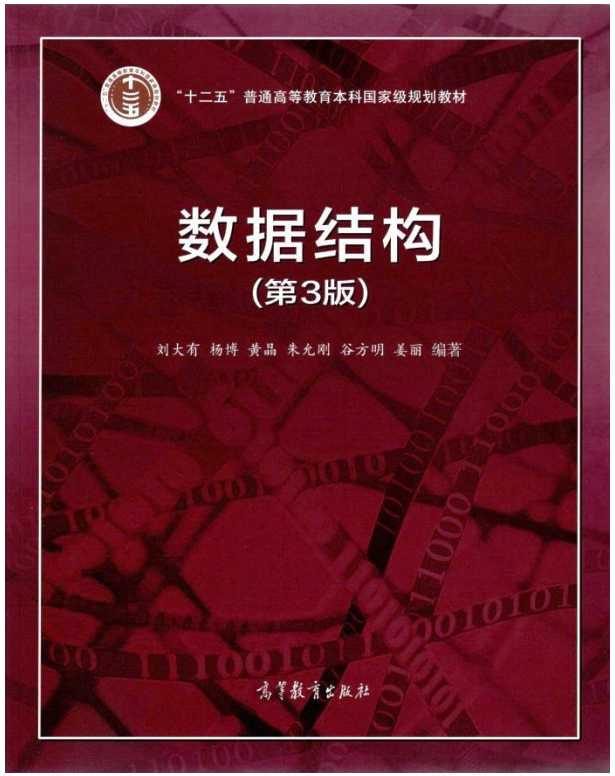



# 图的遍历及应用

- 深度优先搜索
- 广度优先搜索
- 图遍历的应用

数据之法
结构之美
算法之道

zhuyungang@jlu.edu.cn

Last updated on 2023.10

周雨扬

北京大学20级本科生

2019年NOI全国中学生信息学奥赛决赛冠军

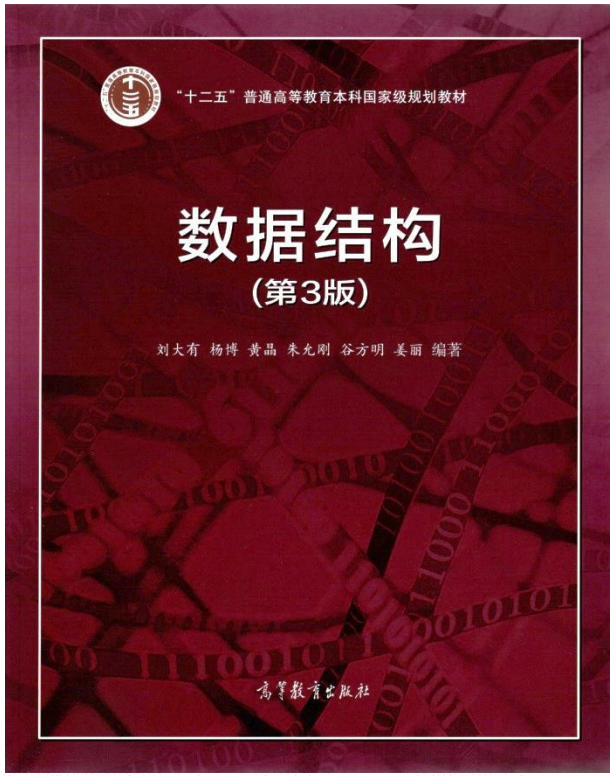
2020年IOI世界中学生信息学奥赛季军

2021年ICPC国际大学生程序设计竞赛亚洲区域赛冠军

2022年ICPC国际大学生程序设计竞赛全球总决赛亚军

有时候程序写完了，没有输出预定的结果，一遍遍地找，还是发现不了错在哪，很崩溃。最痛苦也是最快乐的一次，是找了两个晚上终于把bug找出来……

我一度怀疑当初选择这条路到底值不值？自己究竟行不行？之后我不断调整心态，继续做更多的题，学习更多的算法，终于圆梦赛场。




# 图的遍历及应用

- **深度优先搜索**
- 广度优先搜索
- 图遍历的应用

数据之法
结构之美
算法之道

zhuyungang@jlu.edu.cn

Last updated on 2023.10

# 图的遍历

A

- 从图的某顶点出发，访问图中所有顶点，且使每个顶点恰被访问一次的过程被称为图遍历。
- 图中可能存在回路，且图的任一顶点都可能与其它顶点连通，在访问完某个顶点之后可能会沿着某些边又回到了曾经访问过的顶点。
- 为了避免重复访问，可用一个辅助数组*visited*[ ]记录顶点是否被访问过，数组各元素初始值为0。在遍历过程中，一旦某一个顶点 *i* 被访问，就置 *visited*[*i*] 为 1。
- 图的遍历可以通过深度优先搜索算法和广度优先搜索算法来实现，对应的遍历方式称为深度优先遍历和广度优先遍历。

# 深度优先搜索 DFS ( Depth First Search, DFS)

A

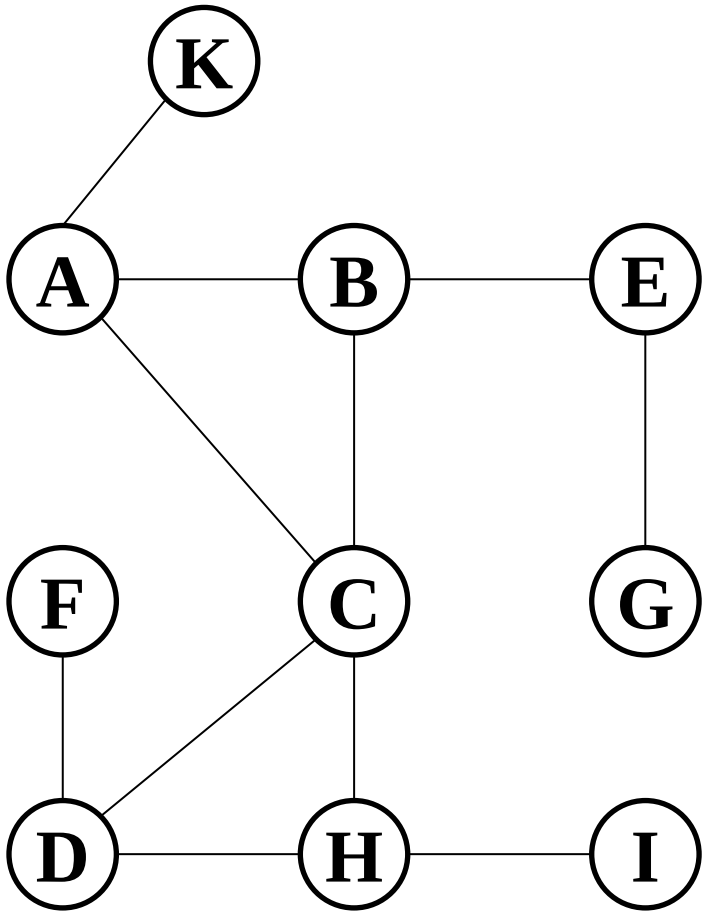
基本思想:

- *DFS* 在访问图中某一起始顶点 $v_0$ 后，由 $v_0$ 出发，访问它的任一邻接顶点 $v_1$；再从 $v_1$ 出发，访问$v_1$的一个未曾访问过的邻接顶点 $v_2$；然后再从 $v_2$ 出发，进行类似的访问，… 如此进行下去，直至到达一个顶点，它不再有未访问的邻接顶点。
- 接着，回退一步，退到前一次刚访问过的顶点，看是否还有其它没有被访问的邻接顶点。如果有，则访问此顶点，之后再从此点出发，进行与前述类似的访问；如果没有，就再退回一步进行搜索。
- 重复上述过程，直到图中所有顶点都被访问过为止。

图深度优先遍历的次序不唯一

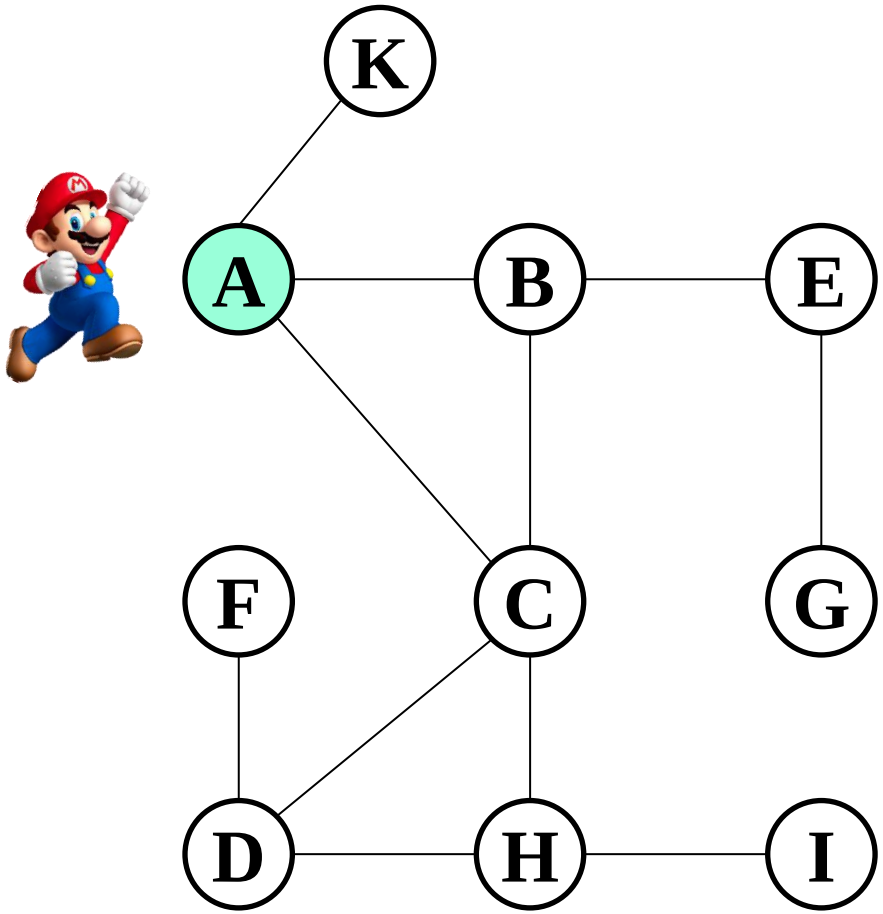
# 深度优先搜索 DFS ( Depth First Search, DFS)

A

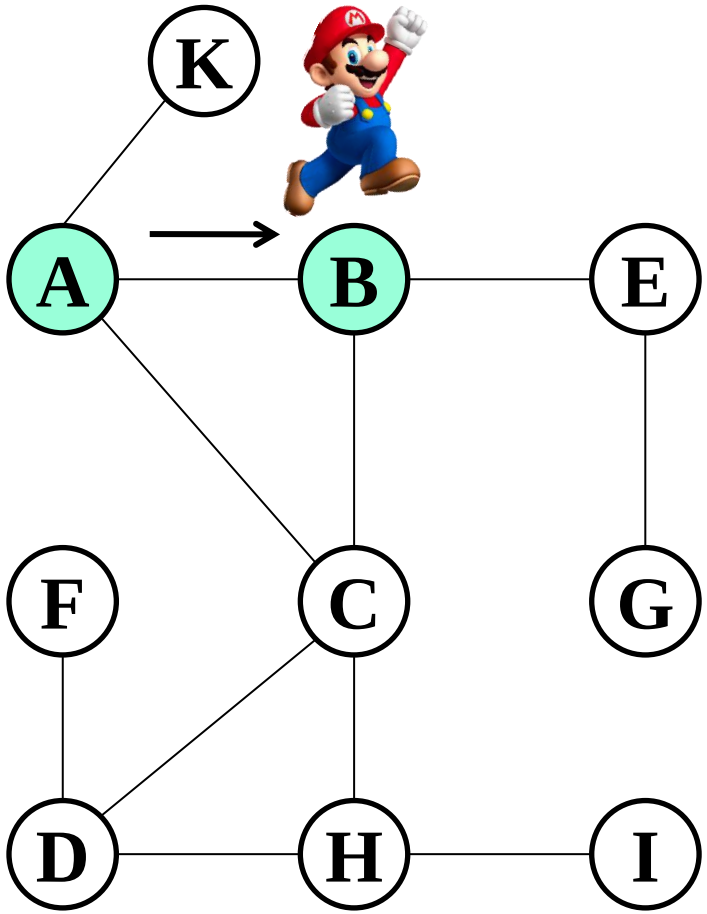


# 深度优先搜索 DFS ( Depth First Search, DFS)

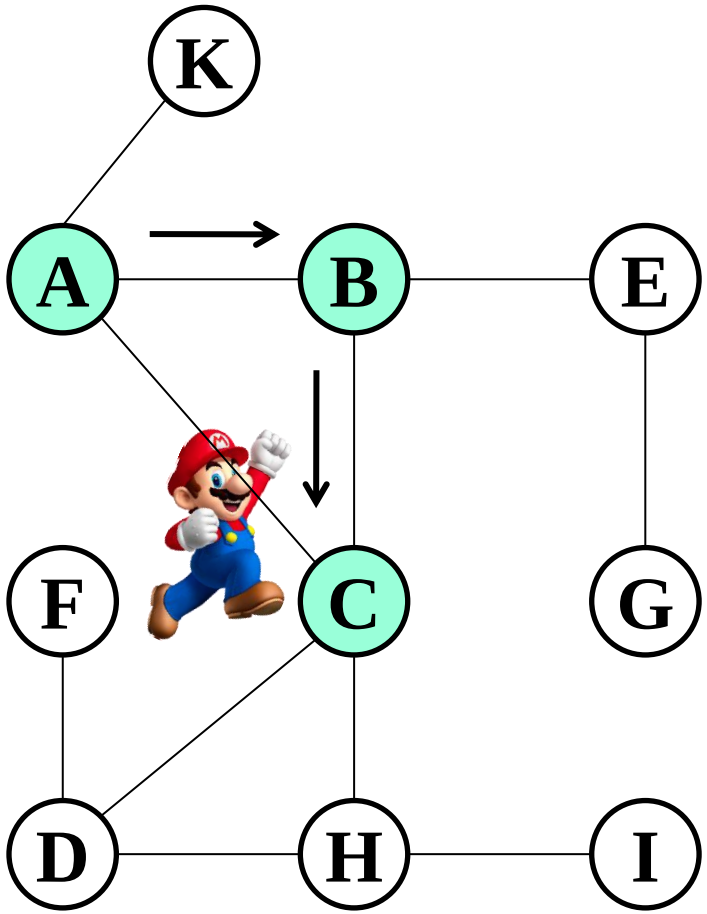
A



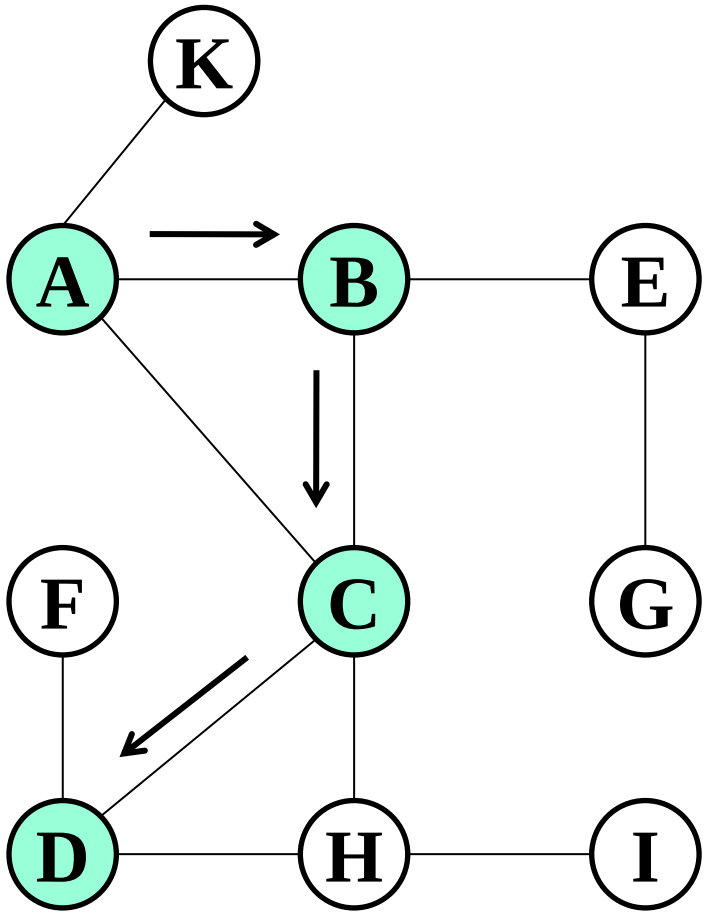
# 深度优先搜索 DFS ( Depth First Search, DFS)



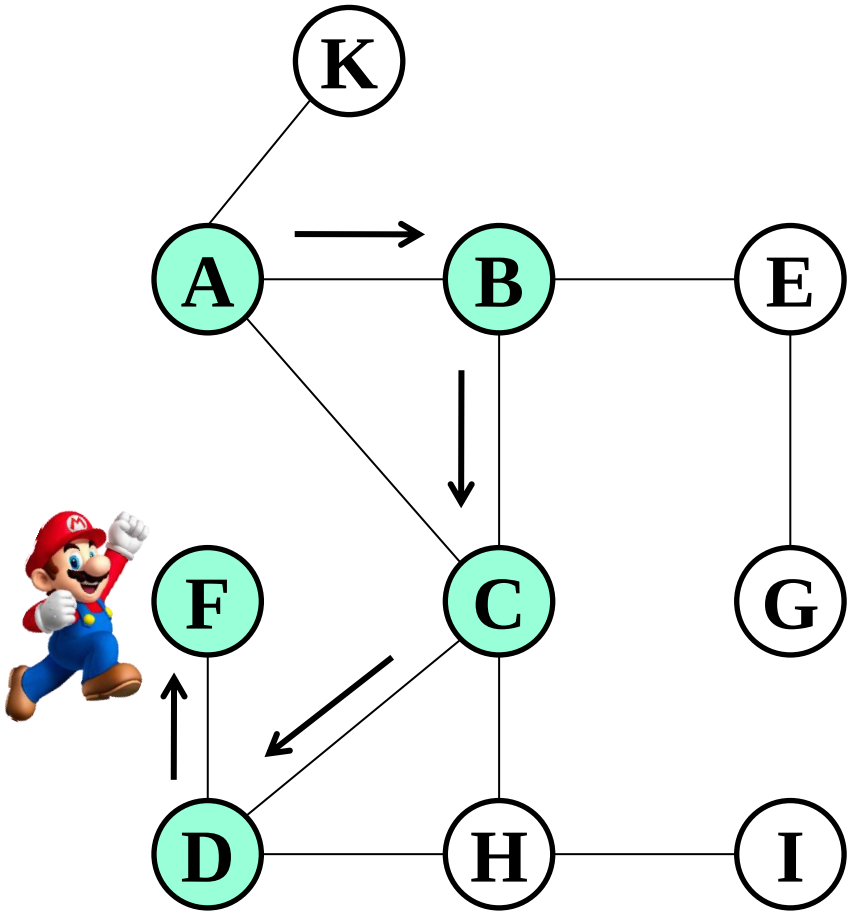
# 深度优先搜索 DFS ( Depth First Search, DFS)



