## 上机实验4：图

#### 班级：2022211305 学号：2022211119 姓名：赵宇鹏

### Email:zhaoyupeng@bupt.edu.cn

1. **实验内容：**

完成了实验所要求的使用邻接表储存图，深度优先遍历图，生成最小生成树以及求最短路径等功能，已完成选做求最短路径要求。

1. **算法与源程序结构：**

首先数据量较大且为稀疏图，采用邻接表对图进行储存，构建顶点域，边结点域以及访问标志数组等，程序有insert\_vertex, insert\_arc, print\_list, dfs, DFSreverse, spanning\_tree, print\_shortest\_path, shortest\_path, free\_graph等函数，分别用来插入点，插入边，打印邻接表，进行深度优先遍历，生成最小生成树以及寻找最短路径。插入顶点非常简单，但是要注意顶点不一定就是字母，测试样例中也有汉字，所以我们需要使用wcscpy函数来移动和储存顶点名;插入边类似于插入链表，每次往邻接表的末端接上一条边，但是由于是无向图，所以需要操作两次，分别对边相关的两个顶点进行插入操作。dfs算法的思路和二叉树的先序遍历算法非常类似，每次都往更深层进行递归访问。最小生成树由于测试样例中给出了开始结点，所以为使输出符合要求本人使用了Prim算法，邻接表的Prim算法思路和邻接矩阵类似，创建一个辅助结构体，每次遍历完成后加入距离最近的顶点即可，这里需要注意的是邻接矩阵的Prim算法中每次遍历需要遍历整个矩阵的一行，而邻接表只需要向后遍历到空即可，由于为系数图，后方节点并不多，时间复杂度低。最短路径算法使用的是迪杰斯特拉算法，思路和最小生成树算法非常类似，不同之处在于迪杰斯特拉算法的修正部分是加入一个新顶点后到达对应结点的距离与原路径相比较，其余部分思路类似，输出最短路径时，因为使用的是Pre数组，所以储存的是结点的前驱，可以使用递归算法一步一步往前递归寻找到开始结点再进行输出。

1. **实验总结**：

完成此次作业大概花费了三天大概二十小时，其中既有感冒发烧的原因，也有对上课时讲的算法不熟练的原因，因为ppt上的算法代码都是针对邻接矩阵写的，因此我花费了大量时间来思考如何将算法转化成适用于邻接表的算法，此外，刚开始构建邻接表时，在插入边时没有对两个顶点都进行插入操作，导致储存和输出的图是一个有向图而非无向图，此外，由于输入输出含有汉字，可是在vscode环境下输入输出汉字都是乱码，据了解，发现是编码问题，需要使用utf-8编码，在改过来之后输出仍然错误，此后又采用了使用wchar\_t，改变编码，改变环境，更换终端等方式，皆无果并因此花费了大量时间，最后使用外部终端来运行程序才得以正常输入和输出汉字，在输出最短路径时，使用case5样例时最短路径输出全为10000，打印出来的路径也只有目的地，由于最短路径的打印思路是逐步往前递归，所以很明显目的地对应的前置结点也没有找到，一开始认为可能是数据量过大导致邻接表储存失败，可是再额外测试了几个短距离最短路径时发现输出没有问题，后经检查发现赋给min的值为10000，可是在case5中最短路径有可能有几万甚至几十万，远远超过了10000的值，这导致在后期比较修正的时候都把10000当成最短路径，导致输出错误，此后将min的值设置为10000000，输出正常。

4、**代码及输出成果**

/\*深度优先遍历图；求最小生成树

要求：1.图为加权无向稀疏图，结构是静态的（不考虑删除边和节点）

2.自己设计图的存储结构。可以考虑邻接矩阵，下三角矩阵，邻接表，或其他存储方式。分析所采用的存储结构的空间复杂度和优缺点。

设计存储结构的使用界面，遍历图和最小生成树算法的代码尽量在不修改的前提下可以修改存储结构（体会一下如何减少模块耦合性，提高系统的灵活性）

3.最多支持10K个节点。

4.（选作）求指定顶点到另一指定顶点之间的最短路径

\*/

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<wchar.h>

#include <locale.h>

#define MAX 10000

//采用邻接表

struct ARCNODE {

    int adjvex;//邻接点域

    int weight;

    struct ARCNODE \*nextarc;//指向下一个边的指针域

};//边结点

struct VNODE{

    wchar\_t vertex[100]; //顶点信息域

    struct ARCNODE \*firstarc;//指向第一条边的指针

};//顶点结点

static int visited[MAX];//访问标志数组

struct graph{

    struct VNODE vex[MAX];//顶点结点数组 ，也可以malloc申请顺序表

    int vexnum;//顶点数

    int arcnum;//边数

};

struct graph \*create\_graph(int num){

    struct graph \*g = (struct graph \*)malloc(sizeof(struct graph));

    g->vexnum = num;

    g->arcnum = 0;

    return g;

