第 4 章 搜索策略

xyfJASON

1 概述

1.1 什么是搜索

概念：依靠经验，利用已有知识，根据问题的实际情况，不断寻找可利用知识，从而构造一条代价最小的推理路线，使问题得以解决的过程称为搜索。

适用情况：不良结构或非结构化问题；难以获得求解所需的全部信息；更没有现成的算法可供求解使用；结构良好，理论上有算法可以的，但问题或算法的复杂性较高。

分类：

- 按是否使用启发式信息：
  - 盲目搜索：按预定的控制策略进行搜索，在搜索过程中获得的中间信息并不改变控制策略。
  - 启发式搜索：在搜索中加入了与问题有关的启发性信息，用于指导搜索朝着最有希望的方向前进，加速问题的求解过程并找到最优解。
- 按问题的表示方式：
  - 状态空间搜索：用状态空间法来求解问题所进行的搜索
  - 与或树搜索：用问题归约法来求解问题时所进行的搜索

1.2 问题的形式

初始状态：s_0

后继函数：

- 操作函数：$\{a_1, a_2, a_3, \ldots\}$
- 路径耗散函数：$\text{PathCost}(s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_i} s_i)$
- 目标测试函数：$\text{GoalTest}(s)$

问题的解：$(s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n)$

1.3 状态空间

状态：表示每一步问题状况的**数据结构**

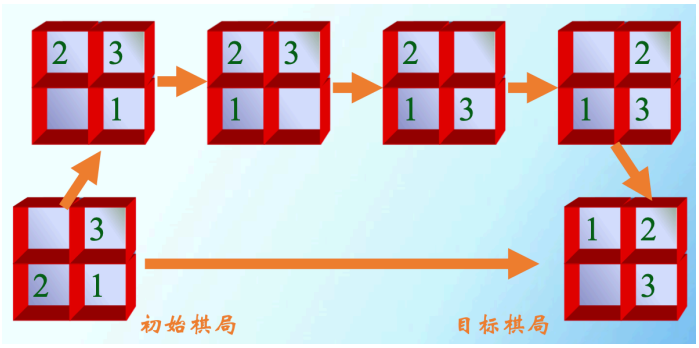
操作：把一种状态变到另一种状态

状态空间：描述问题全部状态和相互关系的空间，常用一个三元组表示：

$$(S, F, G)$$

其中 S 为问题所有初始状态集合，F 为操作集合，G 为目标状态的集合。

状态空间也可以用一个**赋值的有向图**表示，成为状态空间图。节点表示状态，有向边表示操作。



例子（详见 ppt）：三数码、二阶汉诺塔、修道士和野人、猴子摘香蕉。

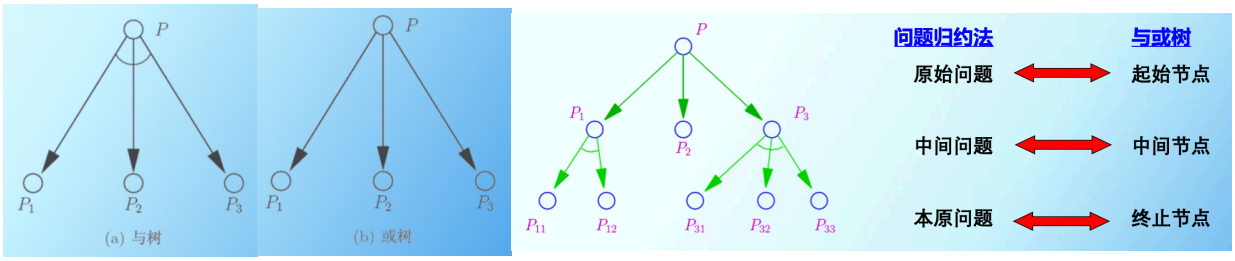
1.4 问题规约法

规约：

- **分解**：若问题 P 可以规约为一组子问题 $P_1, P_2, \ldots, P_n$，且仅当所有子问题都有解时原问题才有解，任何一个子问题无解都会导致原问题无解。“与”
- **等价变换**：若问题 P 可以规约为一组子问题 $P_1, P_2, \ldots, P_n$，只要有一个子问题有解时原问题就有解，仅当所有子问题都无解时原问题才无解。“或”

