搜索求解

主讲：刘夏雷、郭春乐、王亚星
南开大学计算机学院
https://mmcheng.net/xliu/

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

提纲

- 搜索算法基础
- 启发式搜索
- 对抗搜索
- 蒙特卡洛树搜索

课程回顾-逻辑与推理

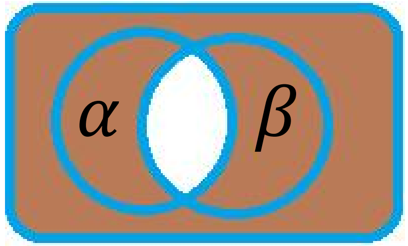
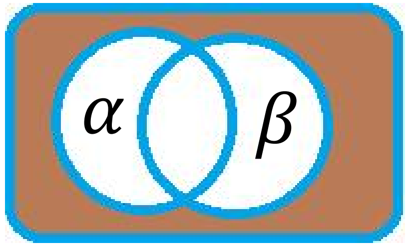
- 命题逻辑
- 谓词逻辑
- 知识图谱推理
- 因果推理

命题逻辑

- **可通过命题联结词对已有命题进行组合，得到新命题。**
  - 假设存在命题p 和q ，下面介绍五种主要的命题联结词：

命题连接符号	表示形式	意义
与(and)	$p \wedge q$	命题合取(conjunction), 即“p且q”
或(or)	$p \vee q$	命题析取(disjunction), 即“p或 q”
非（not）	$\neg p$	命题否定(negation), 即“非p”
条件(conditional)	$p \rightarrow q$	命题蕴含(implication), 即“如果p 则 q”
双向条件 (bi-conditional)	$p \leftrightarrow q$	命题双向蕴含(bi-implication), 即 “p 当且仅当 q”

命题逻辑：若干逻辑等价命题的解释

$(\alpha \longrightarrow \beta) \equiv \neg\beta \longrightarrow \neg\alpha$ (逆否命题)	秋天天气变凉⟶大雁南飞越冬≡大雁没有南飞越冬⟶秋天天气没有变凉
	$x \geq 0 \longrightarrow x^2 \geq 0 \equiv x^2 < 0 \longrightarrow x < 0$
$(\alpha \longrightarrow \beta) \equiv \neg\alpha \vee \beta$ (蕴涵消除)	α为假、则命题恒为真；α为真、则β须为真
$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$ (De Morgan)	
$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$ (De Morgan)	

谓词逻辑：量词

- 全称量词(universal quantifier, ∀)
  - 全称量词用符号∀表示，表示一切的、凡是的、所有的、每一个等。$\forall x$表示定义域中的所有个体，$\forall x P(x)$表示定义域中的所有个体具有性质P
- 存在量词(existential quantifier, ∃)
  - 存在量词用符号∃表示，表示存在、有一个、某些等。$\exists x$表示定义域中存在一个或若干个个体，$\exists x P(x)$表示定义域中存在一个个体或若干个体具有性质P
- 全称量词和存在量词统称为量词。

知识图谱推理: FOIL (First Order Inductive Learner)

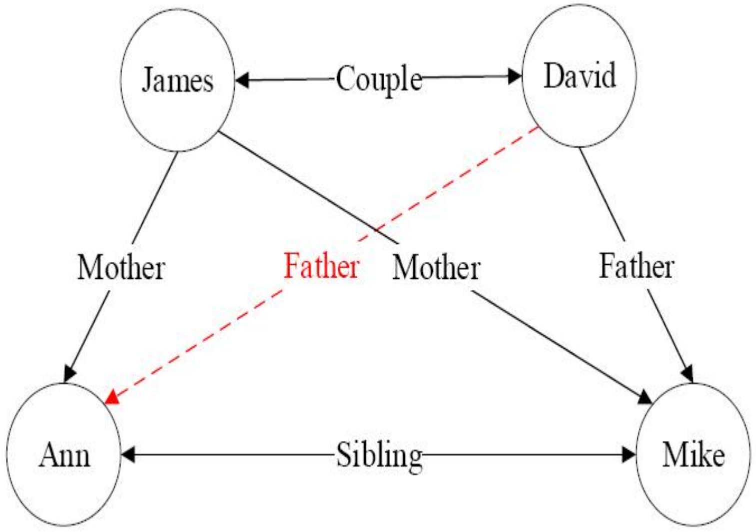
- 推理手段: positive examples + negative examples + background knowledge examples ⟹hypothesis

$$\underbrace{(\forall x)(\forall y)(\forall z)\big(Mother(z,\, y) \wedge\ Couple(x, z)}_{\text{前提约束谓词（学习得到）}} \rightarrow \underbrace{Father(x,\, y)}_{\text{目标谓词（已知）}}\big)$$

知识图谱推理：路径排序

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

Score($Father(David, \ Ann)$)



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）

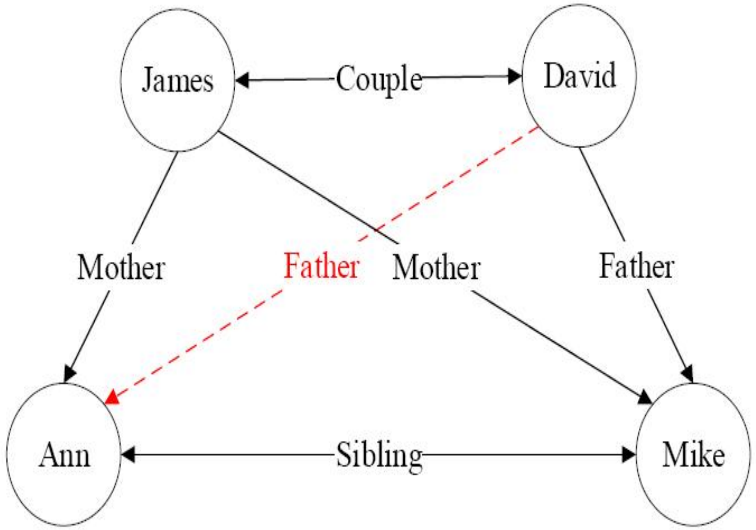
(James, Ann)对应路径：$Mother \rightarrow Sibling$

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

知识图谱推理：路径排序

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

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)：$\{[0,\ 1,\ 0,\ 0],\ -1\}$

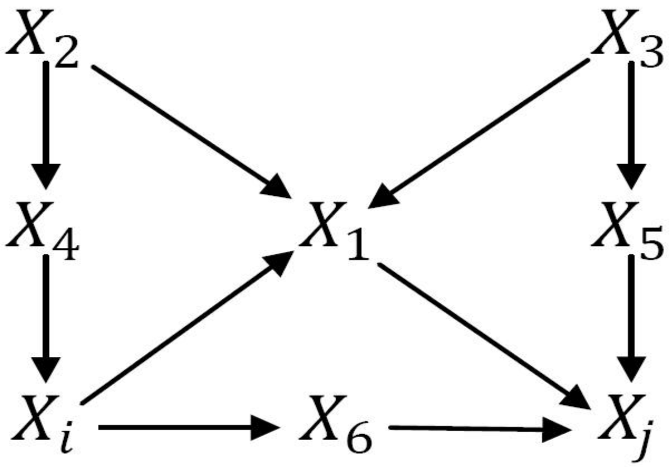
(James, Ann)：$\{[0,\ 0,\ 1,\ 0],\ -1\}$

(James, Mike)：$\{[0,\ 0,\ 1,\ 1],\ -1\}$

4. 依据训练样本，训练分类器M

因果推理：有向无环图（DAG）

- **一个有向无环图唯一地决定了一个联合分布**
- **一个联合分布不能唯一地决定有向无环图**
  - 反过来的结论不成立
  - 如联合分布$P(X_1, X_2) = P(x_1)P(x_2|x_1) = P(x_2)P(x_1|x_2)$

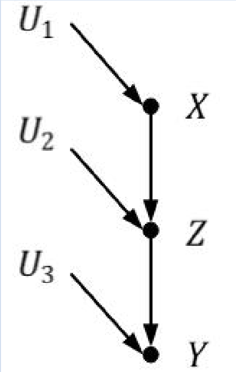
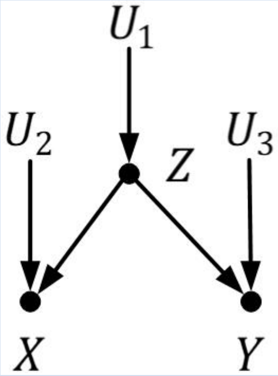
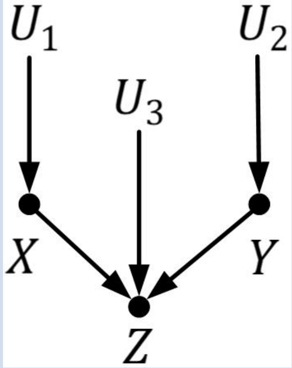


有向无环图DAG

联合分布可表示为：

$$
\begin{aligned}
&P\left(X_1, X_2, X_3, X_4, X_5, X_6, X_i, X_j\right)\\
&= P(X_2) \times P(X_3) \times P(X_1|X_2, X_3, X_i)\\
&\times P(X_4|X_2) \times P(X_5|X_3) \times P(X_6|X_i) \times P(X_i|X_4)\\
&\times P(X_j|X_1,\ X_5,\ X_6)
\end{aligned}
$$

因果推理：*D* -分离

链结构(chain)	分连结构(fork)	汇连（或碰撞）结构(collider)
		
Z和X是相关的	X和Z是相关的	Z和X是相关的
Y和Z是相关的	Y和Z是相关的	Z和Y是相关的
Y和X很有可能是相关的	Y和X很有可能是相关的	Y和X是相互独立的
给定Z时，Y和X是条件独立的	**给定Z时，Y和X是条件独立的**	**给定Z时，Y和X是相关的**