}//创建图

void insert\_vertex(struct graph \*g, int num, wchar\_t v[]){

    wcscpy(g->vex[num].vertex, v);

    g->vex[num].firstarc = NULL;

}//插入顶点，顶点可能为汉字

void insert\_arc(struct graph \*g, int v1, int v2, int weight){

    struct ARCNODE \*p, \*q;

    p = (struct ARCNODE \*)malloc(sizeof(struct ARCNODE));

    p->adjvex = v2;

    p->weight = weight;

    p->nextarc = NULL;

    if(g->vex[v1].firstarc == NULL)

        g->vex[v1].firstarc = p;

    else{

        q = g->vex[v1].firstarc;

        while(q->nextarc != NULL)

            q = q->nextarc;

        q->nextarc = p;

    }

    p = (struct ARCNODE \*)malloc(sizeof(struct ARCNODE));

    p->adjvex = v1;

    p->weight = weight;

    p->nextarc = NULL;

    if(g->vex[v2].firstarc == NULL)

        g->vex[v2].firstarc = p;

    else{

        q = g->vex[v2].firstarc;

        while(q->nextarc != NULL)

            q = q->nextarc;

        q->nextarc = p;

    }

    g->arcnum++;

}//插入边

void print\_list(struct graph \*g){

    int i;

    struct ARCNODE \*p;

    for(i = 0; i < g->vexnum; i++){

        wprintf(L"%d:[%ls]", i, g->vex[i].vertex);

        p = g->vex[i].firstarc;

        while(p != NULL){

            wprintf(L" ->[%ls]%d", g->vex[p->adjvex].vertex, p->weight);

            p = p->nextarc;

        }

        printf("\n");

    }

}//打印邻接表

void dfs(struct graph \*g, int v){

    struct ARCNODE \*w;

    wprintf(L" %ls", g->vex[v].vertex);

    visited[v] = 1;

    for (w = g->vex[v].firstarc; w != NULL; w = w->nextarc){

        if (!visited[w->adjvex])

            dfs(g , w->adjvex);

    }

}//深度优先遍历

void DFSTraverse(struct graph \*g, int v){

    int i;

    for(i = 0; i < g->vexnum; i++)

        visited[i] = 0;

    wprintf(L"DFS(%ls):", g->vex[v].vertex);

    dfs(g, v);

    printf("\n");

}//深度优先遍历

struct closeedge{

    int adjvex;

    int lowcost;

}closeedge[MAX];

void spanning\_tree(struct graph \*g, int start){

    int i, j, k, min;

    int sum = 0;

    struct ARCNODE \*p;

    p = g->vex[start].firstarc;

    for (i = 0; i < g->vexnum; i++){

        closeedge[i].lowcost = MAX;

        closeedge[i].adjvex = start;

    }

    for (p = g->vex[start].firstarc; p; p = p->nextarc){

        closeedge[p->adjvex].lowcost = p->weight;

    }

    closeedge[start].lowcost = 0;

    wprintf(L"spanning tree(%ls):", g->vex[start].vertex);

    for (i = 1; i < g->vexnum; i++){

        min = MAX;

        for (j = 0; j < g->vexnum; j++) {

            if (closeedge[j].lowcost != 0 && closeedge[j].lowcost < min) {

                min = closeedge[j].lowcost;

                k = j;

            }

        }

        closeedge[k].lowcost = 0;

        wprintf(L" %ls-%ls(%d)", g->vex[closeedge[k].adjvex].vertex, g->vex[k].vertex, min);

        sum += min;

        for (p = g->vex[k].firstarc; p; p = p->nextarc) {

            if (closeedge[p->adjvex].lowcost != 0 && p->weight < closeedge[p->adjvex].lowcost){

                closeedge[p->adjvex].lowcost = p->weight;

                closeedge[p->adjvex].adjvex = k;

