中南大学

数据结构试验报告

题 目 实验五 图结构及其应用

学生姓名 张子洋

学 号 8208180512

指导老师 郑瑾

学 院 计算机学院

专业班级 计算机与通信类2212班

完成时间 2023.6

指导老师评定 签名

1. 从键盘输入的数据建立图，并进行深度优先搜索和广度优先搜索（验证性实验）
2. 需求分析

要求：很多涉及图上操作的算法都是以图的遍历操作为基础的。试编写一个程序，演示无向图的遍历操作。

在主程序中提供下列菜单：

(1)“1”代表图的建立；

(2)“2”代表深度优先遍历图；

(3)“3”代表广度优先遍历图；

(4)“0”代表结束。

基本要求

以邻接表为存储结构，实现连通无向图的深度优先和广度优先遍历。以用户指定的结点为起点，分别输出每种遍历下的结点访问序列和相应生成树的边集。

分析：

1. 输入的形式和输入值的范围：字符与int型数字
2. 输出的形式：输出对应遍历的结果字符串。
3. 程序所能达到的功能：建立图，深度优先遍历图，广度优先遍历图。
4. 测试数据：

* DG 9 15
* A B C D E F G H I
* AB AF BG BC CD BI CI DI DG DH DE EH EF FG GH

1. 概要设计
2. 为了实现程序功能，需要定义图的抽象数据类型。

ADT：图（Graph）

Data：顶点的有穷非空集合和边集合

Operation：

Graph(V)：按照顶点集 VV 初始化图。

addVex(v)：在图 G 中添加新顶点 vv。

deleteVex(v)：删除图 G 中顶点 vv 及其相关弧。

addEdge(v1,v2)：在图 G 中添加弧<v1,v2><v1,v2>，若 G 是无向图，则还需要增添对称弧 <v2,v1><v2,v1>。

deleteEdge(v1,v2)：在图 G 中删除弧<v1,v2><v1,v2>，若 G 是无向图，则还需要删除对称弧 <v2,v1><v2,v1>。

DFSTraverse(v)：从顶点 vv 开始对图 G 进行深度优先遍历。

BFSTraverse(v)：从顶点 vv 开始对图 G 进行广度优先遍历。

1. 程序栈操作的实现包含12个函数：

* 主函数main()；
* 查找图节点LocateVex ()；
* 打印图节点visit ()；
* 建图 CreateGraph ()；
* 连通图深度优先遍历DFS ()；
* 非连通图深度优先遍历DFSTraverse()；
* 连通图广度优先遍历BFS()；
* 非连通图广度优先遍历BFSTraverse()；
* 添加队列节点enqueue()；
* 判断队列是否为空queueempty()；
* 移出队列节点dequeue()；
* 创建队列initqueue()；

1. 详细设计

从键盘输入的数据建立图，并进行深度优先搜索和广度优先搜索.c：

//图的建立和遍历  
#include <stdio.h>  
#include <stdlib.h>  
#include <stdbool.h>  
/\*顶点数，边数：5 8  
  行数：5   
测试矩阵：0 1 0 1 1  
1 0 1 1 0  
0 1 0 1 1  
1 1 1 0 1  
1 0 1 1 0\*/   
#define MAXV 100   
#define INF 9999  
#define MaxSize 100  
typedef struct ANode{  
int adjvex;//邻接点  
struct ANode \*nextarc;//指向下一个的指针   
int weight; //权值   
}ArcNode;//边结点   
typedef struct VNode{  
int data;//顶点值   
ArcNode \*firstarc;//指向第一个边结点   
}VNode;//头结点   
typedef struct{  
VNode adjlist[MAXV];//邻接表的头结点数组  
int n,e;//顶点数n和边数e   
}AdjGraph;//完整的图邻接表类型    
typedef struct {  
int data[MaxSize];  
int front, rear;  
} SqQueue;  
  
void InitQueue(SqQueue \*&q) {  
q = (SqQueue \*)malloc(sizeof(SqQueue));  
q->front = q->rear = 0;  
}  
  
bool QueueEmpty(SqQueue \*q) {  
return (q->front == q->rear);  
}  
  
bool enQueue(SqQueue \*&q, int e) {  
if ((q->rear + 1) % MaxSize == q->front)  
return false;  
q->rear++;  
q->data[q->rear] = e;  
return true;  
}  
  
bool deQueue(SqQueue \*&q, int &e) {  
if (q->front == q->rear)  
return false;  
q->front++;  
e = q->data[q->front];  
return true;  
}  
  
