Lecture 13-1
Exercise and Review

1. BFS
2. DFS
3. Topological Sort
4. Strongly Connected Components

Review of BFS

- Problem
  - Input: G=(V, E), $s \in V$
  - Output: Shortest Paths and $\delta(s, \cdot)$
- What does it do?
  - d[]
  - π[]
  - Predecessor Graph
    - $V_\pi = \{u \in V \mid \pi[u] \neq \text{NIL}\} \cup \{s\}$
    - $E_\pi = \{(\pi[u], u)\}$
- Our goal
  - $\delta(s, v) = d[v]$
  - $s \rightarrow v = s \rightarrow \pi[v] + (\pi[v], v)$

The Breadth-First Search (more details)

- *G* is given by its adjacency-lists.
- Initialization:
  - First Part: lines 1 – 4
  - Second Part: lines 5 - 9
- Main Part: lines 10 – 18

- Enqueue(*Q*, *v*): add a vertex *v* to the end of the queue *Q*
- Dequeue(*Q*): Extract the first vertex in the queue *Q*

```
BFS(G, s)
1   for each vertex u∈V[G]-{s}
2         do color[u]<--WHITE
3             d[u]<--∞
4             π[u]<--NIL
5   color[s]<--GRAY
6   d[s]<--0
7   π[s]<--NIL
8   Q<--ϕ
9    ENQUEUE(Q, s)
10   while Q≠ϕ
11      do u<--DEQUEUE(Q)
12          for each v∈Adj[u]
13              do if color[v]=WHITE
14                  then color[v]<--GRAY
15                       d[v]<--d[u]+1
16                       π[v]<--u
17                       ENQUEUE(Q, v)
18          color[u]<--BLACK
```

Important Properties

- Lemma 22.1: $G=(V, E)$, $s \in V$, for any edge$(u, v) \in E$, $\delta(s, v) \le \delta(s, u)+1$.
- Lemma 22.2： $G=(V, E)$, $s \in V$, foe each $v \in V$, $d[v] \ge \delta(s, v)$.
- Lemma 22.3： Let Q=$<v_1, v_2, \ldots, v_r>$, then $d[v_r] \le d[v_1] + 1$， and $d[v_i] \le d[v_{i+1}]$, $1 \le i \le r$-1.
- Corollary 22.4: v_i is enqueued before v_k, $d[v_i] \le d[v_k]$.
- Theorem 22.5： $\delta(s, v)=d[v]$

$$s \rightarrow v = s \rightarrow \pi[v] + (\pi[v]\,,\, v)$$

Exercises

- 22.2-2

vertex u	r	s	t	u	v	w	x	y
d[u]	4	3	1	0	5	2	1	1
π[u]	s	w	u	NIL	r	t	u	u

Exercise 22.2-7

- The diameter of a tree T=(V,E): $\max_{u,v\in V}\{\delta(u,v)\}$
- idea: 以每个点v为源点,广度优先搜索,计算它到其它所有顶点的δ(v,·) ,计算所有δ(·,·) 的最大值.

1 for each vertex $u \in V$

2 do BFS(G, u);

O(V(V+E))

A novel idea

- 任意选一个点u，BFS(G, u)，计算最大的$\delta(u,\cdot)$；设 $\delta(u,v)$最大，BFS(G, v)，计算最大的$\delta(v,\cdot)$即为树的直径。$O(V+E)$
- 对树上任意的两条简单路P_1和P_2，则$P_1 \cap P_2$要么为空要么是一条简单路。

- 引理：v是直径的一个端点。
- 证明：
  - u在直径上。设直径为 $x \to u \to y$，无妨设$\delta(u,x) \geq \delta(u,y)$。

    则对任意的w， $w \to u$只能与$x \to u$或$y \to u$之一有除u之外的公共点。$\delta(u,x) \geq \delta(u,w)$, 否则 $w \to u \to y$ 或 $w \to u \to y$是更长的，矛盾。

Proof (Continued)

