算法设计与分析

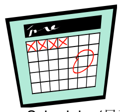
Algorithms Design & Analysis

第十讲：图的算法

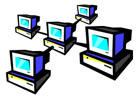
图(Graphs)

- Applications that involve not only a set of items, but also the connections between them (物体及物体之间的连接关系)

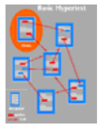
Maps(地图)



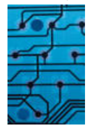
Schedules(日程表)



Computer networks (计算机网络)



Hypertext(超文本)



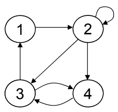
Circuits(电路板)

图的背景知识(Graphs – Background)

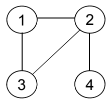
Graphs = a set of nodes (vertices) with edges (links) between them.(节点+边)

Notations:(表示方法)

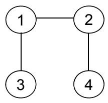
- G = (V, E) – graph (图)
- V = set of vertices |V| = n (节点和节点数目)
- E = set of edges |E| = m (边和边的数目)



Directed Graph (有向图)



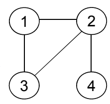
Undirected Graph (无向图)



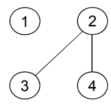
Acyclic Graph (无回路图)

图的其它类型(Other Types of Graphs)

- A graph is **connected** if there is a path between every two vertices (如果任何两个顶点之间存在一个通路,则称图是连接的)

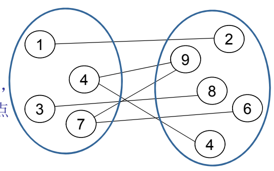


Connected



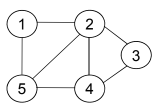
Not connected

- A **bipartite graph** is an undirected graph G = (V, E) in which V = V_1 + V_2 and there are edges only between vertices in V_1 and V_2 (二分图：无向图，由V_1和V_2组成，只在V_1和V_2之间的顶点存在边)

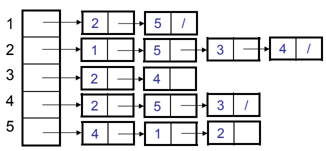


图的表示方法（Graph Representation）

- **Adjacency list representation** of G = (V, E)(邻接表)
  - An array of |V| lists, one for each vertex in V(顺序存储的顶点表)
  - Each list Adj[u] contains all the vertices v such that there is an edge between u and v (边表:一个顶点的边表的每个表目对应与该顶点相关联的一条边)
    - Adj[u] contains the vertices adjacent to u (in arbitrary order) (顺序无关)
  - Can be used for both directed and undirected graphs(适合有向或无向图)

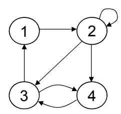


Undirected graph

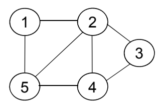


邻接表特性(Properties of Adjacency-List Representation)

- Sum of the lengths of all the adjacency lists （邻接表长度的和→边的数目）
  - Directed graph: (有向图：边出现一次)
    - Edge (u, v) appears only once in u's list |E|
  - Undirected graph: (无向图：边出现两次)
    - u and v appear in each other's adjacency lists: edge (u, v) appears twice 2|E|



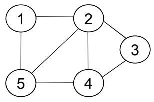
Directed graph



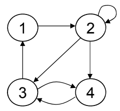
Undirected graph

邻接表特性（Properties of Adjacency-List Representation）

- **Memory required (空间需求)**
  - **$\Theta(V + E)$**
- **Preferred when (适合 |E| << |V|²)**
  - **the graph is sparse: |E| << |V|²**
- **Disadvantage(缺陷)**
  - **no quick way to determine whether there is an edge between node u and v (不容易判断顶点u和v之间是否存在边)**
- **Time to list all vertices adjacent to u: (罗列与u邻接顶点的时间开销)**
  - **Θ(degree(u))**
- **Time to determine if (u, v) ∈ E: (判断(u, v) ∈ E的时间)**
  - **O(degree(u))**



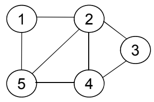
Undirected graph



Directed graph

图的表示方法(Graph Representation)

