```
#include <stdio.h>
#include <stdlib.h>
#include<stdbool.h>//布尔类型 bool 以及两个常量值 true 和 false

#define MAX_VERTEX_NUM 20

//定义相关结构
//图的类型
typedef enum{
    DG,//DG(Directed Graph): 有向图
    DN,//DN(Directed Net)
    UDG,//UDG(Undirected Graph)
    UDN//UDN(Undirected Net): 无向网
} GraphKind;
//弧（边）结点
typedef struct Arcnode
{
    int adjvex; //Adjacent Vertex（邻接顶点）
    int weight;
    struct Arcnode *nextarc;
}Arcnode;
//顶点结点
typedef struct Vnode{
    char data;
    Arcnode *firstarc;
}Vnode,AdjList[MAX_VERTEX_NUM];
//图结构
typedef struct
{
    AdjList vertices;//vertices 是 vertex 的复数形式
    int vex_num,arc_num;
    int kind;
}ALGraph;
//栈结构（DFS）
typedef struct{
    int data[MAX_VERTEX_NUM];
    int top;
}Stack;
//队列结构(BFS)
typedef struct{
    int data[MAX_VERTEX_NUM];
    int front,rear;
}Queue;
//栈&队列操作
```

```
//初始化栈
void initStack(Stack *s){
	s->top=-1;
}
//入栈
void push(Stack *s,int value){
	s->top++;
	s->data[s->top]=value;
}
//出栈
int pop(Stack *s){
	s->top--;
	return s->data[s->top+1];
}
//判断是否栈空
bool isStackEmpty(Stack *s){
	return s->top==-1;
}
//初始化队列
void initQueue(Queue *q){
	q->front=q->rear=0;
}
//入队
void enQueue(Queue *q,int value){
	q->data[q->rear]=value;//队列先进先出
	q->rear++;
}
//出队
int deQueue(Queue *q){
	q->front++;
	return q->data[q->front-1];
}
//判断队列是否为空
bool isQueueEmpty(Queue *q){
	return q->front==q->rear;
}

//查找顶点在顶点数组中位置
int locateVertex(ALGraph *G, char v){
	for(int i=0;i<G->vex_num;i++){
		if(G->vertices[i].data==v)
			return i;
	}
	return -1;
```

```
}
//创建图
void createGraph(ALGraph *G){
    G->vex_num = 8;   // a,b,c,d,e,f,g,h
    G->arc_num = 13; // 边的数量
    G->kind = 0;      // 无向图

    // 初始化顶点
    char vertices[] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'};
    for (int i = 0; i < G->vex_num; i++) {
        G->vertices[i].data = vertices[i];
        G->vertices[i].firstarc = NULL;
    }

    // 添加边（按字母顺序） 边信息：起点，终点，权重
    int edges[][3] = {
        {'a', 'b', 4}, {'a', 'c', 3},
        {'b', 'a', 4}, {'b', 'c', 5}, {'b', 'd', 5}, {'b', 'e', 9},
        {'c', 'a', 3}, {'c', 'b', 5}, {'c', 'd', 5}, {'c', 'h', 5},
        {'d', 'b', 5}, {'d', 'c', 5}, {'d', 'e', 7}, {'d', 'f', 6}, {'d', 'g', 5}, {'d', 'h', 4},
        {'e', 'b', 9}, {'e', 'd', 7}, {'e', 'f', 3},
        {'f', 'd', 6}, {'f', 'e', 3}, {'f', 'g', 2},
        {'g', 'd', 5}, {'g', 'f', 2}, {'g', 'h', 6},
        {'h', 'c', 5}, {'h', 'd', 4}, {'h', 'g', 6}
    };
    for(int i=0;i<26;i++){// 26 条边记录（无向图每条边存两次）
        int u=locateVertex(G,(char)edges[i][0]);
        int v=locateVertex(G,(char)edges[i][1]);
        int w=edges[i][2];

        // 创建新的弧节点
        Arcnode *newArc = (Arcnode *)malloc(sizeof(Arcnode));
        newArc->adjvex = v;
        newArc->weight = w;
        // 找到插入位置（按邻接顶点字母顺序）
        Arcnode *p=G->vertices[u].firstarc;
        Arcnode *prev=NULL;
        while(p!=NULL&&G->vertices[p->adjvex].data<G->vertices[v].data){
            prev=p;//
            p=p->nextarc;
        }
        if(prev==NULL){
            newArc->nextarc=G->vertices[u].firstarc;
            G->vertices[u].firstarc=newArc;
        }
        else{
            newArc->nextarc = prev->nextarc;
            prev->nextarc = newArc;
        }
    }
}
//打印邻接表
void printGraph(ALGraph *G) {
    printf("图的邻接表表示:\n");
    for (int i = 0; i < G->vex_num; i++) {
        printf("%c: ", G->vertices[i].data);
        Arcnode *p = G->vertices[i].firstarc;
        while (p != NULL) {
            printf("->%c(%d) ", G->vertices[p->adjvex].data, p->weight);
            p = p->nextarc;
        }
        printf("\n");
    }
    printf("\n");
}
//深度优先搜索
void dfs(ALGraph *G, int start, int target) {
    printf("深度优先搜索(DFS)从 %c 到 %c:\n",
