lab4实验报告

学号：PB20061338 姓名： 柯志伟

1 实验设备和环境

平台：windows
编程语言：C与C++

2 实验内容及要求

2.1 实验内容

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_5N、 log_7N （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。)

2.2 实验要求

1. 实验设备和环境、实验内容及要求、方法和步骤、结果与分析
2. 比较实际复杂度和理论复杂度是否相同，给出分析

2.3 方法和步骤

1. 以最小堆实现优先队列,分别实现 dijkstra算法，bellman ford算法

以邻接表实现图的数据结构,并使用二维数组维护各边的权重,使用一维数组维护最短路径中各个顶点的前驱顶点,使用的数据结构如下

```
typedef struct node {
    int id;
    struct node* next;
} node,*vertex_list;
```

```
int Graph_weights[Vertex_num_expand][Vertex_num_expand];
vertex_list Graph_edges[Vertex_num_expand];
int Graph_pi[Vertex_num_expand];     // 顶点的前驱, 供
bellman_ford使用
int Graph_d[Vertex_num_expand];      // 目前的最短距离, 供
bellman_ford使用
int Graph_D[Vertex_num_expand][Vertex_num_expand]; // dijkstra
使用
int Graph_PI[Vertex_num_expand][Vertex_num_expand]; // dijkstra
使用
int Heap_nodes[Vertex_num+1]; // 使用堆实现优先队列, 0号元素对应
优先队列的长度
int Heap_size;
```

根据dijkstra算法,bellman ford算法实现具体代码,函数接口如下,详细见代码

```
void dijkstra(int s, int* d, int* pi);   // 传入源s, 储存当前各
顶点的最小距离的数组d,储存前趋顶点数组pi

bool bellman_ford(int s, int *l, int* r); // 传入源s, 以及l,r供
检测到负环时,函数向外传递检测到负环的边的左右顶点序号
```

2. 分别产生顶点数为27,81,243,729, 顶点的度为log5N,$log7N$的随机图(各边权重在[-10,50], 无重边, 可含环)

使用python脚本产生程序的输入数据

```python
import math
import random

vertex_scale = [27, 81, 243, 729]
weights = [i for i in range(-10, 51, 1)]
lines1 = []
lines2 = []

for index, i in enumerate(vertex_scale):
    lines1.clear()
    lines2.clear()
    vertexs = [i for i in range(i)]
    degree1 = int(math.log(i, 5))
    degree2 = int(math.log(i, 7))
    for j in range(i):
        vertex_list = []
        for k in range(degree1):
            vertex = random.choice(vertexs)
            while vertex == j or vertex in vertex_list:
                vertex = random.choice(vertexs)
            vertex_list.append(vertex)
            weight = random.choice(weights)
            lines1.append([str(j), str(vertex), str(weight)])

        with
open("E:\\Savefiles\\Labs\\Algorithm\\lab4\\ex1\\input\\input"
+ str(index + 1) + "1.txt", "w") as f:
            for line in lines1:
                f.write("\t".join(line) + "\n")

        vertex_list = []
        for k in range(degree2):
            vertex = random.choice(vertexs)
            while vertex == j or vertex in vertex_list:
                vertex = random.choice(vertexs)
            vertex_list.append(vertex)
            weight = random.choice(weights)
            lines2.append([str(j), str(vertex), str(weight)])

        with
open("E:\\Savefiles\\Labs\\Algorithm\\lab4\\ex1\\input\\input"
+ str(index + 1) + "2.txt", "w") as f:
            for line in lines2:
                f.write("\t".join(line) + "\n")
```

3. 实现求解所有节点最短路径的johnson算法

扩展原有图，增加节点s，并初始化其到其余顶点的距离为0,由于使用全局变量储存图的信息,使用的数据结构为数组,预留了空间，直接使用预留的空间即可

```
for(int i=0; i< Vertex_num; i++) {
    node* new_node = (node*) malloc(sizeof(node));
    new_node->id = i;
    Graph_weights[Vertex_num][i] = 0;
    if(! Graph_edges[Vertex_num]) {
        new_node->next = nullptr;
        Graph_edges[Vertex_num] = new_node;
    }
    else {
        node* list_head = Graph_edges[Vertex_num];
        new_node->next = list_head->next;
        list_head->next = new_node;
    }
}
```

在扩展图上执行bellman ford算法,同时使用bellman ford算法的返回值判断是否含有负环,如果含有就删除检测到负环时的边,重新执行直到成功

```
// 如果有负环就删除负环中的一条边
    while(!success) {
        success = bellman_ford(Vertex_num, &l, &r);
        node* node_ptr = Graph_edges[l];
        node* node_prev = Graph_edges[l];
        if(!success) {
            cout << "the input graph contains a negative-weight
cycle, try delete the edge ( " << l << ", " << r << " )" <<
endl;
            while(node_ptr) {
                if(node_ptr->id == r) {
                    if(node_ptr == Graph_edges[l]) {
                        Graph_edges[l] = node_ptr->next;
                    }
                    else {
                        node_prev->next = node_ptr->next;
                    }
                    free(node_ptr);
                    break;
                }
                node_ptr = node_ptr->next;
            }
        }
    }
```

