启发式搜索 Heuristic Search (A*)

heuristic 智能搜索 / 根据某一项条件来不断地优化搜索的方向

(一边搜索，一边有所谓的思考在里面　思考型搜索)

基于 BFS 代码

```
def BFS(graph, start, end):

   queue = []
   queue.append([start])
   visited.add(start)

   while queue:
      node = queue.pop() # can we add more intelligence here ?
      visited.add(node)

      process(node)
      nodes = generate_related_nodes(node)
      queue.push(nodes)
```

优化

A* search

```
def AstarSearch(graph, start, end):

   pq = collections.priority_queue() # 优先级 -> 估价函数
   pq.append([start])
   visited.add(start)

   while pq:
      node = pq.pop() # can we add more intelligence here ?
      visited.add(node)

      process(node)
      nodes = generate_related_nodes(node)
      unvisited = [node for node in nodes if node not in visited]
      pq.push(unvisited)
```

启发式 函数

优先级定义函数

估价函数

估价函数

启发式函数：h(n)，它用来评价哪些结点最有希望的是一个我们要找的结点，h(n) 会返回一个非负实数,也可以认为是从结点n的目标结点路径的估计成本。

启发式函数是一种告知搜索方向的方法。它提供了一种明智的方法来猜测哪个邻居结点会导向一个目标。

```java
//Java
    public class AStar
    {
        public final static int BAR = 1; // 障碍值
        public final static int PATH = 2; // 路径
        public final static int DIRECT_VALUE = 10; // 横竖移动代价
        public final static int OBLIQUE_VALUE = 14; // 斜移动代价

        Queue<Node> openList = new PriorityQueue<Node>(); // 优先队列(升序)
        List<Node> closeList = new ArrayList<Node>();

        /**
         * 开始算法
         */
        public void start(MapInfo mapInfo)
        {
            if(mapInfo==null) return;
            // clean
            openList.clear();
            closeList.clear();
            // 开始搜索
            openList.add(mapInfo.start);
            moveNodes(mapInfo);
        }


        /**
         * 移动当前结点
         */
        private void moveNodes(MapInfo mapInfo)
        {
            while (!openList.isEmpty())
            {
                Node current = openList.poll();
                closeList.add(current);
                addNeighborNodeInOpen(mapInfo,current);
                if (isCoordInClose(mapInfo.end.coord))
                {
                    drawPath(mapInfo.maps, mapInfo.end);
                    break;
                }
            }
        }
```

```
        /**
         * 在二维数组中绘制路径
         */
        private void drawPath(int[][] maps, Node end)
        {
            if(end==null||maps==null) return;
            System.out.println("总代价: " + end.G);
            while (end != null)
            {
                Coord c = end.coord;
                maps[c.y][c.x] = PATH;
                end = end.parent;
            }
        }
        /**
         * 添加所有邻结点到open表
         */
        private void addNeighborNodeInOpen(MapInfo mapInfo,Node
current)
        {
            int x = current.coord.x;
            int y = current.coord.y;
            // 左
            addNeighborNodeInOpen(mapInfo,current, x - 1, y,
DIRECT_VALUE);
            // 上
            addNeighborNodeInOpen(mapInfo,current, x, y - 1,
DIRECT_VALUE);
            // 右
            addNeighborNodeInOpen(mapInfo,current, x + 1, y,
DIRECT_VALUE);
            // 下
            addNeighborNodeInOpen(mapInfo,current, x, y + 1,
DIRECT_VALUE);
            // 左上
            addNeighborNodeInOpen(mapInfo,current, x - 1, y - 1,
OBLIQUE_VALUE);
            // 右上
            addNeighborNodeInOpen(mapInfo,current, x + 1, y - 1,
OBLIQUE_VALUE);
            // 右下
            addNeighborNodeInOpen(mapInfo,current, x + 1, y + 1,
OBLIQUE_VALUE);
            // 左下
            addNeighborNodeInOpen(mapInfo,current, x - 1, y + 1,
OBLIQUE_VALUE);
        }
        /**
```

```
         * 添加一个邻结点到open表
         */
        private void addNeighborNodeInOpen(MapInfo mapInfo,Node
current, int x, int y, int value)
        {
            if (canAddNodeToOpen(mapInfo,x, y))
            {
                Node end=mapInfo.end;
                Coord coord = new Coord(x, y);
                int G = current.G + value; // 计算邻结点的G值
                Node child = findNodeInOpen(coord);
                if (child == null)
                {
                    int H=calcH(end.coord,coord); // 计算H值
                    if(isEndNode(end.coord,coord))
                    {
                        child=end;
                        child.parent=current;
                        child.G=G;
                        child.H=H;
                    }
                    else
                    {
                        child = new Node(coord, current, G, H);
                    }
                    openList.add(child);
                }
                else if (child.G > G)
                {
                    child.G = G;
                    child.parent = current;
                    openList.add(child);
                }
            }
        }
```

```
124         /**
125          * 从Open列表中查找结点
126          */
127         private Node findNodeInOpen(Coord coord)
128         {
129             if (coord == null || openList.isEmpty()) return null;
130             for (Node node : openList)
131             {
132                 if (node.coord.equals(coord))
133                 {
134                     return node;
135                 }
136             }
137             return null;
138         }
139
140
141
142
143         /**
144          * 计算H的估值：“曼哈顿”法，坐标分别取差值相加
145          */
146         private int calcH(Coord end,Coord coord)
147         {
148             return Math.abs(end.x - coord.x)
149                     + Math.abs(end.y - coord.y);
150         }
151
152         /**
153          * 判断结点是否是最终结点
154          */
155         private boolean isEndNode(Coord end,Coord coord)
156         {
157             return coord != null && end.equals(coord);
158         }
159
```

```
160
161         /**
162          * 判断结点能否放入Open列表
163          */
164         private boolean canAddNodeToOpen(MapInfo mapInfo,int x, int y)
165         {
166             // 是否在地图中
167             if (x < 0 || x >= mapInfo.width || y < 0 || y >=
    mapInfo.hight) return false;
168             // 判断是否是不可通过的结点
169             if (mapInfo.maps[y][x] == BAR) return false;
170             // 判断结点是否存在close表
171             if (isCoordInClose(x, y)) return false;
172
173
174             return true;
175         }
177
178         /**
179          * 判断坐标是否在close表中
180          */
181         private boolean isCoordInClose(Coord coord)
182         {
183             return coord!=null&&isCoordInClose(coord.x, coord.y);
184         }
185
186
187         /**
188          * 判断坐标是否在close表中
189          */
190         private boolean isCoordInClose(int x, int y)
191         {
192             if (closeList.isEmpty()) return false;
193             for (Node node : closeList)
194             {
195                 if (node.coord.x == x && node.coord.y == y)
196                 {
197                     return true;
198                 }
199             }
200             return false;
201         }
202     }
```