

Lecture 9 树搜索策略

绳伟光

上海交通大学微纳电子学系

2021-11-04



提纲

- 1 树搜索的动机
- 2 典型树搜索策略
- 3 人员分配问题
- 4 旅行商问题
- 5 A^* 算法



提纲

- 1 树搜索的动机
- 2 典型树搜索策略
- 3 人员分配问题
- 4 旅行商问题
- 5 A^* 算法



树搜索的动机

- 相当多的计算问题属于 NPC 问题
- NPC 问题可认为是一类没有有效算法的难解问题
- 相当多的 NPC 问题在规模小时可以穷举解决
- 相当多的 NPC 问题的解结构可以表示为树

当问题的解空间可以组织成树时，穷举问题的求解即变为搜索解空间对应的搜索树！



树搜索问题的解空间

- 树搜索问题的解空间一般表示为一个 n 元组 $\langle x_1, \dots, x_n \rangle$, x_i 的值域为 D_i
- 问题的搜索空间 (解空间) 为笛卡尔乘积空间 $\prod_{i=1}^n D_i$, 大小为 $\Omega(2^n)$
- 显式约束: x_i 值域需满足 D_i
- 隐式约束: 判断 $\langle x_1, \dots, x_n \rangle$ 是否为问题的解
- 搜索树的根节点: $\langle -, \dots, - \rangle$, 第一个子节点: $\langle x_1, \dots, - \rangle$
- 搜索树 k 层节点: $\langle x_1, \dots, x_k, - \dots, - \rangle$, 其 $k+1$ 层子节点: $\langle x_1, \dots, x_k, x_{k+1}, - \dots, - \rangle$



布尔表达式的可满足性

布尔表达式的可满足性问题

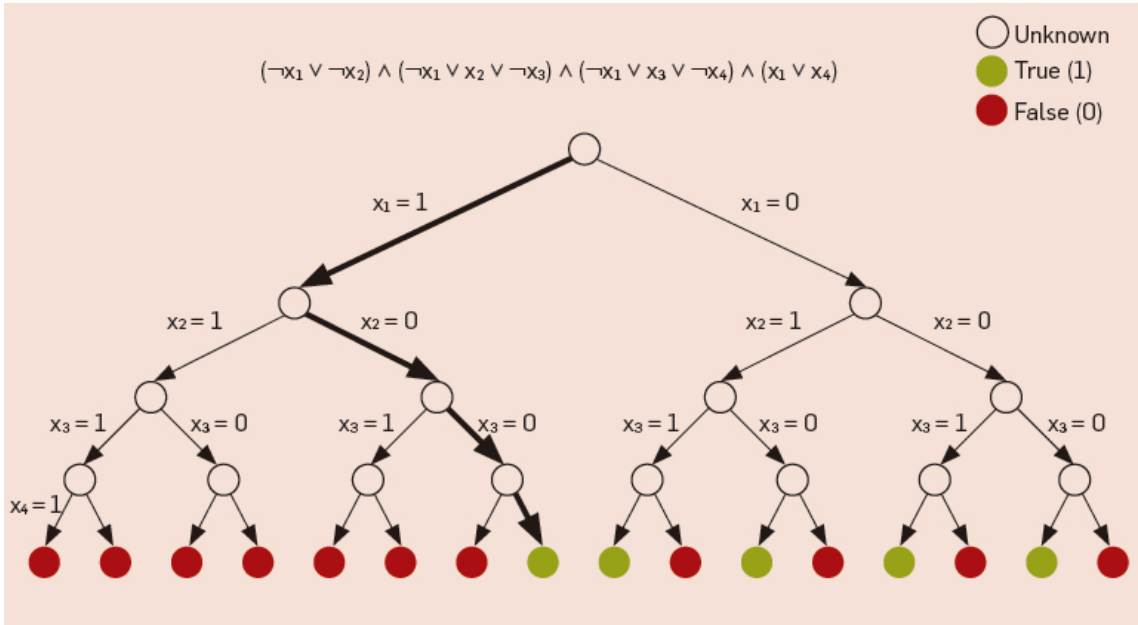
输入： n 个布尔变量 x_1, \dots, x_n 上的 m 个析取表达式 $E_j (1 \leq j \leq m)$ ，其中 $E_j = x_{j_1} \vee \dots \vee x_{j_k}$ ，且所有 x_{j_i} 都是 x_1, \dots, x_n 中某个变量或某个变量的非

输出：变量 x_1, \dots, x_n 的一个赋值使得所有析取表达式 $E_j ((1 \leq j \leq m))$ 取值为真。

- 活节点：以结点 x 为根的子树中可能存在问题的解，称 x 是活结点或扩展节点
- 否则称 x 是死结点



四变量 SAT 问题搜索树示例

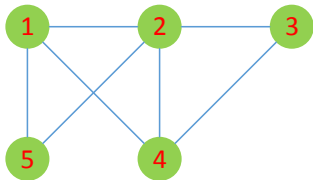


哈密顿环问题

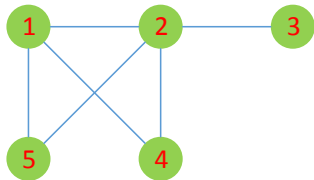
- 哈密顿环问题：给定图 $G = (V, E)$ ，判定是否存在哈密顿环，即从图中任一顶点出发访问其它所有顶点恰好一次后回到出发点
- 哈密顿环问题是 NPC 问题
- TSP 问题是一个变种，求解图中最短的哈密顿环
- 其解可以理解为寻找一个顶点序列满足哈密顿环，可以以树来表示其解空间
- 搜索树的根节点 $\langle v_1, -, \dots, - \rangle$ 表示从任意选定结点 v_1 出发
- 节点 $\langle v_1, \dots, v_k, v_{k+1}, -, \dots, - \rangle$ 是节点 $\langle v_1, \dots, v_k, -, \dots, - \rangle$ 的孩子当且仅当 $v_k v_{k+1} \in E$ 且 $v_{k+1} \notin \{v_1, \dots, v_k\}$
- 不存在孩子的节点是死结点



