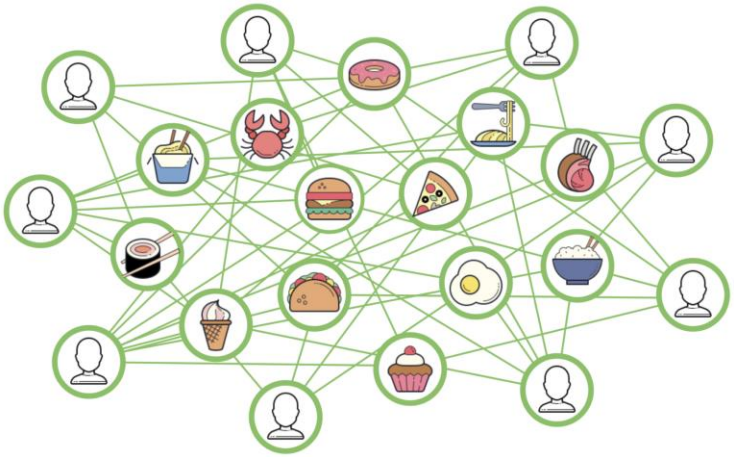
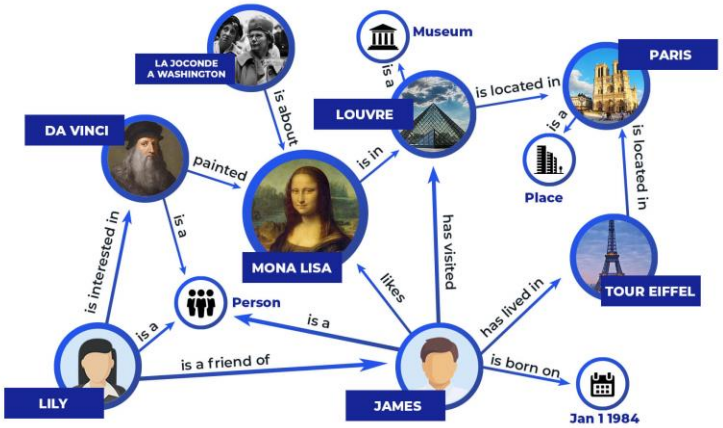
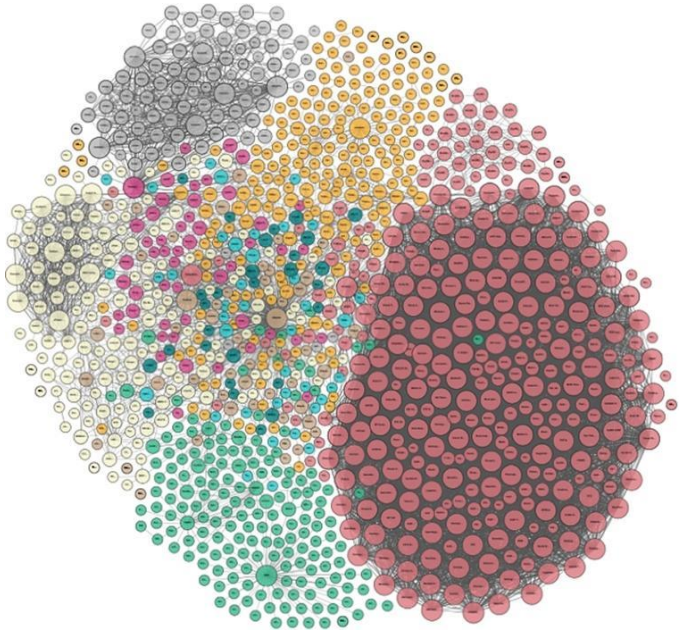
第10章 图





交通网络
分子结构
生物网络
社交网络
……



图网络
知识图谱

学习要点:

- 理解图的定义和与图相关的术语。
- 理解图是一个表示复杂非线性关系的数据结构。
- 掌握图的邻接矩阵表示及其实现方法。
- 掌握图的邻接表表示及其实现方法。
- 了解图的紧缩邻接表表示方法。
- 掌握图的广度优先搜索方法。
- 掌握图的深度优先搜索方法。
- 掌握单源最短路径问题的**Dijkstra**算法。
- 掌握有负权边的单源最短路径问题的**Bellman-Ford**算法。
- 掌握所有顶点对之间最短路径问题的**Floyd**算法。
- 掌握构造最小支撑树的**Prim**算法。
- 掌握构造最小支撑树的**Kruskal**算法。
- 理解图的最大匹配问题的增广路径算法。

10.1 图的基本概念

- 图(Graph)——图G是由两个集合V(G)和E(G)组成　记为G=(V,E)

  其中：V(G)是顶点的非空有限集

  E(G)是边的有限集合，边是顶点的无序对或有序对

- 有向图——有向图G是由两个集合V(G)和E(G)组成

  其中：V(G)是顶点的非空有限集

  E(G)是有向边的有限集合，弧是顶点的有序对，记为<v,w>，v,w是顶点，v为有向边的起点，w为有向边的终点

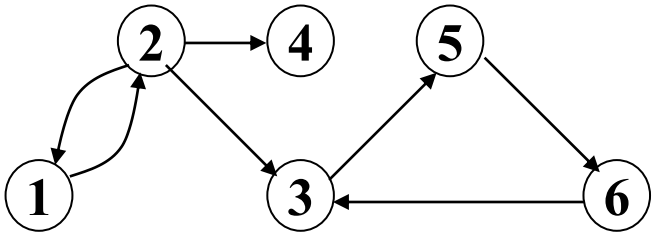
- 无向图——无向图G是由两个集合V(G)和E(G)组成

  其中：V(G)是顶点的非空有限集

  E(G)是边的有限集合，边是顶点的无序对，记为（v,w）或(w,v)，并且(v,w)=(w,v)

  本书约定：不考虑顶点到其自身的边；不允许一条边在图中重复出现！

例

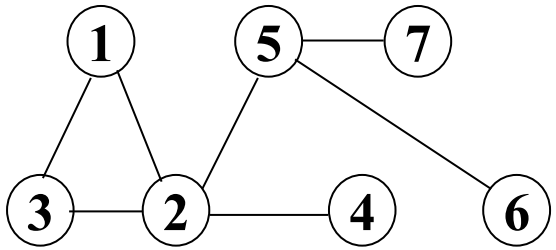


G1

图G1中：V(G1)={1,2,3,4,5,6}

E(G1)={<1,2>, <2,1>, <2,3>, <2,4>, <3,5>, <5,6>, <6,3>}

例



G2

图G2中：V(G2)={1,2,3,4,5,6,7}

E(G1)={(1,2), (1,3), (2,3), (2,4),(2,5), (5,6), (5,7)}

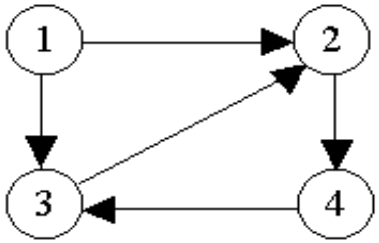
- 完全图——设|V|=n,|E|=e。对有向图G，若e=n(n-1)，则称G为完全的有向图；对无向图G，若e=n(n-1)/2，则称G为完全的无向图。
- 邻接、关联——若(u,v)是一条无向边，则称顶点u和v互为邻接点，或称u和v相邻接。若(u,v)是一条有向边，则称v是u的邻接顶点。
- 顶点的度
  - 无向图中，顶点v的度为关联于该顶点相连的边数，记为D(v)
  - 有向图中，顶点v的度分成入度与出度
    - 入度：以顶点v为终点的边的数目，记为ID(v)
    - 出度：以顶点v为起点的边的数目，记为OD(v)
    - D(v)=ID(v)+OD(v)

无论是有向图还是无向图，顶点数n，边数e和度数之间有如下关系：

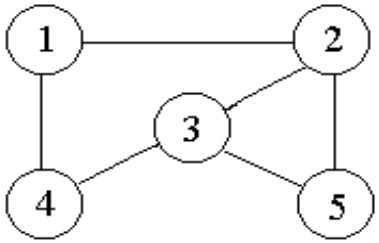
$$e = \frac{1}{2}\sum_{i=1}^{n} D(v_i)$$

- 子图——如果图G(V,E)和图G'(V',E'),满足：

  V'⊆V　　E'⊆E　　则称G'为G的子图

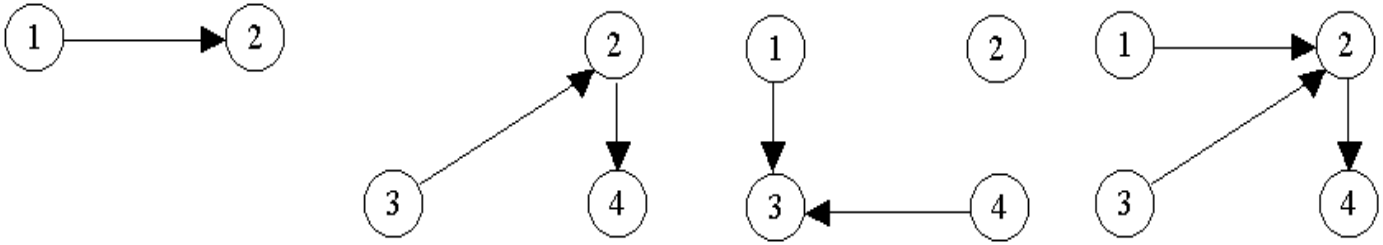


G1

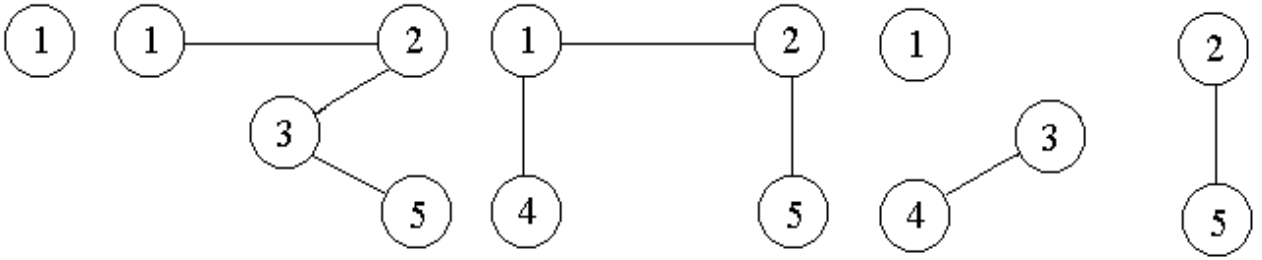


G2

有向图G1的若干子图



无向图G2的若干子图



- 路径：

  在无向图G中，若存在一个顶点序列u(1),u(2),…，u(m),使得(u(i),u(i+1))∈E(G)，i=1,2,…，m-1, 则称该顶点序列为顶点u(1)和u(m)之间的一条路径。其中u(1)称为该路径的起点，u(m)称为该路径的终点。

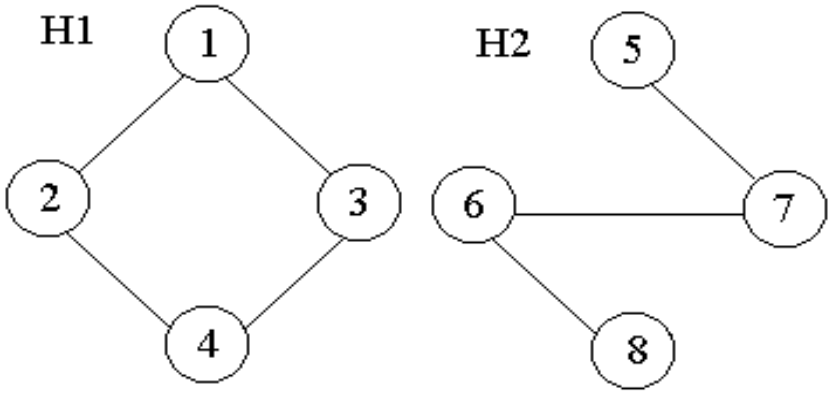
  若图G是有向图，则路径也是有向的，其中每条边(u(i),u(i+1))，i=1,2,…，m-1均为有向边。

  路径的长度：路径所包含的边数m-1称之。

- 简单路—若一条路径上除了起点和终点可能相同外，其余顶点均不相同，则称此路径为一条简单路径。
- 回路—起点和终点相同的简单路径称为简单回路或简单环或圈。
- 有根图—在一个有向图中，若有一个顶点v，从该顶点有路径可以到达图中其它所有顶点,则称此有向图为有根图。v称为该有根图的根。

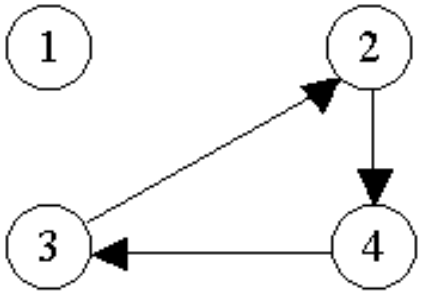
- 连通——无向图G中，若从顶点V到顶点W有一条路径，则说V和W是连通的
- 连通图——无向图中任意两个顶点都是连通的叫连通图
- 连通分支——无向图的极大连通子图叫连通分支

下图有两个连通分支：



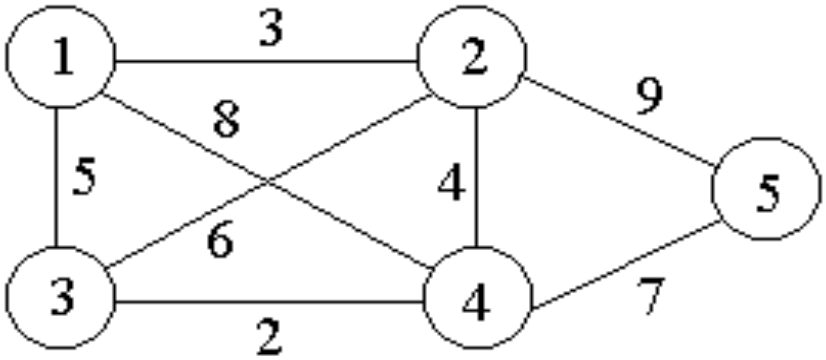
- 强连通图——有向图中，如果对每一对Vi,Vj∈V, Vi≠Vj, 从Vi到Vj 和从Vj到 Vi都存在路径，则称G是强连通图
- 强连通分支——有向图的极大强连通子图叫强连通分支

显然，强连通图只有一个强连通分支，即其自身。非强连通的有向图有多个强连通分支。如下图中的图不是强连通图，但它有2个强连通分支。



赋权图和网络

若无向图的每条边都带一个权，则称相应的图为赋权无向图。同理，若有向图的每条边都带一个权，则称相应的图为赋权有向图。通常，权是具有某种实际意义的数，比如，2个顶点之间的距离，耗费等。赋权无向图和赋权有向图统称为网络。下图就是一个网络的例子。



返回章目录

10.2 抽象数据类型ADT图

ADT图支持的基本运算以有向图为基本模型。

ADT图支持的基本运算如下：

(1)GraphInit (n): 创建有n个孤立顶点的图。
(2)GraphExist (i, j, G): 判断图G中是否存在边(i, j)。
(3)GraphEdges (G): 返回图G的边数。
(4) GraphVertices (G): 返回图G的顶点数。
(5) GraphAdd (i, j, G): 在图G中加入边(i,j),。
(6) GraphDelete (i, j, G):删除图G的边(i,j) 。
(7)Degree (i, G): 返回图G中顶点i的度数。
(8)OutDegree (i, G): 返回图G中顶点i的出度。
(9)InDegree (i, G): 返回图G中顶点i的入度。

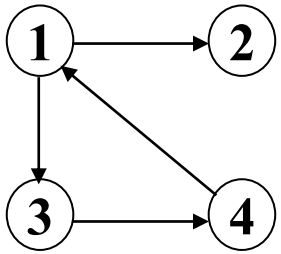
返回章目录

10.3 图的表示法

- 邻接矩阵——表示顶点间邻接关系的矩阵
  - 定义：设G=(V,E)是有n≥1个顶点的图，G的邻接矩阵A是具有以下性质的n阶方阵

$$A[i,j]=\begin{cases}1, 若(v_i,v_j)或<v_i,v_j>\in E(G)\\ 0, \ 其它\end{cases}$$

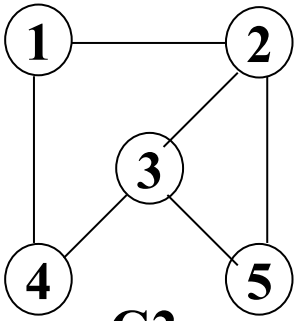
例



G1

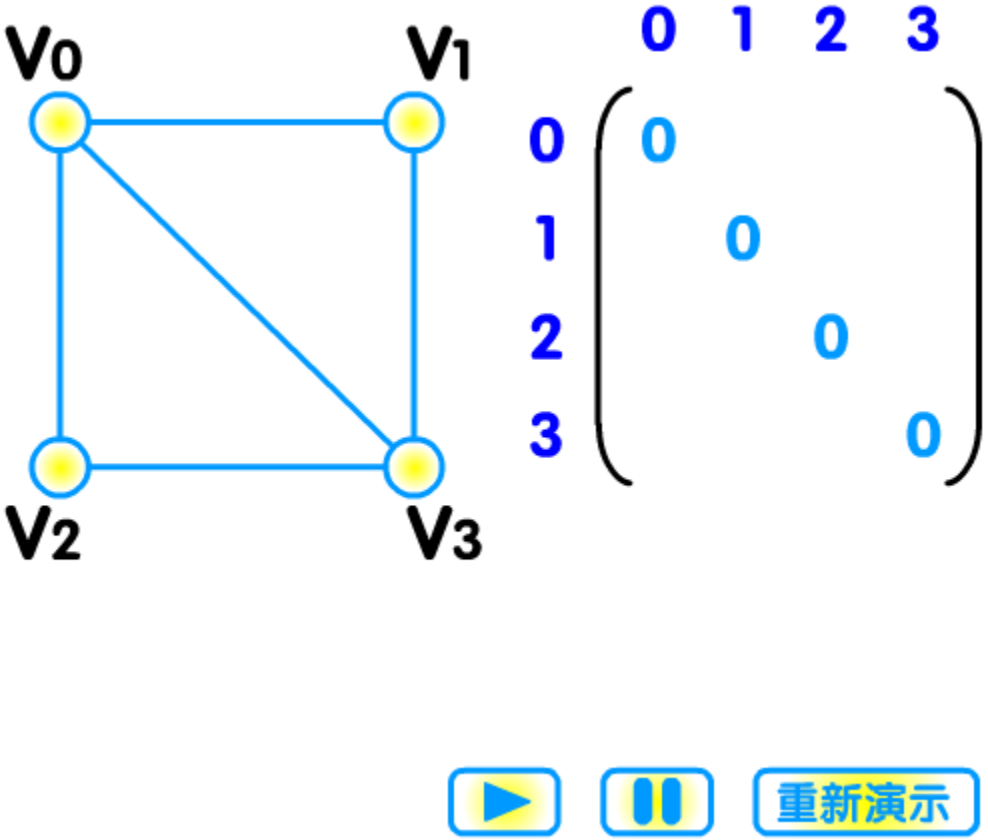
	①	②	③	④
①	0	1	1	0
②	0	0	0	0
③	0	0	0	1
④	1	0	0	0

