实验报告

1. 实验题目： 导航软件

2. 需求分析：

1) 实现目的：

要求在所给的数据集上建立图结构，并在建立的图结构上实现 Dijkstra 算法求解任意两点之间的最短路径；

2) 输入规范：

a) 一个 txt 文档包含若干数据，每行数据以 src dst distance 形式，其中 src 表示源点，dst 表示目标点，distance 表示从源点到目标点有直接连接的路径且路径长度为 distance

b) 对于用户输入有如下规范：

终端显示：“请输入文档名：”，此时输入所读取的数据集的 txt 文档；而后终端接分别着显示“源点：”“目标点：”，此时分别将所要求得两点输入；程序输出结果后显示“请输入 1 确认继续操作，否则输入 0 终止程序：”判断是否继续操作；

3) 实现基本要求：

a) 对于任意两点之间的最短路径求解并输出；

b) 对于最短路径长度的输出；

4) 测试数据：

所提供的 data.txt 文档，其中分别输入 1->2,2->3；

3. 概要设计：

1) 部分函数设计：

CreatGraph(char *filename, ALGraph *G)

操作结果：根据输入的数据集文档，创建有向图 G;

GetDis(ALGraph *G, int src, int dst)

操作结果：根据有向图 G 以及输入的源点 src，目标点 dst，得到 G 中两点之间的距离;

Find(int *mark, int *dis, ALGraph *G)

操作结果：寻找未标记节点中距出发点最近的节点;

UpdateDis(int src, int *dis, int *mark, int *path, ALGraph *G)

操作结果：更新有向图 G 中各点到源点的最短路径的距离及前驱结点;

OutPrint(int src, int dst, int *dis, int *path, ALGraph *G)

操作结果：根据目标点的前驱结点不断迭代输出最短路径;

dijkstra(char *filename)

操作结果：实现最短路径的求解;

2) 主函数设计：

```
1  int main()
```

```
  {
    输入数据集的txt文档名;
    执行dijkstra算法;
    结束运行;
  }
```

4. 详细设计：

1) 图的相关数据结构设计：

```
  typedef enum
{
  DG,
  DN,
  UDG,
  UDN
} GraphKind; // 图的类型

typedef struct ArcNode
{
  int adjvex;               // 该弧所指向的顶点的位置
  struct ArcNode *nextarc; // 指向下一条弧的指针
  int weight;               // 该段弧的权重
} ArcNode;

typedef struct VNode
{
  ArcNode *firstarc; // 指向第一条依附该顶点的弧的指针
} VNode, *AdjList;

typedef struct
{
  AdjList vertices;
  int vexnum;      // 图的当前顶点数
  GraphKind kind; // 图的种类标志
} ALGraph;
```

2) 相关函数详细设计：

a)CreatGraph 函数如下：

```
//有向图的创建
```

```
void CreatGraph(char *filename, ALGraph *G)
{
  // 初始化图的类型与顶点数
  G->kind = DG;
  G->vexnum = 0;
  FILE *fp;
  // 第一次读取文件，获得图的顶点数
  if ((fp = fopen(filename, "r")) == NULL)
  {
    printf("no exist\n");
    return;
  }
  printf("开始更新顶点数\n");
  while (!feof(fp))
  {
    int src, dst, dis; // 源点，目标点，距离
    fscanf(fp, "%d %d %d", &src, &dst, &dis);
    // 更新顶点数
    G->vexnum = (G->vexnum > src) ? G->vexnum : src;
    G->vexnum = (G->vexnum > dst) ? G->vexnum : dst;
  }
  G->vexnum++; // 考虑0
  fclose(fp);
  printf("已更新顶点共%d个\n", G->vexnum + 1);
  G->vertices = (VNode *)malloc(G->vexnum * sizeof(VNode));
  // 初始化图的顶点信息及相关边的信息
  for (int i = 0; i < G->vexnum; i++)
  {
    G->vertices[i].firstarc = NULL;
  }
  // 第二次打开文件存储顶点及边的相关信息
  if ((fp = fopen(filename, "r")) == NULL)
  {
    printf("文件不存在\n");
    return;
  }
  printf("开始读取相关信息\n");
```

```
39    while (!feof(fp))
40    {
41      int src, dst, dis; // 源点，目标点，距离
42      fscanf(fp, "%d %d %d", &src, &dst, &dis);
43      //  作为第一条弧
44      if (G->vertices[src].firstarc == NULL)
45      {
46        ArcNode *p = (ArcNode *)malloc(sizeof(ArcNode));
47        G->vertices[src].firstarc = p;
48        p->nextarc = NULL;
49        p->adjvex = dst;
50        p->weight = dis;
51      } // 向后查找为空的下一条
52      else
53      {
54        ArcNode *p = G->vertices[src].firstarc;
55        while (p->nextarc)
56          p = p->nextarc;
57        ArcNode *q = (ArcNode *)malloc(sizeof(ArcNode));
58        p->nextarc = q;
59        q->nextarc = NULL;
60        q->adjvex = dst;
61        q->weight = dis;
62      }
63    }
64    fclose(fp);
65    printf("读取结束\n");
66  }
```

b)GetDis 函数如下：

```
1   // 得到邻接表中源点 src 到目标点 dst 的距离
2   int GetDis(ALGraph *G, int src, int dst)
3   {
4     int dis = MAX_DIS;
5     ArcNode *p = G->vertices[src].firstarc;
6     while (p)
7     {
8       if (p->adjvex == dst)
```

```
    {
      dis = p->weight;
      break;
    }
    else
      p = p->nextarc;
  }
  return dis;
}
```

