

# 算法实验四

Pb18081616 谭园

## 一、实验要求

### ■实验4.1: Kruskal算法

- 实现求最小生成树的Kruskal算法。无向图的顶点数 $N$ 的取值分别为：8、64、128、512，对每一顶点随机生成 $1 \sim \lfloor N/2 \rfloor$ 条边，随机生成边的权重，统计算法所需运行时间，画出时间曲线，分析程序性能。

### ■实验4.2: 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 $4 \times 2 = 8$ 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。）

## 二、实验环境

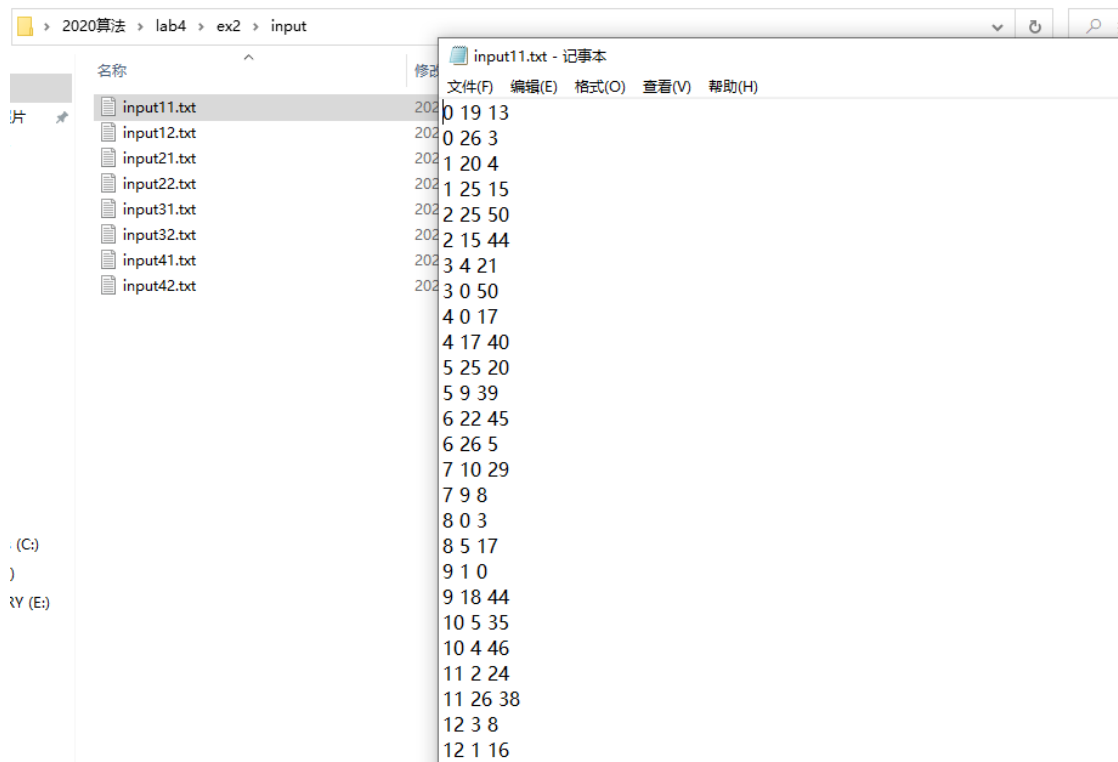
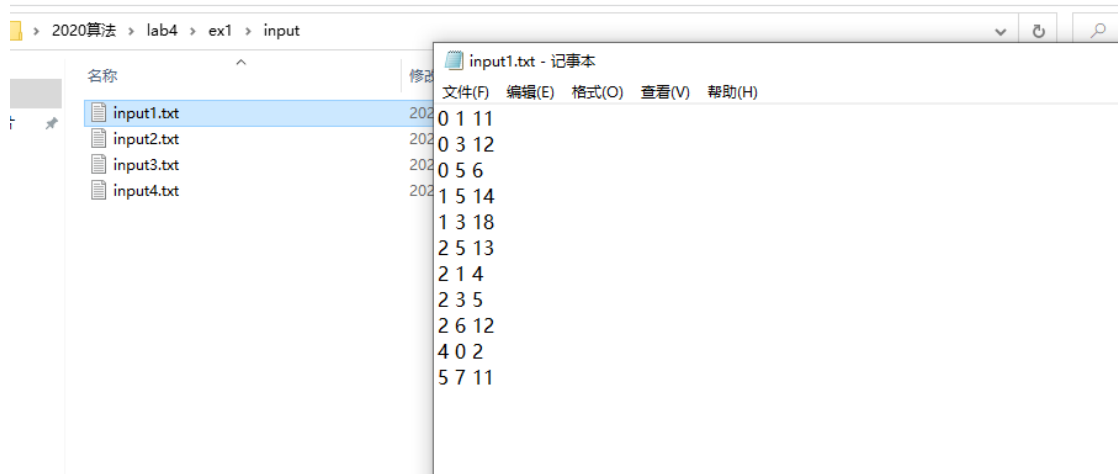
编译环境：DEV C++

