**南 京 师 范 大 学**

**《数据结构》**

**课程设计报告**



**题 目： 排序算法 最短路径**

**学 院： 计算机与电子信息学院**

**专 业： 计算机科学与技术**

**姓 名： 黄艺杰**

**学 号： 21190630**

**任课教师： 吉根林**

**计算机与电子信息学院 制**

**二O二二年 三 月 四 日**

目录

[1 排序算法 1](#_Toc9151)

[1.1 要求 1](#_Toc24743)

[1.2 算法思想 1](#_Toc23180)

[1.3 四种排序方法性能的比较 5](#_Toc16564)

[1.4 程序设计 6](#_Toc8613)

[1.5 程序测试 13](#_Toc22467)

[2 最短路径 15](#_Toc12556)

[2.1 要求 15](#_Toc24309)

[2.2 Dijkstra算法 15](#_Toc25051)

[2.3 程序设计 16](#_Toc17787)

[2.4 程序测试 20](#_Toc24335)

[3 收获与体会 23](#_Toc26788)

# 1 排序算法

### 1.1 要求

编程实现希尔、快速、堆排序、归并排序算法。要求首先随机产生10000个数据存入磁盘文件，然后读入数据文件，分别采用不同的排序方法进行排序并将结果存入文件。

### 1.2 算法思想

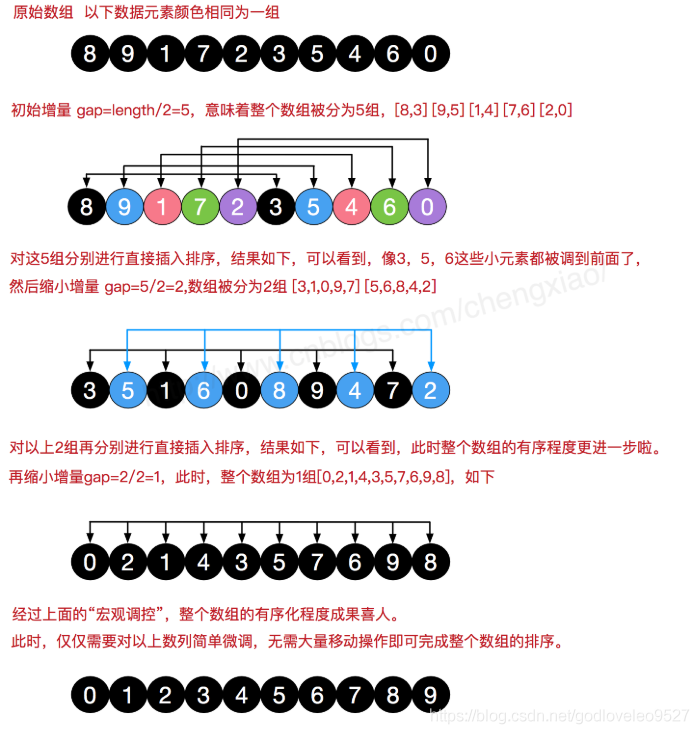
#### 1.2.1 希尔排序

希尔排序是对直接插入排序的一种改进，它利用了插入排序的两个性质：

（1）若待排记录按关键字基本有序，则直接插入排序效率很高；

（2）若待排记录个数很少，则直接插入排序效率也较高。

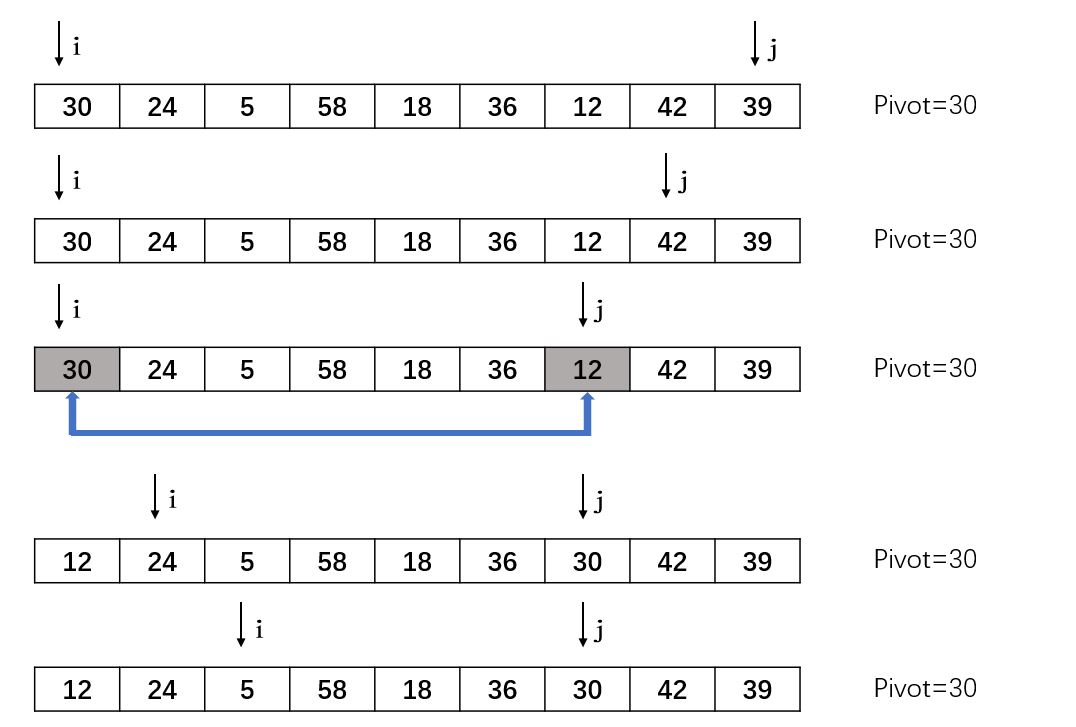
因此希尔排序先将待排序列分为若干个小序列，造这些小序列中进行插入排序；然后逐步扩大小序列的长度，减少小序列的个数，这样使待排序列逐渐处于更有序的状态；最后对全体序列进行一次直接插入排序，从而完成排序。

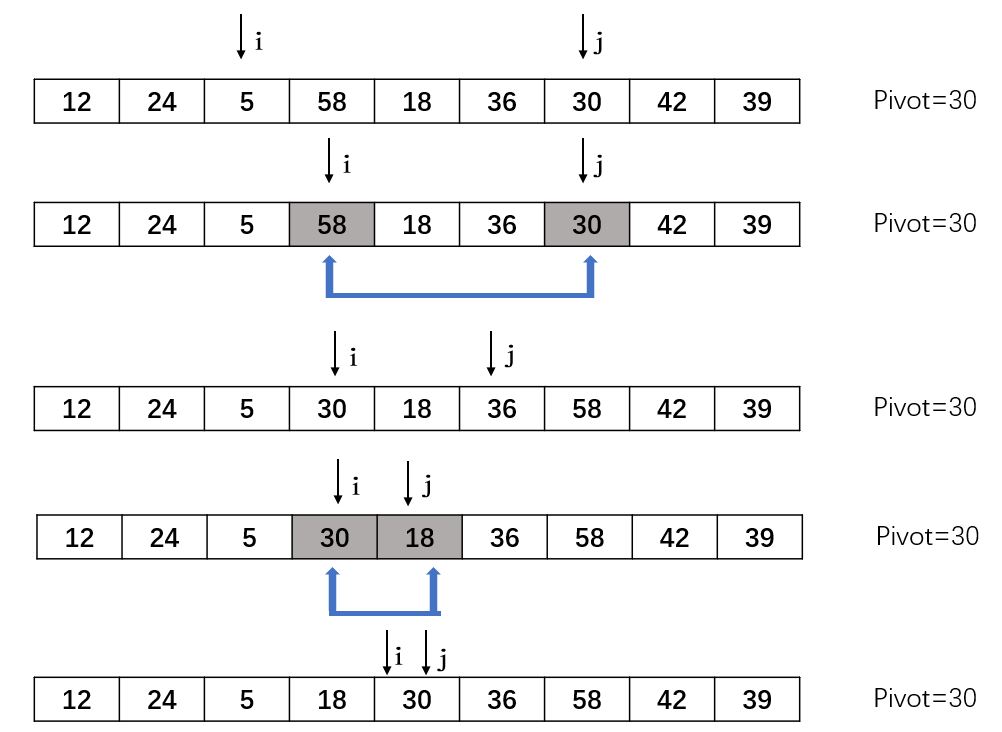


#### 1.2.2 快速排序

1962年，伦敦Elliot Brothers Ltd公司的Tony Hoare发明了快速排序。实际上快速排序名副其实，它几乎是最快的排序算法，被评为20世纪十大算法之一。

快速排序算法的基本思想是从待排序记录序列中选取一个记录（通常选取第一个记录）为枢轴，其关键字值设为k。将关键字值小于k的记录移到前面，而将关键字值大于k的记录移到后面。结果将待排序记录序列分为两个子表，最后将关键字值为k的记录插到分界线处。将这个过程称为划分。通过一次划分后，以关键字值k为基准，将待排序序列分成了两个子表。前面子表中所有记录的关键字值均不大于k，后面子表中所有记录的关键值均不小于k。对划分后的子表继续按上述原则进行划分，直到所有子表的表长不超过1，此时待排序记录序列就变成了一个有序序列。