- **Adjacency matrix representation** of G = (V, E) (相邻矩阵)
  - Assume vertices are numbered 1, 2, … |V| (顶点序号)
  - The representation consists of a matrix $A_{|V| x |V|}$:
  - $a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \ (元素值的确定) \\ 0 & \text{otherwise} \end{cases}$



Undirected graph

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

For undirected graphs matrix A is symmetric:

$a_{ij} = a_{ji}$

$A = A^T$ (无向图的相邻矩阵的对称性)

相邻矩阵的特性(Properties of Adjacency Matrix Representation)

- Memory required (空间需求)
  - $\Theta(V^2)$, independent on the number of edges in G (与边无关)
- Preferred when (适合情况)
  - The graph is dense |E| is close to |V|² (|E|接近|V|²)
  - We need to quickly determine if there is an edge between two vertices (需要快速确定两个顶点之间是否存在边)
- Time to list all vertices adjacent to u:(罗列与u邻接顶点的时间)
  - $\Theta(V)$
- Time to determine if (u, v) ∈ E: (判断(u, v) ∈ E的时间)
  - $\Theta(1)$

有权图(Weighted Graphs)

- **Weighted graphs = graphs for which each edge has an associated weight w(u，v) (图的边被赋予了权值)**

  w: *E* → R, weight function (加权函数)
- **Storing the weights of a graph (权的存储)**
  - **Adjacency list: (邻接表)**
    - **Store w(u,v) along with vertex v in u's adjacency list (在边表表目中)**
  - **Adjacency matrix: (相邻矩阵)**
    - **Store w(u，v) at location (u，v) in the matrix (在元素中)**

图的遍历(Searching in a Graph)

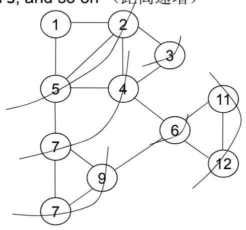
- **Graph searching = systematically follow the edges of the graph so as to visit the vertices of the graph (系统的沿着图的边访问图的顶点)**
- **Two basic graph searching algorithms: (两种基本搜索算法)**
  - **Breadth-first search(广度优先搜索)**
  - **Depth-first search(深度优先搜索)**
  - **The difference between them is in the order in which they explore the unvisited edges of the graph (两种搜索方法的区别在于对图中未访问边的搜索顺序)**
- **Graph algorithms are typically elaborations of the basic graph-searching algorithms (搜索是图的算法的基础，其他有关图的算法基本上是在此基础上的精细化)**

广度优先搜索(Breadth-First Search)(BFS)

- **Input**: (输入)
  - A graph *G* = (V, E) (directed or undirected) (图,有向或无向)
  - A **source** vertex *s* ∈ V (源顶点)
- **Goal**: （目的）
  - **Explore the edges of *G* to "discover" every vertex reachable from s, taking the ones closest to s first (从临近源顶点s最近的顶点开始，通过对图G的边的探索发现从源顶点s能够抵达的每个顶点)**
- **Output**: （输出）
  - d[v] = distance (smallest # of edges) from s to v, for all v ∈ V （从s到v的距离）
  - A "breadth-first tree" rooted at *s* that contains all reachable vertices（广度优先树：以s为根，包含所有可以抵达的顶点）

广度优先搜索（Breadth-First Search）

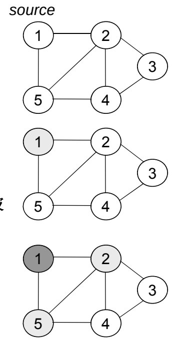
- Discover vertices in increasing order of distance from the source *s* – search in breadth not depth
（广度优先：距离源顶点的距离逐渐增加来搜索相连通的其它顶点）
  - Find all vertices at 1 edge from *s*, then all vertices at 2 edges from *s*, and so on （距离递增）



13

广度优先搜索(Breadth-First Search)

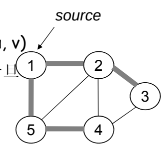
- Keeping track of progress: (过程跟踪)
  - Color each vertex in either **white**, **gray** or **black** (用白、灰和黑来标记顶点）
  - Initially, all vertices are **white** （最初为白）
  - When being discovered a vertex becomes **gray** （刚搜索到，灰）
  - After discovering all its adjacent vertices the node becomes **black** （所有邻接顶点被搜索完，黑）
  - Use FIFO queue *Q* to maintain the set of gray vertices（先进先出队列来保存灰色顶点）



