PJ2 report

程序操作方法

1. 运行pj2_handin.cpp程序后输入0或1之后回车；0代表Walking，1代表Taking bus
2. 输入起始点（范围是0-9）后回车
3. 输入结束点（范围是0-9）后回车 之后是输出结果
4. 以上1-3步是一个无限循环，在第一步输入9可以退出循环

- 程序会输出：**该请求所需的时间，道路的总长度，从start到end所经过的所有点**

算法分析

算法选择

- 使用`Dijkstra`算法来求最短路径，`Dijkstra`算法不能用于计算存在负权值的图；因为在具体的地图中不会出现权重为负数的情况，所以不需要使用时间复杂度较高的`Bellman-Ford`算法，而`Dijkstra`算法的性能要优于`Floyd`算法。

算法实现

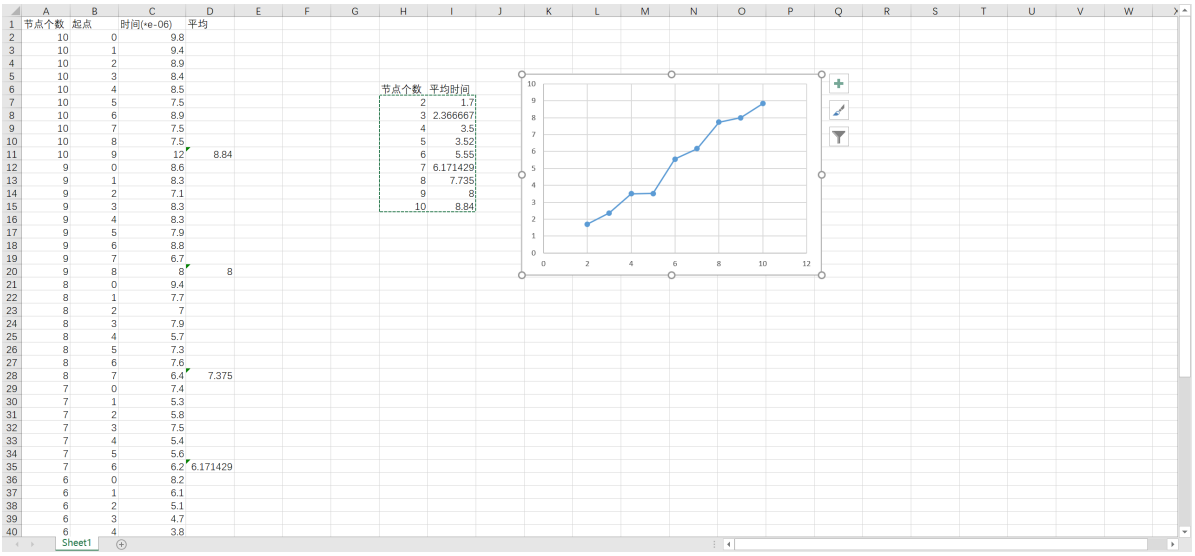
- 使用了`priority_queue`优先队列的数据结构，实际上是堆；在其中使用了插入和取最小值的两个函数。在其中使用了`pair<int, int>`的数据结构以同时记录点的序号和到起始点的距离。
- `length`数组用于记录从起始点到该点当前的最短距离。
- `visual`数组用于记录该节点是否已经被访问过，若已经被访问，则continue；否则继续执行

性能分析

针对`Dijkstra`函数的部分：

- 时间复杂度是O(nlogn)，其中n是图中的节点数（在本程序中为10）
    - 将一个节点插入优先队列的时间复杂度是O(logn)，因为插入一个节点需要调整堆结构，这个过程的时间复杂度是O(logn)。

    - 取出优先队列中的最小节点的时间复杂度是O(logn)
- 因此，每次从优先队列中取出节点的时间复杂度是O(logn)。由于要**遍历所有的节点**，所以总的时间复杂度是O(nlogn)。
- 另外，在对时间进行统计之后把请求的所需时间在excel中大致画了出来，如下图所示。采取的方法是通过改变节点个数以计算不同节点数量的请求所需时间，从而计算程序性能。可能由于节点个数较少，测出的实际值对于nlogn的理论值相比不是很精确（拟合比较困难）。