重新赋值各个权重,并以各个顶点为源在更改权重后的原图上执行dijkstra算法求解各顶点对之间的最短距离，并在求解后还原原本的最短距离

```
for(int i=0; i<Vertex_num; i++) {
        node* node_ptr = Graph_edges[i];
        while(node_ptr) {
            Graph_weights[i][node_ptr->id] += Graph_d[i] -
Graph_d[node_ptr->id];
            node_ptr = node_ptr->next;
        }
}

for(int i=0; i< Vertex_num; i++) {
    dijkstra(i, Graph_D[i], Graph_PI[i]);
    for(int j=0; j<Vertex_num;j++) {
        if(Graph_D[i][j] != Infty) {
            Graph_D[i][j] += Graph_d[j] - Graph_d[i];
        }
    }
}
```

代码实现上的一些细节说明

1. 由于各边的权重在[-10,50]的范围内,考虑图中顶点数最多为729,同时考虑算法执行过程中不会溢出 以 `#define Infty  (2147483647 >> 2)` 作为权重的无穷大

2. 为支持johnson算法扩展图的要求,在表示图的各顶点数组以及权重数组中预留空间,扩展的顶点即为最后一号元素

3. 由于使用具体数值代表无穷大,同时由于图可能不连通,导致点对间的距离为无穷大,最后将结果保存到文件中时,此时根据与无穷大的比较来判断是否点对间的距离为无穷大,因此在使用johnson算法还原原有距离时要对这样的边跳过处理,一面后续与无穷大的比较出错

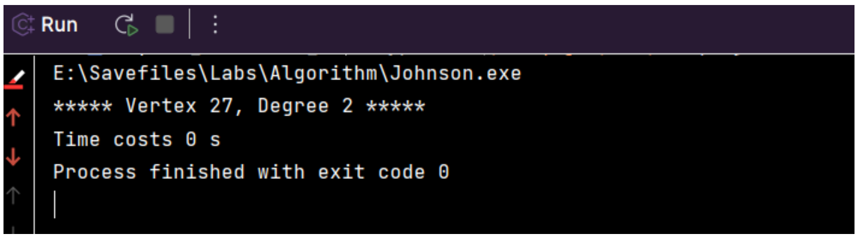
4. 在将结果保存到文件时,以`Infty`代表点对间的距离为无穷大

详细见代码

2.4 结果与分析

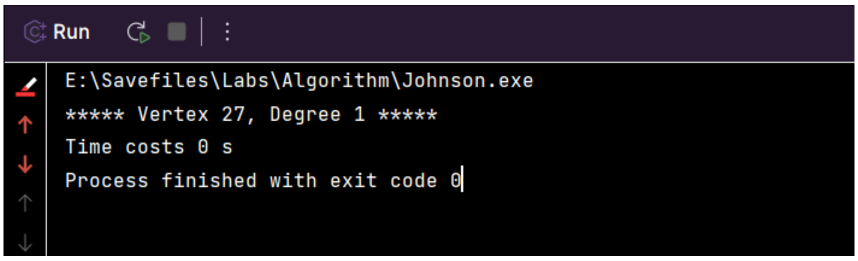
结果展示

input11.txt Vertex 27, Degree 2



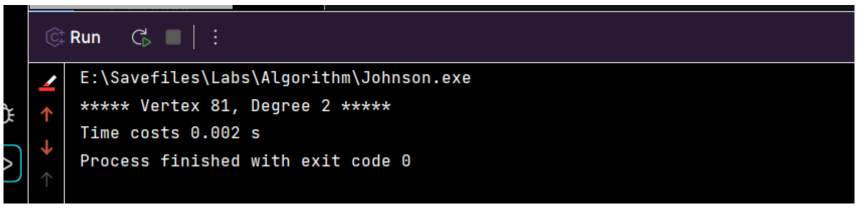
```
E:\Savefiles\Labs\Algorithm\Johnson.exe
***** Vertex 27, Degree 2 *****
Time costs 0 s
Process finished with exit code 0
```

