Artificial Intelligence
人工智能实验

搜索算法

中山大学计算机学院
2024年春季

目录

1.1 基本概念

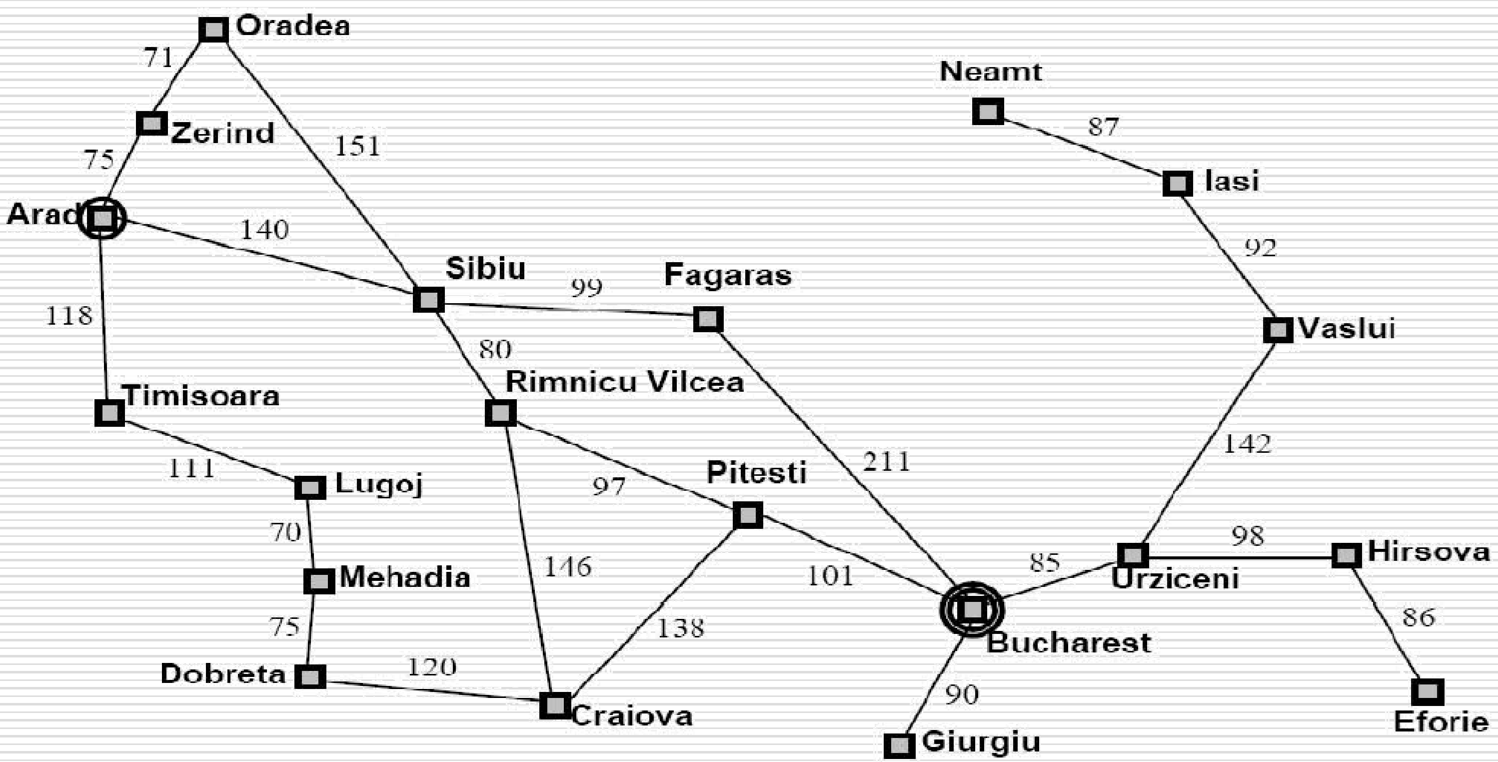
- 问题的定义
  - 用5个组件形式化定义一个search problem：
    - Initial state 初始状态
    - Actions 动作
    - Transition model 转移模型
    - Goal test 目标测试
    - Path cost 路径代价
  - 问题的解（solution）的定义
    - A solution to a problem is an action sequence that leads from the initial state to a goal state.

1.1 基本概念

□ 举例

Currently in Arad, need to get to Bucharest

- **States**: the various cities you could be located in.
- **Actions**: drive between neighboring cities.
- **Initial state**: in Arad
- **Goal**: in Bucharest
- **Solution**: the route, the sequence of cities to travel through to get to Bucharest.



1.1 基本概念

- 求解算法的性能
  - 完备性：当问题有解时，这个算法能否保证找到解。
  - 最优性：搜索策略能否找到最优解。
  - 时间复杂度：找到解所需要的时间，也叫搜索代价
  - 空间复杂度：执行搜索过程中需要多少内存空间

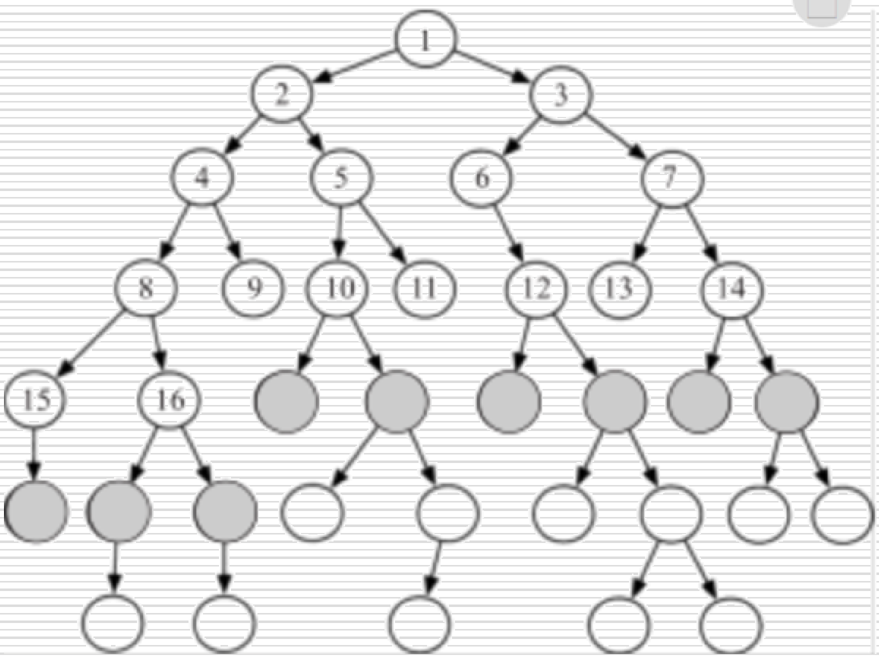
1.2 盲目搜索

1. 宽度优先搜索
2. 深度优先搜索
3. 深度受限搜索
4. 迭代加深搜索
5. 双向搜索

1.2 盲目搜索

1. 宽度优先搜索（BFS）

- 解的深度为d、最大分支为b;



算法 宽度优先搜索 BFS

输入: 初始状态 s_0, 动作集合, 转移模型 T, 目标检测函数

```
初始化队列 queue ← [s0]
while queue 非空 do
    s ← queue.pop()
    for s 的所有可行动作 a do
        获取下一状态 s' = T(s, a)
        if s' 是目标 then
            return 搜索路径
        else
            queue.append(s')
        end if
    end for
end while
return 问题无解
```

- 节点扩展顺序与目标节点的位置无关;
- 用一个先进先出（FIFO）队列实现。

- 完备性;
- 非最优;
- 时间复杂度O(b^d);
- 空间复杂度O(b^d);

