数据结构实验报告三

实验名称：图及其应用

### 实验目的及实验要求

1. 通过完成本实验，掌握图的两种基本的存储结构（邻接矩阵、邻接表），以及图的基本算法实现（建立、遍历），并能运用图结构分析解决一些实际问题。
2. 本实验训练的要点是：图的两种基本存储结构，及各种操作的算法实现（建立、遍历、图的典型应用）。

### 2实验内容及实验步骤（附运行结果截屏）

1. 建立无向图和有向图的邻接矩阵存储，计算顶点的度，并输出图的基本信息。
2. 建立有向图的邻接表存储表示，并根据存储计算顶点的出度和入度，然后输出图的基本信息。
3. 编写完整的程序实现AOV网的拓扑排序。
4. 编程求AOE网的关键路径。
5. 编程实现单源点最短路径的Dijkstra算法。

注：（1）～（2）必做，（3）～（5）选做。

实验步骤：

总体来说，先编写类模板，实现各自的基础结构，之后按照要求编写适当的函数方法（公共接口），最后完成封装。编写主函数直接调用。但这一次考虑到图的处理方式与以往表和树的不同，并没有把所有功能都与类模板绑定到一起而是灵活地选择了合适的处理方式。

核心代码：

//GraphMatrix.h 邻接矩阵表示图

//类的声明

1. template<class T>
2. class GraphMatrix
3. {
4. public:
5. GraphMatrix(int p=0,int e=0):point\_num(p),edge\_num(e) {};
6. bool InsertPoint(char x);
7. void PrintGraph();
8. protected:
9. char point\_name[maxn];
10. double gra[maxn][maxn];
11. int point\_num,edge\_num;
12. };

//插入节点

1. template<class T>
2. bool GraphMatrix<T>::InsertPoint(char x)
3. {
4. if(point\_num==0)
5. {
6. point\_num=1;
7. point\_name[0]=x;
8. }
9. else
10. {
11. point\_name[point\_num]=x;
12. for(int i=0; i<point\_num; i++)
13. {
14. cin>>gra[point\_num][i];
15. if(gra[point\_num][i]!=0)
16. edge\_num++;
17. }
18. point\_num++;
19. }
20. }

//GraphChart.h 邻接表表示图

//类的声明

1. template<class NameType, class DistType> class Graph;
2. template<class NameType, class DistType>
3. class Vertex
4. {
5. public:
6. Vertex()
7. {
8. padjEdge = NULL;
9. m\_vertexName = 0;
10. }
11. ~Vertex()
12. {
13. Edge<DistType> \*pmove = padjEdge;
14. while (pmove)
15. {
16. padjEdge = pmove->m\_pnext;
17. delete pmove;
18. pmove = padjEdge;
19. }
20. }
21. private:
22. friend class Graph<NameType, DistType>;
23. NameType m\_vertexName;
24. Edge<DistType> \*padjEdge;
25. };
26. template<class NameType, class DistType>
27. class Graph
28. {
29. public:
30. Graph(int size = m\_nDefaultSize)
31. {
32. m\_pVertexTable = new Vertex<NameType, DistType>[size];
33. if (m\_pVertexTable == NULL)
34. {
35. exit(1);
36. }
37. m\_numVertexs = 0;
38. m\_nmaxSize = size;
39. m\_nnumEdges = 0;
40. }
41. ~Graph()
42. {
43. Edge<DistType> \*pmove;
44. for (int i = 0; i < this->m\_numVertexs; i++)
45. {
46. pmove = this->m\_pVertexTable[i].padjEdge;
47. if (pmove)
48. {
49. this->m\_pVertexTable[i].padjEdge = pmove->m\_pnext;
50. delete pmove;
51. pmove = this->m\_pVertexTable[i].padjEdge;
52. }
53. }
54. delete[] m\_pVertexTable;
55. }
56. int GetNumEdges()
57. {
58. return m\_nnumEdges / 2;
59. }
60. int GetNumVertexs()
61. {
62. return m\_numVertexs;
63. }
64. bool IsGraphFull() const
65. {
66. //图满的?
67. return m\_nmaxSize == m\_numVertexs;
68. }
70. bool InsertEdge(int v1, int v2, DistType weight = m\_Infinity);
71. bool InsertVertex(const NameType vertex);
72. void PrintGraph();
73. private:
74. Vertex<NameType, DistType> \*m\_pVertexTable;
75. int m\_numVertexs;
76. int m\_nmaxSize;
77. static const int m\_nDefaultSize = 10;
78. static const DistType m\_Infinity = 65536;
79. int m\_nnumEdges;
80. int GetVertexPosTable(const NameType vertex);
81. };

