Graph Traversals: Applications

Topics

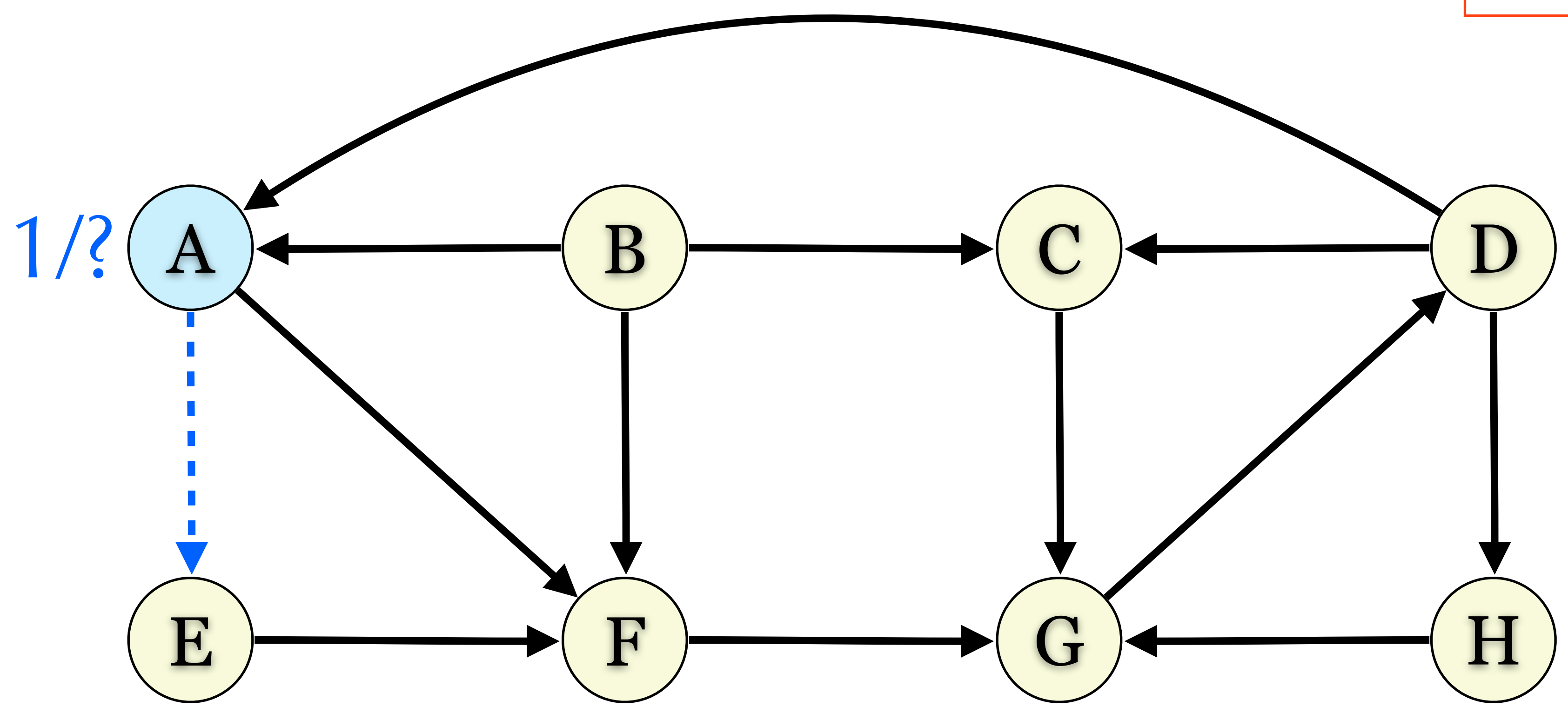
- Cycle detection
- Topological sort
- Shortest paths in DAG
- Shortest paths in unweighted graph
- Bipartite graph check
- Tree diameter
- Connected components
- Finding articulation points and bridges

Cycle Detection

- A cycle $c=\langle v_0,v_1,\ldots,v_k=v_0\rangle$ is in $G=(V,E)$ if $(v_{i-1},v_i)\in E$ for $i\in\{1,\ldots,k\}$.
- How to detect a cycle in G?
- A DFS-based algorithm: check if there is a back edge (u,v)
  - While visiting u, we check (u,v) and we find v is discovered but not visited.

DFS-Based

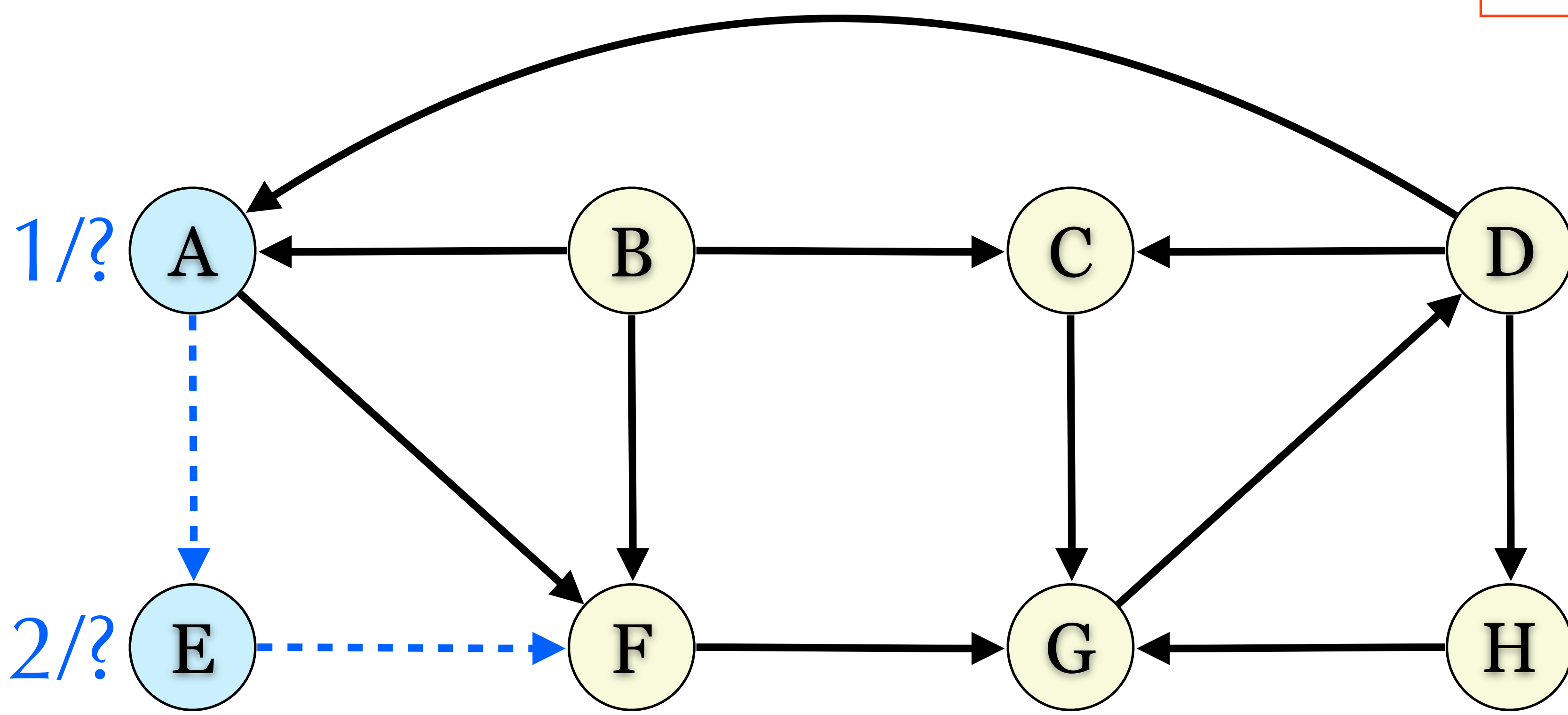
Starting at A



Stack								
	A/0							

DFS-Based

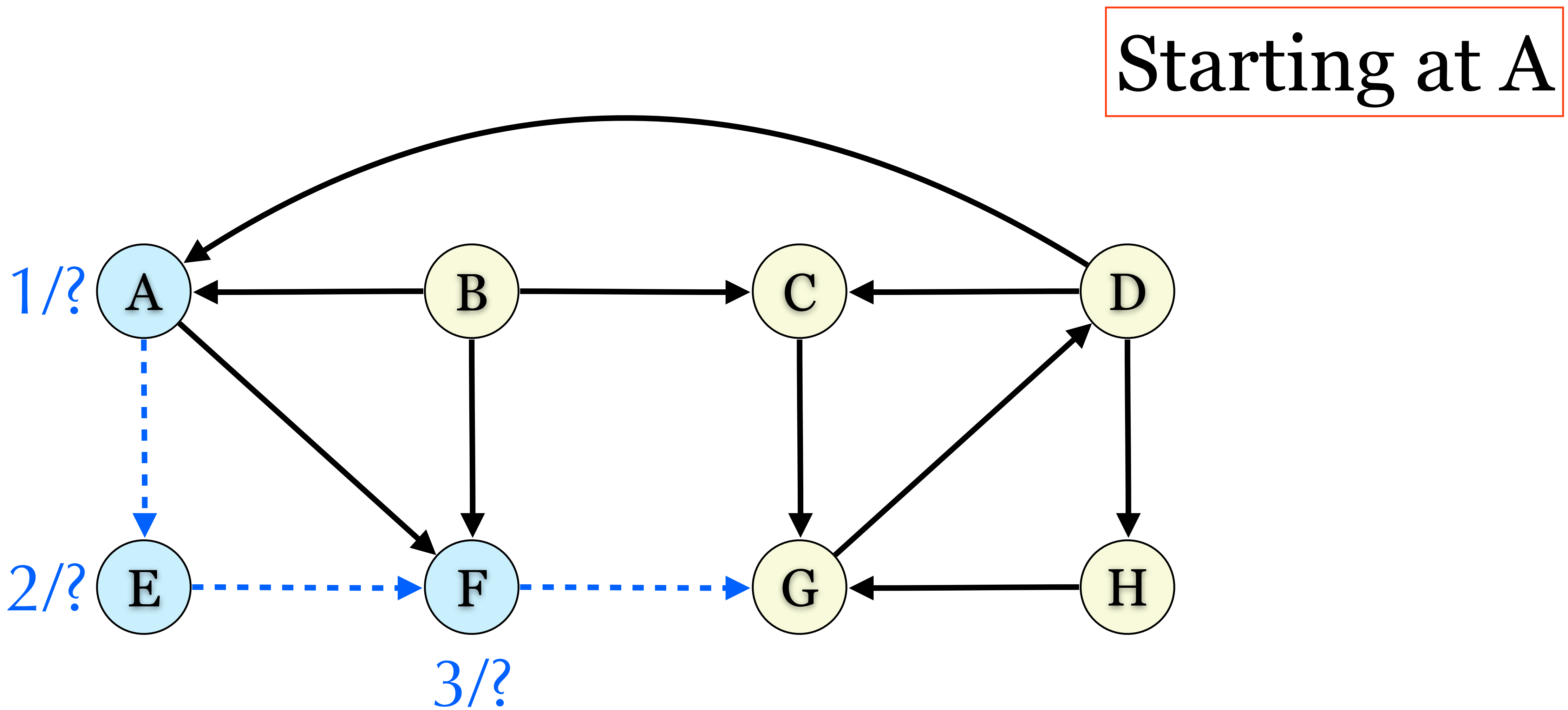
Starting at A



Stack

A/o	E/o						

DFS-Based

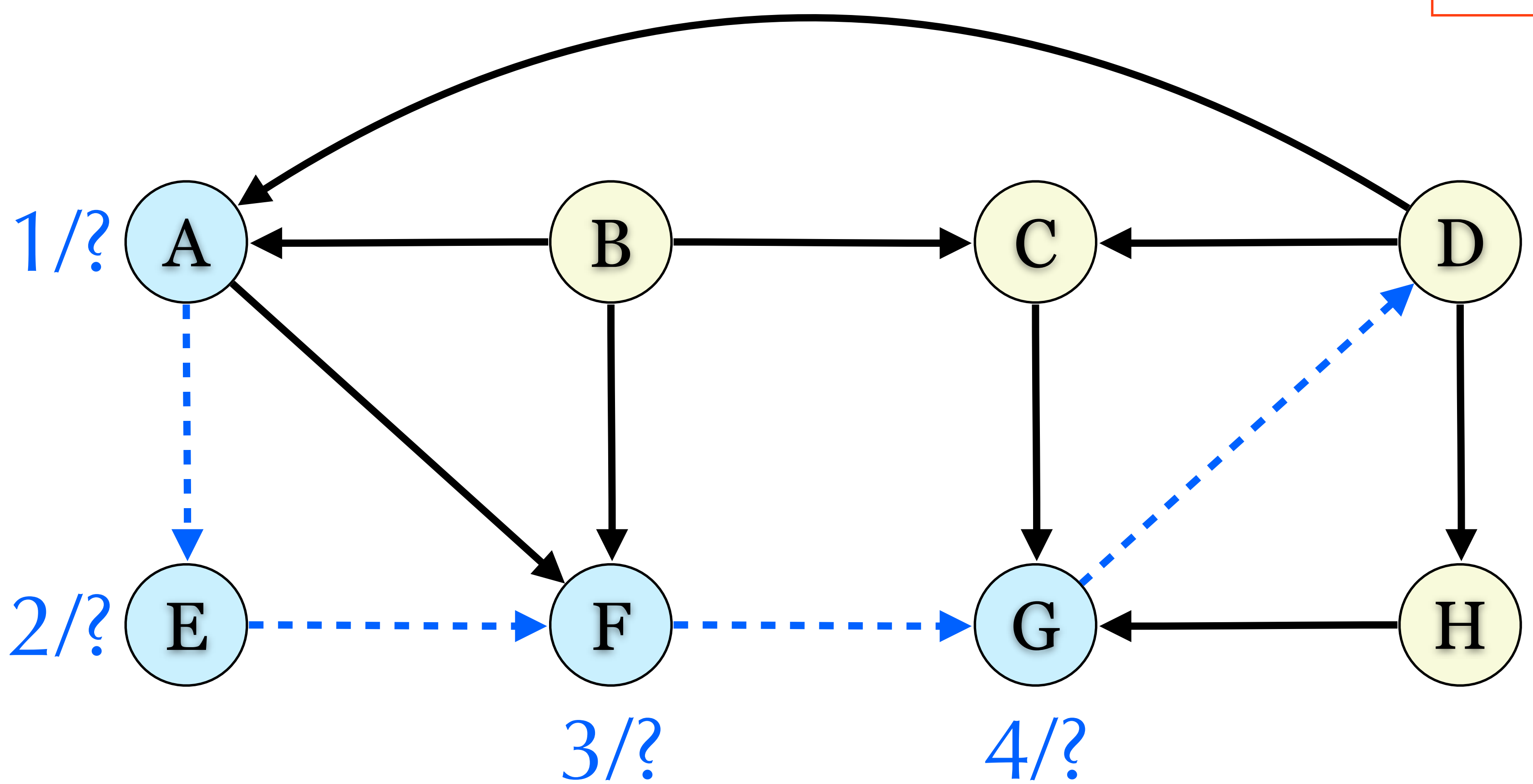


Stack

A/o	E/o	F/o					

DFS-Based

Starting at A

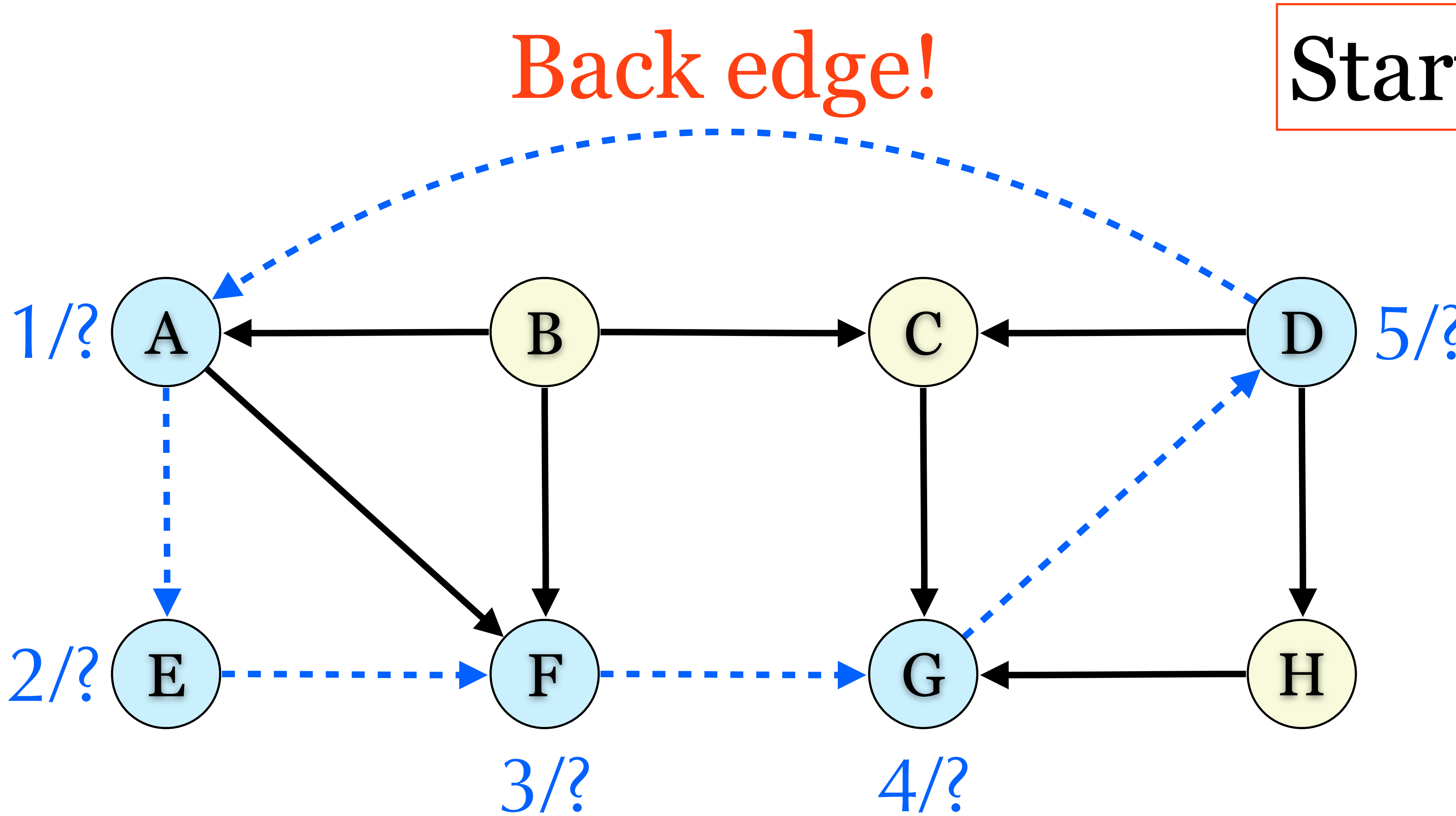


Stack

A/o	E/o	F/o	G/o				

DFS-Based

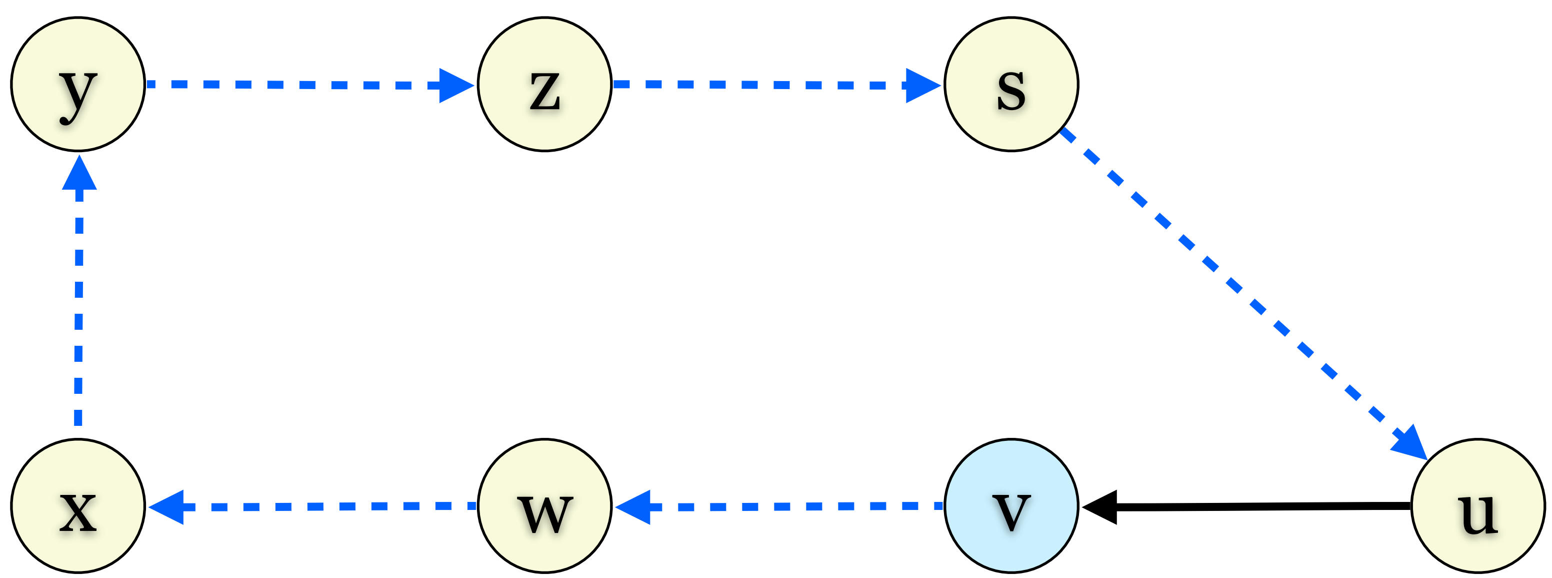
Starting at A



Stack

A/o	E/o	F/o	G/o	D/o			

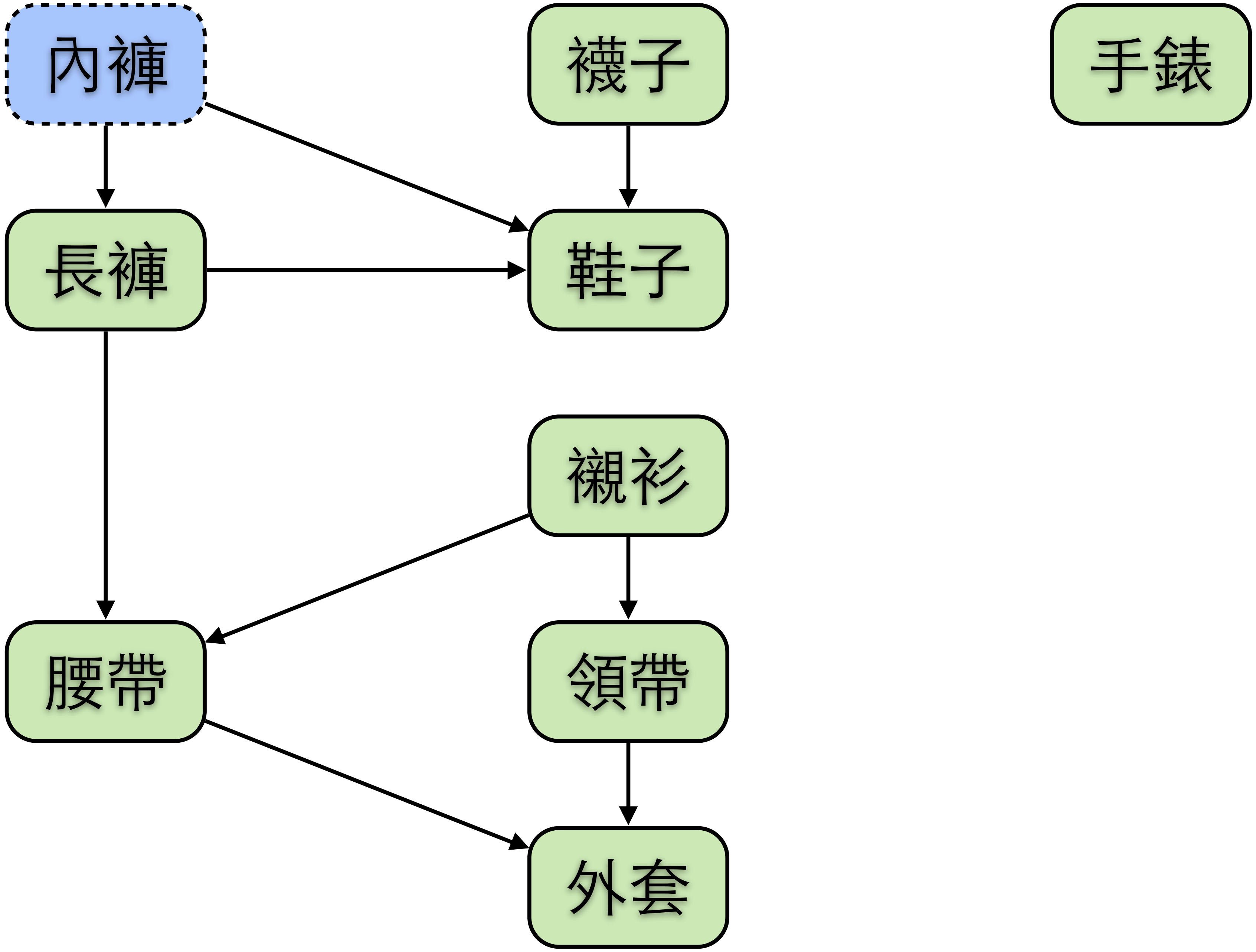
DFS-Based

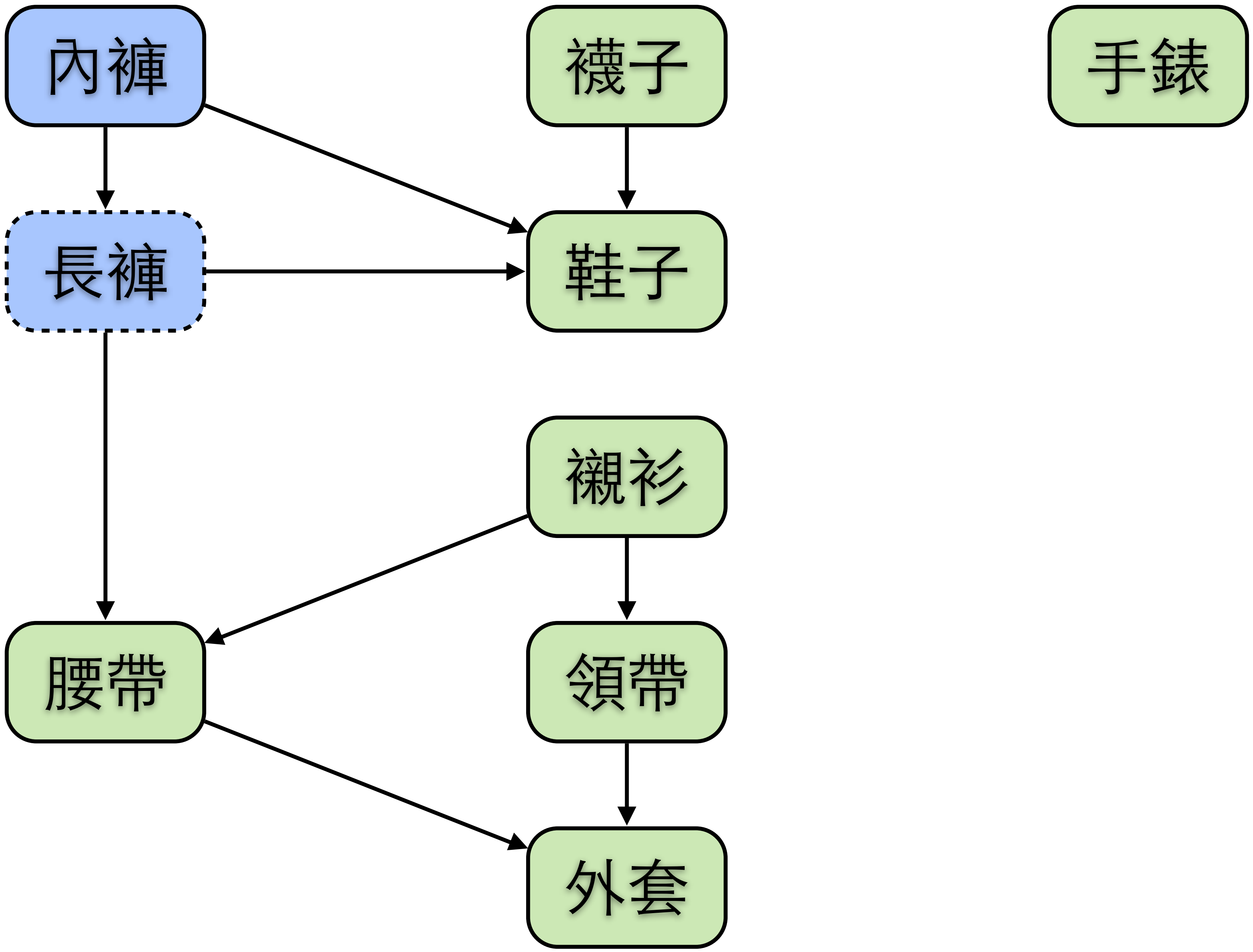


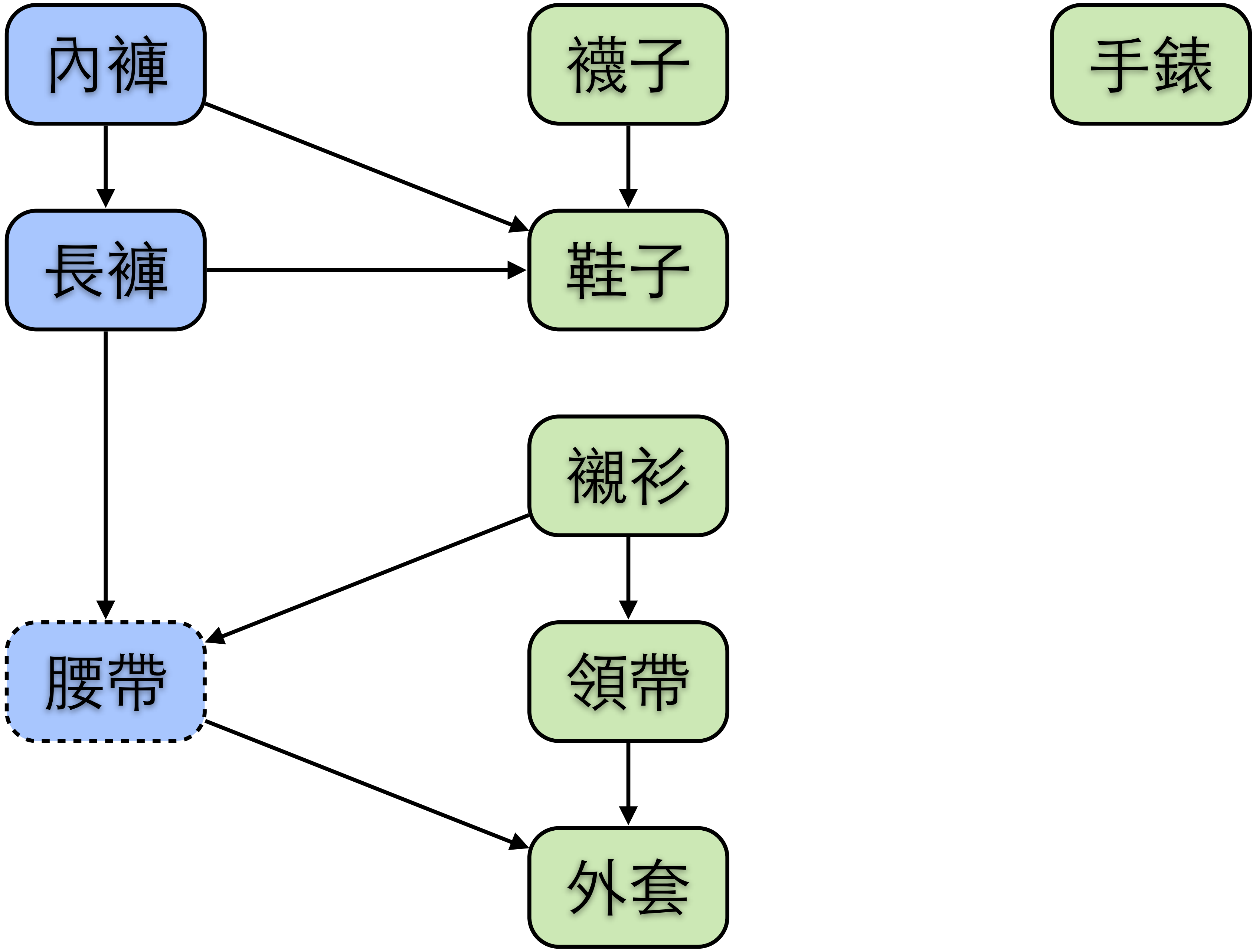
If there is a cycle, and the first discovered vertex is u, then (u,v) must be a back edge.

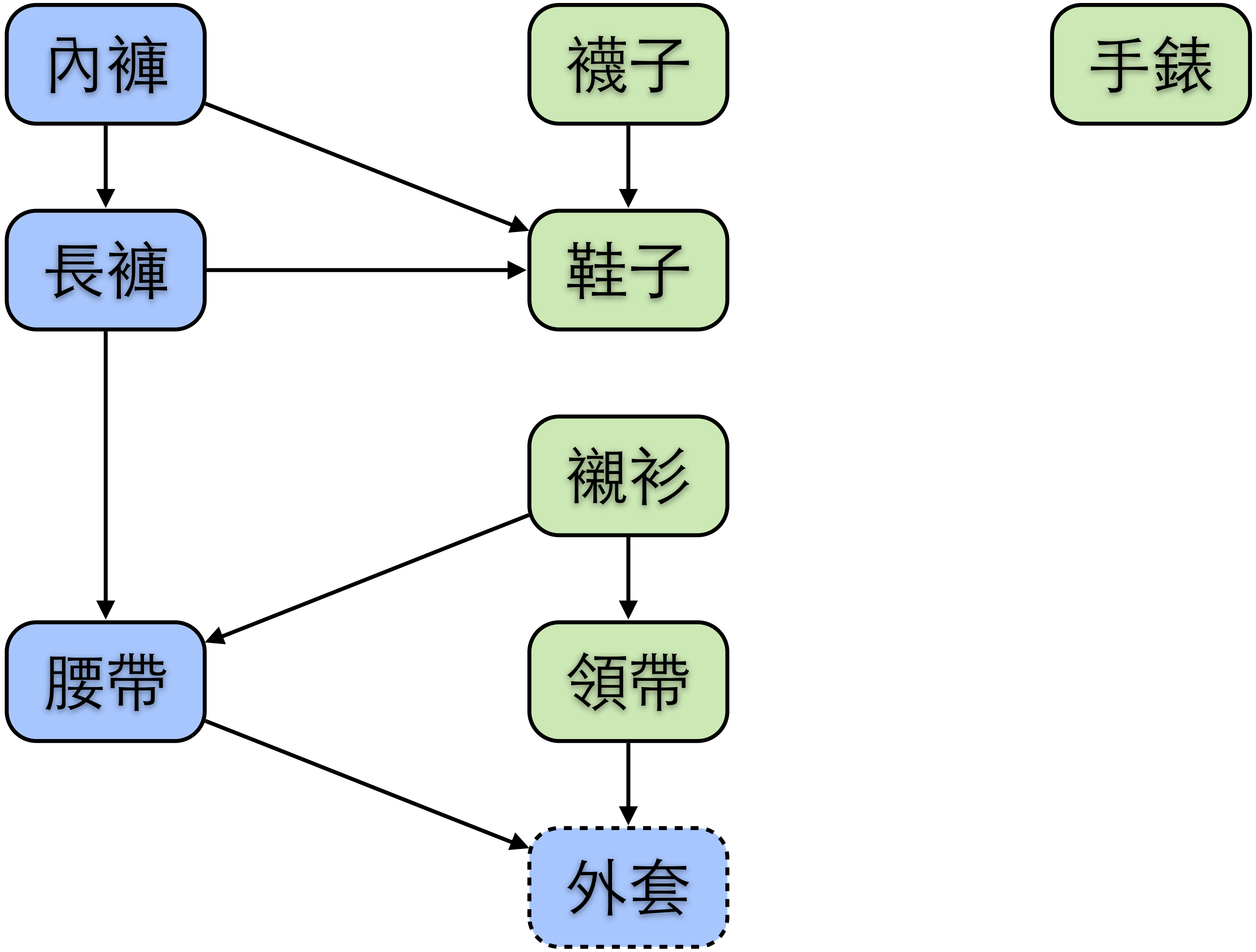
Topological Sort

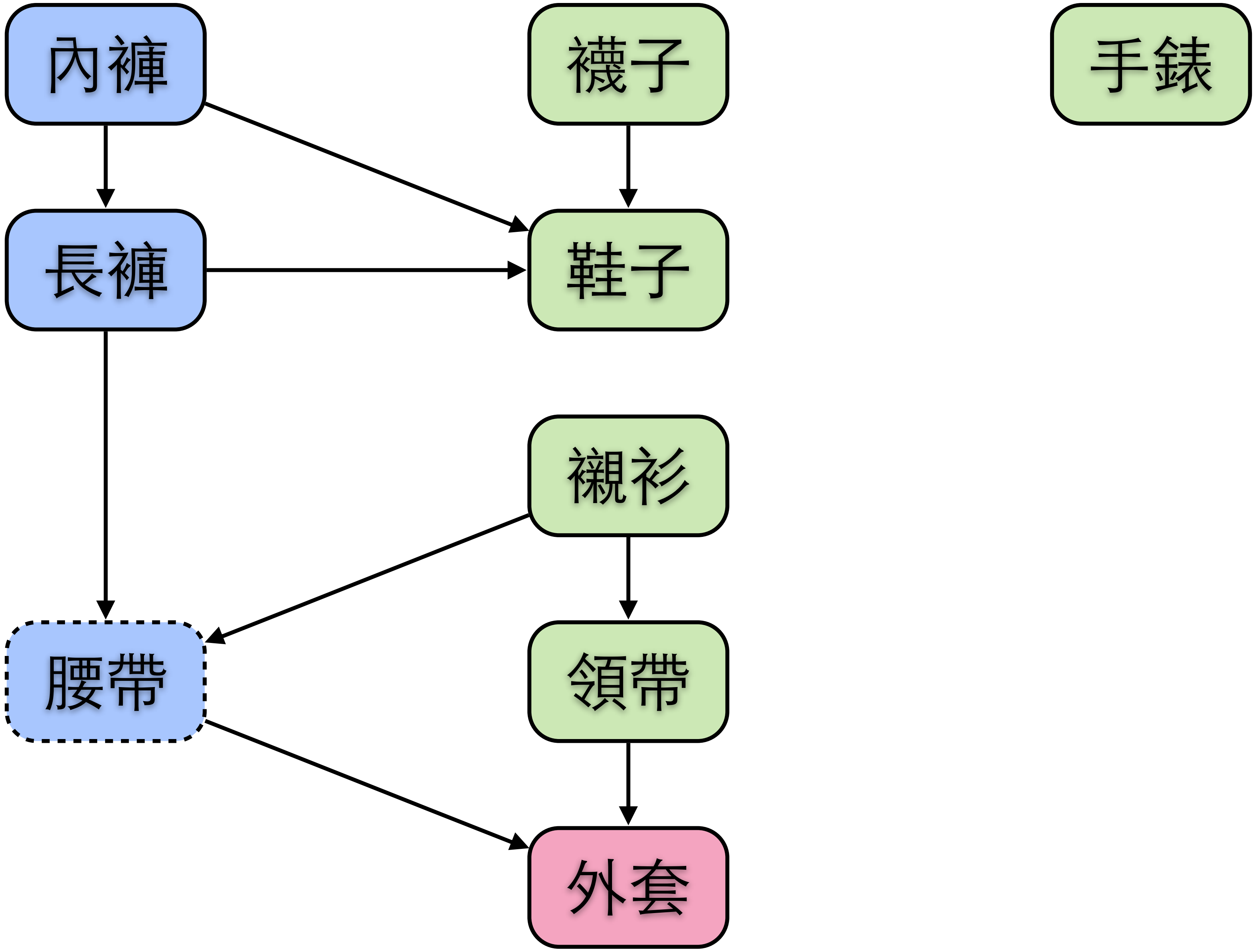
- A topological sort of a directed acyclic graph (DAG) G=(V,E) is a linear ordering of V such that if G contains an edge (u,v), then u appears before v in the ordering.
- Algorithm: O(|V|+|E|)
  - Run DFS on G. abort if any back edge exists
    - Once a vertex v is visited, prepend it to a linked list.

