

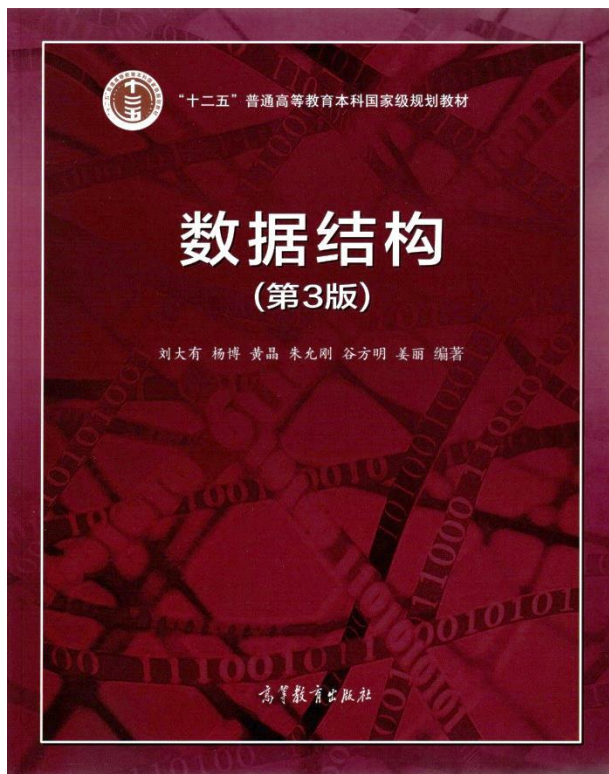


计算机学院王湘浩班
2024级



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





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

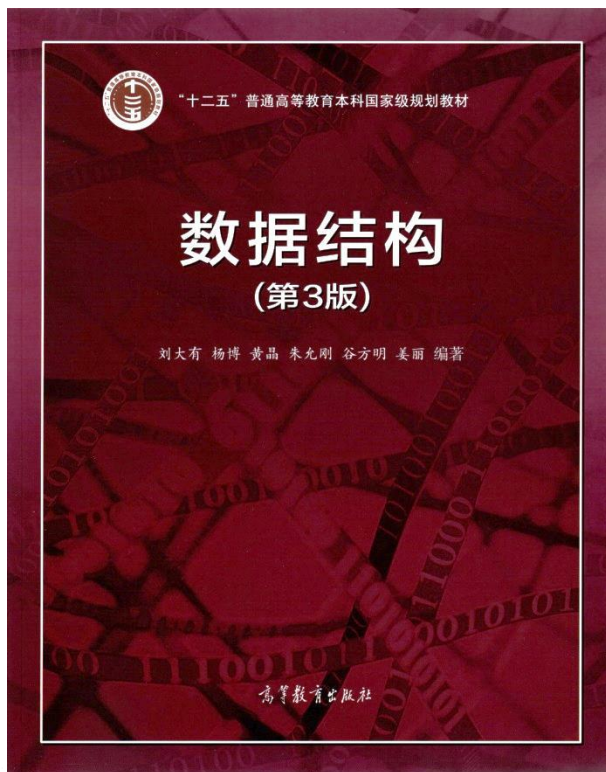
有时候程序写完了，没有输出预定的结果，一遍遍地找，还是发现不了错在哪，很崩溃。最痛苦也是最快乐的一次，是找了两个晚上终于把bug找出来.....

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

- 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

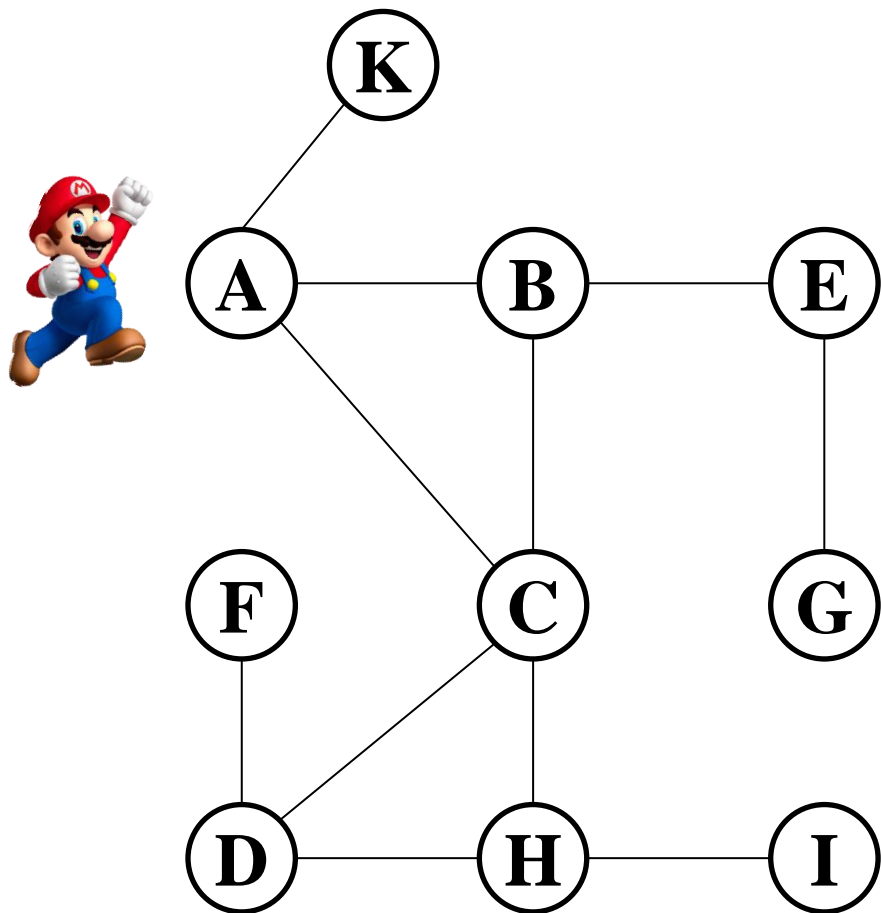
基本思想:

- *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

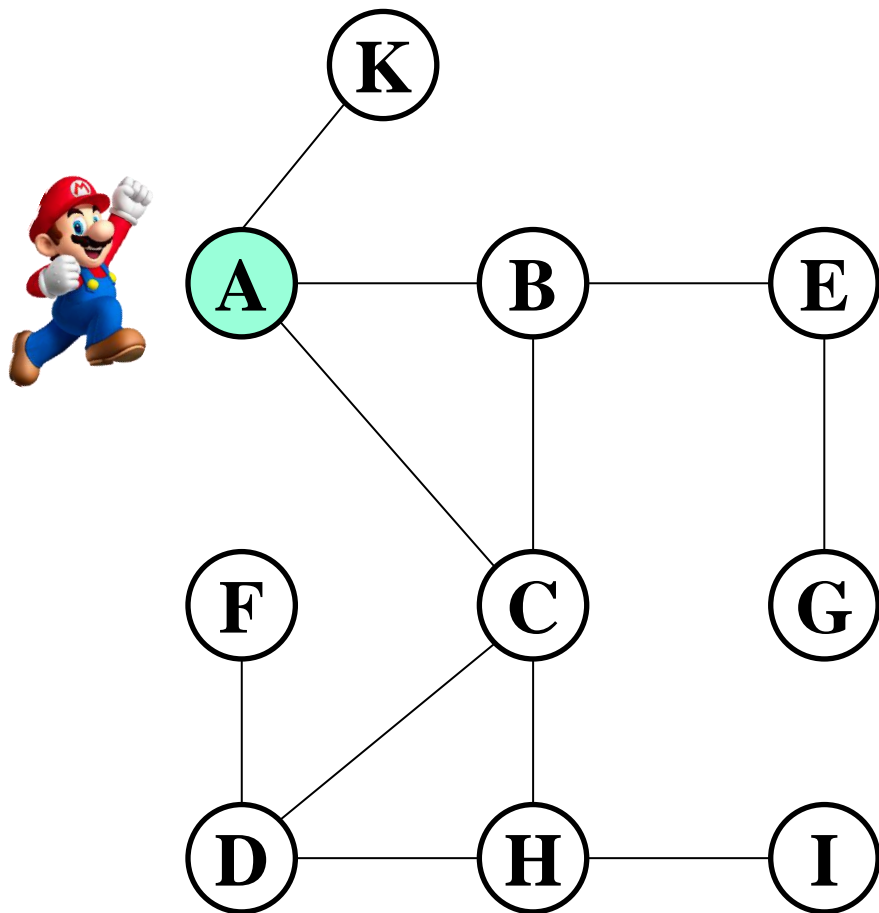
深度优先搜索 DFS (Depth First Search, DFS)

A



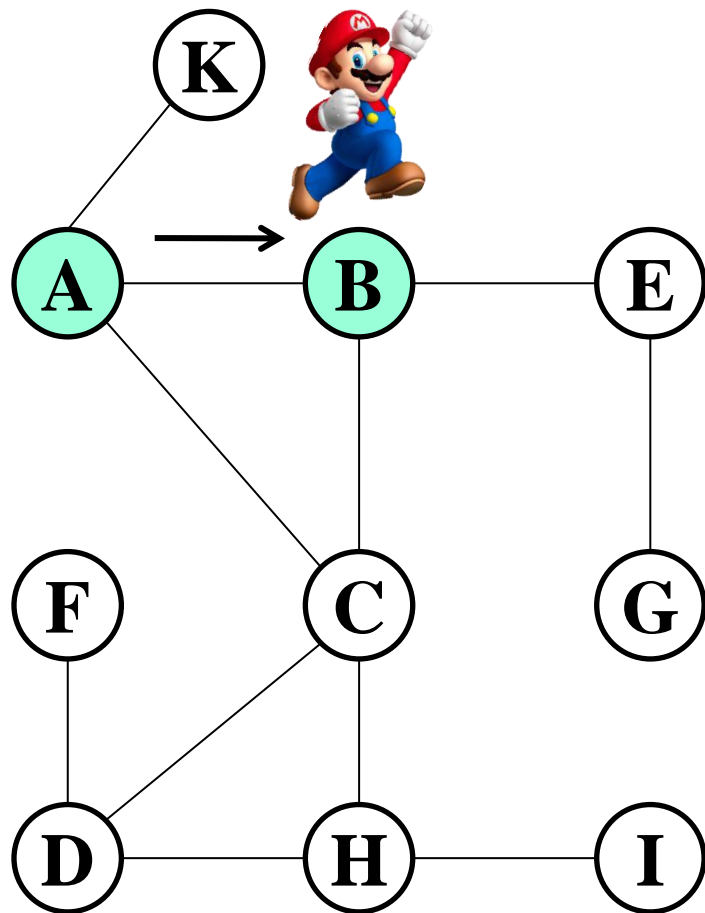
深度优先搜索 DFS (Depth First Search, DFS)

A



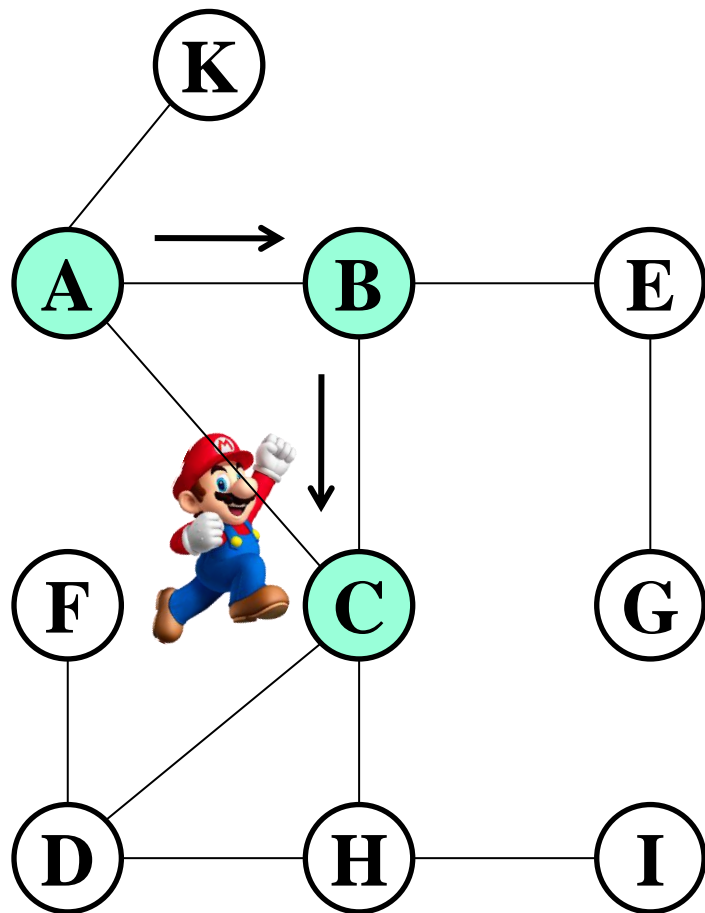
深度优先搜索 DFS (Depth First Search, DFS)

