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~⌊N/2⌋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：log5N、log7N（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。)

二、实验环境

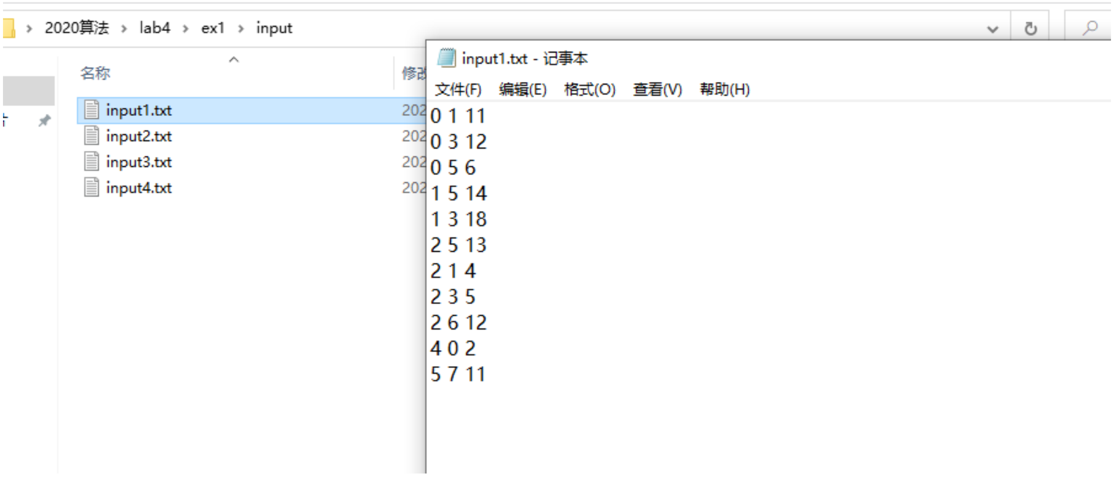
编译环境：DEV C++

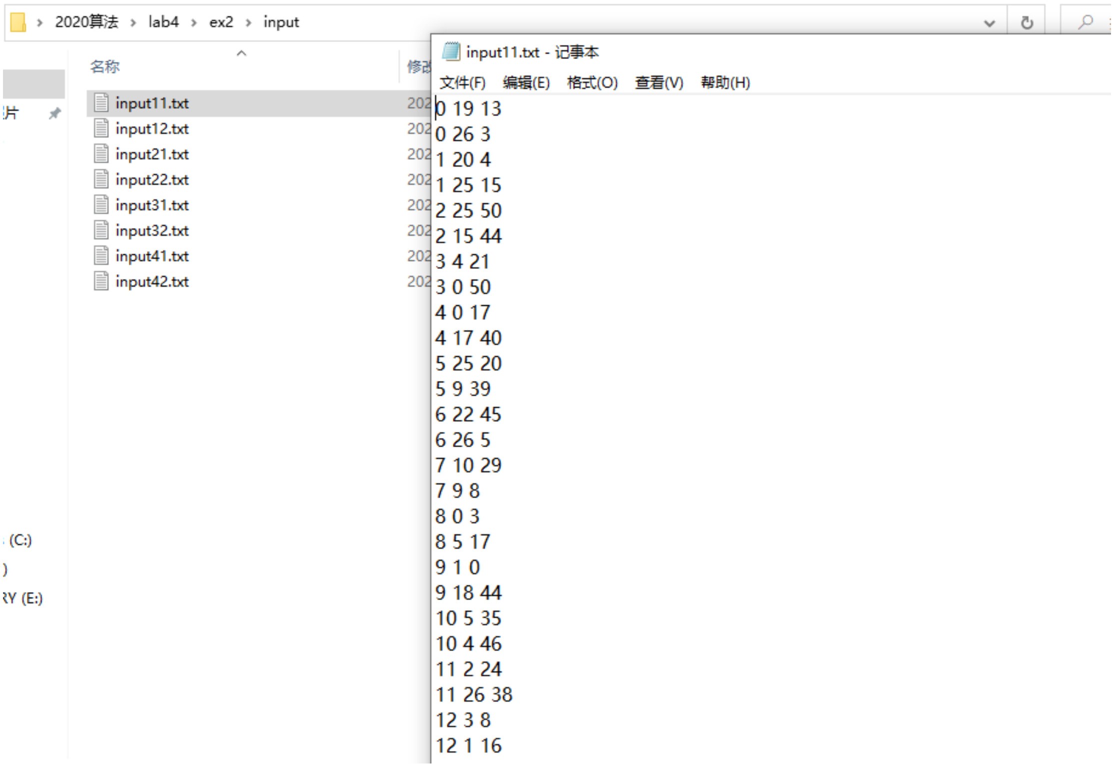
机器内存：16GB

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

```
plinklist kruskal(algraph G, edge* edges) {
    int i;
    pset* x;
    plinklist p, head, rear;
    head = rear = NULL;
    x = (pset*)malloc(G.vexnum * sizeof(pset));
    for (i = 0; i < G.vexnum; i++) {
        x[i] = (pset)malloc(sizeof(sset));
        x[i]->data = i;
        make_set(x[i]);
    }
    heap_sort(edges, G.arcnum);
    for (i = 0; i < G.arcnum; i++) {
        if (find_set(x[edges[i].v1]) != find_set(x[edges[i].v2]))
            p = (plinklist)malloc(sizeof(linklist));
            p->data = i;
            if (head == NULL) {
                head = p;
                rear = p;
            }
            else {
                rear->next = p;
                rear = rear->next;
            }
            union_set(x[edges[i].v1], x[edges[i].v2]);