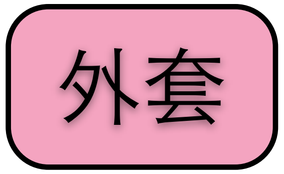

內褲
襪子
手錶
長褲
鞋子
襯衫
腰帶
領帶
外套

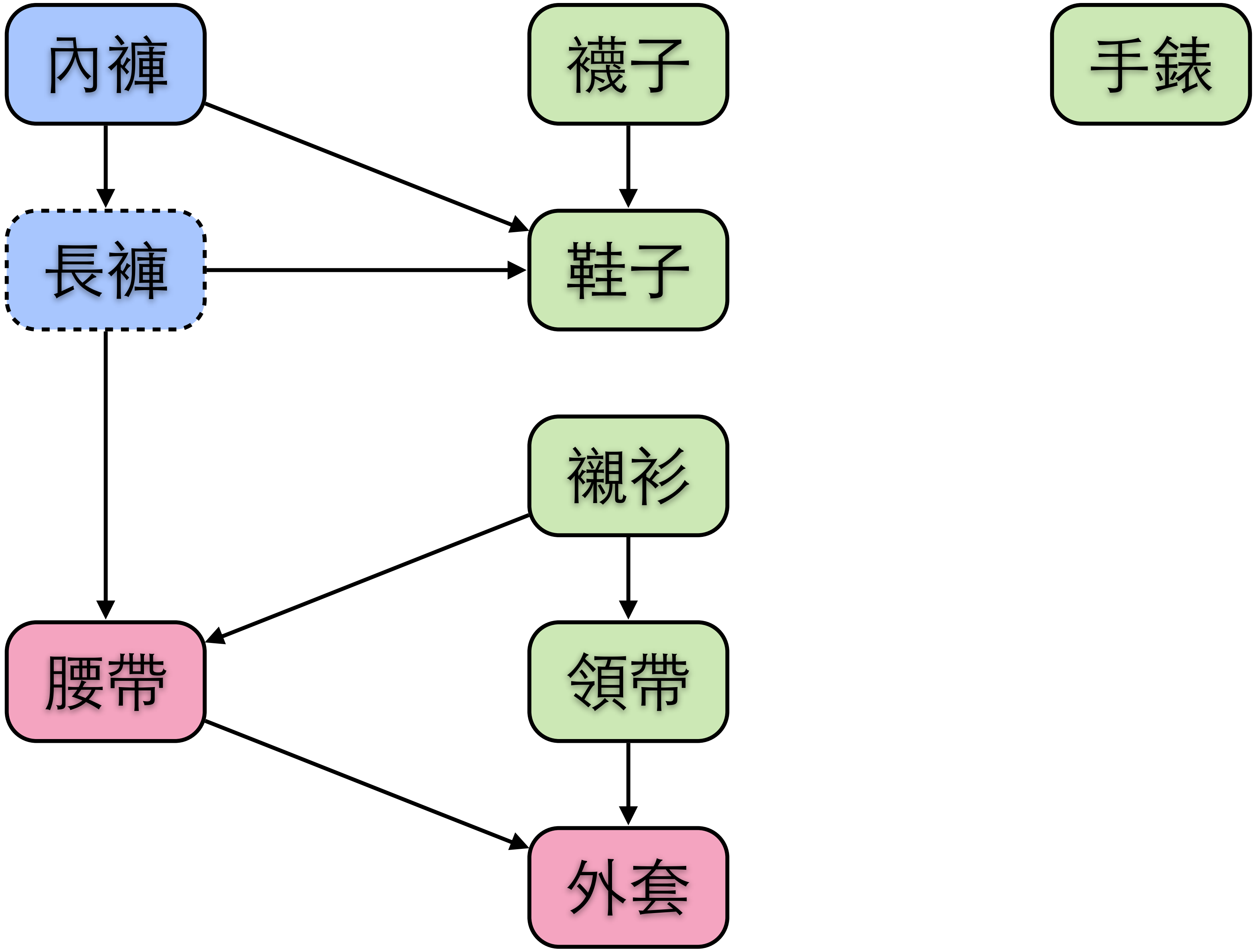

內褲
襪子
手錶
長褲
鞋子
襯衫
腰帶
領帶
外套

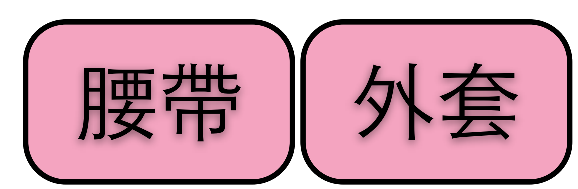

內褲
襪子
手錶
長褲
鞋子
襯衫
腰帶
領帶
外套

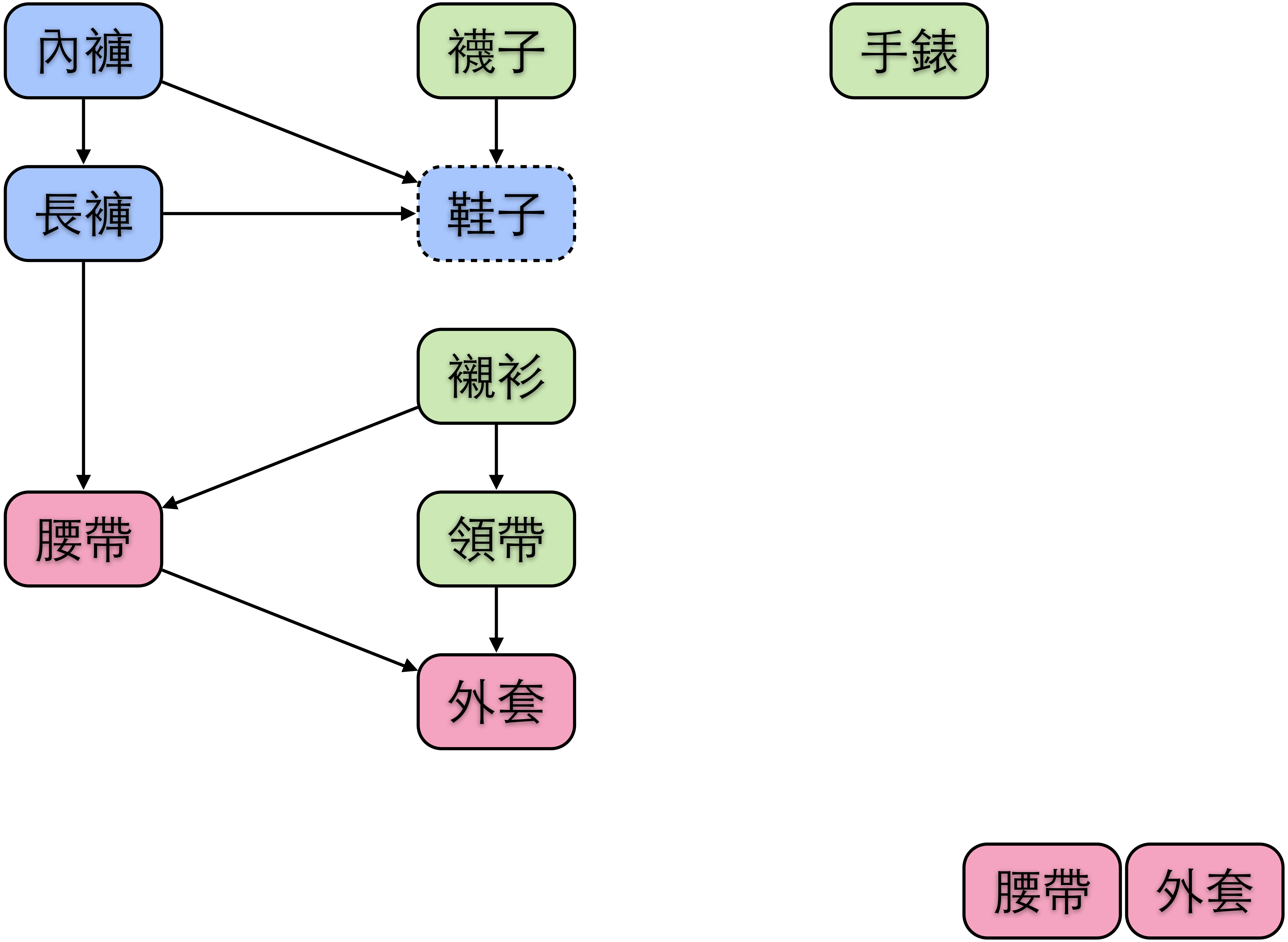

內褲
襪子
手錶
長褲
鞋子
襯衫
腰帶
領帶
外套

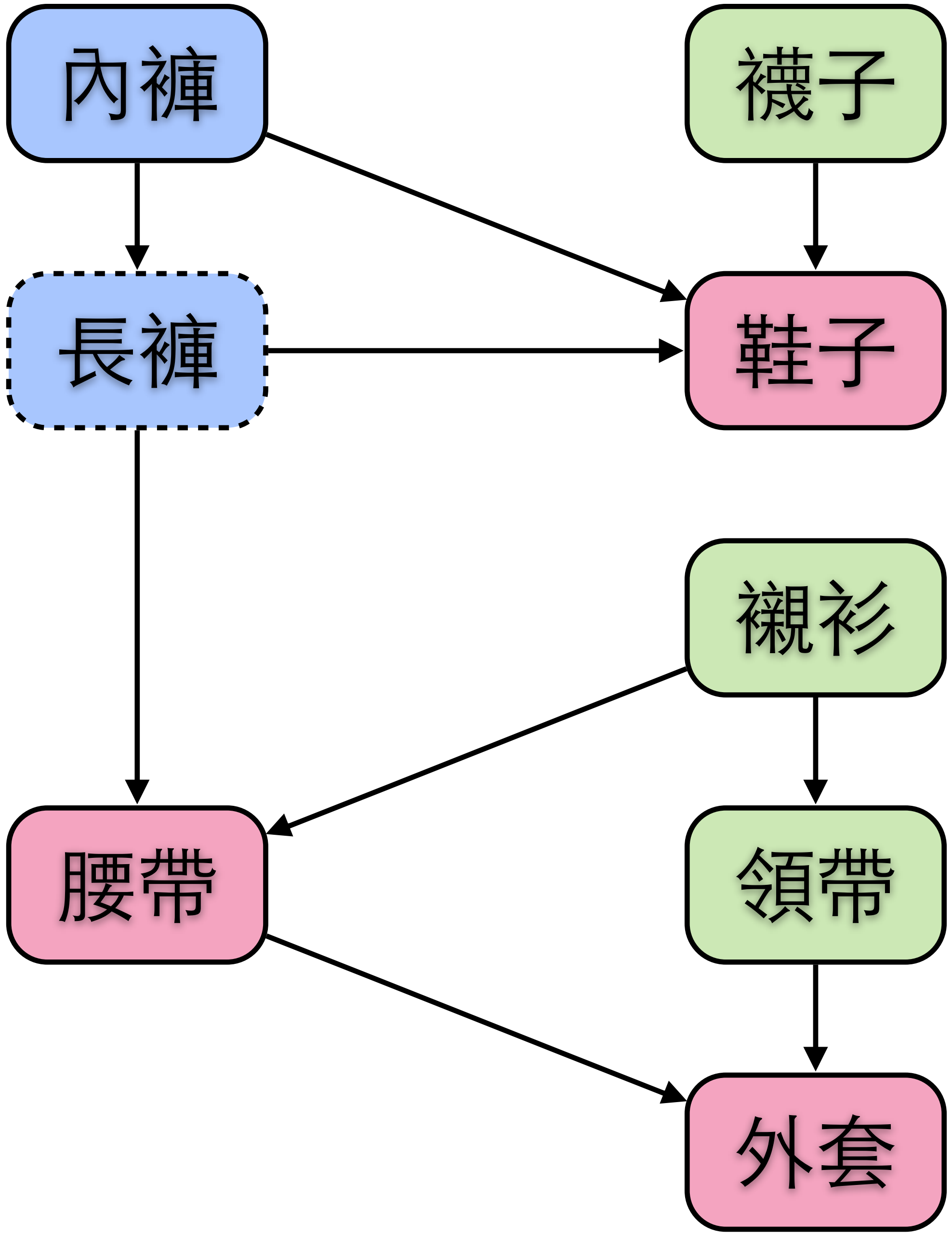

內褲
襪子
手錶
長褲
鞋子
襯衫
腰帶
領帶
外套

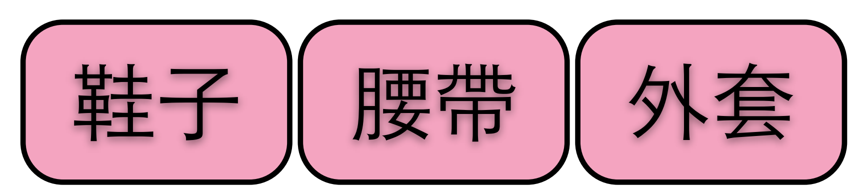

外套

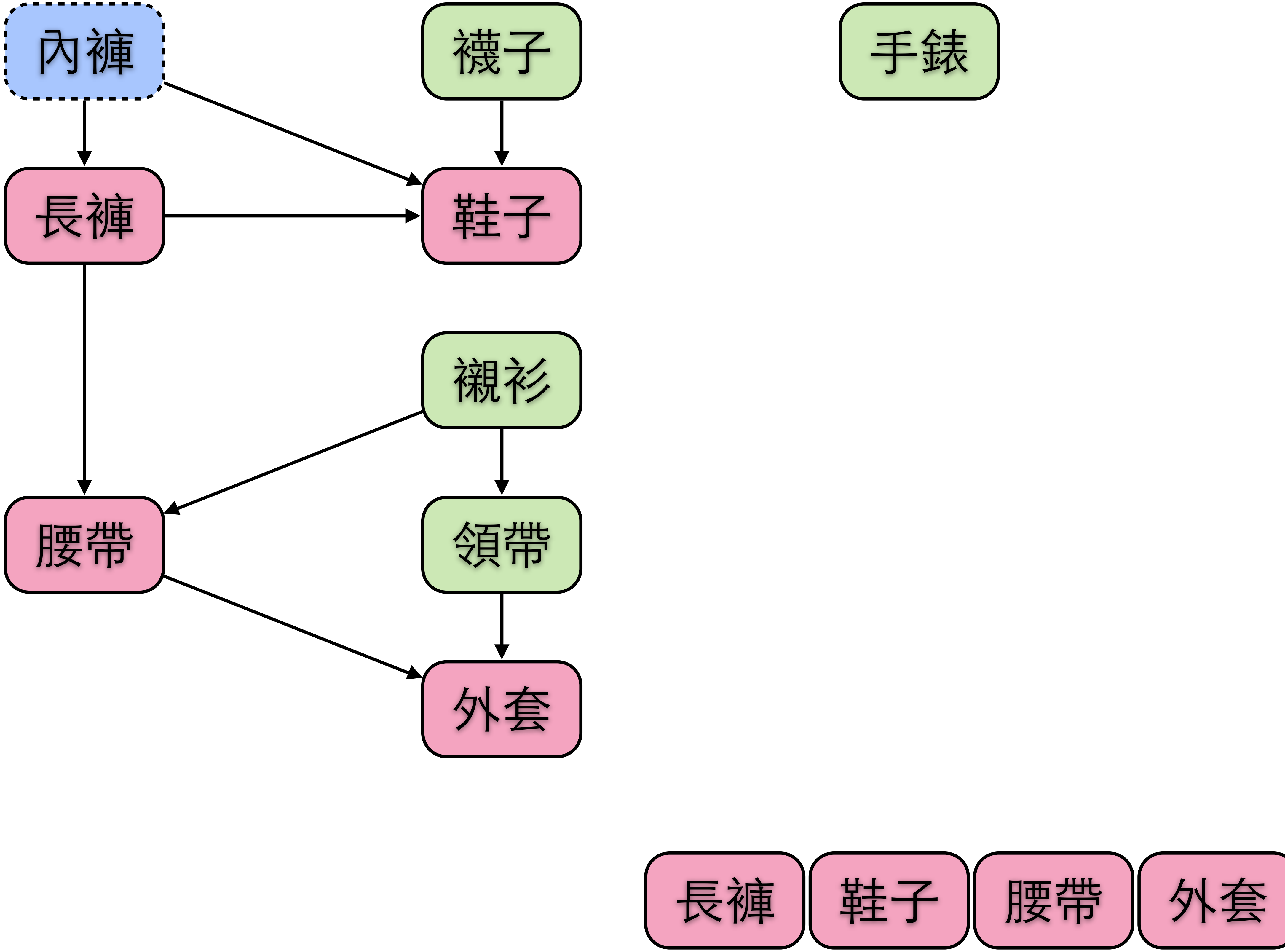

內褲
襪子
手錶
長褲
鞋子
襯衫
腰帶
領帶
外套

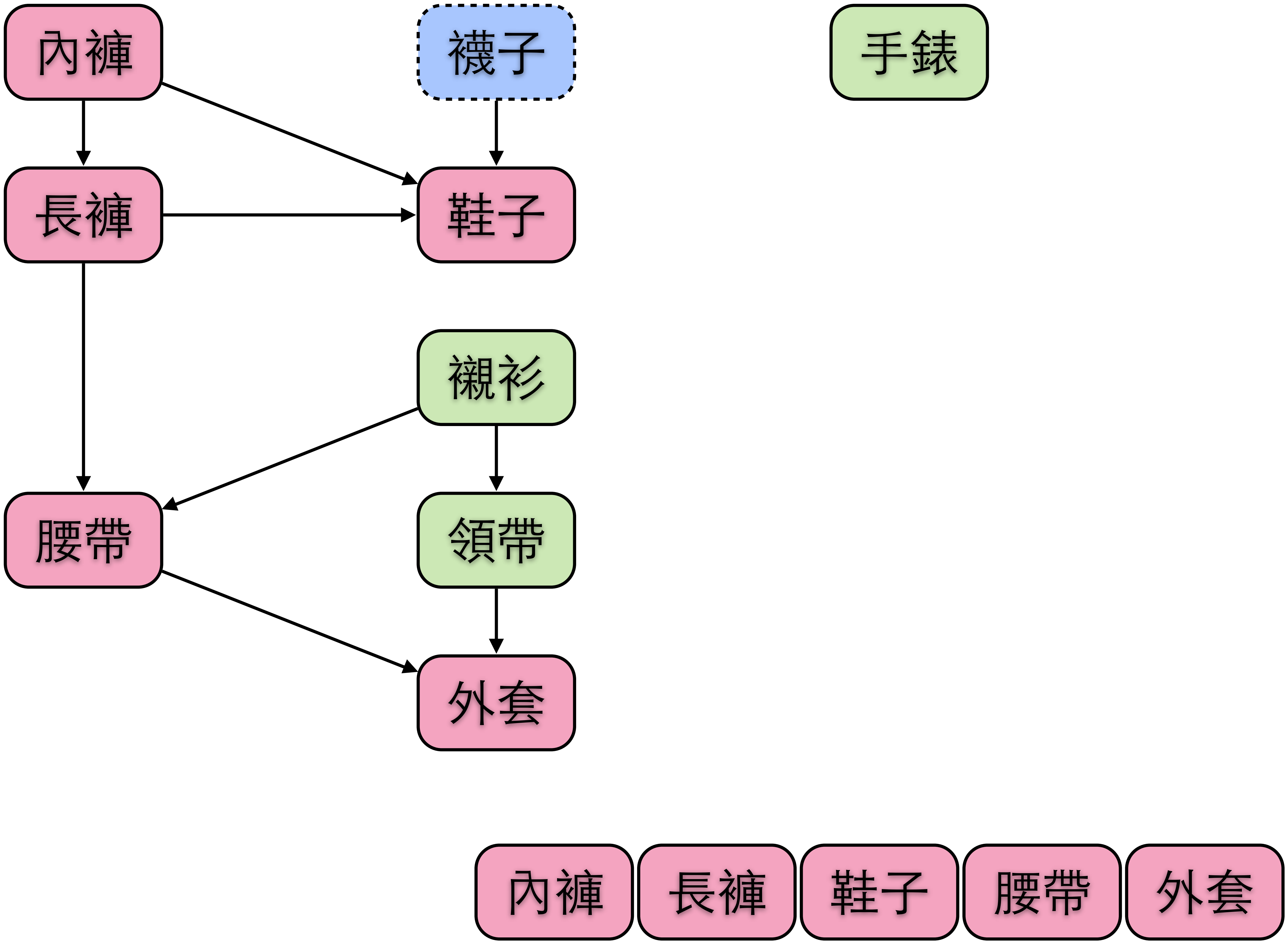

腰帶
外套

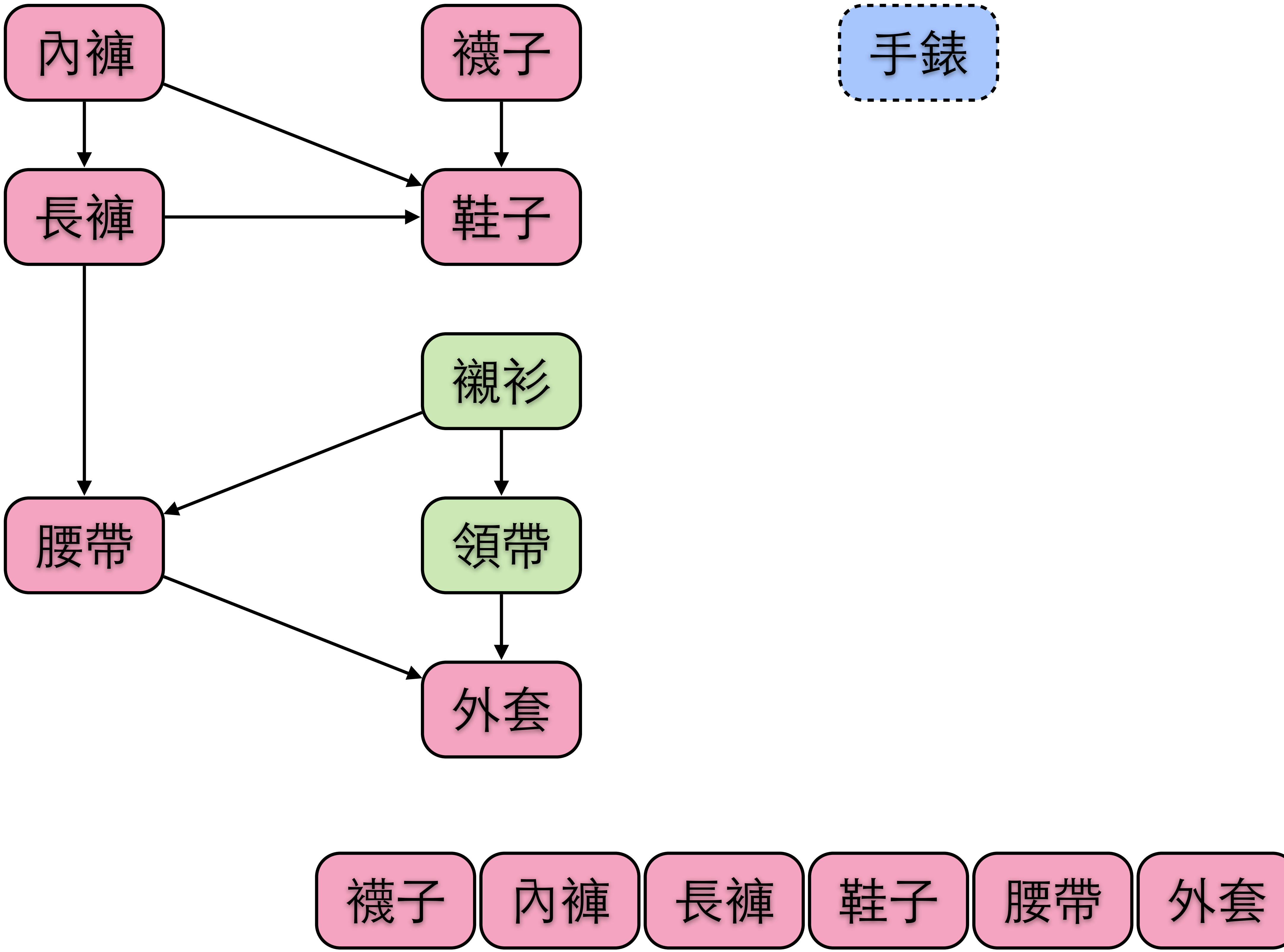

內褲
襪子
手錶
長褲
鞋子
襯衫
腰帶
領帶
外套
腰帶
外套

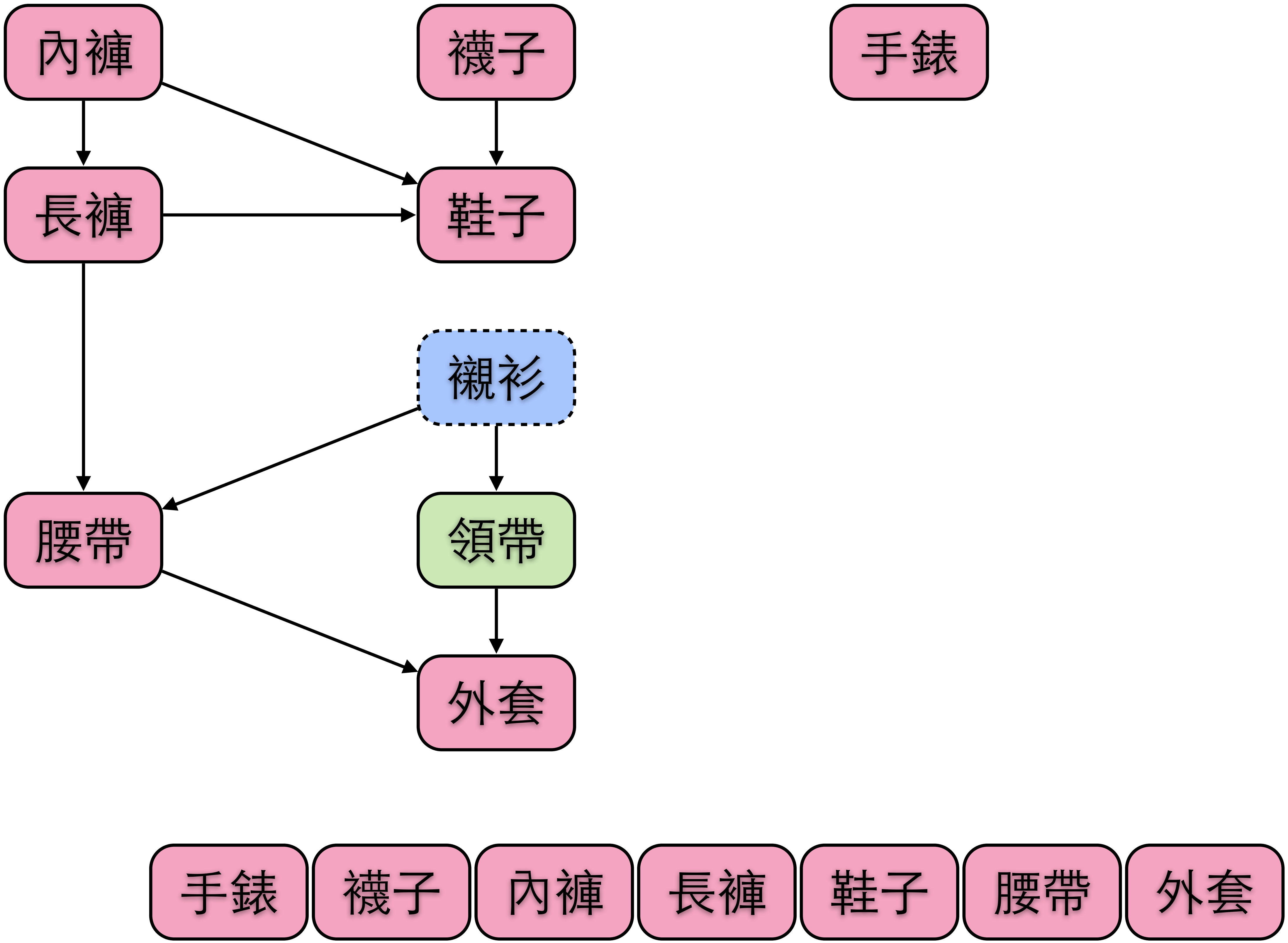

內褲
襪子
手錶
長褲
鞋子
襯衫
腰帶
領帶
外套

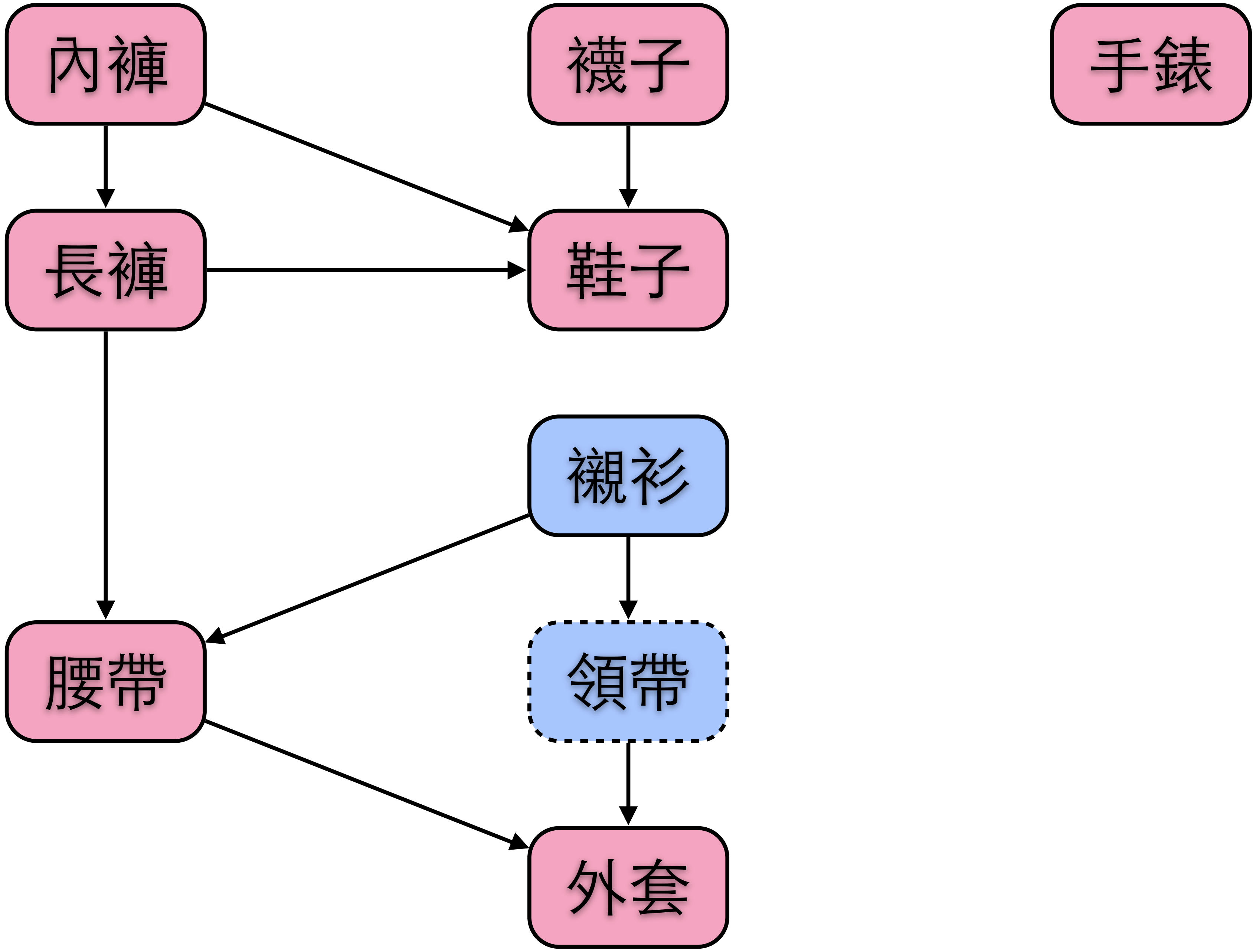

鞋子
腰帶
外套


內褲
襪子
手錶
長褲
鞋子
襯衫
腰帶
領帶
外套
長褲
鞋子
腰帶
外套

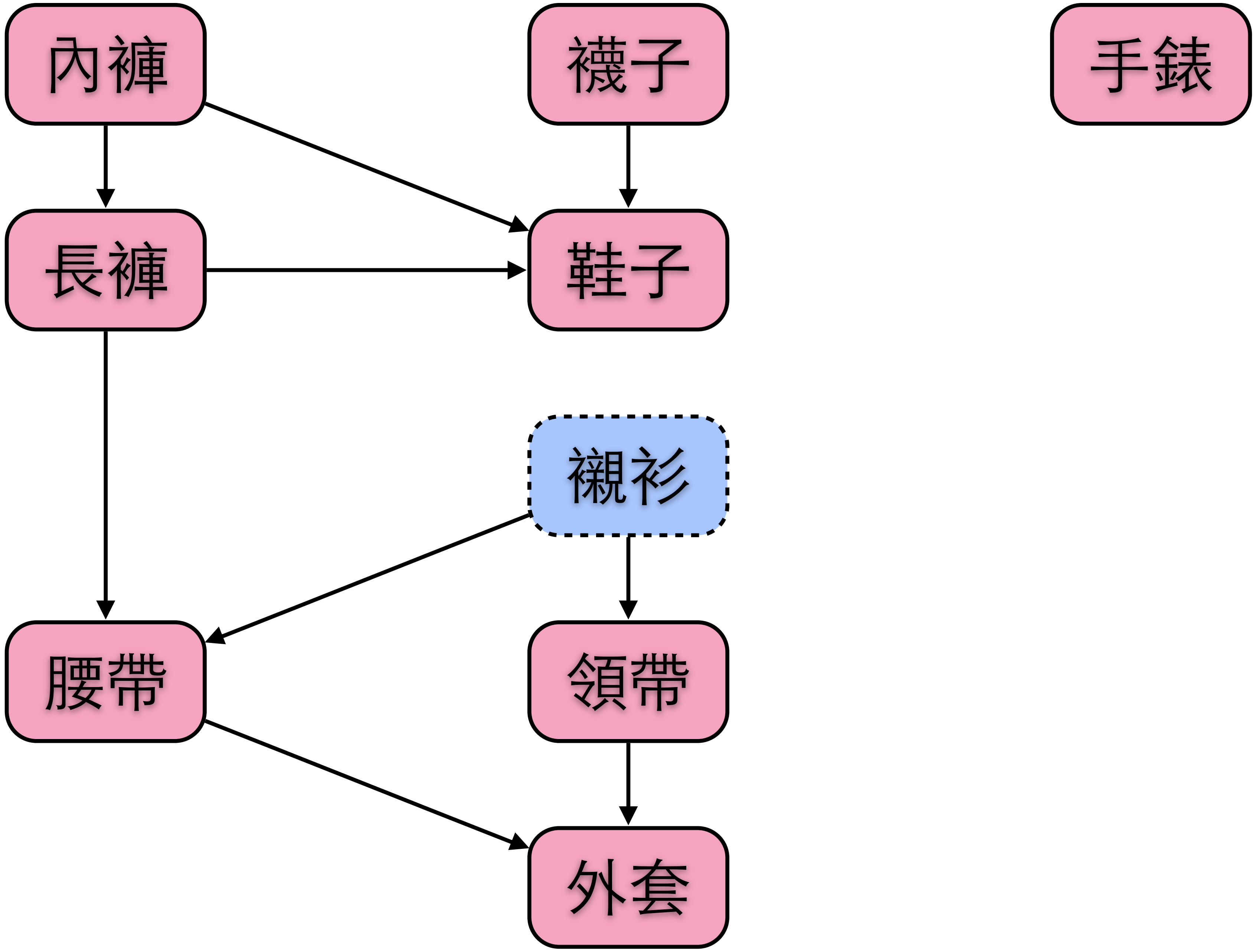

內褲
襪子
手錶
長褲
鞋子
襯衫
腰帶
領帶
外套
內褲
長褲
鞋子
腰帶
外套

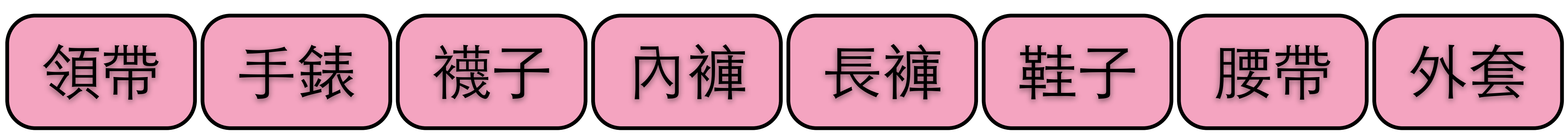

內褲
襪子
手錶
長褲
鞋子
襯衫
腰帶
領帶
外套
襪子
內褲
長褲
鞋子
腰帶
外套

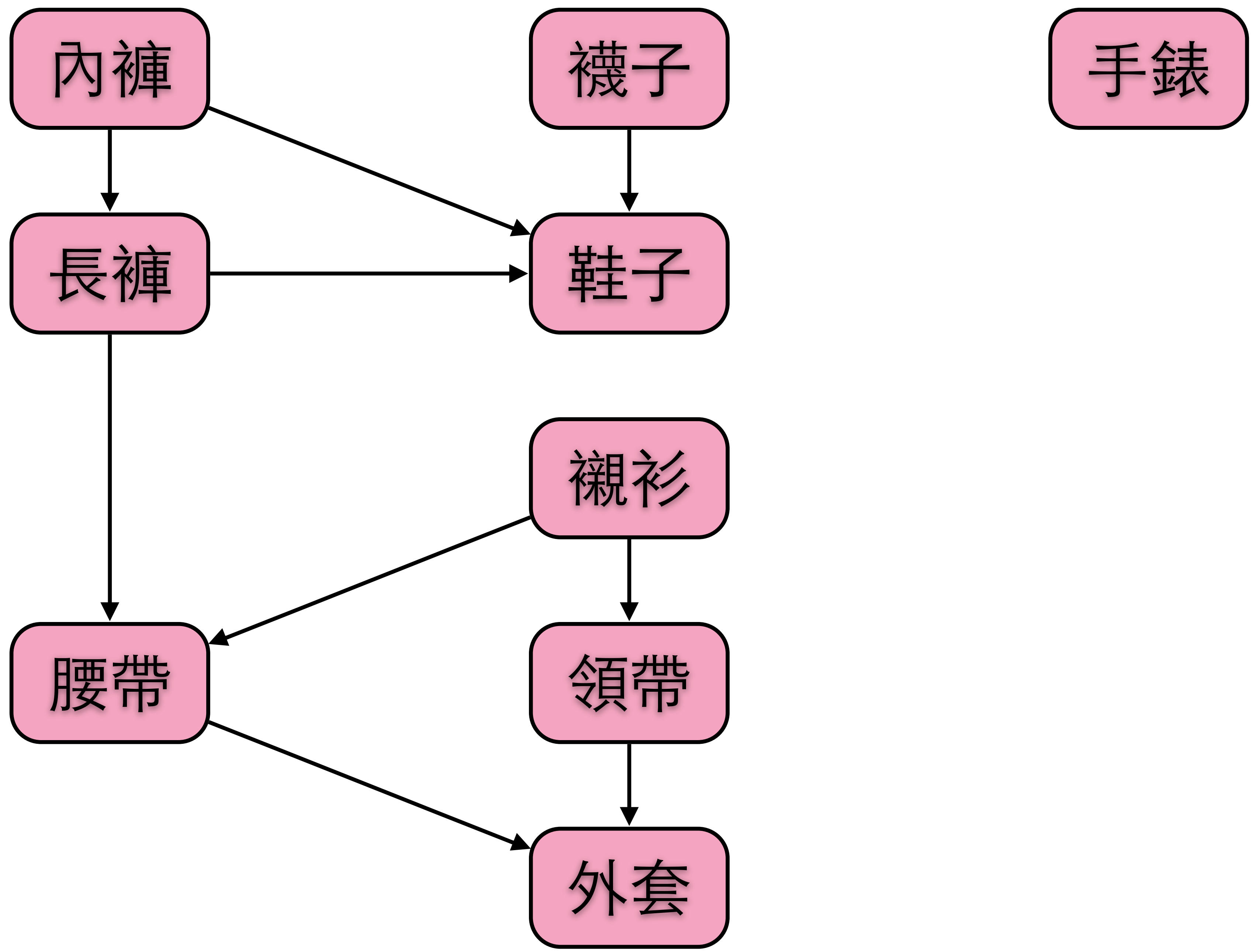

內褲
襪子
手錶
長褲
鞋子
襯衫
腰帶
領帶
外套
手錶
襪子
內褲
長褲
鞋子
腰帶
外套


內褲
襪子
手錶
長褲
鞋子
襯衫
腰帶
領帶
外套


手錶
襪子
內褲
長褲
鞋子
腰帶
外套


內褲
襪子
手錶
長褲
鞋子
襯衫
腰帶
領帶
外套


領帶
手錶
襪子
內褲
長褲
鞋子
腰帶
外套

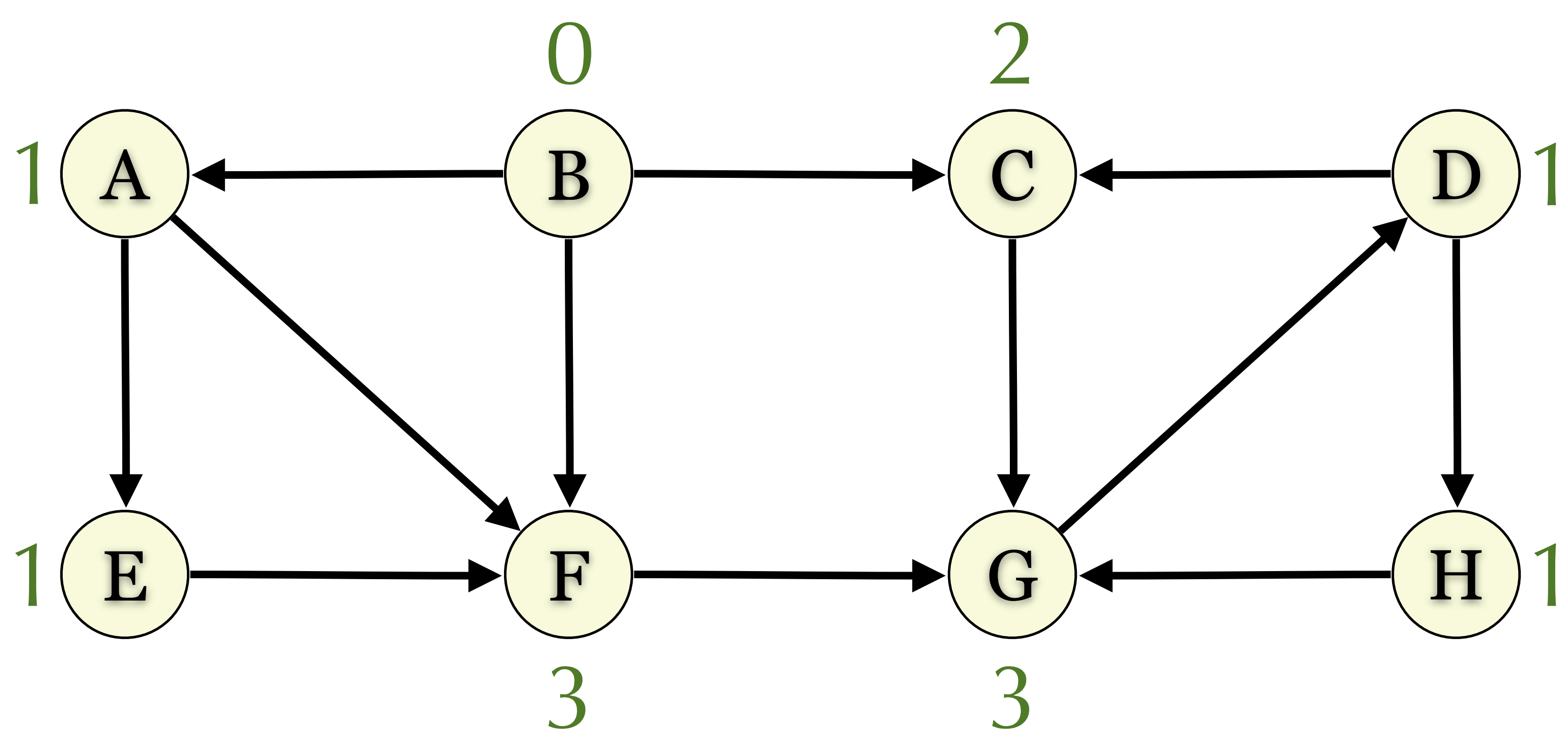

內褲
襪子
手錶
長褲
鞋子
襯衫
腰帶
領帶
外套


襯衫
領帶
手錶
襪子
內褲
長褲
鞋子
腰帶
外套

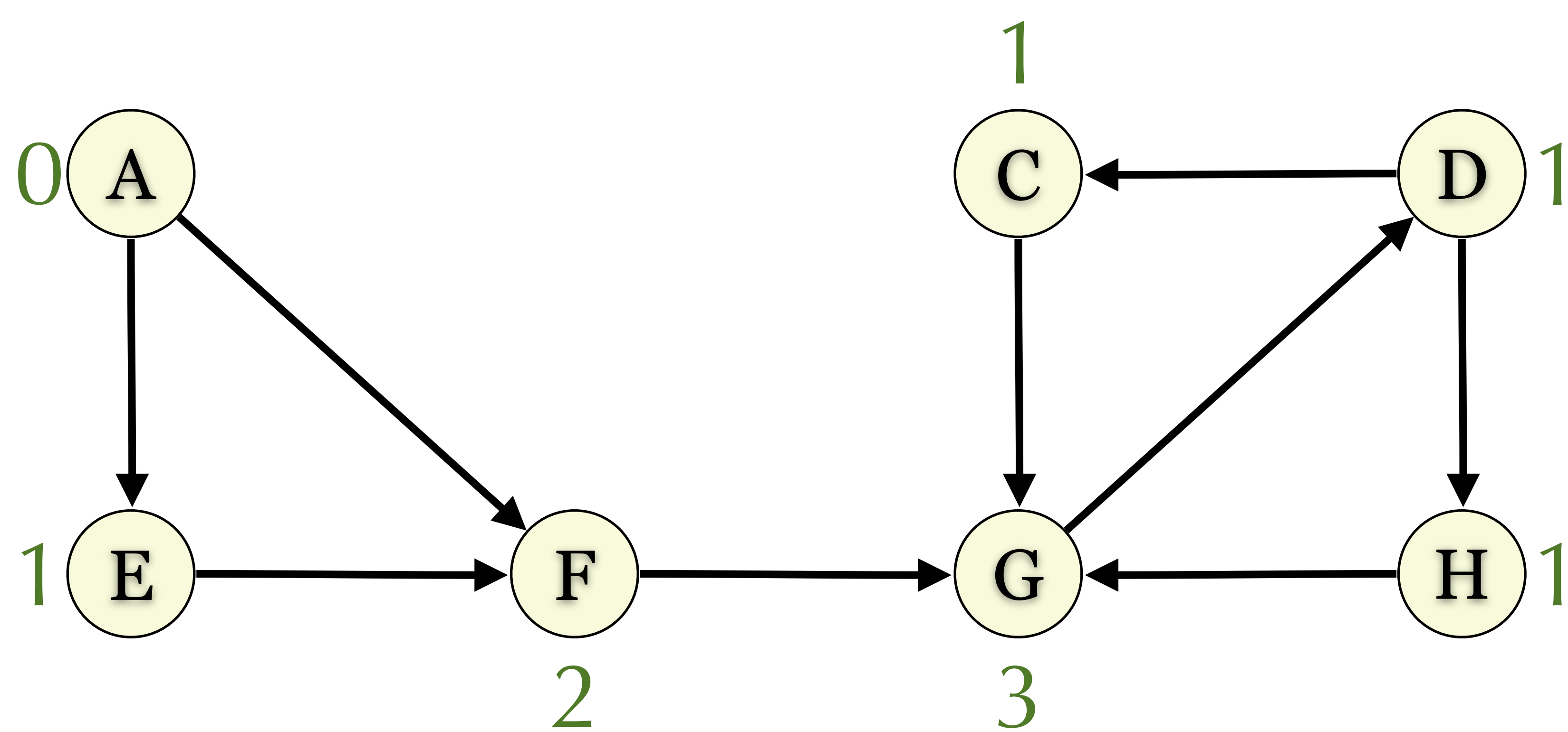
Alternative Solution

- Degree of v: #edge incident to v
- In-degree of v: $|\{(u,v):u\in V\}|$
- Out-degree of v: $|\{(v,u):u\in V\}|$
- Idea: Repeat to remove a vertex whose in-degree is 0.
- If there are no vertices left, then it is a DAG, and we removed the vertices in topological order.