void CreatAdj(AdjGraph\*& G,int A[MAXV][MAXV],int n,int e){  
int i,j;ArcNode \*p;  
G = (AdjGraph\*)malloc(sizeof(AdjGraph));  
for(i=0;i<n;i++)  
G->adjlist[i].firstarc = NULL;  
for(i=0;i<n;i++)  
for(j=n-1;j>=0;j--)//遍历邻接矩阵  
if(A[i][j]!=0&&A[i][j]!=INF){  
p = (ArcNode\*)malloc(sizeof(ArcNode));//创立边结点   
p->adjvex = j;  
p->weight = A[i][j];  
p->nextarc = G->adjlist[i].firstarc;//头插法  
G->adjlist[i].firstarc = p;   
}  
G->n = n;  
G->e = e;   
}  
int visited1[MAXV]={0};  
void DFS(AdjGraph\* G,int v){//深度优先遍历   
ArcNode \*p;  
visited1[v] = 1;//标记已被访问的顶点   
printf("%d ",v);//输出已被访问顶点的编号   
p = G->adjlist[v].firstarc;  
while(p!=NULL){  
if(visited1[p->adjvex]==0){  
DFS(G,p->adjvex);  
}  
p = p->nextarc;   
}  
}  
int visited[MAXV] = {0};  
void BFS(AdjGraph \*G,int v){//广度优先 遍历   
int w,i;  
ArcNode \*p;  
SqQueue \*qu;  
InitQueue(qu);  
int visted[MAXV];  
for(i=0;i<G->n;i++)  
visited[i] = 0;//访问标记数组初始化   
printf("%d ",v);//输出被访问顶点的编号   
visited[v] = 1;//标记已被访问的顶点   
enQueue(qu,v);  
while(!QueueEmpty(qu)){  
deQueue(qu,w);//出队一个顶点  
p = G->adjlist[w].firstarc;//指向第一个边结点  
while(p!=NULL){  
if(visited[p->adjvex]==0){  
printf("%d ",p->adjvex);  
visited[p->adjvex] = 1;  
enQueue(qu,p->adjvex);//访问后该顶点进队   
}  
p = p->nextarc;  
}   
}  
printf("\n");  
}  
  
void printDgraph(AdjGraph \*G)  
{  
    int i;ArcNode \*p;  
    for(i=0;i<G->n;i++){  
     p = G->adjlist[i].firstarc;  
     printf("%3d: ",i);  
     while(p!=NULL){  
     printf("%3d[%d]->",p->adjvex,p->weight);  
     p = p->nextarc;  
}  
printf("Λ\n");  
}  
}  
  
void welcome()//打印选择界面  
{  
    printf("\t\t\t\t\t╔════════════════════════════════╗\n");  
    printf("\t\t\t\t\t║             MENU               ║\n");  
    printf("\t\t\t\t\t║════════════════════════════════║\n");  
    printf("\t\t\t\t\t║         1--Create Graph        ║\n");  
    printf("\t\t\t\t\t║════════════════════════════════║\n");  
    printf("\t\t\t\t\t║         2--DFS    Graph        ║\n");  
    printf("\t\t\t\t\t║════════════════════════════════║\n");  
    printf("\t\t\t\t\t║         3--BFS    Graph        ║\n");  
    printf("\t\t\t\t\t║════════════════════════════════║\n");  
    printf("\t\t\t\t\t║         0--Exit                ║\n");  
    printf("\t\t\t\t\t╚════════════════════════════════╝\n");  
    printf("请输入你的选择:\n");  
}  
int main(){  
AdjGraph \*g;  
int A[MAXV][MAXV];  
int n,e,choice,row;  
int a,b;  
while(1){  
welcome();  
scanf("%d",&choice);  
switch(choice){  
case 1:  
system("cls");  
printf("请输入图的顶点数和边数：");  
scanf("%d%d",&n,&e);  
printf("请输入邻接矩阵的行数：");  
scanf("%d",&row);  
printf("输入矩阵的元素值：(9999视为无穷大)\n");  
for(int i=0;i<row;i++)  
for(int j=0;j<row;j++)  
scanf("%d",&A[i][j]);  
CreatAdj(g,A,n,e);  
printf("建立成功！图如下：\n");  
printDgraph(g);  
break;  
case 2:  
system("cls");  
printf("请输入指定的搜索点：");  
scanf("%d",&a);  
printf("深搜结果如下：\n");  
DFS(g,a);  
for(int i=0;i<MAXV;i++) visited1[i] = 0;  
break;  
case 3:  
system("cls");  
printf("请输入指定的搜索点：");  
scanf("%d",&b);  
printf("广搜结果如下：\n");  
BFS(g,b);  
for(int i=0;i<MAXV;i++) visited[i] = 0;  
break;  
case 0:exit(1);  
default:printf("输入有误！");break;  
}  
}   
return 0;  
}

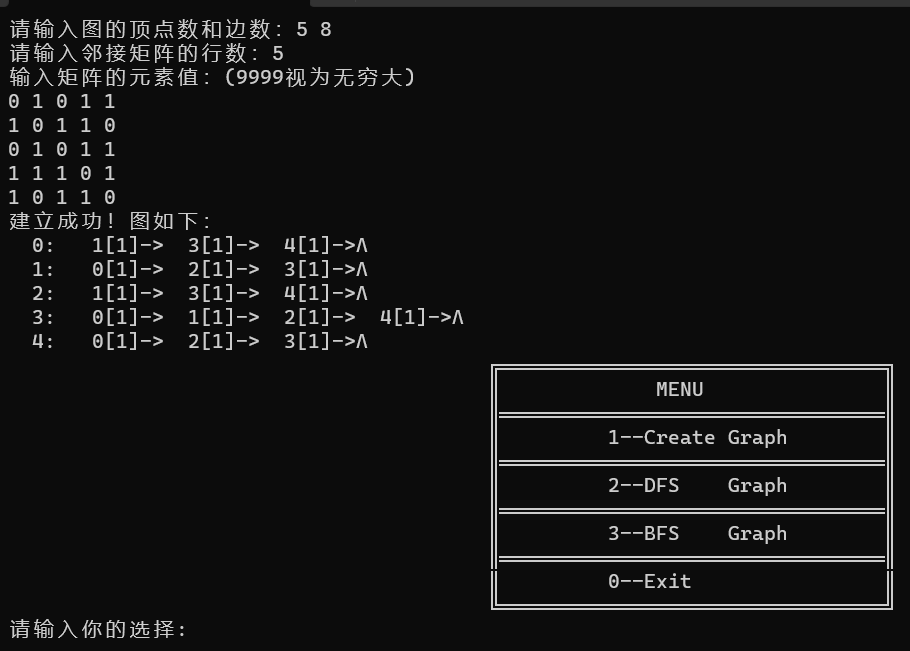
1. 调试分析
2. 采用IDE中自带的调试功能进行调试，手动添加断点和查看程序。
3. 对设计和编码的讨论和分析。该程序实现了顺序栈的操作。分析程序代码的质量，主要从以下几个方面考虑。 