        }
    }
    rear->next = NULL;
    return head;
}
```

```c
void johnson(algraph G, FILE* fp) {
    int i, j;
    int* h;
    int dd;
    algraph Gci;
    arcnode* pa, * ra = NULL;
    Gci.vexnum = G.vexnum + 1;
    Gci.arcnum = G.arcnum + G.vexnum;
    Gci.vertices = (vnode*)malloc(Gci.vexnum * sizeof(vnode));
    for (i = 1; i <= G.vexnum; i++)
        Gci.vertices[i] = G.vertices[i - 1];
    Gci.vertices[0].firstarc = NULL;
    for (i = 1; i <= G.vexnum; i++) {
        pa = (arcnode*)malloc(sizeof(arcnode));
        pa->adjvex = i;
        pa->w = 0;
        if (Gci.vertices[0].firstarc == NULL) {
            Gci.vertices[0].firstarc = pa;
            ra = pa;
        }
        else {
            ra->nextarc = pa;
            ra = ra->nextarc;
        }
    }
    ra->nextarc = NULL;
    h = (int*)malloc(G.vexnum * sizeof(int));
    if (bellman_ford(Gci, 0) == false)
        printf("有负环\n");
    else {
        for (i = 1; i < Gci.vexnum; i++)
            h[i - 1] = Gci.vertices[i].d;
        for (i = 0; i < G.vexnum; i++) {
            pa = G.vertices[i].firstarc;
```

