图的遍历算法

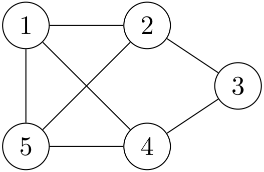
Jun Wu

wujun@yzu.edu.cn

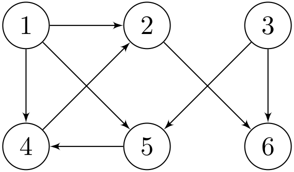
April 11, 2018

图的定义

- 图$G=(V,E)$，即图有两个要素顶点和边。$E\subseteq V\times V$。
- 若边$(u,v)=(v,u)$，则G为无向图。
- 若边$(u,v)\neq(v,u)$，则G为有向图。
- 简单图：无自环，无平行边。



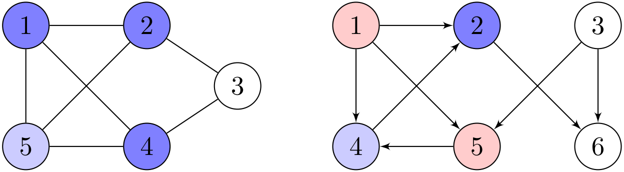
(a) 无向图



(b) 有向图

图: 无向图和有向图的例子

- 顶点u的邻居定义为$\{v|(u,v) \in E\}$，记作$Adj(u)$。
- 对于有向图，$Adj(u)$又称为前向邻居。
  后向邻居$Adj^{-}(u) = \{v|(v,u) \in E\}$。
-



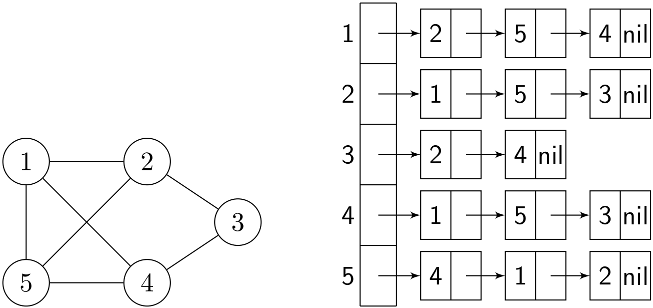
(a) $Adj(5) = \{1,2,4\}$

(b) $Adj(4) = \{2\}$, $Adj^{-}(4) = \{1,5\}$

图: 邻居的例子

邻接表

- 对于每个$u \in V$，用链表将$Adj(u)$串起来；
- 为每个链表建一个链表头；以一维数组存放$|V|$个链表头。
- 存储复杂度$O(|V| + |E|)$。



(a) 无向图 (b) 邻接表

图: 无向图的邻接表

有向图的邻接表

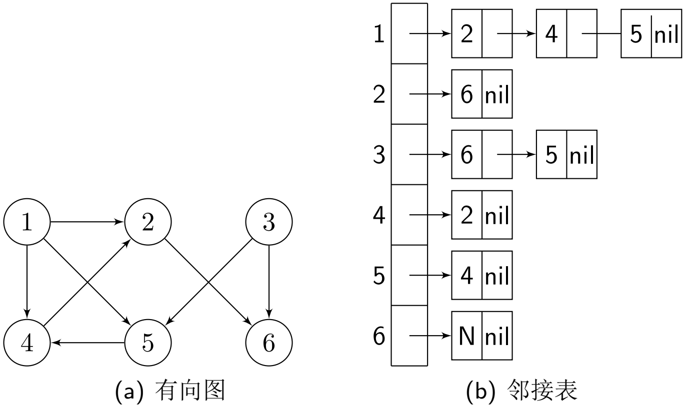
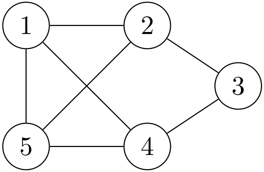


图: 有向图的邻接表

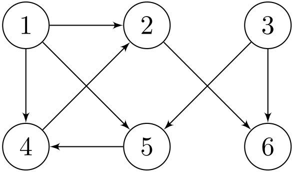
邻接矩阵

- 图$G=(V,E)$, $|V|\times|V|$矩阵A称为G的邻接矩阵，当且仅当

$$a_{ij}=\begin{cases}1, & if\ (i,j)\in E\\ 0, & otherwise\end{cases}$$



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



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

1 图的表示

2 广度优先搜索及应用
- BFS算法
- BFS的应用

3 深度优先搜索及应用

算法思路:

- 算法可以从图中任意点s开始;
- 算法先反问s，然后访问$Adj(s)$中的点;
- 设$Adj(s) = \{v_1, v_2, \cdots, v_k\}$,
- 算法接着依次访问$Adj(v_1), Adj(v_2), \cdots, Adj(v_k)$.

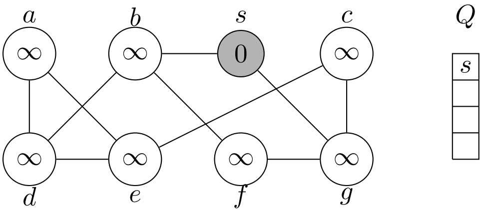
数据结构:

- 采用队列结构;
- 白色顶点：未被访问;
- 灰色顶点：已被访问，但尚有邻居未被访问;
- 黑色顶点：已被访问并且所有邻居也均已被访问。

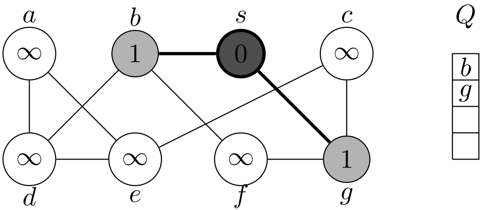
BFS(G,s)

```
for each vertex u ∈ V − {s} do
    color[u] ← White
    d[u] ← ∞
    π(u) ← nil
end for
color[s] ← Gray
d[s] ← 0
π(s) ← nil
Q ← ∅
Enqueue(Q, s)
while Q ≠ ∅ do
    u ← Dequeue(Q)
    for each v ∈ Adj[u] do
        if color[v] = White then
            color[v] ← Gray
            d[v] ← d[u] + 1
            π(v) ← u
            Enqueue(Q, v)
        end if
    end for
    color[u] ← Black
end while
```

