算法实验：Johnson算法

问题提出

- **实验4.1：Johnson算法**
  - 实现求所有点对最短路径的Johnson算法。有向图的顶点数 N 的取值分别为： 27、81、243、729，每个顶点作为起点引出的边的条数取值分别为：$\log_5 N$、$\log_7 N$ （取下整）。图的输入规模总共有4*2=8个，若同一个N，边的两种规模取值相等，则按后面输出要求输出两次，并在报告里说明。(不允许多重边，可以有环。)

其中输入输出目录组织结构如下

- 每种输入规模分别建立txt文件，文件名称为input11.txt， input12.txt,……,input42.txt （第一个数字为顶点数序号（27、81、243、729），第二个数字为弧数目序号（ $\log_5 N$、$\log_7 N$ ））；
- 生成的有向图信息分别存放在对应数据规模的txt文件中；
- 每行存放一对结点i，j序号（数字表示）和w_{ij}，表示存在一条结点i指向结点j的边，边的权值为w_{ij}，权 值范围为[-10,50],取整数。
- Input文件中为随机生成边以及权值，实验首先应判断输入图是否包含一个权重为负值的环路，如果存在，删除负环的一条边，消除负环，实验输出为处理后数据的实验结果，并在实验报告中说明。

- result.txt:输出对应规模图中所有点对之间的最短路径包含结点序列及路径长，不同规模写到不同的txt文件中，因此共有8个txt文件，文件名称为result11.txt,result12.txt,……,result42.txt;每行存一结点的对的最短路径，同一最短路径的结点序列用一对括号括起来输出到对应的txt文件中，并输 出路径长度。若图非连通导致节点对不存在最短路径，该节点对也要单独占一行说明。
- time.txt:运行时间效率的数据，不同规模的时间都写到同个文件。
- example:对顶点为27，边为54的所有点对最短路径实验输出应为：(1,5,2 20)(1,5,9,3 50)……，执行结果与运行时间的输出路径分别为：
  - output/result11.txt
  - output/time.txt

算法原理和实现

算法的实现方法主要有如下步骤

1. 给定图 G = (V, E)，增加一个新的顶点 s，使 s 指向图 G 中的所有顶点都建立连接，设新的图为 G′；
2. 对图 G′ 中顶点 s 使用 Bellman-Ford 算法计算单源最短路径，得到结果 h[] = {h[0], h[1], .. h[V-1]};
3. 对原图 G 中的所有边进行 "re-weight"(即松弛)，即对于每个边 (u, v)，其新的权值为 w(u, v) + (h[u] - h[v])；
4. 移除新增的顶点 s，对每个顶点运行 (优先队列优化的) Dijkstra 算法求得最短路径。 在实现细节上
5. 在graph.h中实现基于稠密方阵为邻接表示的图，以及对应的初始化，松弛操作，还有Dijstra算法。
6. 将预处理数据和给出结果分别在utils.h中的_main()实现。
7. 其他存储和输入另外给出。

实验结果

实验环境：

```
OS: OS X 13 (arm64)
Compiler: clang-1400.0.29.202
```

运行方法

输入数据：

1. 分别为`input{i}{j}.txt`，其中`i=1,2,3,4, j=1,2`，分别代表了 `1:27, 2:81, 3:243, 4:729` 的顶点数, `1:logN/log5, 2:logN/log7`其中`N`代表前一步的顶点数。

输出数据：

1. 计算的最短路径以及其权重，在`output`文件夹中。
2. 每次计算用的时间在`time.txt`文件中。 如下

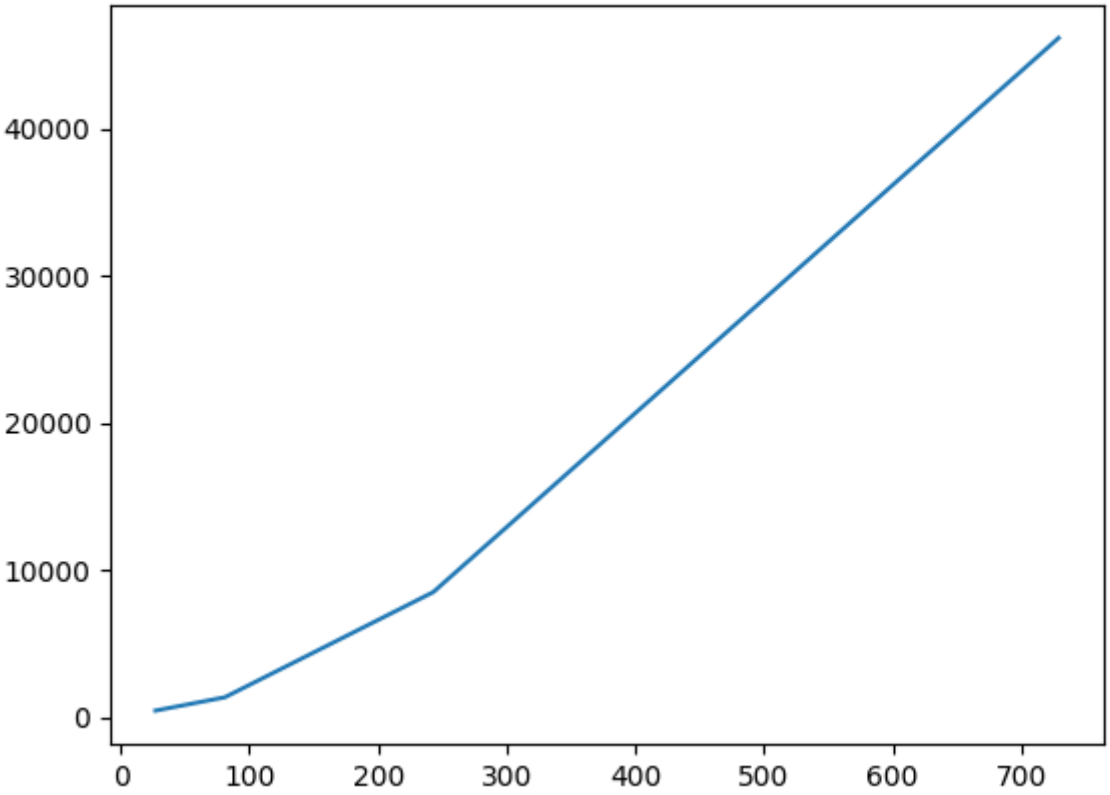
```
n_v     n_deg   milesecond  second
27      2       444         0.000444s
27      1       4963        0.004963s
81      2       1352        0.001352s
81      2       1258        0.001258s
243     3       8505        0.008505s
243     2       458924      0.458924s
729     4       46182       0.046182s
729     3       1181420     1.18142s
```