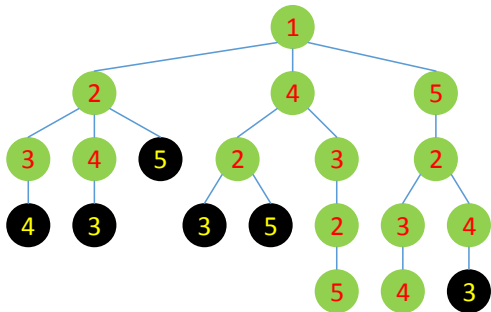
哈密顿环问题解空间的树表示



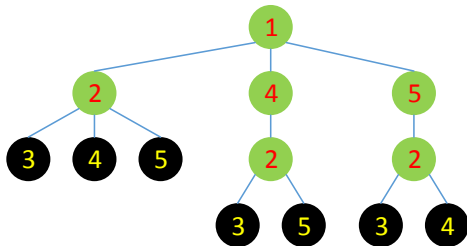
a) 一个存在哈密顿环的图 G_1



b) 一个不存在哈密顿环的图 G_2



c) G_1 的解空间对应的树

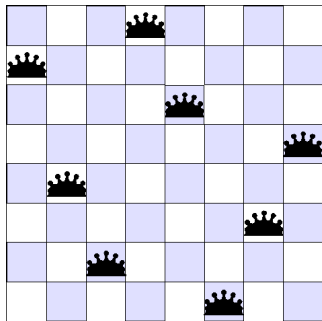


d) G_2 的解空间对应的树

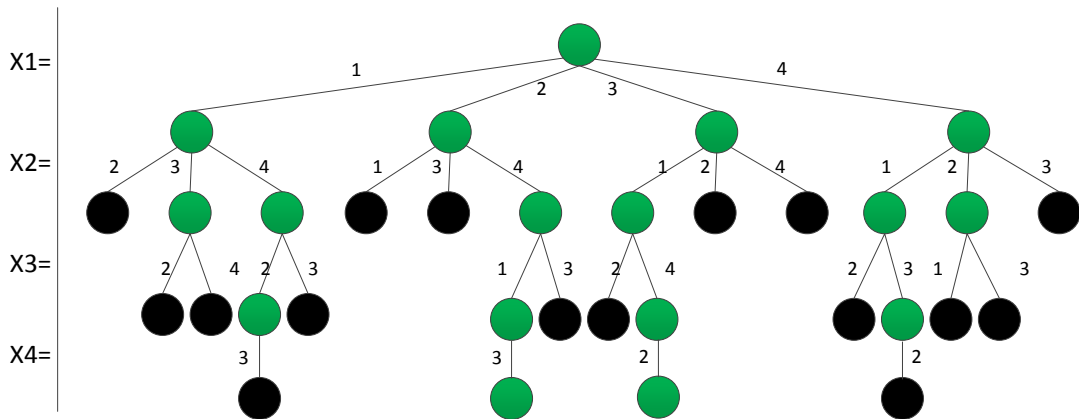


n - 皇后问题

- 在 $n \times n$ 的棋盘上放置 n 个皇后，任意两个皇后不能互相攻击 (处于同行、同列或同一条对角线)
- n - 皇后问题的解可以表示为 n 元组 $\langle x_1, x_2, \dots, x_n \rangle$ ，其中 x_i 表示第 i 行的皇后置于第 x_i 列，下图 8 - 皇后问题的解可表示为: $\langle 4, 1, 5, 8, 2, 7, 3, 6 \rangle$
- 搜索树的根节点 $\langle -, -, \dots, - \rangle$ ，其某个子节点为 $\langle x_1, -, \dots, - \rangle$
- 任意节点 $\langle x_1, \dots, x_k, -, \dots, - \rangle$ 的子节点为 $\langle x_1, \dots, x_k, x_{k+1}, -, \dots, - \rangle$ ，且 $x_{k+1} \notin \{x_1, \dots, x_k\}$



4 - 皇后问题的解空间树表示



8- 魔方问题

- 8- 魔方问题的每次操作空白方格会和同行或同列方格交换
- 8- 魔方问题要求判定是否能从任意格局变换为目标格局

2	3	
5	1	4
6	8	7

A) 起始格局

1	2	3
8		4
7	6	5

B) 目标格局

2	3	
5	1	4
6	8	7

2		3
5	1	4
6	8	7

2	3	4
5	1	
6	8	7

C) 格局之间的父子关系



提纲

- 1 树搜索的动机
- 2 典型树搜索策略
- 3 人员分配问题
- 4 旅行商问题
- 5 A^* 算法

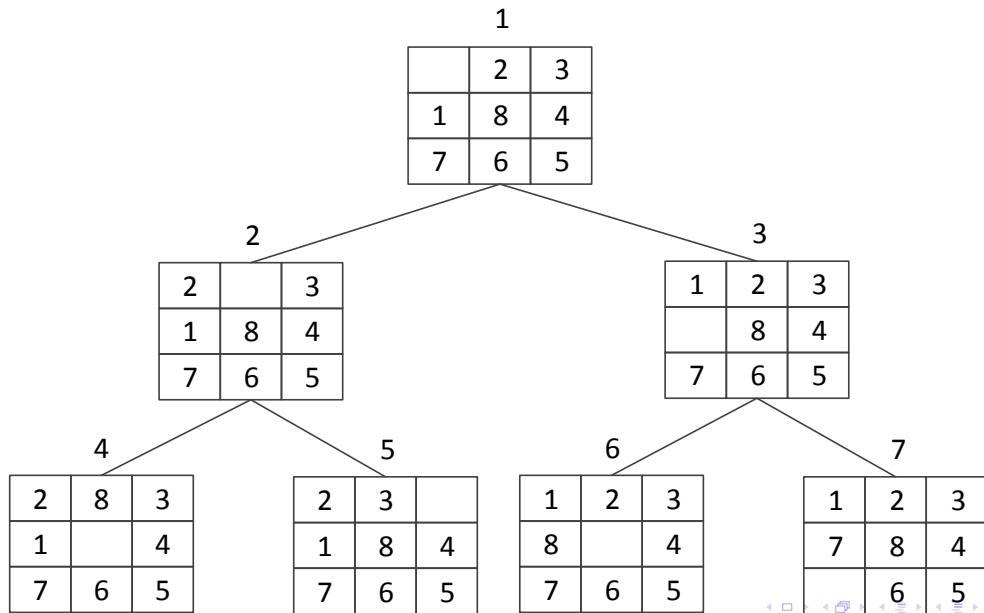


广度优先搜索 BFS

- 构造仅含树根节点的队列 Q
- 如果 Q 的第一个节点 x 是目标节点，输出节点 x 对应的解，算法结束
- 删除队列 Q 的第一个节点 x ，如果以 x 为根的子树可能存在解，将 x 的所有子节点加入队列 Q 的末尾
- 如果队列 Q 为空，则问题无解，算法结束，否则，转到第 2 步