问题归约的实质：从目标（要解决的问题）出发逆向推理，建立子问题以及子问题的子问题，直至最后把初 始问题归约为一个平凡的本原问题集合。

与树/或树：



- 终止节点一定是本原问题对应的节点，即可以直接回答的子问题；而端节点，即叶子节点，不一定是终止节点，即它对应的问题虽然不能分解或变换了，但这个问题无法回答。
- 可解节点：任何终止节点；若或节点的子节点至少一个可解，则该或节点可解；若与节点所有子节点都可解，该与节点可解。
- 不可解节点：任何非终止节点的端节点；若或节点所有子节点都不可解，则该或节点不可解；若与节点的子节点至少一个不可解，则该与节点不可解。
- **解树**：一个由可解节点构成，并且由这些可解节点可以推出初始节点（对应原始问题）为可解节点的子树。

例：三阶汉诺塔问题分解为两个二阶汉诺塔问题和一个单片移动的本原问题；二阶汉诺塔问题分解为三个本原问题。

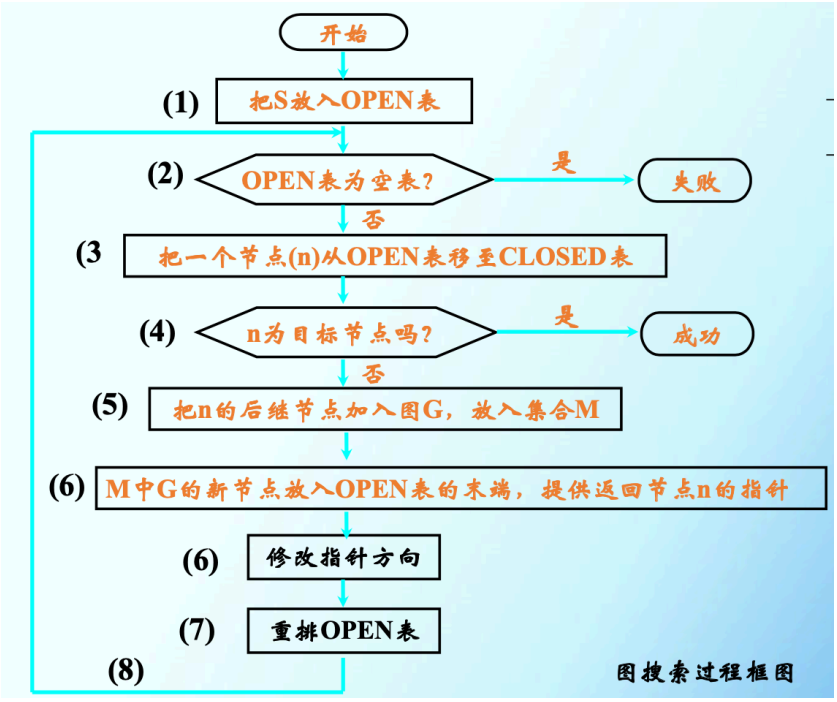
2 状态空间盲目搜索

Open 表：用于存放刚生成的节点，未扩展的节点，Open 表称为未扩展的节点表。

Closed 表：用于存放已经扩展或将要扩展的节点，Closed 表称为已扩展的节点表。

S_0：问题的初始状态；S_g：问题的目标状态。

2.1 一般图搜索过程



2.2 广度优先搜索

思想：从初始节点 S_0 开始，逐层地对节点进行扩展并考察它是否为目标节点，在第 n 层的节点没有全部扩展并考察之前，不对第 $n+1$ 层的节点进行扩展。

Open 表中的节点总是按进入的先后顺序排列，先进入的节点排在前面，后进入的排在后面。——队列

一般算法：

1. 把初始节点 S_0 放入 Open 表，建立一个 Closed 表，置为空；
2. 检查 Open 表是否为空表，若为空，则问题无解，失败退出；
3. 把 Open 表的第 一个节点取出放入 Closed 表，并记该节点为 n；
4. 考察节点 n 是否为目标节点，若是则得到问题的解成功退出；
5. 若节点 n 不可扩展，则转第 2 步；
6. 扩展节点 n， 将其子节点放入 Open 表的**尾部**，并为每个子节点设置指向父节点的指针, 转向第 2 步。

性质：

- 当问题有解时，一定能找到解，且一定是最优解（完备的）
- 搜索效率较低
- 方法与问题无关，具有通用性

2.3 深度优先搜索

思想：从初始节点 S_0 开始扩展，若没有得到目标节点，则选择最新产生的子节点进行扩展，若还是不能到达目标节点，则再对刚才最新产生的子节点进行扩展，一直如此向下搜索。当到达某个子节点，且该子节点既不是目标节点又不能继续扩展时，才选择其兄弟节点进行考察。