A



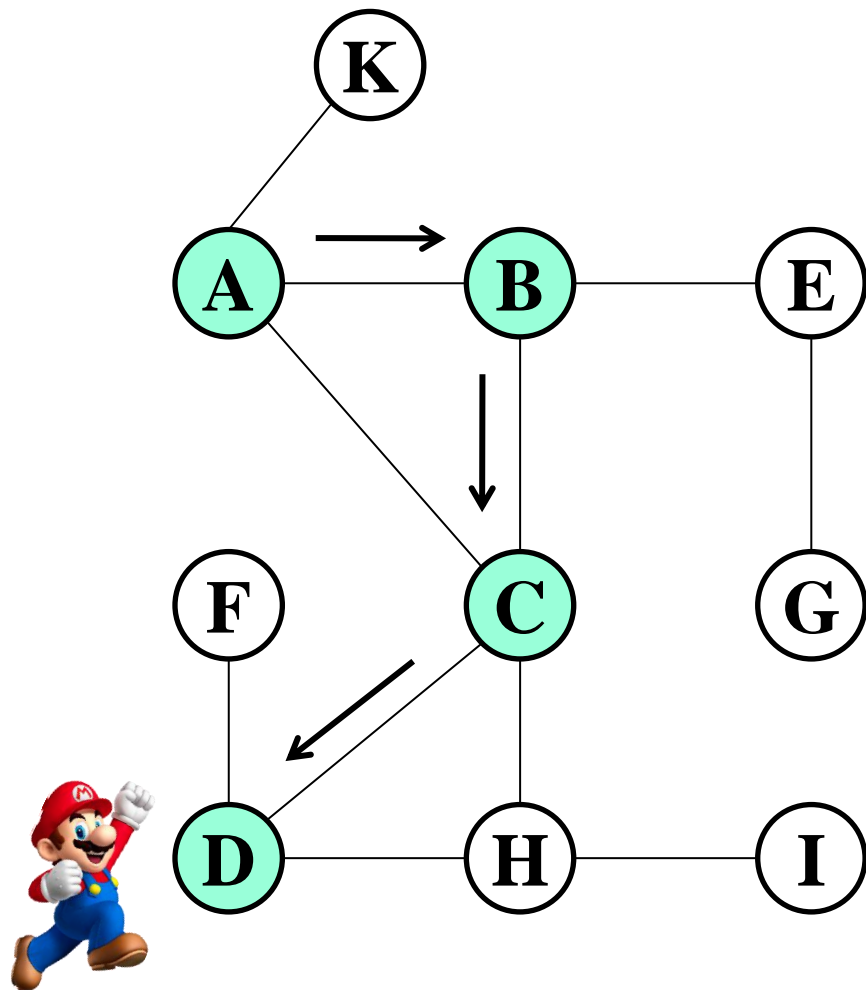
深度优先搜索 DFS (Depth First Search, DFS)

A



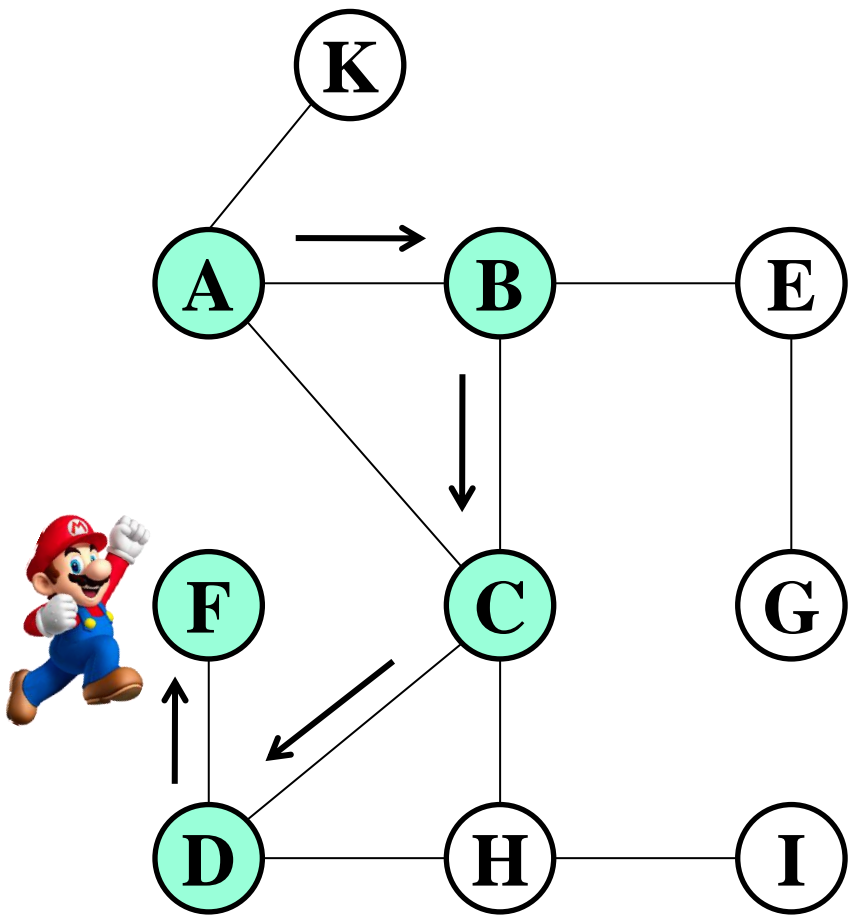
深度优先搜索 DFS (Depth First Search, DFS)

A



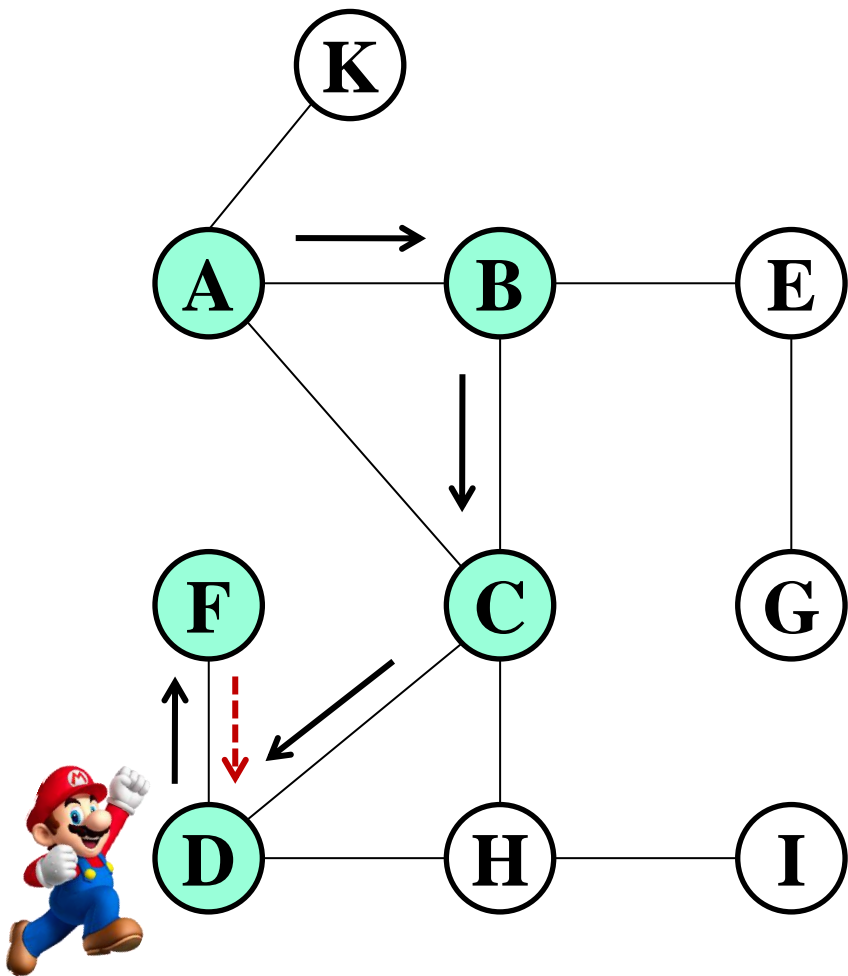
深度优先搜索 DFS (Depth First Search, DFS)

A



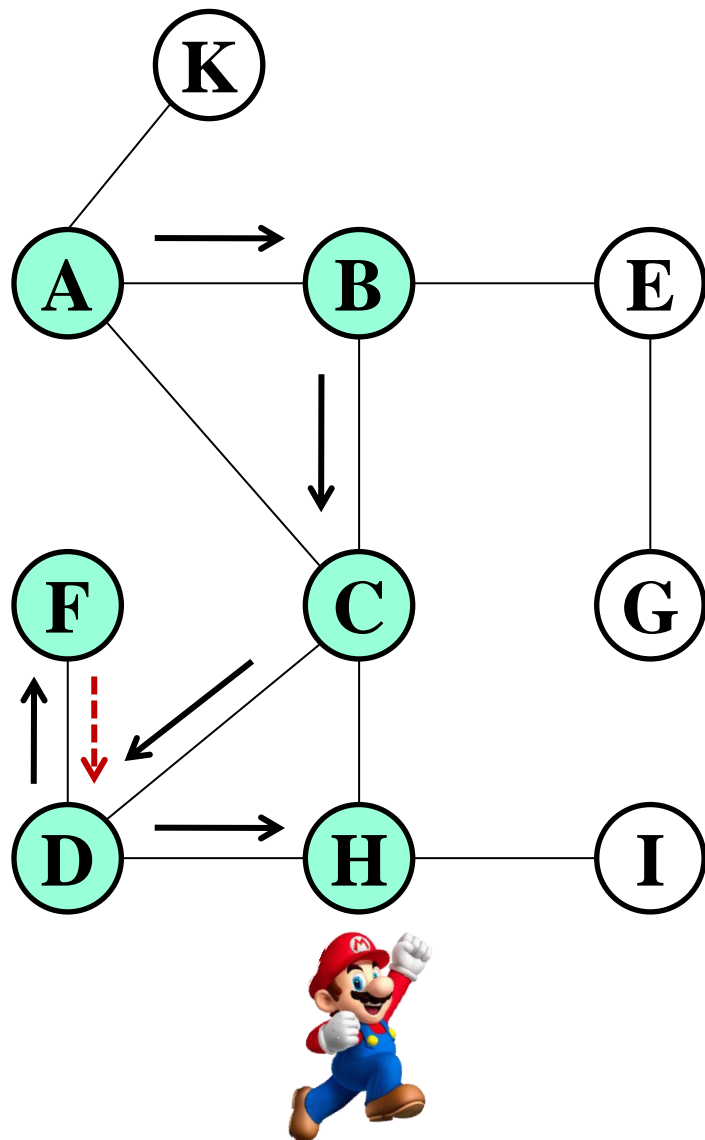
深度优先搜索 DFS (Depth First Search, DFS)

A



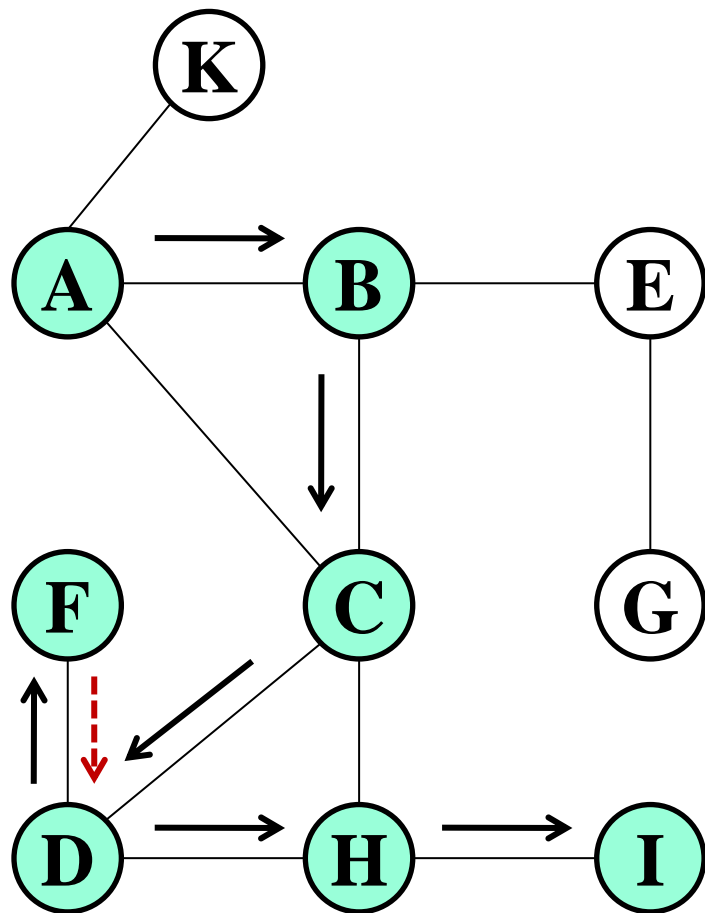
深度优先搜索 DFS (Depth First Search, DFS)