因果推理：D -分离

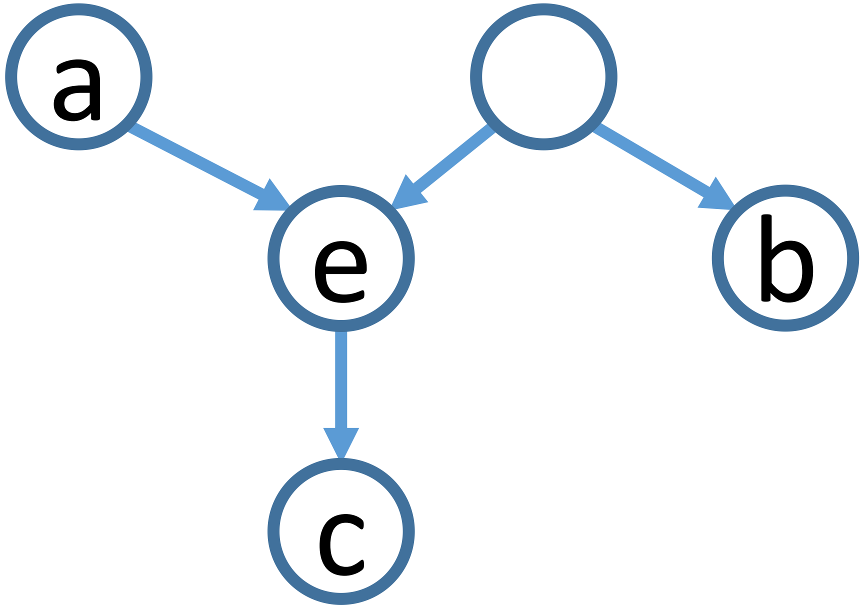
- **D-分离：对于一个DAG图，如果A、B、C是三个集合（可以是单独的节点或者是节点的集合），为了判断A和B 是否是 C 条件独立的， 在DAG图中考虑所有A和B之间的路径(不管方向)。对于其中的一条路径，如果满足以下两个条件中的任意一条，则称这条路径是阻塞（block）的：路径中存在节点X**
  - X是链结构或分连结构节点, 且$X \in C$
  - X是汇连结构节点，并且X或X 后代不包含在 C中

多选题 3分

设置

下面描述中正确的有

- A a和b是条件c下独立的
- B a和b是条件e下独立的
- C a和b是条件f下独立的



提交

因果推理：干预(intervention)和do算子(do-calculus)

- DAG中具有链接箭头的节点之间存在某种“因果关系”。
- 要在 DAG 上引入“因果”的概念，需要引进do 算子
  - do-calculus的意思可理解为“干预”（intervention）
- 在 DAG 中，$do(X_i) = x_i'$，表示将DAG中指向节点X_i的有向边全部切断，并且将X_i的值固定为常数x_i'
- 在这样操作后，所得到新的DAG中变量联合分布为：

$$P(x_1, x_2, \cdots, x_d | do(X_i) = \boldsymbol{x_i}')$$

因果推理：反事实推理 (counterfactual model)

- “反事实”框架是科学哲学家大卫•刘易斯（David Lewis）等人提出的推断因果关系的标准。
- 事实是指在某个特定变量(A)的影响下可观测到的某种状态或结果(B)。“反事实”是指在该特定变量(A) 取负向值时可观测到的状态或结果(B')
- 条件变量对于结果变量的因果性就是A成立时B的状态与A取负向值时“反事实”状态(B')之间的差异。
- 如果这种差异存在且在统计上是显著的，说明条件变量与结果变量存在因果关系。

推理总结

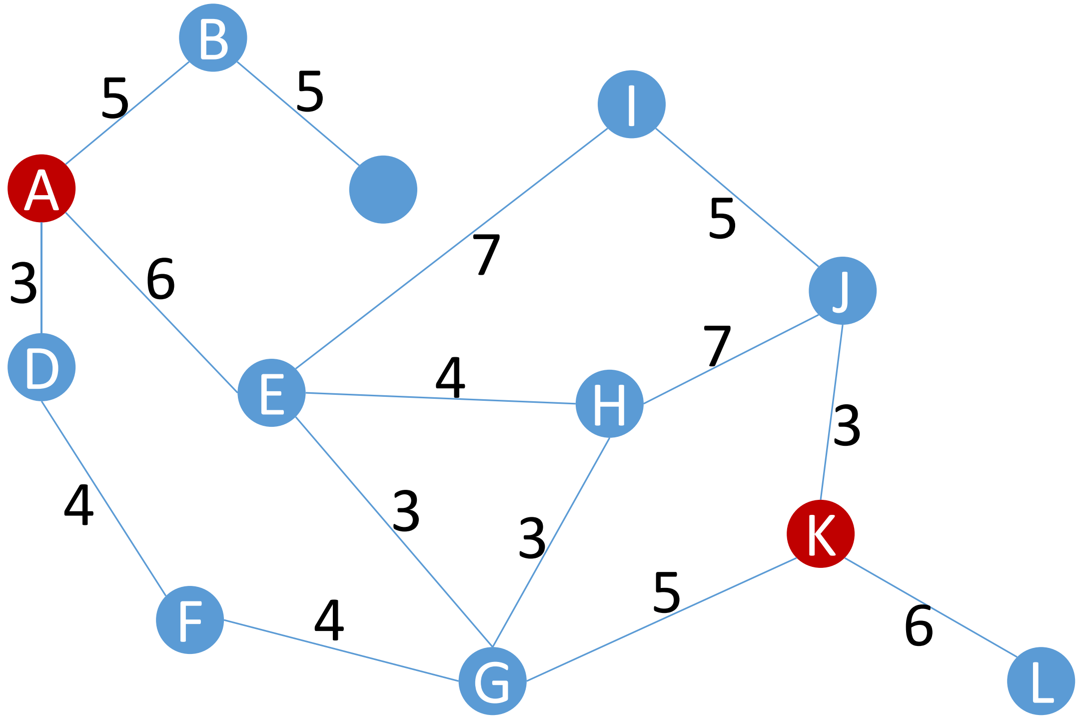
推理方法	推理方式	说明
归纳推理	如果A_i(i为若干取值)，那么B	从若干事实出发推理出一般性规律
演绎推理	如果A，那么B	A是B的前提、但不是唯一前提，因此A是B的充分条件。当然，在特殊情况下A也可为B的充分必要条件
因果推理	因为A，所以B	A是B的唯一前提，因此“如果没有A，那么没有B”也成立。

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

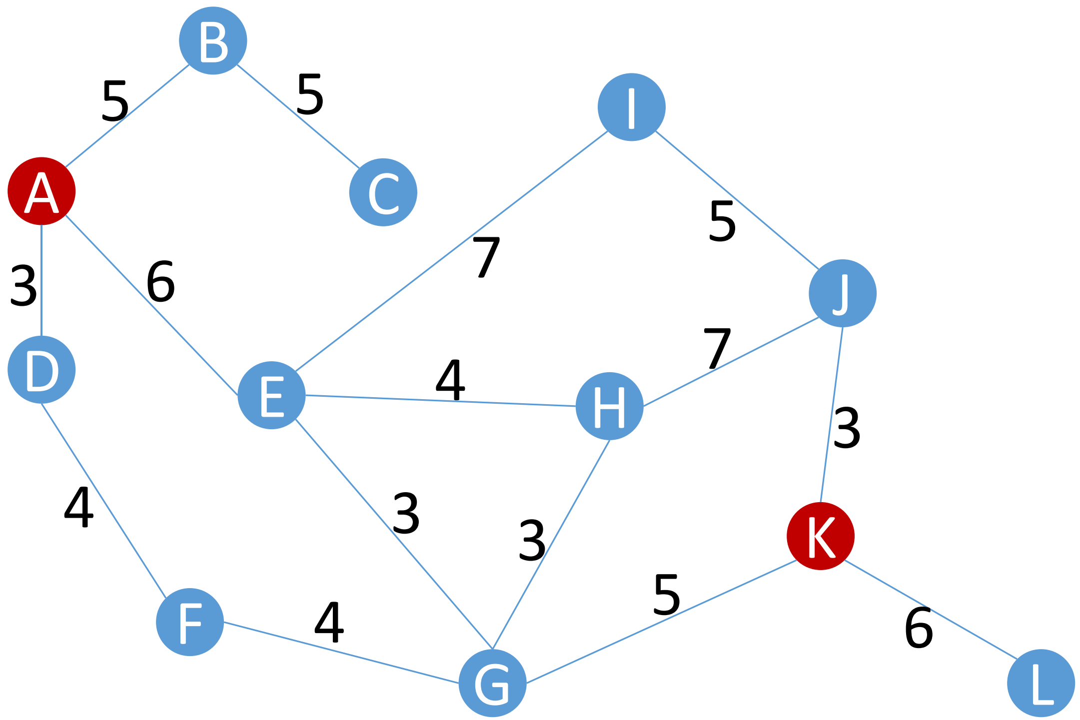
- **状态**
  - 对智能体和环境当前情形的描述。例如，在最短路径问题中，城市可作为状态。将原问题对应的状态称为初始状态。
- **动作**
  - 从当前时刻所处状态转移到下一时刻所处状态所进行操作。一般而言这些操作都是离散的。



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

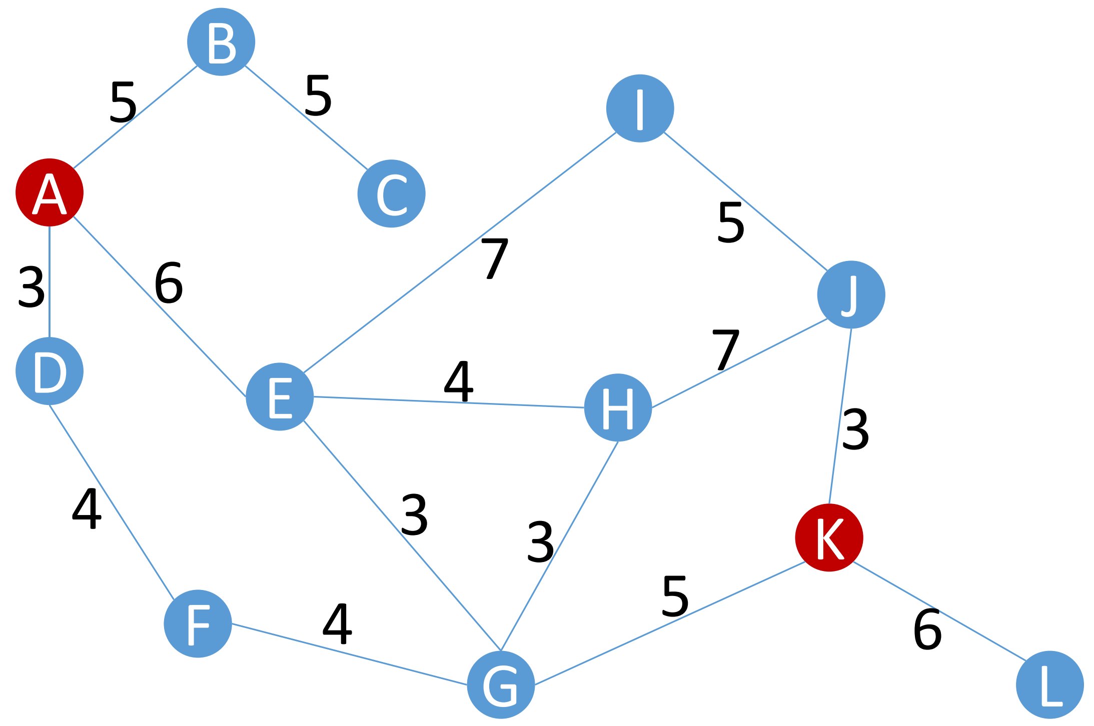
- **状态转移**
  - 对智能体选择了一个动作之后，其所处状态的相应变化
- **路径/代价**
  - 一个状态序列。该状态序列被一系列操作所连接。
  - 如从A到K所形成的路径。
- **目标测试**
  - 评估当前状态是否为目标状态

