实验四 图及其应用

班级：少年班学院少年班 姓名：邱子悦 学号：PB18000028 完成日期：2019.12.13

一. 实验要求

给定无向图G，默认边权为1，完成以下两个搜索算法的应用。

1. DFS的应用

基本要求:

- 参考教材P177-178,算法7.10和7.11，基于邻接矩阵的存储结构，使用非递归的深度优先搜索算法，求 无向连通图中的全部关节点，并按照顶点编号升序输出。

2. BFS的应用

基本要求：

- 基于邻接表的存储结构，依次输出从顶点0到顶点1、2、……、n-1的最短路径和各路径中的顶点信息。

选做要求:

- 将输入的无向图可视化，推荐使用 graphviz 。

二.设计思路

DFS的应用

- 邻接矩阵的定义

```
typedef int AdjMatrix[MAX][MAX];
typedef struct{
    int vexs[MAX];//顶点向量
    AdjMatrix arcs;//邻接矩阵
    int vexnum;//顶点数
}MGraph;
```

- 求邻接点的函数

```
//求下标为v的顶点的第一个邻接点
int FirstAdjVex(MGraph G, int V){
    int i=0;
    while(i<G.vexnum && G.arcs[V][i]==0)
```

```
        i++; //找第v行第一个非0元
    if(i>=G.vexnum)
        return -1;
    else
        return i;
}

//求下标为v的顶点的w之后的下一个邻接点
int NextAdjVex(MGraph G, int V, int w){
    int i=w+1;
    while(i<G.vexnum && G.arcs[V][i]==0)
        i++;
    if(i>=G.vexnum)
        return -1;
    else
        return i;

}
```

- 创建图：这里test1.txt可以换成其他内容

```
void CreateMGraph(MGraph &G){
    int k;
    int vi,vj;
    FILE *fp;
    if((fp=fopen("test1.txt","r"))==NULL)
        printf("file cannot open\n");
    fscanf(fp, "%d", &G.vexnum);
    for(k=0;k<G.vexnum;k++)
        G.vexs[k]=k;
    for(int i=0;i<MAX;i++)
        memset(G.arcs[i], 0, sizeof(int)*MAX);
    while(fscanf(fp, "%d %d", &vi, &vj)!=EOF){
        G.arcs[vi][vj]=G.arcs[vj][vi]=1;
    }
    fclose(fp);
}
```

- 核心代码：把课本的递归方法分解为了状态机

```
typedef struct{
    MGraph G;
    int v0,min,p;
}node;

struct record{
    node a;
    int state;
```

```
    record(node a,int state):a(a),state(state){}
};

void non_recursive_DFSArticul(MGraph G,int v0){
    stack<record> s;
    node* cur=(node*)malloc(sizeof(node));//初始化
    cur->G=G;
    cur->v0=v0;
    int state=0;
    while(1){
        if(state == 2){
            if(s.empty())
                break;
            *cur=s.top().a;
            state=s.top().state; //返回上层状态
            s.pop();
        }
        else if(state == 0){
            visited[cur->v0]=cur->min=++Count;
            cur->p=FirstAdjVex(cur->G, cur->v0);
            if(cur->p!=-1){
                if(visited[cur->p]==0){
                    s.push(record(*cur,1));//保存本层状态
                    cur->v0=cur->p;//更新到下层状态
                    state=0;
                }
                else if(visited[cur->p]<cur->min){
                    cur->min=visited[cur->p];
                    state=3;
                }
                else{
                    state=3;
                }
            }
            else{
                low[cur->v0]=cur->min;
                state=2;
            }
        }
        else if(state == 1){
            if(low[cur->p]<cur->min)
                cur->min=low[cur->p];
            if(low[cur->p]>=visited[cur->v0])
                flag[cur->v0]=TRUE;
            state=3;
        }
        else if(state == 3){
            cur->p=NextAdjVex(cur->G, cur->v0, cur->p);
            if(cur->p!=-1){
```

```
                if(visited[cur->p]==0){
                    s.push(record(*cur,1));//保存本层状态
                    cur->v0=cur->p;         //更新到下层状态
                    state=0;
                }
                else if(visited[cur->p]<cur->min){
                    cur->min=visited[cur->p];
                    state=3;
                }
                else{
                    state=3;
                }
            }
            else{
                low[cur->v0]=cur->min;
                state=2;
            }
        }
    }
}

void FindArticul(MGraph G){
    //查找输出全部关节点
    Count=1;
    visited[0]=1;//0号顶点为生成树的根
    for(int i=1;i<G.vexnum;i++)
        visited[i]=0;
    int v=FirstAdjVex(G,0);
    non_recursive_DFSArticul(G, v);
    if(Count<G.vexnum){
        flag[0]=TRUE;
        while(NextAdjVex(G, 0, v)!=-1){
            v=NextAdjVex(G, 0, v);
            if(visited[v]==0)
                non_recursive_DFSArticul(G, v);
        }
    }
}
```