- u不在直径上。设直径为 $x\to w\to y$，其中w 是$u\to x$出现的第一个$x\to y$上的点。无妨设$\delta(u,x)\geq\delta(u,y)$。则$u\to v$必与直径相交，否则因为$\delta(u,v)\geq\delta(u,x)\geq\delta(u,y)$，则$\delta(w,v)\geq\delta(w,x)$，以致于$\delta(v,y)\geq\delta(x,y)$
- 设$u\to v$与直径相交于$w\to z$。如果z位于$w\to y$上，则$x\to w\to z\to v$是更长的；如果z位于$w\to x$上，则$y\to w\to z\to v$是更长的。所以v必然是x，y之一，即$z=x$。

DFS Algorithm

DFS(G)

for each vertex $u \in V[G]$
    do $color[u] \leftarrow$ WHITE
        $\pi[u] \leftarrow$ NIL
$time \leftarrow 0$
for each vertex $u \in V[G]$
    do if $color[u] =$ WHITE
        then DFS-VISIT(u)

DFS-VISIT(u)

$color[u] =$ GRAY
$d[u] \leftarrow time \leftarrow time + 1$
for each $v \in Adj[u]$
    do if $color[v] =$ WHITE
        then $\pi[v] \leftarrow u$
            DFS-VISIT(v)
$color[u] =$ BLACK
$f[u] \leftarrow time \leftarrow time + 1$

Properties of DFS

- Properties of the DFS
  - $u=\pi[v]$ if and only if DFS-VISIT(v) was called during a search of u's adjacency list.
  - v is a descendant of u if and only if v is discovered during the time in which u is gray.
  - Theorem 22.7 Parenthesis theorem(three cases)
    - Either nesting or disjoint
  - Corollary 22.8 Nesting of descendants' intervals
    - v is a descendant of u if and only if $d[u]<d[v]<f[v]<f[u]$
  - Theorem 22.9 White-path theorem
    - v is a descendant of u if and only if at time $d[u]$, v can be reached from u along a path consisting entirely of white vertices

Classification of edges

- Classification of edges
  - Directed graph: 4 categories.(u,v)
    - Tree edges(树边): $d[u]<d[v]<f[v]<f[u]$
    - Back edges(返回边、反向边): $d[v]<d[u]<f[u]<f[v]$
    - Forward edges(前向边、正向边): $d[u]<d[v]<f[v]<f[u]$
    - Cross edges(交叉边): $d[v]<f[v]< d[u]< f[u]$
    - You should also know the color of v in each category.
  - Undirected graph: 2 categories. (u,v) and $d[u]<d[v]$
    - Tree deges: from u to v
    - Back edges: from v to u

Exercises

- 22.3-2
- 22.3-6

DFS(*G*)

```
for each vertex u ∈ V[G]
    do color[u] ← WHITE
        π[u] ← NIL
time ← 0
Stack S ← ∅
for each vertex u ∈ V[G]
    do if color[u] = WHITE
        Then S.push(u)
While S ≠ ∅
do  color[S.top] ← GRAY
        d[S.top] ← time ← time +1
if exists v ∈ Adj[S.top] &&
    color[v] = WHITE
        then π[v] ← S.top
            S.push(v)
else
    u ← S.pop()
    color[u] ← BLACK
    f[u] ← time ← time + 1
```

22.3.12

- Singly Connected: for all vertices $u,v \in V$, if $u \rightarrow v$, then there is at most one simple path from u to v.
- idea:
  - DFS-VISIT(u) 可以发现u可达的所有顶点,即u到这些点都有路径。
  - 前向边和交叉边（搜索过程中遇到黑点）意味着什么呢？u到某个点有多于1条路径。
  - 这只是u到其它点的情况，单连通要分析任意的顶点对，所以需要分析每个点到其它所有点的情况。即从每个点开始，都做一次全新的DFS-VISIT()。