BFS的例子

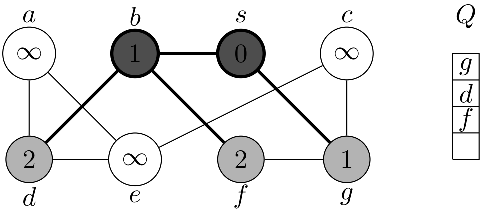


(a) 初始化

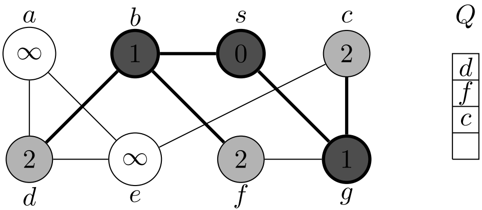


(b) 访问$Adj(s)$后

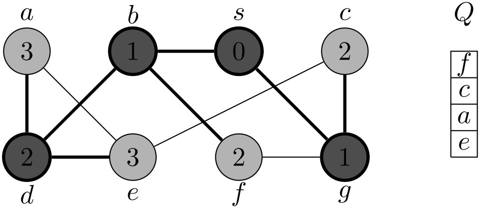
BFS的例子con't 1



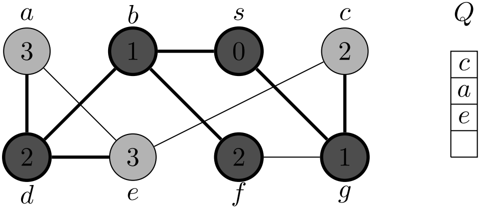
(c) 访问$Adj(b)$后



(d) 访问$Adj(g)$后

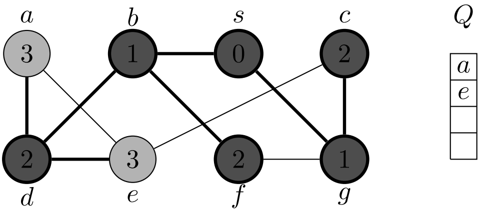


(e) 访问$Adj(d)$后

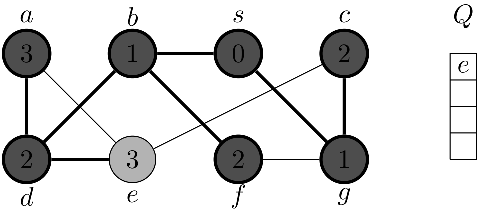


(f) 访问$Adj(f)$后

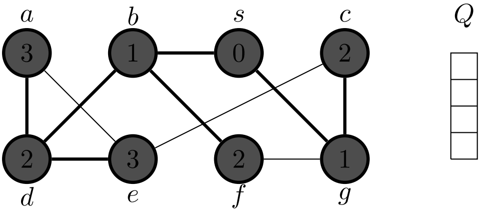
BFS的例子con't 3



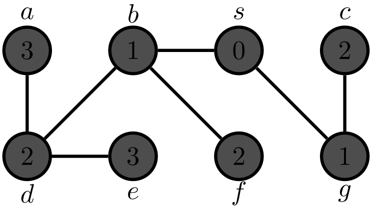
(g) 访问$Adj(c)$后



(h) 访问$Adj(a)$后



(i) 访问$Adj(e)$后



(j) 距离树

图: BFS实例

BFS算法分析

距离树：

- 子图$T=(V,E')$，其中$E'=\{(u,v)\in E|u=\pi(v)\}$是图$G$的以$s$为根的生成树。
- 对于任意$u\in V$，$d(u)$是图G中从s到u的最短路的长度（不加权）。
- 因此，T也被称为距离树。

复杂度：

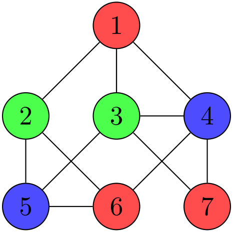
- 每个顶点$u\in V$进入Q一次且仅一次。
- 每个顶点$u\in V$出Q时将访问$Adj(u)$中的点。
- 因此，算法总时间为$O(\sum_{u\in V}|Adj(u))=O(|E|)$。

无向图的二着色问题

Definition 1 (k-着色问题)

实例：图$G=(V,E)$和颜色集C，$|C|=k$。
问题：是否存在顶点V的一个着色，即$c:V\to C$，使得对任意$(u,v)\in E$有$c(u)\neq c(v)$?

- 许多任务调度、资源分配问题可以用着色问题来建模。如排课。
- $k\geq 3$时，着色问题是NP-完全的。



图：一个三着色的例子

Two-colorability(G, s)

for each vertex $u \in V$ **do**
    color$[u] \leftarrow$ White
end for
$Q \leftarrow \emptyset$
color$[s] \leftarrow$ Red
Enqueue(Q, s)
while $Q \neq \emptyset$ **do**
    $u \leftarrow$ Dequeue(Q)
    for each $v \in Adj[u]$ **do**
        if color$[v] =$ White **then**
            color$[v] \leftarrow$ **not**(color$[u]$)
            Enqueue(Q, v)
        else
            if color$[v]$ $=$color$[u]$ **then**
                return(not 2-colorable)
            end if
        end if
    end for
end while
return(2-colorable)

2-着色算法的正确性

如果算法返回2-colorable：

- 每条边被访问可能有两种情况：
- case 1：第10行，两个端点被着成了不同颜色；
- case 2：第13行，两个端点有不同颜色，否则算法将返回not 2-colorable；
- 因此，数组color中存放了一个着色方案。

如果算法返回not 2-colorable：

- $(u, v) \in E \wedge$ color(u) =color(v)；
- 那么将存在从s到u和s到v的红兰交错路；
- 这蕴含了图中存在奇数长度的圈；
- 而所有奇数长的圈都是not 2-colorable。

1. 图的表示

2. 广度优先搜索及应用

3. 深度优先搜索及应用
   - DFS算法
   - DFS算法分析
   - DFS算法应用

深度优先搜索策略

算法思路：

- 从图G的某个顶点s开始；
- 然后访问$Adj(s)$的某个邻居v，弃其它邻居不顾；
- 当v的访问完成后($Adj(v)$中的顶点都被访问后)，再返回s，这个动作称为回溯；
- 接着再访问s的下一个邻居；当所有$Adj(s)$中的顶点访问完后，算法结束。

相关定义：

- 未被访问的顶点为白色；已被访问但未完成的为灰色；已完成的为黑色；
- 时间戳：
- 发现时刻：顶点v首次被访问的时间$d(v)$；
- 完成时刻：顶点v的所有访问完成的时刻$f(v)$；
- 共有$1, 2, \cdots, 2n$个时刻。

DFS的递归算法

DFS(G)

for each vertex $u \in V$ **do**
  color$[u] \leftarrow$ White
  $\pi[u] \leftarrow nil$
end for
time$\leftarrow 0$
for each vertex $u \in V$ **do**
  if color$[u]$ = White **then**
    DFS-Visit(u)
  end if
end for