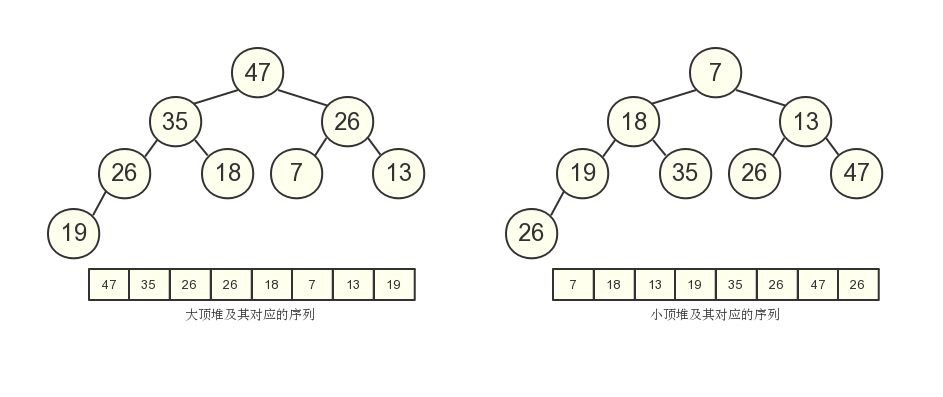
#### 1.2.3 堆排序

堆排序是J. W. J. Williams于1964年提出的。

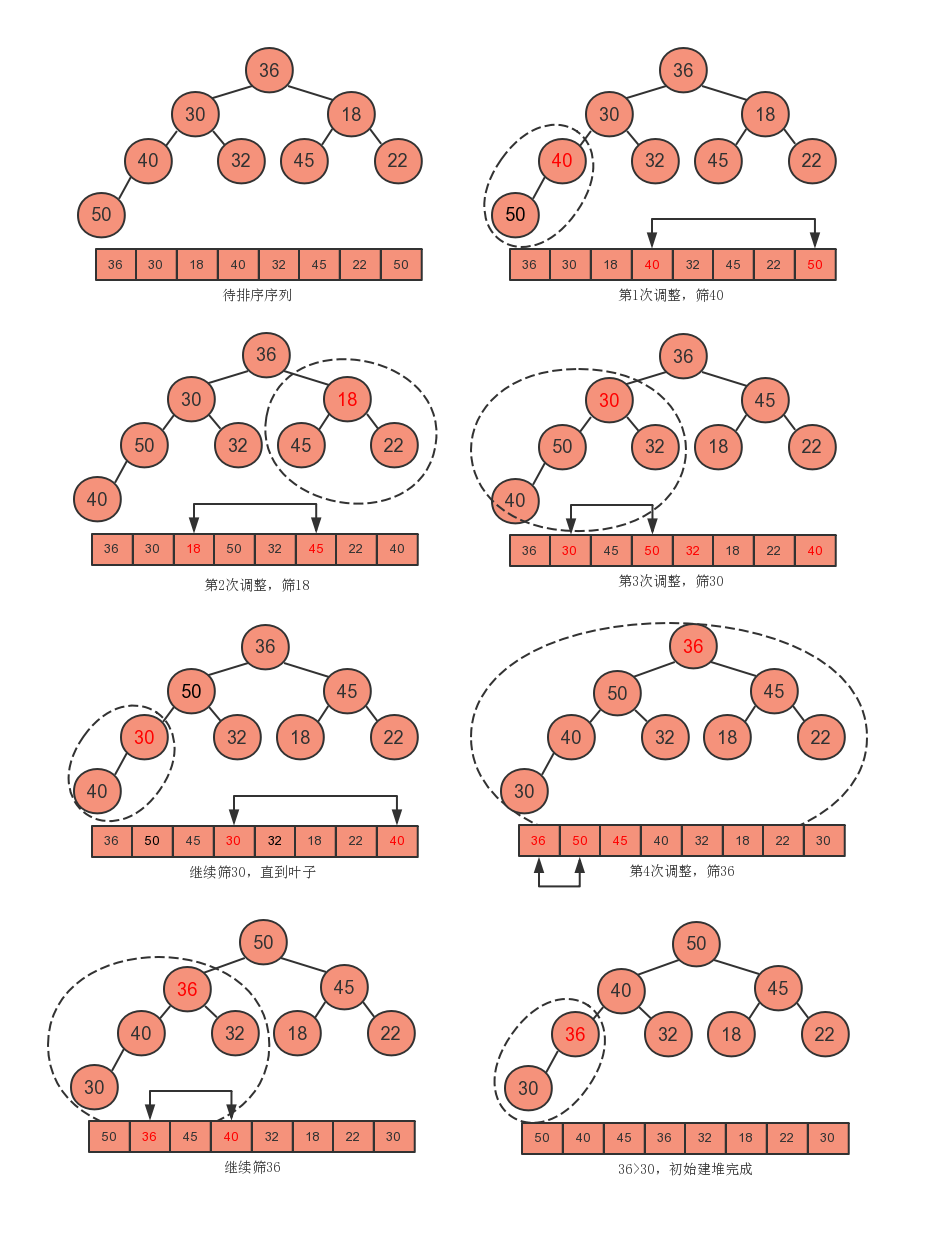
堆是具有下列性质的完全二叉树：每个结点的值都小于或等于其左、右孩子结点的值（小顶堆），或者每个结点的值都大于或等于其左、右孩子结点的值（大顶堆）。

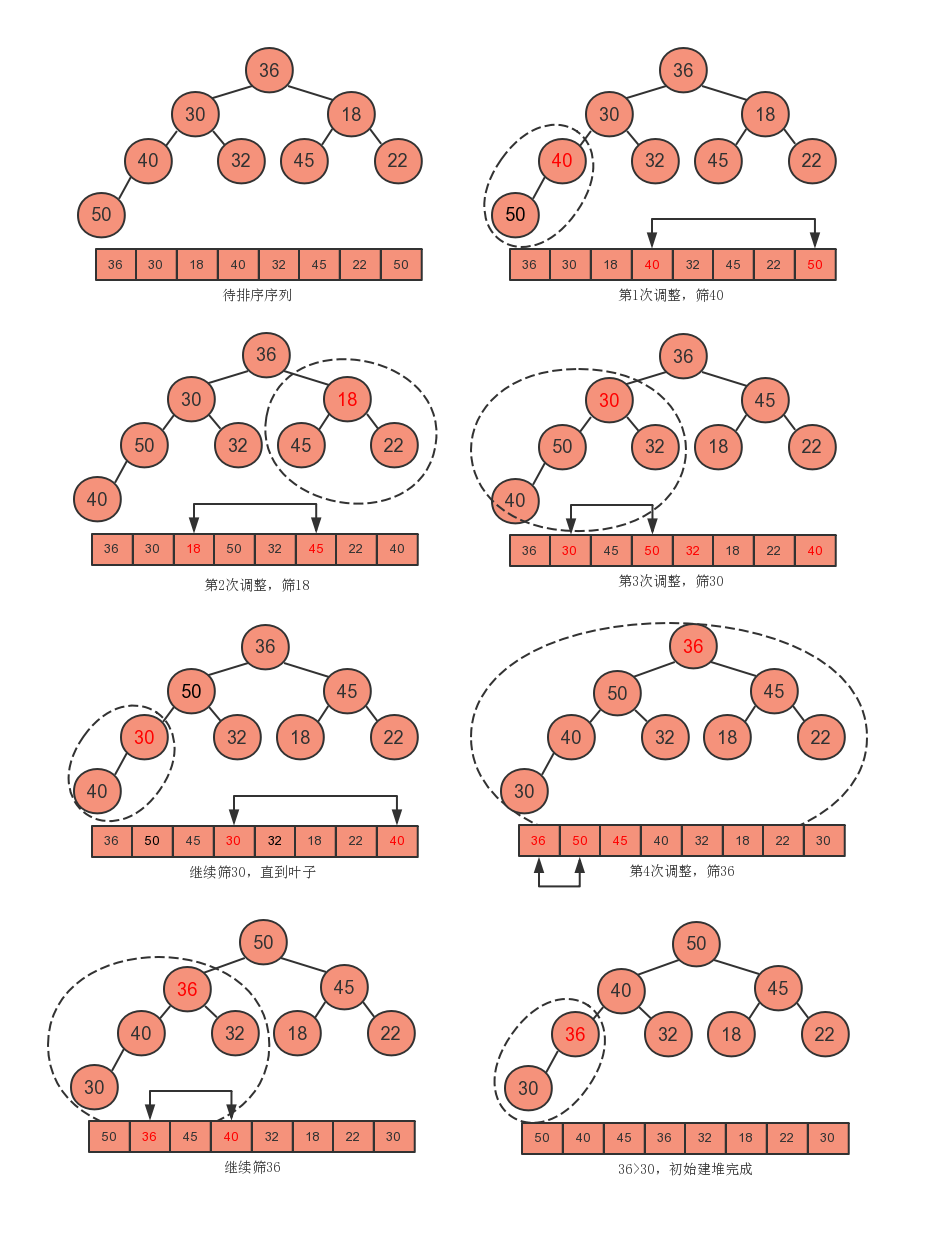
如果将堆按层从上到下，每层从左到右对每个结点从1开始进行编号，那么结点之间满足如下关系：

