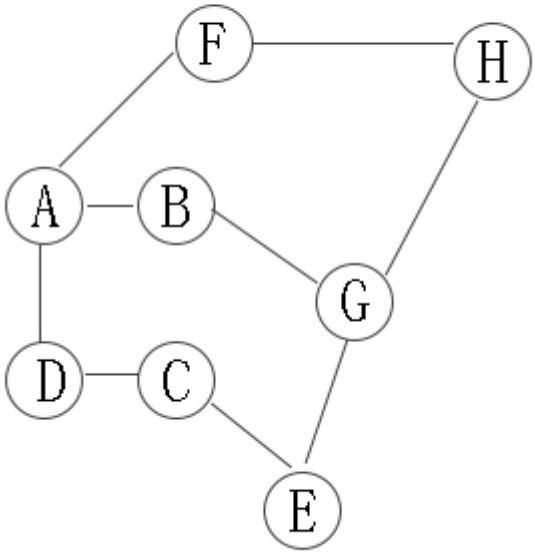
图的深度优先遍历(DFS)和广度优先遍历(BFS)

1. 深度优先遍历　　深度优先遍历(Depth First Search)的主要思想是：　　1、首先以一个未被访问过的顶点作为起始顶点，沿当前顶点的边走到未访问过的顶点；

2、当没有未访问过的顶点时，则回到上一个顶点，继续试探别的顶点，直至所有的顶点都被访问过。

在此我想用一句话来形容 “不到南墙不回头”。

1.1　无向图的深度优先遍历图解 以下"无向图"为例：



对上无向图进行深度优先遍历，从A开始：

第1步：访问A。

第2步：访问B(A的邻接点)。 在第1步访问A之后，接下来应该访问的是A的邻接点，即"B,D,F"中的一个。但在本文的实现中，顶点ABCDEFGH是按照顺序存储，B在"D和F"的前面，因此，先访问B。

第3步：访问G(B的邻接点)。 和B相连只有"G"(A已经访问过了)

第4步：访问E(G的邻接点)。 在第3步访问了B的邻接点G之后，接下来应该访问G的邻接点，即"E和H"中一个(B已经被访问过，就不算在内)。而由于E在H之前，先访问E。

第5步：访问C(E的邻接点)。 和E相连只有"C"(G已经访问过了)。

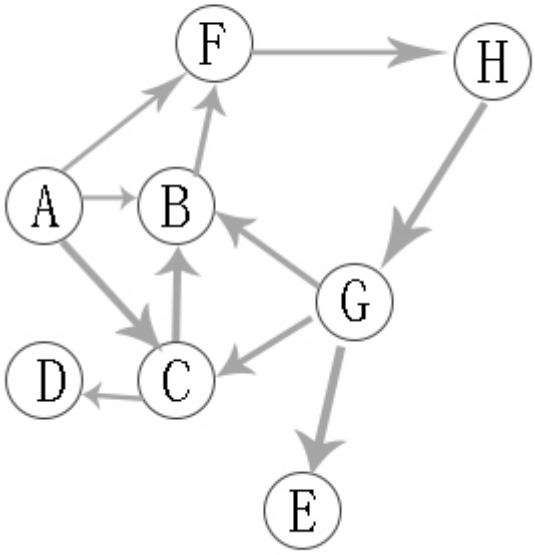
第6步：访问D(C的邻接点)。

第7步：访问H。因为D没有未被访问的邻接点；因此，一直回溯到访问G的另一个邻接点H。

第8步：访问(H的邻接点)F。

因此访问顺序是：A -> B -> G -> E -> C -> D -> H -> F

1.2　有向图的深度优先遍历 有向图的深度优先遍历图解：



对上有向图进行深度优先遍历，从A开始：

第1步：访问A。

第2步：访问(A的出度对应的字母)B。 在第1步访问A之后，接下来应该访问的是A的出度对应字母，即"B,C,F"中的一个。但在本文的实现中，顶点ABCDEFGH是按照顺序存储，B在"C和F"的前面，因此，先访问B。

第3步：访问(B的出度对应的字母)F。 B的出度对应字母只有F。

第4步：访问H(F的出度对应的字母)。 F的出度对应字母只有H。

第5步：访问(H的出度对应的字母)G。

第6步：访问(G的出度对应字母)E。 在第5步访问G之后，接下来应该访问的是G的出度对应字母，即"B,C,E"中的一个。但在本文的实现中，顶点B已经访问了，由于C在E前面，所以先访问C。