参考了课本递归算法

```
void DFSArticul(MGraph G, int v0){
    //从第v0个顶点出发深度优先遍历G，查找并输出关节点
    int min;
    visited[v0]=min=++Count;
    for(int p=FirstAdjVex(G, v0); p!=-1; p=NextAdjVex(G, v0, p)){
        if(visited[p]==0){
            DFSArticul(G, p);
            if(low[p]<min)
```

```
                min=low[p];
            if(low[p]>=visited[v0])
                flag[v0]=TRUE;
        }
        else if(visited[p]<min)
            min=visited[p];
    }
    low[v0]=min;
}
```

和一篇关于递归转非递归的博客：

```
struct record{
  node* a;
  int state;
  record(node* a,int state):a(a),state(state){}
};
void non_recursive_inorder(){
  stack<record> s;
  node* cur=root;    //初始化状态
  int state=0;
  while(1){
    if(!cur){                 //如果遇到null结点，返回上一层
          if(cur == root)break;//如果没有上一层，退出循环
      cur=s.top().a;
      state=s.top().state; //返回上层状态
      s.pop();
    }else if(state == 0){     //状态位0，执行第一个递归inorder(cur->left);
      s.push(record(cur,1));//保存本层状态
      cur=cur->left;         //更新到下层状态
      state=0;
    }else if(state == 1){       //状态为1，执行print和inorder(cur->right)
      printf("%d ",cur->x);
      s.push(record(cur,2));    //保存本层状态
      cur=cur->right;          //进入下层状态
      state=0;
    }else if(state == 2){       //状态2，函数结束，返回上层状态
      if(cur == root)break;   //初始结点的退出状态，遍历结束
      cur=s.top().a;          //返回上层状态
      state=s.top().state;
      s.pop();
    }
  }
  putchar(10);
}
```

- 考虑非递归，用到栈，直接用c++中的库函数

```
#include <stack>
using namespace std;
```

BFS的应用

- 邻接表的定义

```
typedef struct ArcNode{//边结点
    int adjvex; //邻接点的下标
    struct ArcNode *nextarc; //后继链指针
}ArcNode;
typedef struct VNode{//顶点结点
    ArcNode *firstarc;//边链头指针
}AdjList[MAX];

typedef struct{
    AdjList vertices;//邻接表
    int vexnum;//顶点数和边数
    unsigned kind;//图种类标志
}ALGraph;
```

- 创建图

```
void CreateALGraph(ALGraph &G){
    int k,vi,vj;
    FILE *fp;
    if((fp=fopen("test3.txt","r"))==NULL)
        printf("file cannot open\n");
    fscanf(fp, "%d", &G.vexnum);
    for(k=0;k<G.vexnum;k++)
        G.vertices[k].firstarc=NULL;
    ArcNode* p;
    while(fscanf(fp, "%d %d", &vi, &vj)!=EOF){
        p=(ArcNode*)malloc(sizeof(ArcNode));
        p->adjvex=vj;
        p->nextarc=G.vertices[vi].firstarc;//头插法
        G.vertices[vi].firstarc=p;
        //无向图
        p=(ArcNode*)malloc(sizeof(ArcNode));
        p->adjvex=vi;
        p->nextarc=G.vertices[vj].firstarc;
        G.vertices[vj].firstarc=p;
    }
    fclose(fp);
}
```