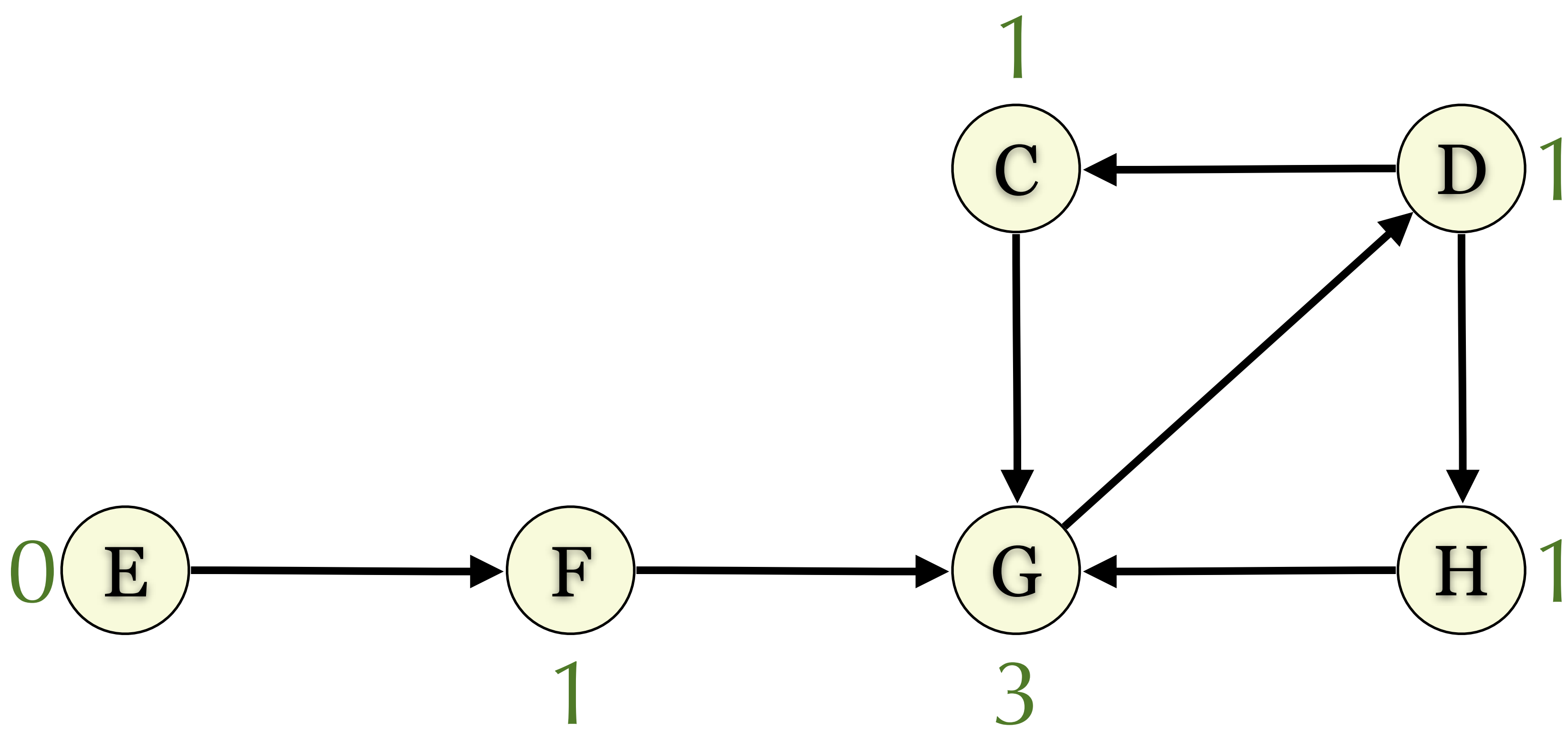
Cycle Detection



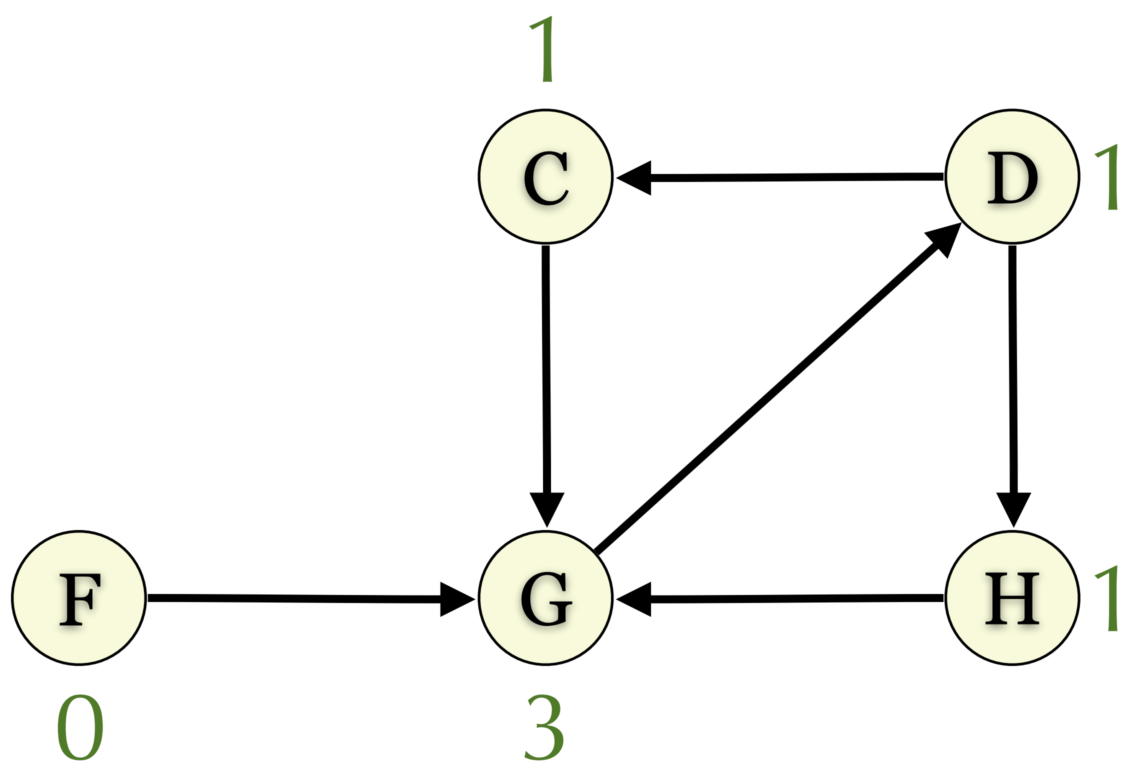
Cycle Detection



Cycle Detection

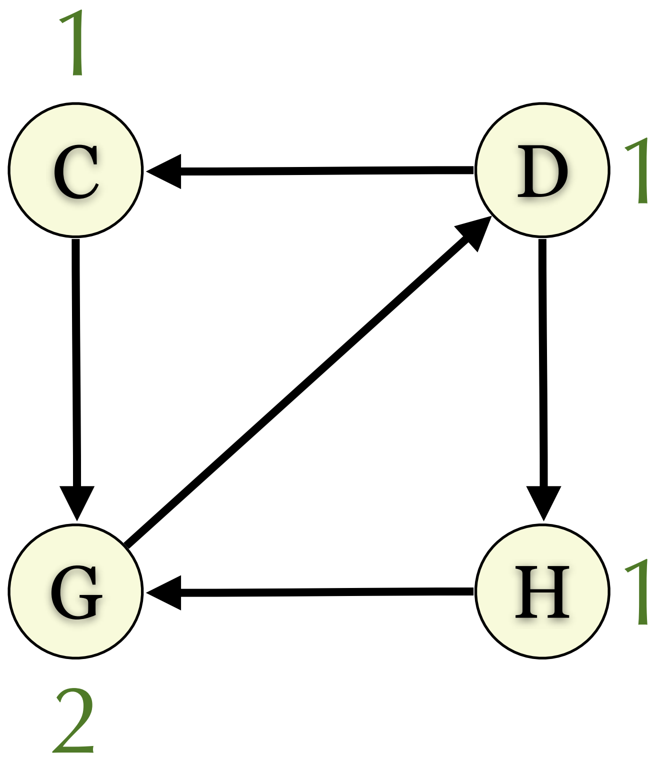


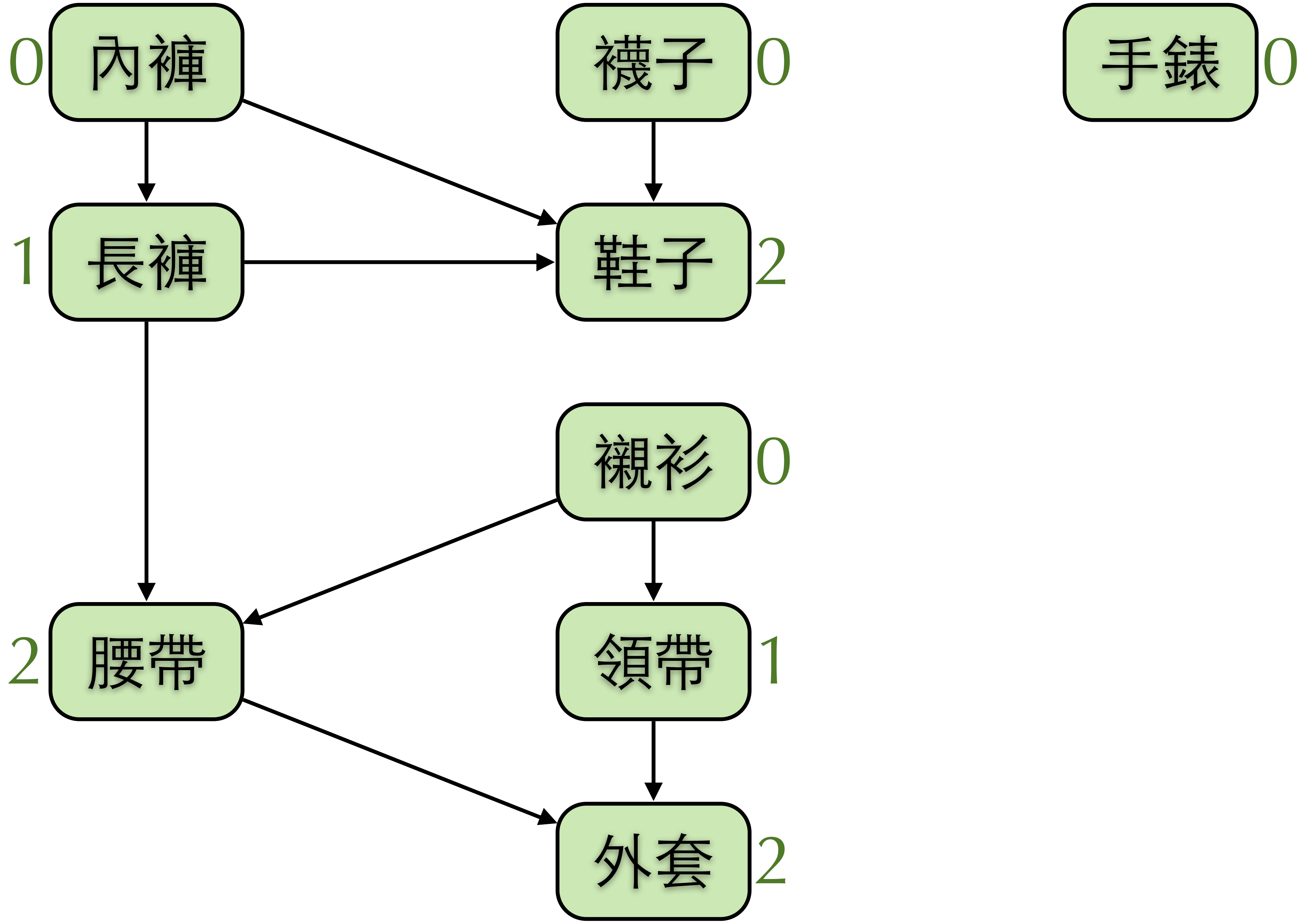
Cycle Detection

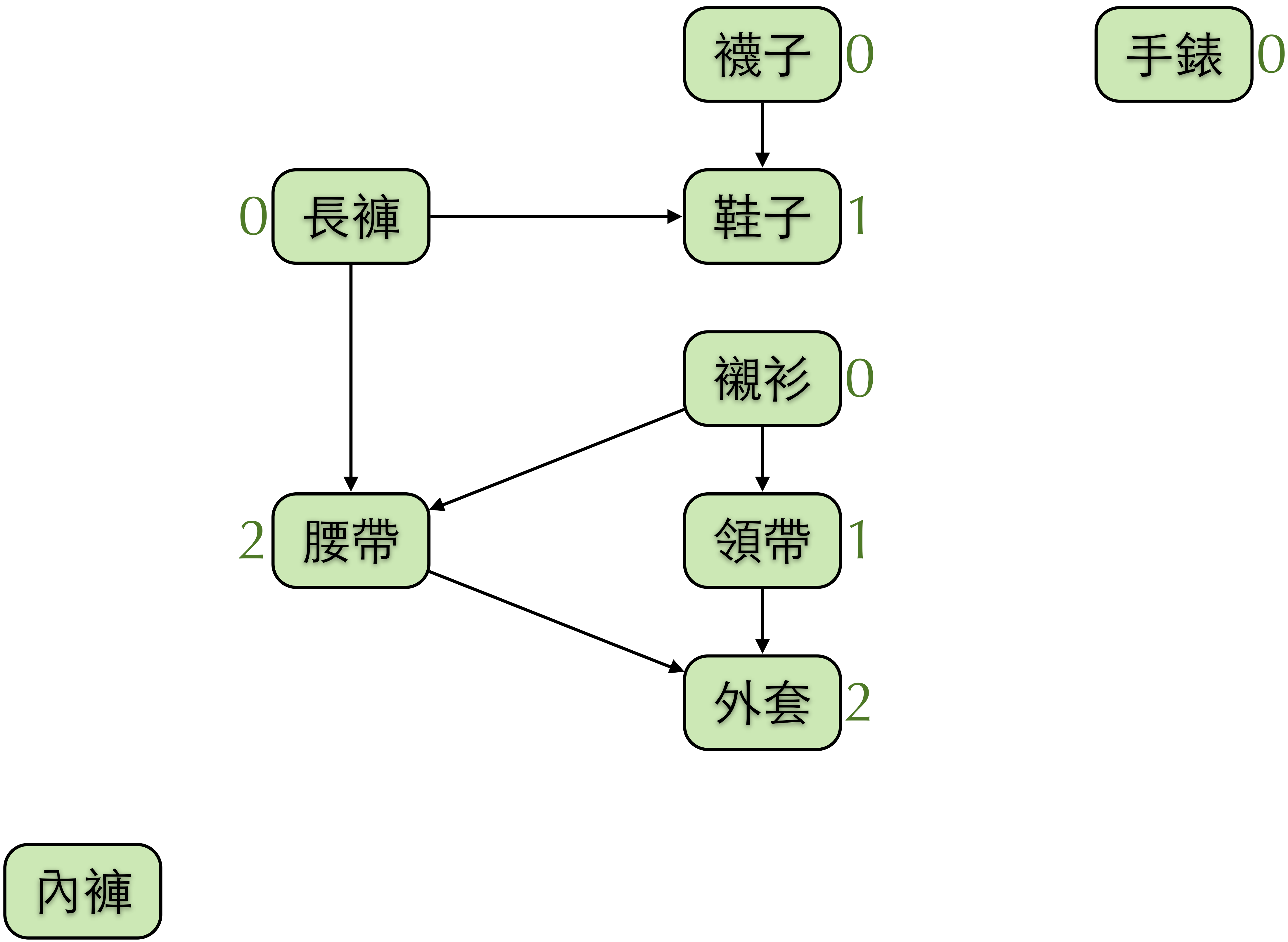


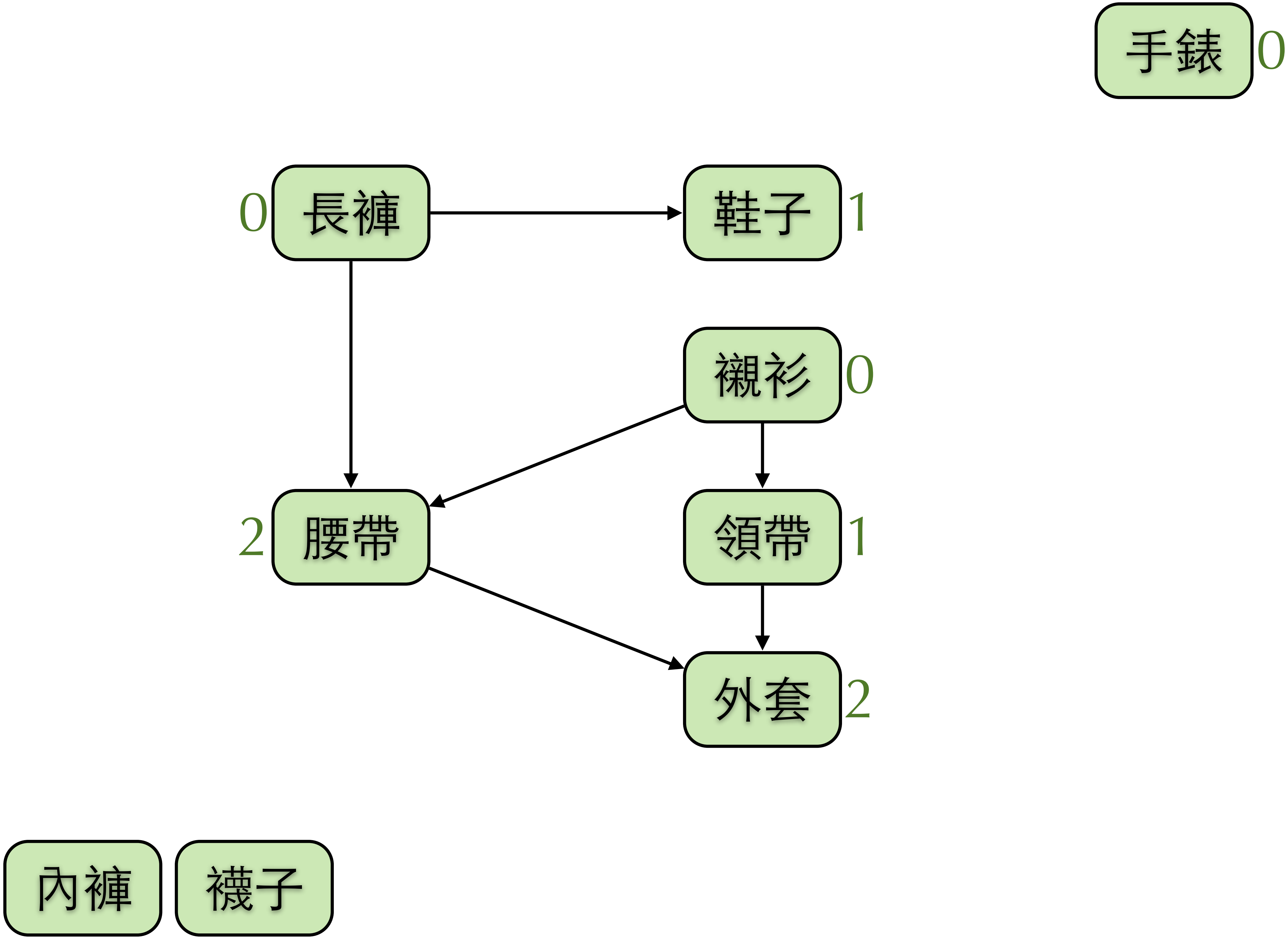
Cycle Detection

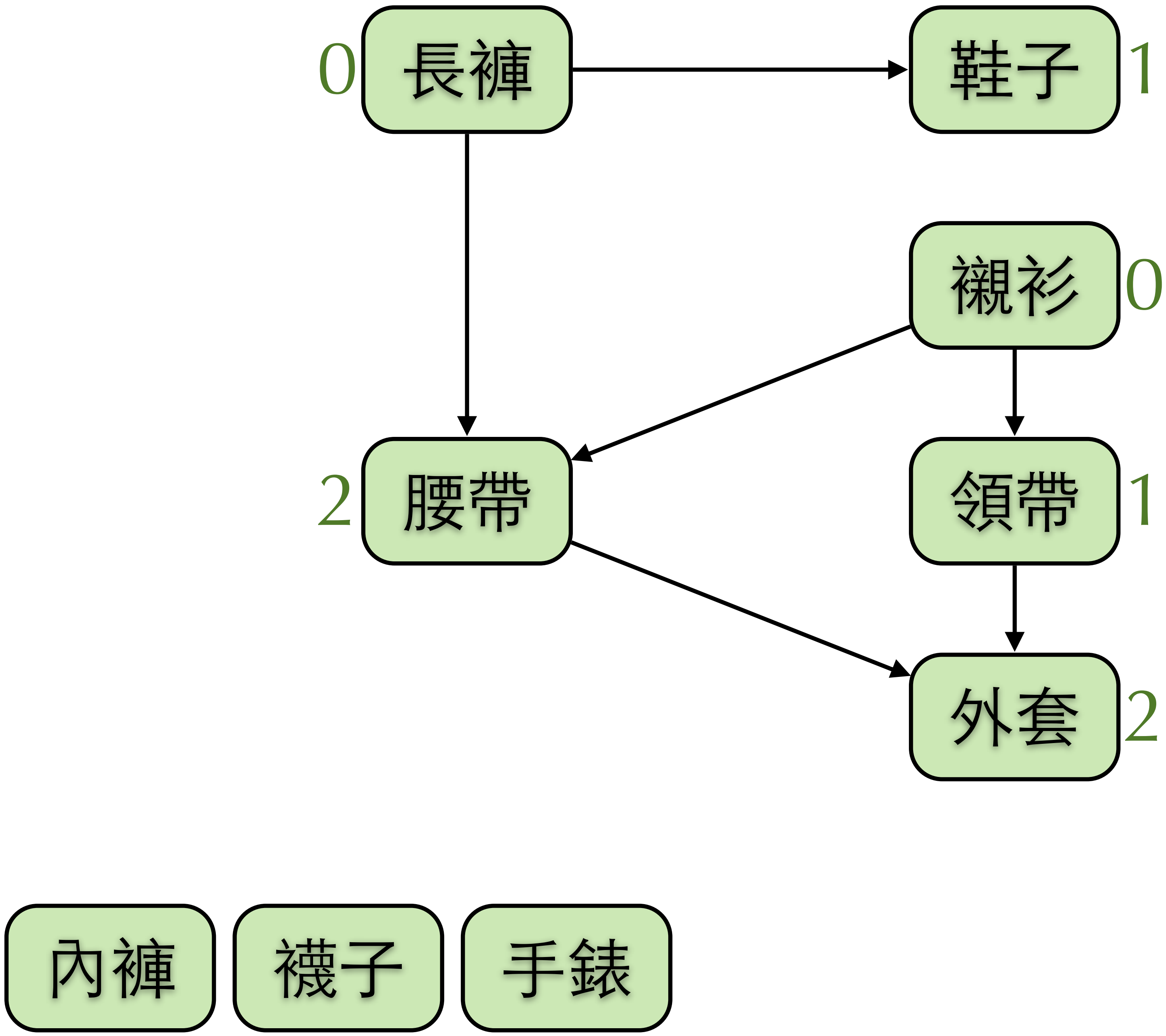
Cycle detected!

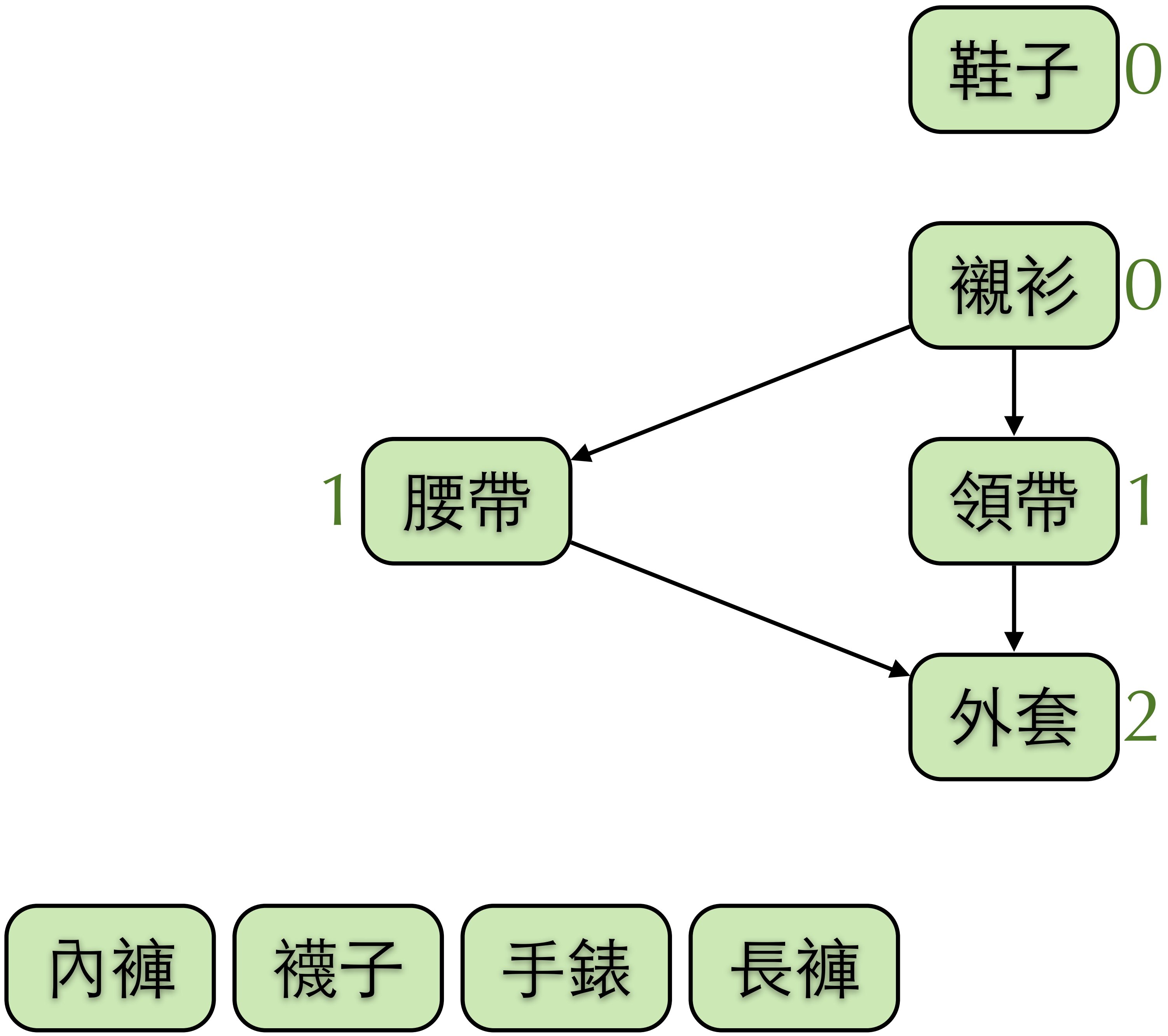


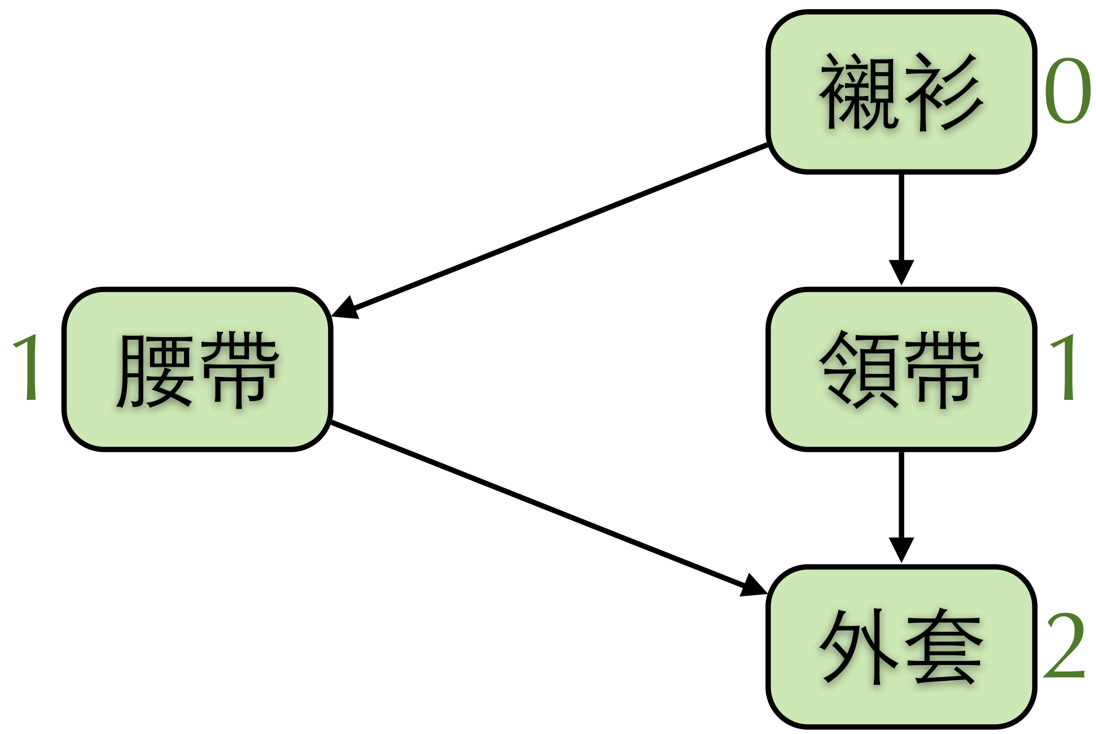

0
內褲
襪子
0
手錶
0
1
長褲
鞋子
2
襯衫
0
2
腰帶
領帶
1
外套
2


襪子 0
手錶 0
0 長褲
鞋子 1
襯衫 0
2 腰帶
領帶 1
外套 2
內褲

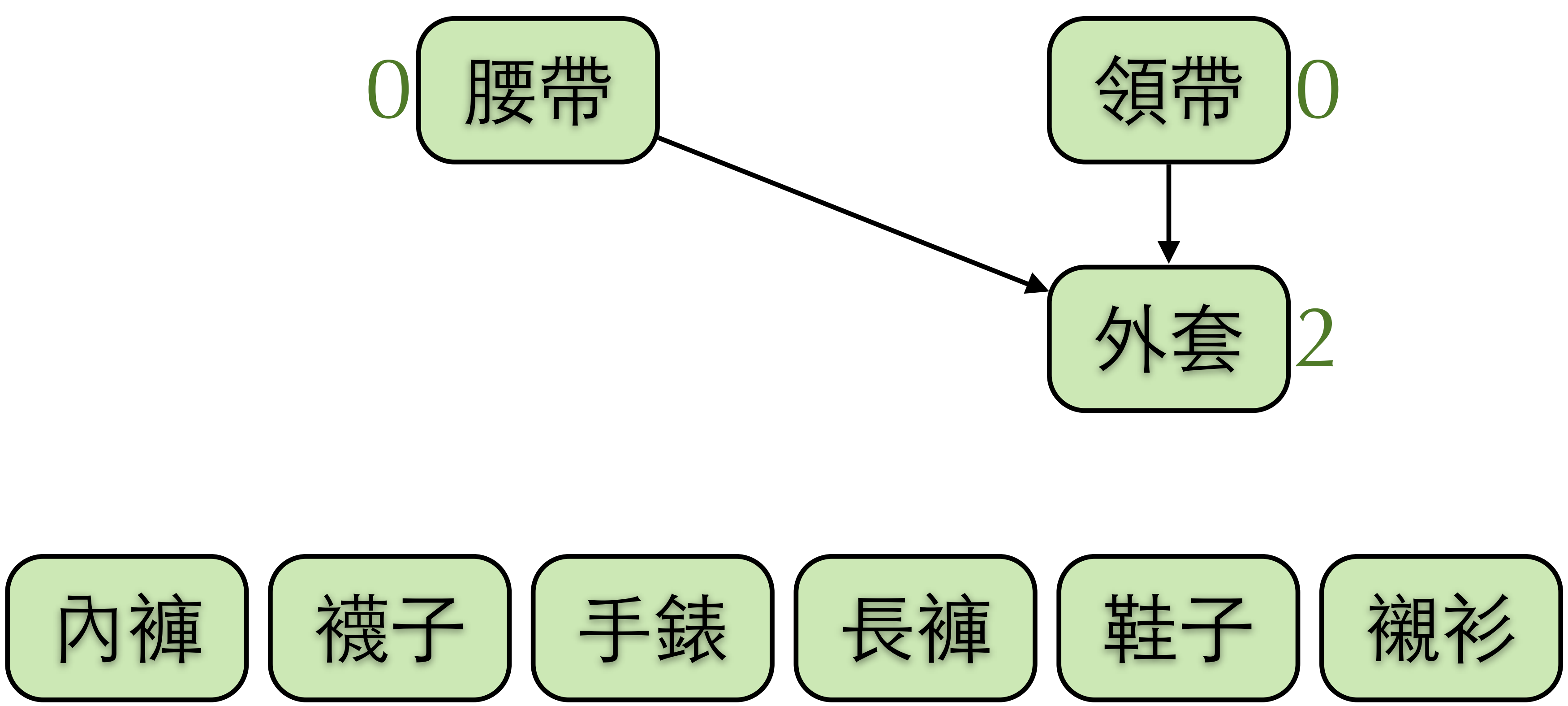

手錶
0
0
長褲
鞋子
1
襯衫
0
2
腰帶
領帶
1
外套
2
內褲
襪子

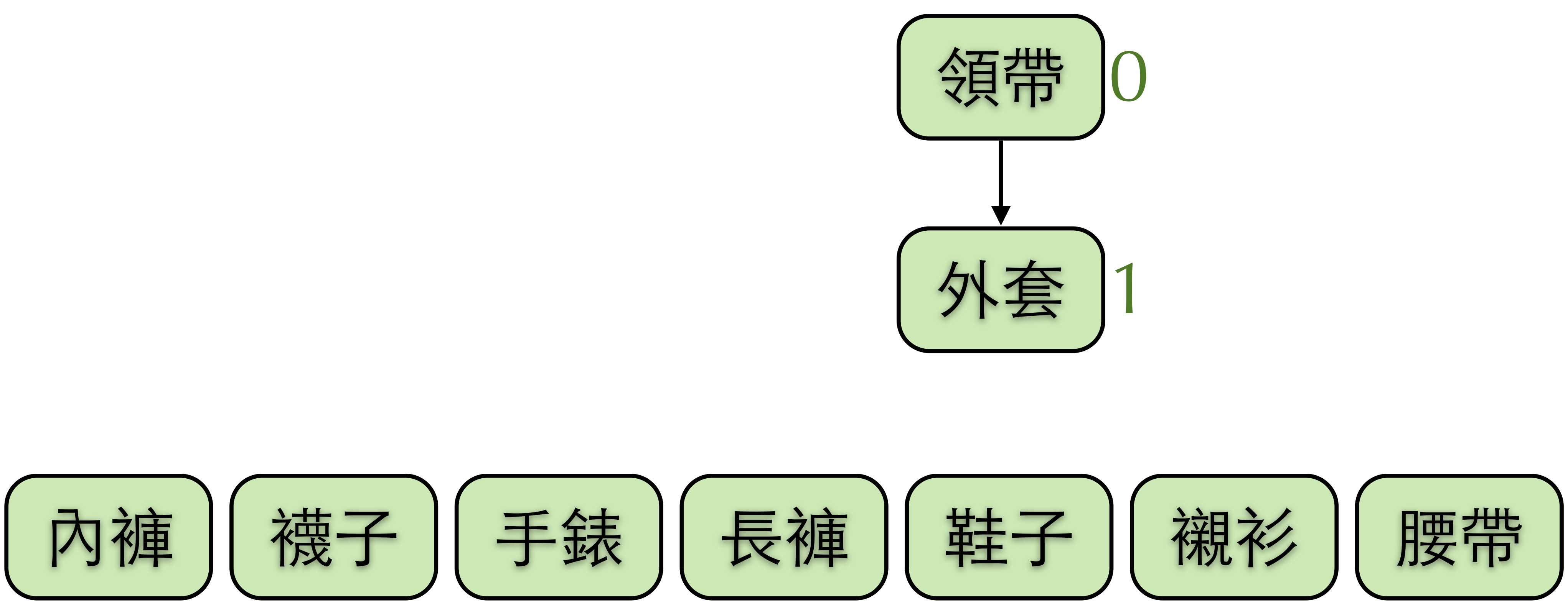

0
長褲
鞋子
1
襯衫
0
2
腰帶
領帶
1
外套
2
內褲
襪子
手錶


鞋子 0
襯衫 0
腰帶 1
領帶 1
外套 2
內褲
襪子
手錶
長褲



內褲 襪子 手錶 長褲 鞋子


0
腰帶
領帶
0
外套
2
內褲
襪子
手錶
長褲
鞋子
襯衫


領帶
0
外套
1
內褲
襪子
手錶
長褲
鞋子
襯衫
腰帶

外套 0

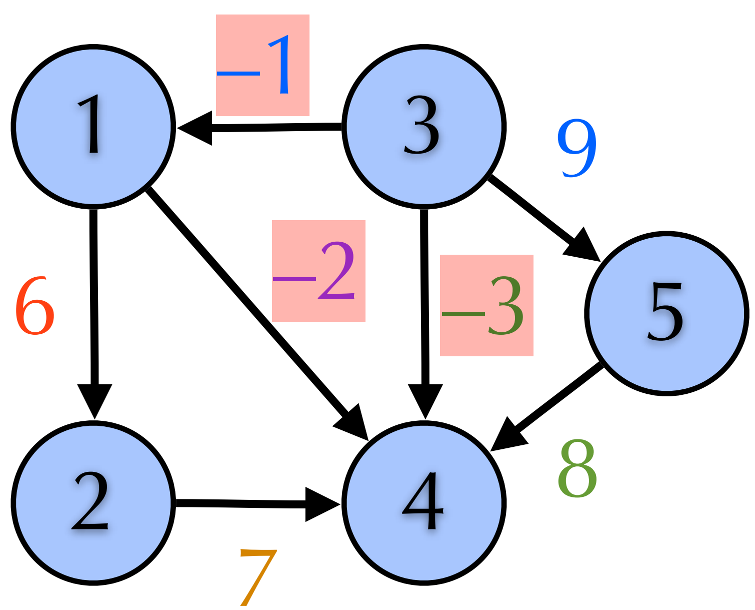
內褲 襪子 手錶 長褲 鞋子 襯衫 腰帶 領帶

內褲 襪子 手錶 長褲 鞋子 襯衫 腰帶 領帶 外套

Shortest Paths

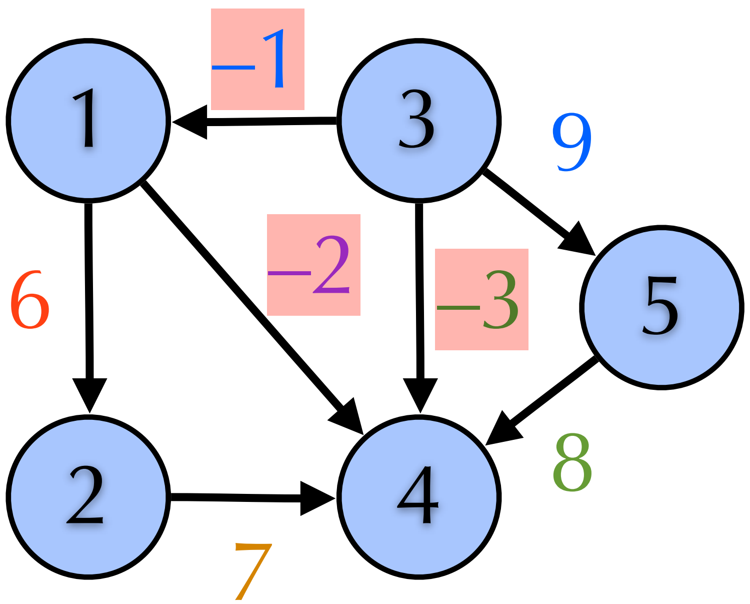
- Weighted graph $G=(V,E)$ with weight w
  - V: set of vertices
  - E: set of edges   directed
  - $w: E \rightarrow R$ can be generalized to paths
    - Weight of path $p=\langle v_0,v_1,\ldots,v_k\rangle$: $w(p)=\Sigma_{1\le i\le k}w(v_{i-1},v_i)$
- $\delta(u,v)=\min_{p:u \rightsquigarrow v}w(p)$ no path: $\delta(u,v)=\infty$
- Goal: Compute $\delta(u,v)$