8 - 魔方的广度优先搜索



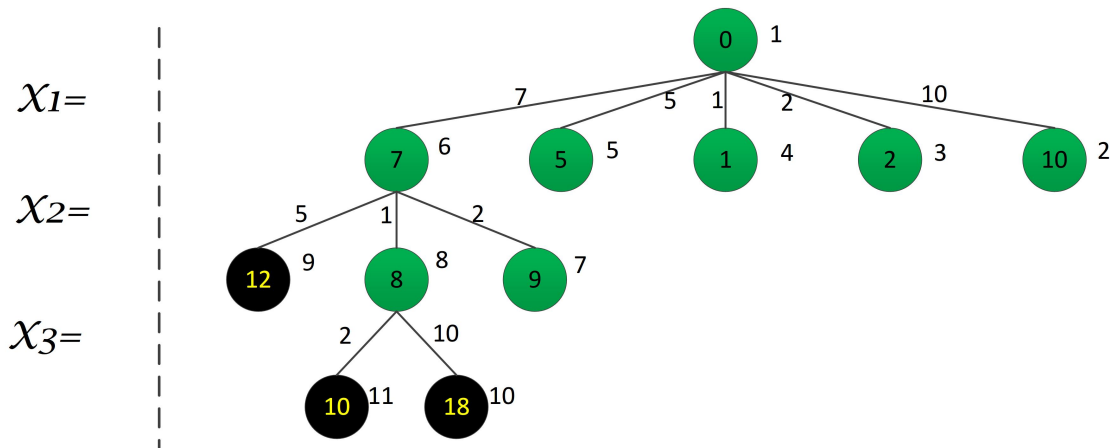
深度优先搜索 DFS

- 构造仅含树根节点的栈 S
- 如果栈顶元素 x 是目标节点，则输出 x 对应的解，算法结束
- 弹出栈顶元素 x ，如果以 x 为根的子树可能存在解，则将 x 的所有子节点依次压入栈顶
- 如果栈 S 为空，问题无解，算法结束，否则，转第 2 步

无论广度还是深度优先搜索，最坏情况下都要访问搜索树中的所有节点！



子集和问题的深度优先搜索



- 子集和问题 $S = \{7, 5, 1, 2, 10\}$, $K = 9$
- 上图圈中数值为子集的和，边值为扩展的元素，圈右边数值为入栈的次序



爬山法

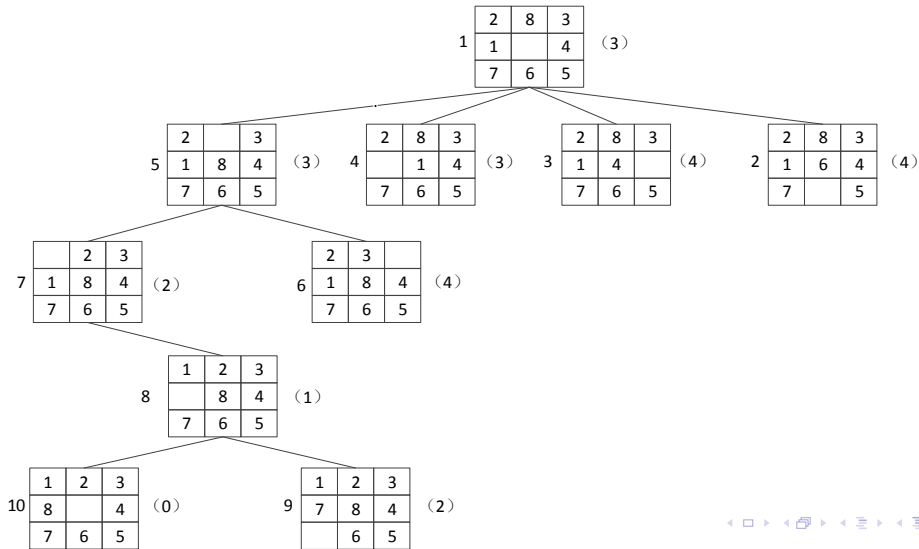
- 构造仅含树根节点的栈 S
- 如果栈顶元素 x 是目标节点，则输出 x 对应的解，算法结束
- 弹出栈顶元素 x ，如果以 x 为根的子树可能存在解，则将 x 的所有子节点按距离目标值由远至近的顺序依次压入栈顶
- 如果栈 S 为空，问题无解，算法结束，否则，转第 2 步

爬山法通过在深度优先搜索过程中每次扩展离目标更近的子节点提高搜索的速度！



8 - 魔方问题的爬山法搜索

以与目标格局不一致的方格数来表示距离目标的距离！



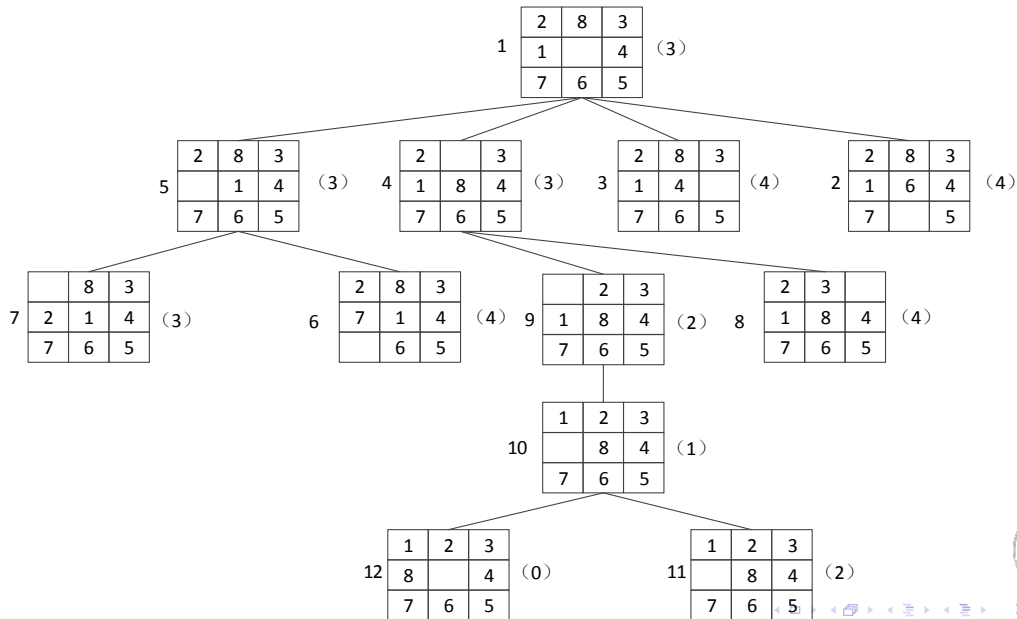
最佳优先搜索

最佳优先搜索从全局范围内选择距离目标最近的节点进行扩展，爬山法则以 DFS 为基础！

- 构造仅含根节点的最小堆 Q
- 如果堆顶元素 x 是目标节点，则输出节点 x 对应的解，算法结束
- 删除堆顶元素 x ，如果以 x 为根的子树可能存在解，将 x 的所有子节点插入堆
- 如果堆 Q 为空，则问题无解，算法结束，否则，转第 2 步



