搜索求解

主讲：郭春乐、刘夏雷

南开大学计算机学院

致谢：本课件主要内容来自浙江大学吴飞教授、

南开大学程明明教授

单选题 1分 设置

1. 下面对一阶归纳推理(FOIL)中信息增益值(information gain)阐释不正确的是( )

A 信息增益值用来判断向推理规则中所加入前提约束谓词的质量

B 在算法结束前,每次向推理规则中加入一个前提约束谓词,该前提约束谓词得到的新推理规则具有最大的信息增益值。

C 在计算信息增益值过程中,需要利用所得到的新推理规则和旧推理规则分别涵盖的正例和反例数目

D 信息增益值大小与背景知识样例数目直接相关

单选题 1分

设置

2. 下面哪个描述的问题不属于因果分析的内容( )

A 如果商品价格涨价一倍,预测销售量(sales)的变化

B 如果广告投入增长一倍,预测销售量(sales)的变化

C 如果放弃吸烟,预测癌症(cancer)的概率

D 购买了一种商品的顾客是否会购买另外一种商品

提交

填空题 2分

设置

3. 常用的知识图谱推理方法有 [填空1] 、 [填空2] 等。

正常使用填空题需3.0以上版本雨课堂

作答

填空题 1分

设置

4. “总体样本上成立的某种关系在分组样本里恰好相反。”是著名的 [填空1] 。

正常使用填空题需3.0以上版本雨课堂

作答

填空题 3分

设置

5. 因果图常见的结构有：[填空1]、[填空2]、[填空3]等。

正常使用填空题需3.0以上版本雨课堂

作答

填空题 3分

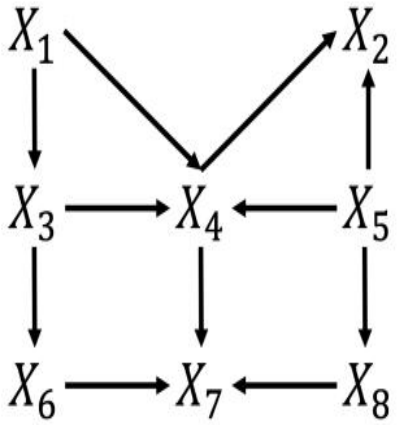
设置

6. 因果分析可被层次化表述为三个层次，即 [填空1] 、[填空2] 、 [填空3] 。

正常使用填空题需3.0以上版本雨课堂

作答

7.下图给出了不同变量之间的依赖关系，请写出因果图$\mathcal{H}$中 8 个变量之间的联合概率形式，并区分哪些变量是内生变量、哪些变量是外生变量。



因果图$\mathcal{H}$

解 使用乘积分解规则：

$$P(X_1, X_2, X_3, X_4, X_5, X_6, X_i, X_j)$$

$$= P(X_1) \times P(X_2|X_4, X_5) \times P(X_3|X_1) \times P(X_4|X_1, X_3, X_5) \times P(X_5)$$
$$\times P(X_6|X_3) \times P(X_7|X_4, X_6, X_8) \times P(X_8|X_5)$$

外生变量：X_1, X_5；内生变量：$X_2, X_3, X_4, X_6, X_7, X_8$

勘误

因此推理规则所覆盖的正例和反例的样本数分别是训练样本中正例和反例的数量

- $m_+ = 0,\ m_- = 0$

推理规则		推理规则涵盖的正例和反例数		FOIL信息增益值
目标谓词	前提约束谓词	正例	反例	信息增益值
$Father(x, y) \leftarrow$	空集	$m_+ = 1$	$m_- = 4$	$FOIL\_Gain$
	$Mother(x, y)$	$\widehat{m_+} = 0$	$\widehat{m_-} = 2$	NA
	$Mother(x, z)$	$\widehat{m_+} = 0$	$\widehat{m_-} = 2$	NA
	$Mother(y, x)$	$\widehat{m_+} = 0$	$\widehat{m_-} = 0$	NA
	$Mother(y, z)$	$\widehat{m_+} = 0$	$\widehat{m_-} = 1$	NA
	$Mother(z, x)$	$\widehat{m_+} = 0$	$\widehat{m_-} = 1$	NA
	$Mother(z, y)$	$\widehat{m_+} = 1$	$\widehat{m_-} = 3$	0.32
	$Sibling(x, y)$	$\widehat{m_+} = 0$	$\widehat{m_-} = 1$	NA
	$Sibling(x, z)$	$\widehat{m_+} = 0$	$\widehat{m_-} = 1$	NA

背景知识样例集合	Sibling(Ann, Mike) Couple(David, James) Mother(James, Ann) Mother(James, Mike)	目标谓词训练样例集合	Father(David, Mike) ¬Father(David, James) ¬Father(James, Ann) ¬Father(James, Mike) ¬Father(Ann, Mike)

勘误

$$score(s,t) = \sum_{\pi_j \in p_l} \theta_j \ father(s \rightarrow t; \pi_j)$$

给定目标关系：$Father(s,t)$

1. 对于目标关系$Father$，生成四组训练样例，一个为正例、三个为负例：

正例：(David, Mike)

负例：(David, James)，(James, Ann)，(James, Mike)

2. 从知识图谱采样得到路径，每一路径链接上述每个训练样例中两个实体：

(David, Mike)对应路径：$Couple \rightarrow Mother$

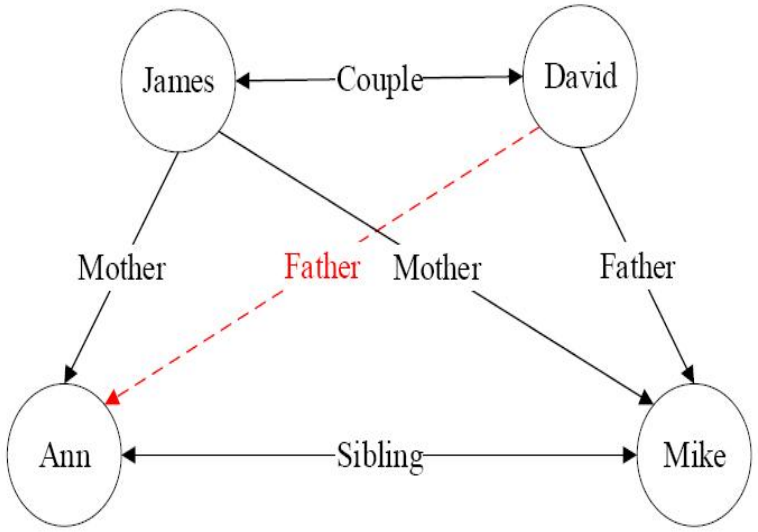
(David, James)对应路径：$Father \rightarrow Mother^{-1}$

（$Mother^{-1}$与 $Mother$为相反关系）

(James, Ann)对应路径：$Mother \rightarrow Sibling$

(James, Mike)对应路径：$Couple \rightarrow Father$

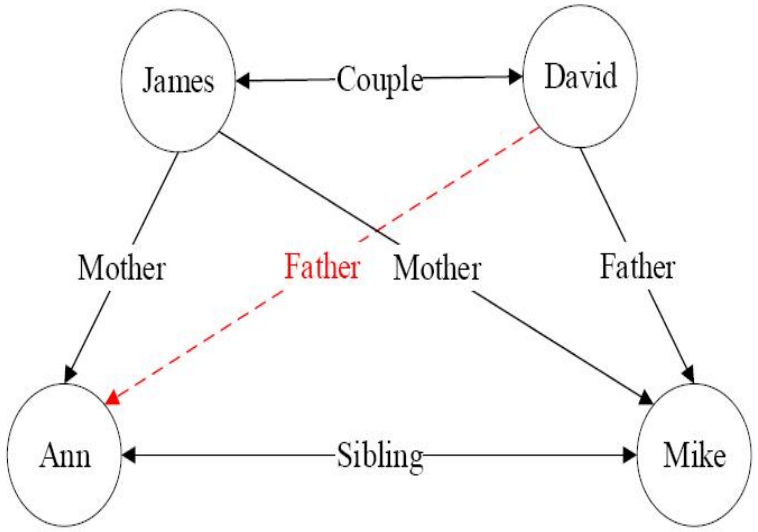
Score($Father(David, Ann)$)



勘误

$$score(s,t) = \sum_{\pi_j \in p_l} \theta_j \ father(s \to t; \pi_j)$$

Score($Father(David, Ann)$)



3. 对于每一个正例/负例，判断上述四条路径可否链接其包含的两个实体，将可链接(记为1)和不可链接(记为0)作为特征，于是每一个正例/负例得到一个四维特征向量：

(David, Mike)：$\{[1, 0, 0, 0], 1\}$

(David, James)：$\{[0, 1, 0, 0], -1\}$

(James, Ann)：$\{[0, 0, 1, 0], -1\}$

(James, Mike)：$\{[0, 0, 1, 1], -1\}$

4. 依据训练样本，训练分类器M

提纲

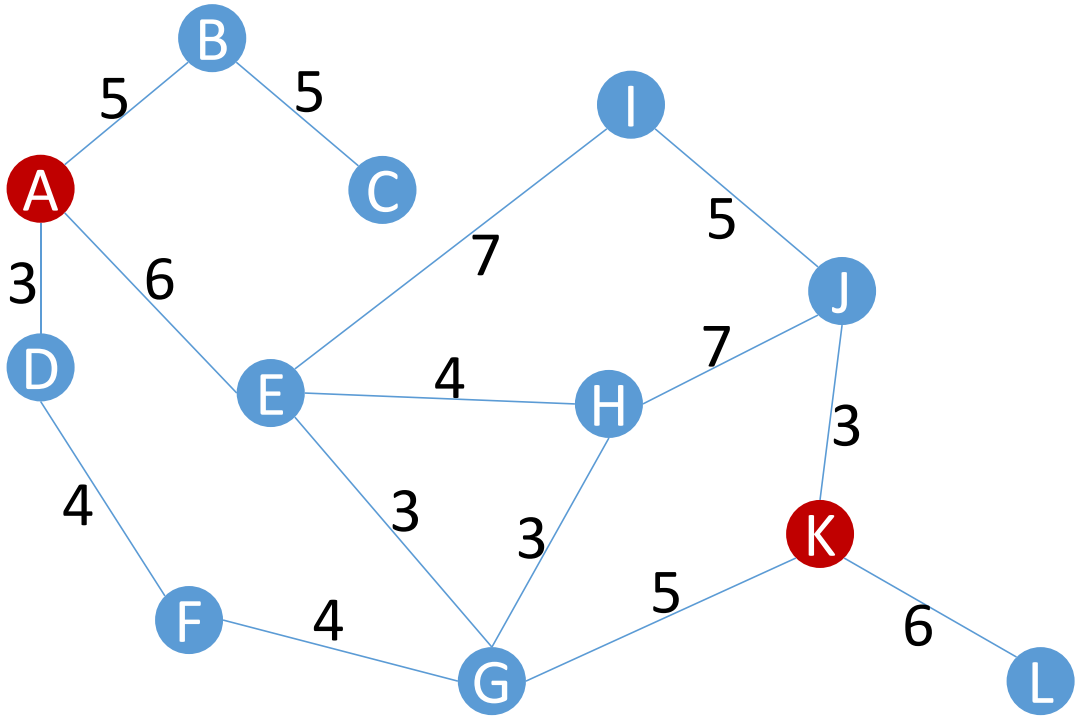
- 搜索算法基础
- 启发式搜索
- 对抗搜索
- 蒙特卡洛树搜索

提纲

- 搜索算法基础
- 启发式搜索
- 对抗搜索
- 蒙特卡洛树搜索

搜索算法的形式化描述

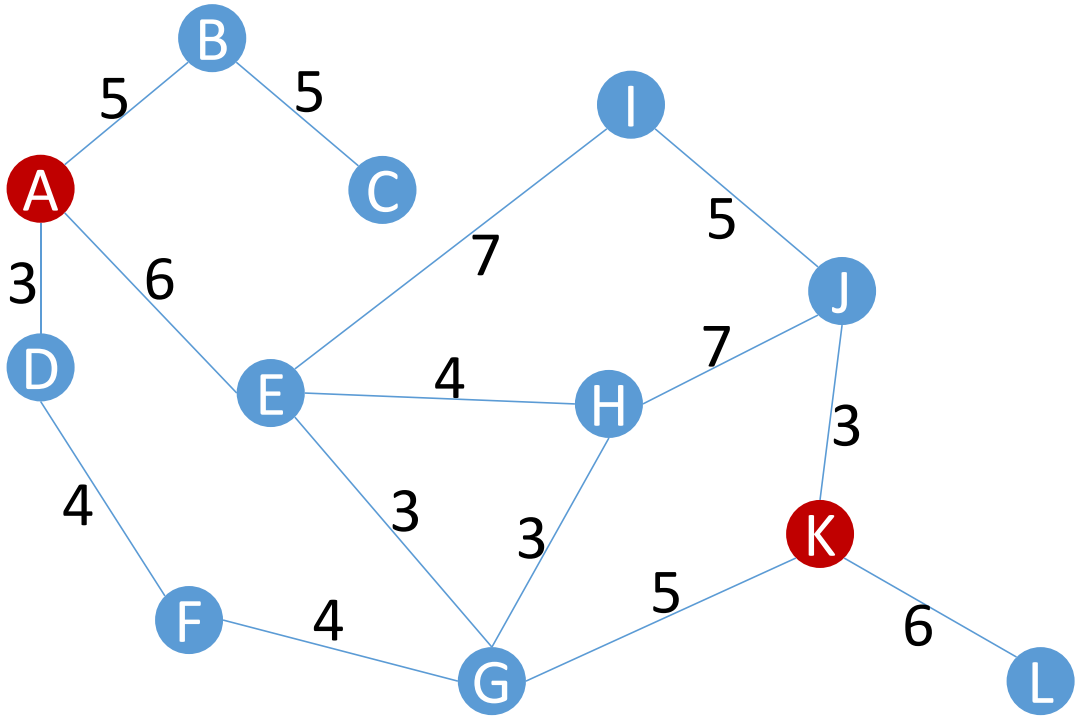
- **状态**
  - 对智能体和环境当前情形的描述。例如，在最短路径问题中，城市可作为状态。将原问题对应的状态称为初始状态。
- **动作**
  - 从当前时刻所处状态转移到下一时刻所处状态所进行操作。一般而言这些操作都是离散的。



问题：寻找从城市A到城市K之间行驶时间最短路线?