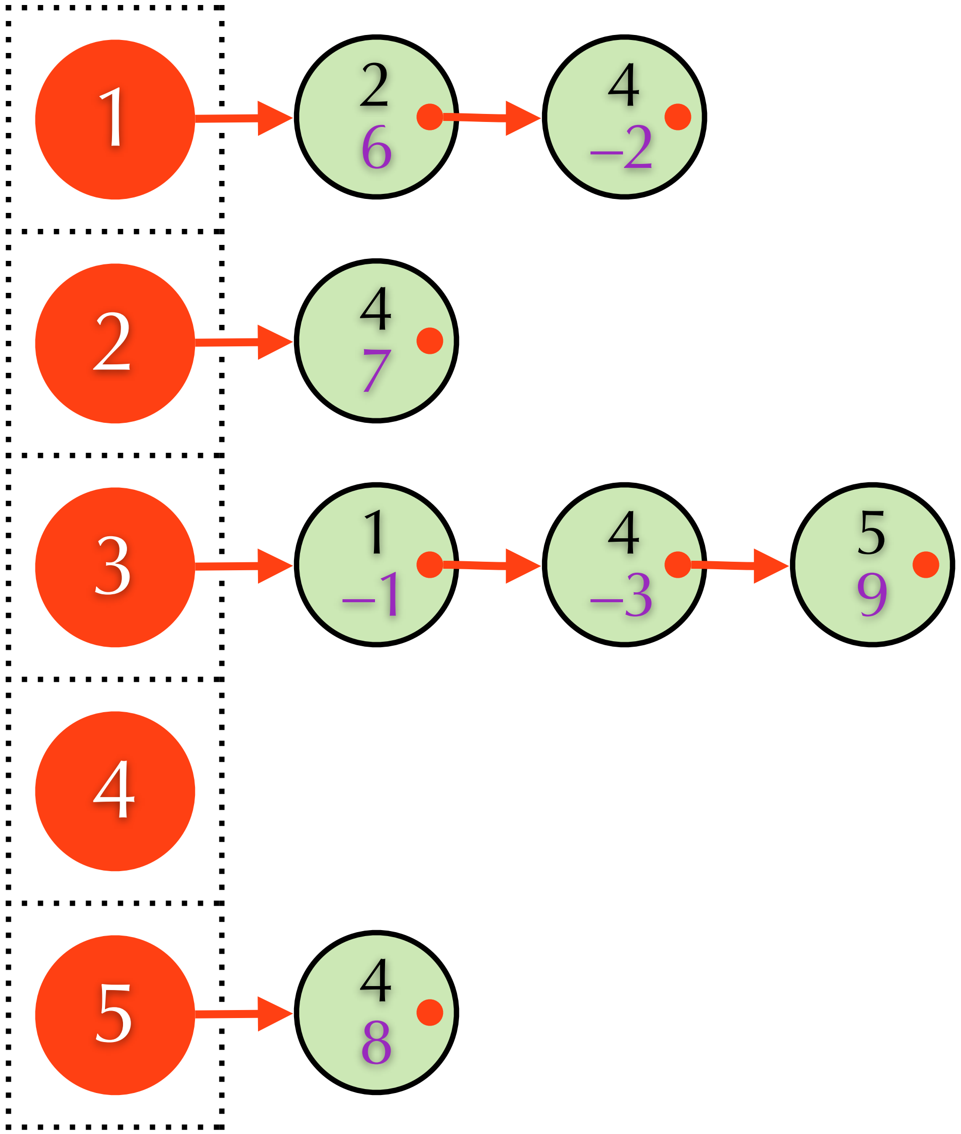
Weighted Adjacency Matrix



	1	2	3	4	5
1	0	6	∞	−2	∞
2	∞	0	∞	7	∞
3	−1	∞	0	−3	9
4	∞	∞	∞	0	∞
5	∞	∞	∞	8	0

Weighted Adjacency List



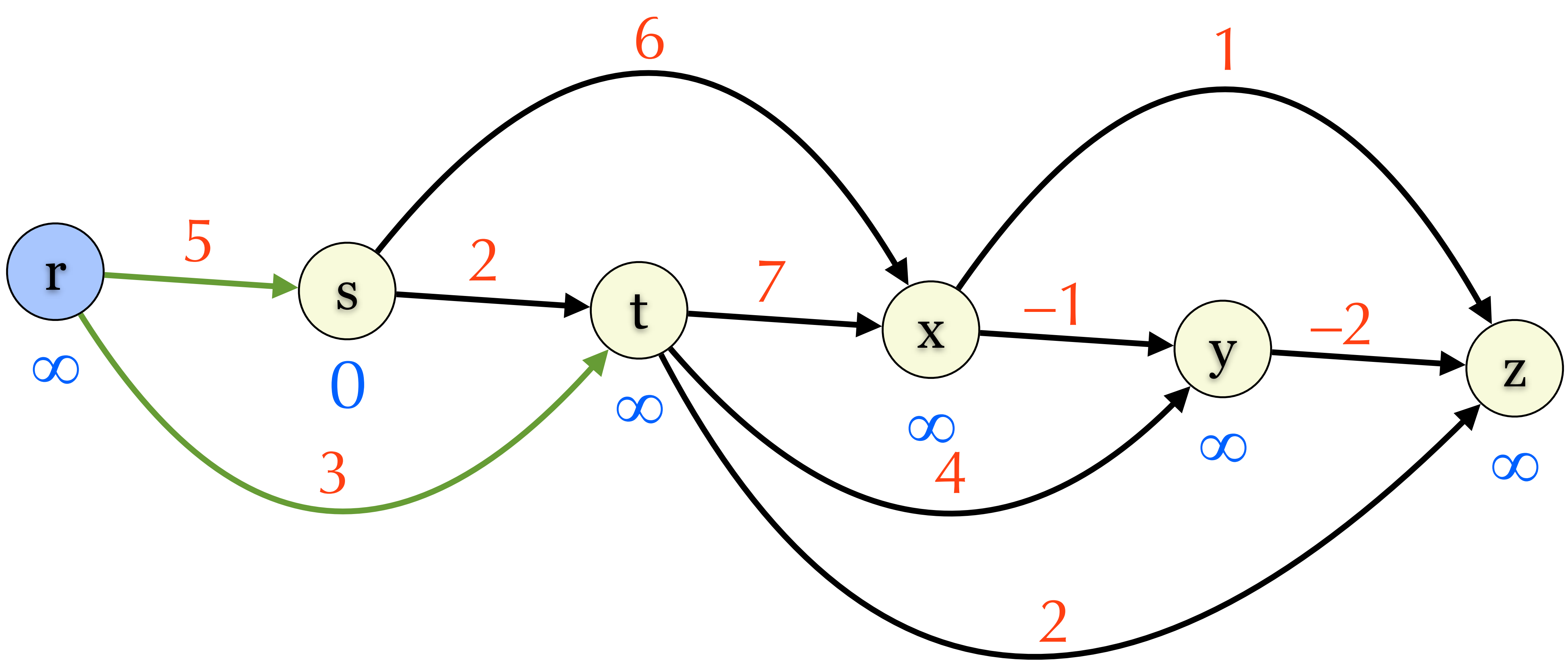


Shortest Path: Directed Acyclic Graph

- source: s
- d[v]: the minimum distance from s to v
- $\langle v_1,\ldots,v_{|V|}\rangle$=Topological-Sort(G)
  for i = 1 to |V| do
      $d[v_i]=\infty$
  d[s]=0
  for i = 1 to |V|−1 do $O(|V|+|E|)$
      for each edge $(v_i,v)\in E$ do
          if $d[v]>d[v_i]+w(v_i,v)$
              $d[v]=d[v_i]+w(v_i,v)$

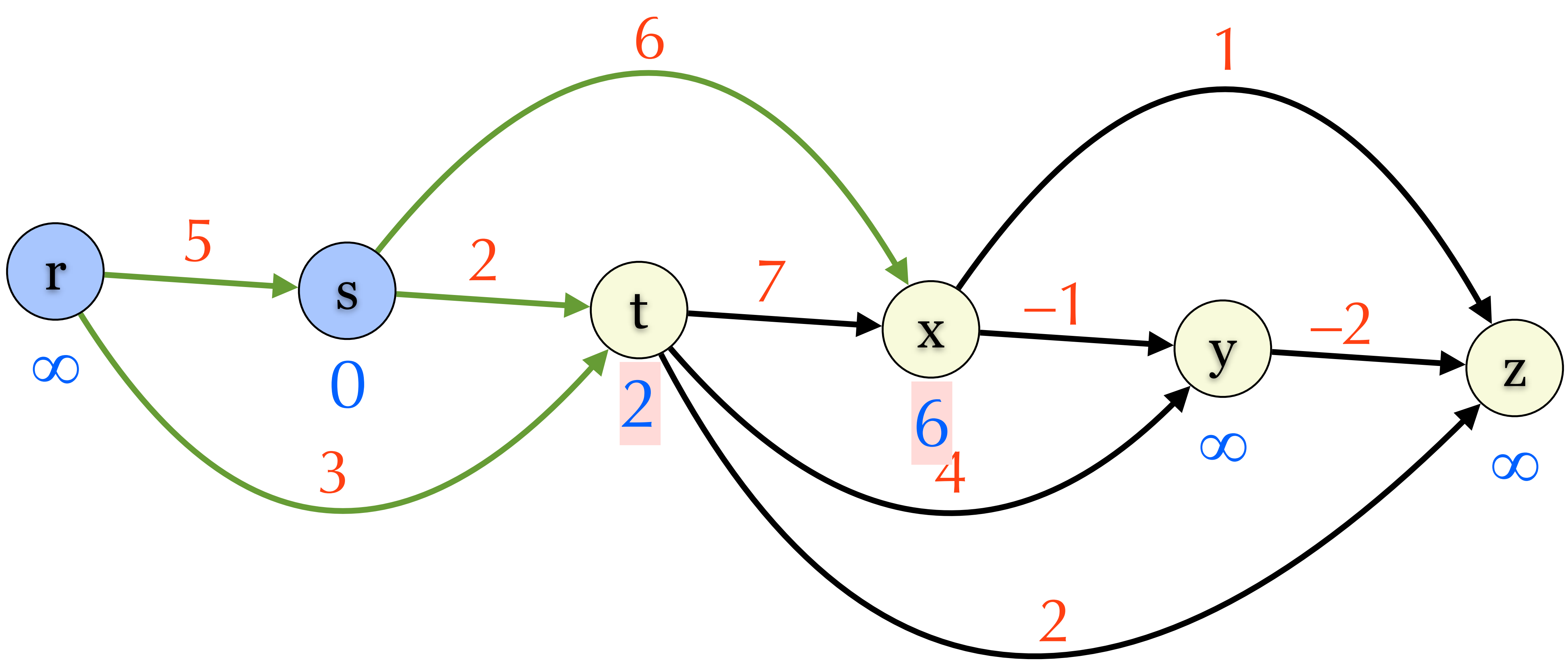
Example

Starting at s



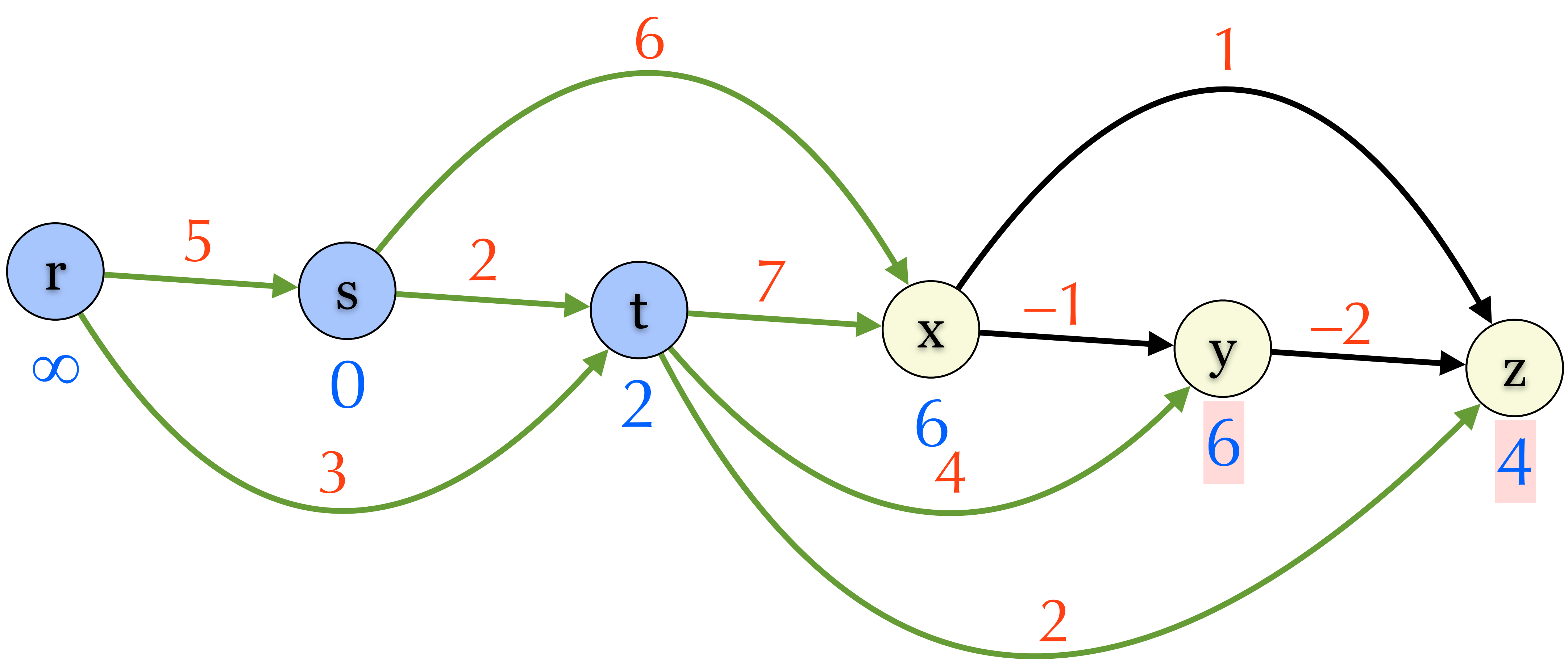
Example

Starting at s



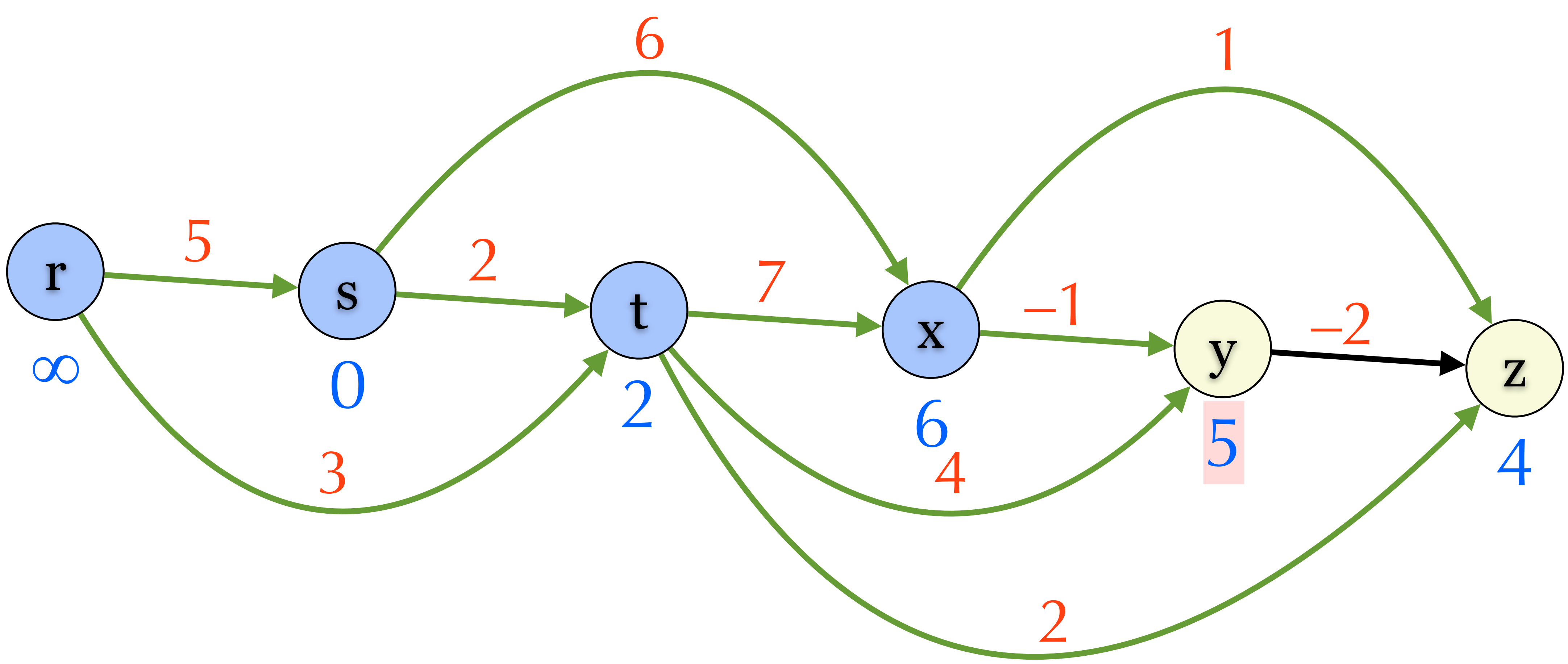
Example

Starting at s



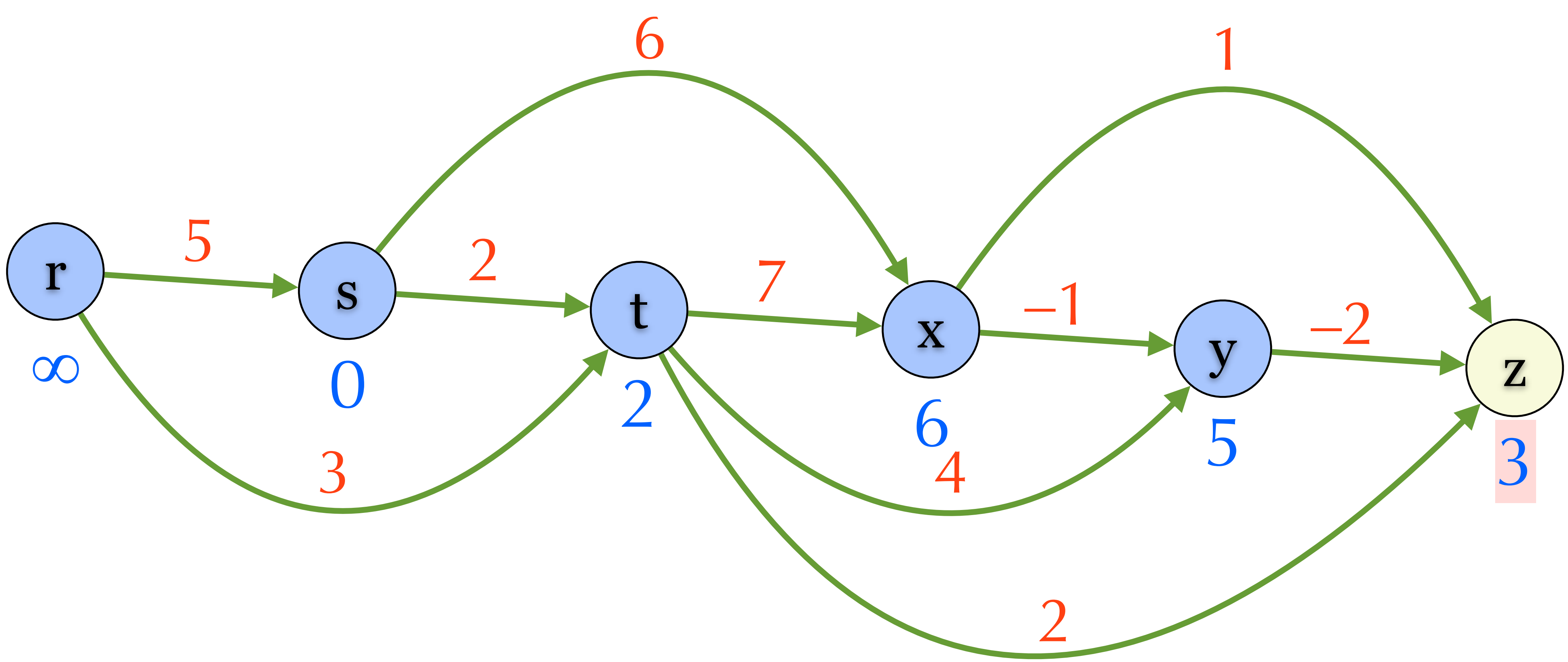
Example

Starting at s



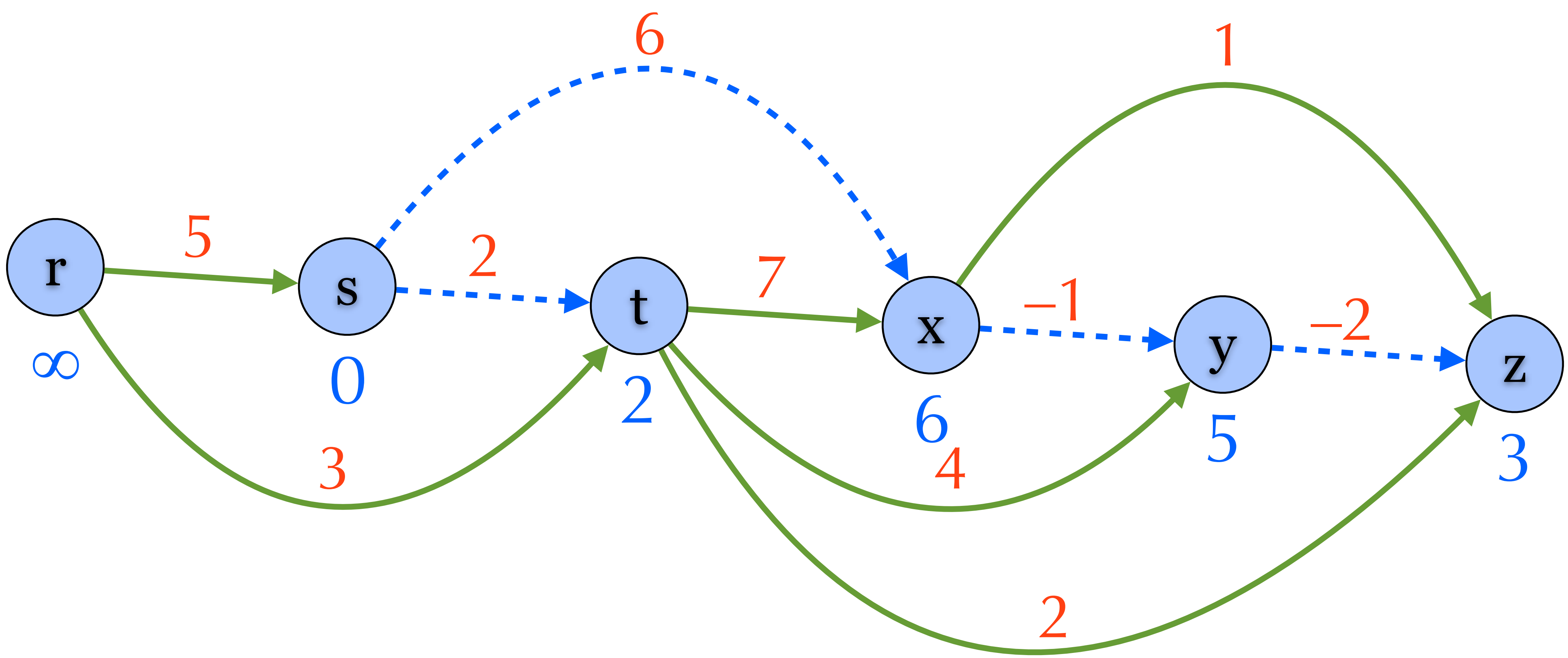
Example

Starting at s



Done

Starting at s

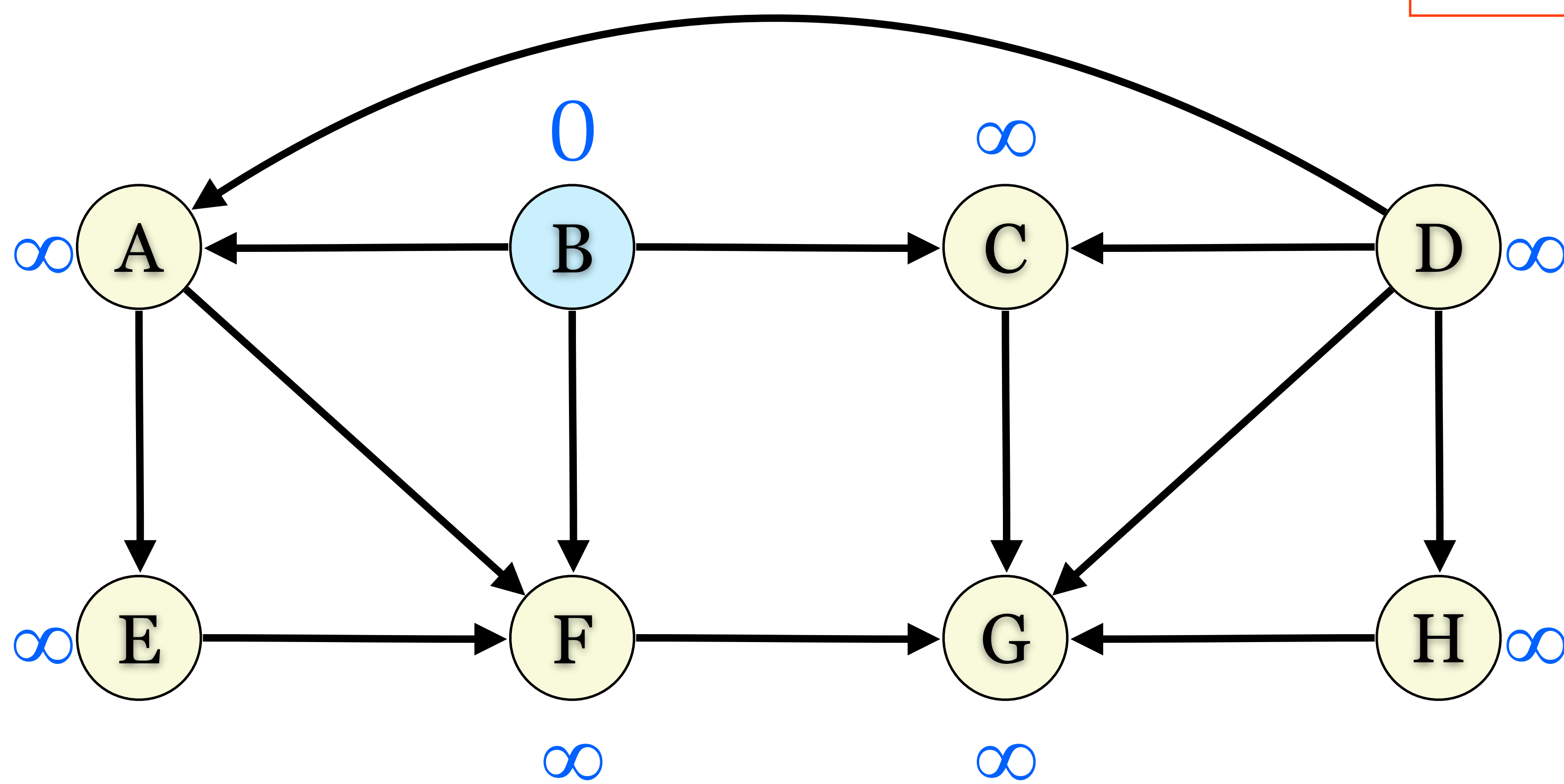


Shortest Paths: Unweighted Graph

- This can be solved by modified BFS
  - Or standard version in CLRS
- Initialization:
  - for each $v \in V$: enqueued[v]=visited[v]=false, $d[v]=\infty$
  - enqueued[s]=true, $d[s]=0$, Q.enqueue(s)
- Main loop: while($Q \neq \emptyset$)
  - u=Q.dequeue()
for each v s.t. $(u,v) \in E$
    if enqueued[v]==false
        enqueued[v]=true, $d[v]=d[u]+1$, Q.enqueue(v)
visited[u]=true