2. 深度优先搜索（DFS）

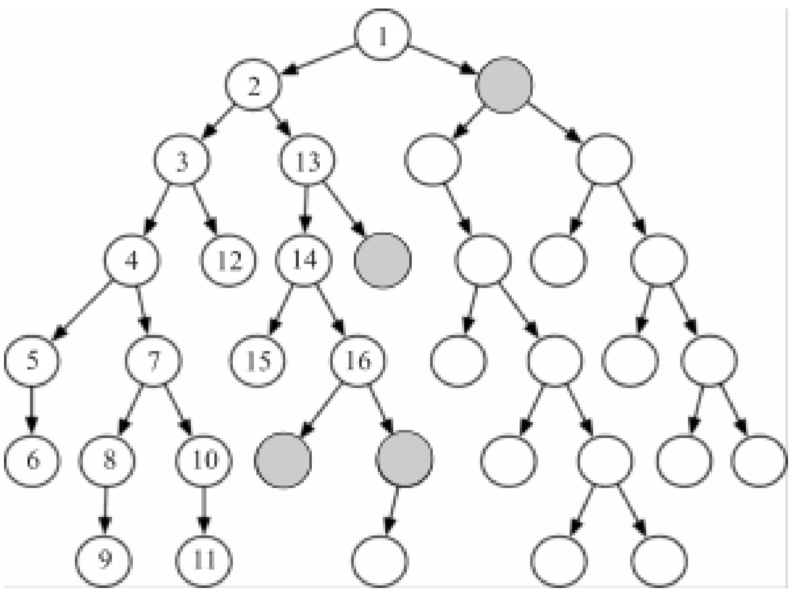


图 3-5 深度优先搜索中节点的扩展顺序

算法 深度优先搜索 DFS

输入: 初始状态 s_0, 动作集合, 转移模型 T, 目标检测函数

```
if s_0 下无可执行动作 then
    return 当前无解
else if s_0 是目标 then
    return 搜索路径
else
    for all s_0 的可行动作 a do
        获取下一状态 s' = T(s_0, a)
        若 s' 未访问, 则以 s' 为初始状态递归执行 DFS
    end for
end if
```

- **节点扩展顺序与目标节点的位置无关；**
- **用一个后进先出堆栈实现。**

1.2 盲目搜索

3. 深度受限搜索（Deep-limited Search）

算法 有界深度搜索 DLS

输入: 初始状态 s_0, 动作集合, 转移模型 T, 目标检测函数, 深度限制 d

```
if d = 0 then
    return 当前无解
else if s0 是目标 then
    return 搜索路径
else
    for all s0 的可行动作 a do
        获取下一状态 s' = T(s0, a)
        若 s' 未访问, 则以 s' 为初始状态, d − 1 为深度限制, 递归执行 DLS
    end for
end if
```

- **若状态空间无限，深度优先搜索可能会出现循环，搜索失败；**
- **通过预定一个深度限制L 来解决这个问题。**

1.2 盲目搜索

4. 迭代加深搜索（Iterative Deepening Search）

算法 迭代加深搜索 IDS

输入: 初始状态 s_0, 动作集合, 转移模型 T, 目标检测函数

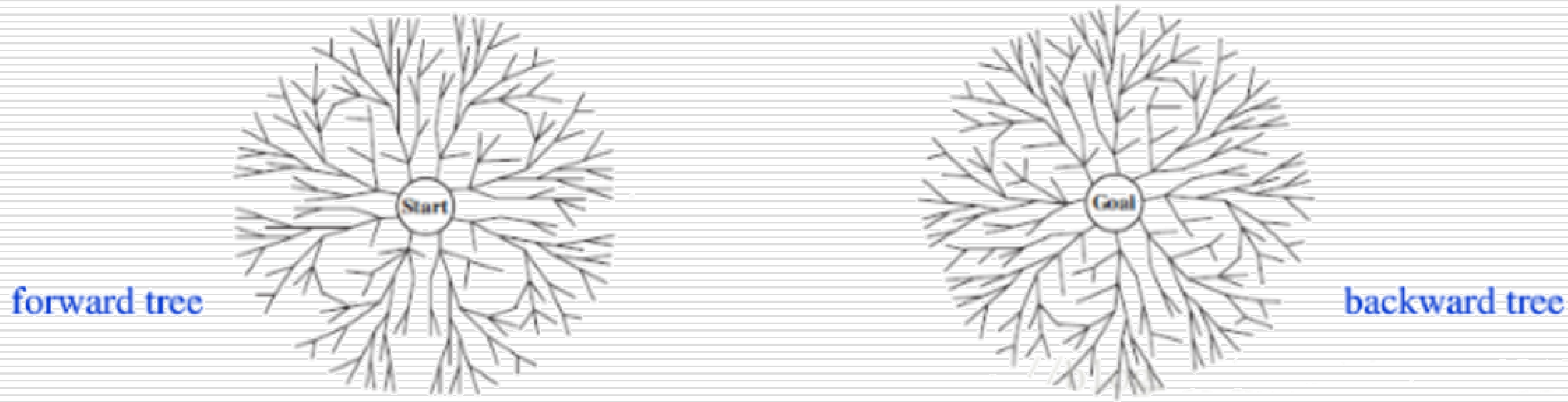
for $d = 1$ **to** $+\infty$ **do**
    以 s_0 为初始状态, d 为深度限制执行 DLS
    if 本次搜索找到解 **then**
        return 搜索路径
    else if 本次搜索所有节点深度小于 d **then**
        return 搜索失败
    end if
end for

- **深度受限搜索的L值难以选择；**
- **迭代加深搜索：以深度优先搜索相同的顺序访问搜索树的节点，但先访问节点的累积顺序实际是宽度优先。**

1.2 盲目搜索

5. 双向搜索（Bidirectional search）

它同时进行两个搜索：一个是从初始状态向前搜索，二另一个则从目标向后搜索。当两者在中间相遇时停止。



- **同时进行两个搜索：一个是从初始状态向前搜索，二另一个则从目标向后搜索；**
- **当两者在中间相遇时停止；**
- **在一定程度上能减小复杂度。**

1.2 盲目搜索

□ 盲目搜索策略评估

Evaluation of Uninformed Tree-search Strategies

无信息树搜索策略评价

Criterion	Breadth First	Uniform Cost	Depth First	Depth Limited	Iterative Deepening	Bidirectional
Complete	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Where
- b -- maximum branching factor of the tree
- d -- depth of the shallowest solution
- m -- maximum depth of the tree
- l -- the depth limit

- a -- complete if b is finite
- b -- complete if step costs ϵ for positive
- c -- optimal if step costs are all identical
- d -- if both directions use breadth-first search

1.3 启发式搜索

1. 基础概念
2. A*
3. IDA*
4. 启发式函数的设计

1.3 启发式搜索

1. 基础概念

- 启发式搜索又叫有信息的搜索，它利用问题所拥有的启发信息来引导搜索，达到减少搜索范围，降低问题复杂度的目的。
  - 无信息搜索对所有的可能路径节点一视同仁，而启发式搜索可以指导搜索向最有希望的方向前进
  - 如何评估一个节点的重要性？ - 评估函数

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

  - 其中g(x)是从初始节点到节点x付出的实际代价；而h(x)是从节点x到目标节点的最优路径的估计代价。h(x)建模了启发式搜索问题中的启发信息，是算法的关键。启发式函数的设计非常重要，合理的定义才能让搜索算法找到一个最优的问题解。

