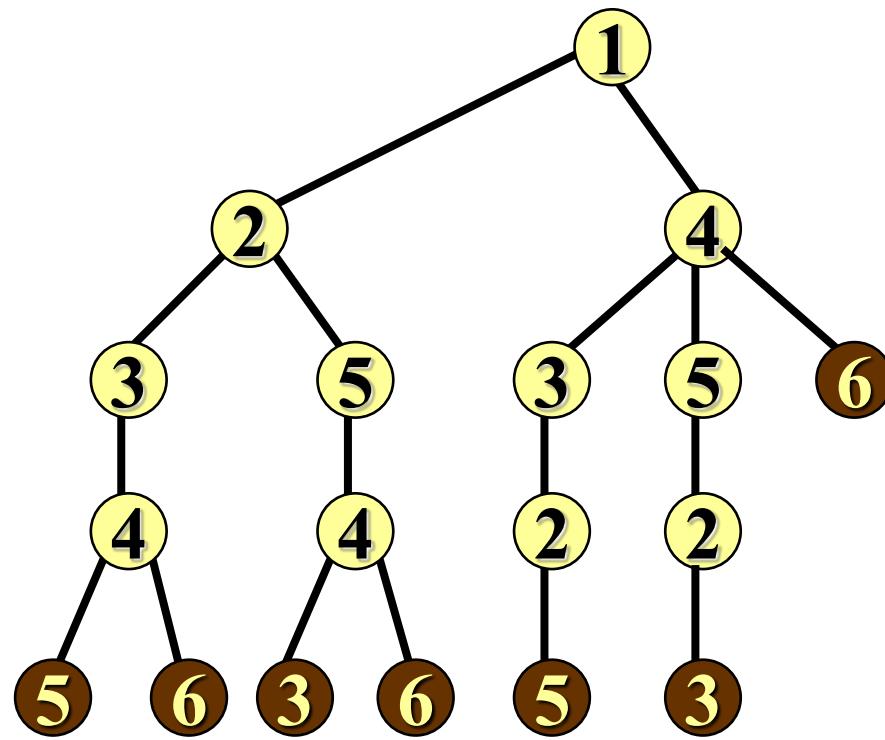
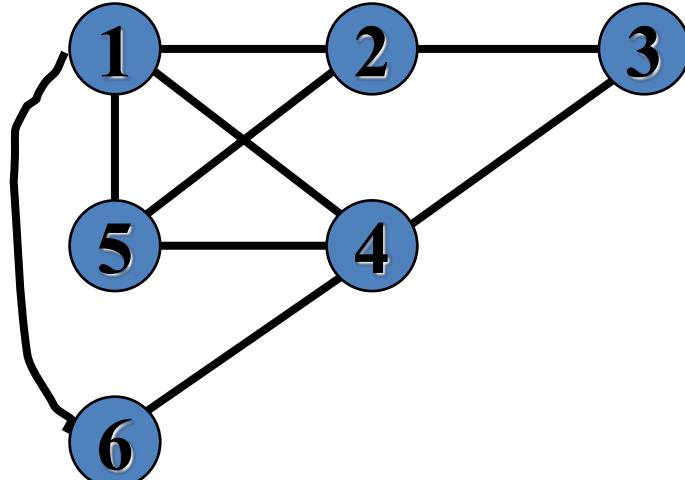
输出: 是否存在 $S' \subseteq S$, 使得 $\text{Sum}(S')=9$



- 例：求解Hamiltonian环问题

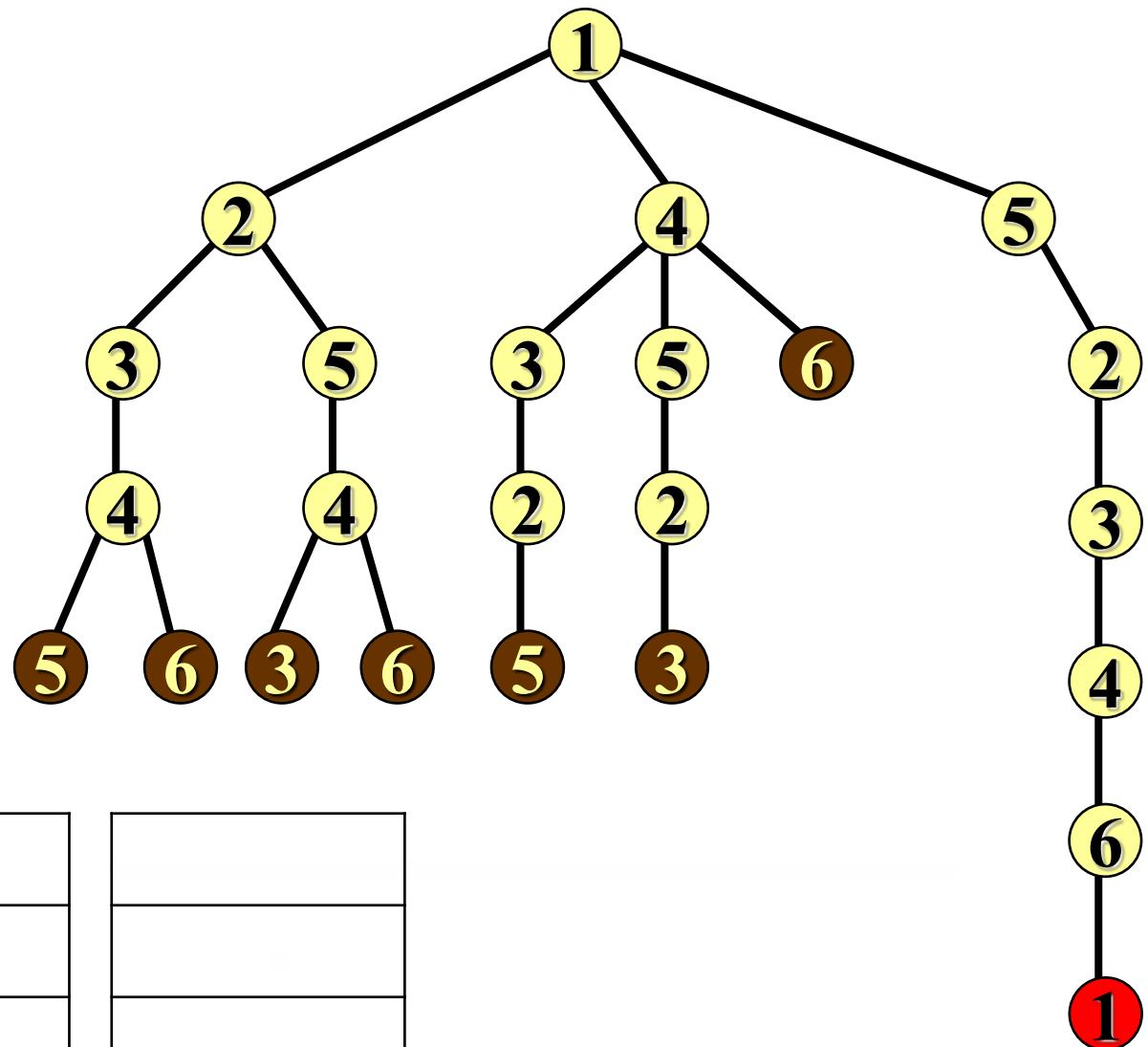
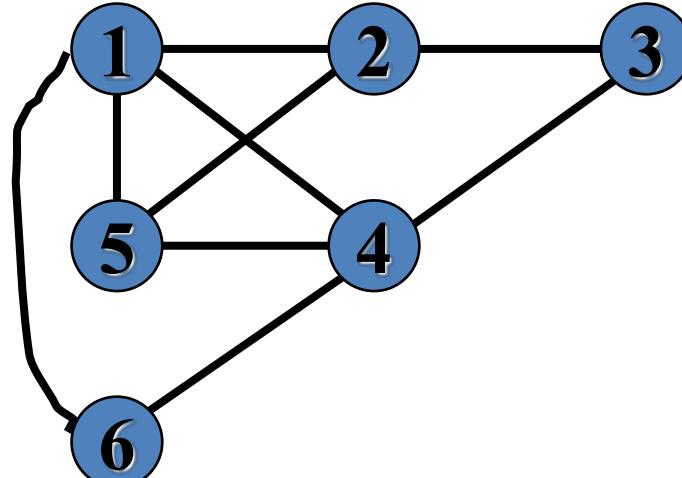


• 例：求解Hamiltonian环问题



| | | | | | | | | |
|-------|----|-----|------|-------|-----|-----|------|-------|
| | | | | | | | | |
| | | | | | | | | |
| 12546 | | 143 | 1432 | 14325 | | | 1452 | 14523 |
| 14 | 14 | 145 | 145 | 145 | 145 | 146 | 146 | 146 |
| 15 | 15 | 146 | 146 | 146 | 146 | 15 | 15 | 15 |
| 16 | 16 | 15 | 15 | 15 | 16 | 16 | 16 | 16 |

- 例：求解Hamiltonian环问题



| | | |
|----|-----|------|
| | | |
| | | |
| | | |
| | | |
| 15 | 152 | 1523 |
| 16 | 16 | 16 |

| | |
|----|-------|
| | |
| | |
| | |
| | |
| 15 | 15234 |
| 16 | 154 |

BFS vs DFS

- 1、如果找对了路， DFS很快就能找到目标节点
- 2、如果目标节点离根节点很近， DFS的效率可能不如BFS
- 3、DFS的空间复杂度比BFS要高， 一般是 $O(d)$
- 4、如果有多个解， DFS找到的解不一定是离根节点最近的解

- 1、如果目标节点离根节点很远， BFS效率比DFS低
- 2、如果目标节点离根节点很近， BFS很快能找到解
- 3、BFS的空间复杂度比DFS高， 一般为 $O(2^d)$
- 4、如果有多个解， BFS找到的一定是离根节点最近的解

思考题1

题目描述

有一个 $n \times m$ 的棋盘，在某个点 (x,y) 上有一个马，要求你计算出马到达棋盘上任意一个点最少要走几步。

输入格式

输入只有一行四个整数，分别为 n, m, x, y 。

输出格式

一个 $n \times m$ 的矩阵，代表马到达某个点最少要走几步（不能到达则输出 -1）。

输入样例#1

3 3 1 1

输出 #1

0 3 2

3 -1 1

2 1 4

思考题2

题目描述

给定一个 $N \times M$ 方格的迷宫，迷宫里有 T 处障碍，障碍处不可通过。在迷宫中移动有上下左右四种方式，每次只能移动一个方格。数据保证起点上没有障碍。给定起点坐标和终点坐标，每个方格最多经过一次，问有多少种从起点坐标到终点坐标的方案。

输入格式

第一行为三个正整数 N 、 M 、 T ，分别表示迷宫的长宽和障碍总数。第二行为四个正整数 SX 、 SY 、 FX 、 FY ， SX 、 SY 代表起点坐标， FX 、 FY 代表终点坐标。接下来 T 行，每行两个正整数，表示障碍点的坐标。

输出格式

输出从起点坐标到终点坐标的方案总数。

输入输出样例

输入：

2 2 1
1 1 2 2
1 2

输出：

1

6.3 搜索的优化



爬山法

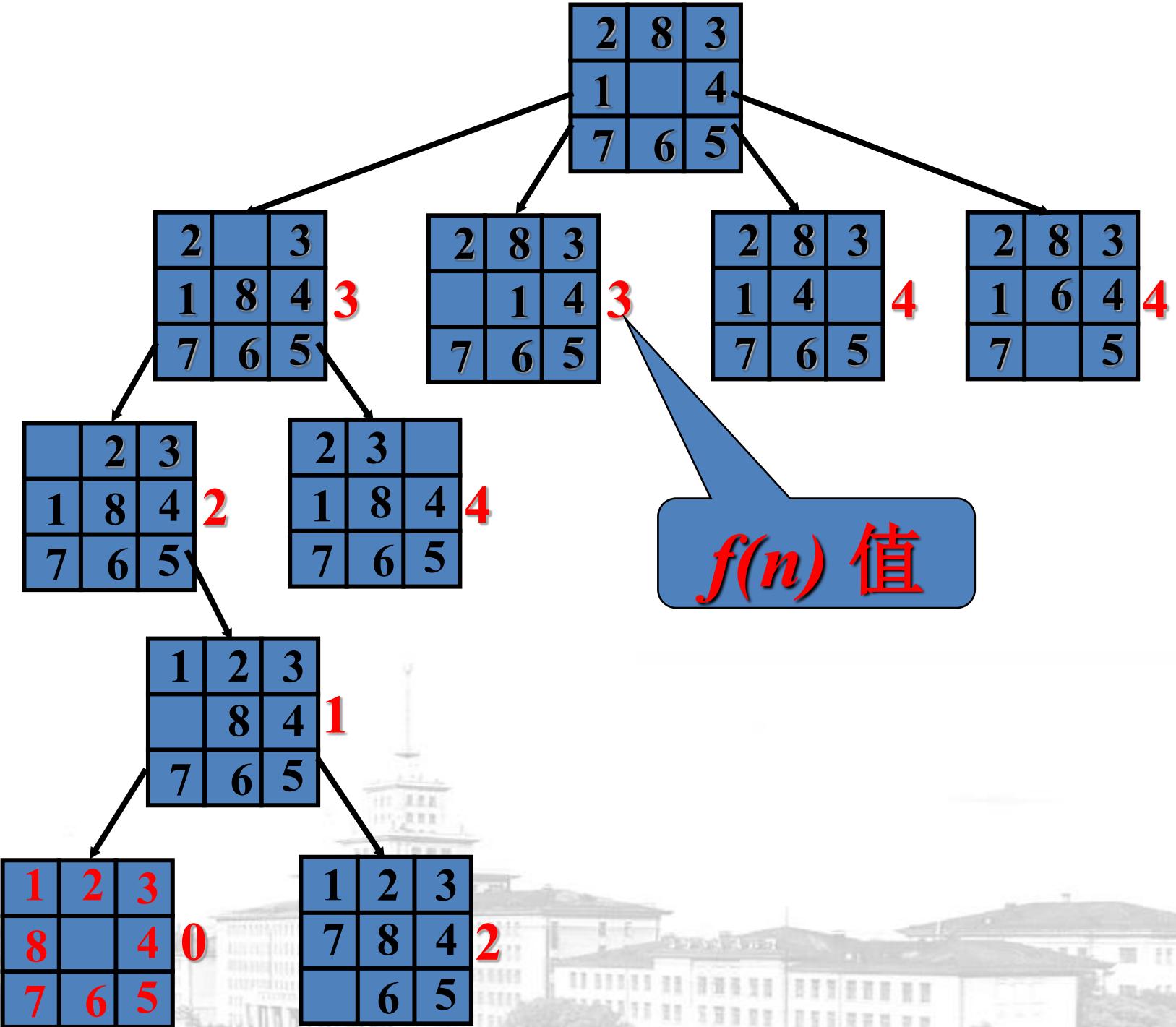
- 基本思想
 - 在深度优先搜索过程中，我们经常遇到多个结点可以扩展的情况，首先扩展哪个？
 - 爬山策略使用贪心方法确定搜索的方向，是优化的深度优先搜索策略。
 - 爬山策略使用启发式测度来排序结点扩展的顺序。
 - 使用什么启发式测度与问题本身相关。

深度优先搜索 + 启发式测度

- 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 |

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 8 | | 4 |
| 7 | 6 | 5 |



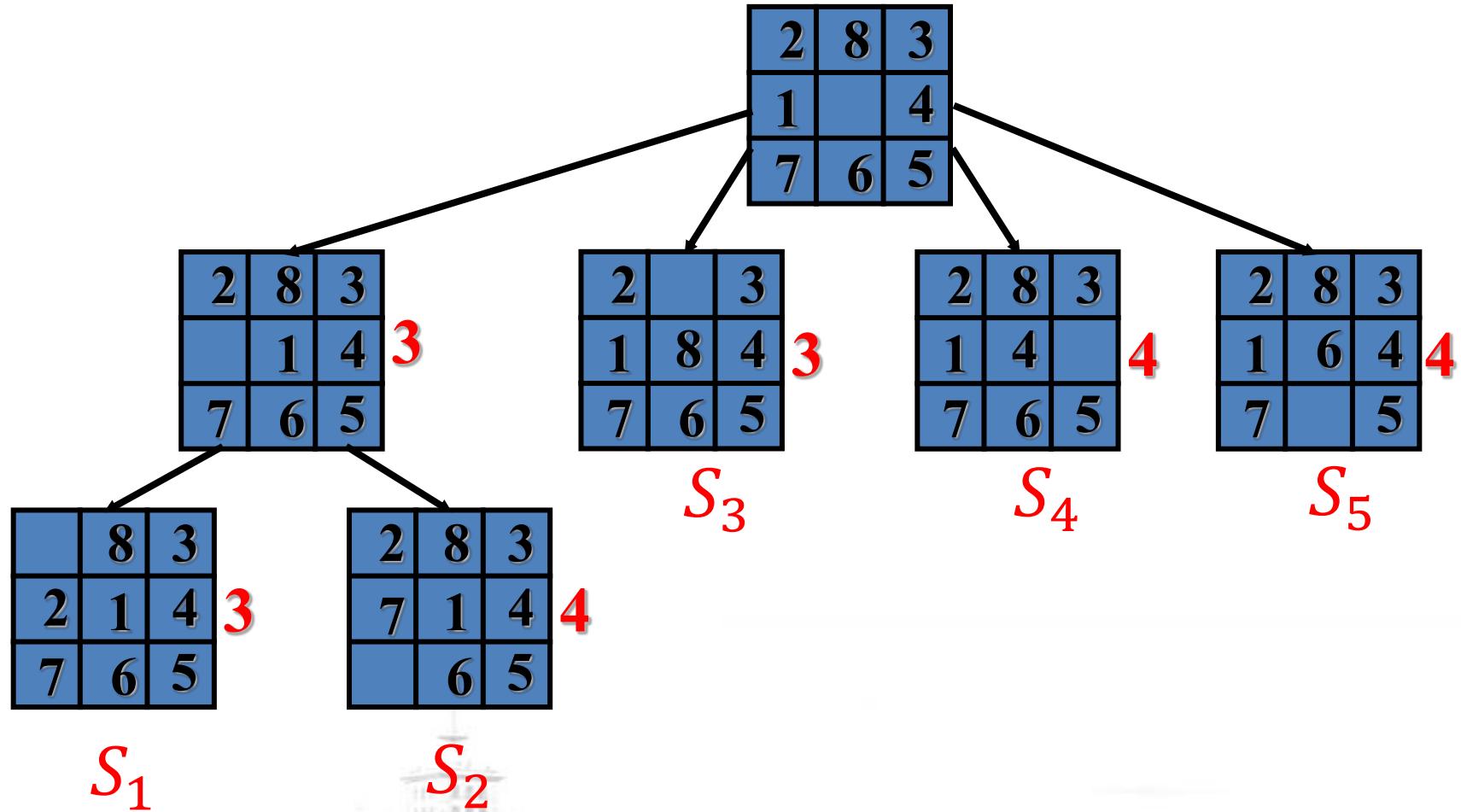
• 爬山法算法

Climb-Search

Input: 问题输入

Output: 问题输出

1. $S \leftarrow \{root\}$ // 栈
2. While not $S.empty()$ Do
3. $x \leftarrow Q.pop()$
4. If x 是目标节点 Then Return x
5. For $v \in sorted(\{x\text{可以扩展的节点}\})$ Do
6. $Q.push(v)$ \\ 按启发式测度从大到小排序
7. Return 无解