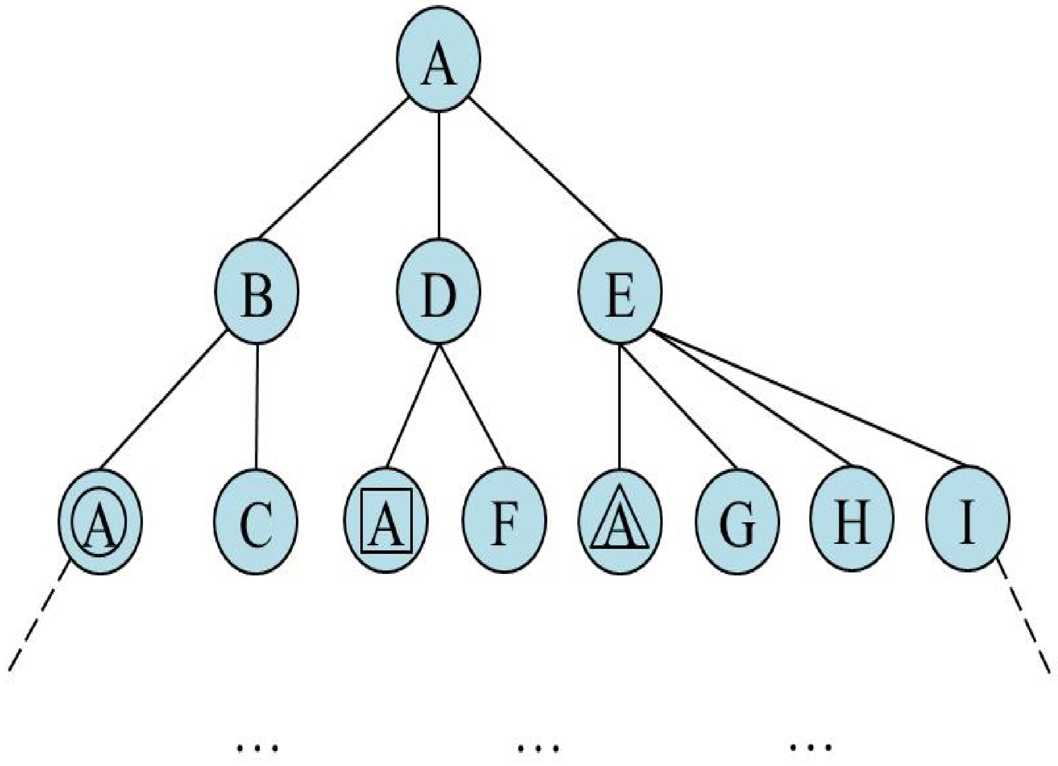


问题：寻找从城市A到城市K之间行驶时间最短路线？

搜索树：用一棵树来记录算法探索过的路径

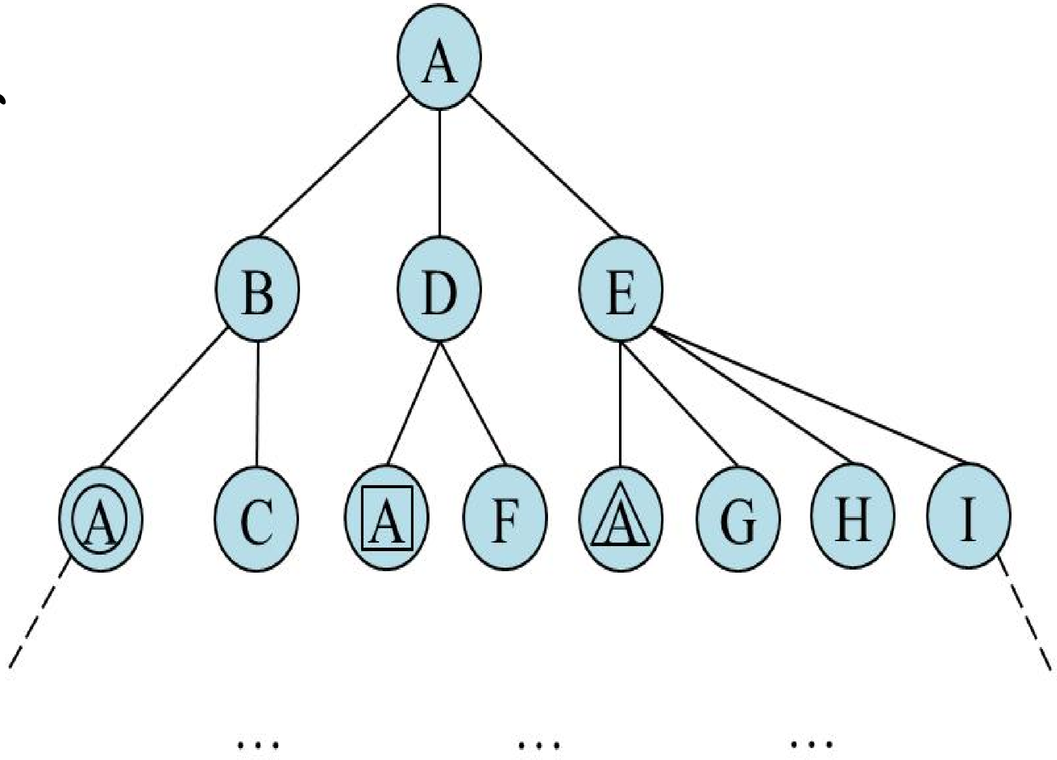
- 搜索算法会时刻记录所有从初始结点出发已经探索过的路径，每次从中选出一条，从该路径末尾状态出发进行一次状态转移，探索一条尚未被探索过新路径。





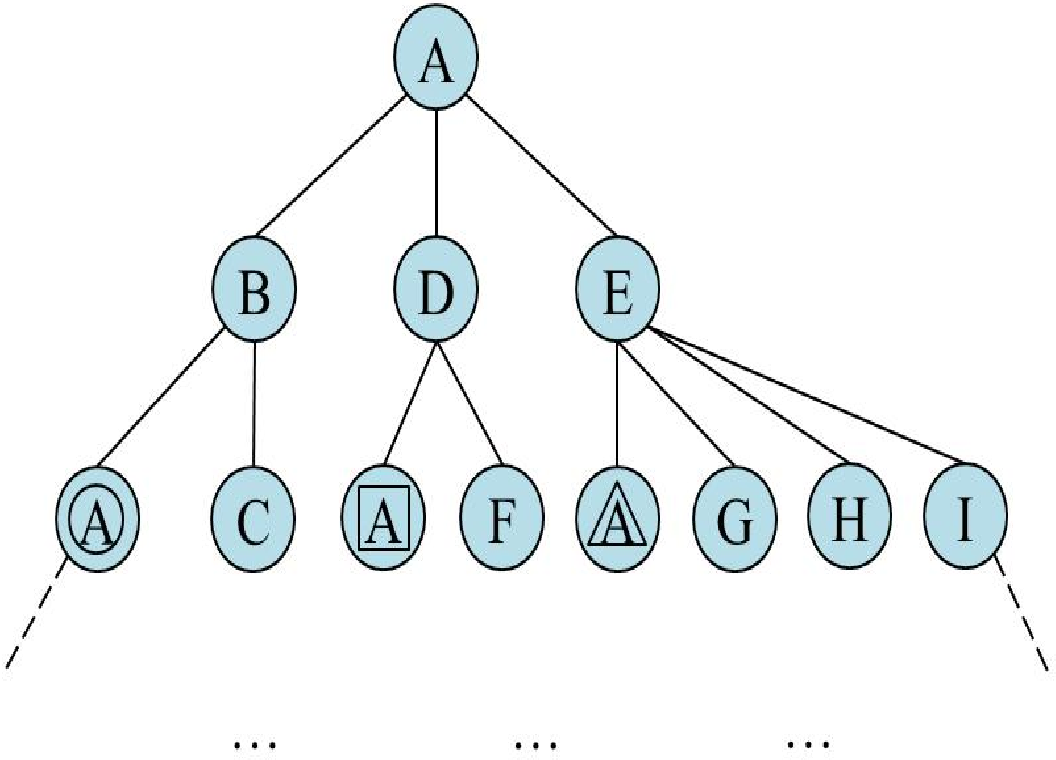
搜索树：用一棵树来记录算法探索过的路径

- **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

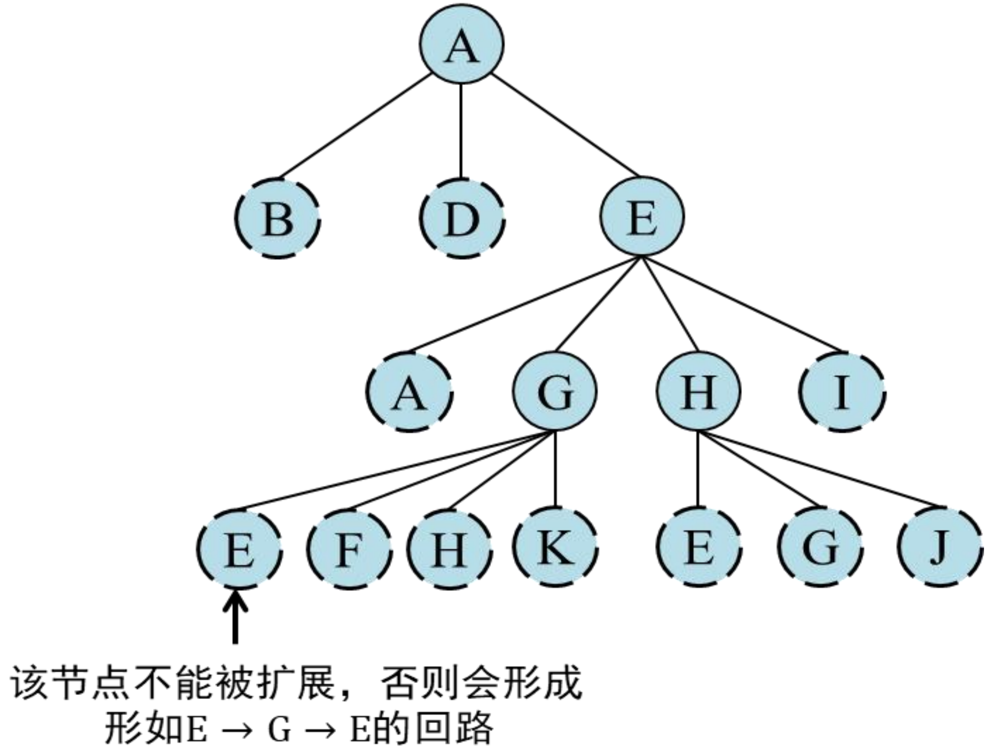
输入： 节点选择函数 pick_from，后继节点计算函数 successor_nodes