- 引理：单连通⇔无前向边交叉边
- 证明：(⇒)有前向边或交叉边，则不单连通
- (⇐)不单连通，则有前向边或交叉边
  - 假设存在某点到另一点之间多于1条简单路。无妨设u到v有两条互不相交的路，
  - $P_x=\langle u,x_1,x_2,\ldots,x_r,v\rangle$, $P_y=\langle u,y_1,y_2,\ldots,y_t,v\rangle$, r,t不同时为0
  - 以u为初始点进行深度优先搜索。
  - 如果$\{x_1,x_2,\ldots,x_r, y_1,y_2,\ldots,y_t\}$不存在v的后代，则边$(x_r,v)$和$(y_t,v)$都不是返回边，而且最多有一条是树边，必有一条是前向边或交叉边。

- 如果 $\{x_1,x_2,\ldots,x_r, y_1,y_2,\ldots,y_{t-1}\}$中存在v的后代顶点，设$x_i$（或$y_j$）是沿着u到v的路上离u最近的v的后代顶点。则$x_{i-1}(y_{j-1})$不是v的后代，则边$(x_{i-1}, x_i)$（或$(y_{j-1}, y_j)$）要么是交叉边，要么是前向边。

Modification(continued)

```
DFS(G)
    for (each vertex u ∈ V[G])
        time ← 0;
     do{ for (each vertex v ∈ V[G])
            do{  color[v] ← WHITE;
                 π[v] ← NIL;
               }
            DFS-VISIT(u)
        }
     Print "G is singly connected"
```

Modification (Continued)

```
DFS-VISIT(u)
    color[u] ← Gray
    time ← time+1
    d[u] ← time
    for each v ∈ Adj[u]
        do if color[v]=White
            Then π[v] ← u
                DFS-VISIT(v)
           if color[v]=Black
            Then Print "G is not singly connected"
                EXIT
    color[u] ← Black
    d[u] ← time ← time+1
```

Topological Sort

- A topological sort of a directed acyclic graph $G = (V, E)$ is a linear ordering of all its vertices such that if G contains an edge (u, v), then u appears before v in the ordering.

The Algorithm

TOPOLOGICAL-SORT(G)

```
call DFS(G) to compute finishing times f[v] for each vertex v
as each vertex is finished, insert it onto the front of a linked list
return the linked list of vertices
```

A Property for DAG

- Lemma 22.11： A directed graph G is acyclic if and only if a depth-first search of G yields no back edges.

Correctness Proof of the Algorithm

- 拓扑排序的定义：图中存在边(u, v)，则在序列中u出现在v的前方。
- 算法总是将后结束的点放在序列的前方。
- 所以只需要证明存在边(u, v)，则v比u先结束。DFS可将图中的边分为四类：
  - 树边： u是v的父亲，$f[u] > f[v]$
  - 前向边： u是v的祖先， $f[u] > f[v]$
  - 交叉边： u和v没关系，$f[u] > f[v]$
  - ~~返回边~~： u是v的后代， $f[u] < f[v]$