机器内存：16GB

时钟主频：2.3GHz

## 三、实验过程

1. 利用 rand 函数生成了所需的 input 文件。



2. 写出 kruskal 算法和 Johnson 算法的实现

```

5
6 plinklist kruskal(algraph G, edge* edges) {
7     int i;
8     pset* x;
9     plinklist p, head, rear;
10    head = rear = NULL;
11    x = (pset*)malloc(G.vexnum * sizeof(pset));
12    for (i = 0; i < G.vexnum; i++) {
13        x[i] = (pset)malloc(sizeof(sset));
14        x[i]->data = i;
15        make_set(x[i]);
16    }
17    heap_sort(edges, G.arcnum);
18    for (i = 0; i < G.arcnum; i++) {
19        if (find_set(x[edges[i].v1]) != find_set(x[edges[i].v2]))
20            p = (plinklist)malloc(sizeof(linklist));
21            p->data = i;
22            if (head == NULL) {
23                head = p;
24                rear = p;
25            }
26            else {
27                rear->next = p;
28                rear = rear->next;
29            }
30            union_set(x[edges[i].v1], x[edges[i].v2]);
31    }
32    rear->next = NULL;
33    return head;
34 }

```

```

40
41 void johnson(algraph G, FILE* fp) {
42     int i, j;
43     int* h;
44     int dd;
45     algraph Gci;
46     arcnode* pa, * ra = NULL;
47     Gci.vexnum = G.vexnum + 1;
48     Gci.arcnum = G.arcnum + G.vexnum;
49     Gci.vertices = (vnode*)malloc(Gci.vexnum * sizeof(vnode));
50     for (i = 1; i <= G.vexnum; i++)
51         Gci.vertices[i] = G.vertices[i - 1];
52     Gci.vertices[0].firstarc = NULL;
53     for (i = 1; i <= G.vexnum; i++) {
54         pa = (arcnode*)malloc(sizeof(arcnode));
55         pa->adjvex = i;
56         pa->w = 0;
57         if (Gci.vertices[0].firstarc == NULL) {
58             Gci.vertices[0].firstarc = pa;
59             ra = pa;
60         }
61         else {
62             ra->nextarc = pa;
63             ra = ra->nextarc;
64         }
65     }
66     ra->nextarc = NULL;
67     h = (int*)malloc(G.vexnum * sizeof(int));
68     if (bellman_ford(Gci, 0) == false)
69         printf("有负环\n");
70     else {
71         for (i = 1; i < Gci.vexnum; i++)
72             h[i - 1] = Gci.vertices[i].d;
73         for (i = 0; i < G.vexnum; i++) {
74             pa = Gci.vertices[i].firstarc;