Open 表是一种栈结构，最先进进入的节点排在最后面，最后进入的节点排最前面。

一般算法：

1. 把初始节点 S_0 放入 Open 表，建立一个 Closed 表，置为空;
2. 检查 Open 表是否为空表，若为空，则问题无解，失败退出;
3. 把 Open 表的第一个节点取出放入 Closed 表，并记该节点为 n;
4. 考察节点 n 是否为目标节点，若是则得到问题的解成功退出;
5. 若节点 n 不可扩展，则转第 2 步;
6. 扩展节点 n，将其子节点放入 Open 表的**首部**，并为每个子节点设置指向父节点的指针，转向第 2 步。

性质：

- 不能保证找到最优解
- 最坏情况下等同于穷举
- 方法与问题无关，具有通用性

改进：有界深度优先搜索

2.4 代价一致搜索

引入：BFS 和 DFS 都假设状态空间中各边的代价都相同（一个单位），但现实中，各边的代价可能不同。称边上有代价的搜索树为**代价树**。

设 $g(n)$ 表示初始节点 S_0 到节点 n 的代价，$c(n_1, n_2)$ 表示从父节点 n_1 到 n_2 的代价，则：

$$g(n_2) = g(n_1) + c(n_1, n_2)$$

最小代价路径与最短路径可能不同。

思想：从初始节点 S_0 开始扩展，若没有得到目标节点，则优先扩展最少代价 $g(i)$ 的节点，一直如此向下搜索。

一般算法：

1. 把初始节点 S_0 放入 Open 表，设 $g(S_0) = 0$，建立一个 Closed 表，置为空;
2. 检查 Open 表是否为空表，若为空，则问题无解，失败退出;
3. 把 Open 表的第一个节点取出放入 Closed 表，并记该节点为 n;

4. 考察节点 n 是否为目标节点，若是则得到问题的解成功退出；
5. 若节点 n 不可扩展，则转第 2 步；
6. 扩展节点 n，生成子节点 $n_i(i=1,2,\ldots)$，将其子节点放入 Open 表，并为每个子节点设置指向父节点的指针，计算各个节点的代价 $g(n_i)$ ，将 Open 表内的节点按 $g(n_i)$ 从小到大排序, 转向第 2 步。

3 状态空间启发式搜索

启发性信息：启发性信息是指那种与具体问题求解过程有关的，并可指导搜索过程朝着最有希望方向前进的控制信息。启发信息的启发能力越强，扩展的无用结点越少，搜索算法越高效。启发信息包括以下 3 种：