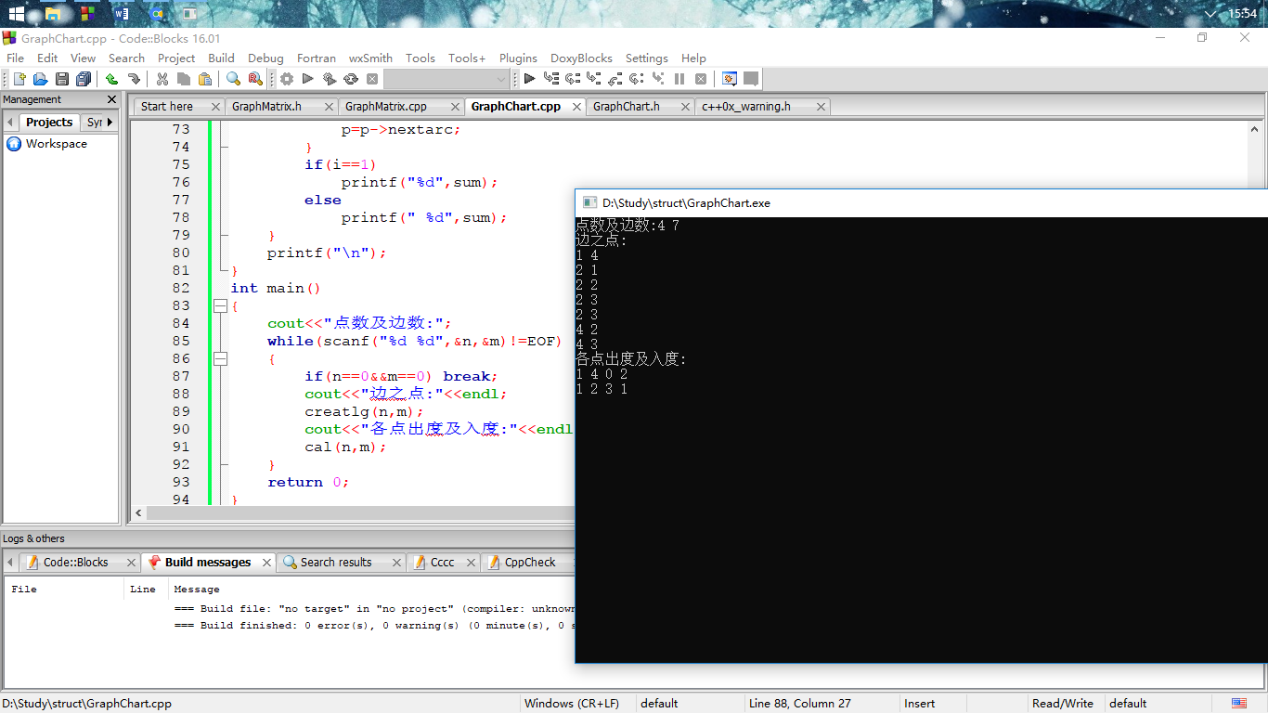
// Dijkstra算法

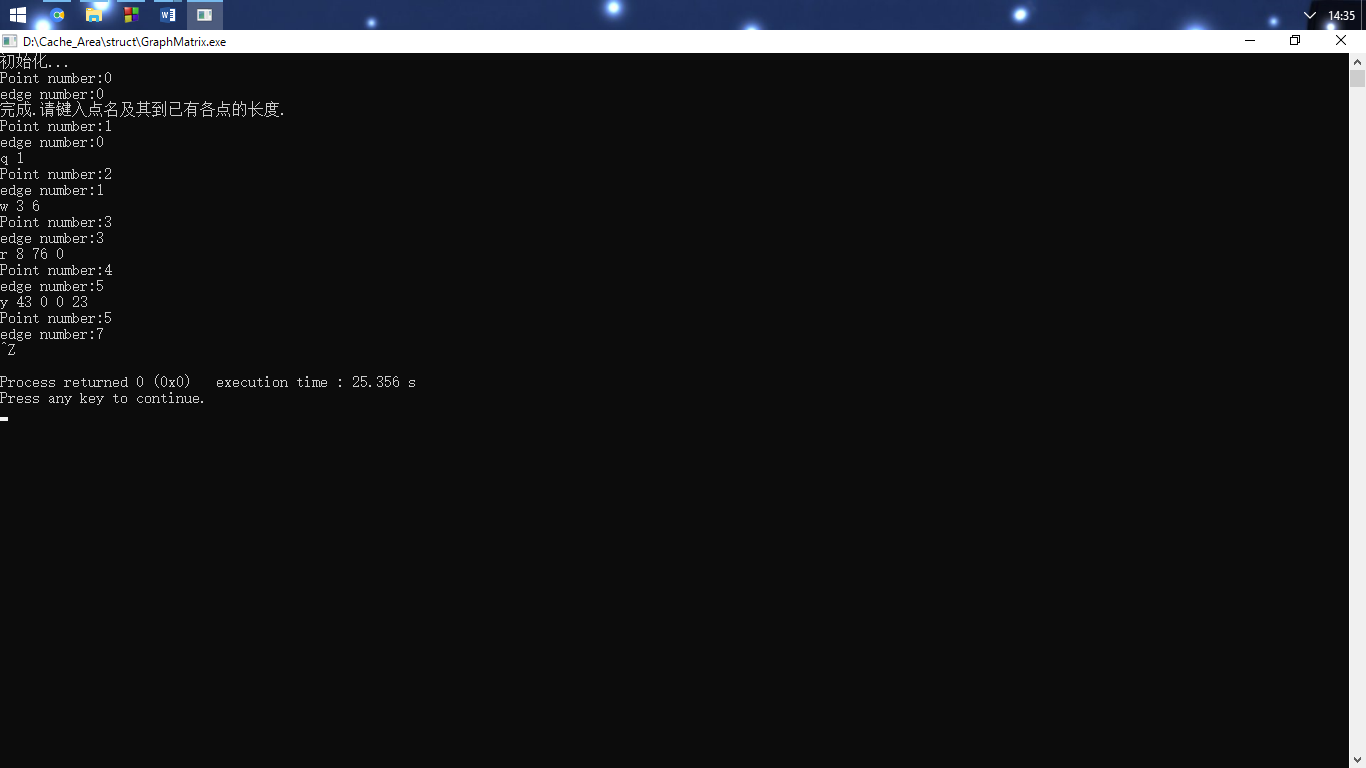
1. void Dijkstra(int n, int v, int \*dist, int \*prev, int c[maxnum][maxnum])
2. {
3. bool s[maxnum];
4. for(int i=1; i<=n; ++i)
5. {
6. dist[i] = c[v][i];
7. s[i] = 0;
8. if(dist[i] == maxint)
9. prev[i] = 0;
10. else
11. prev[i] = v;
12. }
13. dist[v] = 0;
14. s[v] = 1;
15. for(int i=2; i<=n; ++i)
16. {
17. int tmp = maxint;
18. int u = v;
19. for(int j=1; j<=n; ++j)
20. if((!s[j]) && dist[j]<tmp)
21. {
22. u = j;
23. tmp = dist[j];
24. }
25. s[u] = 1;
26. for(int j=1; j<=n; ++j)
27. if((!s[j]) && c[u][j]<maxint)
28. {
29. int newdist = dist[u] + c[u][j];
30. if(newdist < dist[j])
31. {
32. dist[j] = newdist;
33. prev[j] = u;
34. }
35. }
36. }
37. }