```

```

265     }
266     ra->nextarc = NULL;
267     h = (int*)malloc(G.vexnum * sizeof(int));
268     if (bellman_ford(Gci, 0) == false)
269         printf("有负环\n");
270     else {
271         for (i = 1; i < Gci.vexnum; i++)
272             h[i - 1] = Gci.vertices[i].d;
273         for (i = 0; i < G.vexnum; i++) {
274             pa = G.vertices[i].firstarc;
275             while (pa != NULL) {
276                 pa->w = pa->w + h[i] - h[pa->adjvex];
277                 pa = pa->nextarc;
278             }
279         }
280         for (i = 0; i < G.vexnum; i++) {
281             dijkstra(G, i);
282             for (j = 0; j < G.vexnum; j++) {
283                 if (j != i) {
284                     fprintf(fp, "(%d", i);
285                     output(G, i, j, fp);
286                     dd = G.vertices[j].d + h[j] - h[i];
287                     if (dd < 9000000)
288                         fprintf(fp, "          %d)\n", dd);
289                     else
290                         fprintf(fp, "          没有路径可达, 无穷大)\n");
291                 }
292             }
293         }
294     }
295 }
296
297

```

3.得出结果并分析。

## 四、实验结果：单位 (us)

Kruska 算法

Output:

```
result1.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
vernum: 8
arcnum: 11
runtime: 12.800000µs
4 0 2
2 1 4
2 3 5
0 5 6
0 1 11
5 7 11
2 6 12
total weight: 51
第 1 行, 第 1 列 100% Windows (CRLF) ANSI
```

```
result2.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
vernum: 64
arcnum: 660
runtime: 183.800000μs
1 34 1
35 0 1
7 14 1
20 12 1
29 49 1
25 16 1
31 62 1
49 17 1
35 11 1
59 30 1
7 29 1
38 13 1
34 18 1
35 58 1
20 38 1
43 18 1
56 60 1
61 3 1
33 28 1
48 44 1
41 45 1
5 43 1
31 55 1
61 11 1
17 25 1
第 1 行, 第 1 列 100% Windows (CRLF) ANSI
```

```
result3.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
vernum: 128
arcnum: 2742
runtime: 771.100000μs
24 66 1
33 81 1
102 77 1
4 62 1
44 84 1
112 17 1
69 103 1
40 113 1
55 31 1
55 97 1
86 106 1
56 16 1
62 74 1
100 126 1
62 45 1
44 1 1
86 12 1
74 36 1
7 29 1
58 101 1
114 5 1
114 72 1
66 23 1
122 93 1
65 115 1
第 1 行, 第 1 列 100% Windows (CRLF) ANSI
```



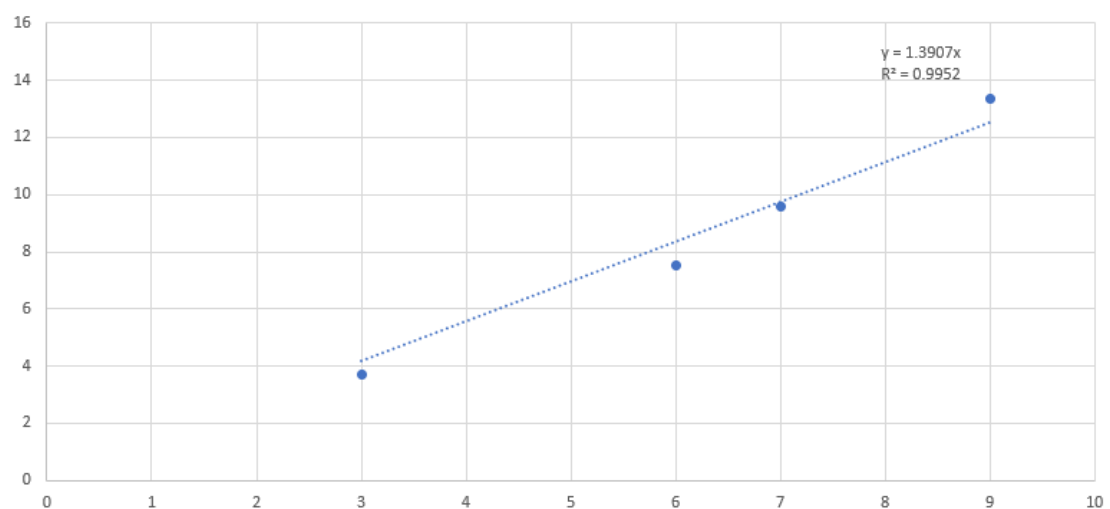
```
result4.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
vernum: 512
arcnum: 43326
runtime: 10372.100000μs
94 271 1
341 177 1
18 115 1
327 470 1
501 313 1
317 480 1
13 495 1
256 163 1
106 448 1
124 507 1
331 498 1
420 292 1
184 377 1
277 373 1
352 118 1
133 309 1
362 142 1
138 282 1
452 51 1
53 415 1
364 261 1
362 338 1
175 110 1
37 157 1
137 270 1
```

