A*算法实验报告

1. 算法设计

   或图通用的搜索算法描述如下：

   设S_0为初始状态，S_g为目标状态，$OPEN=\{\}$, $CLOSED=\{\}$

   1. 将S_0 推入$OPEN$表
   2. 在$OPEN$表中选出一个特定结点n放入$CLOSED$表，并将其从$OPEN$表中移除
   3. 若$n \in S_g$ ，则找到解，解为从S_0到n的路径。
   4. 生成n的后继，若后继结点不在$CLOSED$表中，则加入$OPEN$表
   5. 转到2

   当每次取出结点时选择$OPEN$表尾部结点时，搜索等同于广度优先，选择$OPEN$表头部结点时，等同于深度优先搜索。最佳优先搜索中，使用启发式函数f来对图中结点进行估计以确定最佳的结点。

   定义启发式函数$f(n)$为：

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

   A*算法中规定$h^*(n) \geq h(n) \geq 0$，其中$h^*(n)$定义为从结点n到S_g的实际最小费用。

2. 算法实现介绍

   a 星算法的主要代码部分位于`js/aStar.js`文件中，文件中实现了一个或图通用搜索算法抽象类`searchMethod`, 实现的图搜索算法只需继承该类并实现`addNodeVal`方法为新结点设置其启发值即可。在`searchMethod`类的实现中，$OPEN$表使用了一个堆来实现，每次获取堆中启发值最小的结点进行后继结点的派生。

   `node`类为搜索图中结点类，包含了从S_0到该结点的派生链，以及结点的评估值，提供了生成后继结点的方法。

   `bfs`类为广度优先搜索，实现的`addNodeVal`方法中实现的启发函数为$f(n_j)=f(n_i)+1$, 该启发函数的作用是令搜索图中每一层的结点的值相同，则`searchMethod`类将使用广度优先的方法进行搜索。

   `aStarH1`类使用的$h_1(n)$为放错位置的数字个数。估计函数$f_1(n)=d(n)+h_1(n)$其中$d(n)$表示结点n到S_0的距离。

   `aStarH2`类使用的$h_2(n)$为除空白区域外的放错位置的数字与实际位置的曼哈顿距离。

   使用浏览器打开`puzzle.html`即可使用程序，程序运行过程中将显示当前搜索的状态，$OPEN$表的结点数，总扩展节点数以及下一个评估值最小的结点。程序搜索完毕后可以点击显示路径以显示出最佳路径上的结点状态以及其评估函数值。

   根据要求程序验证了A*算法挑选出来求后继的点n必定满足: $f(n) \leq f^*(S_0)$：

   在每次从$OPEN$表取出用于求后继的点时都会更新搜索图的最大评估值：

```
if (this.currentNode.value > this.maxValue) {
            this.maxValue = this.currentNode.value;
}
```

搜索完成后当前节点到S_0的深度即为$f^*(S_0)$，此时若不符合条件则弹出错误：

```
if (this.currentNode.path.length < this.maxValue) {
            alert("f(n) <= f*(S0) violated!");
            console.log(this.maxValue);
}
```

验证h1(n)的单调性，显示凡A*算法挑选出来求后继的点n_i扩展的一个子结点n_j，检查是否满足：$h(n_i) \leq 1 + h(n_j)$：

每次求后继结点后都会进行单调检查，若不符合则显示错误：

由于$g(n_j) = g(n_i) + 1$，实际检查时的条件为$f(n_j) + 2 \geq f(n_i)$

```
if (child.value + 2 < this.currentNode.value) {
                alert("h(ni) <= 1+h(nj) Monotonicity violated!");
                throw "h(ni) <= 1+h(nj) Monotonicity violated!";
}
```

程序页面每次打开时会随机生成一个初始状态，完全随机时八数码有一般几率随机到无解的状态，因此在`js/puzzle.js`文件中对随机生成的状态进行了检查，检查方法为检查随机获得的状态与还原后的状态的逆序奇偶性是否相同，逆序指的是状态中每个数字前面比它大的数字的个数和。检查部分代码如下：

```
function getInversion(seq) {
    var result = 0;
    for (var i = 0; i < seq.length; i++) {
        for (var k = i + 1; k < seq.length; k++) {
            if (seq[i] > seq[k])
                result++;
        }
    }
    return result;
}

function getParity(inversion, blankSpot) {
    var val = inversion + Math.floor((blankSpot) / 3 + 1) +
Math.floor((blankSpot) % 3 + 1);
    return val % 2;
}

//Returns true if the sequence is not valid!
function checkSequence(seq) {
    var originZeroEle = seq[current_blank];
    var sequence = seq.slice();
    var originSequence = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
    originSequence[originZeroEle] = 0;
    for (var i = 0; i < sequence.length; i++) {
        if (i === current_blank)
```

```
            sequence[i] = 0;
        else if (sequence[i] < originZeroEle)
            sequence[i]++;
        if (originSequence[i] < originZeroEle)
            originSequence[i]++;
    }
    var inversion = getInversion(sequence), originInversion =
getInversion(originSequence);
    return getParity(inversion, current_blank) ===
getParity(originInversion, originZeroEle);
}
```

