Routing_and_Placement

FM算法以及Astar算法实现读取文件进行处理

项目说明文档

划分问题

题目介绍与题目要求

考虑二划分问题，对给定的节点集合进行二划分，划分目标是平衡最小割。可以使用FM算法来实现。提供一个描述节点连接关系的文件prob1.txt

文件示例：

```
19
3 5 7 19
4 5 1
10 11
3 6
…
```

其中第一行表示节点数，比如19。后面跟随19行，表示从1-19个节点分别连接的节点。

迷宫算法布线

题目介绍与题目要求

本题需要实现的是迷宫布线算法。可以采用原始迷宫算法，也可以实现改进的A*迷宫算法。迷宫的全重假设为1。布线后的占据的迷宫位置要成为新的障碍。

本题给定prob4.txt文件，描述的是一个矩阵，每一位表示该坐标上的路径状态：0表示无障碍，1表示障碍，2表示有要连接的引脚。本题需要用迷宫布线算法将引脚都连接上，并且绕过障碍物。

文件示例：

```
0 0 0 0 0 0 0 0 0 0
1 0 0 1 0 0 0 0 1 0
0 0 2 0 0 0 0 0 0 1
0 0 0 2 0 2 0 0 0 1
0 0 0 2 0 0 0 0 0 0
0 0 2 0 0 2 0 0 2 0
```

项目成员

成员名称	学号
郑志宇	20307130176
邱峻蓬	20307130028
任钰浩	20307130243
沈笑涵	20307130063
周翔	20307130188

项目说明

功能说明

我们的项目完成了布局布线算法中的两个部分

- 迷宫布局布线算法
- `FM`算法实现的划分

创新点

1. 使用工厂模式实现主要的功能，各部分抽象程度高，且可扩展性很强
2. `Astar`算法使用了优先队列实现，使用简单的连线策略实现了多数情况下相对优的连线结果。

3. `Astar`生成多点的连线的算法我们采取了以下策略：
    - 增加尝试创建布线的边权重
    - 将已布线边的权重降低为0
4. `FM`算法实现了通过桶型数据结构查找最大增益和相应的更新，在大数据量的背景下对于算法时间复杂度的优化显得极为必要，同时使用多起始点方法改进`FM`算法。

项目使用方法

环境要求

- `visual studio 2019`及以上版本能正常打开项目中的所有文件
- 注意`src`编码格式为`Unicode(UTF-8)`
- 生成一个`Routing.exe` 的可执行文件能够在`windows` 下运行。如果想要生成`Linux`下的`Makefile`，可以修改一下`Makefile`文件。

使用方式

根据自己的环境编写替换`Makefile`，目前项目中的`Makefile`仅仅适用于`window`。

`Windows`：在`file` 文件夹中写好测试文件`file.txt` 后，在项目文件夹下运行以下命令：

```
.\Routing.exe FM file\file.txt
.\Routing.exe Astar file\file.txt
```

在`file`中有一个计算`Astar`布线长度的程序，原理是利用两个存储矩阵的文件中的矩阵做差然后计算布线线长。在`file`文件夹中执行如下命令：

```
.\Calc.exe file1 file2
```

cmd中运行结果示意：

```
# partition迭代次数
FM ..
iteration 0: Cut Size = ...
iteration 1: Cut Size = ...
iteration 2: Cut Size = ...
```

```
FM algorithm .. done. Cut Size = ...
Partition Result:
# Astar输出一个连好线的图
0 0 0 0 0 0 0 0 0 0
1 0 0 1 0 0 0 0 1 0
0 0 2 3 0 0 0 0 0 1
0 0 0 2 3 2 0 0 0 1
0 0 0 2 3 3 0 0 0 0
0 0 2 3 3 2 3 3 2 0
```

文件格式声明

FM算法支持的文件格式

```
19
3 5 7 19
4 5 1
10 11
3 6
...
```

其中第一行表示节点数，比如19。后面跟随19行，表示从1-19个节点分别连接的节点。

Astar算法支持的文件格式

```
0 0 0 0 0 0 0 0 0 0
1 0 0 1 0 0 0 0 1 0
0 0 2 0 0 0 0 0 0 1
0 0 0 2 0 2 0 0 0 1
0 0 0 2 0 0 0 0 0 0
0 0 2 0 0 2 0 0 2 0
```

上面的文件描述的是一个矩阵，每一位表示该坐标上的路径状态：0表示无障碍，1表示障碍，2表示有要连接的引脚。

项目的结构

```
|# 编译项目的Makefile,Windows下使用MinGW32编译
|    Makefile
|# 项目的说明文档
|    README.md
|    README.pdf
|# 项目的可执行文件（Windows平台下）
|    Routing.exe
|# 项目的测试文件以及输出
├——file
|# 项目的README中的图片
├——picture
|# 项目的参考文件
├——projectfile
|# 项目的源文件
└——src
```

项目技术细节

项目顶层文件

此部分由郑志宇同学搭建，接口与功能与小组成员共同商议确定。最终实现了项目的并行推进与项目成员对函数的独立维护。方便项目更容易进行其他功能的扩展。最终有郑志宇同学完成了项目的编译，输出一个`Routing.exe`文件。

| ├── `main.cpp`

| ├── `routing.cpp`

| ├── `routing.h`

顶层的流程

- `parser`类读取文件并解析，得到对应的对象
- `Routing`类根据解析文件的结果实现不同的功能
    - `Astar`连线
    - `FM`划分
- `Routing`类输出一个文件到目标文件夹`file`