A



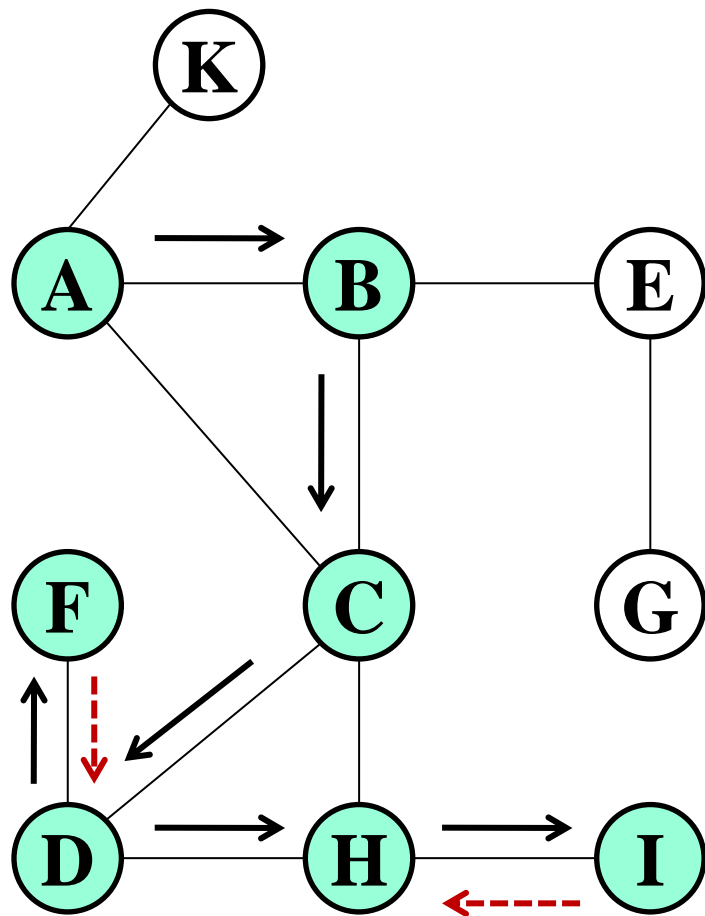
深度优先搜索 DFS (Depth First Search, DFS)

A



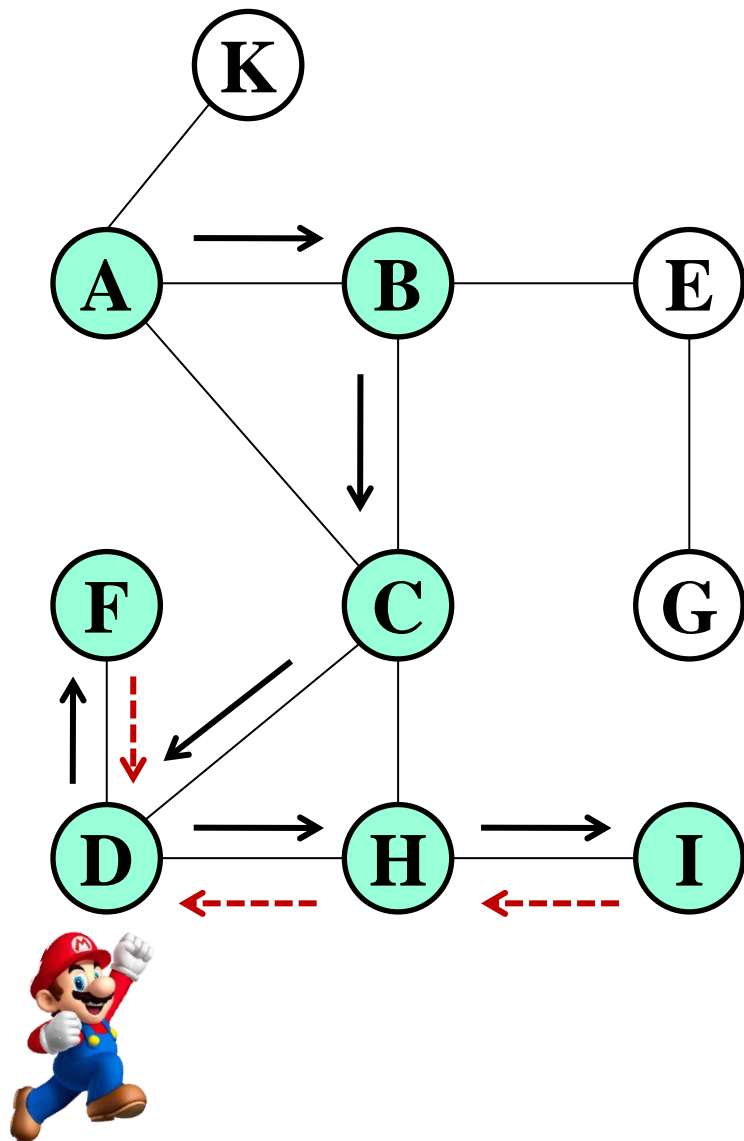
深度优先搜索 DFS (Depth First Search, DFS)

A



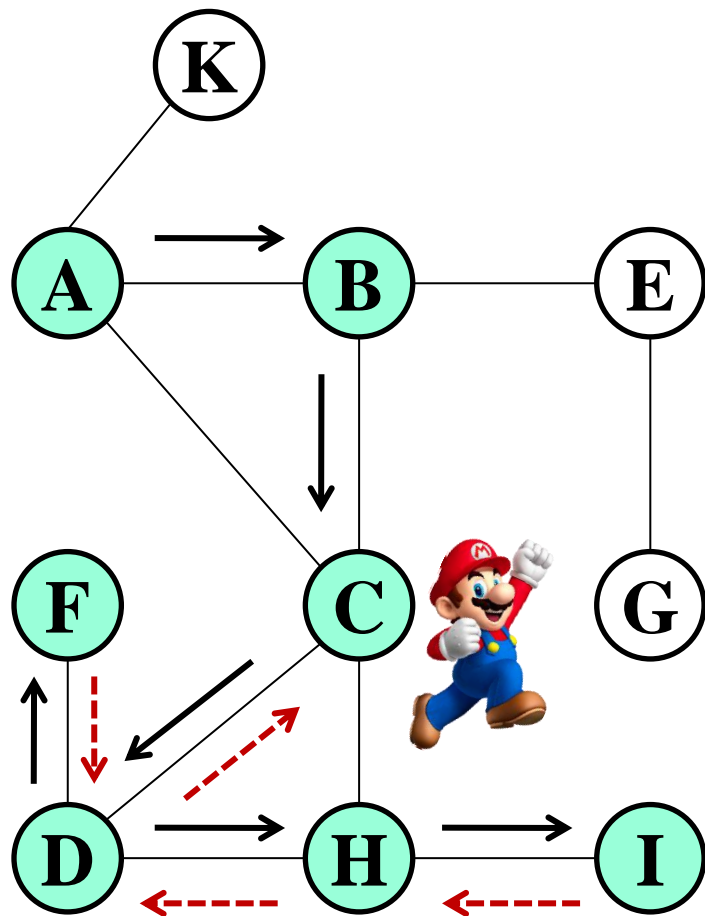
深度优先搜索 DFS (Depth First Search, DFS)

A



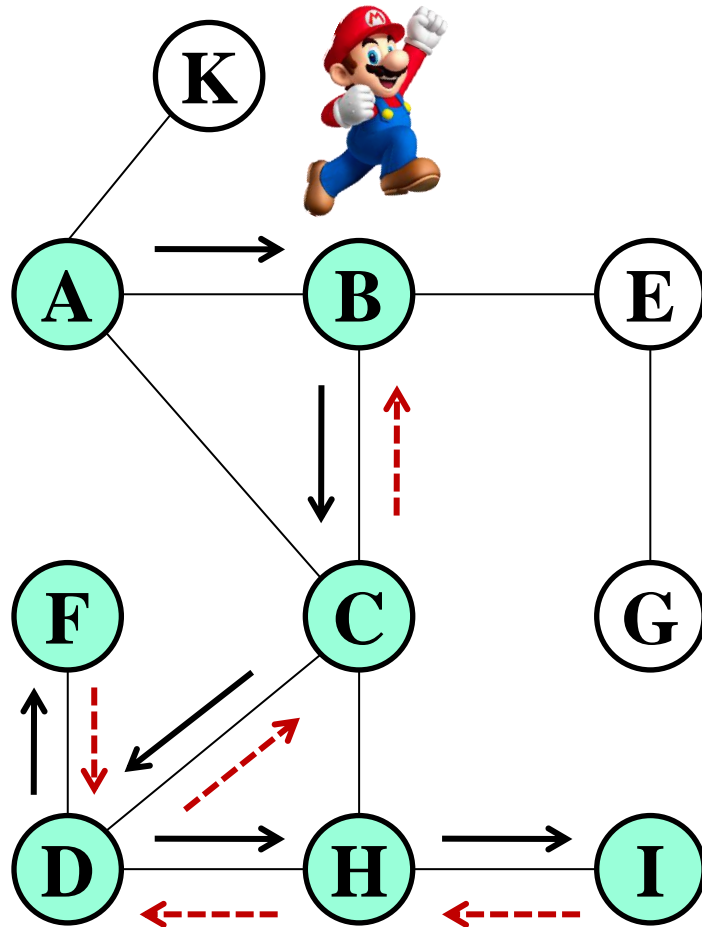
深度优先搜索 DFS (Depth First Search, DFS)

A



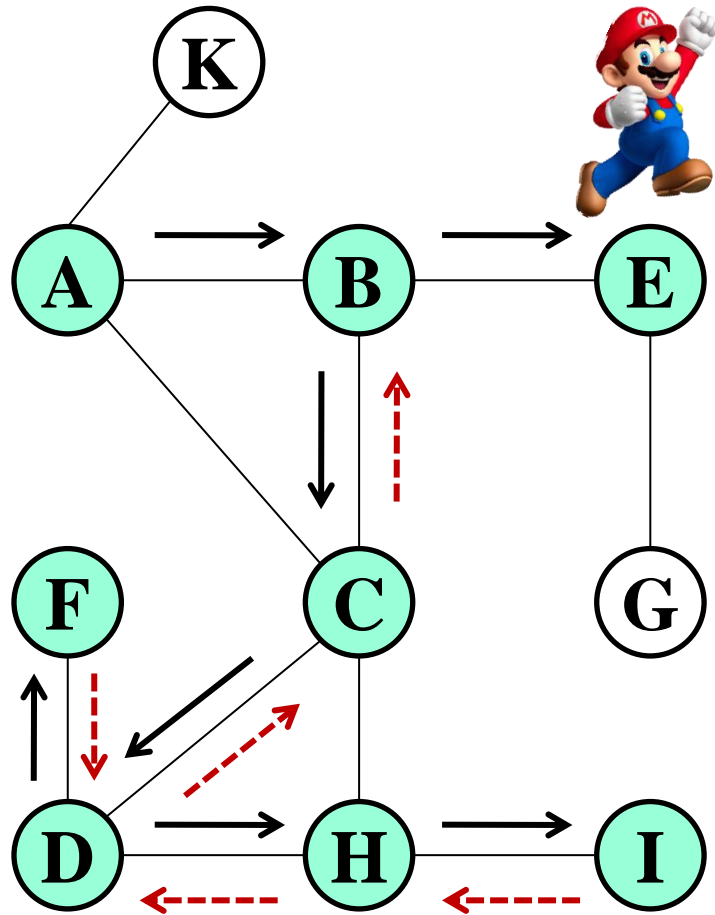
深度优先搜索 DFS (Depth First Search, DFS)

A



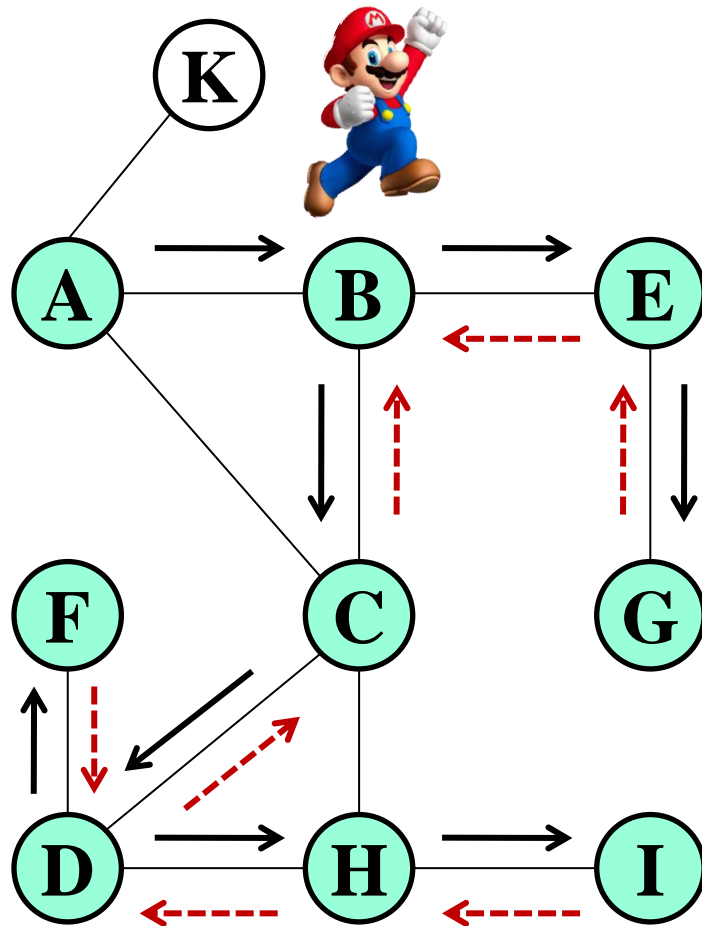
深度优先搜索 DFS (Depth First Search, DFS)