1.3 启发式搜索

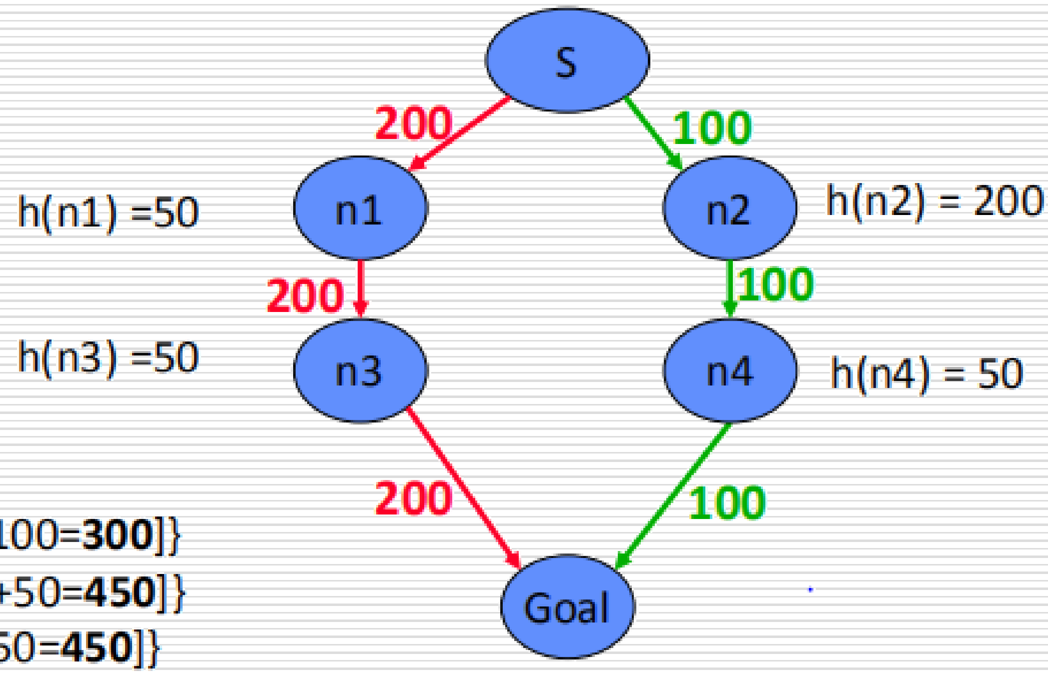
2. A*搜索算法

- A*算法可以看作是BFS算法的升级版，在原有的BFS算法的基础上加入了启发式信息。
  - 算法描述：从起始节点开始，不断查询周围可到达节点的状态并计算它们的f(x)，h(x)与g(x)的值，选取评估函数f(x)最小的节点进行下一步扩展，并同时更新已经被访问过的节点的g(x)，直到找到目标节点；
  - 算法优缺点：拥有BFS速度较快的优点，但是因为它要维护“开启列表”以及“关闭列表”，并且需要反复查询状态。因此它的空间复杂度是指数级的。

2. A*搜索算法

→ step cost = 200

→ step cost = 100



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

{S} → {n1 [200+50=**250**], n2 [200+100=**300**]}

→ {n2 [100+200=**300**], n3 [400+50=**450**]}

→ {n4 [200+50=**250**], n3 [400+50=**450**]}

→ {goal [300+0=**300**], n3 [400+50=**450**]}

1.3 启发式搜索

3. IDA*（迭代加深A*）搜索算法

- IDA* 是迭代加深深度优先搜索算法（IDS）的扩展。因为它不需要去维护表，因此它的空间复杂度远远小于A*。在搜索图为稀疏有向图的时候，它的性能会比A*更好。
  - 算法描述：在算法迭代的每一步，IDA*都进行深度优先搜索，在某一步所有可访问节点对应的最小可估价函数值大于某个给定的阈值的时候，将会剪枝；
  - 算法优缺点：当问题要求空间复杂度比较低的时候，IDA*更有优势。

1.3 启发式搜索

4. 启发式函数设计

- 启发式函数h(n)告诉算法从任何节点到目标节点的最小代价估计值，其选取很大程度影响算法性能。

h(n) 的值	描述	性能变化
h(n) = 0	只有g(n)起作用，退化为Dijkstra算法	保证找到最短路径
h(n) <= h*(n)		保证能找到最短路径
h(n) = h*(n)	只遵循最佳路径不会扩展其它节点	运行速度快并且能找到最短路径
h(n) > h*(n)		不能保证找到最短路径

4. 启发式函数设计

- 启发式函数h(n)告诉算法从任何节点到目标节点的最小代价估计值，其选取很大程度影响算法性能。
- 性质1：可采纳的（admissible）
  - 当评估函数的预估值小于等于真实值时，算法必然可以找到一条从起始节点到最终节点的最短路径。这种性质叫做相容。

$$h(n) \le h^*(n)$$

- 性质2： 单调的 （consistent）
  - 当节点n的评估函数值永远小于等于它的扩展节点n’的评估函数值加上扩展代价时，则启发式函数设计是单调的。

$$h(n) \le \cos t(n,n') + h(n')$$

1.3 启发式搜索

4. 启发式函数设计

- 不同的应用场景下有很多可选择的启发式函数。比如在网格地图中，一般使用以下几种启发式函数：
  - 在正方形网格中，允许向4邻域的移动，使用曼哈顿距离（L1）
  - 在正方形网格中，允许向8邻域的移动，使用对角线距离（L∞）等等

- 启发函数没有限制，大家可以多尝试几种。

1.4 博弈树搜索

1. 两玩家零和博弈问题

2. 博弈树

3. Minimax搜索

4. Alpha-beta剪枝

1.4 博弈树搜索

1. 两玩家零和博弈问题

- 两名玩家轮流行动，进行博弈
  - 有限：行动的个数有限
  - 确定性：不存在随机性
  - 信息完备性：博弈双方知道所处状态的全部信息
  - 零和性：一方的损失相当于另一方的收益，总收益为0
    - 结局有三种可能：玩家A获胜、玩家B获胜、平局（或两种可能，无平局）

1.4 博弈树搜索

1. 两玩家零和博弈问题

- **An example: Rock, Paper, Scissors**
  - Scissors cut paper, paper covers rock, rock smashes scissors
  - Represented as a matrix: Player I chooses a row, Player II chooses a column
  - Payoff to each player in each cell (Pl.I / Pl.II)
  - 1: win, 0: tie, -1: loss
  - so it's zero-sum

Player I \ Player II	R	P	S
R	0/0	-1/1	1/-1
P	1/-1	0/0	-1/1
S	-1/1	1/-1	0/0

1.4 博弈树搜索

2. 博弈树

- 节点（node）：表示问题的状态（state）。
  - 分为内部节点（interior node）和叶子节点（leaf node）
- 扩展节点：行动（action）。
- 双方轮流扩展节点：两个玩家的行动逐层交替出现。
- 博弈树的值：博弈树搜索的目的，找出对双方都是最优的子节点的值。给定叶子节点的效益值，搜索内部节点的值。
- 评价函数：对当前节点的优劣评分。在有深度限制时，原来的内部节点会充当叶子节点的作用，此时以评价函数值作为效益值的估计。

1.4 博弈树搜索

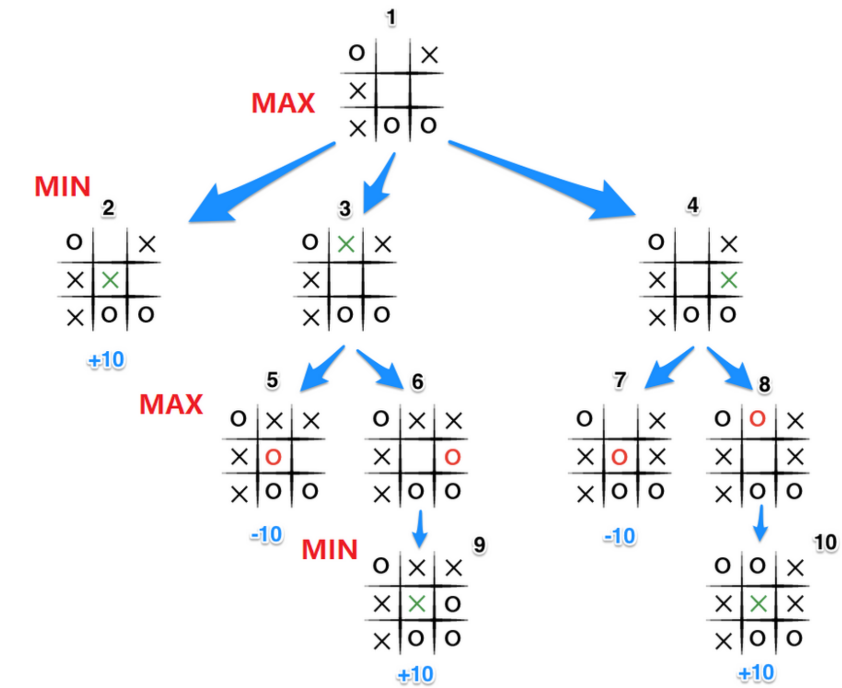
2. 博弈树

- 要提高博弈问题求解程序的效率，应作到如下两点：
  - 改进生成过程，使之只生成好的走步，如按棋谱的方法生成下一步；
  - 改进测试过程，使最好的步骤能够及时被确认。
- 要达到上述目的有效途径是使用启发式方法引导搜索过程，使其只生成可能赢的走步。而这样的博弈程序应具备：
  - 一个好的（即只产生可能赢棋步骤的）生成过程。
  - 一个好的静态估计函数。
- 下面介绍博弈中两种最基本的搜索方法。