Part 1 FM算法实现模块的划分

这部分由邱峻蓬同学和任钰浩同学完成，邱峻蓬同学主要负责FM算法的代码实现和改进。

算法概述

介绍

FM算法的移动过程和KL算法很类似。目标都是讲割代价最小化。但是FM算法要i计算每个独立节点移动的增益，而不是对交换所产生的增益。和KL算法一样，FM算法再每个轮次中选择最好的一次移动。在FM算法的每轮中，一旦一个节点被移动，在本轮中它就被锁定不能再次移动。

术语

以下是和FM算法有关的定义

节点 c 的增益 $\Delta g(c)$ 是 c 移动后割集割边数量所产生的变化。增益 $\Delta g(c)$ 越高，移动节点 c 到其它划分的的优先级越高。形式上，节点增益被定义为：

$$\Sigma_{i=1}^{m}\Delta g_i$$

其中 $E(c)$ 是 c 与不是其所在划分的节点之间边的数量，$I(c)$ 是 c 与其自身所在划分的其它节点之间边的数量。

每轮中的最大增益 G_m 是指 m 次移动所产生的最大节点增益 Δg 的总和所决定的。

$$\Sigma_{i=1}^{m}\Delta g_i$$

类似于在KL算法中，每轮中所有的移动决定了 G_m 和移动顺序 $\{c_1 \dots c_m\}$ 在每轮结束后，即 G_m 和相应的 m 次移动被决定后，再更新移动节点的位置。

流程伪代码

我们为算法流程写了一个简单的伪代码供参考：

```
INPUT: 图 G(V,E)
OUTPUT: 划分后的图 G(V,E)
(A, B) = PARTITION(G) //划分初始化
G_m = \infin
while (G_m > 0) {
    i = 1
```

```
    order = \emptyset
    foreach (c \in V) {
        \Delta g[i][c] = E(c) - I(c) //计算所有节点的初始增益
        status[c] = FREE //将所有节点设为自由
    }
    while (!IS_FIXED(V)) {
        cell = MAX_GAIN(\Delta g[i]) //找到增益最大的节点
        ADD(order, (cell, \Delta g[i])) //记录节点移动序列
        cirtical_net = CRITICAL_NETS(cell) //连接该节点的线网
        if (cell \in A) {
            TRY_MOVE(cell, A, B)
        } else {
            TRY_MOVE(cell, B, A)
        }
        status[cell] = FIXED
        foreach (c \in critical_net, c \neq cell) {
            if (status[c] == FREE) {
                UPDATE_GAIN(\Delta g[i][c]) //更新与尝试移动节点相连的所有节点的增益
            }
        }
        i = i + 1
    }
    (G_m, m) = BEST_MOVES(order) //找到 G_m 最大化时的移动序列
    if (G_m > 0) {
        CONFIRM_MOVES(order, m) //按该移动序列进行实际移动操作
    }
}
```

算法具体实现

```
|  ├── data_structure.h
|  ├── FM_algorithm.h
|  ├── FM_algorithm.cpp
```

节点类的定义

```
class NODE {
public:
    int nodeIndex;                          //节点索引
    NODE_PART Node_Partition;               //节点划分归属
    LOCK_STATE lockedstate;                 //节点锁定情况
    int nodeGain;                           //节点增益
    std::vector<int> ConnectedNode;         //节点连接的其他节点
}
```

对于NODE类，重载了三个构造函数以便灵活赋值。

节点的指针数组类

```
class POINTER_ARRAY {
public:
    std::vector<NODE*> data_array;
        //节点的指针数组

    POINTER_ARRAY();
    POINTER_ARRAY(std::vector<std::vector<int>>& modules);
        //根据parser解析得到的modules进行指针数组的构造
    void copy(POINTER_ARRAY pa);
        //指针数组的复制
    void reset(POINTER_ARRAY pa);
         //指针数组的拷贝赋值
    void recover();
         //指针数组的解锁
    void init_half();
         //指针数组的二分初始化划分
    void init_even();
         //指针数组的奇偶初始化划分
    void init_rand();
         //指针数组的随机初始化划分
    void updateGain();
        //指针数组更新增益
    int updateGain(int i);
        //指针数组更新指定节点增益
    int cutSize();
        //指针数组按照当前划分被切割的边数
};
```

桶结构类

```
class BucketNode {
public:
    int nodeIndex;
    BucketNode* next;
    BucketNode* prev;
    BucketNode(int i,BucketNode* n,BucketNode* p) {
        nodeIndex = i;
        next = n;
        prev = p;
    }
};
class Bucket {
public:
    std::map<int, BucketNode*, std::greater<int>> bucketAtoB;
    std::map<int, BucketNode*, std::greater<int>> bucketBtoA;

    Bucket();
    void load(POINTER_ARRAY& parr);
    int maxGain(NODE_PART partition,POINTER_ARRAY& parr);
    void updateLocal(NODE_PART partition, int i, int updateGain,
int prevGain);
};
```