1. 有效地帮助确定扩展节点的信息；
2. 有效地帮助决定哪些后继节点应被生成的信息；
3. 能决定在扩展一个节点时哪些节点应从搜索树上删除的信息。

估价函数：

$$f(n)=g(n)+h(n)$$

$g(n)$ 是从初始节点到节点 n 的**实际代价**，$h(n)$ 是从节点 n 到目标节点的最优路径的**估计代价**，需要根据问题自身特性来确定，体现的是问题自身的启发性信息，因此也称为**启发函数**。

3.1 贪心算法

思想：按照 $h(n)$ 排序，选择最小的 $h(n)$ 扩展。

> 对比：代价一致算法 v.s. 贪心算法
>
> 代价一致算法按照 **backward cost** $g(n)$ 排序，而贪心算法按照 **forward cost** $h(n)$ 排序。

3.2 A 算法

思想：在状态空间搜索中，如果每一步都利用估价函数 $f(n)=g(n)+h(n)$ 对 Open 表中的节点进行排序，则称 A 算法。它是一种为启发式搜索算法。

类型：

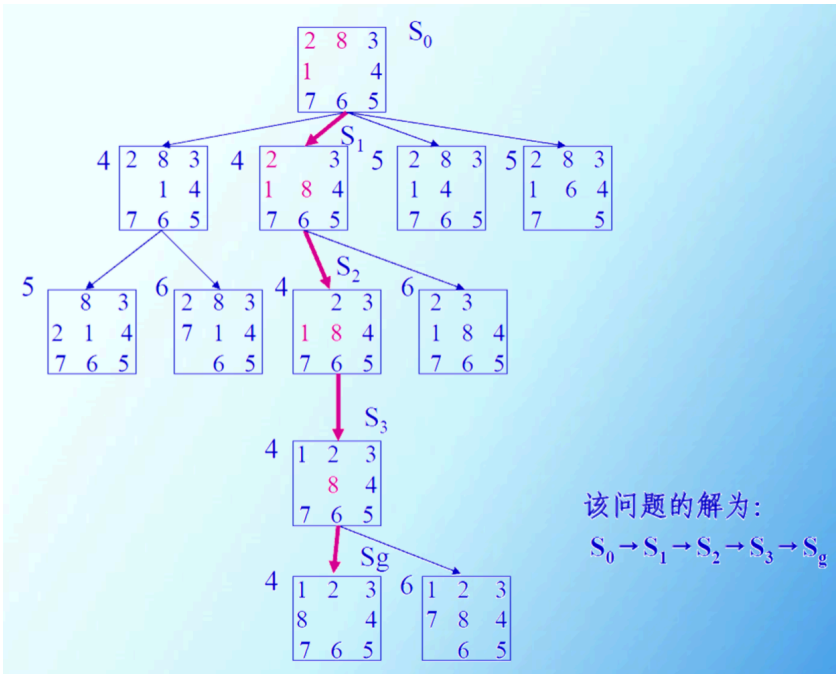
- 全局择优：从 Open 表的所有节点中选择一个估价函数值最小的进行扩展。
- 局部择优：仅从刚生成的子节点中选择一个估价函数值最小的进行扩展。

算法描述：

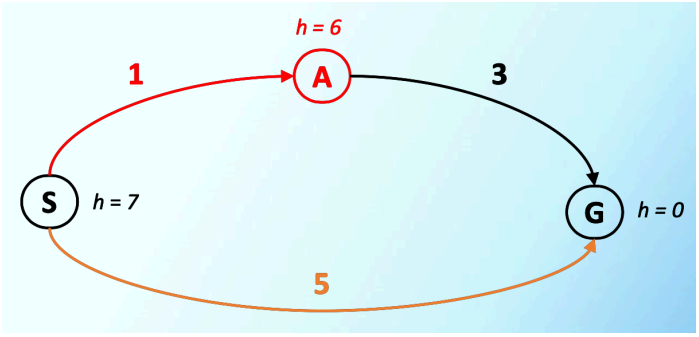
1. 把初始节点 S_0 放入 Open 表中，$f(S_0)=g(S_0)+h(S_0)$；
2. 如果 Open 表为空，则问题无解，失败退出；
3. 把 Open 表的第一个节点取出放入 Closed 表，并记该节点为 n；
4. 考察节点 n 是否为目标节点。若是，则找到了问题的解，成功退出；