- 判断两点之间的连通性

```
int LocateVex(ALGraph G, int v0, int v){
    //v0和v之间连不连通
    ArcNode* p;
    for(p=G.vertices[v0].firstarc;p!=NULL;p=p->nextarc){
        if(p->adjvex==v)
            return 1;
    }
    return MAX;
}
```

- 核心代码：Dijkstra算法

```
void DijkstraALGraph(ALGraph G, int v0, int P[][MAX], int D[]){
    //P存路径
    int v, i, j, w, min, vj, k, l;
    int S[MAX];
    for(v=0;v<G.vexnum;v++){
        S[v]=0;
        D[v]=LocateVex(G, v0, v);
        //printf("D[%d]=%d\n",v,D[v]);
        for(j=0;j<G.vexnum;j++)
            P[v][j]=-1;//初始化存数为-1
        if(D[v]!=MAX){
            P[v][0]=v0;
            P[v][1]=v;
        }
    }
    D[v0]=0;
    //printf("D[%d]=%d\n",v0,D[v0]);
    S[v0]=1;
    for(i=0;i<G.vexnum;i++){
        min=MAX;
        for(w=0;w<G.vexnum;w++)
            if(S[w]==0&&D[w]<min){
                min=D[w];
                vj=w;
            }
        S[vj]=1;
        for(w=0;w<G.vexnum;w++){
            l=LocateVex(G, vj, w);
            if(S[w]==0 && min+l<D[w]){
                D[w]=min+l;
                //printf("D[%d]=%d\n",w,D[w]);
                for(k=0; P[vj][k]!=-1;k++)
                    P[w][k]=P[vj][k];
                P[w][k]=w;
                P[w][k+1]=-1;
            }
```

```
            }
        }
        //输出
        for(k=1;k<G.vexnum;k++){
            printf("%d ",D[k]);
            for(int j=0;P[k][j]!=-1;j++)
                if(P[k][j+1]!=-1)
                    printf("%d->",P[k][j]);
                else
                    printf("%d",P[k][j]);
            printf("\n");
        }
    }
```

图形化

- 输出文件、之后在终端命令行编译：dot pic.dot -T png -o pic.png

```
void CreatePic(){
    FILE *fp,*fq;
    if((fp=fopen("test1.txt","r"))==NULL)
        printf("file cannot open\n");
    int vexnum, vi, vj;
    fscanf(fp, "%d", &vexnum);
    if((fq=fopen("pic.dot","w"))==NULL)
        printf("file cannot open\n");
    char* str="digraph pic{\n";
    fwrite(str, sizeof(char), strlen(str), fq);
    while(fscanf(fp, "%d %d", &vi, &vj)!=EOF){
        putc(vi+'0',fq);
        fputs(" -> ",fq);
        putc(vj+'0',fq);
        fputs(" [arrowhead=\"none\"]\n",fq);
    }
    fputs("}\n",fq);
    fclose(fp);
    fclose(fq);
}
```

三.调试分析&代码测试

- 比较顺利，代码没有遇到太多调试的问题，所以合并了两个模块。
- 最初没能理解状态机的转化，给对所有状态都有存和取，后来理解到递归其实是存储当前状态，进入下一状态，结束后再跳转取出栈中保存的上一状态，并顺序继续执行。

输入输出情况：

Input: test3.txt文件

10 0 4 0 9 1 2 1 3 1 4 2 4 2 6 3 5 5 6 5 9 6 7 7 8

Output:

6 7 2 0->4->1 2 0->4->2 3 0->4->1->3 1 0->4 2 0->9->5 3 0->4->2->6 4 0->4->2->6->7 5 0->4->2->6->7->8 1 0->9