Best-First 搜索策略

- 基本思想

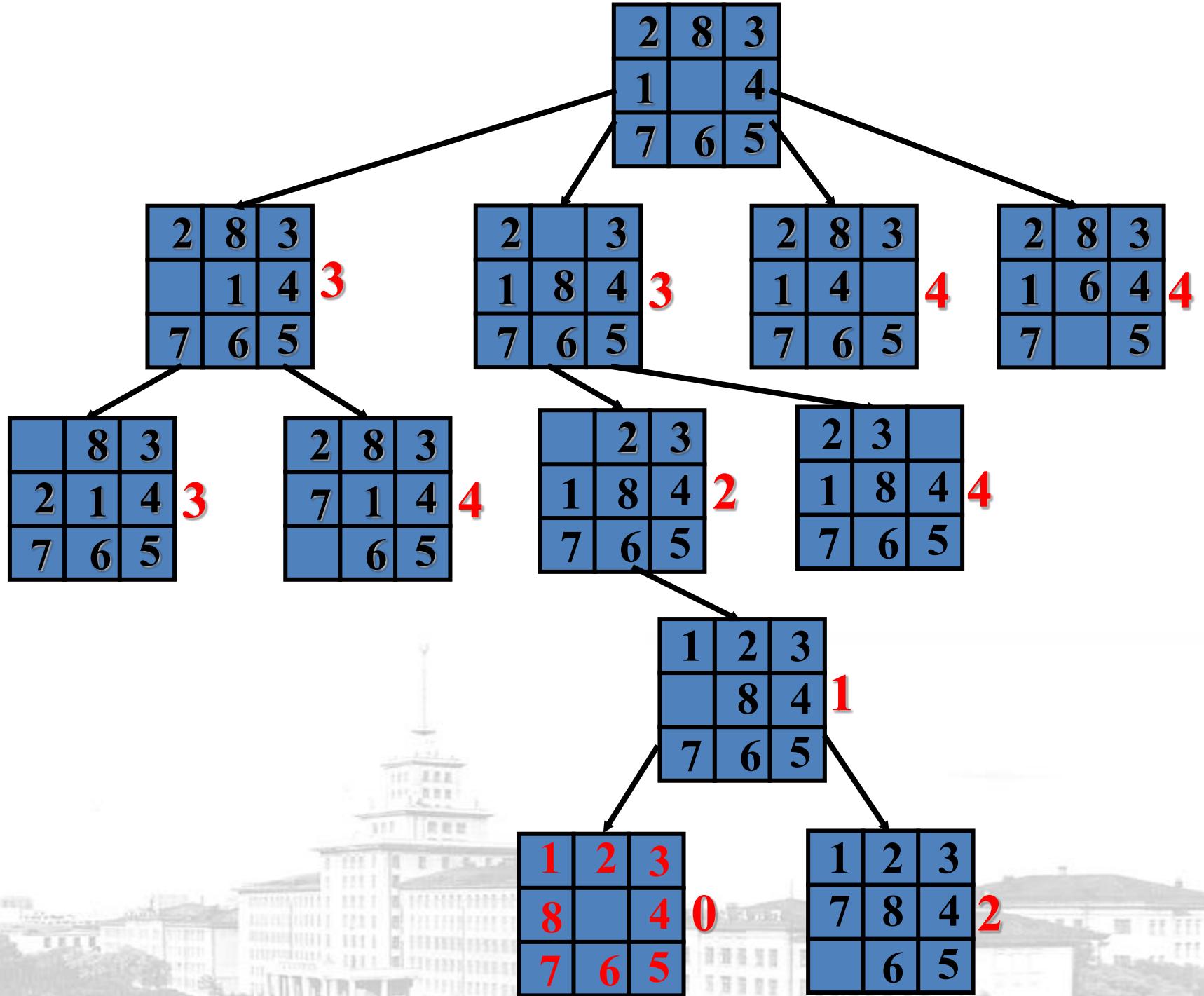
- 结合深度优先和广度优先的优点
- 根据一个评价函数, 在目前产生的所有节点中选择具有最小评价函数值的节点进行扩展.
- 具有全局优化观念, 而爬山策略仅具有局部优化观念.

- Best-First 搜索算法

Best-First-Search

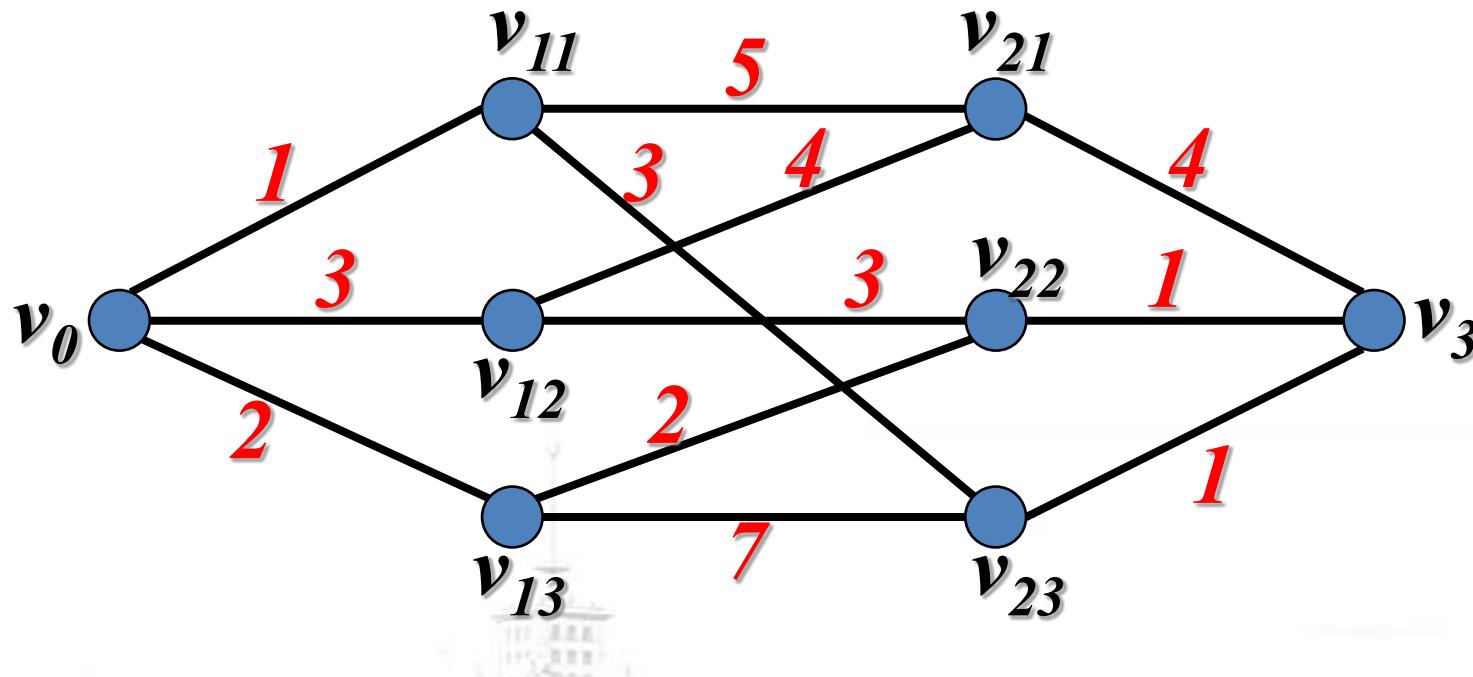
1. $H \leftarrow \{root\}$ // 最小堆
2. While not $H.empty()$ Do
3. $x \leftarrow H.Extract_Min()$
4. If x 是目标节点 Then Return x
5. For $v \in x$ 可以扩展的节点 Do
6. $H.Insert(v)$
7. Return 无解

- 8-Puzzle 问题实例



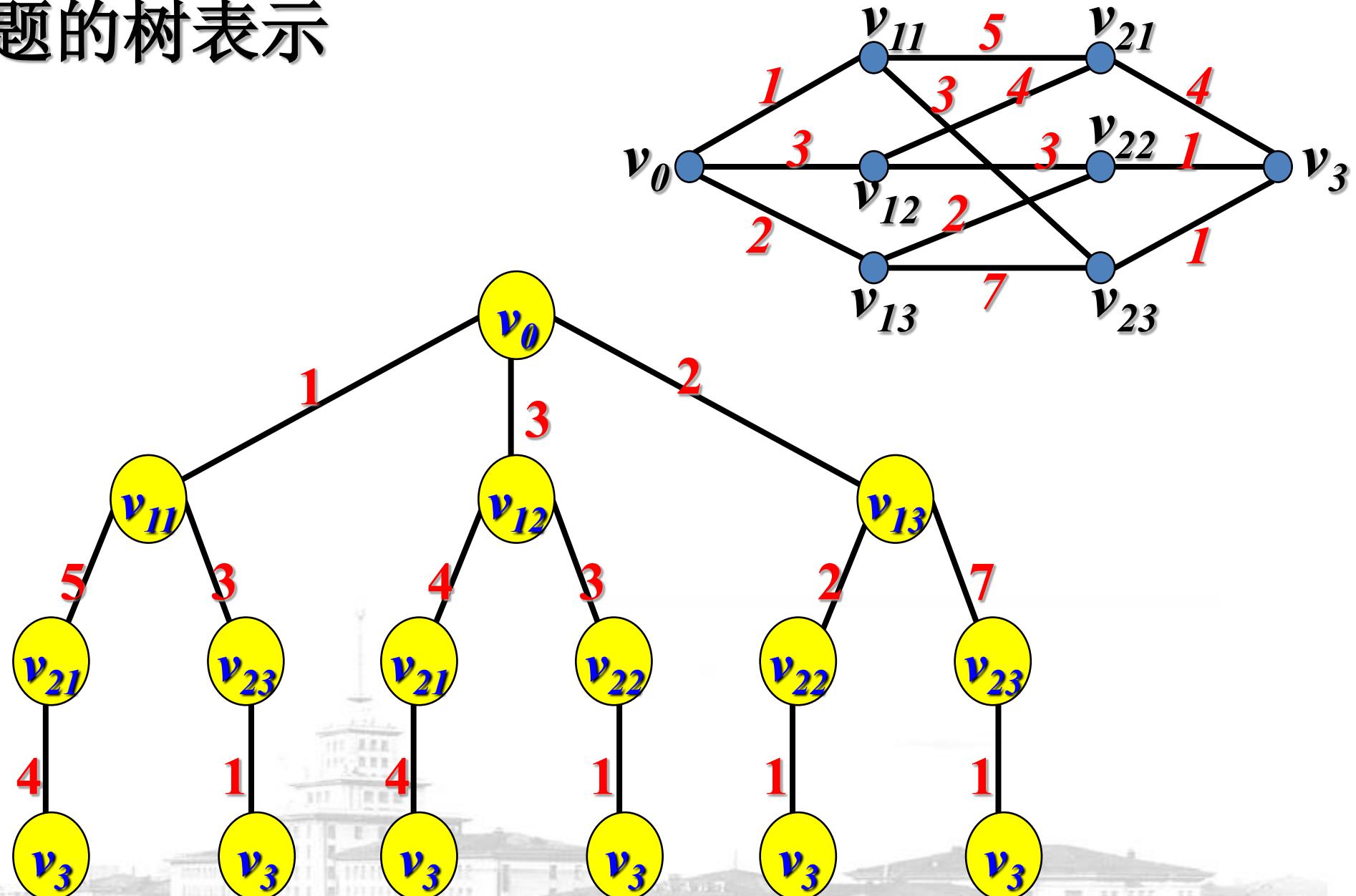
- 基本思想
 - 上述方法很难用于求解优化问题
 - 分支界限策略可以有效地求解组合优化问题
 - 发现优化解的一个界限
 - 缩小解空间, 提高求解的效率
- 举例说明分支界限策略的原理

- 多阶段图搜索问题
 - 输入: 多阶段图

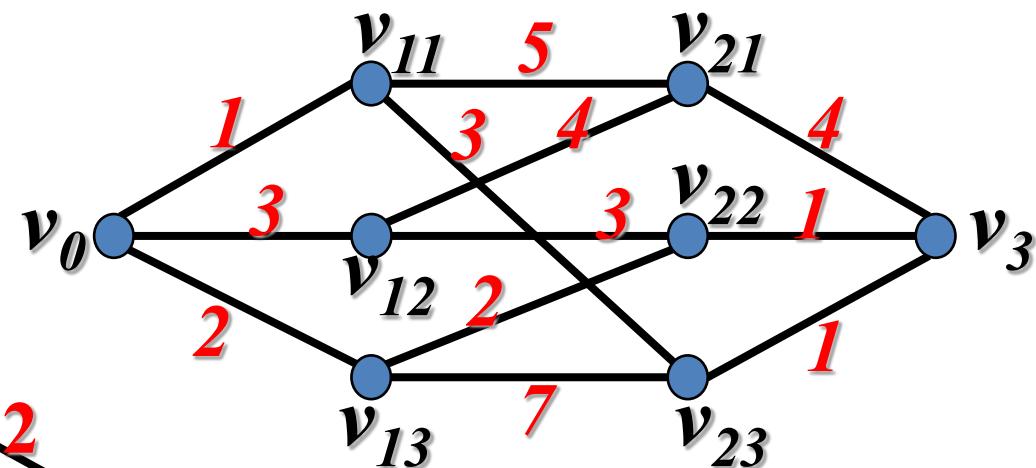
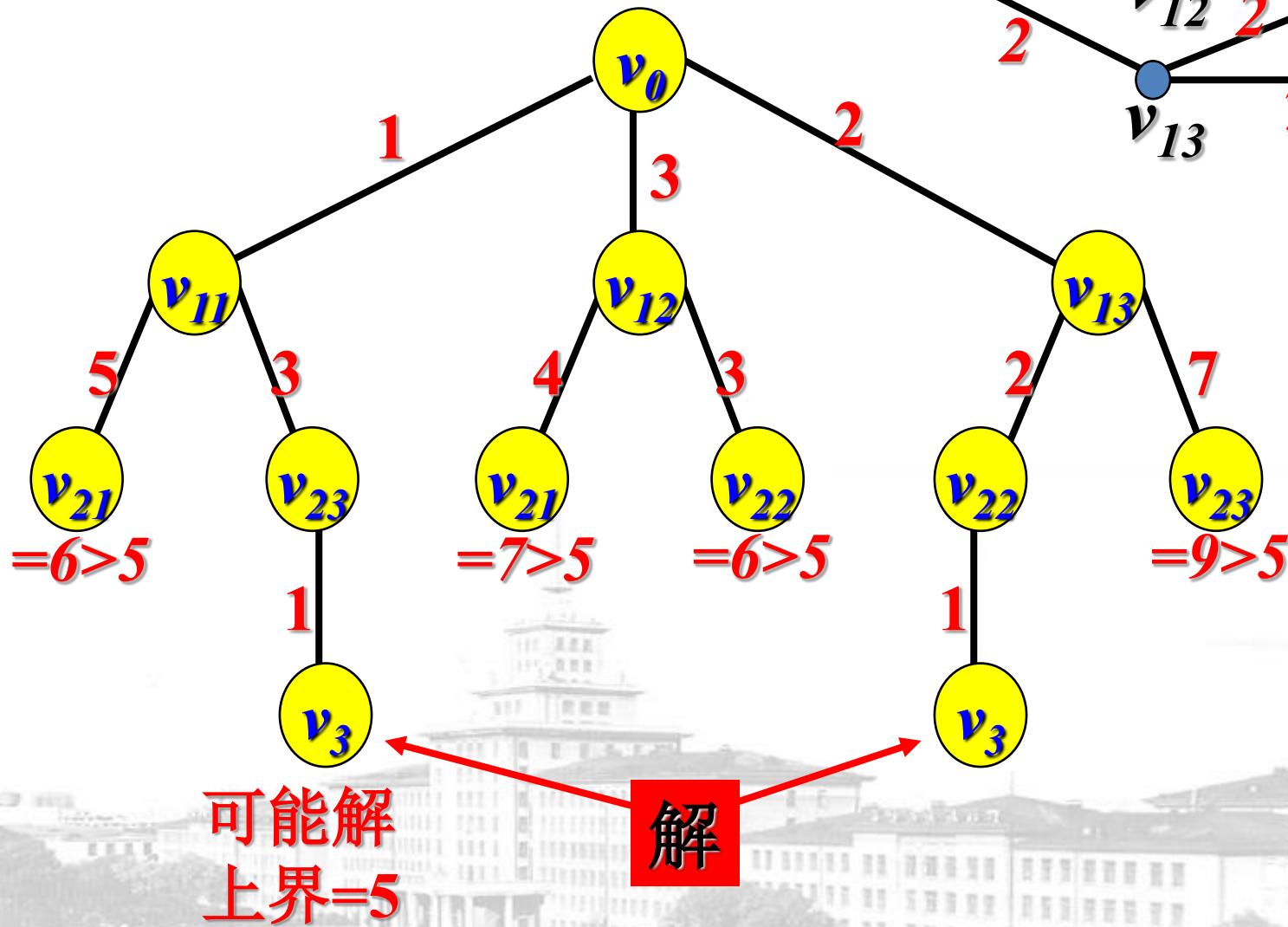