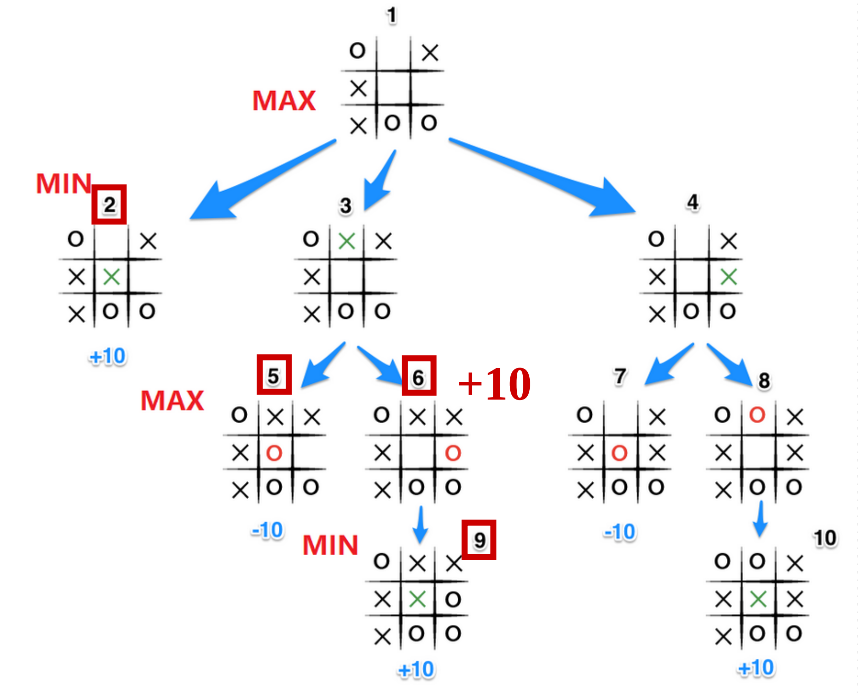
1.4 博弈树搜索

3. Minimax搜索

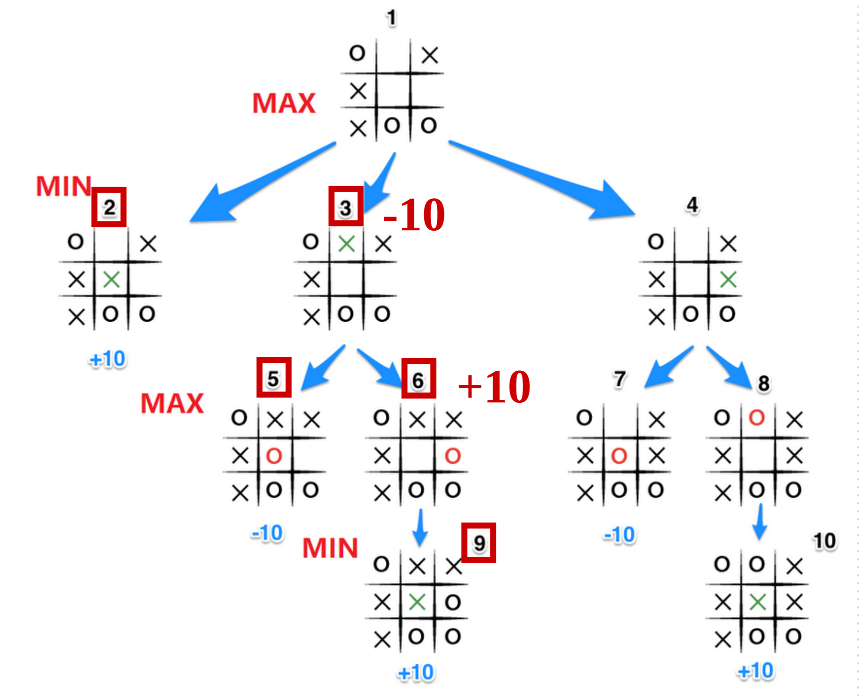
- 假设：
  - 玩家A和玩家B的行动逐层交替；
  - A和B的利益关系对立，即假设A要使分数更大，B就要使分数更小；
  - A和B均采取最优策略。
- Minimax搜索：找到博弈树中内部节点的值，其中Max节点（A）的每一步扩展要使收益最大，Min节点（B）的扩展要使收益最小。