5. 若节点 n 不可扩展，则转第 2 步；
6. 扩展节点 n，生成其子节点 $n_i(i=1,2,\ldots)$，计算每一个子节点的估价值 $f(n_i)(i=1,2,\ldots)$，并为每一个子节点设置指向父节点的指针，然后将这些子节点放入 Open 表中；
7. 根据各节点的估价函数值，对 Open 表中的全部节点按从小到大的顺序重新进行排序；
8. 转第 2 步。

例子：八数码问题：设 $g(n):=d(n)$ 为在搜索树中的深度，$h(n):=W(n)$ 为“不在位”的数码个数。



问题：以下图为例，假若 $h(S)=7, h(A)=6, h(G)=0$，我们会找到 $S\to G$ 而不是更优的 $S\to A\to G$.



出现这个问题的原因是：启发函数的选择不当，导致最优的估计代价甚至大于最差的真实代价。

3.3 A* 算法

为了解决 A 算法的问题，A* 算法给估价函数加入了限制。假设 $f^*(n)$ **是从初始节点 S_0 出发，约束经过节点 n 到达目标节点 S_g 的最小代价**。显然它由两部分组成：从初始节点 S_0 到节点 n 的最小代价 $g^*(n)$ 和从节点 n 到目标节点 S_g 的最小代价 $h^*(n)$：

$$f^*(n)=g^*(n)+h^*(n)$$

记估价函数 $f(n)$ 是对 $f^*(n)$ 的估计值，$g(n)$ 是对 $g^*(n)$ 的估计，$h(n)$ 是对 $h^*(n)$ 的估计。其中，尽管 $g(n)$ 容易计算，但它不一定是 S_0 到 n 的最小代价，因此：$g(n)\geq g^*(n)$.

A* 算法在 A 算法的基础上增加了限制：

1. $g(n)$ 是对 $g^*(n)$ 的估计，且 $g(n) > 0$；
2. $h(n)$ 是 $h^*(n)$ 的下界，即 $h(n) \leq h^*(n)$.

可纳性：对任一状态空间图，当从初始节点到目标节点有路经存在时，如果搜索算法总能在有限步骤内找到一条从初始节点到目标节点的最佳路径，并在此路径上结束，则称该搜索算法是可采纳的。

可以证明，A* 算法是可纳的。证明详见教材和 ppt。

一些重要的性质：

- 若 n 在最优路径上，则

$$g(n) = g^*(n)$$
$$f(n) = g(n) + h(n) = g^*(n) + h(n) \leq g^*(n) + h^*(n) = f^*(n) = f^*(S_0)$$

- 任一时刻，Open 表中必然存在节点 n'，它是最佳路径上的节点（尽管下一步不一定就选择它扩展，换句话说，它不一定是 Open 表中 f 值最小的点）
- 若 n 不在最优路径上，但它被选来扩展了，则一定有

$$f(n) \leq f^*(S_0)$$

  由前两条性质易得。

最优性：在满足 $h(n) \leq h^*(n)$ 的条件下，$h(n)$ 值越大，携带的启发性信息越多，A^* 就越高效。

单调限制：在 A* 算法中，每当扩展一个节点 n 时，都需要检查其子节点是否已在 Open 表或 Closed 表中。对已在 Open 表中的子节点，需要决定是否调整指向其父节点的指针；对已在 Closed 表中的子节点，除需要决定是否调整其指向父节点的指针外，还需要决定是否调整其子节点的后继节点的父指针。如果能够保证，每当扩展一个节点时就已经找到了通往这个节点的最佳路径，就没有必要再去作上述检查。为满足这一要求，我们需要对启发函数 $h(n)$ 增加单调性限制。