- 输出: 从 v_0 到 v_3 的最短路径

- 问题的树表示



- 使用爬山策略进行分支界限搜索

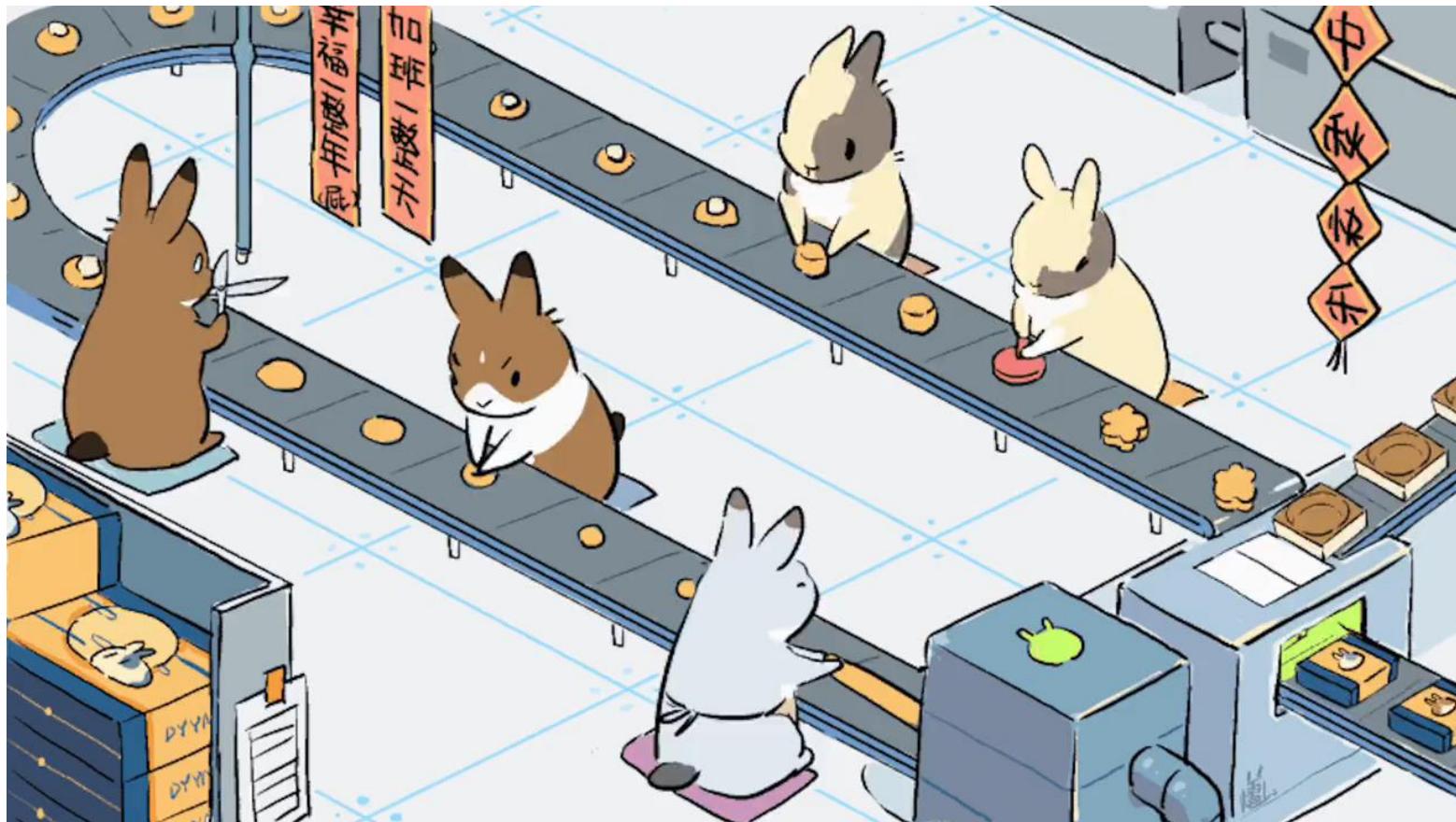


- 分支界限策略的原理
 - 产生分支的机制(使用前面的任意一种策略)
 - 产生一个界限(可以通过发现可能解)
 - 进行分支界限搜索, 即剪除不可能产生优化解的分支.

6.4 剪枝方法论与人员安排问题



人员安排问题



人员安排问题的定义

- 输入

- 人的集合 $P = \{P_1, P_2, \dots\}$

集合 $J = \{J_1, J_2, \dots, J_n\}$,

- 矩阵 $[C_{ij}]$, C_{ij} 是工作 J_i 由人 P_j 完成的可能性

- 输出

- 矩阵 $[X_{ij}]$, $X_{ij} = 1$ 表示人 P_j 被分配到工作 J_i

- 每个人被分配一种工作

| 课程代号 | 课程名称 | 先修课程 |
|----------------|----------|---------------------------------|
| C ₁ | 高等数学 | |
| C ₂ | 程序设计基础 | |
| C ₃ | 离散数学 | C ₁ , C ₂ |
| C ₄ | 数据结构 | C ₃ , C ₂ |
| C ₅ | 高级语言程序设计 | C ₂ |
| C ₆ | 编译方法 | C ₅ , C ₄ |
| C ₇ | 操作系统 | C ₄ , C ₉ |
| C ₈ | 普通物理 | C ₁ |
| C ₉ | 计算机原理 | C ₈ |

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

$P_1 \rightarrow J_1$ 、 $P_2 \rightarrow J_2$ 、 $P_3 \rightarrow J_3$ 是可行解。

$P_1 \rightarrow J_1$ 、 $P_2 \rightarrow J_3$ 、 $P_3 \rightarrow J_2$ 不可能是解。

转换为树搜索问题

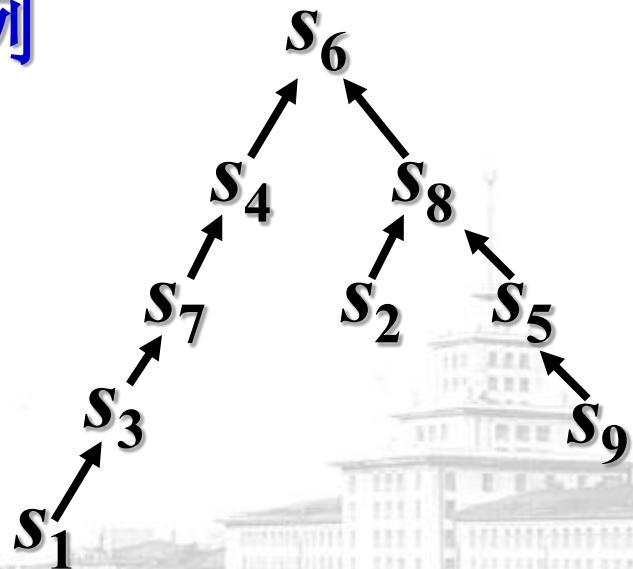
- 拓朴排序

- 输入: 偏序集合(S, \leq)

- 输出: S 的拓朴序列是 $\langle s_1, s_2, \dots, s_n \rangle$,

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

- 例



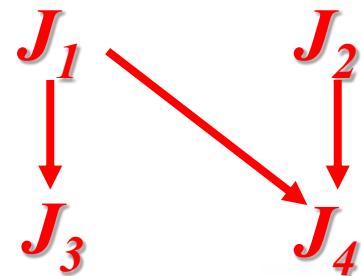
- 拓朴排序:

- $s_1 \ s_3 \ s_7 \ s_4 \ s_9 \ s_5 \ s_2 \ s_8 \ s_6$

- 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.

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

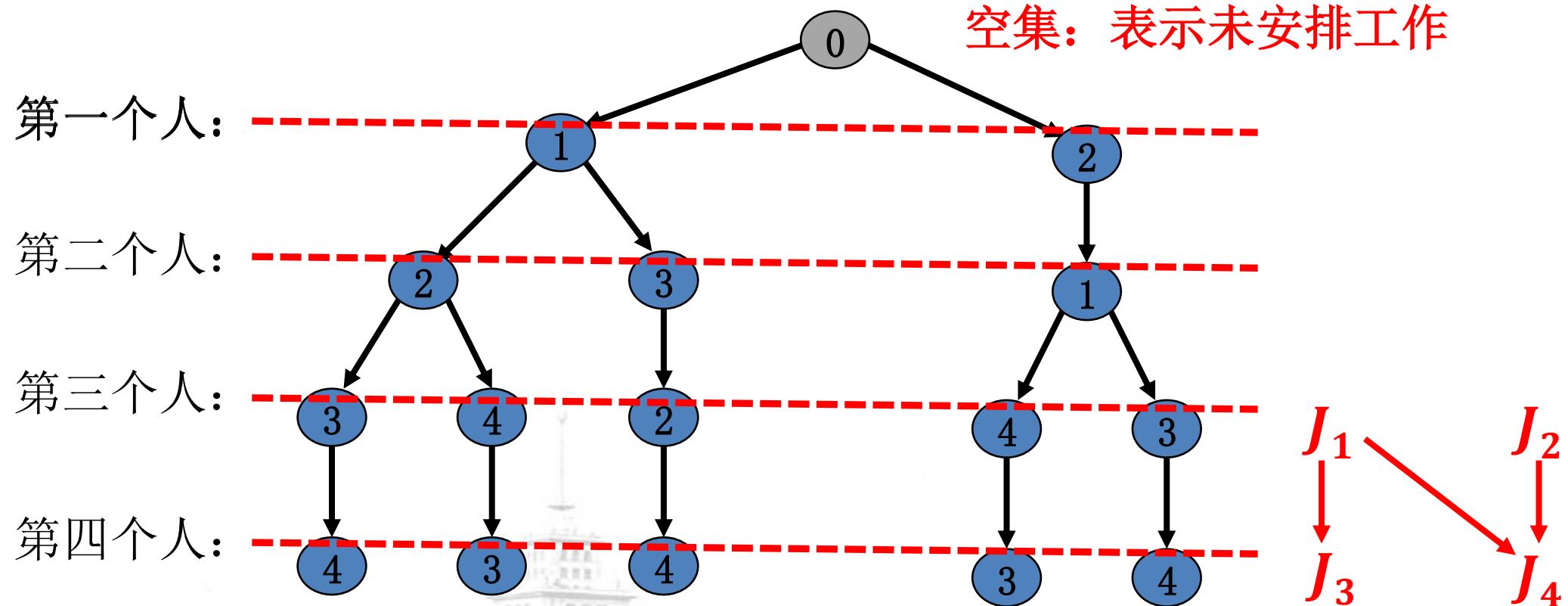


则 (J_1, J_2, J_3, J_4) 、 (J_1, J_2, J_4, J_3) 、 (J_1, J_3, J_2, J_4) 、 (J_2, J_1, J_3, J_4) 、 (J_2, J_1, J_4, J_3) 是拓朴排序序列

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

转换为树搜索问题

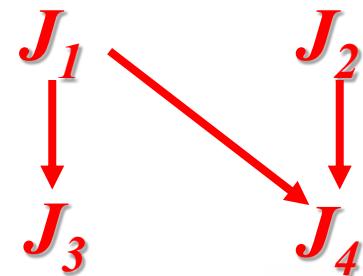
- 问题的树表示（即用树表示所有拓朴排序序列）



- 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.

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

● 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.

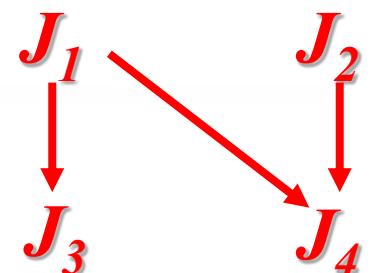
1. 生成树根 $root$;

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；

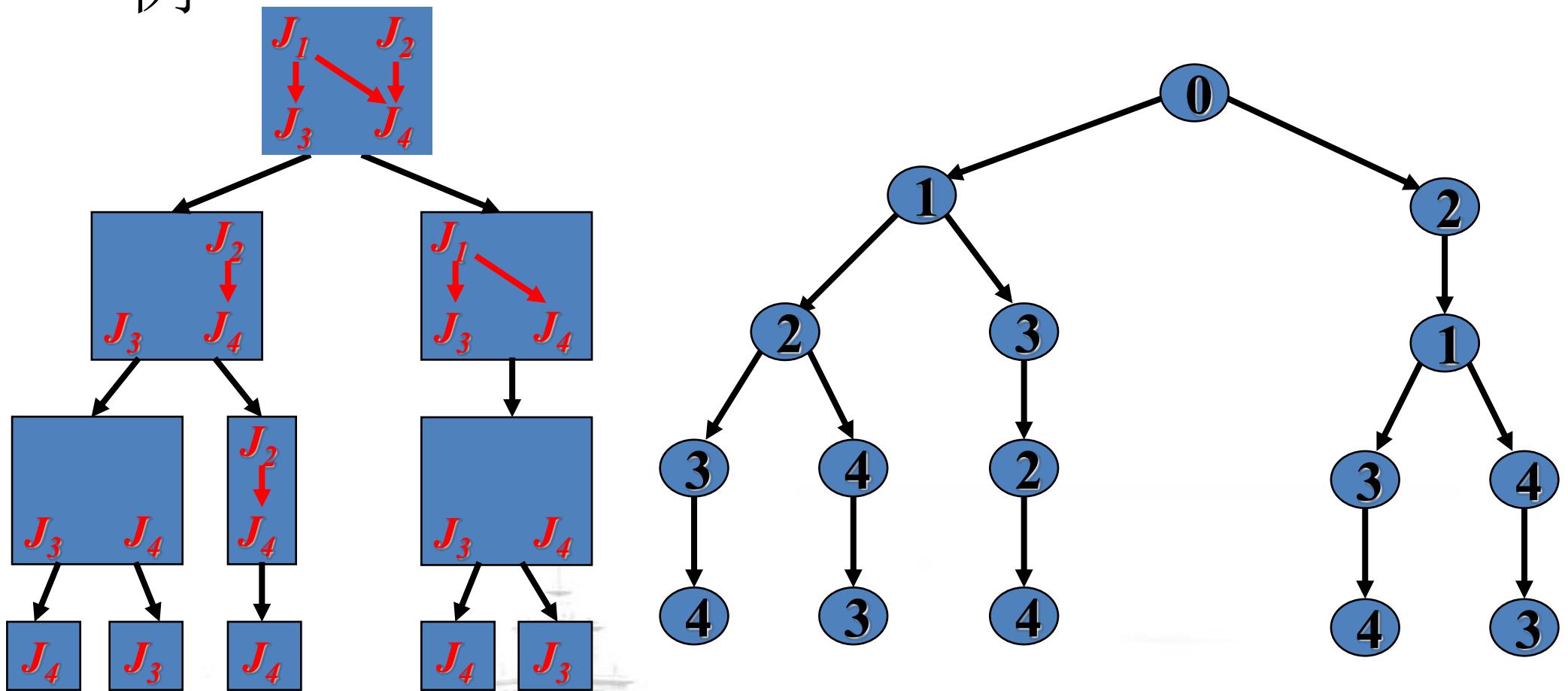
3. For $root$ 的每个字节点 v Do

4. $S=S-\{v\}$;

5. 把 v 作为根, 递归地处理 S .