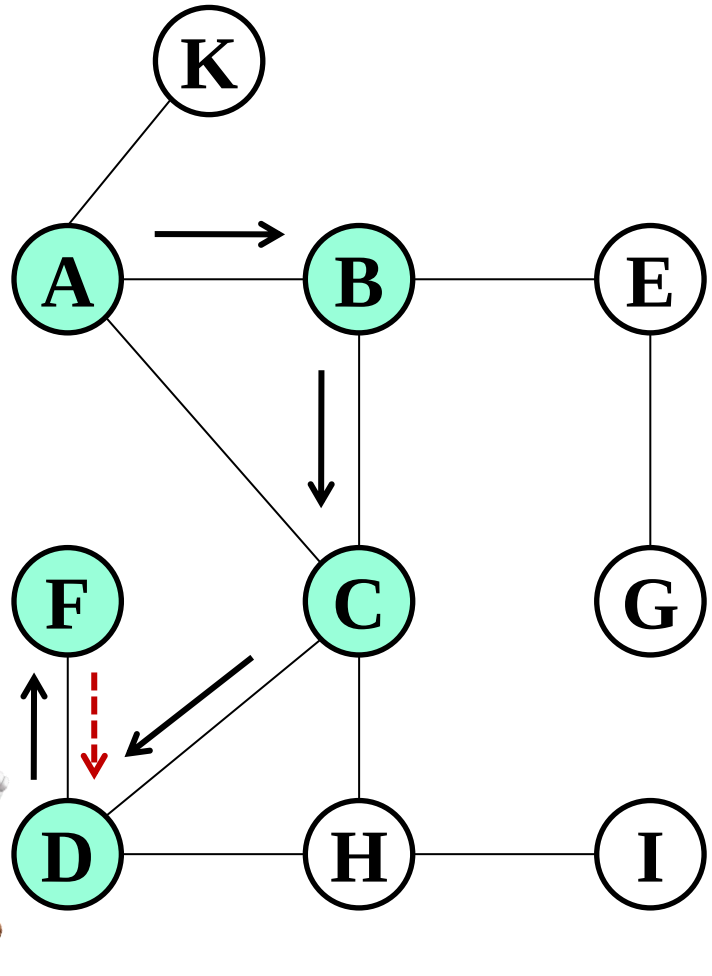
# 深度优先搜索 DFS ( Depth First Search, DFS)



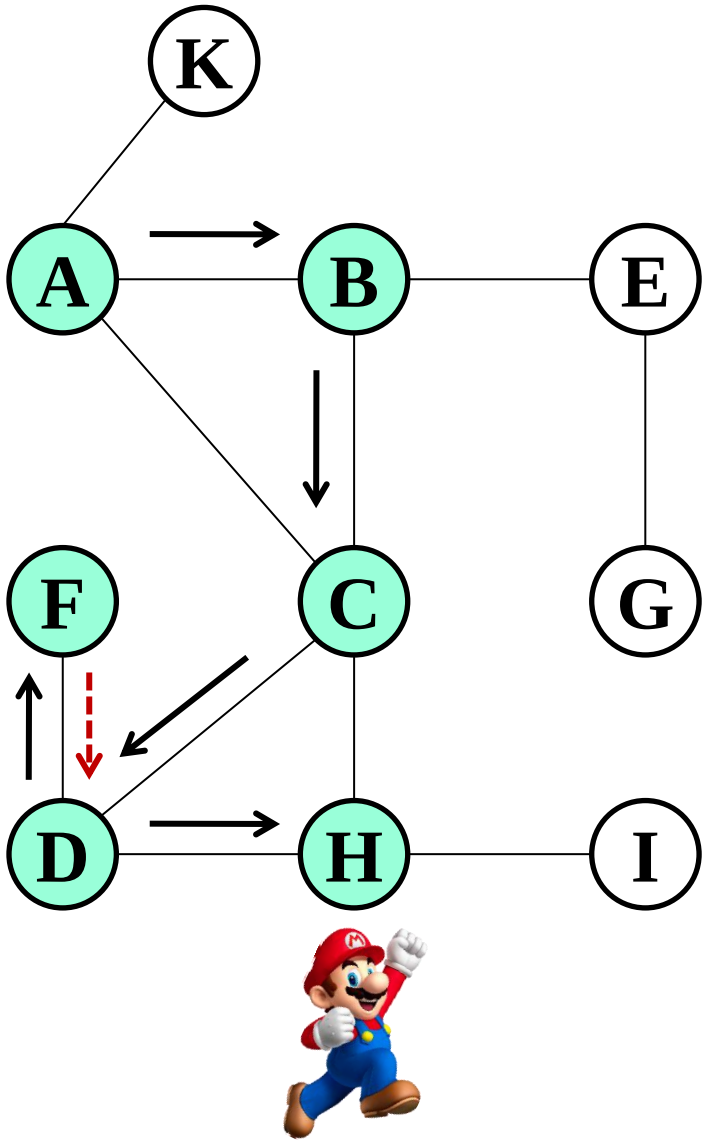

# 深度优先搜索 DFS ( Depth First Search, DFS)



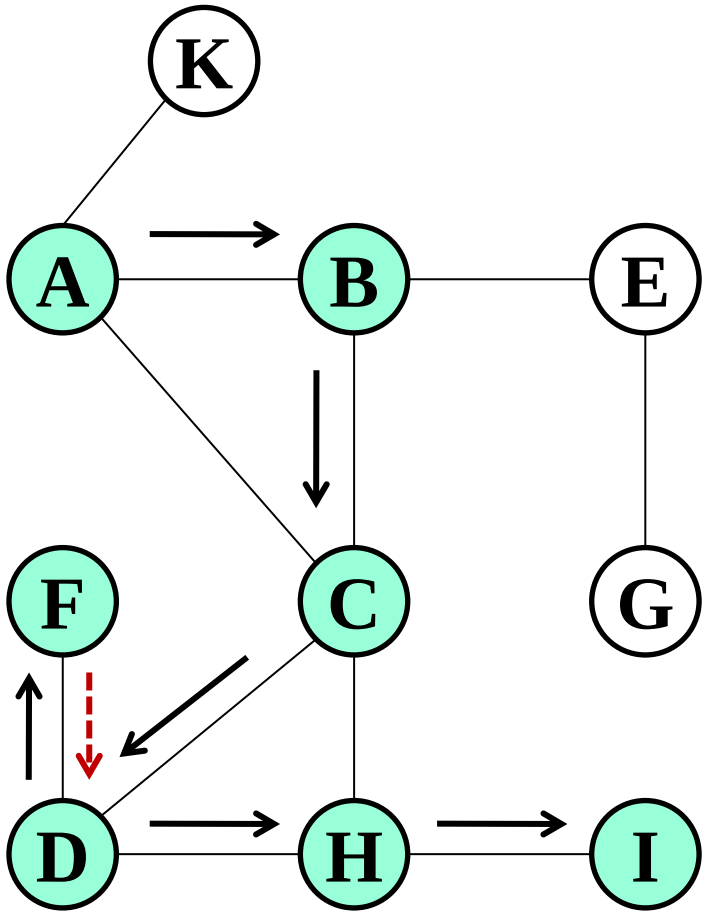
# 深度优先搜索 DFS ( Depth First Search, DFS)



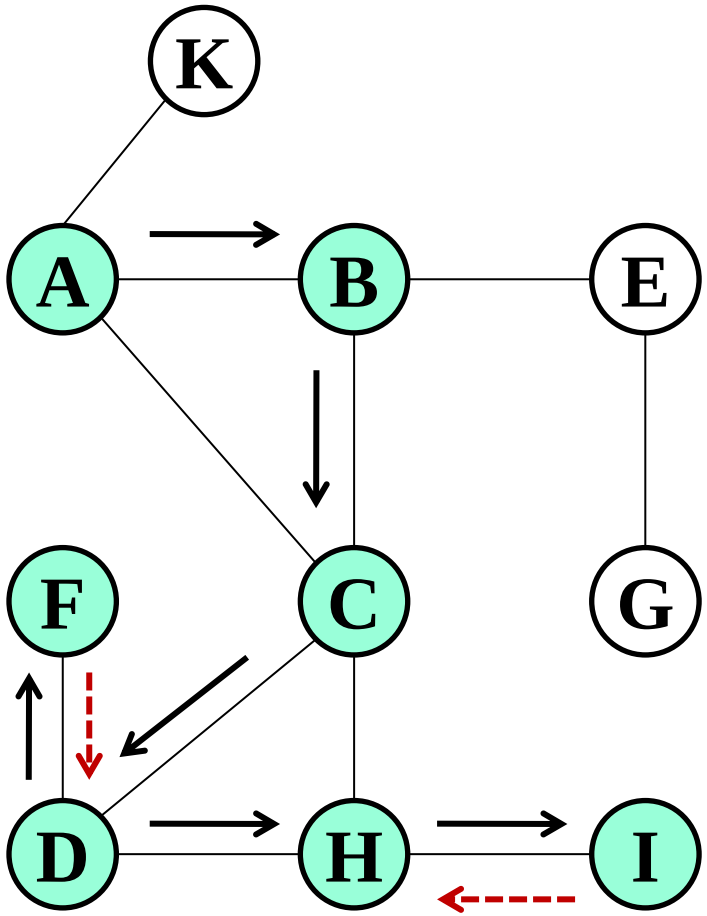
# 深度优先搜索 DFS ( Depth First Search, DFS)



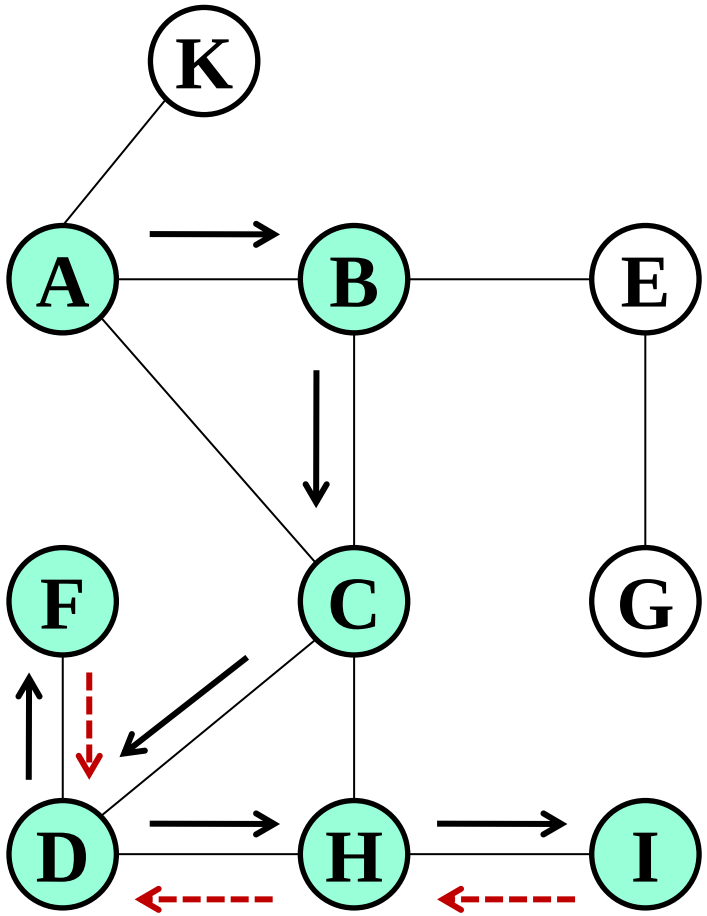
# 深度优先搜索 DFS ( Depth First Search, DFS)



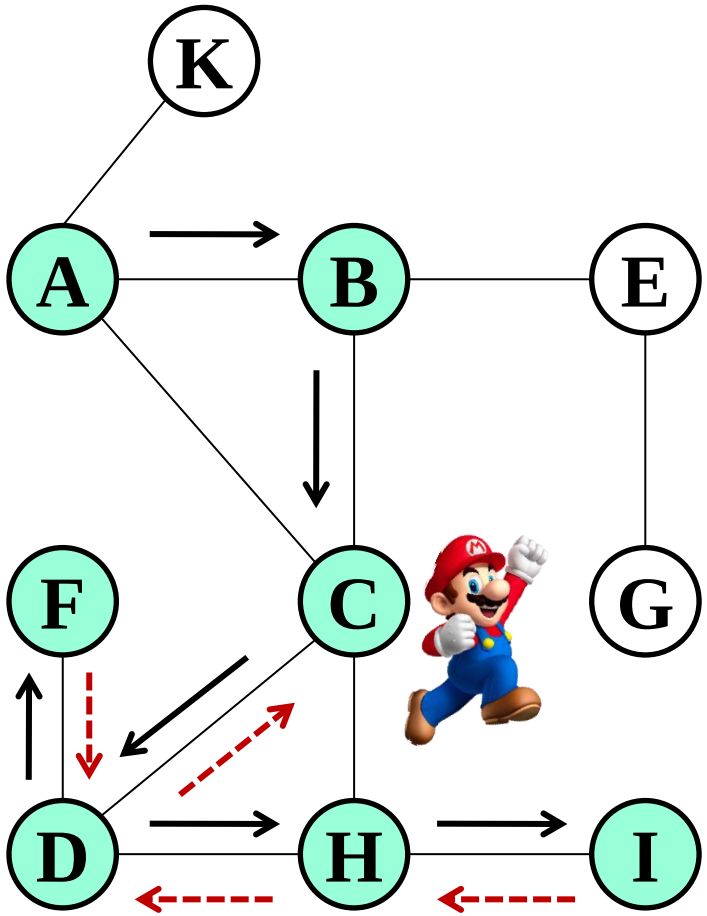

# 深度优先搜索 DFS ( Depth First Search, DFS)



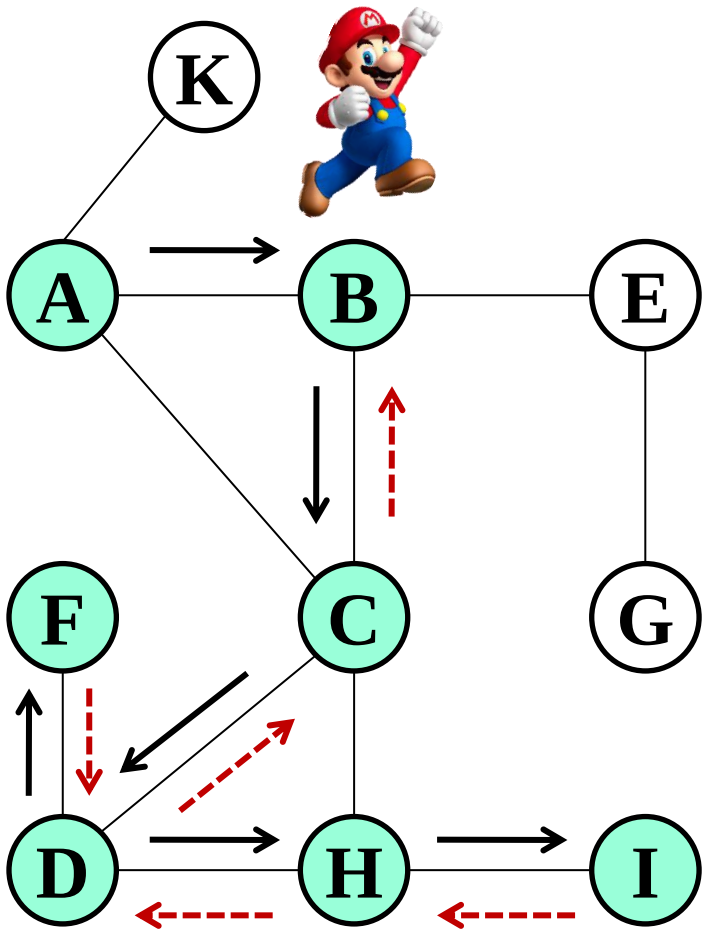

# 深度优先搜索 DFS ( Depth First Search, DFS)



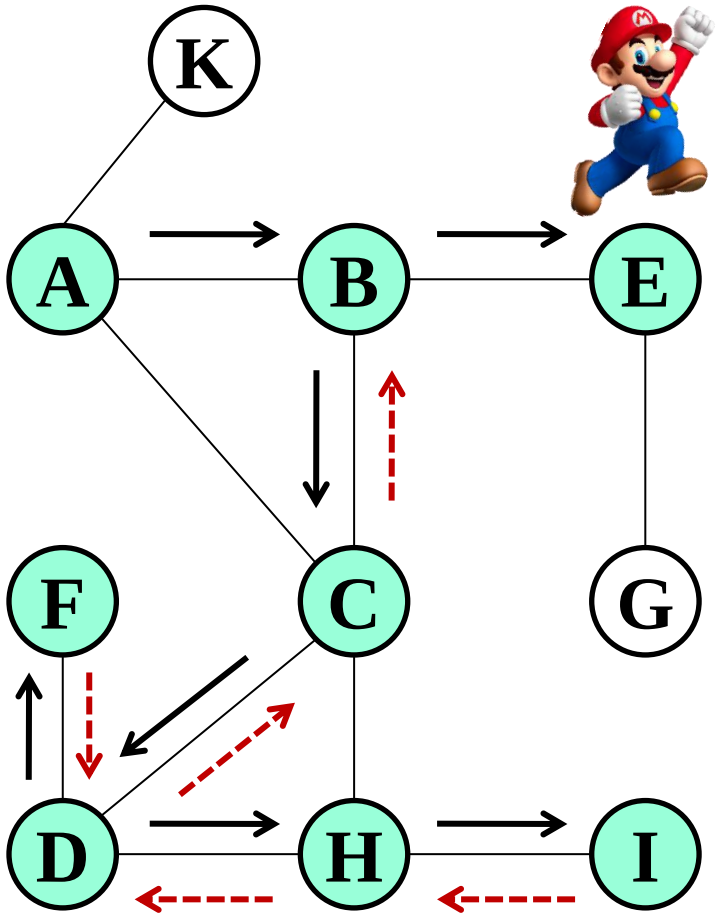

# 深度优先搜索 DFS ( Depth First Search, DFS)



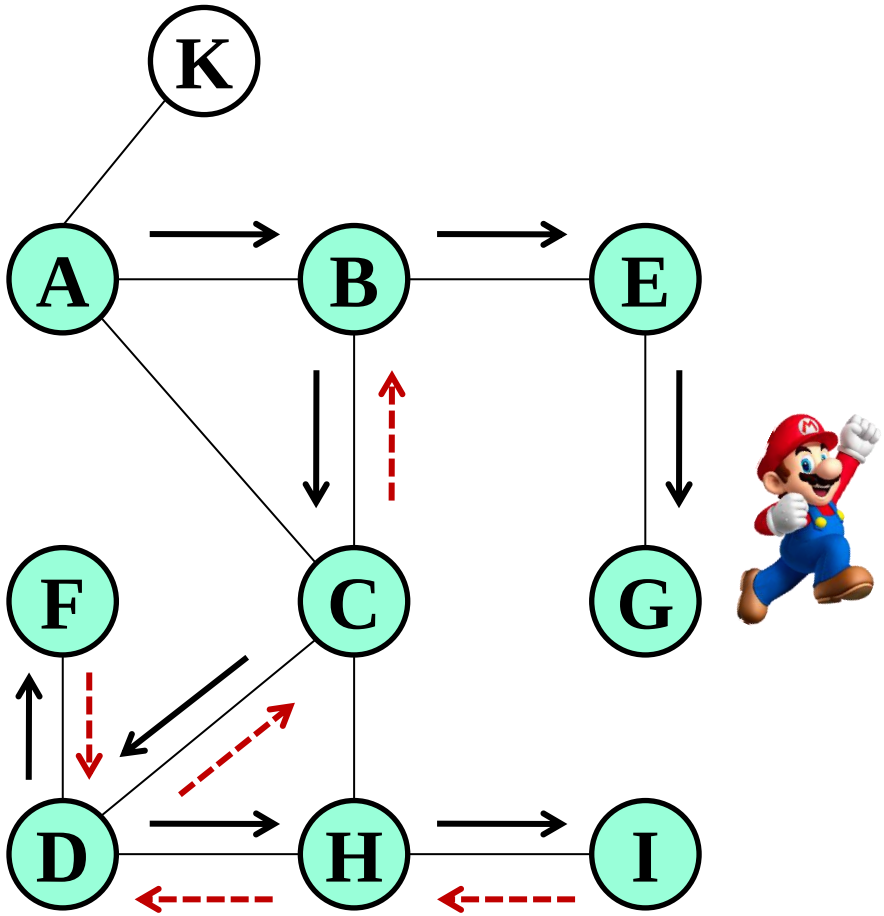
# 深度优先搜索 DFS ( Depth First Search, DFS)



