《人工智能课程设计》实验一：A*算法求解 8 数码问题

装 订 线

学　　号	1953080
姓　　名	田宇
专　　业	计算机科学与技术
授课老师	王俊丽

1 实验概述

1.1 实验目的

熟悉和掌握启发式搜索策略的定义、评价函数 f(n)和算法过程，并利用 A*算法求解 8 数码问题，理解求解流程和搜索顺序。

1.2 实验内容及要求

（1） 以 8 数码问题为例，实现 A*算法的求解程序（编程语言不限），要求设计两种不同的启发函数 h(n)。

（2）设置相同初始状态和目标状态，针对不同的评价函数求得问题的解，比较它们对搜索算法性能的影响，包括扩展节点数、生成节点数和运行时间等。

（3）要求画出结果比较的图表，并进行性能分析。要求界面显示初始状态，目标状态和中间搜索步骤。

（4）要求显示搜索过程，画出搜索过程生成的搜索树，并在每个节点显示对应节点的评价值 f(n)。以红色标注出最终结果所选用的路线。

（5）撰写实验报告，提交源代码（进行注释）、实验报告、汇报 PPT。

2 试验方案整体概述

2.1 总体设计思路与总体框架

总体设计思路：

将问题进行前后端分离，并各自封装成类，其中前端界面部分负责与人进行交互，后端部分完成搜索与查找路径等内容

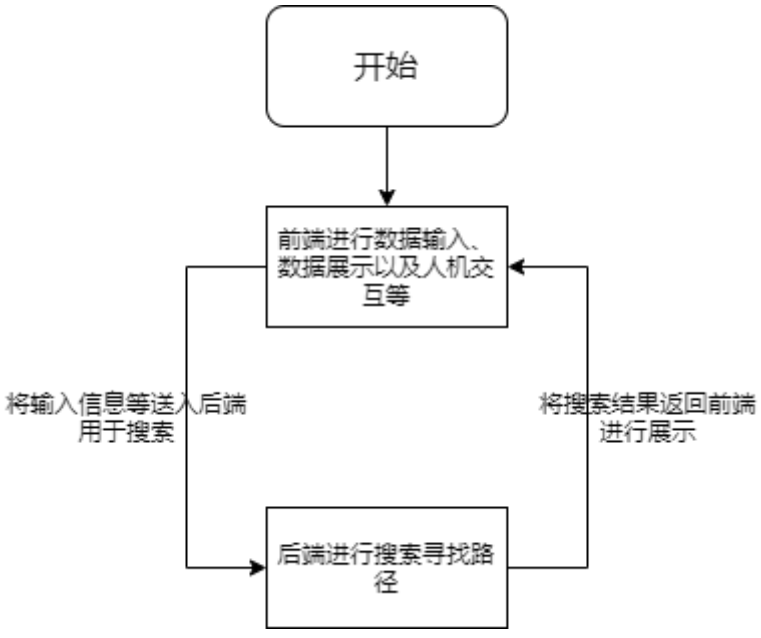


图 1 总体思路流程图

2.1.1 后端部分设计思路

后端部分使用 A*算法，并封装成 Search 类。设置优先队列 open 表、map 容器 close 表和 vector 向量 path。open 表用于存储已扩展出但是还没有访问的节点，使用优先队列进行自动排序；close 表存储已经访问过的节点，避免重复访问；path 用于保存路径。核心算法实现如下：