Exercises

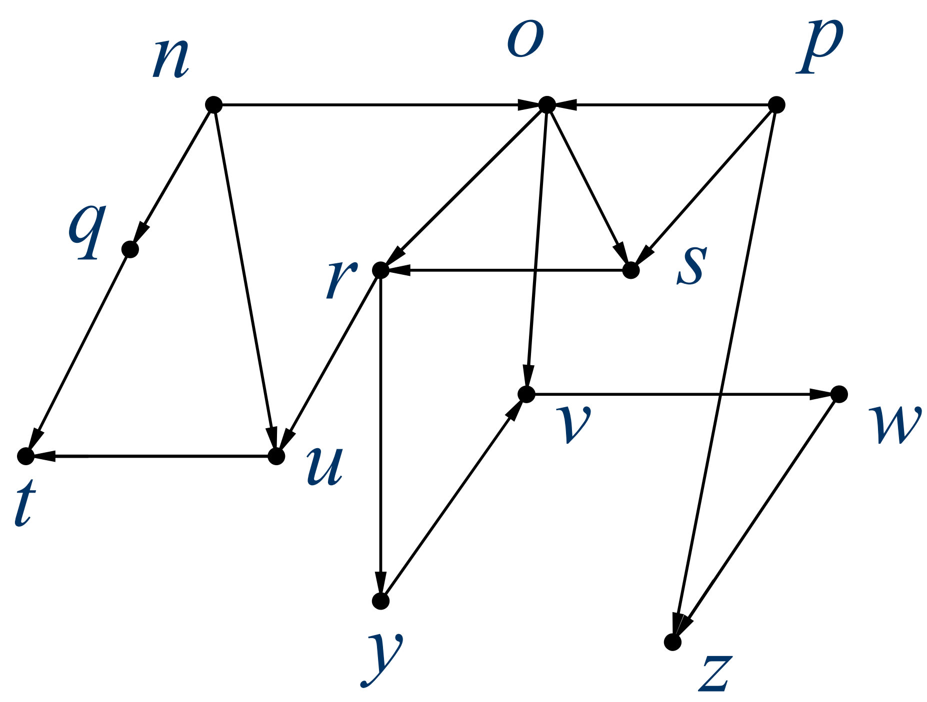
- 22.4-2
- 输入：有向无环图G=(V,E), 两个顶点s,t.
- 输出：s到t的路径数。
- idea：
  - Topological(G)
  - 对每个顶点u，$p[u]$表示$u\rightarrow t$的路径数，初始$p[]\leftarrow 0$, $p[t]=1$.
  - 从t 往前依次扫描每个 顶点u.计算$p[u]=\sum_{v\in Adj[u]} p[v]$.

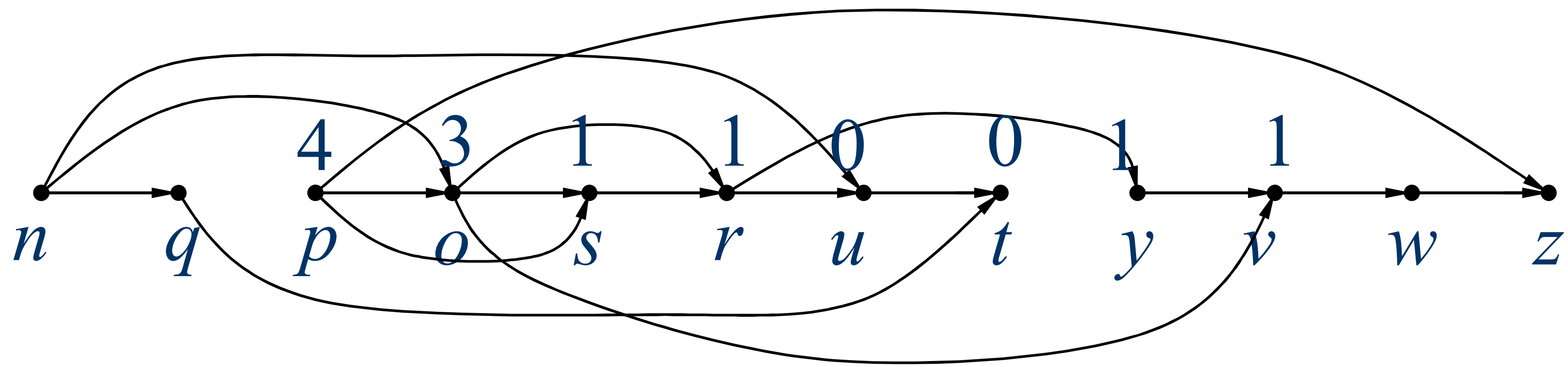
The Algorithm

1. Topologically sort all vertices in G. Assuming that the vertices are $v_1, v_2, \ldots, v_i, \ldots, v_j, \ldots, v_n$ in their topological order, where $s = v_i$, $t = v_j$.
2. Initialize $p[v] \leftarrow 0, \ \forall v_1 \le v \le v_n$.
3. $p[v_j] \leftarrow 1$.
4. **For** $k \leftarrow j$-1 down to i
5. **for each** $v \in \mathrm{Adj}[v_k]$ **do**
6. **if** $v \le v_j$ according to their topological order
7. **then** $p[v_k] \leftarrow p[v_k] + p[v]$.
8. **return** $p[v_i]$.