搜索算法的形式化描述

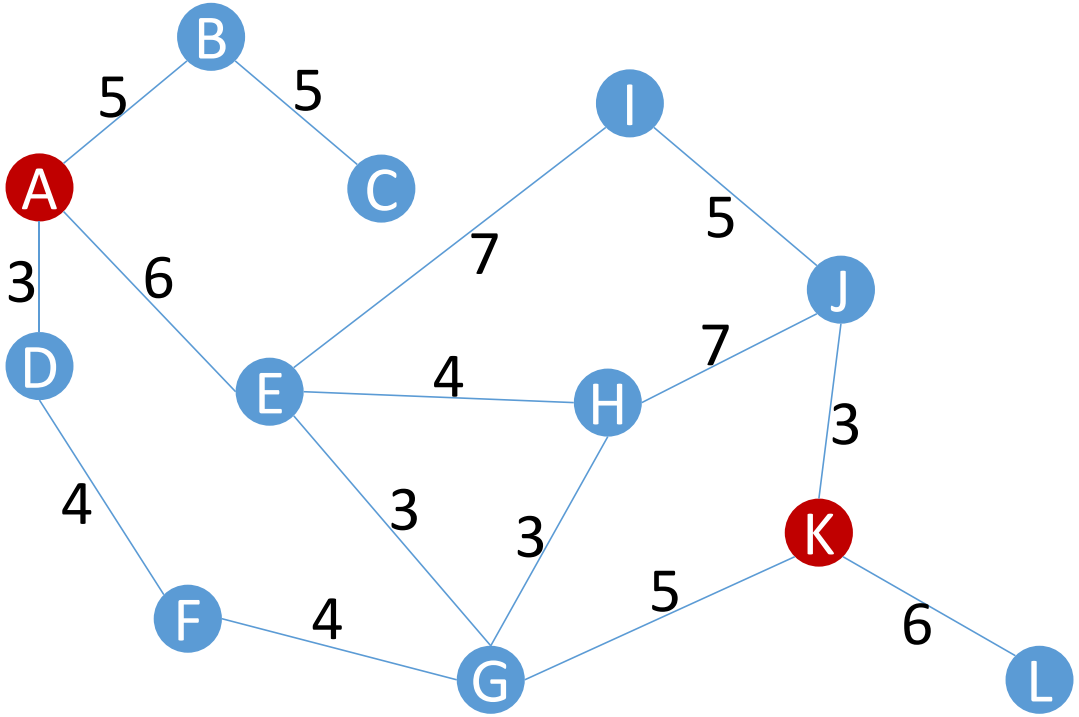
- **状态转移**
  - 对智能体选择了一个动作之后，其所处状态的相应变化
- **路径/代价**
  - 一个状态序列。该状态序列被一系列操作所连接。
  - 如从A到K所形成的路径。
- **目标测试**
  - 评估当前状态是否为目标状态

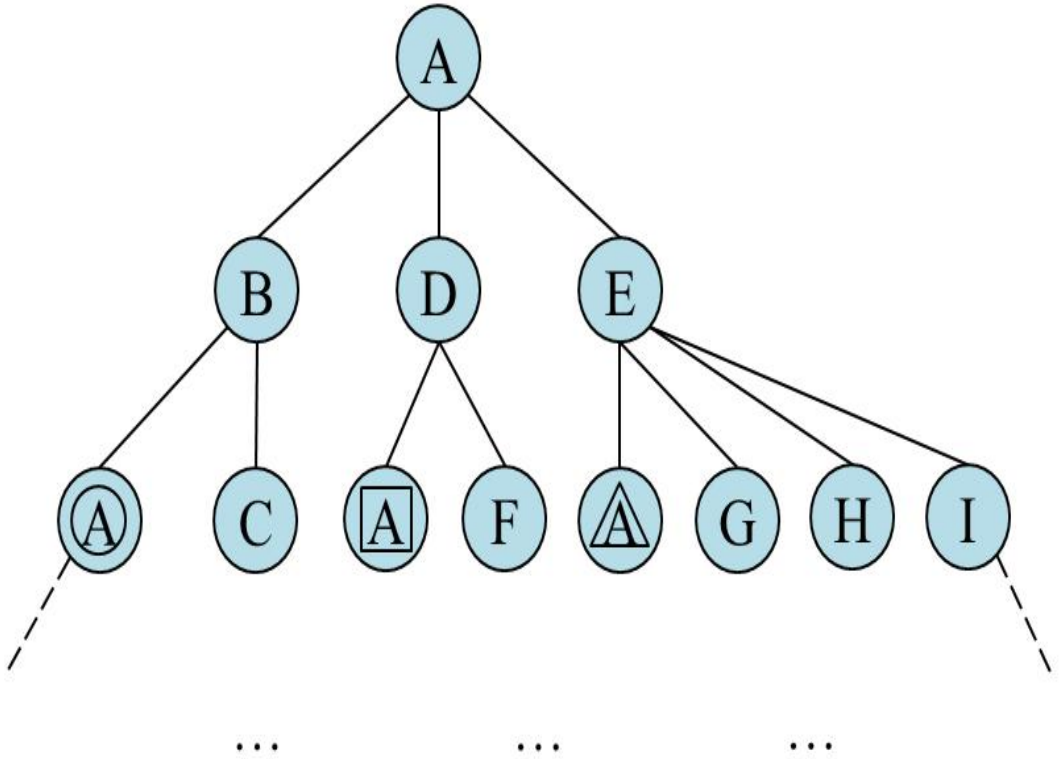


问题：寻找从城市A到城市K之间行驶时间最短路线?

搜索树：用一棵树来记录算法探索过的路径

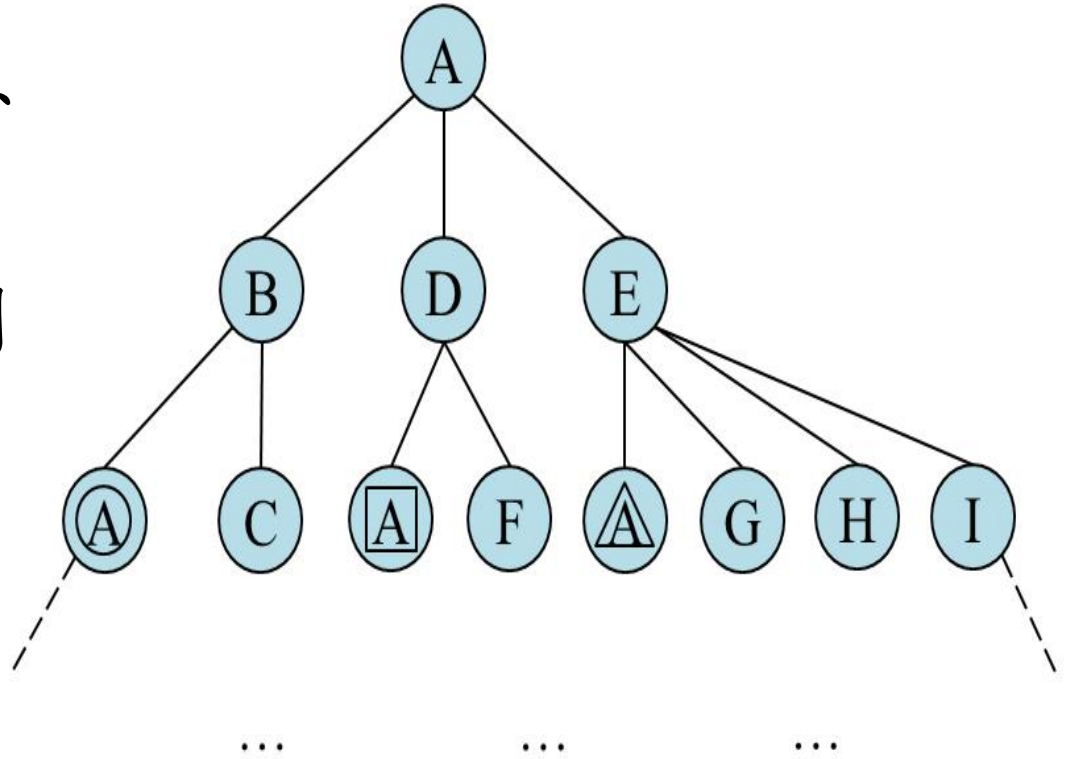
- 搜索算法会时刻记录所有从初始结点出发已经探索过的路径，每次从中选出一条，从该路径末尾状态出发进行一次状态转移，探索一条尚未被探索过新路径。





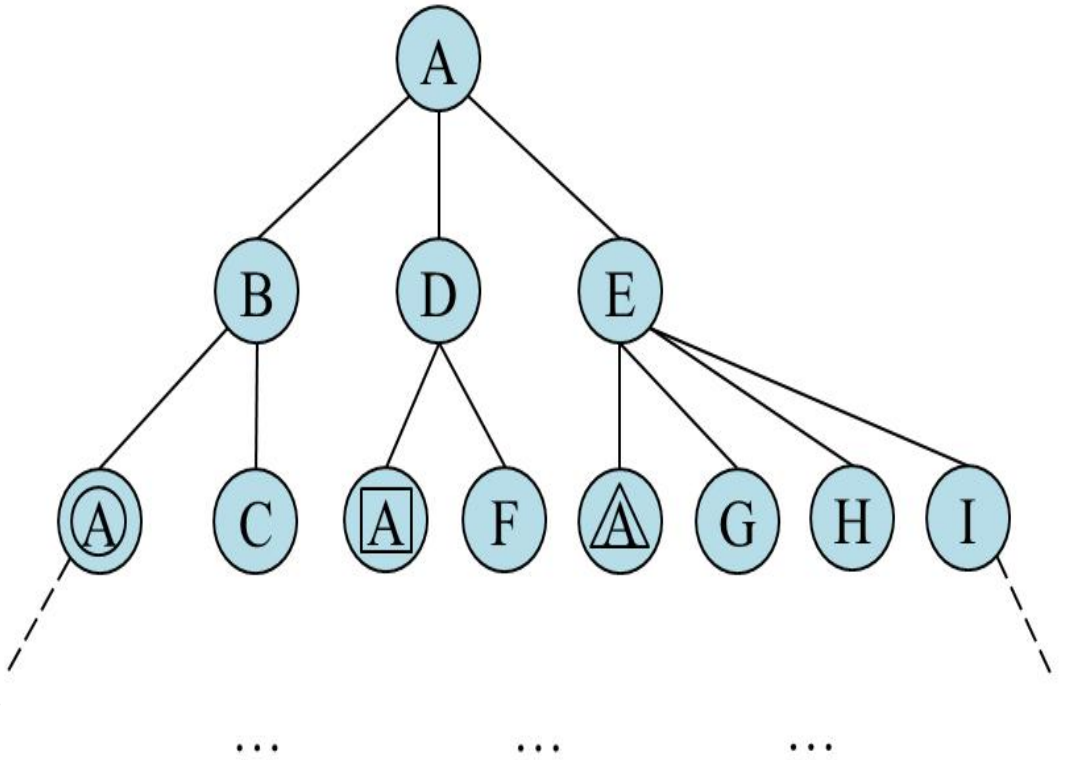
搜索树：用一棵树来记录算法探索过的路径

- 第三层中有三个标号均为A的结点
  - 分别被圆圈、正方形和三角形框住
  - 虽然三个结点对应同一个城市，即所对应状态相同，但是这三个节点在搜索树中却是不同结点
  - 它们分别代表了从初始状态出发到达城市A的三条不同路径。



搜索树：用一棵树来记录算法探索过的路径

- **这三个结点表示的路径分别为**
  - A→B→A、A→D→A和A→E→A
  - 同一个标号一定表示相同的状态，其含义为智能体当前所在的城市
  - 但一个标号可能有多个结点与之对应
  - 不同结点对应从初始状态出发的不同路径
- **搜索算法是一个构建搜索树的过程**
  - 从根结点(初始状态)开始，不断展开每个结点的后继结点，直到某个结点通过了目标测试。



搜索算法的评价指标

完备性	当问题存在解时，算法是否能保证找到一个解。
最优性	搜索算法是否能保证找到的第一个解是最优解。
时间复杂度	找到一个解所需时间。
空间复杂度	在算法的运行过程中需要消耗的内存量。

完备性和最优性刻画了算法找到解的能力以及所求的解的质量，时间复杂度和空间复杂度衡量了算法的资源消耗，它们通常用 O 符号(big O notation)来描述。

符号	含义
b	分支因子，即搜索树中每个节点最大的分支数目
d	根节点到最浅的目标结点的路径长度
m	搜索树中路径的最大可能长度
n	状态空间中状态的数量

搜索树中用于估计复杂度的变量含义

搜索算法框架：树搜索

- 集合$\mathcal{F}$ 用于保存搜索树中可用于下一步探索的所有候选结点
  - 这个集合被称为边缘(fringe)集合，有时也被叫做开表(open list)

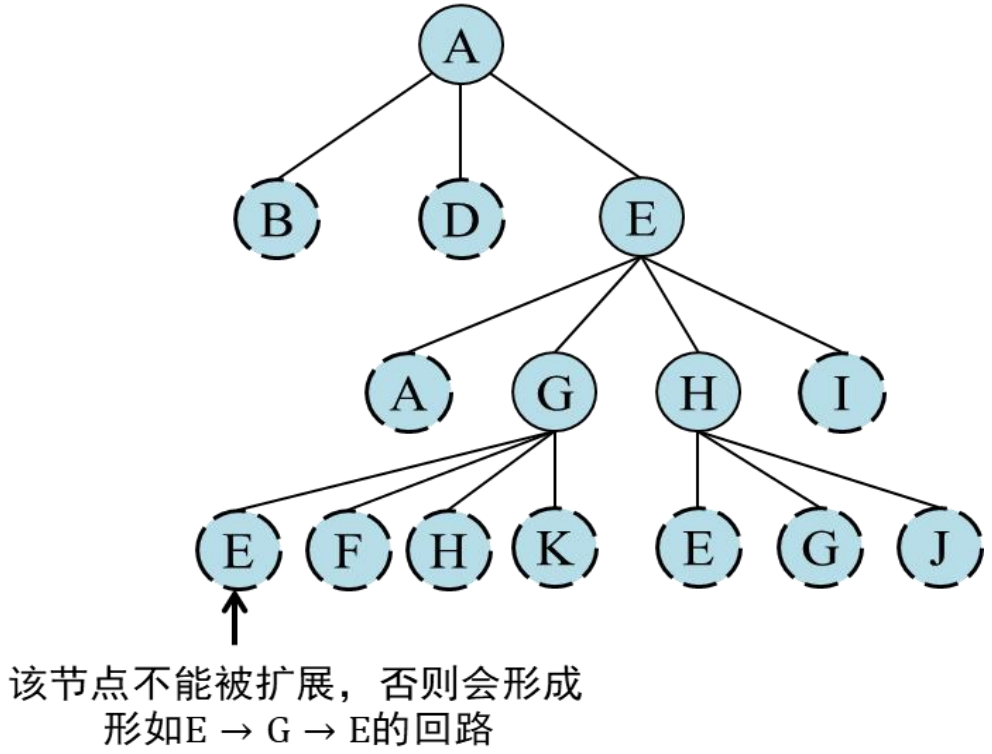
函数： TreeSearch

输入： 节点选择函数 pick_from，后继节点计算函数 successor_nodes
输出： 从初始状态到终止状态的路径

```
1 F ← {根节点}
2 while F ≠ ∅ do
3 |   n ← pick_from(F)
4 |   F ← F − {n}
5 |   if goal_test(n) then
6 |   |   return n.path
7 |   end
8 |   F ← F ∪ successor_nodes(n)
9 end
```

剪枝搜索 - 并不是其所有的后继节点都值得被探索

- 有时候，主动放弃一些后继结点能够提高搜索效率而不会影响最终搜索结果，甚至能解决无限循环(即算法不停机)问题。



正在构建中的搜索树

注意到图中路径A → E → G → E → G → E → …，这意味着在某些搜索策略下(例如深度优先搜索)，算法可能会沿着搜索树的右侧路径在状态E和状态G之间陷入无限循环，即出现环路或回路，搜索算法无法终止，此时算法不具有完备性。

图搜索–不允许环路的存在

函数：GraphSearch
输入：节点选择函数 pick_from，后继节点计算函数 successor_nodes
输出：从初始状态到终止状态的路径

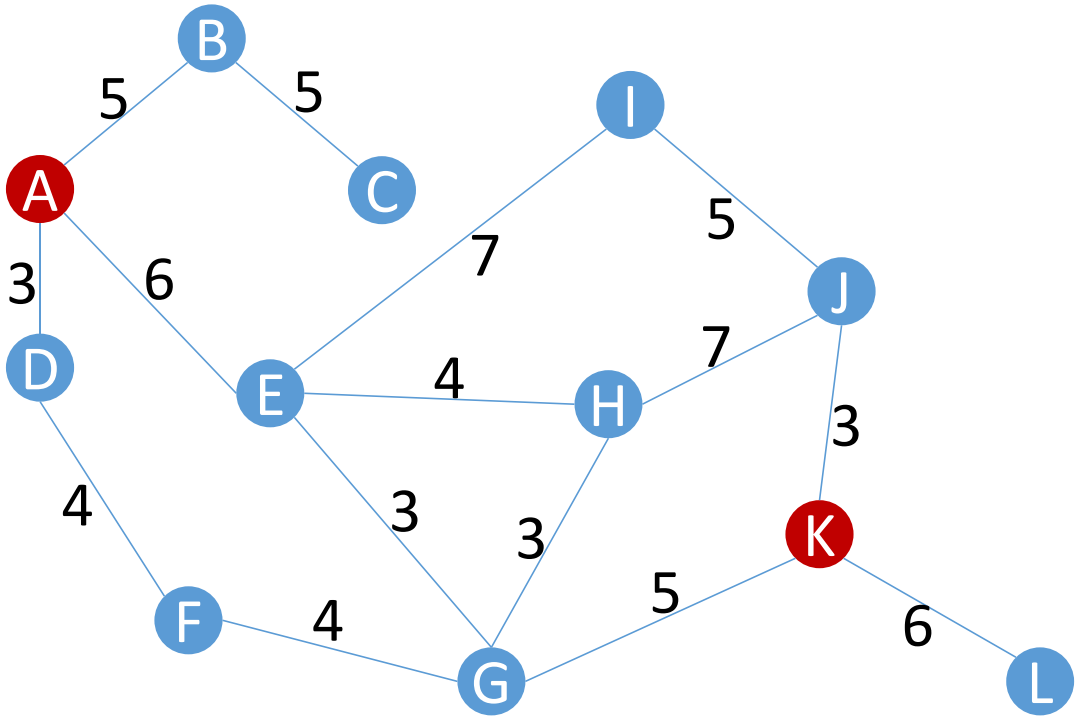
```
1  F ← {根节点}
2  C ← ∅
3  while F ≠ ∅ do
4      n ← pick_from(F)
5      F ← F − {n}
6      if goal_test(n) then
7          return n.path
8      end
9      if n.state ∉ C then
10         C ← C ∪ {n.state}
11         F ← F ∪ successor_nodes(n)
12     end
13 end
```