A



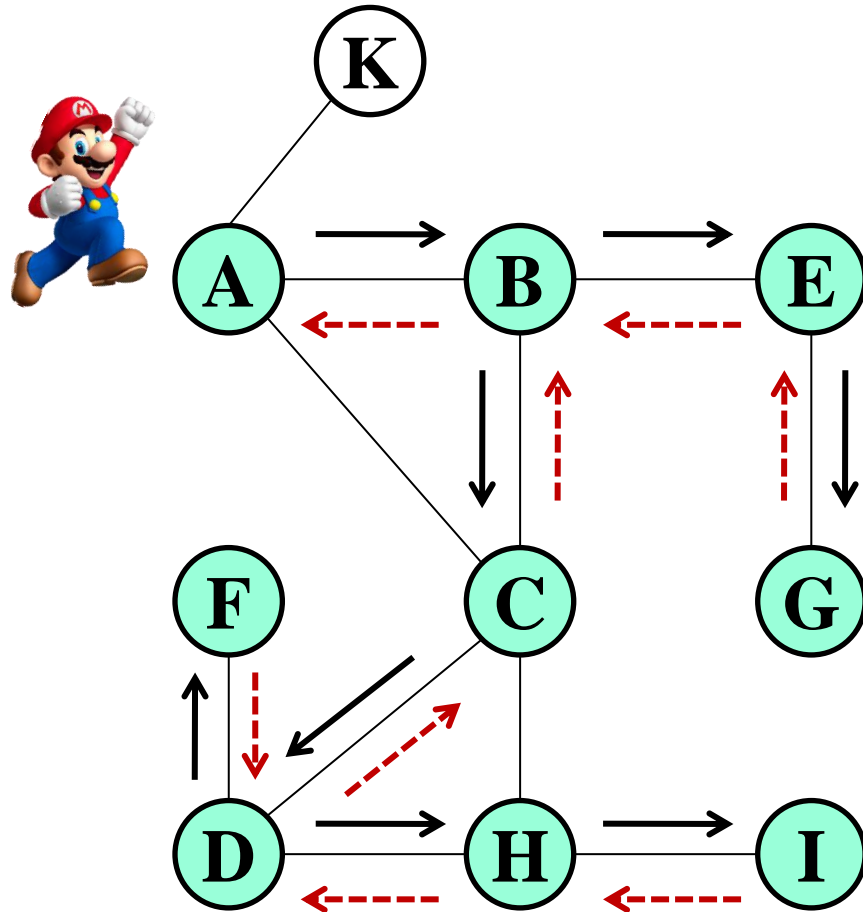
深度优先搜索 DFS (Depth First Search, DFS)

A



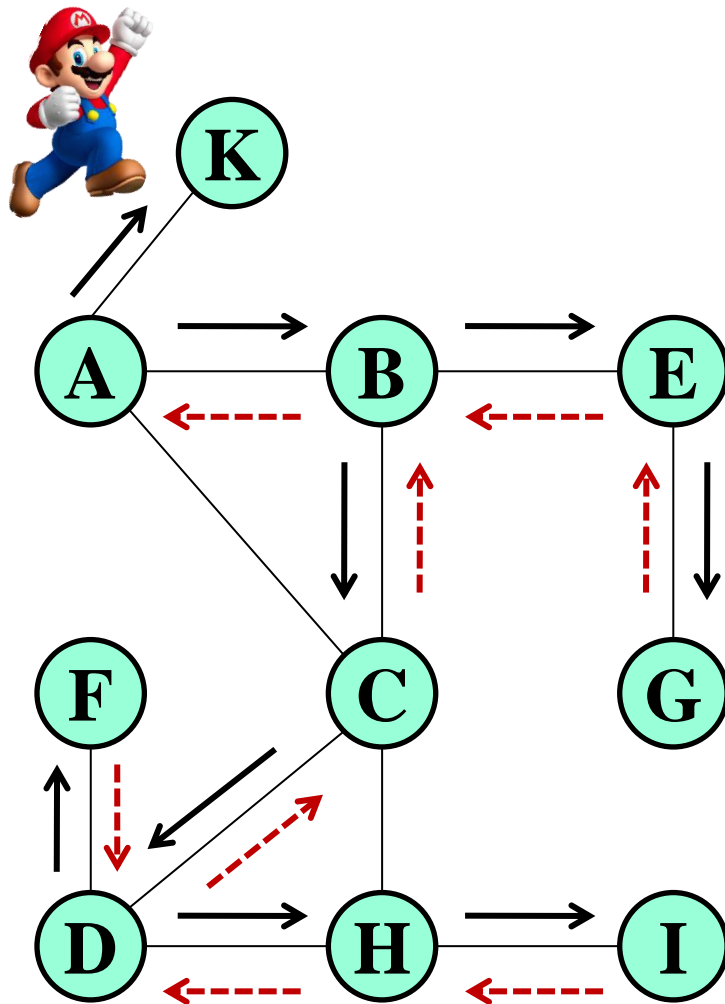
深度优先搜索 DFS (Depth First Search, DFS)

A



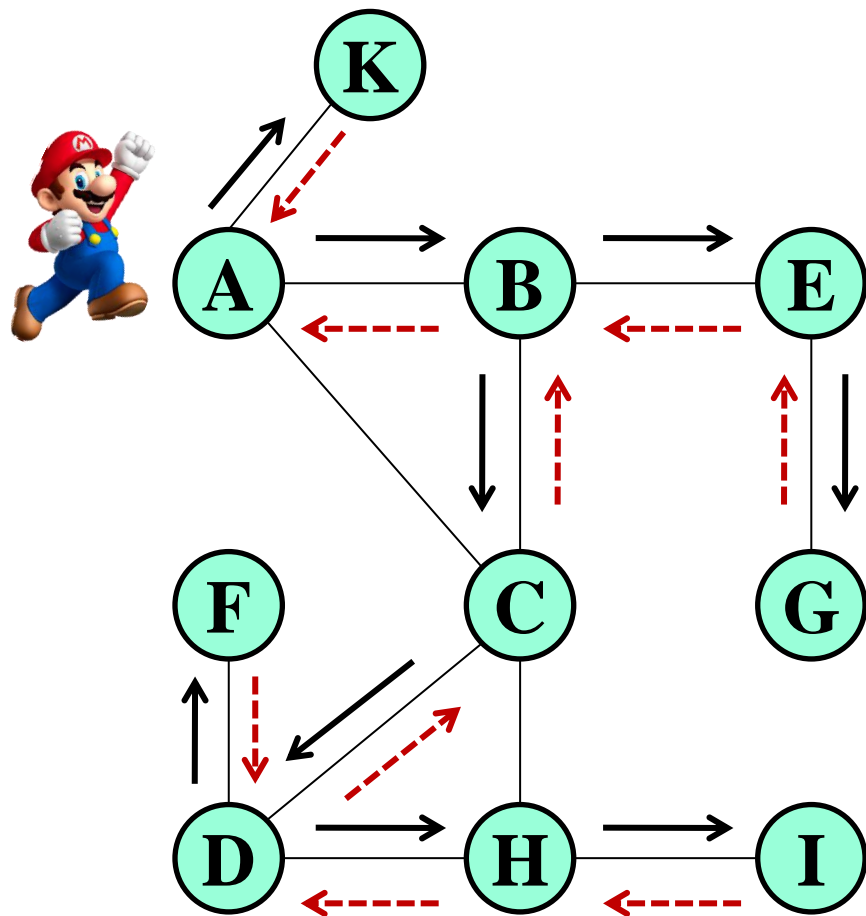
深度优先搜索 DFS (Depth First Search, DFS)

A



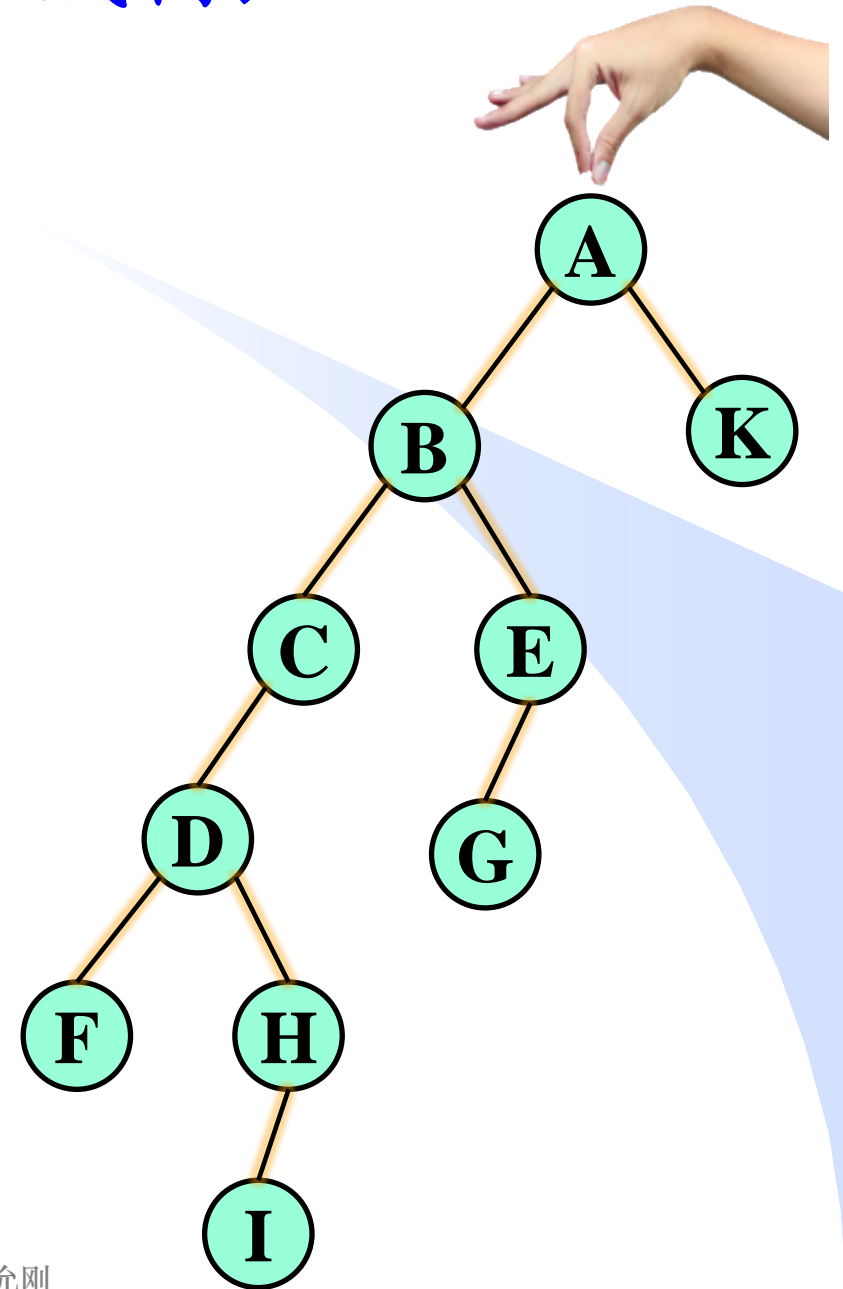
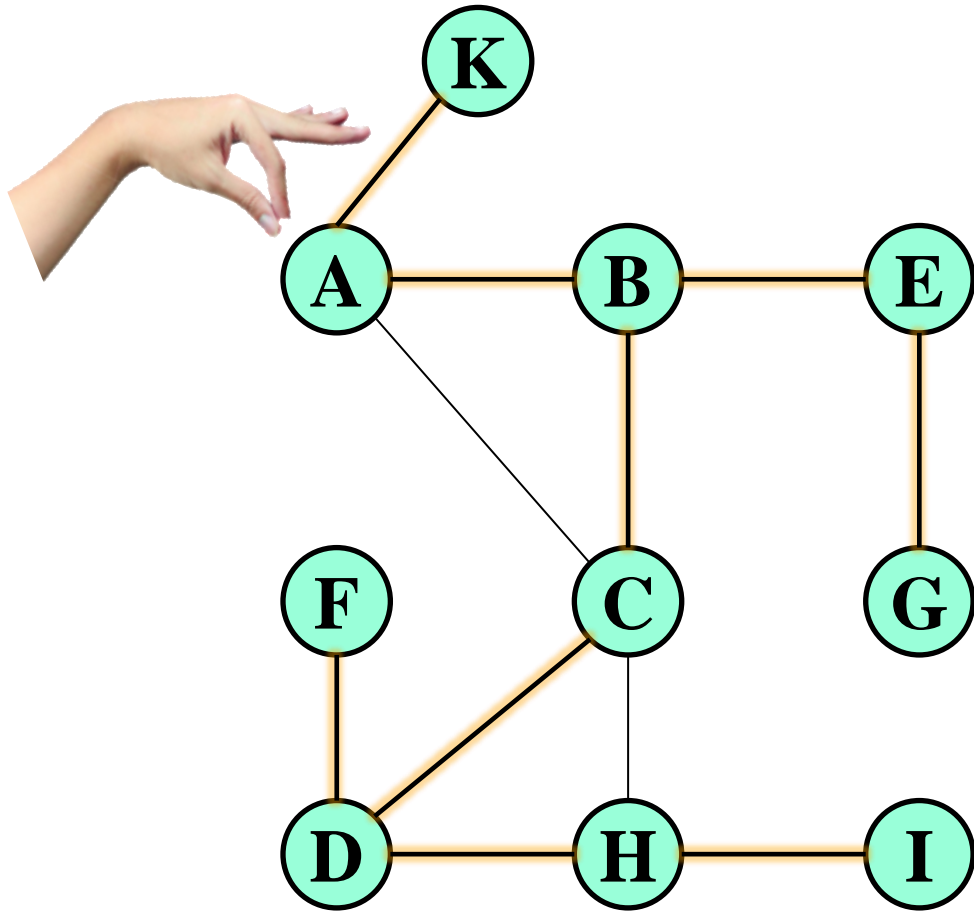
深度优先搜索 DFS (Depth First Search, DFS)