或



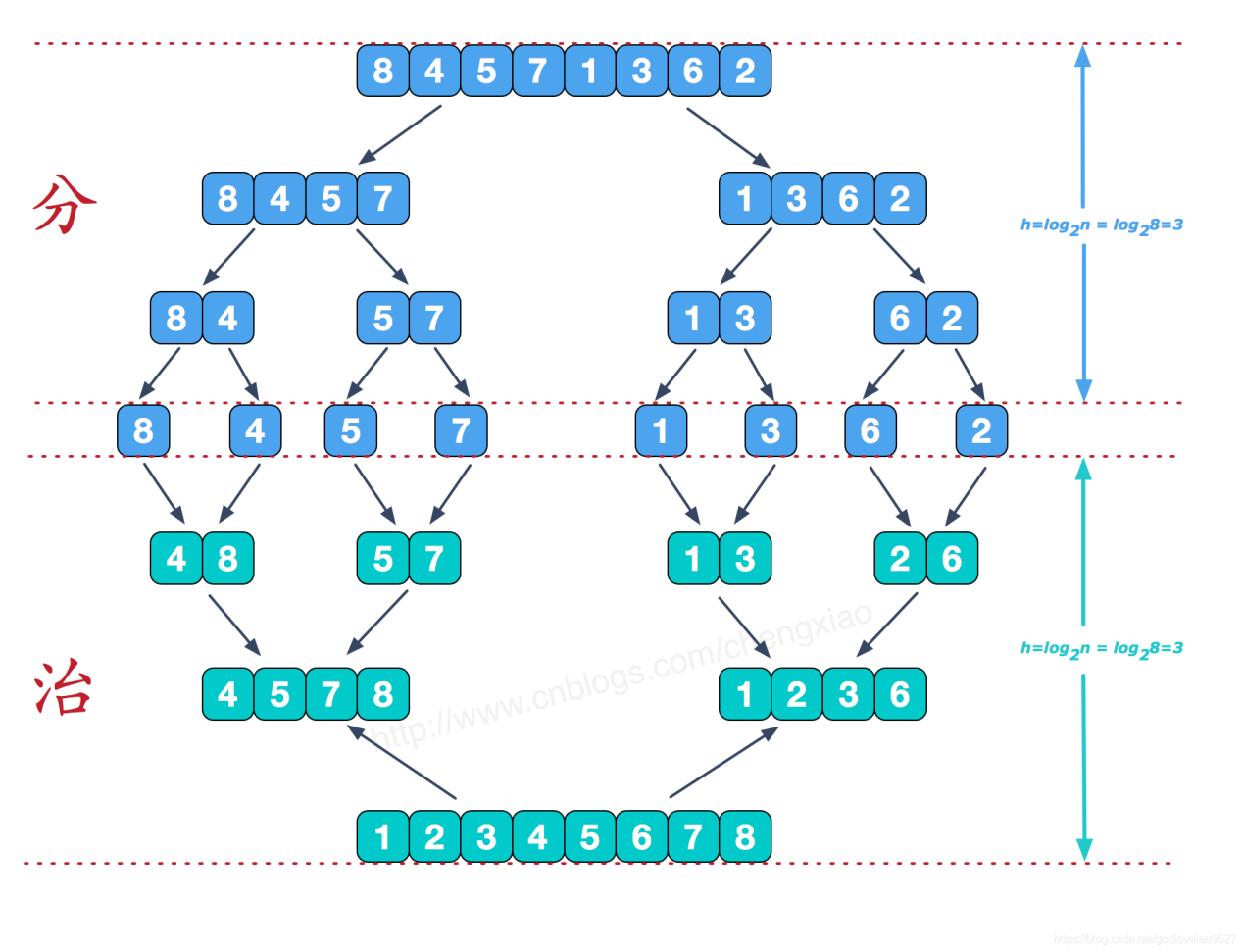
堆排序利用堆（假设利用大顶堆）的特性进行排序的方法，其基本思想是首先用待排序的记录序列构造成一个堆，此时选出堆中所有记录的最大者（如图所示），即堆顶记录；然后将它从堆中移走（通常将堆顶记录和堆中最后一个记录交换），并将剩余的记录再调整成堆，这样又找出了次大的记录，依次类推，直到堆中只有一个记录。





#### 1.2.4 归并排序

二路归并排序时归并排序中最简单的排序方法，其基本思想是将若干有序序列进行两两归并，直至所有待排序记录为一个有序序列。



### 1.3 四种排序方法性能的比较

四种排序的时间复杂度和空间复杂度的比较结果如下图所示

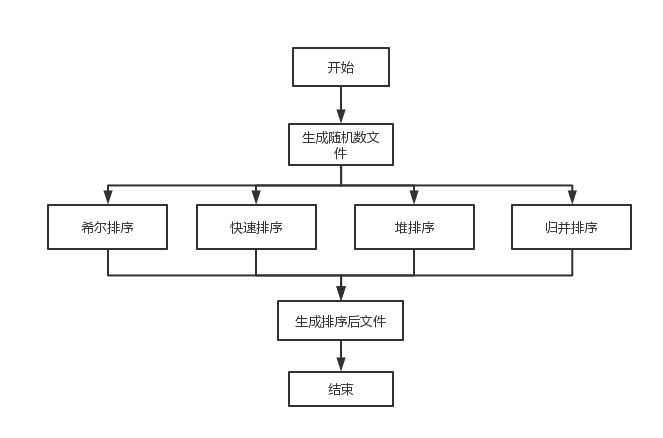
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 排序方法 | 平均情况的  时间复杂度 | 最好情况的  时间复杂度 | 最坏情况的  时间复杂度 | 空间复杂度 |
| 希尔排序 |  |  |  |  |
| 快速排序 |  |  |  |  |
| 堆排序 |  |  |  |  |
| 归并排序 |  |  |  |  |

从稳定性来看，归并排序是稳定的，希尔排序、快速排序和堆排序是不稳定的。

当待排序记录个数n较大，关键字分布较随机，且对稳定性不作要求时，采用快速排序；当待排序记录个数n较大，内存空间允许，且要求排序稳定时，采用归并排序；当待排记录个数n较大，而只要找出最小的前几个记录时，采用堆排序；当带排序记录个数n较小，记录已基本有序时，采用希尔排序。