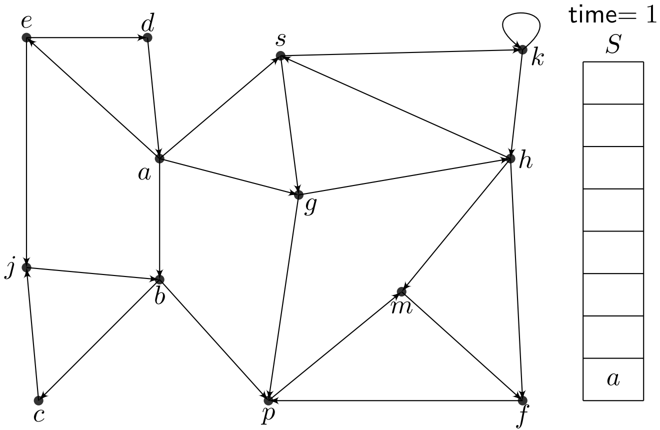
DFS-Visit(s)

color$[s] \leftarrow$ Gray
time$\leftarrow$time$+1$
$d[s] \leftarrow$ time
for each $v \in Adj[s]$ **do**
  if color$[v]$ = White **then**
    $\pi(v) \leftarrow s$
    DFS-Visit(v)
  end if
end for
color$[s] \leftarrow$ Black
$f[s] \leftarrow$time$\leftarrow$time$+1$

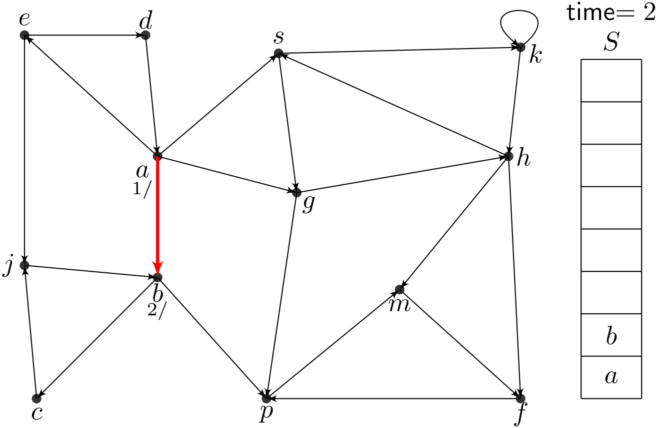
- 时间复杂度：$O(|V| + |E|)$。

DFS-Visit(s)–非递归算法

```
color[s] ← Gray
time ← time+1
d(s) ← time
S ← ∅
Push(S, s)
while S ≠ ∅ do
    u ← Top(s)
    v ← u's next neighbor in Adj(u)
    if v = nil then
        color(u) ← Black
        Pop(S)
        time ← time+1
        f(u) ← time
    else
        if color(v) = White then
            color(v) ← Gray
            time ← time+1
            d(v) ← time
            π(v) ← u
            Push(S, v)
        end if
    end if
end while
```

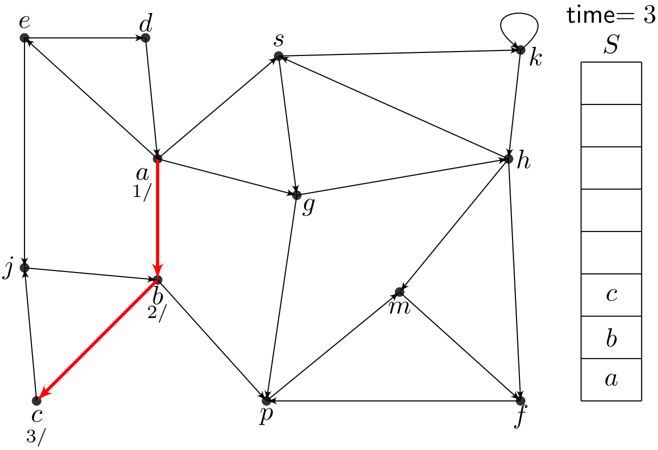


(a)

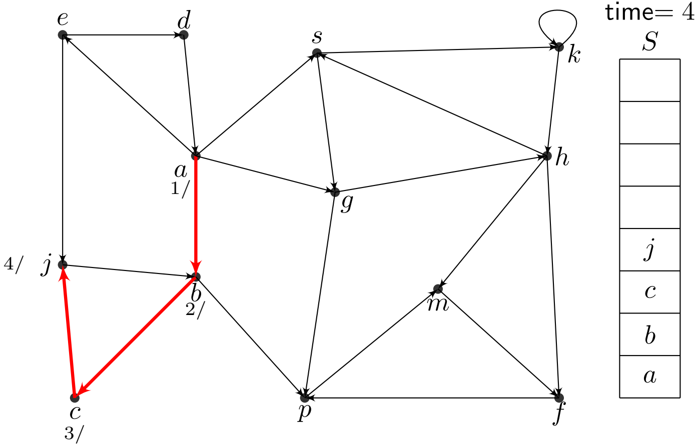


(b)

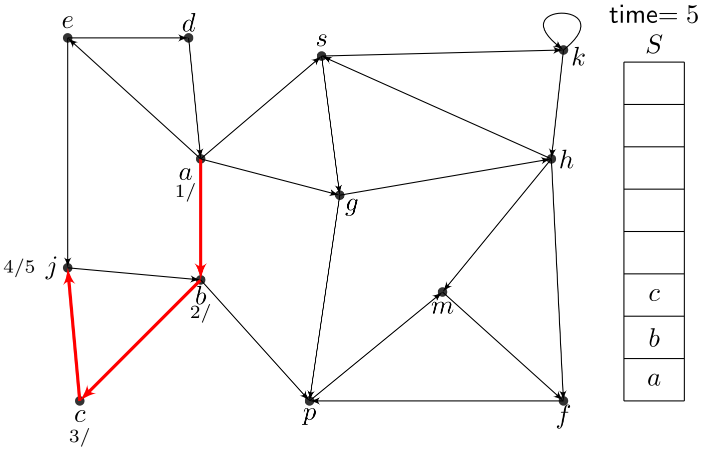
深度优先搜索实例



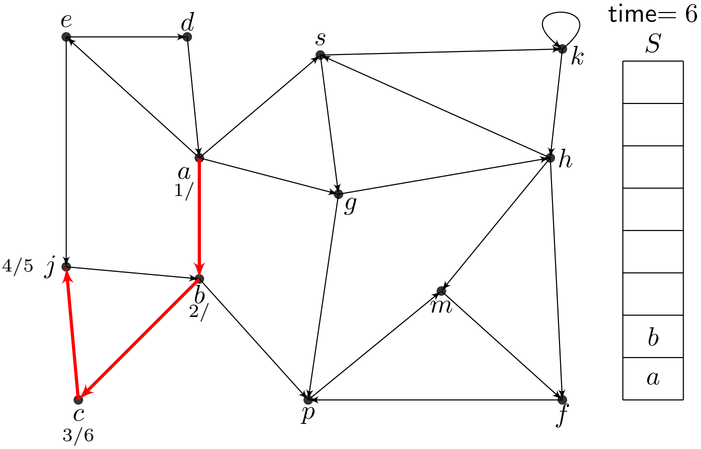
(c)