桶结构中为了减少数据冗余，新定义了桶节点以实现双向链表结构。采用`map`容器来构造桶形结构，使得可以通过增益值作为桶的入口来访问节点，由于`map`容器本身的自排序性，可轻松找到增益最大的入口。
由于实现的目标是二划分，因此需要对两个方向分别维持一个桶形数据结构。

FM类

```
class FM {
public:
    int mincutsize;                                    // FM算法执行后当前
的最小切割数
    void one_swap(
        Bucket& bu,
        POINTER_ARRAY& pointer_array_local,
        POINTER_ARRAY& pointer_array_global,
        int currentBest
    );                                                 // FM算法的一次迭代
```

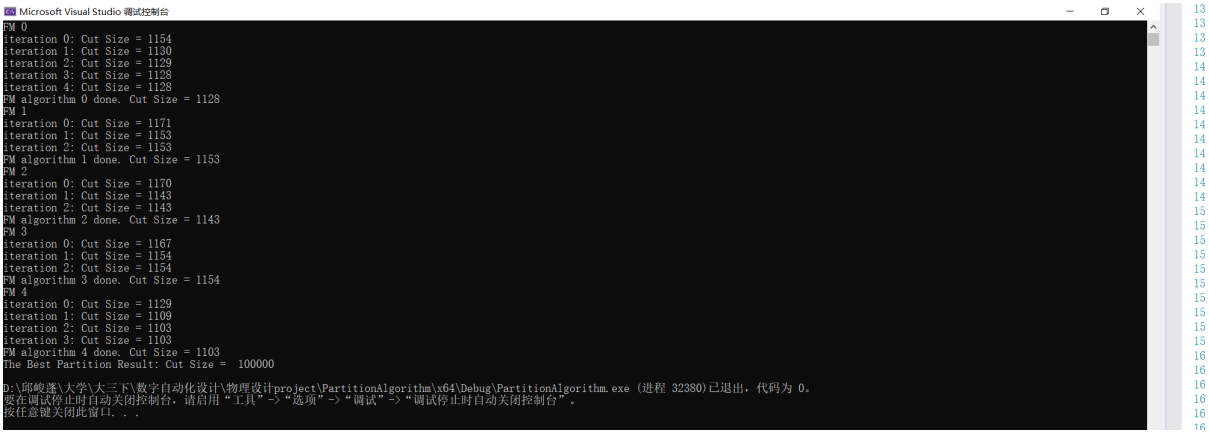
```
    std::vector<std::vector<int>> FM_Algorithm(
        std::vector<std::vector<int>> modules
    );                                                  //FM算法
    std::vector<std::vector<int>> FM_Algorithm_Pertubation(
        std::vector<std::vector<int>> modules
    );                                                  //加入初始状态微扰后改进的FM算法
};
```

one_swap方法即通过桶形数据结构和节点指针数组执行如下过程：从partition A中取得最大增益且未锁定的节点移动至partition B中，更新A节点相连所有节点的增益；从partition B中取得最大增益且未锁定的节点移动至partition A中，更新B节点相连所有节点的增益。直到所有节点都被锁定后，完成一次迭代。
FM算法中宏定义了迭代的上限MaxIteration，但为了加快FM算法的运算过程，假设连续两次迭代都未能成功减小切割数后即结束算法。

由于FM算法基于邻域搜索，因此初始解的情况会非常明显的影响FM算法的最优结果。例如，当FM算法的初始解按照前1000个节点和后1000个节点划分时，FM算法所得最小切割边数为1144。

当FM算法的初始解按照奇偶节点划分时，FM算法所得最小切割边数为1123。

因此，对于FM算法做出多起始点方法的改进，通过随机数产生随机初始平衡划分，多次运行寻找最优结果作为最终的划分结果。因此FM_Algorithm_Pertubation的执行结果具体到每一次都不尽相同，但从数学统计意义上可以给出最优结果。本实现中重复运行5次基础FM算法。一次典型的运行结果如下。



Part 2 `Astar` 算法实现模块的连线

这部分由郑志宇同学和周翔同学完成。

点到点的 `A*` 算法

```
| ├── Astar.h
| ├── Astar.cpp
```

`A*` 算法图的定义

边和节点定义

```cpp
struct Edge
{               // 表示边
    int sid_;  // 边的起始节点
    int eid_;  // 边的结束节点
    double w_; // 边的权重
    Edge() = default;
    Edge(int s, int e, double w)
        : sid_(s), eid_(e), w_(w) {}
};
struct Vertex
{
    int id_;
    double dist_; // 算法实现中，记录第一个节点到这个节点的距离
    double f_;    // f(i)=g(i)+h(i)
    int x_, y_;   // 顶点在地图中的坐标（x, y）
    Vertex() = default;
    Vertex(int id, int x, int y)
```

```
        : id_(id), x_(x), y_(y),
dist_(std::numeric_limits<double>::max()),
f_(std::numeric_limits<double>::max()) {}
};
```

AStarGraph类

```
class AStarGraph
{
public:
    AStarGraph(){};
    void CreateGraph(std::vector<std::vector<int>> &_Maze); //
创建图
    void addEdge(int s, int e, double w);                   //
添加边
    void addVertex(int id, int x, int y);                   //
添加节点
    std::vector<int> AStar(int s, int e);                   //
寻路
    void Initial();                                         //
初始化
    std::vector<int> &getConnectionPoint()                  //
返回需要连接的点的关系
    {
        return ConnectionPoint;
    };
private:
    std::vector<std::vector<Edge>> adj_; // 邻接表
    int v_count_;                        // 顶点数
    std::vector<Vertex> vertexes;        // 记录所有顶点，主要记录坐
标
    double hManhattan(int x1, int y1, int x2, int y2)
    {
        return std::abs(x1 - x2) + std::abs(y1 - y2);
    }
    std::vector<int> ConnectionPoint;
};
```