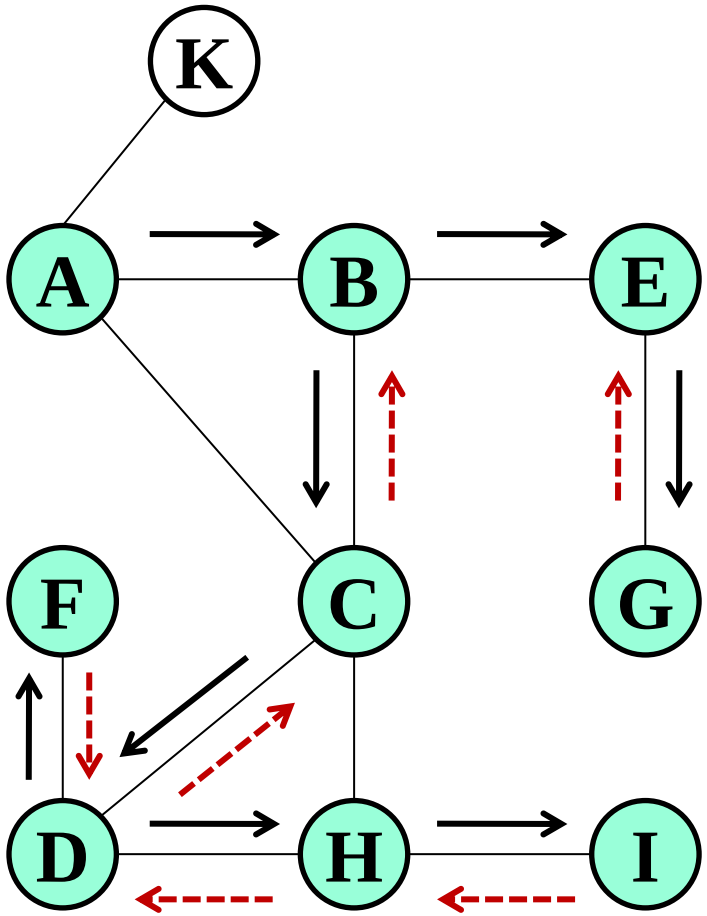
# 深度优先搜索 DFS ( Depth First Search, DFS)



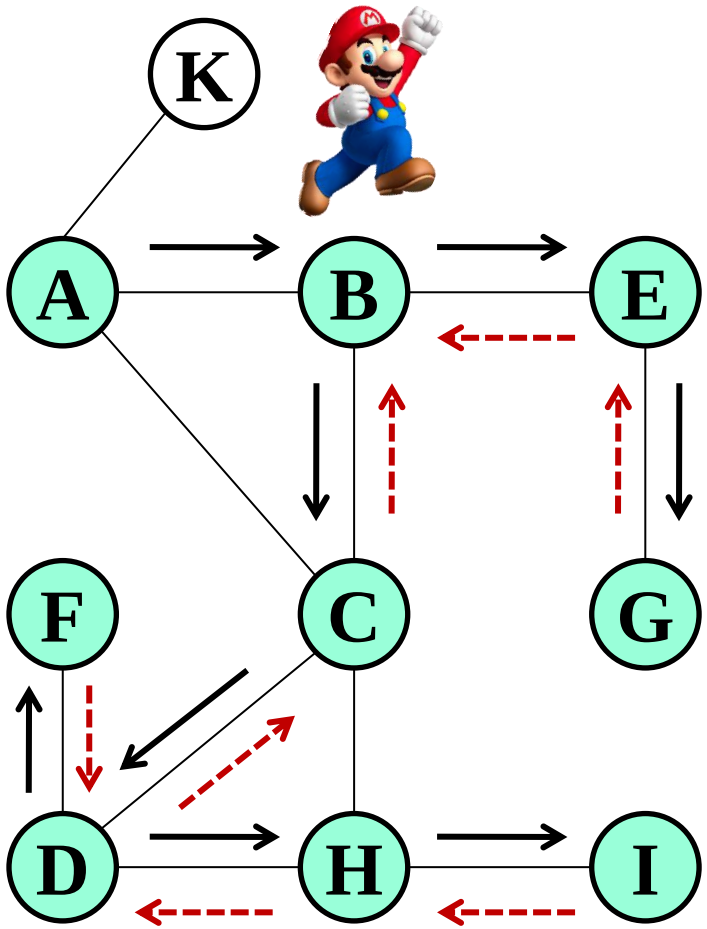
# 深度优先搜索 DFS ( Depth First Search, DFS)



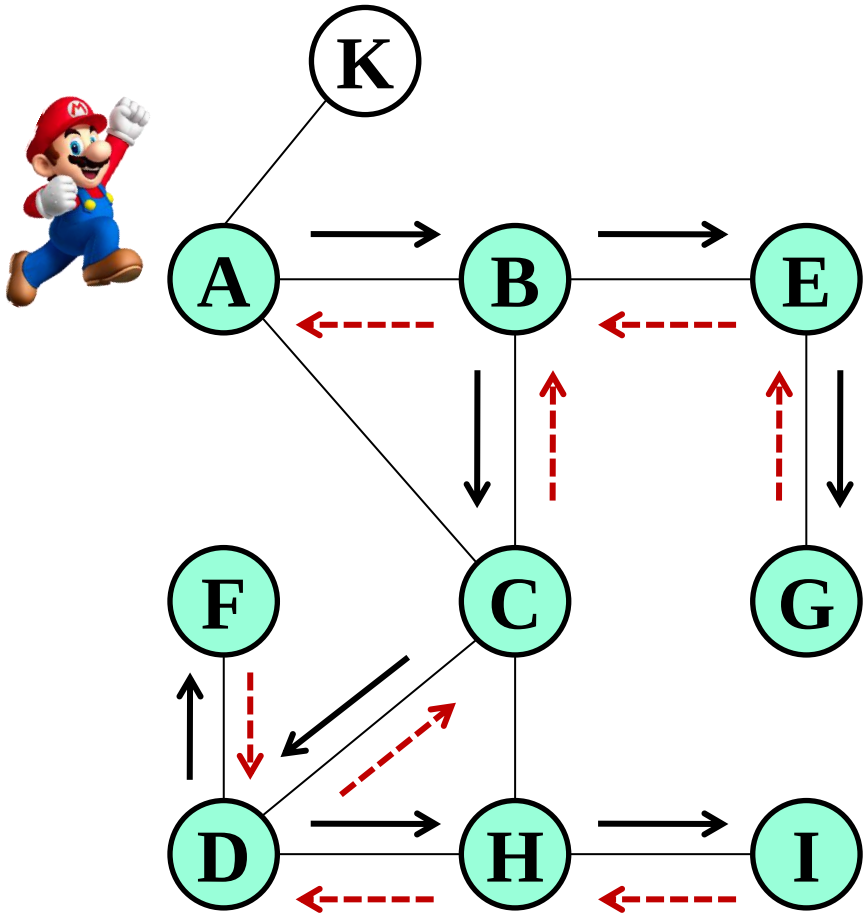
# 深度优先搜索 DFS ( Depth First Search, DFS)



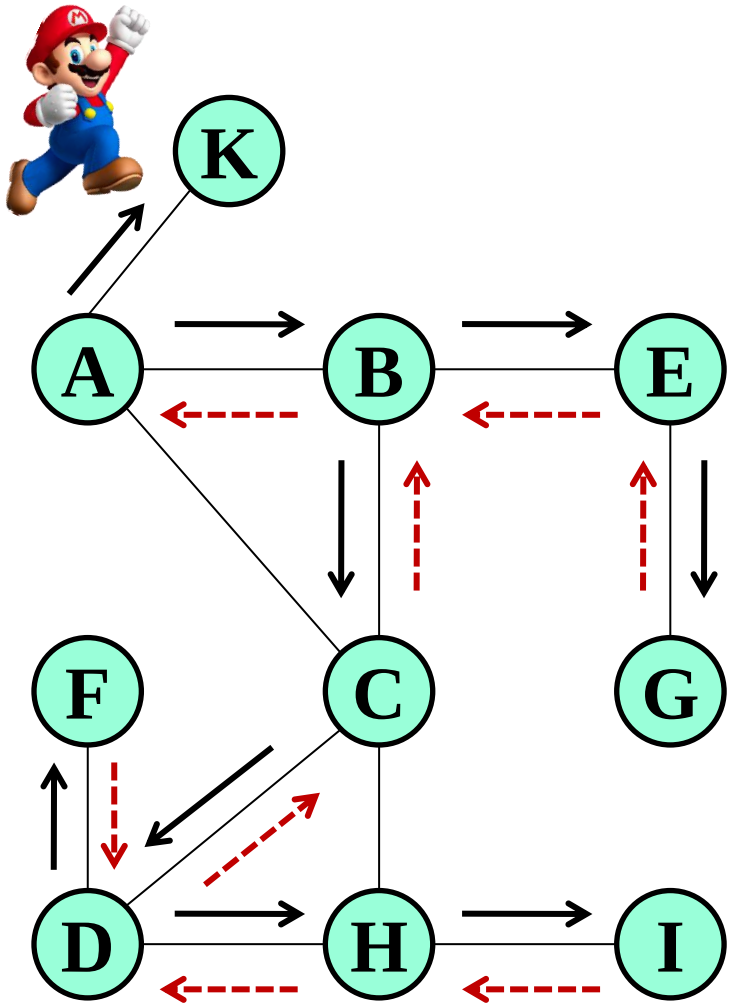

# 深度优先搜索 DFS ( Depth First Search, DFS)



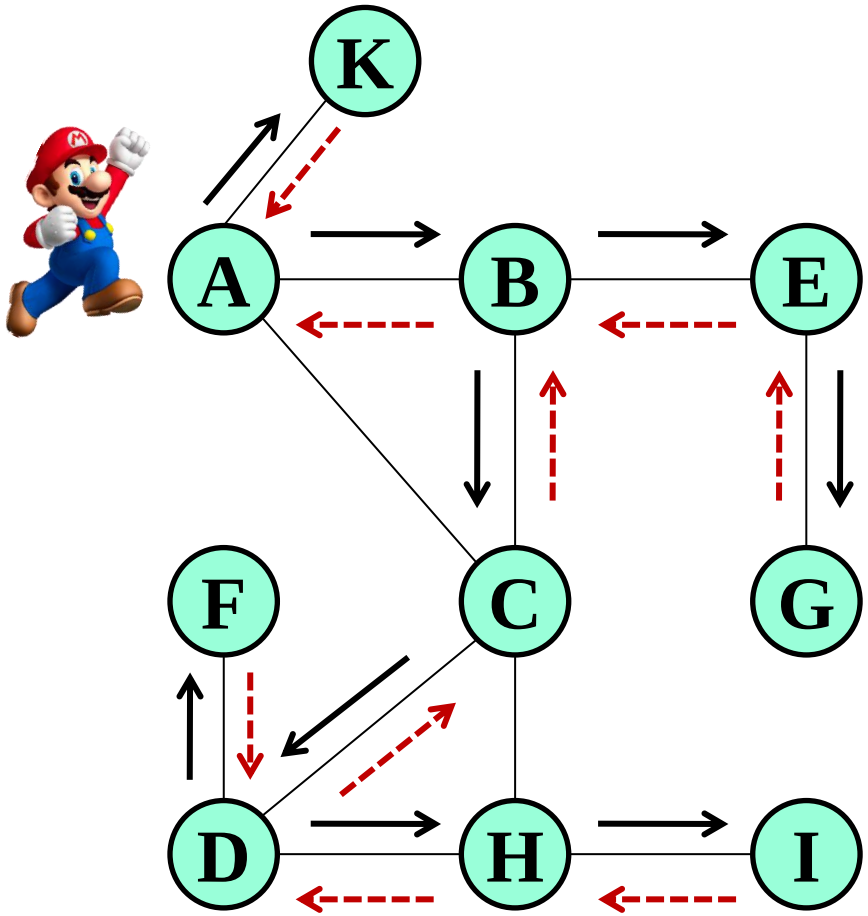
# 深度优先搜索 DFS ( Depth First Search, DFS)



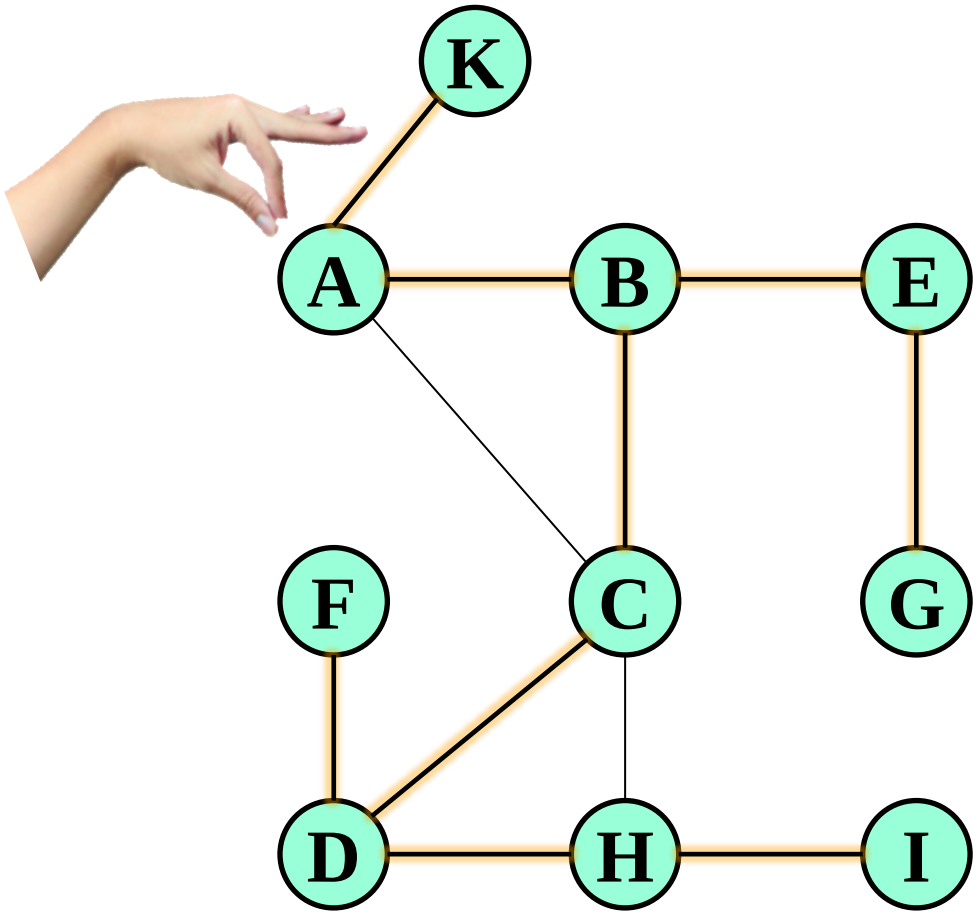
# 深度优先搜索 DFS ( Depth First Search, DFS)