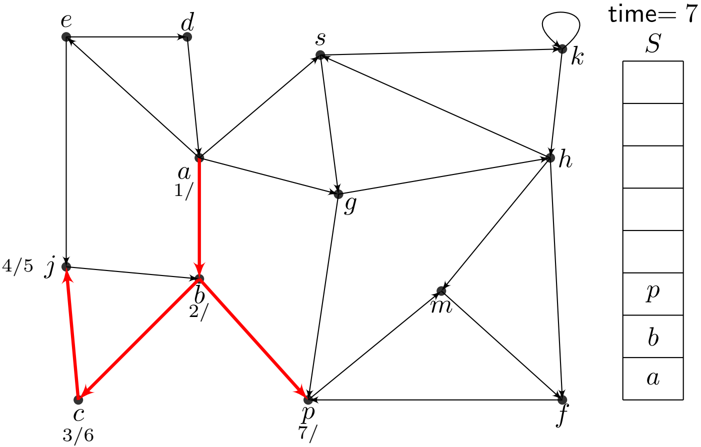
(d)



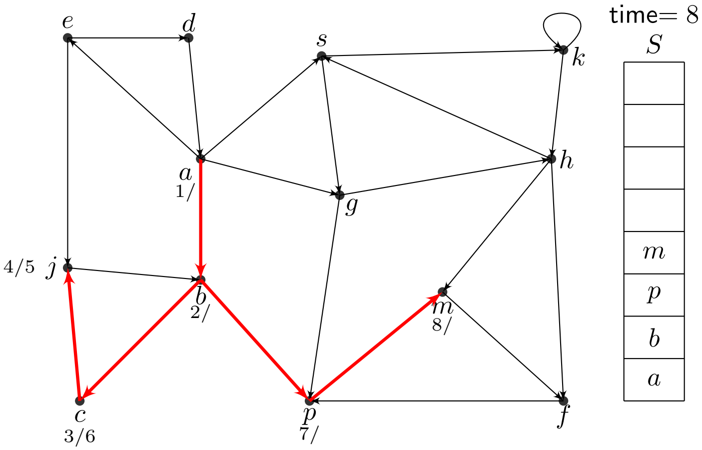
(e)



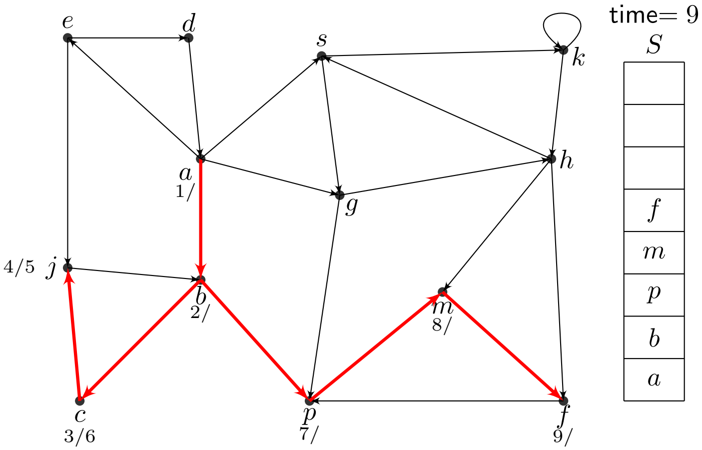
(f)



(g)

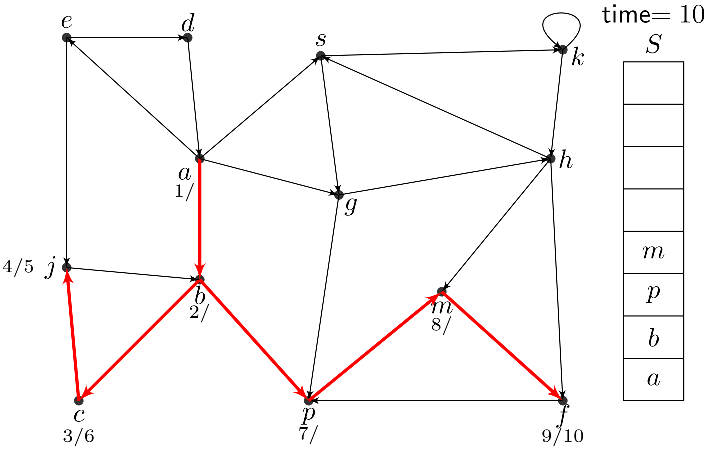


(h)

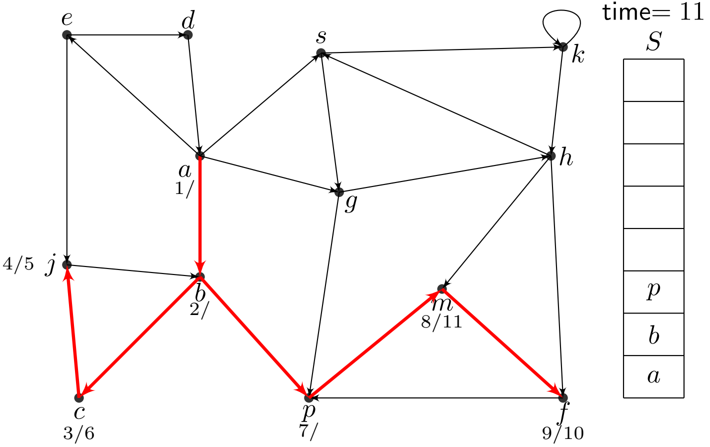


(i)

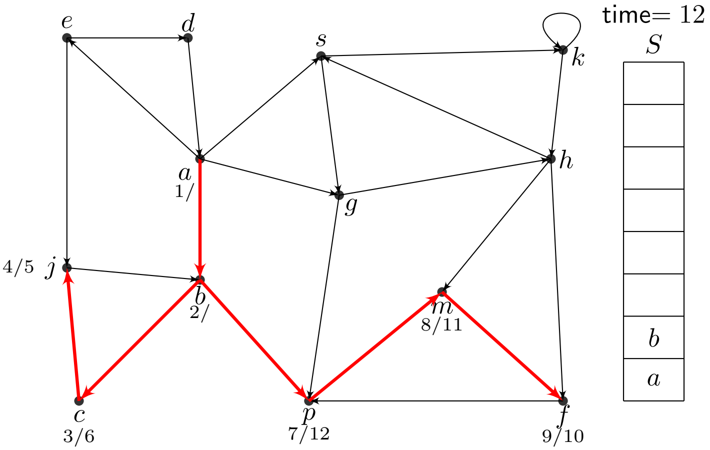
深度优先搜索实例



(j)