input12.txt Vertex 27, Degree 1



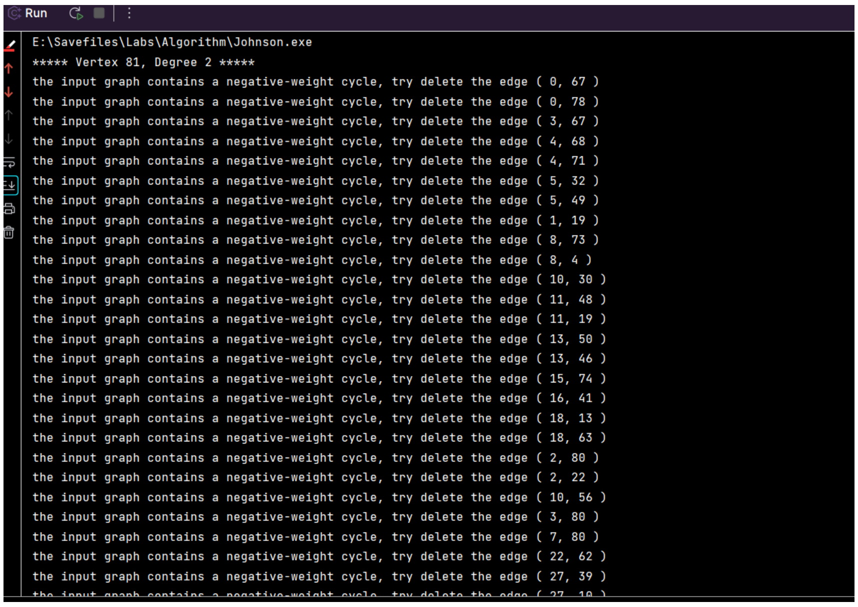
```
E:\Savefiles\Labs\Algorithm\Johnson.exe
***** Vertex 27, Degree 1 *****
Time costs 0 s
Process finished with exit code 0
```

input21.txt Vertex 81, Degree 2

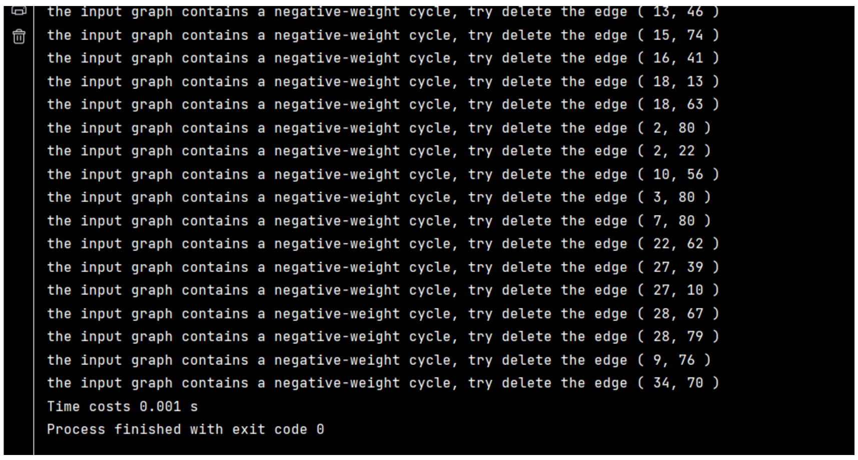


```
E:\Savefiles\Labs\Algorithm\Johnson.exe
***** Vertex 81, Degree 2 *****
Time costs 0.002 s
Process finished with exit code 0
```