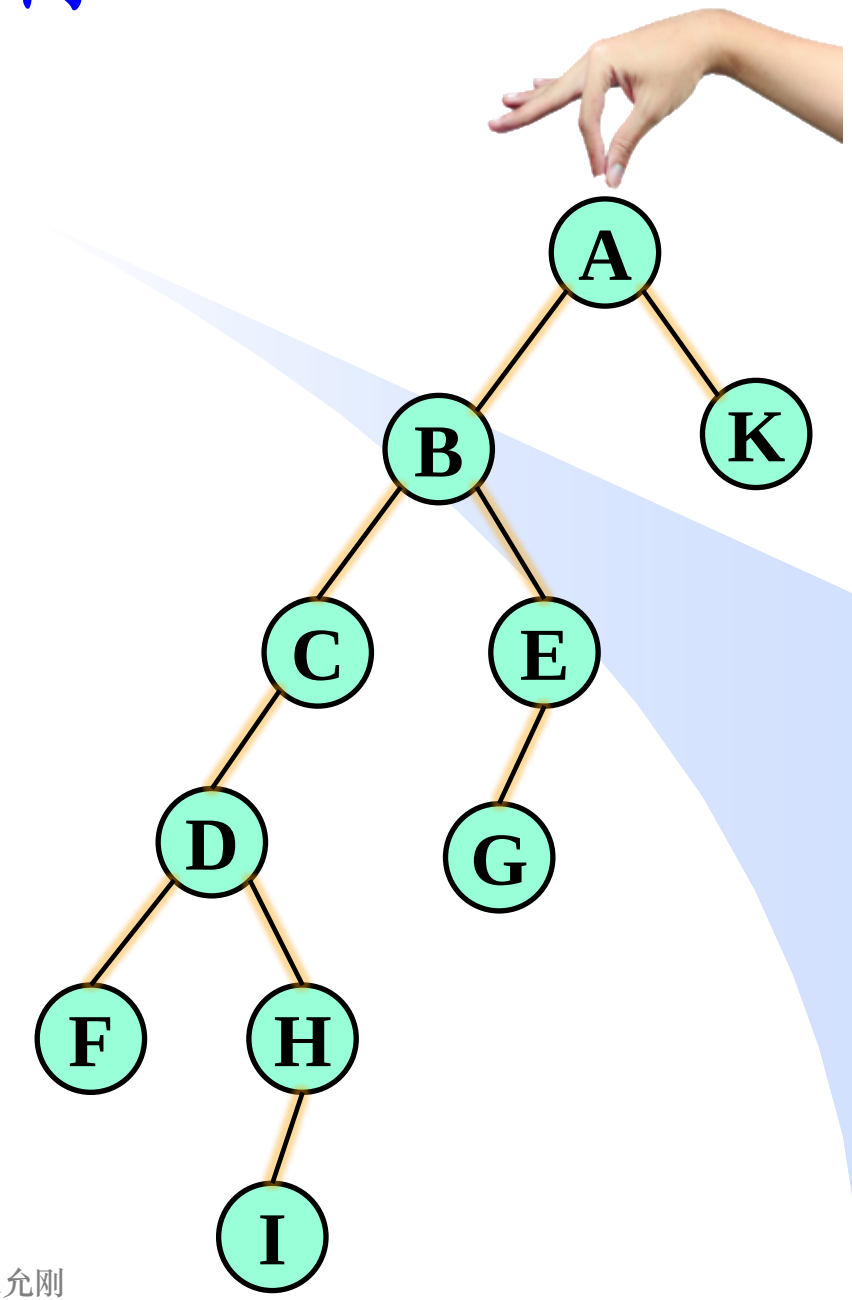


# 深度优先搜索 DFS ( Depth First Search, DFS)



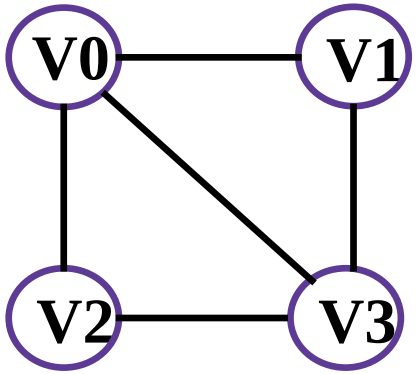
# 深度优先生成树



A

```
void DFS(Vertex*Head, int v, int visited[]){
    //以v为起点进行深度优先搜索，visited数组元素初值为0
    visit(v); visited[v]=1;//访问顶点v
    Edge* p= Head[v].adjacent;  //考察v的所有邻接顶点
    while(p!=NULL){
        int k = p->VerAdj; //考察p的邻接顶点k
        if(visited[k]==0)  //k未被访问
            DFS(Head,k,visited); //以k为起点继续深搜
        p=p->link;
    }
}
```

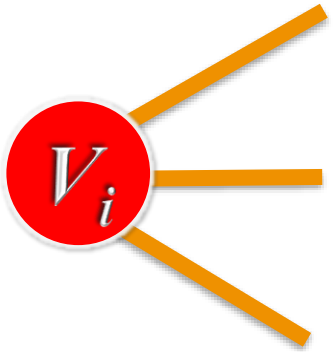



回溯
可行则进
不行则换
换不了则退

# 递归的另一版本

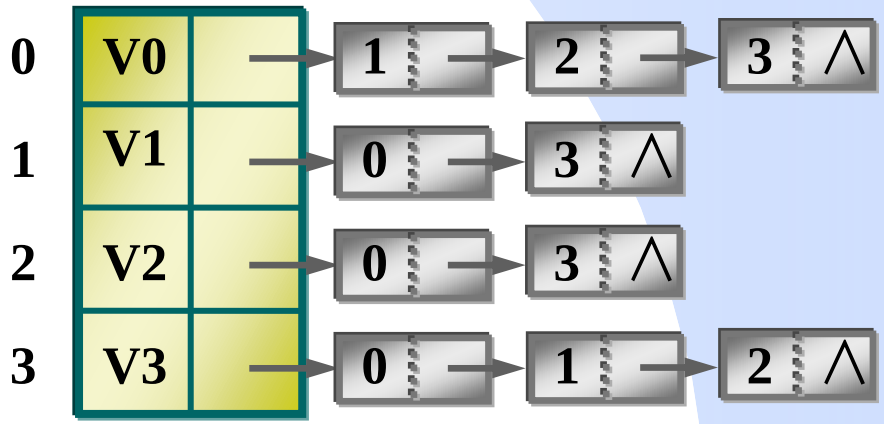
A

```
void DFS(Vertex* Head, int v, int visited[]){
    //以v为起点进行深度优先搜索，vis初值为0
    visit(v); visited[v]=1;  //访问顶点v
    for(Edge* p=Head[v].adjacent; p!=NULL; p=p->link)
       if(visited[p->VerAdj]==0) //考察v的邻接顶点p
          DFS(Head, p->VerAdj, visited);
}
```




$$\sum_{i=1}^{n}(1+d_i)=\sum_{i=1}^{n}1+\sum_{i=1}^{n}d_i$$

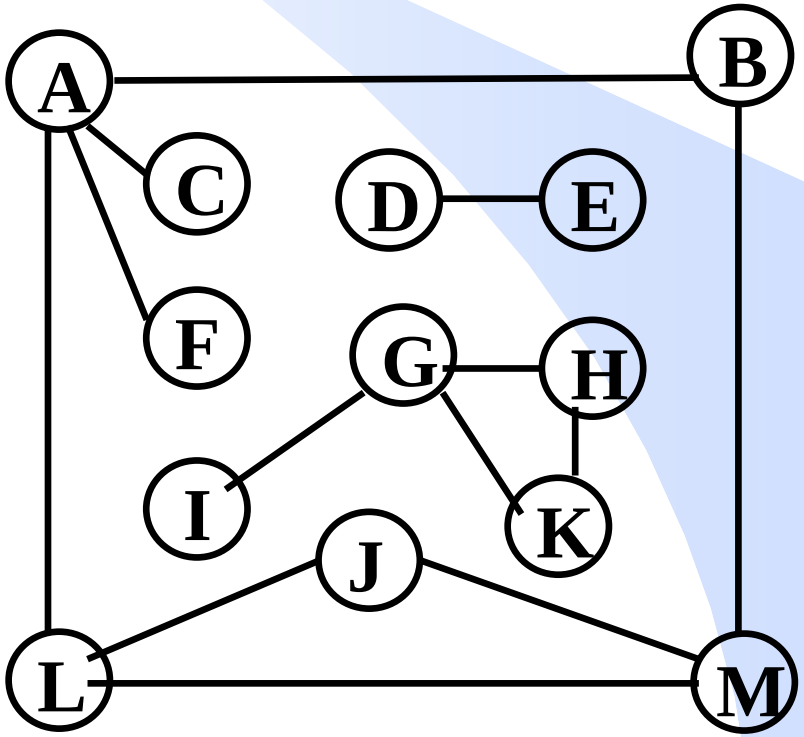
$$=O(n+e)$$



# 非连通图的深度优先遍历——需要多次调用DFS算法

A

```
for(int i=0;i<n;i++) //数组初始化
    visited[i] = 0;
//以每个顶点为起点，试探是否能深搜
for(int i=0;i<n;i++)
   if(visited[i]==0)
      DFS(Head, i, visited);
```




**一次DFS只能遍历一个连通分量**

# 图DFS的非递归算法

B

① 将所有顶点的visited[ ]值置为0, 初始顶点$v_0$压栈；

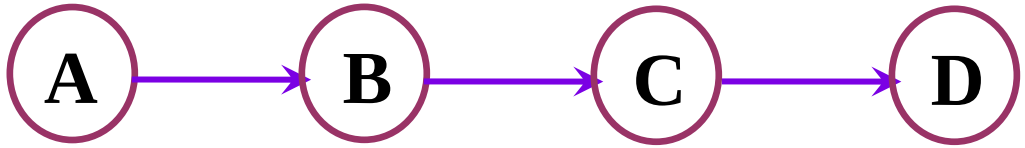
② 若栈为空，则算法结束；

③ 从栈顶弹出一个顶点$v$，如果$v$未被访问过则：

访问$v$，并将visited[$v$]值更新为1；

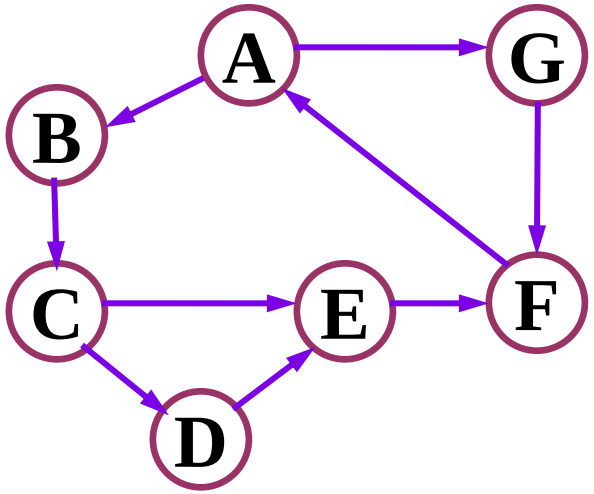
将$v$的未被访问的邻接顶点压栈；

④ 执行步骤②。




顶点弹栈时访问

# 例子




① 将所有顶点的visited[ ]值置为0, 初始顶点$v_0$压栈；

② 若栈为空，则算法结束；

③ 从栈顶弹出一个顶点$v$，如果$v$未被访问过则：

访问$v$，并将visited[$v$]值更新为1；

将$v$的未被访问的邻接顶点压栈；

④ 执行步骤 ② 。

(1) A

(2) G B 访问 A

(3) F B 访问 AG

(4) B 访问 AGF

(5) C 访问 AGFB

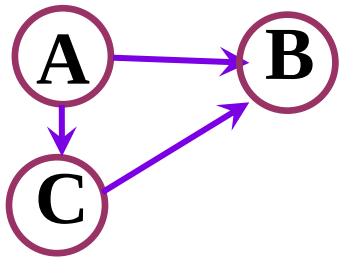
(6) E D 访问 AGFBC

(7) D 访问 AGFBCE

(8) 访问 AGFBCED

# 课下思考

B




① 将所有顶点的visited[ ]值置为0, 初始顶点$v_0$压栈；

② 若栈为空，则算法结束；

③ 从栈顶弹出一个顶点$v$，如果$v$未被访问过则：

访问$v$，并将visited[$v$]值更新为1；

将$v$的未被访问的邻接顶点压栈；

④ 执行步骤②。

---

(1)

| |
|---|
| A |

(2)

| |
|---|
| C |
| B |

访问
A

(3)

| |
|---|
| B |
| B |

访问
AC

(4)

| |
|---|
| |
| B |

访问
ACB

(5)

| |
|---|
| |

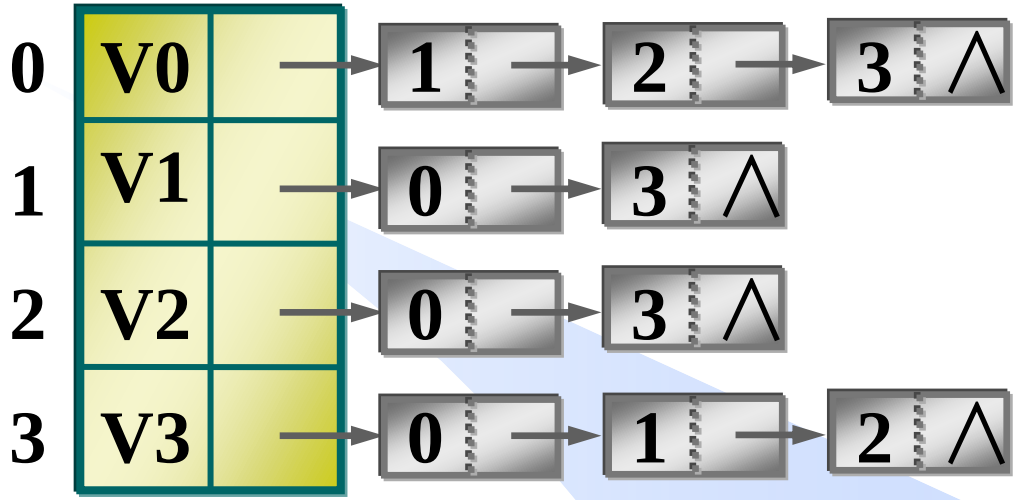
访问
ACB

顶点进栈时已确保未被访问，出栈时为何还判断是否被访问过？

# 图的深度优先遍历的非递归算法

B

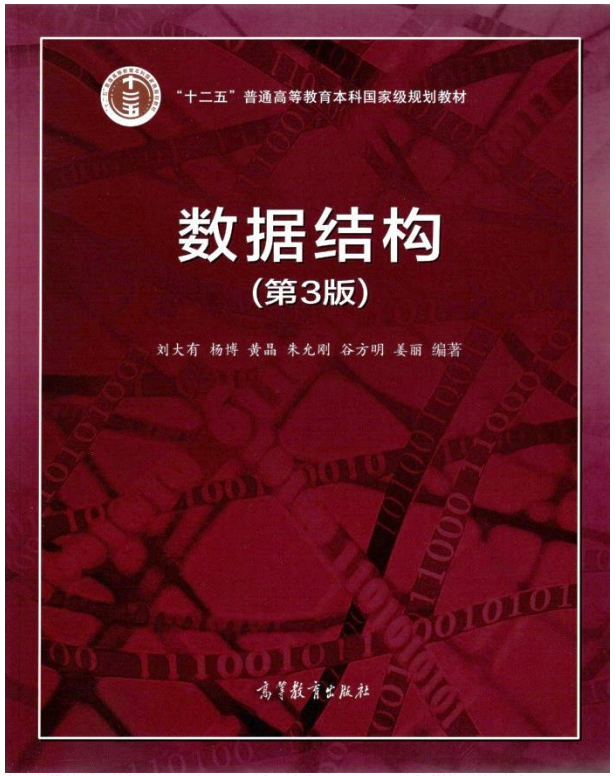
```
void DFS (Vertex* Head, int v, int visited[]){
    Stack S;  //创建栈 S
    for(i=0;i<n;i++)visited[i]=0;
    S.Push(v);  // 将v压入栈中
    while(!S.Empty()){
        v=S.Pop();
        if(visited[v]==0){
            visit(v); visited[v]=1;
            for(Edge* p=Head[v].adjacent; p!=NULL; p=p->link)
                if(visited[p->VerAdj]==0) //考察v的邻接顶点p
                    S.Push(p->VerAdj);
        }
}
```




每次访问一个顶点，共$n$个顶点

扫描每个顶点的边结点

时间复杂度$O(n+e)$




# 图的遍历及应用

- 深度优先搜索
- **广度优先搜索**
- 图遍历的应用

数据之法
结构之美
算法之道

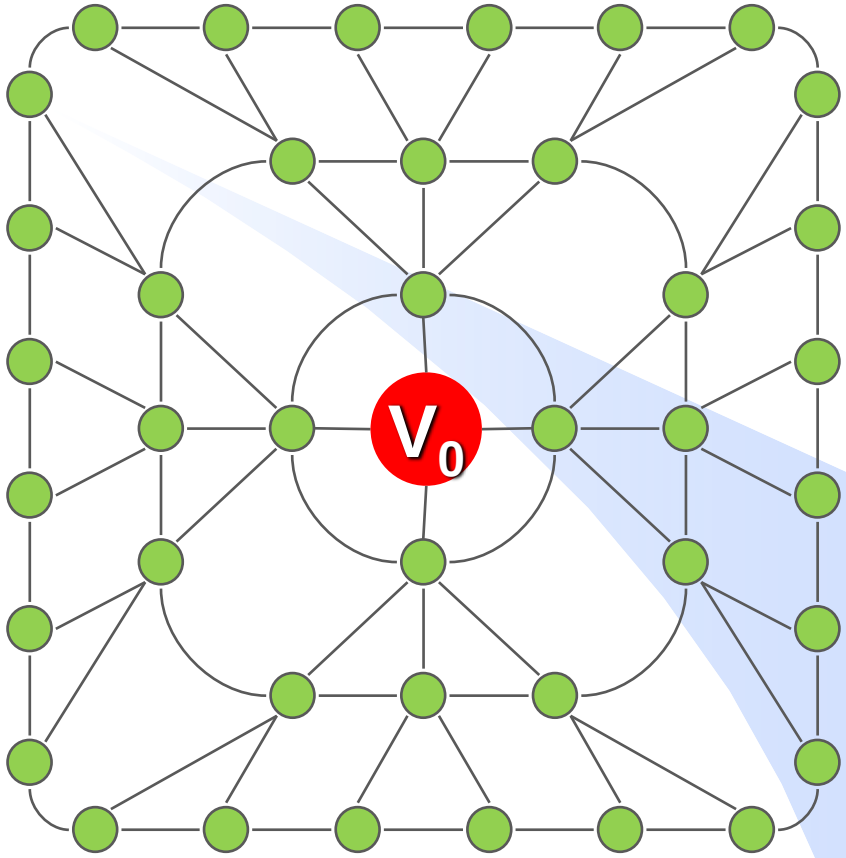
zhuyungang@jlu.edu.cn

Last updated on 2023.10

# 广度优先搜索（Breadth First Search, *BFS*）

A

- ✓ 访问初始点顶点$v_0$；
- ✓ 依次访问$v_0$的邻接顶点$w_1 \ldots w_k$；
- ✓ 然后再依次访问与$w_1 \ldots w_k$的尚未访问的邻接顶点；
- ✓ 再从这些被访问过的顶点出发，逐个访问与它们的尚未访问的邻接顶点……
- ✓ 依此往复，直至连通图中的所有顶点全部访问完。