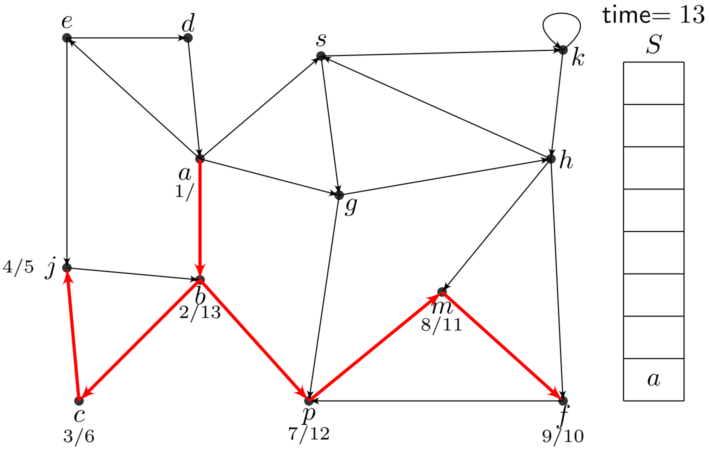


(k)



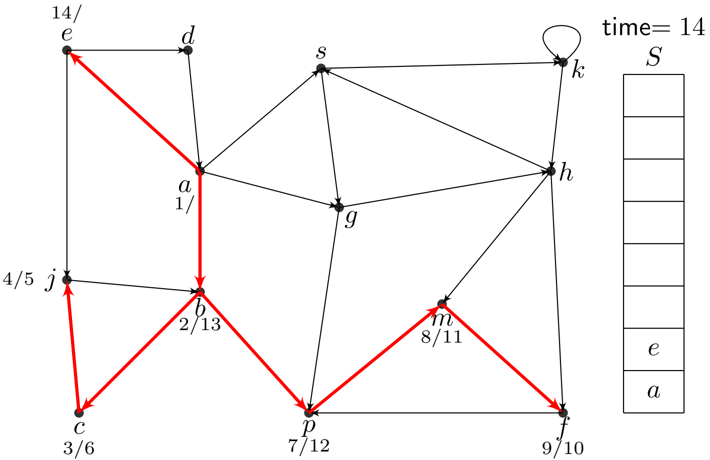
(l)

深度优先搜索实例

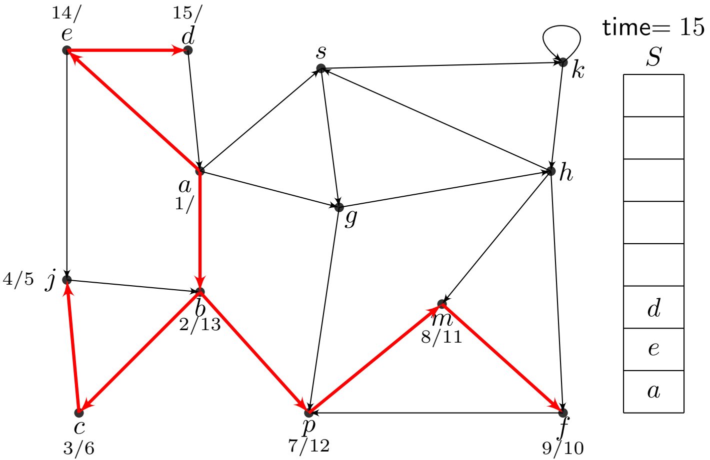


(m)

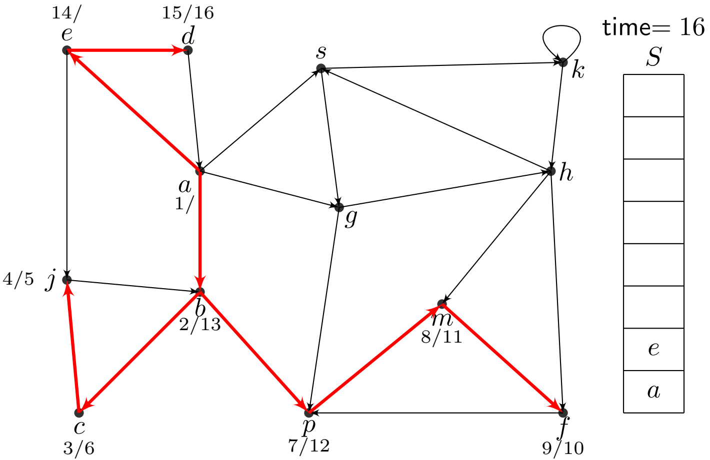
深度优先搜索实例



(n)

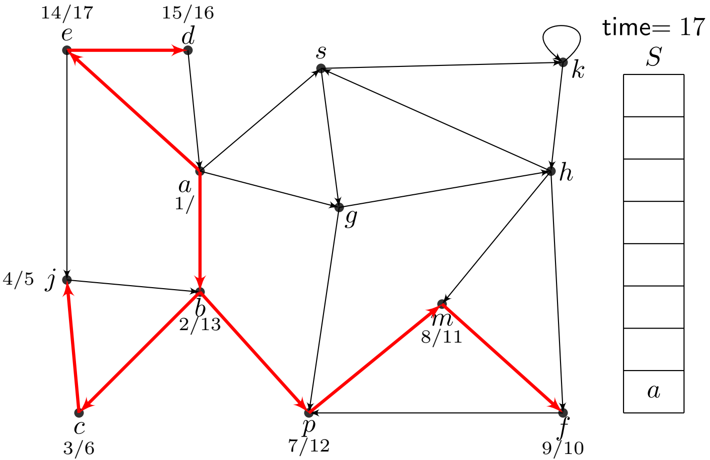


(o)



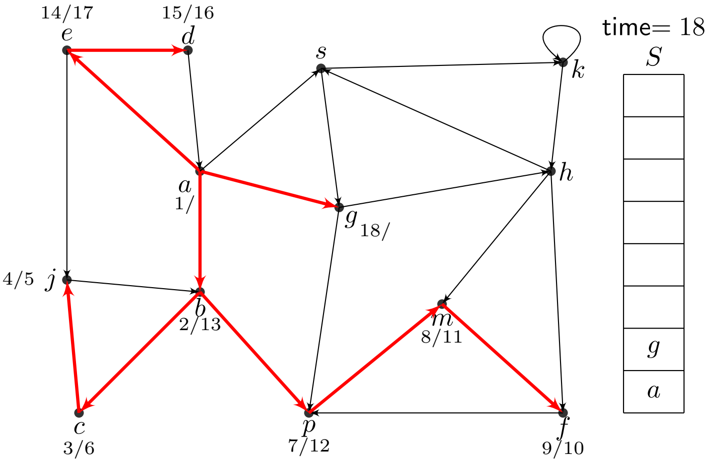
(p)