例



G2

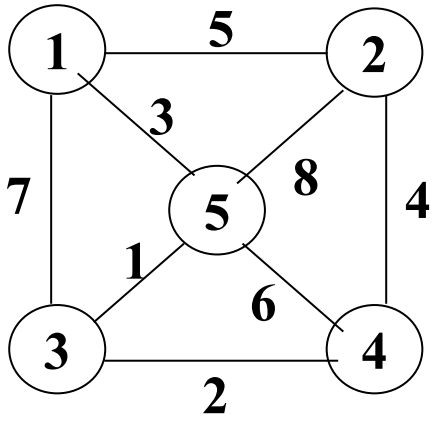
	①	②	③	④	⑤
①	0	1	0	1	0
②	1	0	1	0	1
③	0	1	0	1	1
④	1	0	1	0	0
⑤	0	1	1	0	0


V0
V1
V2
V3
0 1 2 3
0 0
1 0
2 0
3 0
重新演示

- 特点：
  - 无向图的邻接矩阵对称，可压缩存储；有n个顶点的无向图需存储空间为n(n+1)/2
  - 有向图邻接矩阵不一定对称；有n个顶点的有向图需存储空间为n^2
  - 无向图中顶点V_i的度$TD(V_i)$是邻接矩阵A中第i行元素之和
  - 有向图中：
    - 顶点V_i的出度是A中第i行元素之和
    - 顶点V_i的入度是A中第i列元素之和
  - 网络的邻接矩阵可定义为：

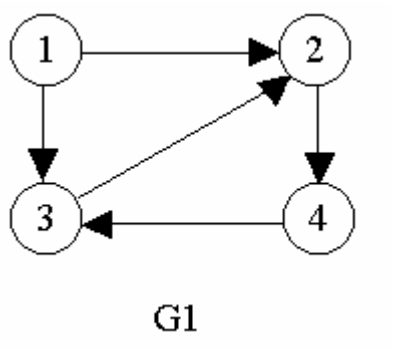
$$A[i,j]=\begin{cases}\omega_{ij}, \text{若}(v_i,v_j)\text{或}<v_i,v_j>\in E(G)\\ \infty, \text{其它}\end{cases}$$

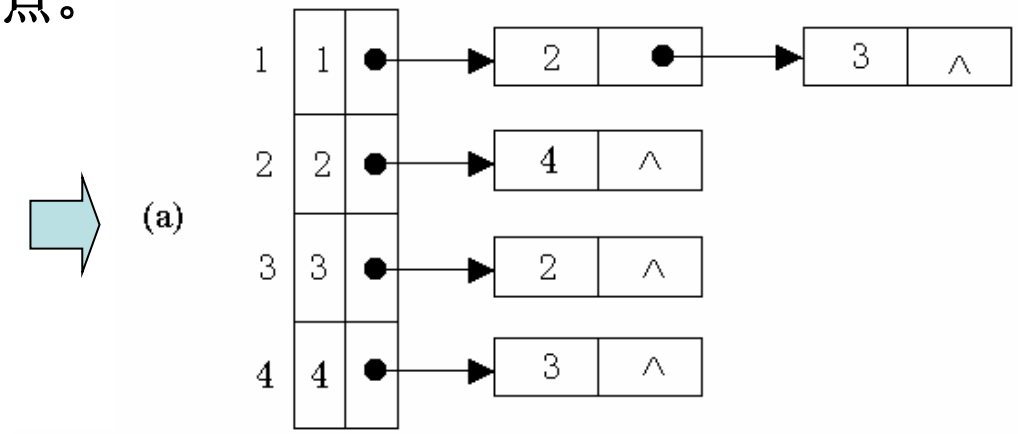
例

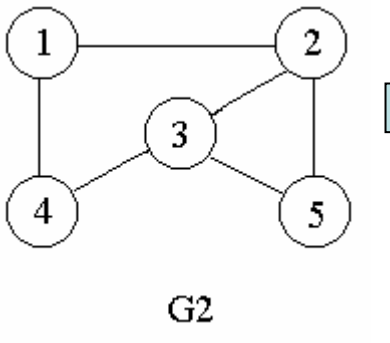


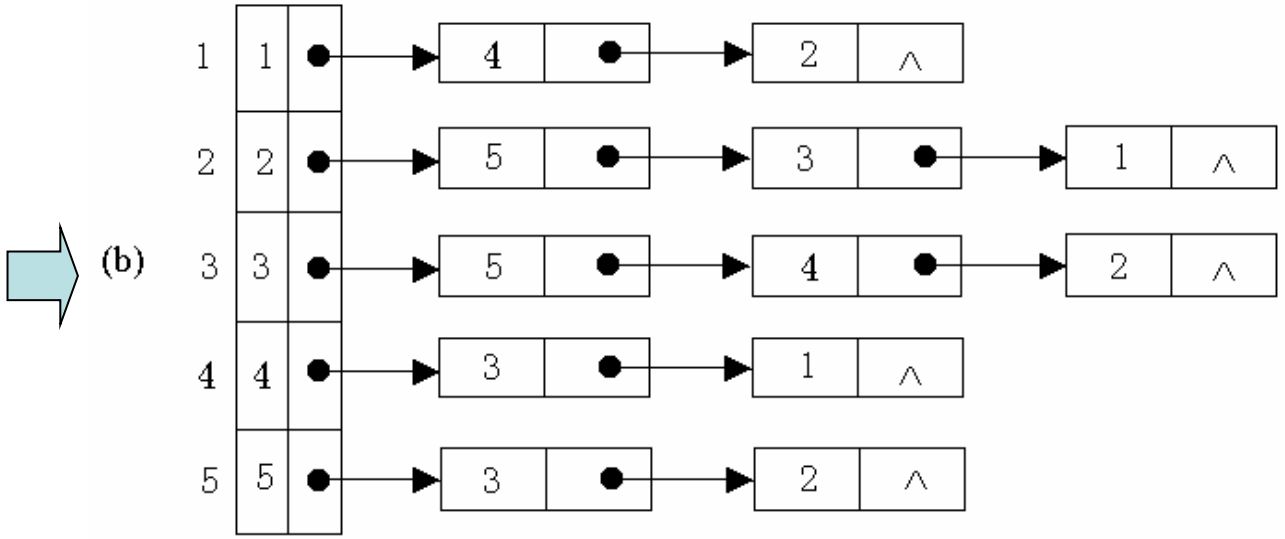
$$
\begin{array}{c} \\ ① \\ ② \\ ③ \\ ④ \\ ⑤ \end{array}
\begin{array}{c}
\begin{array}{ccccc} ① & ② & ③ & ④ & ⑤ \end{array} \\
\begin{bmatrix} \infty & 5 & 7 & \infty & 3 \\ 5 & \infty & \infty & 4 & 8 \\ 7 & \infty & \infty & 2 & 1 \\ \infty & 4 & 2 & \infty & 6 \\ 3 & 8 & 1 & 6 & \infty \end{bmatrix}
\end{array}
$$

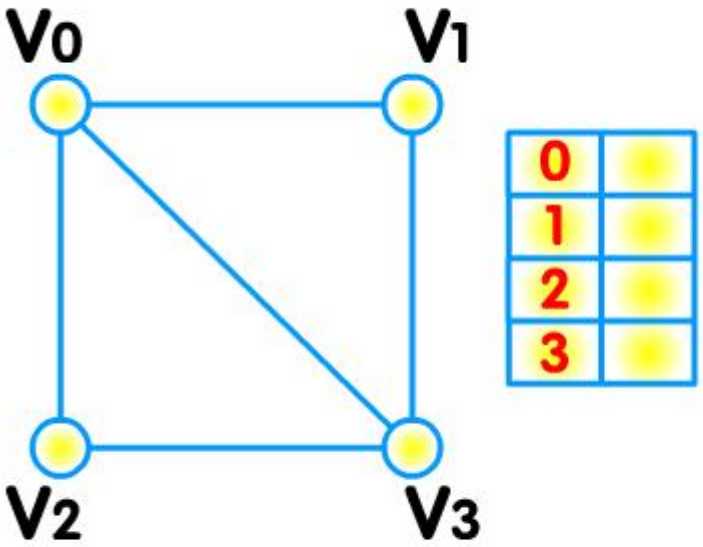
- 邻接表
  - 实现：为图中每个顶点建立一个单链表，第i个单链表存放顶点Vi的所有邻接顶点。



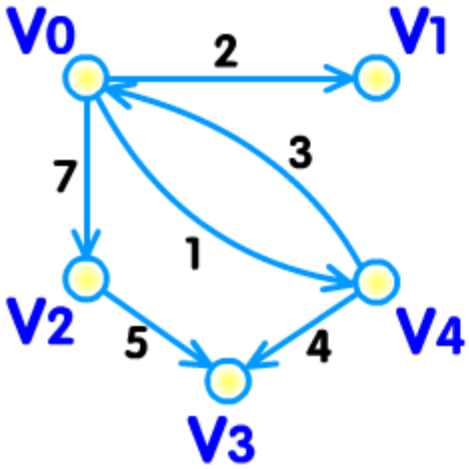








重新演示

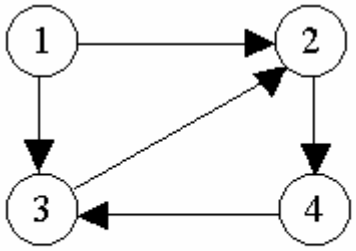


V0	
V1	
V2	
V3	
V4	

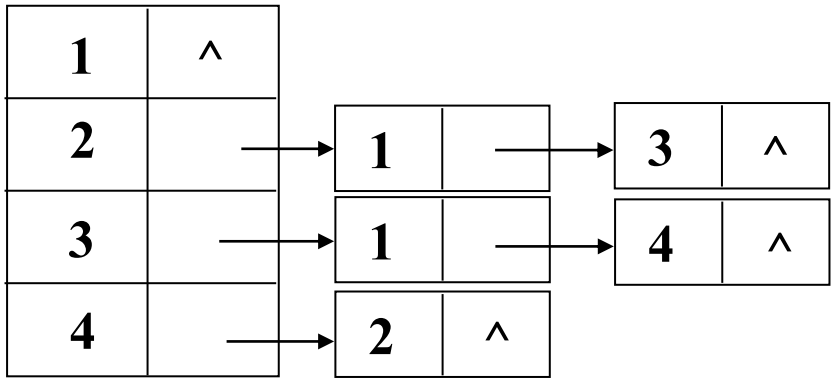
- 特点
  - 无向图中顶点V_i的度为第i个单链表中的结点数
  - 有向图中
    - 顶点V_i的出度为第i个单链表中的结点个数
    - 顶点V_i的入度为整个单链表中邻接点域值是i的结点个数
- 逆邻接表：有向图中对每个结点建立以V_i为终点的边的单链表

求解麻烦！

例



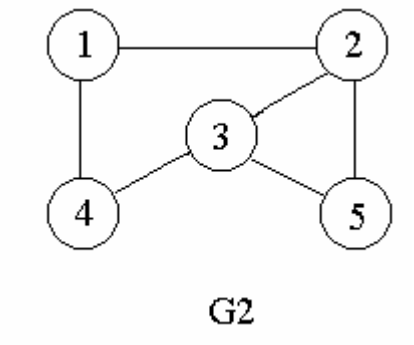
G1



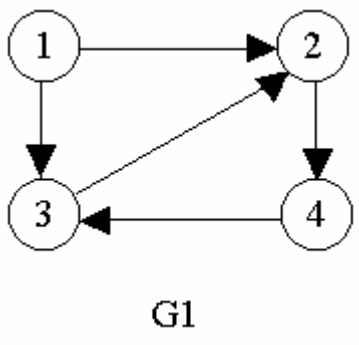
紧缩邻接表

紧缩邻接表将图G的每个顶点的邻接表紧凑地存储在2个一维数组List和h中。其中一维数组List依次存储顶点1,2,…, n的邻接顶点。数组单元h[i]存储顶点i的邻接表在数组List中的起始位置。

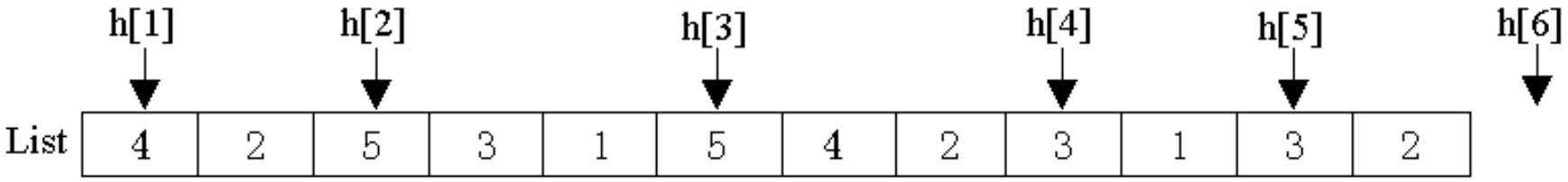
如图G2和G1的紧缩邻接表表示分别如下图(a)和(b)：



G2

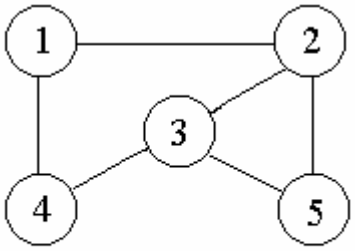


G1

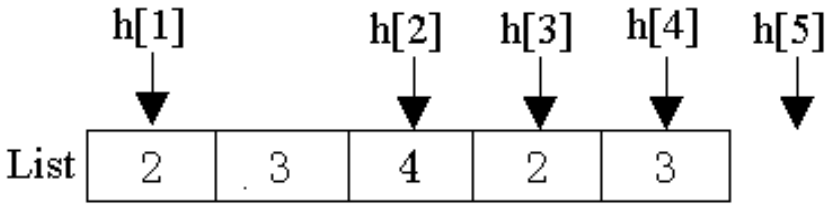


h=[0,2,5,8,10,12]

(a)

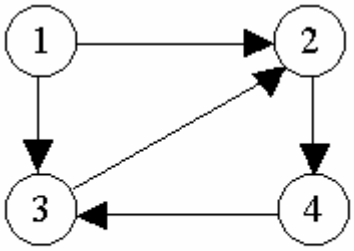


G2



h=[0,2,3,4,5]

(b)



G1

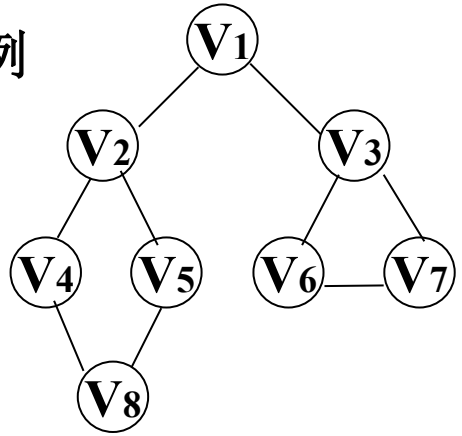
返回章目录

10.4 图的遍历

广度优先搜索(BFS)

- 方法：从图的某一顶点V0出发，访问此顶点后，依次访问V0的各个未曾访问过的邻接顶点；然后分别从这些邻接顶点出发，广度优先遍历图，直至图中所有已被访问的顶点的邻接点都被访问到；若此时图中尚有顶点未被访问，则另选图中一个未被访问的顶点作起点，重复上述过程，直至图中所有顶点都被访问为止