Example

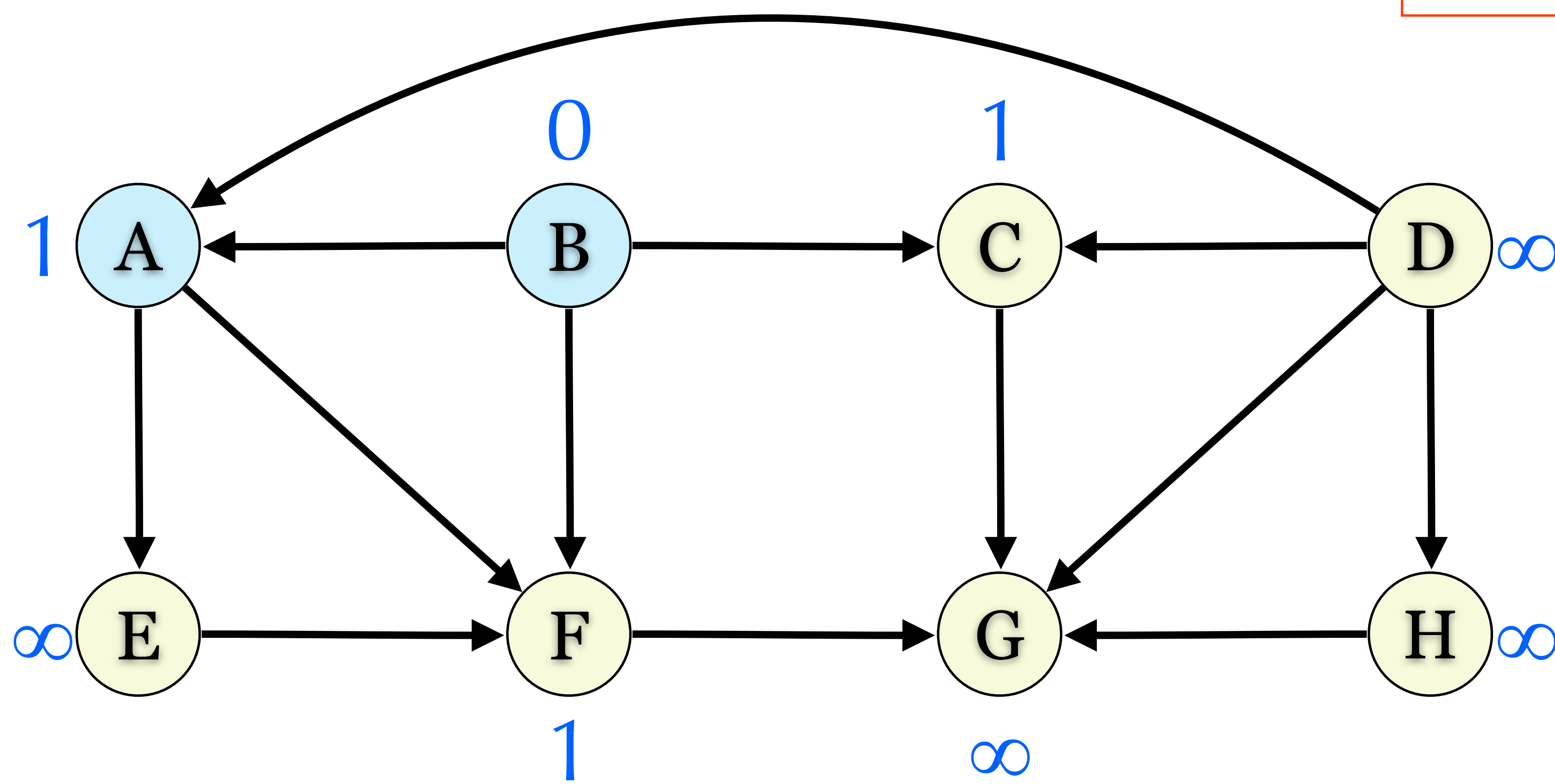
Starting at B



Queue	B							

Example

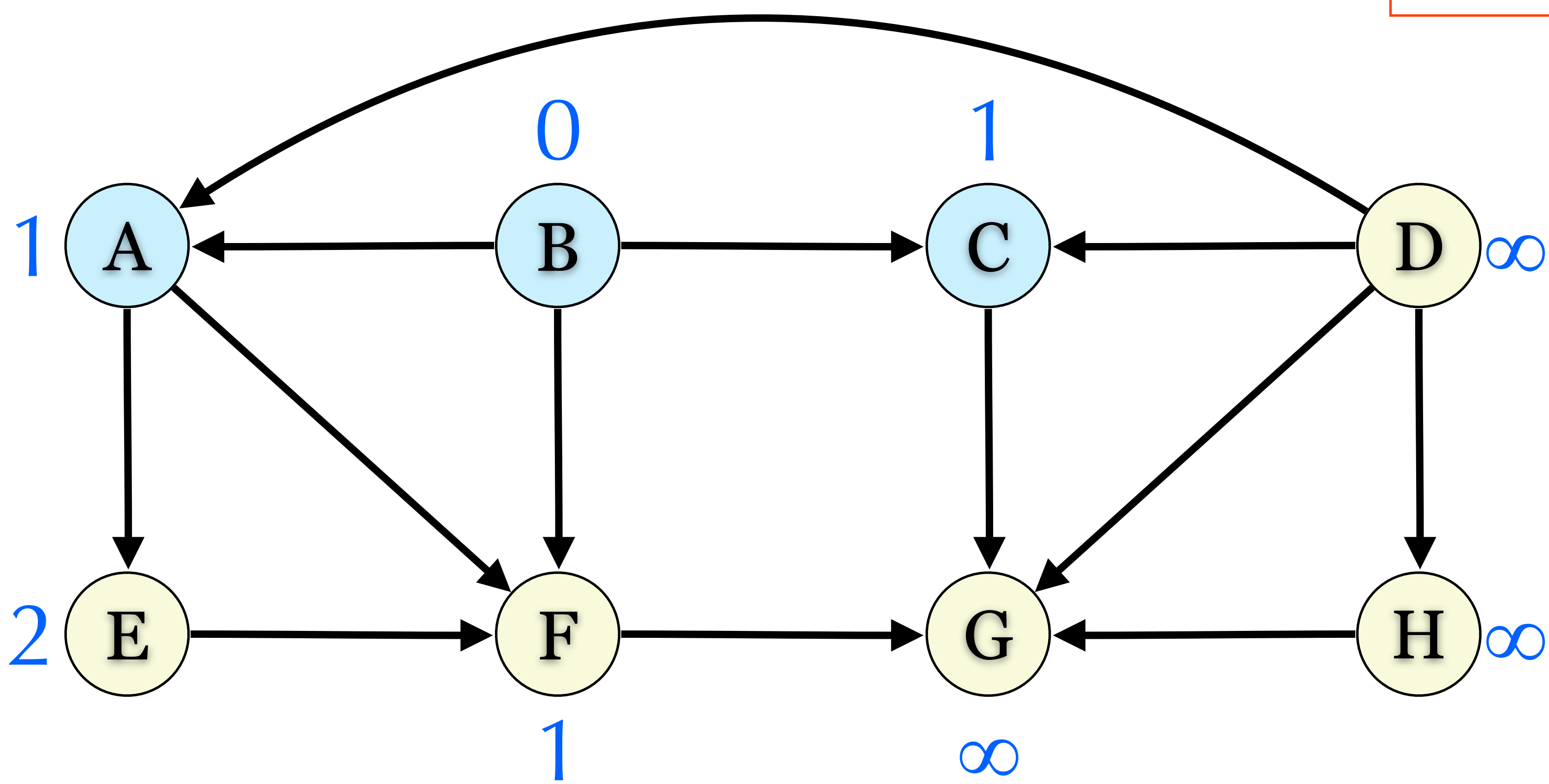
Starting at B



Queue								
	A	C	F					

Example

Starting at B

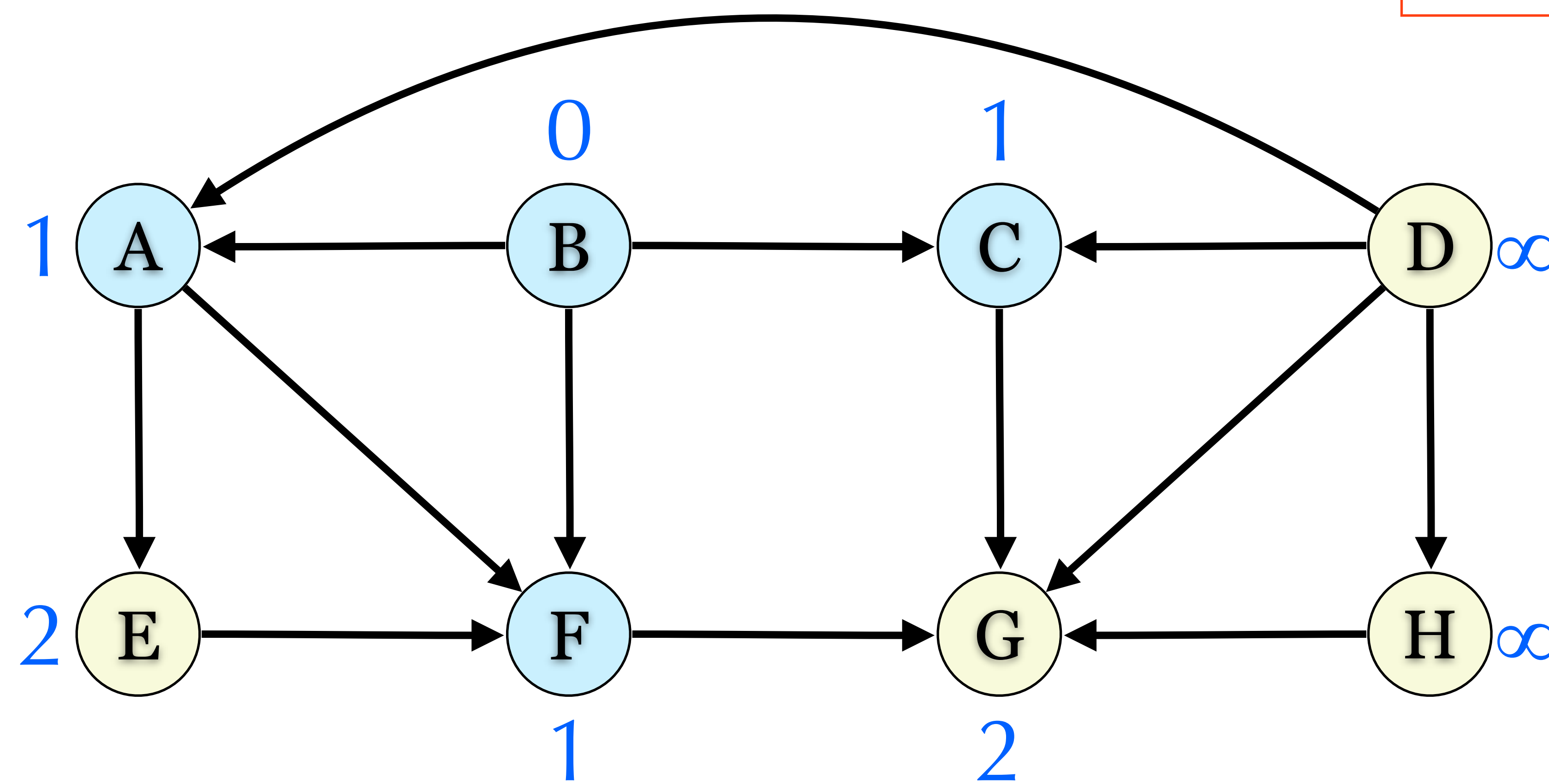


Queue

C	F	E					

Example

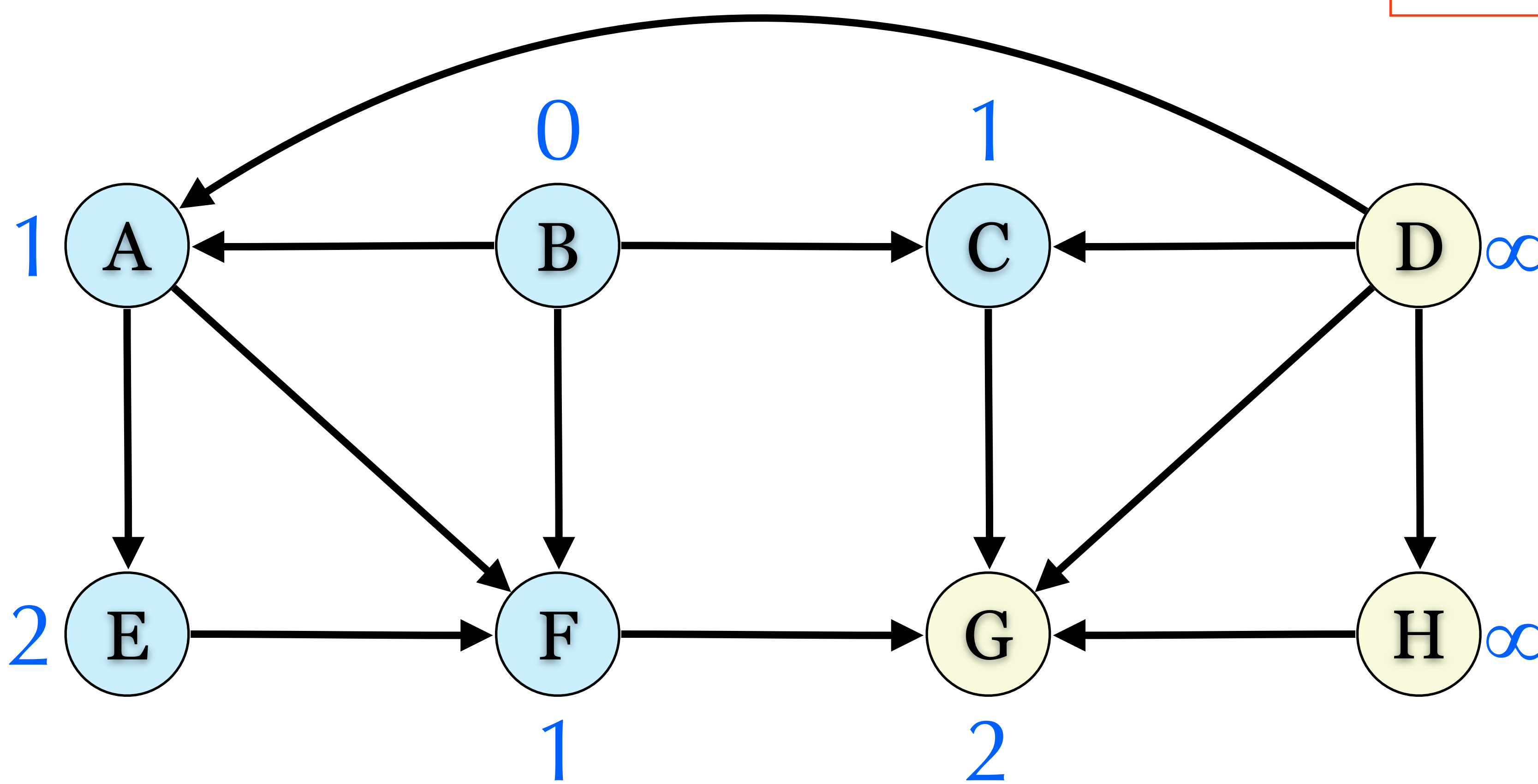
Starting at B



Queue								
	F	E	G					

Example

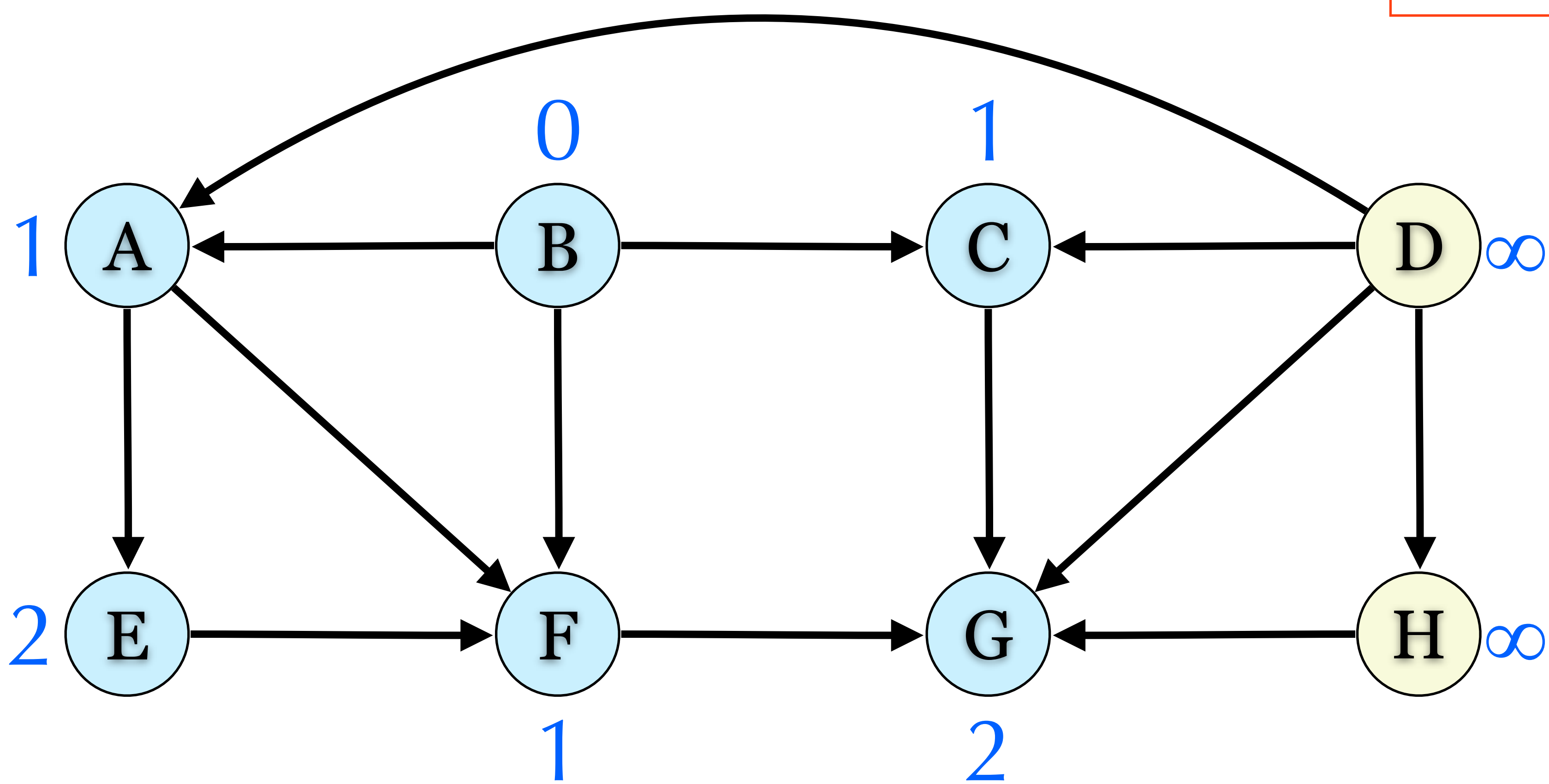
Starting at B



Queue							
E	G						

Example

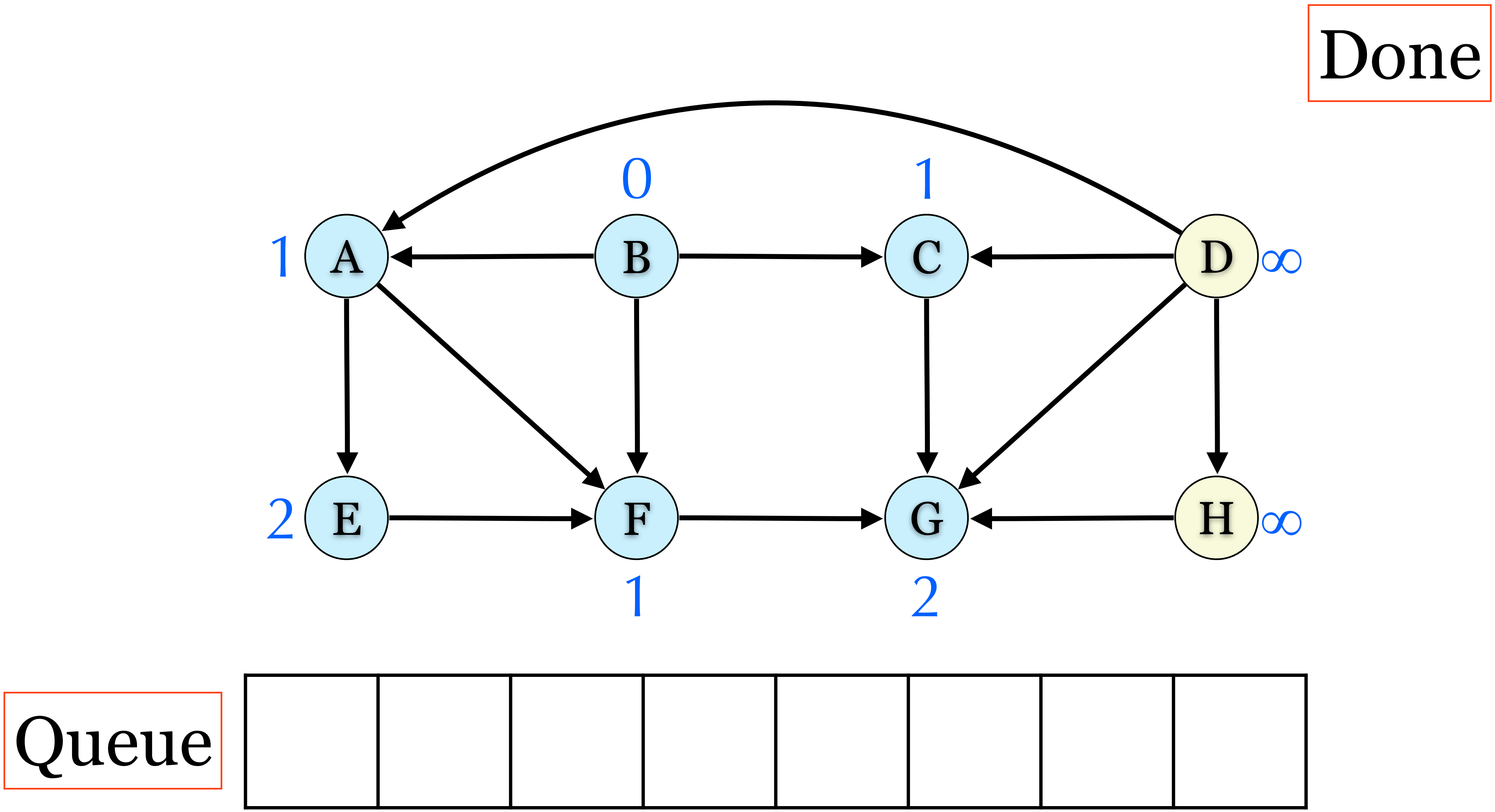
Starting at B



Queue

G							

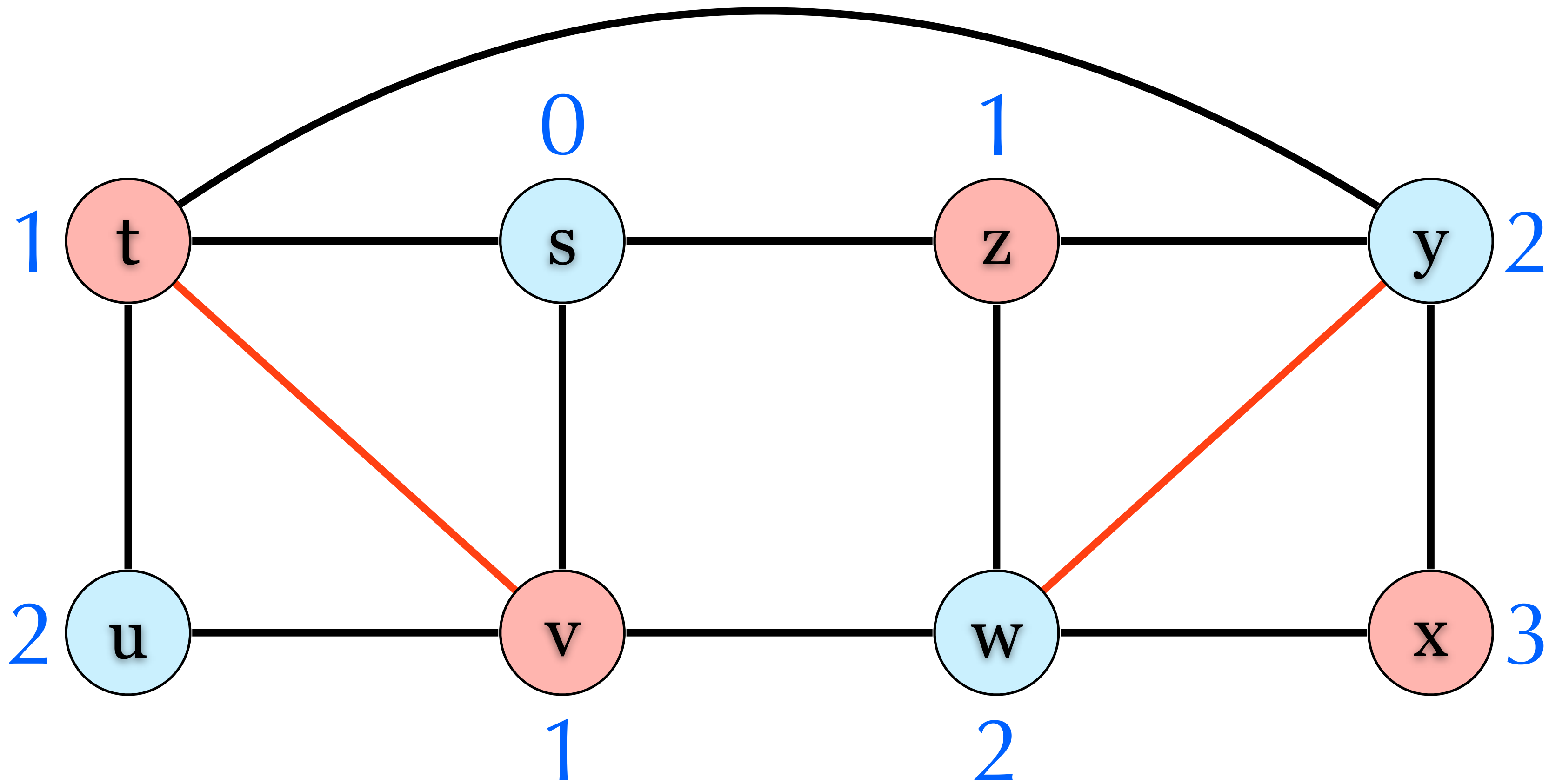
Example



Bipartite Graph Check

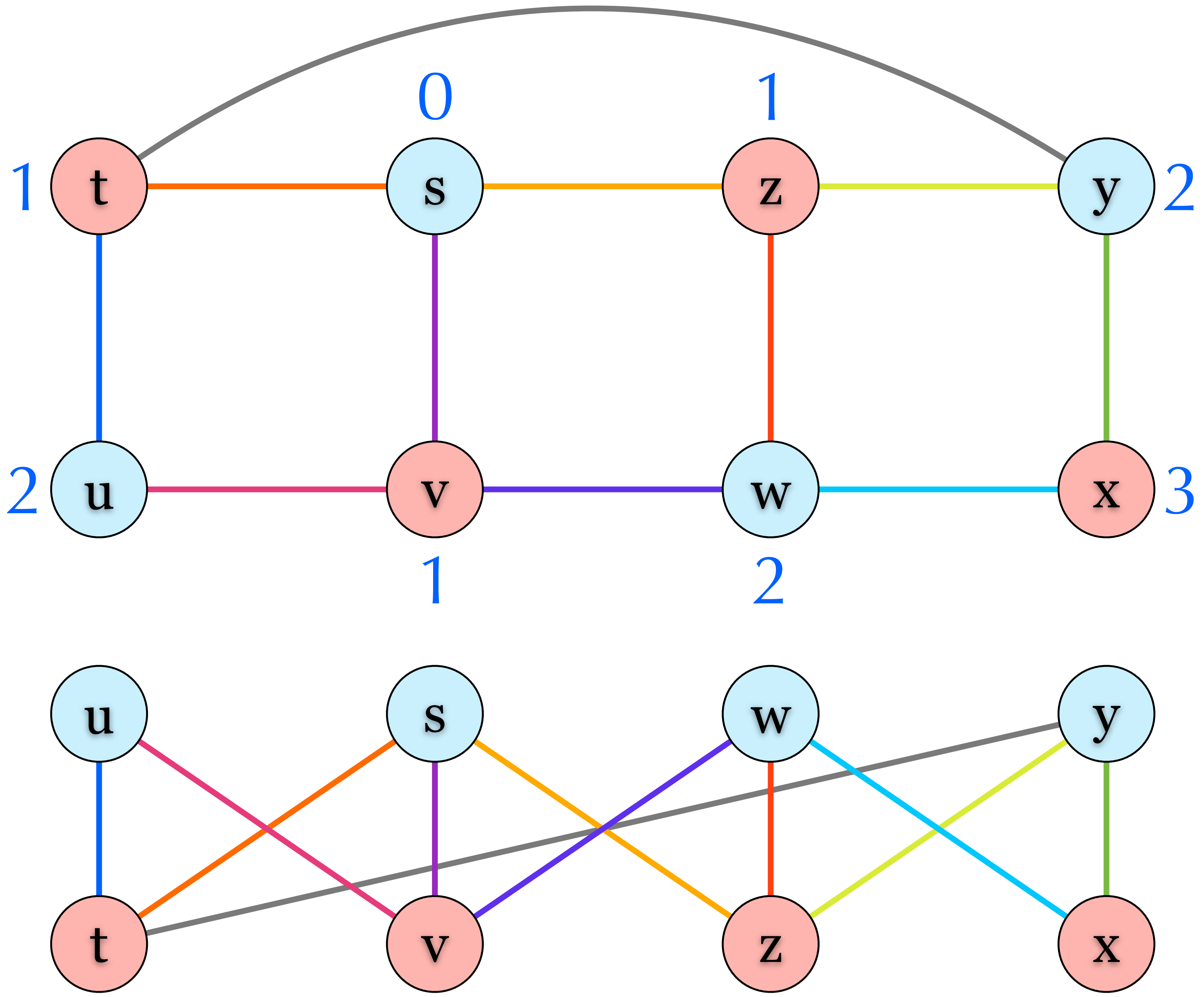
- Bipartite graph $G=(V,E)$ undirected
  - V can be partition into A and B
    - $A \cup B = V$ and $A \cap B = \varnothing$
    - If $\{u,v\} \in E$, then $u \in A \Leftrightarrow v \in B$.
- How to check whether a graph is bipartite?
  - Repeat until $d[v] < \infty$ for every $v \in V$
  - Pick u s.t. $d[u] = \infty$
  - Run BFS from u to compute $\delta(u,v)$
  - If there is an edge $\{u,v\}$ s.t. $2 \mid (d[u]+d[v])$ then the graph is non-bipartite.

Example



Non-bipartite

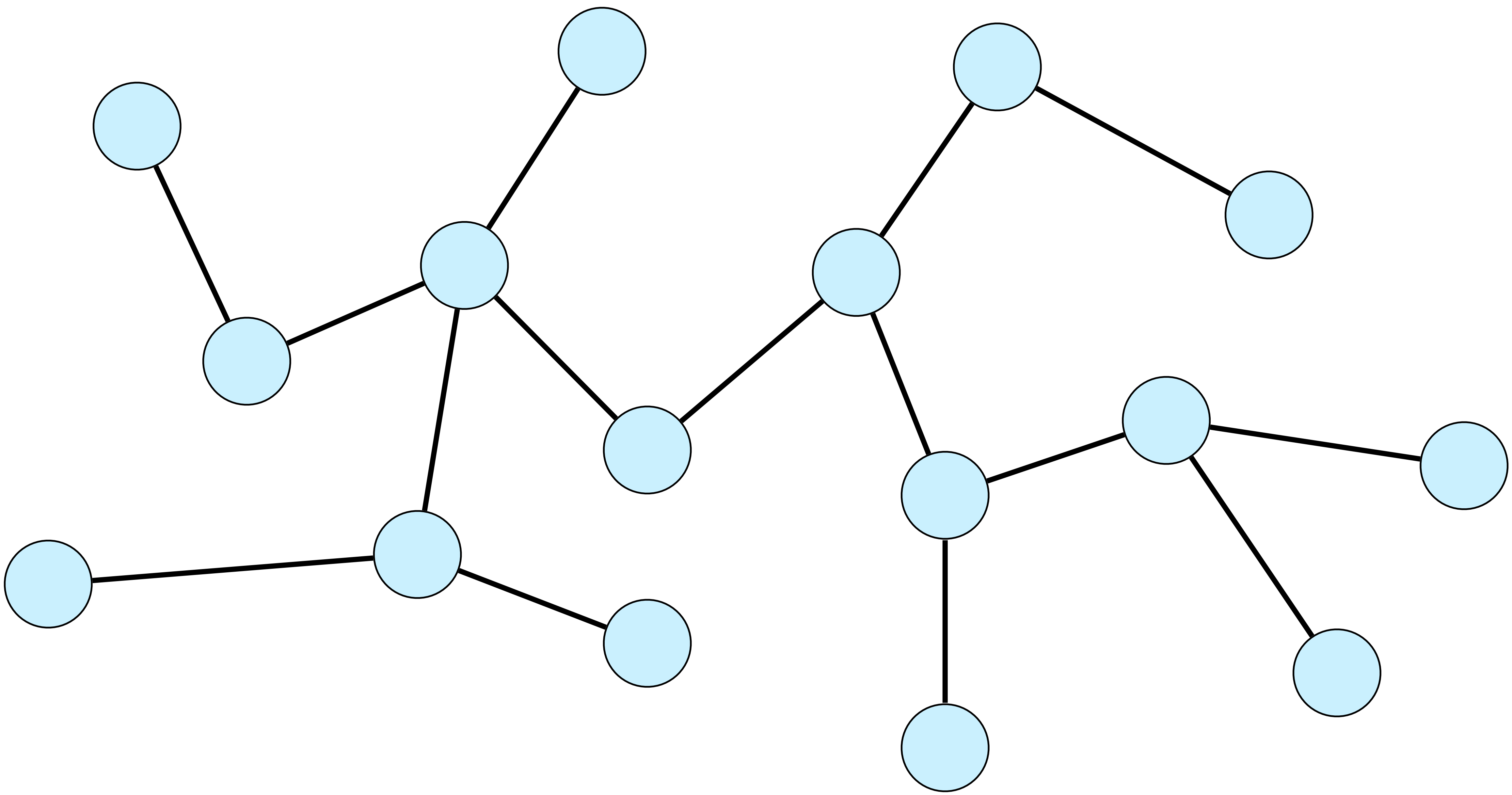
Example



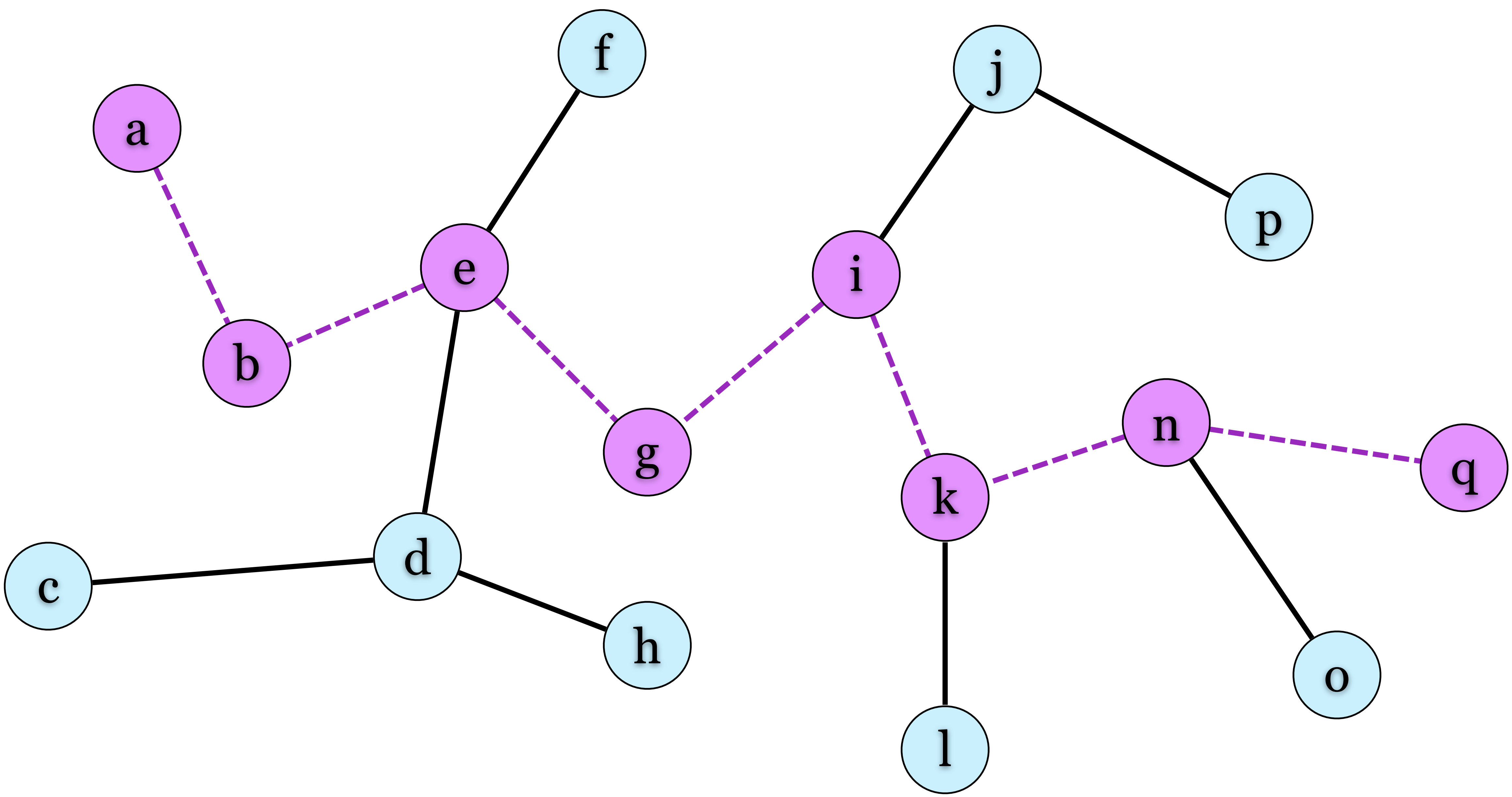
Tree Diameter

- Graph diameter of G: the length of the longest shortest path in G.
- Tree diameter: G=(V,E) is a tree
  - G is undirected unweighted/weighted
  - $|E|=|V|-1$
  - For every pair $u,v \in V$, there exists a path from u to v.

Example

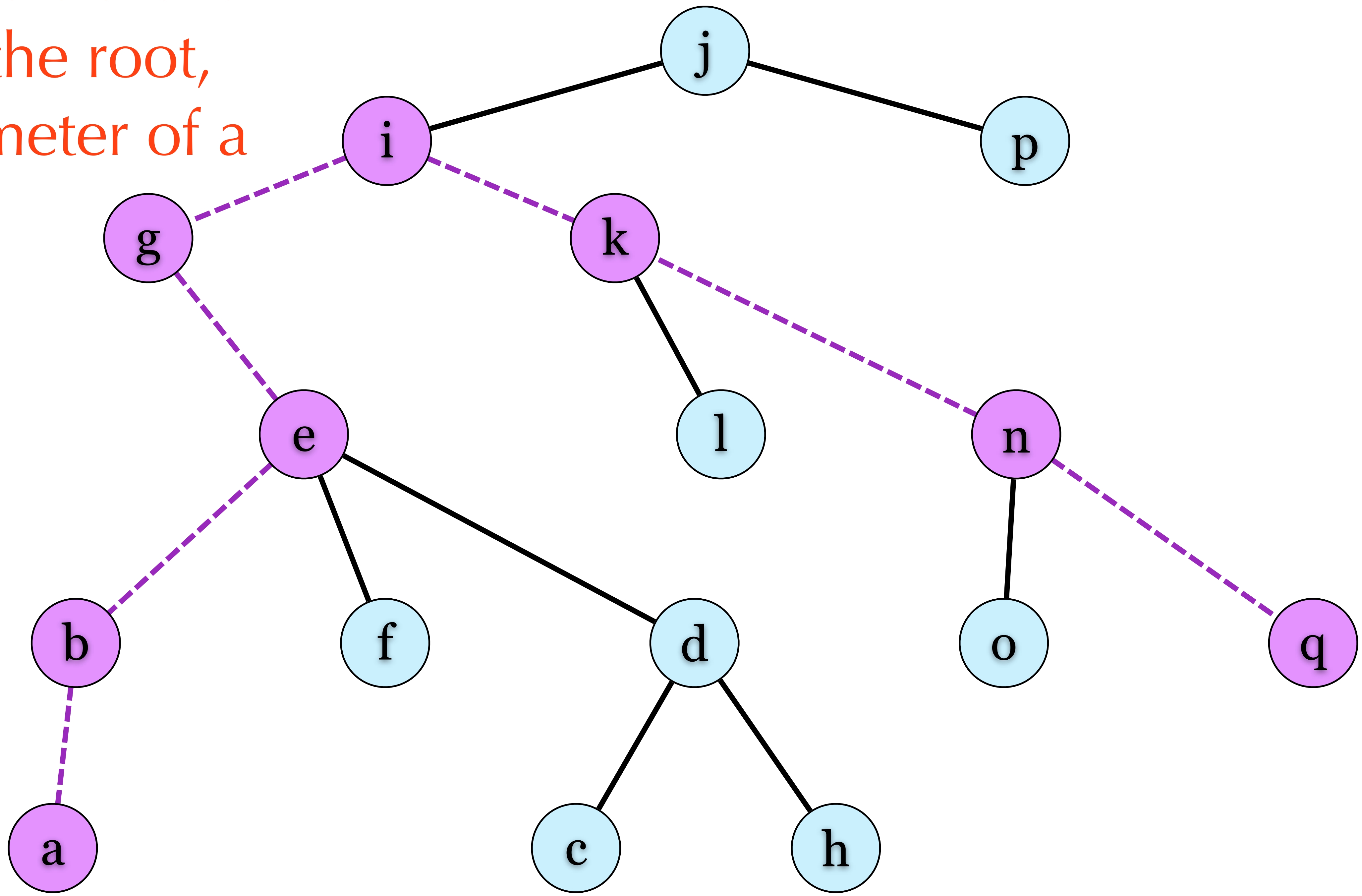


Example



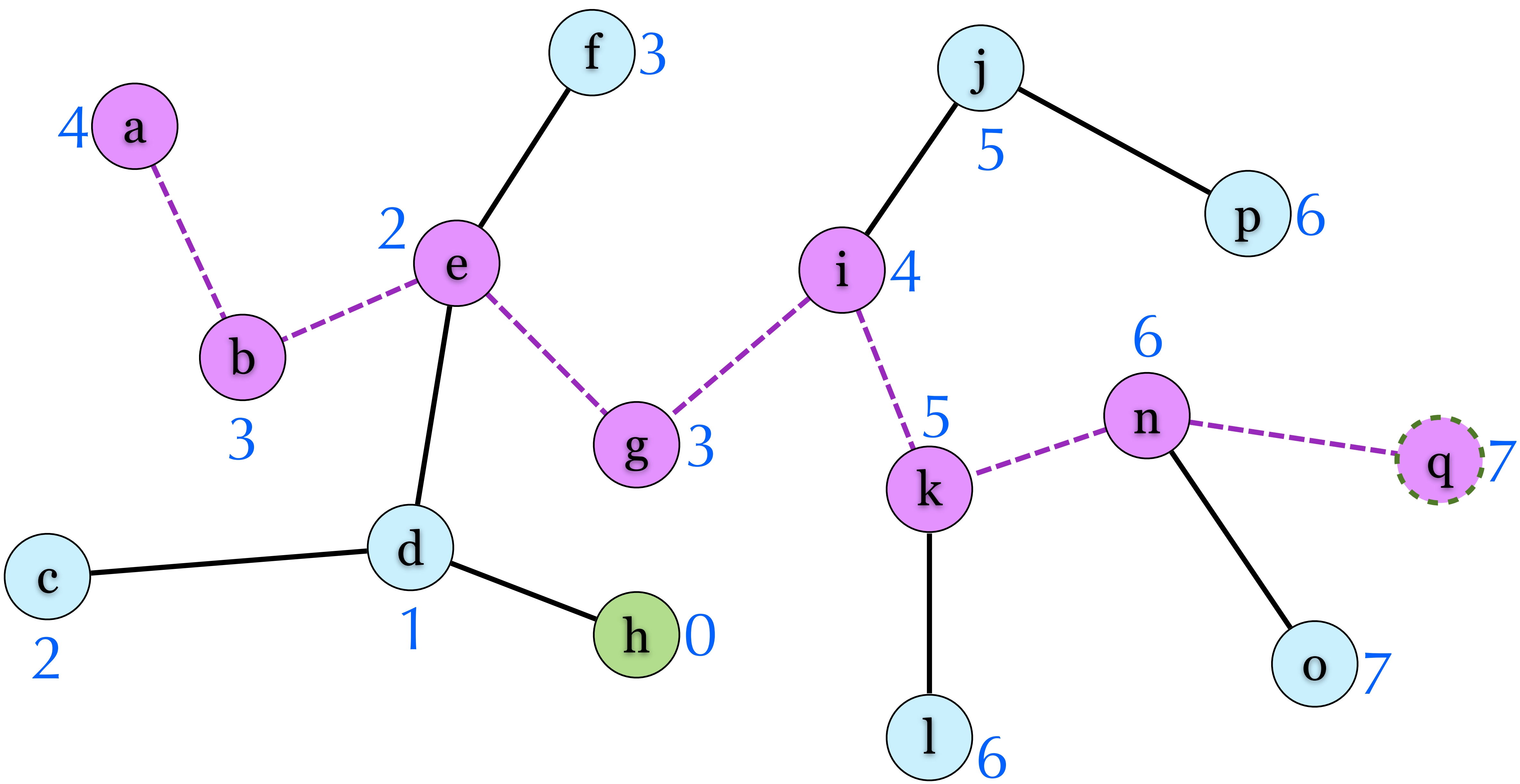
Solution 1: DP

Diameter is either a path pass the root, or the diameter of a subtree.



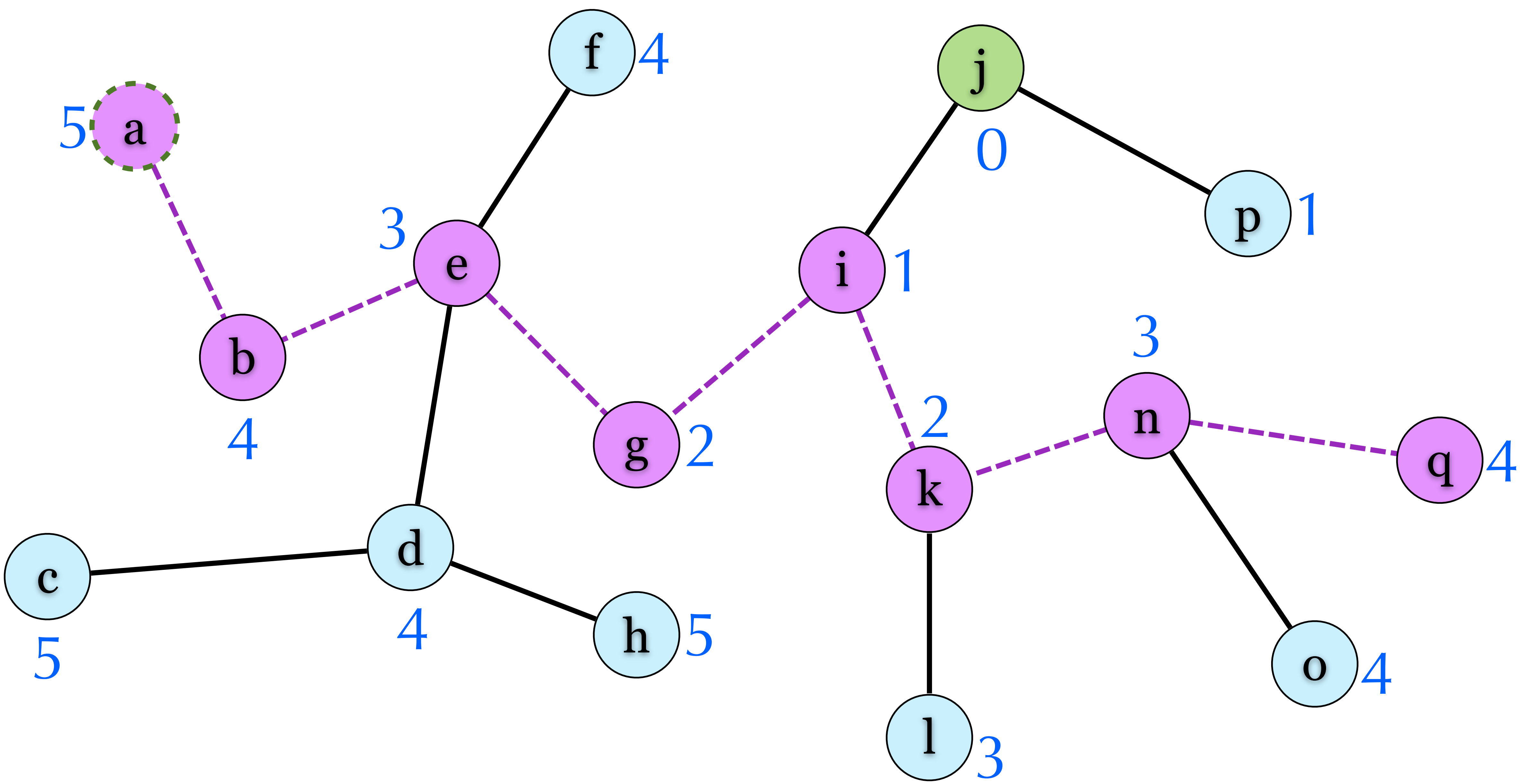
Solution 2: Greedy

The farthest vertex of any v∈V is an end of the diameter.



Solution 2: Greedy

The farthest vertex of any v∈V is an end of the diameter.



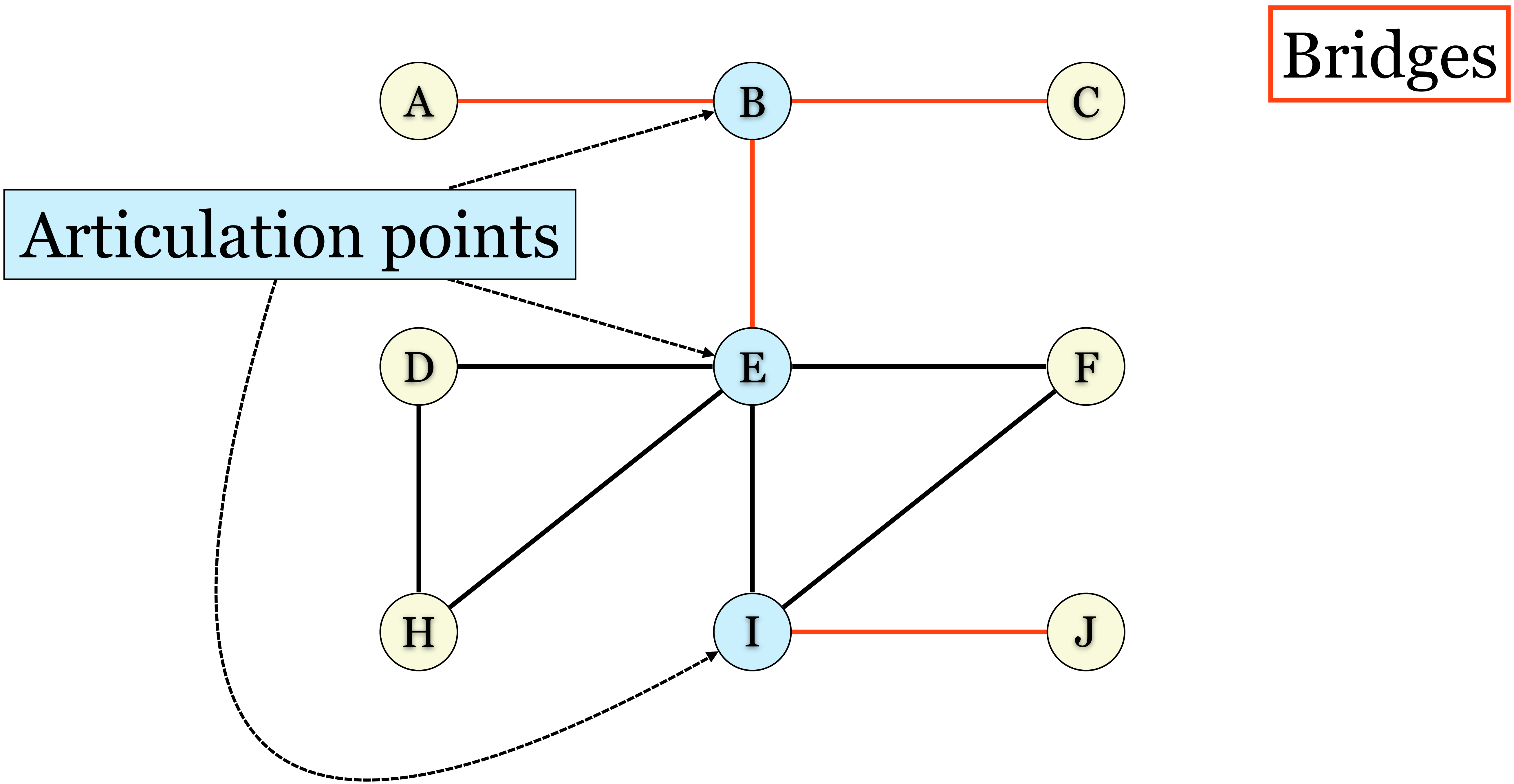
Connected Components

- In an undirected graph G, a connected component $C=(V_C,E_C)$ is a subgraph of G:
  - For $u,v \in V_C$, there is a path from u to v.
  - For $u \in V_C$ and $v \in V \setminus V_C$, there is no path from u to v.
- Use DFS or BFS to compute connected components.
  - Which is better? It depends.

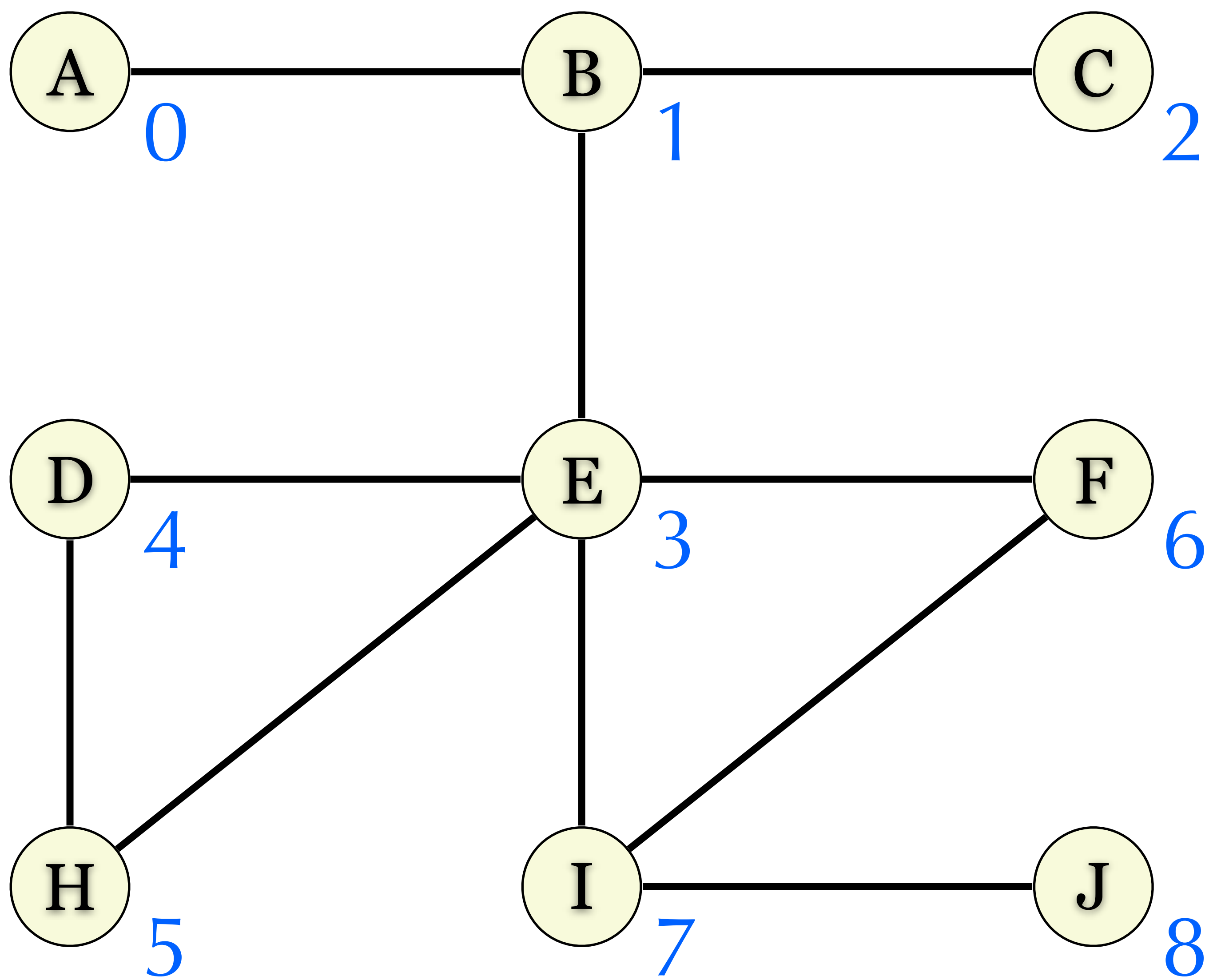
Articulation Points and Bridges

- In undirected graphs,
  - A vertex v is an articulation points if removing v will disconnect the graph.
  - An edge e i s a bridge if removing e will disconnect the graph.
- All articulation points and bridges can be find by a DFS based algorithm.