14

广度优先树(Breadth-First Tree)

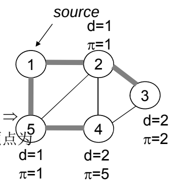
- BFS constructs a breadth-first tree(BFS形成BFT)
  - Initially contains the root (source vertex *s*) (源节点为树根)
  - When vertex v is discovered while scanning the adjacency list of a vertex u ⇒ vertex v and edge (u, v) are added to the tree (在搜索顶点u的临接顶点时一旦发现了顶点v，便将顶点v和边(u,v)加入到树中）
  - u is the **predecessor** (**parent**) of v in the breadth-first tree (在BFT中，u是v的父节点)
  - A vertex is discovered only once ⇒ it has at most one parent (一个顶点只被发现一次，因此只有一个父节点）



15

BFS附加数据结构（BFS Additional Data Structures）

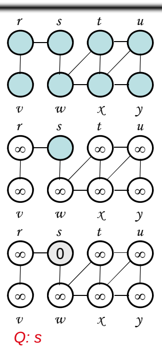
- G = (V, E) represented using adjacency lists（邻接表法表示图）
- color[u] – the color of the vertex for all u ∈ V（顶点颜色）
- π[u] – predecessor of u （顶点u的父顶点）
  - If u = *s* (root) or node u has not yet been discovered ⇒ π[u] = NIL （如果顶点是根顶点或没有被发现，父顶点为空）
- d[u] – the distance from the source *s* to vertex u（距离，从s到u ）
- Use a FIFO queue *Q* to maintain the set of gray vertices （应用FIFO队列*Q*存储灰色顶点）



16

BFS算法：BFS(V, E, *s*)

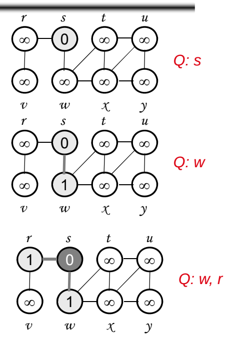
1. **for each** u ∈ V - {*s*}
2. **do** color[u] ← WHITE
3. d[u] ← ∞
4. π[u] = NIL
5. color[*s*] ← GRAY
6. d[*s*] ← 0
7. π[*s*] = NIL
8. Q ← ∅
9. Q ← ENQUEUE(Q, *s*)



17

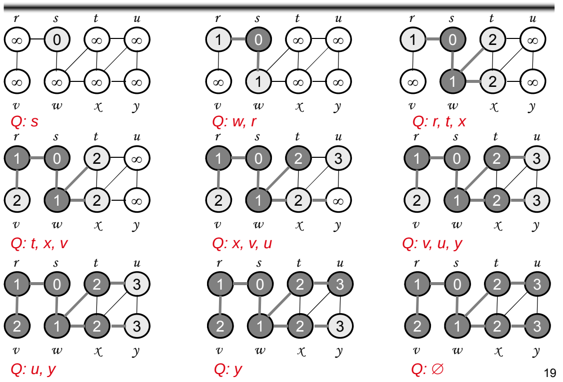
BFS(V, E, *s*)

10. **while** Q ≠ ∅
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



18

Example



19

BFS算法分析（Analysis of BFS）

for each u ∈ V - {s}
  do color[u] ← WHITE
    d[u] ← ∞
    π[u] = NIL

$O(V)$

color[s] ← GRAY
d[s] ← 0
π[s] = NIL
Q ← ∅
Q ← ENQUEUE(Q, s)

$\Theta(1)$

20

Analysis of BFS

while Q ≠ ∅
  do u ← DEQUEUE(Q) — $\Theta(1)$
    for each v ∈ Adj[u] — Scan Adj[u] for all vertices in the graph （搜索所有与u邻接的顶点）
      do if color[v] = WHITE
        then color[v] = GRAY
          d[v] ← d[u] + 1
          π[v] = u
          ENQUEUE(Q, v) — $\Theta(1)$
  color[u] ← BLACK