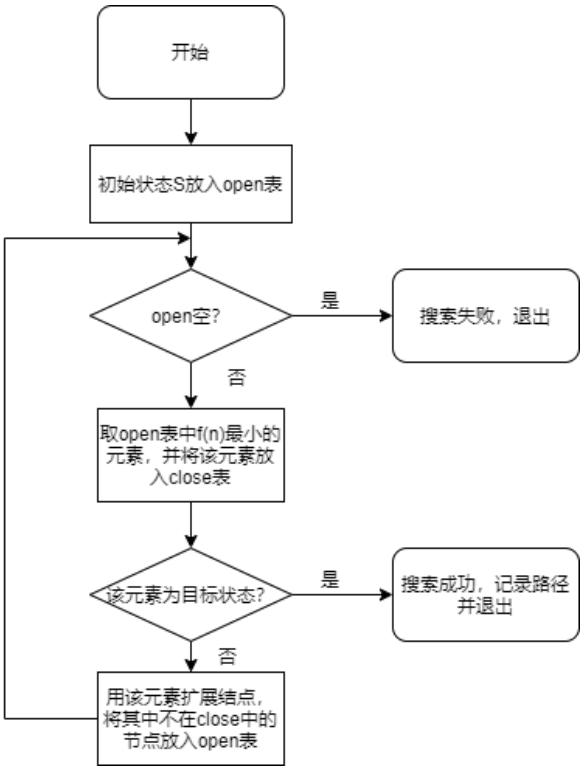


图 2 内核部分设计思路流程图

（1）先将起始状态放入 Open 表

（2）将 Open 表中 f(n)最小的元素取出放入 close 中，若已经为目标结点，则搜索完成，记录路径并退出

（3）使用上一步找到的元素扩展节点，如果新节点不在 Close 中，就将其放入 Open 表

（4）回到 2

找到目标状态后，从后向前回溯记录路径并存储在 path 中，记录完成后退出并将搜索信息送入前端。

2.1.2 前端设计思路

UI 界面初始状态如图 3，界面上从左到右、从上到下设置有：帮助工具栏，初始、当前、目标 3 个状态，搜索过程文本框，h(n)选择框，6 个功能按钮，时间及步数展示。

图 3 UI 界面初始状态

输入起始和目标状态，或直接使用“随机生成”按钮可以完成状态设置，选择 h(n)之后点击“开始搜索”按钮进行搜索，此时如果状态有错会给出报错提示，搜索完成后会有完成提示，同时会对搜索时间和步数信息进行展示。搜索结束后可以使用“单步执行”或“自动执行”（每 500ms 进行一步）按钮对搜索过程进行展示，展示结果为最短路径；或使用“搜索树”按钮展示搜索树，此时调用 graphviz 生成搜索树图片并展示，同时为了避免图片过大仅展示一部分。

在运行的不同阶段，会对相应的按钮进行禁用，避免造成功能冲突。

图 4、5 搜索完成信息展示

图 6 自动执行展示

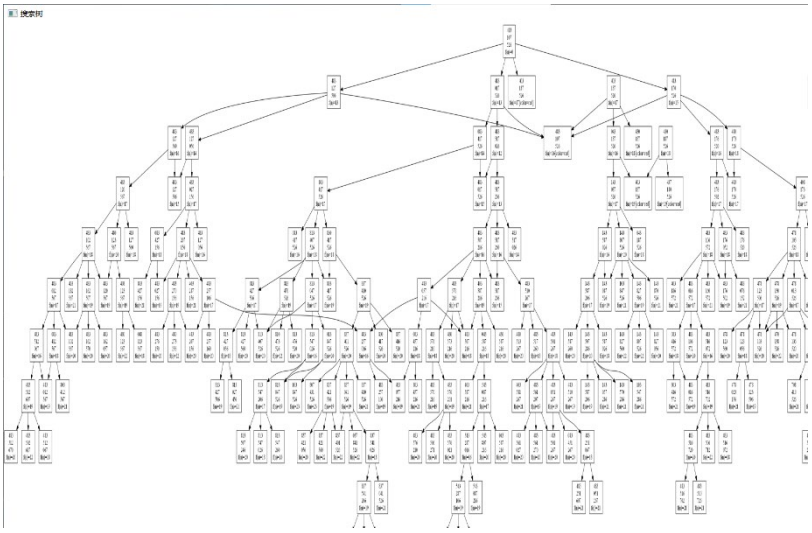
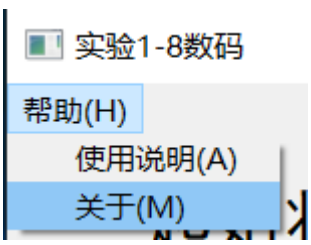


图 7 部分搜索树展示

在帮助菜单中可以查看“使用说明”和“关于”:



装 订 线

图 8、9 帮助菜单展示

输入错误或无解则会给出错误提示：

图 7 输入无解

图 8 输入错误

2.2 核心算法及基本原理

本实验的核心算法为 A*搜索算法。

A*算法是路径规划的常用算法，是一种高效率的有信息搜索算法，其高效性基于其特殊的评估函数上。A*算法的评估函数 f(n)定义为经过节点 n 的最小路径的预期花费，由两部分组成，f(n)=g(n)+h(n)，其中 g(n)表示从起点到达当前节点的实际代价，h(n)称为启发函数，表示预期的从该点到终点的代价。每次搜索时选择 f(n)最小的节点进行扩展，由此可以在期望最短的时间内找到最优解。