输出： 从初始状态到终止状态的路径

```
𝓕 ← {根节点}
while 𝓕 ≠ ∅ do
    n ← pick_from(𝓕)
    𝓕 ← 𝓕 − {n}
    if goal_test(n) then
        return n.path
    end
    𝓕 ← 𝓕 ∪ successor_nodes(n)
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

- **图搜索中，边缘集合中所有产生环路的节点都要被剪枝**
  - 但不会排除所有潜在的可行解
  - 在状态数量有限情况下，采用图搜索策略的算法也是完备的。
  - 维护一个集合，用于存储所有被扩展过的节点状态，这个集合被称为闭表（closed list）。
  - 完备却不具有最优性。

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

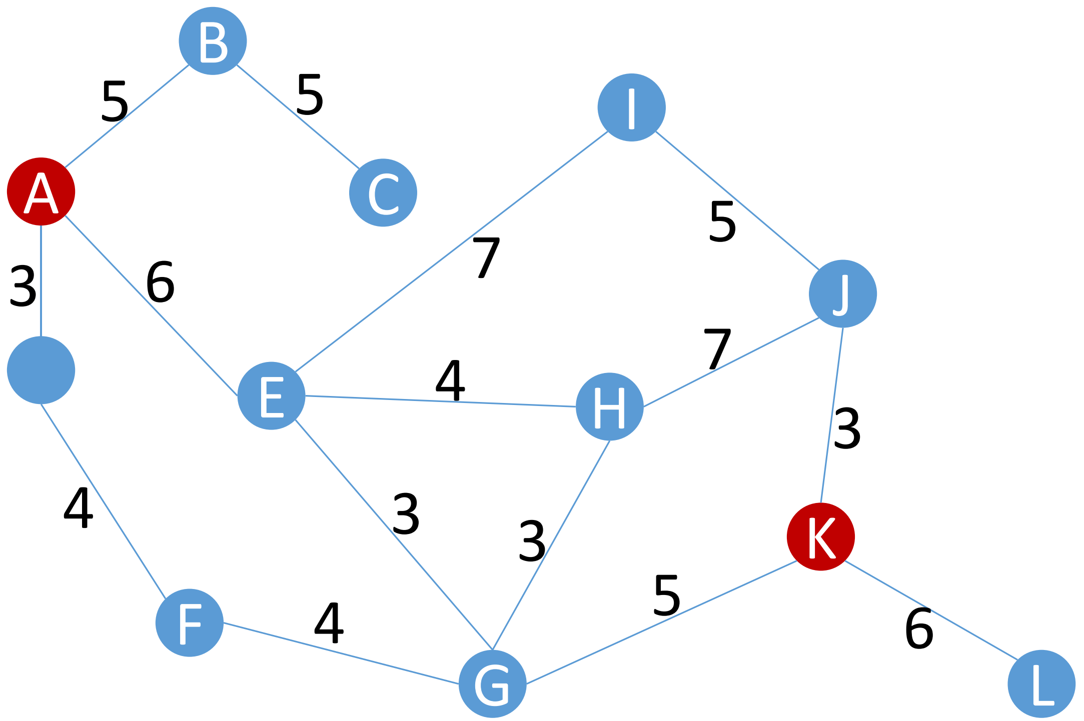
$\mathcal{F} \leftarrow \{$根节点$\}$
$\mathcal{C} \leftarrow \emptyset$
while $\mathcal{F} \neq \emptyset$ **do**
    $n \leftarrow$ `pick_from(`$\mathcal{F}$`)`
    $\mathcal{F} \leftarrow \mathcal{F} - \{n\}$
    if `goal_test(`n`)` **then**
        return $n.path$
    end
    if $n.state \notin \mathcal{C}$ **then**