如果启发函数满足以下两个条件：

1. $h(S_g) = 0$；
2. 对任意节点 n_i 及其任一子节点 n_j，都有 $0 \leq h(n_i) - h(n_j) \leq c(n_i, n_j)$。其中 $c(n_i, n_j)$ 是 n_i 到其子节点 n_j 的边代价，则称 $h(n)$ 满足单调限制。

4 与/或树搜索

4.1 与/或树的一般搜索

与/或树的搜索过程实际上是一个不断寻找解树的过程。其一般搜索过程如下：

1. 把原始问题作为初始节点 S_0，并把它作为当前节点；
2. 应用分解或等价变换操作对当前节点进行扩展；
3. 为每个子节点设置指向父节点的指针；

4. 选择合适的子节点作为当前节点，反复执行第 2 步和第 3 步，在此期间需要多次调用可解标记过程或不可解标记过程，直到初始节点被标记为可解节点或不可解节点为止。

上述搜索过程将形成一颗与/或树，这种由搜索过程所形成的与/或树称为搜索树。

4.2 广度优先搜索

略

4.3 深度优先搜索

可以添加一个深度限制，略。

4.4 启发式搜索

解树的代价：

1. 若 n 为终止节点，$h(n)=0$
2. 若 n 为或节点，且子节点为 $n_1,\dots,n_k$，则 $h(n)=\min_{1\le i\le k}\{c(n,n_i)+h(n_i)\}$
3. 若 n 为与节点，且子节点为 $n_1,\dots,n_k$，则
   - 和代价法：$h(n)=\sum_{i=1}^{k}[c(n,n_i)+h(n_i)]$
   - 最大代价法：$h(n)=\max_{1\le i\le k}\{c(n,n_i)+h(n_i)\}$
4. 若 n 是端节点，但非终止节点，则 $h(n)=+\infty$
5. 根节点代价即为解树的代价

希望树：为找到最佳解树，搜索过程的任何时刻都应该选择那些最有希望成为最佳解树一部分的节点进行扩展。由于这些节点及其父节点所构成的与或树最有可能成为最佳解树的一部分，因此称之为希望树。与或树的启发式搜索过程就是不断地选择、修正希望树的过程，在该过程中，希望树是不断变化的。

希望树定义：

1. 初始节点 S_0 在希望树 T 中；
2. 如果节点 x 在希望树 T 中，则：
   (a) 如果 x 是具有子节点 $y_1,y_2,\dots,y_k$ 的“或”节点，则具有 $\min\{c(x,y_i)+h(y_i)\}$ $(i=1,2,\dots,k)$ 值的那个子节点 y_i 也应在 T 中；
   (b) 如果 x 是“与”节点，则 x 的全部子节点都在希望树 T 中。

与或树启发式**搜索过程**：

1. 把初始节点 S_0 放入 Open 表中，计算 $h(S_0)$；
2. 计算希望树 T；
3. 依次在 Open 表中取出 T 的端节点放入 Closed 表，并记该节点为 n；

4. 如果节点 n 为终止节点，则做下列工作：
   (a) 标记节点 n 为可解节点；
   (b) 在 T 上应用可解标记过程，对 n 的先辈节点中的所有可解节点进行标记；
   (c) 如果初始解节点 S_0 能够被标记为可解节点，则 T 就是最优解树，成功退出；
   (d) 否则，从 Open 表中删去具有可解先辈的所有节点。
   (e) 转第 2 步。
5. 如果节点 n 不是终止节点，但可扩展，则做下列工作：
   (a) 扩展节点 n，生成 n 的所有子节点；
   (b) 把这些子节点都放入 Open 表中，并为每一个子节点设置指向父节点 n 的指针；
   (c) 计算这些子节点及其先辈节点的 h 值；
   (d) 转第 2 步。
6. 如果节点 n 不是终止节点，且不可扩展，则做下列工作:
   (a) 标记节点 n 为不可解节点；
   (b) 在 T 上应用不可解标记过程，对 n 的先辈节点中的所有不可解节点进行标记；
   (c) 如果初始节点 S_0 能够被标记为不可解，则问题无解，失败退出；
   (d) 否则，从 Open 表中删去具有不可解先辈的所有节点；
   (e) 转第 2 步。

例子：见 ppt.