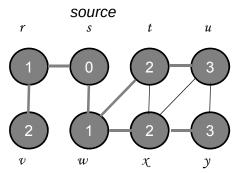
- Each vertex is scanned only once, when the vertex is dequeued（只搜索一次）
  - Sum of lengths of all adjacency lists = Θ(E)（邻接表长度）
  - Scanning operations: O(E) （搜索操作）

- Total running time for BFS = O(V + E) 运行时间

21

最短路径（Shortest Paths Property）

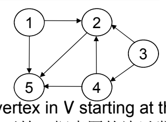
- BFS finds the shortest-path distance from the source vertex *s* ∈ V to each node in the graph
（BFS确定了从源节点到其他顶点的最短路径）
- Shortest-path distance = δ(*s*, u) （最短路径）
  - Minimum number of edges in any path from *s* to u（从*s*到u最小边的数目）



22

深度优先搜索（Depth-First Search）

- **Input:** （输入，图，没有源顶点）
  - G = (V, E) (No source vertex given!)
- **Goal**: （目的）
  - Explore the edges of G to “discover” every vertex in V starting at the most current visited node （从当前访问顶点开始，探索图的边以发现图中的每个顶点）
  - Search may be repeated from multiple sources(搜索从多个源重复)

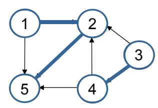


- **Output:** （输出）
  - 2 **timestamps** on each vertex: （每个顶点有两个时间标记）
    - d[v] = discovery time （发现时间）
    - f[v] = finishing time (done with examining v’s adjacency list) （结束时间，检查v所有的邻接表）
  - Depth-first forest（深度优先森林）

23

深度优先搜索（Depth-First Search）

- Search “deeper” in the graph whenever possible **(尽可能往深度搜索)**
- Edges are explored out of the most recently discovered vertex that still has unexplored edges **(最近发现的顶点v的是否有没有搜索边?)**



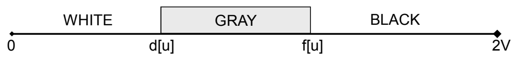
- After all edges of v have been explored, the search “backtracks” from the parent of v （**顶点v的所有边搜索完后，回朔到v的父顶点搜索**）
- The process continues until all vertices reachable from the original source have been discovered（**这个过程不断继续，直到从源顶点可以抵达的所有顶点都被发现**）
- If undiscovered vertices remain, choose one of them as a new source and repeat the search from that vertex （**如果图中存在还没有发现的顶点，选择其中之一作为新的源顶点重复上述搜索过程**）
- DFS creates a “depth-first forest” （**DFS产生的是DFF**）

24

DFS附加数据结构（DFS Additional Data Structures）

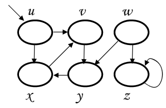
- Global variable: time-step （全局变量：时间步骤）
  - Incremented when nodes are discovered/finished （在顶点发现和搜索完成时增加）
- color[u] – similar to BFS （颜色变量，和BFS类似）
  - White before discovery, gray while processing and black when finished processing（白：发现前，灰：发现处理中，黑：处理完成）
- π[u] – predecessor of u （顶点u的父顶点）
- d[u], f[u] – discovery and finish times （发现、完成时间）

$1 \le d[u] < f[u] \le 2|V|$



DFS算法：DFS(V, E)

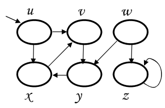
```
for each u ∈ V
    do color[u] ← WHITE
       π[u] ← NIL
time ← 0
for each u ∈ V
    do if color[u] = WHITE
         then DFS-VISIT(u)
```



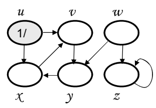
- Every time DFS-VISIT(u) is called, u becomes the root of a new tree in the depth-first forest （ DFS-VISIT(u)每次被调用， u是深度优先森林中一棵新树的根节点）

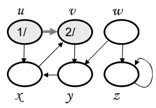
DFS-VISIT(u)

```
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)
color[u] ← BLACK
time ← time + 1
f[u] ← time
```