A



深度优先支撑树（生成树）

B

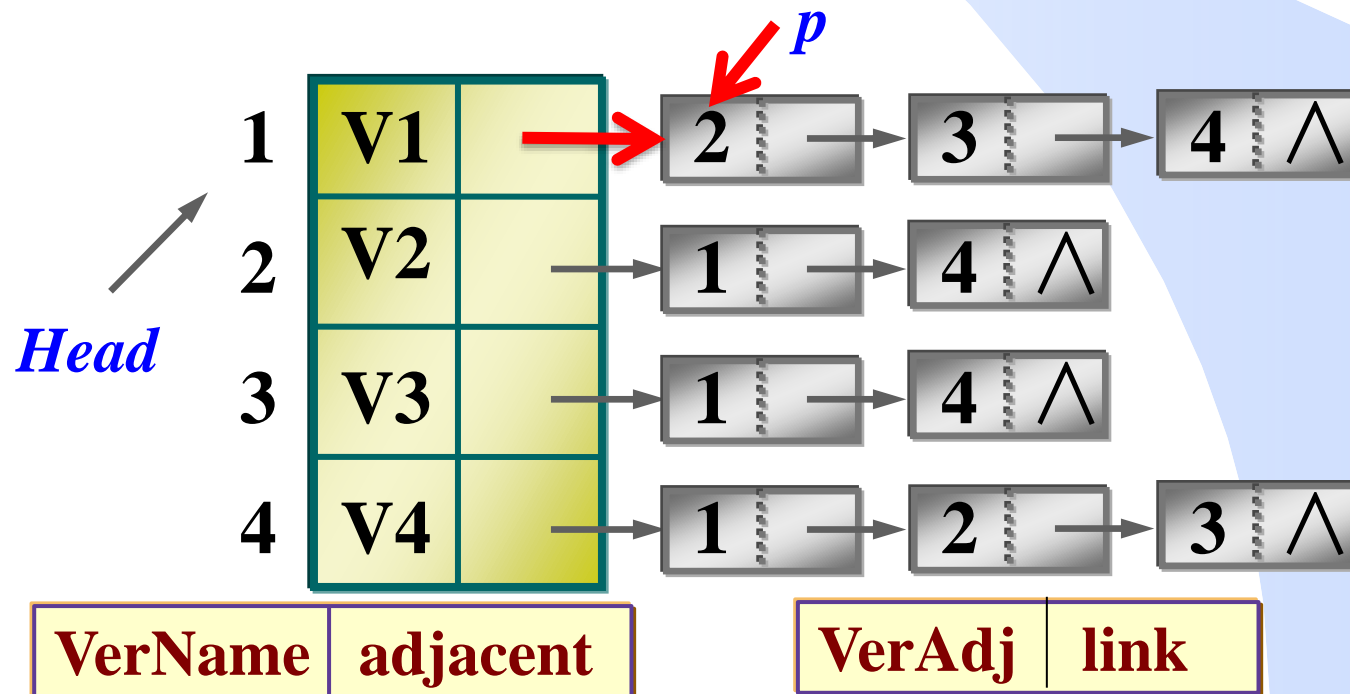
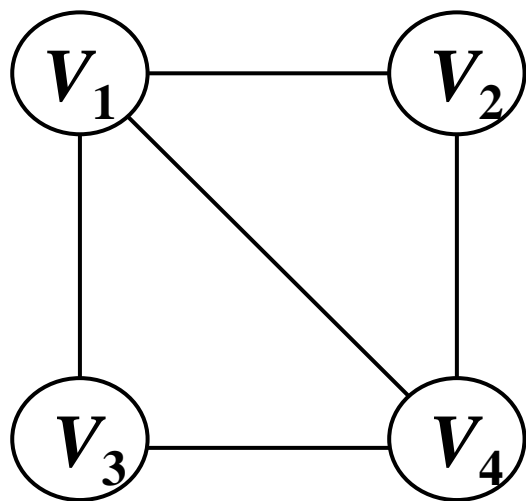


图的深度优先搜索算法

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

```

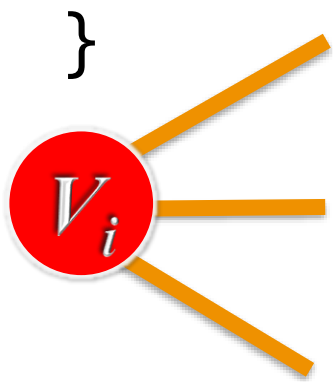
graph LR
    1[1 V1] --> 2[2]
    2 --> 3[3]
    3 --> 4[4]
    p((p)) --> 2
  
```



回溯
可行则进
不行则换
换不了则退

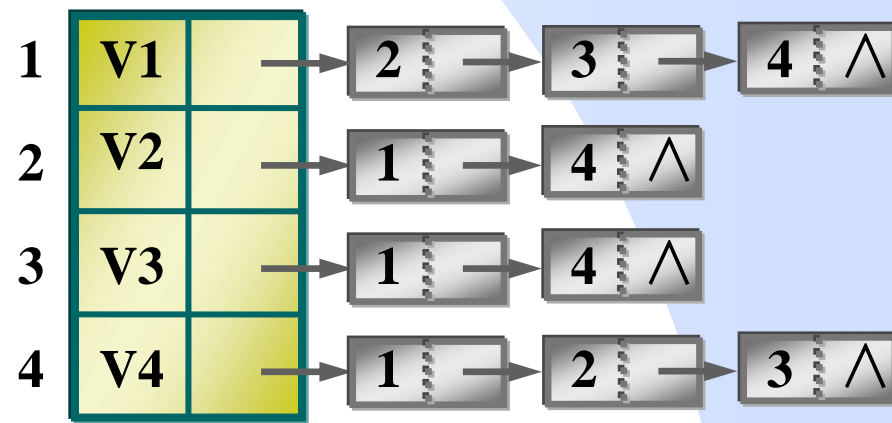
时间复杂度分析

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

```
    visited[i]=0;
```

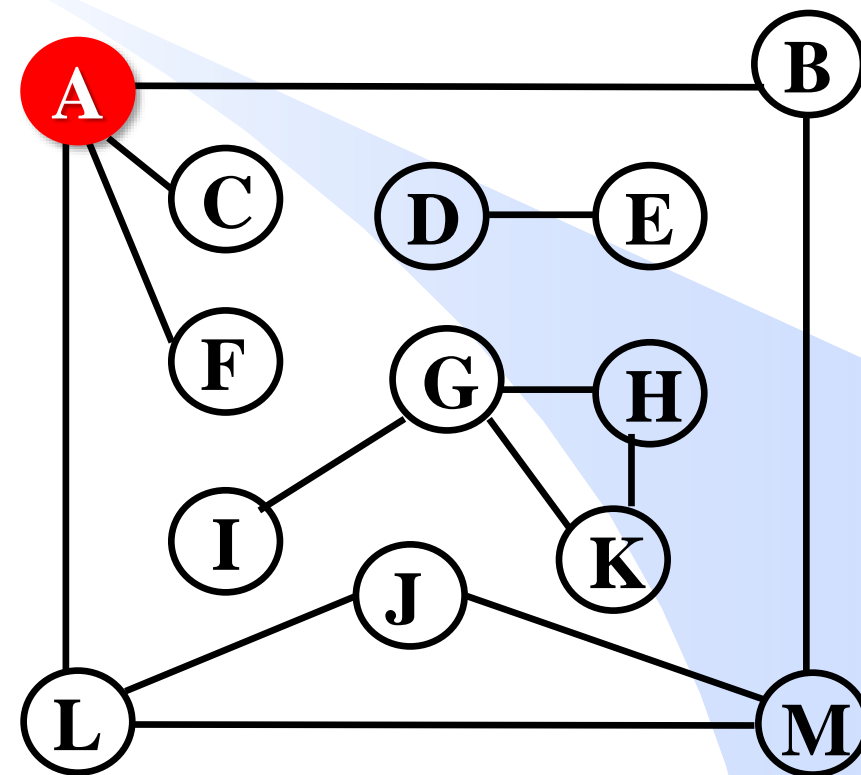
```
//以每个顶点为起点，试探是否能深搜
```

```
for(int i=0; i<n; i++)
```

```
    if(visited[i]==0)
```

```
        DFS(Head, i, visited);
```

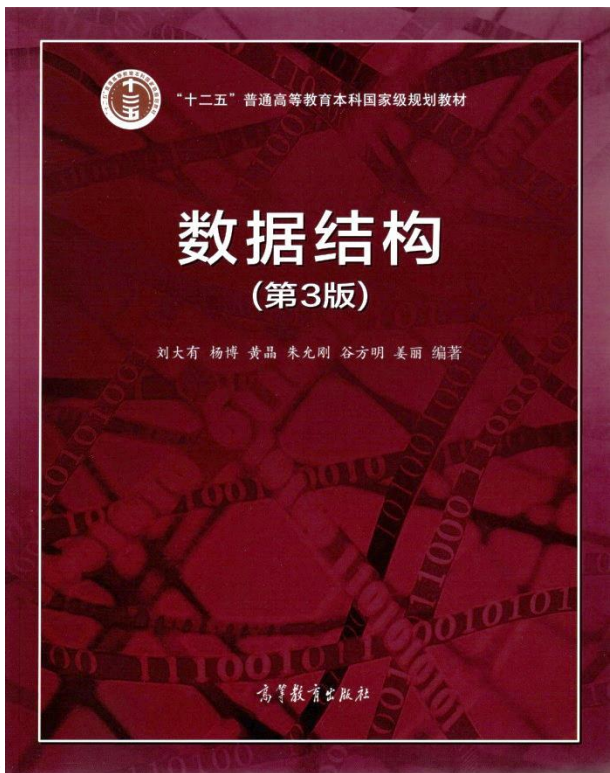
一次DFS只能遍历一个连通分量





图的遍历及应用

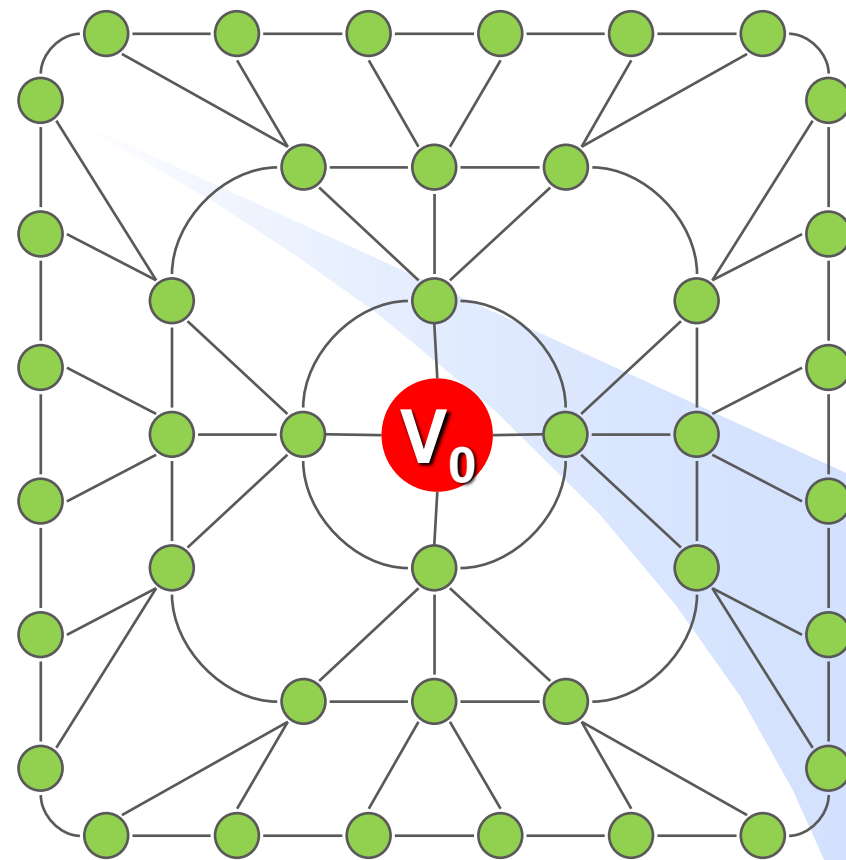
- 深度优先搜索
- **广度优先搜索**
- 图遍历的应用



数据之法
结构之美
算法之道

广度优先搜索 (Breadth First Search, *BFS*)