3. Minimax搜索

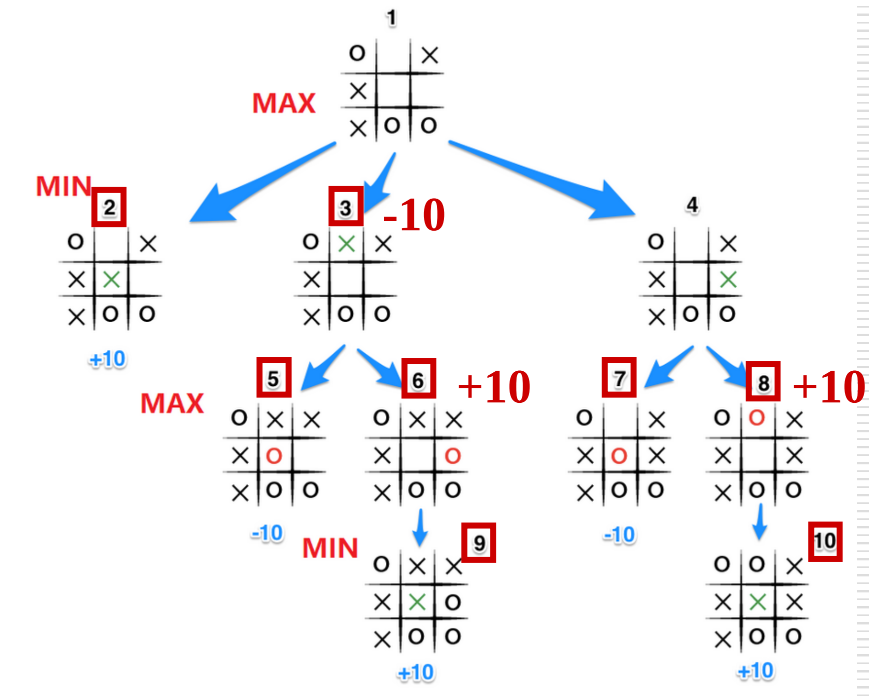


3. Minimax搜索



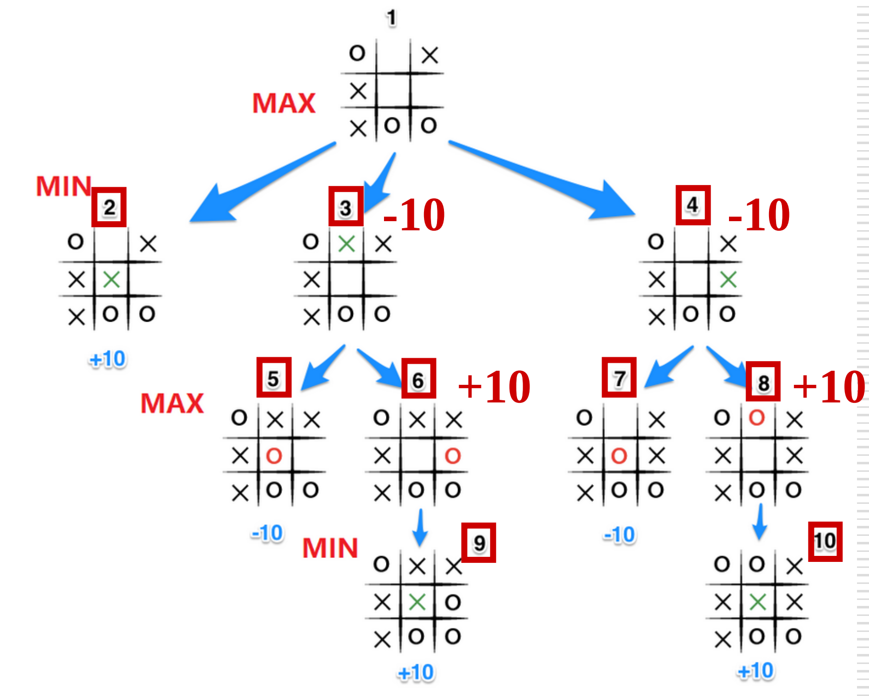
1.4 博弈树搜索

3. Minimax搜索



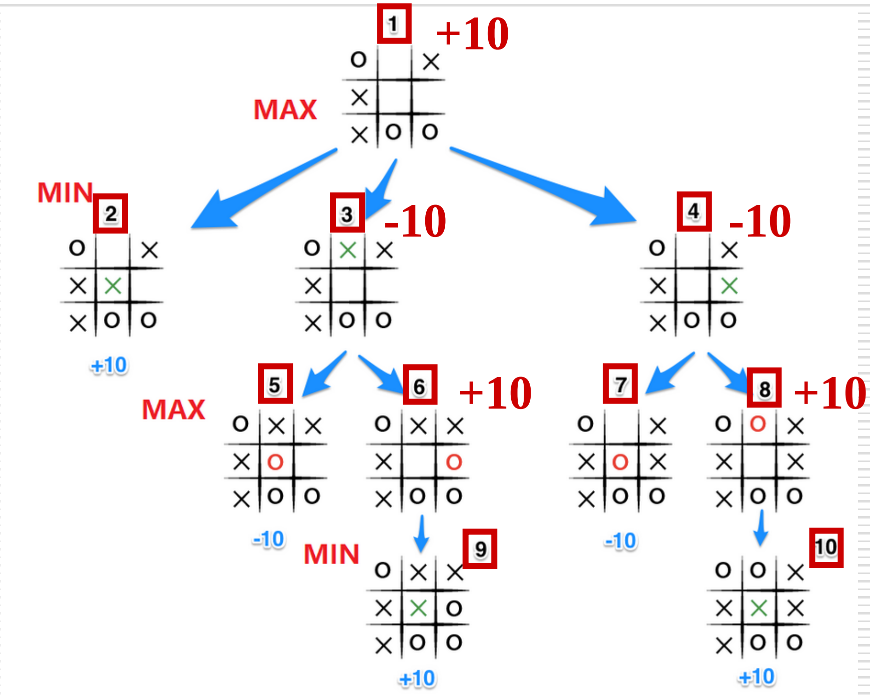
1.4 博弈树搜索

3. Minimax搜索



1.4 博弈树搜索

3. Minimax搜索



1.4 博弈树搜索

3. Minimax搜索

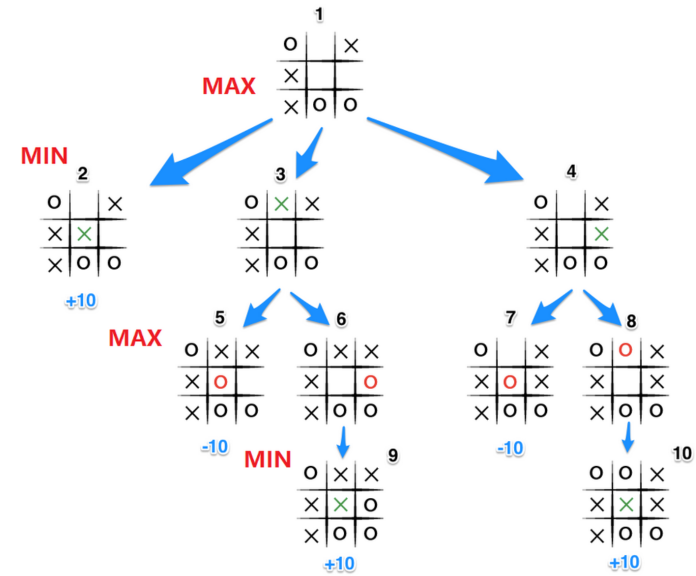
function MINIMAX-DECISION(*state*) **returns** *an action*
  return arg max$_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT(*state*, *a*))

---

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



4. Alpha-beta剪枝

- Minimax搜索：随着博弈的进行，必须检查的游戏状态的数目呈指数增长；
  - **我们只需要知道博弈过程所对应路径上的节点值；**
- Alpha-beta剪枝：剪掉不可能影响决策的分支，尽可能地消除部分搜索树。
  - **Max节点记录alpha值，Min节点记录beta值**
  - **Max节点的alpha剪枝：效益值 ≥ 任何祖先Min节点的beta值**
  - **Min节点的beta剪枝：效益值 ≤ 任何祖先Max节点的alpha值**

4. Alpha-beta剪枝

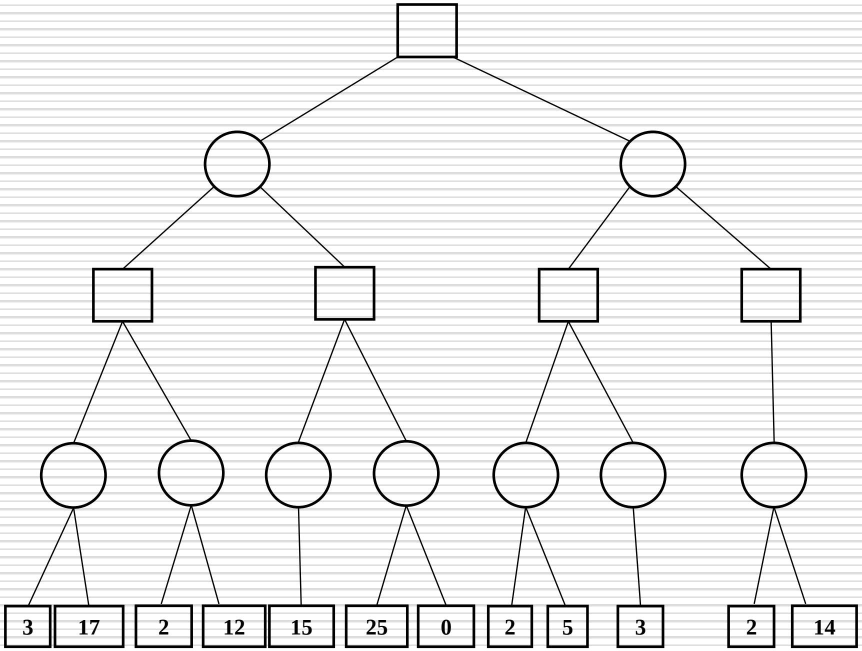
```
function ALPHA-BETA-SEARCH(state) returns an action
   v ← MAX-VALUE(state, −∞, +∞)
   return the action in ACTIONS(state) with value v
```

```
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
```