### 1.4 程序设计

#### 1.4.1 设计过程



1、编写Sort.h文件，里面包含希尔排序，快速排序、堆排序、归并排序的算法实现。

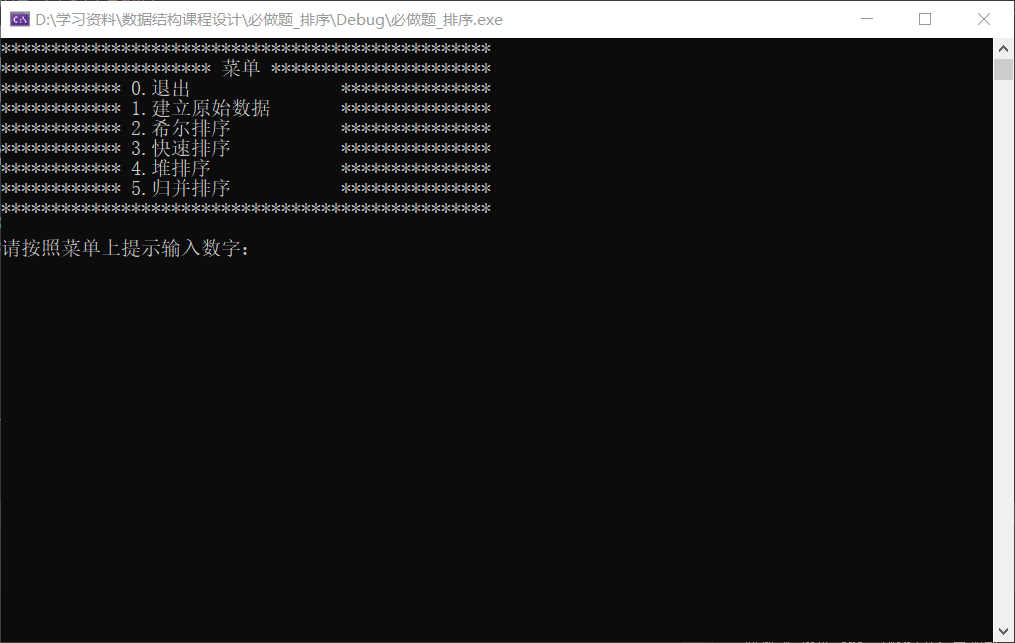
1. #pragma once
2. #include <iostream>
3. #include <time.h>
4. #include <windows.h>
5. **using** **namespace** std;
7. //希尔排序算法
8. **void** ShellSort(**int** r[], **int** n)
9. {
10. **for** (**int** d = n / 2; d >= 1; d = d / 2)
11. {
12. **for** (**int** i = d; i < n; i++)
13. {
14. **int** temp = r[i];
15. **int** j;
17. **for** (j = i - d; j >= 0 && temp < r[j]; j = j - d)
18. r[j + d] = r[j];
19. r[j + d] = temp;
20. }
21. }
22. }
24. //快速排序
25. **int** Partition(**int** r[], **int** i, **int** j)
26. {
27. **int** temp = r[i];
28. **while** (i < j)
29. {
30. **while** (i < j && r[j] >= temp)
31. j--;
32. **if** (i < j)
33. r[i++] = r[j];
34. **while** (i < j && r[i] <= temp)
35. i++;
36. **if** (i < j)
37. r[j--] = r[i];
38. }
39. r[i] = temp;
40. **return** i;
41. }
43. **void** QuickSort(**int**  r[], **int** i, **int** j)
44. {
45. **if** (i < j)
46. {
47. **int** pivot = Partition(r, i, j);
48. QuickSort(r, i, pivot - 1);
49. QuickSort(r, pivot + 1, j);
50. }
51. }
53. //堆排序中的筛选算法
54. **void** Sift(**int**  r[], **int** k, **int** m)
55. {
56. **int** i = k;
57. **int** j = 2 \* i + 1;
59. **while** (j <= m)
60. {
61. **if** (j < m && r[j] < r[j + 1])
62. j++;
63. **if** (r[i] > r[j])
64. **break**;
65. **else**
66. {
67. **int** temp = r[i];
68. r[i] = r[j];
69. r[j] = temp;
70. i = j;
71. j = i \* 2 + 1;
72. }
73. }
74. }
76. //堆排序算法
77. **void** HeapSort(**int** r[], **int** n)
78. {
79. **for** (**int** i = n / 2 - 1; i >= 0; i--)
80. Sift(r, i, n - 1);
81. **for** (**int** i = 1; i < n; i++)
82. {
83. **int** temp = r[0];
84. r[0] = r[n - i];
85. r[n - i] = temp;
86. Sift(r, 0, n - i - 1);
87. }
88. }
90. //一次归并算法
91. **void** Merge(**int** r[], **int** r1[], **int** s, **int** m, **int** t)
92. {
93. **int** i = s;
94. **int** j = m + 1;
95. **int** k = s;
97. **while** (i <= m && j <= t)
98. **if** (r[i] <= r[j])
99. r1[k++] = r[i++];
100. **else**
101. r1[k++] = r[j++];
102. **if** (i <= m)
103. **while** (i <= m)
104. r1[k++] = r[i++];
105. **else**
106. **while** (j <= t)
107. r1[k++] = r[j++];
108. }
110. //一趟归并排序算法
111. **void** MergePass(**int** r[], **int** r1[], **int** n, **int** h)
112. {
113. **int** i = 0;
115. **while** (i <= n - 2 \* h + 1)
116. {
117. Merge(r, r1, i, i + h - 1, i + 2 \* h - 1);
118. i += 2 \* h;
119. }
120. **if** (i < n - h + 1)
121. Merge(r, r1, i, i + h - 1, n);
122. **else**
123. **for** (**int** k = i; k <= n; k++)
124. r1[k] = r[k];
125. }
127. //归并排序的非递归算法
128. **void** MergeSort(**int** r[], **int** n)
129. {
130. **int** h = 1;
131. **int**\* r1 = **new** **int**[n];
133. **while** (h < n)
134. {
135. MergePass(r, r1, n - 1, h);
136. h = 2 \* h;
137. MergePass(r1, r, n - 1, h);
138. h = 2 \* h;
139. }
140. }

2、编写主函数，设计功能菜单

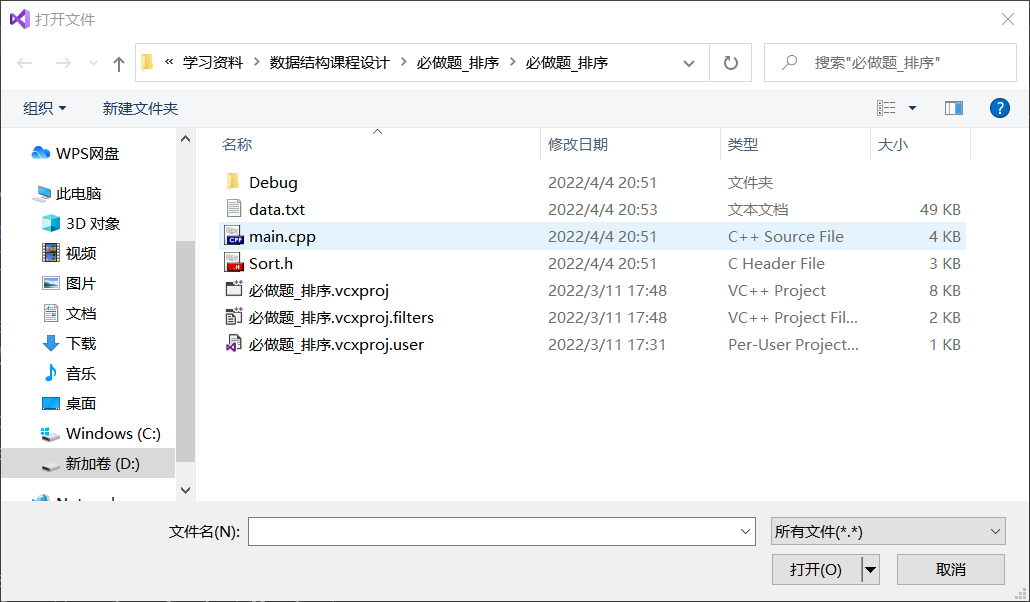
1. #include <iostream>
2. #include <fstream>
3. #include <time.h>
4. #include "Sort.h"
5. #define N 10000
6. **using** **namespace** std;
7. **void** SetData()
8. {
9. fstream fout("data.txt", ios::out);
10. **if** (!fout)
11. {
12. cout << "无法打开文件！" << endl;
13. exit(1);
14. }
15. **else**
16. {
17. srand((unsigned)time(NULL));
18. **for** (**int** i = 0; i < N; i++)
19. {
20. **int** m = rand() % N;
21. fout << m << " ";
22. **if** ((i + 1) % 20 == 0)
23. fout << endl;
24. }
25. cout << "10000个随机数生成完毕，并已储存到data.txt文件里" << endl;
26. }
27. fout.close();
28. }
30. //写入文件
31. **void** WriteFile(**int** r[], **int** n, **const** **char**\* filename)
32. {
33. fstream fout(filename, ios::out);
34. **if** (!fout)
35. {
36. cout << "无法打开文件！" << endl;
37. exit(1);
38. }
39. **else**
40. {
41. **for** (**int** i = 0; i < n; i++)
42. {
43. fout << r[i] << " ";
44. **if** ((i + 1) % 15 == 0)
45. fout << endl;// '\n';
46. }
47. }
48. fout.close();
49. }
51. //读取文件
52. **int**\* ReadFile(**int** n, **const** **char**\* filename)
53. {
54. fstream fin(filename, ios::in);
55. **int**\* s = **new** **int**[n];
57. **if** (!fin)
58. {
59. cout << "无法打开文件！" << endl;
60. exit(1);
61. }
62. **else**
63. {
64. **for** (**int** i = 0; i < n; i++)
65. fin >> s[i];
66. }
67. fin.close();
68. **return** s;
69. }
71. **void** Menu()
72. {
73. **while** (1)
74. {
75. system("cls");
76. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
77. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 菜单 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
78. cout << "\*\*\*\*\*\*\*\*\*\*\*\* 0.退出               \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
79. cout << "\*\*\*\*\*\*\*\*\*\*\*\* 1.建立原始数据       \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
80. cout << "\*\*\*\*\*\*\*\*\*\*\*\* 2.希尔排序           \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
81. cout << "\*\*\*\*\*\*\*\*\*\*\*\* 3.快速排序           \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
82. cout << "\*\*\*\*\*\*\*\*\*\*\*\* 4.堆排序             \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
83. cout << "\*\*\*\*\*\*\*\*\*\*\*\* 5.归并排序           \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
84. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
85. cout << endl;
86. cout << "请按照菜单上提示输入数字：" << endl;
87. **int** choose = 0;
88. cin >> choose;
89. cout << endl;
90. **switch** (choose)
91. {
92. **case** 0:
93. exit(0);
94. **case** 1:
95. SetData();
96. **break**;
97. **case** 2:
98. {
99. **int**\* r = ReadFile(N, "data.txt");
100. ShellSort(r, N);
101. WriteFile(r, N, "ShellSort.txt");
102. cout << "希尔排序执行完毕，结果已存储到ShellSort.txt中" << endl;
103. **break**;
104. }
105. **case** 3:
106. {
107. **int**\* r = ReadFile(N, "data.txt");
108. QuickSort(r, 0, N - 1);
109. WriteFile(r, N, "QuickSort.txt");
110. cout << "快速排序执行完毕，结果已经存储到QuickSort.txt中" << endl;
111. **break**;
112. }
113. **case** 4:
114. {
115. **int**\* r = ReadFile(N, "data.txt");
116. HeapSort(r, N);
117. WriteFile(r, N, "HeapSort.txt");
118. cout << "堆排序执行完毕，结果已经存储到HeapSort.txt中" << endl;
119. **break**;
120. }
121. **case** 5:
122. {
123. **int**\* r = ReadFile(N, "data.txt");
124. MergeSort(r, N);
125. WriteFile(r, N, "MergeSort.txt");
126. cout << "归并排序执行完毕，结果已经存储到MergeSort.txt中" << endl;
127. **break**;
128. }
130. }
131. }
132. }
134. **int** main()
135. {
136. Menu();
137. **return** 0;
138. }