对运行时间进行拟合分析，将数据进行对数处理

3	3.678072
6	7.521993
7	9.590774
9	13.34042

得到如下拟合图

运行时间-规模拟合图



如图可以看出，数据基本拟合的很好，所以实际时间复杂度基本满足理论时间复杂度

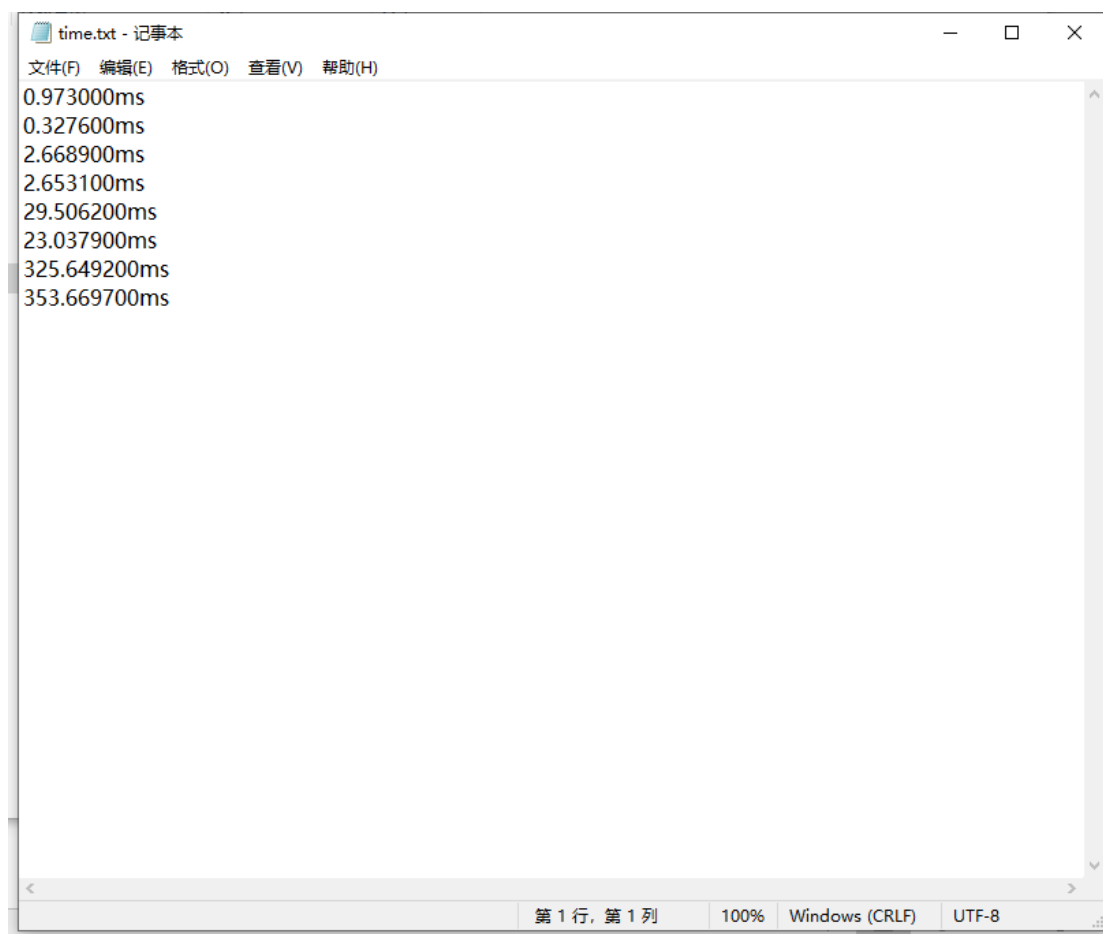
Johnson 算法:

Outout-result:

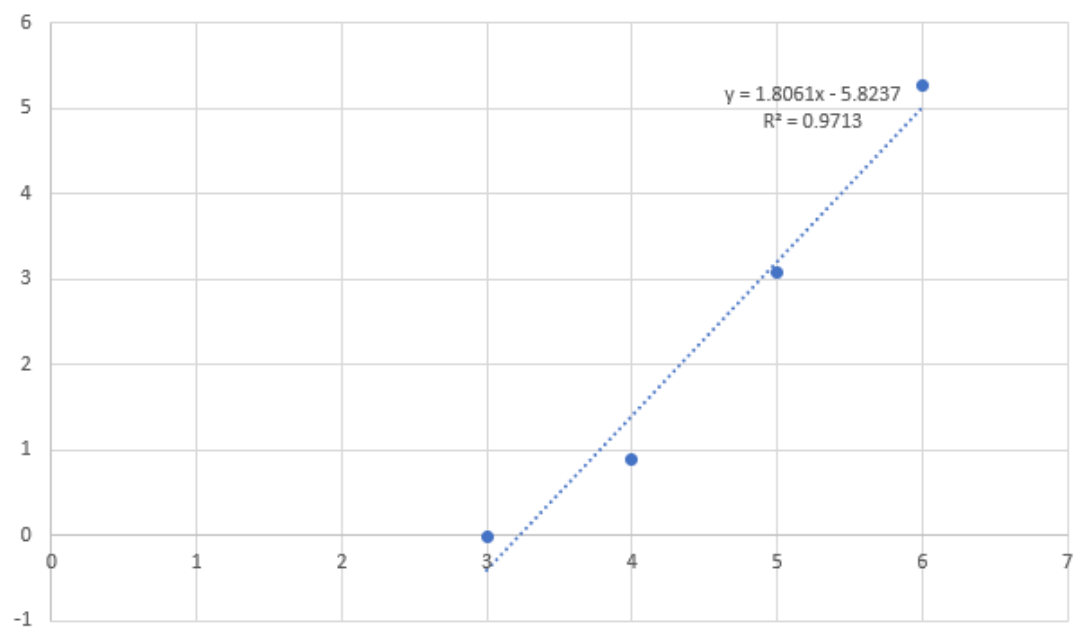
The screenshot shows a Windows file explorer window with the address bar set to "电脑 > 桌面 > 2020算法 > lab4 > ex2 > output". The left pane displays a list of files: result11.txt, result12.txt, result21.txt, result22.txt, result31.txt, result32.txt, result41.txt, result42.txt, and time.txt. The right pane shows the content of result11.txt, which is a text file named "result11.txt - 记事本". The file content consists of a series of lines, each representing a path and its weight, such as "(0,26,21,1 47)", "(0,2 没有路径可达, 无穷大)", "(0,19,12,3 71)", etc. The file size is indicated as 20.1 KB.

```
result11.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
(0,26,21,1 47)
(0,2 没有路径可达, 无穷大)
(0,19,12,3 71)
(0,19,12,3,4 92)
(0,19,18,7,10,5 135)
(0,26,25,16,6 112)
(0,19,18,7 69)
(0,8 没有路径可达, 无穷大)
(0,19,18,7,9 77)
(0,19,18,7,10 98)
(0,11 没有路径可达, 无穷大)
(0,19,12 63)
(0,13 没有路径可达, 无穷大)
(0,14 没有路径可达, 无穷大)
(0,19,18,15 79)
(0,26,25,16 73)
(0,26,25,16,17 103)
(0,19,18 36)
(0,19 13)
(0,26,21,1,20 89)
(0,26,21 16)
(0,26,25,16,6,22 157)
(0,23 没有路径可达, 无穷大)
(0,26,21,24 66)
(0,26,25 37)
(0,26 3)
(1,20,10,4,0 78)
(1,2 没有路径可达, 无穷大)
```

