第四次上机实验报告

一、 问题介绍

实现用于稀疏图的 Johnson 算法，解决边权值可能为负值的图的所有路径对的最短路径问题，输出最短路径和最短路径长度，并随机生成图分析算法复杂度。

二、 算法思想及实现

- 算法思想

  Johnson 算法：

  要求任一结点对之间的最短路径，只需将单源最短路径的算法运行 G.V 次。由于 Dijkstra 算法比 Bellman-Ford 算法效率高，但需要边权值非负，于是给图 G 增加结点 s 和边(s,v)且边权值 w(s,v)=0 构造图 G'，利用 Bellman-Ford 算法的结果，重新赋予权重，使边权值非负。再调用 G.V 次 Dijkstra 算法，消去调整值即可得到最短路径。

  Bellman-Ford 算法：

  解决一般情况下的单源最短路径问题。对图进行初始化后，对每条边进行 G.V-1 次松弛操作，即可得到单源最短路径，最后检查是否有权重为负值的环路。

  Dijkstra 算法：

  解决边权值非负的单源最短路径问题。对图进行初始化后，按 d 值将顶点排成最小堆。每次移除 d 值最小的顶点，维持最小堆的性质，并对此顶点所连的边进行松弛操作，直到移除所有的顶点，所得 d 值即为单源最短路径长度。

- 实现

  稀疏图用邻接链表存储，顶点存储在数组中，每条链表连接顶点所连的边。图、顶点、边均用结构体表示。

  边：
```
struct edge{
        int num;
        int weight;
        edge* neighbor;
    };
```
  存储指向的顶点编号、权重值、下一条边。

  顶点：
```
struct vertex{
        int d;
        int num;
        int heapindex;
        vertex* parent;
        edge* neighbor;
    };
```
  存储最短路径估计值 d、顶点编号、在 Dijkstra 算法中的最小堆中的下标、父结点、所连的边。

```
图：struct graph{
        vertex* chain[1000];
        int V;
    };
```

存储图的邻接链表及顶点数量。

三、 实验中问题、理解与解决方法

- 由于要使对最小堆的操作 DECREASE_KEY 时间复杂度为 O(lgV)，需已知结点在最小堆中的位置，则应在结点及其堆元素间相互保存指向对方的句柄，在结构体 vertex 中加上下标 heapindex，并在堆算法中维护 heapindex 的值。
- 每次调用 Dijkstra 算法，结点的 d 值和 parent 值会改变，需用二维数组存储每次的结果。且由于函数不能返回两个数组，故将数组定义为全局变量。
- 在初始化图时，d 值的无穷值若定为 INT_MAX，则加上正数后会变成负数（由于 int 值的存储规则），故应将其定为较小的数，如 99999999。
- 在 Dijkstra 算法中，注意不要对连接已移除最小堆的顶点的边再进行松弛操作。
- 由于图中不能出现负权值的环路，为提高有效的图的比例，在随机生成图时增加正的边权值的比例。

四、 测试结果

使用课本上的示例进行测试，输出任一对结点间的最短路径和最短路径权值之和：

```
选择执行的操作：1.测试(书上示例) 2.分析性能(随机生成图)
1
最短路径长度矩阵：
0       1       -3      2       -4
3       0       -4      1       -1
7       4       0       5       3
2       -1      -5      0       -2
8       5       1       6       0
从结点1到结点1的最短路径
1
从结点1到结点2的最短路径
1       5       4       3       2
从结点1到结点3的最短路径
1       5       4       3
从结点1到结点4的最短路径
1       5       4
从结点1到结点5的最短路径
1       5
从结点2到结点1的最短路径
2       4       1
从结点2到结点2的最短路径
2
从结点2到结点3的最短路径
2       4       3
```

```
从结点2到结点4的最短路径
2       4
从结点2到结点5的最短路径
2       4       1       5
从结点3到结点1的最短路径
3       2       4       1
从结点3到结点2的最短路径
3       2
从结点3到结点3的最短路径
3
从结点3到结点4的最短路径
3       2       4
从结点3到结点5的最短路径
3       2       4       1       5
从结点4到结点1的最短路径
4       1
从结点4到结点2的最短路径
4       3       2
从结点4到结点3的最短路径
4       3
从结点4到结点4的最短路径
4
从结点4到结点5的最短路径
4       1       5
```

```
从结点5到结点1的最短路径
5       4       1
从结点5到结点2的最短路径
5       4       3       2
从结点5到结点3的最短路径
5       4       3
从结点5到结点4的最短路径
5       4
从结点5到结点5的最短路径
5
```