`A*`算法是对`Dijkstra`算法的优化

在`Dijkstra`算法中，需要创建了优先队列，从优先队列中取出节点，再从这个节点扩散到其他节点（类似广度优先搜索），优先队列的优先依据是根据起始点到队列中节点最近的一个。即队列中的节点总是离起始点最近的先出队，这样很容易在一开始就跑偏。

当走到某个节点时，已知的是起始点到该节点的距离$g(i)$，`Dijkstra`算法判断依据就只有这一个，那么再增加一个当前节点到终点的距离，结合$g(i)$可以实现防止过度跑偏。

在电路图中，虽然不知道中间某个点距离终点的最短路，但是我们可以将地图置于坐标轴中，通过计算节点之间的曼哈顿距离来代替欧几里得距离，曼哈顿距离就是计算两点之间横纵坐标的距离之和，只涉及到加减法和符号转换。这里得到的距离记为$h(i)$，作为启发函数。

而$g(i)+h(i)$就可以当做最终的估价函数，优先队列中，估价函数值最低的优先出列。

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

连接多个点的策略

`routing.h`

`routing.cpp`

`connect`函数

```
void connect(std::vector<std::vector<int>>& Maze, int source,
int target, std::vector<int> &parent)
{
    if (source == target) return;
    int x = target % Maze[0].size();
    int y = target / Maze[0].size();
    if (Maze[y].at(x) != 2) {
        Maze[y].at(x) = 3;
    }
    connect(Maze, source, parent[target], &parent);
}
```

`AStarGraph.AStar(int s, int e)`会传出全部的搜索结果`parent`，是一个大小为`n*n`的向量。如果直接通过`for`循环更改`Maze`的所有的参数，可能会出现两个引脚之间存在多条连线的情况。这里使用递归，由目标节点向前推进，直到回到开始节点为止。期间可能会经过其他引脚，此时不改变`Maze`中该引脚位置的参数。否则将`Maze`上的参数改为`3`，表示连线经过该点。

`performAstar`函数

```
void Routing::performAstar()
{
    Astar.CreateGraph(Maze);
    std::vector<int> ConnectionPoint =
Astar.getConnectionPoint();

    for (int i = 0; i < ConnectionPoint.size() - 1; i++)
    {
        Astar.Initial();
        std::vector<int> parent =
Astar.AStar(ConnectionPoint[i], ConnectionPoint[i + 1]);
        connect(Maze, ConnectionPoint[i], ConnectionPoint[i +
1], parent);
    }
    Astar.drawGrid(Maze);
}

```

由于每次利用`Astar`算法对两个引脚进行连线后，都会改变图`Maze`上节点的参数，因此在进行下一次布线之前，首先需要使用`Astar.Initial()`函数进行初始化，将节点的`dist_`与`f_`恢复到最大值，之后再进行后续节点的`connect`操作。

连接的策略

采用的策略

我们在连接点的时候采用的策略会自动优化一部分线长。但是由于我们实质上并没有尝试引入斯坦纳点去解决最小生成子树的问题，所以连出的线中仍有可以优化的部分。实际上我们可以优化点之间连接的顺序来实现这个工作。由于时间关系，我们将这个工作放在以后。

实际上我们在连接点的时候，我们使用了这样的策略：

1. 需要连线的点按照从上到下，从左到右的顺序进行记录。这种记录方式实际上会导致一些连线的浪费问题。我们考虑使用优先队列来实现这样一个连线的问题或许会

更好一些。但程序中并没有这样做。

2. 已经连过的走线我们将它的代价视为`0`，这样节点在同样的曼哈顿距离下会优先选择连好的线。
3. 没有连过的走线我们将它的代价设为`1.1`。这样实际上会让节点之间在连线时在有可能的情况下不会因为路径进入队列的顺序问题而产生额外的走线。

结果的对比

下面是我们优化过程中得到的中间文件的对比情况，供参考：

使用`Calcdifference.cpp`编译了一个`Calc.exe`文件来实现这种代价的对比。

使用方式：

```
Calc.exe file1 file2
```

实现两个文件中矩阵作差并求出线长的差距。

```
C:\Users\18064\projects\FMandAstar\file>Calc.exe nomodified.txt modified1.txt
Matrix Size: 40*40
Sum of differences: 22

C:\Users\18064\projects\FMandAstar\file>Calc.exe nomodified.txt modified2.txt
Matrix Size: 40*40
Sum of differences: 51
```

对比了最初没有做优化以及加入了策略2和策略3之后得到的线长的优化长度。可以看到优化还是较为明显的。以下是总线长的计算。

```
C:\Users\18064\projects\FMandAstar\file>Calc.exe nomodified.txt prob4.txt
Matrix Size: 40*40
Sum of differences: 219

C:\Users\18064\projects\FMandAstar\file>Calc.exe modified1.txt prob4.txt
Matrix Size: 40*40
Sum of differences: 197

C:\Users\18064\projects\FMandAstar\file>Calc.exe modified2.txt prob4.txt
Matrix Size: 40*40
Sum of differences: 168
```