深度优先搜索实例



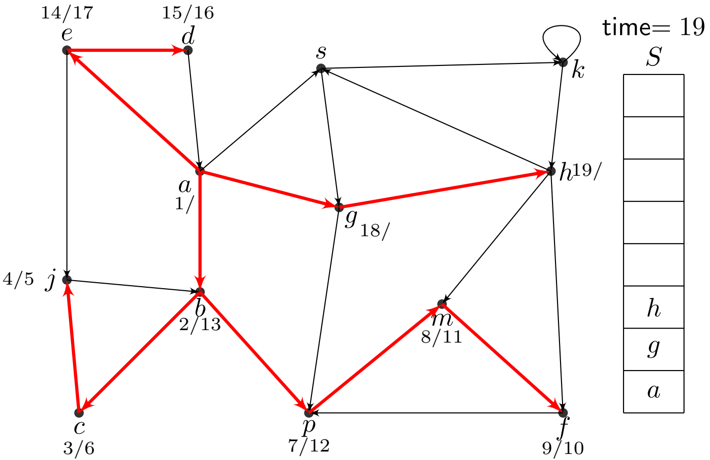
(q)

深度优先搜索实例



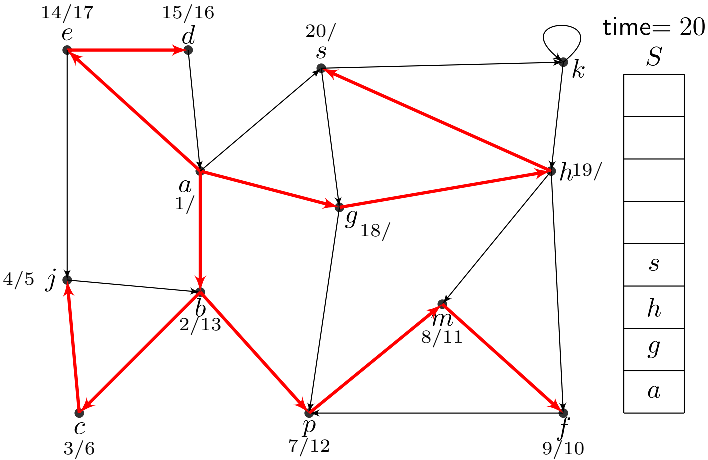
(r)

深度优先搜索实例



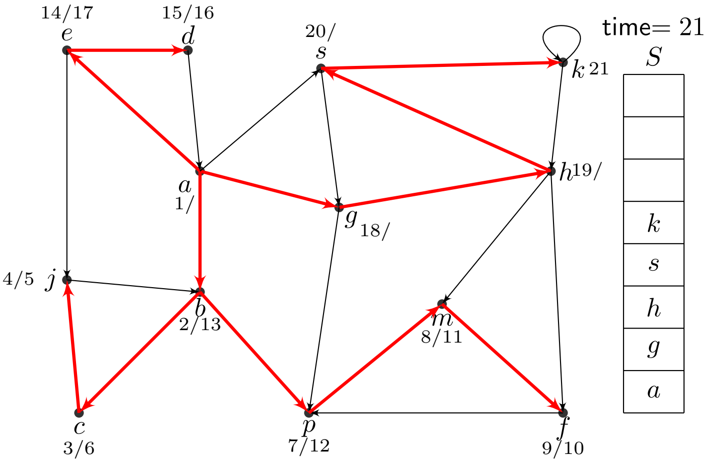
time= 19
S
14/17
e
15/16
d
s
k
a
1/
h19/
g
18/
4/5
j
b
2/13
m
8/11
c
3/6
p
7/12
f
9/10
h
g
a

(s)

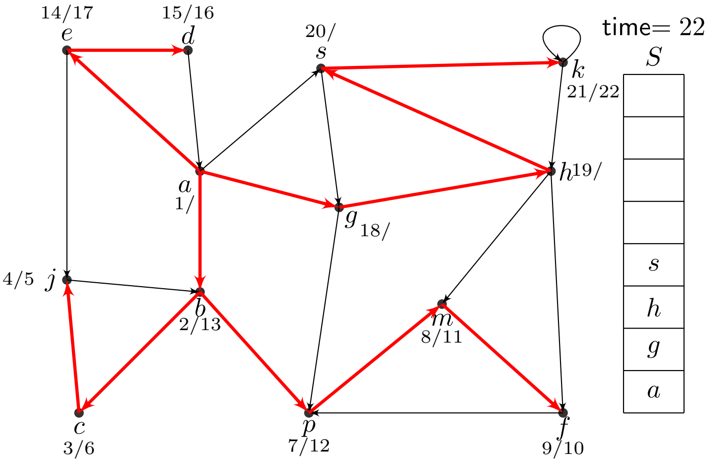

14/17
15/16
20/
time= 20
S
e
d
s
k
h 19/
a
1/
g
18/
4/5 j
b
2/13
m
8/11
c
3/6
p
7/12
f
9/10
s
h
g
a

(t)

深度优先搜索实例

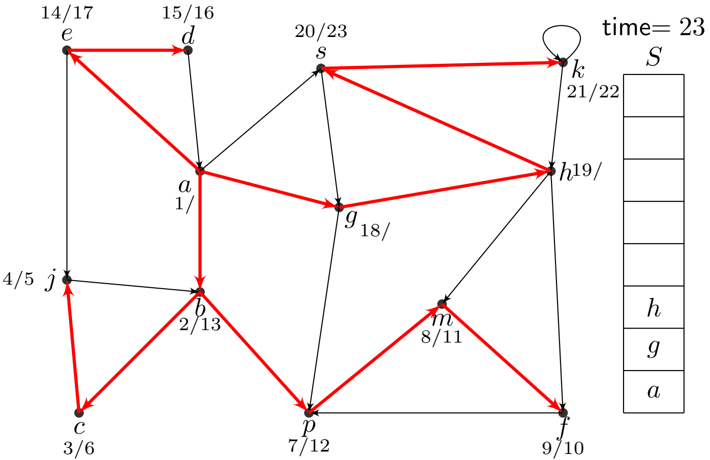

14/17
e
15/16
d
20/
s
time= 21
S
k 21
a
1/
h19/
g 18/
4/5 j
b
2/13
m
8/11
c
3/6
p
7/12
f
9/10
k
s
h
g
a

(u)


14/17
e
15/16
d
20/
s
k
21/22
time= 22
S
a
1/
h19/
g
18/
4/5
j
b
2/13
m
8/11
c
3/6
p
7/12
f
9/10
s
h
g
a

(v)