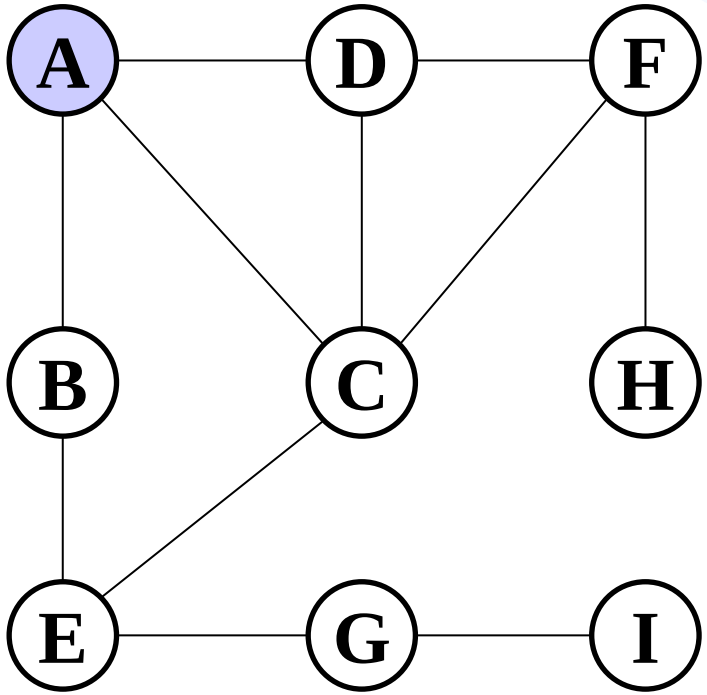



啊~五环，你比四环多一环

# 广度优先搜索 ( Breadth First Search, *BFS* )

A

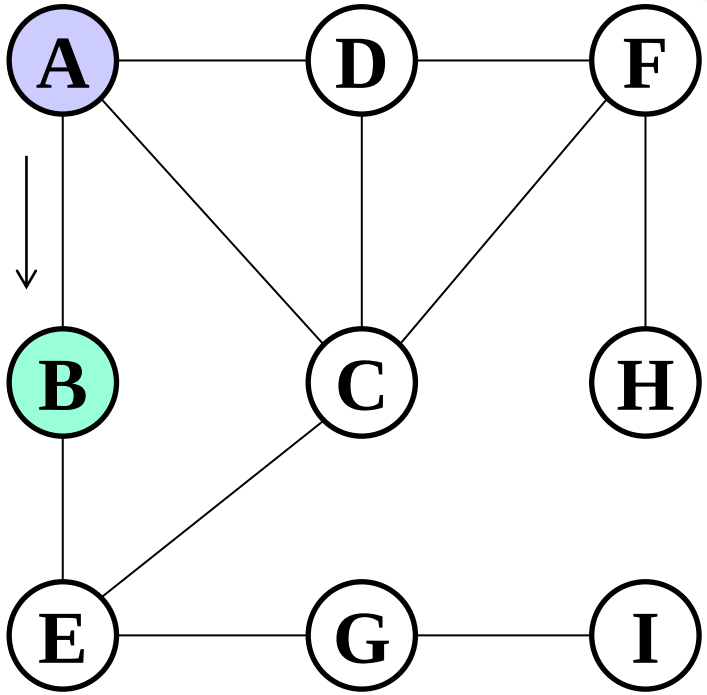
地毯式搜索
层层推进



# 广度优先搜索 ( Breadth First Search, *BFS* )

A

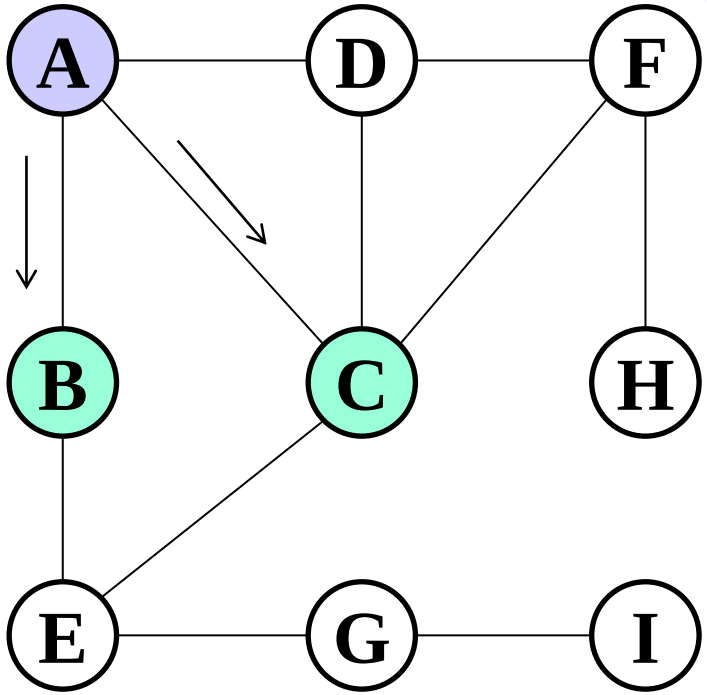
地毯式搜索
层层推进



# 广度优先搜索 ( Breadth First Search, *BFS* )

A

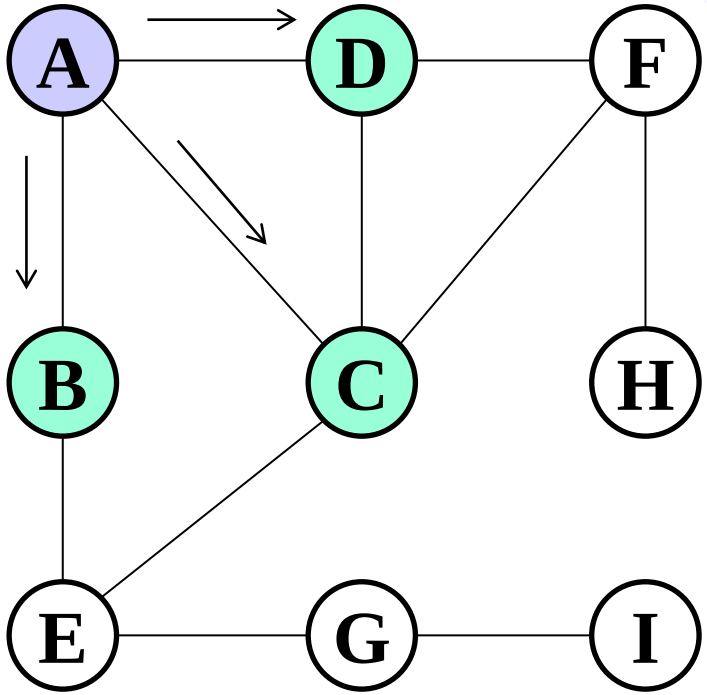
地毯式搜索
层层推进



# 广度优先搜索 ( Breadth First Search, *BFS* )

A

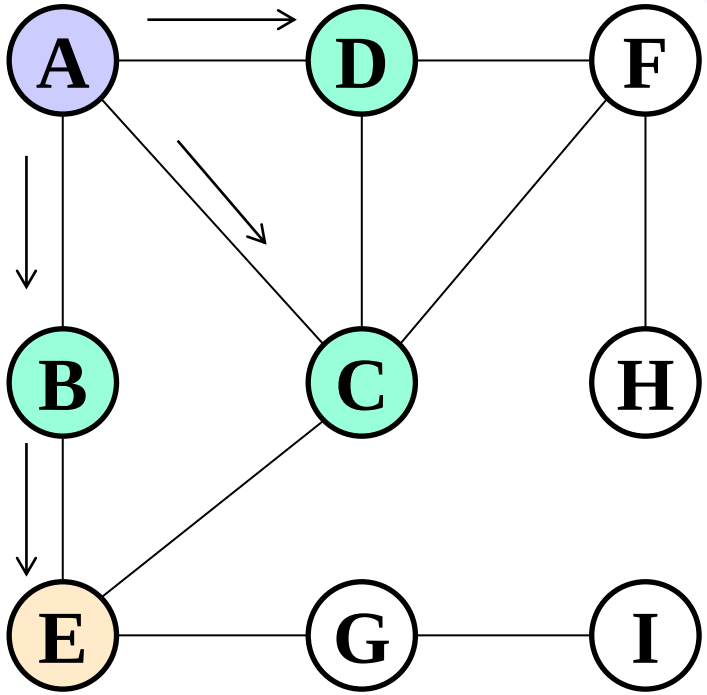
地毯式搜索
层层推进



# 广度优先搜索 ( Breadth First Search, *BFS* )

A

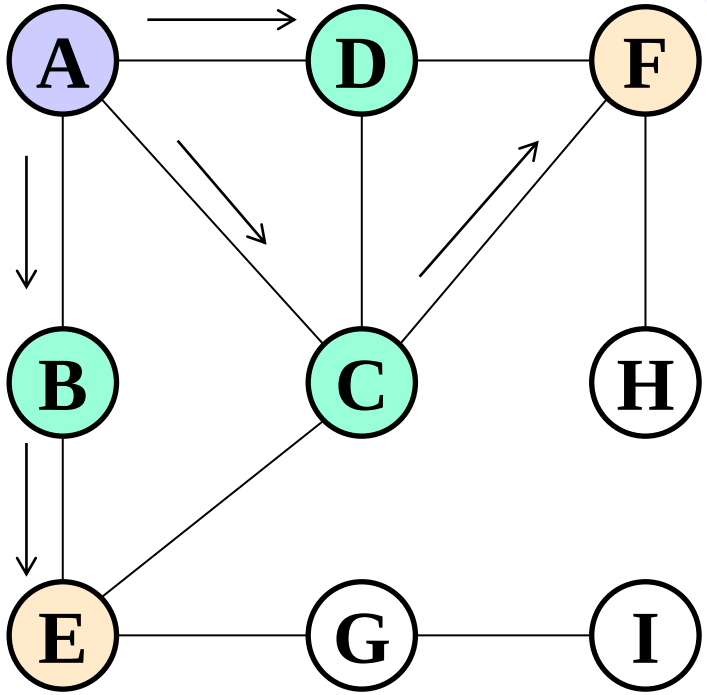
地毯式搜索
层层推进



# 广度优先搜索 ( Breadth First Search, *BFS* )

A

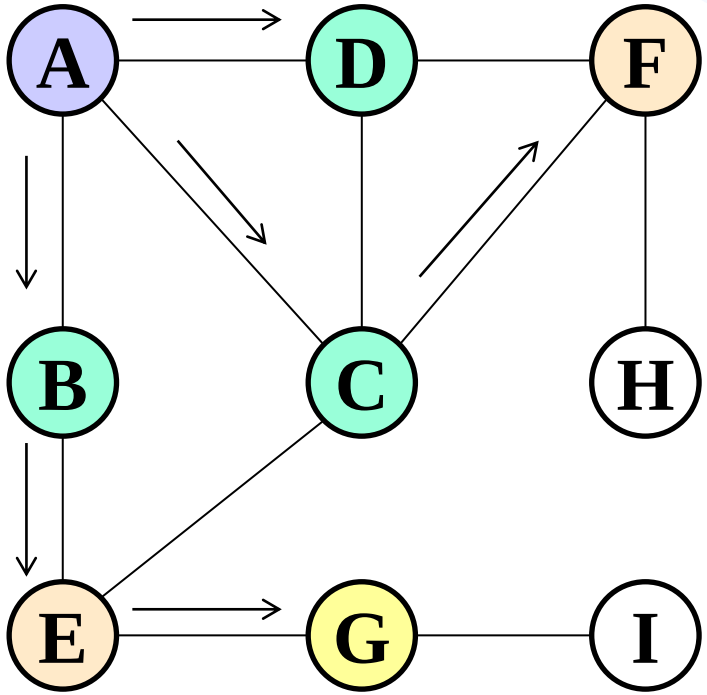
地毯式搜索
层层推进



# 广度优先搜索 ( Breadth First Search, *BFS* )

A

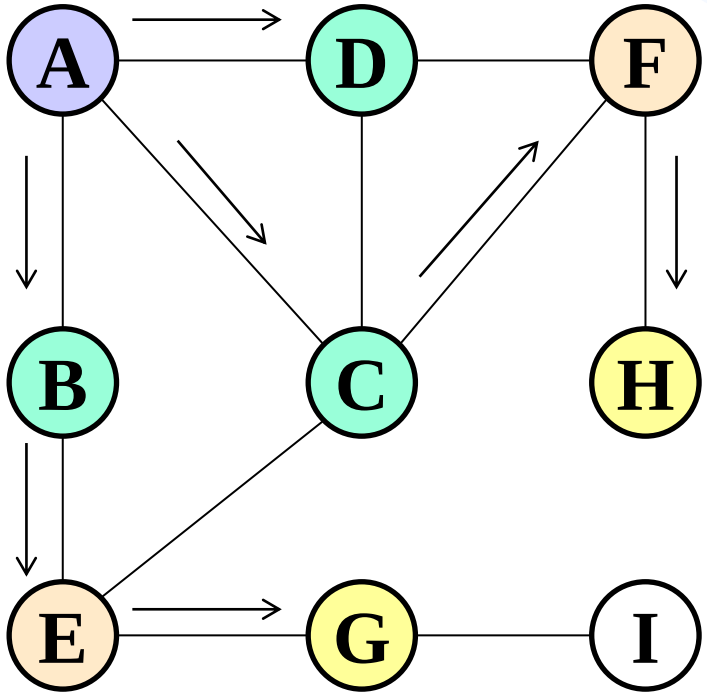
地毯式搜索
层层推进



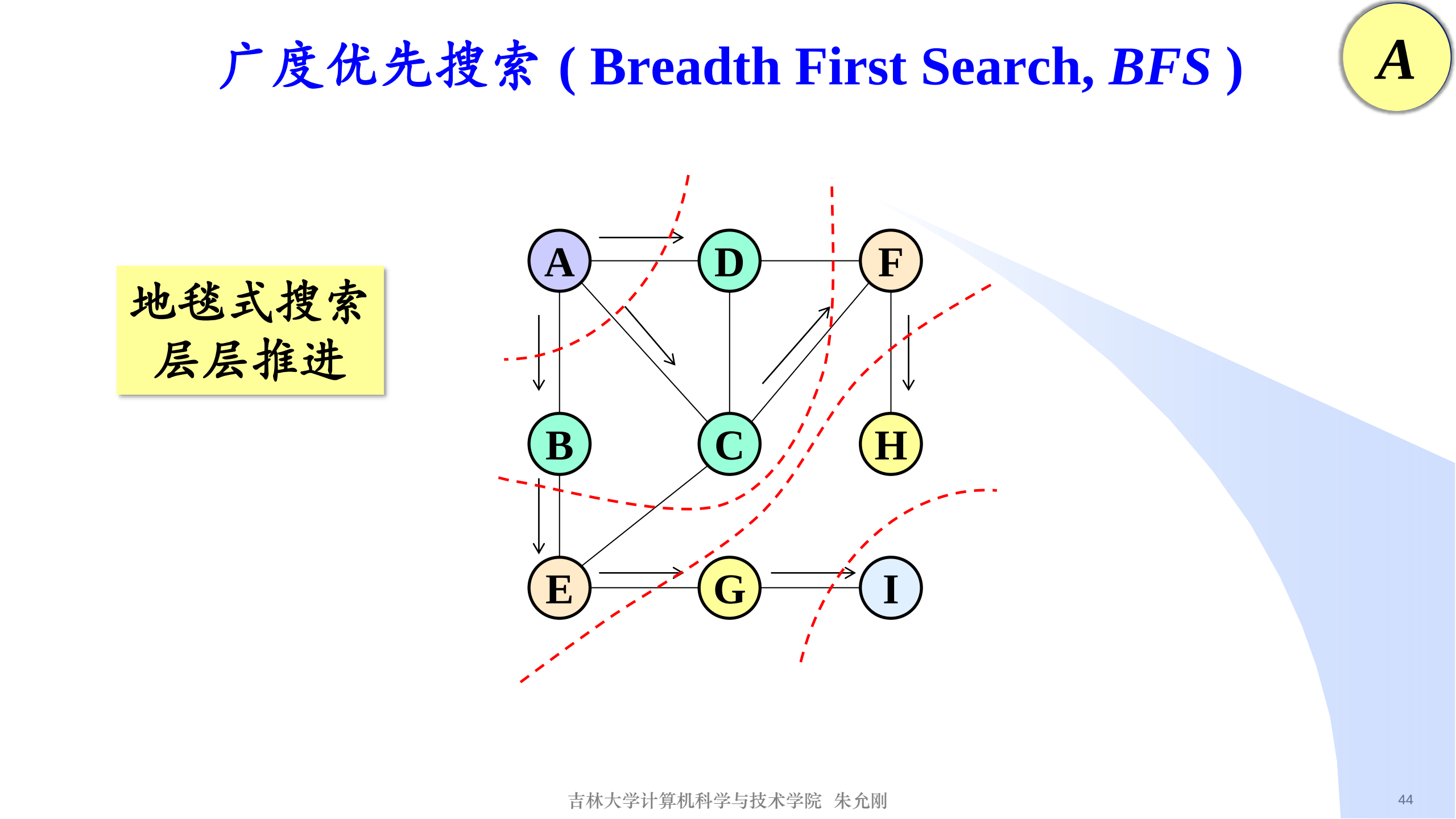
# 广度优先搜索 ( Breadth First Search, *BFS* )