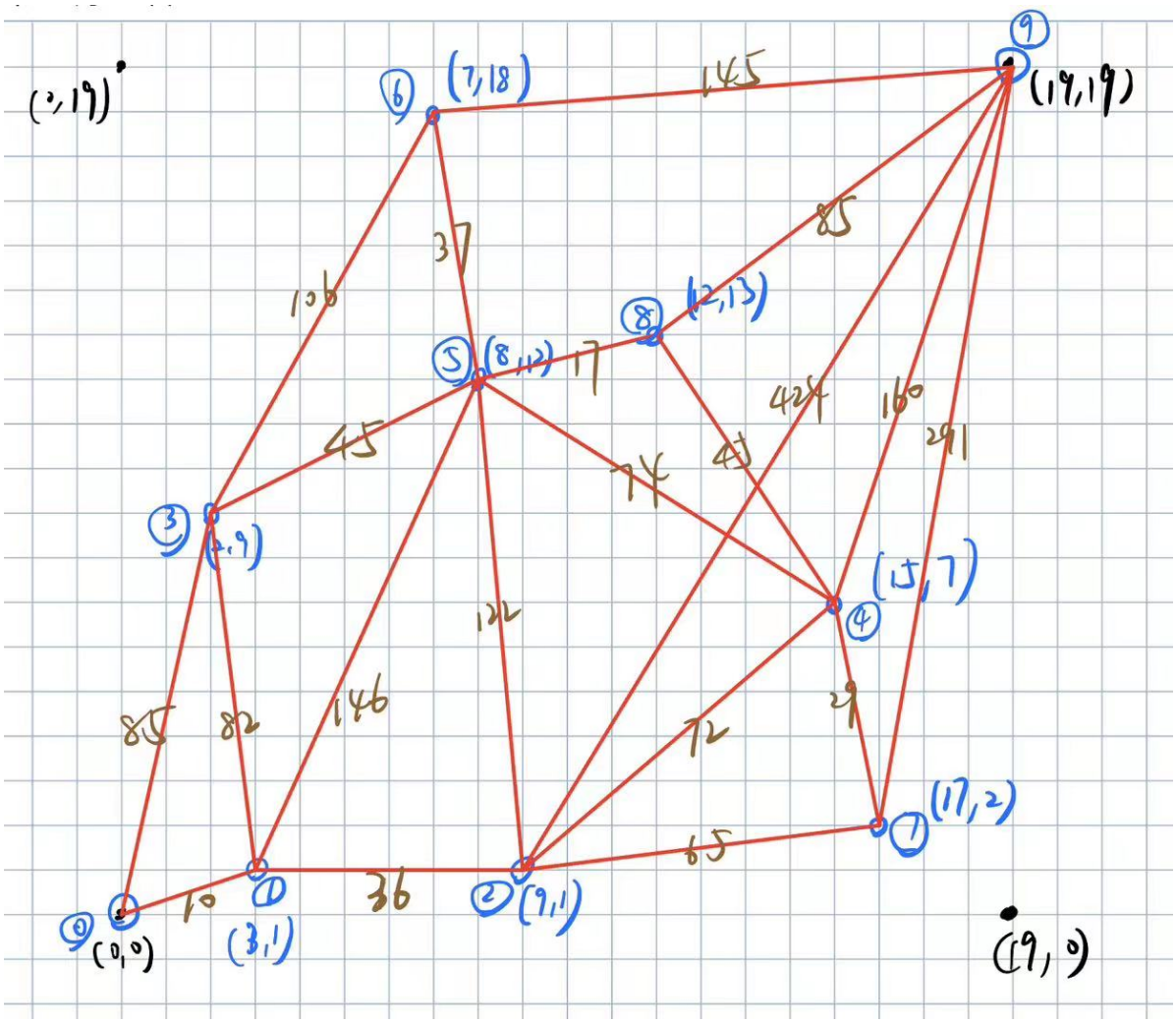
节点个数	起点	时间(*e-06)	平均
10	0	9.8	
10	1	9.4	
10	2	8.9	
10	3	8.4	
10	4	8.5	
10	5	7.5	
10	6	8.9	
10	7	7.5	
10	8	7.5	
10	9	12	8.84
9	0	8.6	
9	1	8.3	
9	2	7.1	
9	3	8.3	
9	4	8.3	
9	5	7.9	
9	6	8.8	
9	7	6.7	
9	8	8	8
8	0	9.4	
8	1	7.7	
8	2	7	
8	3	7.9	
8	4	5.7	
8	5	7.3	
8	6	7.6	
8	7	6.4	7.375
7	0	7.4	
7	1	5.3	
7	2	5.8	
7	3	7.5	
7	4	5.4	
7	5	5.6	
7	6	6.2	6.171429
6	0	8.2	
6	1	6.1	
6	2	5.1	
6	3	4.7	
6	4	3.8	