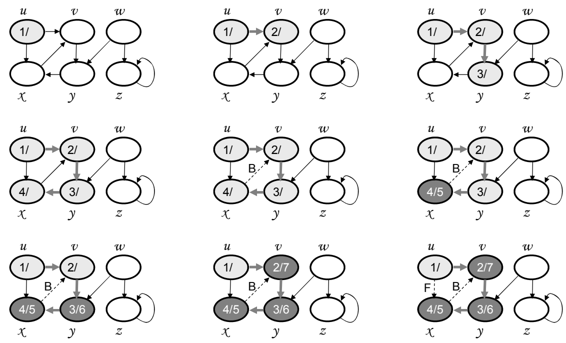


time = 1

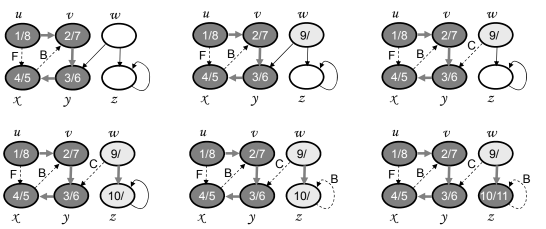


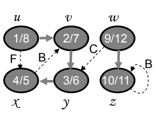


Example



Example (cont.)



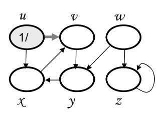


The results of DFS may depend on:（DFS的结果与下列因素有关：）
- The order in which nodes are explored in procedure DFS （顶点的搜索顺序）
- The order in which the neighbors of a vertex are visited in DFS-VISIT（ DFS-VISIT中顶点邻接点的访问顺序）

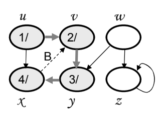
边的类型(Edge Classification)

- **Tree edge (reaches a WHITE vertex): (树边：抵达白色顶点)**
  - **(u, v) is a tree edge if v was first discovered by exploring edge (u, v)**

  (边(u, v) 是一树边，如果v是在探索边(u, v)时首次被发现)

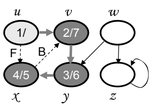


- **Back edge (reaches a GRAY vertex): （回边：抵达灰色顶点）**
  - **(u, v), connecting a vertex u to an ancestor v in a depth first tree （边(u, v)连接顶点u至其一个深度优先树的先辈顶点v）**
  - **Self loops (in directed graphs) are also back edges （有向图中的自回路也是回边）**

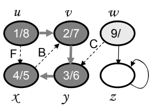


Edge Classification

- Forward edge (reaches a BLACK vertex & d[u] < d[v]):（前向边，抵达黑色顶点，且d[u] （ d[v]）
  - Non-tree edges (u, v) that connect a vertex u to a descendant v in a depth first tree
  （非树边(u, v)，连接顶点u和它在深度优先树上的一个后裔顶点v ）



- Cross edge (reaches a BLACK vertex & d[u] > d[v]):（横跨边：抵达黑色顶点，且d[u] > d[v]）
  - Can go between vertices in same depth-first tree (as long as there is no ancestor / descendant relation) or between different depth-first trees（可以跨越在不同的深度优先树的顶点或同一深度优先树不同顶点间（只要没有遗传关系））



算法分析（Analysis of DFS(V, E)）

```
for each u ∈ V
    do color[u] ← WHITE
        π[u] ← NIL
time ← 0
for each u ∈ V
    do if color[u] = WHITE
        then DFS-VISIT(u)
```

Θ(V)

Θ(V) – exclusive of time for DFS-VISIT（DFS—VISIT执行时间）

DFS-VISIT(u)分析

```
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)
color[u] ← BLACK
time ← time + 1
f[u] ← time
```

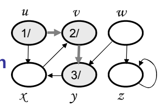
DFS-VISIT is called exactly once for each vertex（对于每个顶点DFS-VISIT被调用一次）

Each loop takes |Adj[v]| （循环次数）

Total: $\underbrace{\Sigma_{v\in V} |Adj[v]|}_{\Theta(E)} + \Theta(V) = \Theta(V + E)$

DFS特性（Properties of DFS）

- u = π[v] ⇔ DFS-VISIT(v) was called during a search of u's adjacency list（u = π[v] ⇔ DFS-VISIT(v) 在搜索u的邻接表时被调用）