8 - 魔方问题的最佳优先搜索

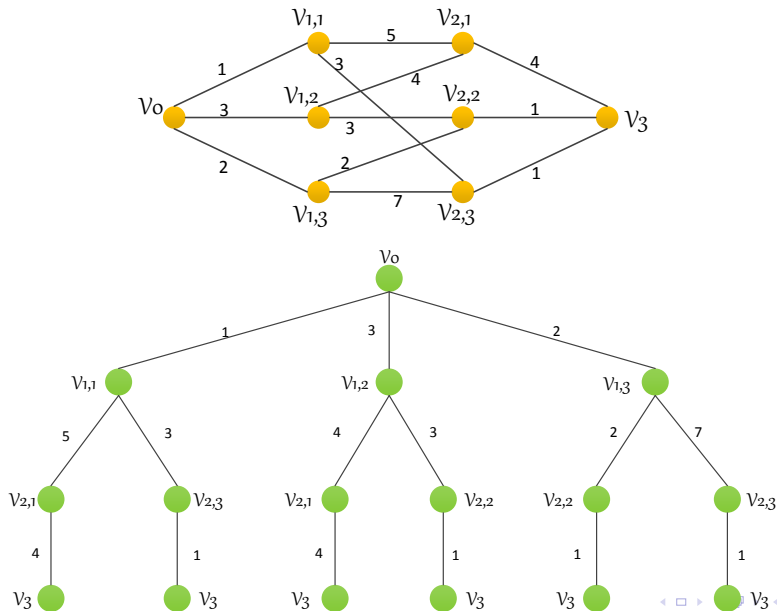


分支限界法

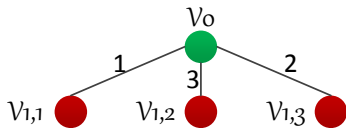
- 与前面算法不同，分支限界法主要用于求解优化问题
- 分支限界法利用已经发现的可行解的代价剪除不能取得优化解的分支，避免对不能产生优化解的分支进行搜索
- 分支限界法用于求解最小化问题，求解最大化问题需要适当变形
- 分支限界法首先要求找到一个解，再用找到的解去剪枝

应用：在 CGO2020 的论文 “Optimizing occupancy and ILP on the GPU using a combinatorial approach” 中，设计了基于分支限界法的调度算法，可针对 GPU 同时优化占用率和 ILP 指标

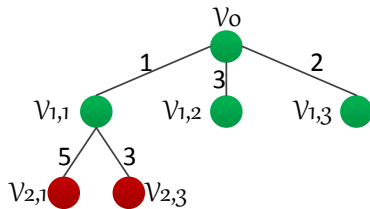
最短路径问题及其解空间树



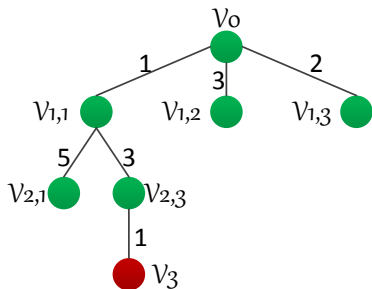
最短路径问题的分支限界搜索



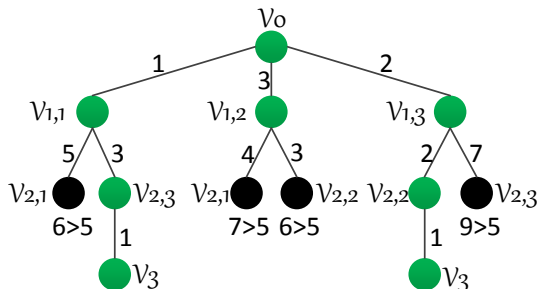
a) 爬山法第一次扩展



b) 爬山法第二次扩展



c) 爬山法第三次扩展得到可行解



d) 利用可行解进行剪枝搜索



提纲

- 1 树搜索的动机
- 2 典型树搜索策略
- 3 人员分配问题
- 4 旅行商问题
- 5 A^* 算法



偏序关系

定义 1 (非严格偏序/自反偏序)

给定集合 S , \leq 是 S 上的二元关系, 若 \leq 满足:

- 1) 自反性: $\forall a \in S$, 有 $a \leq a$
- 2) 反对称性: $\forall a, b \in S$, $a \leq b$ 且 $b \leq a$, 则 $a = b$
- 3) 传递性: $\forall a, b, c \in S$, $a \leq b$ 且 $b \leq c$, 则 $a \leq c$

定义 2 (严格偏序/反自反偏序)

给定集合 S , $<$ 是 S 上的二元关系, 若 $<$ 满足:

- 1) 反自反性: $\forall a \in S$, 有 $a \not\leq a$
- 2) 非对称性: $\forall a, b \in S$, $a < b \implies b \not< a$
- 3) 传递性: $\forall a, b, c \in S$, $a < b$ 且 $b < c$, 则 $a < c$

偏序不要求集合内任意元素间的相互可比较性, 只要求部分排序!

人员分配问题

人员分配问题

- 输入：人员集合 $P = \{P_1, P_2, \dots, P_n | P_1 < P_2 < \dots < P_n\}$ ，工作集合 $J = \{J_1, J_2, \dots, J_n\}$ 及其上的偏序关系 \leq ，代价矩阵 $C = (c_{ij})_{n \times n}$ ， c_{ij} 表示 J_j 分配给 P_i 的代价
- 输出：矩阵 $(x_{ij})_{n \times n}$ ，使得如下三个条件成立且 $\sum_{i,j} x_{ij} \cdot c_{ij}$ 最小：
 - 1) $x_{ij} \in \{0, 1\}$ ， $x_{ij} = 1$ 表示将 J_j 分配给 P_i ，0 表示未分配
 - 2) $\sum_j x_{kj} = 1$ 且 $\sum_i x_{ik} = 1$ 对 $\forall 1 \leq k \leq n$ 成立，即每人只分配一项工作，每项工作仅分配给一人
 - 3) 若 $x_{is} = 1$ ， $x_{jt} = 1$ ，且 $J_s \leq J_t$ ，则 $P_i < P_j$

人员分配问题 (续)

人员分配问题需注意：首先人员需要有序，如果第 s 项工作分配给第 i 个人员，第 t 项工作分配给第 j 个人员且 $J_s \leq J_t$ ，则需要 $P_i < P_j$ ！