A

地毯式搜索
层层推进

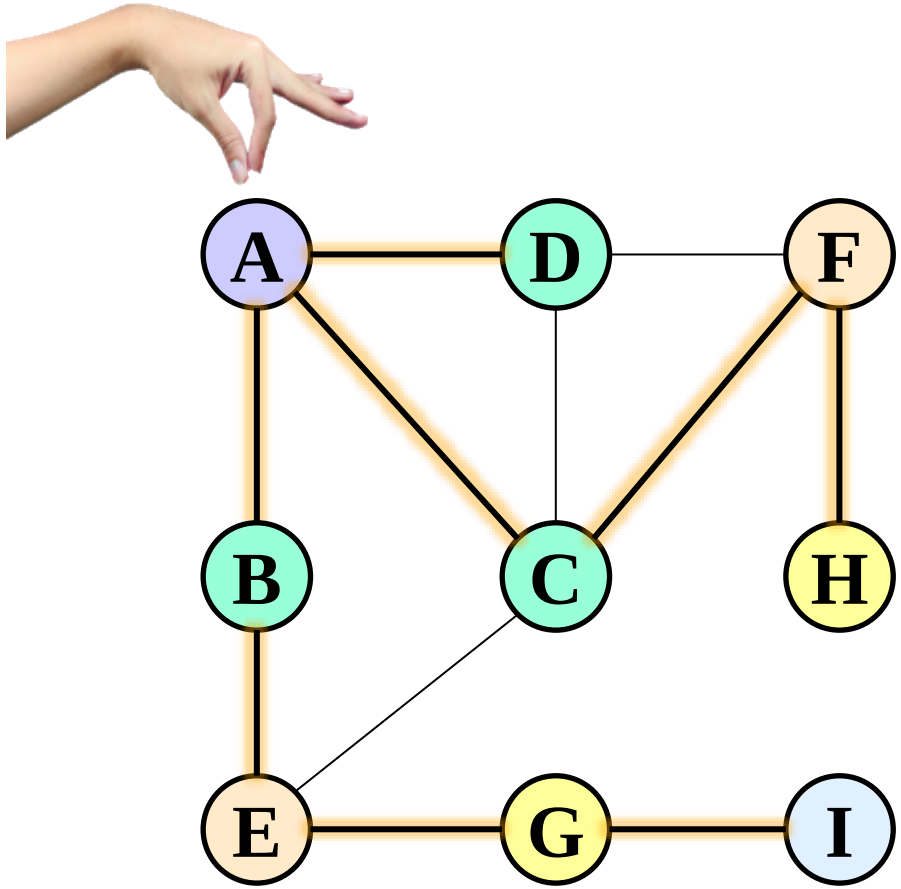


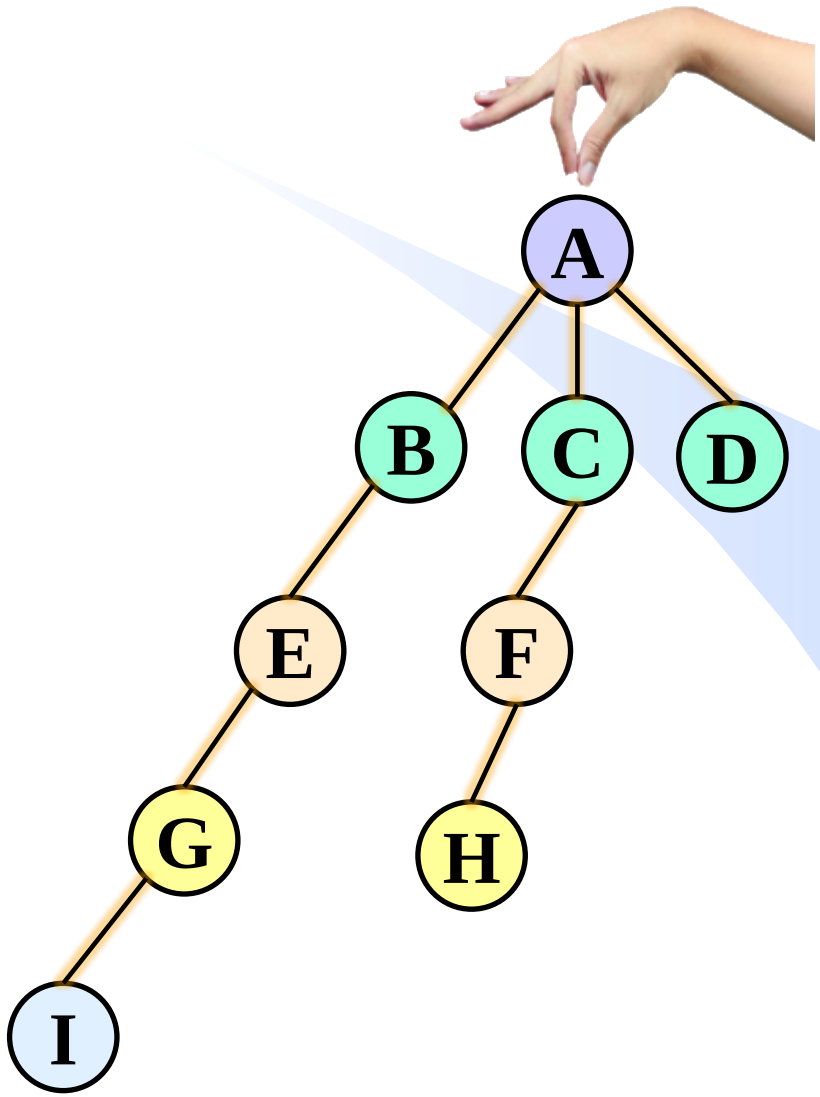
# 广度优先搜索 ( Breadth First Search, *BFS* )




地毯式搜索
层层推进

# 广度优先生成树



# 广度优先搜索 ( Breadth First Search, *BFS* )

A

- 图的广度优先遍历类似于树的层次遍历，是一种分层的搜索过程，每前进一层可能访问一批顶点，不像深度优先搜索那样有时需要回溯。因此，广度优先搜索不是一个递归过程，也不需要递归算法。
- 为了实现逐层访问，算法中使用一个队列，存储还未被处理的顶点，并用以确定正确的访问次序。
- 与深度优先搜索过程一样，为避免重复访问，需要一个辅助数组 ***visited*[ ]**，标识一个顶点是否被访问过。

# 广度优先搜索 ( Breadth First Search, *BFS* )

A

① 将所有顶点的visited[ ]值置为0，访问起点v，置visited[*v*]=1，*v*入队；

② 检测队列是否为空，若队列为空，则算法结束；

③ 出队一个顶点*v*，考察其每个邻接顶点*w*：

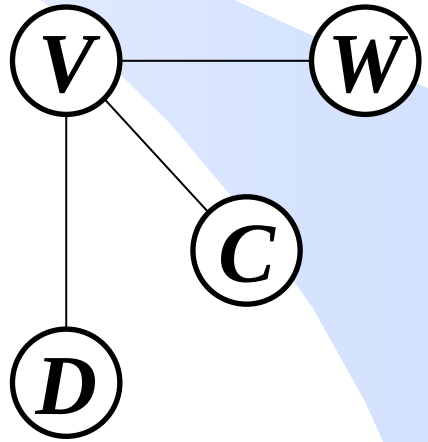
如果*w*未被访问过，则

访问*w*；

将visited[*w*]值更新为1；

将*w*入队；

④ 执行步骤②。




顶点入队时访问

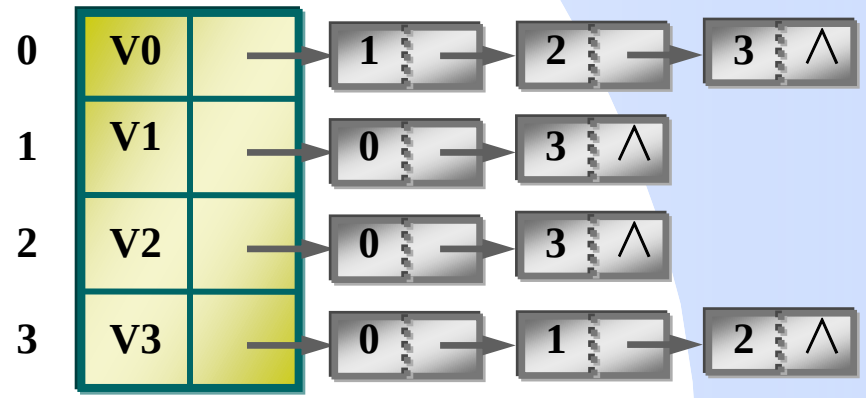
A

```
void BFS(Vertex* Head, int v, int n, int visited[]){
    Queue Q;  //创建队列Q, 队列需预先实现
    for(int i=0; i<n; i++) visited[i]=0;
    visit(v); visited[v]=1;//访问点v
    Q.Enqueue(v);              //起点v入队
    while(!Q.Empty()){
        v=Q.Dequeue();     //出队一个点
        for(Edge* p=Head[v].adjacent; p!=NULL; p=p->link)
            if(visited[p->VerAdj]==0){ //考察v的邻接顶点p
                visit(p->VerAdj);
                visited[p->VerAdj]=1;
                Q.Enqueue(p->VerAdj);
            } //end if
    } //end while
} //end BFS
```

for循环：扫描每个顶点的边结点

每个顶点被出队一次

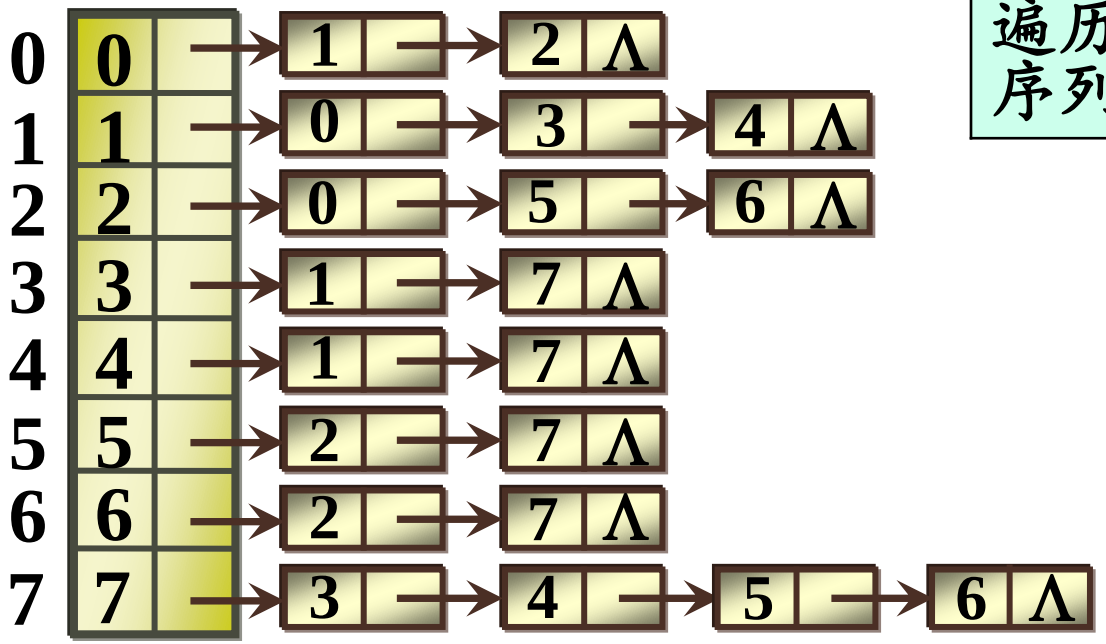
时间复杂度O(*n*+*e*)



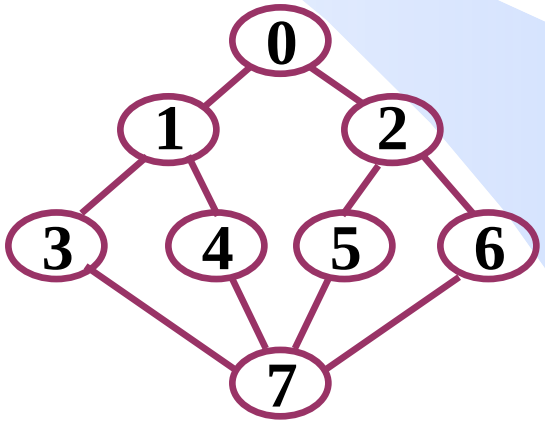
```
while(!Q.Empty()){
    v=Q.Dequeue();  //出队一个点
    for(Edge* p=Head[v].adjacent; p!=NULL; p=p->link)
        if(vis[p->VerAdj]==0){ //考察v的邻接顶点p
            printf("%d ", p->VerAdj); vis[p->VerAdj]=1;Q.Enqueue(p->VerAdj);
        }
}
```

# 顶点入队时访问

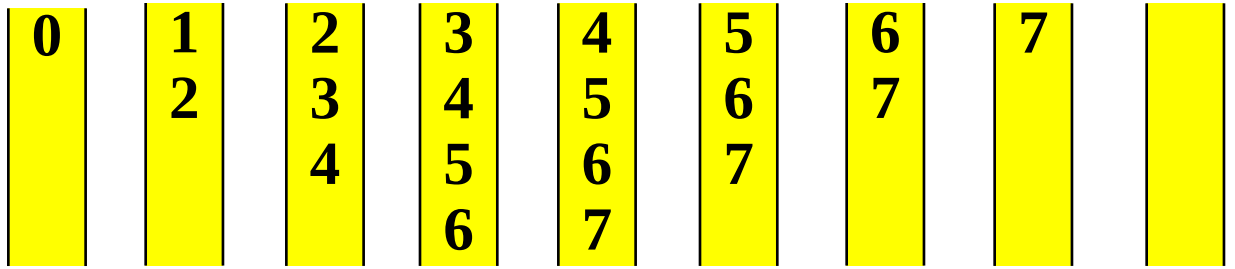
A




| 遍历序列 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|




队列 Q 的变化，上面是队头，下面是队尾。



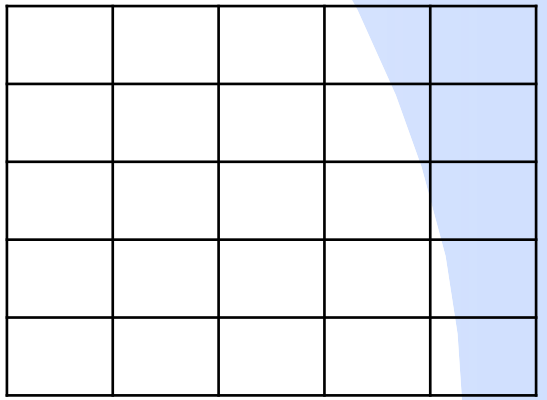
# 广度优先搜索 ——邻接矩阵存图

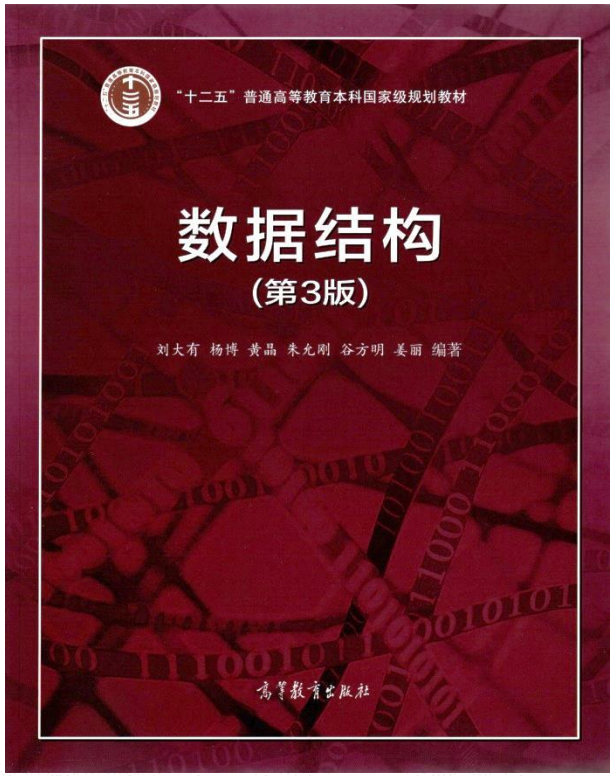
A

```
void BFS(int G[N][N],int v,int n,int vis[])//邻接矩阵存图
    Queue Q;  //创建队列Q
    for(int i=0; i<n; i++) visited[i]=0;
    visit(v); visited[v]=1;//访问点v
    Q.Enqueue(v);  //起点v入队
    while(!Q.Empty()){
        v=Q.Dequeue();    //出队一个点
        for(int w=0; w<n; w++)
            if(G[v][w]==1 && visited[w]==0){//考察v的邻接顶点
                visit(w); visited[w]=1;
                Q.Enqueue(w);
            } //end if
    } //end while
} //end BFS
```

for循环：扫描顶点 $i$ 的邻接顶点需遍历矩阵第 $i$ 行，时间$O(n)$

时间复杂度$O(n^2)$






# 图的遍历及应用

- 深度优先搜索
- 广度优先搜索
- **图遍历的应用**

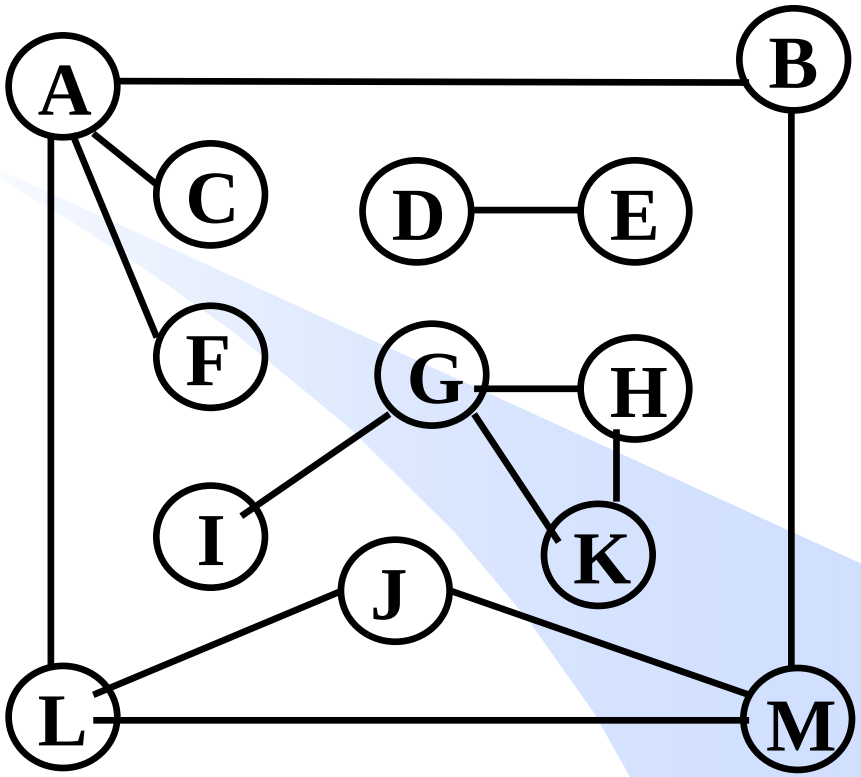
数据之法
结构之美
算法之道

zhuyungang@jlu.edu.cn

Last updated on 2023.10

# 判断无向图是否连通及连通分量数目【谷歌、微软、苹果、英特尔面试题】

A




## 方案1：DFS

```
for(int i=0;i<n;i++) visited[i]=0;
int cnt=0; //连通分量数目
for(int i=0;i<n;i++)
    if(visited[i]==0){
        DFS(Head, i, visited);
            cnt++; //每遍历一个连通分量，计数器加1
    }
```

每调用一次DFS，遍历一个连通分量
cnt等于1即为连通图

时间复杂度O(*n*+*e*)