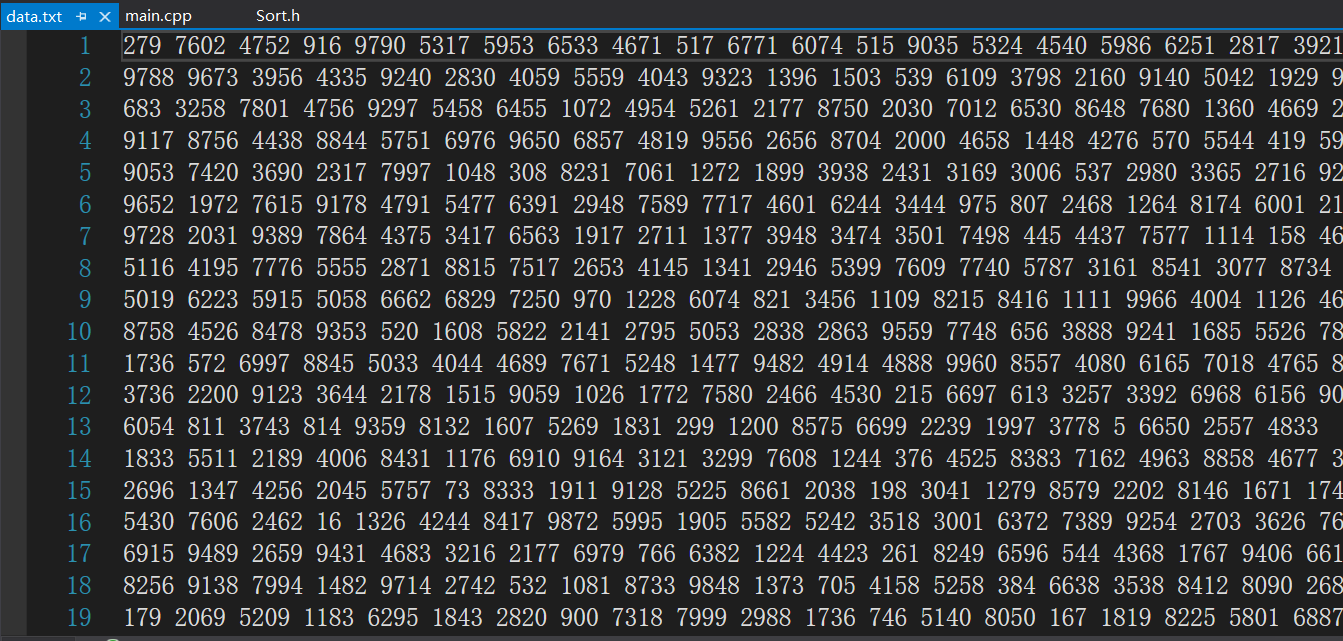
### 1.5 程序测试

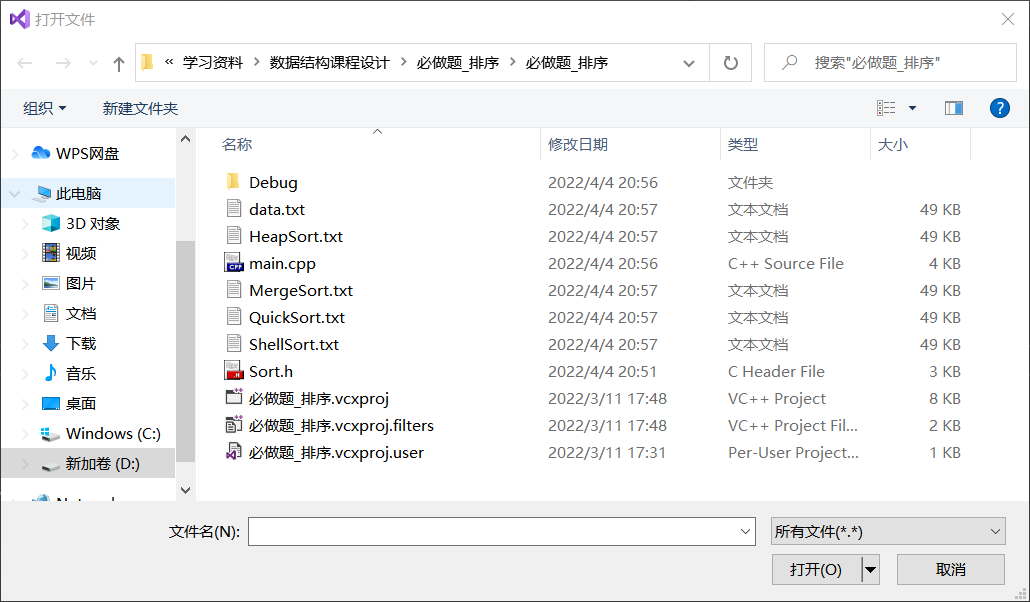
打开程序后点击运行，弹出如下菜单：

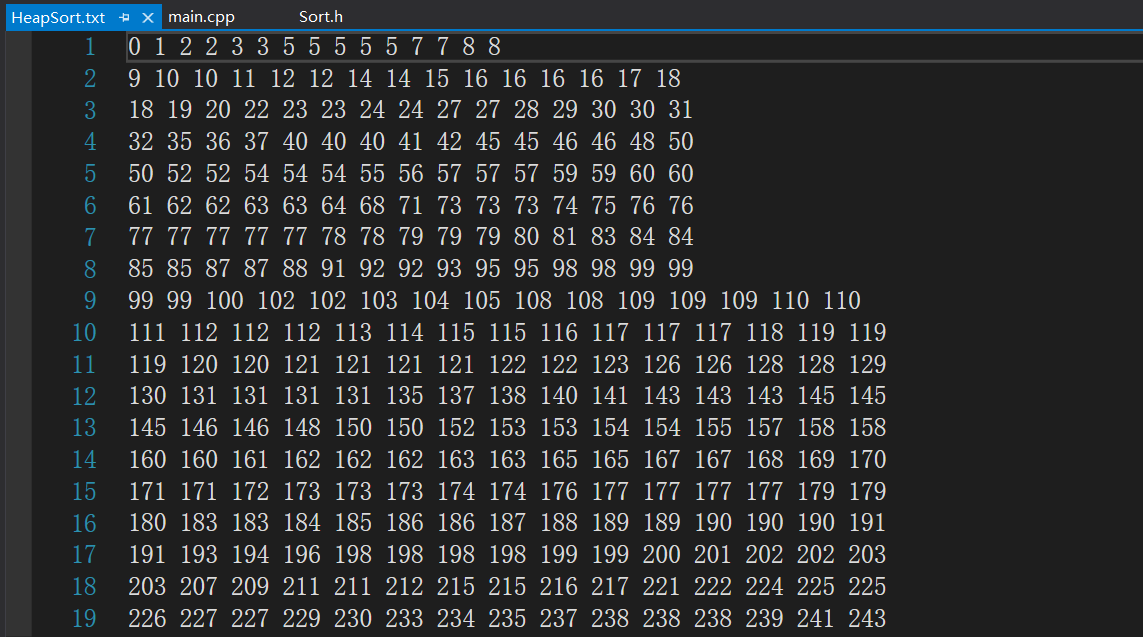


按要求输入数字1，选择随机生成10000个随机数，并存在data.txt文件中，此时文件夹中出现一个名为data.txt的文件，打开后全为随机数。



之后分别输入2、3、4、5，选择不同的排序算法，生成不同的排好序的文件





最后输入0，退出程序。

# 2 最短路径

### 2.1 要求

设有n(n>10)个城市之间的交通图。假设任意两个城市之间不一定有直接交通路线，权表示乘车时间。要求事先将交通图信息将存入磁盘文件中，要求从某城市出发到其他城市的最少乘车时间和乘车路线。要求将结果以图形的方式在屏幕上输出。

### 2.2 Dijkstra算法

基本思想：每次找到离源点（如1号结点）最近的一个顶点，然后以该顶点为中心进行扩展，最终得到源点到其余所有点的最短路径。

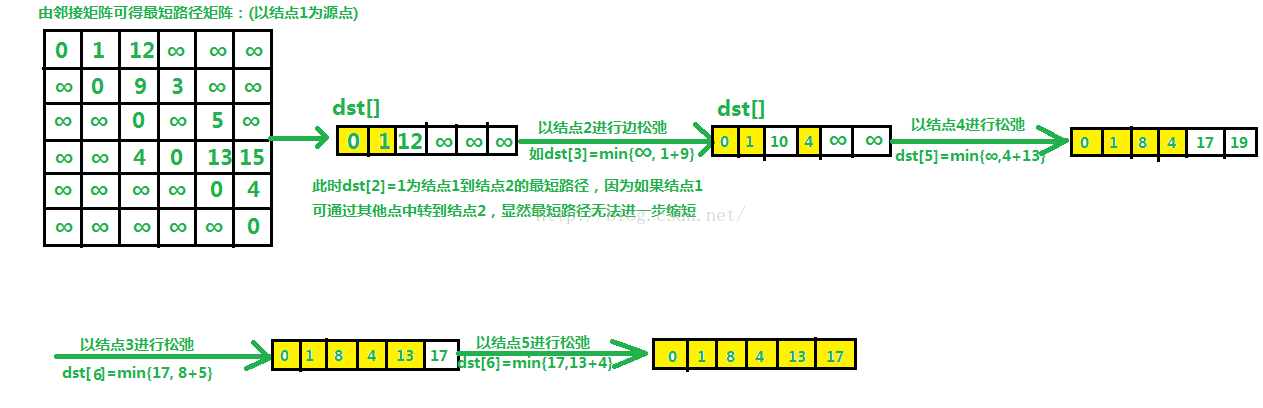
基本步骤：

1，设置标记数组book[]：将所有的顶点分为两部分,已知最短路径的顶点集合P和未知最短路径的顶点集合Q，很显然最开始集合P只有源点一个顶点。book[i]为1表示在集合P中；