节点个数	平均时间
2	1.7
3	2.366667
4	3.5
5	3.52
6	5.55
7	6.171429
8	7.735
9	8
10	8.84

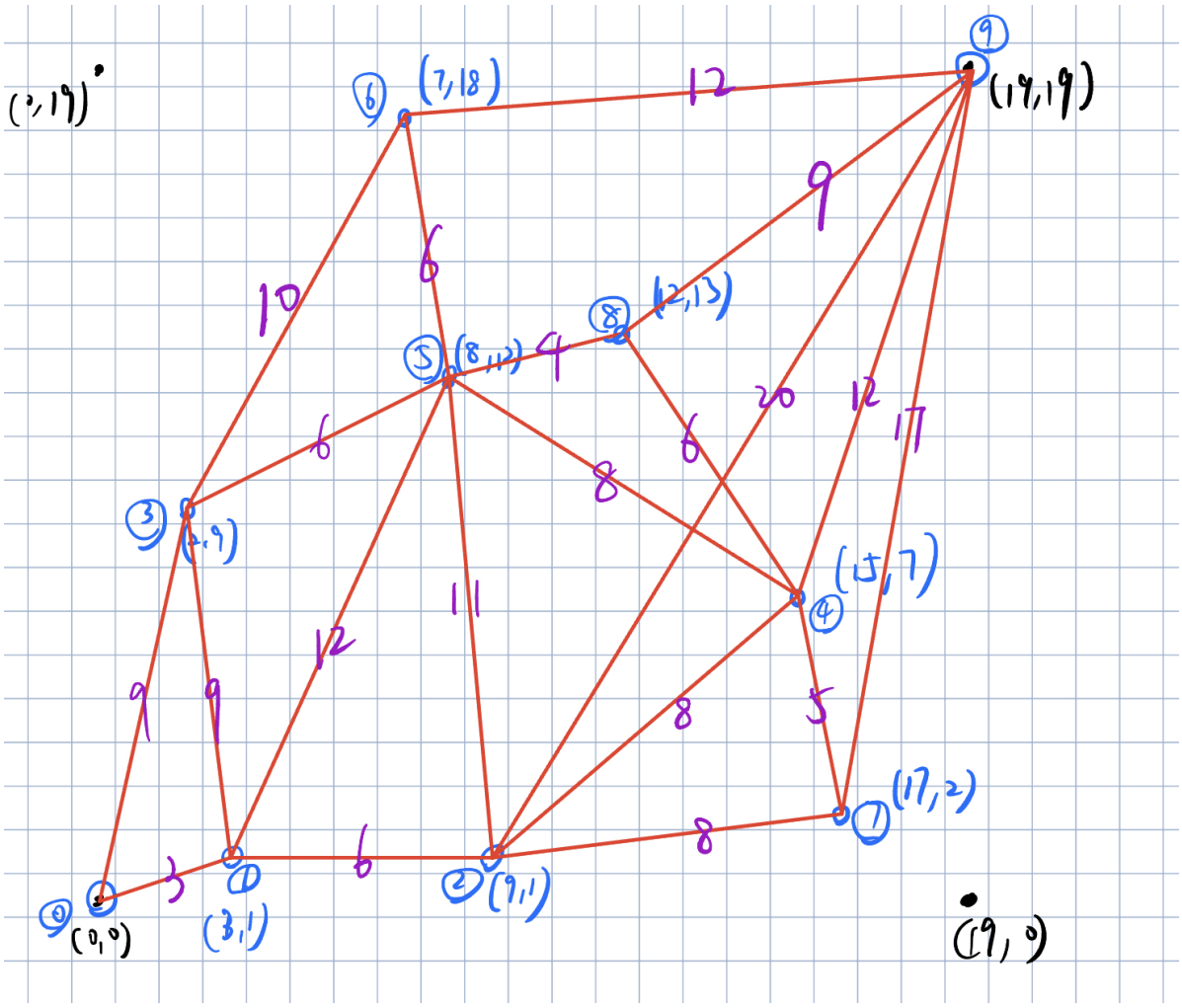
道路图片

其中边上的权重即经过该道路所需时间

- **Walking**


(7,18)
(19,19)
(12,13)
(8,11)
(4,9)
(15,7)
(17,2)
(9,1)
(3,1)
(0,0)
(19,0)
145
106
37
85
45
17
42
160
291
74
41
122
85
82
146
72
29
65
10
36