input22.txt Vertex 81, Degree 2

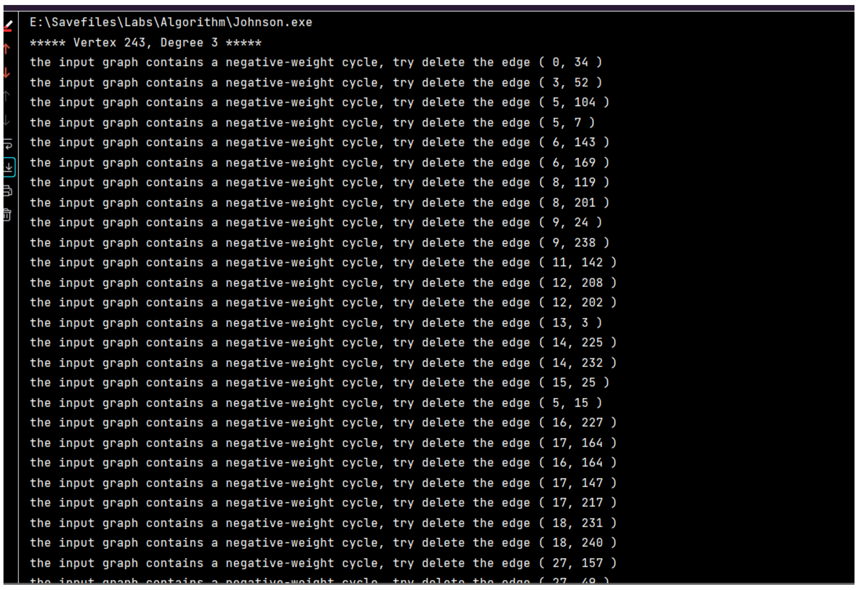


```
E:\Savefiles\Labs\Algorithm\Johnson.exe
***** Vertex 81, Degree 2 *****
the input graph contains a negative-weight cycle, try delete the edge ( 0, 67 )
the input graph contains a negative-weight cycle, try delete the edge ( 0, 78 )
the input graph contains a negative-weight cycle, try delete the edge ( 3, 67 )
the input graph contains a negative-weight cycle, try delete the edge ( 4, 68 )
the input graph contains a negative-weight cycle, try delete the edge ( 4, 71 )
the input graph contains a negative-weight cycle, try delete the edge ( 5, 32 )
the input graph contains a negative-weight cycle, try delete the edge ( 5, 49 )
the input graph contains a negative-weight cycle, try delete the edge ( 1, 19 )
the input graph contains a negative-weight cycle, try delete the edge ( 8, 73 )
the input graph contains a negative-weight cycle, try delete the edge ( 8, 4 )
the input graph contains a negative-weight cycle, try delete the edge ( 10, 30 )
the input graph contains a negative-weight cycle, try delete the edge ( 11, 48 )
the input graph contains a negative-weight cycle, try delete the edge ( 11, 19 )
the input graph contains a negative-weight cycle, try delete the edge ( 13, 50 )
the input graph contains a negative-weight cycle, try delete the edge ( 13, 46 )
the input graph contains a negative-weight cycle, try delete the edge ( 15, 74 )
the input graph contains a negative-weight cycle, try delete the edge ( 16, 41 )
the input graph contains a negative-weight cycle, try delete the edge ( 18, 13 )
the input graph contains a negative-weight cycle, try delete the edge ( 18, 63 )
the input graph contains a negative-weight cycle, try delete the edge ( 2, 80 )
the input graph contains a negative-weight cycle, try delete the edge ( 2, 22 )
the input graph contains a negative-weight cycle, try delete the edge ( 10, 56 )
the input graph contains a negative-weight cycle, try delete the edge ( 3, 80 )
the input graph contains a negative-weight cycle, try delete the edge ( 7, 80 )
the input graph contains a negative-weight cycle, try delete the edge ( 22, 62 )
the input graph contains a negative-weight cycle, try delete the edge ( 27, 39 )
the input graph contains a negative-weight cycle, try delete the edge ( 27, 10 )
```

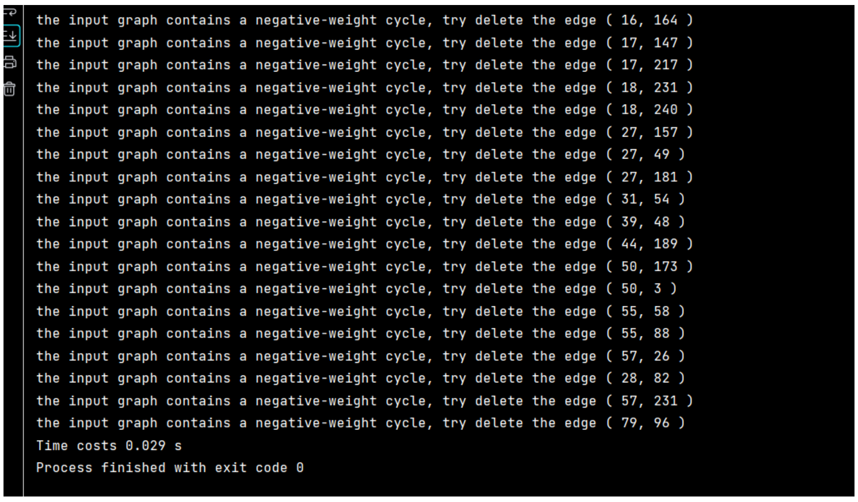


```
the input graph contains a negative-weight cycle, try delete the edge ( 13, 46 )
the input graph contains a negative-weight cycle, try delete the edge ( 15, 74 )
the input graph contains a negative-weight cycle, try delete the edge ( 16, 41 )
the input graph contains a negative-weight cycle, try delete the edge ( 18, 13 )
the input graph contains a negative-weight cycle, try delete the edge ( 18, 63 )
the input graph contains a negative-weight cycle, try delete the edge ( 2, 80 )
the input graph contains a negative-weight cycle, try delete the edge ( 2, 22 )
the input graph contains a negative-weight cycle, try delete the edge ( 10, 56 )
the input graph contains a negative-weight cycle, try delete the edge ( 3, 80 )
the input graph contains a negative-weight cycle, try delete the edge ( 7, 80 )
the input graph contains a negative-weight cycle, try delete the edge ( 22, 62 )
the input graph contains a negative-weight cycle, try delete the edge ( 27, 39 )
the input graph contains a negative-weight cycle, try delete the edge ( 27, 10 )
the input graph contains a negative-weight cycle, try delete the edge ( 28, 67 )
the input graph contains a negative-weight cycle, try delete the edge ( 28, 79 )
the input graph contains a negative-weight cycle, try delete the edge ( 9, 76 )
the input graph contains a negative-weight cycle, try delete the edge ( 34, 70 )
Time costs 0.001 s
Process finished with exit code 0
```