使用 $h^*(n)$表示从 n 节点到目标点的真实花费，根据 h(n)和 $h^*(n)$的大小关系可以对启发函数进行评估：

（1）如果 h(n) < $h^*(n)$到目标状态的实际距离，这种情况下，搜索的点数多，搜索范围大，效率低。但能得到最优解。

（2）如果 h(n) = $h^*(n)$，即距离估计 h(n)等于最短距离，那么搜索将严格沿着最短路径进行， 此时的搜索效率是最高的。

（3）如果 h(n) > $h^*(n)$，搜索的点数少，搜索范围小，效率高，但不能保证得到最优解。

其中若对于每一个节点 n 都有 0<=h(n)<=$h^*(n)$，则称 h(n)是可采纳的。

在本题程序中,我采用了两种不同的估价函数,并比较其运行效果。

（1）$h_0(n)$：该启发函数定义为当前状态每一个格子与其目标位置之间的曼哈顿距离之和。由于每个格子到达其目标位置至少需要的代价为其曼哈顿距离，因此满足 $h_0(n)$<=$h^*(n)$，该启发函数可采纳。

（1）$h_1(n)$：该启发函数定义为当前状态与目标状态相比，位置不正确的格子的数量。由于位置不正确的格子至少经过一次移动才能到达目标位置，因此满足 $h_1(n)$<=$h^*(n)$，该启发函数可

采纳。

2.3 模块设计

2.3.1 后端部分模块设计

后端部分将搜索信息封装为两个类，其中 State 类用来描述搜索过程的某一个状态的信息，Search 类用来描述一个搜索的过程极其相关信息。（详情见注释）

【State 类定义】:

```
class State
{
private:
    string st;//存储该状态的字符串，本实验中使用长度为 9 的字符串存储状态信息
    string pre;//存储该节点的前驱的字符串
    int f, g, h;//分别对应评估函数的 f(n)，g(n)，h(n)

public:
    State() {}
    State(string st, string pre = "", int g = 0, int h = 0);//构造函数，利用默认参数实现 1~4 参数
的构造
    void set(string st, string pre = "", int g = 0, int h = 0);//设置(更新)状态函数
    bool operator<(const State temp) const;//重载<号，根据 f 值比较大小，用于优先队列排序
    bool operator==(const State temp) const;//重载==号
    // friend void Search::GetSolution();

    //声明友元类
    friend class Search;
    friend class MainWindow;
};
```

【Search 类定义】:

```
class Search
{
private:
    static const int STATE_SIZE = 9;//状态大小
    static const int MOV_NUM = 4;//移动情况的数量，因为只有上下左右 4 种移动方式，因此该值为 4

    bool error_state;//用于描述输入状态是否有错
    bool finded;//用于判断是否能找到解，或是否已经找到解
    string origin_state, end_state;//起始字符串和目标字符串
    int run_time;//用于记录搜索时间

    priority_queue<State> open;//优先队列类型的 open 表
    map<string,string> close;//map 类型的 close 表，其中键为该状态的字符串，值为其前驱状态的字符串
    vector<State> all_path;//用于记录所有搜索中所有进入 close 的节点
    vector<string> path;//用于记录最佳路径
    int h_func;//用于记录选择哪个启发函数
    /*  0 1 2
```

```
        3 4 5
        6 7 8 */
    //mov 数组表示每一个位置的可能的移动位置，其中-1 表示不可达
    int mov[STATE_SIZE][MOV_NUM] = {
        {-1, 1, -1, 3},
        {0, 2, -1, 4},
        {1, -1, -1, 5},
        {-1, 4, 0, 6},
        {3, 5, 1, 7},
        {4, -1, 2, 8},
        {-1, 7, 3, -1},
        {6, 8, 4, -1},
        {7, -1, 5, -1}};

    static const int GET_H_NUM = 2;//记录启发函数总数量，本题共用到 2 种启发函数
    int (*get_h)(string st,string end_state);//启发函数的函数指针，用于简化操作
public:
    Search(){}
    Search(string origin_state, string end_state = "123456780", int h_func = 0);//构造函数，利用
默认参数实现 1~3 参数的构造
    void set(string origin_state, string end_state = "123456780", int h_func = 0);//设置(更
新)Search 的函数，与上面构造函数的不同点在于需要清空(clear)open、clpse 等容器
    void GetPath();//寻找并记录最佳路径
    bool CheckState();//判断输入是否合法，在前端中有类似实现，此处用于单独调试后端时使用
    void GetSolution();//搜索答案

    //声明友元类和友元函数(启发函数)
    friend class MainWindow;
    friend int get_h_0(string st,string end_state);//启发函数 h_0(n)
    friend int get_h_1(string st,string end_state);//启发函数 h_1(n)
};
```