* 正确性。在一定的数据范围内，该程序能实现所需功能，所以正确性是没有问题的。
* 健壮性。在一定的数据输入范围内，该程序能较好的实现链表的操作。但是如果输入数 据非法，该程序还是可能会产生一些预想不到的输出结构，或是不做任何处理。所以， 该程序的健壮性有待进一步的提高。要综合考虑一些情况，当输入有误时，应返回一个 表示错误的值，并中止程序的执行，以便在更高的抽象层次上进行处理。

1. 使用说明

按照屏幕提示，输入对应的数据，按下ENTER键后，根据屏幕提示输入对应功能的数字。

1. 测试程序的运行结果



1. 心得体会

在进行图的遍历时，我们通常会使用深度优先遍历或广度优先遍历等算法。深度优先遍历可以让我们从某一节点开始，一直遍历到最深处，直到无法继续下去后再返回，并继续遍历其他分支。而广度优先遍历则是逐层遍历节点，从某一层的节点依次遍历到下一层节点。这些算法在实现时需要根据具体场景进行选择，以便更好地达到预期的效果。总的来说，图的建立和遍历都是非常重要的。

附录：源程序文件清单

各程序源代码文件随本实验报告电子版一起打包，存放在文件夹从键盘输入的数据建立图，并进行深度优先搜索和广度优先搜索中。

文件清单如下：

从键盘输入的数据建立图，并进行深度优先搜索

和广度优先搜索.c…………………………………………图的建立与遍历的实现

1. 利用最小生成树算法解决通信网的总造价最低问题（设计性实验）
2. 需求分析

需求：

若在n个城市之间建通信网络，只需架设n−1条线路即可。如何以最低的经济代价建设这个通信网是一个网的最小生成树问题。

基本要求

以邻接表为存储结构，利用Prim算法或Kruskal算法求网的最小生成树。

1. 输入的形式和输入值的范围：图类型，点数，边数，权值
2. 输出的形式：打印最小生成树的边
3. 程序所能达到的功能：利用Prim算法求图的最小生成树并打印
4. 测试数据：

\*输入数据示例：

8 11

a

b

c

d

e

f

g

h

1 2 8

1 3 3

2 4 2

3 4 1

4 5 6

4 6 14

5 6 4

5 7 12

6 8 9

6 7 2

7 8 11\*/

1. 概要设计
2. 为了实现程序功能，需要定义图的抽象数据类型。

ADT：图（Graph）

Data：顶点的有穷非空集合和边集合

typedef struct ArcNode{

int adjvex;

int weigth;

struct ArcNode \*nextarc;

}ArcNode; //邻接表结构

typedef struct VNode{

VertexType vexdata;

ArcNode \*firstarc;

}VNode; //点节点

typedef struct{

VNode \*vertices;

int vexnum;

int arcnum;

GraphKind kind[4];

}ALGraph; //图结构

typedef struct{

int first;

int tail;

int weigth;

}edge; // 边结构

Operation：

Graph(V)：按照顶点集 VV 初始化图。

addVex(v)：在图 G 中添加新顶点 vv。

deleteVex(v)：删除图 G 中顶点 vv 及其相关弧。

addEdge(v1,v2)：在图 G 中添加弧<v1,v2><v1,v2>，若 G 是无向图，则还需要增添对称弧 <v2,v1><v2,v1>。

deleteEdge(v1,v2)：在图 G 中删除弧<v1,v2><v1,v2>，若 G 是无向图，则还需要删除对称弧 <v2,v1><v2,v1>。

DFSTraverse(v)：从顶点 vv 开始对图 G 进行深度优先遍历。

BFSTraverse(v)：从顶点 vv 开始对图 G 进行广度优先遍历。

1. 程序栈操作的实现包含4个函数：

* 主函数main()；
* 查找节点LocateVex ()；
* 创建图 CreateGraph ()；
* Prim算法求最小生成树 MiniSpanTree\_PRIM ()；