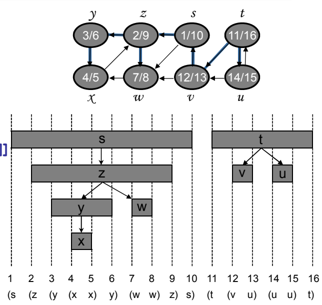


- Vertex v is a descendant of vertex u in the depth first forest ⇔ v is discovered during the time in which u is gray （深度优先树上顶点v是顶点u的后裔⇔顶点v在顶点u处于灰色状态时被发现）

括号定理（Parenthesis Theorem）

In any DFS of a graph G, for all u, v, exactly one of the following holds:（在图的任何DFS，对于任何u, v，下列情况之一成立）

1. [d[u], f[u]] and [d[v], f[v]] are disjoint, and neither of u and v is a descendant of the other （[d[u], f[u]] 和 [d[v], f[v]]不邻接， u和v 互不为后裔）
2. [d[v], f[v]] is entirely within [d[u], f[u]] and v is a descendant of u（[d[u], f[u]] 包含[d[v], f[v]]， v是u 的后裔）
3. [d[u], f[u]] is entirely within [d[v], f[v]] and u is a descendant of v（ [d[v], f[v]] 包含[d[u], f[u]] ， u 是v的后裔）



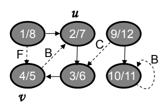
Well-formed expression: parenthesis are properly nested （括号嵌套）

DFS其他特性(Other Properties of DFS)

Corollary (推论)

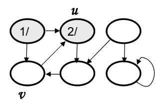
Vertex v is a proper descendant of u
⇔ d[u] < d[v] < f[v] < f[u]

顶点v是顶点u的后裔⇔ d[u] < d[v] < f[v] < f[u]



Theorem (White-path Theorem) (定理)

In a depth-first forest of a graph G, vertex v is a descendant of u if and only if at time d[u], there is a path u ⇒ v consisting of only white vertices. (在图G的任何深度优先森林中，顶点v是顶点u的后裔，当且仅当在时刻d[u]，存在一条u ⇒ v的路径只包含有白色顶点）



拓扑排序（Topological Sort）

Topological sort of a directed acyclic graph G = (V, E): a linear order of vertices such that if there exists an edge (u, v), then u appears before v in the ordering. （拓扑排序：找出有向无回路图G = (V, E)中顶点的一个线性序，使得(u, v)如果是图中的一条边，那么在这个线性序中u在v前出现）

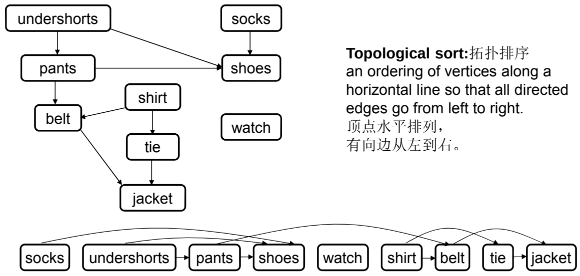
- Directed acyclic graphs (DAGs) （有向无回路图）
  - Used to represent precedence of events or processes that have a **partial order** （表示事件的先后或有顺序的过程）

a before b, b before c } a before c

b before c, a before c } What about a and b?

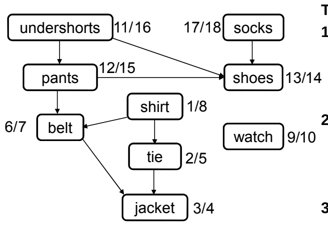
Topological sort helps us establish a **total order** （拓扑排序建立完整的顺序）

拓扑排序（Topological Sort）



Topological sort:拓扑排序
an ordering of vertices along a horizontal line so that all directed edges go from left to right.
顶点水平排列，
有向边从左到右。

拓扑排序（Topological Sort）



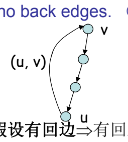
TOPOLOGICAL-SORT(V, E)

1. **Call DFS(V, E) to compute finishing times f[v] for each vertex v （调用DFS(V, E)，对每个顶点v计算其完成时间f[v] ）**
2. **When each vertex is finished, insert it onto the front of a linked list（顶点完成，将之插入到连接表前端）**
3. **Return the linked list of vertices （返回顶点的连接表）**