Example



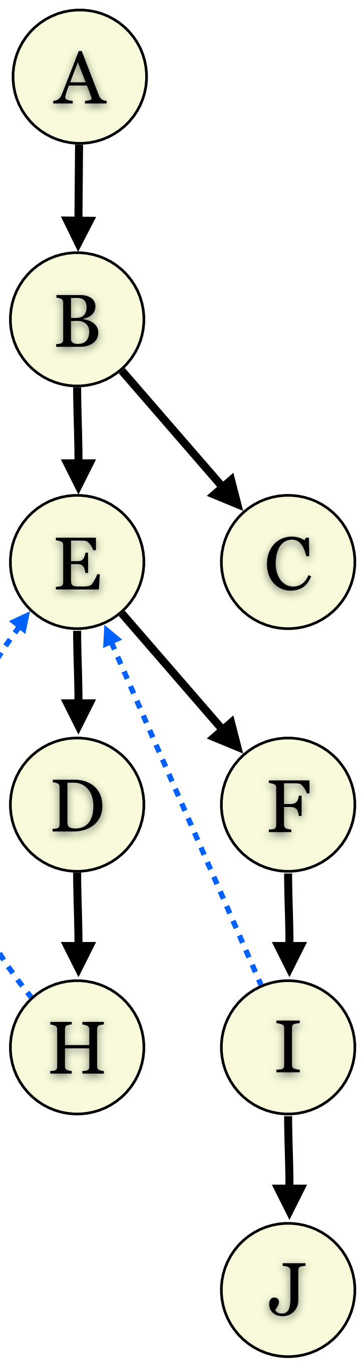
DFS Based Algorithm



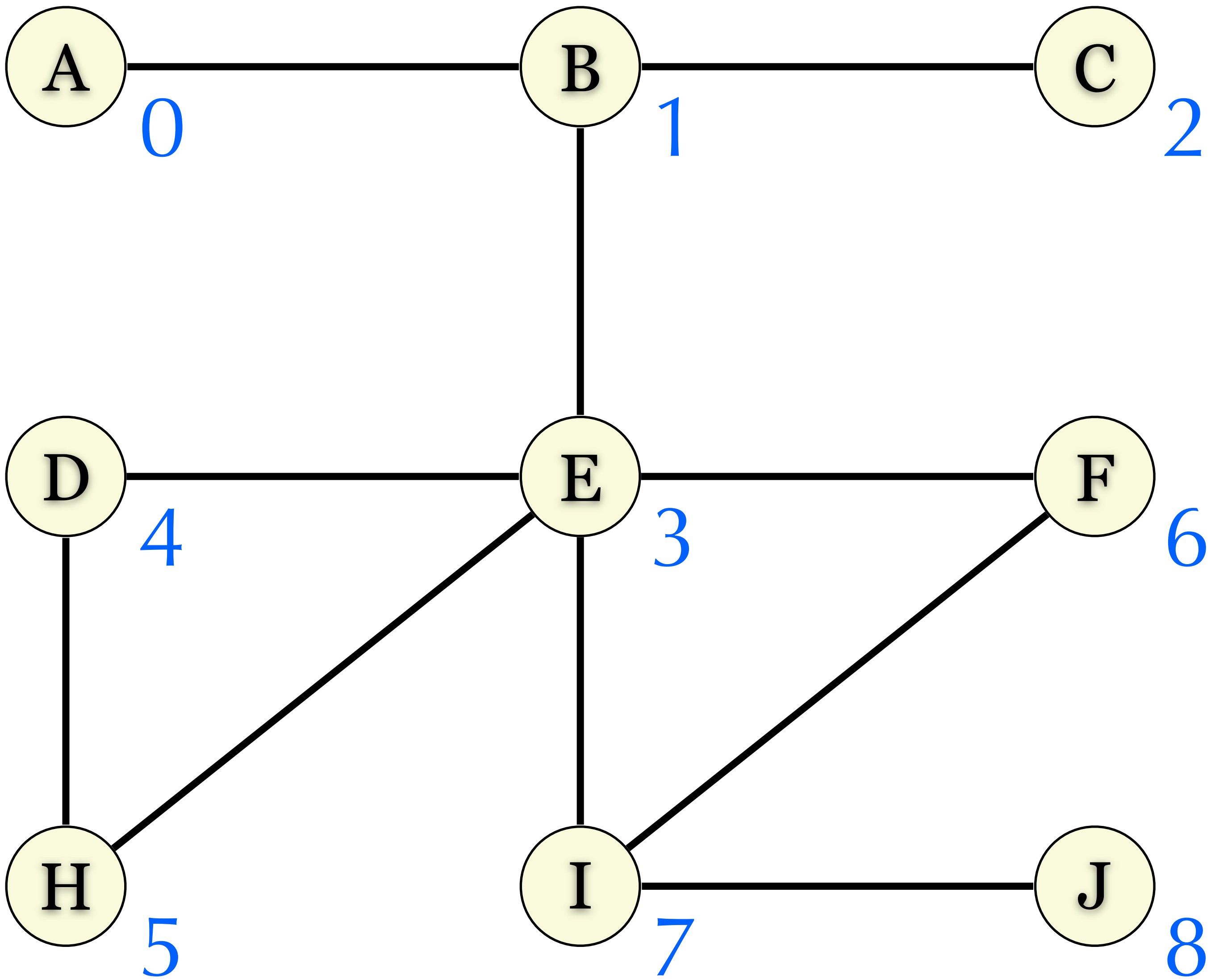
Compute
DFS order

Order

DFS tree

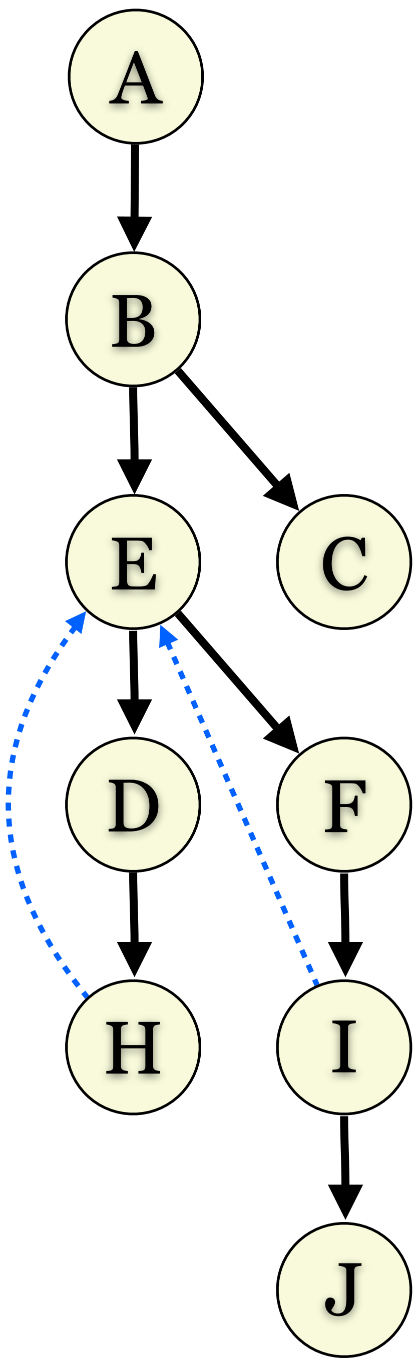


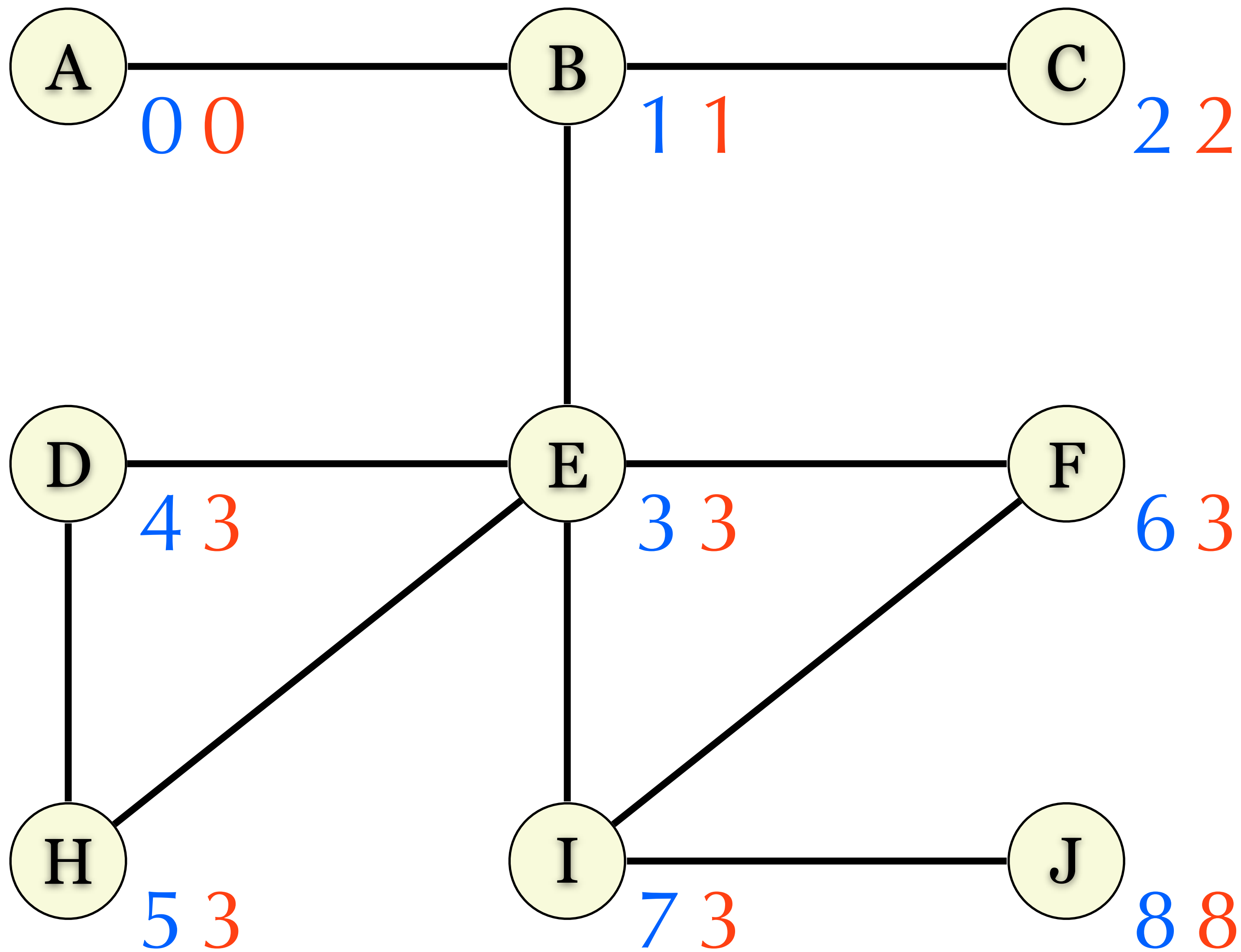
DFS order



Low Value

DFS tree





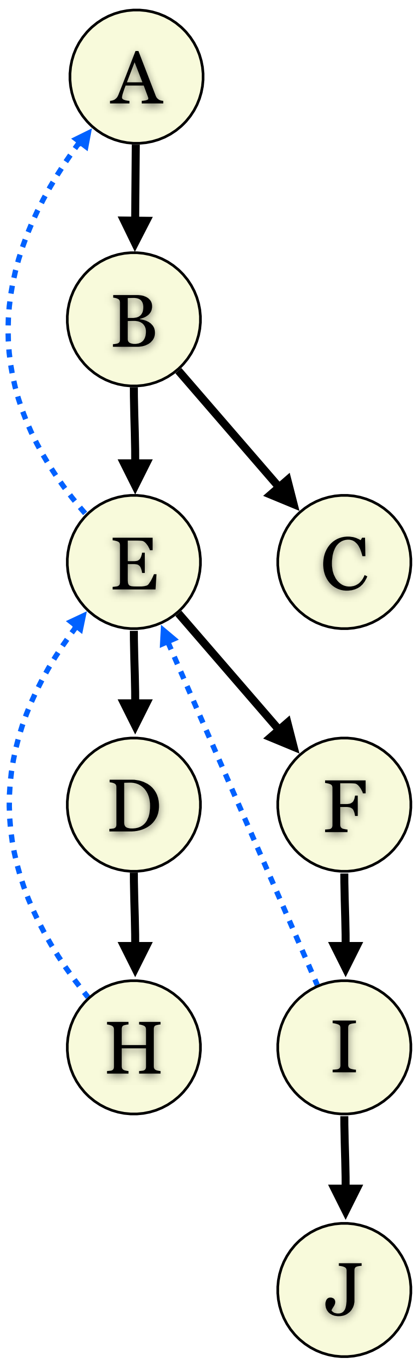
DFS order

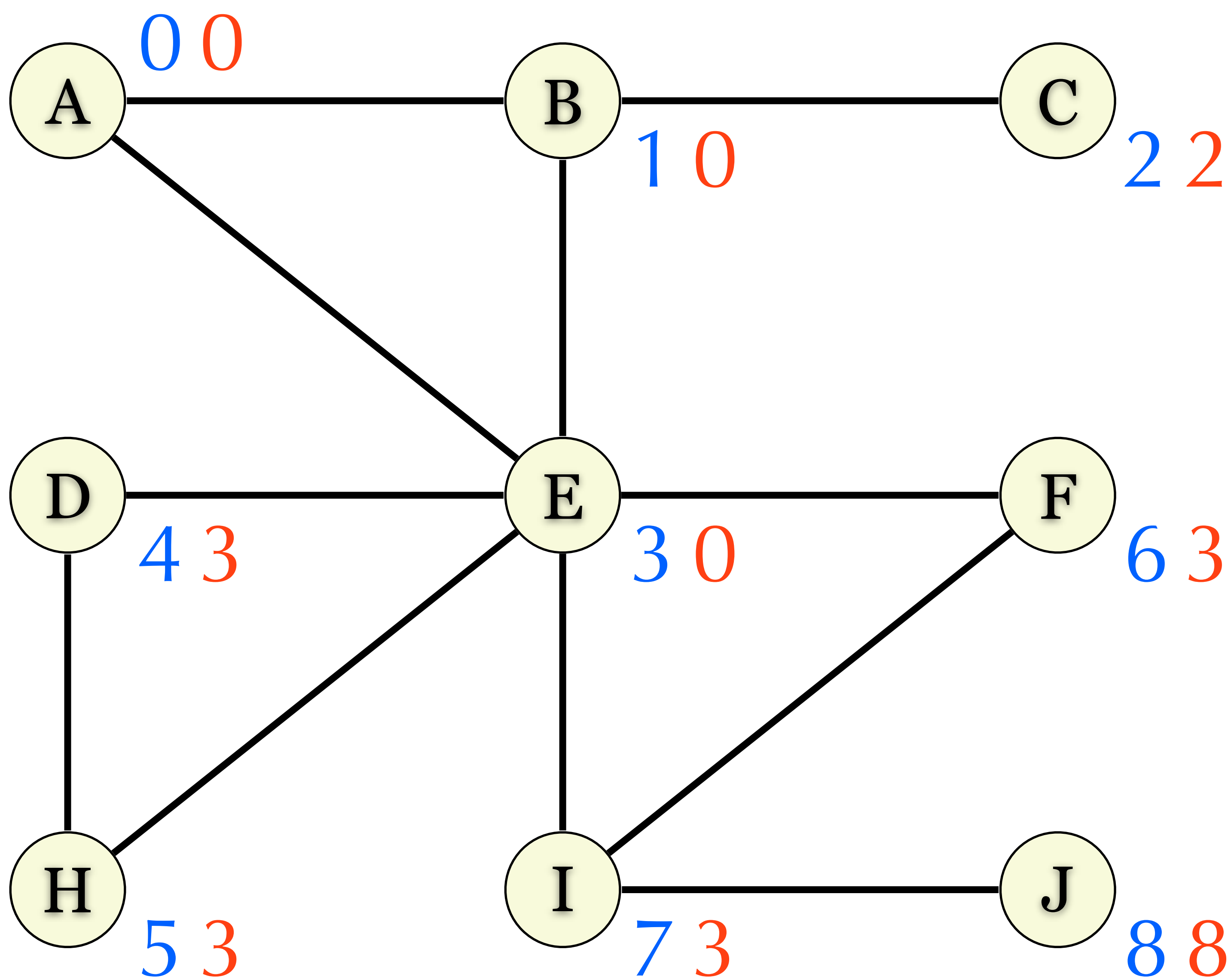
Compute
Low value

Low value of v: The lowest order of nodes which are read during DFS-visit(v).