提纲

- 搜索算法基础
- 启发式搜索
- 对抗搜索
- 蒙特卡洛树搜索

搜索算法：启发式搜索(有信息搜索)

- 在搜索的过程中利用与所求解问题相关的辅助信息，其代表算法为**贪婪最佳优先搜索**(Greedy best-first search)和**A*搜索**。



寻找从城市A到城市K之间最短路线？

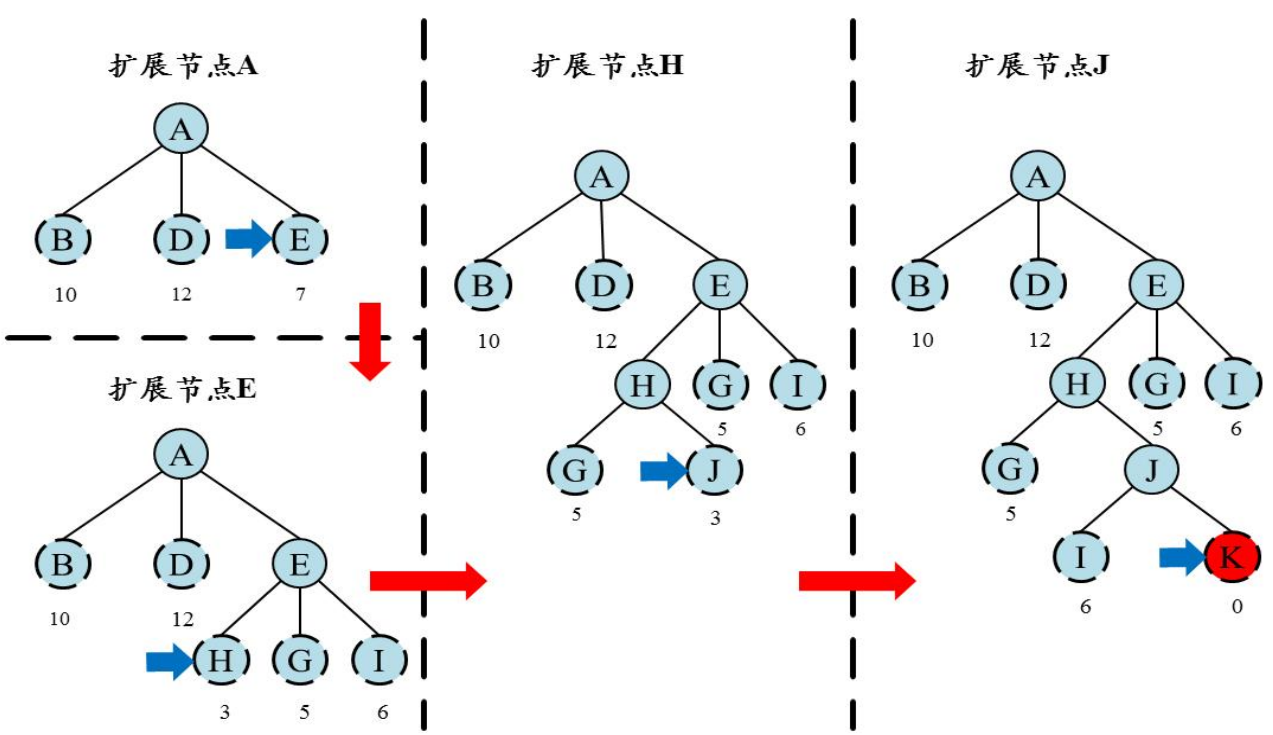
搜索算法：启发式搜索(有信息搜索)

辅助信息	所求解问题之外、与所求解问题相关的特定信息或知识。	
评价函数$f(n)$ (evaluation function)	从当前节点n出发，根据评价函数来选择后续结点。	下一个结点是谁?
启发函数$h(n)$ (heuristic function)	从结点n到目标结点之间所形成路径的最小代价值，这里用两点之间的直线距离。	完成任务还需要多少代价?

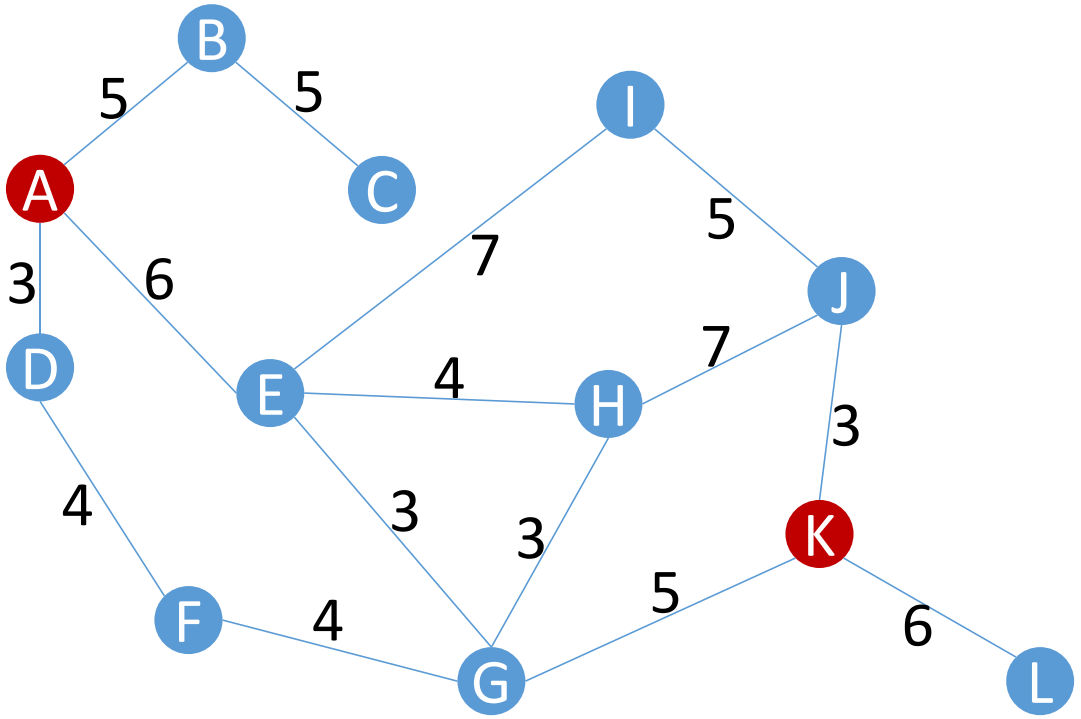
- **贪婪最佳优先搜索：评价函数$f(n)$=启发函数$h(n)$**
- **辅助信息(启发函数)**
  - 任意一个城市与终点城市K之间的直线距离

搜索算法：贪婪最佳优先搜索

- 贪婪最佳优先搜索：评价函数$f(n)$=启发函数$h(n)$
  - 例：启发函数为任意一个城市与终点城市K之间的直线距离



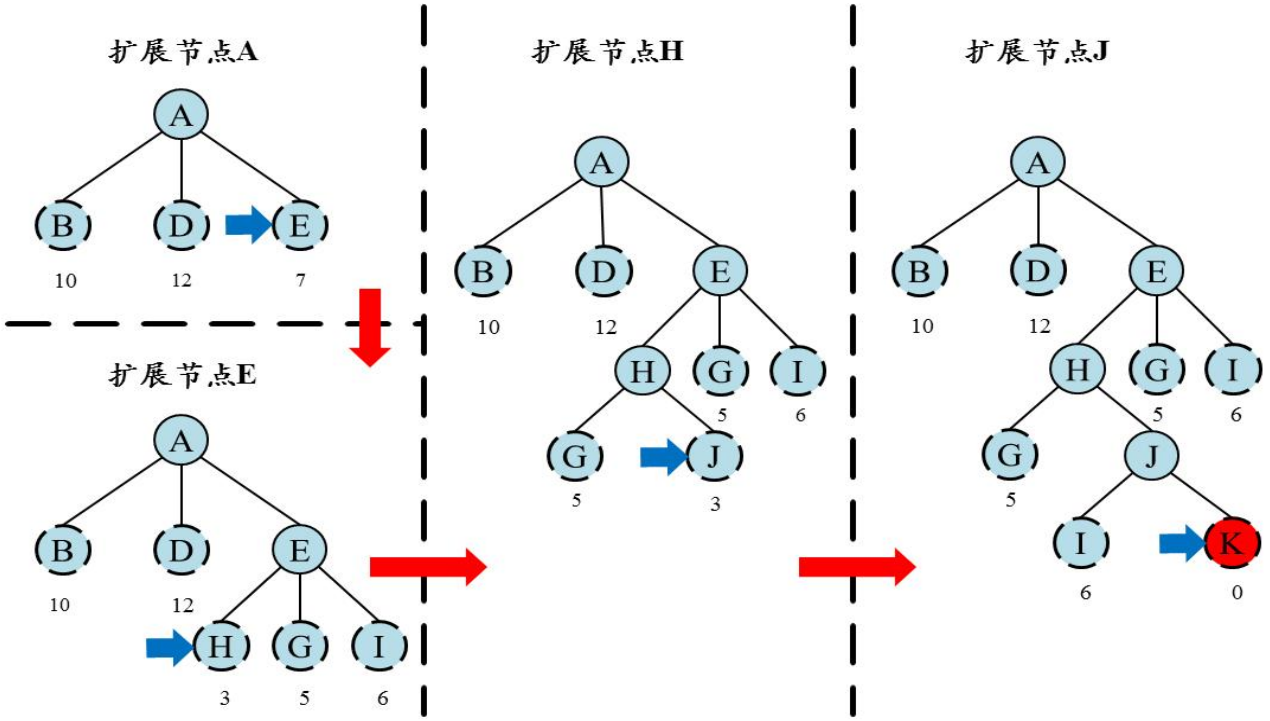
贪婪最佳优先搜索的过程



寻找从城市A到城市K之间最短路线？

搜索算法：贪婪最佳优先搜索

- 贪婪最佳优先搜索：评价函数$f(n)$=启发函数$h(n)$
  - 例：启发函数为任意一个城市与终点城市K之间的直线距离



贪婪最佳优先搜索的过程

算法找到了一条从起始结点到终点结点的路径A → E → H → J → K，但这条路径并不是最短路径，实际上最短路径为A → E → G → K。

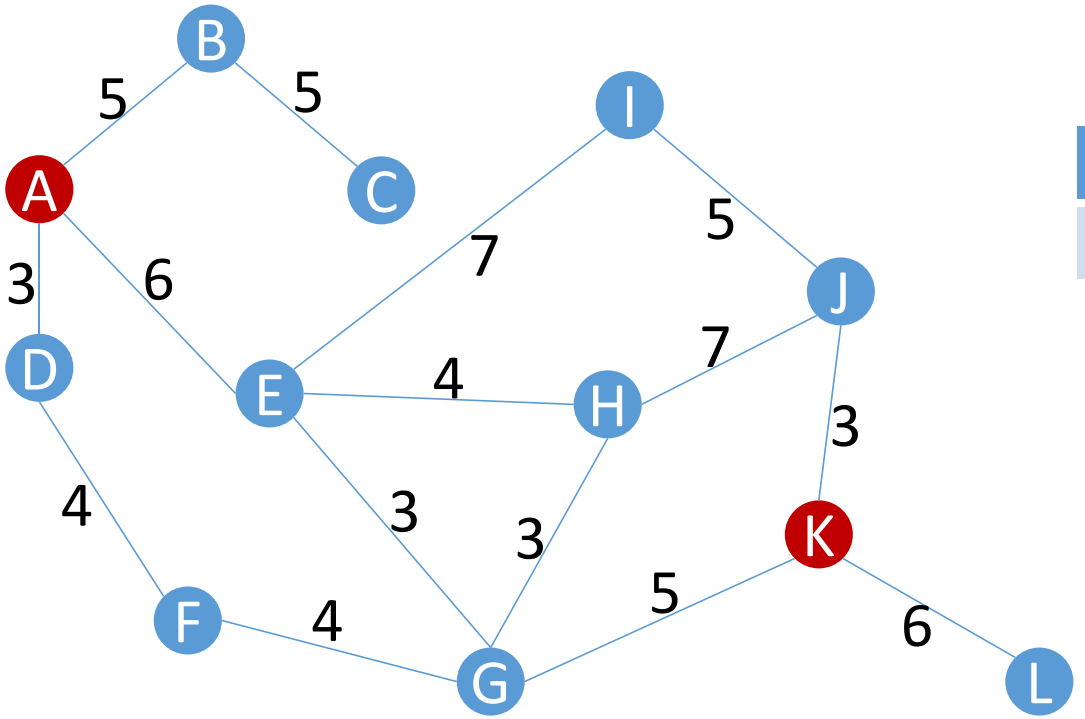
搜索算法：A*算法

- **评价函数：$\mathbf{f}(\boldsymbol{n}) = \mathbf{g}(\boldsymbol{n}) + \boldsymbol{h}(\boldsymbol{n})$**
  - $\mathbf{g}(\boldsymbol{n})$表示从起始结点到结点$\boldsymbol{n}$的开销代价值
  - $\boldsymbol{h}(\boldsymbol{n})$表示从结点$\boldsymbol{n}$到目标结点路径中所估算的最小开销代价值
  - $\mathbf{f}(\boldsymbol{n})$可视为经过结点$\boldsymbol{n}$、具有最小开销代价值的路径。

$$\underbrace{f(n)}_{\text{评价函数}} = \underbrace{g(n)}_{\substack{\text{起始结点到结点}n\text{代价} \\ \text{(当前最小代价)}}} + \underbrace{h(n)}_{\substack{\text{结点}n\text{到目标结点代价} \\ \text{(后续估计最小代价)}}}$$

搜索算法：A*算法

$$\underbrace{f(n)}_{\text{评价函数}} = \underbrace{g(n)}_{\text{起始结点到结点}n\text{代价}(\text{当前最小代价})} + \underbrace{h(n)}_{\text{结点}n\text{到目标结点代价}(\text{后续估计最小代价})}$$