2.3.2 前端部分模块设计

前端部分使用 C++ Qt 进行设计，其中 UI 界面部分使用 Qt creator 进行绘制，后端使用 Qt 中的多种类进行编写，并使用信号—槽机制进行信号的传输与响应。其中画搜索树部分使用 graphviz 生成搜索树图片并展示。

前端最主要的 MainWindow 类定义如下：（详情见注释）

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

public slots:
    //帮助菜单的“关于”选项的槽函数
    void MsgAbout();

    //帮助菜单的“使用说明”选项的槽函数
```

```
    void MsgManual();

private:
    Ui::MainWindow *ui;

    static const int STATE_SIZE=9;//状态大小为 9
    static const int SLEEP_TM=500;//默认自动执行间隔为 500ms

    Search my_search;//实例化查找对象
    vector<State> all_path;
    vector<string> path;//存储路径
    QLineEdit *origin_state[STATE_SIZE],*now_state[STATE_SIZE],*end_state[STATE_SIZE];//分别与 ui
的起始、当前、结束状态相对应
    int h_func;//记录所使用的启发函数
    int path_cnt;//路径计数器，用于展示路径
    TreeWindow *pTreeWindow;
    string origin_state_str,end_state_str;

    //初始化 ui 对象的参数
    void InitMembers();

    //初始化信号槽连接情况
    void InitConnections();

    //设置一个状态中所有的 QLineEdit 的可读状态
    void SetStateReadonly(QLineEdit *state[STATE_SIZE],bool flag=true);

    //检查某一状态的输入是否合法
    bool CheckInput(string input);

    //将一个状态的所有输入转为一个 QString 返回
    QString GetInput(QLineEdit *input[STATE_SIZE]);

    //将一个状态写入一个字符串
    void WriteState(QString str,QLineEdit *state[STATE_SIZE]);

    //获得一个随机状态
    string GetRandState();

    //清空一个状态
    void ClearState(QLineEdit *state[STATE_SIZE]);

    //自己封装的 Sleep 函数，因为 Qt 中 Sleep 函数不能正常工作
    void MySleep(int slp_tm);

    //判断是否有解
    bool HaveSolu(string ,string);

    //获得搜索树
    void GetTree();

private slots://ui 中按钮等的槽函数
    //“随机生成”按钮的槽函数
    void on_btn_randstr_clicked();

    //“开始搜索”按钮的槽函数
```

```
    void on_btn_start_clicked();

    //“单步执行”按钮的槽函数
    void on_btn_step_run_clicked();

    //“自动执行”按钮的槽函数
    void on_btn_auto_run_clicked();

    //“初始化”按钮的槽函数
    void on_btn_clear_clicked();

    //“搜索树”按钮的槽函数
    void on_btn_tree_clicked();

    //h(n)_0 的 checkbox 的槽函数
    void on_cbox_h0_clicked();

    //h(n)_1 的 checkbox 的槽函数
    void on_cbox_h1_clicked();