1. 详细设计

利用最小生成树算法解决通信网的总造价最低问题.c：

1. #include<stdio.h>
2. #include<stdlib.h>
3. #define MaxVertices 100
4. #define INF 65535
5. int graph[MaxVertices][MaxVertices];
6. //建立边表
7. typedef struct node
8. {
9. int adjvex; //指向边表结点
10. struct node \*nextarc; //指向下一条边，没有则为NULL
11. int info; //权值
12. }ArcNode;
13. //建立顶点表
14. typedef struct
15. {
16. char data[MaxVertices]; //顶点数据
17. ArcNode \*firstarc; //指向该顶点的第一个边表结点
18. }VerNode;
19. //建立图
20. typedef struct
21. {
22. VerNode adjlist[MaxVertices];//顶点数组
23. int n; //总顶点的个数
24. int e; //总的边数
25. }ALGraph;
26. //建立邻接表
27. void CrateGraph(ALGraph \*G)
28. {
29. int i,j, k;
30. int v1, v2; //输入的两个边
31. int weight; //权值
32. printf("请输入邻接表的总顶点数和边数：");
33. scanf("%d %d", &G->n, &G->e);
34. for (i = 1; i <= G->n; i++) //prim最小生成树初始化
35. {
36. for (j = 1; j <= G->e; j++)
37. {
38. graph[i][j] = INF;
39. }
40. }
41. fflush(stdin);
42. printf("请输入顶点（回车结束一个输入）\n");
43. for (i = 0; i < G->n; i++)
44. {
45. gets(G->adjlist[i].data);
46. G->adjlist[i].firstarc = NULL; //置头结点后第一条边为空
47. }
48. printf("请根据顶点的位置从1开始输入两两相邻的边和权值(中间用空格隔开)\n");
49. for (k = 0; k < G->e; k++)
50. {
51. fflush(stdin);//清空输入缓冲区
52. printf("第%d个连接点：", k + 1);
53. scanf("%d %d %d", &v1, &v2, &weight); //输入相邻的边和权值
54. graph[v1][v2] = weight;
55. graph[v2][v1] = weight;//类似于构造邻接矩阵
56. v1--; v2--; //按顶点排序从1开始，减去1方便数组计算
57. ArcNode \*s = (ArcNode \*)malloc(sizeof(ArcNode));
58. s->info = weight;
59. s->adjvex = v2;
60. s->nextarc = G->adjlist[v1].firstarc;
61. G->adjlist[v1].firstarc = s;//相当于利用邻接矩阵建立邻接表
62. }
63. }
64. //打印邻接表
65. void DisGraph(ALGraph \*G)
66. {
67. int i;
68. printf("序号\t顶点名称\t\t边表\n");
69. for (i = 0; i < G->n; i++)
70. {
71. printf("%d\t", i+1); //打印序号
72. printf("%s\t\t->", G->adjlist[i].data); //打印顶点
73. while (G->adjlist[i].firstarc)
74. {
75. printf("%d(%d)->", G->adjlist[i].firstarc->adjvex+1, G->adjlist[i].firstarc->info);
76. G->adjlist[i].firstarc = G->adjlist[i].firstarc->nextarc;
77. }
78. printf("NULL\n");
79. }
80. }
81. //prim最小生成树
82. int Prim(int graph[][MaxVertices], int n)
83. {
84. int lowcost[MaxVertices], closest[MaxVertices];
85. /\*
86. lowcost[i]记录以i为终点的边的最小权值，当lowcost[i]=0时表示终点i加入生成树
87. closest[i]记录对应lowcost[i]的起点，当closest[i]=0时表示起点i加入生成树
88. \*/
89. int i, j, min, minid, sum = 0;
90. for (i = 2; i <= n; i++) //默认选择1号节点加入生成树，从2号节点开始初始化
91. {
92. lowcost[i] = graph[1][i]; //最短距离初始化为其他节点到1号节点的距离
93. closest[i] = 1; //标记所有节点的起点皆为默认的1号节点
94. }
95. closest[1] = 0; //标记1号节点加入生成树
96. for (i = 2; i <= n; i++) //n个节点至少需要n-1条边构成最小生成树
97. {
98. min = INF;
99. for (j = 1; j <= n; j++) //找满足条件的最小权值边的节点minid
100. {
101. if (lowcost[j] < min && lowcost[j] != 0) //边权值较小且不在生成树中
102. {
103. min = lowcost[j];
104. minid = j;
105. }
106. }
107. printf("边(%d,%d)的权为：%d\n", closest[minid], minid, min); //输出生成树边的信息:起点，终点，权值
108. sum += min; //累加权值
109. lowcost[minid] = 0; //标记节点minid加入生成树
110. for (j = 2; j <= n; j++) //更新当前节点minid到其他节点的权值
111. {
112. if (graph[minid][j] < lowcost[j]) ///发现更小的权值
113. {
114. lowcost[j] = graph[minid][j]; //更新权值信息
115. closest[j] = minid; //更新最小权值边的起点
116. }//改变U后，U到其他顶点的权值发生变化，进行更新
117. }
118. }
119. return sum; //返回最小权值和
120. }
121. int main()
122. {
123. ALGraph G;
124. CrateGraph(&G);
125. printf("\n");
126. DisGraph(&G);
127. printf("\n最小生成树的权值之和:%d\n", Prim(graph, G.n));
128. return 0;
129. }
130. /\*输入数据示例：
131. 8 11
132. a
133. b
134. c
135. d
136. e
137. f
138. g
139. h
140. 1 2 8
141. 1 3 3
142. 2 4 2
143. 3 4 1
144. 4 5 6
145. 4 6 14
146. 5 6 4
147. 5 7 12
148. 6 8 9
149. 6 7 2