Time Complexity

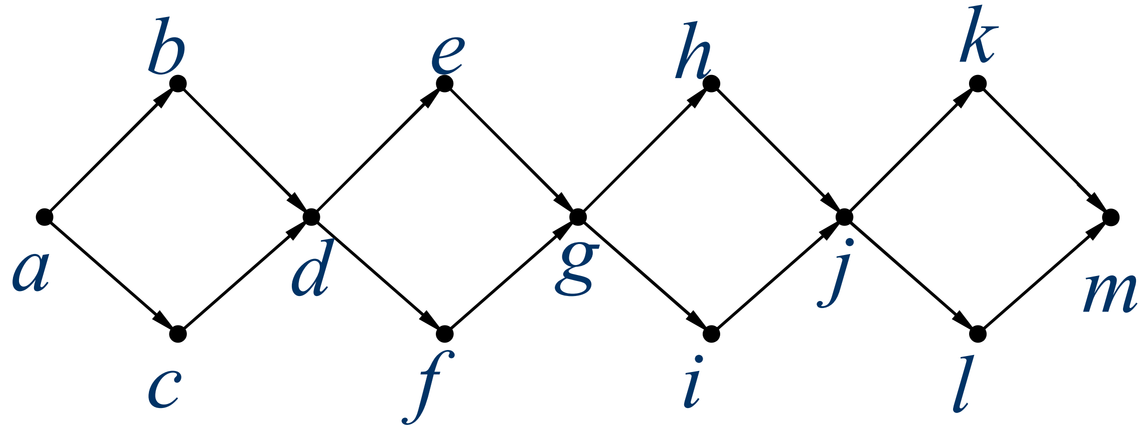
- Step 1: $O(V+E)$
- Step 2: $O(V)$
- Step 4~7: $O(V+E)$
- Total time complexity: $O(V+E)$.

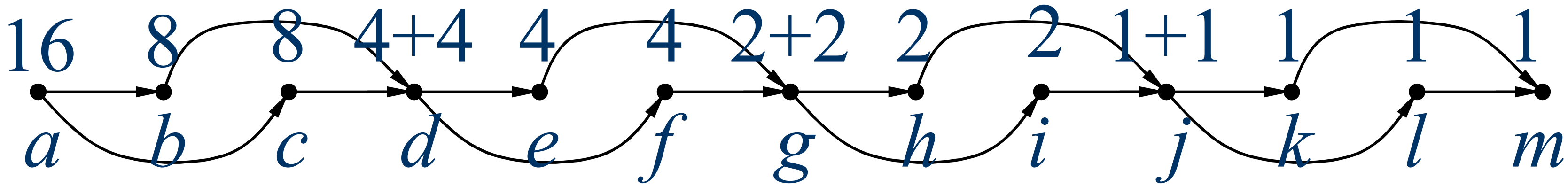




There are 4 distinct paths from *p* to *v*.

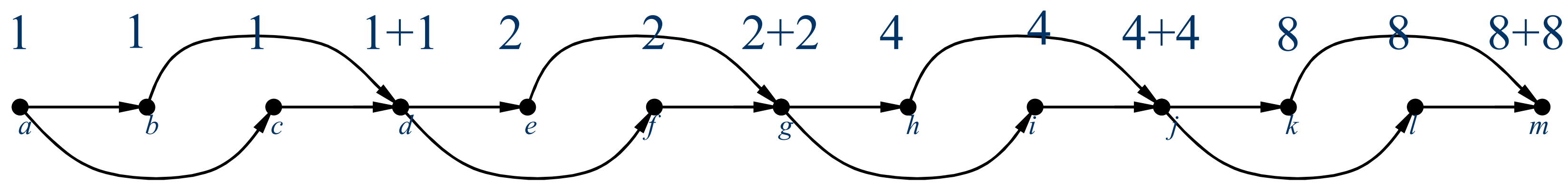
Another example





There are 16 distinct paths from *a* to *m*.