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{n.state\}$
        $\mathcal{F} \leftarrow \mathcal{F} \cup$ `successor_nodes(`n`)`
    end
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?

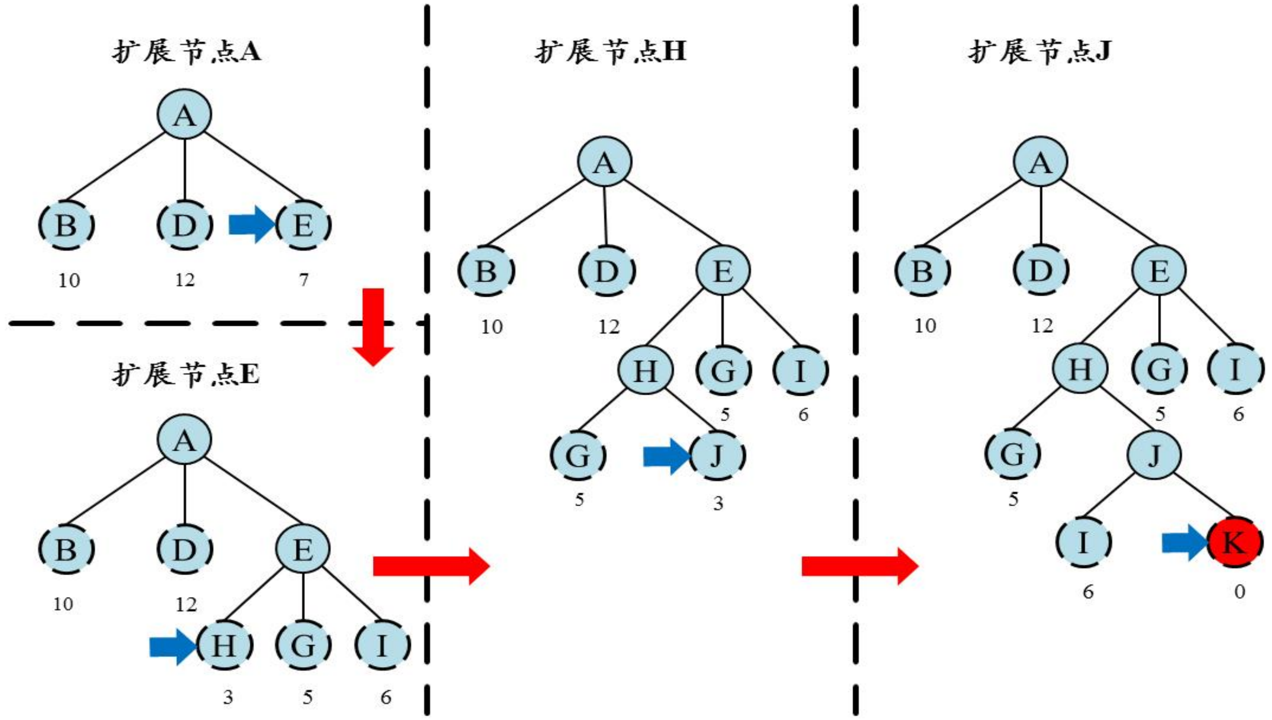
- **贪婪最佳优先搜索：评价函数$f(n)$=启发函数$h(n)$**
- **辅助信息(启发函数)**
  - 任意一个城市与终点城市K之间的直线距离

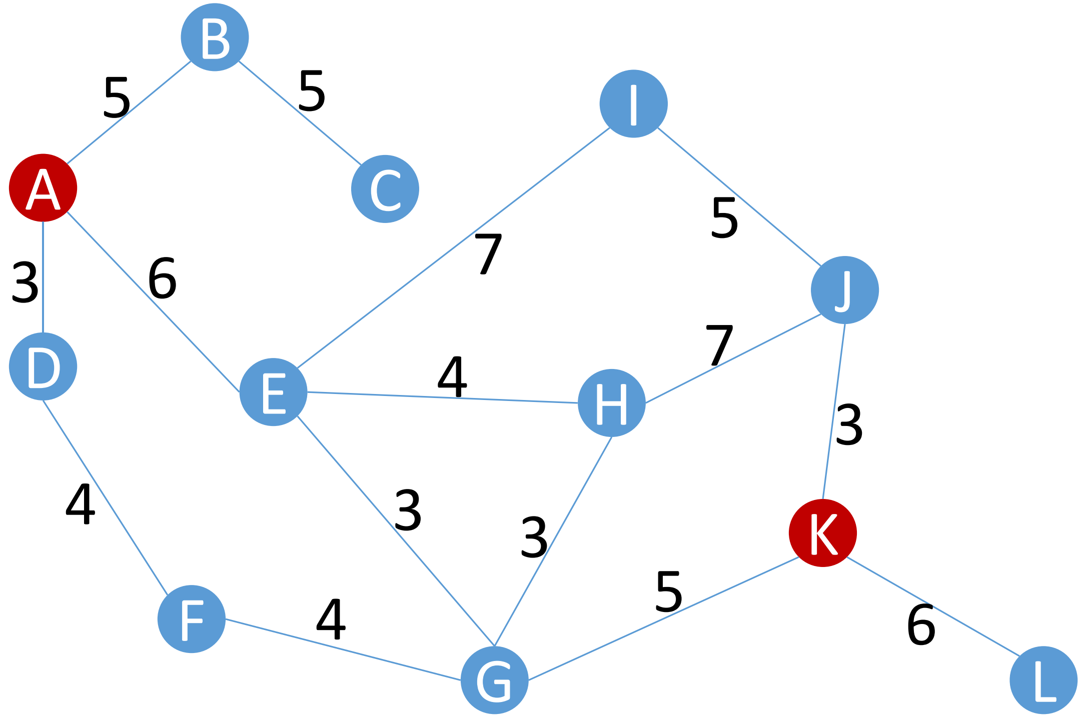
状态	A	B	C	D	E	F	G	H	I	J	K	L
$h(n)$	13	10	6	12	7	8	5	3	6	3	0	6

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：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



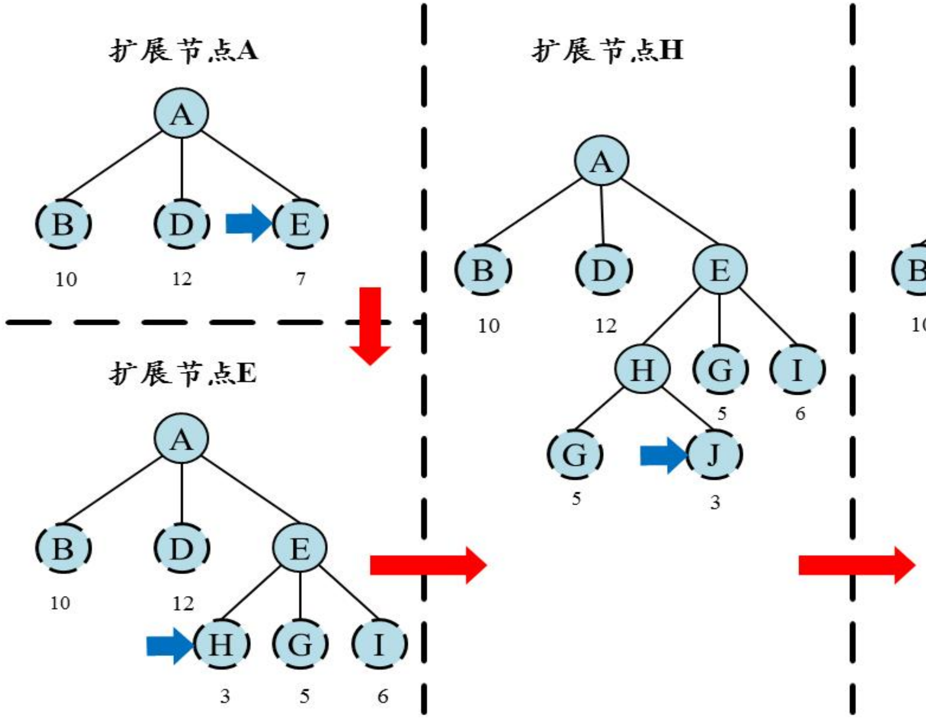


贪婪最佳优先搜索的过程

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。

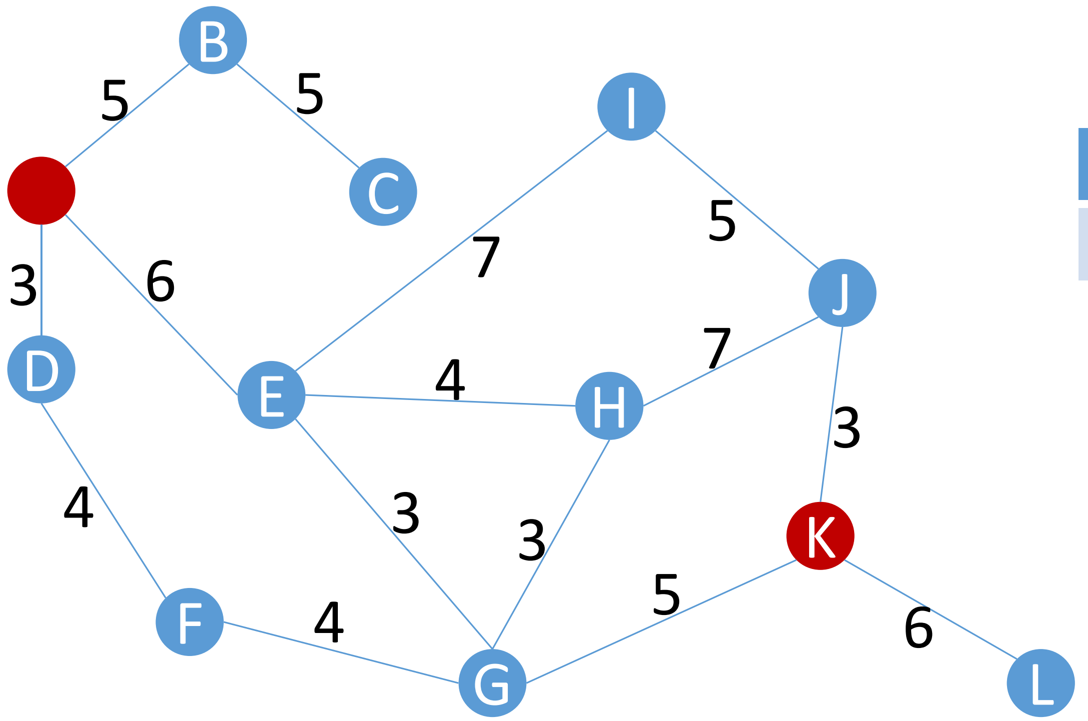
搜索算法：A*算法

- 评价函数：$\mathrm{f}(\boldsymbol{n})=\mathrm{g}(\boldsymbol{n})+\boldsymbol{h}(\boldsymbol{n})$
  - $\mathrm{g}(\boldsymbol{n})$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