- ✓ 访问初始点顶点 v_0 ;
- ✓ 依次访问 v_0 的邻接顶点 $w_1...w_k$;
- ✓ 然后再依次访问与 $w_1...w_k$ 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。



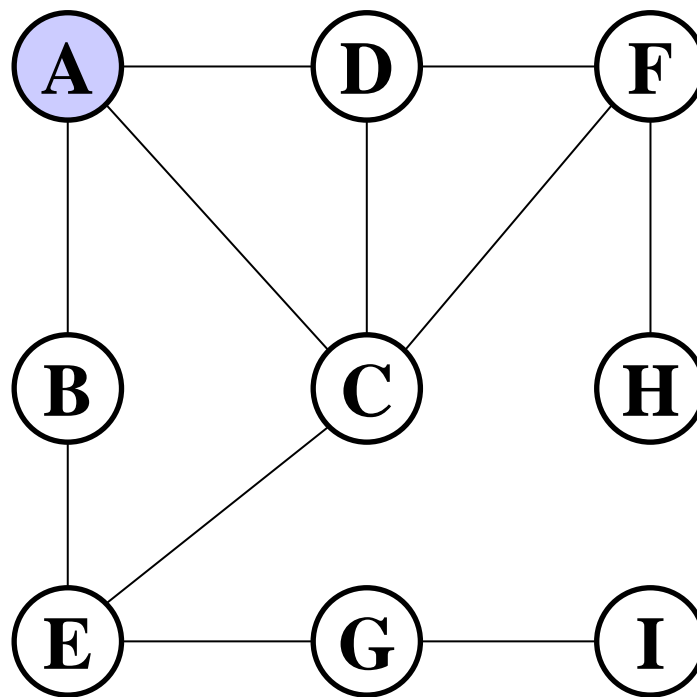
啊~五环, 你比四环多一环



广度优先搜索 (Breadth First Search, *BFS*)

A

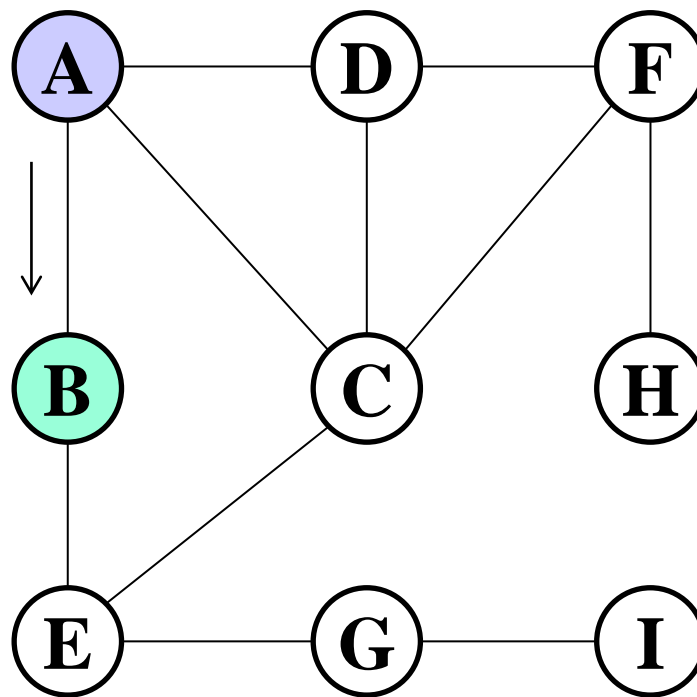
地毯式搜索
层层推进



广度优先搜索 (Breadth First Search, *BFS*)

A

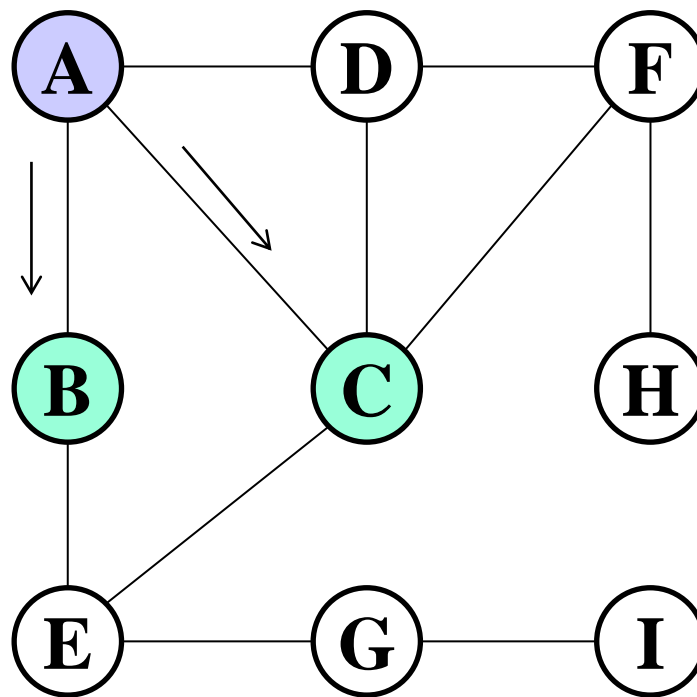
地毯式搜索
层层推进



广度优先搜索 (Breadth First Search, *BFS*)

A

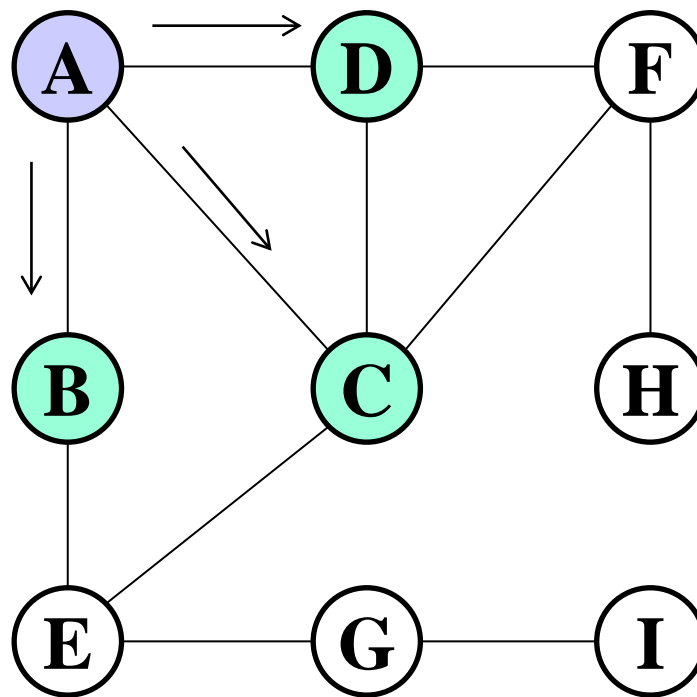
地毯式搜索
层层推进



广度优先搜索 (Breadth First Search, *BFS*)

A

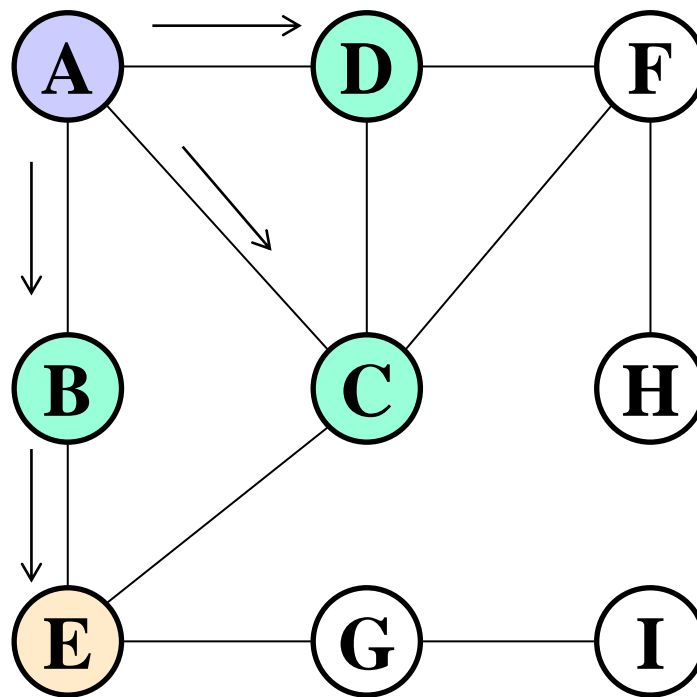
地毯式搜索
层层推进



广度优先搜索 (Breadth First Search, *BFS*)

A

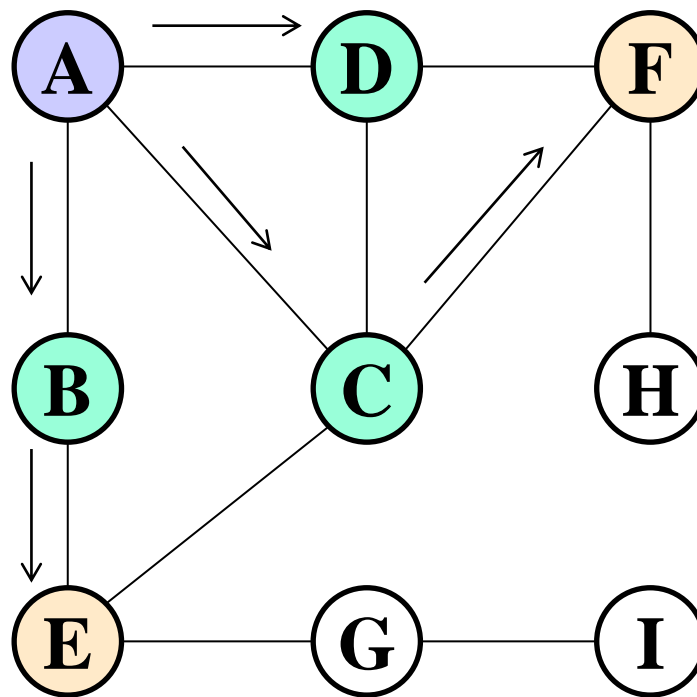
地毯式搜索
层层推进



广度优先搜索 (Breadth First Search, *BFS*)

A

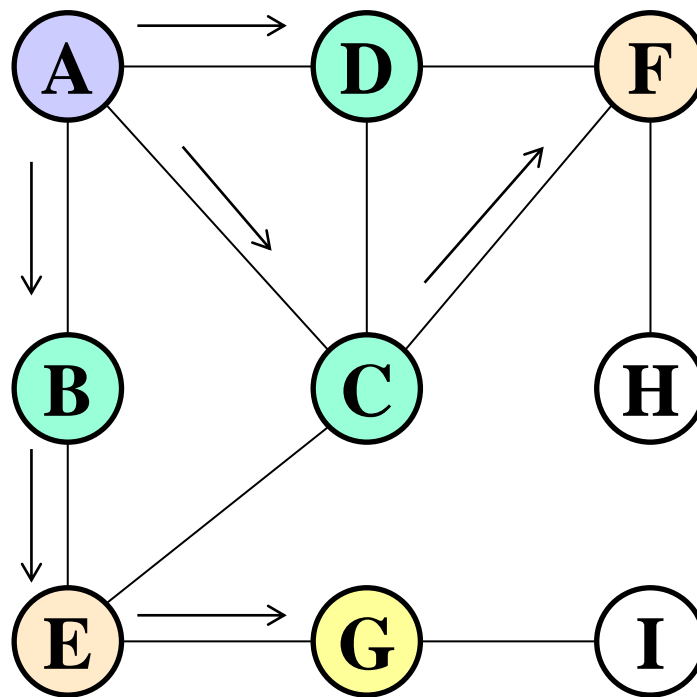
地毯式搜索
层层推进



广度优先搜索 (Breadth First Search, *BFS*)