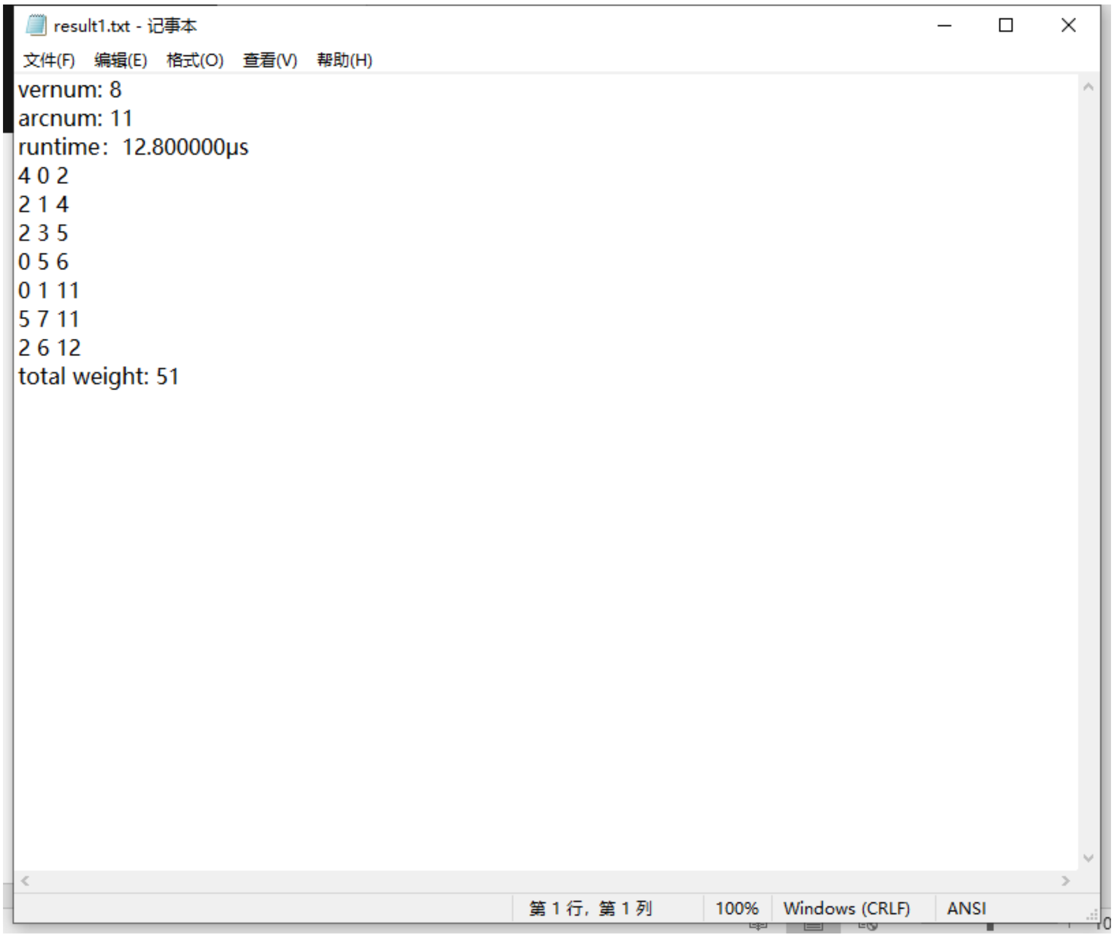
```c
    }
    ra->nextarc = NULL;
    h = (int*)malloc(G.vexnum * sizeof(int));
    if (bellman_ford(Gci, 0) == false)
        printf("有负环\n");
    else {
        for (i = 1; i < Gci.vexnum; i++)
            h[i - 1] = Gci.vertices[i].d;
        for (i = 0; i < G.vexnum; i++) {
            pa = G.vertices[i].firstarc;
            while (pa != NULL) {
                pa->w = pa->w + h[i] - h[pa->adjvex];
                pa = pa->nextarc;
            }
        }
        for (i = 0; i < G.vexnum; i++) {
            dijkstra(G, i);
            for (j = 0; j < G.vexnum; j++) {
                if (j != i) {
                    fprintf(fp, "(%d", i);
                    output(G, i, j, fp);
                    dd = G.vertices[j].d + h[j] - h[i];
                    if (dd < 9000000)
                        fprintf(fp, "        %d)\n", dd);
                    else
                        fprintf(fp, "        没有路径可达，无穷大)\n");
                }
            }
        }
    }
}
```