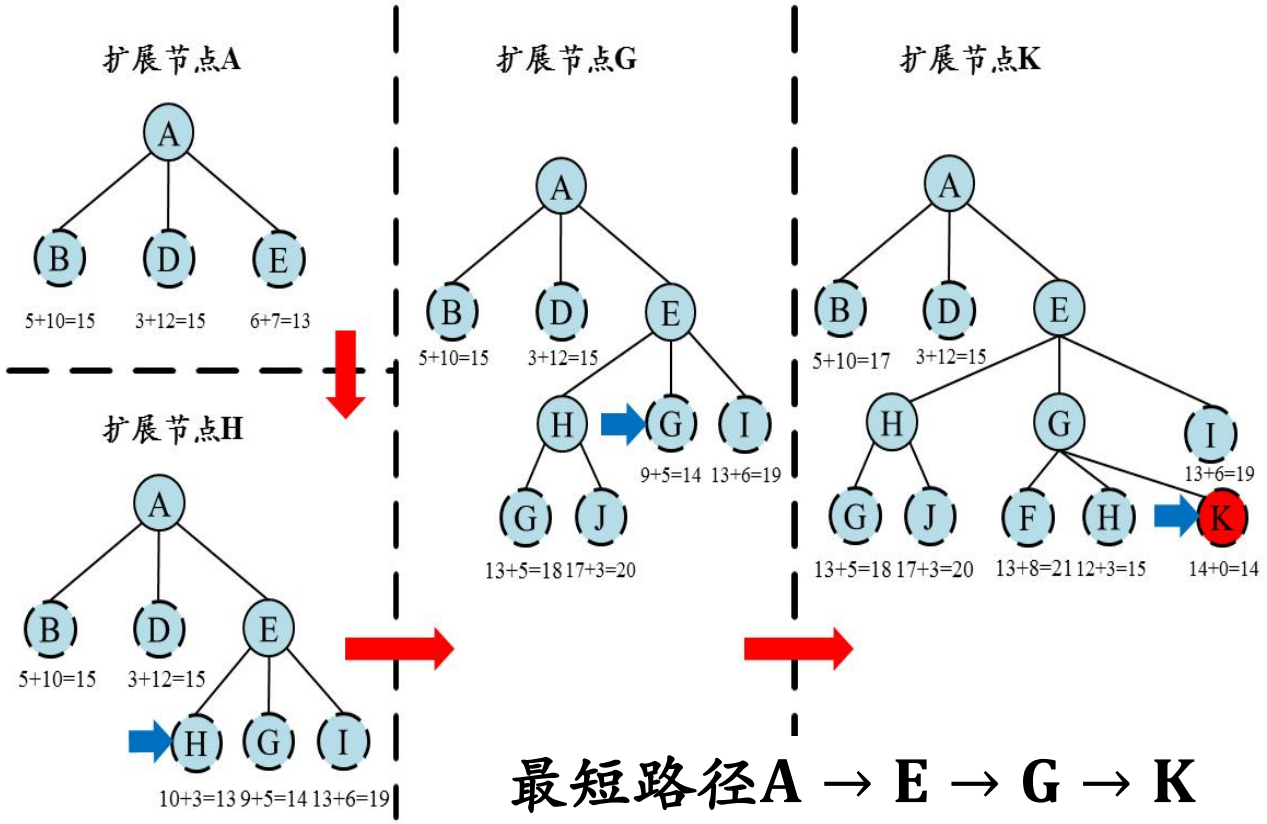


状态	A	B	C	D	E	F	G	H	I	J	K	L
$h(n)$	13	10	6	12	7	8	5	3	6	3	0	6

辅助信息：任意一个城市与终点城市K之间的直线距离

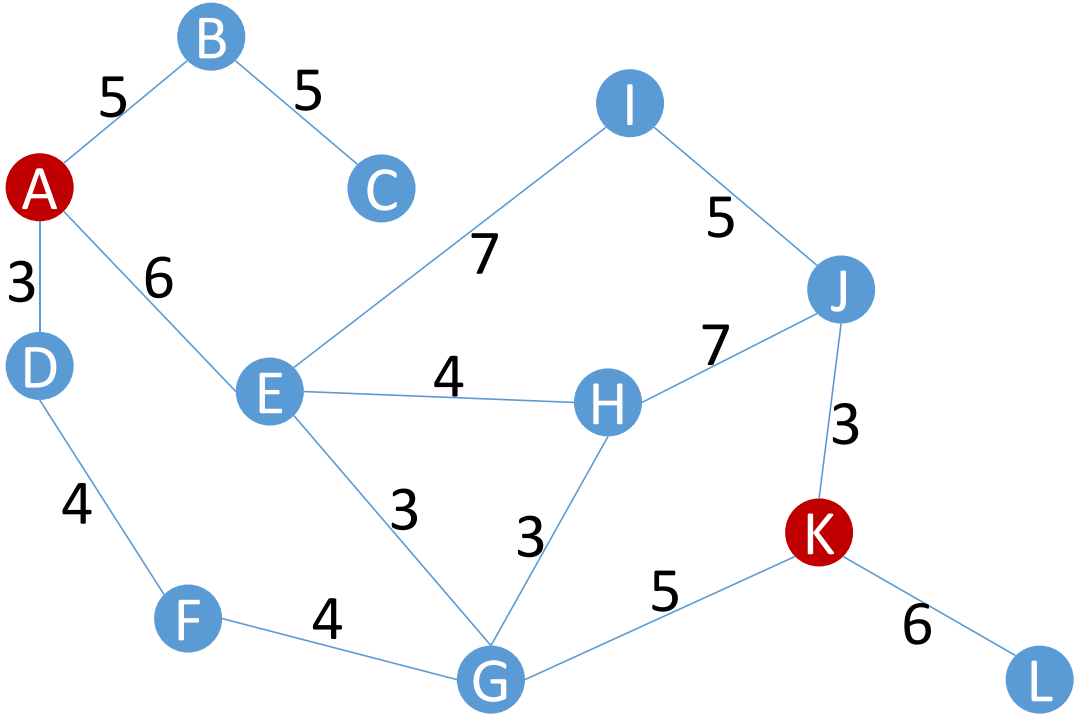
寻找从城市A到城市K之间最短路线？

搜索算法：A*算法



最短路径A → E → G → K

A*算法的搜索过程



寻找从城市A到城市K之间最短路线？

状态	A	B	C	D	E	F	G	H	I	J	K	L
$h(n)$	13	10	6	12	7	8	5	3	6	3	0	6

辅助信息：任意一个城市与终点城市K之间的直线距离

搜索算法：A*算法性能分析

- A*算法的完备性和最优性取决于搜索问题和启发函数的性质

符号	含义
$h(n)$	节点n的启发函数取值
$g(n)$	从起始节点到节点n所对应路径的代价
$f(n)$	节点n的评价函数取值
$c(n,a,n')$	从节点n执行行动a到达节点n'的单步代价
$\mathrm{h}^*(n)$	从节点n出发到达终止节点的最小代价

辅助信息	所求解问题之外、与所求解问题相关的特定信息或知识。	
评价函数$f(n)$ (evaluation function)	从当前节点n出发，根据评价函数来选择后续结点。	下一个结点是谁?
启发函数$h(n)$ (heuristic function)	结点n到目标结点之间所形成路径的最小代价值，这里用两点之间的直线距离。	完成任务至少还需要多少代价?

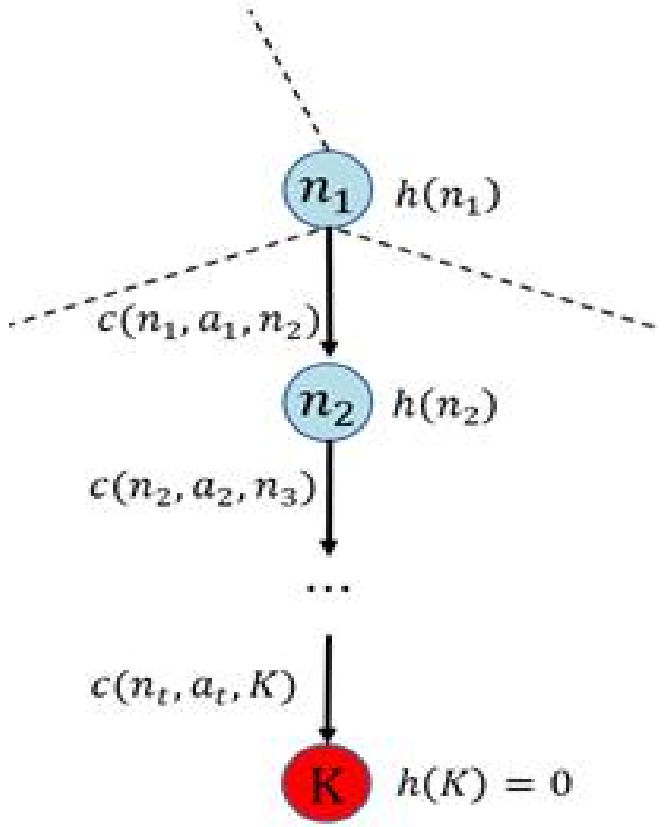
搜索算法：A*算法性能分析

- **启发函数需要满足如下两种性质：**
  - **可容性(admissible)**：对于任意结点n，有$h(n) \leq h^*(n)$，如果n是目标结点，则有$h(n) = 0$。$h^*(n)$是从结点n出发到达终止结点所付出的(实际)最小代价。可以这样理解满足可容性的启发函数，启发函数不会过高估计(over-estimate)从结点n到终止结点所应该付出的代价。
  - **一致性(consistency)**：启发函数的一致性指满足条件$h(n) \leq c(n, a, n') + h(n')$，这里$c(n, a, n')$表示结点$n$通过动作a到达其相应的后继结点$n'$的代价(三角不等式原则)。

搜索算法：A*算法性能分析

- **满足一致性的启发函数一定满足可容性**
  - 如图中以n_1为根结点的子树，假设从该节点到达终止结点代价最小的路径为$n_1 \to n_2 \to \cdots \to K$。由于$h(K) = 0$且启发函数满足一致性条件，故可以推导如下：

$$
\begin{aligned}
h(n_1) &\le h(n_2) + c(n_1, a_1, n_2) \\
&\le h(n_3) + c(n_2, a_2, n_3) + c(n_1, a_1, n_2) \\
&\le \cdots \\
&\le c(n_1, a_1, n_2) + c(n_2, a_2, n_3) + \cdots + c(n_t, a_t, \mathrm{K}) \\
&= h^*(n_1)
\end{aligned}
$$



从n_1结点到终止结点的最短路径

搜索算法：A*算法的完备性

- **完备性：如果在起始点和目标点间有路径解存在，那么一定可以得到解，如果得不到解那么一定说明没有解存在**
  - 在一些常见的搜索问题中，状态数量是有限的，此时图搜索A*算法和排除环路的树搜索A*均是完备的，即一定能够找到一个解。在更普遍的情况下，如果所求解问题和启发函数满足以下条件，则A*算法是完备的：
    - 搜索树中分支数量有限，即每个节点的后继结点数量是有限的
    - 单步代价的下界是一个正数
    - 启发函数有下界

搜索算法：树搜索A*算法的最优性

- **如果启发函数是可容的，那么树搜索的A*算法满足最优性**

证明：启发函数满足可容性时，假设树搜索的A*算法找到的第一个终止结点为n。对于此时边缘集合中任意结点n'，根据算法每次扩展时的策略，即选择评价函数取值最小的边缘节点，有$f(n) \leq f(n')$。

由于A*算法对评价函数定义为$f(n) = g(n) + h(n)$，且$h(n) = 0$，有$f(n) = g(n) + h(n) = g(n)$，$f(n') = g(n') + h(n')$，综合可得$g(n) \leq g(n') + h(n') \leq g(n') + h^*(n')$。

此时扩展其他任何一个边缘结点都不可能找到比结点n所对应路径代价更小的路径，因此结点n对应的是一条最短路径，即算法满足最优性。

搜索算法：图搜索A*算法性能分析

- **如果A*算法采用的是图搜索策略，那么即使启发函数满足可容性条件，也不能保证算法最优性。**
  - 例子：如果有多个结点对应同一个状态(即存在多条到达同一个城市的路径)，图搜索算法只会扩展其中的一个结点，而剪枝其余的结点，然而最短路径有可能经过其中某些被剪枝结点，导致算法无法找到这些最短路径。

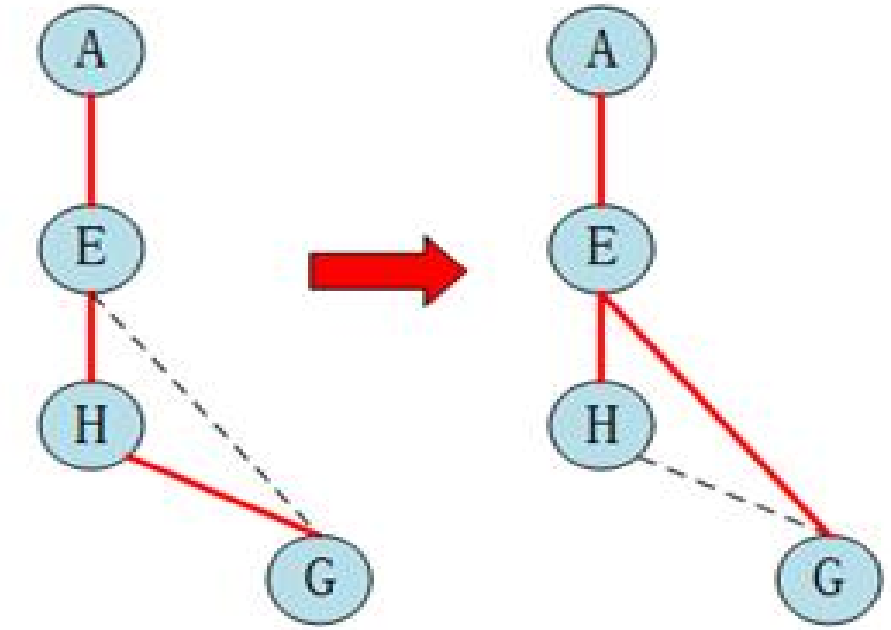
状态	A	B	C	D	E	F	G	H	I	J	K	L
$h(n)$	13	10	6	**12→0**	7	**8→0**	**5→0**	3	6	3	0	6

表 3.2 每个状态(城市)所对应启发函数取值

此时启发函数仍然满足可容性，但图搜索的A*算法会优先扩展结点A → D → F → G，而放弃最短路径经过的节点A → E → G。

搜索算法：A*算法性能分析

- **图搜索A*算法满足最优性(方法一)**
  - 当算法从不同路径扩展同一结点时，不保留最先探索的那条路径，而是保留代价最小的那条路径。



修改后图搜索A*算法扩展A→E→G结点，红色实线表示当前搜索树中的边，虚线表示不在搜索树中的边

搜索算法：A*算法性能分析

- **图搜索A*算法满足最优性(方法二)**
  - 要求启发函数满足一致性

引理3.1： 启发函数满足一致性条件时，给定一个从搜索树中得到的结点序列，每个结点是前一个结点的后继，那么评价函数对这个序列中的结点取值按照结点出现的先后次序而依次单调非减。

考虑序列中结点n，对结点n采取动作a而得到的后继节点n'。由于启发函数满足一致性条件，因此有$h(n) \leq c(n,a,n')+h(n')$。于是有：

$$f(n') = g(n')+h(n') = g(n)+c(n,a,n')+h(n') \geq g(n)+h(n) = f(n)$$

搜索算法：A*算法性能分析

- **图搜索A*算法满足最优性(方法二)**
  - 要求启发函数满足一致性

引理3.1： 启发函数满足一致性条件时，给定一个从搜索树中得到的结点序列，每个结点是前一个结点的后继，那么评价函数对这个序列中的结点取值按照结点出现的先后次序而依次单调非减。

引理3.2： 假设状态 加入搜索树时对应的结点为 ，在结点 被加入搜索树后，若对应状态 的任何结点 ′出现在边缘集合中，那么必然有 ( ) ≤ (′)。

算法的最优性： 对于任意一个状态t，它第一次被加入搜索树时的路径必然是最短路径。

搜索算法：A*算法性能分析

引理3.2： 假设状态s加入搜索树时对应的结点为n，在结点n被加入搜索树后，若对应状态s的任何结点n'出现在边缘集合中，那么必然有$f(n) \leq f(n')$。

这个性质说明，每个状态被加入搜索树时，其路径代价必然小于等于任何将来出现在边缘集合中通向该状态的路径的代价。

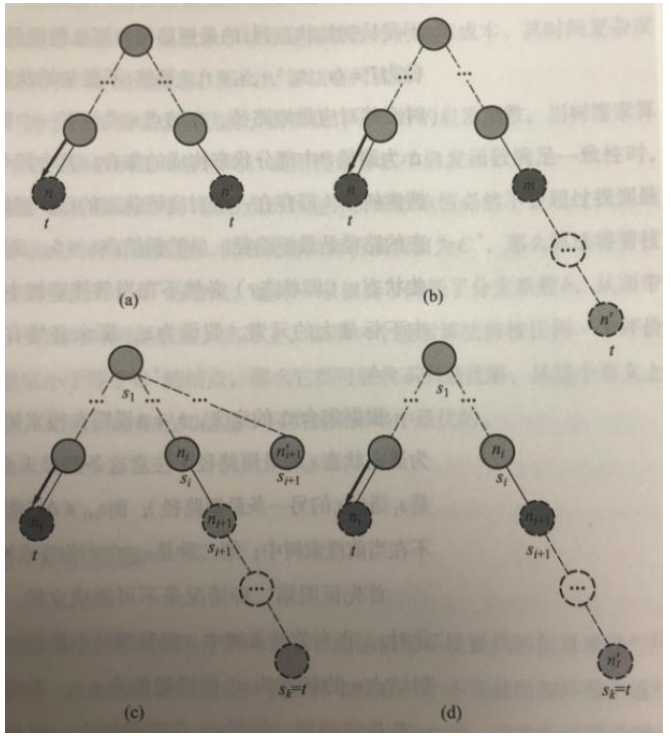


图3.8 图搜索A*算法最优性证明过程示意图。其中已在搜索树中的结点用实线表示，不在搜索树中的结点用虚线表示，可扩展的边缘结点用深灰色标出。节点的代号标注在结点内，结点对应的状态标注在相应结点的下方。

搜索算法：A*算法性能分析

- **最优性：对于任意一个状态t，它第一次被加入搜索树时的路径必然是最短路径。**
  - 为了让A*算法发挥优势，需要设计一个好的启发函数。当树搜索算法中启发函数满足可容性时，或图搜索算法中启发函数满足一致性时，算法扩展任意结点时，该结点所对应评价函数取值必然不会超过找到最优解结点的评价函数值。