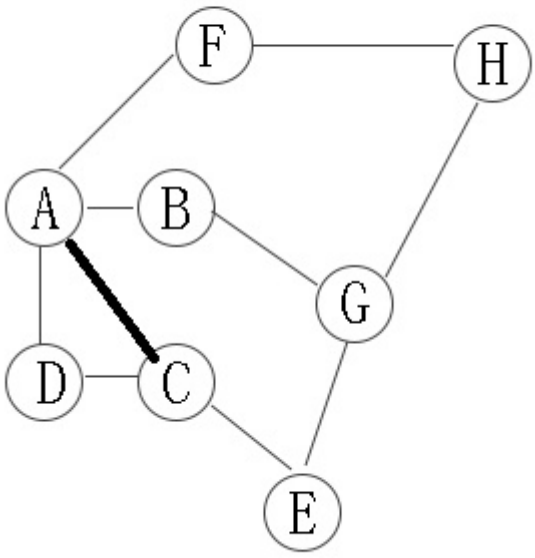
第7步：访问(C的出度对应的字母)D。

第8步：访问(C的出度对应字母)D。 在第7步访问C之后，接下来应该访问的是C的出度对应字母，即"B,D"中的一个。但在本文的实现中，顶点B已经访问了，所以访问D。

第9步：访问E。D无出度，所以一直回溯到G对应的另一个出度E。

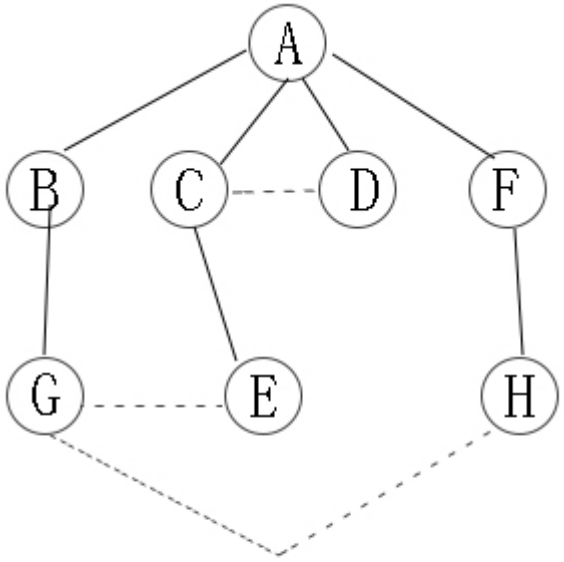
因此访问顺序是：A -> B -> F -> H -> G -> C -> D -> E