# 判断无向图是否连通及连通分量数目【谷歌、微软、苹果、英特尔面试题】

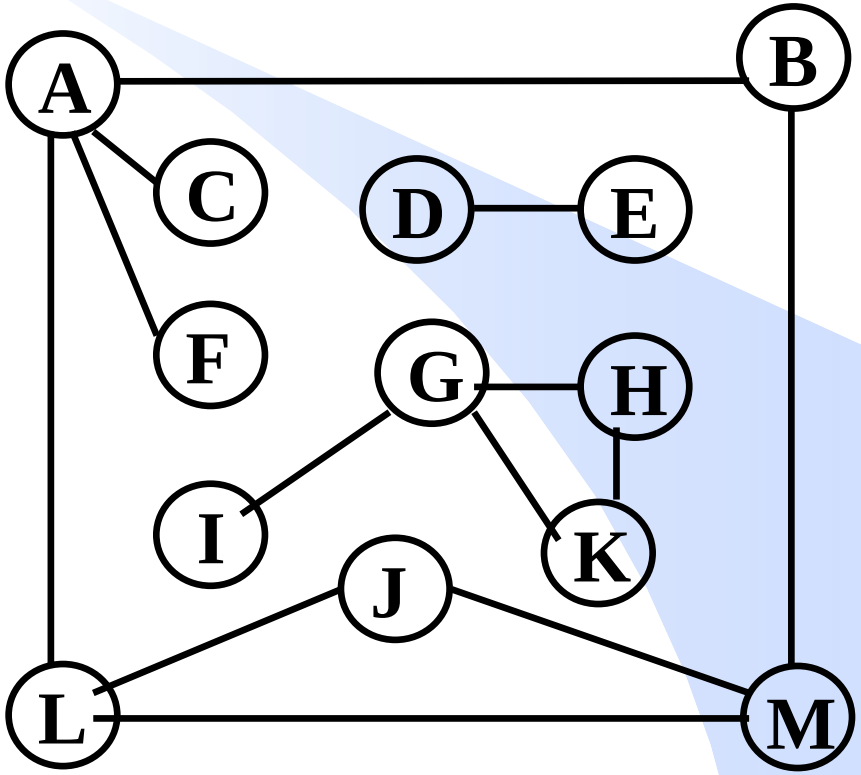
B

方案2：并查集

连通分量⇔集合

每次读入一条边(x,y)，Union(x,y);
扫描完所有边后：集合数即连通分量数

时间复杂度O(*n*+*e*)



# 判断图中顶点u到v是否存在路径【华为、谷歌、微软、苹果、亚马逊面试题】

A

- 以u为起点遍历，看遍历过程中是否经过v

```
bool DFS(Vertex* Head, int u, int v, int visited[]){
    visited[u]=1;   //访问顶点u
    if(u==v) return true; //走到顶点v了
    for(Edge* p=Head[u].adjacent; p!=NULL; p=p->link)
        if(visited[p->VerAdj]==0) //考察u的邻接顶点
            if(DFS(Head,p->VerAdj,v,visited)) return true;
    return false;
}
```

# 课下思考

B

- 给定$n$个顶点的有向无环图，输出顶点0到顶点$n$-1的所有路径
  【字节跳动、阿里、谷歌、微软、苹果面试题】

# 课下思考

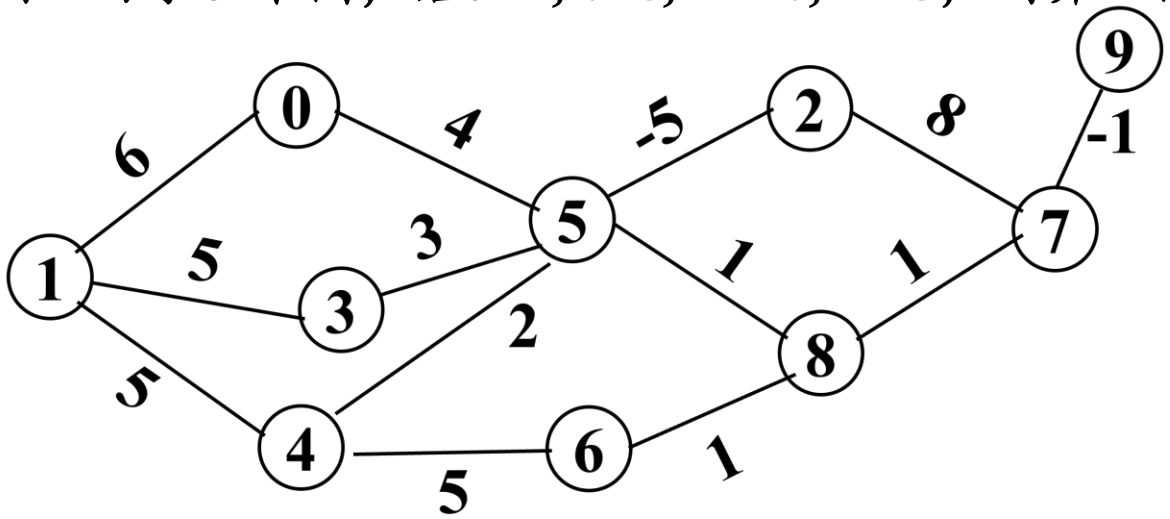
B

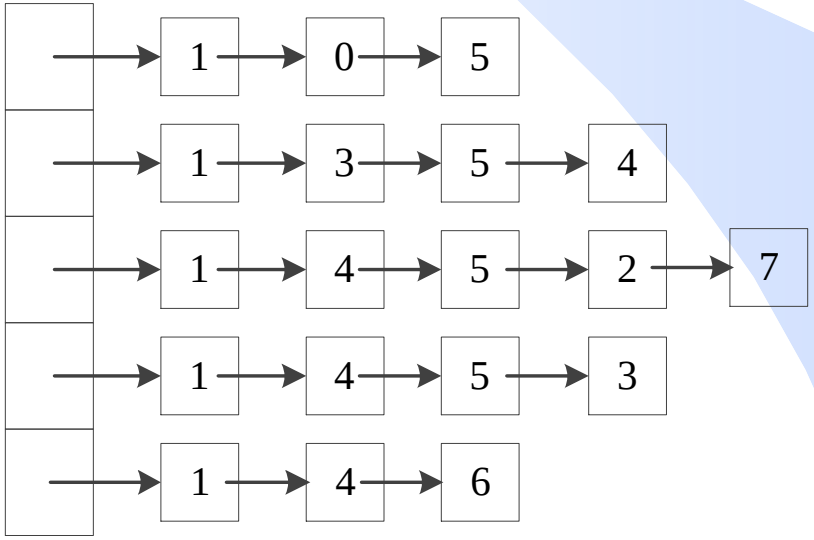
一个带权图G以邻接表作为存储结构，顶点编号为0至*n*-1，边权可正可负。请设计一个算法，找出从指定顶点*u*出发的、不经过顶点*v*(*v*≠*u*)的、包含总顶点数不超过*m* (*m* ≥ 2)的、长度等于*L*的所有简单路径，并将每条路径保存在一个单链表中，所有单链表的头指针存入一个指针数组。即实现如下函数：

```
int FindPath(Vertex Head[ ], int n, int u, int v, int L, int m, ListNode* PathList[ ])
```

其中Head为图G的顶点表，n、u、v、L、m含义如题目所述，PathList为存储各路径链表头指针的数组，函数返回值为满足条件的路径条数。

例如对于左下图，若u=1, v=8, L=10, m=5，则算法得到右下图所示的结果。




（1）给出算法的基本设计思想。

（2）根据设计思想，采用ADL、C/C++、Java等语言描述算法，关键之处给出注释。

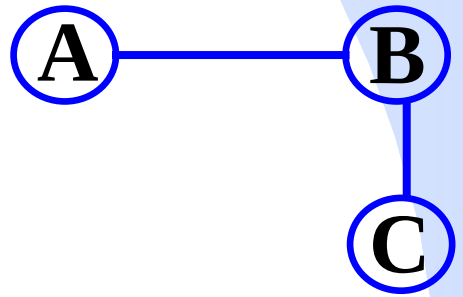
（3）给出算法的时间复杂度。【2020级期末考试题，15分】

# 判断无向图中是否有环

B

（1）深度优先搜索，遍历过程中遇到之前访问过的顶点（不能是当前访问点的前驱），即有环

```
bool DFS(Vertex* Head, int v, int visited[], int pre){
    //以v为起点DFS，pre为v之前访问的结点
    visited[v]=1;   //访问顶点v
    for(Edge* p=Head[v].adjacent; p!=NULL; p=p->link)
        if(visited[p->VerAdj]==0) { //考察v的邻接顶点
            if(DFS(Head,p->VerAdj,visited,v)) return true;
        }
        else if(p->VerAdj!=pre) return true;
        //p之前被访问过，且不是pre的邻接顶点
    return false;
}
```



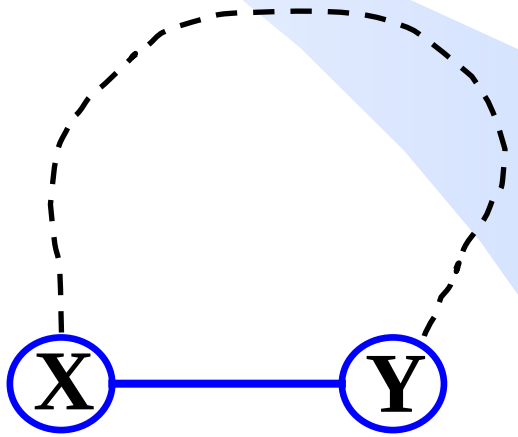
# 判断无向图中是否有环

B

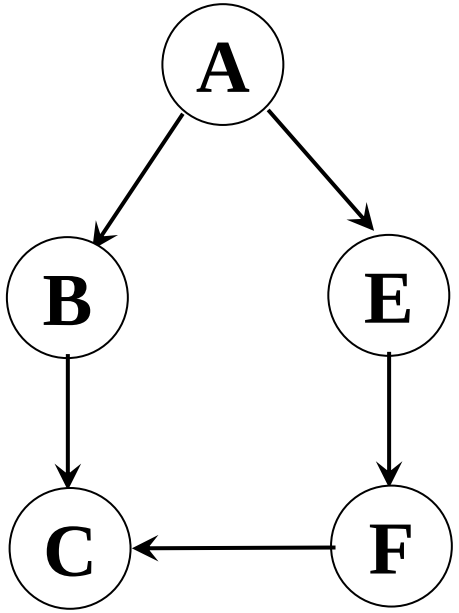
## （2）并查集（连通关系⇔等价关系）

每次读入一条边(x,y)

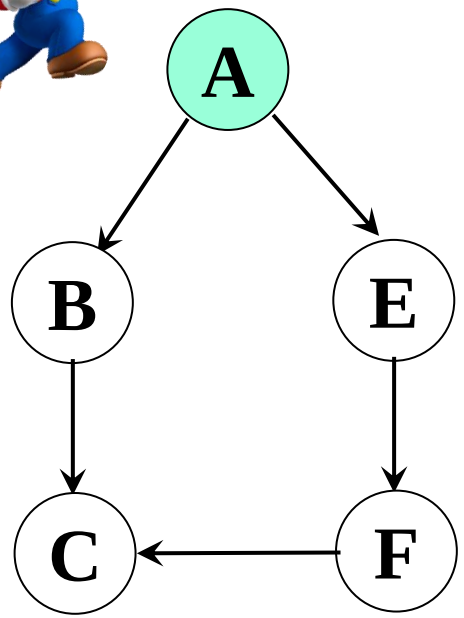
```
if(Find(x)==Find(y))
    return true;
else Union(x, y);
```



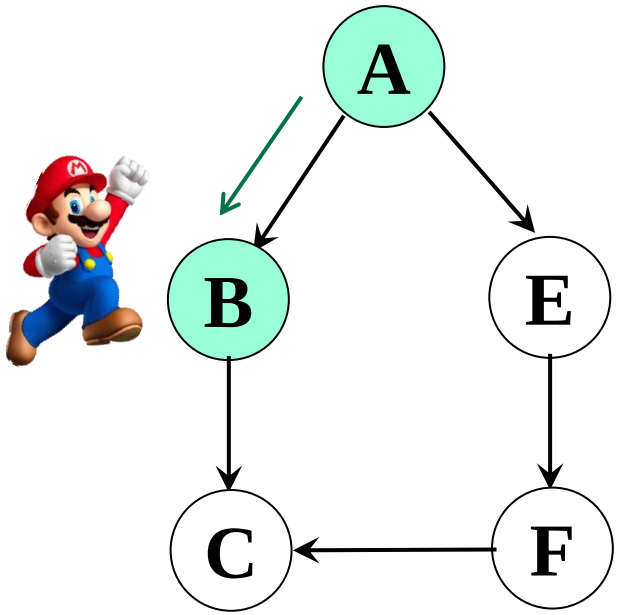
# 判断有向图中是否有环



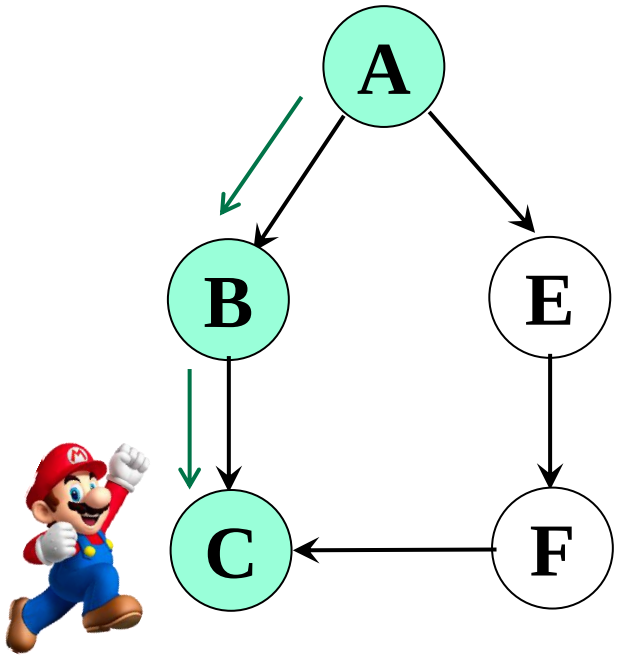
# 判断有向图中是否有环



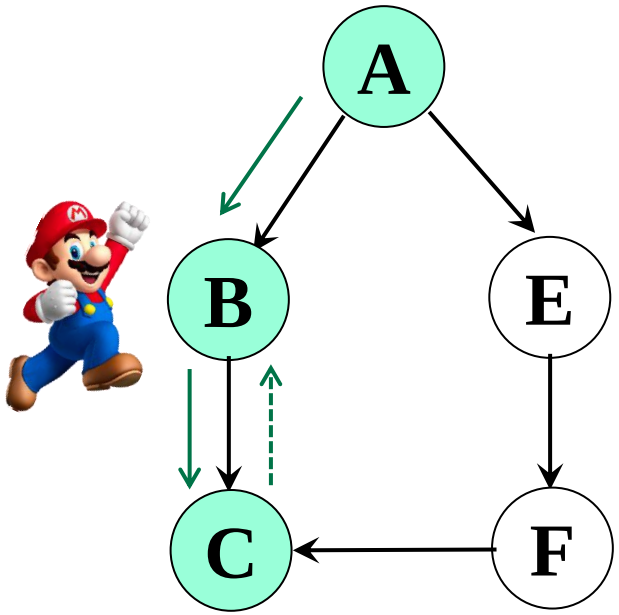
# 判断有向图中是否有环



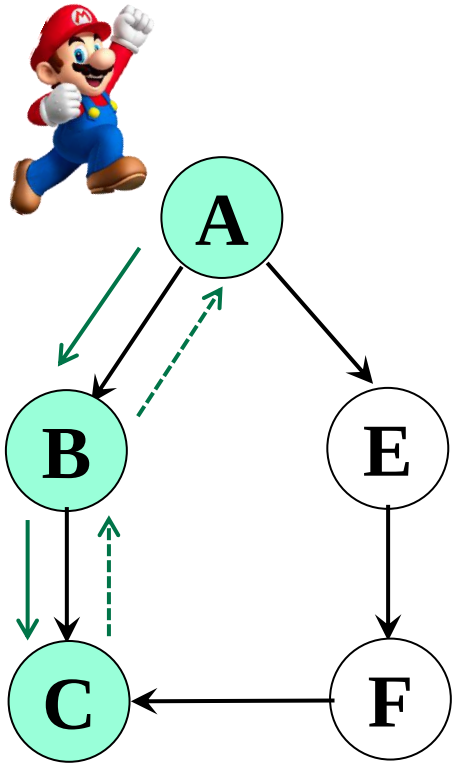
# 判断有向图中是否有环



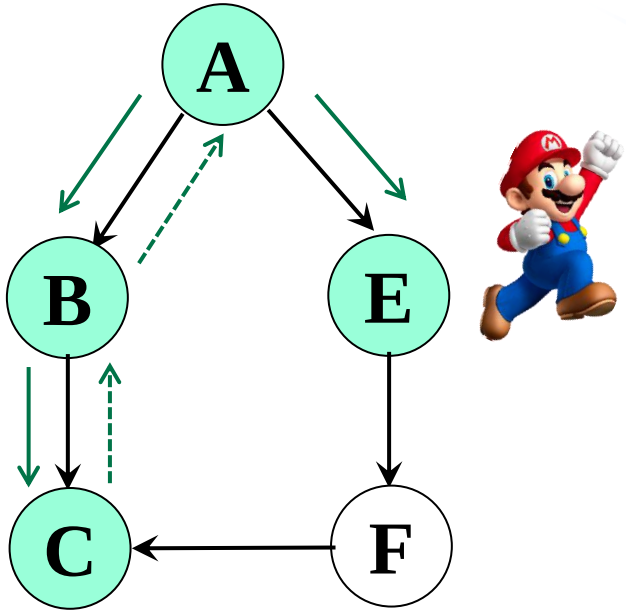
# 判断有向图中是否有环



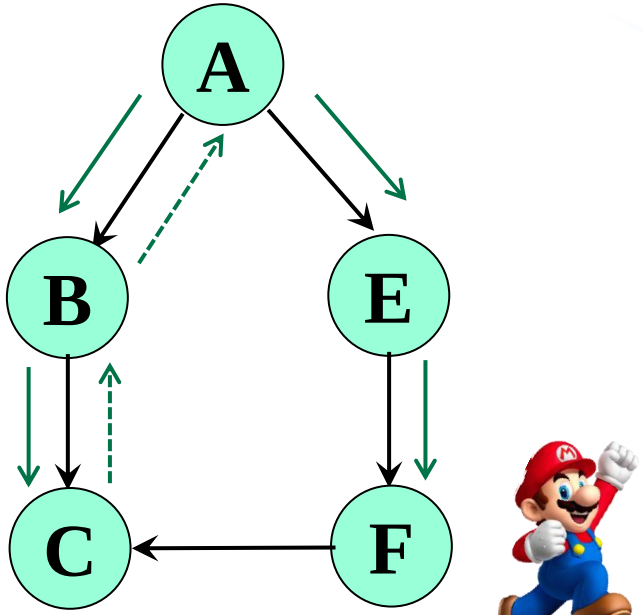
# 判断有向图中是否有环



# 判断有向图中是否有环

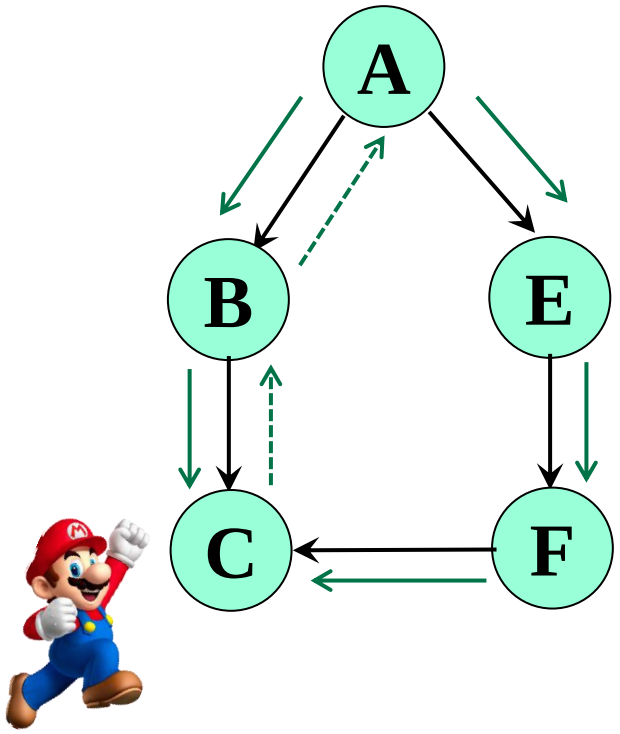


# 判断有向图中是否有环



# 判断有向图中是否有环

B




有环：DFS过程中遇到已访问过的点，且该点在当前正在探索的路线上。

对于已访问的点，应区分是否在当前正在探索的路径上

# 判断有向图中是否有环

B

有向图：在深度优先搜索过程中，一个顶点会经历3种状态：

- *visited*[*i*]=0，顶点*i*尚未被访问到。
- *visited*[*i*]=1，顶点*i*已经被访问过，但对于它的遍历尚未结束。该顶点还有若干邻接顶点尚未访问，当前算法正在递归地深入探索该顶点的某一邻接顶点。顶点*i*在当前探索的路径上。
- *visited*[*i*]=2，顶点*i*的所有邻接顶点已完成遍历，其自身的遍历也已结束。

在DFS过程中：

✓当前正在探索的路线（路径）上的点，visited值是1;

✓已经探索完的路线（路径）上的点，visited值是2;

✓还没探索的路线（路径）上的点，visited值是0.