Another Example

DFS tree

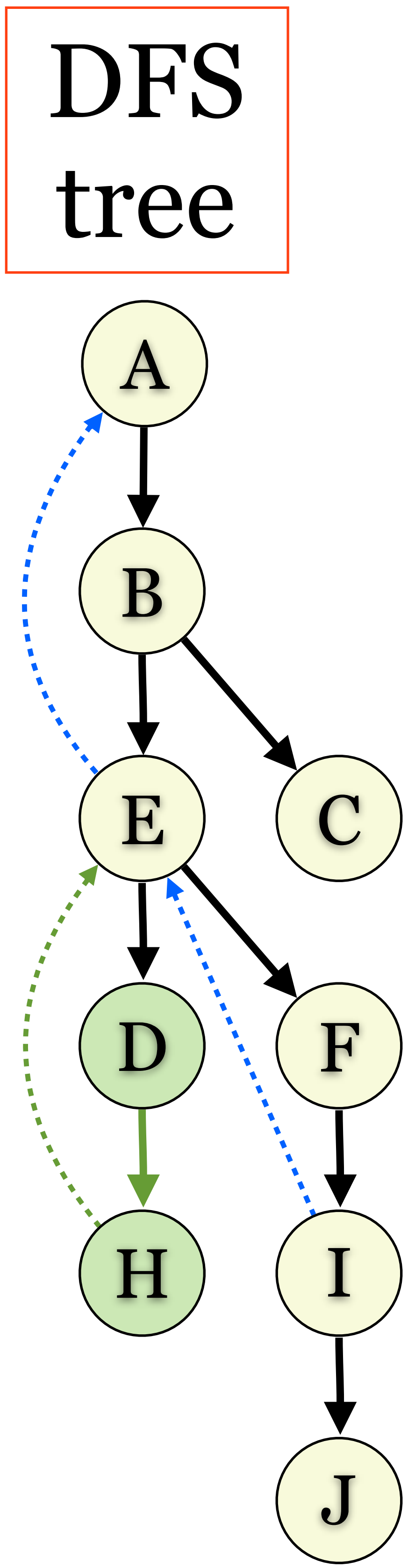


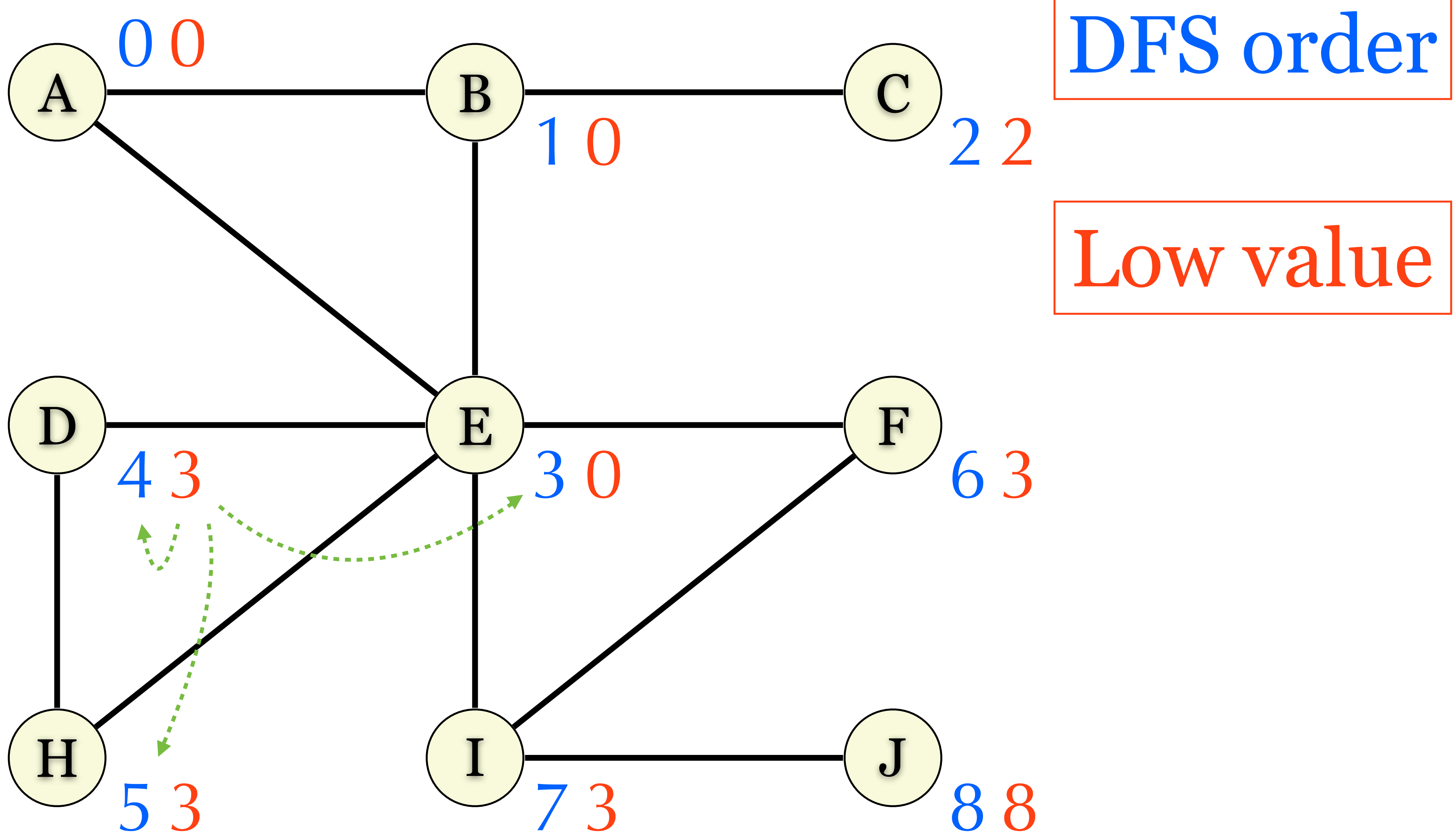


DFS order

Low value

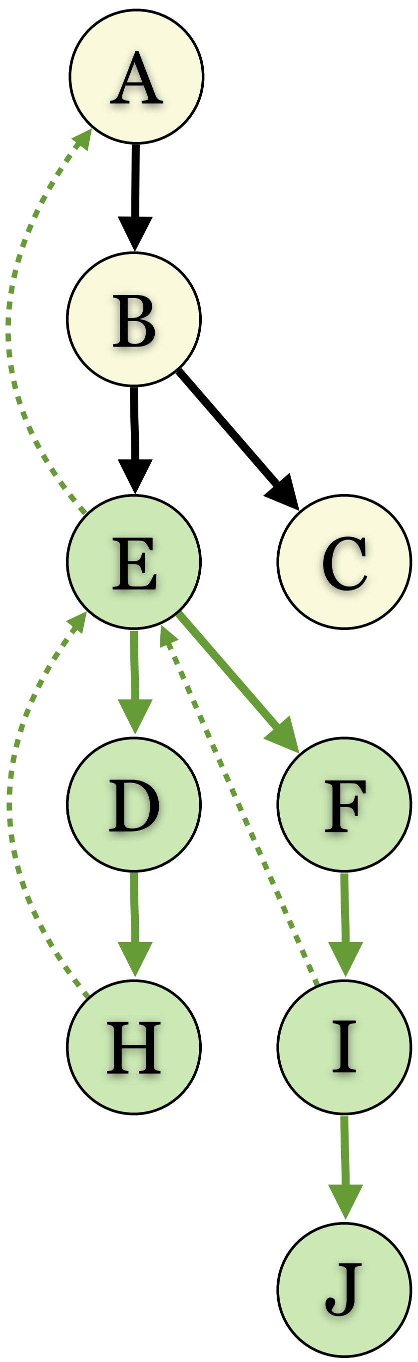
Another Example

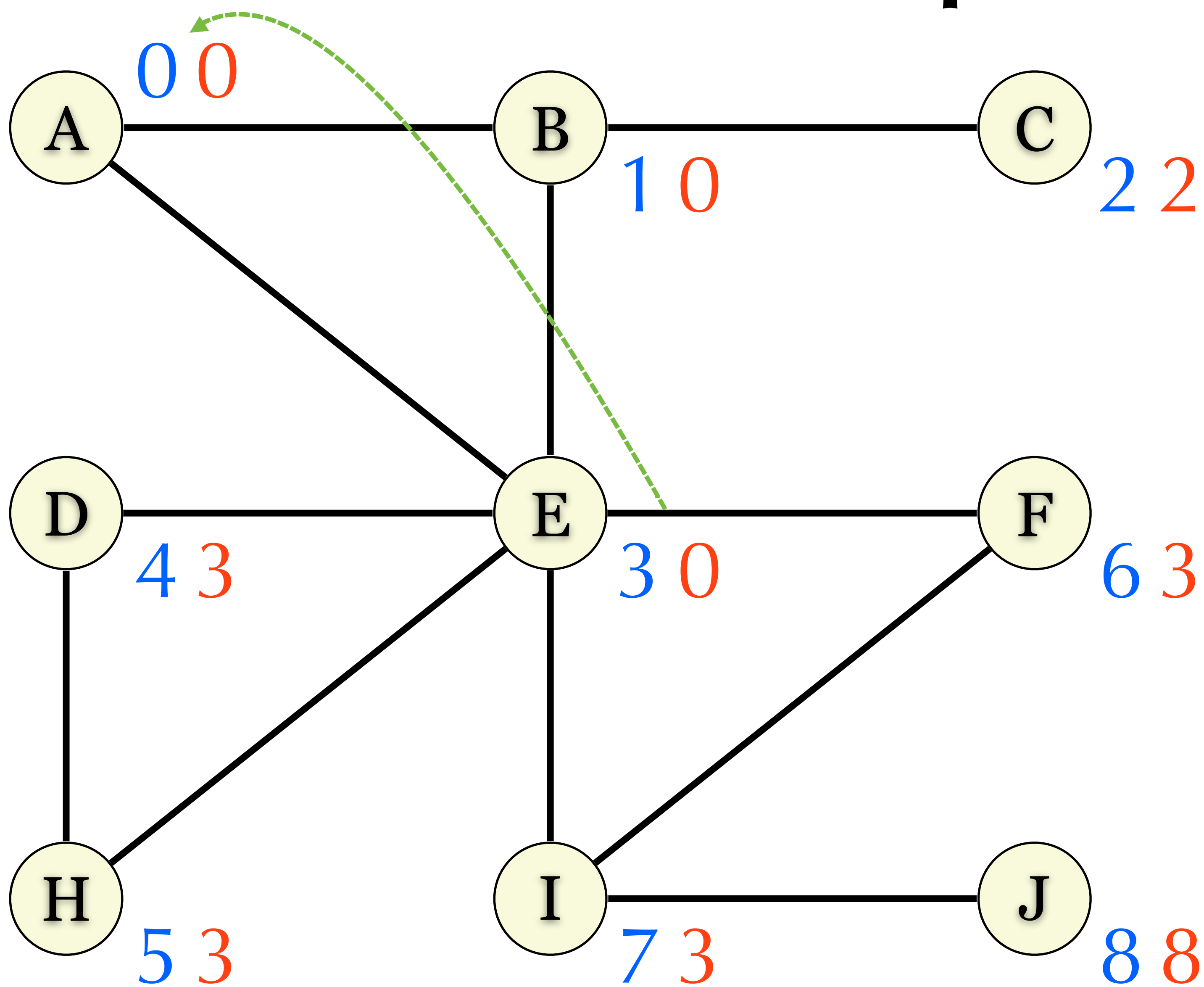




Another Example

DFS tree

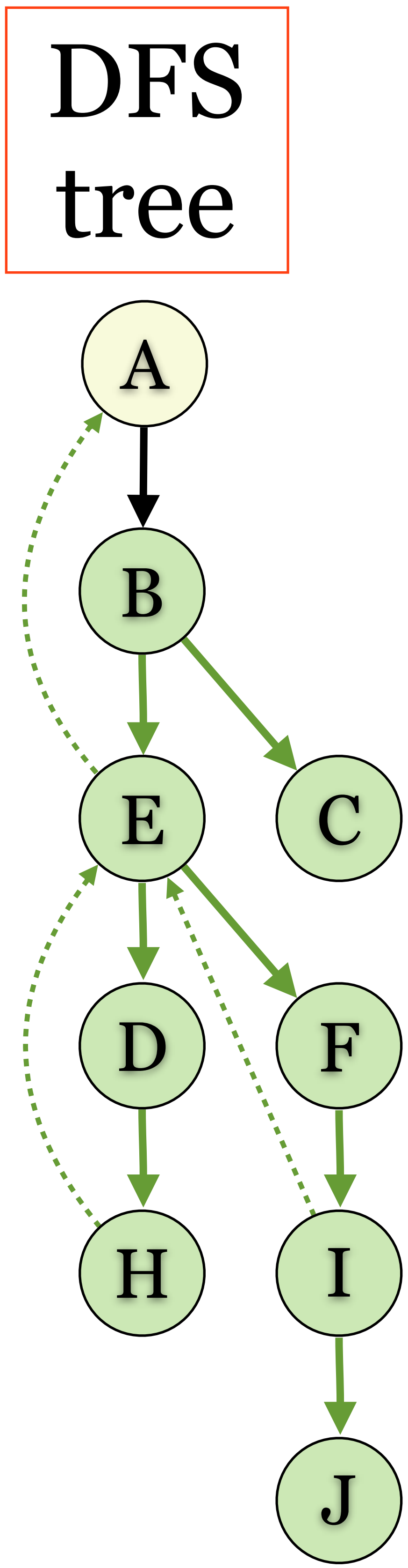


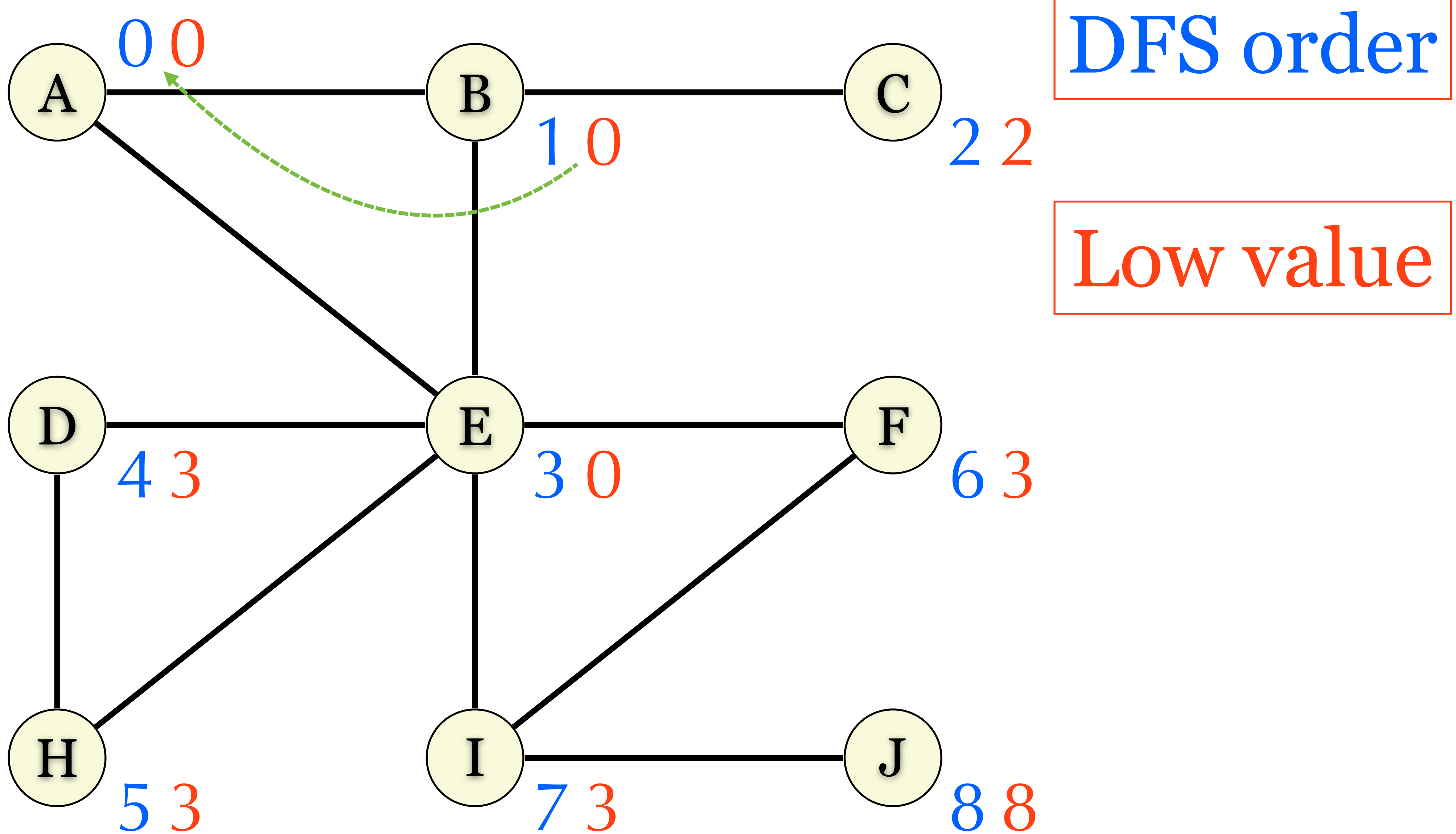


DFS order

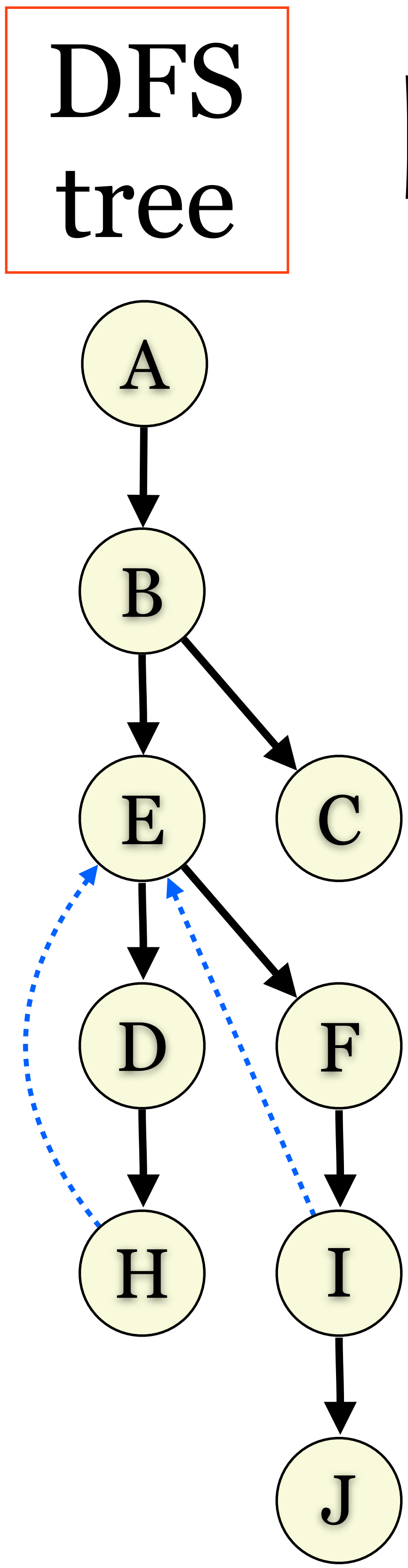
Low value

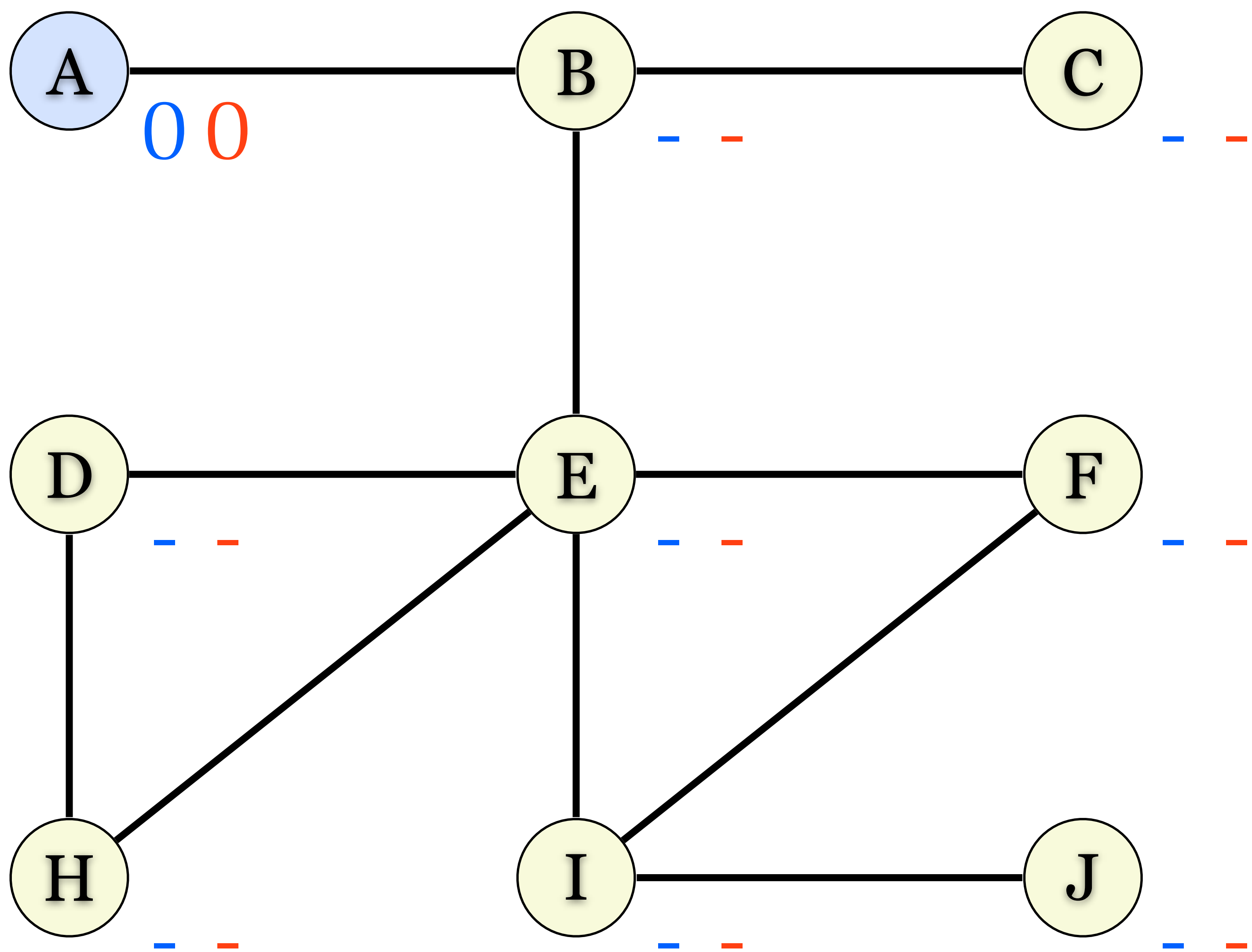
Another Example





DFS Based Algorithm



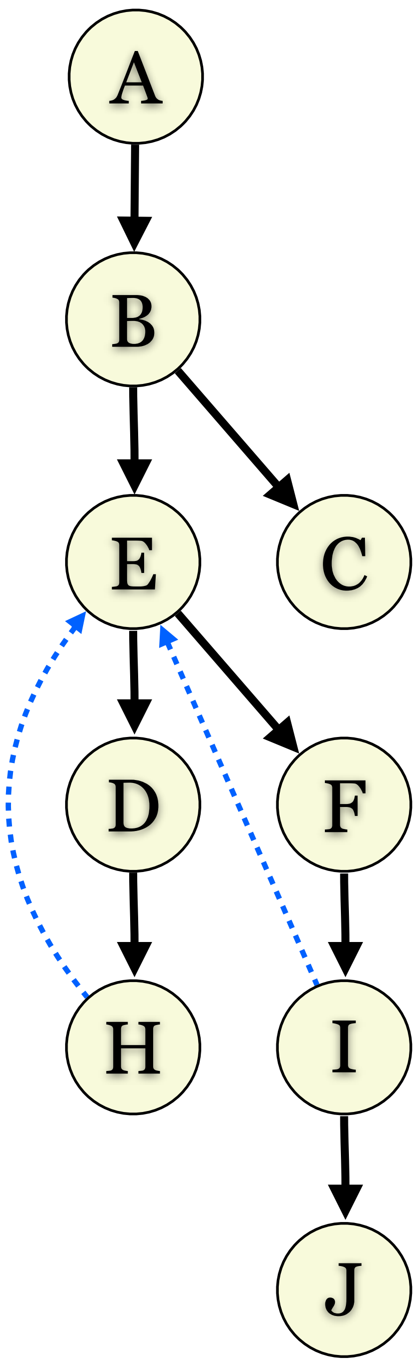


DFS order

Compute Low value

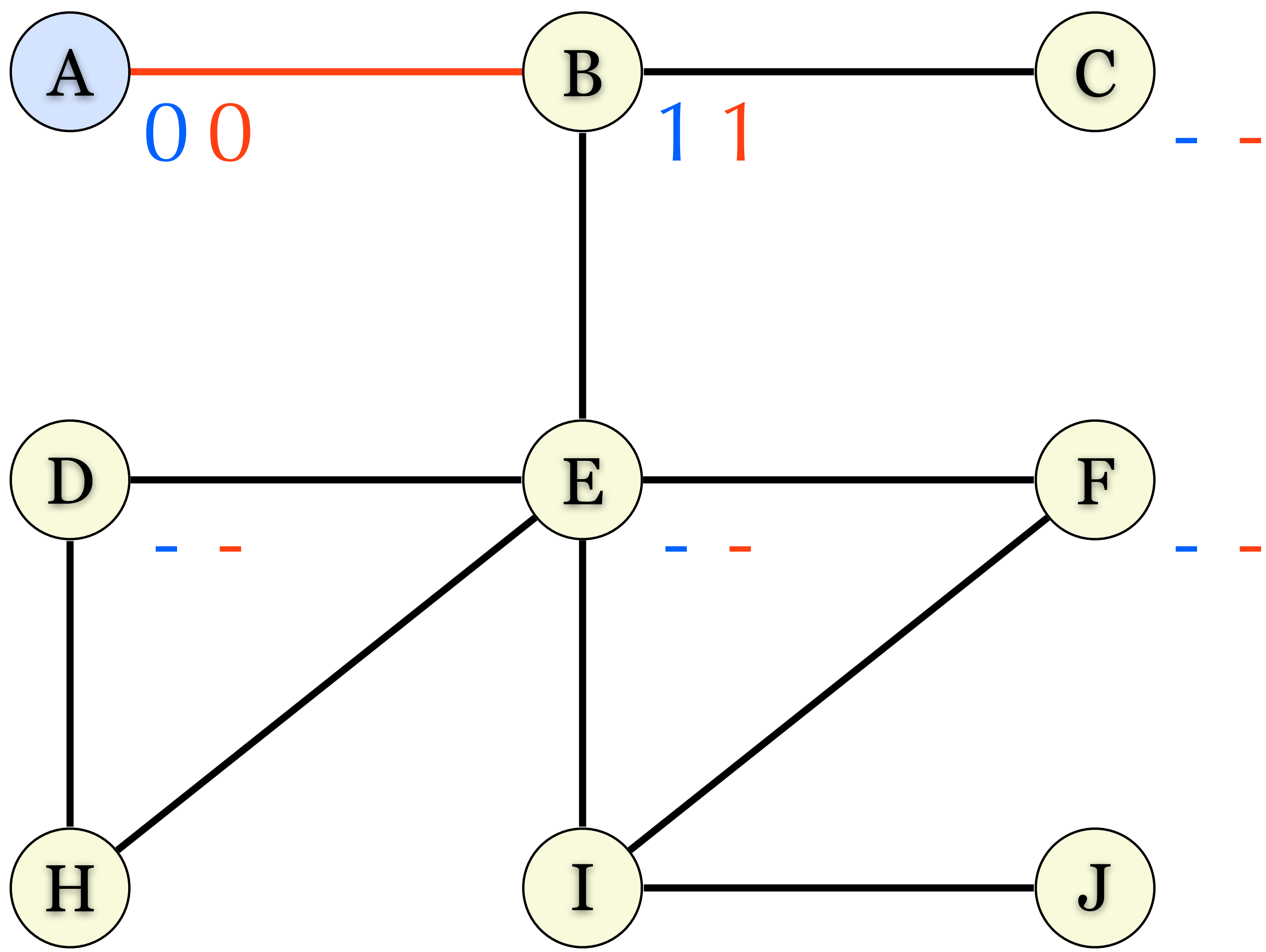
DFS Based Algorithm

DFS tree



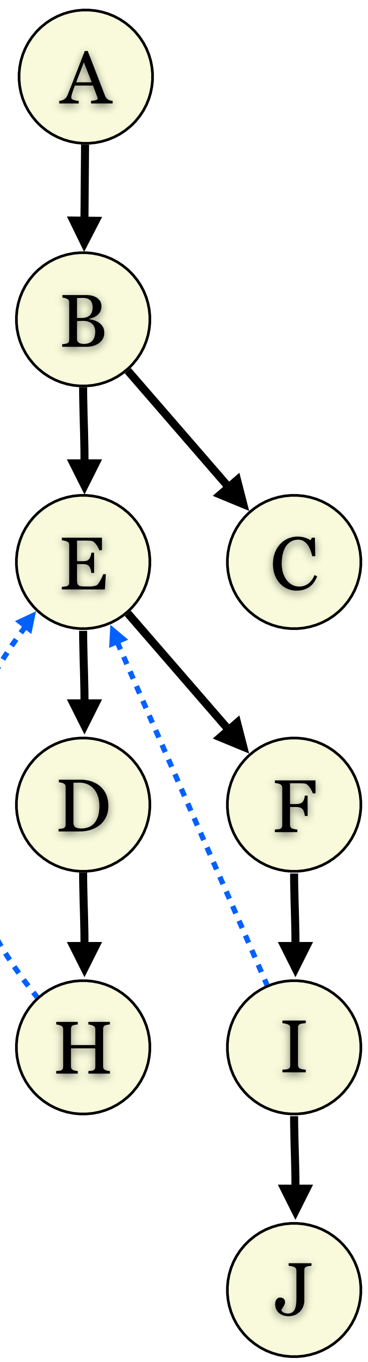
DFS order

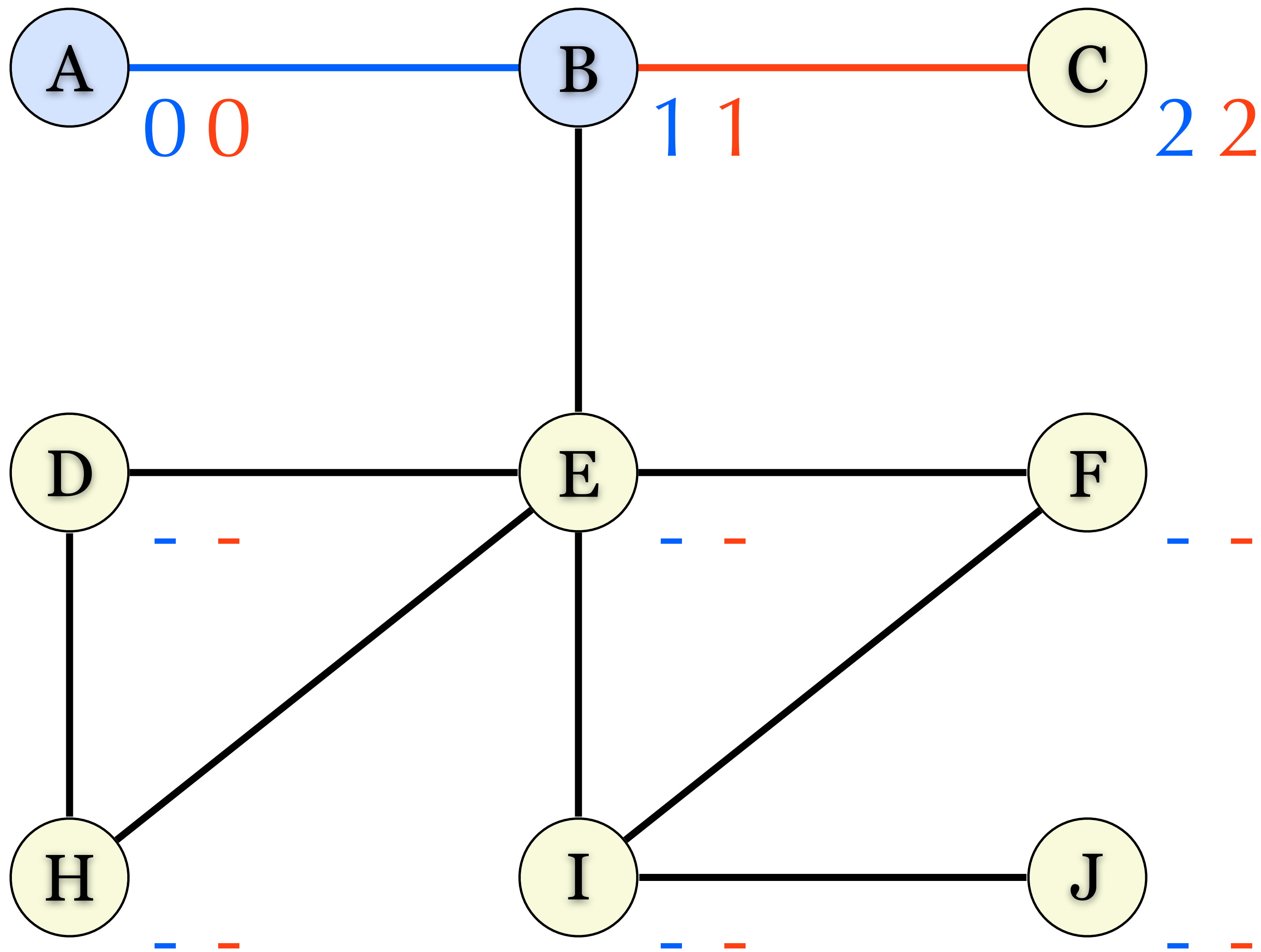
Compute
Low value



DFS Based Algorithm

DFS tree



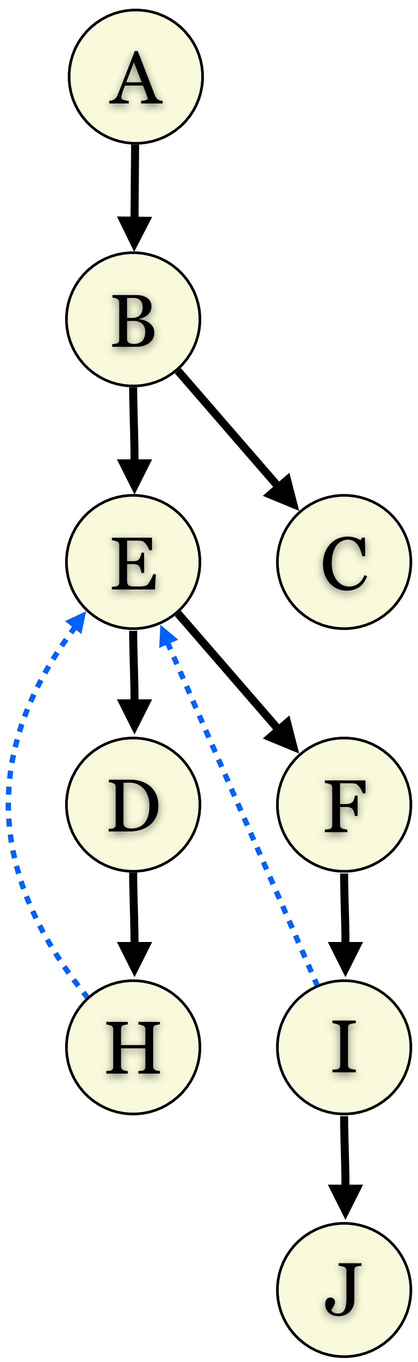


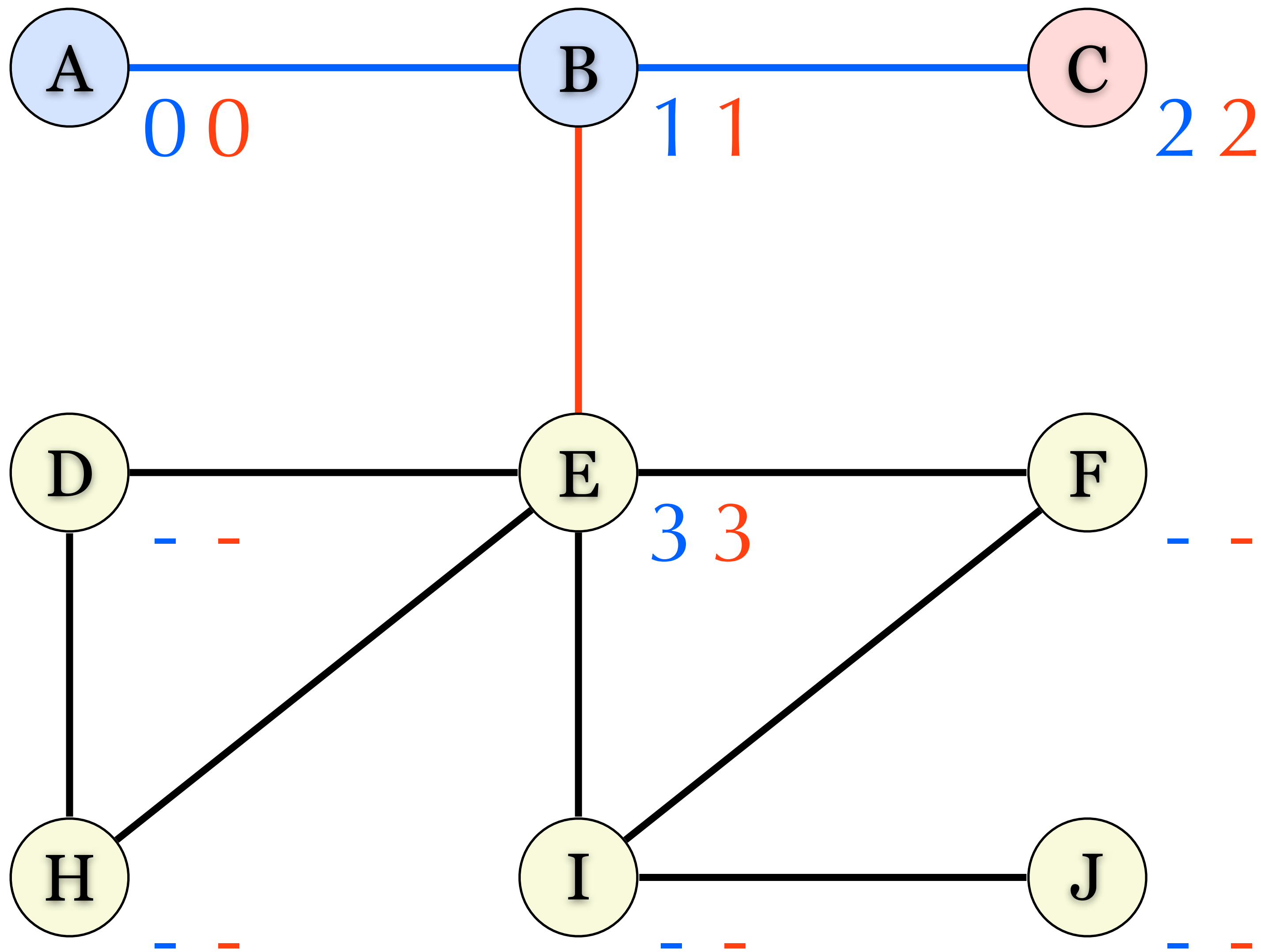
DFS order

Compute Low value

DFS Based Algorithm

DFS tree



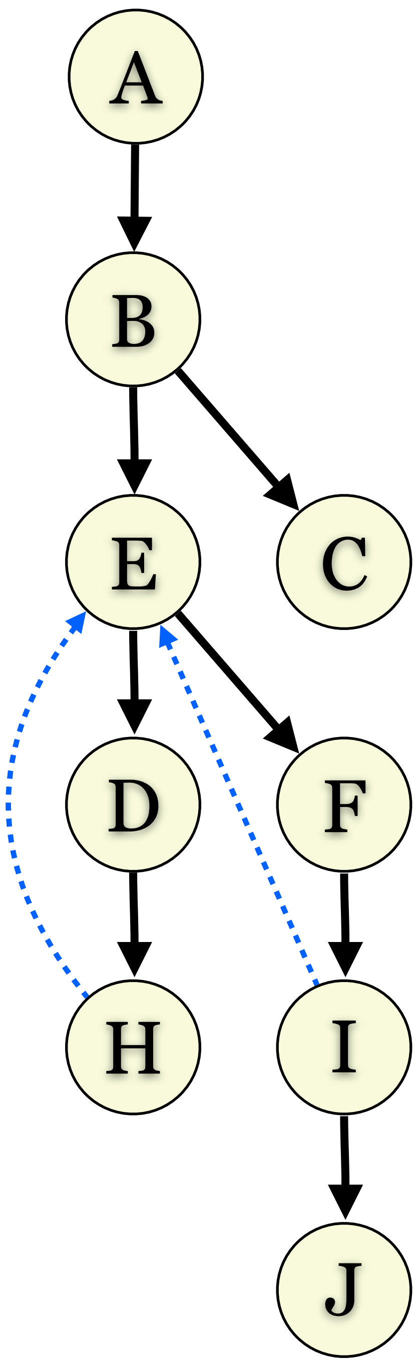


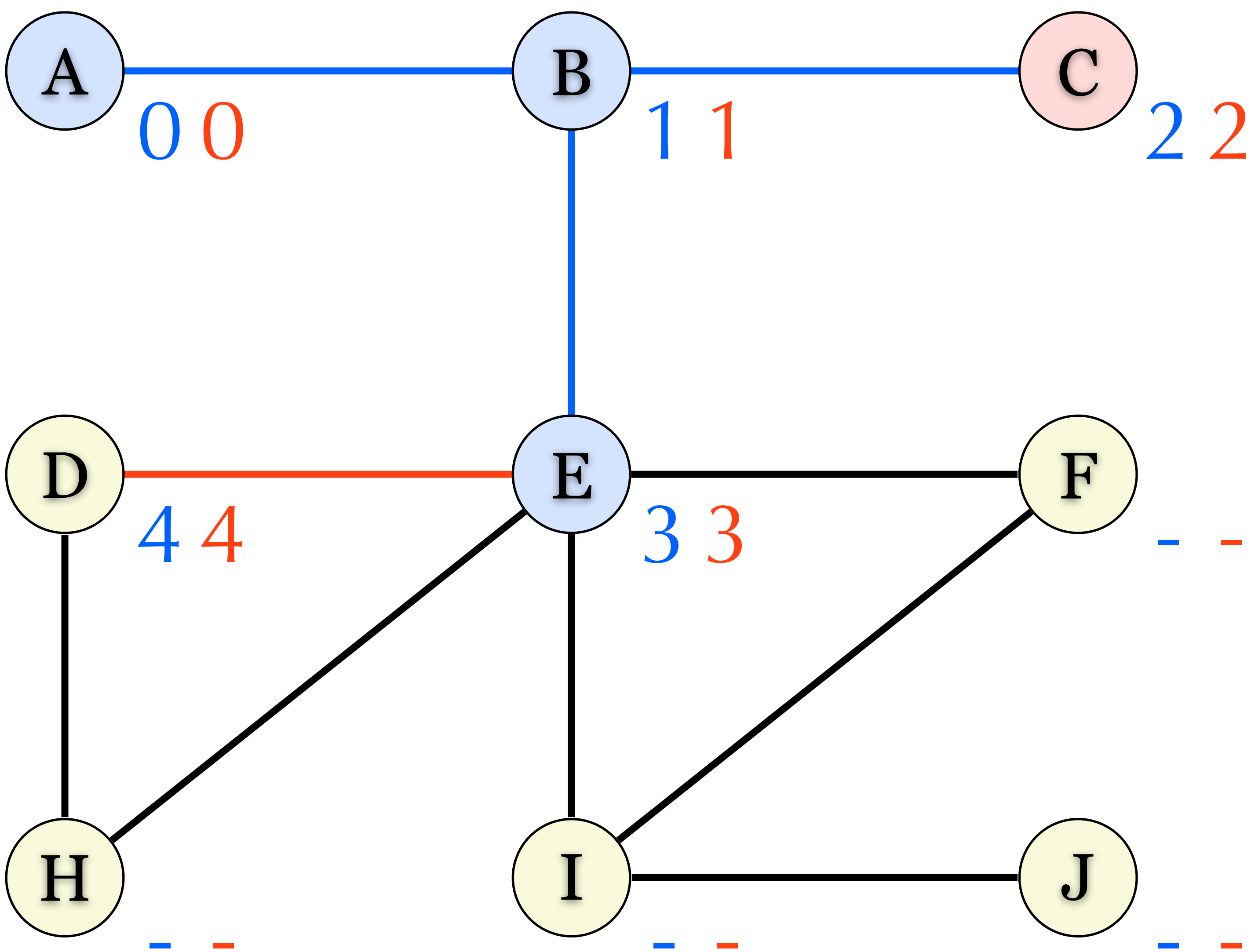
DFS order

Compute Low value

DFS tree

DFS Based Algorithm

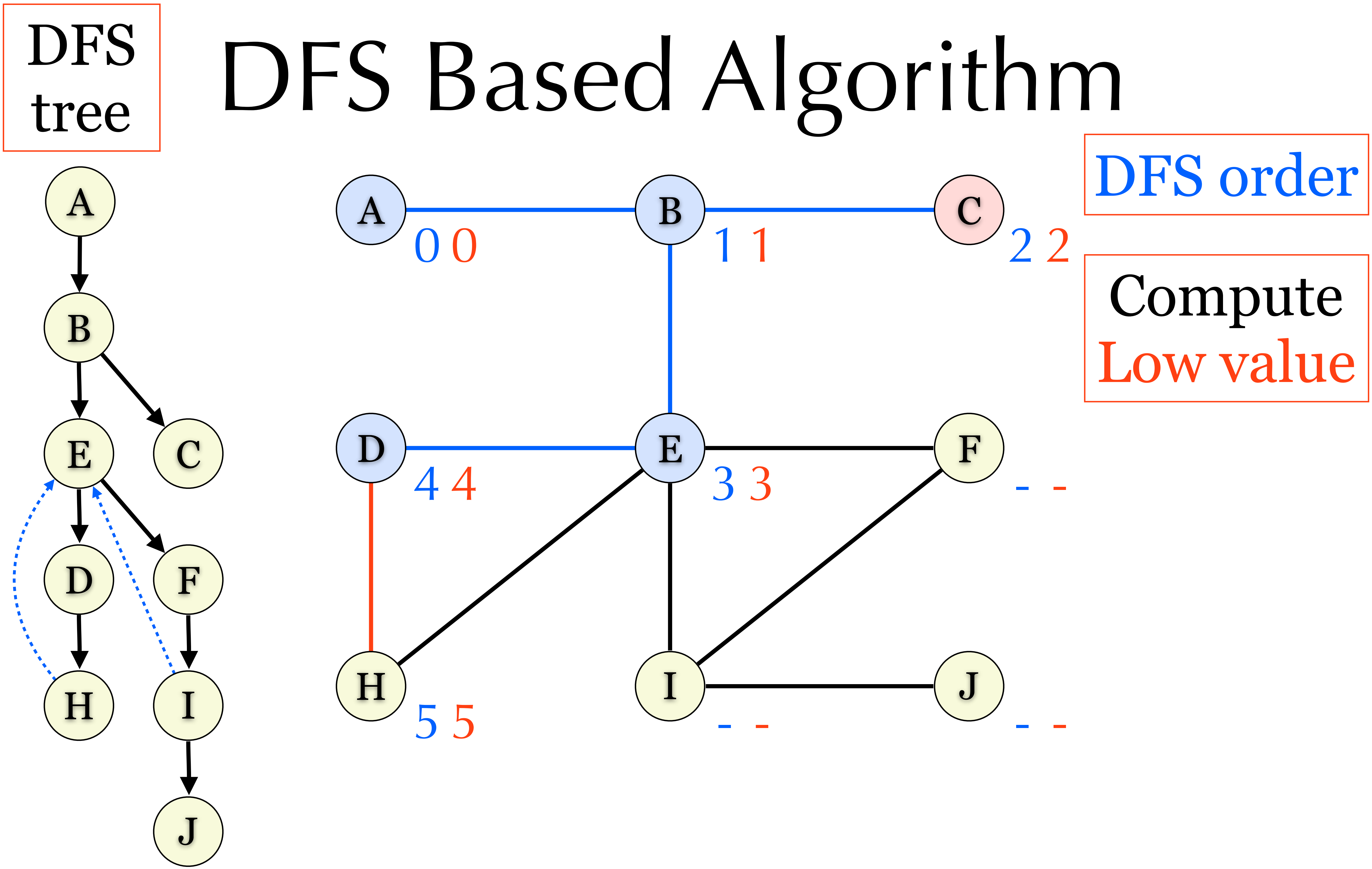




DFS order

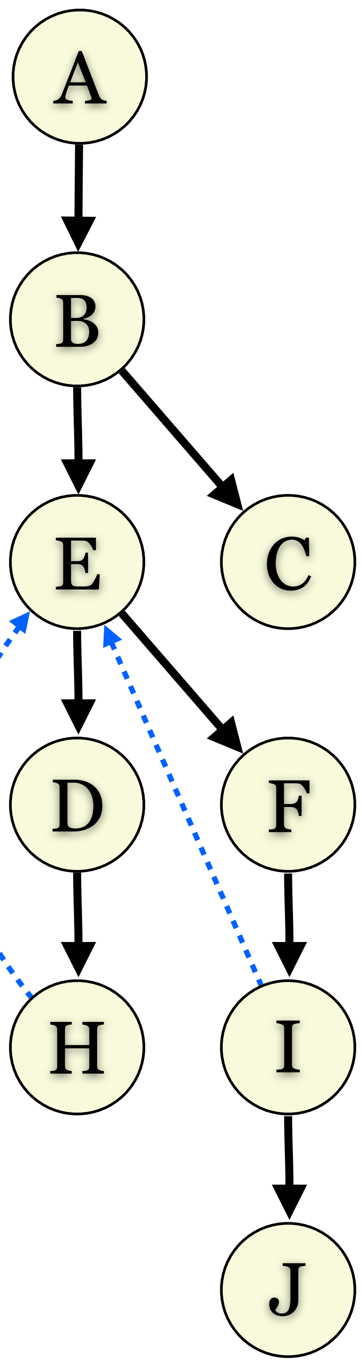
Compute
Low value

DFS Based Algorithm



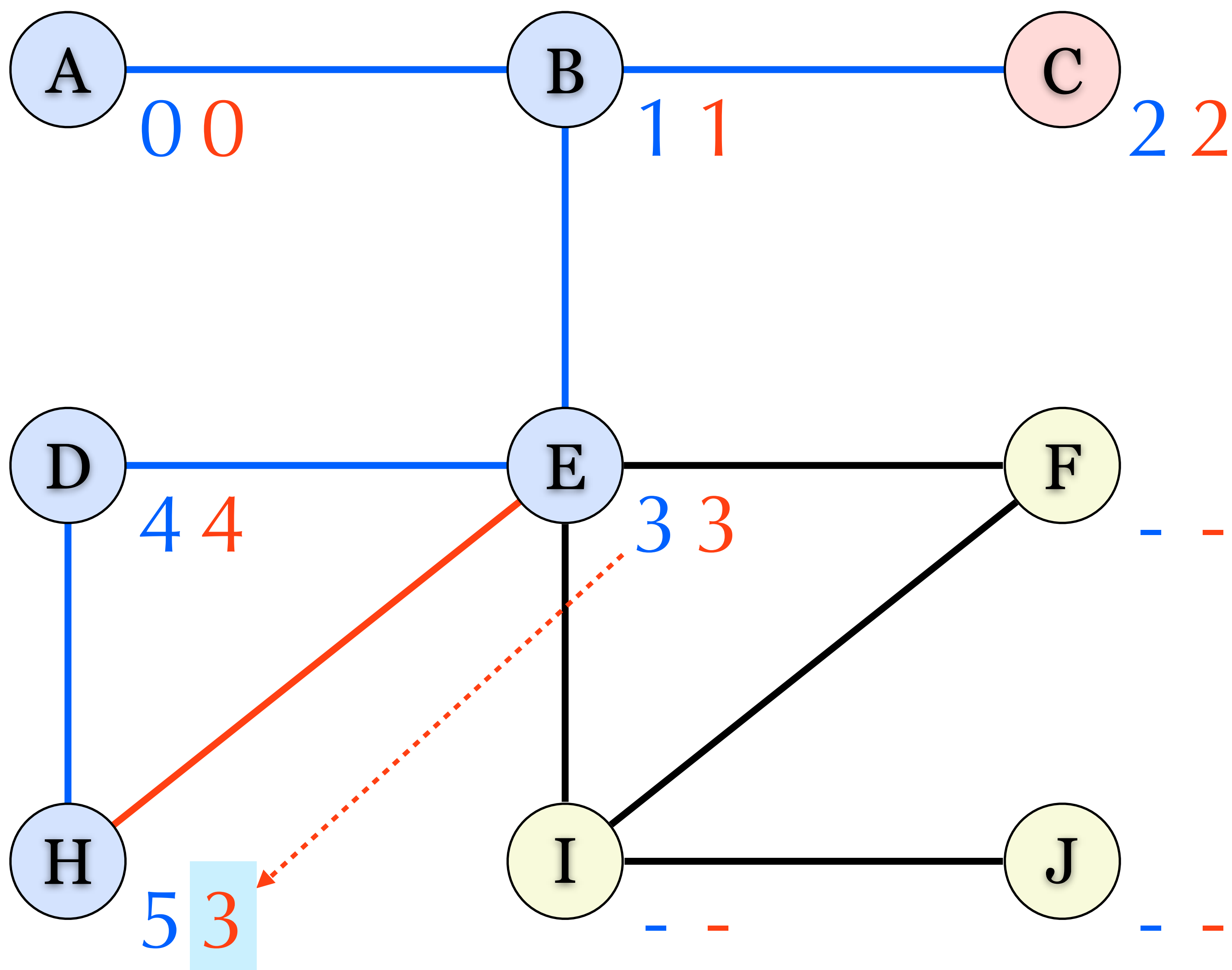
DFS Based Algorithm

DFS tree



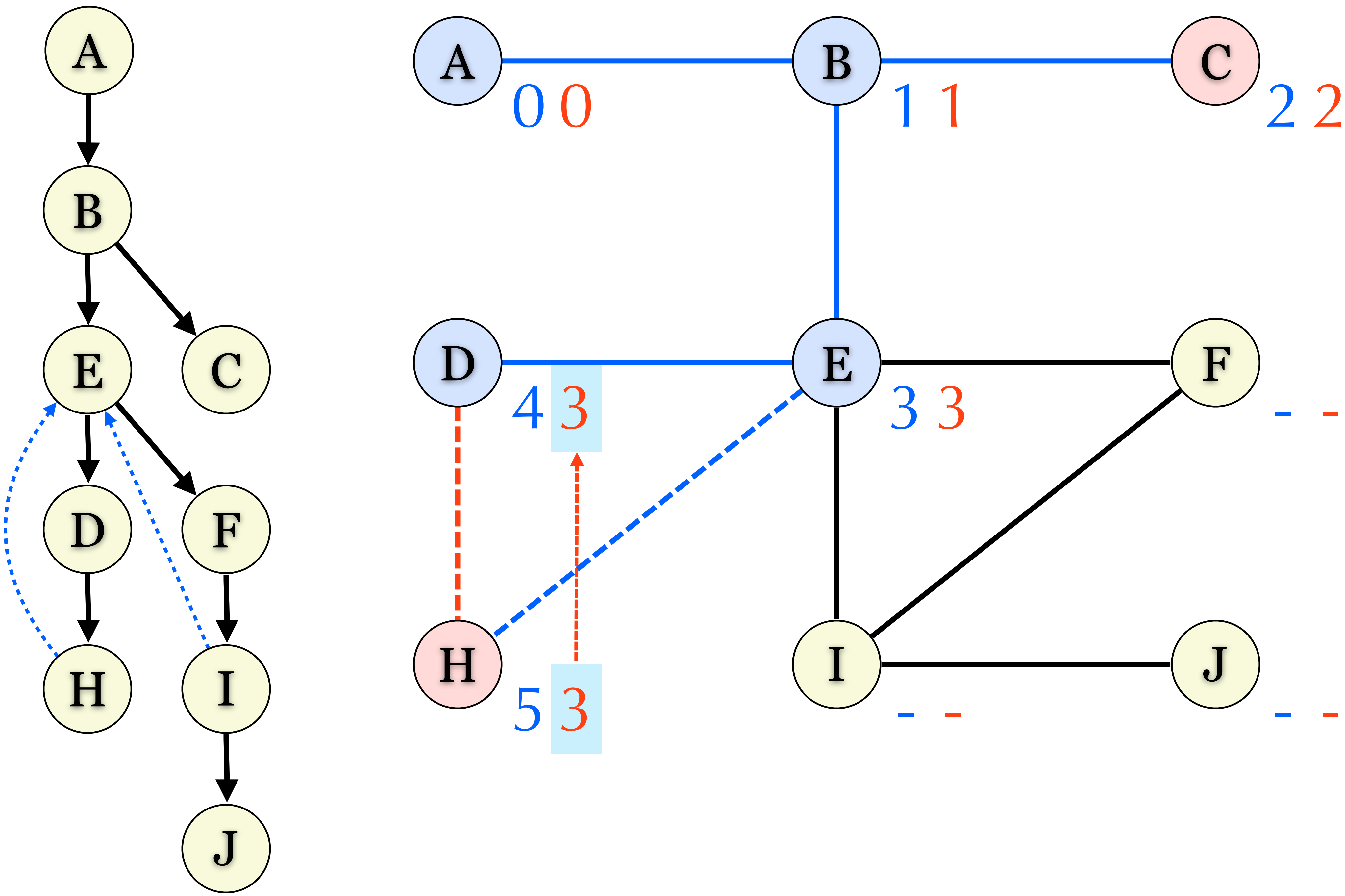
DFS order

Compute Low value



DFS Based Algorithm

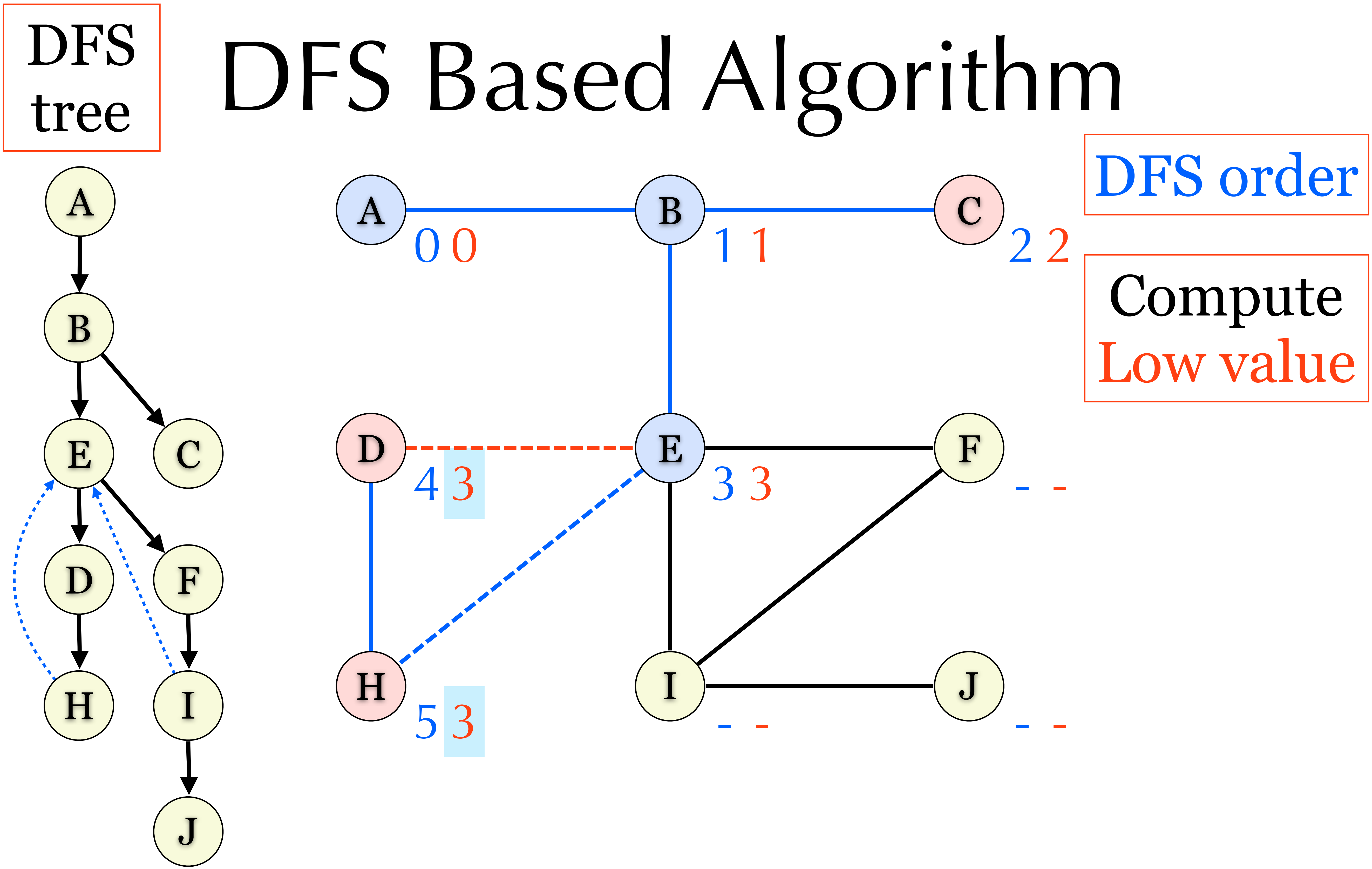
DFS tree



DFS order

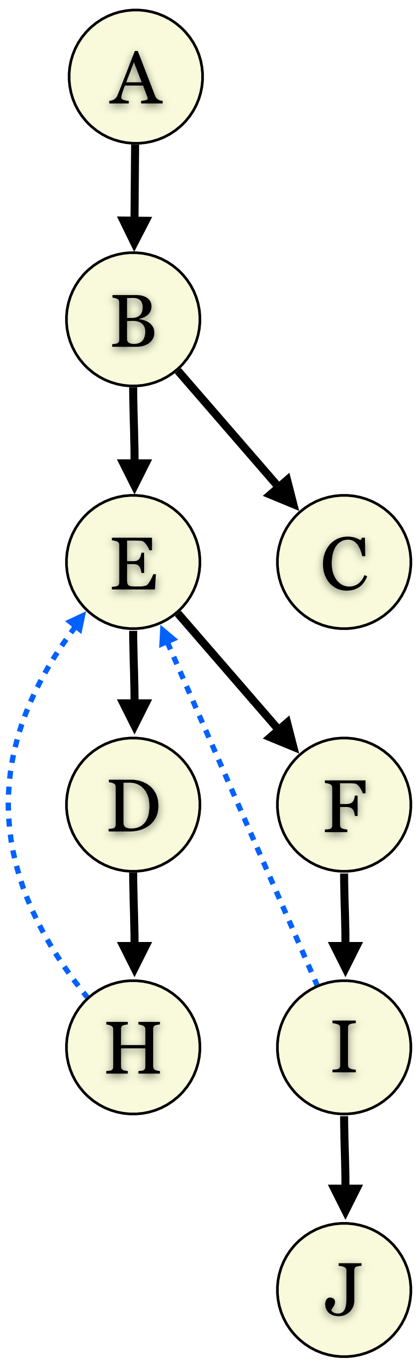
Compute
Low value

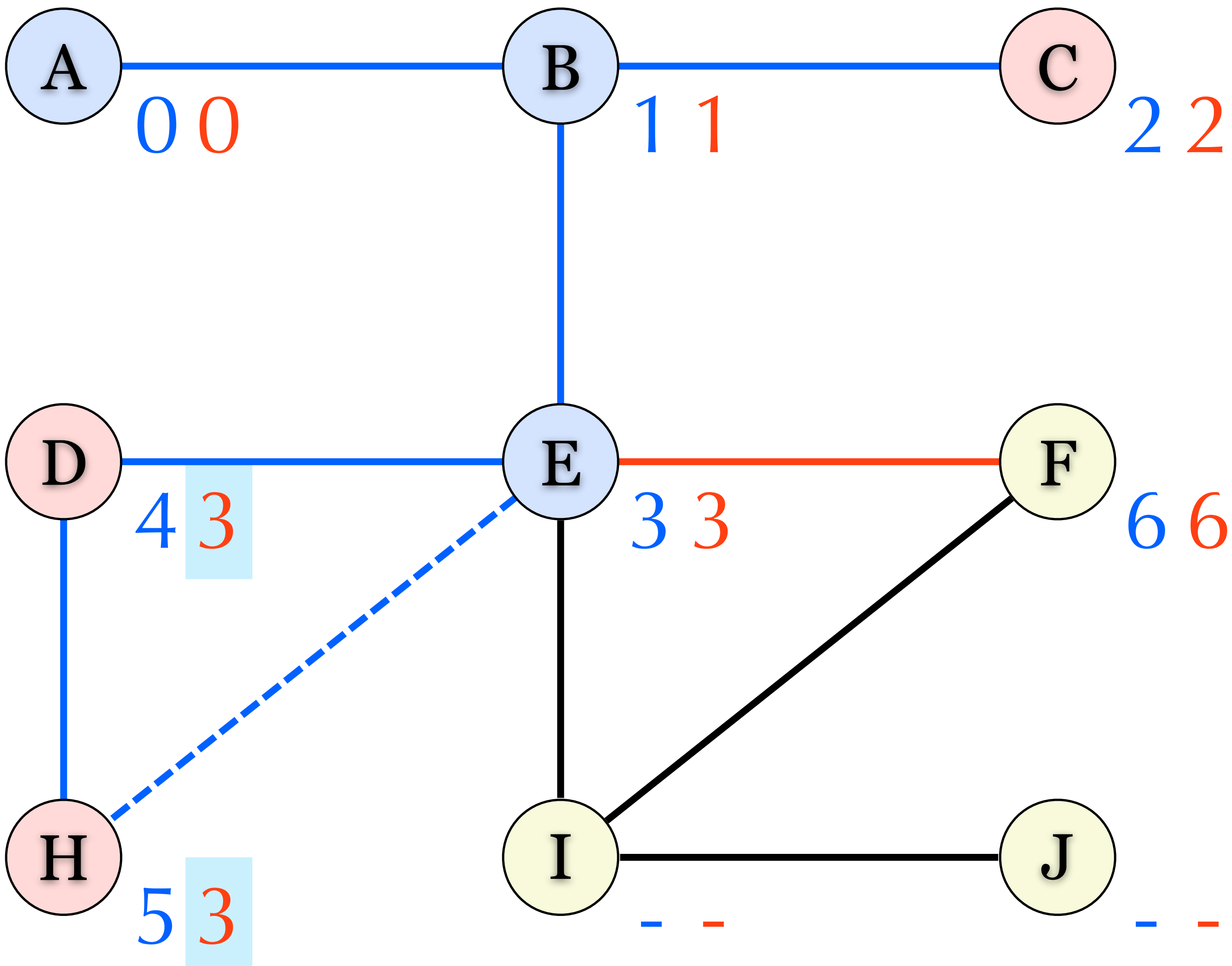
DFS Based Algorithm



DFS Based Algorithm