c)Find 函数如下：

```
// 寻找未标记节点中距出发点最近的节点
int Find(int *mark, int *dis, ALGraph *G)
{
  int k = 0;
  while (mark[k] != 0)
    k++;
  for (int i = 0; i < G->vexnum; i++)
  {
    if (mark[i] == 0 && dis[k] > dis[i])
      k = i;
  }
  return k;
}
```

d)UpdateDis 函数如下：

```
// 更新距离
void UpdateDis(int src, int *dis, int *mark, int *path, ALGraph *G)
{
  int n = 0;
  int k = src; // 从源点开始寻找最近的未标记节点
  dis[k] = 0;
  while ((n++) <= G->vexnum)
  {
    for (int i = 0; i < G->vexnum; i++)
    {
      // 对于未标记节点
      if (!mark[i])
```

```
13        {
14          // 更新最短距离以及前驱结点
15          if (dis[i] > dis[k] + GetDis(G, k, i))
16          {
17            dis[i] = dis[k] + GetDis(G, k, i);
18            path[i] = k;
19          }
20        }
21      }
22      //
23      k = Find(mark, dis, G);
24      mark[k] = 1; // 标记该节点
25    }
26  }
```

e)OutPrint 函数如下；

```
1  // 根据目标点以及前驱结点输出路径
2  void OutPrint(int src, int dst, int *dis, int *path, ALGraph *G)
3  {
4    int k, n = 0;
5    int *l; // 输出路径
6    l = (int *)malloc(G->vexnum * sizeof(int));
7    l[0] = dst; // 从目标点开始记录
8    k = dst;
9    // 向前回溯
10   while (k != src)
11   {
12     l[n++] = path[k];
13     k = path[k];
14   }
15   printf("最短路径为:");
16   for (int i = n - 1; i >= 0; i--)
17   {
18     if (i == n - 1)
19       printf("%d", l[i]);
20     else
21       printf("->%d", l[i]);
22   }
```

```
23    printf("\n");
24    printf("最短路径长度为:%d\n", dis[dst]);
25  }
```

f)dijkstra 函数如下：

```
1   // 迪杰斯特拉算法实现最短路径的求解
2   void dijkstra(char *filename)
3   {
4     // 读取文件建立相关有向图
5     ALGraph *G = (ALGraph *)malloc(sizeof(ALGraph));
6     CreatGraph(filename, G);
7
8     int *dis, *path, *mark;
9     mark = (int *)malloc((G->vexnum) * sizeof(int));
10    // 标记各点是否被访问
11    dis = (int *)malloc((G->vexnum) * sizeof(int));
12    // 记录各点到源点的最短距离
13    path = (int *)malloc((G->vexnum) * sizeof(int));
14    // 记录最短路径的前驱结点
15
16    // 源点，目标点，结束标志
17    int src, dst, choose = 1;
18    while (choose)
19    {
20      // 初始化记录数组
21      for (int i = 0; i < G->vexnum; i++)
22      {
23        mark[i] = 0;
24        dis[i] = MAX_DIS;
25        path[i] = -1;
26      }
27      printf("源点:");
28      scanf("%d", &src);
29      printf("目标点:");
30      scanf("%d", &dst);
31
32      UpdateDis(src, dis, mark, path, G);
33      OutPrint(src, dst, dis, path, G);
```

```
34      printf("\n");
35      printf("请输入1确认继续操作，否则输入0终止程序：");
36      scanf("%d", &choose);
37    }
38
39    // 释放内存
40    for (int i = 0; i < G->vexnum; i++)
41    {
42      ArcNode *p = G->vertices[i].firstarc, *q = NULL;
43      if (p != NULL)
44        q = p->nextarc;
45      while (q)
46      {
47        free(p);
48        p = q;
49        q = q->nextarc;
50      }
51      free(p);
52    }
53    free(G);
54    free(dis);
55    free(path);
56    free(mark);
57  }
```

5. 调试分析：

1) 本次实验主要是对迪杰斯特拉算法的应用，在调试过程中出现的问题基本在于对于空间分配考虑不周，其余问题相较而言较为简单；

2) 由于采取邻接表的储存形式，相较于邻接矩阵而言，在顶点数较大的稀疏图中对于空间的利用更为充分；

3) 采取了朴素的迪杰斯特拉算法，时间复杂度为 O(n2)，如果时间足够还可以对于算法进行相应的优化，使其时间复杂度降到 O(E*log(|V|))；

4) 经验体会：对于程序运行时的分配空间问题要时刻注意；

6. 用户手册：

1) 本程序的运行环境为 DOS 操作系统，执行文件为 Navigation_software.exe;

2) 进入程序即显示相应的操作提示，即可得到目标求解；

```
 *  Executing task: d:\Codefiled\CODE_C\C_
Single\Data_Structure\bin\Navigation_softw
are.exe

请输入txt文档名:data.txt
开始更新顶点数
已更新顶点共10000个
开始读取相关信息
读取结束
源点:1
目标点:2
最短路径为:1->6333->1488->1714->4551->1615-
>2312->5833->5802
最短路径长度为:156

请输入1确认继续操作,否则输入0终止程序:1
```

7. 测试结果：

测试结果截图如下：

```
请输入txt文档名:data.txt
开始更新顶点数
已更新顶点共10000个
开始读取相关信息
读取结束
源点:1
目标点:2
最短路径为:1->6333->1488->1714->4551->1615-
>2312->5833->5802
最短路径长度为:156

请输入1确认继续操作,否则输入0终止程序:1
源点:2
目标点:3
最短路径为:2->7546->9411->830->3033->9527->
7608->9373
最短路径长度为:160

请输入1确认继续操作,否则输入0终止程序:0
结束运行 *  Press any key to close the term
```

8. 附录：

源程序文件名清单：

Navigation_software.c

Navigation_software.h

Dijkstra.h