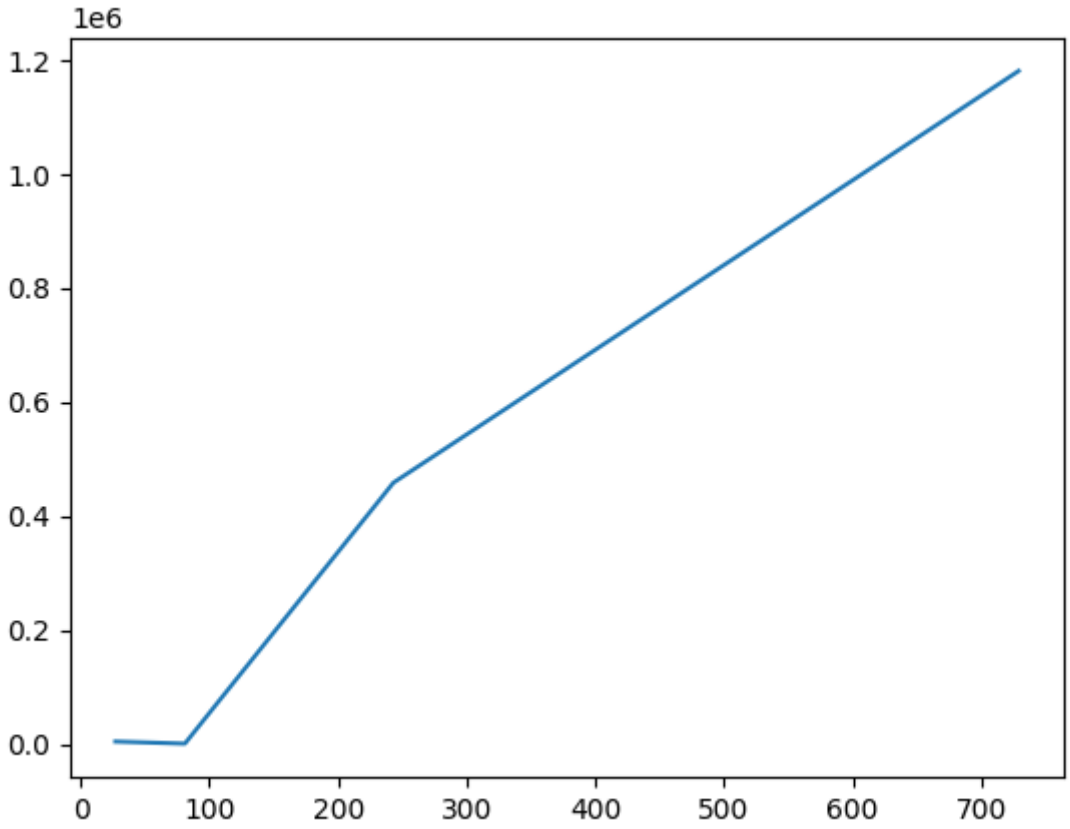
分析比较

可以做出如下的分析

1. 度的变化 可知，度的变化会带来一个量级左右的扩大

2. 和顶点数的变化 顶点数的增加带来超线性的增长.





复杂度分析

空间复杂度：Bellman-Ford 我们只需要一个额外的 h 数组，Dijkstra 需要 $O(V)$ 的空间不赘述。如果每一轮我们都直接将求出的最短路都输出而不是存在一个 $O(V2)$ 的数组中的话，那么额外空间就是 $O(V)$ 的，存图的空间为 $O(V+E)$ 总复杂度为 $O(V+E)$。

时间复杂度：因为Bellman-Ford 为 $O(VE)$ ，并且Dijskra算法的复杂度为$O(V*V+E)$(使用优先队列 优化的 Dijkstra 则总复杂度为 $O(V(E+V)\log E)$)。

缺点和不足

1. 十分难生成有效的图，采用了python的`networkx`模块来做。
2. 时间上因为难以处理输入和输出关系，在运行时直接实现了，这对最终时间可能有很大影响。

总结

这次实验提高了编程能力和实验能力。