DFS tree

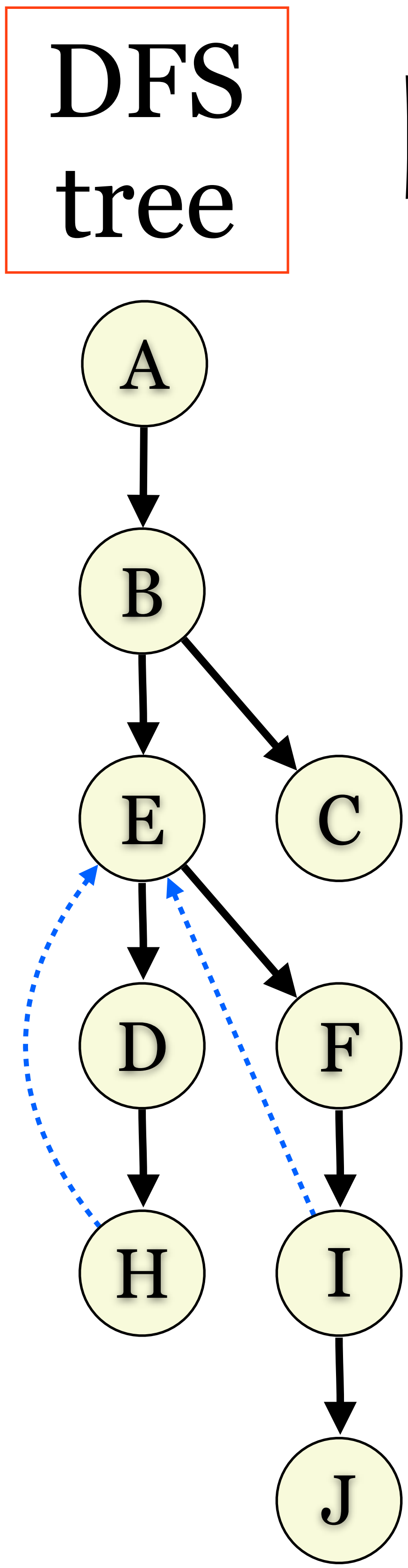


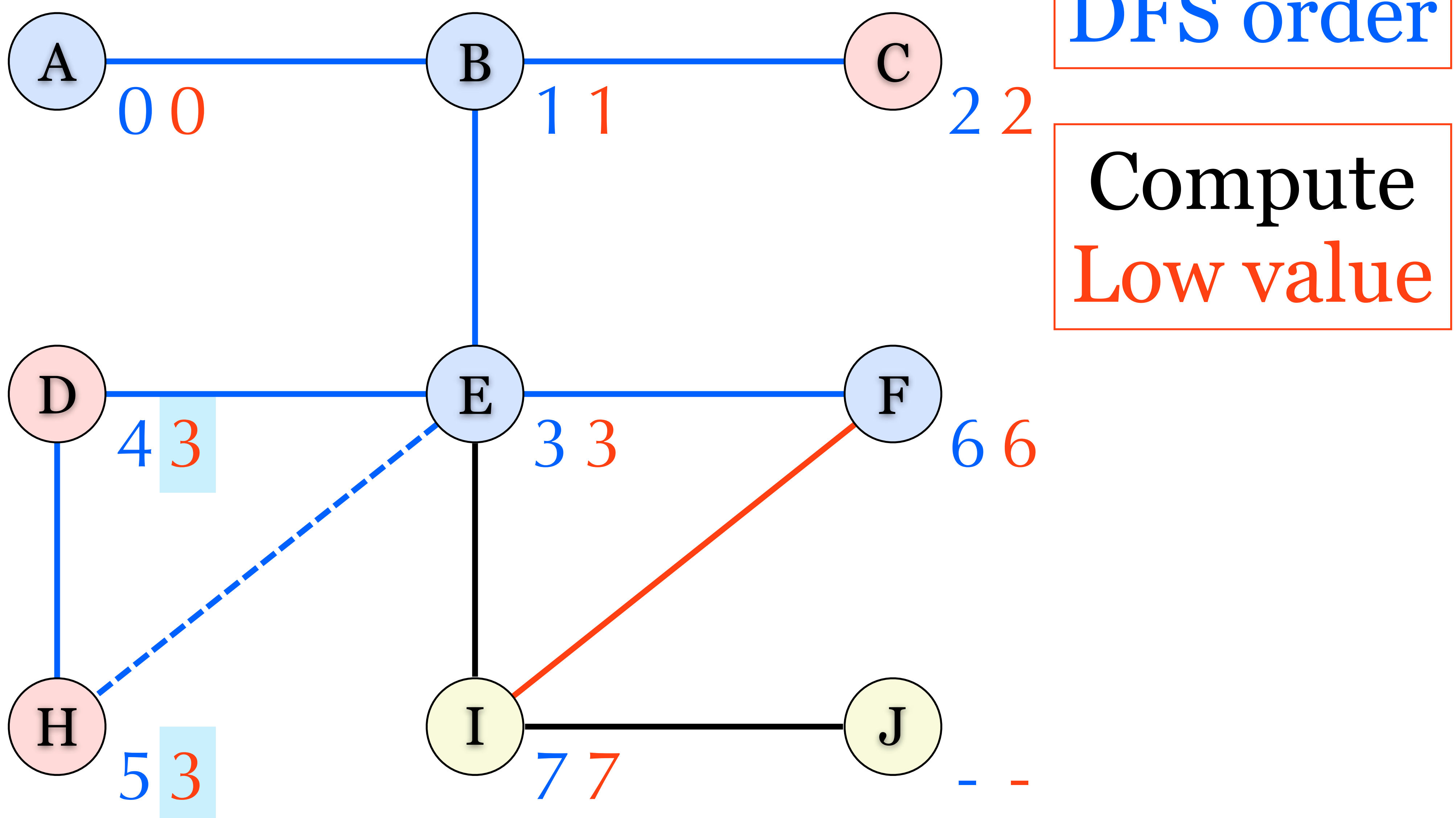


DFS order

Compute
Low value

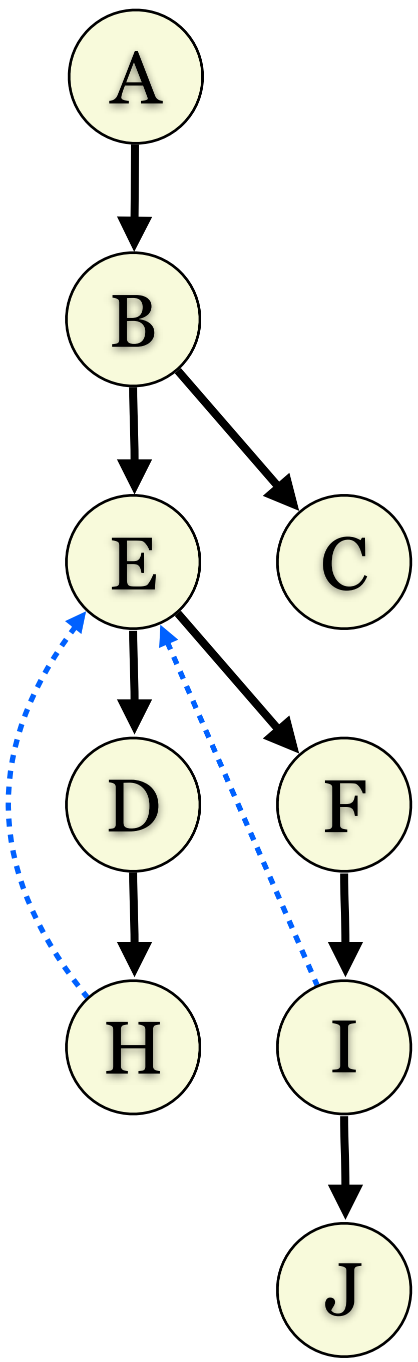
DFS Based Algorithm

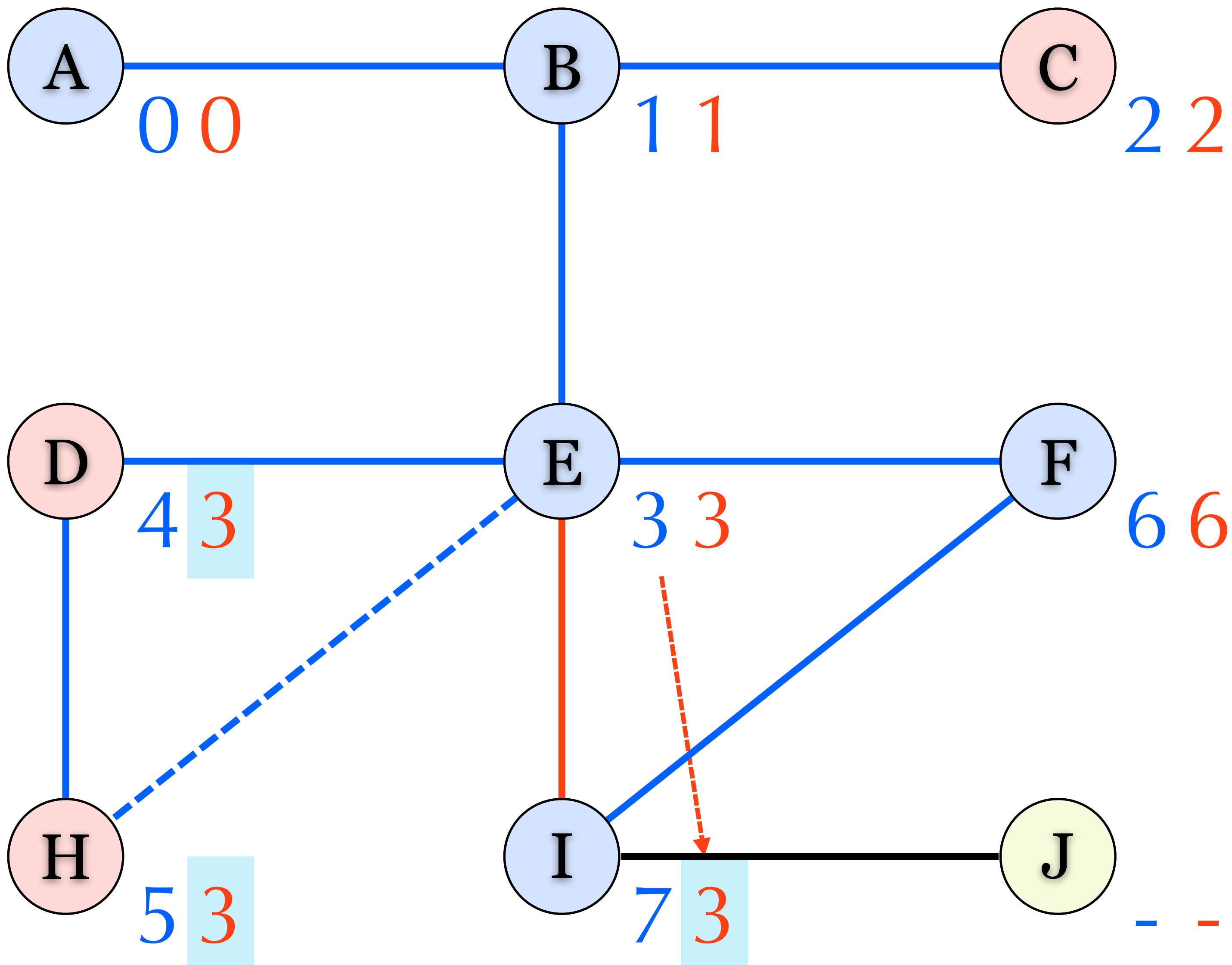




DFS Based Algorithm

DFS tree



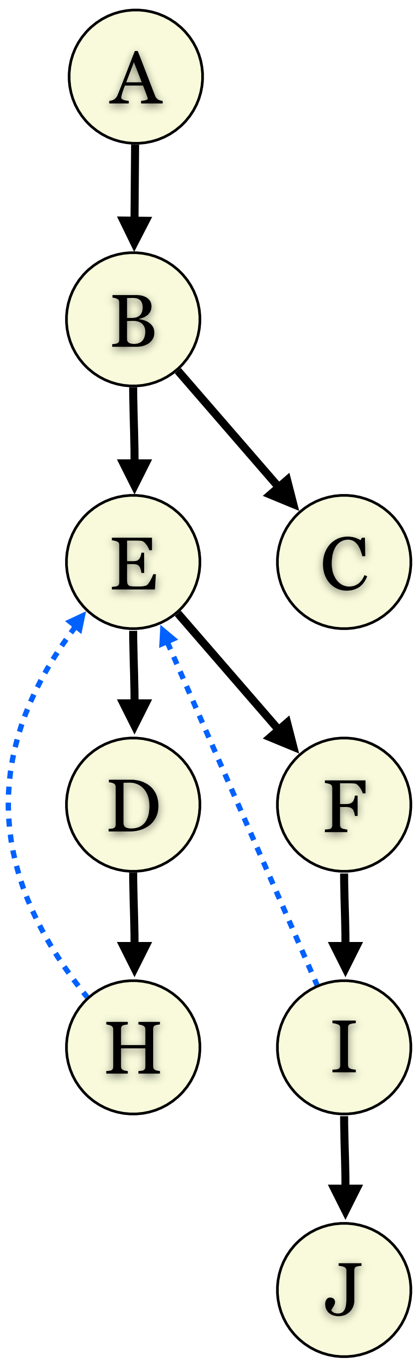


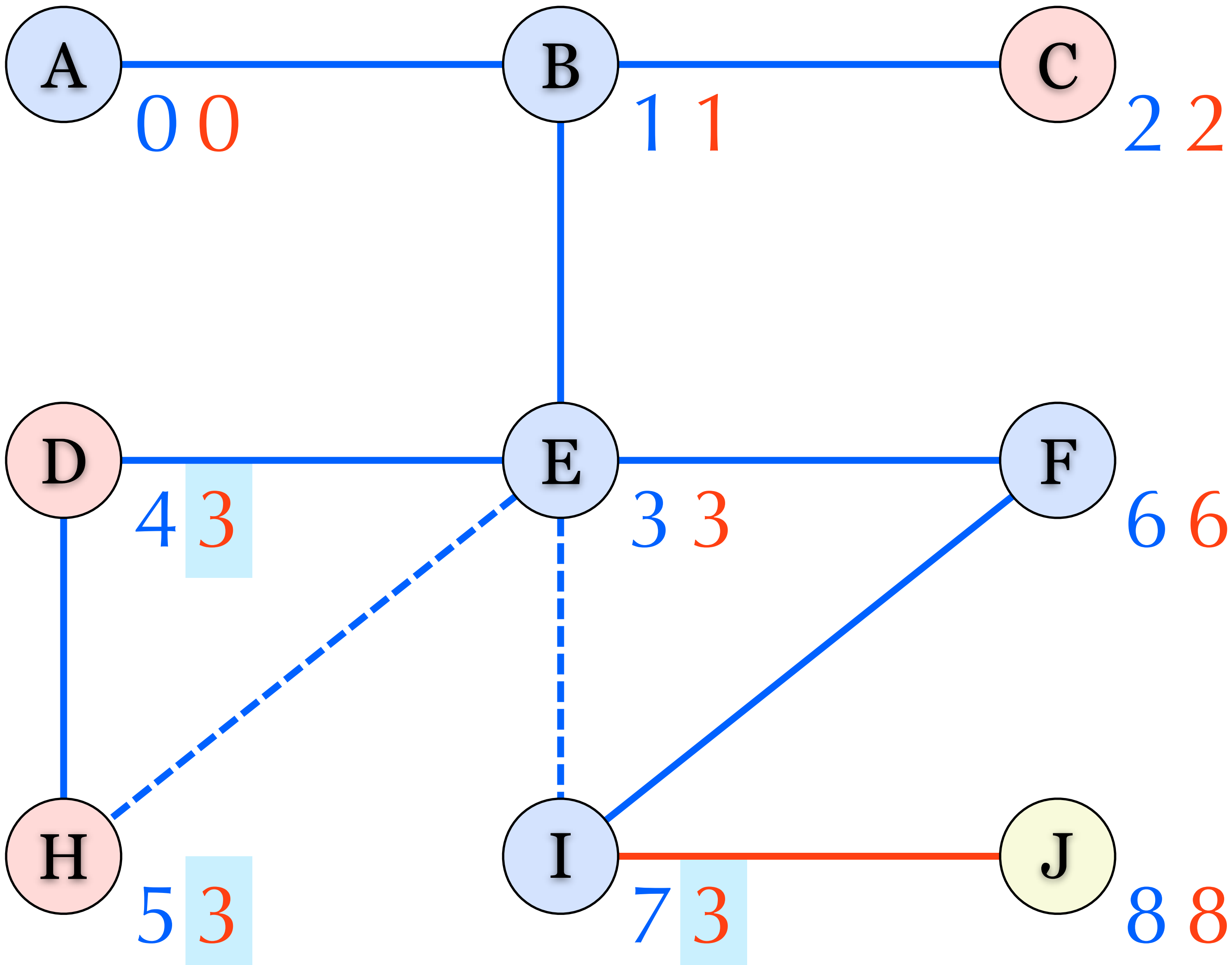
DFS order

Compute
Low value

DFS Based Algorithm

DFS tree



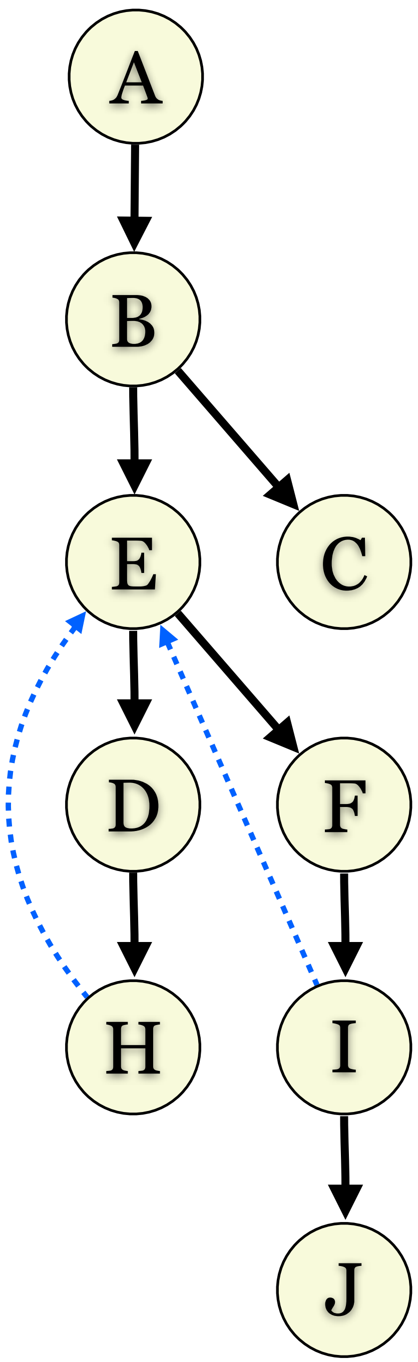


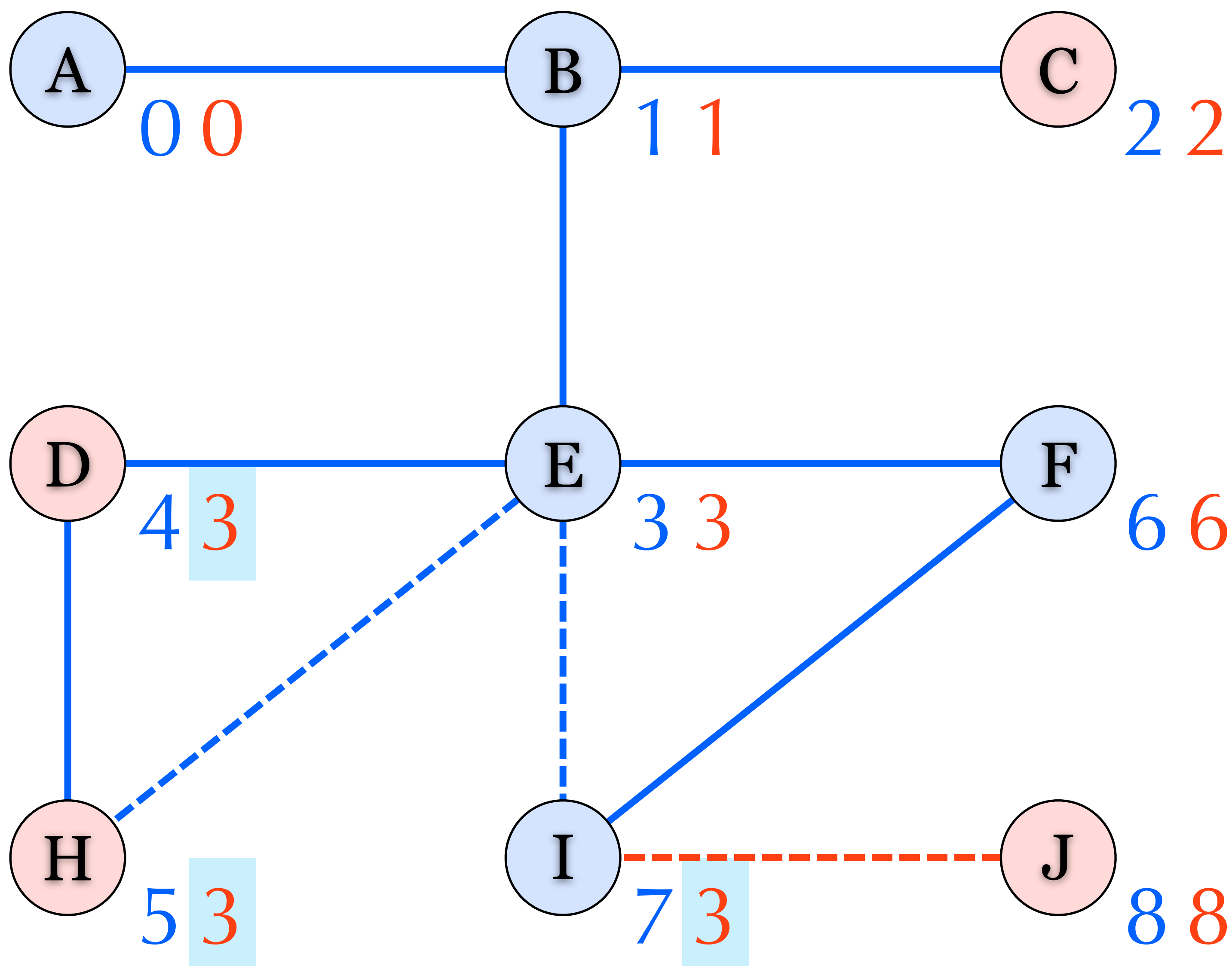
DFS order

Compute
Low value

DFS tree

DFS Based Algorithm



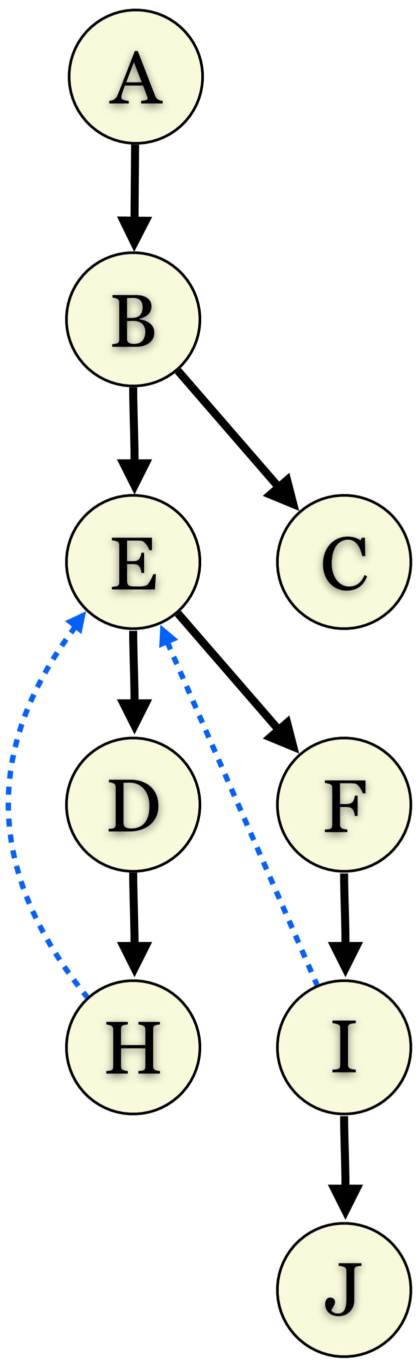


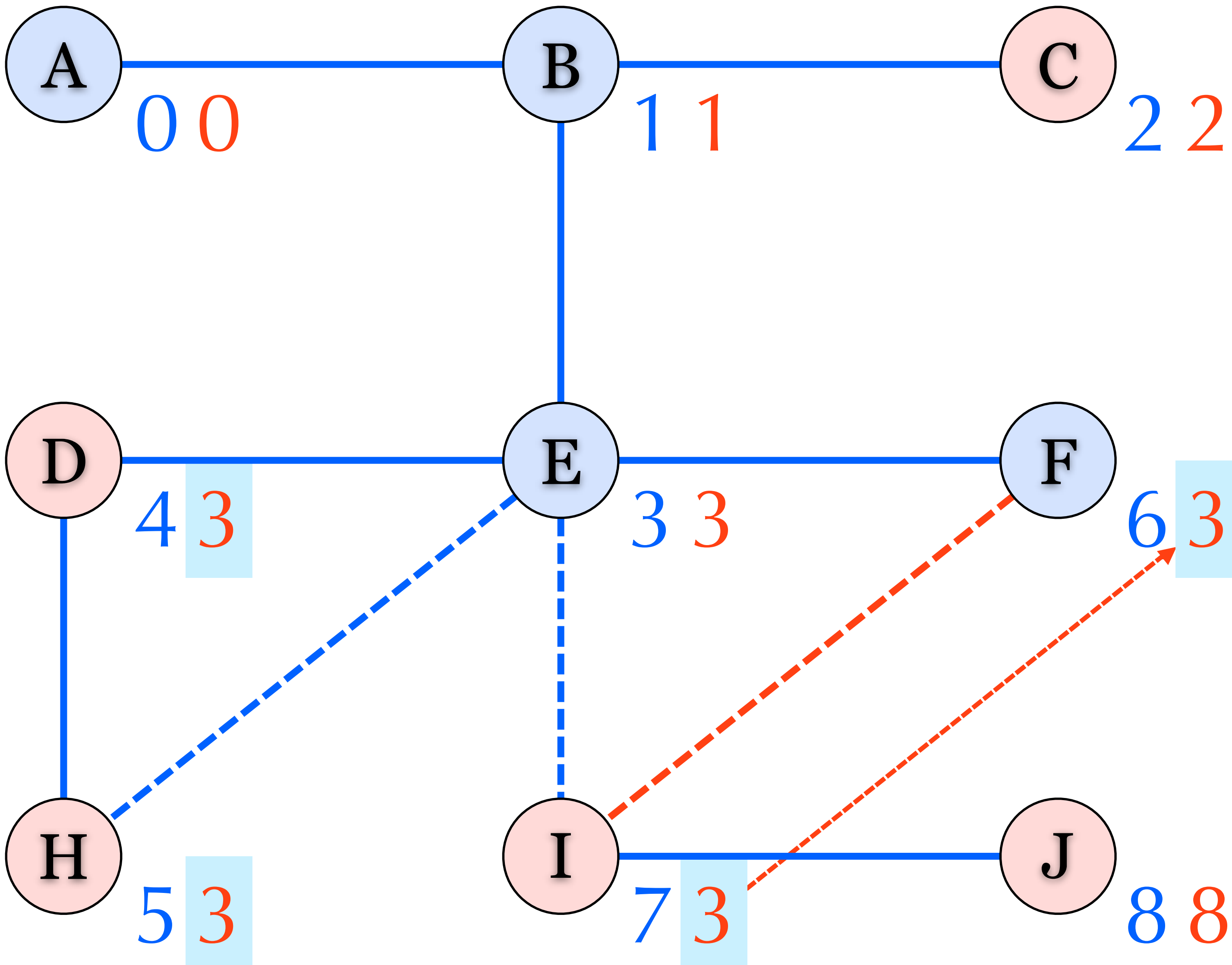
DFS order

Compute
Low value

DFS Based Algorithm

DFS tree



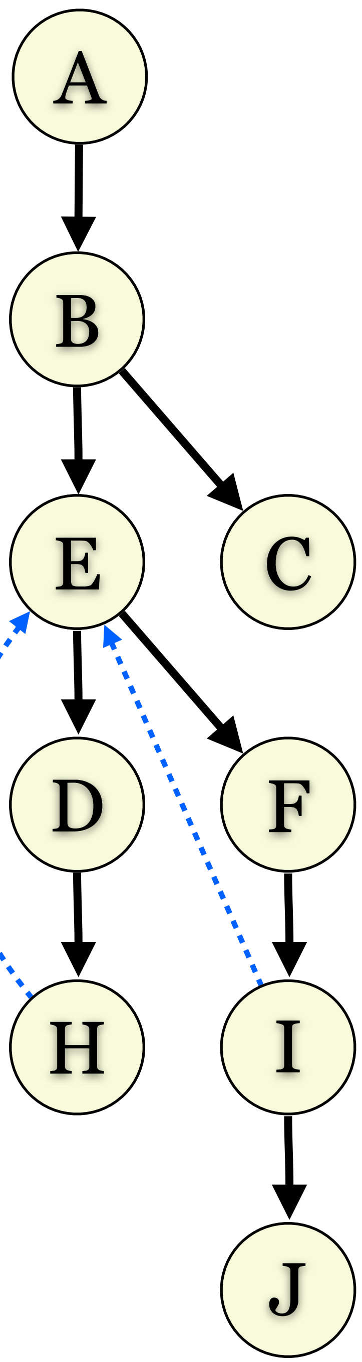


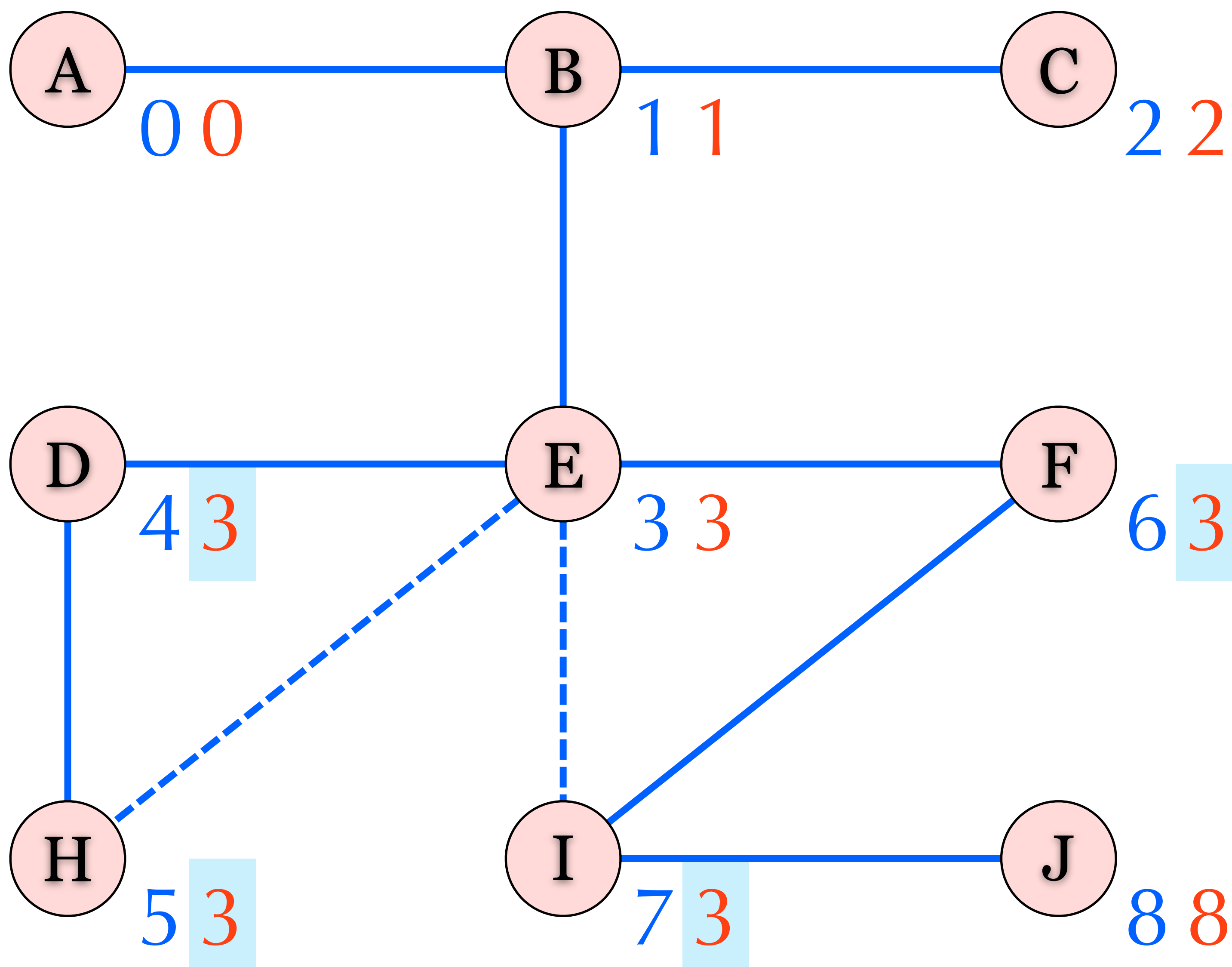
DFS order

Compute
Low value

DFS Based Algorithm

DFS tree





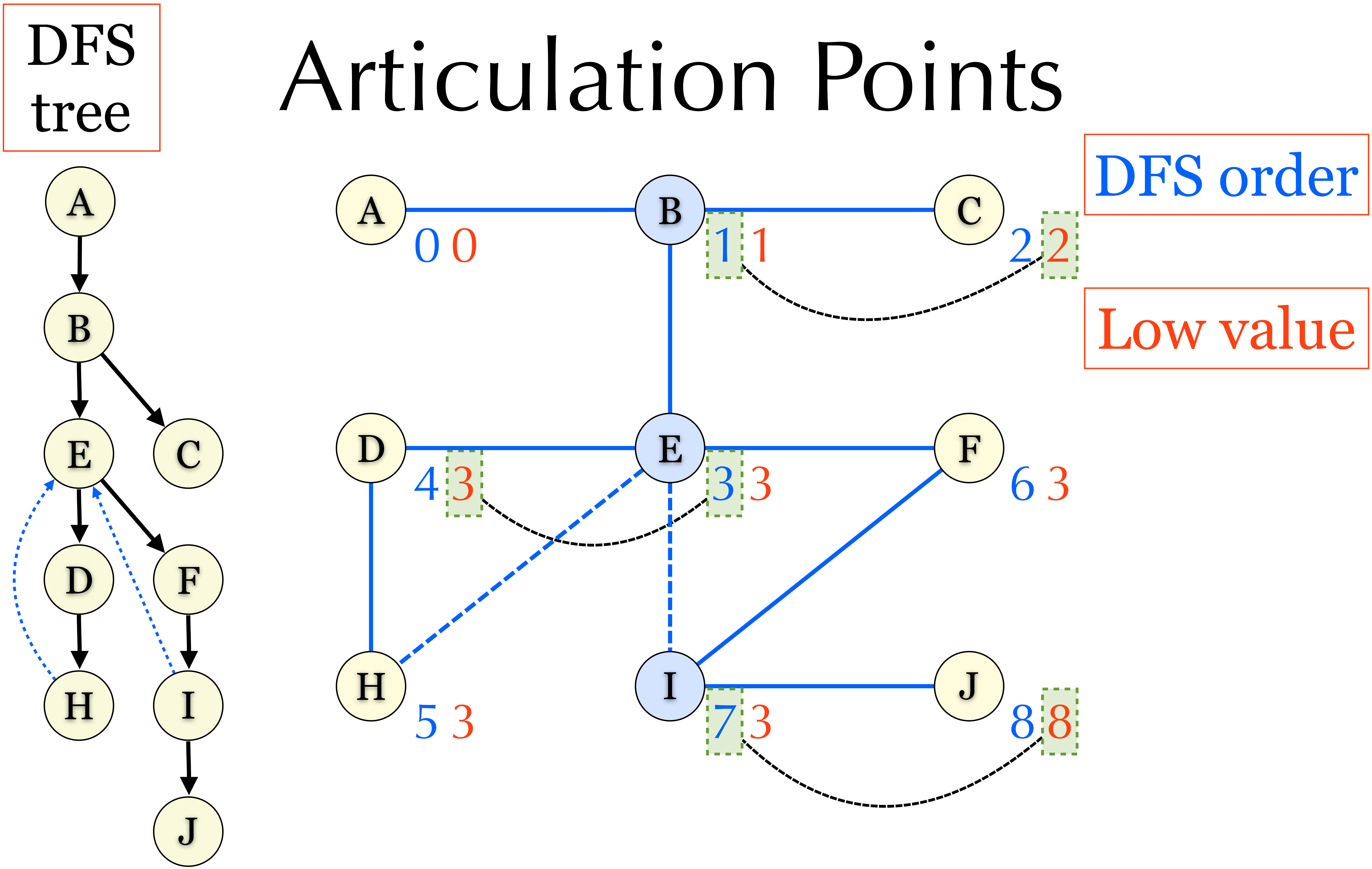
DFS order

Compute Low value

Determine Articulation Points

- If u is root
  - u is an articulation point if and only if u has more than one child in DFS tree.
- If u is not root
  - u is an articulation point if and only if there exist an edge (u,v) and the DFS order of u is at most the low value of v.

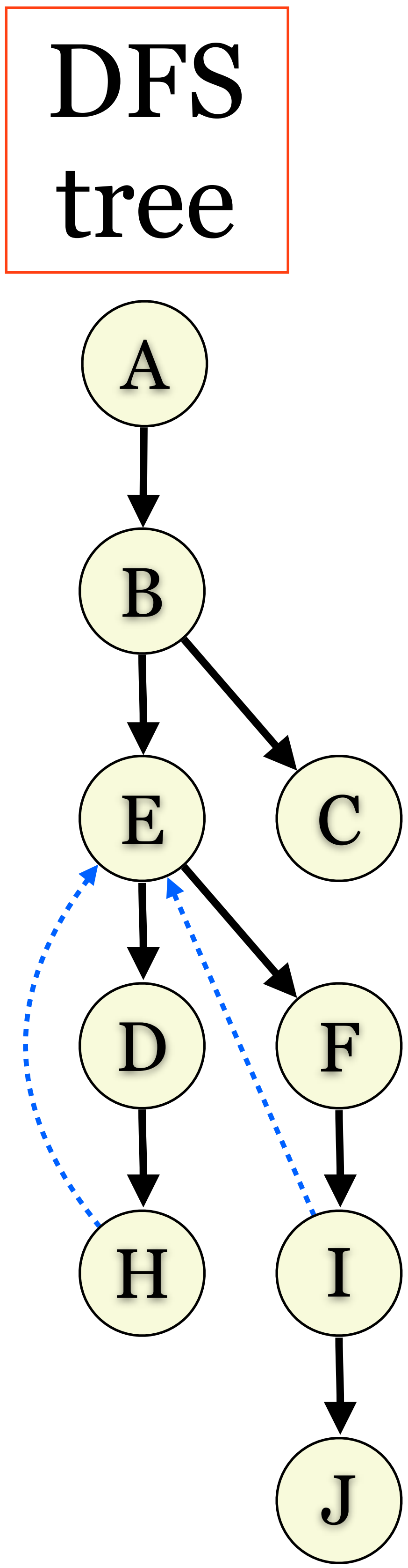
Articulation Points

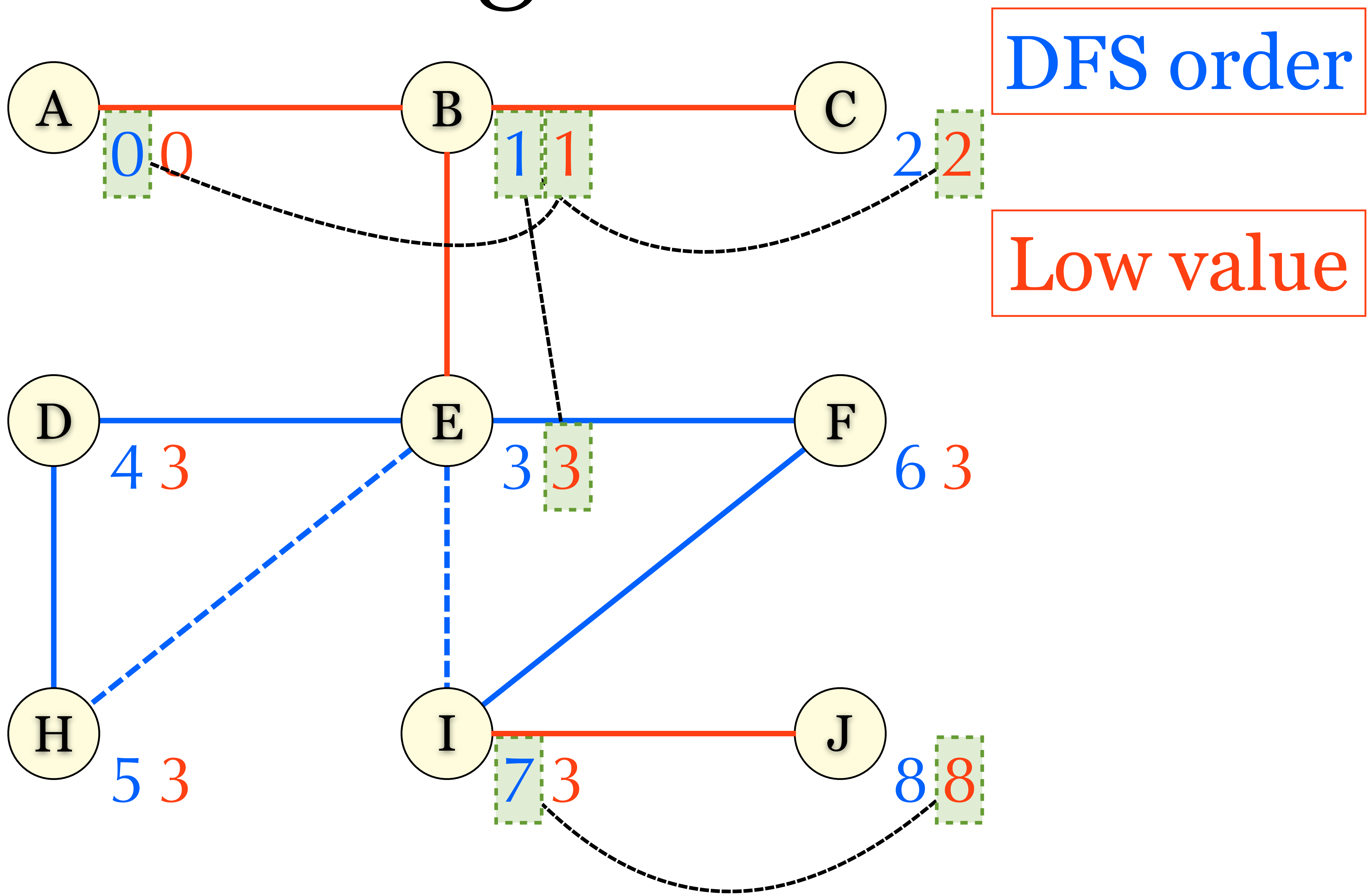


Determine Bridges

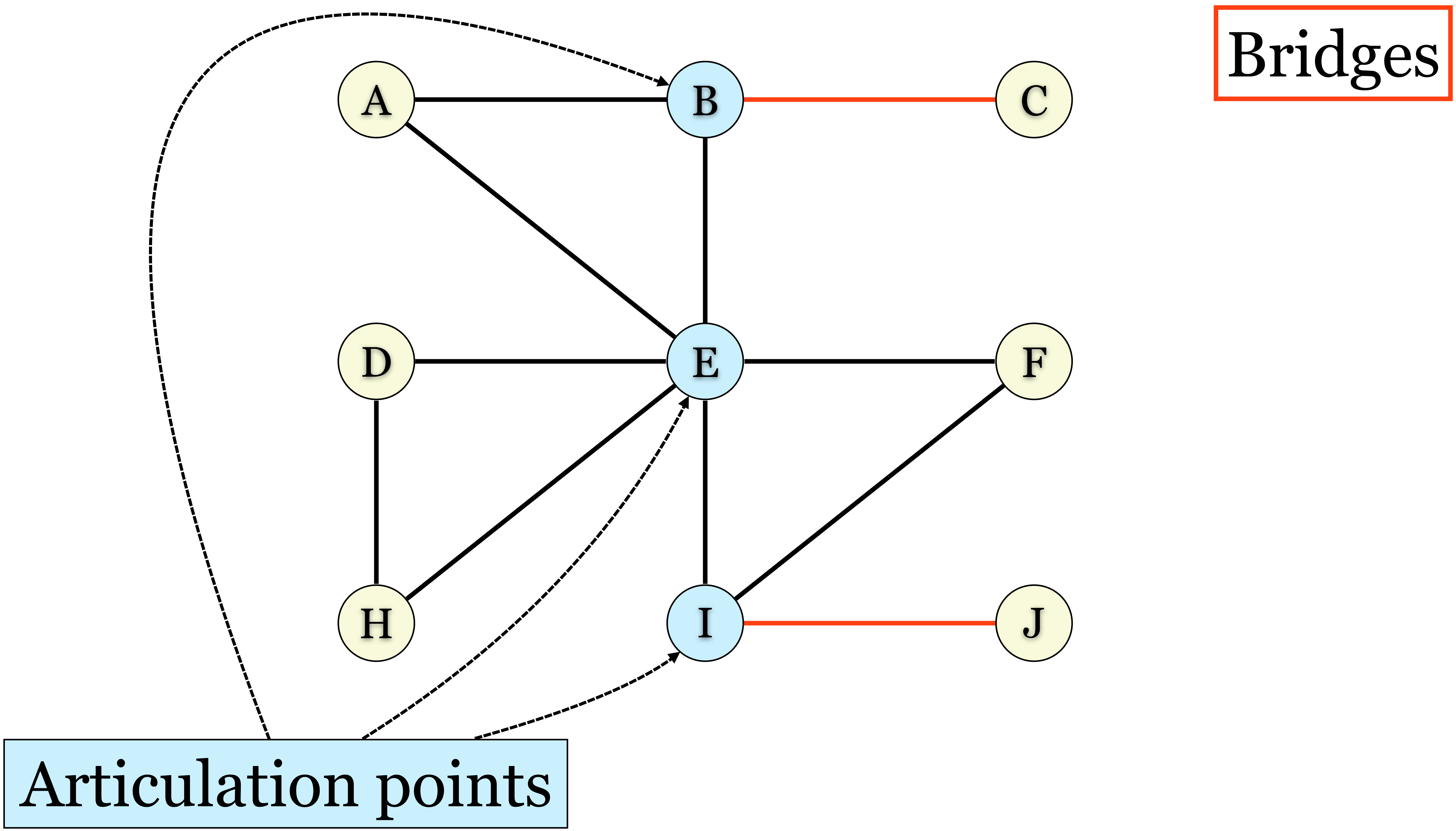
- (u,v) is a bridge if and only if the DFS order of u is less than the low value of v.

Bridges


DFS
tree
A
B
C
E
D
F
H
I
J


A
B
C
D
E
F
H
I
J
0 0
1 1
2 2
4 3
3 3
6 3
5 3
7 3
8 8
DFS order
Low value

Example 2



DFS Based Algorithm