input31.txt Vertex 243, Degree 3

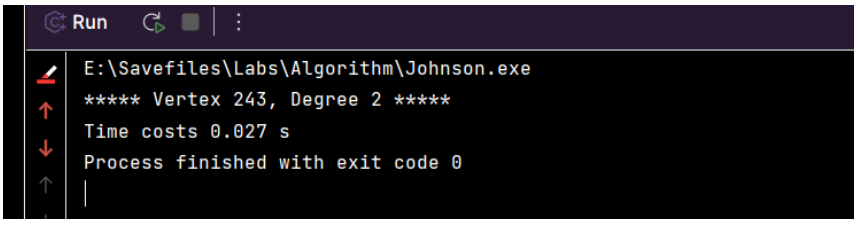


```
E:\Savefiles\Labs\Algorithm\Johnson.exe
***** Vertex 243, Degree 3 *****
the input graph contains a negative-weight cycle, try delete the edge ( 0, 34 )
the input graph contains a negative-weight cycle, try delete the edge ( 3, 52 )
the input graph contains a negative-weight cycle, try delete the edge ( 5, 104 )
the input graph contains a negative-weight cycle, try delete the edge ( 5, 7 )
the input graph contains a negative-weight cycle, try delete the edge ( 6, 143 )
the input graph contains a negative-weight cycle, try delete the edge ( 6, 169 )
the input graph contains a negative-weight cycle, try delete the edge ( 8, 119 )
the input graph contains a negative-weight cycle, try delete the edge ( 8, 201 )
the input graph contains a negative-weight cycle, try delete the edge ( 9, 24 )
the input graph contains a negative-weight cycle, try delete the edge ( 9, 238 )
the input graph contains a negative-weight cycle, try delete the edge ( 11, 142 )
the input graph contains a negative-weight cycle, try delete the edge ( 12, 208 )
the input graph contains a negative-weight cycle, try delete the edge ( 12, 202 )
the input graph contains a negative-weight cycle, try delete the edge ( 13, 3 )
the input graph contains a negative-weight cycle, try delete the edge ( 14, 225 )
the input graph contains a negative-weight cycle, try delete the edge ( 14, 232 )
the input graph contains a negative-weight cycle, try delete the edge ( 15, 25 )
the input graph contains a negative-weight cycle, try delete the edge ( 5, 15 )
the input graph contains a negative-weight cycle, try delete the edge ( 16, 227 )
the input graph contains a negative-weight cycle, try delete the edge ( 17, 164 )
the input graph contains a negative-weight cycle, try delete the edge ( 16, 164 )
the input graph contains a negative-weight cycle, try delete the edge ( 17, 147 )
the input graph contains a negative-weight cycle, try delete the edge ( 17, 217 )
the input graph contains a negative-weight cycle, try delete the edge ( 18, 231 )
the input graph contains a negative-weight cycle, try delete the edge ( 18, 240 )
the input graph contains a negative-weight cycle, try delete the edge ( 27, 157 )
the input graph contains a negative-weight cycle, try delete the edge ( 27, 49 )
```



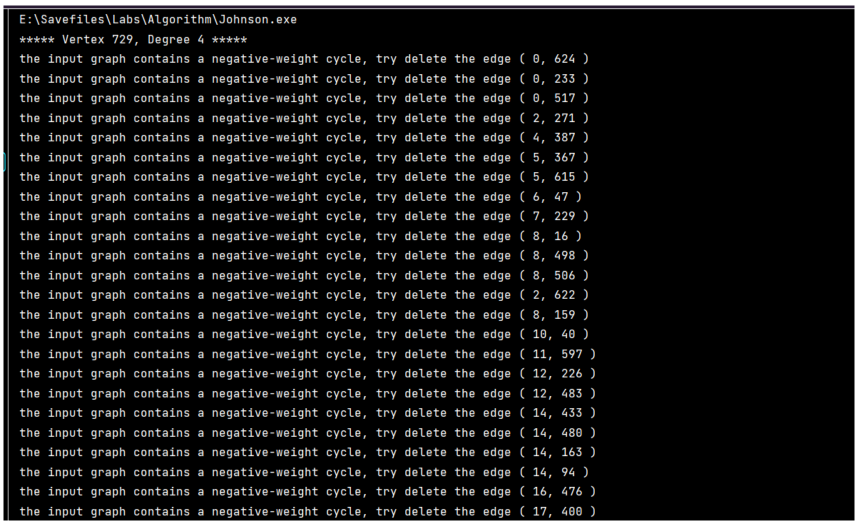
```
the input graph contains a negative-weight cycle, try delete the edge ( 16, 164 )
the input graph contains a negative-weight cycle, try delete the edge ( 17, 147 )
the input graph contains a negative-weight cycle, try delete the edge ( 17, 217 )
the input graph contains a negative-weight cycle, try delete the edge ( 18, 231 )
the input graph contains a negative-weight cycle, try delete the edge ( 18, 240 )
the input graph contains a negative-weight cycle, try delete the edge ( 27, 157 )
the input graph contains a negative-weight cycle, try delete the edge ( 27, 49 )
the input graph contains a negative-weight cycle, try delete the edge ( 27, 181 )
the input graph contains a negative-weight cycle, try delete the edge ( 31, 54 )
the input graph contains a negative-weight cycle, try delete the edge ( 39, 48 )
the input graph contains a negative-weight cycle, try delete the edge ( 44, 189 )
the input graph contains a negative-weight cycle, try delete the edge ( 50, 173 )
the input graph contains a negative-weight cycle, try delete the edge ( 50, 3 )
the input graph contains a negative-weight cycle, try delete the edge ( 55, 58 )
the input graph contains a negative-weight cycle, try delete the edge ( 55, 88 )
the input graph contains a negative-weight cycle, try delete the edge ( 57, 26 )
the input graph contains a negative-weight cycle, try delete the edge ( 28, 82 )
the input graph contains a negative-weight cycle, try delete the edge ( 57, 231 )
the input graph contains a negative-weight cycle, try delete the edge ( 79, 96 )
Time costs 0.029 s
Process finished with exit code 0
```