2，设置最短路径数组dst[]并不断更新：初始状态下，令dst[i] = edge[s][i](s为源点，edge为邻接矩阵)，很显然此时dst[s]=0，book[s]=1。此时，在集合Q中可选择一个离源点s最近的顶点u加入到P中。并依据以u为新的中心点，对每一条边进行松弛操作(松弛是指由结点s-->j的途中可以经过点u，并令dst[j]=min{dst[j], dst[u]+edge[u][j]})，并令book[u]=1;

3，在集合Q中再次选择一个离源点s最近的顶点v加入到P中。并依据v为新的中心点，对每一条边进行松弛操作(即dst[j]=min{dst[j], dst[v]+edge[v][j]}),并令book[v]=1;

4，重复3，直至集合Q为空。



### 2.3 程序设计

#### 2.3.1 设计过程

###### 2.3.1.1前期分析

1. 本题中的难点在于图形化界面输出路径，在此，我使用了适合在C++上运行的Easyx库，完成了简单的图像绘制，并加入了中国地图作为背景，使图像更加优美。
2. **void** drawGraph(MGraph pGraph)
3. {
4. initgraph(862, 583,SHOWCONSOLE);
5. IMAGE img;
6. loadimage(&img, L"C:\\Users\\莫辞\\source\\repos\\数据结构课设\\数据结构课设/chinamap1.jpg");
7. putimage(0, 0, &img);
8. **int** x1, y1, x2, y2;
9. setlinecolor(RGB(255, 0, 0));
10. setlinestyle(0, 3, 0);
11. **wchar\_t** str[100];
12. **for** (**int** i = 1; i < pGraph.vertexNum; ++i)
13. {
14. **for** (**int** j = 0; j < i; ++j)
15. {
16. **if** (pGraph.graphEdge[i][j] > 0 && pGraph.graphEdge[i][j] < MINFINITE)
17. {
18. x1 = pGraph.graphVertex[i].x;
19. y1 = pGraph.graphVertex[i].y;
20. x2 = pGraph.graphVertex[j].x;
21. y2 = pGraph.graphVertex[j].y;
22. line(x1, y1, x2, y2);
23. swprintf(str, \_T("%d"), pGraph.graphEdge[i][j]);
24. outtextxy((x1 + x2 - 10) / 2, (y1 + y2 - 20) / 2, (**LPCTSTR**)str);
25. }
26. }
27. }
28. setfillcolor(RGB(0, 0, 255));
29. **int** radio = 8;
30. **for** (**int** i = 0; i < pGraph.vertexNum; ++i)
31. {
32. fillcircle(pGraph.graphVertex[i].x, pGraph.graphVertex[i].y, radio);
33. //outtextxy(pGraph.graphVertex[i].x - 5, pGraph.graphVertex[i].y - 5, pGraph.graphVertex[i].data);
34. }
35. MinimumSpanningTree(pGraph);
36. \_getch();
37. //closegraph();
38. };

###### 2.3.1.2 编程过程

1.邻接矩阵结点的设计

分别设计边集和点集的结构体

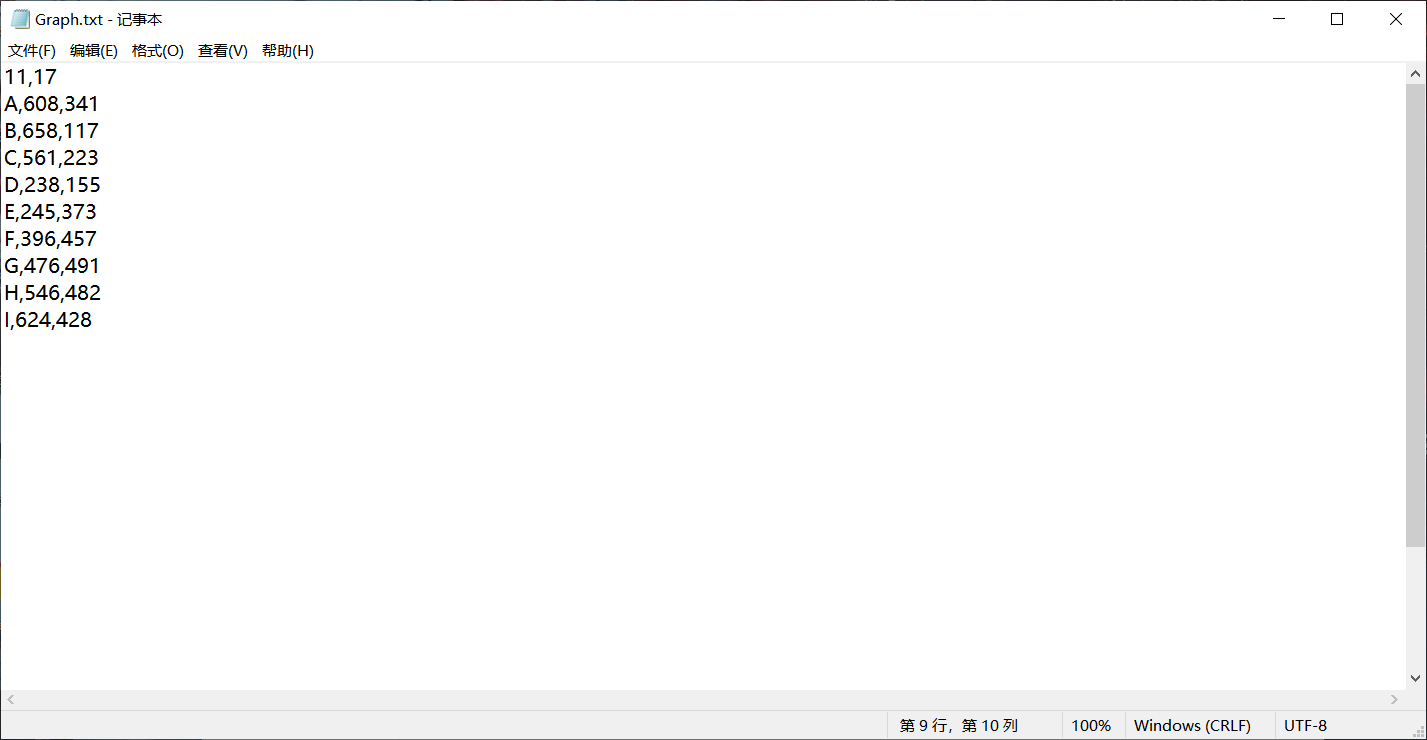
1. **typedef** **struct**
2. {
3. **int** edgeBegin;
4. **int** edgeEnd;
5. **int** wight;
6. }EdgeWight;
8. **typedef** **struct**
9. {
10. **char** data;
11. **int** x;
12. **int** y;
13. }Vertex;
15. **typedef** **struct**
16. {
17. Vertex graphVertex[MAXVEX];
18. EdgeType graphEdge[MAXVEX][MAXVEX];
19. **int** vertexNum;
20. **int** edgeNum;
21. }MGraph;
22. Dijkrast算法实现与绘制
23. **void** Dijkstra(MGraph pGraph)
24. {
25. //记录最短路径的值的数组
26. **int** shortesPath[MAXVEX];
27. **int** visitVertex[MAXVEX];
28. //记录使用过顶点的集合
29. **int** fromVertex[MAXVEX];
30. **int** x1, x2, y1, y2;
31. setlinecolor(RGB(255, 255, 0));
32. setlinestyle(0,3,0);
33. **for** (**int** i = 0; i < pGraph.vertexNum; ++i)
34. {
35. shortesPath[i] = pGraph.graphEdge[v][i];
36. fromVertex[i] = v;
37. visitVertex[i] = 0;
38. }
39. visitVertex[v] = 1;
40. **for** (**int** i = 1; i < pGraph.vertexNum; ++i)
41. {
42. **int** minWight = MINFINITE;
43. **int** min = -1;
44. **for** (**int** j = 0; j < pGraph.vertexNum; ++j)
45. {
46. **if** (visitVertex[j] == 0 && minWight > shortesPath[j])
47. {
48. minWight = shortesPath[j];
49. min = j;
50. }
51. }
52. visitVertex[min] = 1;
53. **for** (**int** j = 0; j < pGraph.vertexNum; ++j)
54. {
55. **if** (pGraph.graphEdge[min][j] != MINFINITE)
56. {
57. **if** (!visitVertex[j]&&shortesPath[min] + pGraph.graphEdge[min][j] < shortesPath[j])
58. {
59. shortesPath[j] = shortesPath[min] + pGraph.graphEdge[min][j];
60. fromVertex[j] = min;
61. }
62. }
63. }
64. }
65. printf("最短路径时间：\n");
66. **for** (**int** i = 0; i < pGraph.vertexNum; ++i)
67. {
68. printf("%-3d ", shortesPath[i]);
69. }
70. printf("\n经过城市：\n");
71. **for** (**int** i = 0; i < pGraph.vertexNum; ++i)
72. {
73. //printf("%-3d ", fromVertex[i]);
74. Print(fromVertex[i]);
75. printf(" ");
76. }
77. printf("\n");
78. printf("\n");
79. printf("\n");
80. **for** (**int** i = 0; i < pGraph.vertexNum; ++i)
81. {
82. setlinecolor(RGB(255, 255, 0));
83. setlinestyle(0, 3, 0);
84. **if** (i != v)
85. {
86. **int** j = i;
87. **int** k = i;
88. //printf("%c--->", pGraph.graphVertex[i].data);
89. Print(i);
90. printf("--->");
91. **while** (fromVertex[j] != v)
92. {
93. //printf("%c---->", pGraph.graphVertex[fromVertex[j]].data);
94. Print(fromVertex[j]);
95. printf("--->");
96. j = fromVertex[j];
97. \_getch();
98. x1 = pGraph.graphVertex[k].x;
99. y1 = pGraph.graphVertex[k].y;
100. x2 = pGraph.graphVertex[j].x;
101. y2 = pGraph.graphVertex[j].y;
102. line(x1, y1, x2, y2);
103. k = j;
104. }
105. //printf("%c\n", pGraph.graphVertex[v].data);
106. Print(v);
107. printf("    最短路径时间：%-3d ", shortesPath[i]);
108. printf("\n");
109. \_getch();
110. x1 = pGraph.graphVertex[v].x;
111. y1 = pGraph.graphVertex[v].y;
112. x2 = pGraph.graphVertex[j].x;
113. y2 = pGraph.graphVertex[j].y;
114. line(x1, y1, x2, y2);
116. }
117. \_getch();
118. setlinecolor(RGB(255, 0, 0));
119. //wchar\_t str[100];
120. **for** (**int** m = 1; m < pGraph.vertexNum; ++m)
121. {
122. **for** (**int** n = 0; n < m; ++n)
123. {
124. **if** (pGraph.graphEdge[m][n] > 0 && pGraph.graphEdge[m][n] < MINFINITE)
125. {
126. x1 = pGraph.graphVertex[m].x;
127. y1 = pGraph.graphVertex[m].y;
128. x2 = pGraph.graphVertex[n].x;
129. y2 = pGraph.graphVertex[n].y;
130. line(x1, y1, x2, y2);
131. //swprintf(str, \_T("%d"), pGraph.graphEdge[i][j]);
132. //outtextxy((x1 + x2 - 10) / 2, (y1 + y2 - 20) / 2, (LPCTSTR)str);
133. }
134. }
135. }
136. }
137. }

