EXP1 Report

实验1.1

启发式函数

用曼哈顿距离作为启发式函数

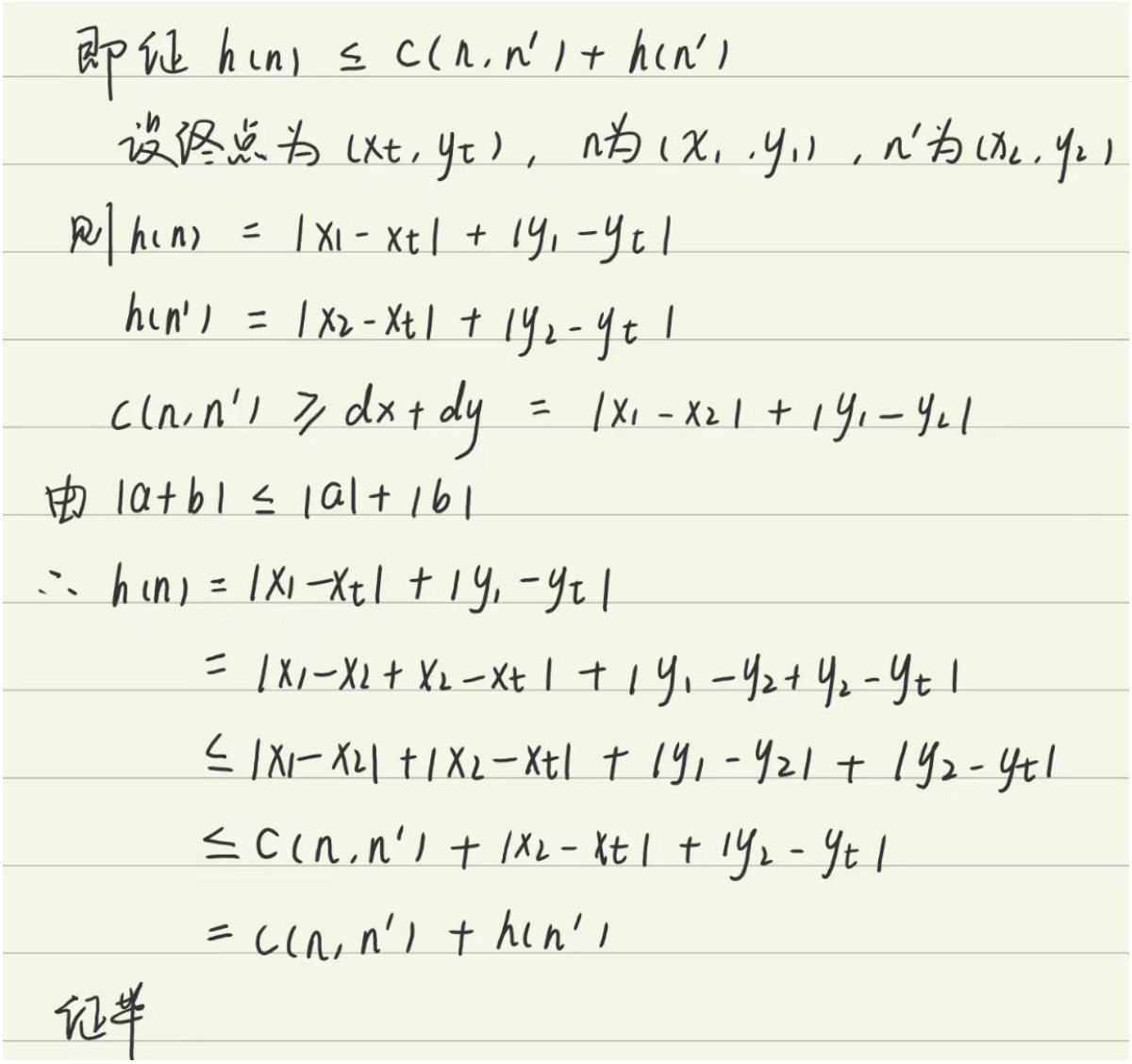
```
1  int Heuristic_Funtion(pair<int, int> point) {
2      return abs(point.first - end_point.first) + abs(point.second
   - end_point.second);
3  }
```