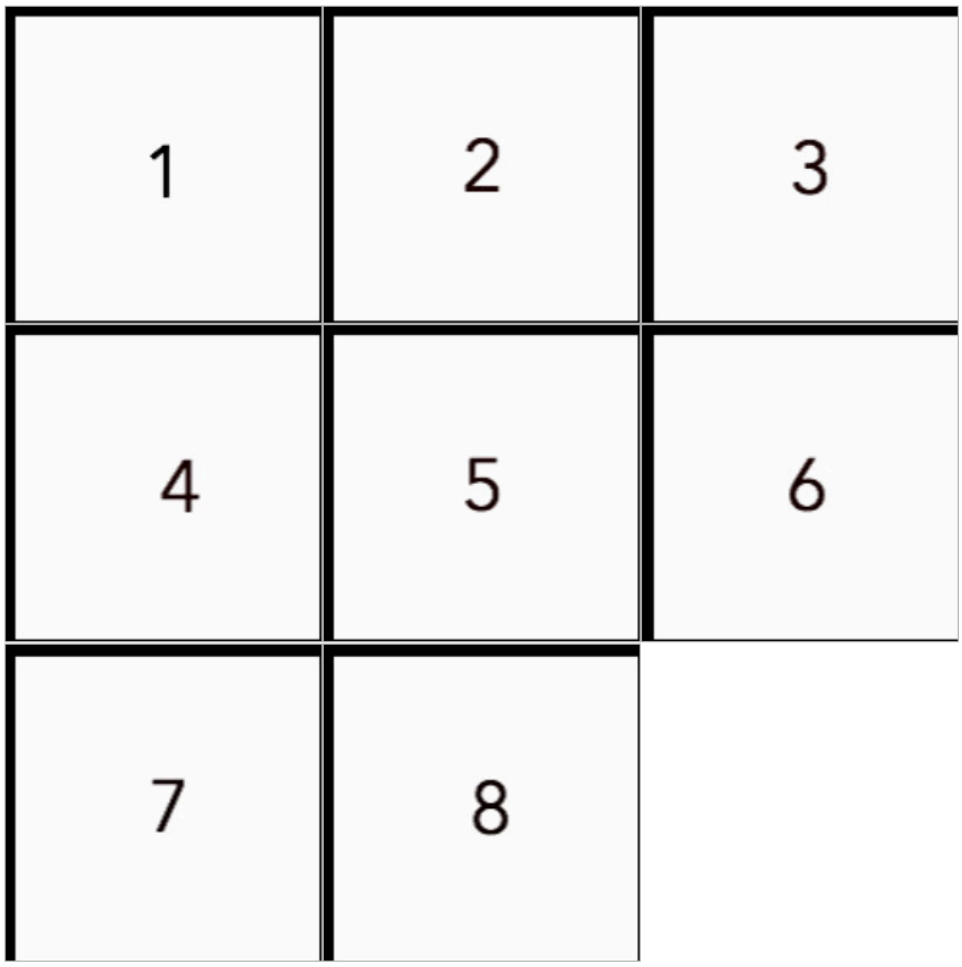
3. 测试结果

   启动程序后使用相同的初始状态进行测试，广度优先的结果如下：

广度优先算法

选择你自己的图片： 上传文件

搜索完成。步数：126240, 路径长度：25



还原 显示路径

总扩展结点数: 126240

OPEN表结点数：30594

当前节点评估值：24

OPEN表评估函值最小结点如下：

9,1,3
4,2,5
7,8,6

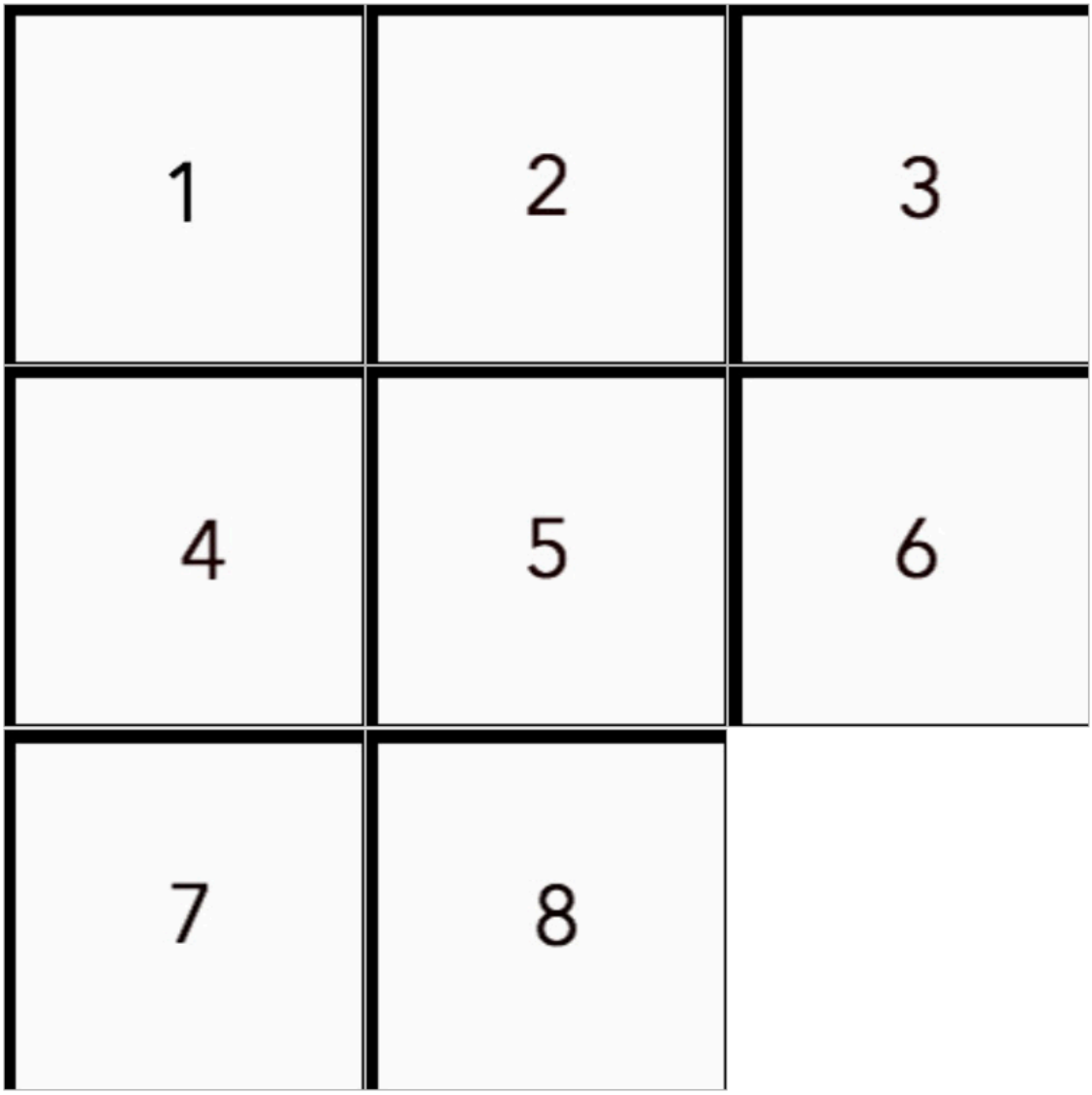
h_1的结果如下：

A*算法 N1

A星算法 H1

选择你自己的图片： 上传文件

搜索完成。步数：15348, 路径长度：25



还原 显示路径

总扩展结点数: 15348

OPEN表结点数：8576

当前节点评估值：25

OPEN表评估函值最小结点如下：

9,8,2

1,3,7
6,5,4

h_2的结果如下：

A星算法 H2

选择你自己的图片： 上传文件

搜索完成。步数： 1139, 路径长度： 25

1	2	3
4	5	6
7	8	

还原 显示路径

总扩展结点数: 1139

OPEN表结点数： 660

当前节点评估值： 25

当前节点评估值：25

OPEN表评估函值最小结点如下：

9,2,5
4,3,7
1,8,6

可观察到由于$h_2(n) \geq h_1(n)$，h_2的找到解的速度更高，而两张启发函数下的 A*算法执行的速度都显著高于非启发式的广度优先算法。