A

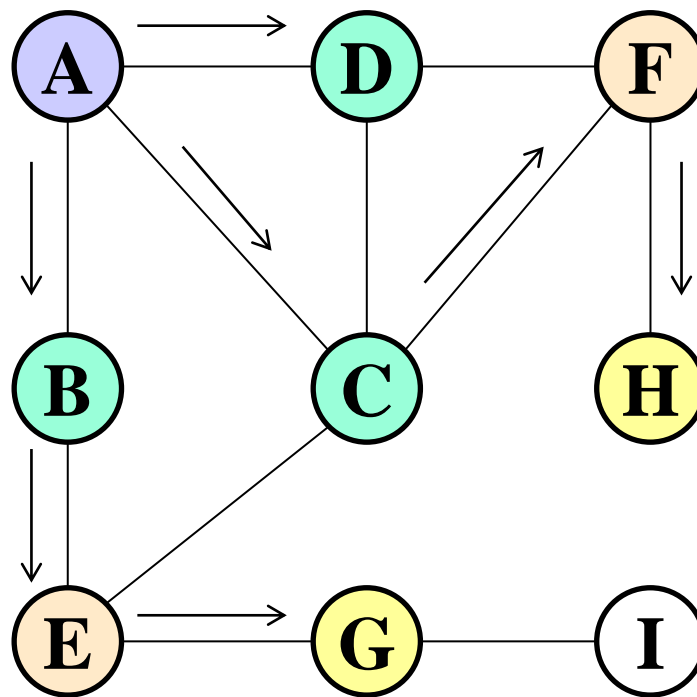
地毯式搜索
层层推进



广度优先搜索 (Breadth First Search, *BFS*)

A

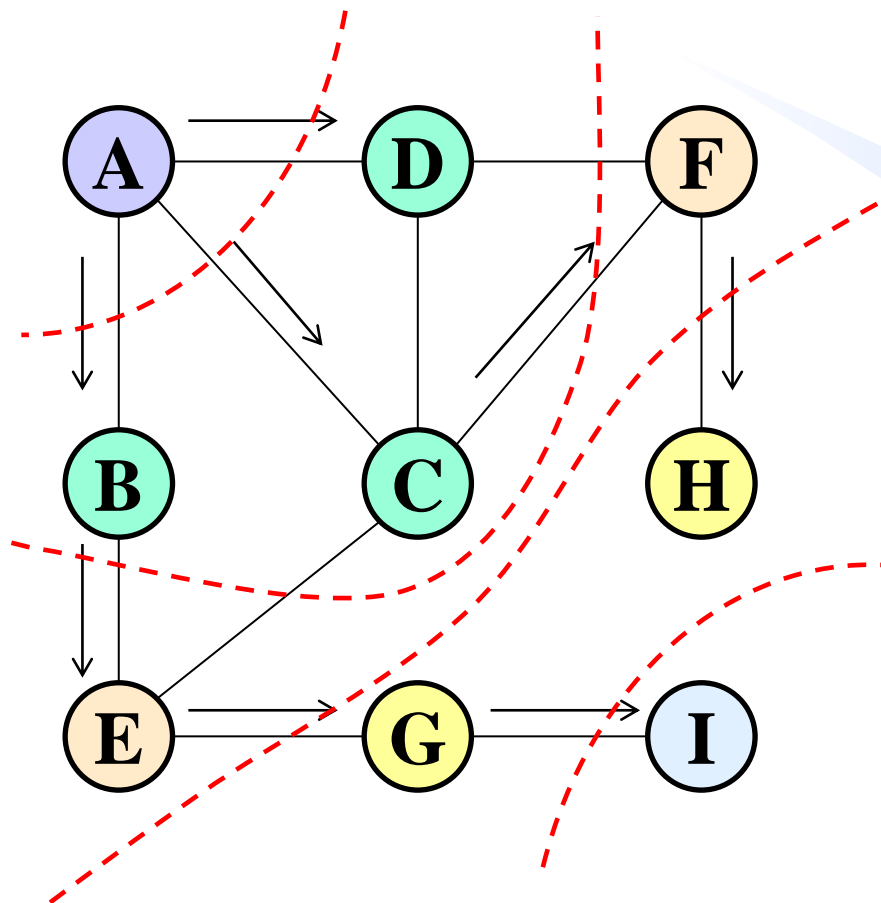
地毯式搜索
层层推进



广度优先搜索 (Breadth First Search, *BFS*)

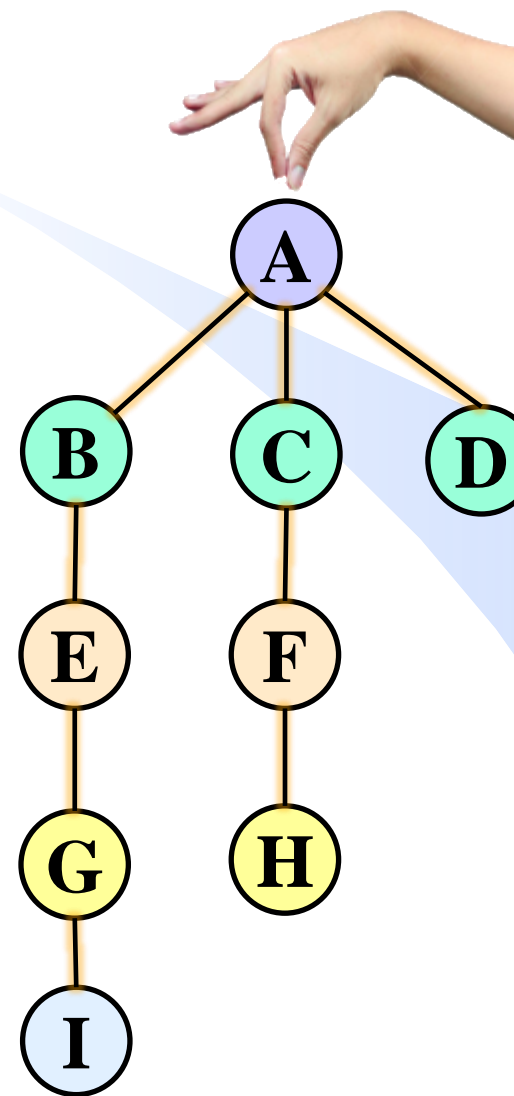
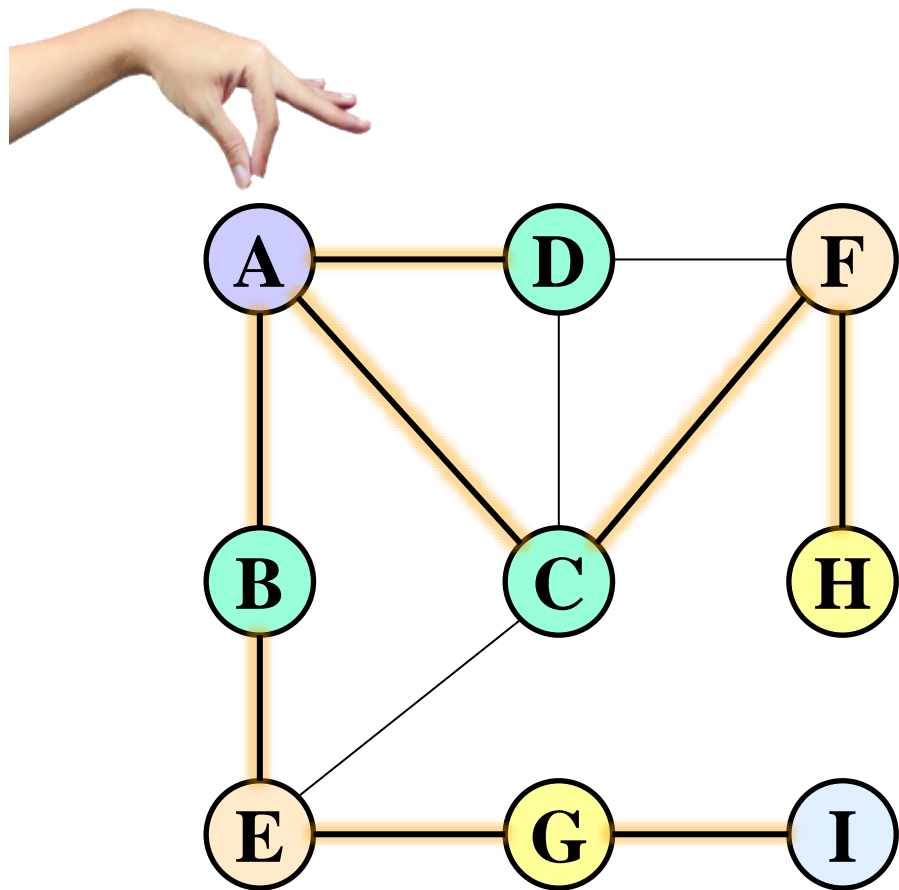
A

地毯式搜索
层层推进



广度优先支撑树（生成树）

B



广度优先搜索 (Breadth First Search, *BFS*)

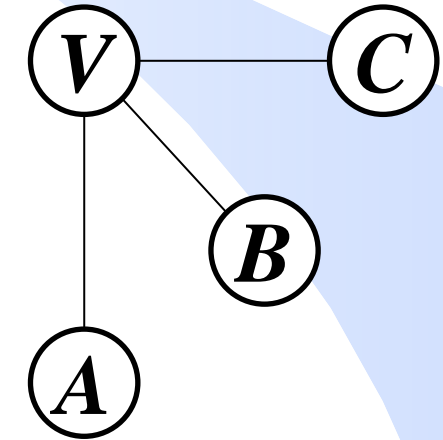
A

- 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

广度优先搜索 (Breadth First Search, *BFS*)

```
void BFS(Vertex* Head, int v, int n, int visited[]){
```

```
    Queue<int> 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
```

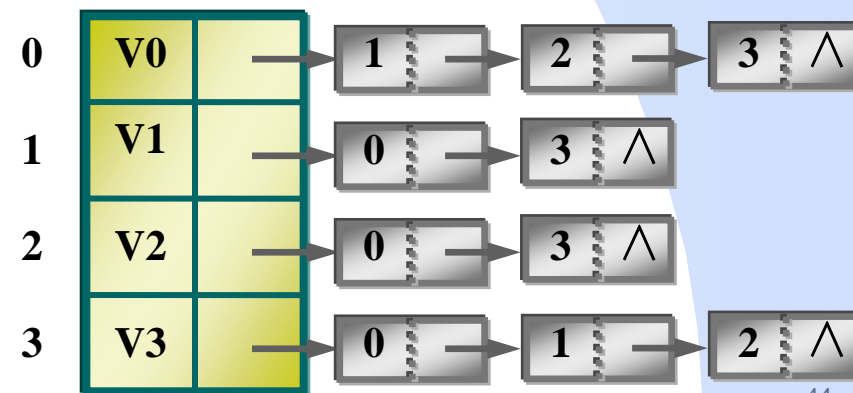
```
        } //end while
```

```
    } //end BFS
```