5 博弈树的启发式搜索

5.1 概述

博弈的**概念**：博弈是一类具有智能行为的竞争活动，如下棋、战争等。

博弈的**类型**：

- 双人完备信息博弈：两位选手（例如 MAX 和 MIN）对垒，轮流走步，每一方不仅知道对方已经走过的棋步，而且还能估计出对方未来的走步。
- 机遇性博弈：存在不可预测性的博弈，例如掷币等。

博弈树

若把双人完备信息博弈过程用图表示出来，就得到一棵与/或树，这种与/或树被称为博弈树。在博弈树中，那些下一步该 MAX 走步的节点称为 MAX 节点， 下一步该 MIN 走步的节点称为 MIN 节点。

博弈树的**特点**

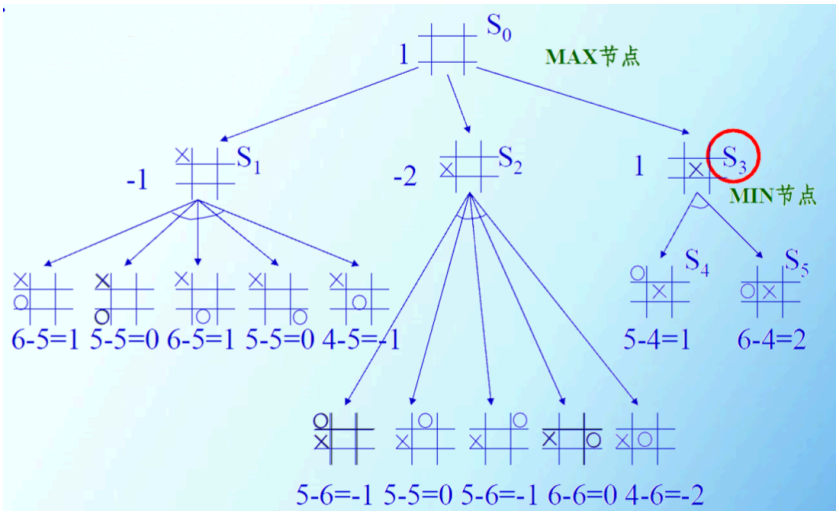
- 博弈的初始状态是初始节点；
- 博弈树中的“或”节点和“与”节点是逐层交替出现的；
- 整个博弈过程始终站在某一方的立场上，例如 MAX 方。所有能使自己一方获胜的终局都是本原问题，相应的节点是可解节点；所有使对方获胜的终局都是不可解节点。

5.2 极大极小过程

极大极小过程：

- 叶节点的估值：那些对 MAX 有利的节点，其估价函数取正值；那些对 MIN 有利的节点，其估价函数取负值；那些使双方均等的节点，其估价函数取接近于 0 的值。
- 非叶节点的估值：从叶节点开始向上倒推——对于 MAX 节点，由于 MAX 方总是选择估值最大的走步，因此，**MAX 节点的倒推值应该取其后继节点估值的最大值**。 对于 MIN 节点，由于 MIN 方总是选择使估值最小的走步，因此 **MIN 节点的倒推值应取其后继节点估值的最小值**。这样一步一步的计算倒推值，直至求出初始节点的倒推值为止。这一过程称为极大极小过程。

例子：一字棋游戏，详见 ppt.