人员分配问题等效为寻找一个工作的优化排序 $J_{k_1}, J_{k_2}, \dots, J_{k_n}$ ，使得人员分配方案 $P_1 \rightarrow J_{k_1}, P_2 \rightarrow J_{k_2}, \dots, P_n \rightarrow J_{k_n}$ 的代价最低。这是一个 NP - 完全问题！

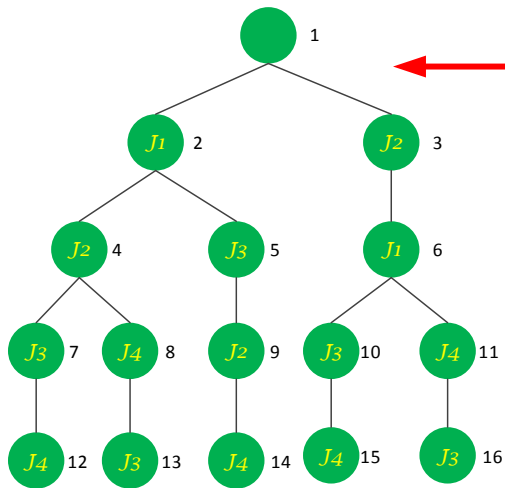


人员分配问题解空间的树表示

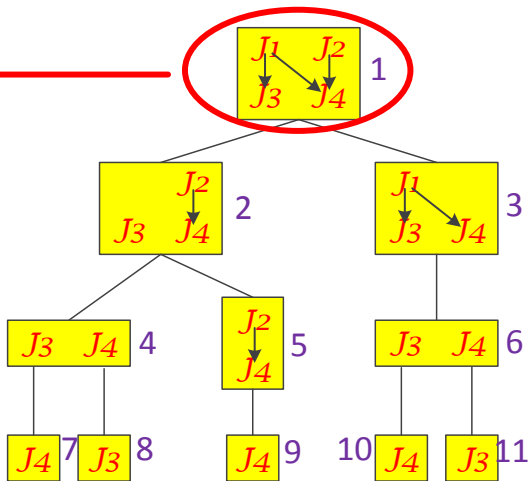
- 搜索树中位于第 k 层的每个节点用 k 元组 $\langle j_1, \dots, j_k \rangle$ 表示, 其中 $k \in [1, n]$
- 第 $k+1$ 层的节点 $\langle j_1, \dots, j_k, j_{k+1} \rangle$ 是节点 $\langle j_1, \dots, j_k \rangle$ 的子节点, 当且仅当 $j_{k+1} \in \mathcal{I} \setminus \{j_1, \dots, j_k\}$ 且不存在 $j_j \in \mathcal{I} \setminus \{j_1, \dots, j_k\}$ 使得 $j_j \neq j_{k+1}$ 且 $j_j \leq j_{k+1}$, 即不存在处于 j_k 和 j_{k+1} 之间的任务



人员分配问题解空间的树表示 (续)



a) 所有拓扑排序的树表示

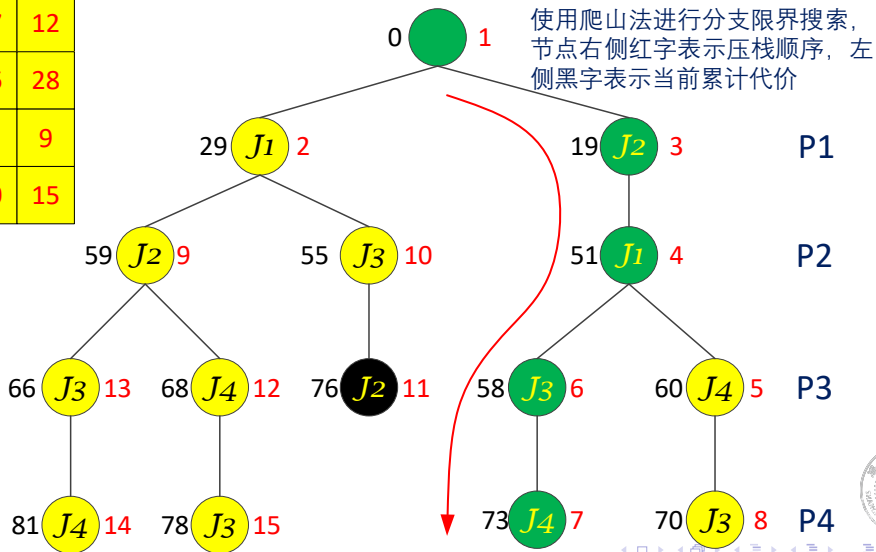


b) 偏序关系的层次分解



人员分配问题的分支限界求解

C P e r s o n	Job →			
	29	19	17	12
	32	30	26	28
	3	21	7	9
	18	13	10	15



分支限界算法的优化

引理 1

给定人员分配问题的代价矩阵 $C = (c_{ij})_{n \times n}$, 将 C 的第 k 行 (列) 减去常数 a 得矩阵 $C' = (c'_{ij})_{n \times n}$, $X = (x_{ij})_{n \times n}$ 是问题的一个可行解, 则 X 在 C 下是最优解当且仅当 X 在 C' 下是最优解。

证 X 在 C 下的代价为 $cost(X, C) = \sum_{i,j} x_{ij} \cdot c_{ij}$, 同理

$$\begin{aligned} cost(X, C') &= \sum_{i,j} x_{ij} \cdot c'_{ij} \\ &= \sum_{i,j, i \neq k} x_{ij} \cdot c'_{ij} + \sum_j x_{kj} \cdot c'_{kj} \\ &= \sum_{i,j, i \neq k} x_{ij} \cdot c_{ij} + \sum_j x_{kj} \cdot (c_{kj} - a) \\ &= \sum_{i,j} x_{ij} \cdot c_{ij} - a \sum_j x_{kj} \\ &= cost(X, C) - a \quad \left(\sum_j x_{kj} = 1 \right) \end{aligned}$$



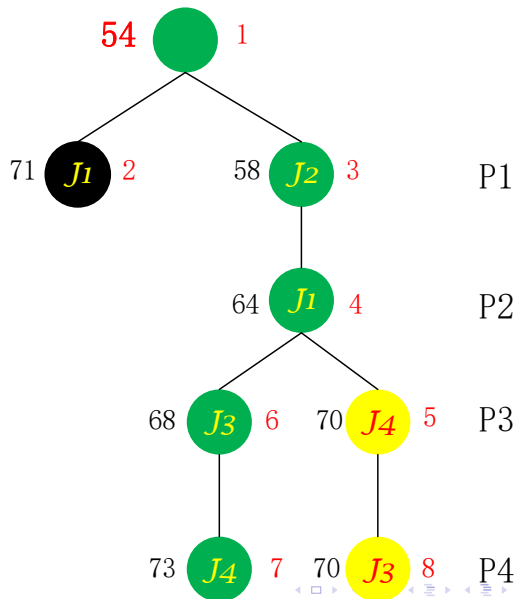
优化后的人员分配问题分支限界搜索

$$C = \begin{bmatrix} 29 & 19 & 17 & 12 \\ 32 & 30 & 26 & 28 \\ 3 & 21 & 7 & 9 \\ 18 & 13 & 10 & 15 \end{bmatrix} \begin{matrix} -12 \\ -26 \\ -3 \\ -10 \end{matrix}$$