Part 3 文件解析以及`FM`算法和`Astar`算法实现结果的显示

该部分由任钰浩和沈笑涵同学完成。

文件解析

该部分将记录数据信息的.txt文件读入并转化为`FM`算法和`Astar`算法需要的的数据结构
详见以下文件：

`| ├── parser.cpp`

`| ├── parser.h`

其中`Maze`中存`Astar`算法所需要的数据，`Modules`中存`FM`算法所需要的数据，`erroinfo`中存读取文件时发现的文件格式错误信息，通过相应的`getMaze``getModules``returninfo`等结构访问数据与信息

```
class parser
{
public:
    int type;
    parser(std::string, std::string);
    int parse();
    std::string returninfo();
    std::string getFileName();
    std::vector<std::vector<int>>& getMaze();
    std::vector<std::vector<int>>& getModules();
private:
    std::string filename;
    std::string erroinfo;
    std::vector<std::vector<int>> Maze;
    std::vector<std::vector<int>> Modules;
};
```

结果显示

该部分实现了将前两部分`FM`和`Astar`算法的结果汇总，并完成相关布线代价的计算，并将最终布线结果输出到txt文件中。其部分详见以下文件：

`| ├── routing.cpp`

`| ├── routing.h`

其中，`routing.h`中构造了一个`Routing`的class，基本结构如下：

```
class Routing
{
public:
    // 完成解析输出的类型
    int type;
    std::string filename;
    // 构造Routing类
    Routing(parser& p)
    {
        Maze = p.getMaze();
        modules = p.getModules();
        filename = p.getFileName();
        type = p.type;
    };
    // FM算法
    void performFM();
    // Astar算法
    void performAstar();
    // 输出文件
    void outputfile();
private:
    void connect(std::vector<std::vector<int>>& Maze,
    int source, int target, std::vector<int> &parent);
    // 矩阵
    std::vector<std::vector<int>> Maze;
    // 模块连接关系
    std::vector<std::vector<int>> modules;
    // 划分
    std::vector<std::vector<int>> partition;
    // Astar代理
    AStarGraph Astar;
    // FM算法实现代理
    FM fm;
};
```

其中，`modules`为通过`parser`对输入解析后得到的结果，`type`标志了该解析文件后的输出类型，
`Maze`和`fm`分别存储`Astar`和`FM`算法得到的结果。

`int cost_of_routing_Astar()`和`int cost_of_FM();`两个函数为计算`Astar`和`FM`算法得到的结果的代价。具体来说，`Astar`算法的代价为布线线长，`FM`算法的代价为切割边的值。
最后，通过`outputfile()`函数将结果写入到txt文件中。