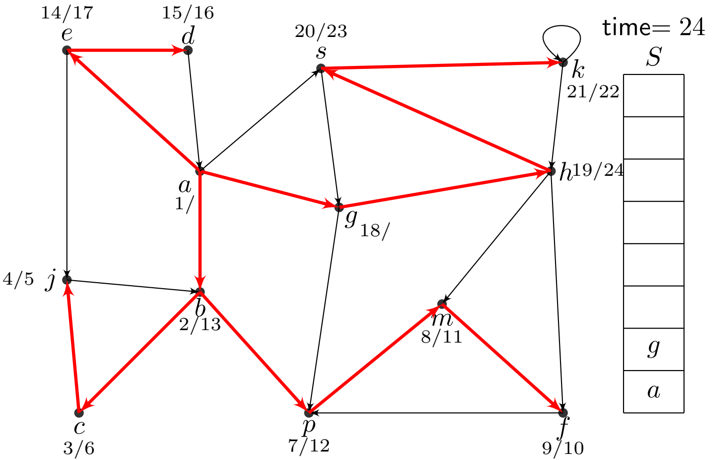
深度优先搜索实例



14/17
e
15/16
d
20/23
s
k
21/22
time= 23
S
a
1/
h19/
g
18/
4/5 j
b
2/13
m
8/11
c
3/6
p
7/12
f
9/10
h
g
a

(w)

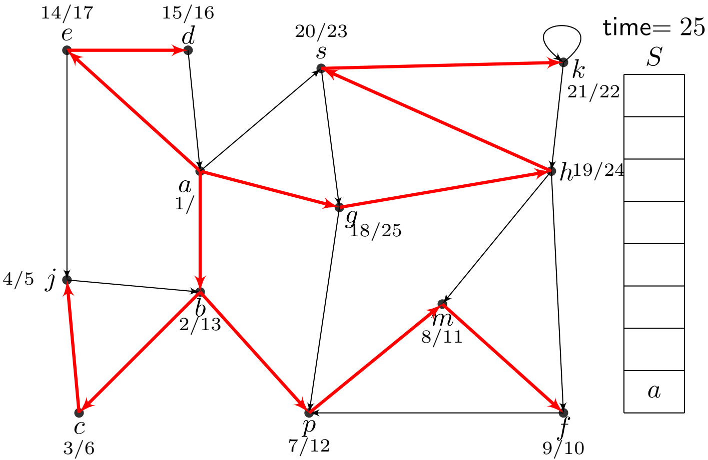
深度优先搜索实例



14/17
e
15/16
d
20/23
s
k
21/22
time= 24
S
h19/24
a
1/
g
18/
4/5
j
b
2/13
m
8/11
g
a
c
3/6
p
7/12
f
9/10

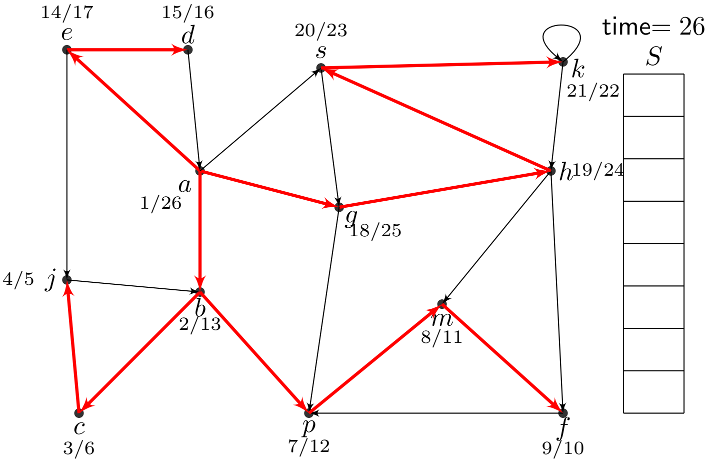
(x)

深度优先搜索实例



14/17
e
15/16
d
20/23
s
k
21/22
h19/24
a
1/
q
18/25
4/5
j
b
2/13
m
8/11
c
3/6
p
7/12
f
9/10
time= 25
S
a

(y)



(z)

图: DFS实例和DFS树或森林

区间套定理

Theorem 2

在算法DFS(G)结束时，顶点u的访问区间$[d(u), f(u)]$包含顶点v的访问区间$[d(v), f(v)]$当且仅当u 是v的祖先。如果u和v没有直接关系，它们的访问区间彼此不相交。

Proof.

- u是v祖先$\Longleftrightarrow$算法先访问u再访问v且v先于u出栈。
- 因此，$d(u) < d(v) < f(v) < f(u)$。
- 假设u和v没有直接关系，且$d(u) < d(v)$，那么在DFS树上没有从u到v路径，即u将在v进栈之前完成访问。
- 反之，若u和v的访问区间不相交，不妨设$d(u) < f(u) < d(v)$。那么在u完成访问时v尚未入栈，意味着v不可能成为u 的子孙。

□

白路径定理

Theorem 3

在算法*DFS*(G)执行过程中，顶点v成为顶点u的后代，当且仅当在时刻$d(u)$，图中存在一条从u到v的由白色顶点构成的路径。

Proof.

- $\Rightarrow$若v是u的后代，那么存在$u = v_0, v_1, \cdots, v_n = v$使得前一个点是后一个点的父亲；
- 根据区间套定理，$d(u) = d(v_0) < d(v_1) < \cdots < d(v_n) = d(v)$。
- 因此，在$d(u)$时刻，该序列构成一条白路径。
- $\Leftarrow$设在$d(u)$时刻，$u = v_0, v_1, \cdots, v_n = v$为一条白路径。
- 归纳基础：v_1必然是u的后代。
- 若$v_1, v_2, \cdots, v_k (1 < k < n)$是$u$的后代，那么$v_{k+1}$必然也是$u$的后代。

$\square$

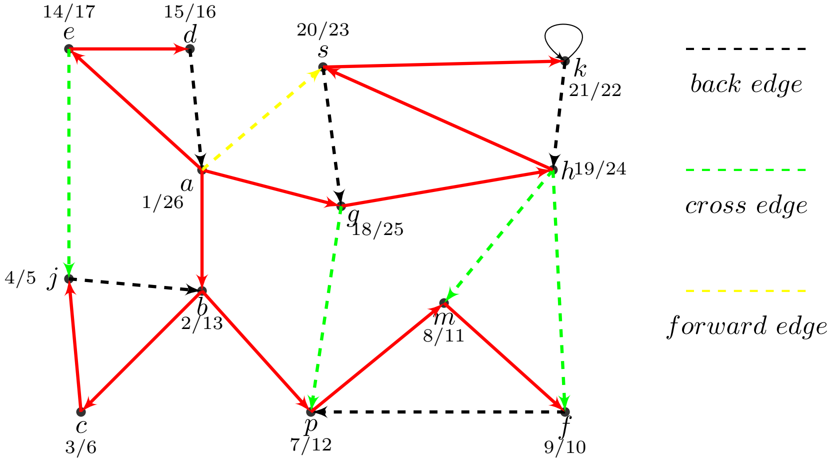
DFS树将图中非树中的边分成3类：

- 反向边：从某顶点出发指向该顶点的祖先；
- 前向边：从某顶点出发指向该顶点后代；
- 交叉边：从某顶点出发指向没有直系亲属关系的顶点。

当DFS访问u的邻居v时，若color$(v) \neq$ White，那么(u,v)不属于DFS树：

- color(v) = Gray，则(u,v)为反向边；
- color(v) = Black，且$d(u) < d(v)$，则(u,v)为前向边；
- color(v) = Black，且$d(u) > d(v)$，则(u,v)为交叉边。