An Example Again



There are 16 distinct paths from *a* to *m*.

22.4-3

- Given an algorithm that determines whether or not a given undirected graph G=(V,E) contains a cycle. *O*(V), independent of |E|.
- DFS, stops whenever encounters an back edge.

22.4-5

- Note:
- 首先要计算每个点的入度。O(V+E)
- 当删一个点时，其邻接点的入度要减1。O(V+E)
- 记录入度为0的点.

22.4-5 Proof of Correctness

- Induction on |V|. |V|=2, only one edge, trivial.
- Suppose it is true for |V|<n.
- When |V|=n. Select a vertex s of in-degree zero.(must exist. Otherwise, all vertices have non-zero in-degrees. Start from a vertex and backtrack along in-edges. Since V is limited, the procedure must end at an in-edge which leaves a vertex already encountered, thus implies a cycle).
- According to our algorithm, s will be at the most left of the list L. And the other part L' of the list is exactly the list of the graph G' obtained by deleting s and the edges leaving it. By induction hypothesis, all edges in G' pointing from left to right in the list L'(thus also in L). Considering that all edges in G are either those in G' or those leaving s, we completes the proof.

– for any $(u,v)\in E$, the in-degree of v can not be zero before deleting u, which implies u appears in front of v.

Strongly Connected Component

- A strongly connected component of a directed graph $G = (V, E)$ is a vertex induced sub-graph $G' = (V', E')$ of G, such that :
  - Every pair of vertices in V' are reachable from each other in G';
  - Any other sub-graph that contains more vertices than G' **does not** satisfy (1).
- Given a directed graph $G = (V, E)$, the transpose of G is the graph $G^T = (V, E^T)$, where $E^T = \{(v, u) | (u, v) \in E\}$.

The component graph

- The component graph $G^{SCC} = (V^{SCC}, E^{SCC})$ of a directed graph $G = (V, E)$ is defined as follows:
  - Suppose that G has strongly connected components C_1, C_2, …, C_k
  - the vertex set $V^{SCC} = \{v_i \mid v_i$ corresponds to component C_i of G, $i = 1,2, \ldots, k\}$
  - the edge set $E^{SCC} = \{(v_i, v_j) \mid G$ contains a directed edge (x, y) for some $x \in C_i$ and some $y \in C_j$, $i, j = 1, 2, \ldots, k, i \neq j\}$

An important property

- Lemma 22.13 :The component graph $G^{\mathrm{SCC}} = (V^{\mathrm{SCC}}, E^{SCC})$ is a directed acyclic graph.

The Algorithm

```
STRONGLY-CONNECTED-COMPONENTS(G)
1   call DFS(G) to compute finishing times f[u] for each vertex u
2   compute G^T
3   call DFS(G^T), but in the main loop of DFS, consider the vertices
            in order of decreasing f[u] (as computed in line 1)
4   output the vertices of each tree in the depth-first forest formed in line 3 as a
            separate strongly connected component
```

Crucial Lemma

- Lemma 22.14 Let C and C' be two distinct strongly connected components in directed graph $G = (V, E)$. Suppose that there is an edge $(u, v) \in E$, where $u \in C$ and $v \in C'$. Then $f(C) > f(C')$.
- Corollary 22.15： Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$. Suppose that there is an edge $(u, v) \in E^{T}$, where $u \in C$ and $v \in C'$. Then $f(C) < f(C')$.

Final Destination
Correctness Proof of the Algorithm

- Theorem 22.16： STRONGLY-CONNECTED-COMPONENTS(G) correctly computes the strongly connected components of a directed graph G.