7 8 11\*/

4.调试分析

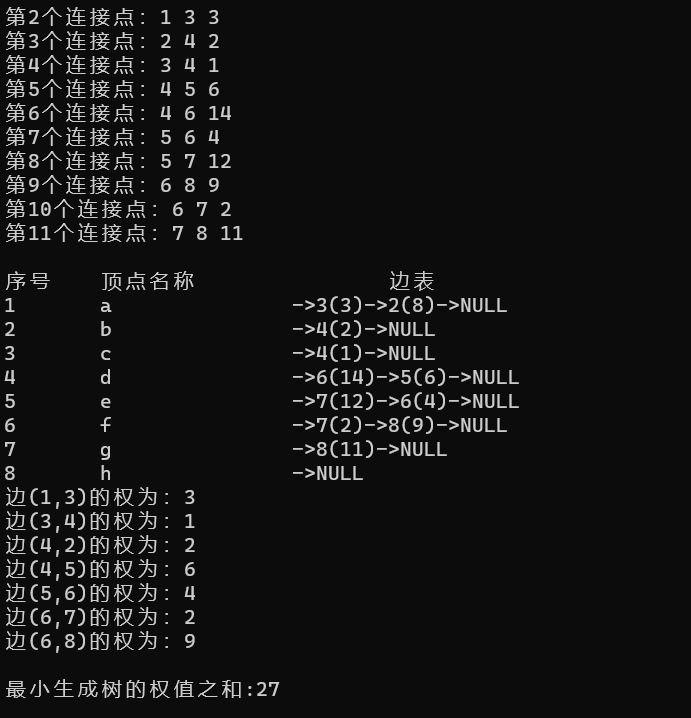
1. 采用IDE中自带的调试功能进行调试，手动添加断点和查看程序。
2. 对设计和编码的讨论和分析。该程序实现了城市链表的操作。分析程序代码的质量，主要从以下几个方面考虑。 

* 正确性。在一定的数据范围内，该程序能实现所需功能，所以正确性是没有问题的。
* 健壮性。在一定的数据输入范围内，该程序能较好的实现链表的操作。但是如果输入数 据非法，该程序还是可能会产生一些预想不到的输出结构，或是不做任何处理。所以， 该程序的健壮性有待进一步的提高。要综合考虑一些情况，当输入有误时，应返回一个 表示错误的值，并中止程序的执行，以便在更高的抽象层次上进行处理。

5.使用说明

按照屏幕提示，将文件放入对应文件夹中。按ENTER键继续，结果生成在文件' result.txt'中。

6.测试程序的运行结果



1. 心得体会

最小生成树算法是一种贪心算法，其基本思路是按照边权重从小到大依次构建生成树，不断扩大生成树的范围，直到生成一颗覆盖全部节点的生成树。这种贪心策略可以使得生成的最小生成树中边的数量最小，从而实现通信网络建设中的成本最小化。总的来说，通过实践最小生成树算法解决通信网的总造价最低问题，我深刻领会到算法设计的实际应用，并且对算法的时间复杂度、优化等方面有了更为深入的了解。

附录：源程序文件清单

各程序源代码文件随本实验报告电子版一起打包，存放在文件夹利用最小生成树算法解决通信网的总造价最低问题中。

文件清单如下：

利用最小生成树算法解决通信网的总造价最低问题.c…………………包含图的建立与遍历基本操作的头文件

1. 导游问题（综合性实验）
2. 需求分析

需求：

给出一张某公园的导游图，游客通过终端询问可知：

(1) 从某一景点到另一个景点的最短路径；

(2) 游客从公园大门进入，选一条最佳路线，使游客可以不重复的游览各景点，最后回到出口。

基本要求

(1) 将导游图看作一张带权无向图，顶点表示公园的各个景点，边表示各景点之间的道路，边上的权值表示距离，选择适当的数据结构。

(2) 为游客提供图中任意景点相关信息的查询。

(3) 为游客提供任意两个景点之间最短的简单路径。

(4) 为游客选择最佳游览路径。

分析：

1. 输入的形式和输入值的范围：将导游图数据放入文件“data.txt”，根据屏幕提示输入对应功能的数字和字符。
2. 输出的形式：与输入对应的字符串。
3. 程序所能达到的功能：

显示景点，查看介绍，最短路径，最佳路线。

1. 测试数据：

1

2

大门

3

景点A

景点B

4

5

1. 概要设计
2. 为了实现程序功能，需要定义图的抽象数据类型。

ADT：图（Graph）

Data：顶点的有穷非空集合和边集合

typedef struct ArcNode{

int adjvex;

int weigth;

struct ArcNode \*nextarc;

}ArcNode; //邻接表结构

typedef struct VNode{

char vexdata[NAME];

char information[introduce];

ArcNode \*firstarc;

}VNode; // 点结构

typedef struct{

VNode \*vertices;

int vexnum;

int arcnum;

}ALGraph; //图结构

typedef struct{

int first;

int tail;

int weigth;