-3

$$a = 12 + 26 + 3 + 10 + 3 = 54$$

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



提纲

- 1 树搜索的动机
- 2 典型树搜索策略
- 3 人员分配问题
- 4 旅行商问题
- 5 A^* 算法



旅行商问题

- 给定加权有向图 $G = (V, E)$ ，边上的加权函数为 $W : E \rightarrow R+$
- 从图 G 中任意顶点出发，沿 G 中的边访问所有顶点恰一次，回到出发点，所经过的路径称为 G 的一个哈密顿环
- 哈密顿环的代价是环中所有边的权值之和
- 旅行商问题要求在加权完全有向图上找出代价最小的哈密顿环
- 旅行商问题是一个著名的 NP —完全问题



旅行商问题解空间的树表示

- 在旅行商问题解空间的树表示中，根节点表示所有的可行解，每个节点是 G 的一个邻接矩阵表示 (或变形)
- 通过节点的邻接矩阵可以计算出任意后代节点表示的可行解的代价的一个下界
- 搜索方法为：利用爬山法优先扩展当前节点的子节点中下界最小的节点，直到发现一个可行解
- 剪枝策略：利用找出的可行解的代价，继续搜索其它节点，剪除不可能产生优化解的分支，直到搜索树中不再有可扩展节点



TSP 问题邻接矩阵表示及解代价下界计算

i \ j	1	2	3	4	5	6	7	
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



i \ j	1	2	3	4	5	6	7
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	∞

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



TSP 问题搜索树子节点的扩展方法

- 扩展方法为利用图中的一条边 (i, j) 将所有可行解分为两组，使用边 (i, j) 的可行解构成一组，不使用 (i, j) 的可行解构成一组
- 两个分组分别对应根节点的左右子节点
- 为提高分支限界搜索的剪枝能力，要求：左子节点的代价下界缓慢增长，右子节点代价快速增长
- 如果令 $c_{ij} = 0$ ，则左子节点下界增长 0
- 此时右侧分支不使用边 (i, j) ，则其必使用一条从 i 出发的边和一条进入顶点 j 的边，则右侧分支下界至少增长： $f(i, j) = \min_{k \neq i} c_{kj} + \min_{k \neq j} c_{ik}$
- 故此，选取当前邻接矩阵中权值为 0 且 $f(i, j)$ 达到最大值的边 (i, j) 划分解空间



TSP 问题解空间的划分边的选择

$i \backslash j$	1	2	3	4	5	6	7	
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

→


$i \backslash j$	1	2	3	4	5	6	7	
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	∞	

-7
-1
-4

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

由右侧邻接矩阵, $f(1, 2) = 6 + 0 = 6$, $f(2, 1) = 12 + 0 = 12$, $f(3, 5) = 17 + 1 = 18$, $f(4, 6) = 32 + 0 = 32$, $f(5, 6) = 3 + 0 = 3 \dots$, 最大为 $f(4, 6)$, 含边 $f(4, 6)$ 的可行解下界 96, 不含边 $f(4, 6)$ 的下界为 $96 + 32 = 128$!

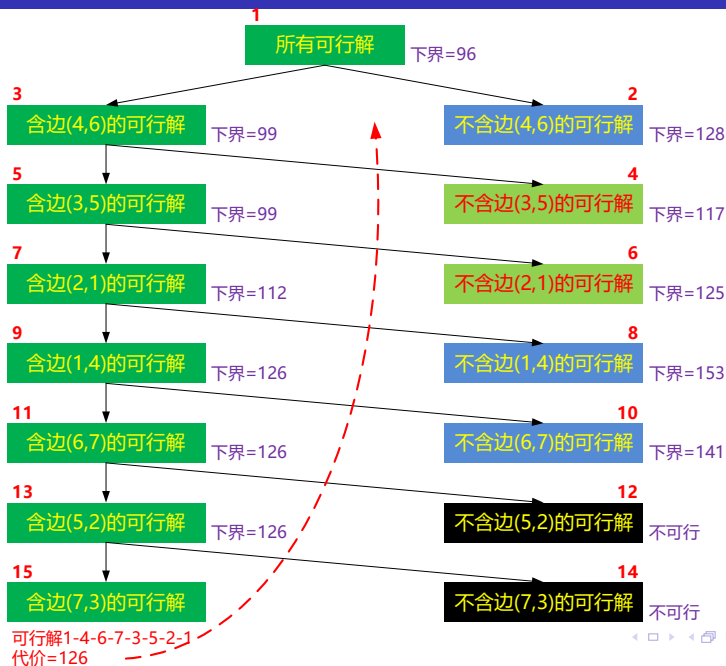
TSP 问题邻接矩阵的更新

$i \setminus j$	1	2	3	4	5	6	7		$i \setminus j$	1	2	3	4	5	6	7
1	∞	0	83	9	30	∞	50		1	∞	0	83	9	30	∞	50
2	0	∞	66	37	17	∞	26		2	0	∞	66	37	17	∞	26
3	29	1	∞	19	0	∞	5		3	29	1	∞	19	0	∞	5
4	∞	∞	∞	∞	∞	∞	∞		4	∞	∞	∞	∞	∞	∞	∞
5	3	21	56	7	∞	∞	28		5	0	18	53	4	∞	∞	25
6	0	85	8	∞	89	∞	0		6	0	85	8	∞	89	∞	0
7	18	0	0	0	58	∞	∞		7	18	0	0	0	58	∞	∞

$$LB=96+3=99$$

对左侧分支，由于任意可行解要使用边 (4,6)，所以所有从 4 出发，所有进入 6 的边均不再使用，设为 ∞ ；同理边 (6,4) 也不再使用，设为 ∞ ；接下来需将不含 0 的行 (第 5 行) 继续变换，下界变为 99！