项目测试

测试文件1 `prob1.txt`

输出结果 `outputprob1.txt`

```
partition size: 1000:
```

```
2  1 3 5 6 7 8 9 12 13 16 18 19 20 24 26 27 28 31 32 33 37 40 41 42
   49 54 55 56 58 59 60 67 69 70 72 73 74 76 78 80 81 83 87 88 89 92
   93 94 96 98 100 102 103 104 107 109 110 111 113 116 119 121 122
   126 128 130 132 135 139 143 146 148 149 151 152 154 156 158 160
   161 165 166 168 169 171 173 175 179 180 181 183 184 186 189 195
   196 197 201 203 205 206 207 211 213 214 215 216 217 220 221 222
   223 224 226 231 234 235 238 240 241 243 245 246 251 252 253 254
   256 260 266 267 270 271 272 274 279 282 283 284 285 286 289 290
   293 295 296 297 298 299 301 303 304 305 306 307 308 309 310 312
   319 320 323 324 325 327 328 330 332 334 335 336 337 342 343 344
   345 346 347 350 351 356 357 358 360 363 366 367 373 376 377 382
   384 387 388 393 395 397 402 404 405 406 409 410 414 415 416 418
   421 424 425 426 429 436 437 438 442 444 446 448 453 455 457 458
   459 461 464 466 471 472 473 474 476 480 484 487 488 490 494 495
   498 499 502 503 506 507 508 510 512 517 518 520 523 526 531 532
   533 535 537 538 540 542 544 545 548 549 550 551 553 554 555 557
   558 562 565 566 567 569 570 578 580 581 582 583 584 586 588 590
   592 594 595 597 598 599 600 602 605 606 608 609 610 611 612 614
   616 617 618 621 622 625 626 627 629 633 636 637 638 640 642 643
   646 648 650 651 653 654 655 656 657 658 659 661 662 663 664 666
   669 672 673 679 682 685 686 687 688 689 691 692 696 699 701 703
   706 707 711 712 713 715 717 718 719 720 722 723 724 727 728 730
   731 732 733 734 737 738 739 744 746 747 748 749 753 755 757 758
   761 763 768 769 772 773 775 776 777 778 780 781 787 788 790 791
   793 796 798 803 805 806 807 808 810 811 812 813 820 821 823 824
   825 828 829 830 831 832 833 835 840 841 842 844 846 850 853 856
   859 860 861 864 865 867 869 870 872 878 881 882 884 886 887 888
   890 892 894 896 902 903 904 906 907 910 914 916 919 920 924 925
   926 927 928 930 931 935 936 937 940 941 944 946 949 952 955 958
   960 962 964 965 967 970 971 975 976 978 984 986 990 996 998 999
   1000 1002 1003 1005 1008 1011 1012 1013 1014 1020 1021 1022 1023
   1024 1025 1026 1028 1032 1033 1034 1039 1040 1041 1043 1044 1047
   1049 1052 1054 1055 1060 1063 1064 1072 1075 1079 1081 1082 1083
   1086 1087 1088 1089 1090 1091 1093 1098 1105 1112 1116 1117 1118
   1125 1126 1129 1132 1136 1137 1138 1140 1142 1145 1146 1147 1148
   1150 1156 1157 1159 1160 1161 1167 1168 1169 1170 1171 1172 1173
   1176 1177 1178 1179 1180 1181 1182 1189 1190 1191 1192 1193 1194
   1195 1196 1198 1199 1202 1203 1204 1206 1207 1208 1209 1210 1211
   1216 1217 1218 1220 1221 1228 1231 1232 1234 1237 1239 1240 1241
   1243 1244 1245 1246 1247 1248 1250 1251 1252 1253 1256 1258 1259
   1263 1264 1265 1267 1268 1273 1274 1276 1277 1280 1281 1285 1287
   1288 1290 1293 1294 1297 1299 1300 1302 1304 1306 1308 1311 1313
   1314 1315 1316 1317 1319 1320 1322 1324 1325 1327 1329 1331 1332
```

```
1334 1337 1338 1339 1340 1341 1345 1352 1353 1354 1355 1356 1357
1358 1361 1362 1363 1365 1367 1370 1371 1373 1377 1380 1381 1382
1385 1386 1388 1390 1392 1393 1398 1399 1400 1402 1410 1411 1417
1418 1421 1424 1425 1427 1428 1429 1430 1431 1434 1436 1439 1440
1441 1442 1443 1444 1445 1446 1449 1450 1456 1457 1458 1460 1464
1466 1467 1469 1470 1471 1472 1473 1480 1482 1483 1484 1486 1488
1492 1494 1496 1497 1502 1503 1504 1505 1506 1507 1508 1510 1511
1518 1520 1521 1523 1524 1527 1528 1531 1534 1538 1541 1546 1548
1549 1554 1556 1557 1561 1563 1564 1566 1567 1569 1570 1573 1576
1579 1580 1583 1584 1585 1586 1588 1589 1591 1592 1595 1597 1598
1601 1602 1603 1605 1606 1607 1608 1609 1610 1611 1615 1622 1623
1625 1627 1629 1631 1632 1636 1640 1642 1644 1648 1649 1650 1651
1652 1653 1654 1655 1658 1659 1660 1661 1664 1665 1667 1670 1671
1680 1681 1682 1683 1684 1688 1689 1690 1692 1694 1696 1698 1699
1700 1701 1704 1705 1706 1707 1708 1710 1711 1714 1715 1720 1721
1724 1727 1728 1730 1732 1733 1737 1740 1743 1744 1747 1748 1750
1752 1753 1756 1760 1762 1768 1770 1772 1773 1775 1776 1778 1779
1781 1783 1784 1785 1786 1787 1788 1791 1793 1800 1803 1804 1805
1806 1807 1809 1812 1815 1819 1820 1821 1823 1824 1827 1829 1830
1834 1837 1838 1840 1841 1842 1845 1846 1848 1849 1851 1855 1860
1861 1863 1864 1865 1866 1867 1870 1871 1872 1873 1874 1887 1888
1889 1892 1894 1895 1896 1899 1900 1901 1902 1903 1904 1905 1906
1908 1909 1912 1913 1915 1918 1921 1924 1925 1927 1928 1929 1930
1937 1938 1939 1940 1942 1946 1948 1949 1952 1953 1954 1956 1959
1961 1963 1965 1966 1968 1969 1970 1972 1974 1977 1978 1979 1983
1984 1987 1988 1992 1993 1994 1995 1996 1997 1998 1999