}edge; // 边结构

Operation：

Graph(V)：按照顶点集 VV 初始化图。

addVex(v)：在图 G 中添加新顶点 vv。

deleteVex(v)：删除图 G 中顶点 vv 及其相关弧。

addEdge(v1,v2)：在图 G 中添加弧<v1,v2><v1,v2>，若 G 是无向图，则还需要增添对称弧 <v2,v1><v2,v1>。

deleteEdge(v1,v2)：在图 G 中删除弧<v1,v2><v1,v2>，若 G 是无向图，则还需要删除对称弧 <v2,v1><v2,v1>。

DFSTraverse(v)：从顶点 vv 开始对图 G 进行深度优先遍历。

BFSTraverse(v)：从顶点 vv 开始对图 G 进行广度优先遍历。

1. 本程序包含6个函数：

* 主函数main()；
* 查找节点search ()；;
* 计算最佳游览路径route ()；
* 打印最佳游览路径optimal\_sightseeing\_route ()；
* 求最短路径dijkstra ()；
* 创建图 CreateGraph ();

1. 详细设计

导游问题.c：

#include<stdio.h>

#include<stdlib.h>

#define UNVISITED 0

#define VISITED 1

#define INFINITY INT\_MAX

#define UNSELECTED 0

#define SELECTED 1

typedef struct

{

int symbol;//景点代号

const char \*name;//景点名字

const char \*introduction;//景点简介

}spot; //景点类型

typedef struct

{

spot v,w; //边的端点

int info; //对带权图，为权值，此处为距离

}ArcInfo; //存储边信息

typedef struct

{

spot \*vexs; //顶点数组，spot是景点类型

int \*\*arcs; //关系数组，此图为带权图，则为权值或INFINITY

int n,e; //顶点数和边数

int \*tags; //标志数组，可用于在图的遍历中标记顶点访问与否

}MGraph;

typedef struct

{

int prev; //当前最短路径上该顶点的前驱顶点的位序

int lowcost;//当前最短路径的长度

}DistInfo;//V-U中定点的当前最短路径信息

int LocateVex(MGraph G,spot v)//找顶点位置

{

int i;

for(i=0;i<G.n;i++)

if(v.symbol==G.vexs[i].symbol) return i;

return -1;

}

int InitGraph(MGraph &G,spot \*vexs,int n)

{

//初始化含n个顶点且无边的图G

int i,j,info;

if(n<0||(n>0&&vexs==NULL)) return 0;

info=INFINITY; //带权图

G.n = n;

G.e = 0; //顶点数和边数

if(G.n==0) return 1; //空图

if((G.vexs=(spot\*)malloc(n\*sizeof(spot)))==NULL) return 0;

for(i=0;i<G.n;i++) G.vexs[i]=vexs[i];

if((G.arcs=(int \*\*)malloc(n\*sizeof(int\*)))==NULL) //分配指针数组

return 0;

for(i=0;i<n;i++) //分配每个指针所指向的数组

if((G.arcs[i]=(int\*)malloc(n\*sizeof(int)))==NULL)

return 0;

if((G.tags=(int\*)malloc(n\*sizeof(int)))==NULL) return 0;

for(i=0;i<n;i++){ //初始化标志数组和关系数组

G.tags[i]=UNVISITED;

for(j=0;j<n;j++) G.arcs[i][j]=info;

}

return 1;

}

int CreateGraph(MGraph &G,spot \*vexs,int n,ArcInfo \*arcs,int e)

{

//创建含n个顶点和e条边的无向带权图图G，vexs为顶点信息，arcs为边信息

if(n<0||e<0||(n>0&&vexs==NULL)||(e>0&&arcs==NULL)) return 0;

int i,j,k;

spot v,w;

if(InitGraph(G,vexs,n)!=1) return 0; //初始化

G.e = e;//边数

for(k=0;k<G.e;k++){//建立关系数组

v=arcs[k].v; w=arcs[k].w; //读入边(v,w)

i=LocateVex(G,v);j=LocateVex(G,w);//确定v和w在顶点数组中的位序i和j

if(i<0||j<0) return 0;

G.arcs[i][j]=G.arcs[j][i]=arcs[k].info;

}

return 1;

}

int FirstAdjVex(MGraph G,int k)

{//求图G中k顶点的第一个邻接顶点的位序

int i;

if(k<0||k>=G.n) return -1;

for(i=0;i<G.n;i++){//查找第一个值非零且非INFINITY的元素

if(G.arcs[k][i]!=0&&G.arcs[k][i]!=INFINITY) return i;

}

return -1;//k顶点无邻接顶点

}

int NextAdjVex(MGraph G,int k,int m)

{

//m顶点为k顶点的邻接顶点，求图中k顶点相对于m顶点的下一个邻接顶点的位序

int i;

if(k<0||k>=G.n||m<0||m>=G.n) return 0;

for(i=m+1;i<G.n;i++)

if(G.arcs[k][i]!=0&&G.arcs[k][i]!=INFINITY)

return i;

return -1;

}

int visit(int k,MGraph G)

{

//访问结点信息（读取景点信息）

if(k<0||k>=G.n) return 0;

printf("景点序号：%d\n",G.vexs[k].symbol);

printf("景点名字：%s\n",G.vexs[k].name);

printf("景点简介：%s\n",G.vexs[k].introduction);

return 1;

}

int Dijkstra(MGraph G,int i,DistInfo\* &Dist)

{

//求图G中从i顶点到其他所有顶点的最短路径，并由Dist返回

//迪杰斯特拉算法

int j,m,k,min,p;

Dist = (DistInfo\*)malloc(G.n\*sizeof(DistInfo));

for(j=0;j<G.n;j++){

//初始化

Dist[j].lowcost = INFINITY;