TSP 问题的分支限界搜索



提纲

- 1 树搜索的动机
- 2 典型树搜索策略
- 3 人员分配问题
- 4 旅行商问题
- 5 A^* 算法

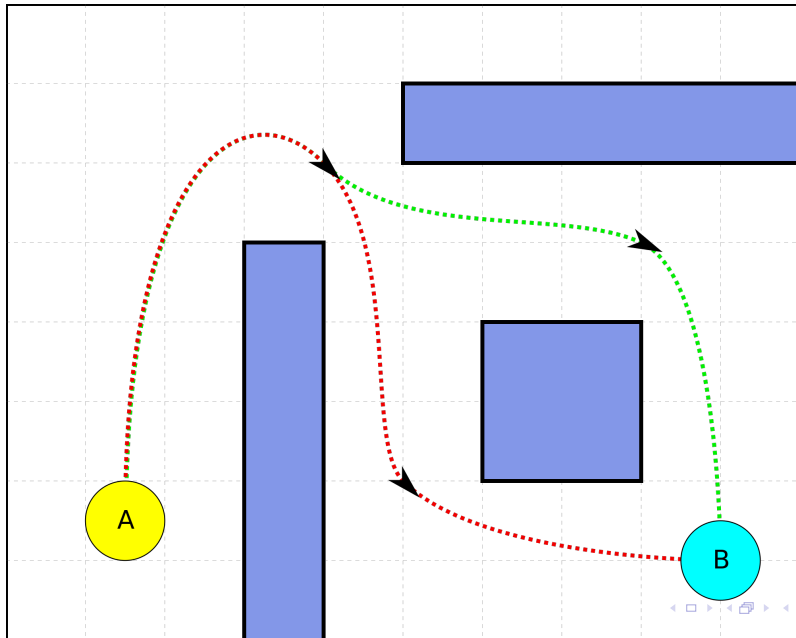


A* 算法简介

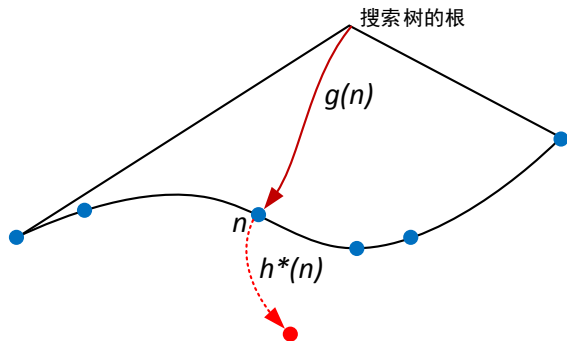
- A* 算法是一种有效的树搜索策略，在人工智能领域比较流行
- A* 算法一般针对最小化问题，如果是最大化问题，需要进行变换
- A* 算法强调在特定情况下，已找到的可行解必定是最优解，因此搜索可以提前终止
- 作为对比，分支限界法需要先找到一个可行解，然后进行剪枝，直到不存在可扩展节点
- 典型应用：电脑游戏中的寻路



A* 算法游戏寻路



A* 算法搜索过程的中间状态



符号表示:

1. Q : 当前可扩展顶点集合
2. n : 任意可扩展顶点
3. $g(n)$: 根节点到 n 节点的代价
4. $h^*(n)$: 从 n 扩展到达目标节点的最小代价
5. $h(n)$: $h^*(n)$ 的估计值, $h(n) \leq h^*(n)$
6. $f^*(n)$: $f^*(n) = g(n) + h^*(n)$, 从根节点出发经节点 n 到达目标节点的最小代价
7. $f(n)$: $f(n) = g(n) + h(n)$, $f(n)$ 的估计值, $f(n) \leq f^*(n)$



A* 算法基本规则

A* 算法基本规则如下：

- A* 算法采用最佳优先策略，每次总扩展代价 $f(n)$ 最小的节点
- 节点代价定义 $f(n) = g(n) + h(n)$ ，其中 $g(n)$ 是从根节点扩展到达 n 的代价， $h(n)$ 是从 n 扩展到达目标节点的最小代价 $h^*(n)$ 的估计值
- 确保 $h(n) \leq h^*(n)$
- 当且仅当选中的扩展节点是目标节点时，输出该节点对应的优化解，算法终止
- A* 算法在判定节点的后代不会产生优化解时，可以终止搜索该扩展节点



A* 算法优化性的证明

定理 1

A* 算法输出优化问题的优化解。

证 将问题优化解记为 s , A* 算法终止时选中的扩展节点 $t \in Q$ 是目标节点, 则 $f(t) = \min_{n \in Q} f(n)$ 。由于 t 是目标节点, 所以 $0 \leq h(t) \leq h^*(t) = 0$ 。于是, $f(t) = g(t) + h(t) \leq g(t) + h^*(t) = f^*(t)$ 。

另一方面, 由于 Q 包含当前所有可扩展节点, 所以优化解 s 必然通过 Q 中的一个节点扩展得到, 即 $f^*(s) = \min_{n \in Q} f^*(n) \geq \min_{n \in Q} f(n)$, 不等式成立是因为估计值 $h(n) \leq h^*(n)$ 。而 $\min_{n \in Q} f(n) = f(t)$, 所以 $f(t) \leq f^*(s)$ 。

由于 s 是优化解, 而 t 是一个可行解, 故 $f^*(s) \leq g(t)$, $g(t)$ 是算法中到 t 的实际代价, 而 $g(t) = f(t)$, 则 $f^*(s) \leq f(t)$ 。综上, $f^*(s) = f(t)$, 得证。



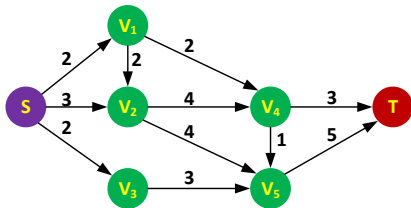
A* 算法求解 $S-T$ 最短路径问题

- $g(n)$: 从 S 出发到达 n 时经过路径的权值之和
- $h^*(n)$: 从 n 出发到 T 的最短路径的代价
- $h(n)$: 关联顶点 n 的所有出边的最小权值, 显然 $h(n) \leq h^*(n)$
- $f(n) = g(n) + h(n)$: 搜索树中节点 n 的代价
- 剪枝策略 1: 如果已获得代价为 $cost$ 的可行解, 则所有 $f(n) \geq cost$ 的节点 n 为死节点
- 剪枝策略 2: A^* 算法扩展过程中可能先后两次到达同一顶点 x , 设第一次到达代价为 $g_1(x)$, 第二次到达为 $g_2(x)$, 如果 $g_2(x) \leq g_1(x)$, 则第一次扩展在搜索树中对应的节点是死结点; 如果 $g_2(x) \geq g_1(x)$, 则第二次扩展对应的节点是死节点

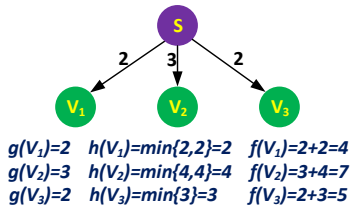
注: 因为算法保证 $h(n) \leq h^*(n)$, 所以 $f(n)$ (中的 $h(n)$) 通过剪枝策略 1 实现了对搜索路径的剪枝!