  - $\mathrm{f}(\boldsymbol{n})$可视为经过结点$\boldsymbol{n}$、具有最小开销代价值的路径。

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

搜索算法：A*算法

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

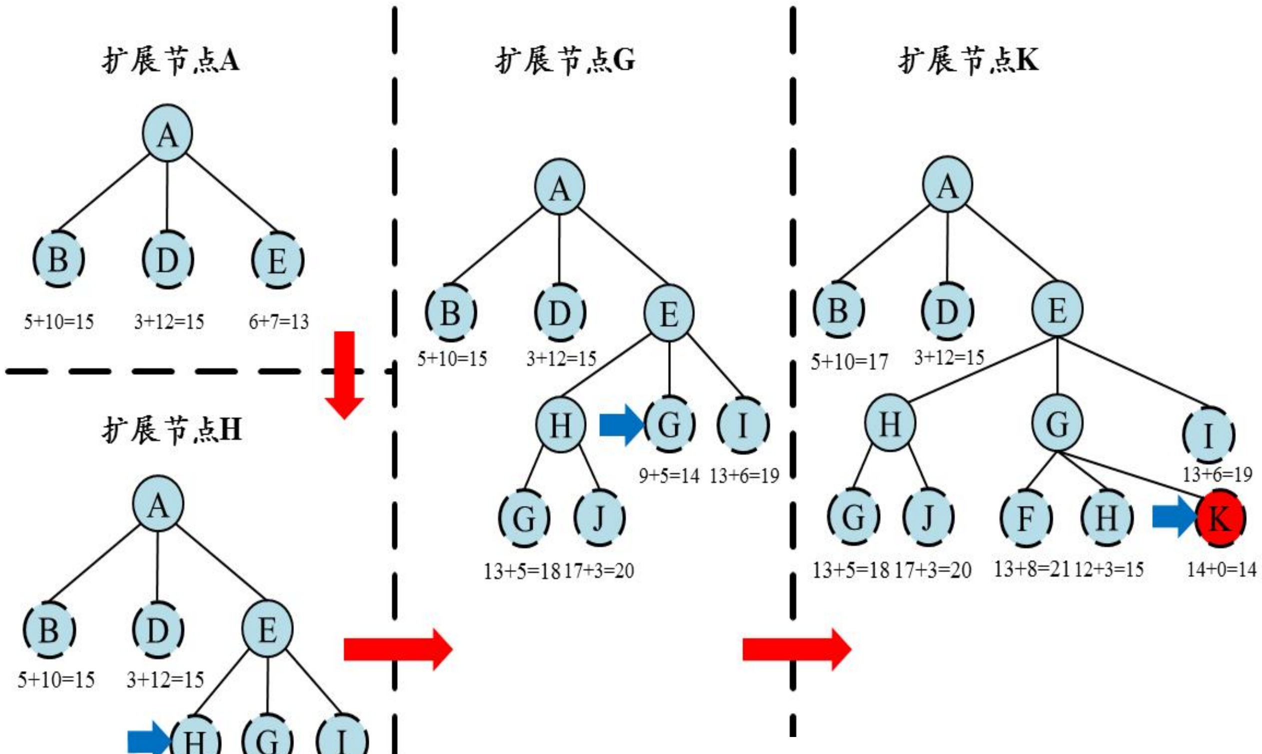


状态	A	B	C	D	E	F	G	H	I	J	K	L
$h(n)$	13	10	6	12	7	8	5	3	6	3	0	6

辅助信息：任意一个城市与终点城市K之间的直线距离

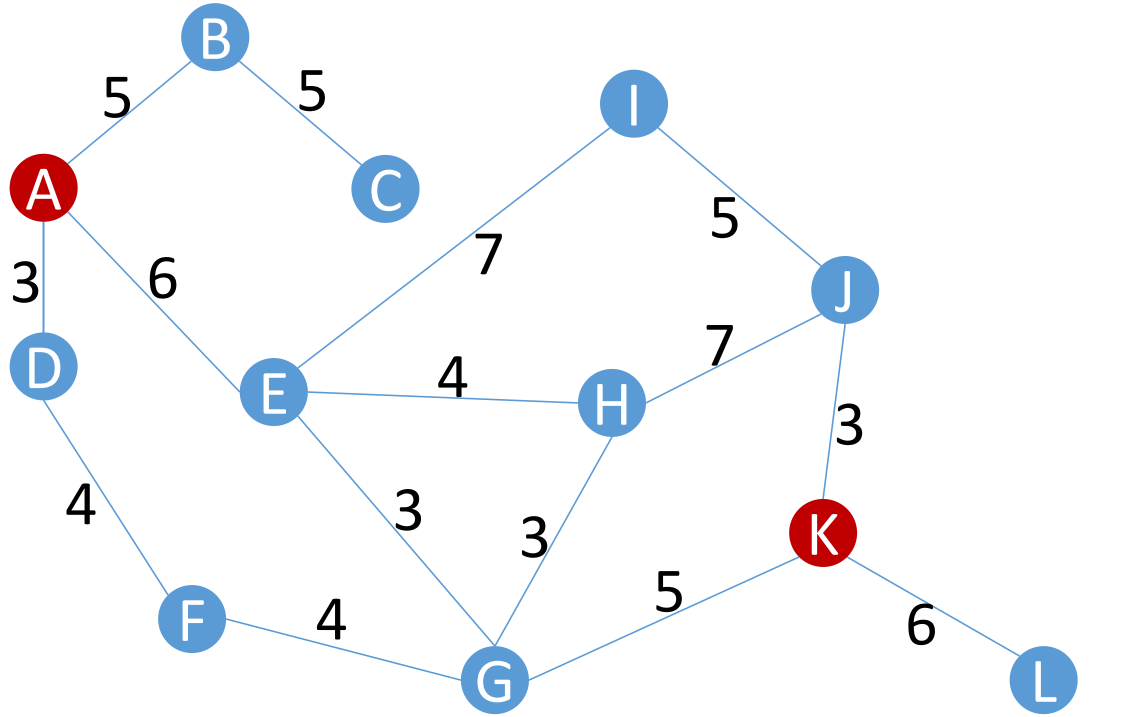
寻找从城市A到城市K之间最短路线？

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？

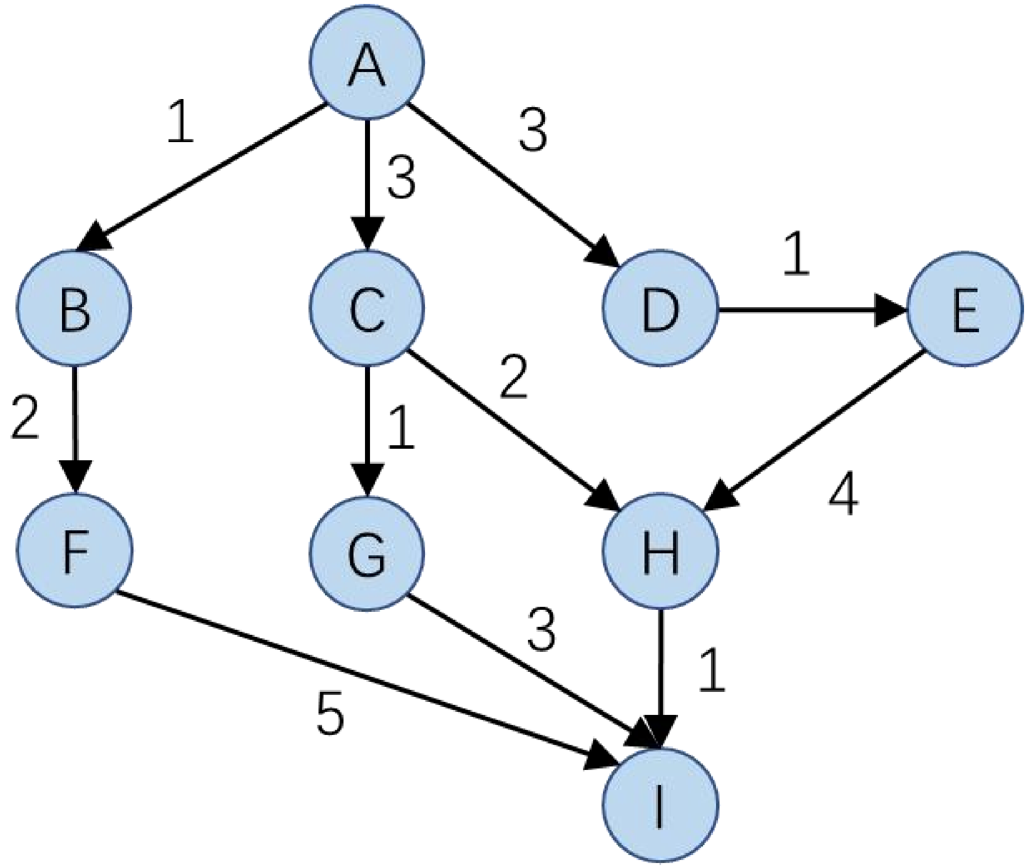
状态	A	B	C	D	E	F	G	H	I	J	K	L
$h(n)$	13	10	6	12	7	8	5	3	6	3	0	6

辅助信息：任意一个城市与终点城市K之间的直线距离

课上习题

如图1所示，假设每个节点代表一个状态，节点之间的箭头表示状态转移关系，箭头旁的数字表示状态转移的代价。若使用以下搜索算法寻找从状态A到状态I的路径，请画出算法终止（找到第一条路径）时的搜索树，并在搜索树中标出节点的扩展顺序，以及找到的路径。若有多个节点拥有相同的扩展优先度，则优先扩展对应路径字典序较小的节点。

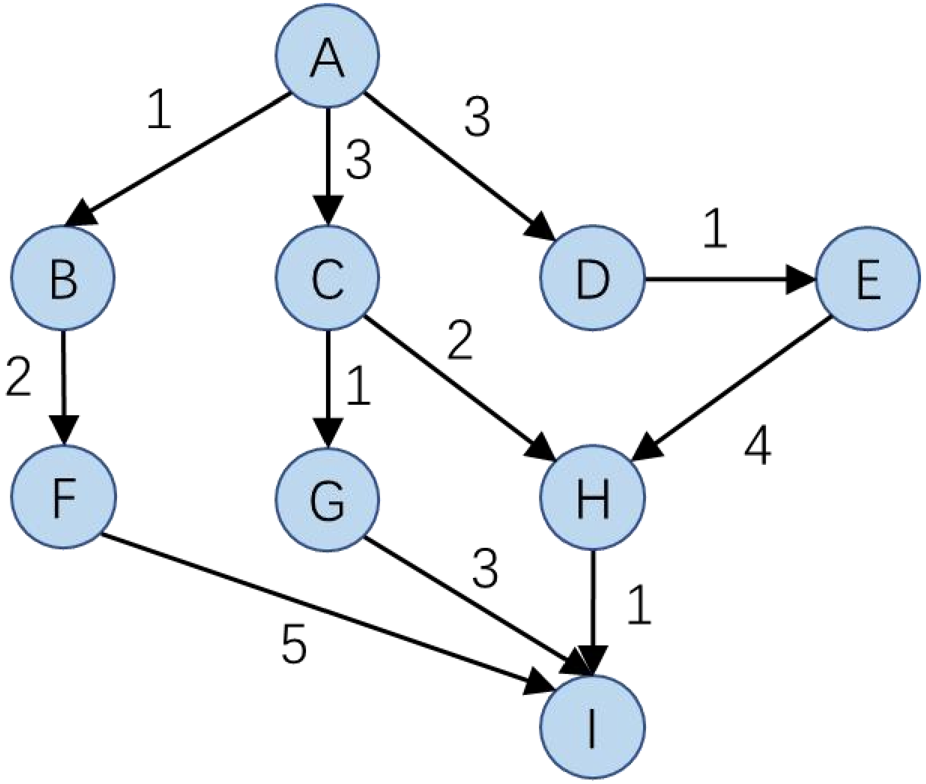
- 1)基于树搜索的广度优先搜索。