G.tags[j] = UNSELECTED;

}

for(j=0;j<G.n;j++){

//源点i引出的所有弧的信息存入Dist

if(G.arcs[i][j]!=INFINITY){

Dist[j].prev=i;

Dist[j].lowcost=G.arcs[i][j];

}

}

Dist[i].prev=-1;Dist[i].lowcost=0;//源点i信息存入Dist

G.tags[i]=SELECTED;//初始集合U仅含源点i

for(m=1;m<G.n;m++){ //按路径长度升序，依次求源点到其他定点的最短路径

min=INFINITY;k=0;

for(j=0;j<G.n;j++)

{

if(UNSELECTED==G.tags[j]&&Dist[j].lowcost<min)

{

k=j;

min=Dist[k].lowcost; //此处求得在V-U集合中的最短路径

}

}

G.tags[k]=SELECTED; //将k顶点加入集合U

for(p=FirstAdjVex(G,k);p>=0;p=NextAdjVex(G,k,p)){ //更新Dist数组，这里做了一个简单优化，不用从0开始循环

if(UNSELECTED==G.tags[p]&&Dist[k].lowcost+G.arcs[k][p]<Dist[p].lowcost){

Dist[p].lowcost = Dist[k].lowcost+G.arcs[k][p]; //k点的邻接点中距离最小的点p

Dist[p].prev=k;

}

}

}

return 1;

}

void Dispath(MGraph G,DistInfo \*Dist,int k)

{

//沿Dist数组的prev域，可递归获得源点到k定点的最短路径

if(k==-1) return ;

Dispath(G,Dist,Dist[k].prev);//逆向递归获得路径上的顶点

printf("%d.%s ",G.vexs[k].symbol,G.vexs[k].name); //正向输出当前路径上的顶点

}

void initdata(spot \*vexs,ArcInfo \*arcs)

{

vexs[0].symbol=1;

vexs[0].name="大门";

vexs[0].introduction="公园的正门，雄伟壮观的大理石做的门，体现了公园的宏伟壮观";

vexs[1].symbol=2;

vexs[1].name="荷花池";

vexs[1].introduction="一到夏日，满池的荷花鲜艳绽放";

vexs[2].symbol=3;

vexs[2].name="九曲桥";

vexs[2].introduction="九曲桥蜿蜒在荷花池中央，游客可在桥上近距离赏花";

vexs[3].symbol=4;

vexs[3].name="湖中亭";

vexs[3].introduction="建在荷花池中央，便于游客停下休息";

vexs[4].symbol=5;

vexs[4].name="游乐场";

vexs[4].introduction="各种游乐设施齐全，可供游客尽情玩耍，释放自我";

vexs[5].symbol=6;

vexs[5].name="纪念塔";

vexs[5].introduction="这是纪念红军的纪念塔，满含红色革命精神";

vexs[6].symbol=7;

vexs[6].name="观景台";

vexs[6].introduction="观景台是地势公园最高的地方，在此，可以鸟瞰到整个公园的风景";

vexs[7].symbol=8;

vexs[7].name="餐厅";

vexs[7].introduction="餐厅内有各种食物，可为游客解渴充饥";

vexs[8].symbol=9;

vexs[8].name="休息区";

vexs[8].introduction="配备房间，游客累了能在此休息";

vexs[9].symbol=10;

vexs[9].name="自行车道";

vexs[9].introduction="有各种款式的自行车，也有专门的自行车道，方便1骑行";

vexs[10].symbol=11;

vexs[10].name="游船港";

vexs[10].introduction="有双人、多人复古风游船，能让游客尽情地享受水上的快乐时光";

//边信息

arcs[0].v=vexs[0];

arcs[0].w=vexs[1];

arcs[0].info=1;

arcs[1].v=vexs[1];

arcs[1].w=vexs[2];

arcs[1].info=3;

arcs[2].v=vexs[1];

arcs[2].w=vexs[3];

arcs[2].info=4;

arcs[3].v=vexs[2];

arcs[3].w=vexs[3];

arcs[3].info=6;

arcs[4].v=vexs[3];

arcs[4].w=vexs[4];

arcs[4].info=1;

arcs[5].v=vexs[1];

arcs[5].w=vexs[5];

arcs[5].info=4;

arcs[6].v=vexs[2];

arcs[6].w=vexs[5];

arcs[6].info=4;

arcs[7].v=vexs[3];

arcs[7].w=vexs[5];

arcs[7].info=5;

arcs[8].v=vexs[3];

arcs[8].w=vexs[6];

arcs[8].info=6;

arcs[9].v=vexs[2];

arcs[9].w=vexs[7];

arcs[9].info=7;

arcs[10].v=vexs[5];

arcs[10].w=vexs[6];

arcs[10].info=9;

arcs[11].v=vexs[5];

arcs[11].w=vexs[7];

arcs[11].info=6;

arcs[12].v=vexs[5];

arcs[12].w=vexs[8];

arcs[12].info=4;

arcs[13].v=vexs[5];

arcs[13].w=vexs[9];

arcs[13].info=6;

arcs[14].v=vexs[3];

arcs[14].w=vexs[9];

arcs[14].info=7;

arcs[15].v=vexs[6];

arcs[15].w=vexs[10];

arcs[15].info=3;

arcs[16].v=vexs[7];

arcs[16].w=vexs[8];

arcs[16].info=4;

arcs[17].v=vexs[8];

arcs[17].w=vexs[9];

arcs[17].info=5;

arcs[18].v=vexs[9];

arcs[18].w=vexs[10];

arcs[18].info=5;