Running time: Θ(V + E)

引理（Lemma）

A directed graph is **acyclic** ⇔ a DFS on G yields no back edges. （有向图是无回路的⇔深度优先搜索DFS不产生回边）



Proof:

"⇒": acyclic ⇒ no back edge （无回路⇒无回边）

- Assume **back edge** ⇒ prove **cycle**（反证法:假设有回边⇒有回路）
- Assume there is a back edge (u, v) （ (u, v)是回边）

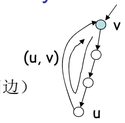
⇒v is an ancestor of u （ v是u的祖先）

⇒ there is a path from v to u in G (v ⇨ u) （从v到u有通路v ⇨ u）

⇒ v ⇨ u + the back edge (u, v) yield a cycle （通路v ⇨ u与回边(u, v)构成回路）

Lemma

A directed graph is **acyclic** ⇔ a DFS on G yields no back edges.



Proof:

"⇐": no back edge ⇒ acyclic （ 无回边⇒无回路）

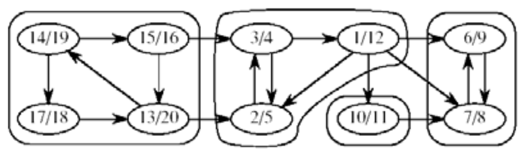
- Assume **cycle** ⇒ prove **back edge** （假设回路⇒回边）
- Suppose G contains cycle c（假设图G有回路c ）
- Let v be the first vertex discovered in c, and (u, v) be the preceding edge in c （顶点v在c中最先发现， (u, v) 是c中的前向边）
- At time d[v], vertices of c form a white path v ⇨ u （在d[v]时刻，c的顶点形成v ⇨ u白色通路）
- u is descendant of v in depth-first forest (by white-path theorem) （在深度优先森林中， u是v的后裔）

⇒ (u, v) is a back edge （ (u, v) 是回边）

强连通分支(Strongly Connected Components)

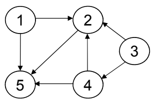
Given directed graph G = (V, E):

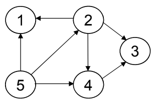
A **strongly connected component** (**SCC**) of G is a maximal set of vertices C ⊆ V such that for every pair of vertices u, v ∈ C, we have both u ⇨ v and v ⇨ u. (有向图G = (V, E)中的强连通分支是它的顶点的极大集C ⊆ V， 在该集中，每一对顶点u, v ∈ C有u ⇨ v 和 v ⇨ u）



图的转置(The Transpose of a Graph)

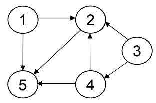
- G^T = **transpose** of G
  - G^T is G with all edges reversed （所有边转向）
  - $G^T = (V, E^T)$, $E^T = \{(u, v) : (v, u) \in E\}$
- If using adjacency lists: we can create G^T in $\Theta(V + E)$ time (应用临接表，转置时间)

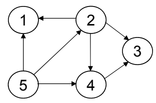




确定强连通分支(Finding the SCC)

- **Observation:** G and G^T have the same SCC's(G^T和G有相同的SCC)
  - u and v are reachable from each other in G ⇔ they are reachable from each other in G^T
- Idea for computing the SCC of a DAG G = (V, E): (有向无回路图的SCC确定方法)
  - Make two depth first searches: one on G and one on G^T (在G和G^T做深度搜索)

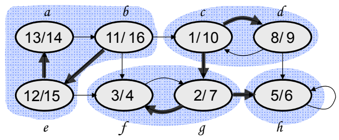




STRONGLY-CONNECTED-COMPONENTS(G)