3.得出结果并分析。

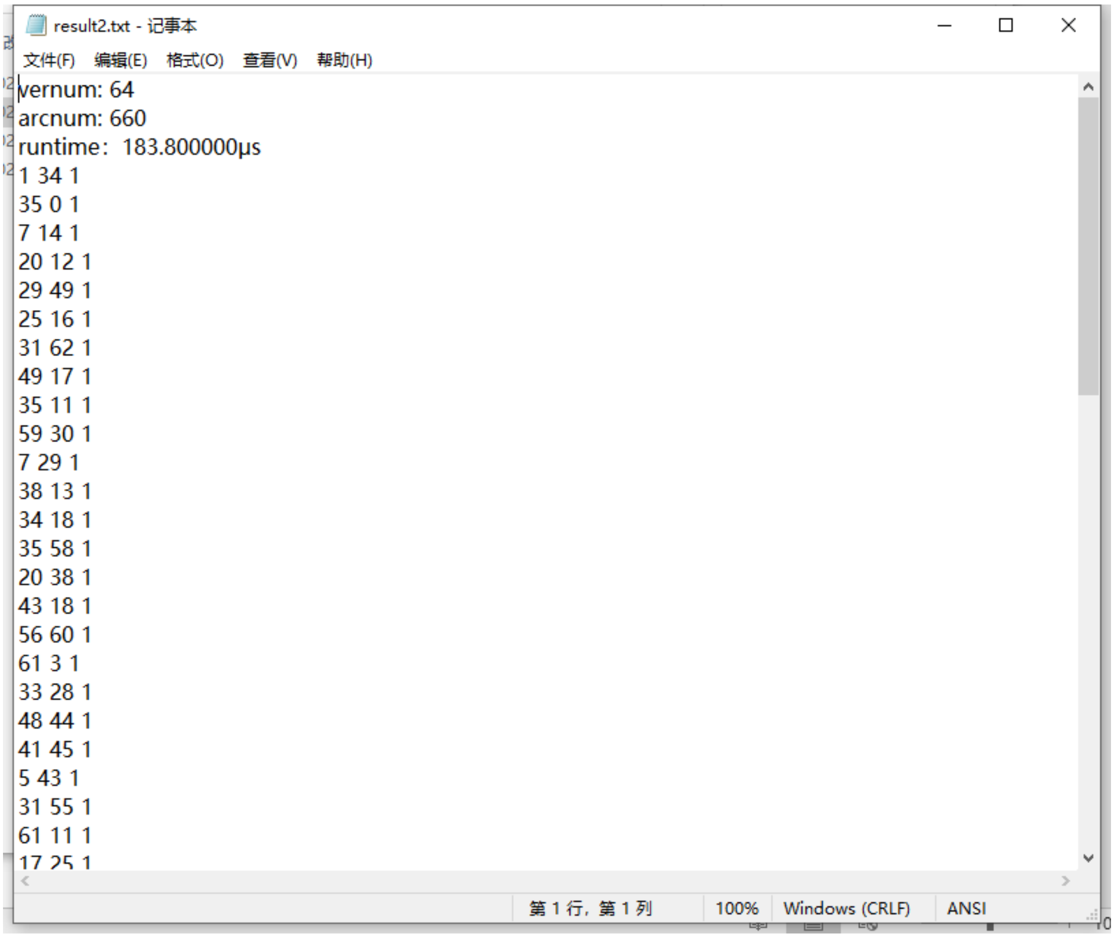
四、实验结果：单位（us）

Kruska 算法

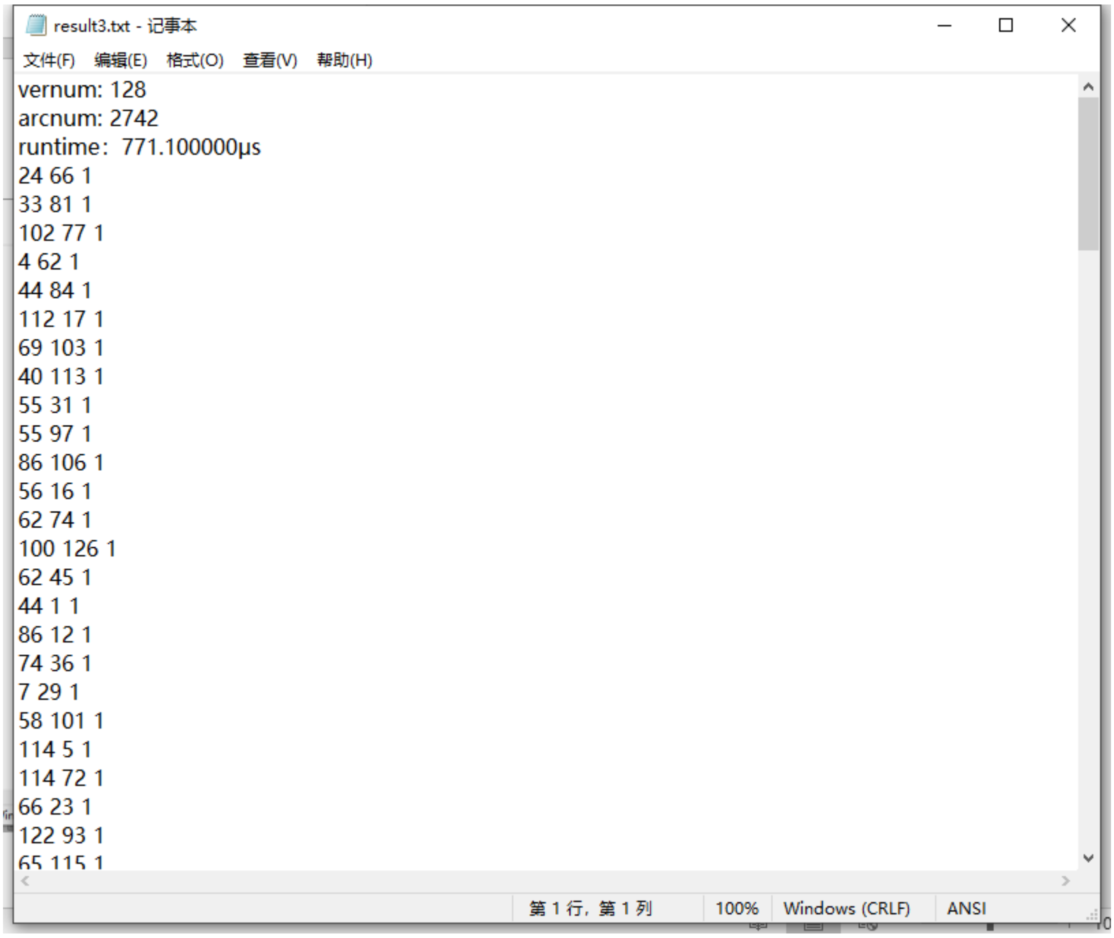
Output：




```
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
```




```
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
```