1.4 博弈树搜索

4. Alpha-beta剪枝

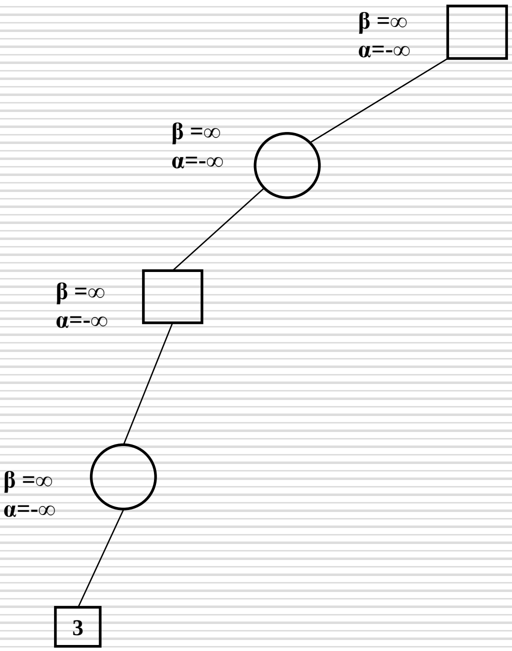


方形:Max节点
圆形:Min节点

请写出对该博弈树进行搜索的过程，要求使用结合Alpha-beta剪枝的深度优先Minimax算法。

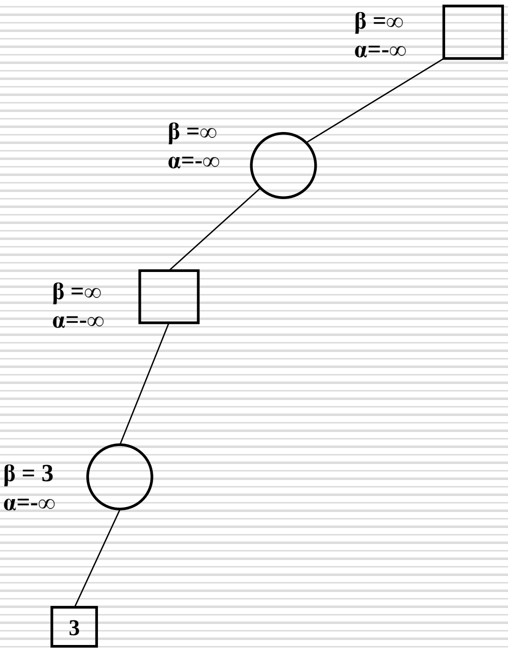
1.4 博弈树搜索

4. Alpha-beta剪枝



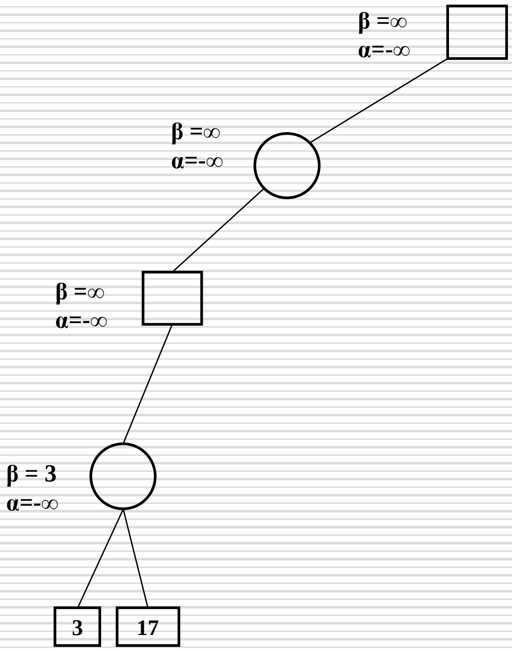
1.4 博弈树搜索

4. Alpha-beta剪枝



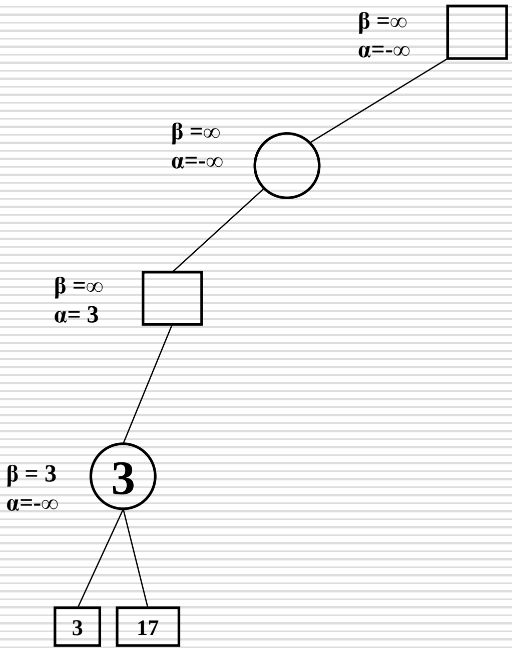
1.4 博弈树搜索

4. Alpha-beta剪枝



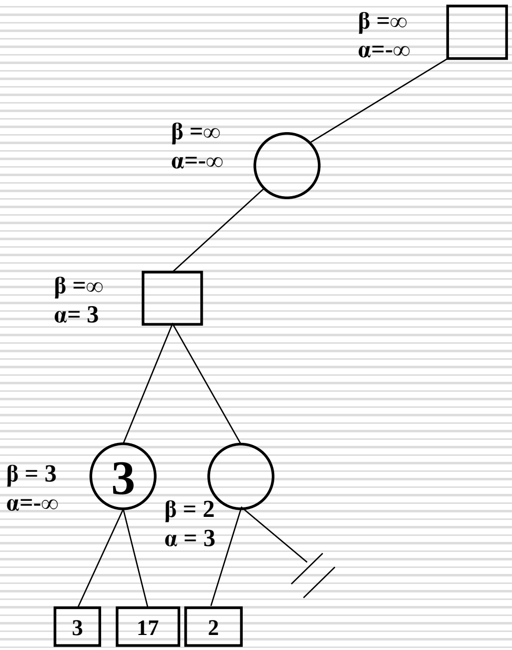
1.4 博弈树搜索

4. Alpha-beta剪枝

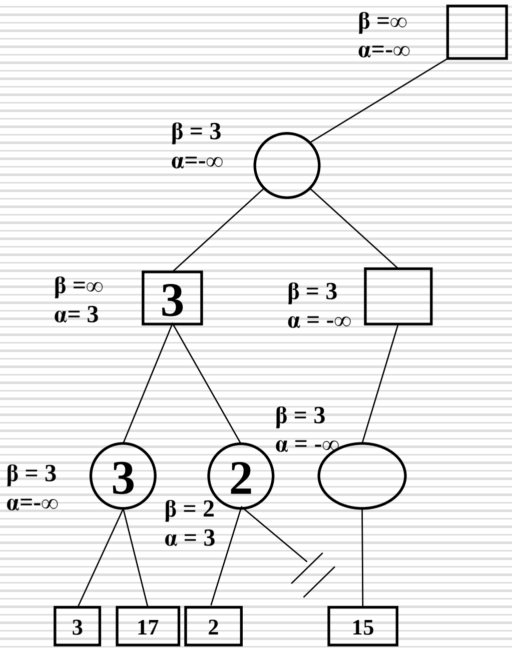


1.4 博弈树搜索

4. Alpha-beta剪枝

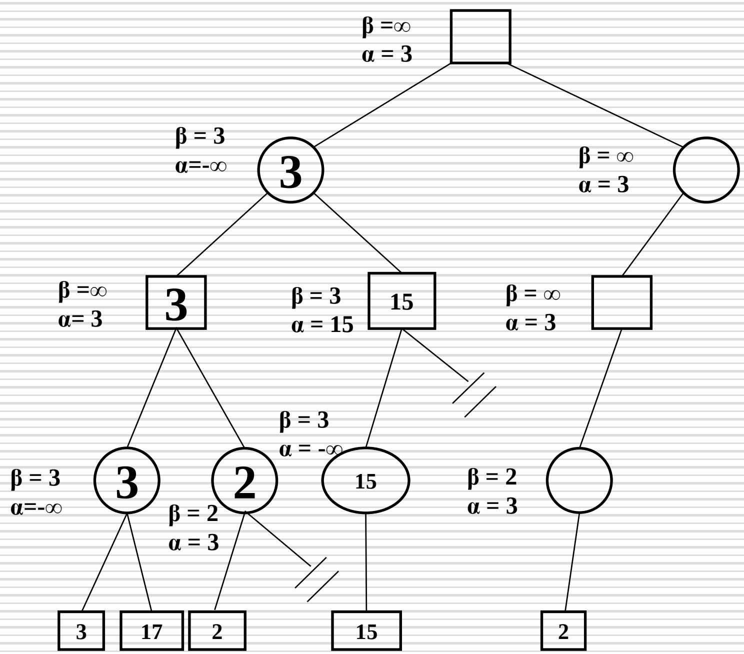


4. Alpha-beta剪枝



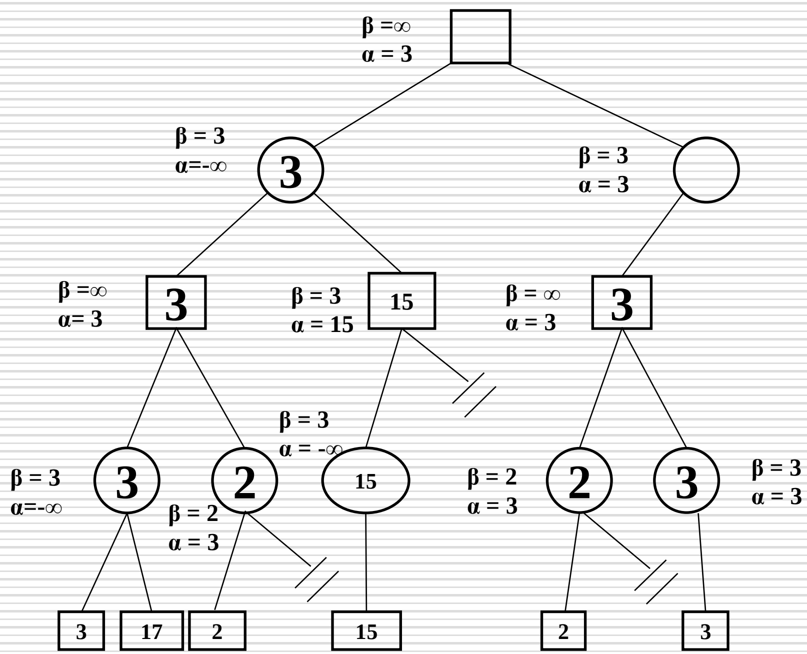
1.4 博弈树搜索

4. Alpha-beta剪枝



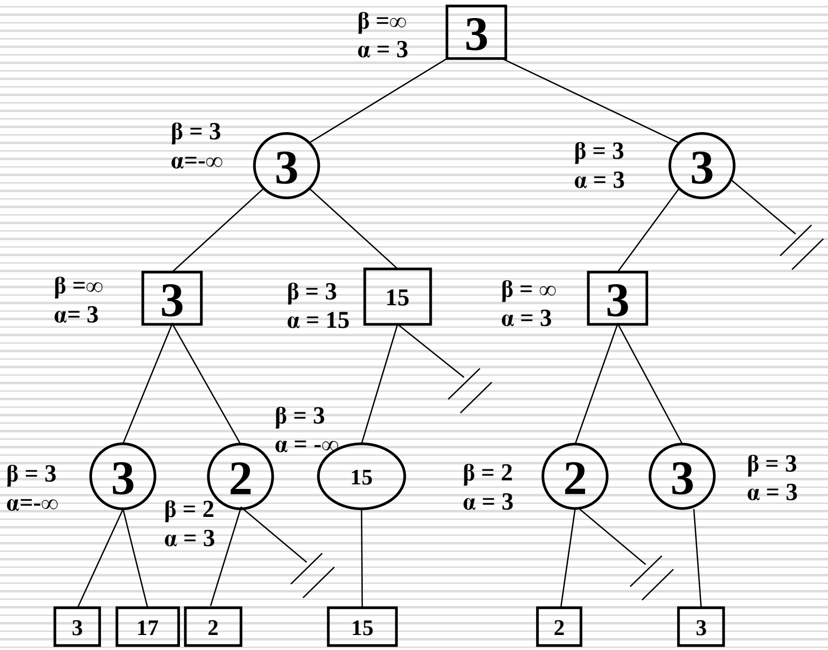
1.4 博弈树搜索

4. Alpha-beta剪枝



1.4 博弈树搜索

4. Alpha-beta剪枝



2.1 利用盲目搜索解决迷宫问题

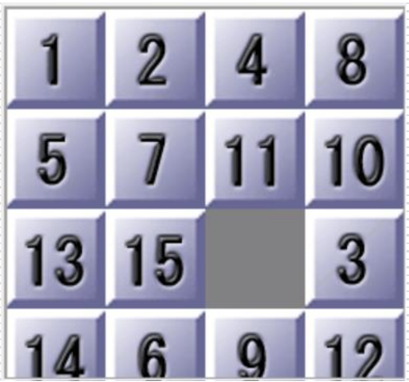
- 尝试利用DFS、BFS、深度受限算法、迭代加深算法、双向搜索算法解决迷宫问题:
  - S表示起点;
  - E表示终点;
  - 1表示墙;
  - 0是可通行。

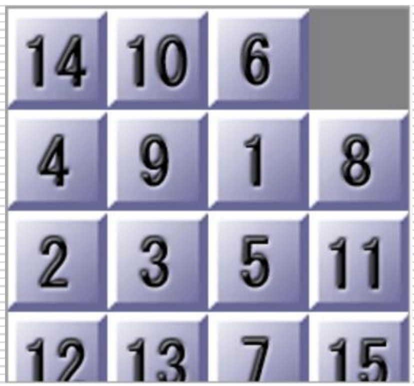