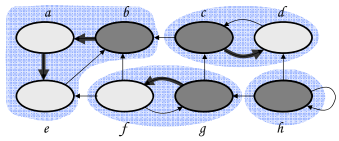
1. call DFS(G) to compute finishing times f[u] for each vertex u (在图上执行深度优先搜索，计算每个顶点u的完成时刻f[u])
2. compute G^T (构成转置图G^T)
3. call DFS(G^T), but in the main loop of DFS, consider vertices in order of decreasing f[u] (as computed in first DFS) (从具有最大f[u]的顶点开始，在G^T上执行深度优先搜索，如果深度优先搜索不能到达所有的顶点，则在余下的顶点中找一个f[u]最大的顶点，开始下一次深度优先搜索）
4. output the vertices in each tree of the depth-first forest formed in second DFS as a separate SCC（在最终得到的森林中的每一棵树对应一个强连通分支）

Example



DFS on the initial graph G

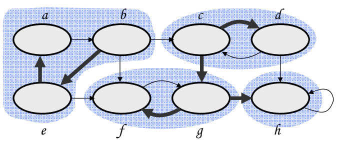
b	e	a	c	d	g	h	f
16	15	14	10	9	7	6	4

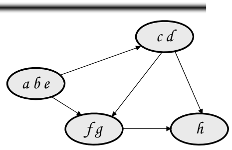


DFS on G^T:
- start at b: visit a, e
- start at c: visit d
- start at g: visit f
- start at h

Strongly connected components: $C_1 = \{a, b, e\}$, $C_2 = \{c, d\}$, $C_3 = \{f, g\}$, $C_4 = \{h\}$

分支图（Component Graph）





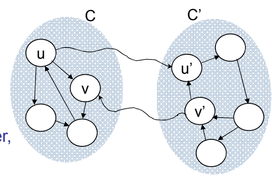
- The **component graph** $G^{SCC} = (V^{SCC}, E^{SCC})$: （表示方法）
  - $V^{SCC} = \{v_1, v_2, \ldots, v_k\}$, where v_i corresponds to each strongly connected component C_i
  - There is an edge $(v_i, v_j) \in E^{SCC}$ if G contains a directed edge (x, y) for some $x \in C_i$ and $y \in C_j$
- The component graph is a DAG （分支图是一个有向无回路图）

引理1（Lemma 1）

Let C and C' be distinct SCC's in G （ C和C'是图G的SCC）
Let u, v ∈ C, and u', v' ∈ C'
Suppose there is a path u ⇨ u' in G （假设图G中存在通路u ⇨ u' ）
Then there cannot also be a path v' ⇨ v in G.
（则不可能有v' ⇨ v ）

Proof (反证法)

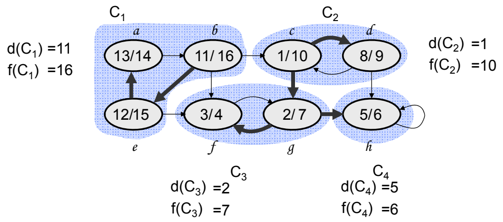
- Suppose there is a path v' ⇨ v (假设有v' ⇨ v)
- There exists u ⇨ u' ⇨ v'
- There exists v' ⇨ v ⇨ u
- u and v' are reachable from each other, so they are not in separate SCC's: contradiction! (和双向可连通，与他们分别在不同的SCC相矛盾）



符号标记（Notations）

- Extend notation for d (starting time) and f (finishing time) to sets of vertices $U \subseteq V$:
  - $d(U) = \min_{u \in U} \{ d[u] \}$ (earliest discovery time)（最早发现时间）
  - $f(U) = \max_{u \in U} \{ f[u] \}$ (latest finishing time)（最晚结束时间）

（用于SCC）

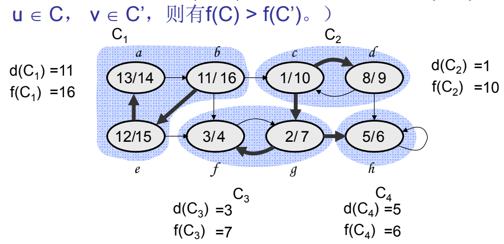


49

引理2（Lemma 2）

- Let C and C' be distinct SCCs in a directed graph G = (V, E). If there is an edge $(u, v) \in E$, where $u \in C$ and $v \in C'$ then $f(C) > f(C')$.

（C和C'是有向图G = (V, E)的SCC，如果存在边$(u, v) \in E$，$u \in C$， $v \in C'$，则有$f(C) > f(C')$。）



50