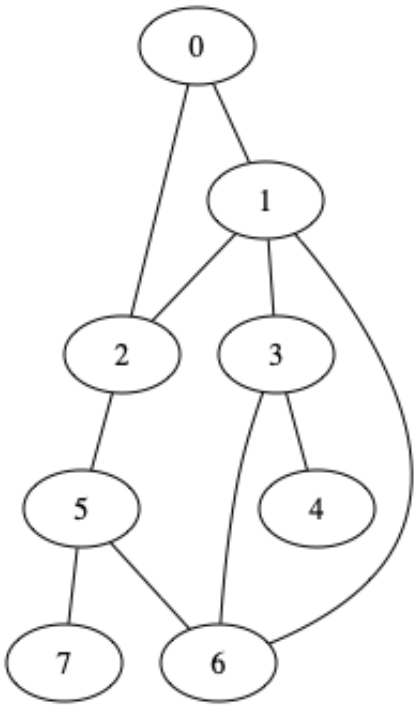
```
/Users/zengjing/CLionProjects/untitled21/cmake-build-debug/untitled21
6 7
2 0->4->1
2 0->4->2
3 0->4->1->3
1 0->4
2 0->9->5
3 0->4->2->6
4 0->4->2->6->7
5 0->4->2->6->7->8
1 0->9
```

图形化

```
digraph pic{
0 -> 4 [arrowhead="none"]
0 -> 9 [arrowhead="none"]
1 -> 2 [arrowhead="none"]
1 -> 3 [arrowhead="none"]
1 -> 4 [arrowhead="none"]
2 -> 4 [arrowhead="none"]
2 -> 6 [arrowhead="none"]
3 -> 5 [arrowhead="none"]
5 -> 6 [arrowhead="none"]
5 -> 9 [arrowhead="none"]
6 -> 7 [arrowhead="none"]
7 -> 8 [arrowhead="none"]
}

```

命令行编译：dot pic.dot -T png -o pic.png



Input: test1.txt文件

8 0 1 0 2 1 2 1 3 1 6 2 5 3 4 3 6 5 6 5 7

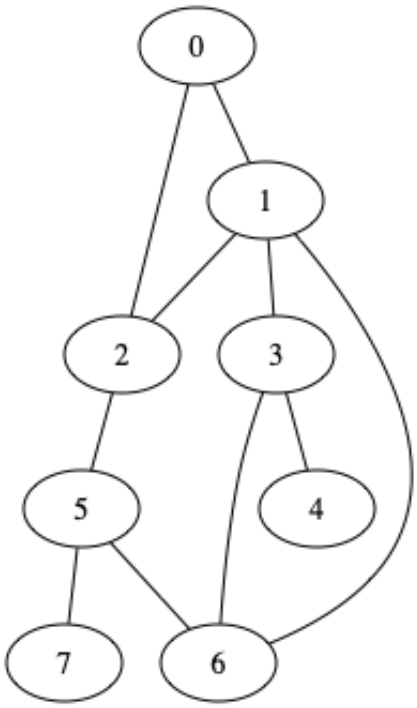
Output:

3 5 1 0->1 1 0->2 2 0->1->3 3 0->1->3->4 2 0->2->5 2 0->1->6 3 0->2->5->7

```
/Users/zengjing/CLionProjects/untitled21/cmake-build-debug/untitled21
3 5
1 0->1
1 0->2
2 0->1->3
3 0->1->3->4
2 0->2->5
2 0->1->6
3 0->2->5->7
```

图形化：pic.dot文件

```
digraph pic{
0 -> 1 [arrowhead="none"]
0 -> 2 [arrowhead="none"]
1 -> 2 [arrowhead="none"]
1 -> 3 [arrowhead="none"]
1 -> 6 [arrowhead="none"]
2 -> 5 [arrowhead="none"]
3 -> 4 [arrowhead="none"]
3 -> 6 [arrowhead="none"]
5 -> 6 [arrowhead="none"]
5 -> 7 [arrowhead="none"]
}
```



四.实验总结

第一次尝试较大的图实现，使用CLion书写和执行。

囊括了邻接矩阵、邻接表两种储存结构，涉及到了递归转非递归求关节点、求最短路径、图形化等等，极大地锻炼了能力。

从最初的手忙脚乱到后来功能清晰、细节处理到位，收获很大，体会到了思考全面以及迅速找到出错的地方的方法。

实验进行较为顺利，按时完成实验。

六.附录

main.c //主程序