提纲

- 搜索算法基础
- 启发式搜索
- 对抗搜索
- 蒙特卡洛树搜索

对抗搜索

- 对抗搜索(Adversarial Search)也称为博弈搜索(Game Search)
- 在一个竞争的环境中，智能体(agents)之间通过竞争实现相反的利益，一方**最大化**这个利益，另外一方**最小化**这个利益。

对抗搜索：主要内容

- **最小最大搜索(Minimax Search)**
  - 最小最大搜索是在对抗搜索中最为基本的一种让玩家来计算最优策略的方法
- **Alpha-Beta剪枝搜索(Pruning Search)**
  - 一种对最小最大搜索进行改进的算法，即在搜索过程中可剪除无需搜索的分支节点，且不影响搜索结果。
- **蒙特卡洛树搜索(Monte-Carlo Tree Search)**
  - 通过采样而非穷举方法来实现搜索。

对抗搜索

- **本课程目前主要讨论在确定的、全局可观察的、竞争对手轮流行动、零和游戏(zero-sum)下的对抗搜索**
  - 所谓零和博弈是博弈论的一个概念，属非合作博弈。指参与博弈的各方，在严格竞争下，一方的收益必然意味着另一方的损失，博弈各方的收益和损失相加总和永远为“零”，双方不存在合作的可能。与“零和”对应，“双赢博弈”的基本理论就是“利己”不“损人”，通过谈判、合作达到皆大欢喜的结果。

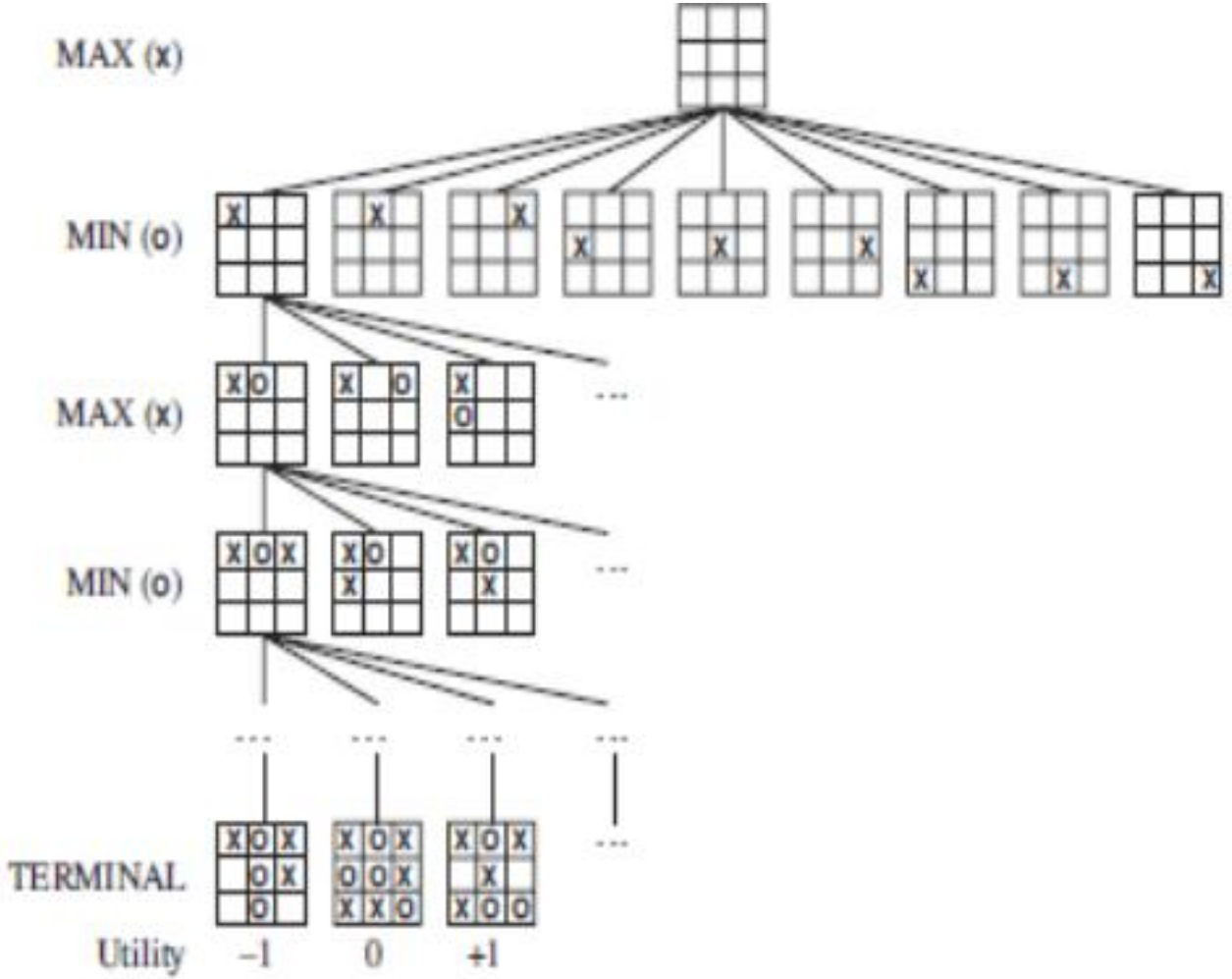
对抗搜索

- 本课程目前主要讨论在确定的、全局可观察的、竞争对手轮流行动、零和游戏(zero-sum)下的对抗搜索
- 两人对决游戏 (MAX and MIN, MAX先走) 可如下形式化描述，从而将其转换为对抗搜索问题

初始状态S_0	游戏所处于的初始状态
玩家$PLAYER(s)$	在当前状态s下，该由哪个玩家采取行动
行动$ACTIONS\ (s)$	在当前状态s下所采取的可能移动
状态转移模型$RESULT\ (s,a)$	在当前状态s下采取行动a后得到的结果
终局状态检测$TERMINAL-TEST\ (s)$	检测游戏在状态s是否结束
终局得分$UTILITY\ (s,p)$	在终局状态s时，玩家p的得分。

对抗搜索

- Tic-Tac-Toe游戏的对抗搜索
  - MAX先行，可在初始状态的9个空格中任意放一个X
  - MAX希望游戏终局得分高、MIN希望游戏终局得分低
  - 所形成游戏树的叶子结点有$9! = 362{,}880$,国际象棋的叶子节点数为10^{40}



Tic-Tac-Toe中部分搜索树

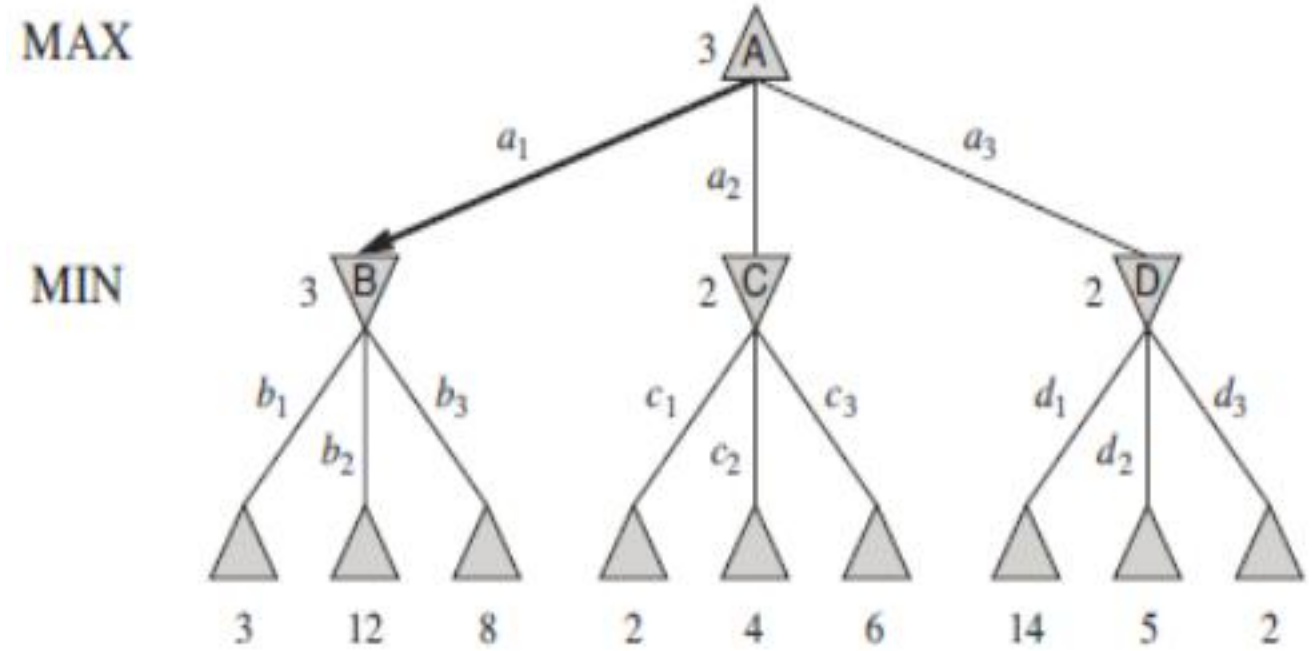
对抗搜索：minimax算法

- **给定一个游戏搜索树，minimax算法通过每个节点的minimax值来决定最优策略。**
  - MAX希望最大化minimax值，而MIN则相反

$$\textsc{Minimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \end{cases}$$

对抗搜索：minimax算法

- **给定一个游戏搜索树，minimax算法通过每个节点的minimax值来决定最优策略。**
  - 通过minimax算法，我们知道，对于MAX而言采取a_1行动是最佳选择，因为这能够得到最大minimax值(收益最大)。



对抗搜索：minimax算法

```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))
```

```
function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s, a)))
  return v
```

```
function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s, a)))
  return v
```

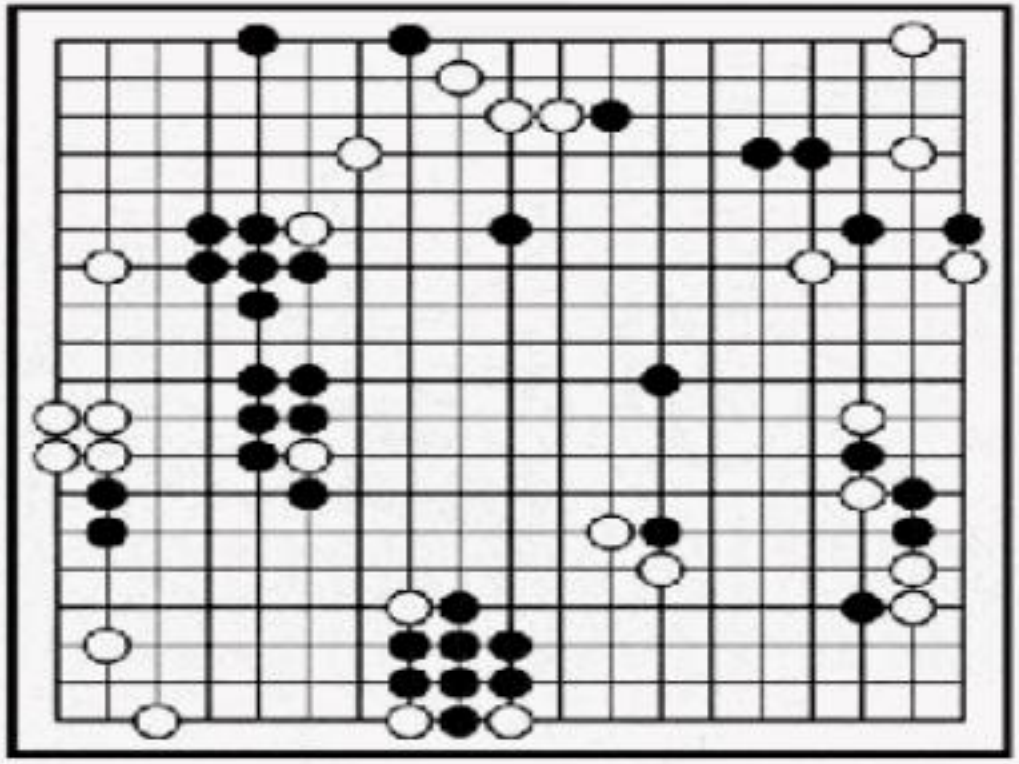
对抗搜索：minimax算法

- **Complete ?**
  - Yes (if tree is finite)
- **Optimal?**
  - Yes (against an optimal opponent)
- **Time complexity ?**
  - $O(b^m)$
  - m 是游戏树的最大深度
  - 在每个节点存在b个有效走法
- **Space complexity ?**
  - $O(b \times m)$ (depth-first exploration)

For chess, b ≈ 35, m ≈100
for "reasonable" games
→ exact solution
completely infeasible

对抗搜索： minimax算法

- 枚举当前局面之后每一种下法，然后计算每个后续局面的赢棋概率，选择概率最高的后续局面



对抗搜索：minimax算法

- **优点:**
  - 算法是一种简单有效的对抗搜索手段
  - 在对手也“尽力而为”前提下，算法可返回最优结果
- **缺点:**
  - 如果搜索树极大，则无法在有效时间内返回结果
- **改善:**
  - 使用alpha-beta pruning算法来减少搜索节点
  - 对节点进行采样、而非逐一搜索 (i.e., MCTS)

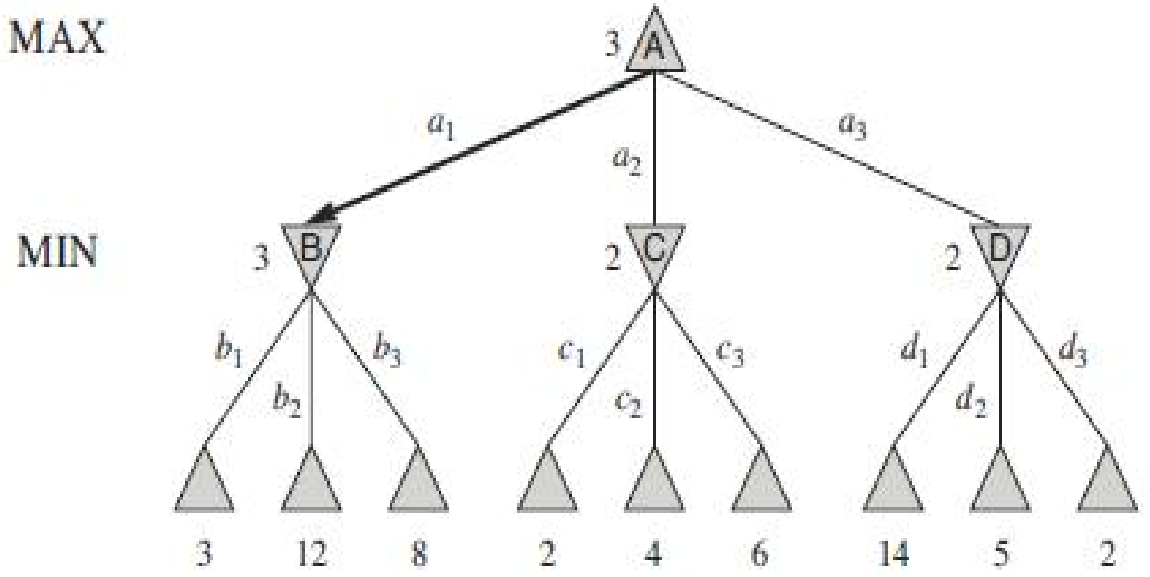
对抗搜索：Alpha-Beta 剪枝搜索

- 在极小化极大算法(minimax算法)中减少所搜索的搜索树节点数。该算法和极小化极大算法所得结论相同，但剪去了不影响最终结果的搜索分枝。

$$MINIMAX(root)$$
$$= \max(\min(3.12,8), \min(2,x,y), \min(14,5,2))$$
$$= \max(3, \min(2,x,y), 2)$$
$$= \max(3,z,2) = 3$$

$$where\ z = \min(2,x,y) \leq 2$$

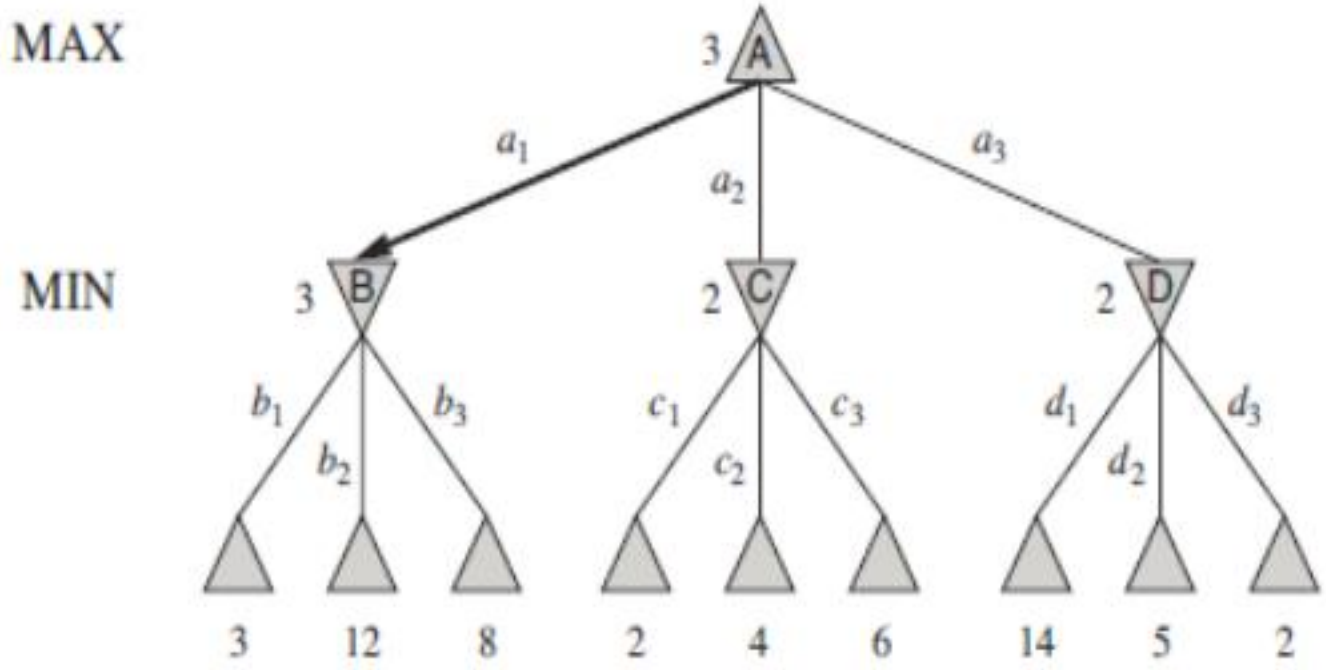
可以看出：根节点(即 MAX选手)的选择与x和y 两个值无关(因此，x和y可以被剪枝去除)