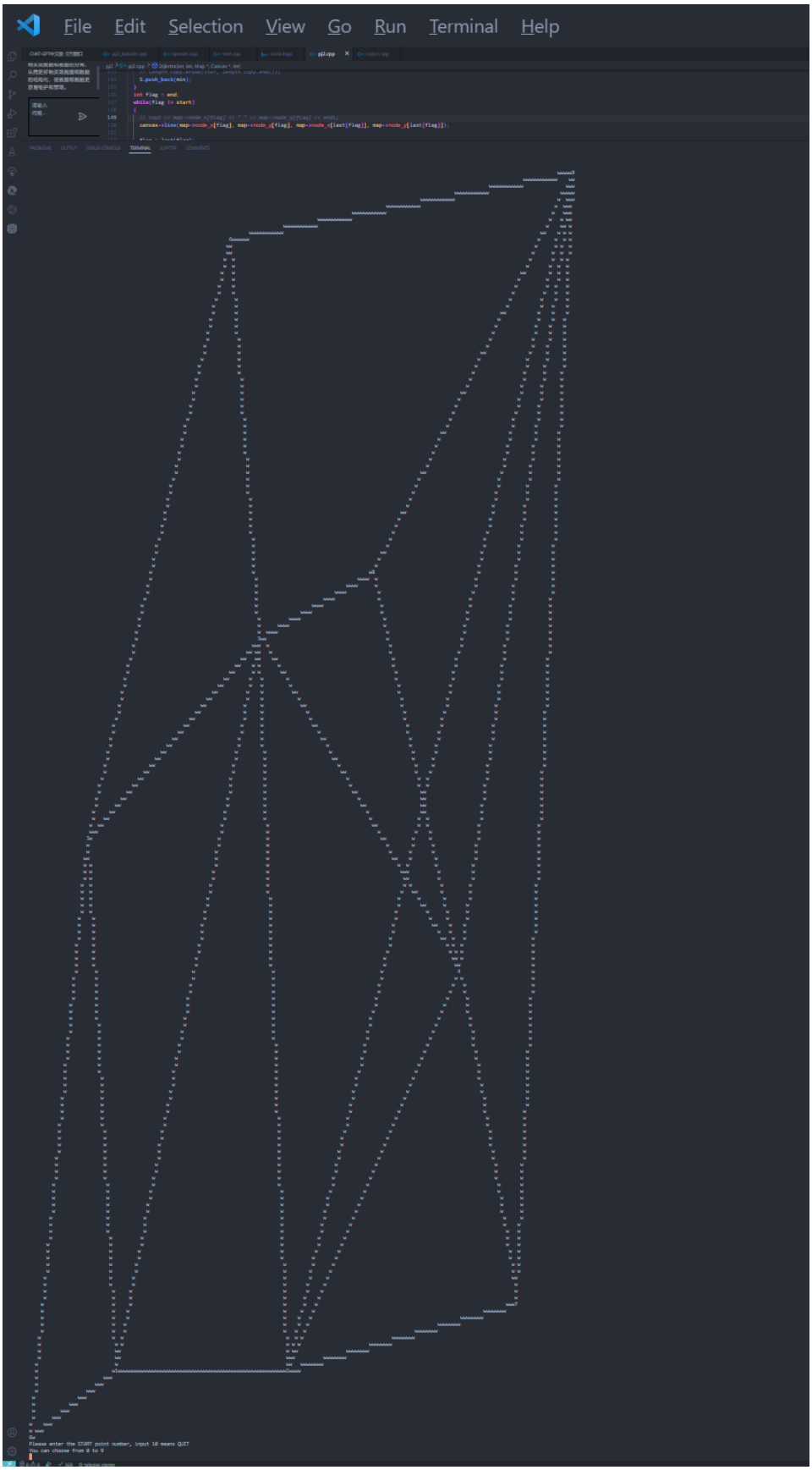
- **Taking bus**



PS

在PJ面测的时候采用了使用终端直接打出上述路径的Bresenham画线算法，但是这样的实现在终端中比较抽象，需要缩小很多才能看得比较清楚，故现在的程序没有采用之前的方法。以下这两张图就是之前程序的输出结果。

- 整个Map


File
Edit
Selection
View
Go
Run
Terminal
Help
Please enter the START point number, input 10 means QUIT
You can choose from 0 to 9

- 从0-9的最优路径