边分类的例子


14/17
e
15/16
d
20/23
s
k
21/22
h19/24
a
1/26
g
18/25
4/5 j
b
2/13
m
8/11
c
3/6
p
7/12
f
9/10
back edge
cross edge
forward edge

拓扑排序

- 拓扑排序：将偏序集线性化；
- 任意偏序集对应着一张DAG图。

Topological-Sort(G)

调用DFS(G)对图G进行深度优先搜索。
在DFS进行过程中，当一个顶点完成时，将它输出并插入到已输出序列的前面。
按序列的顺序输出各顶点。
END

拓扑排序算法的正确性

- 只需证明对于任意边(u, v)，算法输出u在输出v之前；
- 根据算法，只要证明$f(u) > f(v)$即可；下面分两种情况讨论：
- case 1：DFS过程先发现u。
  - 根据白路径定理，v必然是u的后代，
  - 再由区间套定理知$f(u) > f(v)$；
- case 2：DFS过程先发现v。
  - 由于图G是DAG图，因此图中不存在v到u的有向路；
  - 根据白路径定理，u必然不是v的后代，
  - 再由区间套定理知，$d(v) < f(v) < d(u) < f(u)$。

有向图的强连通分支

Definition 4

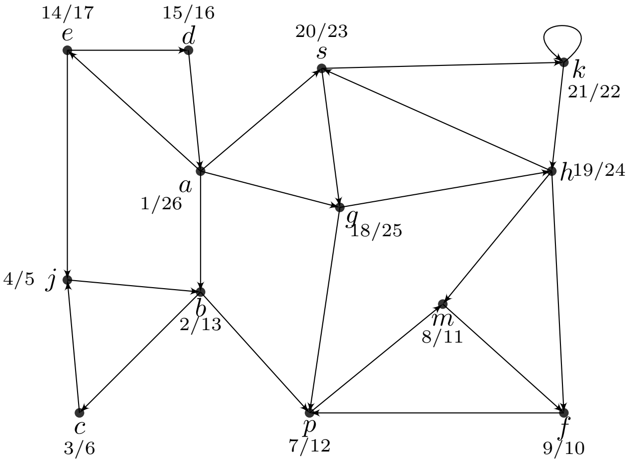
- 有向图$G=(V,E)$，若$\forall u,v\in V$存在从u到v的有向路，则称G是强连通的。
- 有向图G的基图G'是将G中的边去掉方向后的无向图。
- 若有向图G的基图是连通图，则称G是弱连通的。
- 若有向图G的一个子图是强连通的，则称该子图是强连通子图。
- 有向图的最大强连通子图，称为该有向图的一个强连通分支。
- 有向图的强连通分支问题就是把一个有向图的顶点划分为若干个不相交的强连通分支。

强连通分支算法

Strongly-Connected(G)

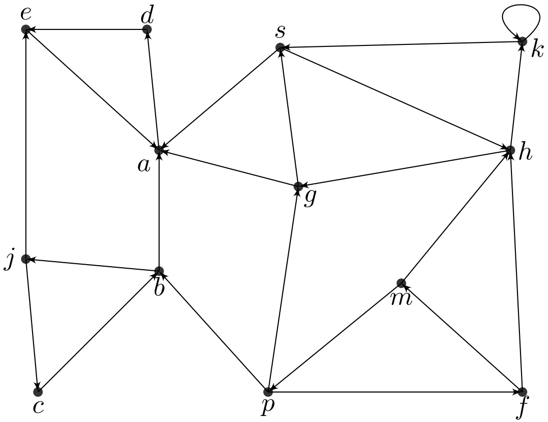
1: 对图G进行DFS搜索并标出各顶点u的访问开始和完成时刻$d[u]/f[u]$.
2: 构造图G的转置图G^T，G^T是把G中每条边反向后得到的图。
3: 从有最大完成时刻的顶点u出发对G^T进行一轮DFS。所有访问到的顶点形成一个强连通分支并且被输出。如果还有未访问到的顶点，则在这些未访问到的顶点中重复这一步直到所有点都被某一轮DFS输出。
4: **END**

强连通分支算法的例子



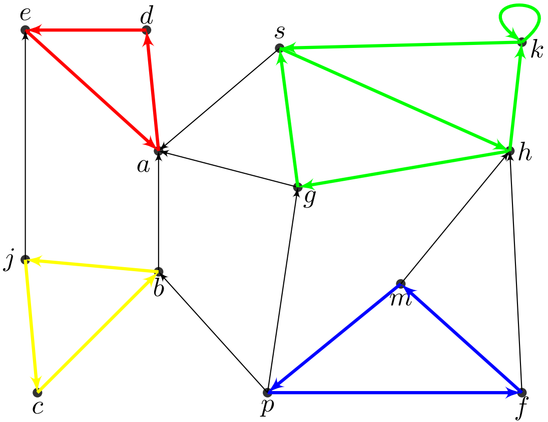
(a) 算法第一步

强连通分支算法的例子



(b) 算法第二步

强连通分支算法的例子



(c) 算法第三步

- 图G的强连通分支图$G^C = (V^C, E^C)$定义如下：
- V^C中的每个点对应G中一个强连通分支；
- 若$(i, j) \in E^C$，那么G中存在边连接强连通分支i和j。
- 注意：若存在多条边连接i和j，这些边必定有相同的方向。

图: 强连通分支图的例子

强连通分支算法的正确性

Lemma 5

设$G^C = (V^C, E^C)$是图G的强连通分支图。若$(i, j) \in E^C$，那么*DFS*(G)过程中，在分支i和j的所有点中最后完成的点必在分支i中。

Proof.

- 分两种情况讨论：
- case 1：DFS最先发现的点x在分支i中。
  - 那么对于i和j中的任意点y,存在从x到y的白路径。
  - 根据白路径定理和区间套定理，$f(y) < f(x)$.
- case 2：DFS最先发现的点x在分支j中。
  - 对于j中任意点y，存在从x到y的白路径；根据白路径定理，y是x的后代。
  - 对于i中任意点z，不存在x到z的白路径；故z和x无直接关系。再由区间套定理知，$f(z) > f(x)$。

□

强连通分支算法的正确性

- 若x是最后完成访问的顶点，x在分支i中，
- 根据上述引理，分支i必然没有进入的边；
- 由于转置操作不影响强连通分支的划分，
- 因此，在G^T中，分支i必然没有出去的边。
- 所以，在算法第二步从x出发DFSG^T时，i中的点均将被访问到，且被限制在i。
- 即分支i被正确的分离。
- 以此类推，第二步的每一轮DFS都将正确的分离出一个强连通分支。