}

void welcom()

{

printf("\t\t\t\t\t\t╔════════════════════════════════╗\n");

printf("\t\t\t\t\t\t║ 公园导游系统 ║\n");

printf("\t\t\t\t\t\t║════════════════════════════════║\n");

printf("\t\t\t\t\t\t║ 1-查看景点信息 ║\n");

printf("\t\t\t\t\t\t║════════════════════════════════║\n");

printf("\t\t\t\t\t\t║ 2-查看两景点间最短路径 ║\n");

printf("\t\t\t\t\t\t║════════════════════════════════║\n");

printf("\t\t\t\t\t\t║ 3-退出系统 ║\n");

printf("\t\t\t\t\t\t╚════════════════════════════════╝\n");

}

void spotList()

{

printf("\t\t\t\t\t\t\*\*\*\*\*\*\*\*\*\*\*景点列表\*\*\*\*\*\*\*\*\*\*\*\* \n");

printf("\t\t\t\t\t ------------------------------------------\n");

printf("\t\t\t\t\t 1.大门 2.荷花池 3.九曲桥 4.湖中亭\n");

printf("\t\t\t\t\t 5.游乐场 6.纪念塔 7.观景台 8.餐厅 \n");

printf("\t\t\t\t\t 9.休息区 10.自行车道 11.游船港 \n");

printf("\t\t\t\t\t ------------------------------------------\n");

}

int main()

{

int n=11,e=19,sum=0,flag=0;

spot \*vexs;

ArcInfo \*arcs;

int \*path; //用于存储路径

int length; //当前路径长度

int choice=INFINITY,i,j;

vexs=(spot\*)malloc(n\*sizeof(spot)); //分配空间

arcs=(ArcInfo\*)malloc(e\*sizeof(ArcInfo));

path=(int\*)malloc(e\*sizeof(int));

length=0;

MGraph G;

DistInfo \*Dist; //定义数组

initdata(vexs,arcs);

CreateGraph(G,vexs,n,arcs,e);

while(1)

{

welcom();

spotList();

printf(" \n| 请选择功能 |\n");

scanf("%d",&choice);

switch(choice)

{

case 1:

system("cls");

spotList();

printf("你想查看哪个景点的信息（1-%d）？\n",G.n);

scanf("%d",&i);

while(i<=0||i>G.n)

{

printf("你输入的数据不合法，请重新输入\n");

scanf("%d",&i);

}

visit(i-1,G);

break;

case 2:

system("cls");

spotList();

for(i=0;i<n;i++)

{ //初始化标志数组

G.tags[i]=UNVISITED;

}

printf("输入第一个点的序号（1-%d）\n",G.n); //算最短路径

scanf("%d",&i);

while(i<=0||i>G.n)

{

printf("你输入的数据不合法，请重新输入\n");

scanf("%d",&i);

}

printf("输入第二个点的序号（1-%d）\n",G.n);

scanf("%d",&j);

while(j<=0||j>G.n)

{

printf("你输入的数据不合法，请重新输入\n");

scanf("%d",&j);

}

Dijkstra(G,i-1,Dist); //迪杰斯特拉算法求出Dist数组

if(Dist[j-1].lowcost==INFINITY)

{

printf("没有连通\n");

break;

}

else

{

printf("路线：");

Dispath(G,Dist,j-1); //递归输出

printf("\n");

printf("路径长度为:%d米\n",Dist[j-1].lowcost\*100);

break;

}

case 3:exit(1);

default:system("cls");printf("输入数据不合法，请重新选择\n");

}

}

return 0;

}

1. 调试分析
2. 采用IDE中自带的调试功能进行调试，手动添加断点和查看程序。
3. 对设计和编码的讨论和分析。该程序实现了城市链表的操作。分析程序代码的质量，主要从以下几个方面考虑。 

* 正确性。在一定的数据范围内，该程序能实现所需功能，所以正确性是没有问题的。
* 健壮性。在一定的数据输入范围内，该程序能较好的实现链表的操作。但是如果输入数 据非法，该程序还是可能会产生一些预想不到的输出结构，或是不做任何处理。所以， 该程序的健壮性有待进一步的提高。要综合考虑一些情况，当输入有误时，应返回一个 表示错误的值，并中止程序的执行，以便在更高的抽象层次上进行处理。

1. 使用说明

将导游图数据放入文件“data.txt”中，按照屏幕提示，输入对应功能的字符实现相应的功能。

1. 测试程序的运行结果



1. 心得体会

我认为在编写某公园的导游图程序时，需要掌握文件读写、命令行交互和图像显示等相关技术，同时需要关注游客的使用体验，让程序能够方便快捷地帮助游客了解公园的相关信息。

我们需要从文件中读取公园的导游图和相关资源，例如景点介绍、地图数据、照片等。这些信息需要存储在程序中，并且能够方便高效地随时读取和更新。。

附录：源程序文件清单

各程序源代码文件随本实验报告电子版一起打包，存放在文件夹导游问题中。

文件清单如下：

导游问题.c…………………………………………导游问题的实现