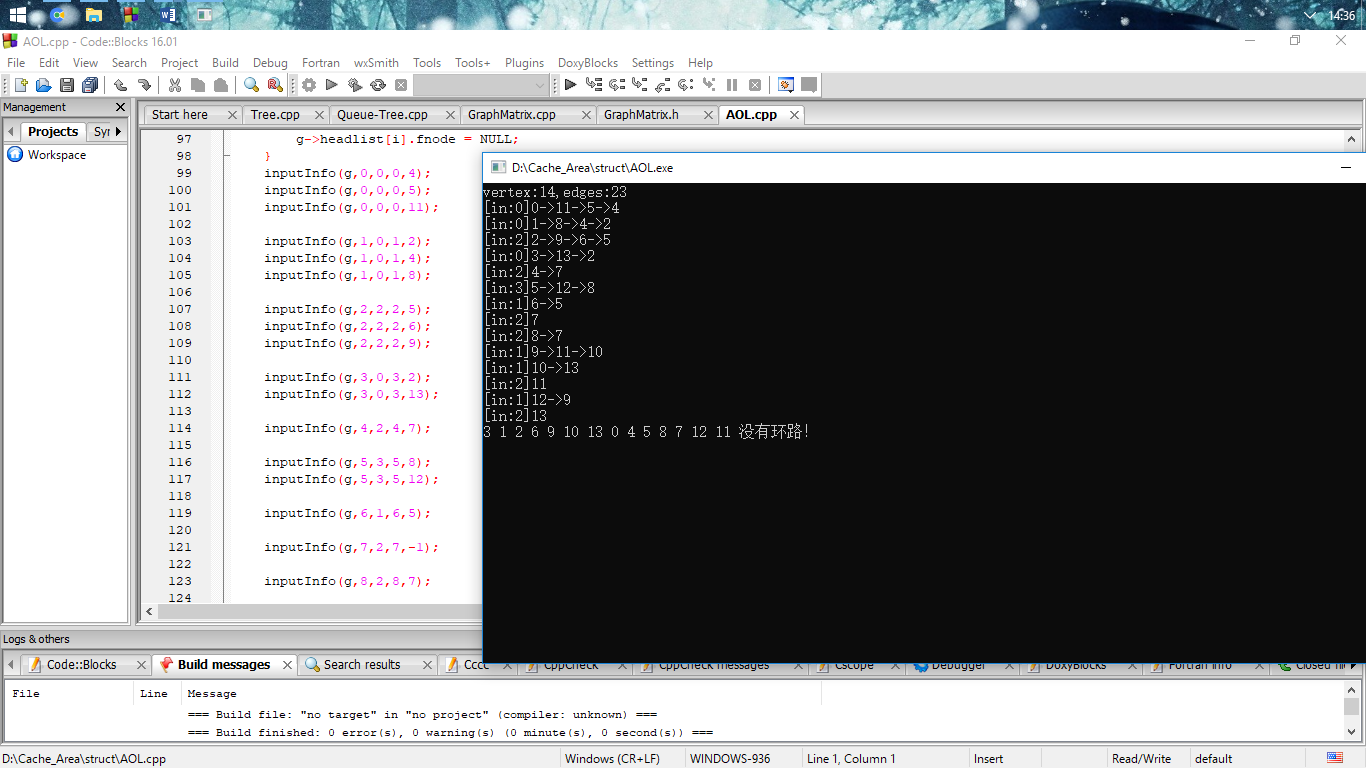
//AOL

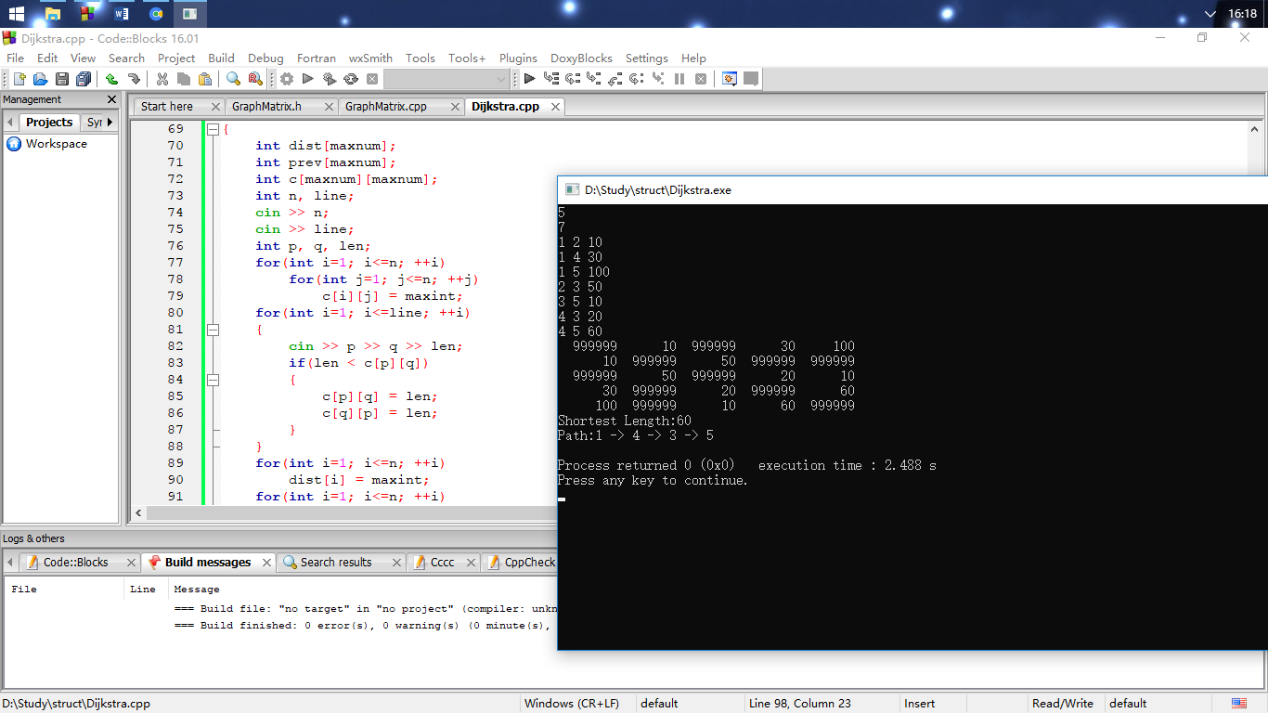
1. int topGraph(graph g)
2. {
3. EdgeNode \*e;
4. int i,k,gettop;
5. int top = 0 ;
6. int count = 0;
7. int \*stack;
8. stack = (int \*)malloc(g->numVertexes \* sizeof(int));
9. for(i=0; i<g->numVertexes; i++)
10. {
11. if(g->headlist[i].in == 0) //把入度为0的，即没有入度的点入栈
12. stack[++top] = i;
13. }
14. while(top)
15. {
16. gettop = stack[top--];
17. printf("%d ",gettop);
18. count++;
19. for(e = g->headlist[gettop].fnode; e ; e=e->next) //一次遍历链表，减少各个子节点的入度
20. {
21. k = e->data;
22. if(!(--g->headlist[k].in))
23. stack[++top] = k;
24. }
25. }
26. if(count < g->numVertexes)
27. return ERROR;
28. else
29. return OK;
30. }

截图：









3 实验体会（实验遇到的问题及解决方法）

图的算法刚接住的时候感觉很难，但是当了解后发现它的解决方式一般是固定的而不像表和树可以有自己的完全不同的解决方法（虽然大多数情况下并不一定好）。所以遇到的问题就是在刚开始的时候无从下手，然后翻了几遍书后再在网上看了一些相关的博客，逐渐了解了这方面的解决思路。