input32.txt Vertex 243, Degree 2

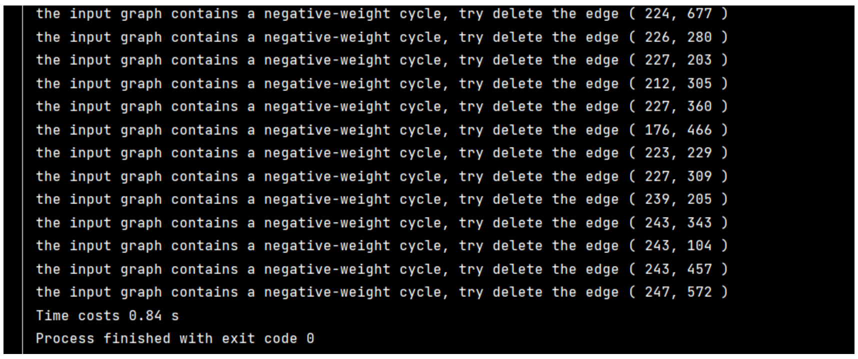


```
E:\Savefiles\Labs\Algorithm\Johnson.exe
***** Vertex 243, Degree 2 *****
Time costs 0.027 s
Process finished with exit code 0
```

input41.txt Vertex 729, Degree 4

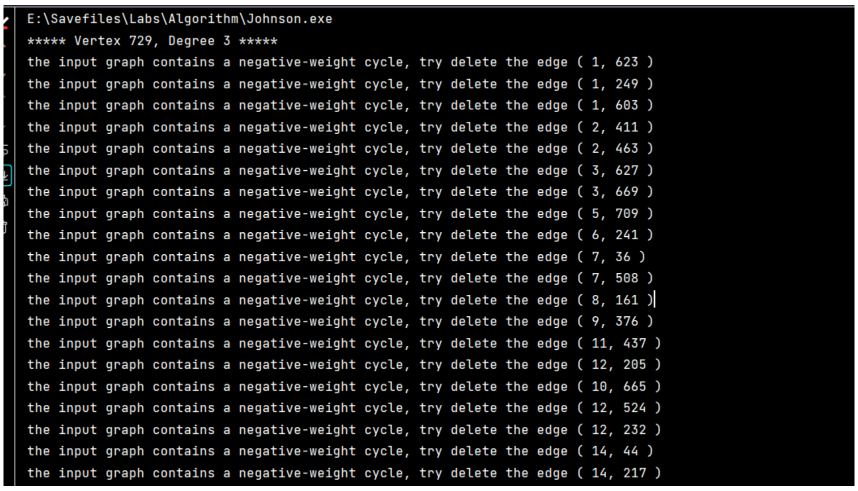


```
E:\Savefiles\Labs\Algorithm\Johnson.exe
***** Vertex 729, Degree 4 *****
the input graph contains a negative-weight cycle, try delete the edge ( 0, 624 )
the input graph contains a negative-weight cycle, try delete the edge ( 0, 233 )
the input graph contains a negative-weight cycle, try delete the edge ( 0, 517 )
the input graph contains a negative-weight cycle, try delete the edge ( 2, 271 )
the input graph contains a negative-weight cycle, try delete the edge ( 4, 387 )
the input graph contains a negative-weight cycle, try delete the edge ( 5, 367 )
the input graph contains a negative-weight cycle, try delete the edge ( 5, 615 )
the input graph contains a negative-weight cycle, try delete the edge ( 6, 47 )
the input graph contains a negative-weight cycle, try delete the edge ( 7, 229 )
the input graph contains a negative-weight cycle, try delete the edge ( 8, 16 )
the input graph contains a negative-weight cycle, try delete the edge ( 8, 498 )
the input graph contains a negative-weight cycle, try delete the edge ( 8, 506 )
the input graph contains a negative-weight cycle, try delete the edge ( 2, 622 )
the input graph contains a negative-weight cycle, try delete the edge ( 8, 159 )
the input graph contains a negative-weight cycle, try delete the edge ( 10, 40 )
the input graph contains a negative-weight cycle, try delete the edge ( 11, 597 )
the input graph contains a negative-weight cycle, try delete the edge ( 12, 226 )
the input graph contains a negative-weight cycle, try delete the edge ( 12, 483 )
the input graph contains a negative-weight cycle, try delete the edge ( 14, 433 )
the input graph contains a negative-weight cycle, try delete the edge ( 14, 480 )
the input graph contains a negative-weight cycle, try delete the edge ( 14, 163 )
the input graph contains a negative-weight cycle, try delete the edge ( 14, 94 )
the input graph contains a negative-weight cycle, try delete the edge ( 16, 476 )
the input graph contains a negative-weight cycle, try delete the edge ( 17, 400 )
```