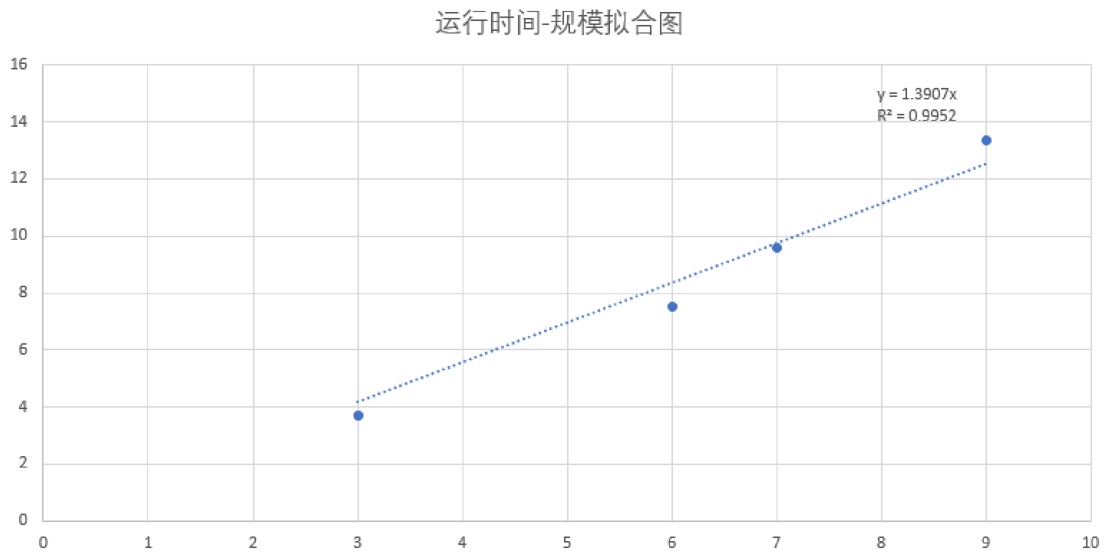
vernum: 128
arcnum: 2742
runtime: 771.100000μs
24 66 1
33 81 1
102 77 1
4 62 1
44 84 1
112 17 1
69 103 1
40 113 1
55 31 1
55 97 1
86 106 1
56 16 1
62 74 1
100 126 1
62 45 1
44 1 1
86 12 1
74 36 1
7 29 1
58 101 1
114 5 1
114 72 1
66 23 1
122 93 1
65 115 1

```
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
第 1 行，第 1 列    100%    Windows (CRLF)    ANSI
```