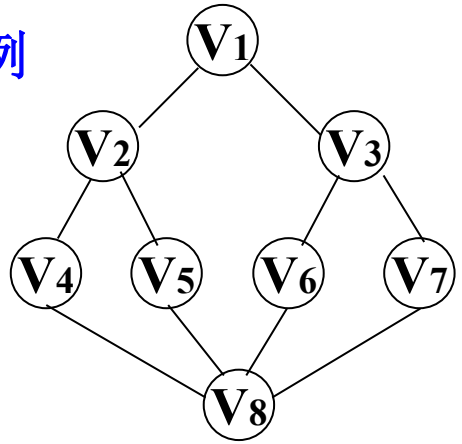
例



类似于：树的层次遍历！

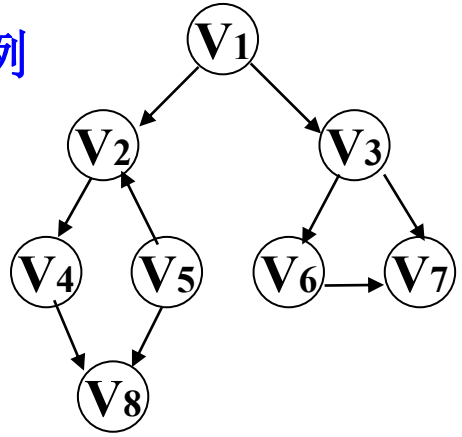
广度遍历：V1⇒ V2 ⇒V3 ⇒ V4 ⇒V5 ⇒V6 ⇒V7 ⇒V8

例



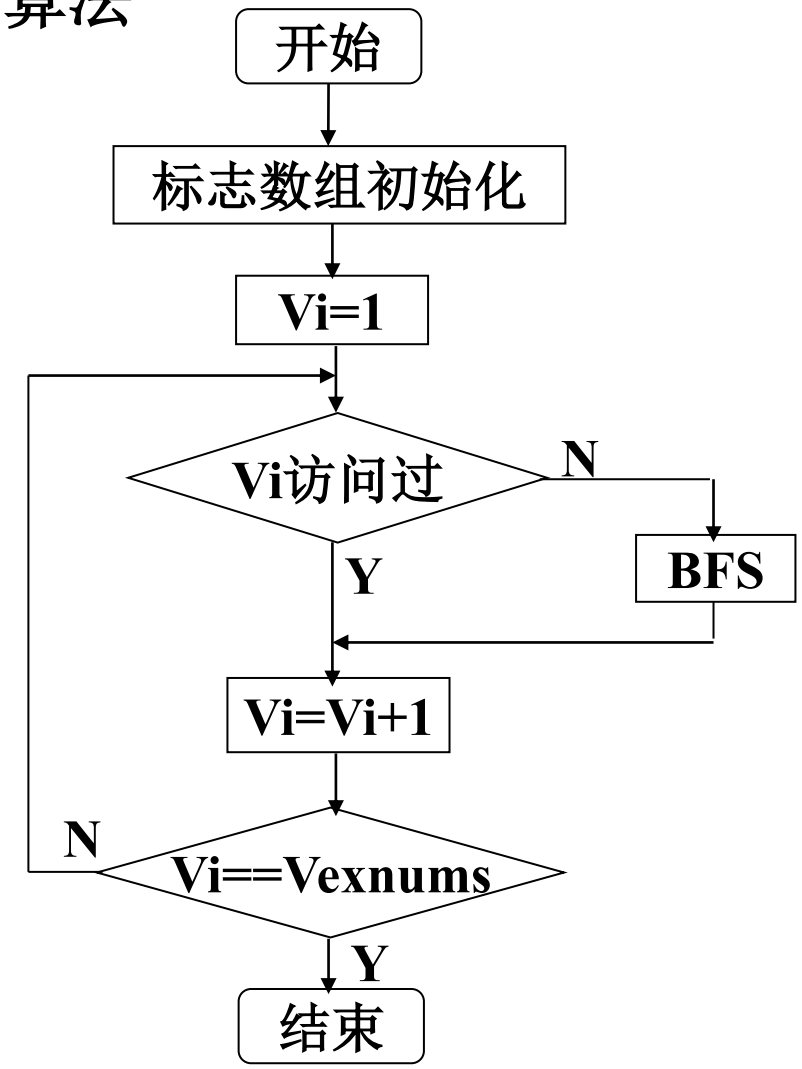
广度遍历：V1⇒ V2 ⇒V3 ⇒ V4 ⇒V5 ⇒V6 ⇒V7 ⇒V8

例

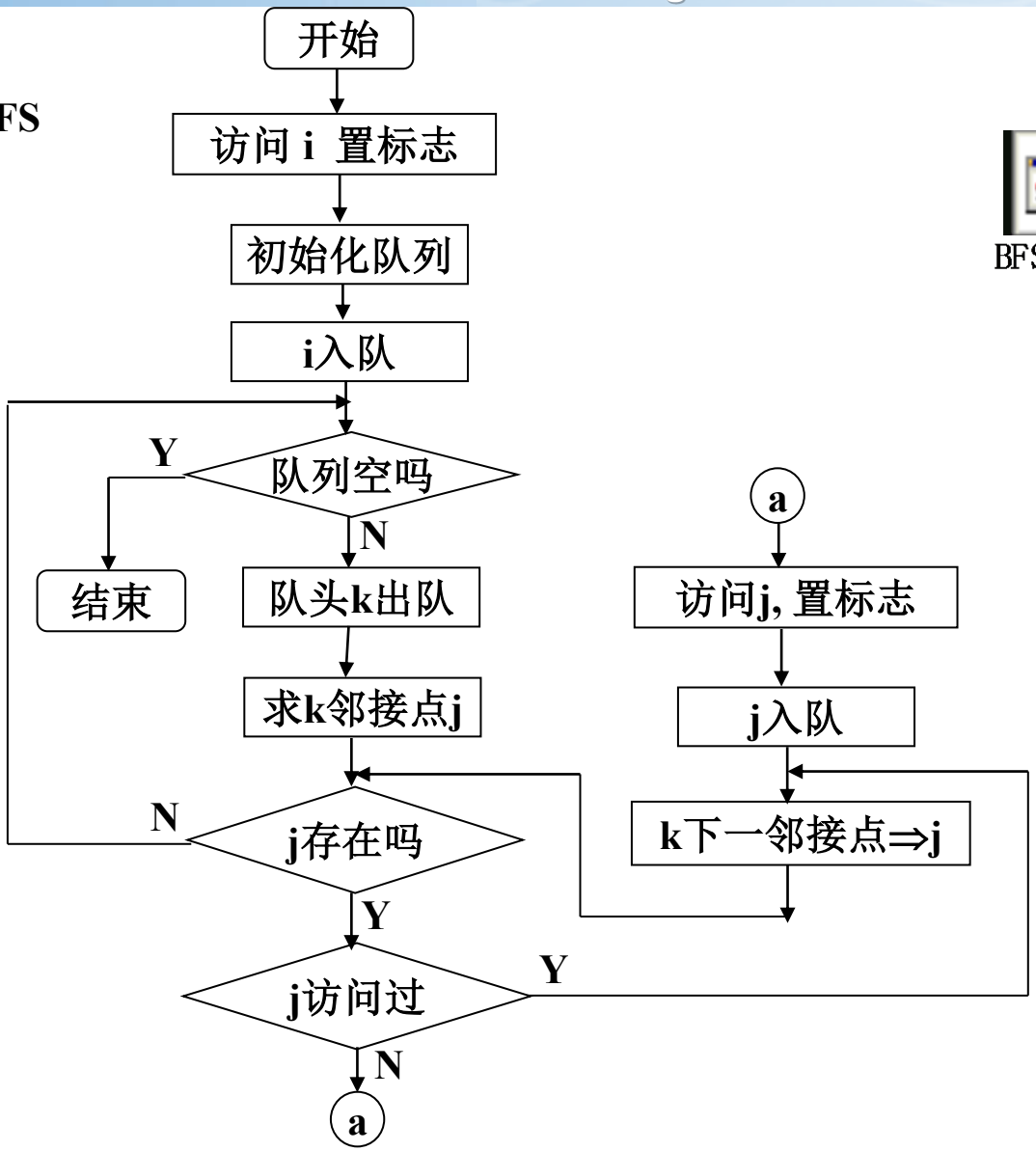


广度遍历：V1⇒ V2 ⇒V3 ⇒ V4 ⇒V6 ⇒V7 ⇒V8 ⇒V5

- 广度优先遍历算法

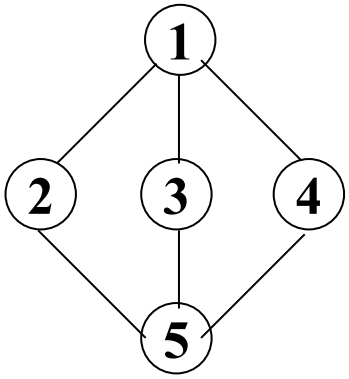


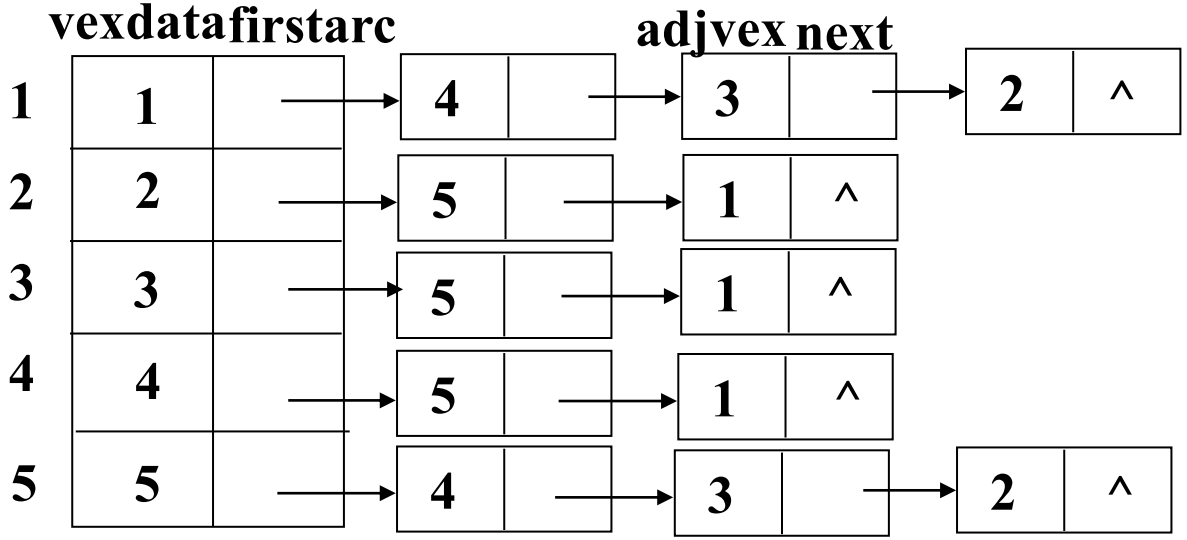
BFS

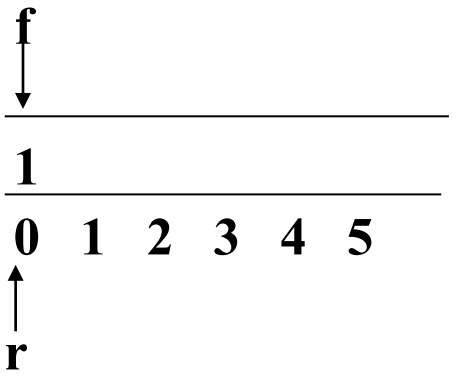

开始
访问 i 置标志
初始化队列
i入队
队列空吗
Y
N
结束
队头k出队
求k邻接点j
j存在吗
N
Y
j访问过
Y
N
a
a
访问j, 置标志
j入队
k下一邻接点⇒j


BFS.txt

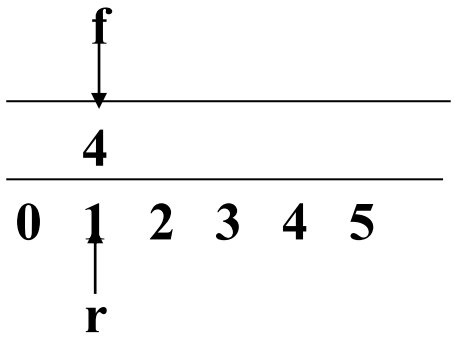
例



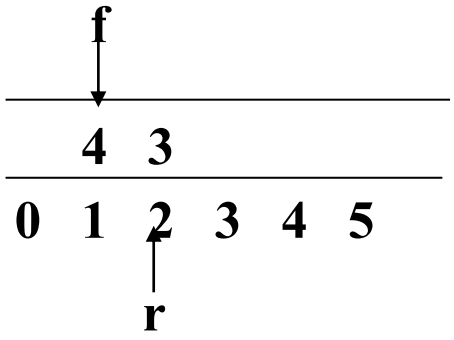




遍历序列：1

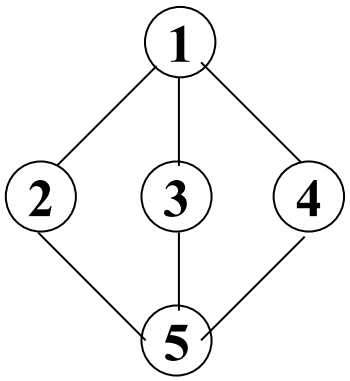


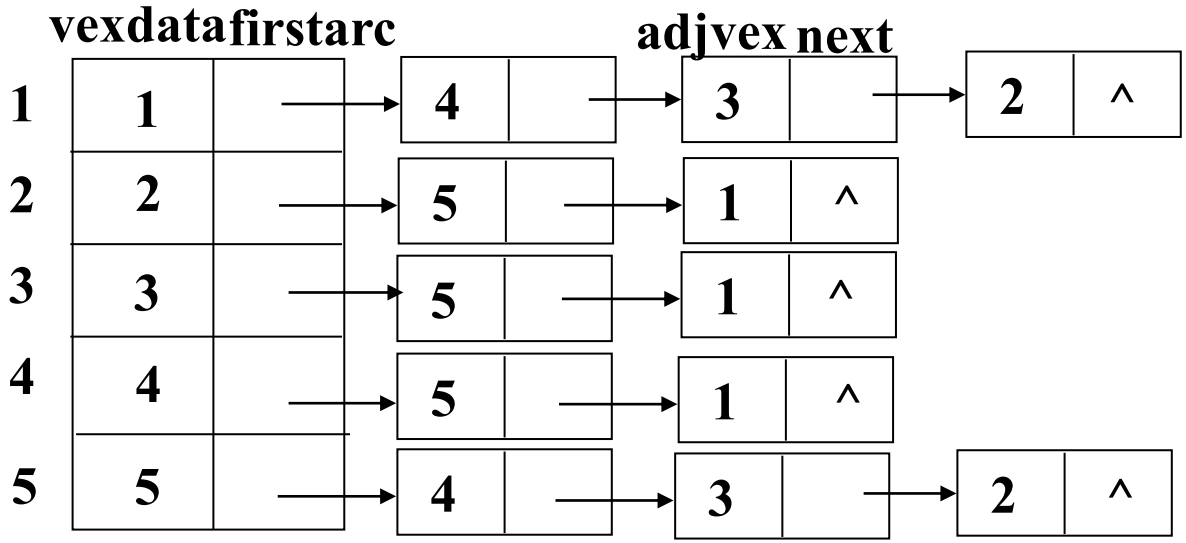
遍历序列：1 4

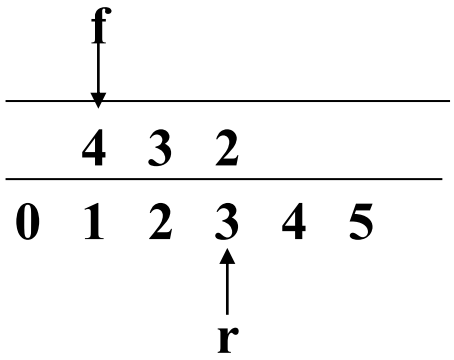


遍历序列：1 4 3

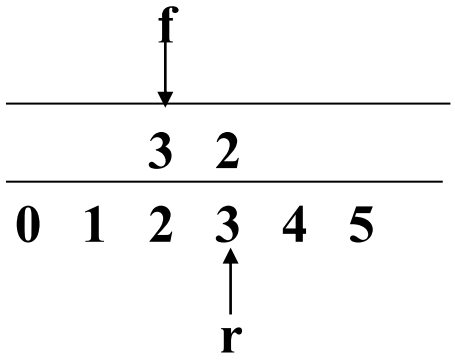
例



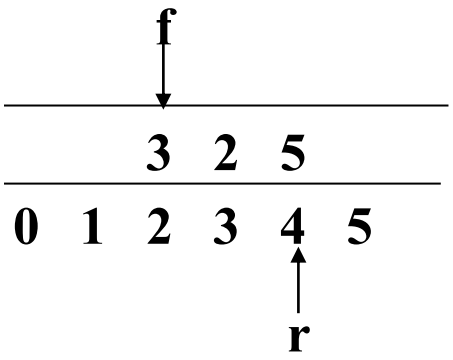




遍历序列：1 4 3 2

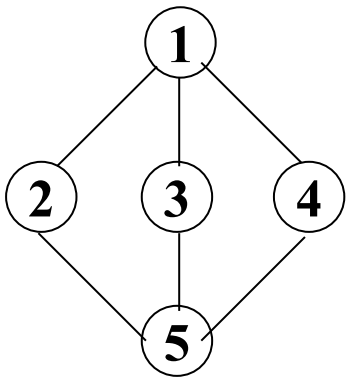


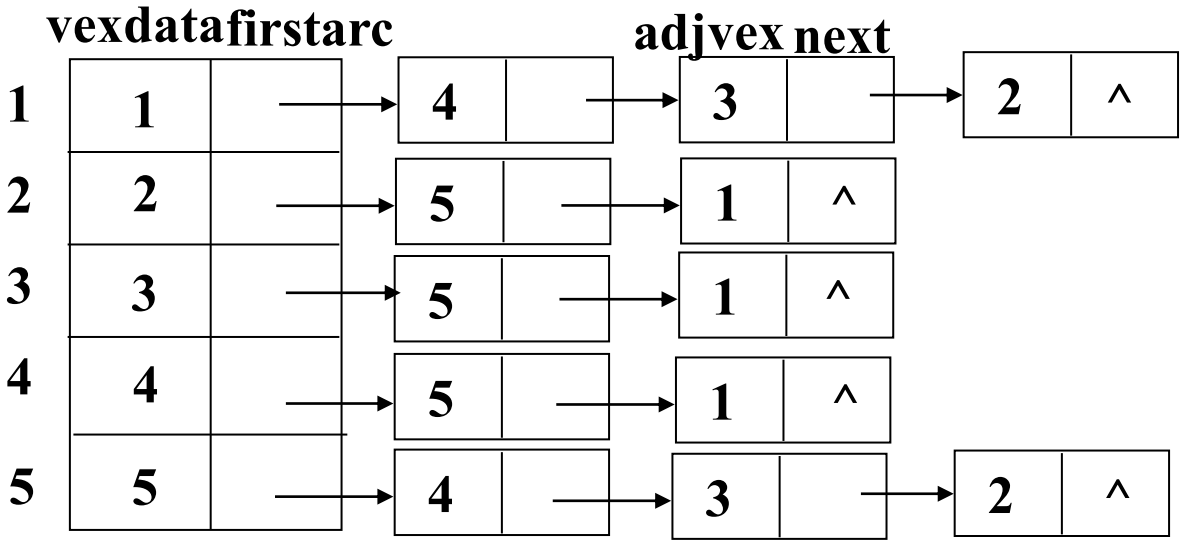
遍历序列：1 4 3 2

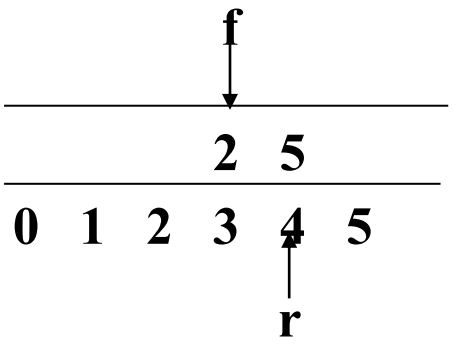


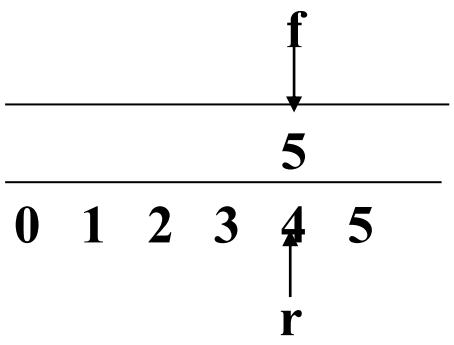
遍历序列：1 4 3 2 5

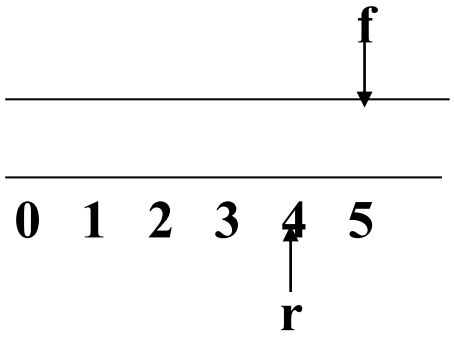
例











遍历序列：1 4 3 2 5

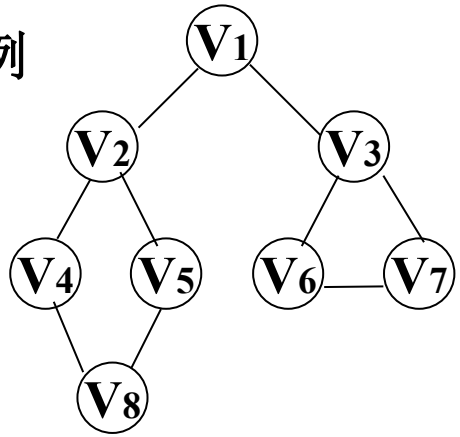
遍历序列：1 4 3 2 5

遍历序列：1 4 3 2 5

10.4 图的遍历

- 深度优先遍历(DFS)
  - 方法：从图的某一顶点V0出发，访问此顶点；然后依次从V0的未被访问的邻接点出发，深度优先遍历图，直至图中所有和V0相通的顶点都被访问到；若此时图中尚有顶点未被访问，则另选图中一个未被访问的顶点作起点，重复上述过程，直至图中所有顶点都被访问为止