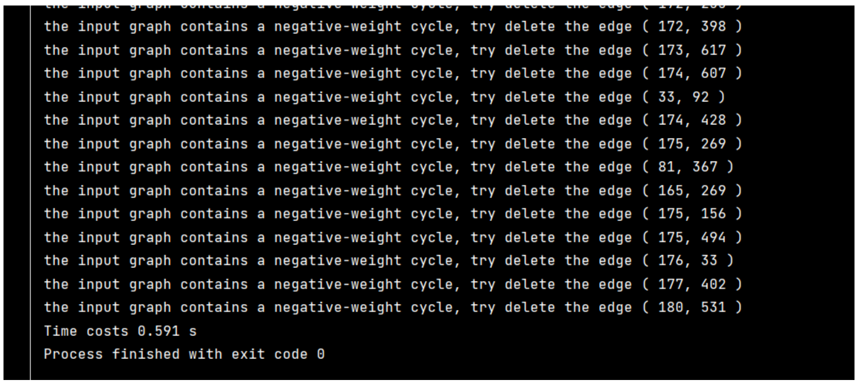


```
the input graph contains a negative-weight cycle, try delete the edge ( 224, 677 )
the input graph contains a negative-weight cycle, try delete the edge ( 226, 280 )
the input graph contains a negative-weight cycle, try delete the edge ( 227, 203 )
the input graph contains a negative-weight cycle, try delete the edge ( 212, 305 )
the input graph contains a negative-weight cycle, try delete the edge ( 227, 360 )
the input graph contains a negative-weight cycle, try delete the edge ( 176, 466 )
the input graph contains a negative-weight cycle, try delete the edge ( 223, 229 )
the input graph contains a negative-weight cycle, try delete the edge ( 227, 309 )
the input graph contains a negative-weight cycle, try delete the edge ( 239, 205 )
the input graph contains a negative-weight cycle, try delete the edge ( 243, 343 )
the input graph contains a negative-weight cycle, try delete the edge ( 243, 104 )
the input graph contains a negative-weight cycle, try delete the edge ( 243, 457 )
the input graph contains a negative-weight cycle, try delete the edge ( 247, 572 )
Time costs 0.84 s
Process finished with exit code 0
```

input42.txt Vertex 729, Degree 3



```
E:\Savefiles\Labs\Algorithm\Johnson.exe
***** Vertex 729, Degree 3 *****
the input graph contains a negative-weight cycle, try delete the edge ( 1, 623 )
the input graph contains a negative-weight cycle, try delete the edge ( 1, 249 )
the input graph contains a negative-weight cycle, try delete the edge ( 1, 603 )
the input graph contains a negative-weight cycle, try delete the edge ( 2, 411 )
the input graph contains a negative-weight cycle, try delete the edge ( 2, 463 )
the input graph contains a negative-weight cycle, try delete the edge ( 3, 627 )
the input graph contains a negative-weight cycle, try delete the edge ( 3, 669 )
the input graph contains a negative-weight cycle, try delete the edge ( 5, 709 )
the input graph contains a negative-weight cycle, try delete the edge ( 6, 241 )
the input graph contains a negative-weight cycle, try delete the edge ( 7, 36 )
the input graph contains a negative-weight cycle, try delete the edge ( 7, 508 )
the input graph contains a negative-weight cycle, try delete the edge ( 8, 161 )
the input graph contains a negative-weight cycle, try delete the edge ( 9, 376 )
the input graph contains a negative-weight cycle, try delete the edge ( 11, 437 )
the input graph contains a negative-weight cycle, try delete the edge ( 12, 205 )
the input graph contains a negative-weight cycle, try delete the edge ( 10, 665 )
the input graph contains a negative-weight cycle, try delete the edge ( 12, 524 )
the input graph contains a negative-weight cycle, try delete the edge ( 12, 232 )
the input graph contains a negative-weight cycle, try delete the edge ( 14, 44 )
the input graph contains a negative-weight cycle, try delete the edge ( 14, 217 )
```



```
the input graph contains a negative-weight cycle, try delete the edge ( 172, 398 )
the input graph contains a negative-weight cycle, try delete the edge ( 173, 617 )
the input graph contains a negative-weight cycle, try delete the edge ( 174, 607 )
the input graph contains a negative-weight cycle, try delete the edge ( 33, 92 )
the input graph contains a negative-weight cycle, try delete the edge ( 174, 428 )
the input graph contains a negative-weight cycle, try delete the edge ( 175, 269 )
the input graph contains a negative-weight cycle, try delete the edge ( 81, 367 )
the input graph contains a negative-weight cycle, try delete the edge ( 165, 269 )
the input graph contains a negative-weight cycle, try delete the edge ( 175, 156 )
the input graph contains a negative-weight cycle, try delete the edge ( 175, 494 )
the input graph contains a negative-weight cycle, try delete the edge ( 176, 33 )
the input graph contains a negative-weight cycle, try delete the edge ( 177, 402 )
the input graph contains a negative-weight cycle, try delete the edge ( 180, 531 )
Time costs 0.591 s
Process finished with exit code 0
```