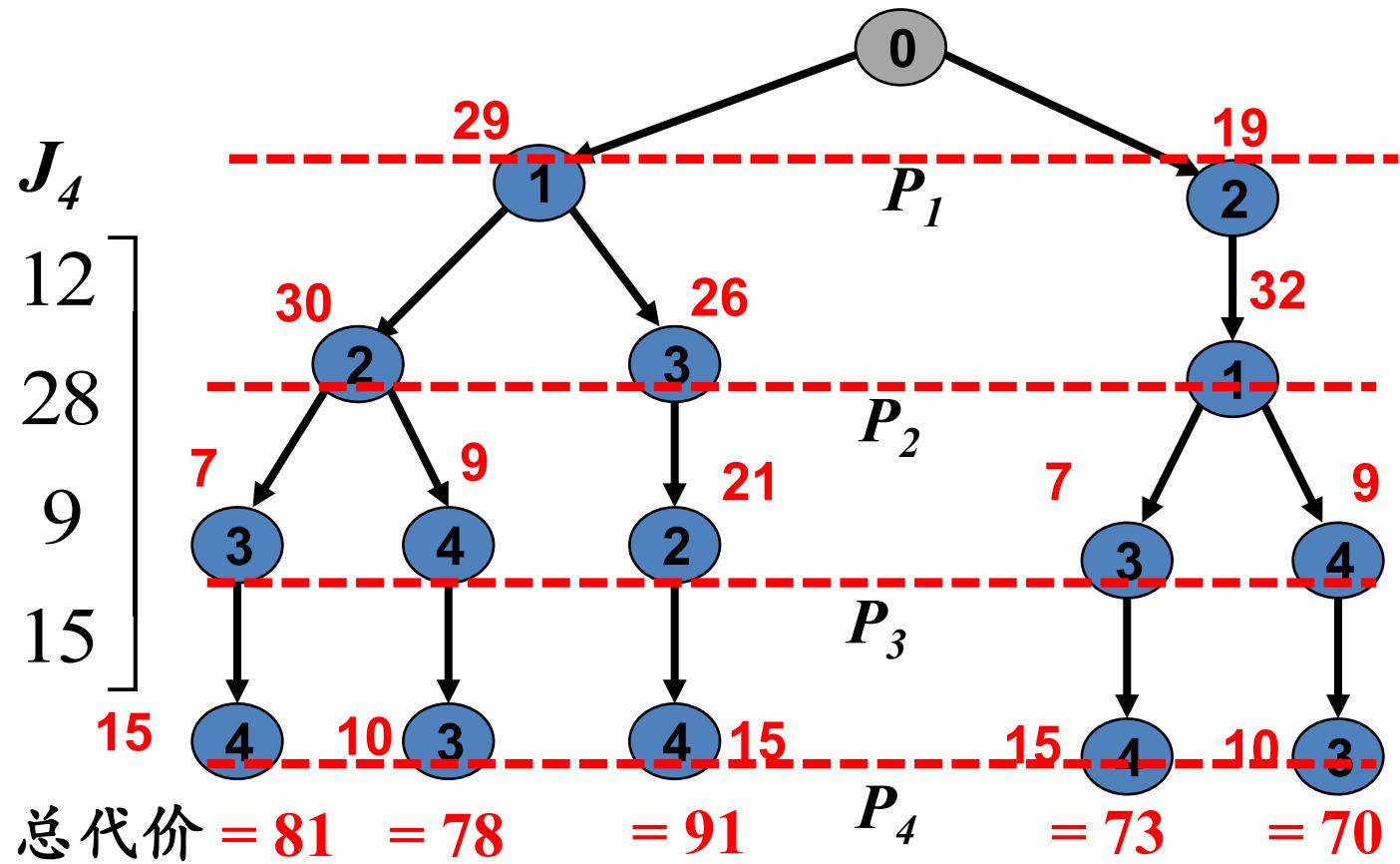


• 例



代价矩阵

| | 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 |



人员安排问题的所有可行解都已求出（穷举所有的可能解）

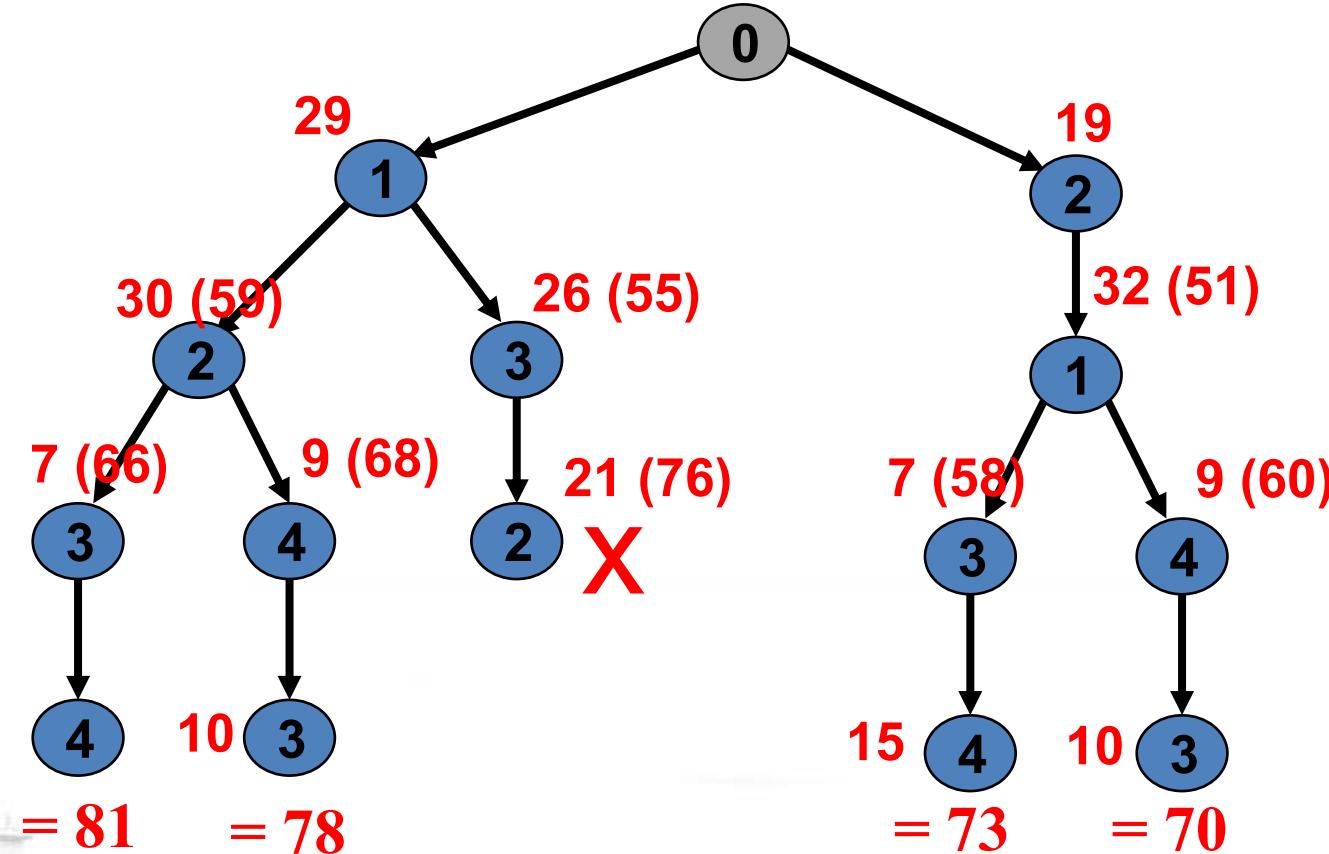
有没有更有效的算法？

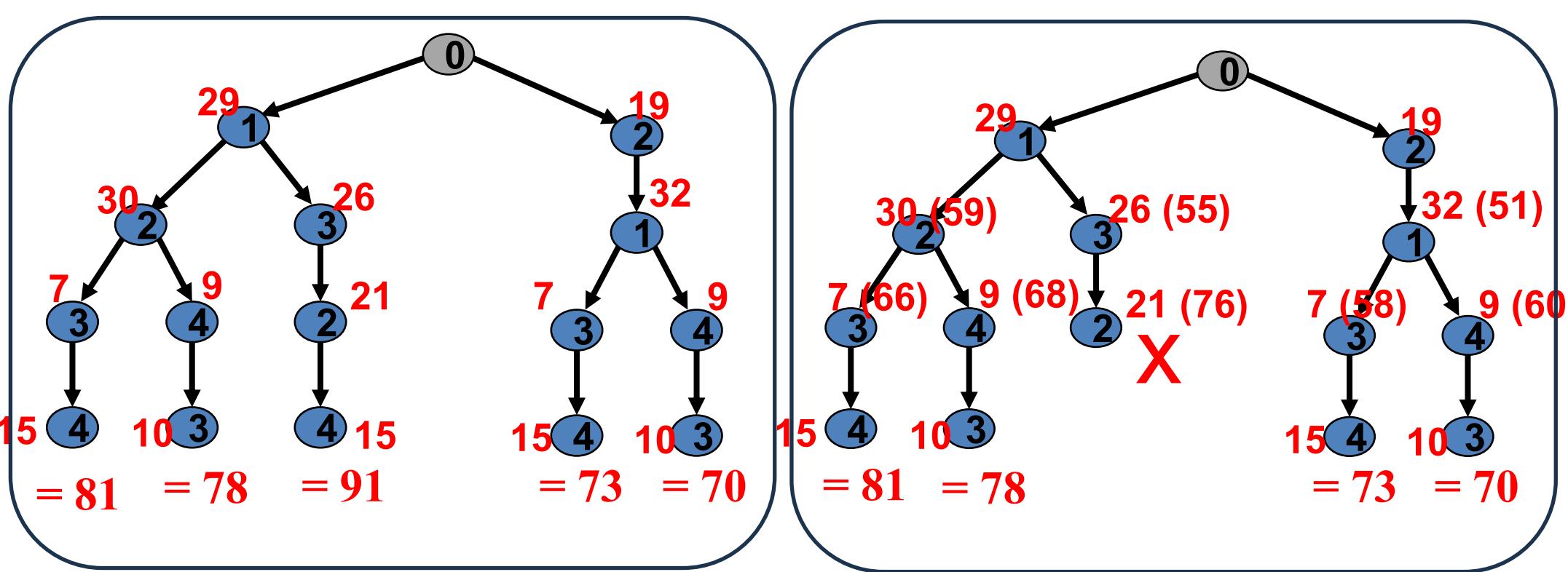
分支界限

代价矩阵

| | 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 |

总代价





分支界限似乎没有减少太多的搜索状态。

还能不能优化?

考虑 $f(x) = x_1 + x_2$, $x_1 \in \{2, 4\}$, $x_2 \in \{8, 16\}$

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.

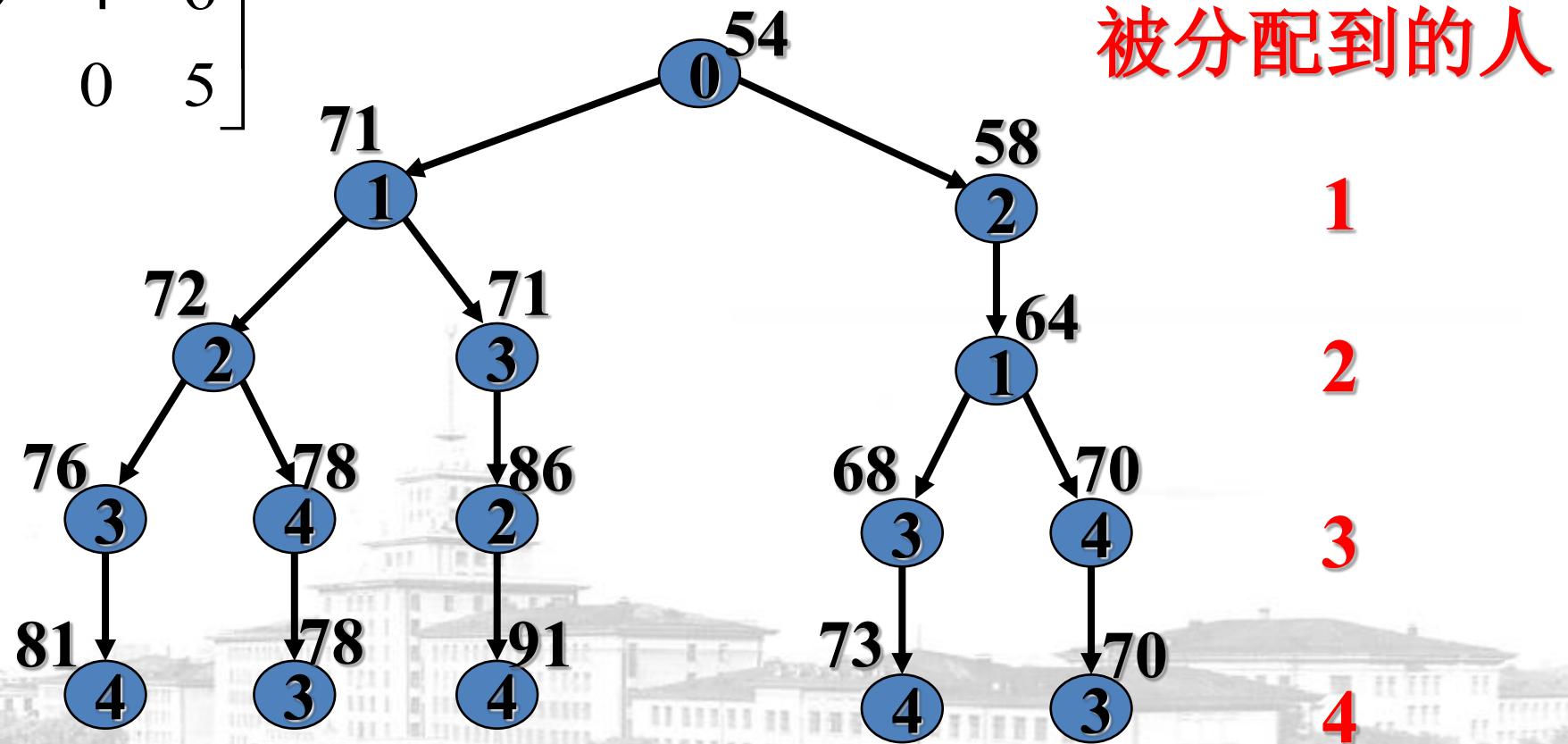
$$\begin{array}{c} J_1 \quad J_2 \quad J_3 \quad J_4 \\ P_1 \left[\begin{array}{cccc} 29 & 19 & 17 & 12 \end{array} \right] -12 \\ P_2 \left[\begin{array}{cccc} 32 & 30 & 26 & 28 \end{array} \right] -26 \\ P_3 \left[\begin{array}{cccc} 3 & 21 & 7 & 9 \end{array} \right] -3 \\ P_4 \left[\begin{array}{cccc}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

$$\xrightarrow{\text{Red Arrow}} \begin{bmatrix} 17 & 7 & 5 & 0 \\ 6 & 4 & 0 & 2 \\ 0 & 18 & 4 & 6 \\ 8 & 3 & 0 & 5 \end{bmatrix} \downarrow \begin{bmatrix} 17 & 4 & 5 & 0 \\ 6 & 1 & 0 & 2 \\ 0 & 15 & 4 & 6 \\ 8 & 0 & 0 & 5 \end{bmatrix}$$

解代价下界
 $=12+26+3+10+3=54$

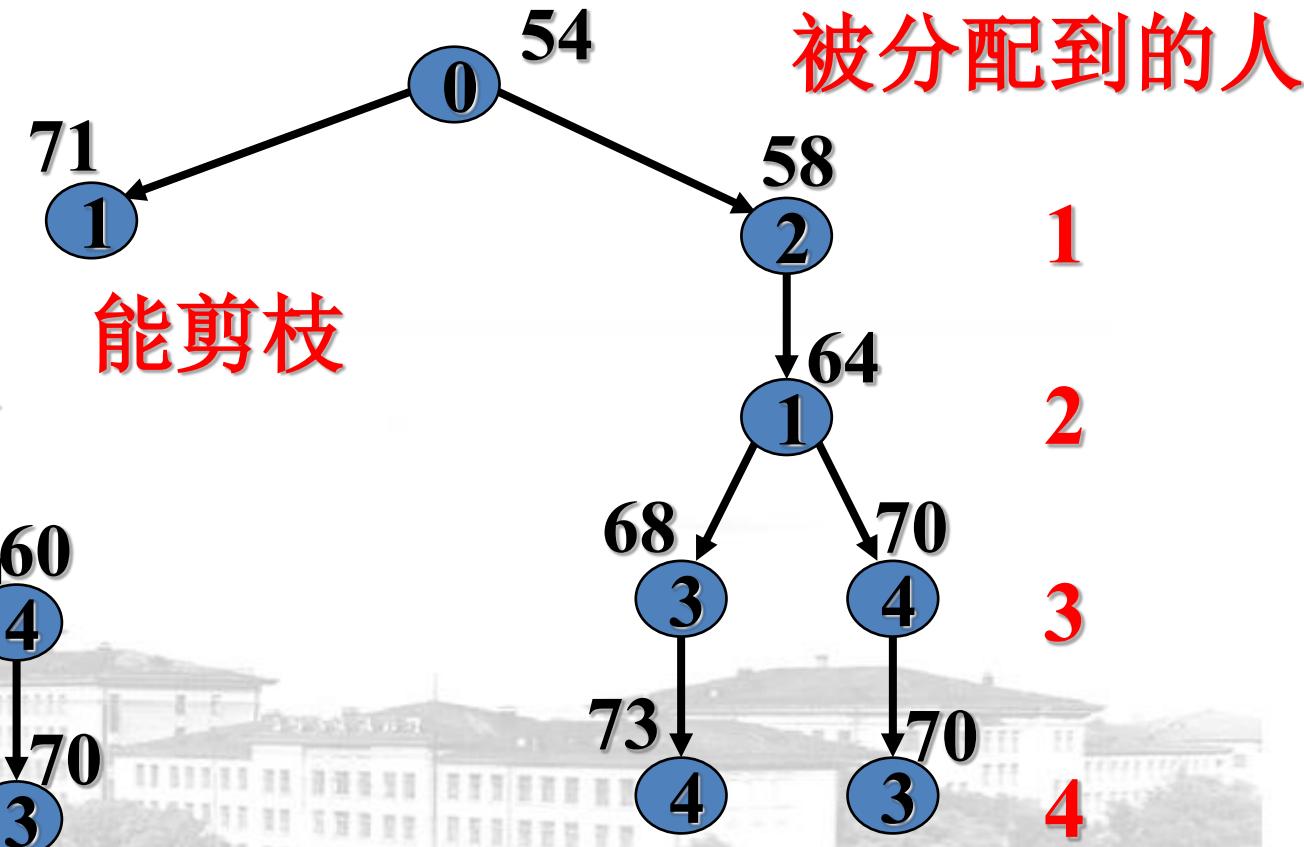
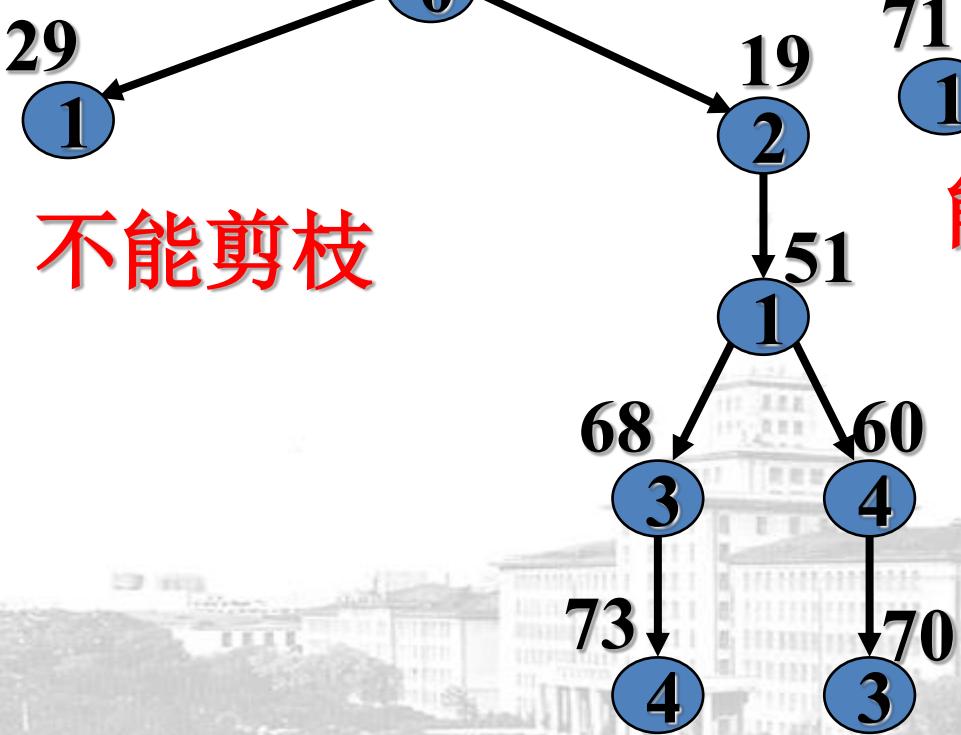
解空间的加权树表示