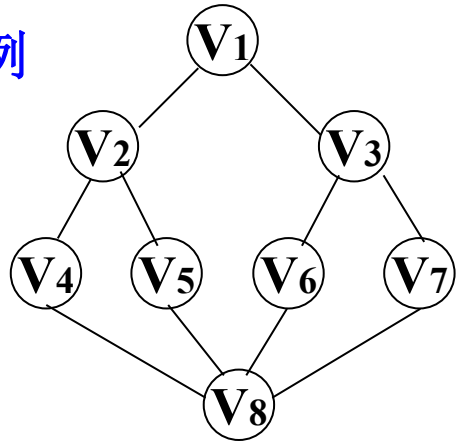
例



类似于：树的前序遍历！

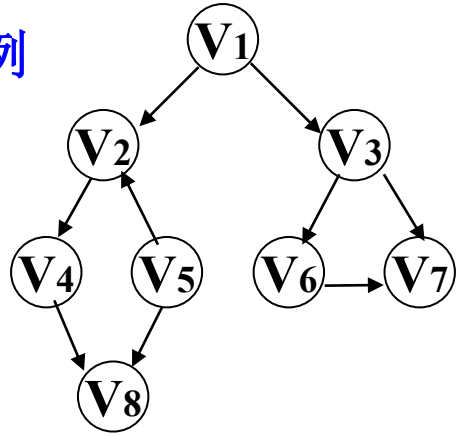
深度遍历：V1⇒ V2 ⇒V4 ⇒ V8 ⇒V5 ⇒V3 ⇒V6 ⇒V7

例



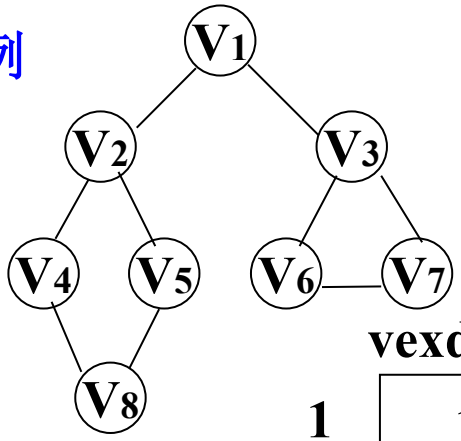
深度遍历：V1⇒ V2 ⇒V4 ⇒ V8 ⇒V5 ⇒V6 ⇒V3 ⇒V7

例



深度遍历：V1⇒ V2 ⇒V4 ⇒ V8 ⇒V3 ⇒V6 ⇒V7 ⇒V5

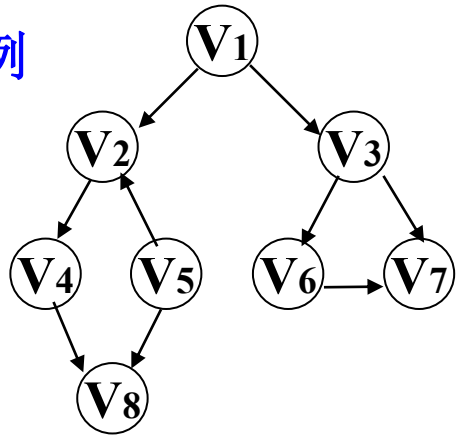
例



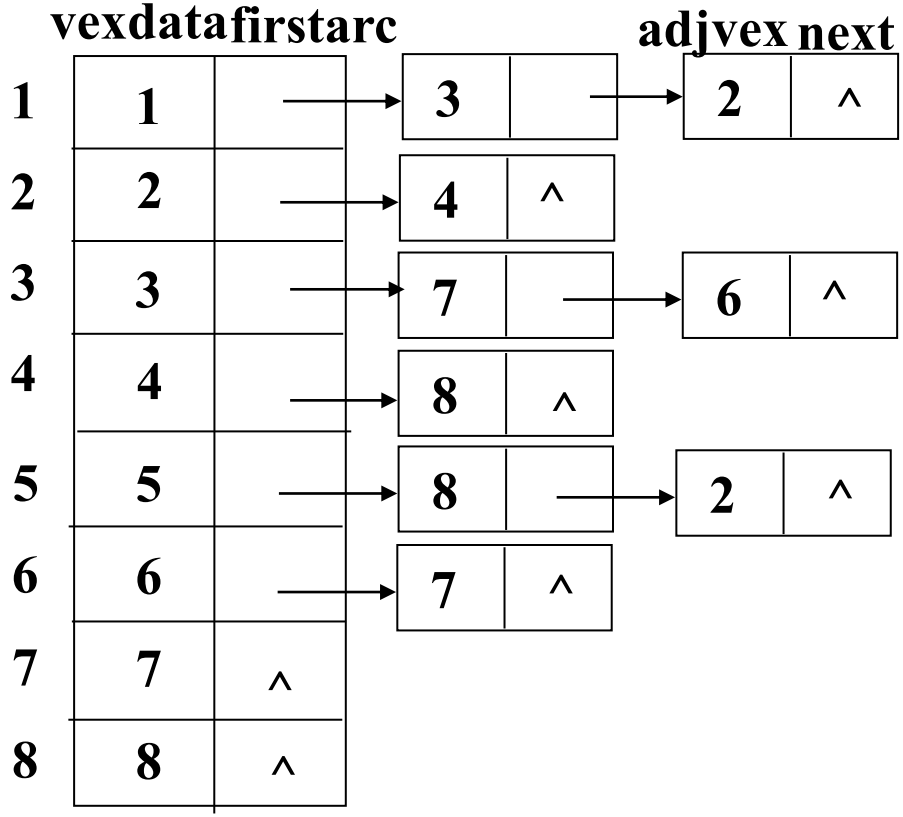
深度遍历：V1⇒V3 ⇒V7 ⇒V6 ⇒V2 ⇒V5 ⇒V8 ⇒V4

	vexdata	firstarc	adjvex	next
1	1	→	3	2 ^
2	2	→	5	4 → 1 ^
3	3	→	7	6 → 1 ^
4	4	→	8	2 ^
5	5	→	8	2 ^
6	6	→	7	3 ^
7	7	→	6	3 ^
8	8	→	5	4 ^

例



深度遍历：V1⇒V3 ⇒V7 ⇒V6 ⇒V2 ⇒V4 ⇒V8 ⇒V5

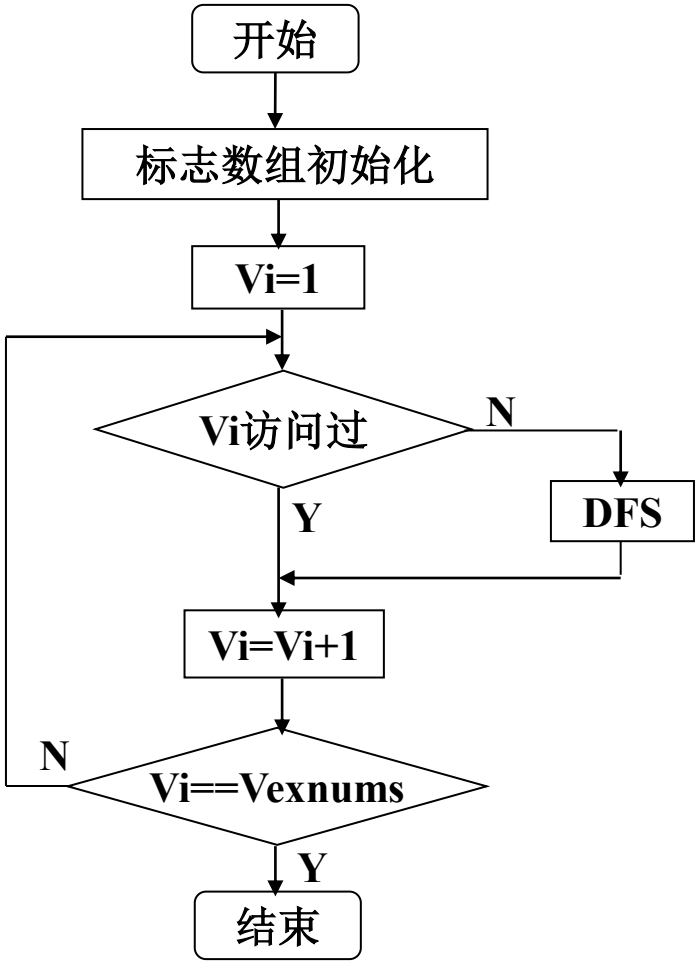


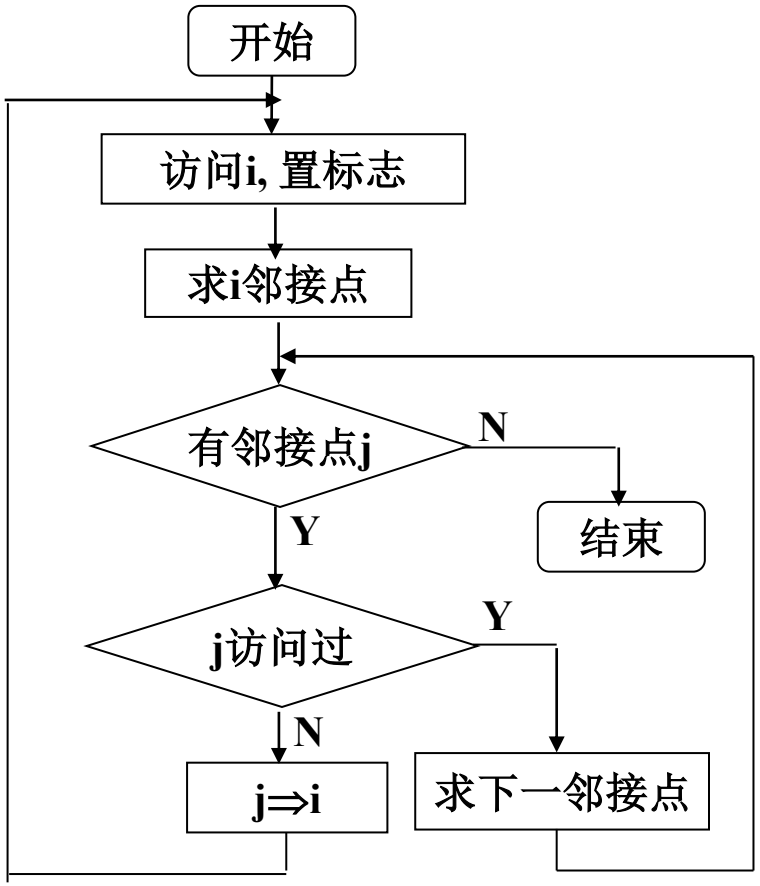
深度优先遍历算法

递归算法

DFS.txt

DFS算法演示

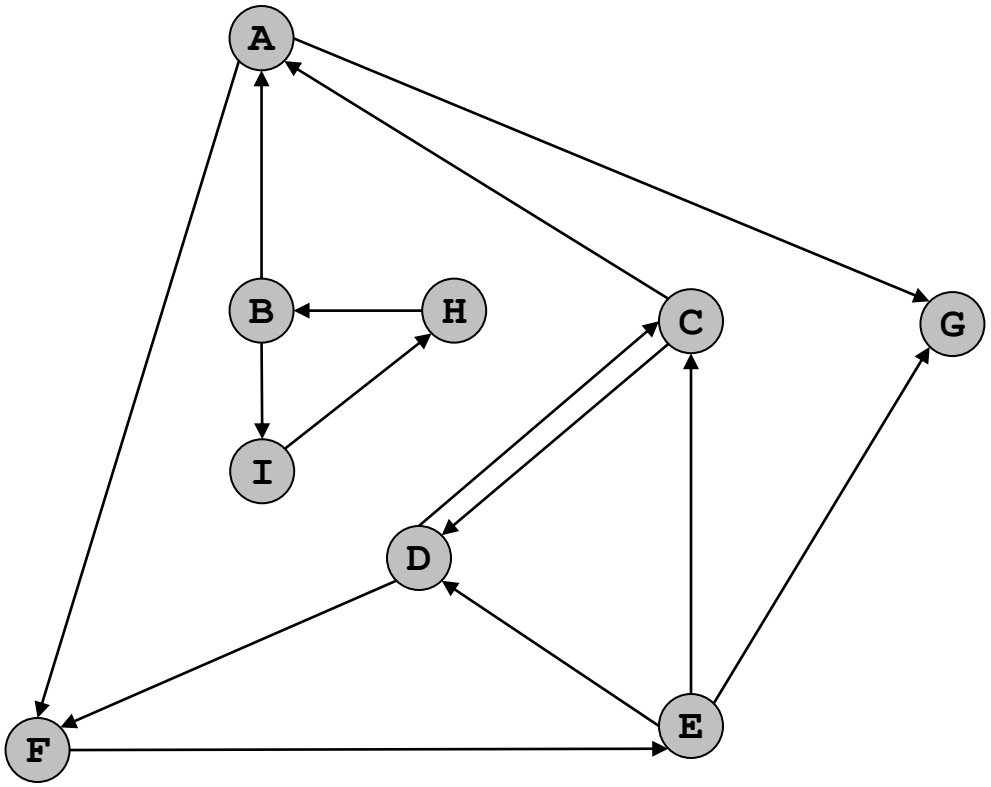




DFS

返回章目录

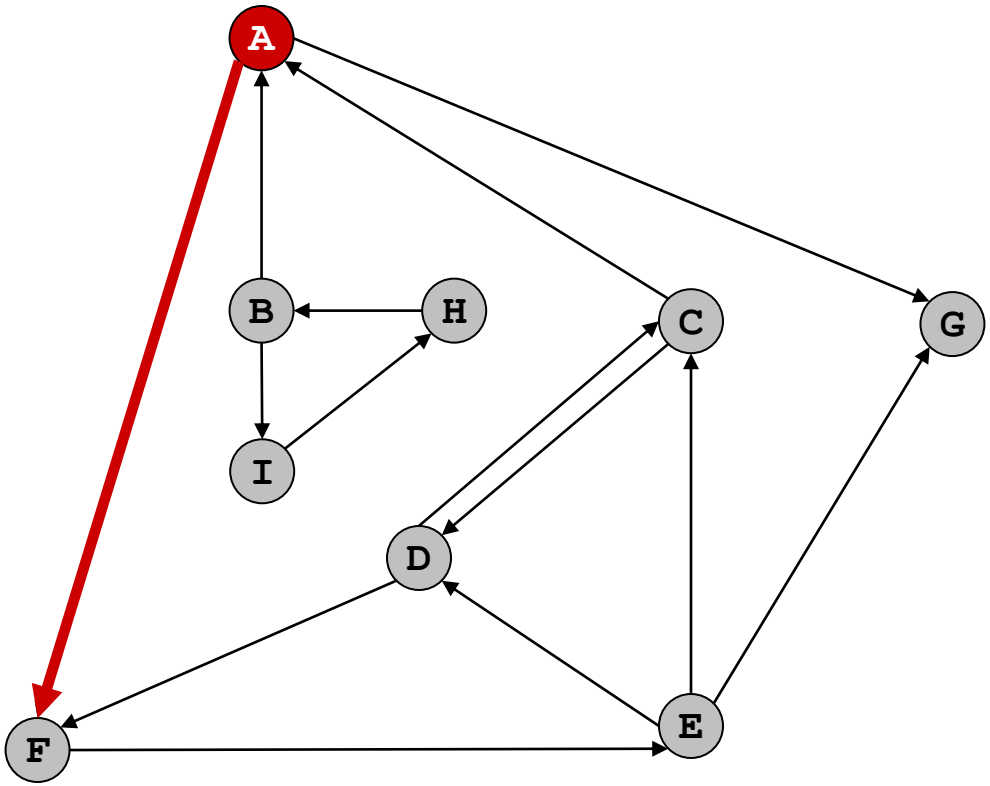
DFS算法演示



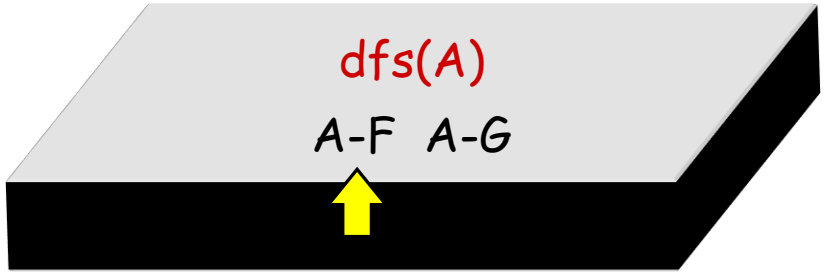
Adjacency Lists

```
A:   F G
B:   A H
C:   A D
D:   C F
E:   C D G
F:   E
G:
H:   B
I:   H
```