Exercise 22.5-5

- 下一次DFS-VISIT()时候的交叉边。

DFS Algorithm

DFS(*G*)

```
for each vertex u∈V[G]
    do color[u] ← WHITE
  π[u] ← NIL
  SCC[u] ← 0
time ← 0; SCCnum ← 0
for each vertex u∈V[G]
    do if color[u] = WHITE
  then SCCnum++
         NewAdj[SCCnum] ←∅
         DFS-VISIT(u)
  Return NewAdj
```

DFS-VISIT(*u*)

```
   color[u] = GRAY
   SCC[u] ← SCCnum
   d[u] ← time ← time +1
for each v ∈ Adj[u]
      do if color[v] = WHITE
          then π[v] ← u
               DFS-VISIT(v)
          if color[v] = BLACK&&
           SCC[v]!= SCC[u]&&
           SCC[v]∉ NewAdj[SCC[u]]
               NewAdj[SCC[u]].add(SCC[v])
  color[u] = BLACK
  f[u] ← time ← time + 1
```

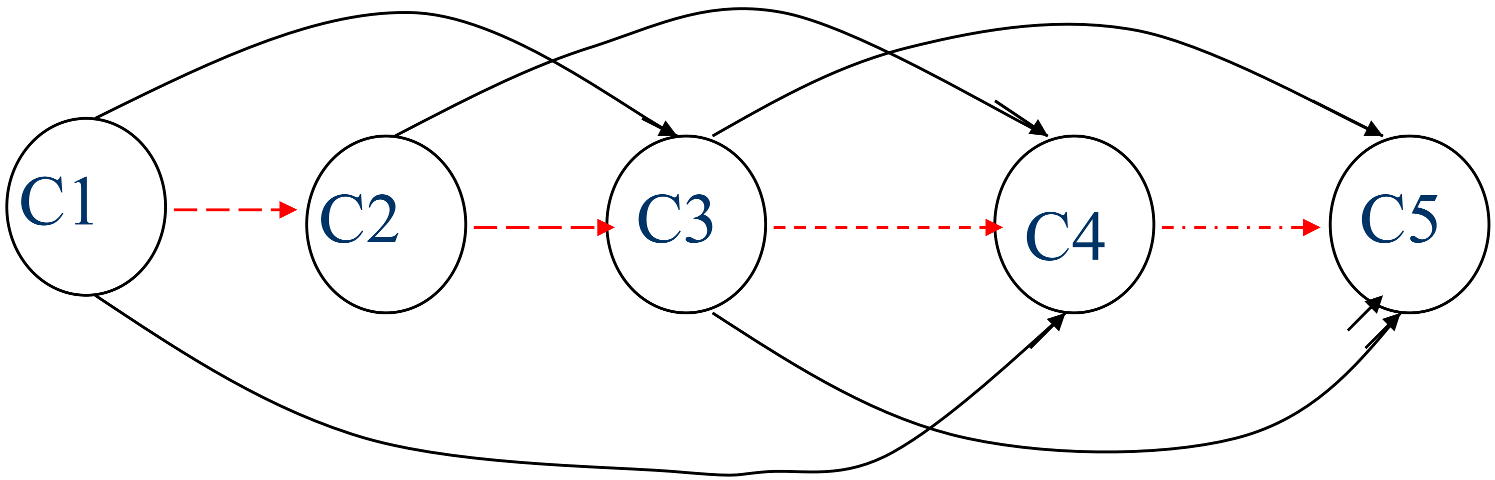
Semi-connected Graph

- A directed graph $G = (V, E)$ is semi-connected if for all pairs of vertices $u, v \in V$, we have $u \leadsto v$ (v is reachable from u) or $v \leadsto u$ (u is reachable from v) or both.
- How to determine whether or not a directed graph is semi-connected? (Exercise 22.5-7)

An Observation

- Given a directed graph $G = (V, E)$, its strongly connected component graph is denoted as $G^{SSC} = (V^{SCC}, E^{SCC})$. Then G is semi-connected if and only if for any pair of vertices $u, v \in V^{SSC}$, either u is reachable from v or v is reachable from u (notice that there **can not** be paths in **both** directions).