#### 2.3.2 问题及解决方案

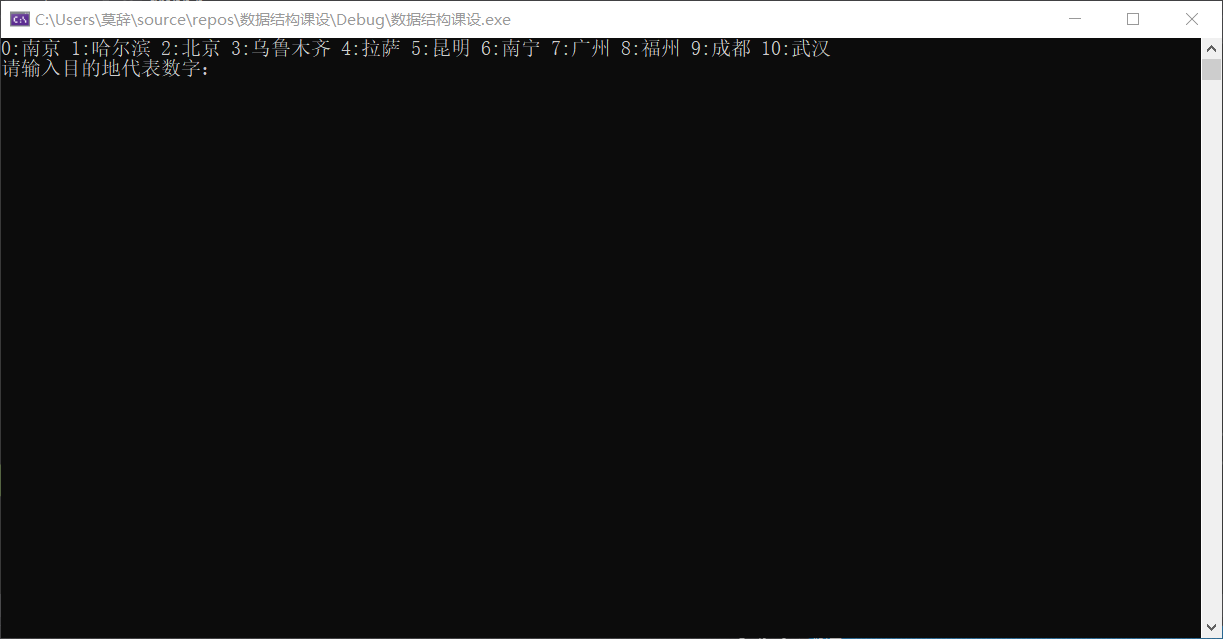
1.在对于背景图片的添加的过程中，因为识别不了路径导致前期执行时程序一直无法运行，之后得知要用绝对路径，并且在对于字符“/”，要用两个“/”才能成功转意。对于不同点的坐标位置，需要用相对精确的像素点的位置来表示。