# 判断有向图中是否有环

B

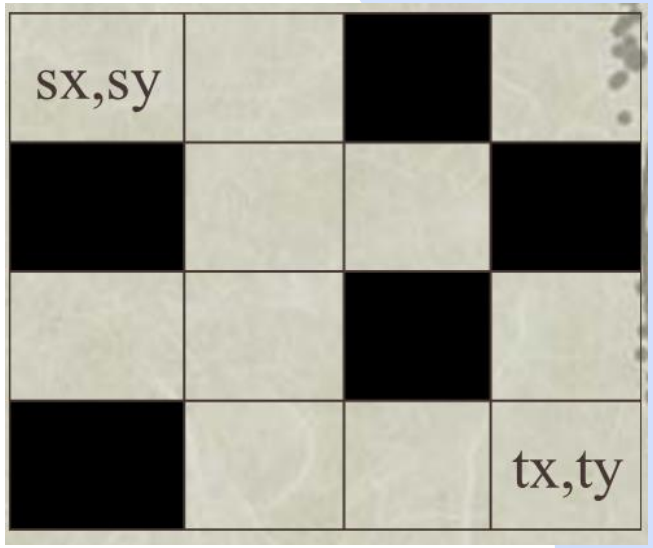
在DFS过程中，一个顶点的visited值按照0、1、2的顺序变化。

✓遍历开始前每个顶点visited值都是0;

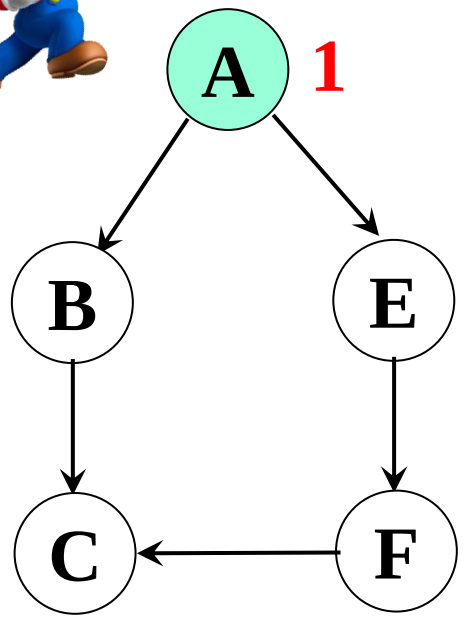
✓当一个顶点刚被发现时,其visited值置为1,算法递归地遍历其邻接顶点;

✓当一个顶点所有邻居都完成遍历,该点visited值变为2,其自身也完成遍历.

```
void DFS(Vertex* Head, int v, int visited[]){//vis初值为0
    visit(v); visited[v]=1;  //访问顶点v
    for(Edge*p=Head[v].adjacent; p!=NULL; p=p->link)
       if(visited[p->VerAdj]==0)
            DFS(Head, p->VerAdj, visited);
    visited[v]=2;
}
```

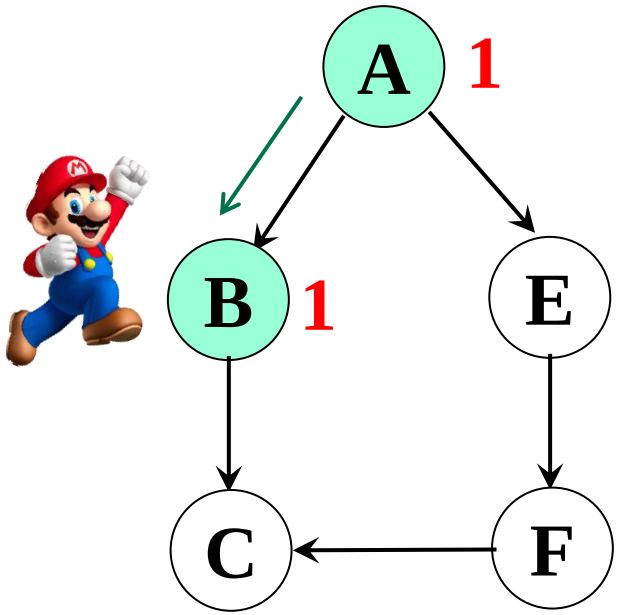


# 判断有向图中是否有环



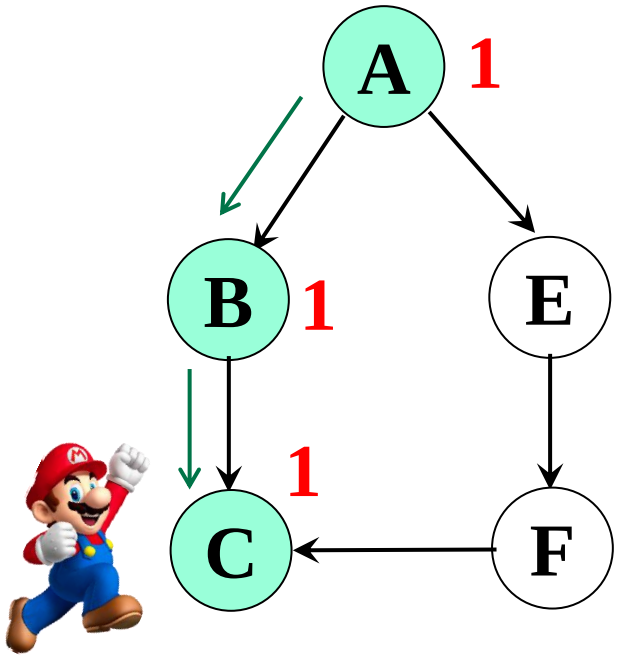

```
void DFS(Vertex* Head, int v, int visited[]){
   visit(v); visited[v]=1;//访问顶点v
   for(Edge*p=Head[v].adjacent; p; p=p->link)
        if(visited[p->VerAdj]==0) DFS(Head, p->VerAdj, visited);
   visited[v]=2;
}
```

# 判断有向图中是否有环



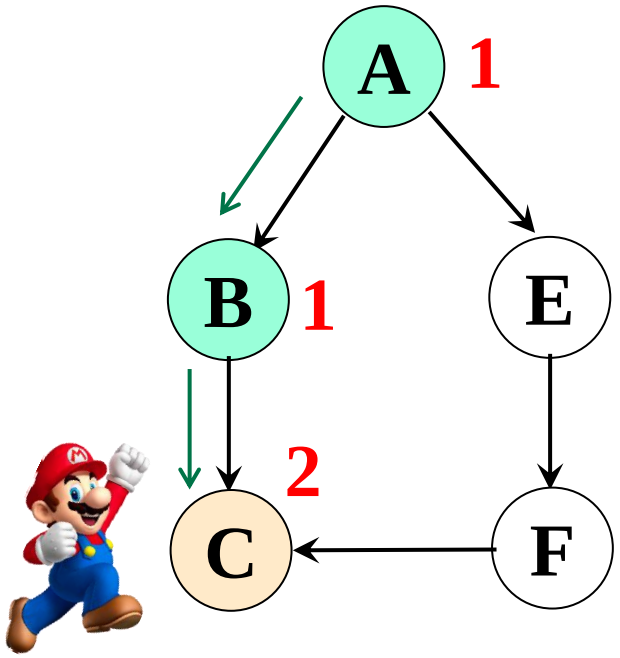

```
void DFS(Vertex* Head, int v, int visited[]){
    visit(v); visited[v]=1;//访问顶点v
    for(Edge*p=Head[v].adjacent; p; p=p->link)
         if(visited[p->VerAdj]==0) DFS(Head, p->VerAdj, visited);
    visited[v]=2;
}
```

# 判断有向图中是否有环



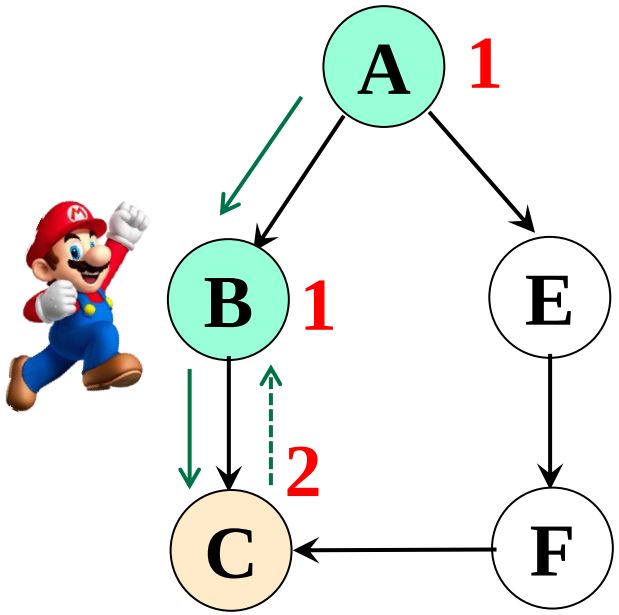

```
void DFS(Vertex* Head, int v, int visited[]){
    visit(v); visited[v]=1;//访问顶点v
    for(Edge*p=Head[v].adjacent; p; p=p->link)
         if(visited[p->VerAdj]==0) DFS(Head, p->VerAdj, visited);
    visited[v]=2;
}
```

# 判断有向图中是否有环



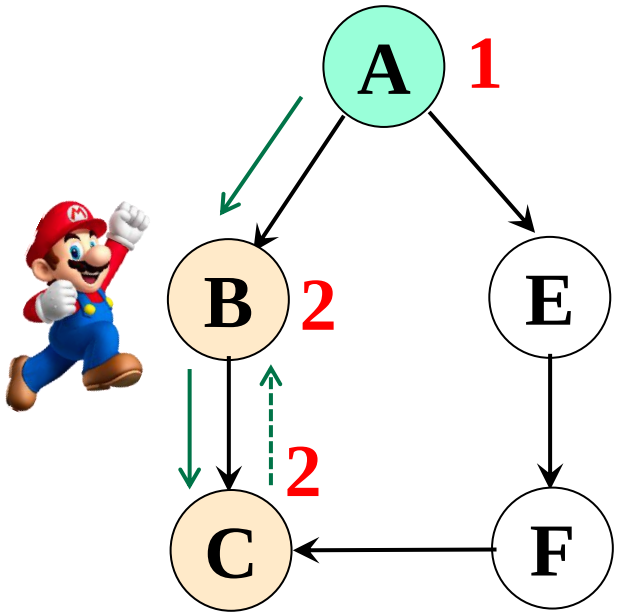

```
void DFS(Vertex* Head, int v, int visited[]){
    visit(v); visited[v]=1;//访问顶点v
    for(Edge*p=Head[v].adjacent; p; p=p->link)
         if(visited[p->VerAdj]==0) DFS(Head, p->VerAdj, visited);
    visited[v]=2;
}
```

# 判断有向图中是否有环



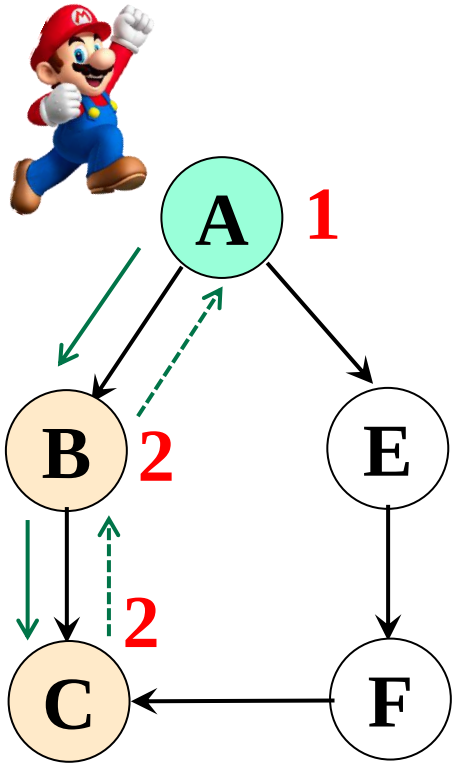

```
void DFS(Vertex* Head, int v, int visited[]){
    visit(v); visited[v]=1;//访问顶点v
    for(Edge*p=Head[v].adjacent; p; p=p->link)
         if(visited[p->VerAdj]==0) DFS(Head, p->VerAdj, visited);
    visited[v]=2;
}
```

# 判断有向图中是否有环



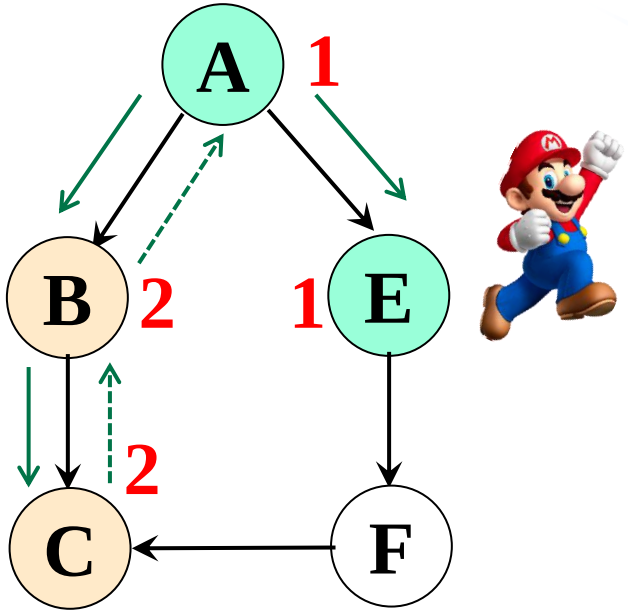

```
void DFS(Vertex* Head, int v, int visited[]){
    visit(v); visited[v]=1;//访问顶点v
    for(Edge*p=Head[v].adjacent; p; p=p->link)
         if(visited[p->VerAdj]==0) DFS(Head, p->VerAdj, visited);
    visited[v]=2;
}
```

# 判断有向图中是否有环




```
void DFS(Vertex* Head, int v, int visited[]){
    visit(v); visited[v]=1;//访问顶点v
    for(Edge*p=Head[v].adjacent; p; p=p->link)
         if(visited[p->VerAdj]==0) DFS(Head, p->VerAdj, visited);
    visited[v]=2;
}
```

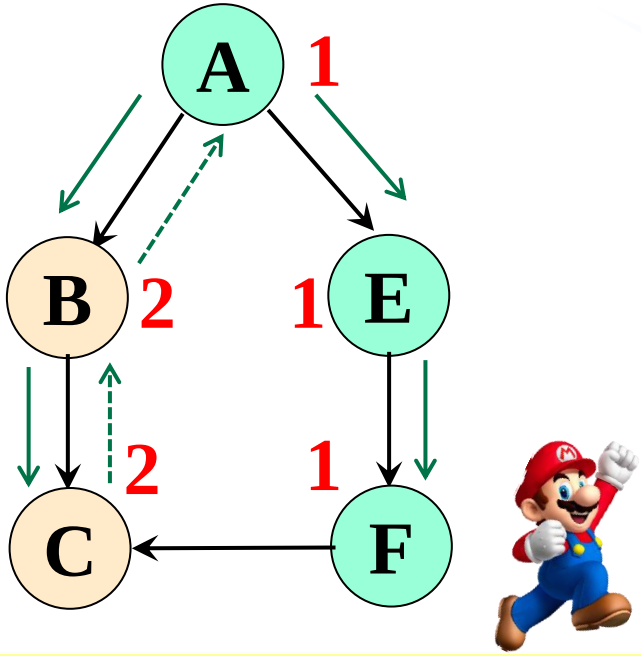
# 判断有向图中是否有环




```
void DFS(Vertex* Head, int v, int visited[]){
    visit(v); visited[v]=1;//访问顶点v
    for(Edge*p=Head[v].adjacent; p; p=p->link)
         if(visited[p->VerAdj]==0) DFS(Head, p->VerAdj, visited);
    visited[v]=2;
}
```

# 判断有向图中是否有环

B






有环：DFS过程中遇到已访问过的且visited值为1的点

在DFS过程中：

✓ 当前正在探索的路线（路径）上的点，visited值是1;

✓ 已经探索完的路线（路径）上的点，visited值是2;

✓ 还没探索的路线（路径）上的点，visited值是0.

}

# 判断有向图中是否有环

B

```
bool HasCircle(Vertex* Head, int v, int visited[]){
    visited[v]=1;   //访问顶点v
    for(Edge*p=Head[v].adjacent; p!=NULL; p=p->link){
        int k=p->VerAdj;
        if(visited[k]==0)
            if(HasCircle(Head,k,visited)==true) return true;
        if(visited[k]==1) return true;
    }
    visited[v]=2;
    return false;
}
```