2.广度优先遍历　广度优先遍历(Depth First Search)的主要思想是：类似于树的层序遍历。
2.1　无向图的广度优先遍历图解：



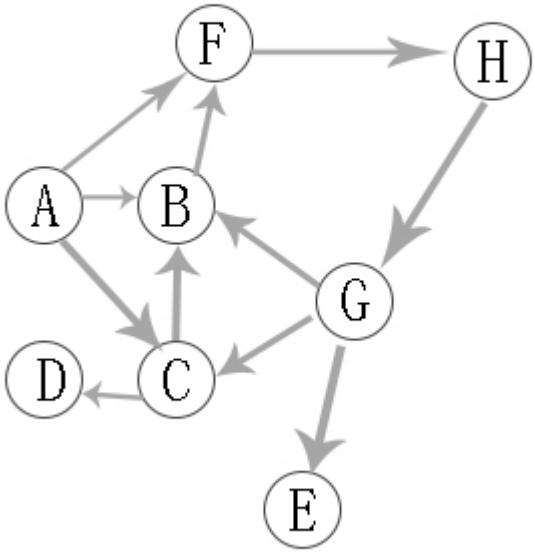
从A开始，有4个邻接点，“B，C，D，F”，这是第二层；

在分别从B，C，D，F开始找他们的邻接点，为第三层。以此类推。

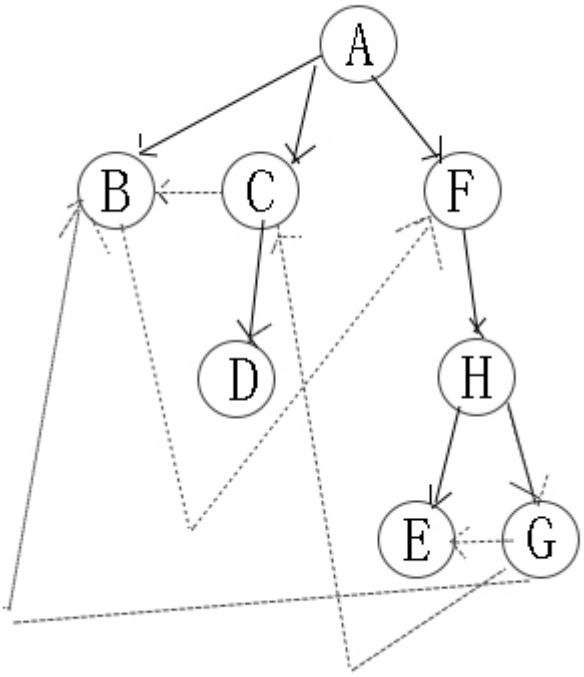


因此访问顺序是：A -> B -> C -> D -> F -> G -> E -> H

2.2　有向图的广度优先遍历图解：



与无向图类似 。可以参考。



因此访问顺序是：A -> B -> C -> F -> D -> H -> E -> G

广度优先遍历

```c
/*************************************************************************
    > File Name: dfs.c
    > Author: Sakura7301
    > Email: sakuraduck@foxmail.com
    > Github: https://github.com/Sakura7301
    > Created Time: 2021年07月30日 星期五 14时31分31秒
 ************************************************************************/

#include<stdio.h>

#define MAX_GRAPH 100
#define MAX_QUEUE 30

typedef struct
{
    char vex[MAX_GRAPH];//顶点
    int edge[MAX_GRAPH][MAX_GRAPH];//邻接矩阵
    int n;//顶点数
    int e;//边数
}GRAPH;


int visited[MAX_GRAPH];

void create(GRAPH *G)//创建图
{
    printf("请输入顶点数：\n");
    scanf("%d",&G->n);
    printf("请输入边数：\n");
    scanf("%d",&G->e);
```

```
    getchar();
    int i,j,k,w;
    printf("请输入端点（char类型）：\n");
    for(i=0;i<G->n;i++)
    {
        scanf("%c",&G->vex[i]);
    }
    for(i=0;i<G->n;i++)
    {
        for(j=0;j<G->n;j++)
        {
            G->edge[i][j]=0;//初始化邻接矩阵
        }
    }
    printf("请输入边：：\n");
    for(k=0;k<G->e;k++)
    {
        scanf("%d%d%d",&i,&j,&w);//k j 是坐标，w是权值
        G->edge[i][j]=w;
        G->edge[j][i]=w;//无向图的邻接矩阵是对称的
    }
}

void dfs(GRAPH *G,int i)//深度优先遍历（递归）
{
    int j;
    printf("访问顶点：%c\n",G->vex[i]);
    visited[i]=1;//标志已访问的顶点
    for(j=0;j<G->n;j++)
    {
        //判断条件，该坐标未被访问过并且权值不为0
        if(G->edge[i][j]!=0 && visited[j]==0)
        {
            dfs(G,j);//满足条件则递归访问顶点即可
        }
    }
}

void bfs(GRAPH *G,int k)//广度优先遍历（队列）
{
    printf("广度优先遍历结果如下：\n");
    int quene[MAX_QUEUE];//使用数组模拟队列
    int front=-1;//队头
    int rear=-1;//队尾
    int flag=0;
    int i,j;
    for(i=0;i<MAX_GRAPH;i++)
    {
        visited[i]=0;//初始化
    }
    printf("访问顶点%c\n",G->vex[k]);
    visited[k]=1;//标记已访问的顶点
    rear=(rear+1)% MAX_QUEUE;//队尾指针后移
    quene[rear]=k;//入队
    front=rear;
    flag++;
    while(flag>0)
    {
        i=quene[front];//出队
        front=(front+1)% MAX_QUEUE;//队头指针后移
        flag--;
```

```
        for(j=0;j < G->n;j++)
        {
            if(G->edge[i][j]!=0 && visited[j]==0)
            {
                printf("访问顶点%c\n",G->vex[j]);
                visited[j]=1;//标记已访问的顶点
                rear=(rear+1)% MAX_QUEUE;
                quene[rear]=j;//入队
                flag++;
            }
        }
    }
    printf("遍历结束\n");
}
int main()
{
    for(int i=0;i<MAX_GRAPH;i++)
    {
        visited[i]=0;//初始化
    }
    GRAPH G;
    create(&G);
    printf("深度优先遍历结果如下：\n");
    dfs(&G,0);
    printf("遍历结束\n");
    bfs(&G,0);
    return 0;
}
```