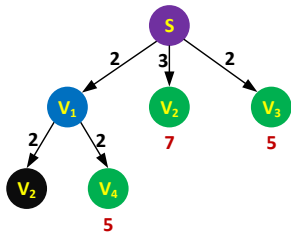
A* 算法解 $S-T$ 最短路径问题的搜索过程



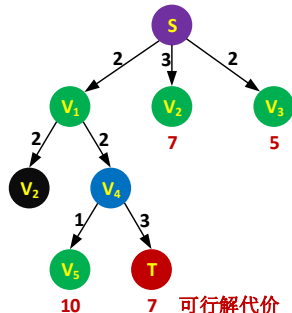
a) 最短路径的图



b) 第一步：扩展根节点



c) 第二步：扩展代价最小的节点 V_1

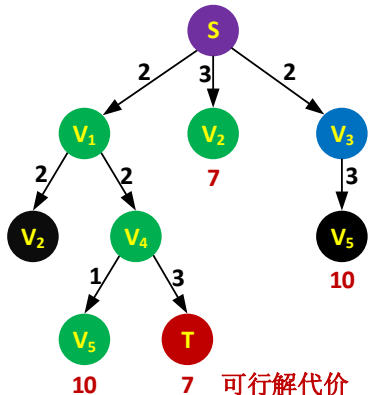


可行解代价

d) 第三步：扩展代价最小的节点 V_4

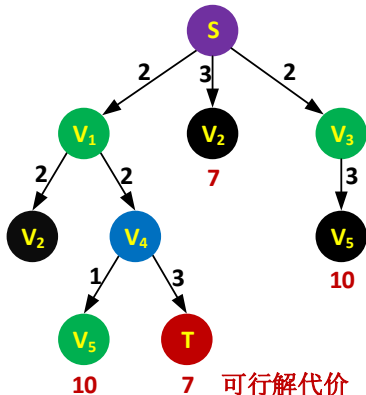


A* 算法求解最短路径问题的搜索过程 (续)



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

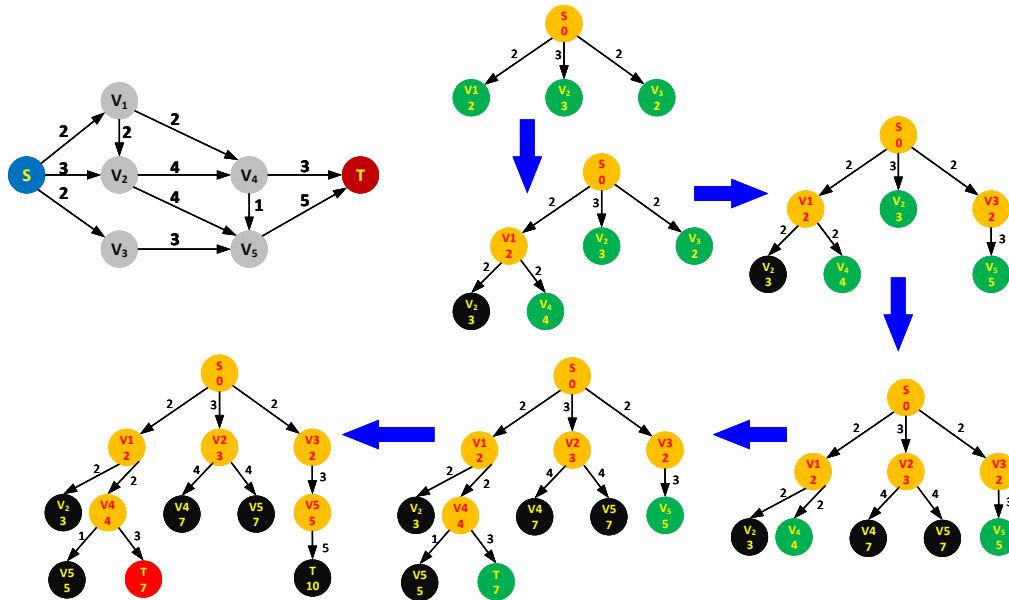
e) 第四步：扩展代价最小的节点V3



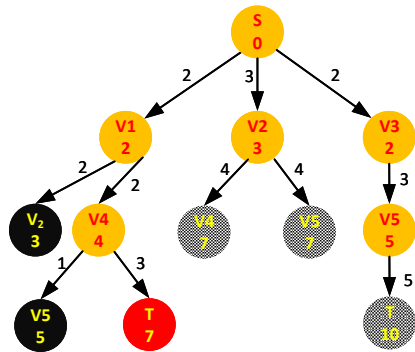
f) 第五步：扩展代价最小的节点V2

第五步扩展 V_2 以后，最后一步扩展目标节点 T ，扩展节点为目标节点时 A* 算法终止，返回最优解 7！

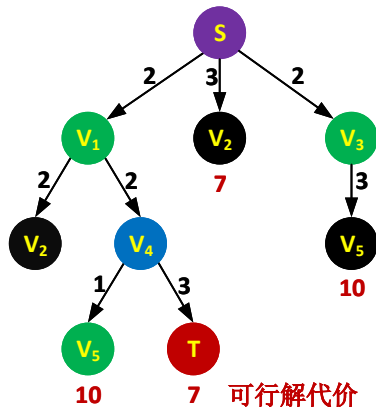
Dijkstra 算法求解最短路径过程



Dijkstra 算法与 A* 算法对比



a) Dijkstra算法的搜索树



b) A*算法的搜索树

Dijkstra 算法至少多搜索了灰色节点!



A* 算法 $h(n)$ 的讨论

- $h(n) = 0$ ，此时只有 $g(n)$ 起作用，此时 A* 算法演变成 Dijkstra 算法，能保证找到最短路径
- 如果 $h(n) \leq h^*(n)$ ，那么 A* 算法保证能找到一条最短路径，且 $h(n)$ 越小，需要扩展的点越多，运行速度越慢
- 如果 $h(n)$ 正好等于 $h^*(n)$ ，那么 A* 算法将只遵循最佳路径而不会扩展到其他任何结点，能够运行地很快
- 如果 $h(n) \geq h^*(n)$ ，则 A* 算法不能保证找到一条最短路径，但运行得更快
- $h(n)$ 比 $g(n)$ 大很多，则只有 $h(n)$ 起作用，此时 A* 算法演变成贪婪最佳优先搜索算法 (Greedy Best-First-Search)



小结

小结：

- 典型树搜索策略 (理解)
- 分支限界法应用 (重点)
- A^* 算法原理及应用 (重点)