| | | | |
|----|----|---|---|
| 17 | 4 | 5 | 0 |
| 6 | 1 | 0 | 2 |
| 0 | 15 | 4 | 6 |
| 8 | 0 | 0 | 5 |



用初始代价矩阵与代价下界矩阵计算最优代价对比

| | | | |
|----|----|----|----|
| 29 | 19 | 17 | 12 |
| 32 | 30 | 26 | 28 |
| 3 | 21 | 7 | 9 |
| 18 | 13 | 10 | 15 |



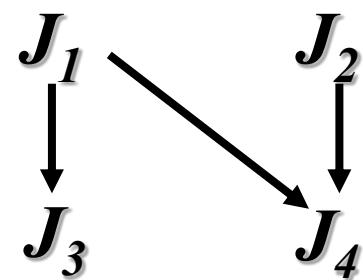
| | | | |
|----|----|---|---|
| 17 | 4 | 5 | 0 |
| 6 | 1 | 0 | 2 |
| 0 | 15 | 4 | 6 |
| 8 | 0 | 0 | 5 |

- 分支界限搜索(使用爬山法)算法

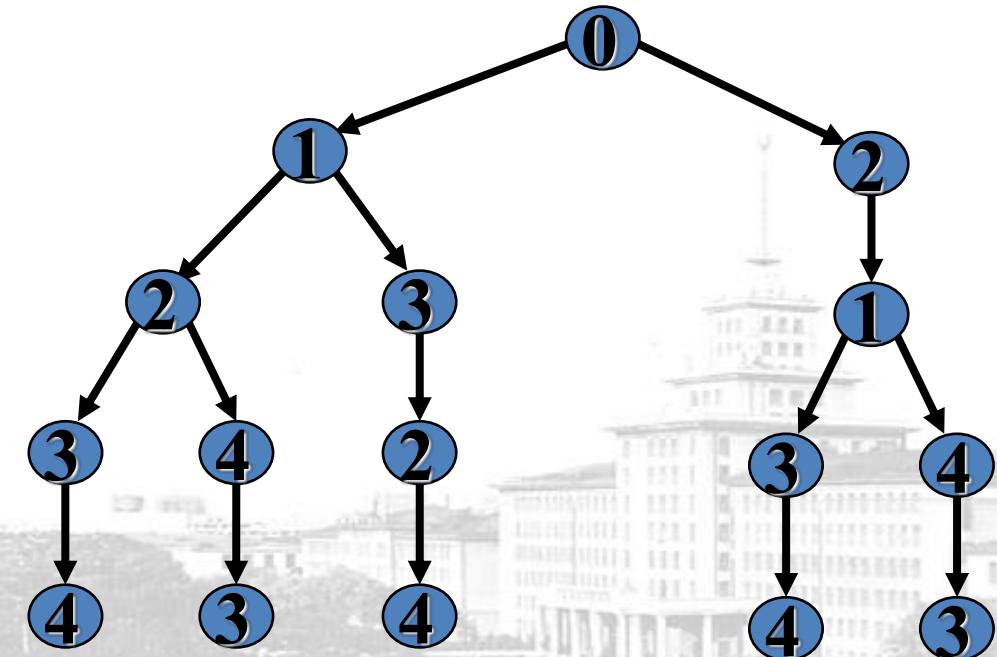
1. 建立根节点, 其权值为解代价下界;
2. 使用爬山法, 类似于拓朴排序序列树生成算法求解问题, 每产生一个节点, 其权值为加工后的代价矩阵对应元素加其父节点权值;
3. 一旦发现一个可能解, 将其代价作为界限, 循环地进行分支界限搜索: 剪掉不能导致优化解的解, 使用爬山法继续扩展新增节点, 直至发现优化解.

• 例

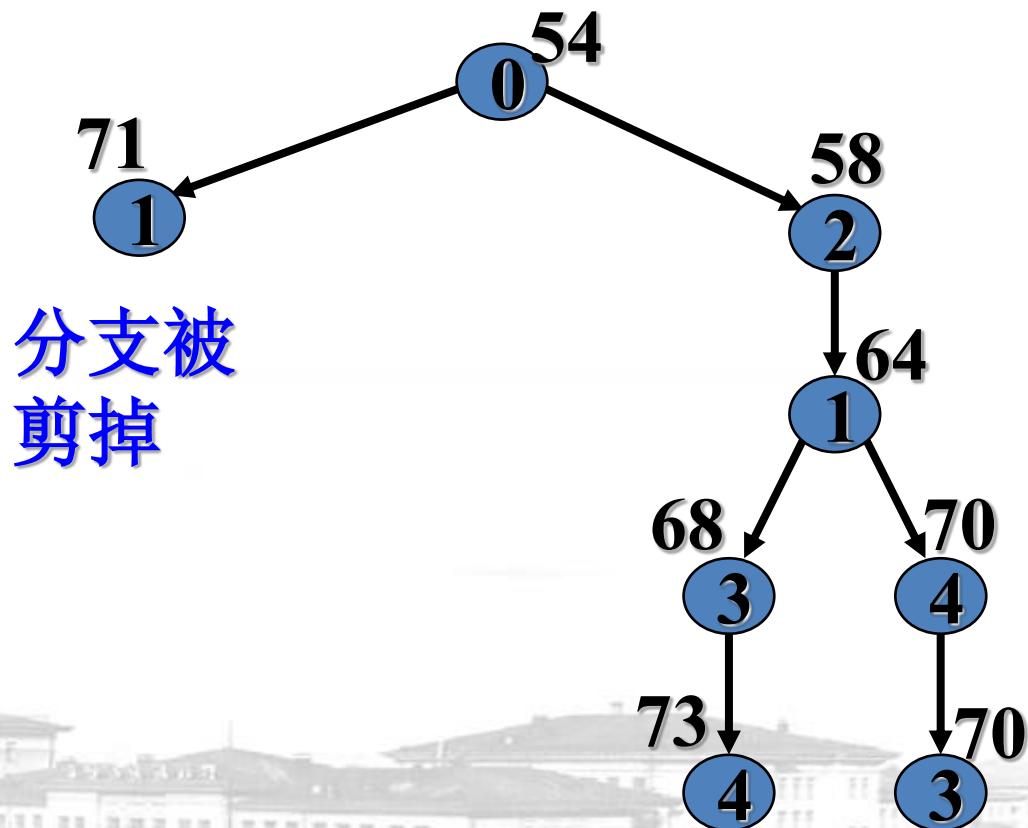
$$\{P_1, P_2, P_3, P_4\}$$



| | | | |
|----|----|---|---|
| 17 | 4 | 5 | 0 |
| 6 | 1 | 0 | 2 |
| 0 | 15 | 4 | 6 |
| 8 | 0 | 0 | 5 |



分支被
剪掉



6.5旅行商问题



问题的定义

输入：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(即权值之和)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分解空间，得到二叉树
- 划分过程：
 - 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
 - 选择图上边 (i, J) 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

| $j = 1$ | 2 | 3 | 4 | 5 | 6 | 7 | | |
|---------|----------|----------|----------|----------|----------|----------|----------|-----|
| $i=1$ | ∞ | 3 | 93 | 13 | 33 | 9 | 57 | -3 |
| 2 | 4 | ∞ | 77 | 42 | 21 | 16 | 34 | -4 |
| 3 | 45 | 17 | ∞ | 36 | 16 | 28 | 25 | -16 |
| 4 | 39 | 90 | 80 | ∞ | 56 | 7 | 91 | -7 |
| 5 | 28 | 46 | 88 | 33 | ∞ | 25 | 57 | -25 |
| 6 | 3 | 88 | 18 | 46 | 92 | ∞ | 7 | -3 |
| 7 | 44 | 26 | 33 | 27 | 84 | 39 | ∞ | -26 |
| | | | | | -7 | -1 | | -4 |

- 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

| | | $j = 1$ | 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 | 32 | 83 | 66 | ∞ | 49 | 0 | 80 | |
| 5 | 3 | 21 | 56 | 7 | ∞ | 0 | 28 | |
| 6 | 0 | 85 | 8 | 42 | 89 | ∞ | 0 | |
| 7 | 18 | 0 | 0 | 0 | 58 | 13 | ∞ | |

- 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
- ✓ 右子节点为不包括边(4, 6)的所有解集合

| | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|
| ∞ | 0 | 83 | 9 | 30 | 6 | 50 |
| 0 | ∞ | 66 | 37 | 17 | 12 | 26 |
| 29 | 1 | ∞ | 19 | 0 | 12 | 5 |
| 32 | 83 | 66 | ∞ | 49 | 0 | 80 |
| 3 | 21 | 56 | 7 | ∞ | 0 | 28 |
| 0 | 85 | 8 | 42 | 89 | ∞ | 0 |
| 18 | 0 | 0 | 0 | 58 | 13 | ∞ |

所有解的集合 L.B=96

包括边(4, 6)
的所有解集合

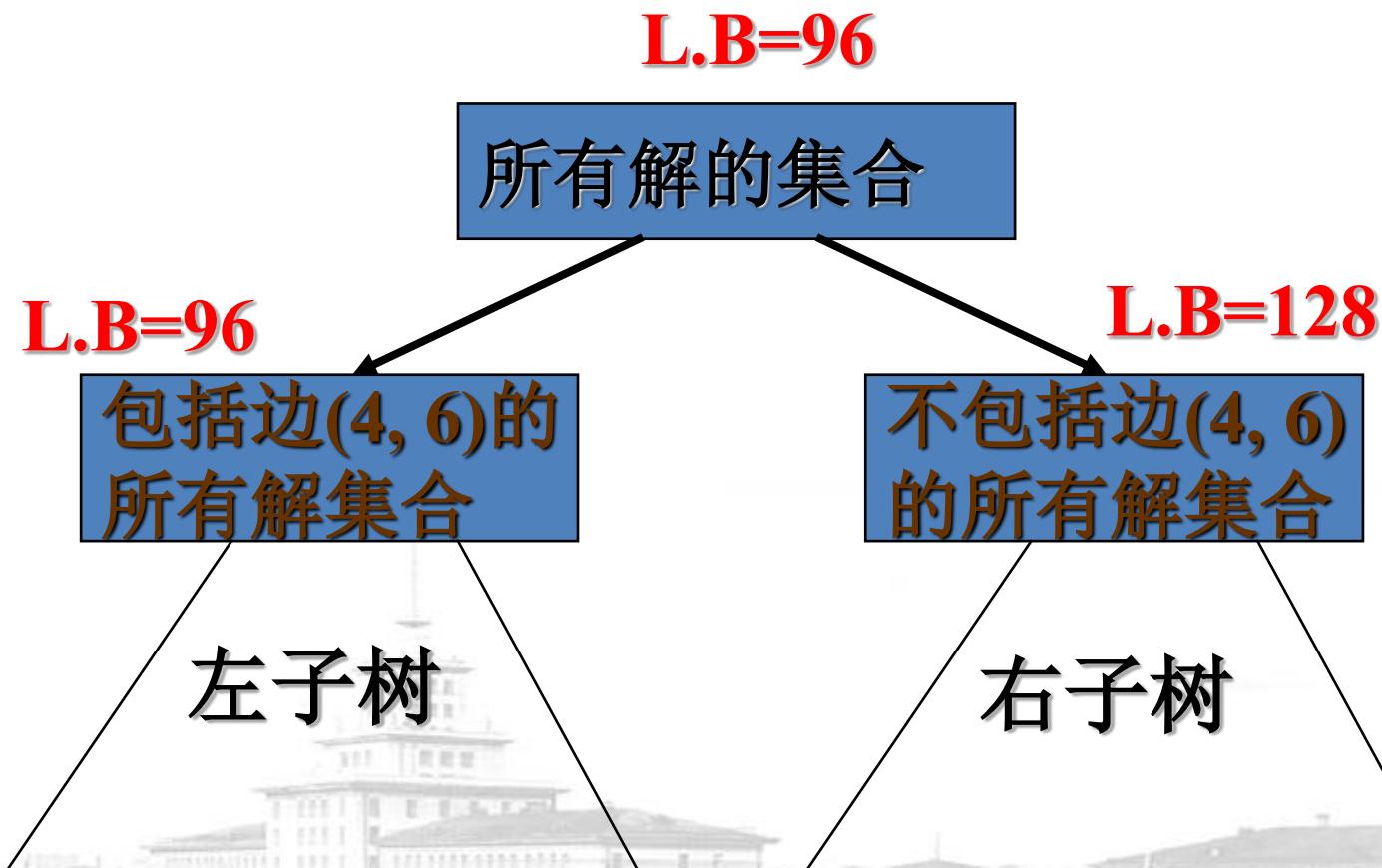
不包括边(4, 6)
的所有解集合

➤ 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.

| | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|
| ∞ | 0 | 83 | 9 | 30 | 6 | 50 |
| 0 | ∞ | 66 | 37 | 17 | 12 | 26 |
| 29 | 1 | ∞ | 19 | 0 | 12 | 5 |
| 32 | 83 | 66 | ∞ | 49 | 0 | 80 |
| 3 | 21 | 56 | 7 | ∞ | 0 | 28 |
| 0 | 85 | 8 | 42 | 89 | ∞ | 0 |
| 18 | 0 | 0 | 0 | 58 | 13 | ∞ |

➤ 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

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

➤ 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

| | $j = 1$ | 2 | 3 | 4 | 5 | |
|-------|----------|----------|----------|----------|----------|----------|
| $i=1$ | ∞ | 0 | 83 | 9 | 30 | 7 |
| 2 | 0 | ∞ | 66 | 37 | 17 | 26 |
| 3 | 29 | 1 | ∞ | 19 | 0 | 5 |
| | | | | | | |
| 5 | 0 | 18 | 53 | 4 | ∞ | 25 |
| 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

| | $j = 1$ | 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 | 32 | 83 | 66 | ∞ | 49 | ∞ | 80 |
| 5 | 3 | 21 | 56 | 7 | ∞ | 0 | 28 |
| 6 | 0 | 85 | 8 | 42 | 89 | ∞ | 0 |
| 7 | 18 | 0 | 0 | 0 | 58 | 13 | ∞ |