            }

        }

    }

    printf("\n");

    printf("Weight of tree: %d\n", sum);

}

void free\_graph(struct graph \*g){

    int i;

    struct ARCNODE \*p, \*q;

    for(i = 0; i < g->vexnum; i++){

        p = g->vex[i].firstarc;

        while(p != NULL){

            q = p;

            p = p->nextarc;

            free(q);

        }

    }

}

void print\_shortest\_path(struct graph \*g, int prev[], int start, int end){

    if(prev[end] != -1 && end != start){

        print\_shortest\_path(g, prev, start, prev[end]);

    }

    if(end == start)

        wprintf(L"%ls", g->vex[end].vertex);

    else

        wprintf(L"->%ls", g->vex[end].vertex);

}

void shortest\_path(struct graph \*g, int start, int end){

    wprintf(L"shortest path(%ls->%ls):", g->vex[start].vertex, g->vex[end].vertex);

    int i, j, k, min;

    struct ARCNODE \*p;

    int prev[MAX];

    int final[MAX];

    int D[MAX];

    for(i = 0; i < g->vexnum; i++){

        final[i] = 0;

        D[i] = 10000000;

        prev[i] = -1;

    }

    for(p = g->vex[start].firstarc; p; p = p->nextarc){

        D[p->adjvex] = p->weight;

        prev[p->adjvex] = start;

    }

    final[start] = 1;

    D[start] = 0;

    for(i = 1; i < g->vexnum; i++){

        min = 10000000;

        for(j = 0; j < g->vexnum; j++){

            if(!final[j] && D[j] < min){

                min = D[j];

                k = j;

            }

        }

        final[k] = 1;

        for(p = g->vex[k].firstarc; p; p = p->nextarc){

            if(!final[p->adjvex] && (min + p->weight) < D[p->adjvex]){

                D[p->adjvex] = min + p->weight;

                prev[p->adjvex] = k;

            }

        }

    }

    min = D[end];

    print\_shortest\_path(g, prev, start, end);

    printf("\n");

    printf("Distance: %d\n", min);

}//使用迪杰斯特拉算法求最短路径

int main(){

    setlocale(LC\_ALL, "");

    int start, end;

    char command[100];

    scanf("%s", command);

    while((strcmp(command, "CREATE"))){

        printf("输入错误\n");

        scanf("%s", command);

    }

    int num;

    printf("创建图\n");

    scanf("%d", &num);

    struct graph \*g = create\_graph(num);

    scanf("%s", command);

    while(strcmp(command, "BYE")){

        if(!strcmp(command, "CREATE")){

           printf("输入错误\n");

        }

        else if(!strcmp(command, "VERTEX")){

            int v;

            wchar\_t vertex[100];

            fflush(stdout);  // 刷新标准输出流，确保提示语句被立即显示

            printf("插入顶点\n");

            scanf("%d", &v);

            wscanf(L"%ls", vertex);

            insert\_vertex(g, v, vertex);

        }

        else if(!strcmp(command, "EDGE")){

            int v1, v2, weight;

            fflush(stdout);  // 刷新标准输出流，确保提示语句被立即显示

            printf("插入边\n");

            scanf("%d%d%d", &v1, &v2, &weight);

            insert\_arc(g, v1, v2, weight);

        }

        else if(!strcmp(command, "DUMP")){

            print\_list(g);

        }

        else if(!strcmp(command, "DFS")){

            fflush(stdout);  // 刷新标准输出流，确保提示语句被立即显示

            start = 0;

            printf("开始DFS\n");

            scanf("%d", &start);

            DFSTraverse(g, start);

        }

        else if(!strcmp(command, "SPANNING-TREE")){

            fflush(stdout);  // 刷新标准输出流，确保提示语句被立即显示

            printf("开始寻找最小生成树\n");

            scanf("%d", &start);

            spanning\_tree(g, start);

        }

        else if(!strcmp(command, "SHORTEST-PATH")){

            printf("开始寻找最小路径\n");

            scanf("%d", &start);

            scanf("%d", &end);

            shortest\_path(g, start, end);

        }

        else

            printf("输入错误\n");

        scanf("%s", command);

    }

    free\_graph(g);

    return 0;

}