时间复杂度 $O(n+e)$

for循环：扫描每个顶点的边结点

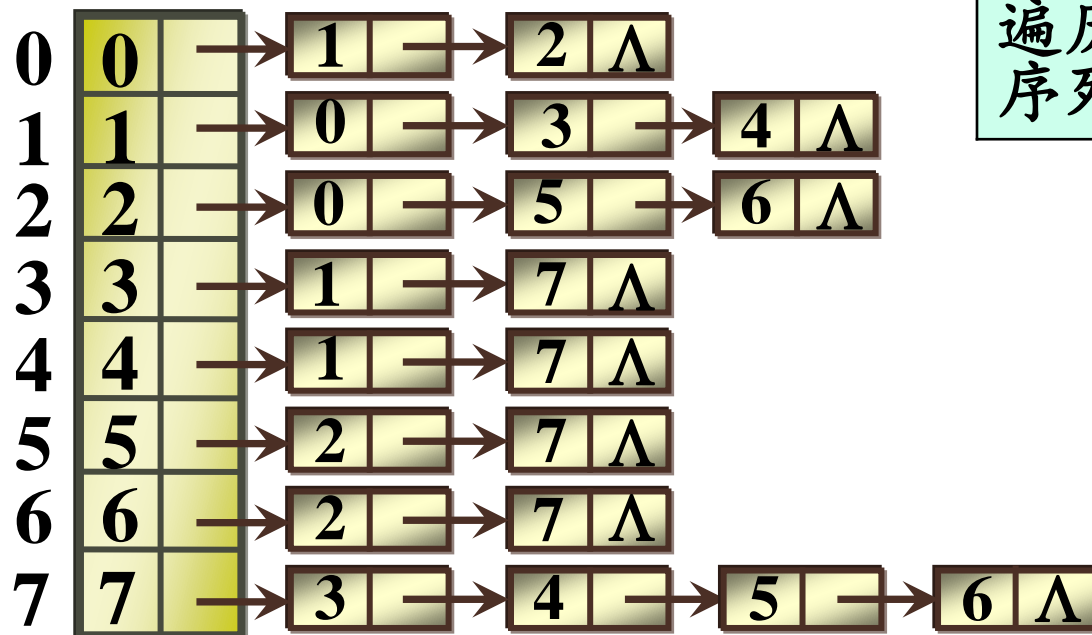
每个顶点被出队一次



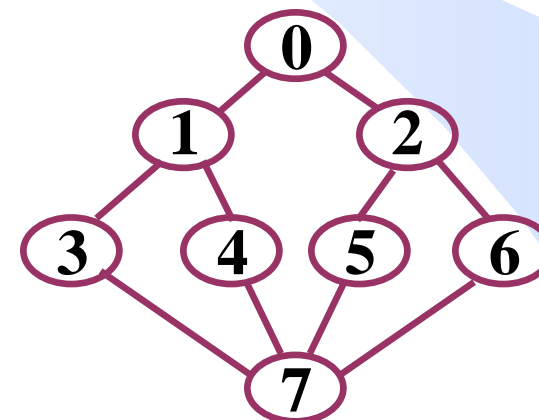
顶点入队时访问

A

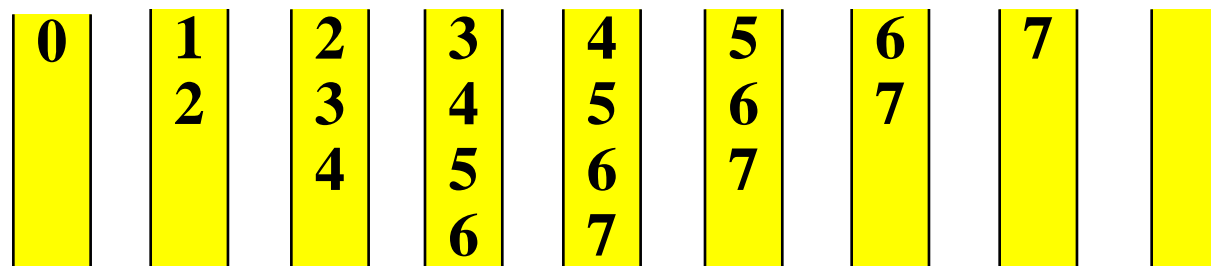
```
while(!Q.Empty()){
    v=Q.dequeue(); //出队一个点
    for(Edge* p=Head[v].adjacent; p!=NULL; p=p->link)
        if(vis[p->VerAdj]==0){ //考察v的邻接顶点p
            visit(p->VerAdj); vis[p->VerAdj]=1;Q.enqueue(p->VerAdj);
        }
}
```



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（版本2）

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

版本1和版本2
哪个效率更高?

出队时访问

广度优先搜索 —— 邻接矩阵存图

```
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);
```

```
            }
```

```
        }
```

```
    }
```

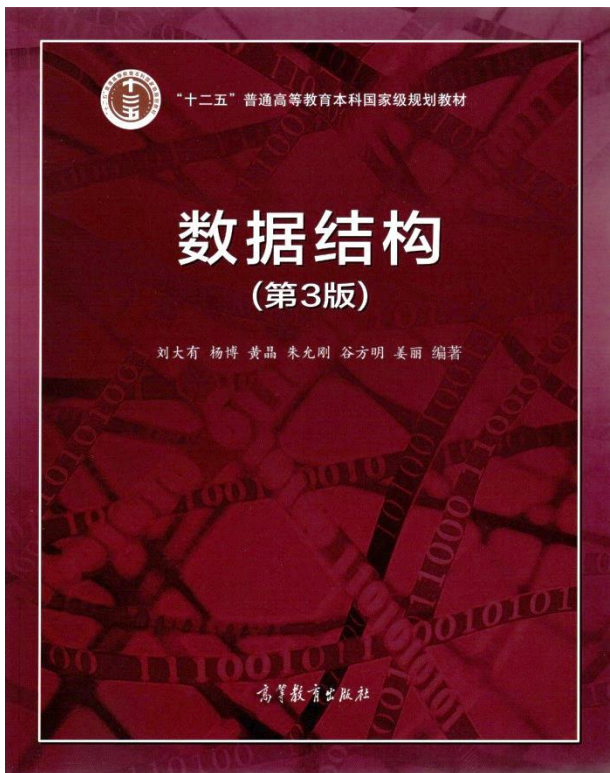
for循环：扫描顶点 i 的邻接顶点需遍历矩阵第 i 行，时间 $O(n)$

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

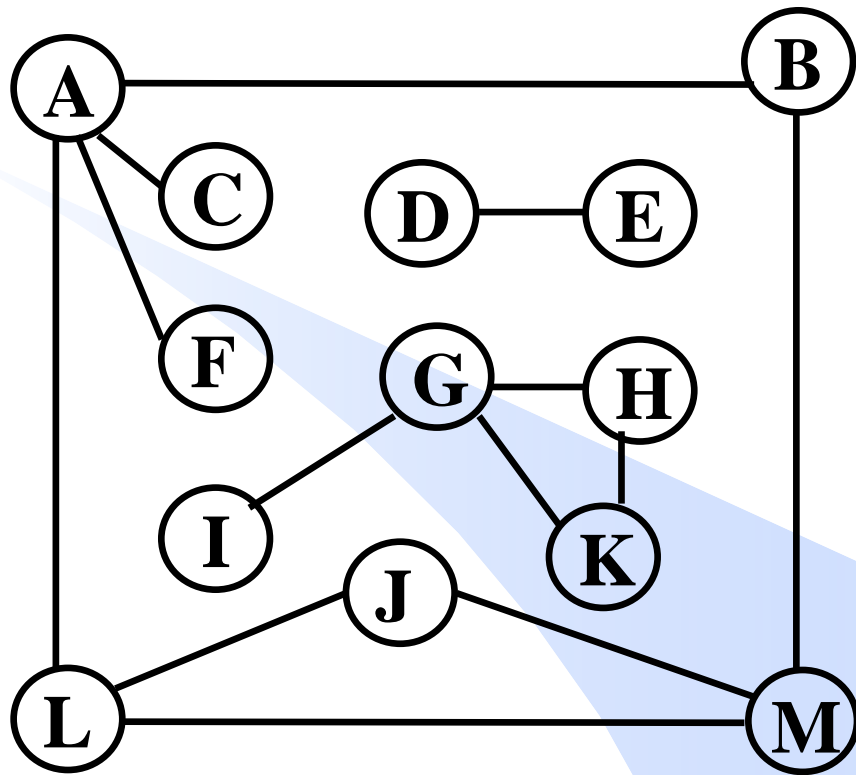
zhuyungang@jlu.edu.cn

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
    }
```



时间复杂度 $O(n+e)$

每调用一次DFS，遍历一个连通分量

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】

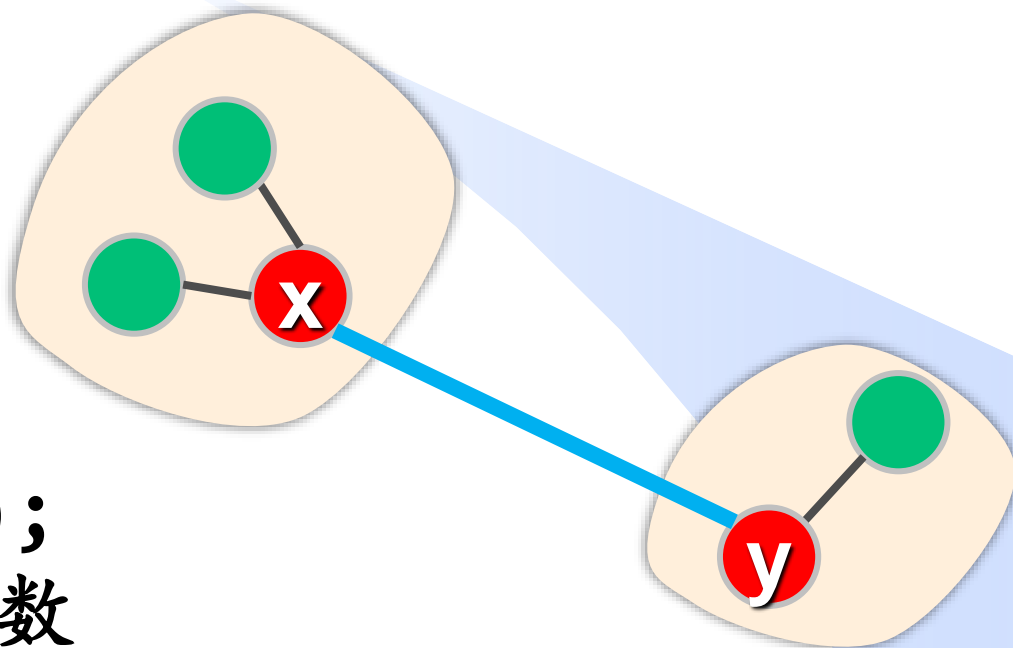
方案2：并查集

连通关系 \Leftrightarrow 等价关系

连通分量 \Leftrightarrow 并查集

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

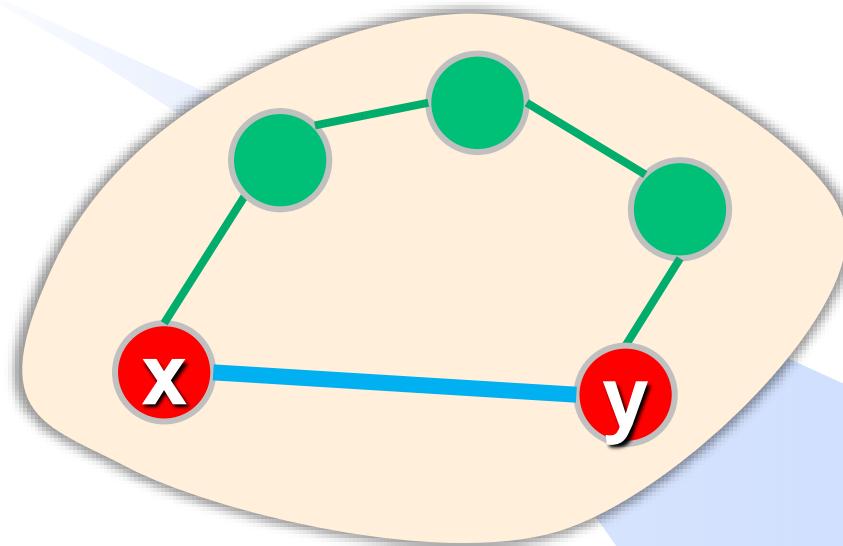
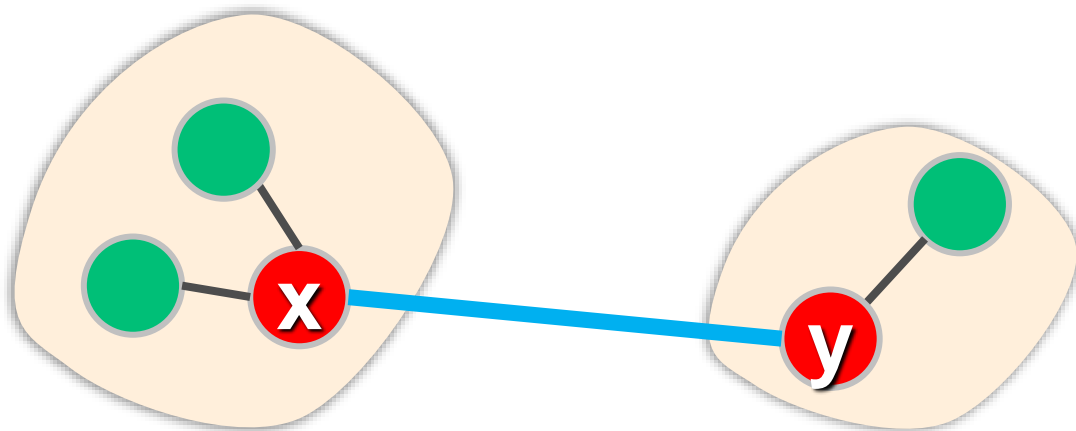
时间复杂度 $O(n+e)$



判断无向图中是否有环

并查集（连通关系 \Leftrightarrow 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】

➤ 以 u 为起点做DFS，看搜索过程中是否经过 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

给定 n 个顶点的有向无环图，输出顶点 u 到顶点 v 的所有路径。

【字节跳动、阿里、谷歌、微软、苹果面试题】



自愿性质OJ练习题

- ✓ [LeetCode 684](#) (并查集无向图判环)
- ✓ [POJ 1985](#) (树的直径)
- ✓ [HDU 4514](#) (无向图判环)