与课本上结果相同。

五、 性能分析

随机生成图，自定义图的顶点个数，对每个顶点随机选择顶点相连（注意不能重复也不能选择自身），保证了随机性，故每条边可固定连五条边，不影响性能的分析。边权值在[-1,100]间随机选择整数，减少含有负权值的环路的比例。在分析性能时只记录算法本身的运行时间，不输出最短路径和最短路径权重之和。

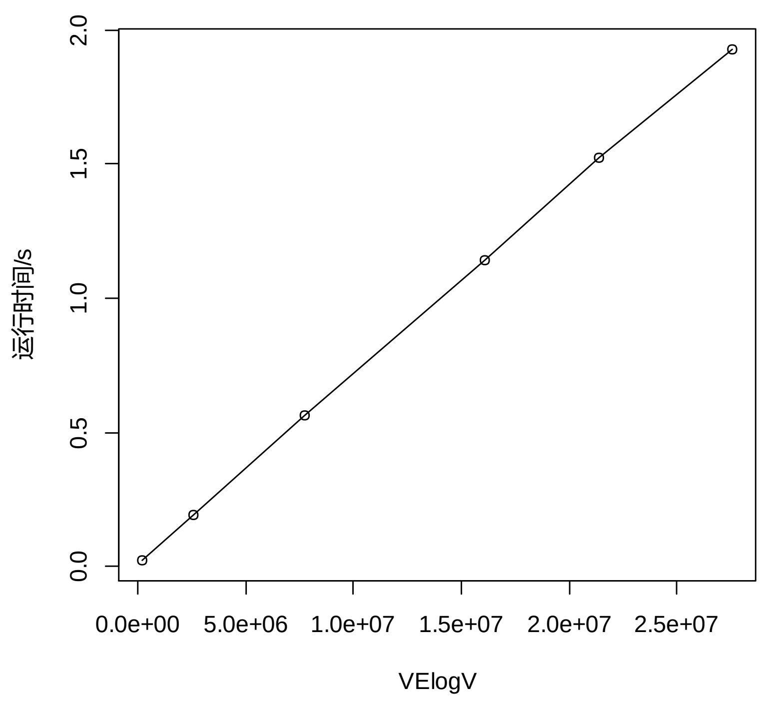
示例：

```
选择执行的操作：1.测试<书上示例> 2.分析性能<随机生成图>
2
输入矩阵规模
100
总时间：0.027s
请按任意键继续. . .
```

作时间关于顶点数 V 和边数 E 的表格：

		时间 1/s	时间 2/s	时间 3/s	平均时间/s
顶点数 V	100	0.031	0.015	0.032	0.026
边数 E	500				
顶点数 V	300	0.187	0.188	0.203	0.193
边数 E	1500				
顶点数 V	500	0.563	0.562	0.563	0.563
边数 E	2500				
顶点数 V	700	1.141	1.125	1.156	1.141
边数 E	3500				
顶点数 V	800	1.484	1.566	1.519	1.523
边数 E	4000				
顶点数 V	900	1.953	1.937	1.891	1.927
边数 E	4500				

作曲线图 运行时间-顶点数*边数*log(顶点数)



由图可知，曲线图为一条直线。

六、 实验结论

Johnson 算法的时间复杂度为 O(VElogV)，与课本上结论相符。