partition size: 1000:
```

```
4  0 2 4 10 11 14 15 17 21 22 23 25 29 30 34 35 36 38 39 43 44 45 46
   47 48 50 51 52 53 57 61 62 63 64 65 66 68 71 75 77 79 82 84 85 86
   90 91 95 97 99 101 105 106 108 112 114 115 117 118 120 123 124
   125 127 129 131 133 134 136 137 138 140 141 142 144 145 147 150
   153 155 157 159 162 163 164 167 170 172 174 176 177 178 182 185
   187 188 190 191 192 193 194 198 199 200 202 204 208 209 210 212
   218 219 225 227 228 229 230 232 233 236 237 239 242 244 247 248
   249 250 255 257 258 259 261 262 263 264 265 268 269 273 275 276
   277 278 280 281 287 288 291 292 294 300 302 311 313 314 315 316
   317 318 321 322 326 329 331 333 338 339 340 341 348 349 352 353
   354 355 359 361 362 364 365 368 369 370 371 372 374 375 378 379
   380 381 383 385 386 389 390 391 392 394 396 398 399 400 401 403
   407 408 411 412 413 417 419 420 422 423 427 428 430 431 432 433
   434 435 439 440 441 443 445 447 449 450 451 452 454 456 460 462
   463 465 467 468 469 470 475 477 478 479 481 482 483 485 486 489
   491 492 493 496 497 500 501 504 505 509 511 513 514 515 516 519
   521 522 524 525 527 528 529 530 534 536 539 541 543 546 547 552
   556 559 560 561 563 564 568 571 572 573 574 575 576 577 579 585
   587 589 591 593 596 601 603 604 607 613 615 619 620 623 624 628
   630 631 632 634 635 639 641 644 645 647 649 652 660 665 667 668
   670 671 674 675 676 677 678 680 681 683 684 690 693 694 695 697
   698 700 702 704 705 708 709 710 714 716 721 725 726 729 735 736
   740 741 742 743 745 750 751 752 754 756 759 760 762 764 765 766
   767 770 771 774 779 782 783 784 785 786 789 792 794 795 797 799
   800 801 802 804 809 814 815 816 817 818 819 822 826 827 834 836
   837 838 839 843 845 847 848 849 851 852 854 855 857 858 862 863
   866 868 871 873 874 875 876 877 879 880 883 885 889 891 893 895
   897 898 899 900 901 905 908 909 911 912 913 915 917 918 921 922
   923 929 932 933 934 938 939 942 943 945 947 948 950 951 953 954
   956 957 959 961 963 966 968 969 972 973 974 977 979 980 981 982
   983 985 987 988 989 991 992 993 994 995 997 1001 1004 1006 1007
   1009 1010 1015 1016 1017 1018 1019 1027 1029 1030 1031 1035 1036
   1037 1038 1042 1045 1046 1048 1050 1051 1053 1056 1057 1058 1059
   1061 1062 1065 1066 1067 1068 1069 1070 1071 1073 1074 1076 1077
   1078 1080 1084 1085 1092 1094 1095 1096 1097 1099 1100 1101 1102
   1103 1104 1106 1107 1108 1109 1110 1111 1113 1114 1115 1119 1120
   1121 1122 1123 1124 1127 1128 1130 1131 1133 1134 1135 1139 1141
   1143 1144 1149 1151 1152 1153 1154 1155 1158 1162 1163 1164 1165
   1166 1174 1175 1183 1184 1185 1186 1187 1188 1197 1200 1201 1205
   1212 1213 1214 1215 1219 1222 1223 1224 1225 1226 1227 1229 1230
   1233 1235 1236 1238 1242 1249 1254 1255 1257 1260 1261 1262 1266
   1269 1270 1271 1272 1275 1278 1279 1282 1283 1284 1286 1289 1291
   1292 1295 1296 1298 1301 1303 1305 1307 1309 1310 1312 1318 1321
```

```
1323 1326 1328 1330 1333 1335 1336 1342 1343 1344 1346 1347 1348
1349 1350 1351 1359 1360 1364 1366 1368 1369 1372 1374 1375 1376
1378 1379 1383 1384 1387 1389 1391 1394 1395 1396 1397 1401 1403
1404 1405 1406 1407 1408 1409 1412 1413 1414 1415 1416 1419 1420
1422 1423 1426 1432 1433 1435 1437 1438 1447 1448 1451 1452 1453
1454 1455 1459 1461 1462 1463 1465 1468 1474 1475 1476 1477 1478
1479 1481 1485 1487 1489 1490 1491 1493 1495 1498 1499 1500 1501
1509 1512 1513 1514 1515 1516 1517 1519 1522 1525 1526 1529 1530
1532 1533 1535 1536 1537 1539 1540 1542 1543 1544 1545 1547 1550
1551 1552 1553 1555 1558 1559 1560 1562 1565 1568 1571 1572 1574
1575 1577 1578 1581 1582 1587 1590 1593 1594 1596 1599 1600 1604
1612 1613 1614 1616 1617 1618 1619 1620 1621 1624 1626 1628 1630
1633 1634 1635 1637 1638 1639 1641 1643 1645 1646 1647 1656 1657
1662 1663 1666 1668 1669 1672 1673 1674 1675 1676 1677 1678 1679
1685 1686 1687 1691 1693 1695 1697 1702 1703 1709 1712 1713 1716
1717 1718 1719 1722 1723 1725 1726 1729 1731 1734 1735 1736 1738
1739 1741 1742 1745 1746 1749 1751 1754 1755 1757 1758 1759 1761
1763 1764 1765 1766 1767 1769 1771 1774 1777 1780 1782 1789 1790
1792 1794 1795 1796 1797 1798 1799 1801 1802 1808 1810 1811 1813
1814 1816 1817 1818 1822 1825 1826 1828 1831 1832 1833 1835 1836
1839 1843 1844 1847 1850 1852 1853 1854 1856 1857 1858 1859 1862
1868 1869 1875 1876 1877 1878 1879 1880 1881 1882 1883 1884 1885
1886 1890 1891 1893 1897 1898 1907 1910 1911 1914 1916 1917 1919
1920 1922 1923 1926 1931 1932 1933 1934 1935 1936 1941 1943 1944
1945 1947 1950 1951 1955 1957 1958 1960 1962 1964 1967 1971 1973
1975 1976 1980 1981 1982 1985 1986 1989 1990 1991
```

```
cut size: 1113
```