- 2)基于图搜索的深度优先搜索。

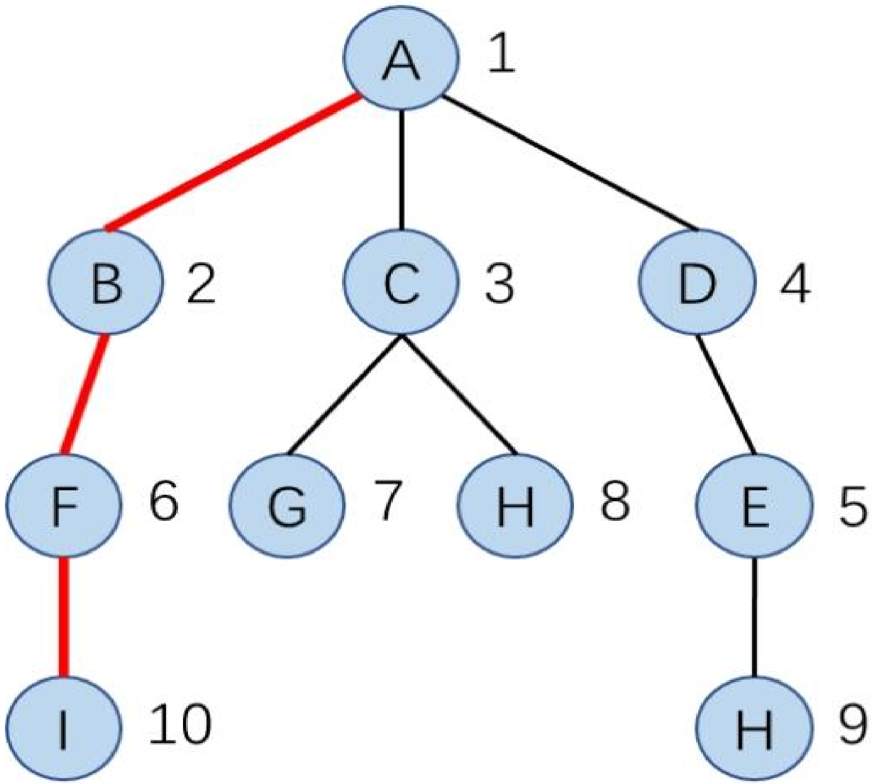


状态转移图

课上习题

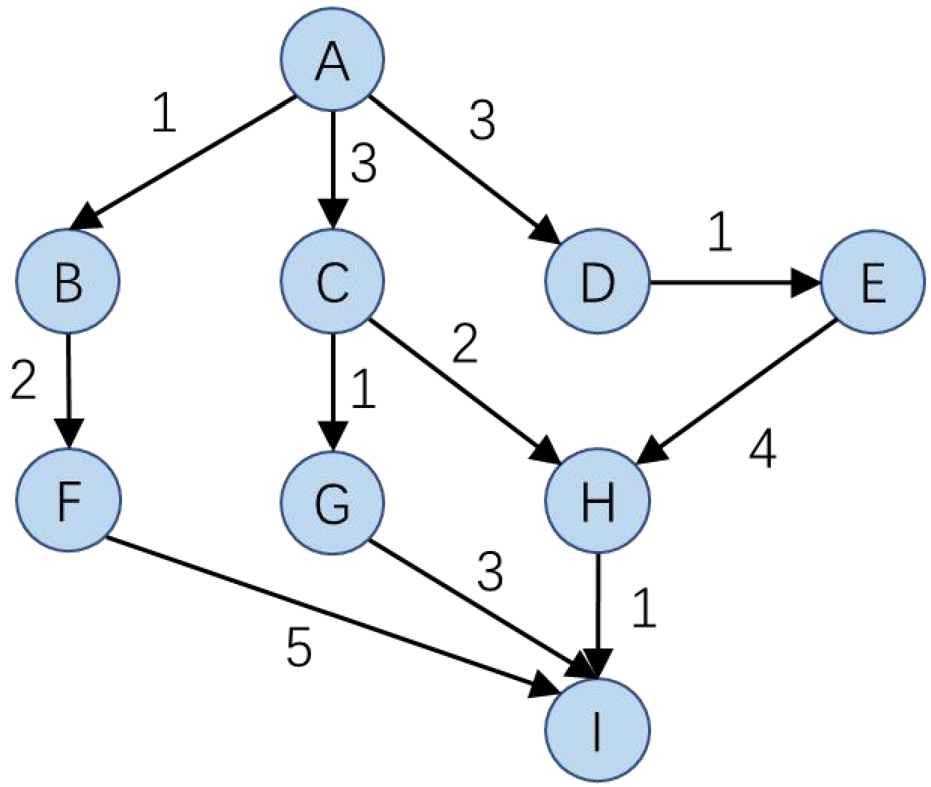
(1)

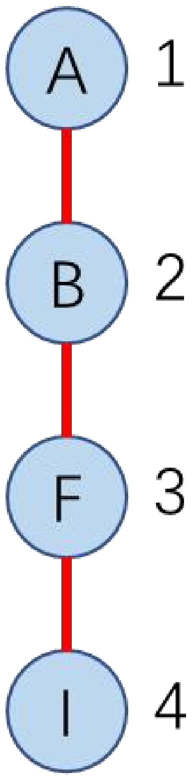




课上习题

(2)





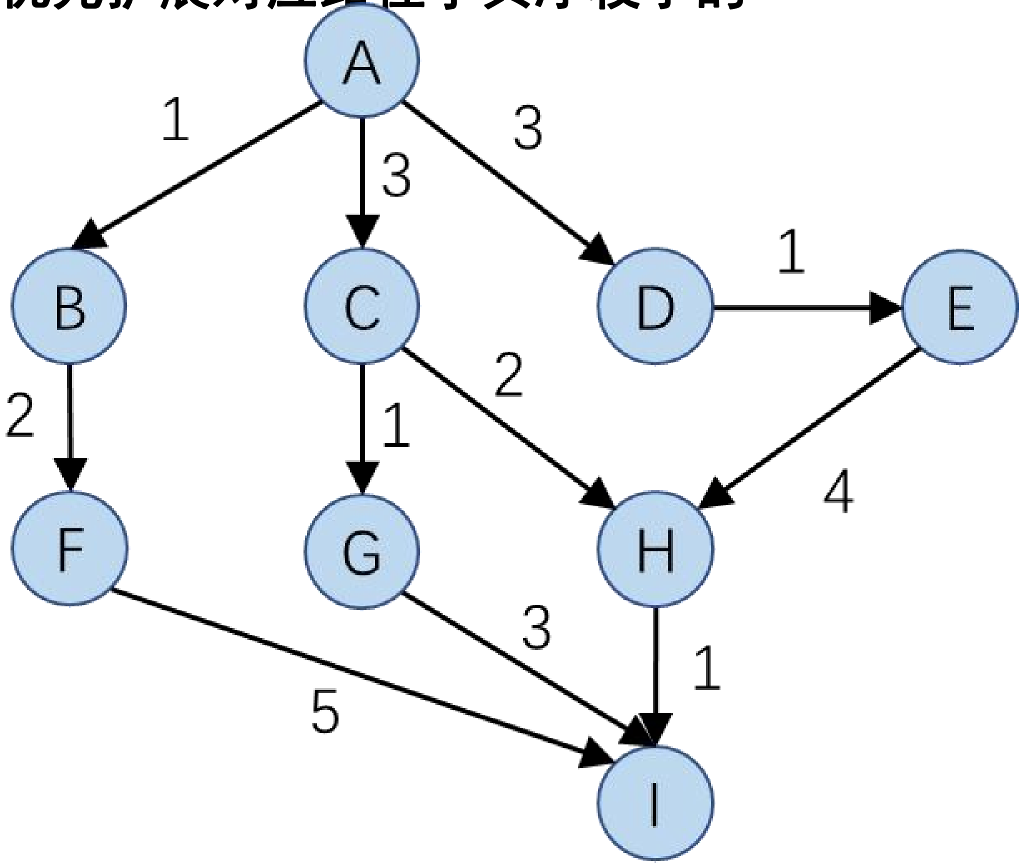
课上习题

考虑图1中的问题，给定每个状态的启发函数如表1所示。若仍以状态A为初始状态、状态I为终止状态，请分别使用以下算法求解从A到I的路径，并按照第6题中的方法画出搜索树。若有多个节点拥有相同的扩展优先度，则优先扩展对应路径字典序较小的节点。

- (1)基于树搜索的贪婪最佳优先搜索。
- (2)基于图搜索的A*算法。

状态	A	B	C	D	E	F	G	H	I
启发函数	5	4	3	2	5	5	2	1	0

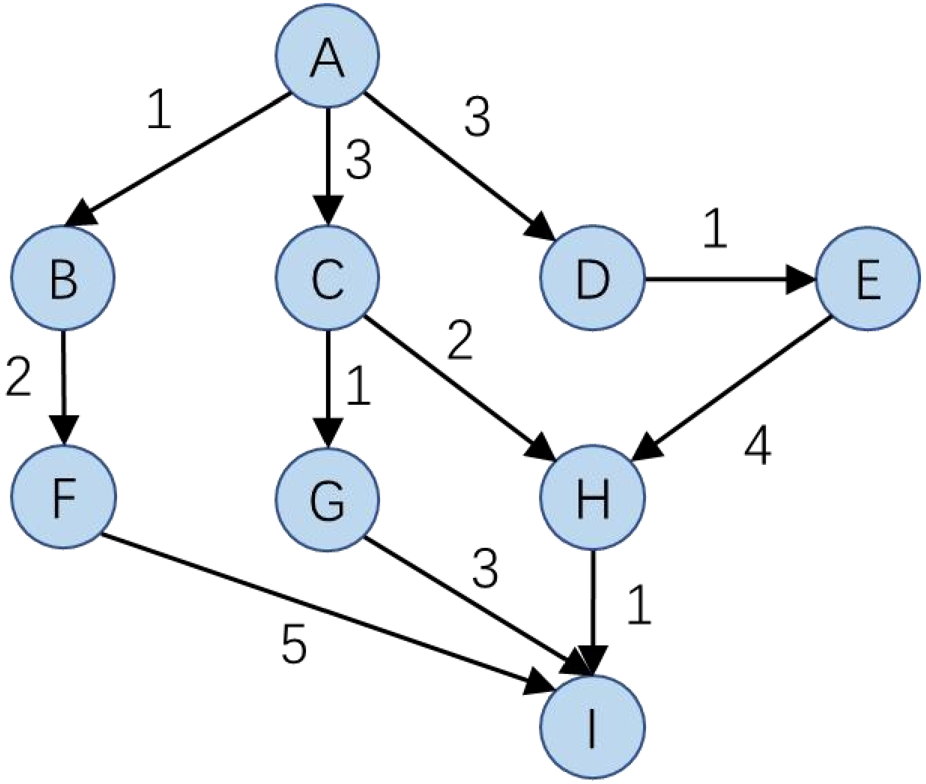
启发函数的取值

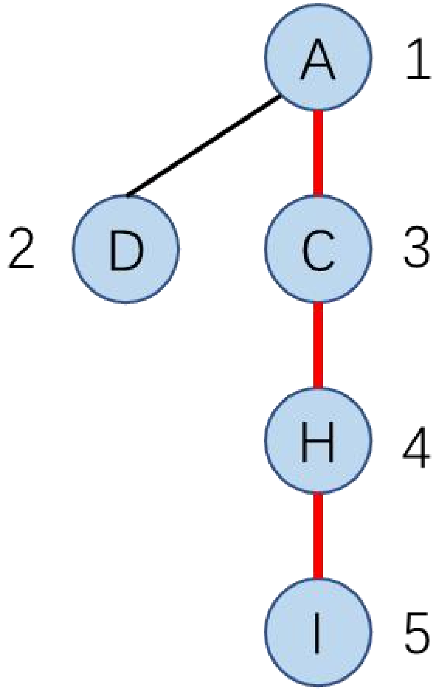


状态转移图

课上习题

(1)



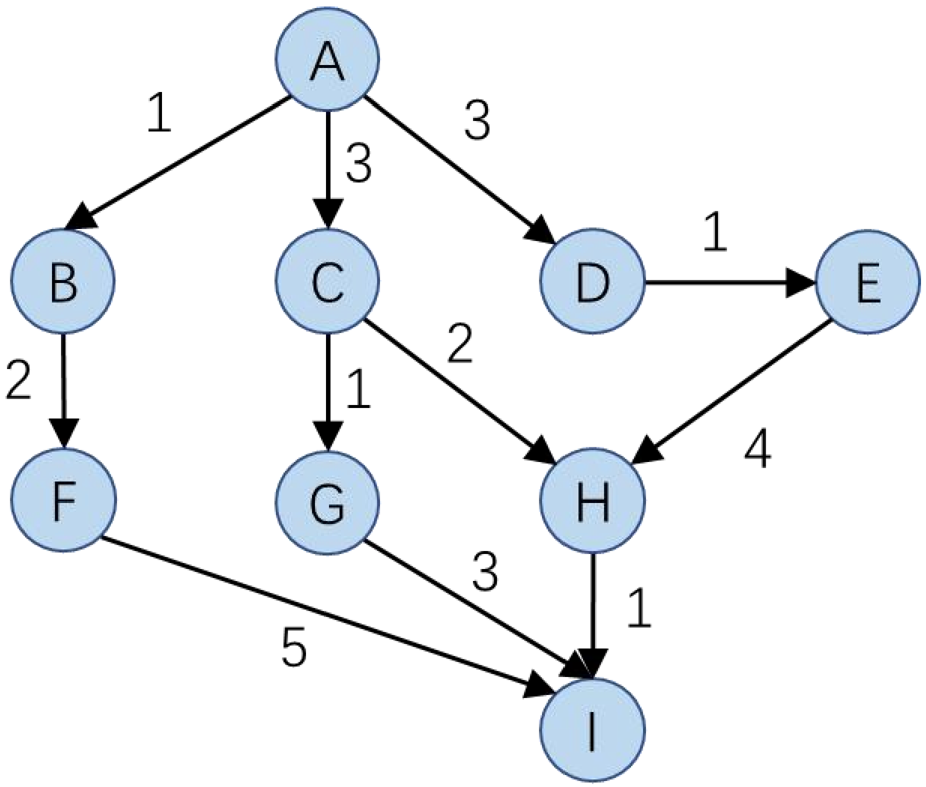


状态	A	B	C	D	E	F	G	H	I
启发函数	5	4	3	2	5	5	2	1	0

启发函数的取值

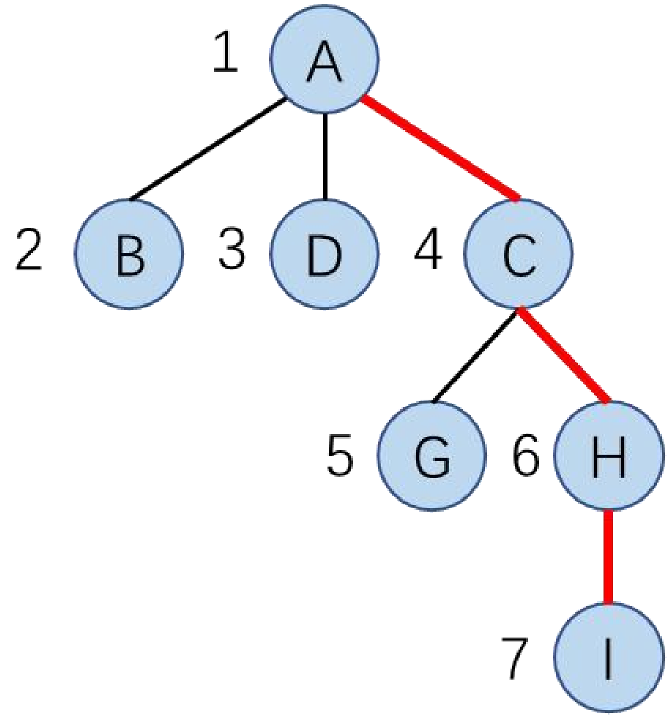
课上习题

(2)



状态	A	B	C	D	E	F	G	H	I
启发函数	5	4	3	2	5	5	2	1	0

启发函数的取值



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。