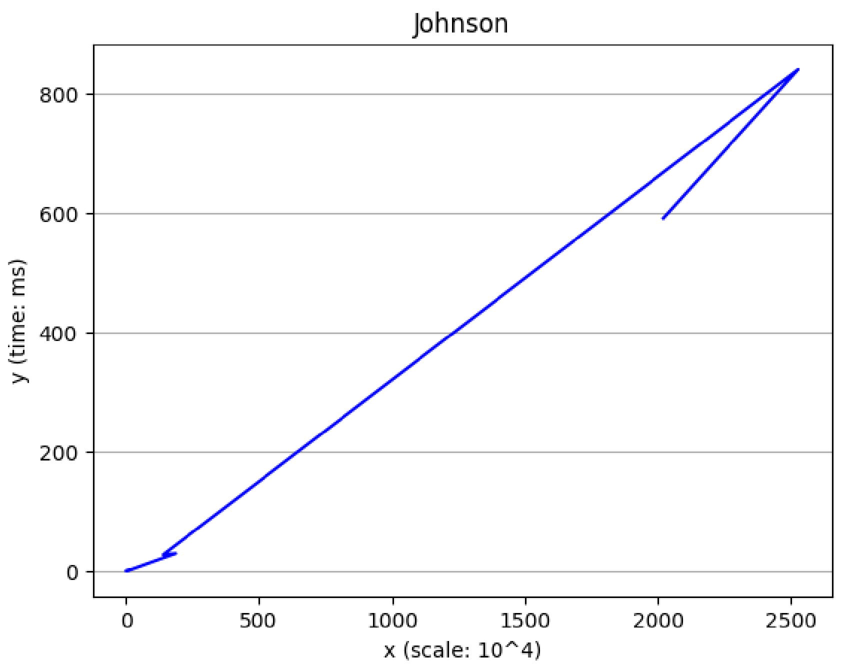
数据分析

图的规模以及运行时间如下:

顶点数	边数	V(V+E)lgV	运行时间(s)
27	54	1.03e+04	0
27	27	6.93e+03	0
81	162	1.24e+05	0.002
81	162	1.24e+05	0.001
243	729	1.87e+06	0.029
243	486	1.40e+06	0.027
729	2916	2.52e+07	0.84
729	2187	2.02e+07	0.591

以预期时间复杂度绘制折线图如下:

```python
import numpy as np
import matplotlib.pyplot as plt
## johnson算法
time = np.array([0,0,0.002,0.001,0.029,0.027,0.84,0.591])
vertex = np.array([27,27,81,81,243,243,729,729])
degree = np.array([2,1,2,2,3,2,4,3])

edges = vertex * degree

scales = vertex*(vertex+edges)*np.log2(vertex)


x = scales / 10000
y = time*1000

plt.title("Johnson")

plt.plot(x, y,'b')
plt.xlabel("x (scale: 10^4)")
plt.ylabel("y (time: ms)")
plt.grid(axis='y')
plt.plot()
```



johnson算法主体依靠dijkstra算法获得各点对之间的最短距离,而dijkstra算法的性能很大程度上依赖于优先队列的实现,本次实验中以二叉堆实现的优先队列的EXTRACT-MIN和DECREASE-KEY的时间复杂度均为lgN,在dijkstra算法中两者执行的次数分别为V,E,故dijkstra算法的时间复杂度为O((V+E)lgV),因此johnson算法的时间复杂度为O(V(V+E)lgV),由图可知除异常的一个数据外，其余数据大致在一条直线上,由于本次实验中的图为随机生成,且由于可能产生负圈导致最后的图的规模与原图有一定偏差,最终有一定误差,因此该图可以较好的符合该时间复杂度