可以看到结果中两组划分个数相同，且最小切割边数目为1113。

测试文件2 `prob4.txt`

输出结果 `outputprob4.txt`

```
0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 2 3 3 3 0 1
0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0
0 3 3 3 3 2 0 1
```

```
3  0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3
   3 3 1 0 0 0 0 0
4  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 1
   0 0 0 1 0 0 0 0
5  0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
   1 0 0 0 0 1 0 0
6  0 0 0 0 0 0 0 0 0 0 2 3 3 3 3 3 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 3 3
   3 3 3 3 3 3 2
7  0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   0 0 0 0 1 0 0 2
8  1 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
   0 1 0 0 0 0 0 3
9  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   0 0 0 3 3 3 3 3
10 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   3 3 3 3 0 0 0 0
11 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
   3 3 3 3 1 0 0 0
12 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0
   1 0 0 3 3 3 3 2
13 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
   0 0 0 3 3 3 3 3
14 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 2 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3
   3 3 3 3 1 0 0 0
15 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 3 2 0 0 0 0 0 0 1
   0 0 0 0 0 0 0 0
16 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 3 1 0 0 0 0 0 1
   0 0 0 1 0 0 0 0
17 0 0 0 0 1 0 0 0 1 0 1 1 0 0 3 3 3 3 3 3 3 3 3 3 3 2 0 0 0 0 0 0 0
   0 1 0 0 1 0 0 0
18 0 0 0 0 0 0 0 0 1 0 0 0 0 2 3 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0
19 0 0 1 1 0 0 1 0 0 0 0 0 0 0 3 3 3 3 3 2 0 0 0 0 0 0 0 0 0 0 0 1 0
   0 1 0 0 0 1 0
20 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 3 3 0 0 0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 1 0 0
21 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   0 1 0 0 0 0 1 0
22 0 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
   0 0 0 0 0 1 0 0
23 0 0 0 0 0 0 0 0 0 2 3 3 3 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0
```

```
24 1 0 0 0 0 0 0 0 0 0 0 0 3 3 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0
   0 0 0 0 0 0 0 0 0
25 0 0 0 0 0 0 0 1 0 0 0 0 0 3 3 3 3 2 0 0 0 0 0 0 1 0 0 0 1 0 0 0
   0 0 0 0 0 0 0 0 0
26 0 0 0 0 1 0 0 0 0 0 0 2 3 3 3 3 3 3 3 0 1 0 0 0 0 0 0 0 0 0 0 0
   1 0 0 0 0 0 0 0 0
27 0 0 0 2 3 3 3 3 3 3 3 3 3 3 3 3 1 0 0 0 1 0 0 0 0 0 3 3 3 1 0 2 3
   3 3 3 0 0 0 0 0
28 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 1 3 3 3 3 0
   0 0 3 3 1 0 0 0
29 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
   0 0 0 3 3 0 0 0
30 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 0 3
   3 3 3 3 2 1 0 0
31 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3
   1 0 0 0 1 0 0 0
32 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 3 3 3 3 3 1 0 0
   0 1 0 0 0 0 0 0
33 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 1 0 0 0 0 0 0
   0 0 0 0 0 0 0 1
34 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0
   1 0 0 0 0 0 0 0
35 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0
36 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0
37 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
   0 0 0 0 1 0 0 0
38 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
   0 0 0 0 0 0 0 1
39 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0
40 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0
```

`Astar`代价如下：

```
C:\Users\18064\projects\FMandAstar\file>Calc.exe nomodified.txt prob4.txt
Matrix Size: 40*40
Sum of differences: 219

C:\Users\18064\projects\FMandAstar\file>Calc.exe modified1.txt prob4.txt
Matrix Size: 40*40
Sum of differences: 197

C:\Users\18064\projects\FMandAstar\file>Calc.exe modified2.txt prob4.txt
Matrix Size: 40*40
Sum of differences: 168
```

在我们使用的简单的优化策略之下，总连线线长为168。

项目总结

我们的项目旨在完成布局布线算法中的两个关键部分：迷宫布局布线算法和使用FM算法进行划分。在项目中，我们引入了工厂模式来实现主要功能，以提高代码的抽象程度和可扩展性。

在迷宫布局布线算法中，我们采用了Astar算法，并使用优先队列来实现快速搜索和路径选择。我们还实现了简单的连线策略，以在大多数情况下得到相对优秀的连线结果。对于Astar算法生成多点的连线，我们采用了一些策略来优化结果。首先，我们增加了尝试创建布线的边的权重，以鼓励算法在这些位置上进行连线。其次，已经布线的边的权重被降低为0，以避免重复布线。

在FM算法的实现中，我们采用了桶型数据结构来查找最大增益并进行相应的更新。这种方法在处理大规模数据时，对算法的时间复杂度进行了优化。此外，我们还引入了多起始点的方法来改进FM算法的性能和效果。

通过我们的项目，我们取得了以下创新点和成果：

1. 引入工厂模式，提高了代码的抽象程度和可扩展性。
2. Astar算法的优先队列实现和连线策略，实现了大多数情况下较好的连线结果。
3. 通过增加边权重和降低已布线边的权重，优化了Astar算法生成多点连线的结果。
4. FM算法的桶型数据结构和多起始点方法，提高了算法在大数据量背景下的时间复杂度和性能。

总的来说，我们的项目在布局布线算法中取得了一定的创新和成果，提高了算法的效率和质量。然而，仍有进一步的改进空间，可以继续研究和优化算法，以满足更复杂和多样化的布局布线需求。