➤ 计算右子树根的代价下界

✓ 矩阵的第4行不包含0

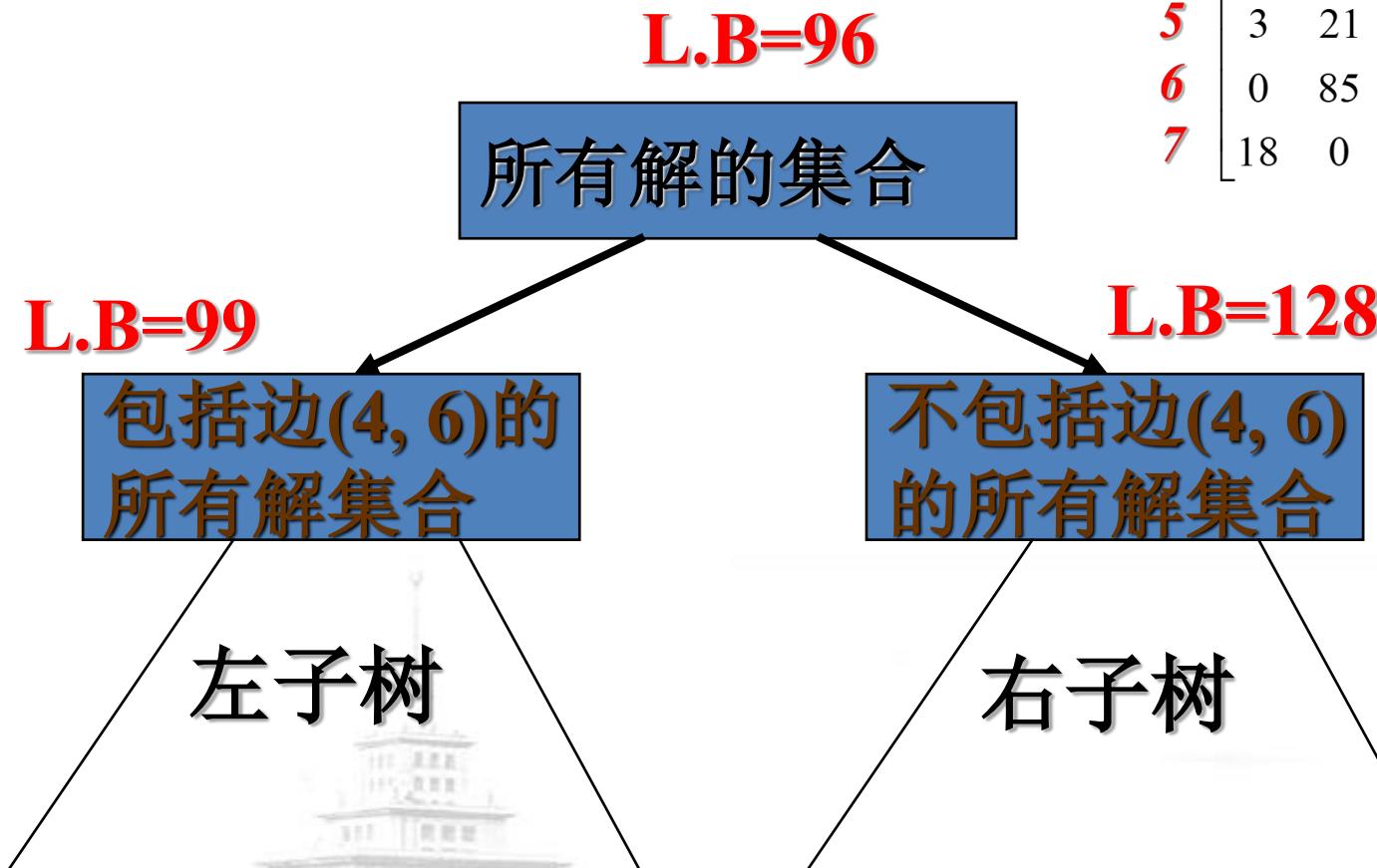
✓ 第4行元素减32

✓ 结果矩阵如下

| | $j = 1$ | 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 |
| 5 | 3 | 21 | 56 | 7 | ∞ | 0 | 28 |
| 6 | 0 | 85 | 8 | 42 | 89 | ∞ | 0 |
| 7 | 18 | 0 | 0 | 0 | 58 | 13 | ∞ |

| <i>j</i> | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|------------|----------|----------|----------|----------|----------|----------|----------|
| <i>i=1</i> | ∞ | 0 | 83 | 9 | 30 | 6 | 50 |
| 2 | 0 | ∞ | 66 | 37 | 17 | 12 | 26 |
| 3 | 29 | 1 | ∞ | 19 | 0 | 12 | 5 |
| 4 | 32 | 83 | 66 | ∞ | 49 | ∞ | 80 |
| 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

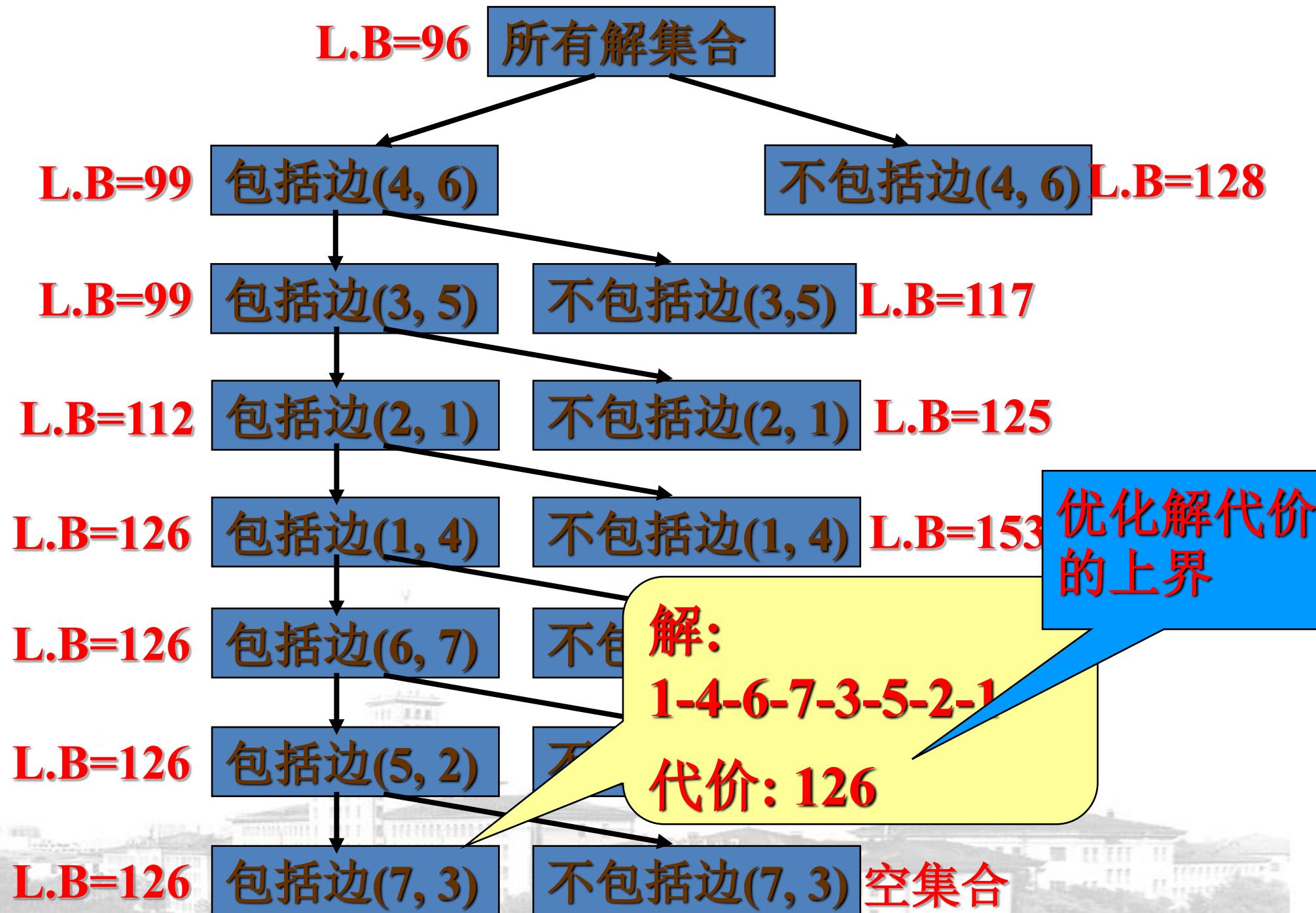


➤ 使用爬山策略扩展左子树根

- ✓ 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

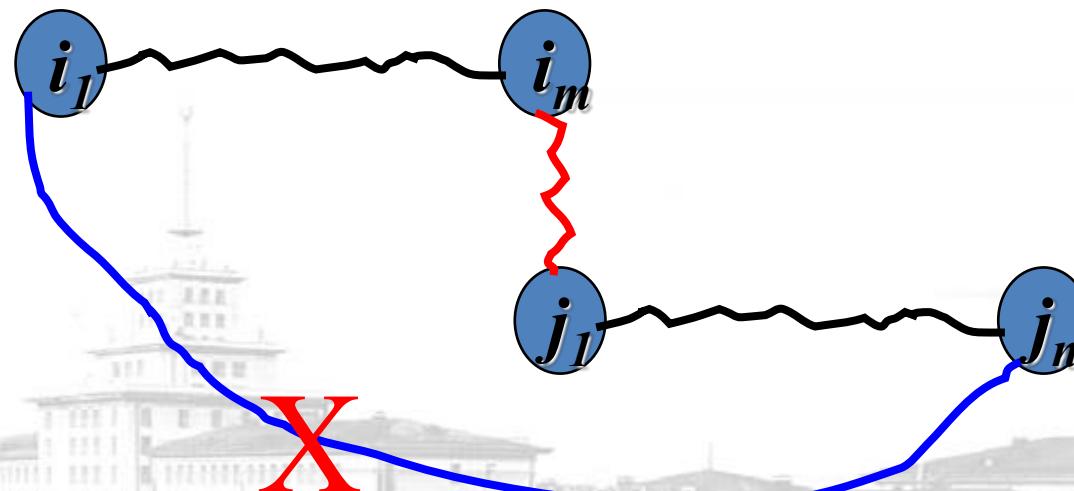
➤ 目前树扩展为:

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



注意

如果 $i_1-i_2-\dots-i_m$ 和 $j_1-j_2-\dots-j_n$ 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.



6.6 A*算法



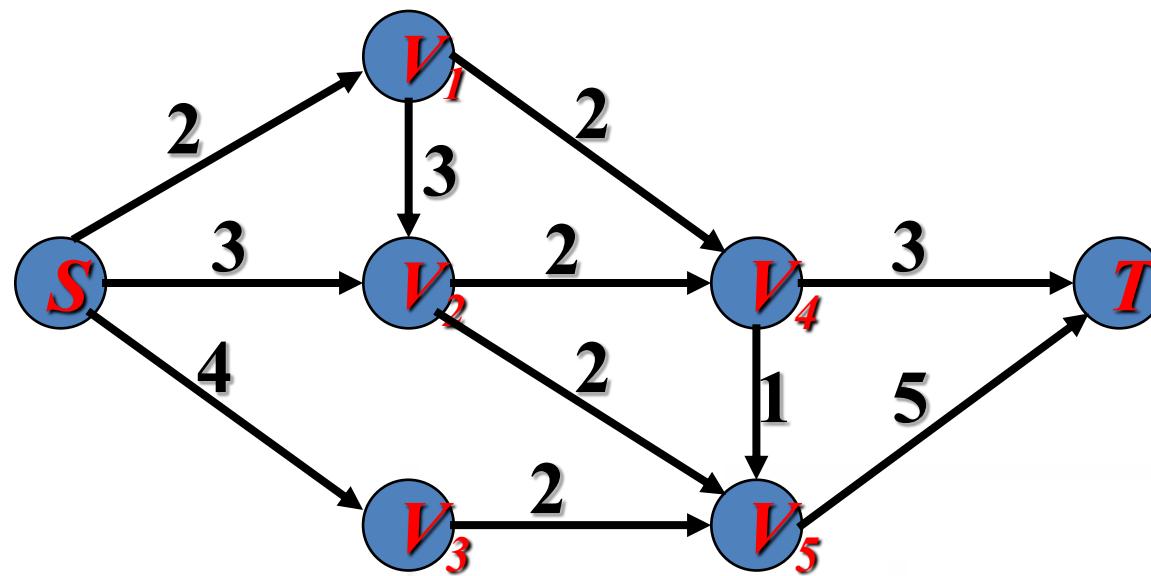
A*算法的基本思想

- 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

- 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
 - $h(n) \leq h^*(n)$ 总为真
 - $f(n) = g(n) + h(n) \leq g(n) + h^*(n) = f^*(n)$ 定义为 n 的代价

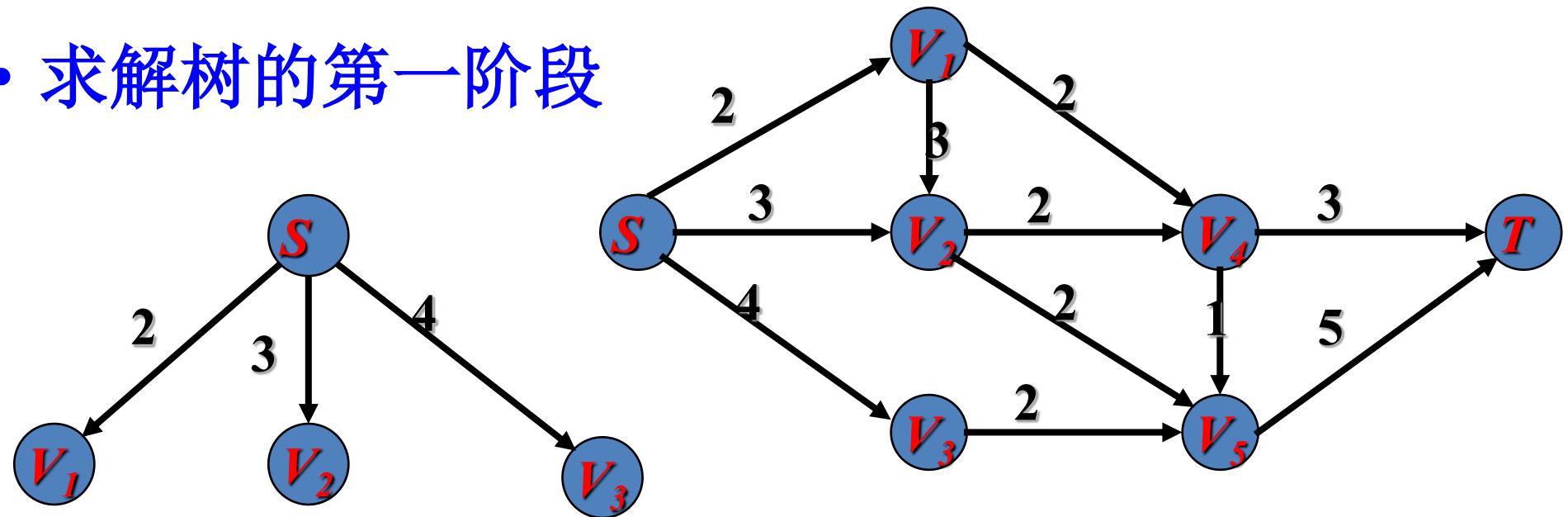
例1. 最短路径问题:

- 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



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

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

• 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 (可为阶段目标节点, 可为最终目标节点) .

证明 $f(t)=g(t)=f^*(t)$ (满足该等式, t 为最终目标节点)是优化解代价.

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

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

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

(4). $f(t)$ 是一个可能解, $f(t) \geq f^*(s)$, $f(t)=f^*(s)$.

也可用反证法解定理1：使用Best-First策略搜索树，如果A*如选择的节点是目标节点，则该节点表示的解是优化解。
证明（反证法）：

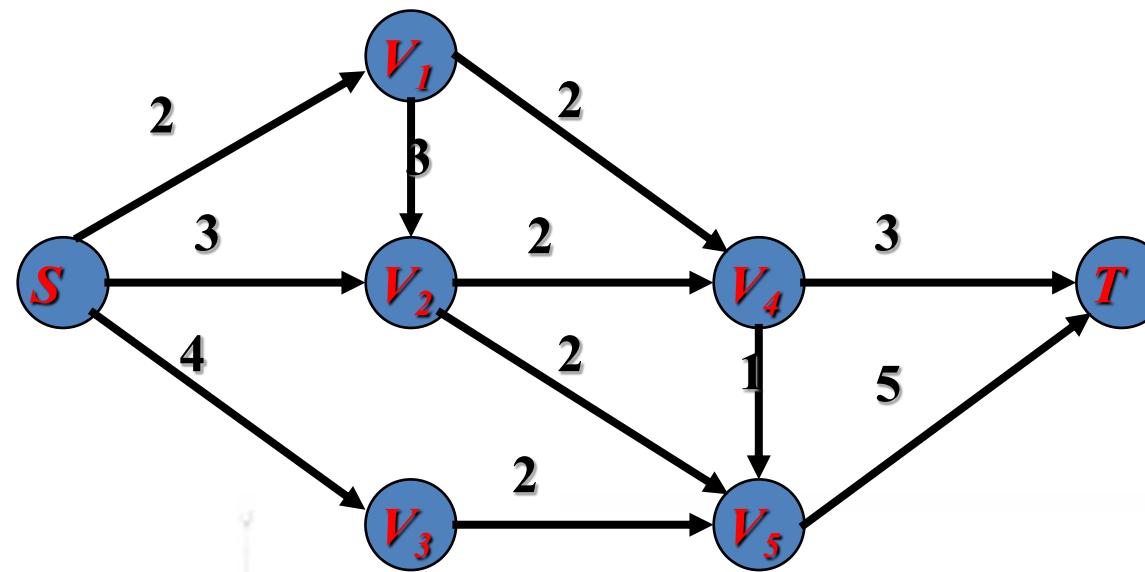
- (1) 假设A*算法首先找到的路径 $P(n_1, n_2, \dots, T_P)$ 来到目标节点，而路径P并非真正的最小代价路径。
- (2) 若真正的最小代价路径 $A(n_1, n_2, \dots, n_p, \dots, T_A)$ ，对于A中任意节点 n_i ，有 $f(n_i) \leq f^*(n_i) = g(T_A) < g(T_P) = f(T_P)$ ，即 $f(n_i) < f(T_P)$
- (3) 按照Best-First算法，如果 $f(n_i) < f(T_P)$ ，那么应该选择扩展 n_i 节点（包括 T_A ）而非 T_P 节点，与Best-First策略相矛盾。故A*算法选择的节点是目标节点，则该节点表示的解是优化解。

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.