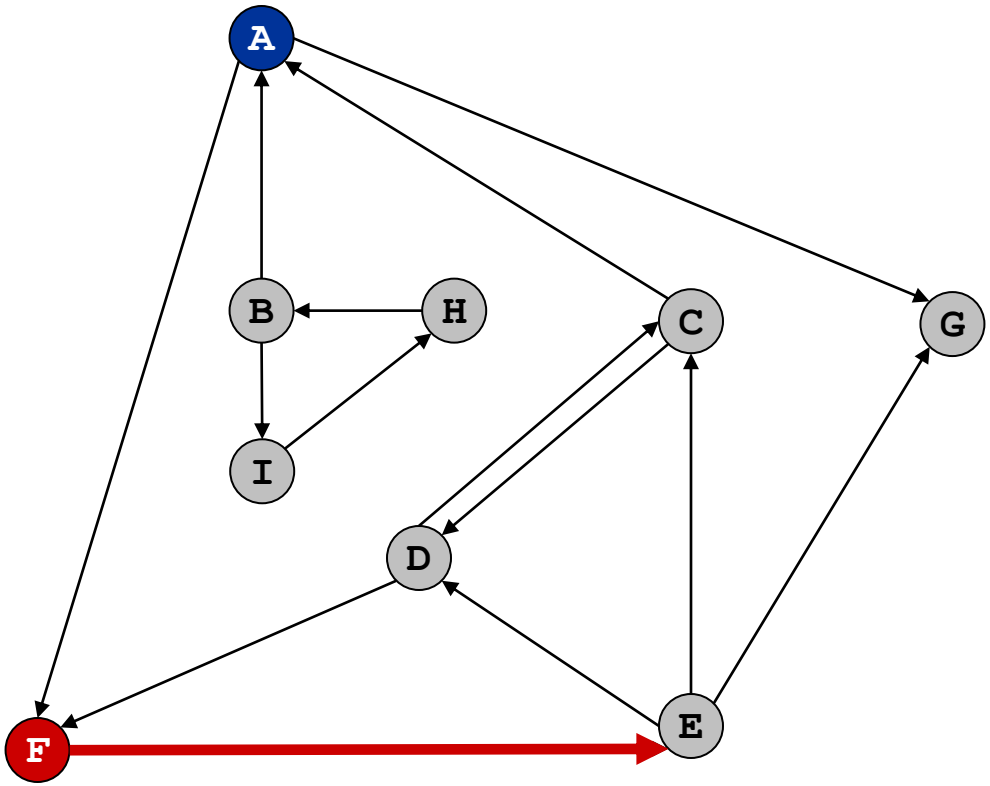
DFS算法演示



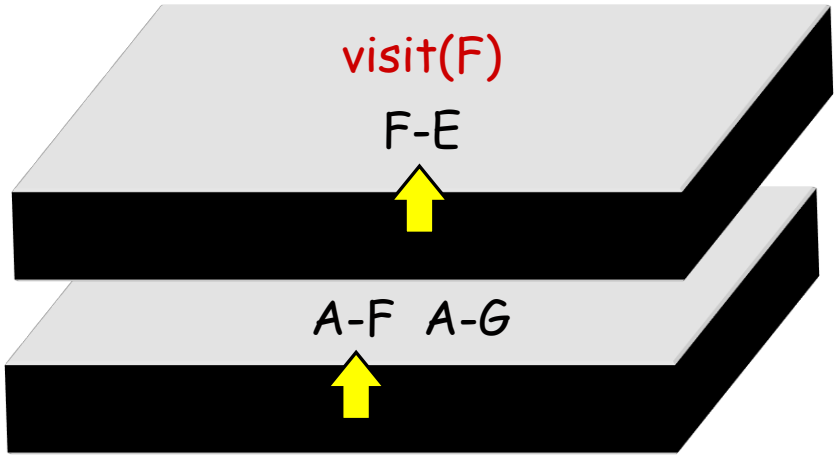
Function call stack:



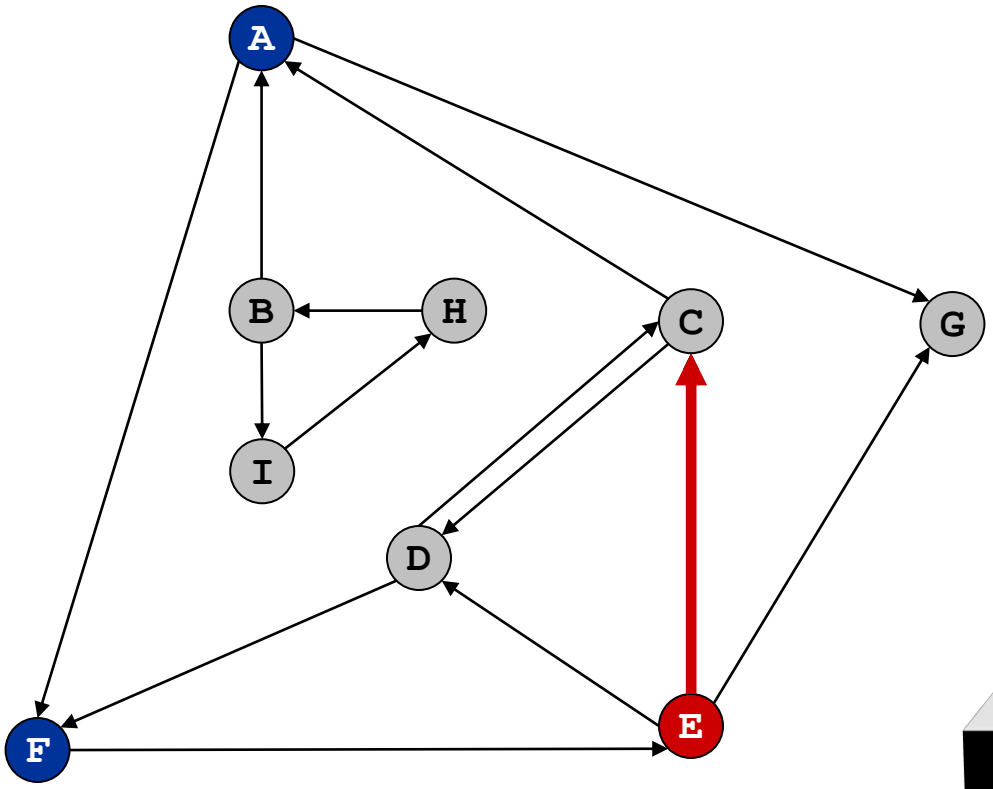
DFS算法演示

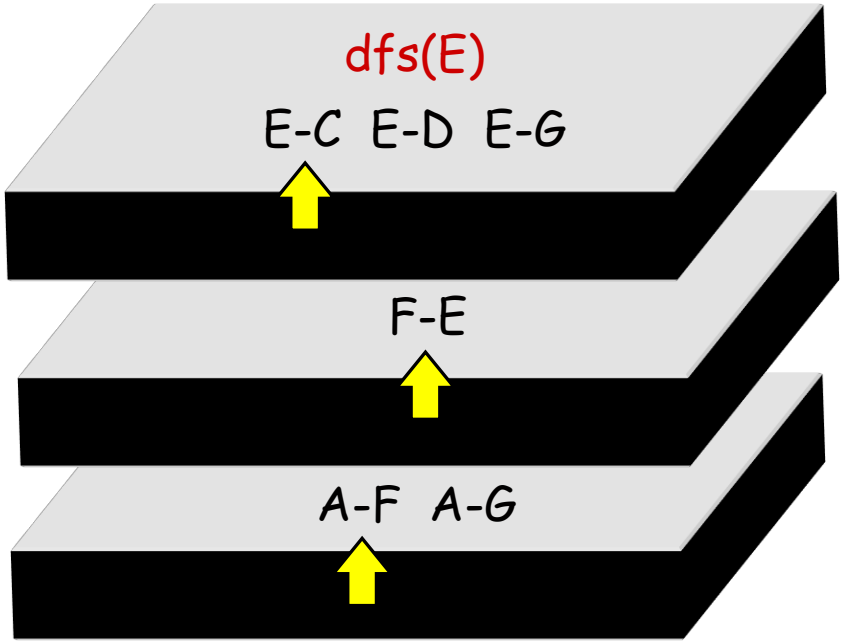


Function call stack:



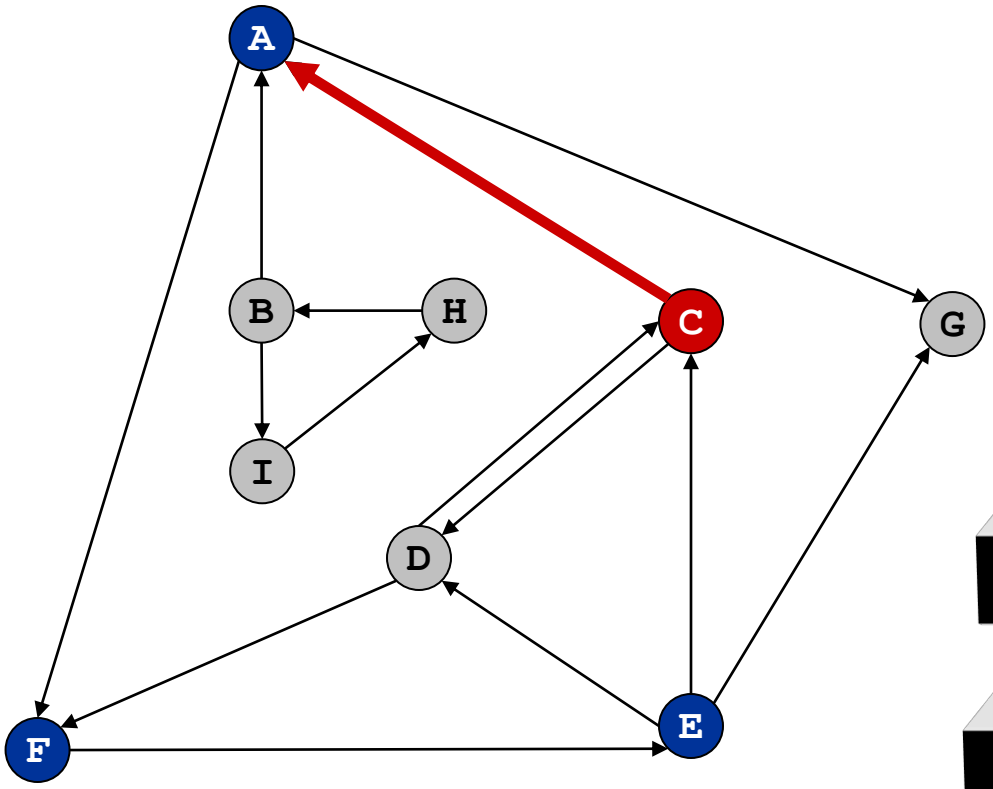
DFS算法演示

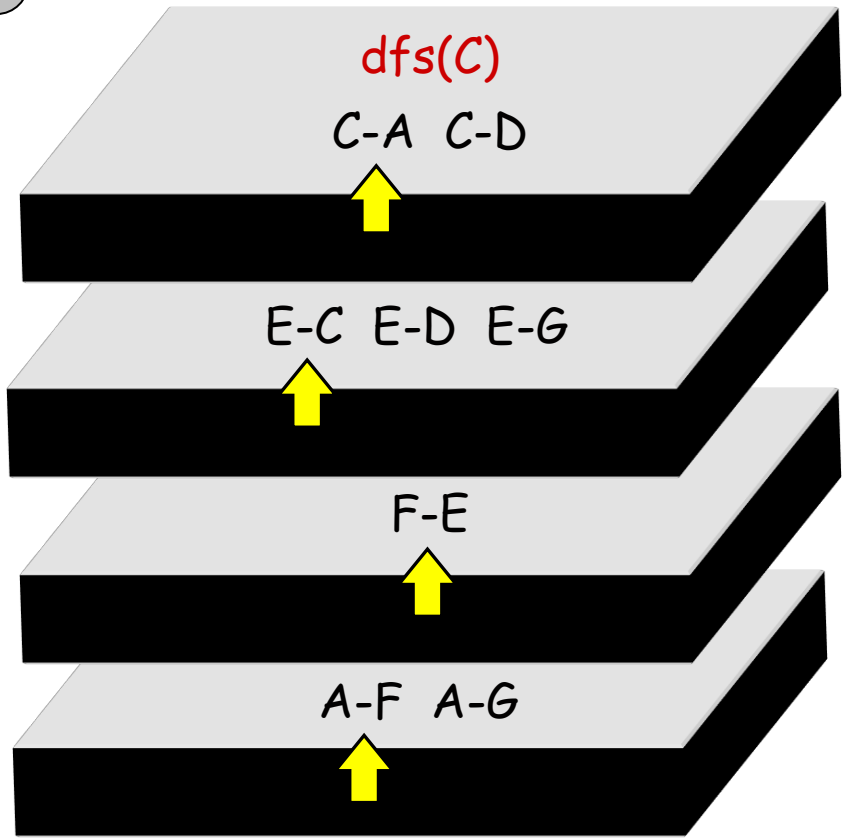




Function call stack:

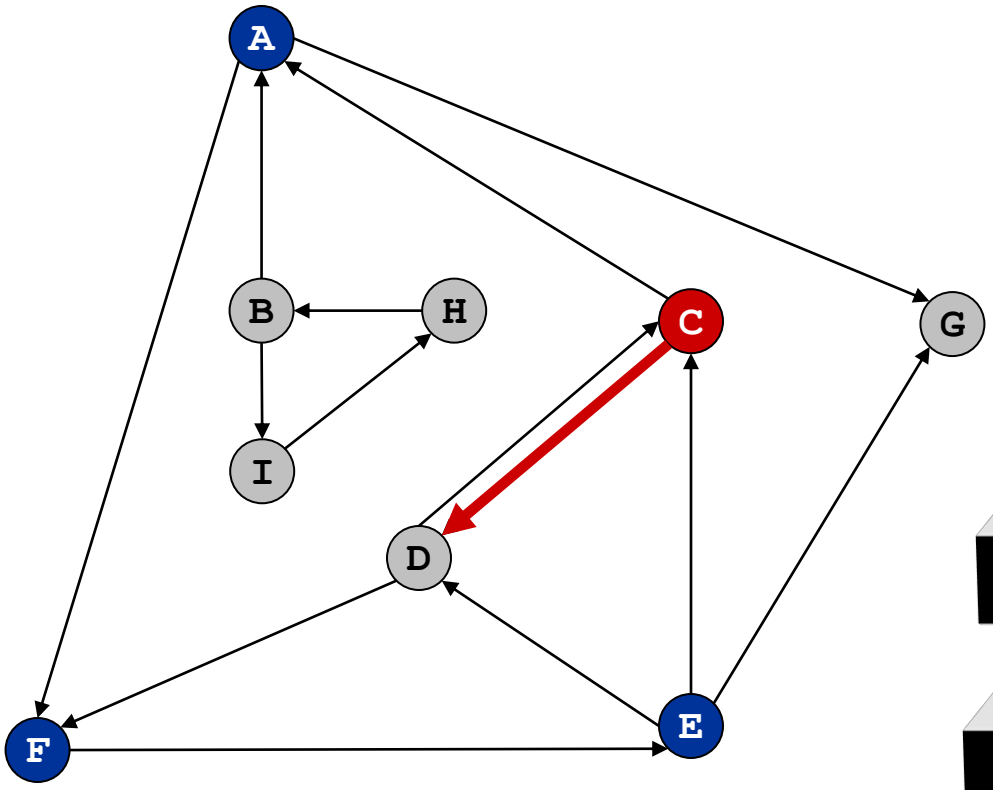
DFS算法演示

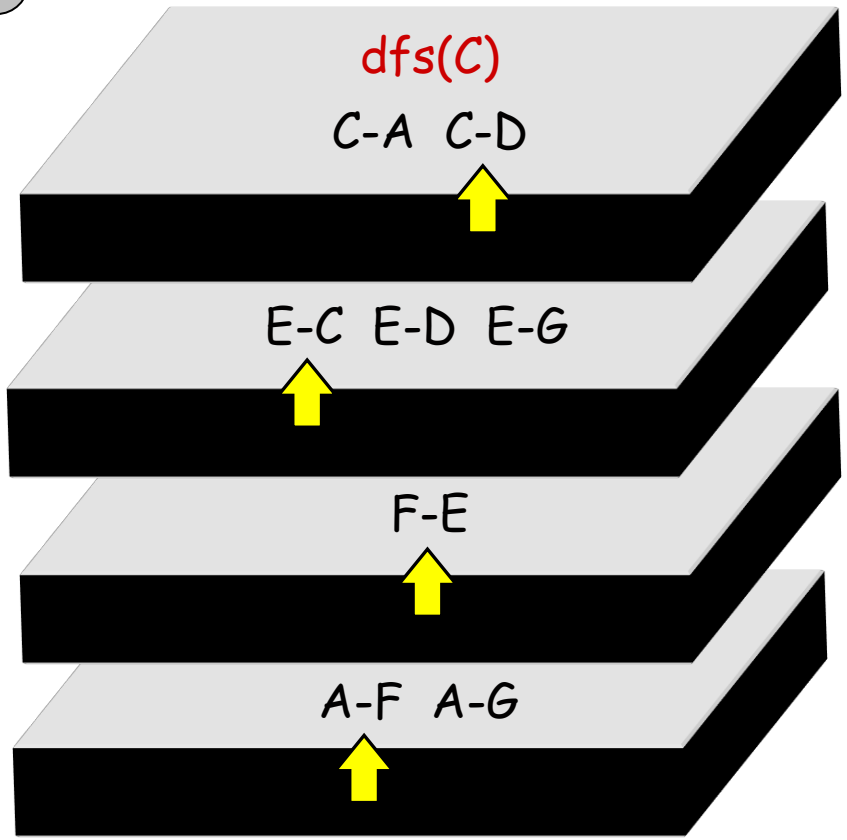




Function call stack:

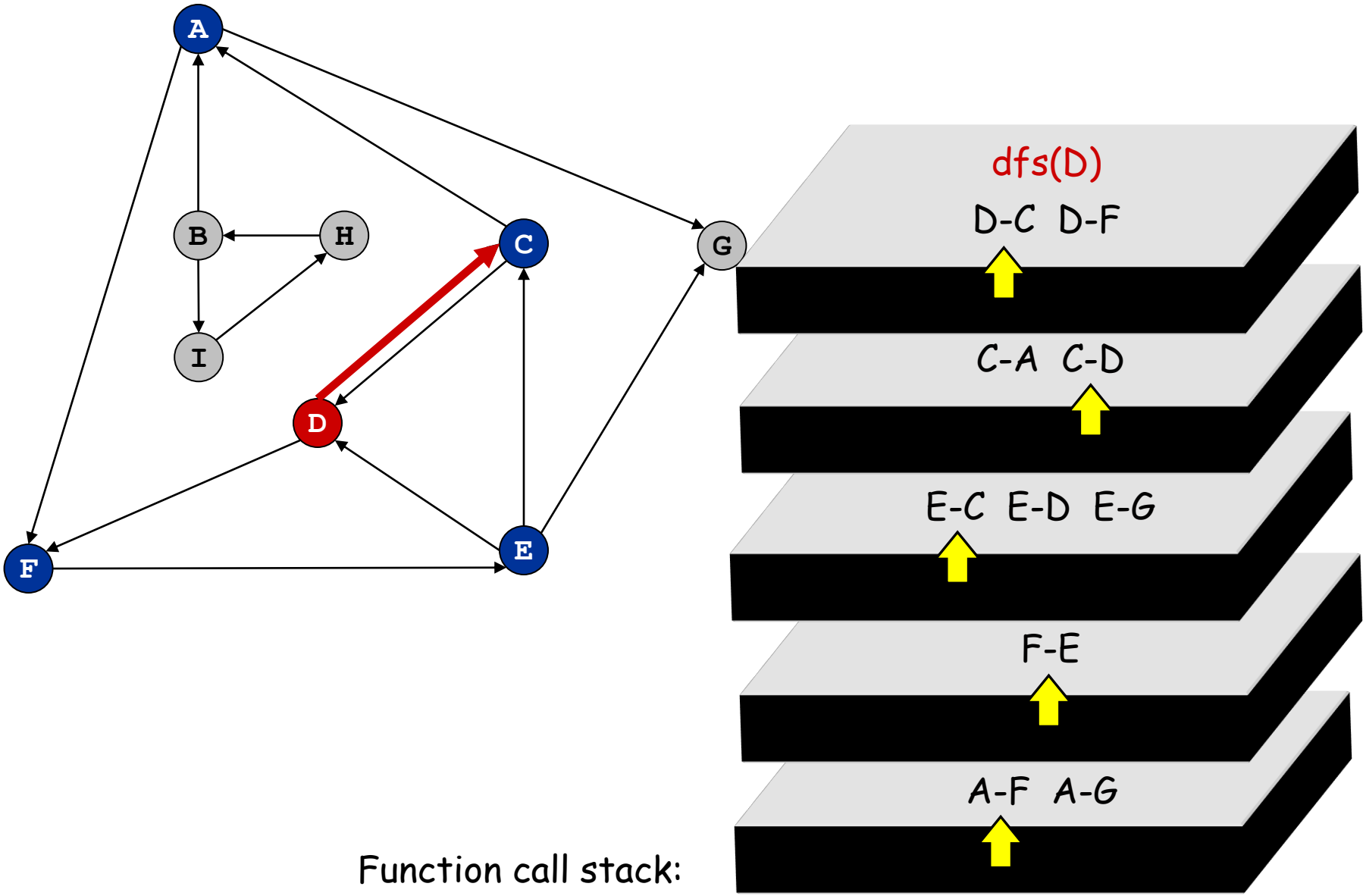
DFS算法演示



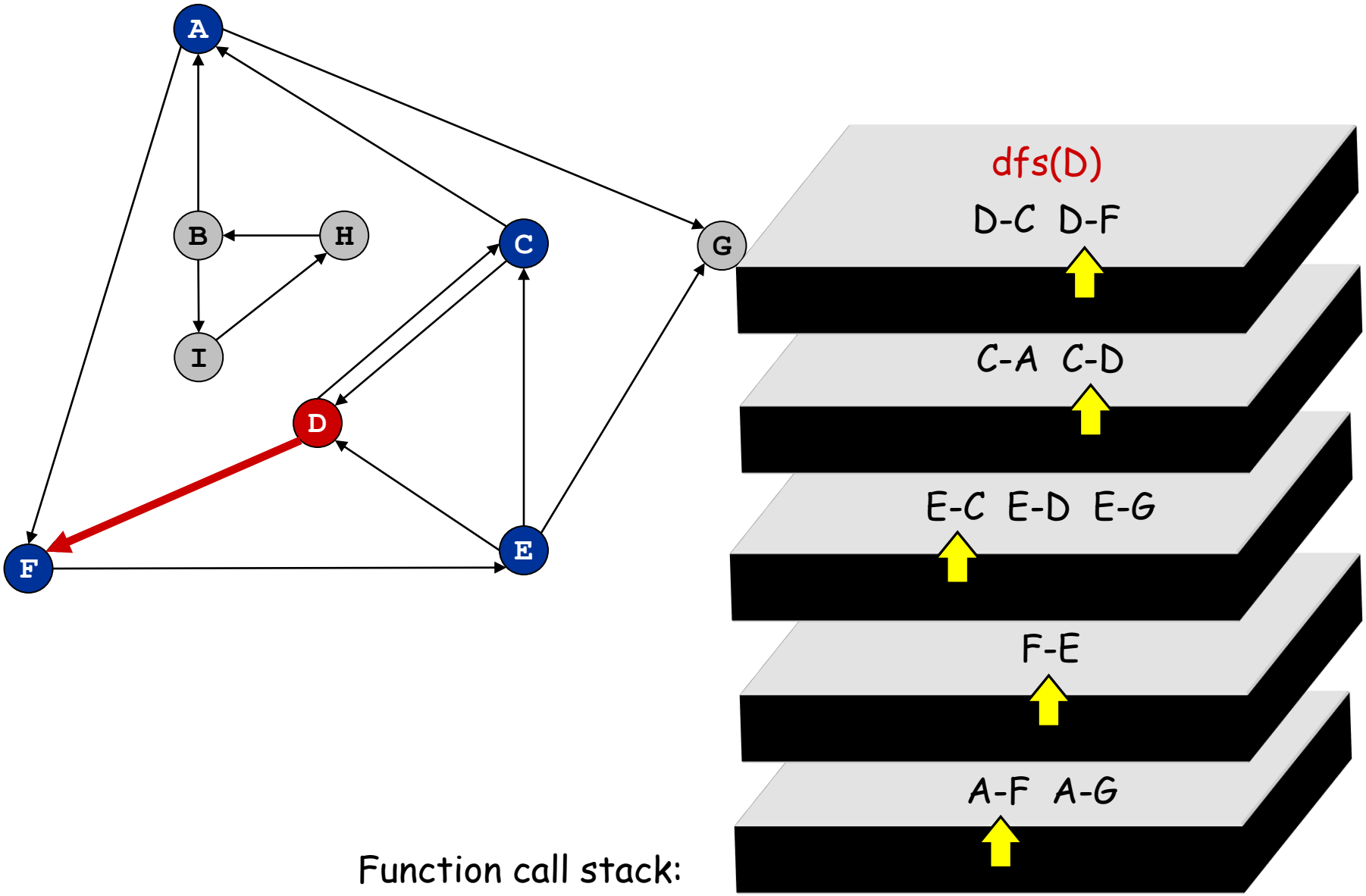


Function call stack:

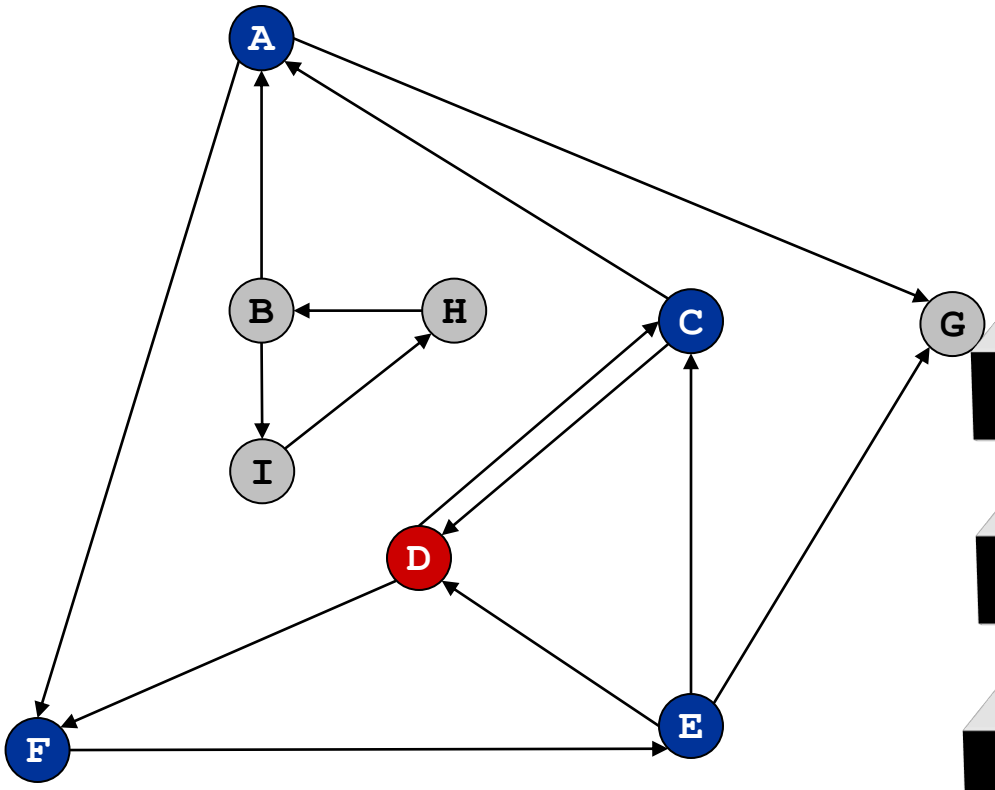
DFS算法演示

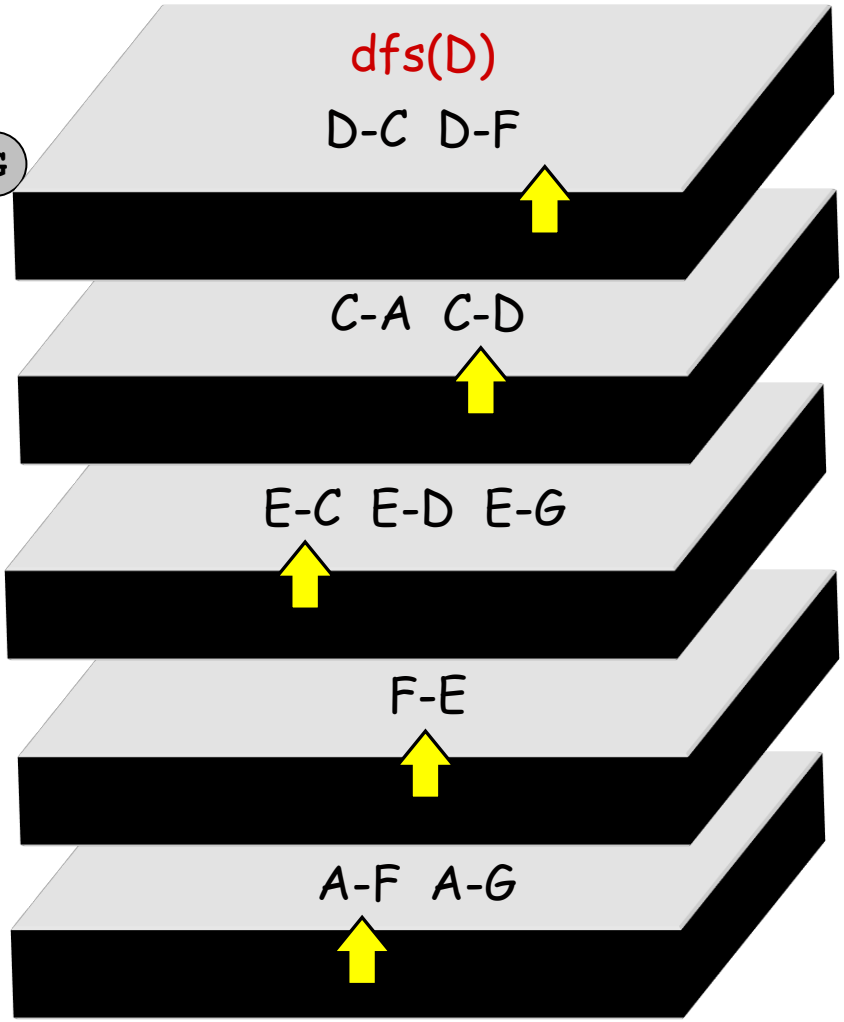

A
B
H
C
G
I
D
F
E
dfs(D)
D-C D-F
C-A C-D
E-C E-D E-G
F-E
A-F A-G
Function call stack:

DFS算法演示


A
B
H
C
G
I
D
F
E
dfs(D)
D-C D-F
C-A C-D
E-C E-D E-G
F-E
A-F A-G
Function call stack:

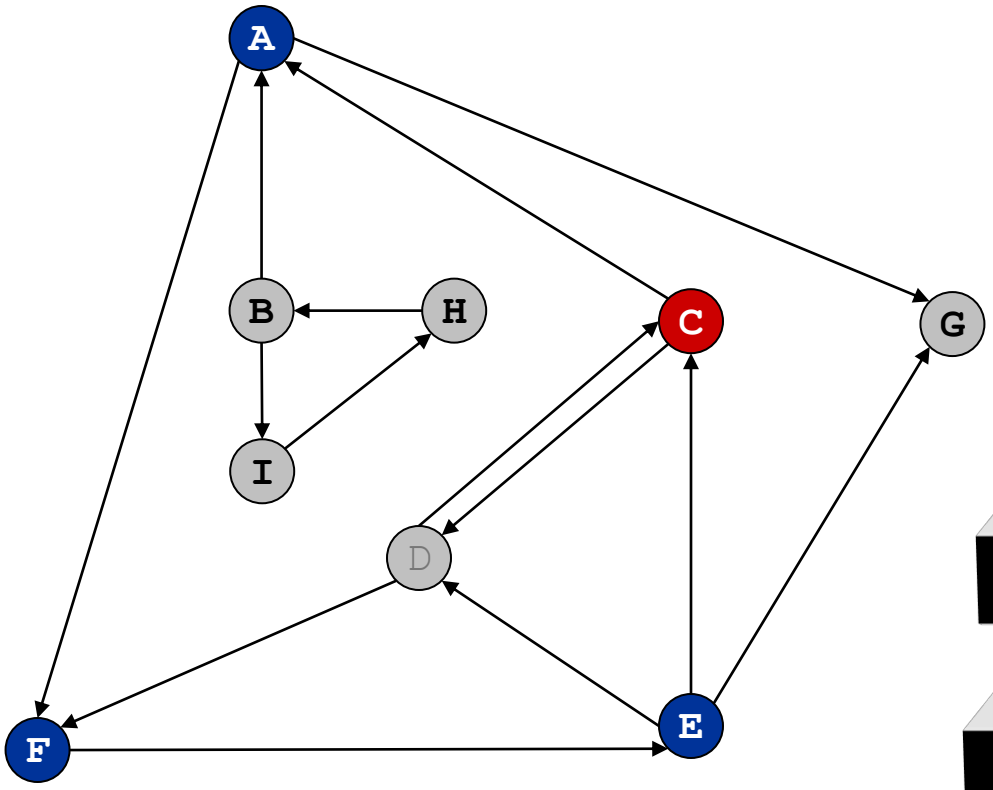
DFS算法演示

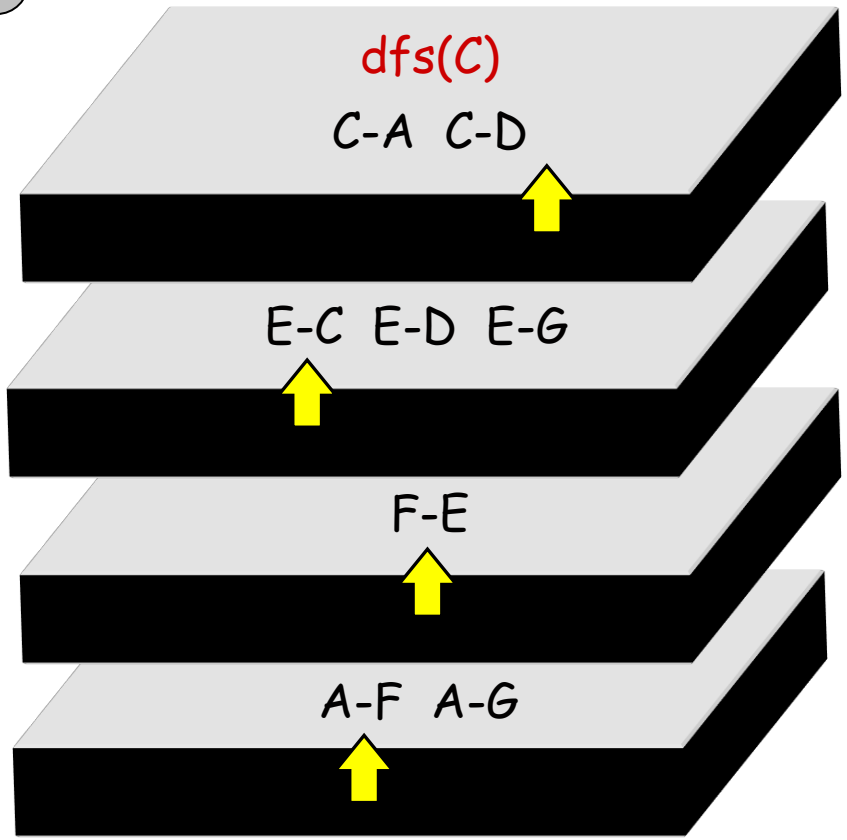




Function call stack:

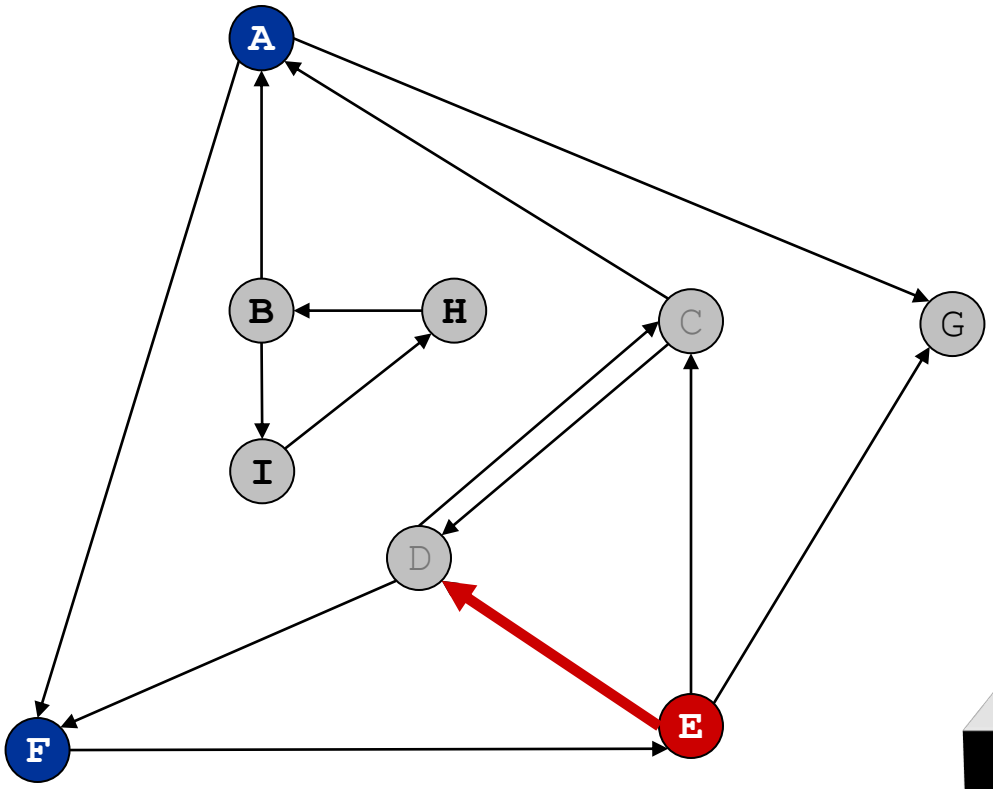
DFS算法演示

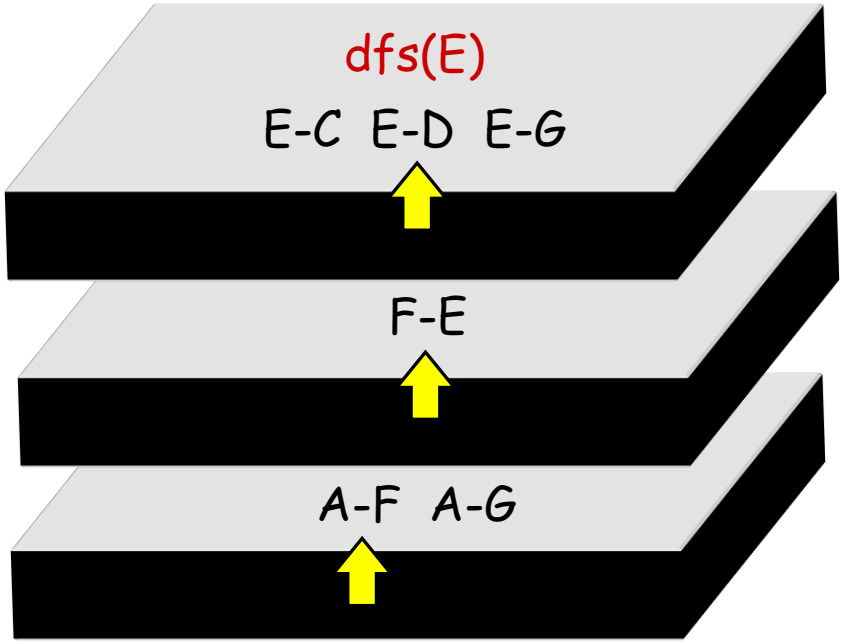




Function call stack:

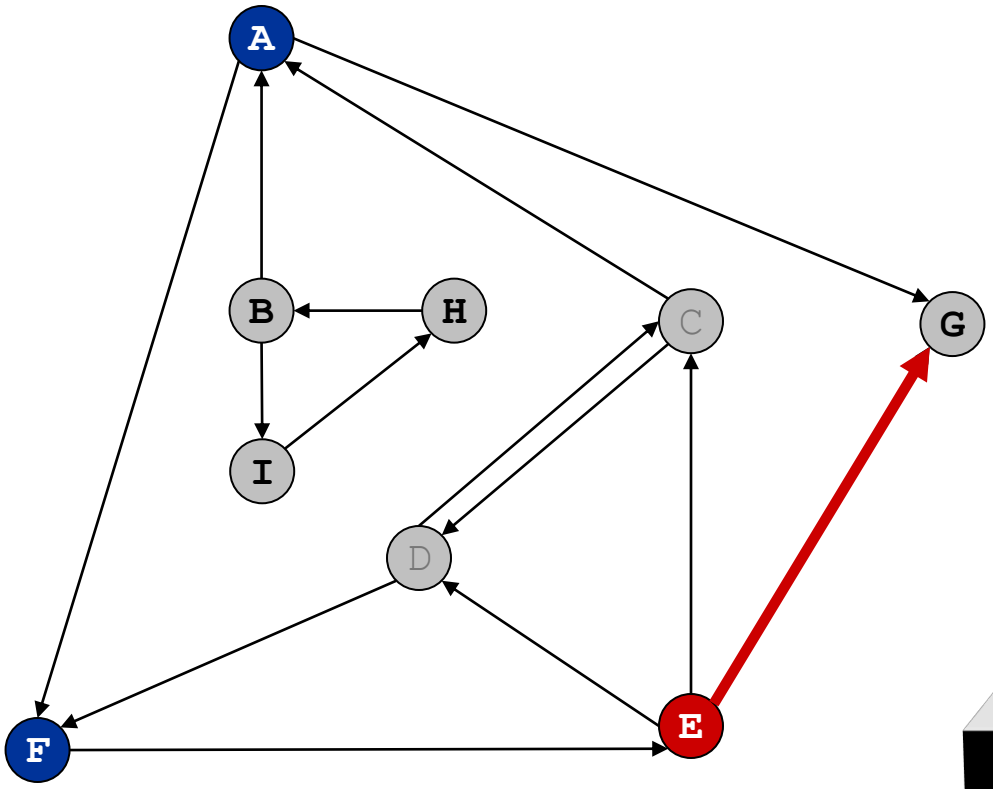
DFS算法演示

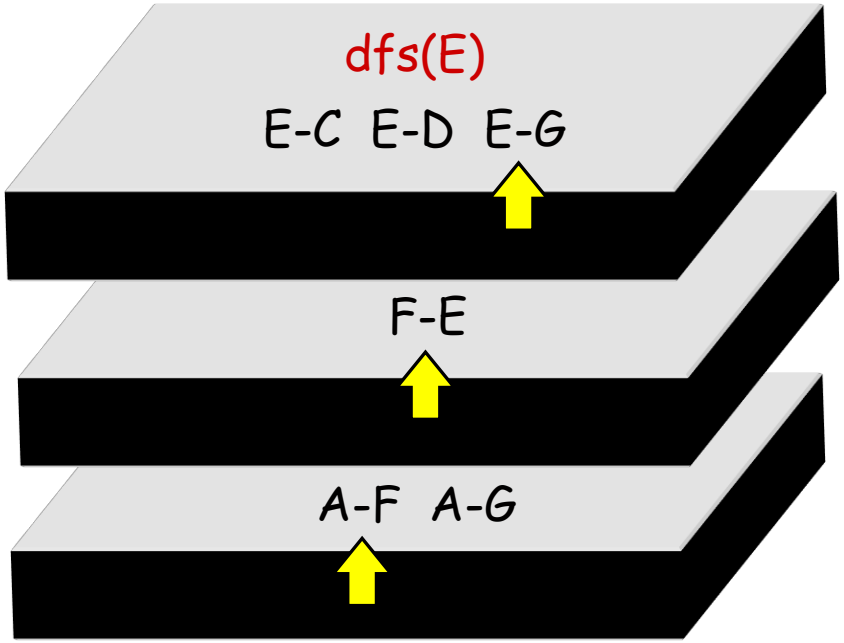




Function call stack:

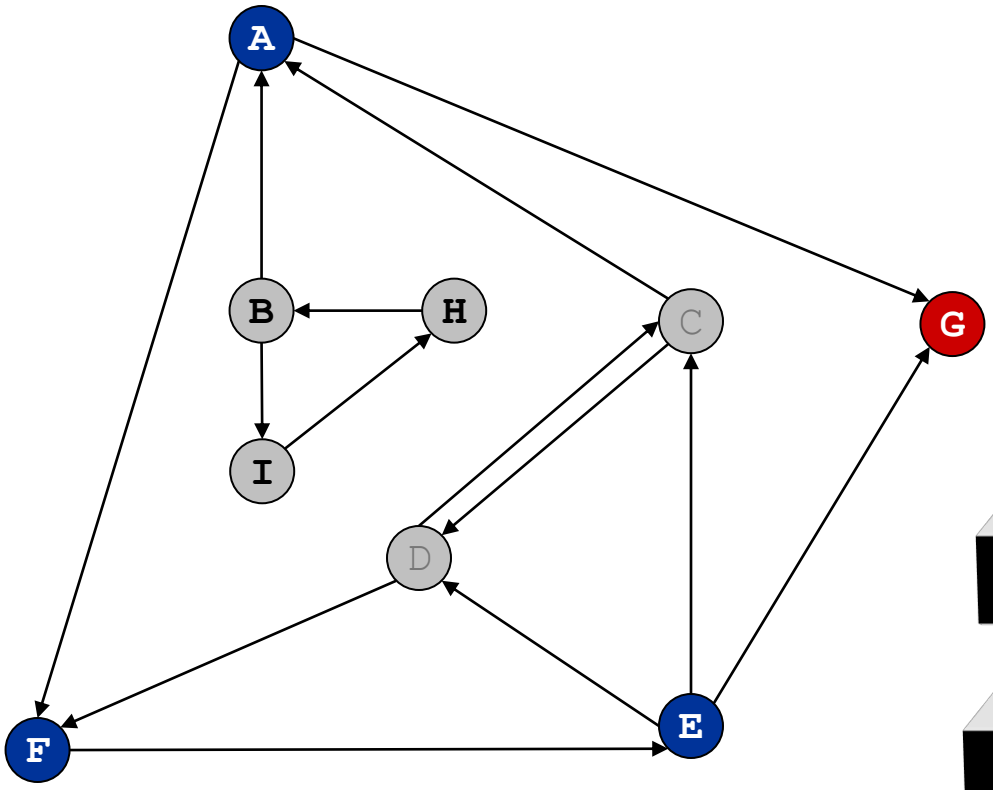
DFS算法演示

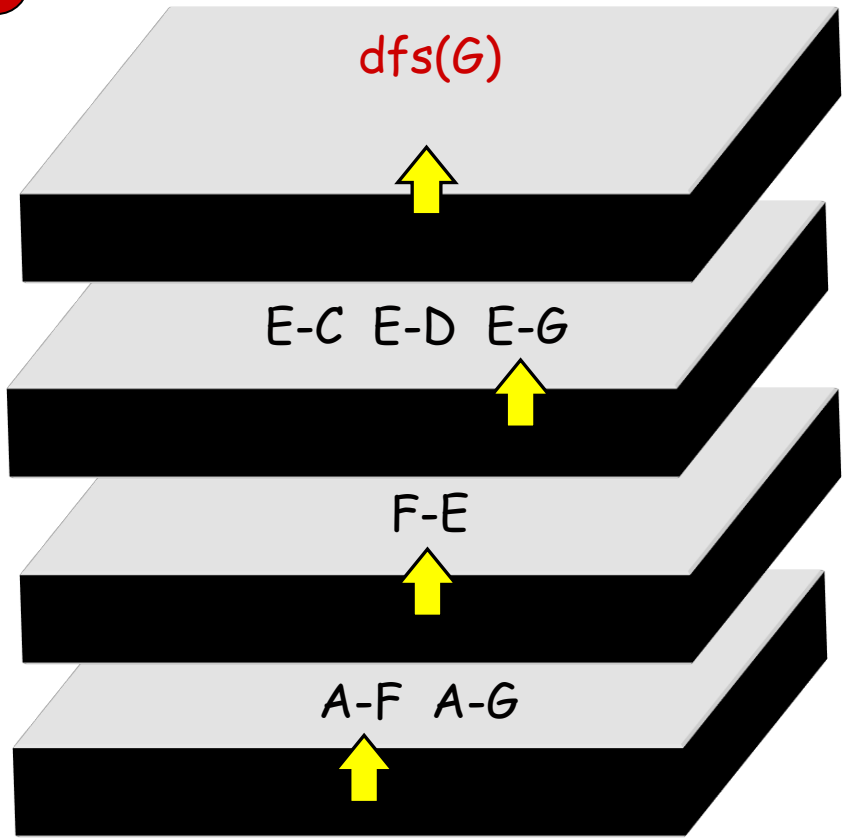




Function call stack:

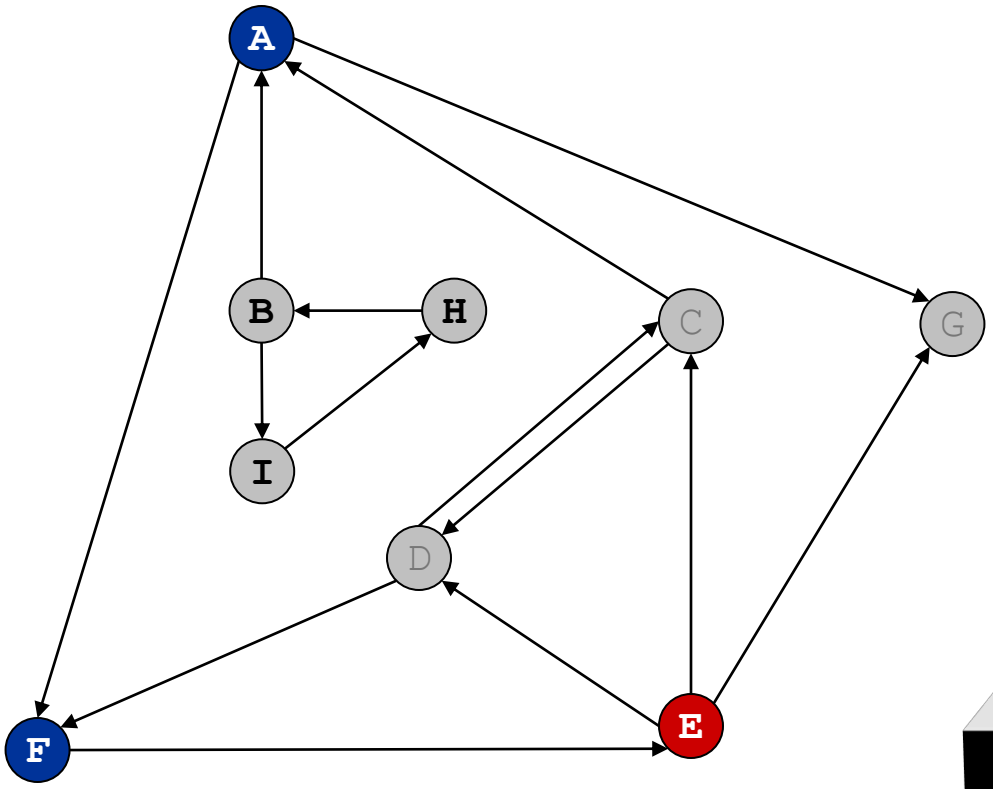
DFS算法演示

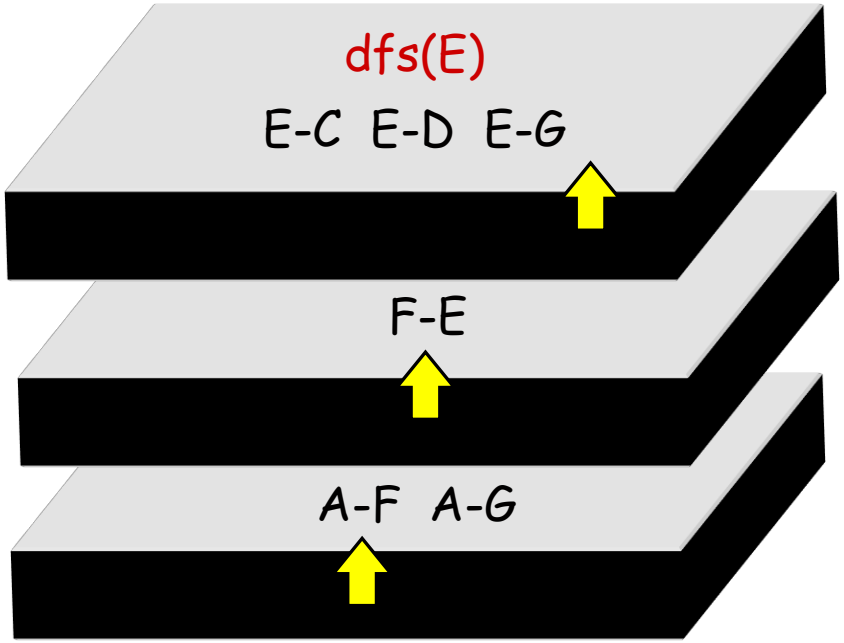




Function call stack:

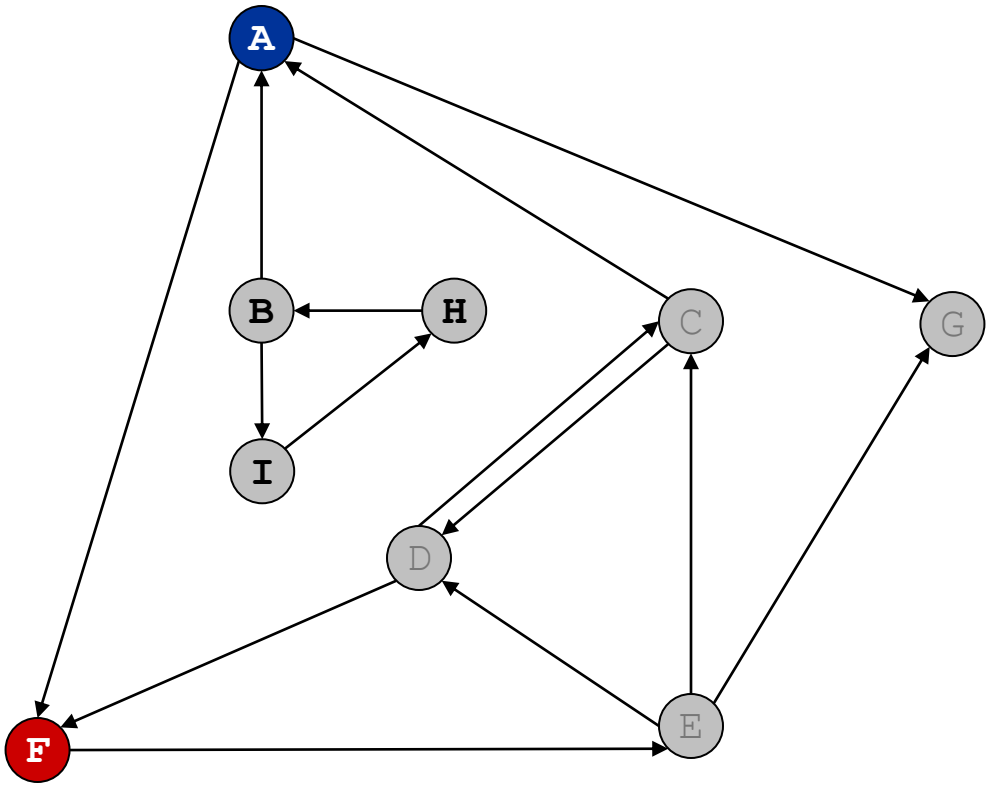
DFS算法演示



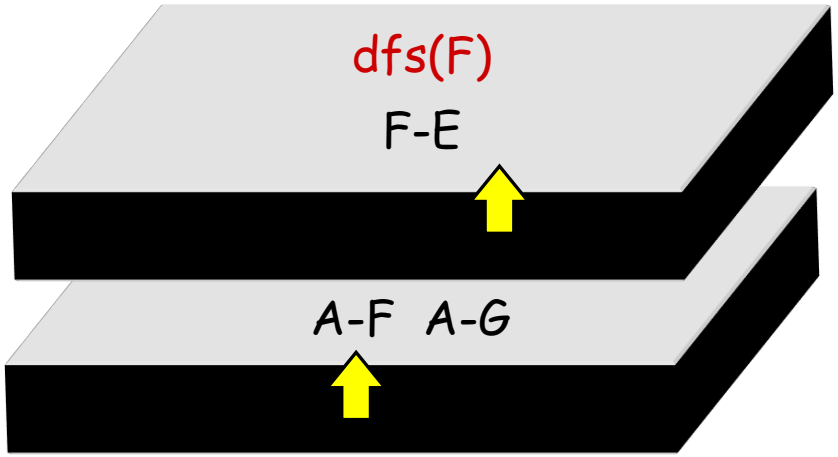


Function call stack:

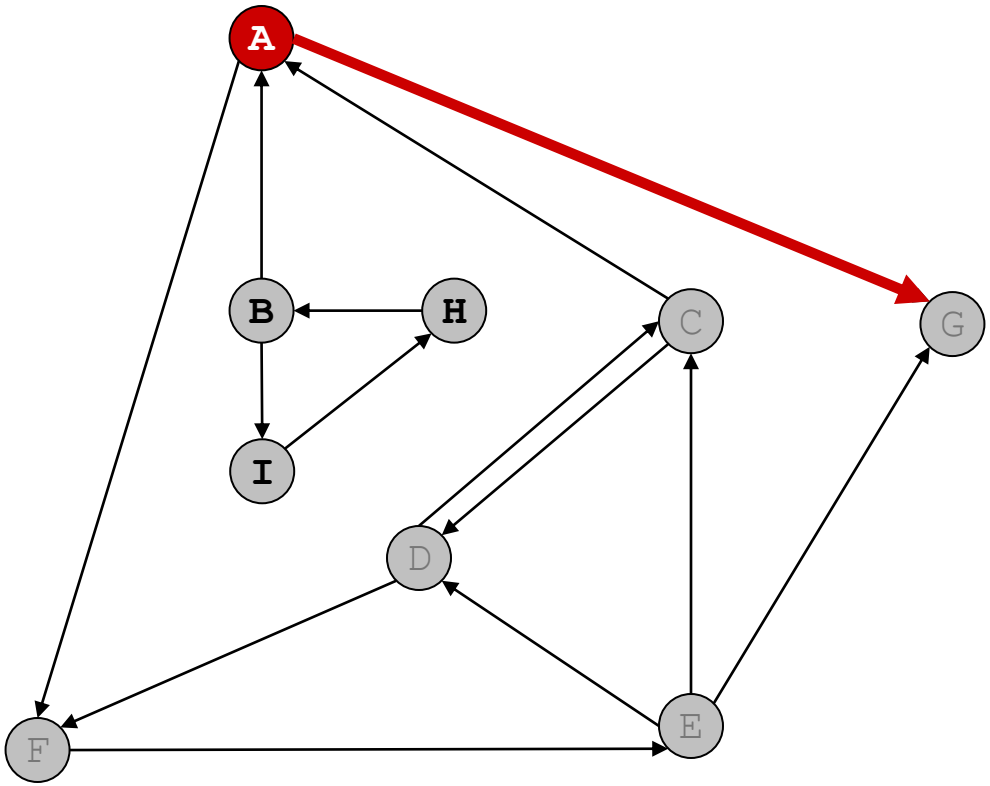
DFS算法演示



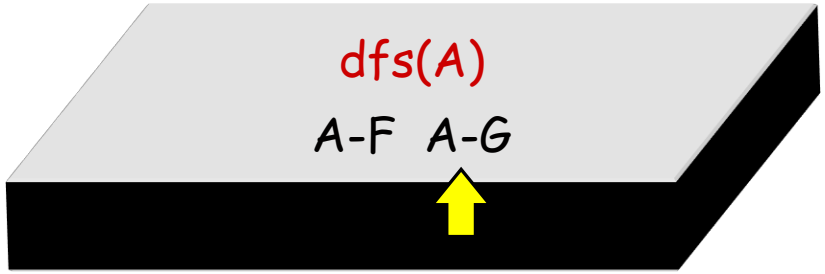
Function call stack:



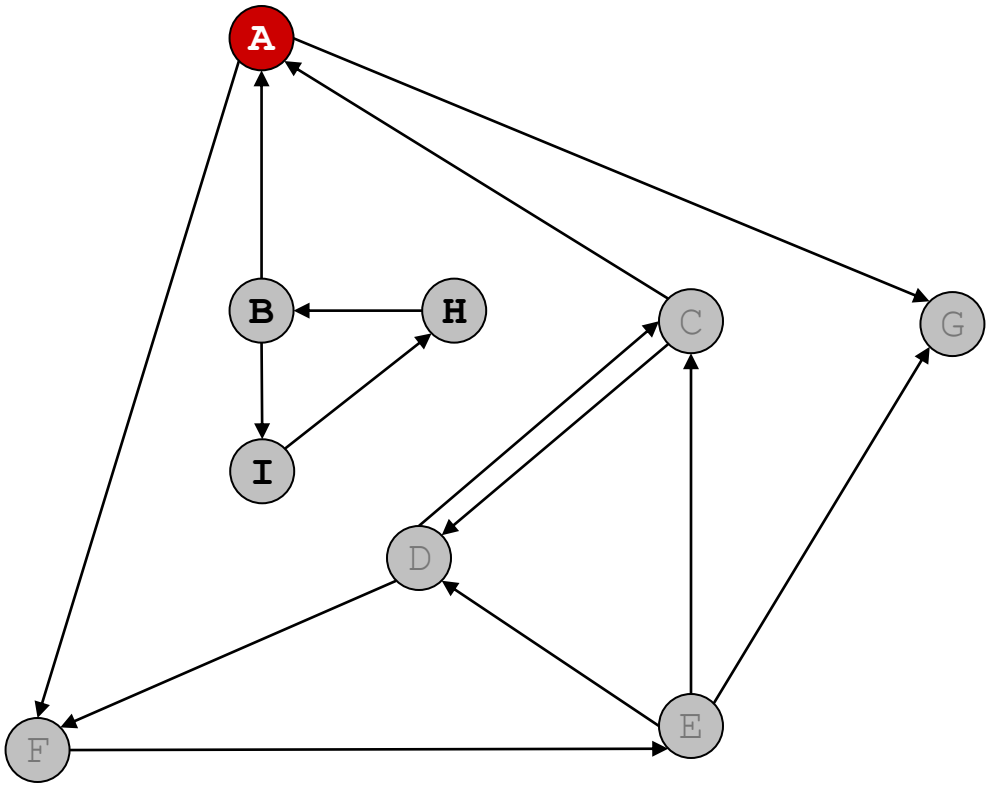
DFS算法演示



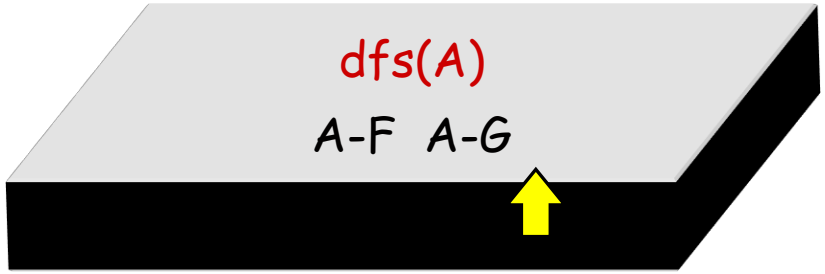
Function call stack:



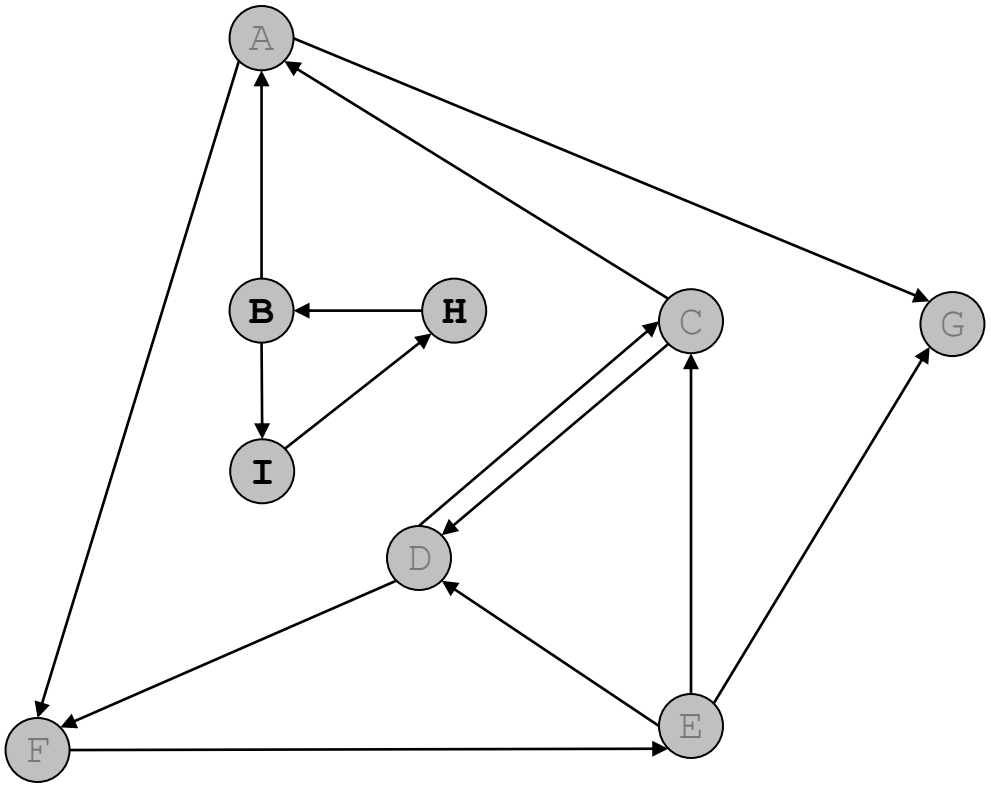
DFS算法演示



Function call stack:



DFS算法演示



Nodes reachable from A: A, C, D, E, F, G

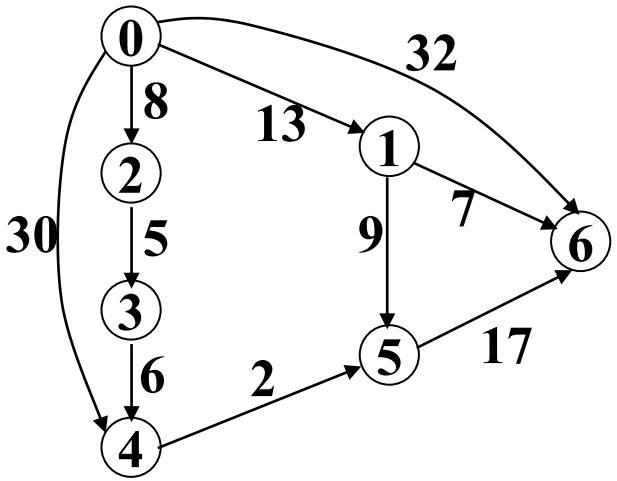
返回

10.5 最短路径

- 问题提出

用带权的有向图表示一个交通运输网，图中：
顶点——表示城市
边——表示城市间的交通联系
权——表示此线路的长度或沿此线路运输所花的时间或费用等
问题：从某顶点出发，沿图的边到达另一顶点所经过的路径中，各边上权值之和最小的一条路径——最短路径

- 单源最短路径



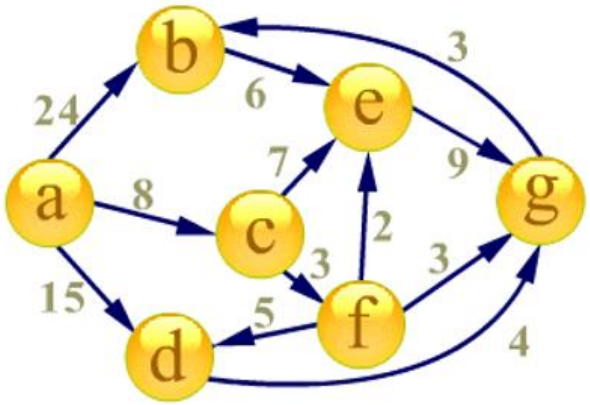
最短路径	长度
<V0,V1>	13
<V0,V2>	8
<V0,V2,V3>	13
<V0,V2,V3,V4>	19
<V0,V2,V3,V4,V5>	21
<V0,V1,V6>	20

Dijkstra算法思想

按路径长度递增次序产生最短路径算法：
把V分成两组：
（1）S：已求出最短路径的顶点的集合
（2）V-S=T：尚未确定最短路径的顶点集合
将T中顶点按最短路径递增的次序加入到S中，
保证：（1）从源点V0到S中各顶点的最短路径长度都不大于
从V0到T中任何顶点的最短路径长度
（2）每个顶点对应一个距离值
S中顶点：从V0到此顶点的最短路径长度
T中顶点：从V0到此顶点的只包括S中顶点作中间
顶点的最短路径长度

依据：可以证明V0到T中顶点Vk的最短路径，或是从V0到Vk的
直接路径的权值；或是从V0经S中顶点到Vk的路径权值之和

- 求最短路径步骤
  - 初使时令 S={V0},T={其余顶点}，T中顶点对应的距离值
    - 若存在<V0,Vi>，距离值为<V0,Vi>弧上的权值
    - 若不存在<V0,Vi>，距离值为∞
  - 从T中选取一个其距离值为最小的顶点W，加入S
  - 对T中顶点的距离值进行修改：若加进W作中间顶点，从V0到Vi的距离值比不加W的路径要短，则修改此距离值
  - 重复上述步骤，直到S中包含所有顶点，即S=V为止



$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \end{array} \begin{pmatrix} 0 & 24 & 8 & 15 & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty & 6 & \infty & \infty \\ \infty & \infty & 0 & \infty & 7 & 3 & \infty \\ \infty & \infty & \infty & 0 & \infty & \infty & 4 \\ \infty & \infty & \infty & \infty & 0 & \infty & 9 \\ \infty & \infty & \infty & 5 & 2 & 0 & 10 \\ \infty & 3 & \infty & \infty & \infty & \infty & 0 \end{pmatrix}$$

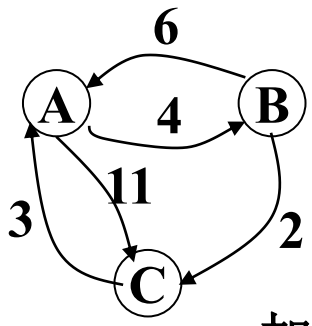
- 算法实现
  - 图用带权邻接矩阵存储a[ ][ ]
  - 数组dist[]存放当前找到的从源点V0到每个终点的最短路径长度，其初态为图中直接路径权值
  - 数组pre[]表示从V0到各终点的最短路径上，此顶点的前一顶点的序号；若从V0到某终点无路径，则用0作为其前一顶点的序号
- 算法描述


Dijkstra.txt

- 算法分析：$T(n)=O(n^2)$

- 所有顶点对之间的最短路径
  - 方法一：每次以一个顶点为源点，重复执行Dijkstra算法
    n次—— $T(n)=O(n^3)$
  - 方法二： Floyd算法
    - 算法思想：逐个顶点试探法
    - 求最短路径步骤
      - 初始时设置一个n阶方阵，令其对角线元素为0，若存在弧<Vi,Vj>，则对应元素为权值；否则为∞
      - 逐步试着在原直接路径中增加中间顶点，若加入中间点后路径变短，则修改之；否则，维持原值
      - 所有顶点试探完毕，算法结束

例



初始：
$$\begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{pmatrix}$$

路径：

	AB	AC
BA		BC
CA		

$$path=\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

加入A：
$$\begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix}$$

路径：

	AB	AC
BA		BC
CA	CAB	

$$path=\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

加入B：
$$\begin{pmatrix} 0 & 4 & 6 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix}$$

路径：

	AB	ABC
BA		BC
CA	CAB	

$$path=\begin{pmatrix} 0 & 0 & 2 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

加入C：
$$\begin{pmatrix} 0 & 4 & 6 \\ 5 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix}$$

路径：

	AB	ABC
BCA		BC
CA	CAB	

$$path=\begin{pmatrix} 0 & 0 & 2 \\ 3 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

- 算法实现
  - 图用邻接矩阵存储
  - 二维数组c[ ][ ]存放最短路径长度
  - path[i][j]是从Vi到Vj的最短路径上Vj前一顶点序号
- 算法描述


Floyd. txt

这样看来， Floyd算法似乎没带来更多的好处？？！

- 算法分析：$T(n)=O(n^3)$

实际上，从实现代码来看：
Floyd算法的代码比用Dijkstra算法要简明得多！！！

返回章目录