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

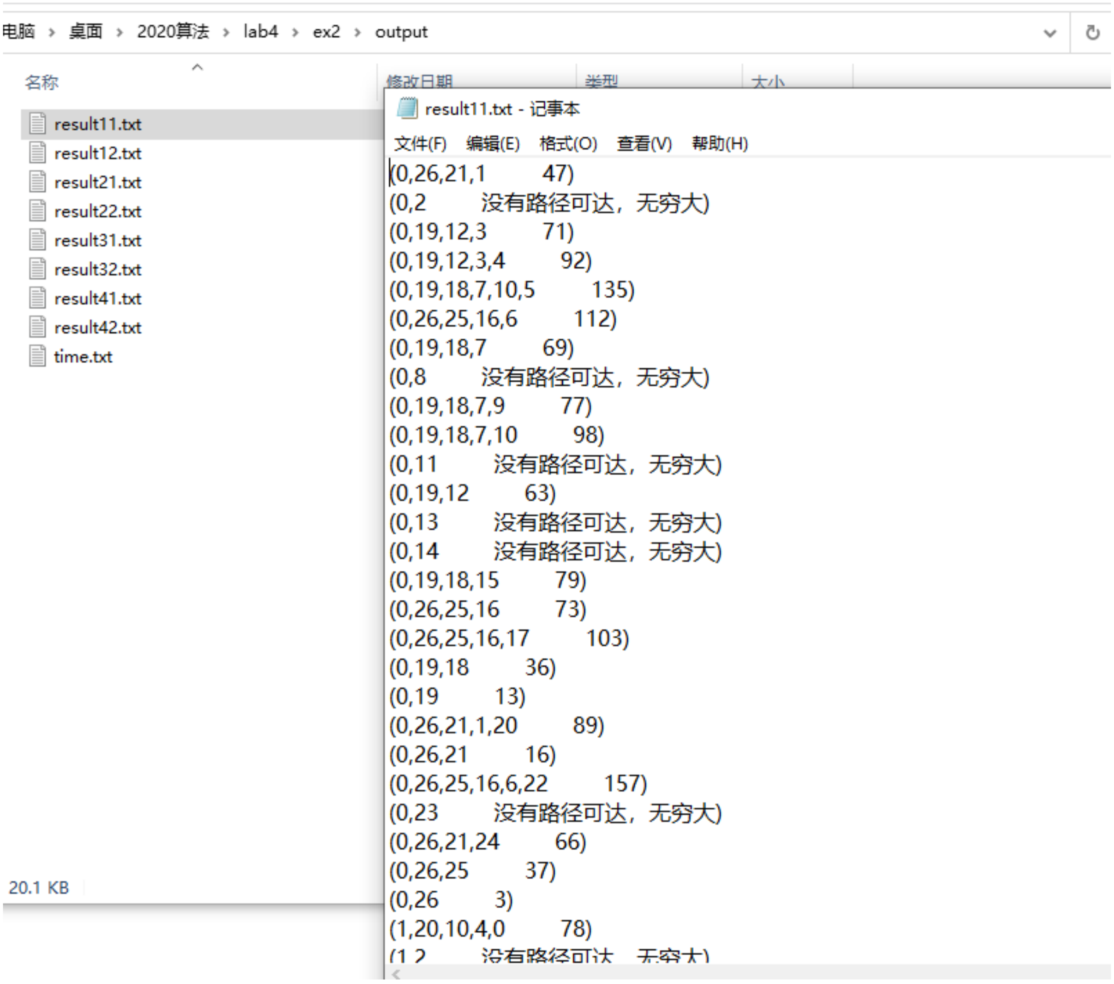
得到如下拟合图



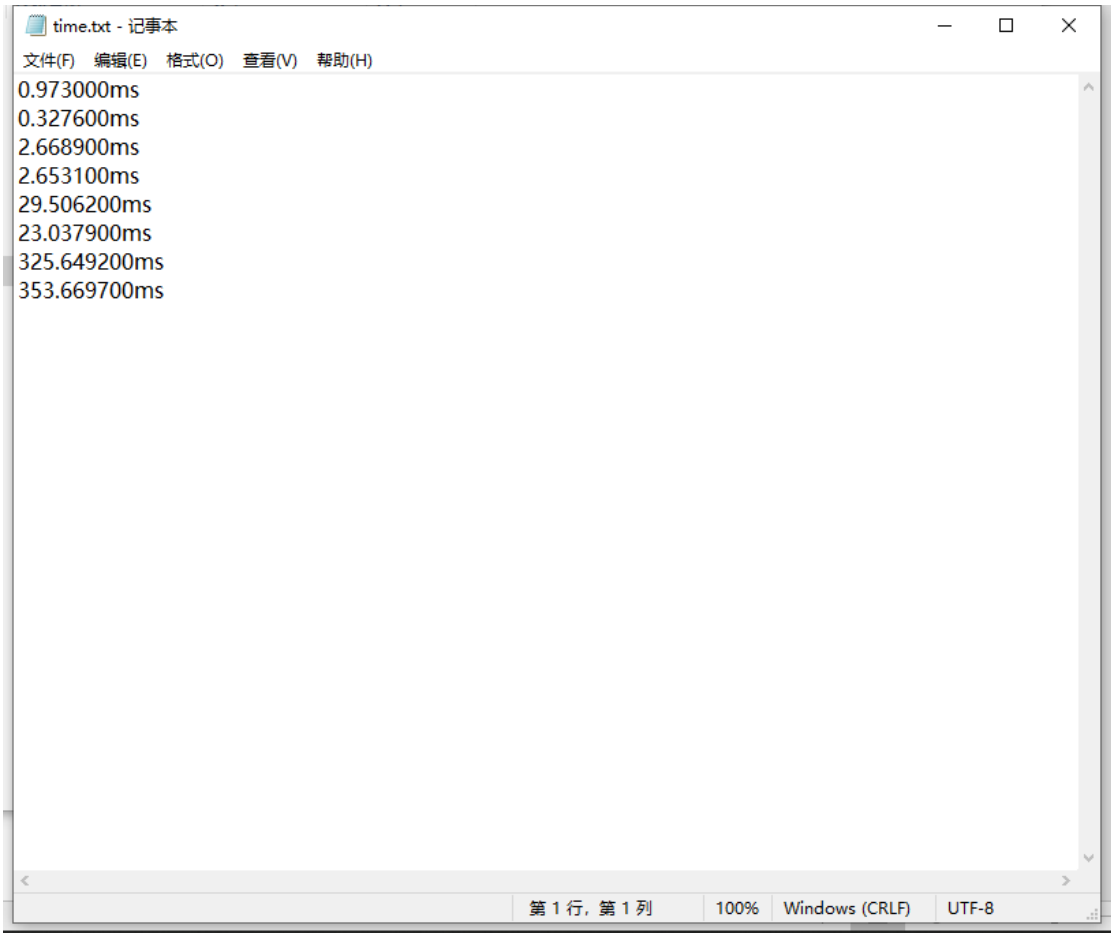