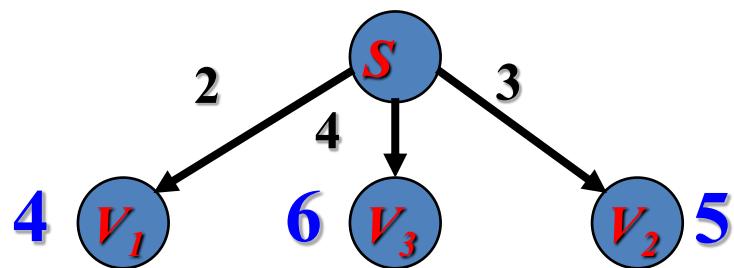
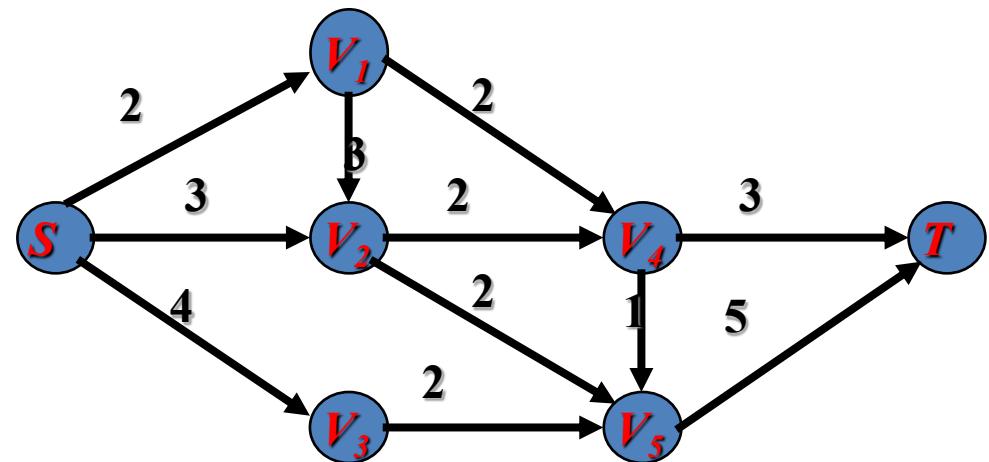
应用A*算法求解最短路径问题

- 问题的输入:



- A*算法的执行全过程

Step 1



$$g(V_1) = 2$$

$$g(V_3) = 4$$

$$g(V_2) = 3$$

$$h(V_1) = \min\{2, 3\} = 2$$

$$h(V_3) = \min\{2\} = 2$$

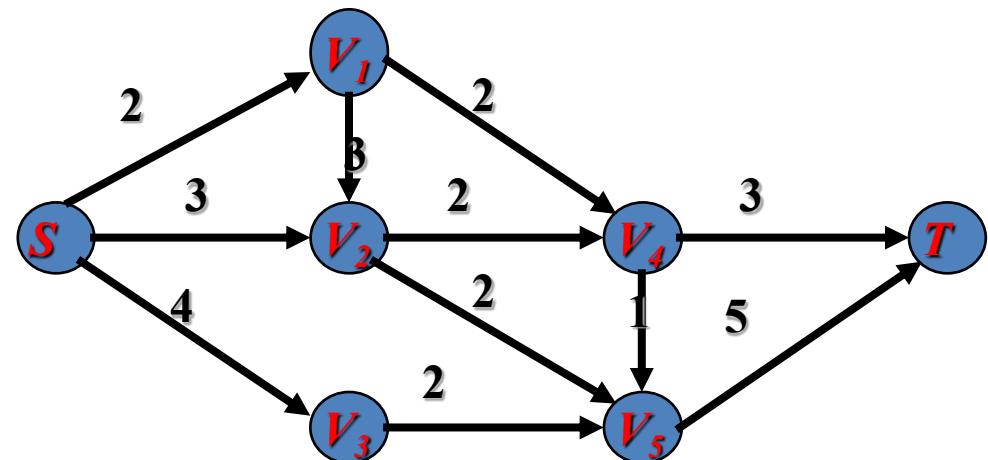
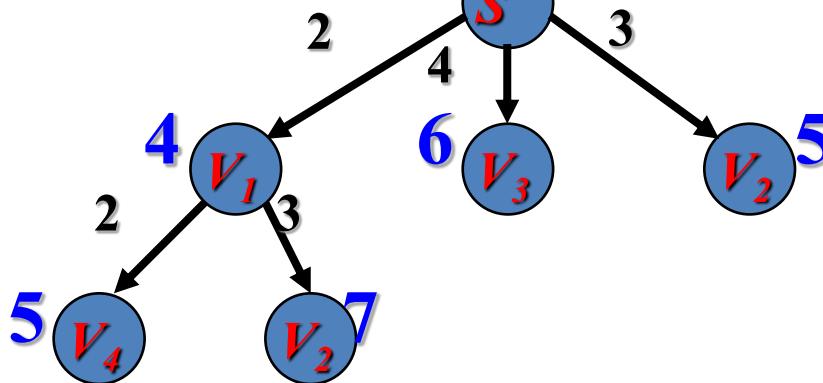
$$h(V_2) = \min\{2, 2\} = 2$$

$$f(V_1) = 2 + 2 = 4$$

$$f(V_3) = 4 + 2 = 6$$

$$f(V_2) = 3 + 2 = 5$$

Step 2. 扩展 V_1



$$g(V_4) = 2 + 2 = 4$$

$$g(V_2) = 2 + 3 = 5$$

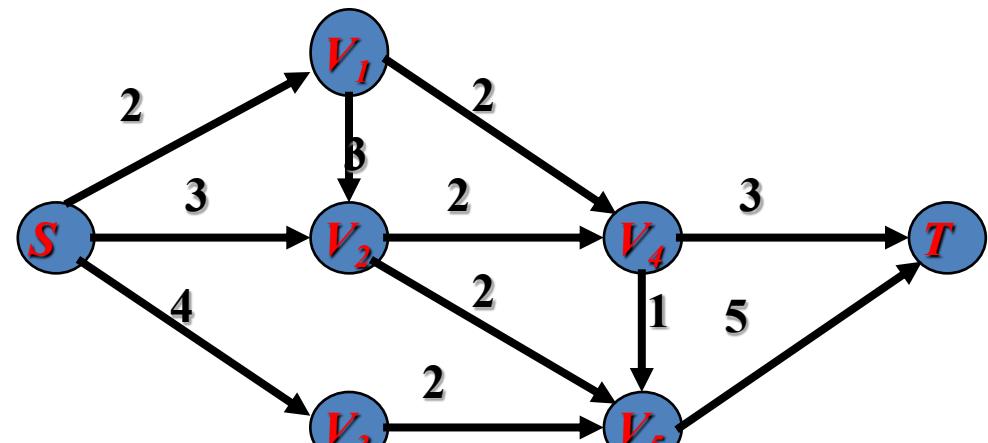
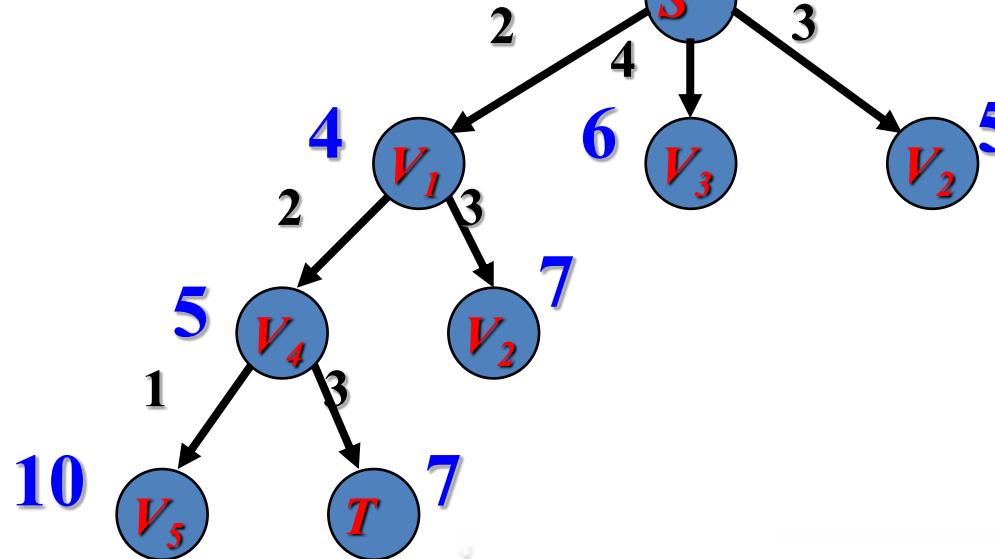
$$h(V_4) = \min\{3, 1\} = 1$$