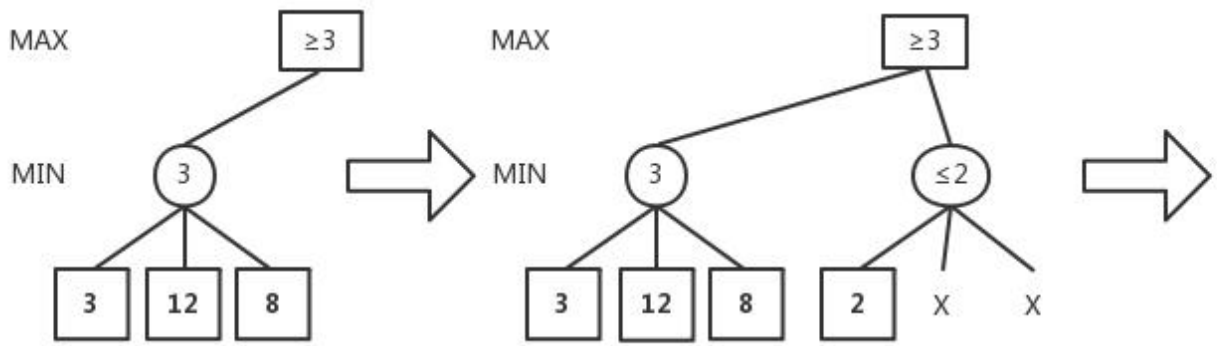
对抗搜索：Alpha-Beta 剪枝搜索

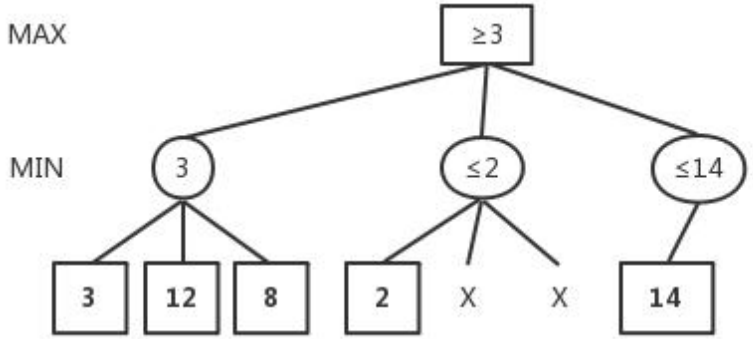
- 在极小化极大算法(minimax算法)中减少所搜索的搜索树节点数。该算法和极小化极大算法所得结论相同，但剪去了不影响最终结果的搜索分枝。

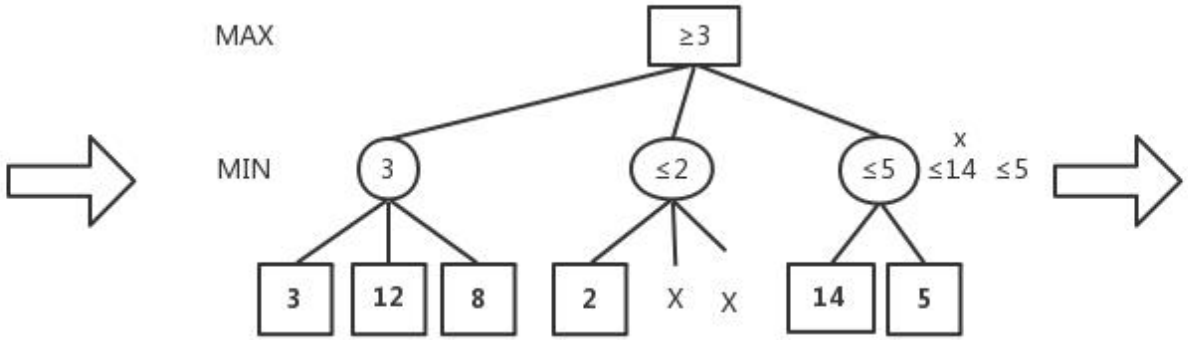
图中MIN选手所在的节点C下属分支4和6与根节点最终优化决策的取值无关，可不被访问。

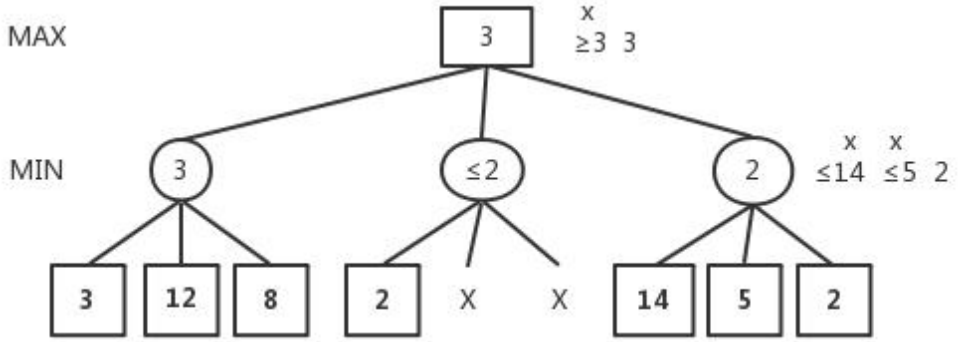


对抗搜索：Alpha-Beta 剪枝搜索



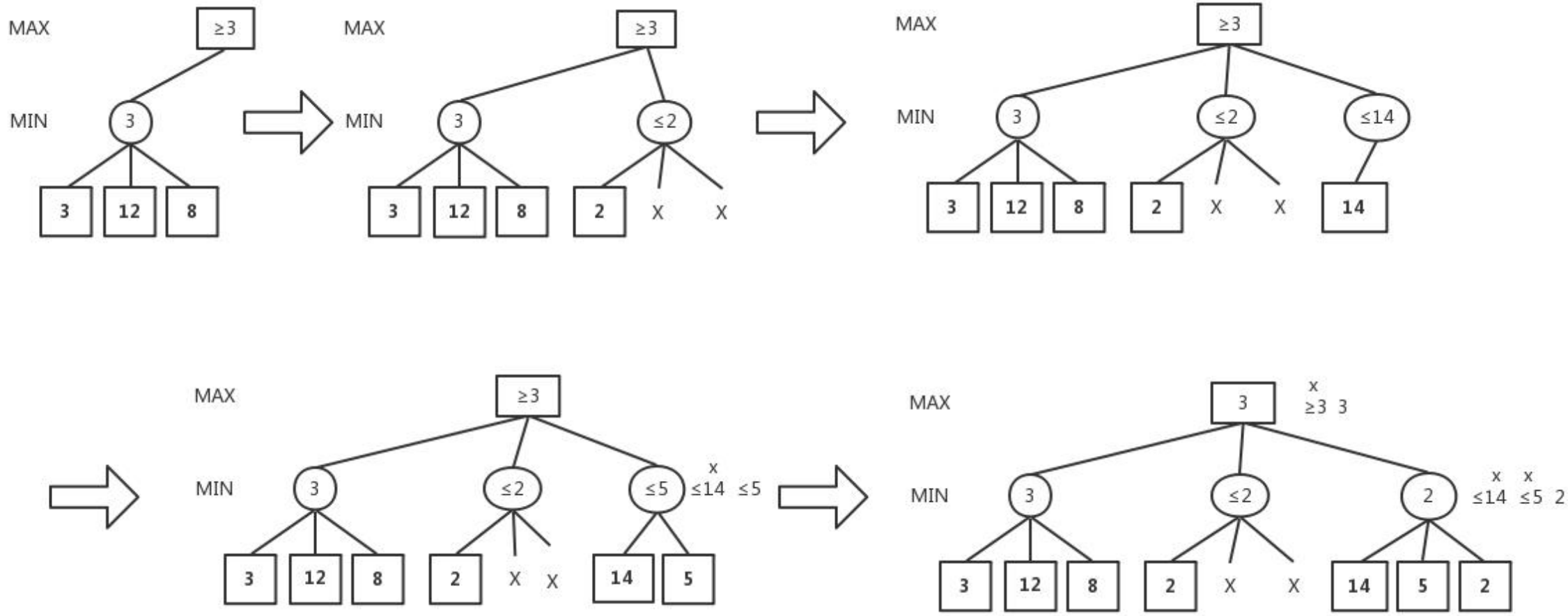






Alpha值(α)	MAX节点目前得到的最高收益
Beta值(β)	MIN节点目前可给对手的最小收益
α和β的值初始化分别设置为$-\infty$和∞	

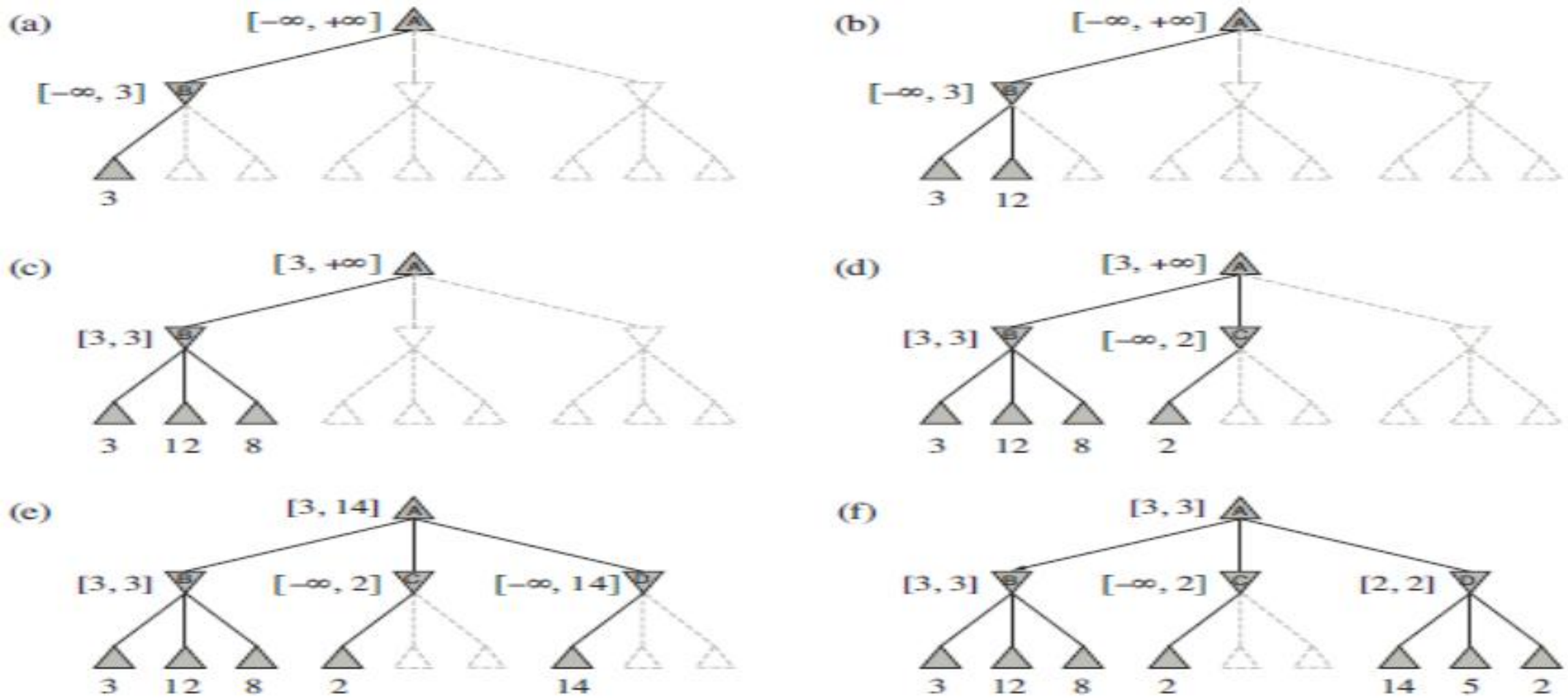
对抗搜索：Alpha-Beta 剪枝搜索



Alpha值(α)	MAX节点目前得到的最高收益
Beta值(β)	MIN节点目前可给对手的最小收益
α和β的值初始化分别设置为$-\infty$和∞	

对抗搜索： Alpha-Beta 剪枝搜索

- 从α和β的变化来理解剪枝过程

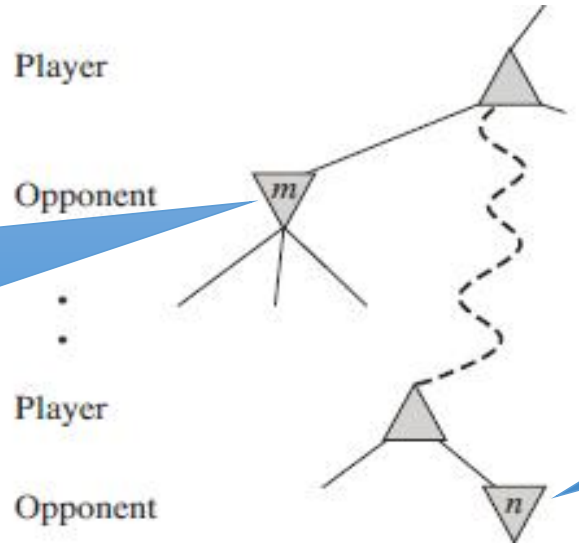


对抗搜索：如何利用Alpha-Beta 剪枝

Alpha值(α)	玩家MAX(根节点)目前得到的最高收益
	假设n是MIN节点，如果n的一个后续节点可提供的收益小于α，则n及其后续节点可被剪枝
Beta值(β)	玩家MIN目前给对手的最小收益
	假设n是MAX节点，如果n的一个后续节点可获得收益大于β，则n及其后续节点可被剪枝
α和β的值初始化分别设置为$-\infty$和∞	