A Sketch



Notice that if the above is a topological sort of G^{SCC}, then
edges can only point from the left to right. So, if G is semi-connected,
Path can only point from left to right,
then C1 must reach each C2…,C5, C2 must reach each C3,…,C5,
…, C4 must reach C5. Then the red edges must exist, Vice versa.

The Algorithm Outline

1. Compute the strongly connected components of G.
2. Construct the component graph G^{SCC} of G.
3. Topological Sort G^{SCC}.
4. Judge whether there is an edge from C_i to C_{i+1}, for $0<i<m$, where C_i is indexed by topological sort, and m is the number of SCCs.

- *Time complexity:* $O(V + E)$

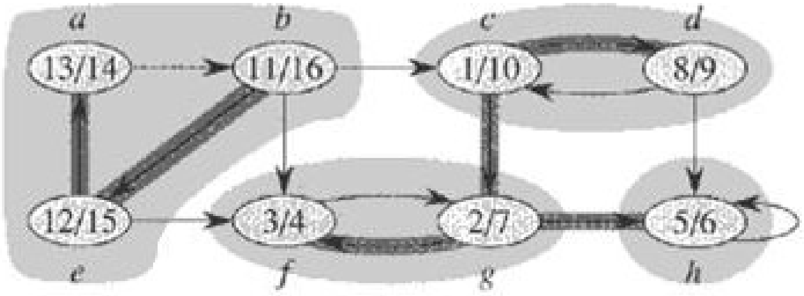
Another method

- In step 3, generate topological sort by using the method that each time find a vertex of zero in-degree. Then G is semi-connected if and only if in each run, exactly one vertex of zero in-degree exist.

- Proof?

An Example

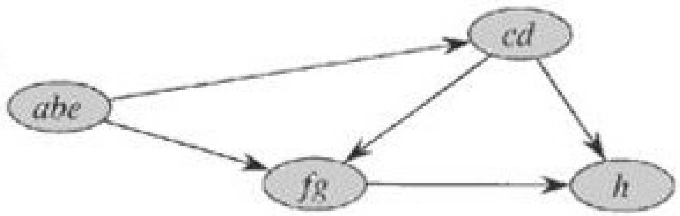
(a)



(a): The graph G with its SCCs shaded

(b)

(c)



(c): The component graph $G^{SCC} = (V^{SCC}, E^{SCC})$ of G

Remarks

- Until now, we have introduced
  - Directed graph
  - Singly connected graph
  - Semi-connected graph
  - Strongly connected graph

Problem22-3

- Euler tour： traverses each edge exactly once，it may visit a vertex more than once.
- in-degree(v)=out-degree(v)
- Proof: (sketch)
- 设C是一条欧拉回路，顶点v在C中每出现一次，其入度和出度都增加1。
- 反之,设p=< v_0, v_1, v_2, …, v_k >是最长的通路,则v_0=v_k,否则v_k的入度比出度多1，而且必存在边e=(v_k,v), e不在p上,与p最长矛盾。
- 若p不包含图中所有边，由G的连通性知，必存在边e=(v_i,v_j)$\notin p$, 从而< v_i, v_{i+1}, v_{i+2}, …, v_i ,v_j>是更长的通路。

Search an Euler Tour

- 从一个顶点出发，一直往下找，一定能回到这个点。
- 关键是如何把找到的圈并起来.
- 用一个链表存储当前找到的圈;另一个链表存储已经找到的圈.

```
Euler(G)
For each e∈E
   color[e] ← white
For each u∈V
   color[u] ← white
EP←∅
EP.add(u)
color[u] ← gray
while exist u∈EP&& color[u] =gray
       L←∅
       C ← Findcycle(u)
EP.insert(u, C)
Return EP

Findcycle(u)
        if exist v∈ Adj [u] && color[(u, v)] =white
             L.add(v)
             color[(u, v)] ←black
             color[v] ←gray
             Findcycle(v)
        color[u] ←black
        Return L
```