    //“输出路径”文本框的槽函数
    void autoScroll();
    };
```

2.4 其他创新内容及优化算法

（1）open 表使用优先队列，close 表使用 map，相较于使用 vector 的结构，大大提高了运行速度。

（2）使用逆序对奇偶性的特点可以直接判断当前输入是否有解，可以在搜索进行之前先进行判断，避免在无解的输入上花费太大的搜索时间和空间。

（3）使用函数指针优化代码，避免在选择启发函数时使用过多条件判断语句造成代码可读性降低。

（4）在综合考虑下使用字符串存储搜索的状态，与使用一个长整数相比，可以减少操作难度提升可读性；与使用一个整型数组存储相比，可以降低空间开销。

3 实验过程

3.1 环境说明

3.1.1 操作系统

Windows 10

3.1.2 开发语言

C++, Qt, dot(用于 graphviz 画图)

3.1.2 开发环境

开发环境: Qt Creater 5.9.0, VS 2019

编译器: Desktop Qt 5.9.0 MinGW 32bit

3.2 源代码文件清单及主要函数清单

3.2.1 头文件

(1) kernel.h

用于定义后端搜索所用的类(State, Search), 声明后端所用的头文件及其他所用函数。

(2) mainwindow.h

用于定义前端 UI 界面所用的类(MainWindow), 声明前端所用的头文件, 其中需要包含 kernel.h, 用于完成前后端交互。

(3) treewindow.h

用于定义画树子窗体的类(TreeWindow), 声明其所用的头文件, 用于展示画树的图片。

3.2.2 源文件

(1) kernel.cpp

后端处理用源文件，包含后端两个类中函数的实现及用到其他函数的实现，是本题的算法核心。

(2) mainwindow.cpp

前端主界面源文件，包含前端主窗体类 MainWindow 中函数的实现。

(3) treewindow.cpp

画树子窗体源文件，用于实现画树子窗体类 TreeWindow 中的函数。

(4) main.cpp

主函数源文件，用于展示主窗体。

3.2.3 其他文件

mainwindow.ui：用 Qt creator 绘制的前端主窗体 UI，包含帮助工具栏，初始、当前、目标 3 个状态，搜索过程文本框，h(n)选择框，6 个功能按钮，时间及步数展示等部分。

treewindow.ui：用 Qt creator 绘制的画树子窗体 UI，包含一个用于画树的 label。

tree.dot：由程序生成的树状图描述脚本，用于 graphviz 识别并画树。

tree.png：由 graphviz 画出的树状图，用于在子窗体中显示。

装 订 线

3.3 实验结果展示

3.3.1 UI 界面展示

具体内容可以通过运行 Eight_Figure_Puzzles.exe 得到，此处只展示一次运行结果（如图 10）

图 10 运行结果

3.3.2 效率分析

通过 8 种不同的随机生成的起始与目标状态下 2 种 h(n)运行时间的进行比较，得到如下表格：

实验编号	总步数	$f_0(n)$用时/ms	$f_1(n)$用时/ms
1	17	0	2
2	29	68	349
3	26	8	130
4	30	12	498
5	25	11	110
6	26	9	126
7	26	17	135
8	25	5	87

和		130	1437

表 1 运行时间对比图

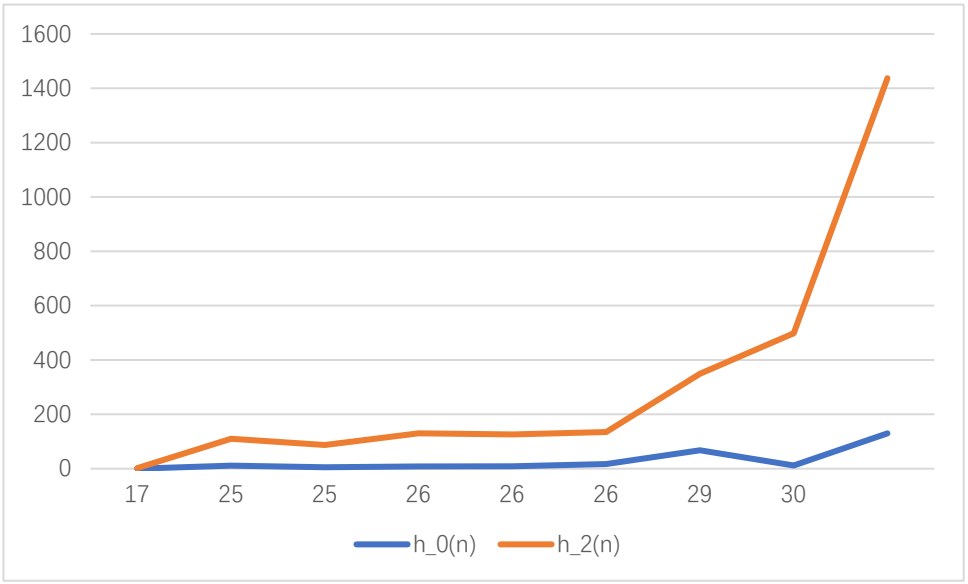


图 11 2 种启发函数运行时间随最优解步数的折线图