如图可以看出，数据基本拟合的很好，所以实际时间复杂度基本满足理论时间复杂度

Johnson 算法:

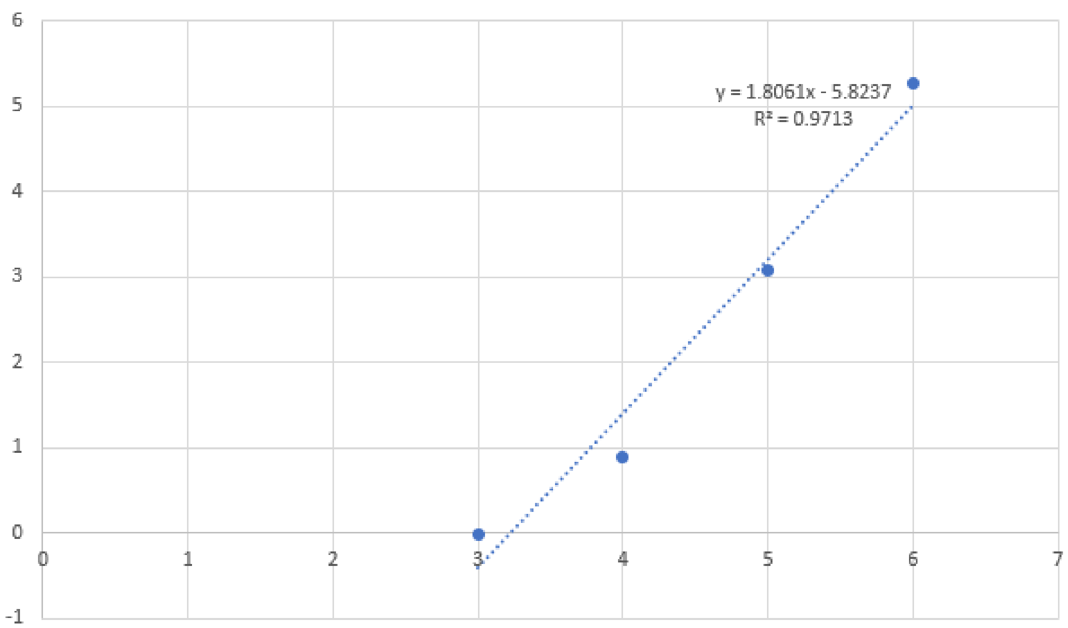
Outout-result:



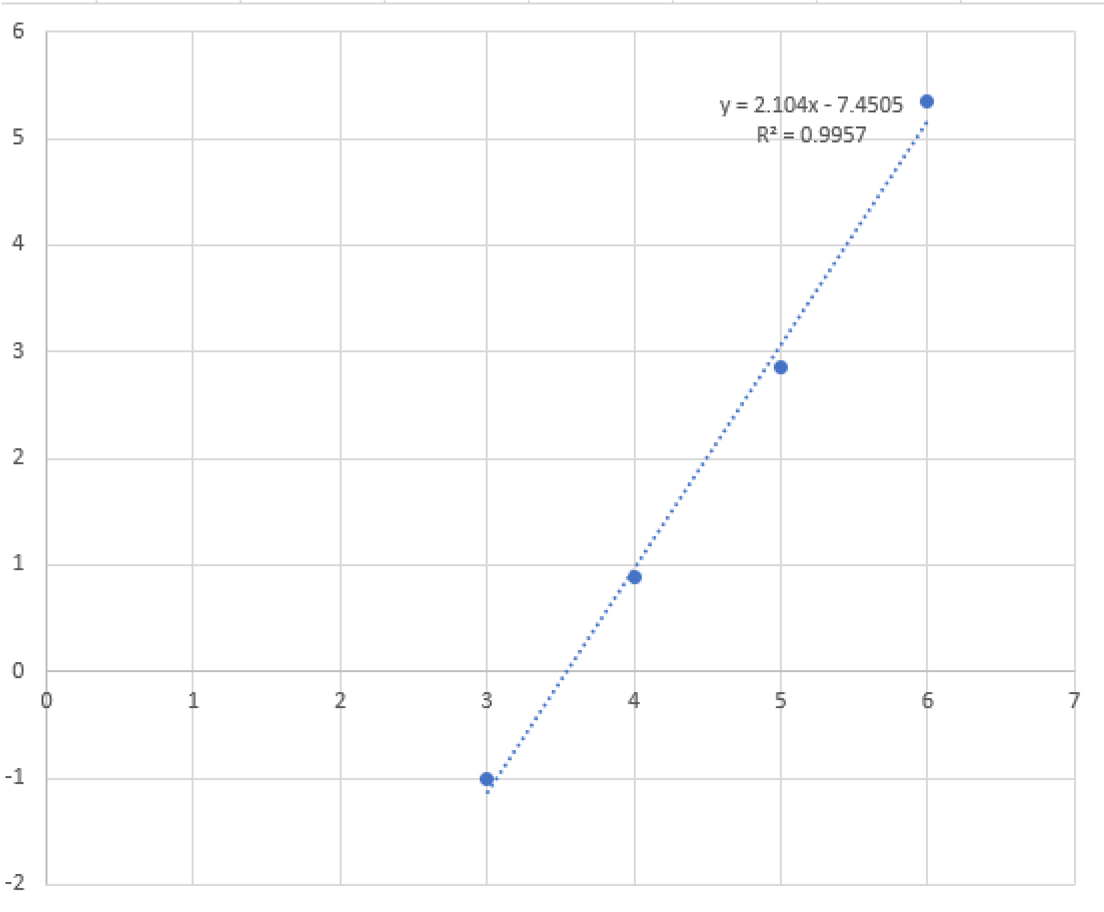
Output-time


time.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
0.973000ms
0.327600ms
2.668900ms
2.653100ms
29.506200ms
23.037900ms
325.649200ms
353.669700ms
第 1 行，第 1 列 100% Windows (CRLF) UTF-8

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