```
111111111111111111111111111111111111
1000000000000000000000000000000000S1
101111111111111111111111101111111101
101100010001000000111111100011000001
101101010101011110111111111011011111
101101010101000000000000011011000001
101101010101010111100111000011111101
101001010100010000110111111110000001
101101010111111110110000000011011111
101101000110000000111111111011000001
100001111110111111100000011011111101
111111000000100000001111011010000001
100000011111101111101000011011011111
101111110000001000000011111011000001
100000000111111011111111111011001101
111111111100000000000000000011111101
1E0000000001111111111111111000000001
111111111111111111111111111111111111
```

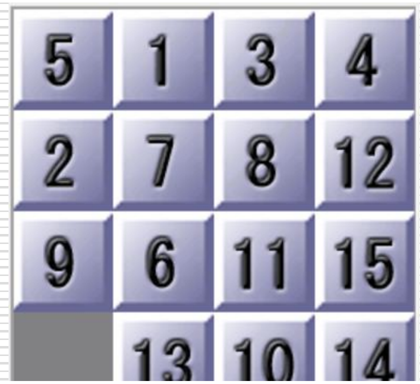
- **无需提交**

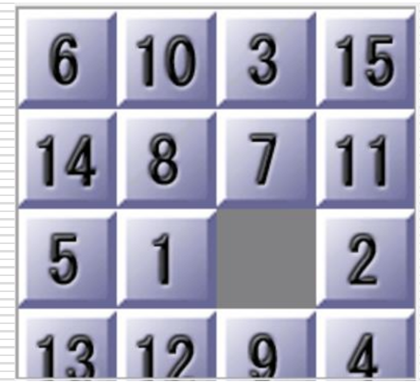
2.2 利用启发式搜索解决15-Puzzle问题

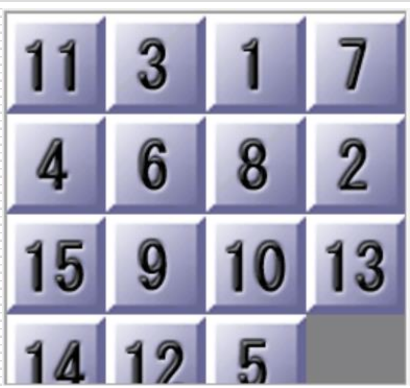
- 尝试使用A*与IDA*算法解决15-Puzzle问题，启发式函数可以自己选取，最好多尝试几种不同的启发式函数**（无需提交）**