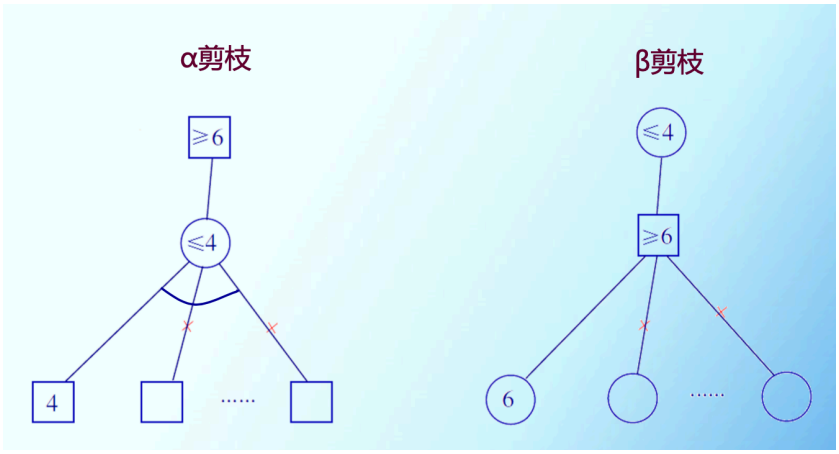


5.3 $\alpha - \beta$ 剪枝

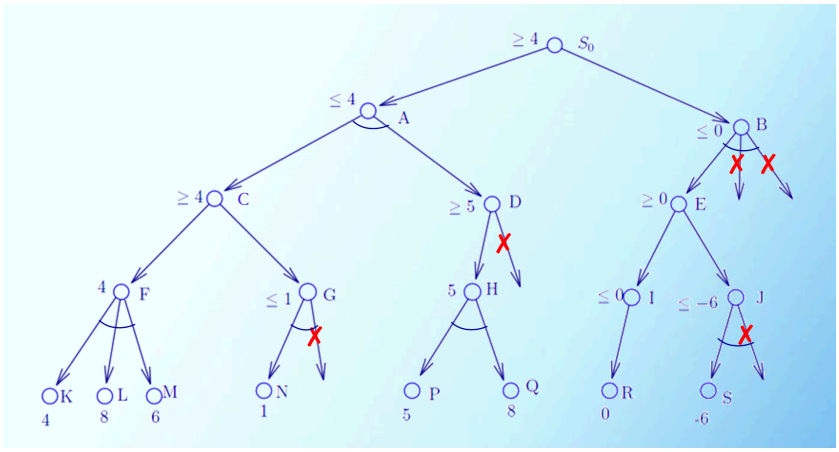
概念：在极大极小搜索过程中，剪去一些没用的分枝，减少了搜索空间，使得算法在同样时间内可以搜索更深层，这种技术被称为 $\alpha - \beta$ 剪枝。

剪枝方法与规则：

1. MAX 节点（或节点）的 α 值为当前子节点的最大倒推值；
2. MIN 节点（与节点）的 β 值为当前子节点的最小倒推值；
3. $\alpha - \beta$ 剪枝的规则如下：
   (a) β 剪枝：任何 MAX 节点 n 的 α 值大于或等于它先辈节点的 β 值，则 n 以下的分枝可停止搜索，并令节点 n 的倒推值为 α。这种剪枝称为 β 剪枝。
   (b) α 剪枝：任何 MIN 节点 n 的 β 值小于或等于它先辈节点的 α 值，则 n 以下的分枝可停止搜索， 并令节点 n 的倒推值为 β。这种剪枝称为 α 剪枝。


α剪枝
β剪枝
≥6
≤4
4
.....
≤4
≥6
6
.....

例子:


≥ 4 S_0
≤ 4 A
≤ 0 B
≥ 4 C
≥ 5 D
≥ 0 E
4 F
≤ 1 G
5 H
≤ 0 I
≤ −6 J
K L M N P Q R S
4 8 6 1 5 8 0 -6