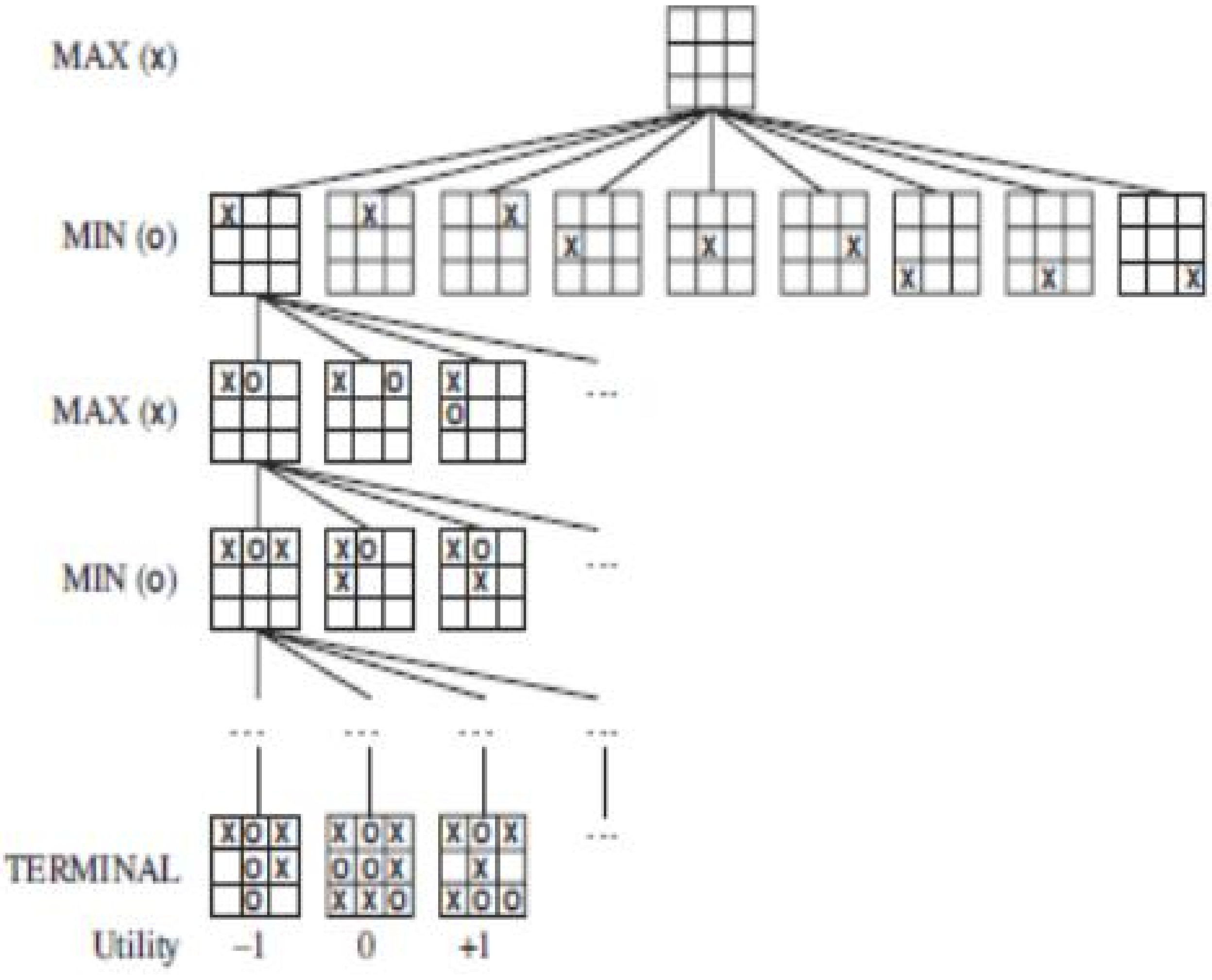
对抗搜索

- 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

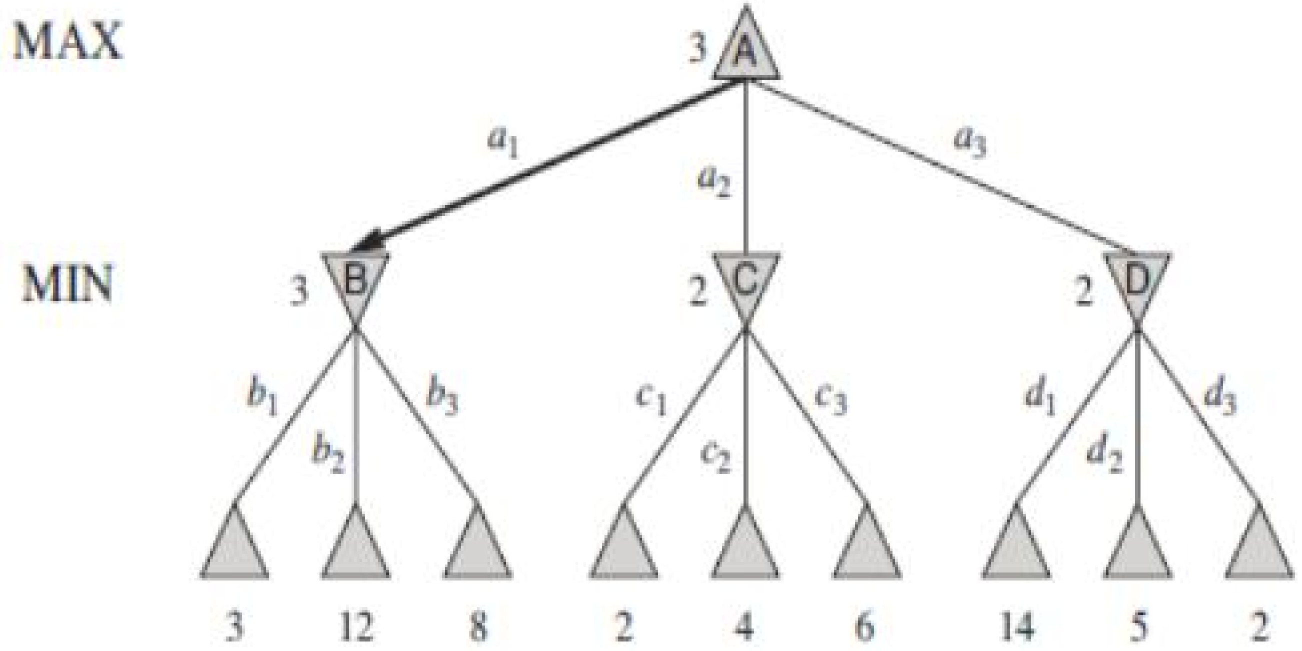
对抗搜索：minimax算法

- **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

$$
\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}
$$

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$_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT(*state*, *a*))

---

function MAX-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for each *a* **in** ACTIONS(*state*) **do**
    $v \leftarrow$ 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*)
  $v \leftarrow \infty$
  for each *a* **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(*s*, *a*)))
  return v

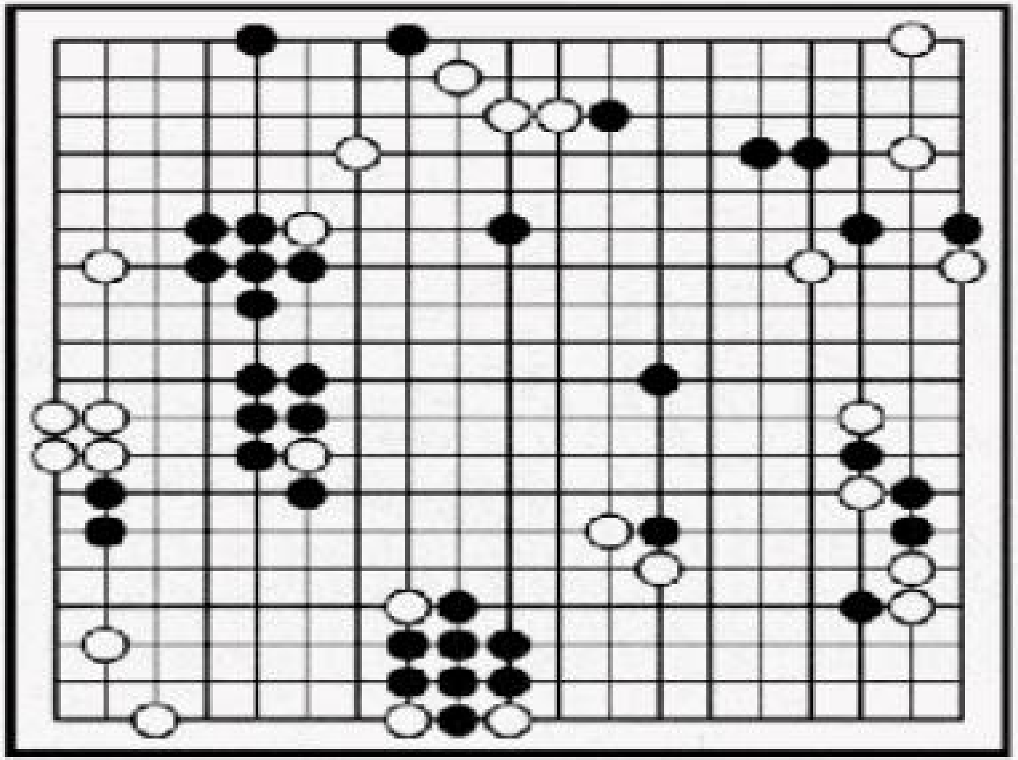
对抗搜索：minimax算法

- **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)

谢谢！