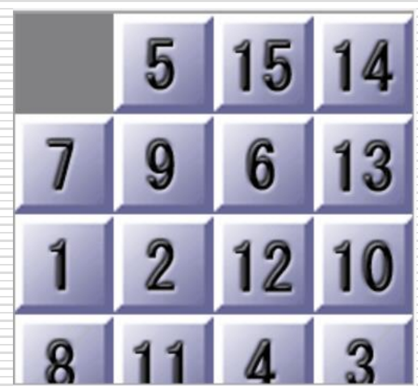








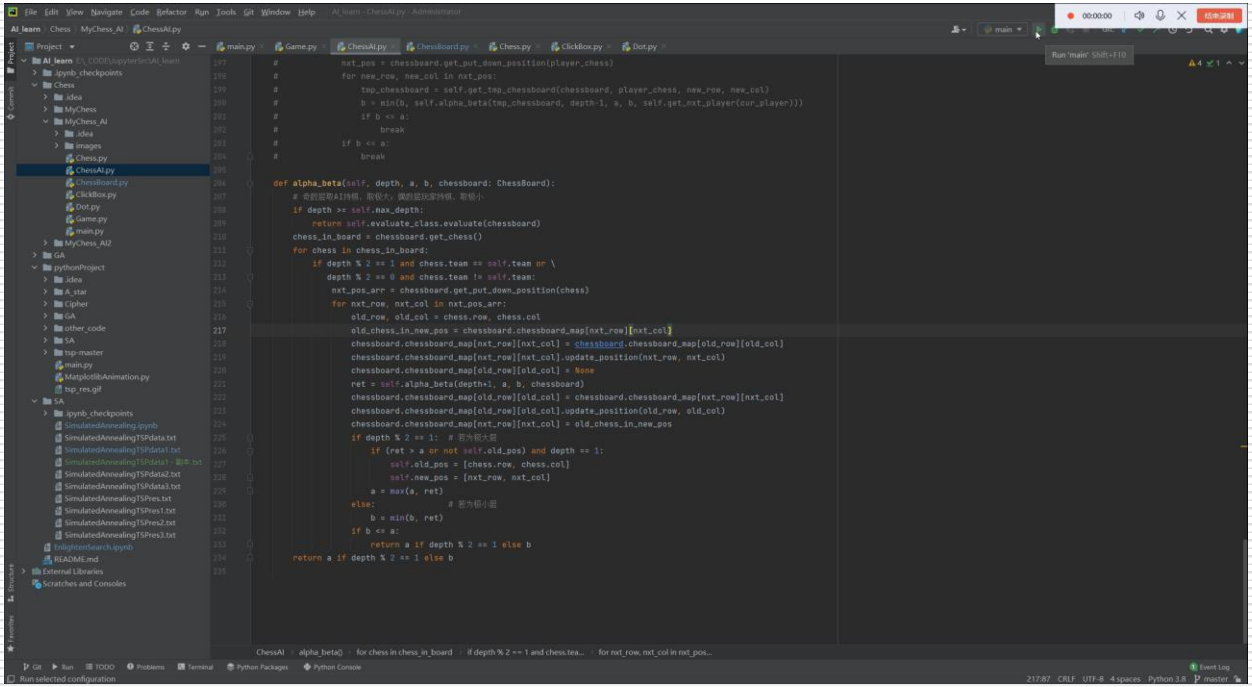




2.3 利用博弈树搜索实现象棋AI

- 编写一个中国象棋博弈程序，要求用alpha-beta剪枝算法，可以实现两个AI对弈。
  - 一方由人类点击或者AI算法控制。
  - 一方由内置规则AI控制。
  - 算法支持红黑双方互换

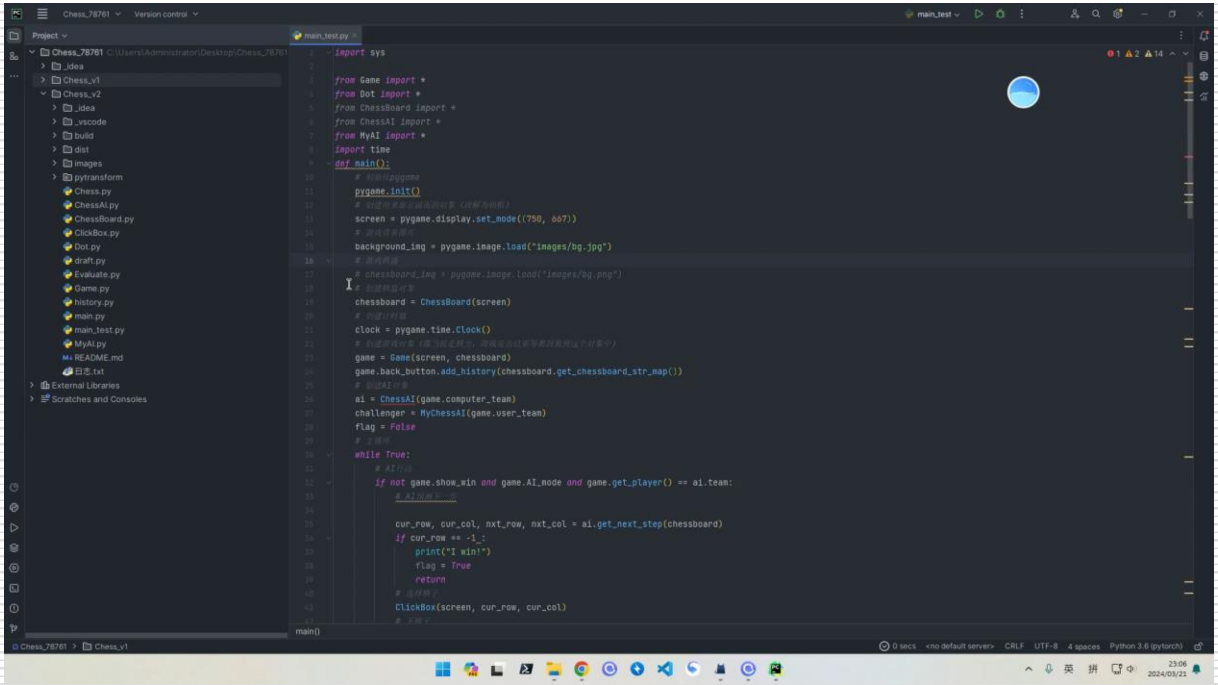
人类
Vs
内置规则



2.3 利用博弈树搜索实现象棋AI

- 编写一个中国象棋博弈程序，要求用alpha-beta剪枝算法，可以实现两个AI对弈。
  - 一方由人类点击或者AI算法控制。
  - 一方由内置规则AI控制。
  - 算法支持红黑双方互换

AI算法
Vs
内置规则



3. 作业提交说明

- 压缩包命名为："学号_姓名_作业编号"，例：20240326_张三_实验5。
- 每次作业文件下包含两部分：code文件夹和实验报告PDF文件。
  - code文件夹：存放实验代码；
  - PDF文件格式参考发的模板。
- 如果需要更新提交的版本，则在后面加_v2，_v3。如第一版是"学号_姓名_作业编号.zip"，第二版是"学号_姓名_作业编号_v2.zip"，依此类推。
- 截至日期：**2024年4月16日晚24点**。
- 提交邮箱：zhangyc8@mail2.sysu.edu.cn。