在图中$m > n$，因此n右边节点及后续节点就被剪枝掉了

对手节点(min节点)在这里可提供的最大收益是？



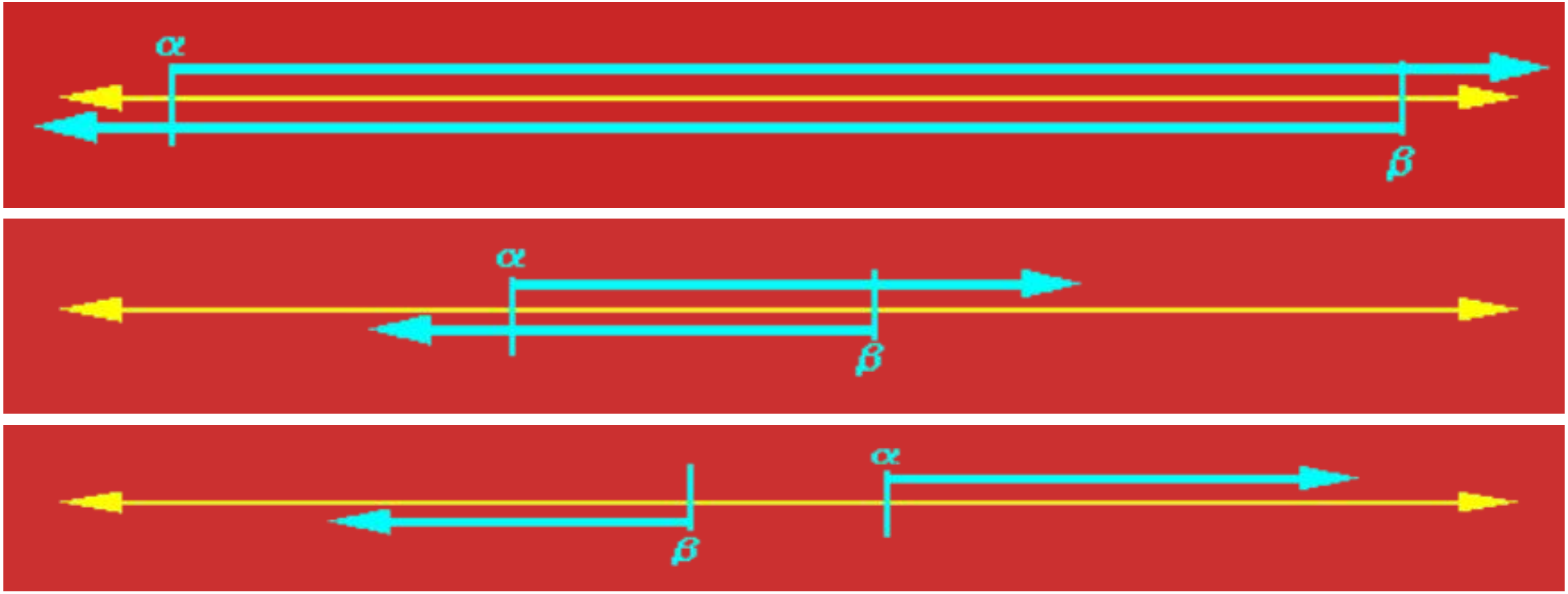
对手节点(min节点)在这里可提供的最大收益是？

对抗搜索：如何利用Alpha-Beta 剪枝

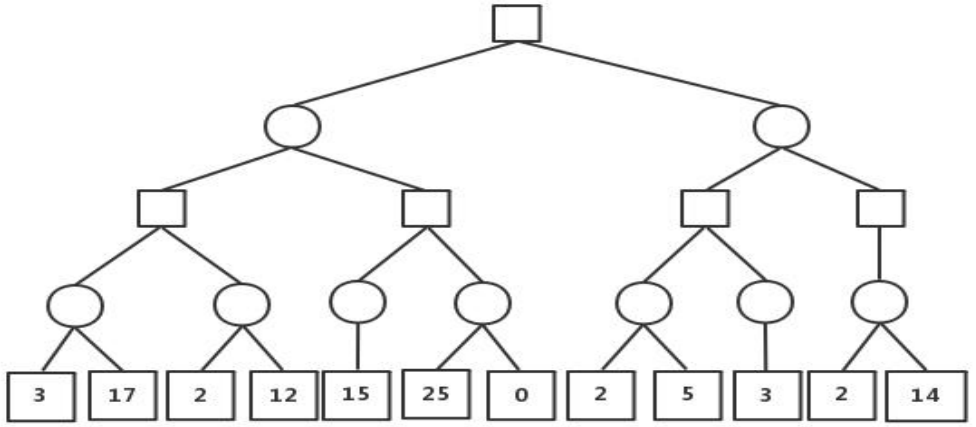
- α为可能解法的最大上界
- β 为可能解法的最小下界
  - 如果节点N是可能解法路径中的一个节点，则其产生的收益一定满足如下条件：$\alpha \leq reward(N) \leq \beta$(其中$reward(N)$是节点 产生的收益)
  - 每个节点有两个值，分别是 和 。节点的 和 值在搜索过程中不断变化。其中， 从负无穷大$(-\infty)$逐渐增加、 从正无穷大(∞)逐渐减少。如果一个节点中 > ，则该节点的后续节点可剪枝。

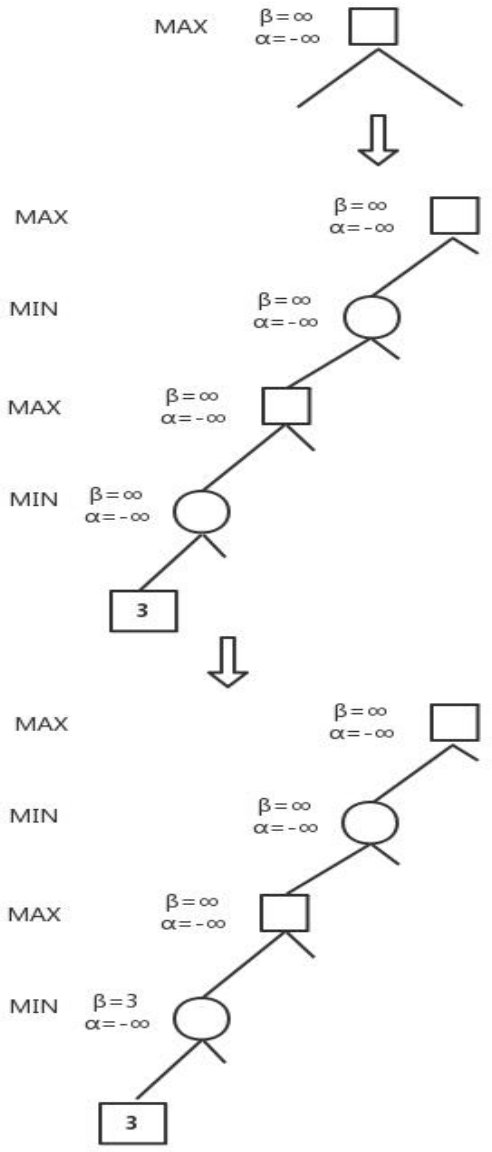
对抗搜索：如何利用Alpha-Beta 剪枝

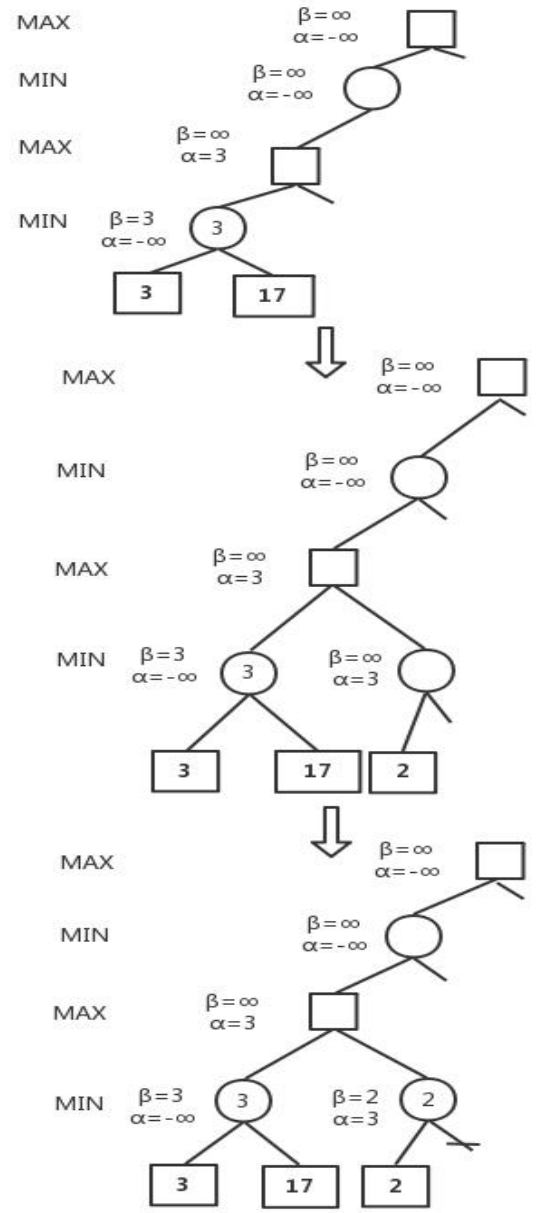
- α为可能解法的最大上界
- β 为可能解法的最小下界



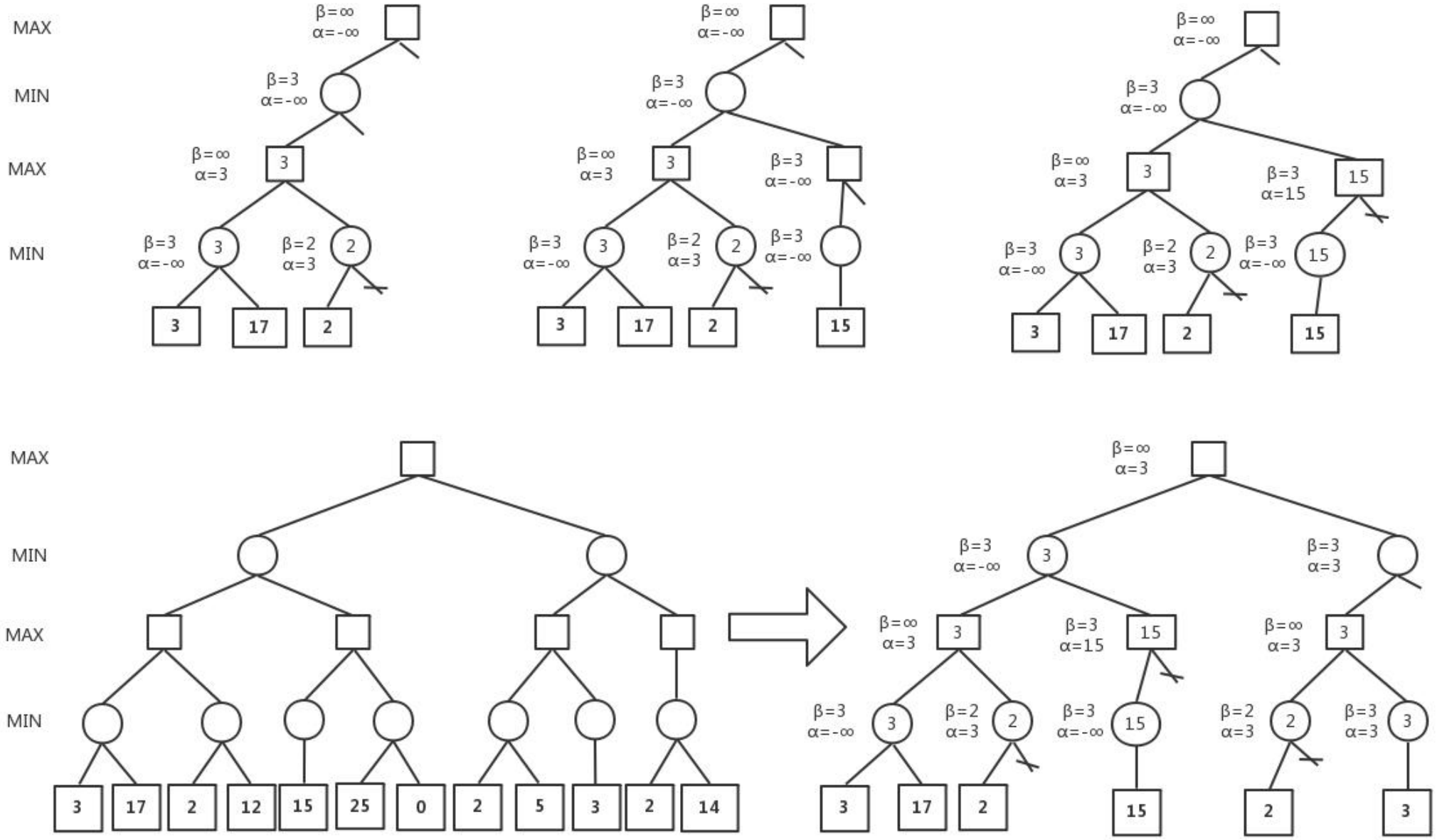
对抗搜索：Alpha-Beta 剪枝搜索示意







对抗搜索：Alpha-Beta 剪枝搜索示意



对抗搜索：Alpha-Beta 剪枝搜索的算法描述

```
function ALPHA-BETA-SEARCH(state) returns an action
    v ← MAX-VALUE(state, −∞, +∞)
    return the action in ACTIONS(state) with value v
```

```
function MAX-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for each a in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v
```