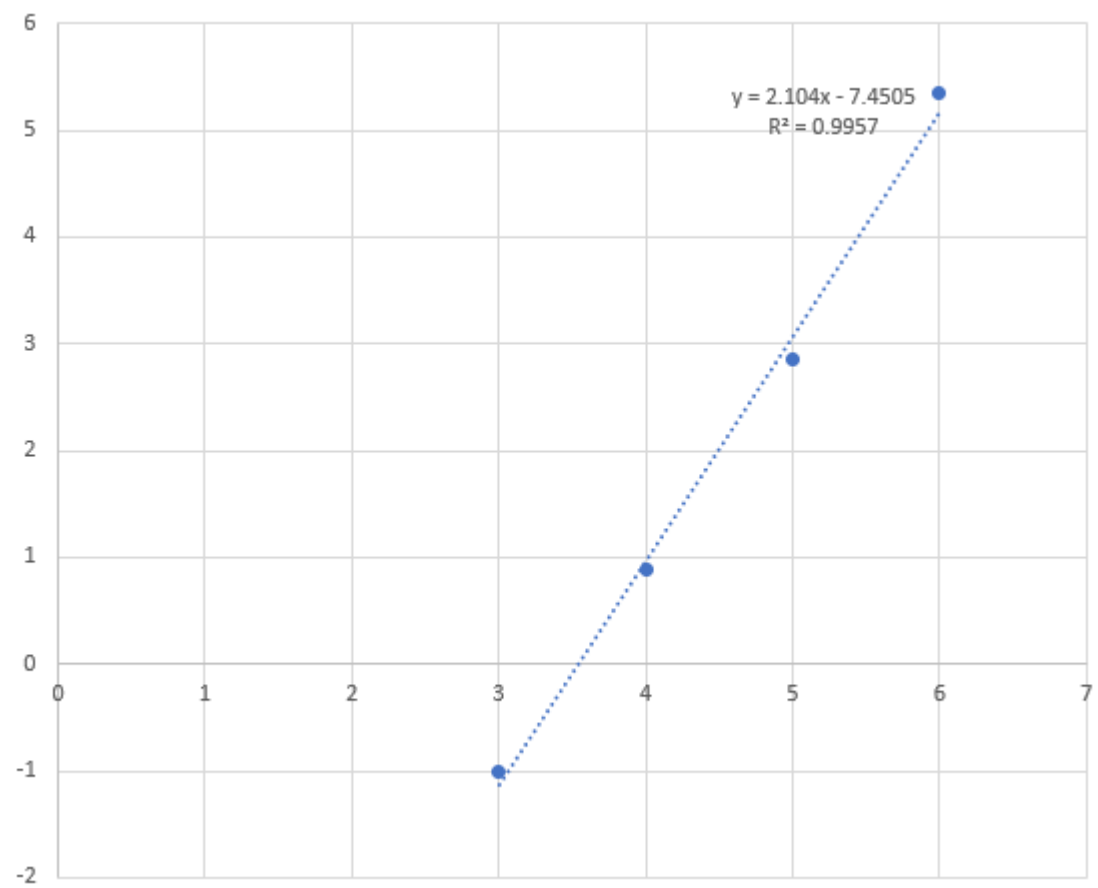
Output-time



运行时间-规模图 (5 为底)



运行时间-规模图 (7 为底)



数据也是经过对数处理之后得出的拟合图像，拟合的很好，说明实际时间复杂度基本符合理论时间复杂度