           G->vertices[start].data, G->vertices[target].data);

    bool visited[MAX_VERTEX_NUM]={false};//标记访问与否
    int parent[MAX_VERTEX_NUM];//标记前驱结点
    Stack stack;
    initStack(&stack);

    for(int i=0;i<G->vex_num;i++)
        parent[i]=-1;

    push(&stack,start);
    visited[start]=true;

    printf("访问顺序: ");

    while(!isStackEmpty(&stack)){
        int current=pop(&stack);//都是用的顶点的值
        printf("%c ", G->vertices[current].data);
        if (current == target) {
            printf("\n 找到目标顶点 %c!\n", G->vertices[target].data);
            break;
        }

        Arcnode *p=G->vertices[current].firstarc;
        Stack temp;
        initStack(&temp);
        while(p){// 先将所有邻居按顺序压入临时栈
            if(visited[p->adjvex]==false)
                push(&temp,p->adjvex);
            p=p->nextarc;
        }

        while(!isStackEmpty(&temp)){// 从临时栈弹出并压入主栈（实现逆序）
            int neighbor=pop(&temp);
            if(!visited[neighbor]){
                parent[neighbor]=current;
                visited[neighbor]=true;
                push(&stack,neighbor);
            }
        }
    }
    // 重建路径
    printf("路径: ");
    int path[MAX_VERTEX_NUM];
    int path_length = 0;
    int current = target;

    while (current!=-1)
    {
        path[path_length]=current;
        path_length++;
        current=parent[current];
    }
    for(int i=path_length-1;i>=0;i--){
        printf("%c", G->vertices[path[i]].data);
        if (i > 0)
            printf(" -> ");
    }
    printf("\n");
}
//广度优先搜索
void bfs(ALGraph *G, int start, int target){
    printf("广度优先搜索 (BFS) 从 %c 到 %c:\n",G->vertices[start].data,
G->vertices[target].data);
```

```
bool visited[MAX_VERTEX_NUM]={false};
int parent[MAX_VERTEX_NUM];
for (int i = 0; i < G->vex_num; i++)
        parent[i] = -1;
Queue queue;
initQueue(&queue);

enQueue(&queue,start);
visited[start]=true;

printf("访问顺序: ");

while(!isQueueEmpty(&queue)){
        int current=deQueue(&queue);
        printf("%c ", G->vertices[current].data);

        if (current == target) {
                printf("\n 找到目标顶点 %c!\n", G->vertices[target].data);
                break;
        }

        Arcnode *p=G->vertices[current].firstarc;
        while(p){// 按字母顺序入队
                if(!visited[p->adjvex]){
                        parent[p->adjvex]=current;
                        visited[p->adjvex]=true;
                        enQueue(&queue,p->adjvex);
                }
                p=p->nextarc;
        }
}
printf("路径: ");
int path[MAX_VERTEX_NUM];
int path_length = 0;
int current = target;

while (current != -1) {
        path[path_length++] = current;
        current = parent[current];
}

for (int i = path_length - 1; i >= 0; i--) {
        printf("%c", G->vertices[path[i]].data);
```

```
            if (i > 0) printf(" -> ");
        }
        printf("\n");
}
//释放图的内存
void freeGraph(ALGraph *G) {
        for (int i = 0; i < G->vex_num; i++) {
            Arcnode *p = G->vertices[i].firstarc;
            while (p != NULL) {
                Arcnode *temp = p;
                p = p->nextarc;
                free(temp);
            }
        }
}
//主函数
int main() {
        ALGraph G;

        // 创建图
        createGraph(&G);

        // 打印图的邻接表
        printGraph(&G);

        // 查找顶点位置
        int start = locateVertex(&G, 'a');
        int target = locateVertex(&G, 'g');

        if (start == -1 || target == -1) {
            printf("顶点不存在!\n");
            return -1;
        }

        // 执行 DFS
        dfs(&G, start, target);

        // 执行 BFS
        bfs(&G, start, target);

        // 释放内存
        freeGraph(&G);

        return 0;
```

```
}
```