$$h(V_2) = \min\{2, 2\} = 2$$

$$f(V_4) = 4 + 1 = 5$$

$$f(V_2) = 5 + 2 = 7$$

Step 3. 扩展 V_4



$$g(V_5) = 2 + 2 + 1 = 5$$

$$g(T) = 2 + 2 + 3 = 7$$

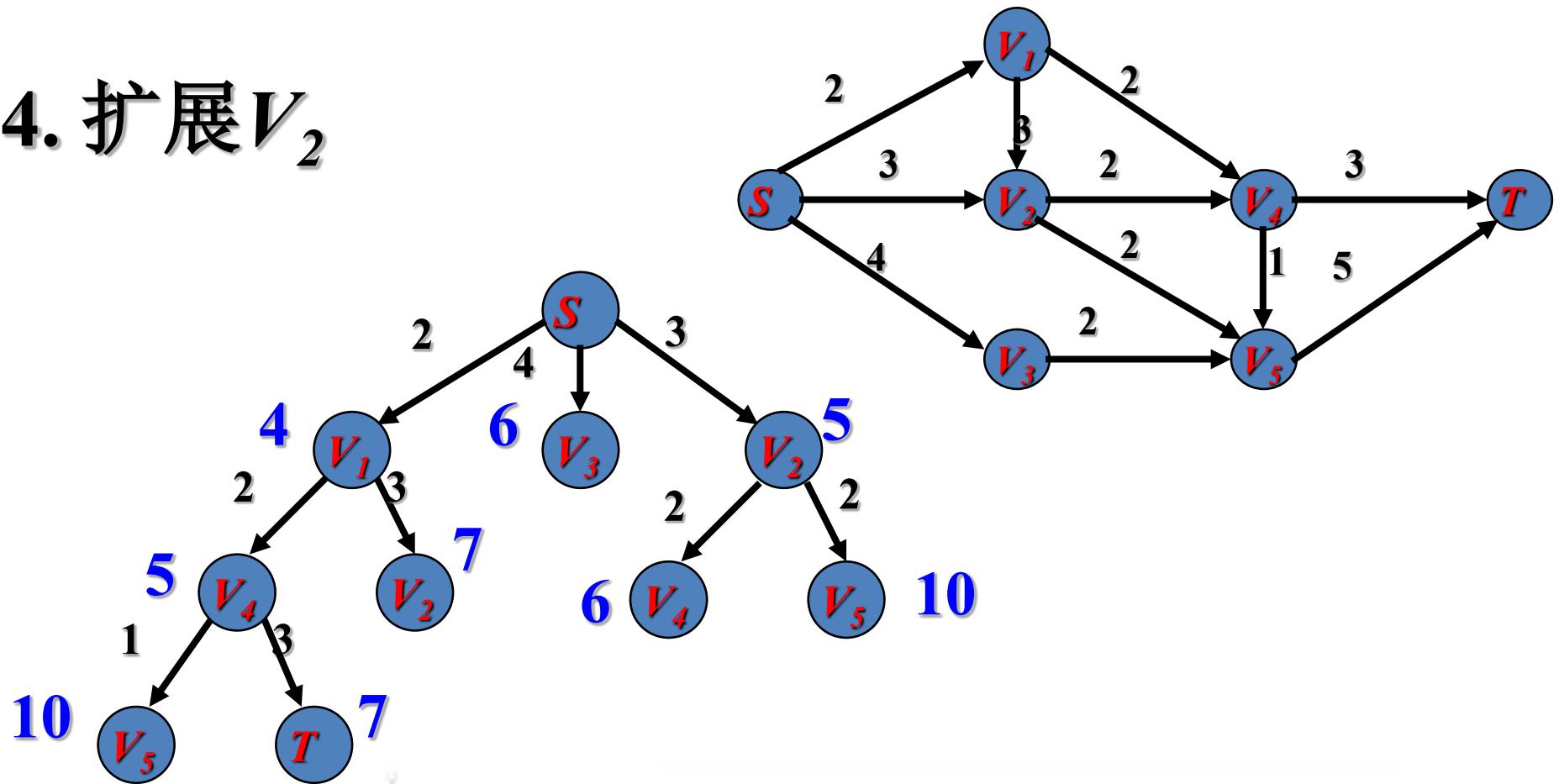
$$h(V_5) = \min\{5\} = 5$$

$$h(T) = 0$$

$$f(V_5) = 5 + 5 = 10$$

$$f(T) = 7 + 0 = 7$$

Step 4. 扩展 V_2



$$g(V_4) = 3 + 2 = 5$$

$$g(V_5) = 3 + 2 = 5$$

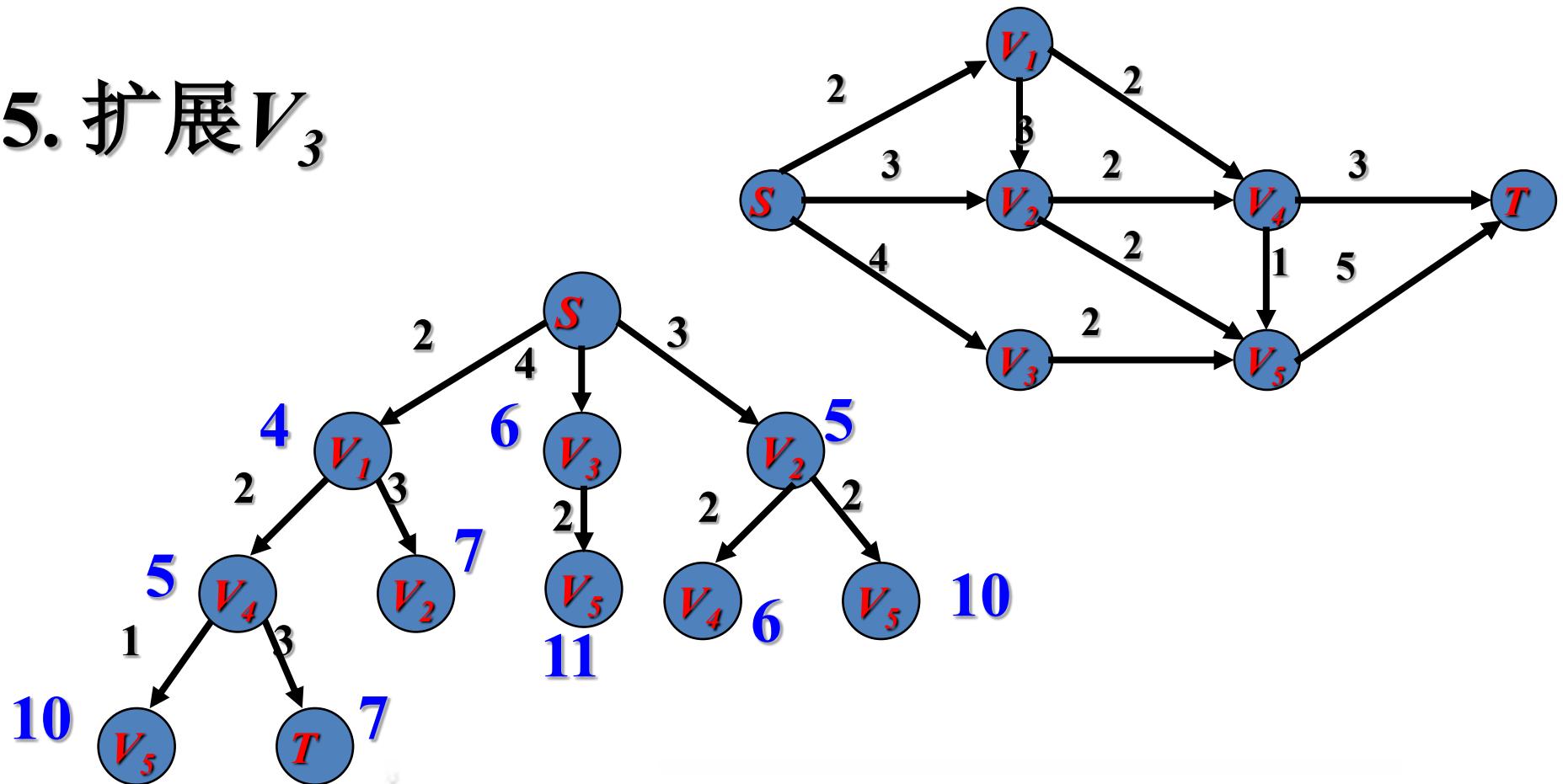
$$h(V_4) = \min\{1, 3\} = 1$$

$$h(V_5) = \min\{5\} = 5$$

$$f(V_4) = 5 + 1 = 6$$

$$f(V_5) = 4 + 5 = 10$$

Step 5. 扩展 V_3

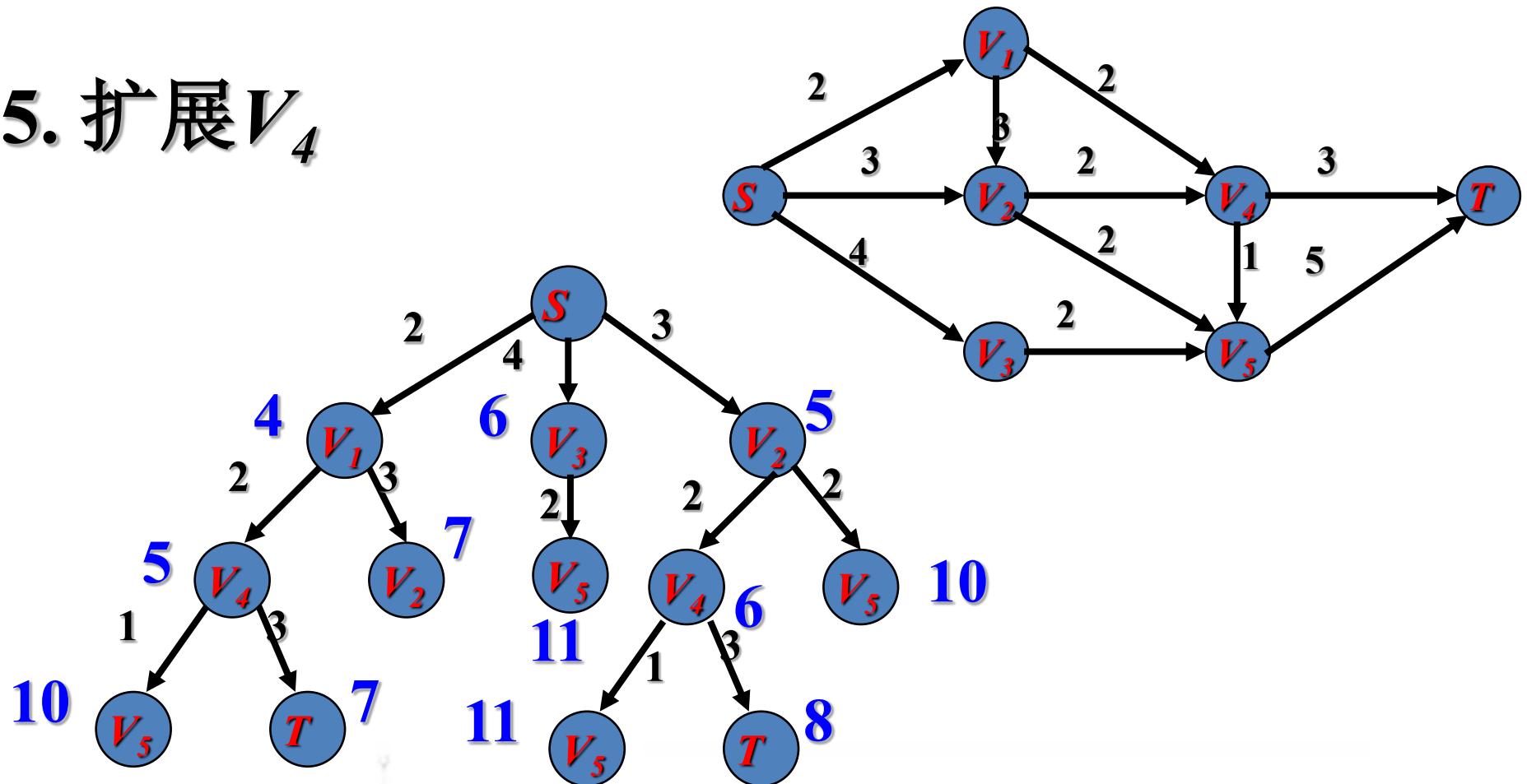


$$g(V_5) = 4 + 2 = 6$$

$$h(V_5) = \min\{5\} = 5$$

$$f(V_5) = 4 + 5 = 11$$

Step 5. 扩展 V_4



$$g(V_5) = 3 + 2 + 1 = 6$$

$$g(T) = 3 + 2 + 3 = 8$$

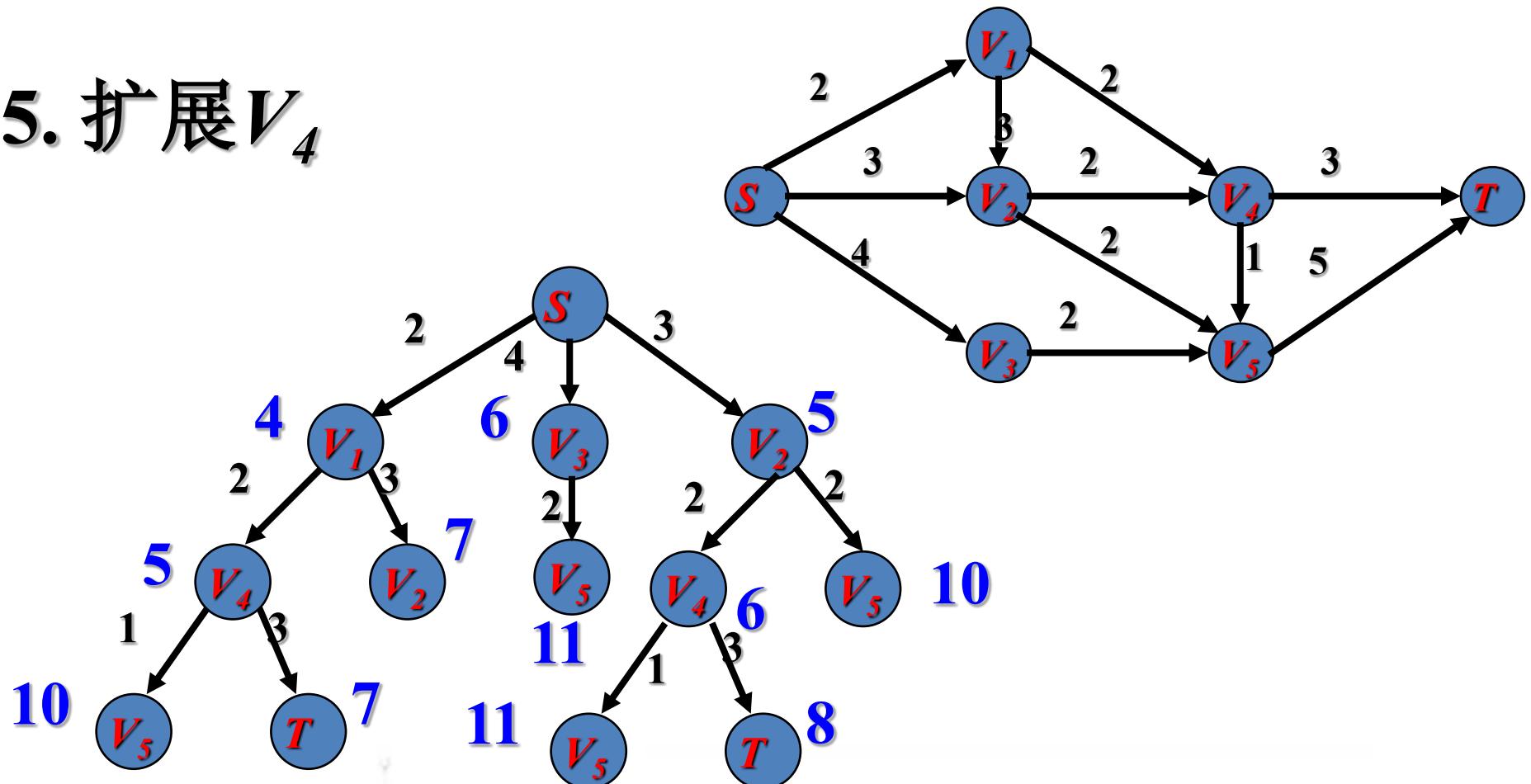
$$h(V_5) = \min\{5\} = 5$$

$$h(T) = 0$$

$$f(V_5) = 6 + 5 = 11$$

$$f(T) = 8 + 0 = 8$$

Step 5. 扩展 V_4



$$g(V_5) = 3 + 2 + 1 = 6$$

$$h(V_5) = \min\{5\} = 5$$

$$f(V_5) = 6 + 5 = 11$$

$g(T) = 0$ 因为 T 是目标节点, 所以我们得到解: $f(T) = 8 + 0 = 8$

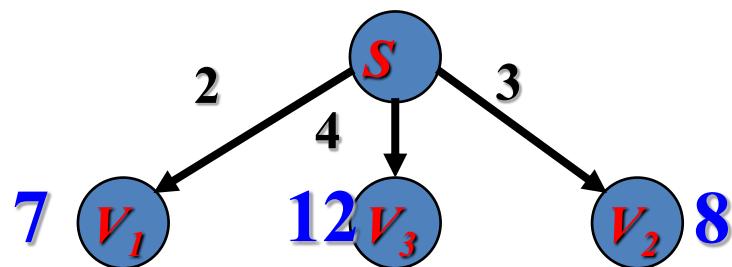
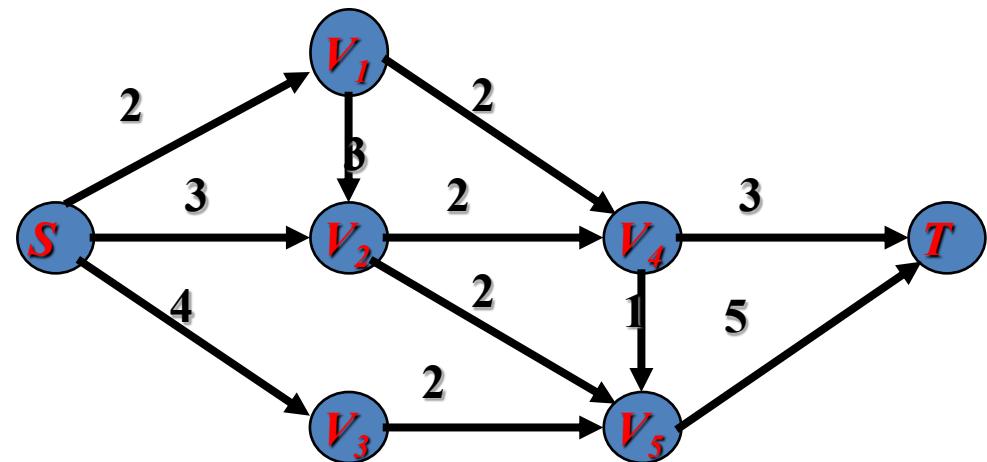
$$S \rightarrow V_1 \rightarrow V_4 \rightarrow T$$

A*算法的启发函数

$h(n) \leq h^*(n)$ 怎么选？

- 必须比从 n 到目标节点的最小代价小
- 在满足条件下的情况下， $h(n)$ 越接近 $h^*(n)$ ， 启发性越好， 能更快找到最优解
- $h(n)$ 离 $h^*(n)$ 越远， 接近0， 启发性越差， 需要搜索的空间越大， 效率越差

Step 1



$$g(V_1) = 2$$

$$g(V_3) = 4$$

$$g(V_2) = 3$$

$$h(V_1) = 5$$

$$h(V_3) = 7$$

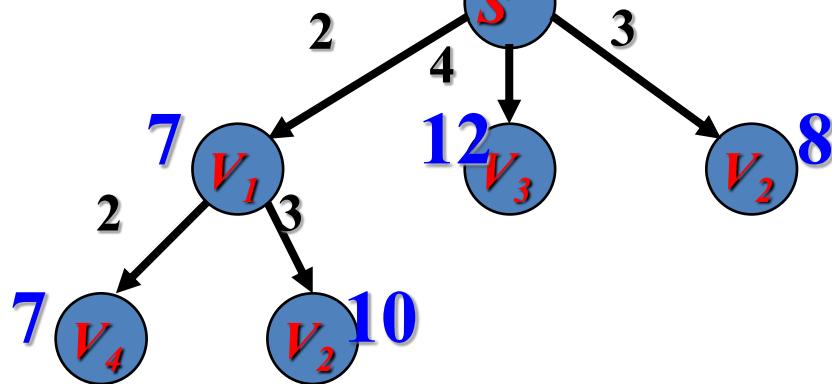
$$h(V_2) = 5$$

$$f(V_1) = 2 + 5 = 7$$

$$f(V_3) = 4 + 7 = 12$$

$$f(V_2) = 3 + 5 = 8$$

Step 2. 扩展 V_1



$$g(V_4) = 2 + 2 = 4$$

$$g(V_2) = 2 + 3 = 5$$

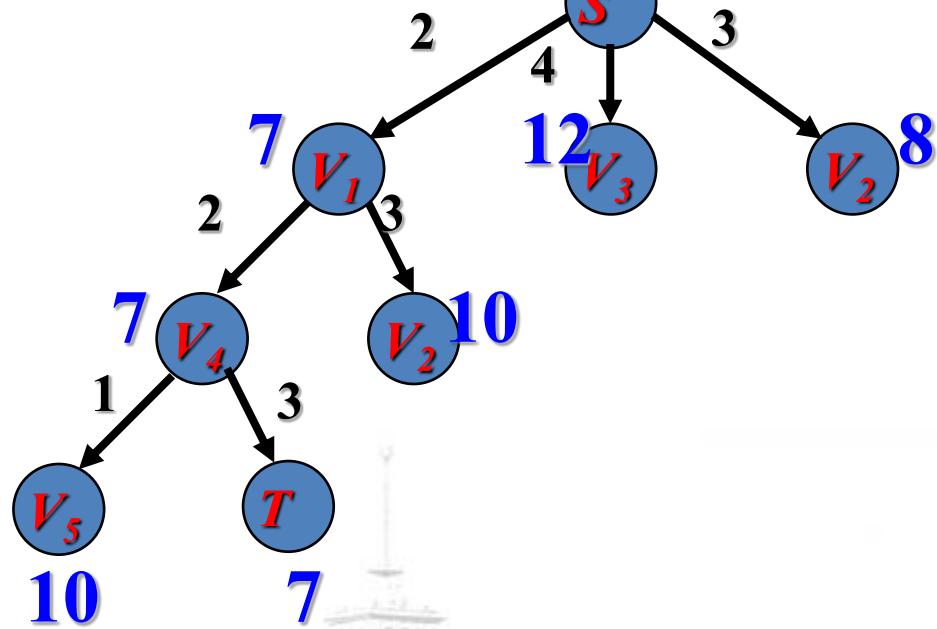
$$h(V_4) = 3$$

$$h(V_2) = 5$$

$$f(V_4) = 4 + 3 = 7$$

$$f(V_2) = 5 + 5 = 10$$

Step 4. 扩展 V_4



$$g(V_5) = 2 + 2 + 1 = 5$$

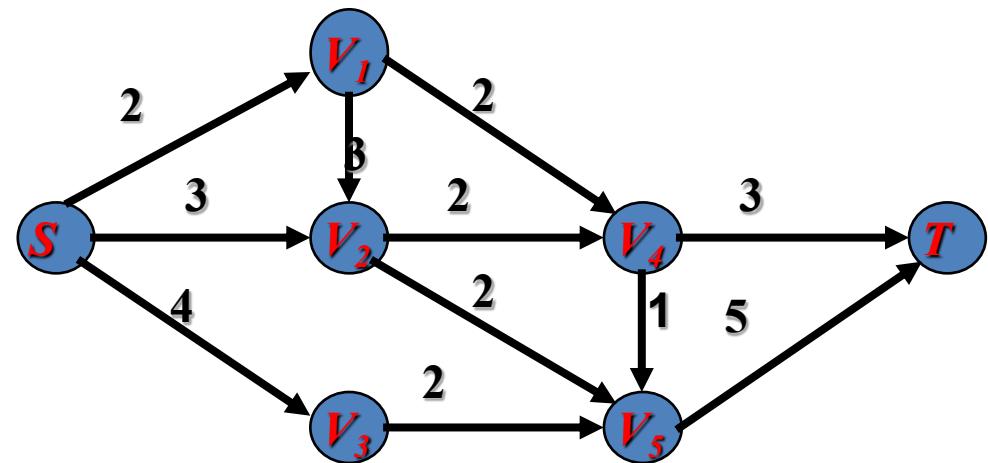
$$g(T) = 2 + 2 + 3 = 7$$

$$h(V_5) = 5$$

$$h(T) = 0$$

$$f(V_5) = 5 + 5 = 10$$

$$f(T) = 7 + 0 = 7$$



思考题3

基因序列可以表示为一条由 8 个字符组成的字符串，其中每个字符都是 'A'、'C'、'G' 和 'T' 之一。假设我们需要调查从基因序列 start 变为 end 所发生的基因变化。一次基因变化就意味着这个基因序列中的一个字符发生了变化。

例如， "AACCGGTT" --> "AACCGGTA" 就是一次基因变化。

另有一个基因库 bank 记录了所有有效的基因变化，只有基因库中的基因才是有效的基因序列。 (变化后的基因必须位于基因库 bank 中)

给你两个基因序列 start 和 end，以及一个基因库 bank，请你找出并返回能够使 start 变化为 end 所需的最少变化次数。如果无法完成此基因变化，返回 -1。 (注意：起始基因序列 start 默认是有效的，但是它并不一定会出现在基因库中。)

示例：

输入： start = "AAAAACCC", end = "AACCCCCC",
bank = ["AAAAACCC", "AAACCCCC", "AACCCCCC"]

输出： 3