admissible

本题背景是一个网格问题，在网格问题中，只允许横向或者竖向移动，而曼哈顿距离采用的就是两点横纵坐标分别相减的绝对值之和，这也就保证了其不会高估实际代价，因此是admissible的

consistent

是consistent的，证明如下：



算法主要思路

评估函数f=g+h

- g是从出发到当前不重复路径的格子数
- h是曼哈顿距离，作为启发式函数

将初始格加入open_lilst，然后每次循环在open_list中选出评估函数g+h最小的格子进行扩展

> 由于open_list是优先队列，直接取出队头即可

- 若选出的格子已是终点：直接结束
- 若选出的格子已经耗尽体力（没有食物补充）：出队，并判断下一个队头
- 若不是上面两种情况：对其相邻格进行拓展，判断能否加入优先队列，拓展完成将当前格出队（已经经过）

> 类似于BFS，只是是按照相邻格拓展的

判断能否加入优先队列

相邻格也有可能是已经走过的格子，而最短路径中是不会走重复的格子的，因此需要通过格子结点的comefrom来回溯其父节点链，其相邻格是否在父节点链中出现，出现则不能加入优先队列

重复如上过程，直到优先队列为空或者到达终点

与一致代价搜索比较

将启发式函数设为0，退化为一致代价搜索

发现运行时间显著变长，算法性能不佳

而A*搜索不仅有较快的运行速度，在启发式函数一致时也能保证全局最优性

实验1.2

算法实现过程

建立棋盘，建立博弈树，设置搜索深度，设置评估函数

- 达到搜索深度限制时，返回当前棋盘得分
- 达成终局条件时，返回当前棋盘得分
- 其余情况：

  创建并遍历所有子节点，评估分数，并判断是否需要剪枝

  > 评估得分函数写在构造函数中，创建子节点的过程也就自动完成分数评估

- max层：

  max层需要得到最大值，父节点是min层需要最小值，因此在求值过程中，如果子节点max向上返回的值已经大于beta，说明之前已经有更小值返回到父节点min，该max结点已经不可能返回比beta更小的值（因为max只会越来越大），可以剪枝，直接返回评估值

- min层：

  min层需要得到最小值，父节点是max层需要最大值，因此在求值过程中，如果子节点min向上返回的值已经小于alpha，说明之前已经有更大值返回到父节点max，该min结点已经不可能返回比alpha更大的值（因为min只会越来越小），可以剪枝，直接返回评估值

实验结果

以第一个为例：


```
lab1 > Alpha_Beta > output > output_1.txt
1    R (1,8) (4,8)
2
```

输出的结果是将车从1，8移到4，8

也就是如下图所示：

会将死黑棋，下法合理

alpha-beta剪枝的影响

alphabeta剪枝减去了搜索树中大量不需要搜索的空间，由于搜索空间巨大(原来是$O(b^m)$)，减掉一个结点会将其后续的搜索全部节约掉，极大提高搜索效率

例如，在实验过程中，将搜索深度设置为4需要花费四分钟左右来得到结果

而设置为3只需要仅仅30s左右，由此可见，减掉不需要搜索的搜索树结点带来的效率收益是巨大的

评估函数的设计思路和效果

来自于三方面：

- 棋力

  不同棋子在不同位置的价值

- 棋子价值

  棋子固有价值

- 行棋可能性

  棋子可能的行动带来的价值

  在本实验中我使用吃掉棋子的价值不同来衡量，例如吃掉对面的将价值就很高

将这三个方面综合考虑，得到两边玩家不同的评估值，然后以我方为视角，用我方评估值减去对方评估值，作为在我方视角下最后的评估值

需要注意的是：

- 在本实验中，我对行棋吃掉士和象的价值进行了添加，仅代表我自身对吃掉它的评估，所以会导致下一步行棋有所变化
- 在本实验中，行棋可能性并未充分考虑到吃掉对面棋子后是否自身也会被吃掉，所以可能造成下一步行棋攻击倾向过于强烈的情况，导致自身被吃，甚至比较下来还更亏