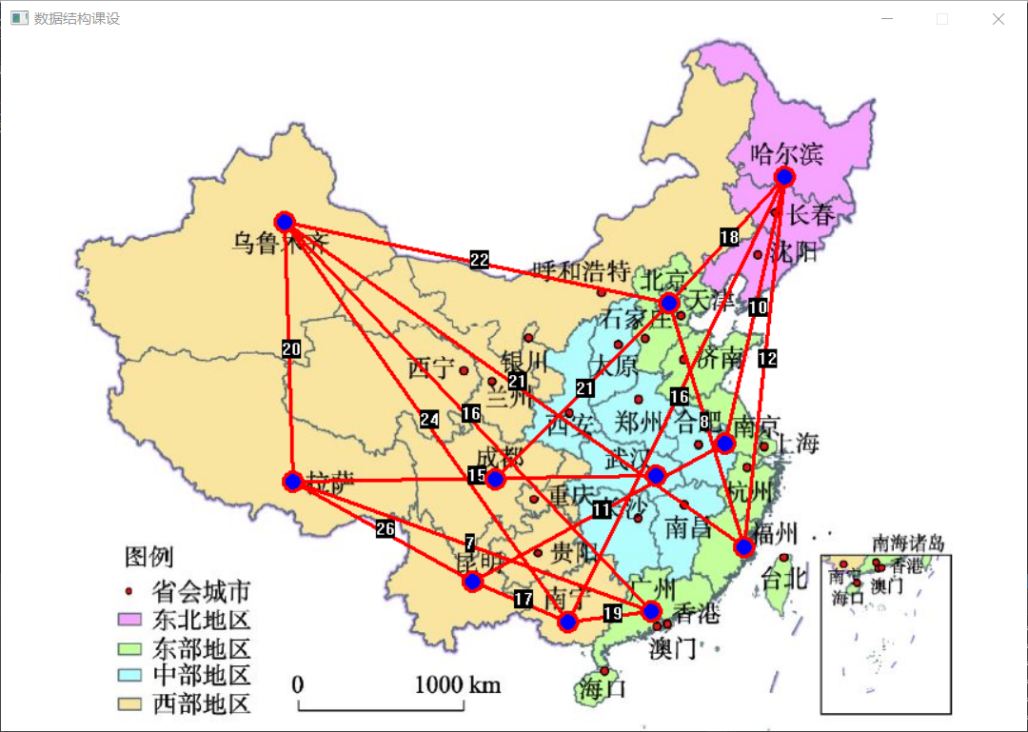
### 2.4 程序测试

首先，在“Graph.txt”文件中储存了点的个数，边数，各个点位置的坐标以及权值。

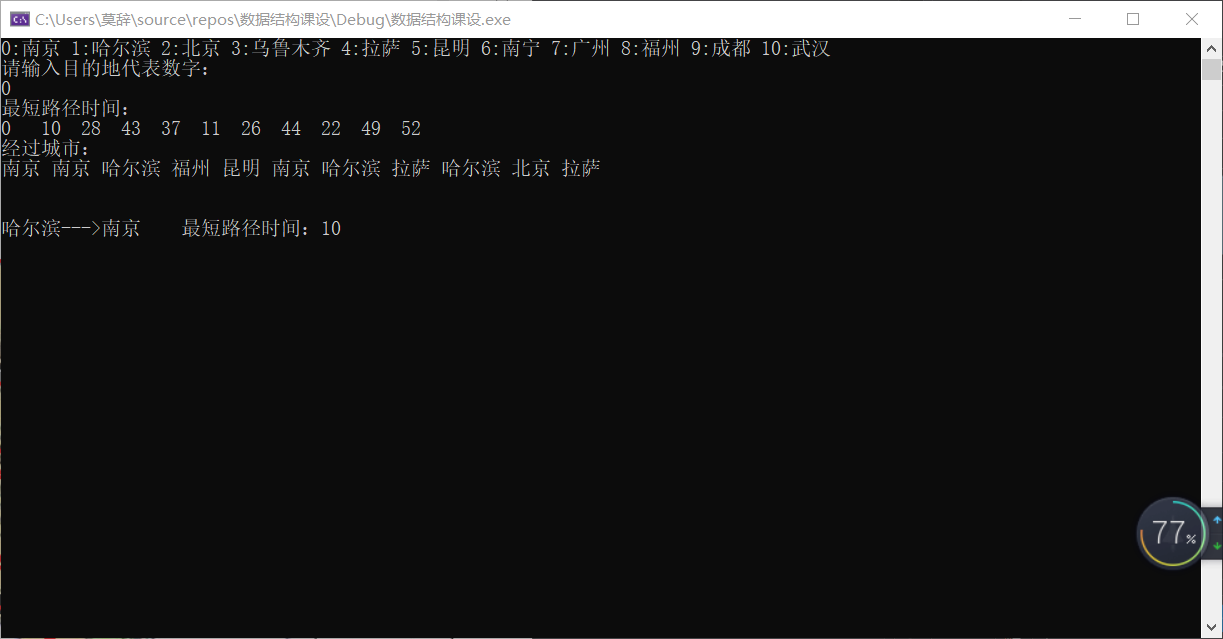


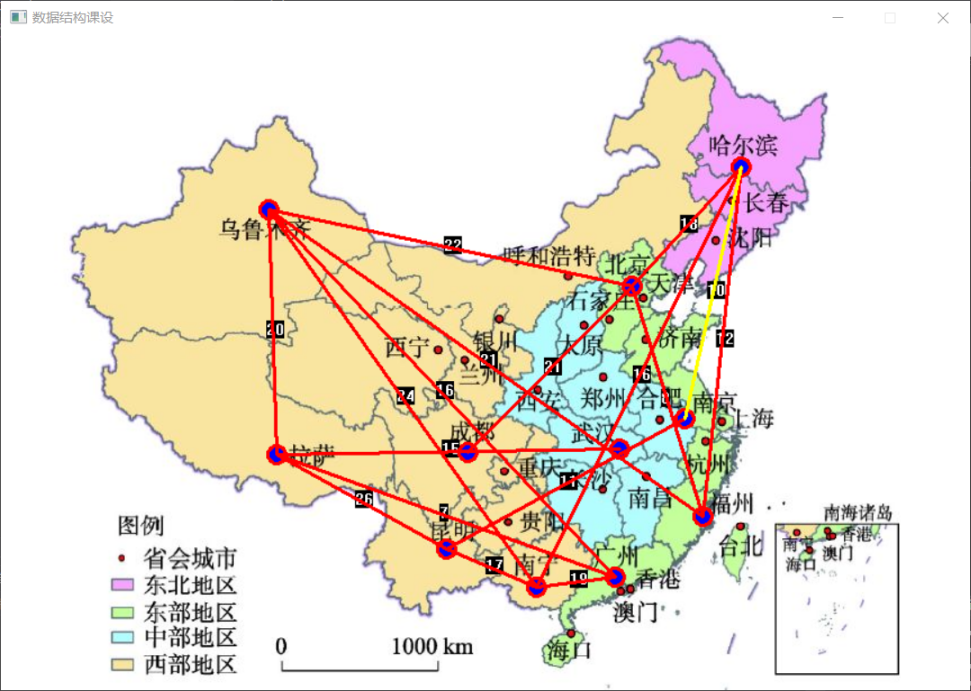
之后运行程序，弹出控制台，按照提示选择目的地，之后会弹出图形界面。

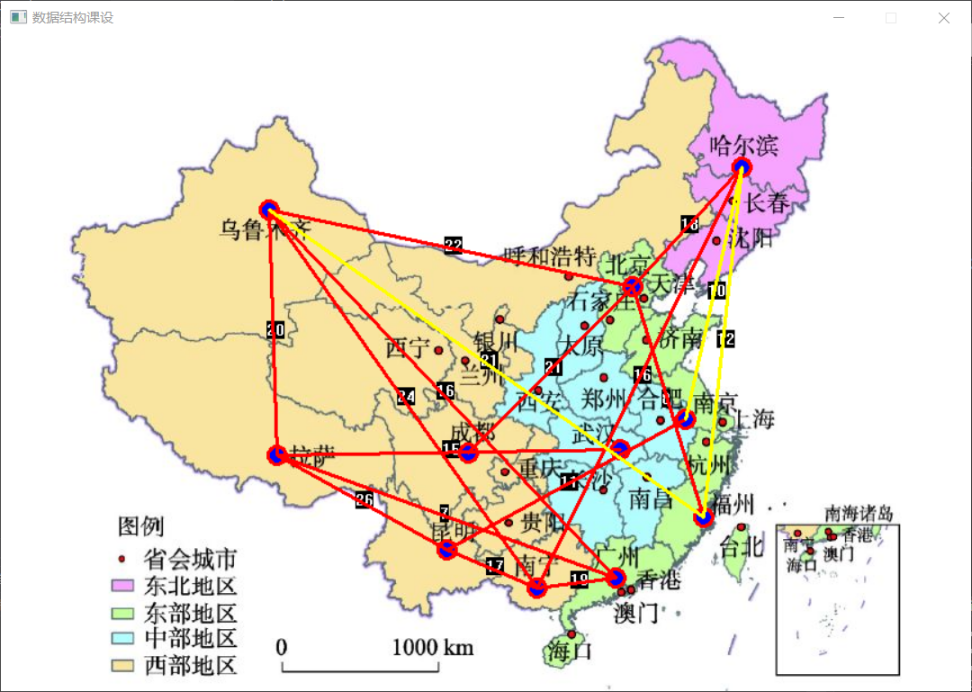




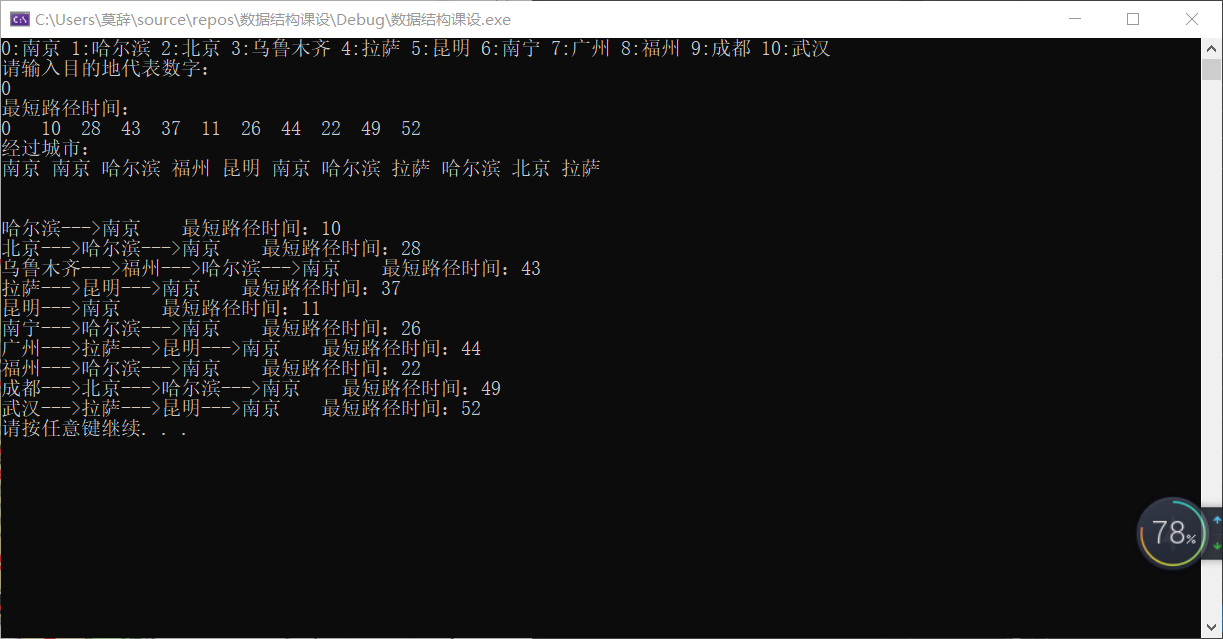
在选择完目的地后，将会随着按任意键持续逐步弹出路线，并在图形窗口以黄线覆盖红线绘制，直到到达目的地，路线刷新。







直到输出所有路线。



# 3 收获与体会

本次课程设计的选题是排序算法和最短路径问题，此次课程设计不仅要熟悉排序算法和Dijkstra算法，更多的是考研如何把这些算法融会贯通到具体应用上。第一个设计的难点在于文件的读写，第二个课设的难点在于图形化输出。

排序算法是数据结构中的一块重要内容，各种排序都有其优缺点，应视具体情况进行选择。图形化的界面对于用户友好，非常适合来用表示一些路线、动态规划。

将画图函数与我的需求相结合是我遇到的最大困难，通过查阅一些资料，以及不断的调试，最终让图形化输出基本符合我的期望。

通过这次数据结构课程设计，我对代码编写和图形化界面设计有了更深入的了解。对于文件的读写操作以及各种排序算法的实现及其性能、Dijkstra算法有了更深刻的认识，同时也学习到了用Easyx库中的函数进行简单的绘制图形。