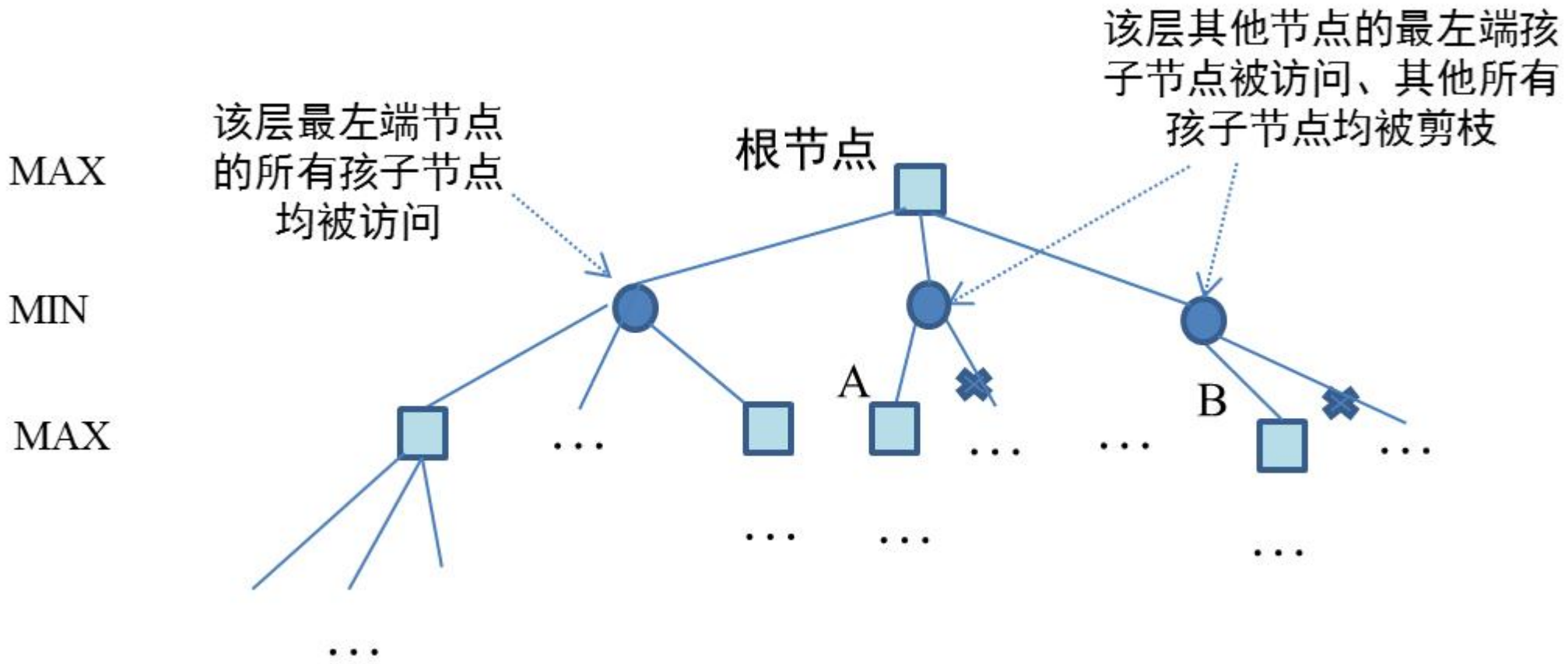
```
function MIN-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for each a in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```

对抗搜索：Alpha-Beta 剪枝搜索的性质

- 剪枝本身不影响算法输出结果
- 节点先后次序会影响剪枝效率
- 如果节点次序“恰到好处”，Alpha-Beta剪枝的时间复杂度为$O(b^{\frac{m}{2}})$，最小最大搜索的时间复杂度为$O(b^m)$

b	分支因子，即搜索树中每个节点最大的分支数目
m	搜索树中路径的最大可能长度

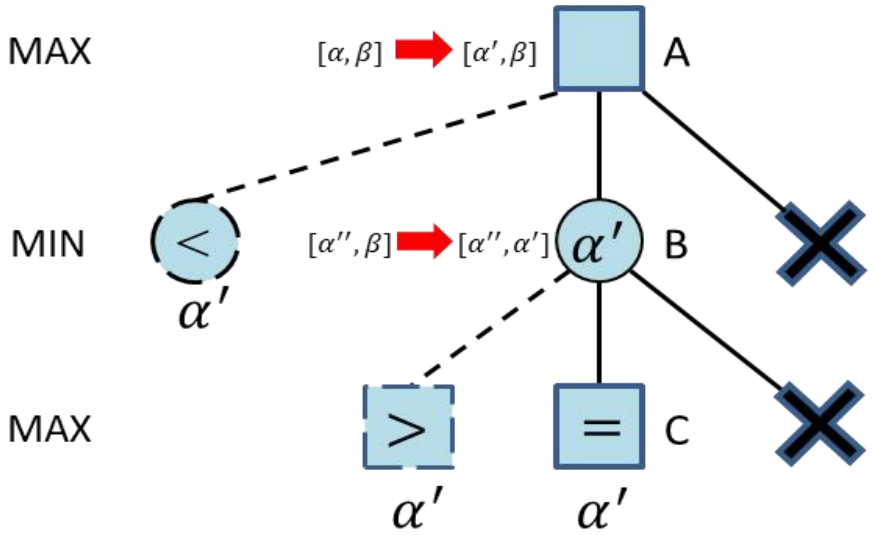
对抗搜索：Alpha-Beta 剪枝算法性能分析

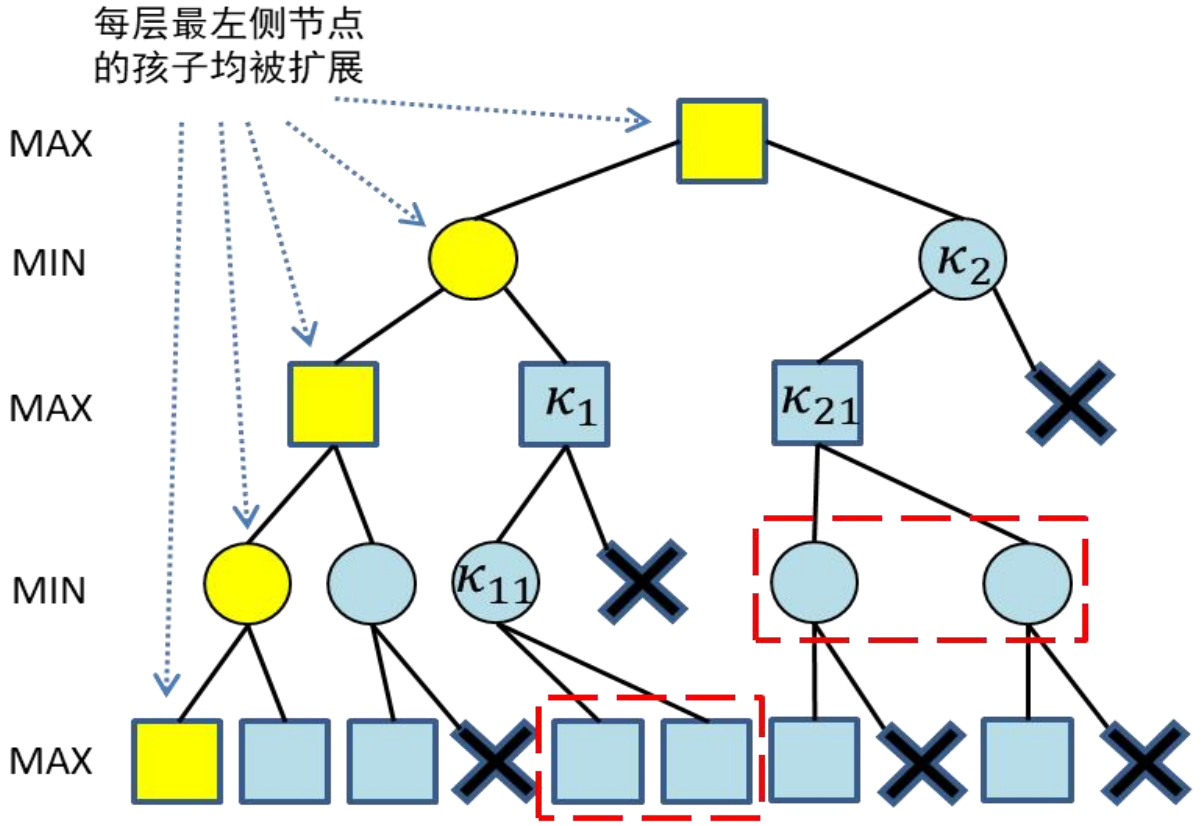


最优状况下的剪枝结果示意图，每一层最左端结点的所有孩子结点均被访问，其他节点仅有最左端孩子结点被访问、其他孩子结点被剪枝。

对抗搜索：Alpha-Beta 剪枝算法性能分析

- 观察：如果一个节点导致了其兄弟节点被剪枝，可知其孩子节点必然被扩展

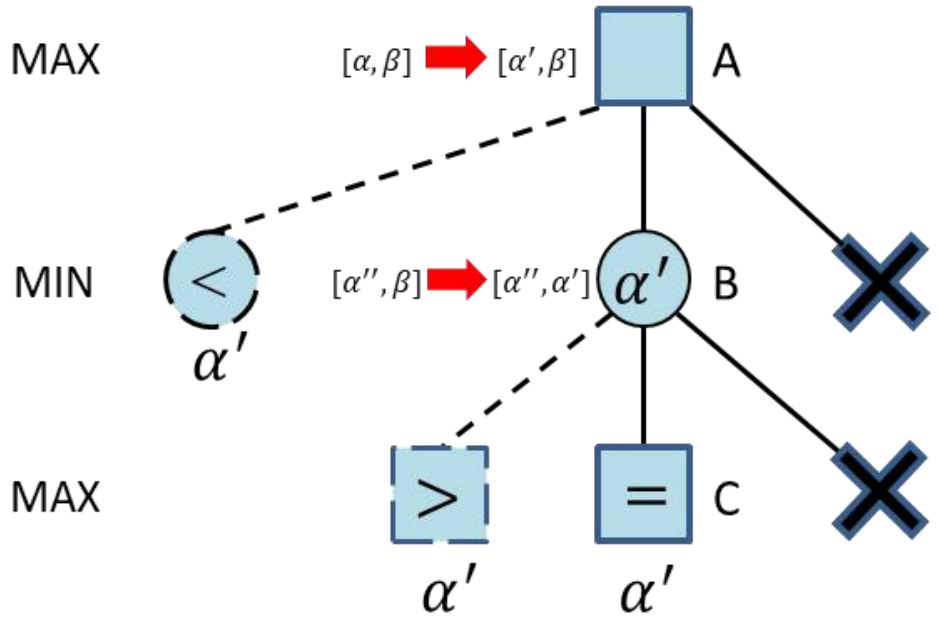




κ_{11}和κ_{21}两个节点分别导致了其右端的兄弟被剪枝，于是可以肯定κ_{11}和κ_{21}全部孩子节点必然被扩展。

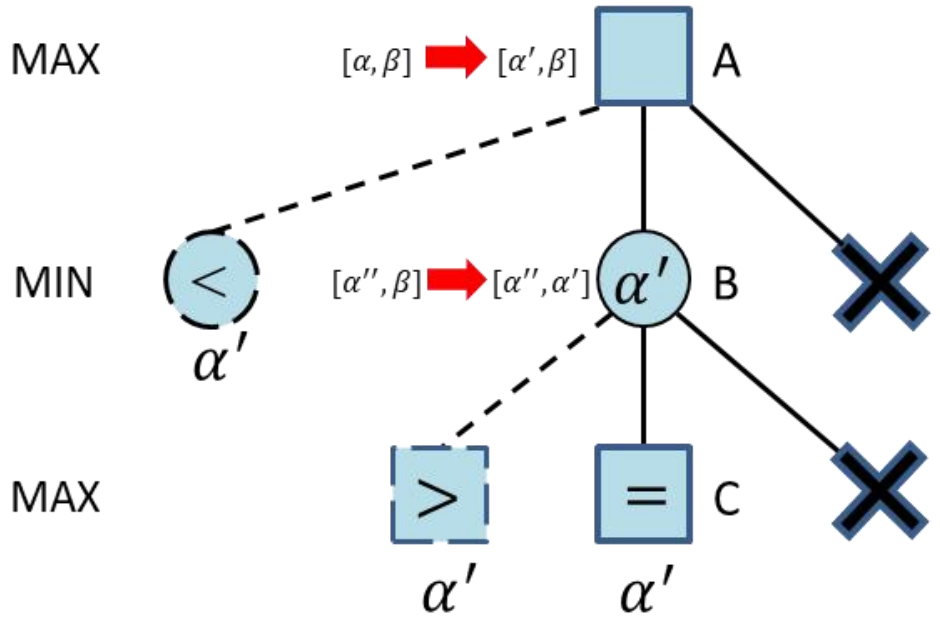
对抗搜索：Alpha-Beta 剪枝算法性能分析

- 不妨假设A结点为MAX节点。
  - 假设节点A对应的初值是 [α, β]
  - 如果A的孩子结点B导致了B右端的兄弟结点被剪枝。由于剪枝直到B的收益分数计算完后才发生，可知结点B必然是A已扩展孩子结点中收益分数最大的结点；同时根据剪枝的发生条件，可知B的收益分数α'必然满足'>。



对抗搜索：Alpha-Beta 剪枝算法性能分析

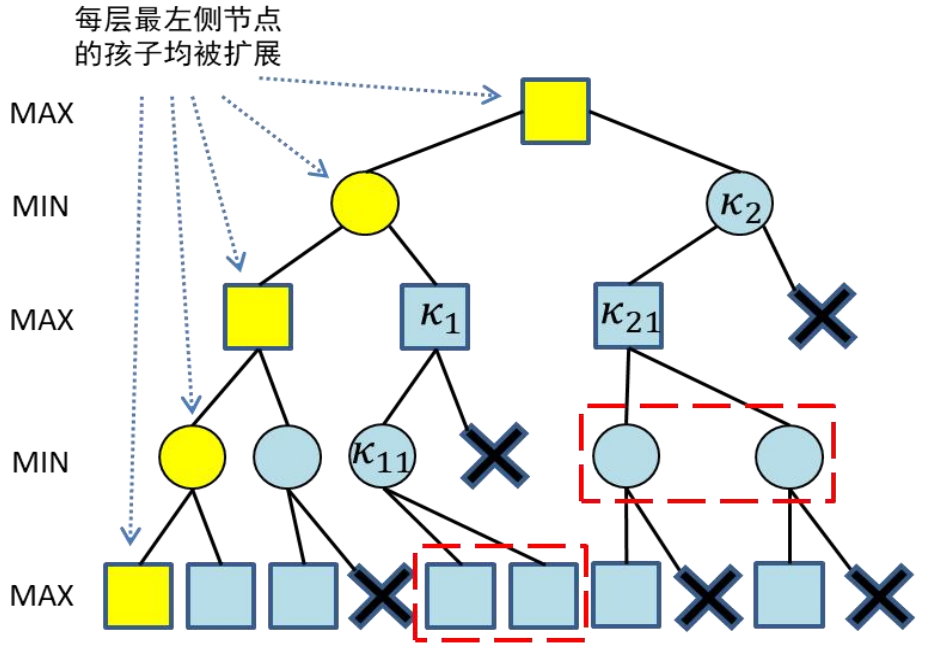
- **不妨假设A结点为MAX节点。**
  - 假设扩展结点B时的上下界为[α″,β]，其中α″来自结点B被扩展时结点A的下界alpha。根据假设，B结点至少有一个孩子结点(C结点)被扩展，因此α″≤α′，同时又根据α′>β，因此推出α″<β。如果结点C导致B的其余子结点被剪枝，仿照对结点B的分析，可知结点C是结点B已扩展子结点中收益分数最小的一个。于是，结点B的收益分数必然是结点C的收益分数，即minimax(β)= minimax(α′)=α′，同时C的收益分数必然满足α′<α″，该式与前面的不等式矛盾。



由此不难归纳出结论：如果某个结点导致了其右侧兄弟结点被剪枝，那么该节点的所有孩子结点必然已被全部扩展

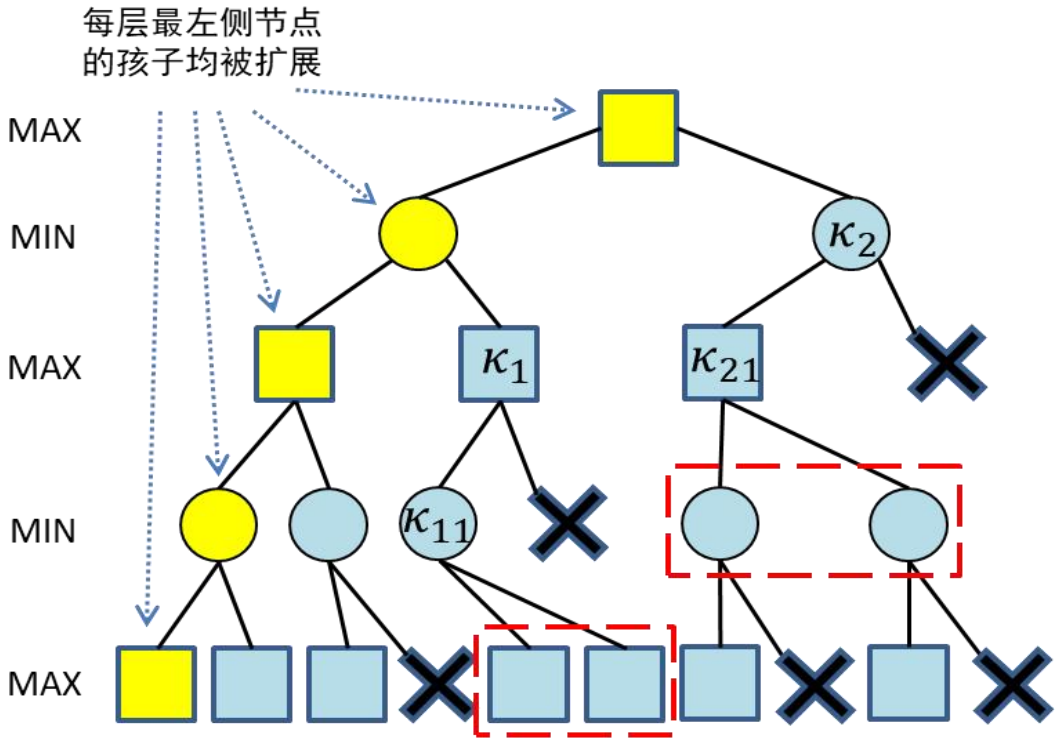
对抗搜索：Alpha-Beta 剪枝算法性能分析

- 下图展示了alpha-beta剪枝算法效率最优的情况
  - 为了方便说明，假设图中每个结点恰好有 b 个子节点
  - 搜索树每层最左端结点的孩子结点必然全部被扩展。其他结点的孩子结点被扩展情况分为如下两类
    - 1)只扩展其最左端孩子结点；
    - 2)它是其父亲结点唯一被扩展的子结点，且它的所有子结点(如果存在)一定被扩展。



对抗搜索：Alpha-Beta 剪枝算法性能分析

- 最左侧子节点外其他子节点
  - 1)只扩展其最左端孩子结点；
  - 2)它是其父亲结点唯一被扩展的子结点，且它的所有子结点(如果存在)一定被扩展。



[illegible]1和[illegible]2两个结点属于上述第一类结点，即仅扩展它们的最左端孩子结点。[illegible]11和[illegible]21两个结点属于上述第二类结点，即它们是其父亲结点唯一被扩展的孩子结点，且它的所有孩子结点一定被扩展。

对抗搜索：Alpha-Beta 剪枝算法性能分析

- 最左侧子节点外其他子节点
  - 1)只扩展其最左端孩子结点；
  - 2)它是其父亲结点唯一被扩展的子结点，且它的所有子结点(如果存在)一定被扩展。

Alpha-Beta剪枝在最优效率下扩展的结点数量为$O(b^{\frac{m}{2}})$，比起原本最小最大搜索的$O(b^m)$有显著提升。