对同一启发函数进行纵向对比，发现基本上随着最优解步数的提升，搜索用时随之增大，但也存在着一定的波动性，因为即使最优解步数相同，不同的起始与目标状态也会导致不同的搜索时间。

对 2 种不同启发函数进行横向对比，发现在相同步数下，$h_0(n)$的搜索时间明显比 $h_1(n)$少很多，搜索快很多，分析该题使用曼哈顿距离作为启发函数，可以得到更好的搜索效果。

4 总结

4.1 存在问题及解决方案

（1）最初进行实验时后端搜索使用 vector 存储 open 表和 close 表，且每次从 open 表中选元素都进行一次 sort 排序，但是在进行测试时经常造成程序卡死，即使步数较少的的搜索也要进行较长时间。经过分析，该方法复杂度比较低，因此经过思考改用优先队列存储 open 表，使用 map 存储 close 表，从而实现快速搜索。

（2）最初进行实验时没有先使用逆序对奇偶性的方法直接判断输入是否有解，结果存在较多的无解情况，且每次出现无解都需要进行大量搜索，花费极大；因此经过资料搜索与分析，采用逆序对奇偶性的方法在搜索前进行判断，降低了无必要花费。

（3）最初进行实验时启发函数也封装在类中，但是在使用过程中造成了代码复杂度的提升，且根据实际需求分析，启发函数应该由用户提供，因此将启发函数移出 Search 类，使代码更加简洁且符合实际需求。

4.2 心得体会

（1）本次实验使用 A*搜索解决了 8 数码问题，在算法的实际使用中大大增强了对该算法的认识与理解，同时也对不同启发函数的效率差距有了一定的认识。

（2）实验中学习使用基于 Qt 的 UI 设计，在开发带有用户界面的应用程序上有了新的认识，编程能力得到提升

（3）实验中学习使用 STL 的各种工具进行程序开发，大大提高了开发效率和代码可读性，程序运行速度也有所提升。

（4）实验中学习使用 graphviz 绘制树状图，同时配套学习 dot 脚本描述语言，对于树状图的生成原理有了更多认识，同时也能够通过程序生成一些常用图片类型。

4.3 后续改进方向

（1）可以在 A*算法的基础上加上合理的剪枝，使搜索效率进一步提高。

（2）可以改变程序结构，将其改变为一款拼图游戏，且是带有 AI 提示的拼图游戏。

（3）可以改变程序结构，将 8 数码扩展为任意行列的“数码”类程序。

（4）由于时间限制，本题在实现过程中对 dot 脚本以及 graphviz 的学习了解有限，造成最后程序的画图部分存在一点小 bug，应该在后续学习中对其进行完善处理。

4.4 实验总结

通过本次实验，我在实践中对 A*搜索算法的原理与实现有了更深刻的认识，并通过不同启发函数的效率对比认识到选择合理的启发函数的重要性。同时通过实验中对 Qt、C++ STL、graphviz&dot 的学习与研究，我学到了许多新知识，并由此能够用更快的速度开发具有前端界面的实用程序。

5 参考文献

[1] Stuart J. Russell, Peter Norvig. 世界计算机教材精选 人工智能 一种现代的方法 第 3 版. 北京：清华大学出版社, 2018.07.

6 成员分工与自评

6.1 成员分工

成员：1953080 田宇(组长)

分工：由组长田宇同学实现本次实验的全部工作（包括程序设计、前后端代码编写实现、报告撰写、汇报 PPT 制作等）

6.2 自评

本次实验总体完成情况良好，能够完成实验要求实现的全部内容，但是由于没有合理安排好实验时间，最后时间有点紧张，导致画树部分仍有一点小 bug 未解决。

同时由于本次实验组内仅有我一个成员，缺少同学一起讨论，导致在实验过程中走了一些弯路，好在最后及时通过查阅资料以及讨论解决。

总体来看，在独立完成本次实验的过程中对于实验的整体有了全方面的认识，同时对于多种能力都有所提升，缺点是缺乏团队合作。