Lecture 15: Applications of DFS: SCC

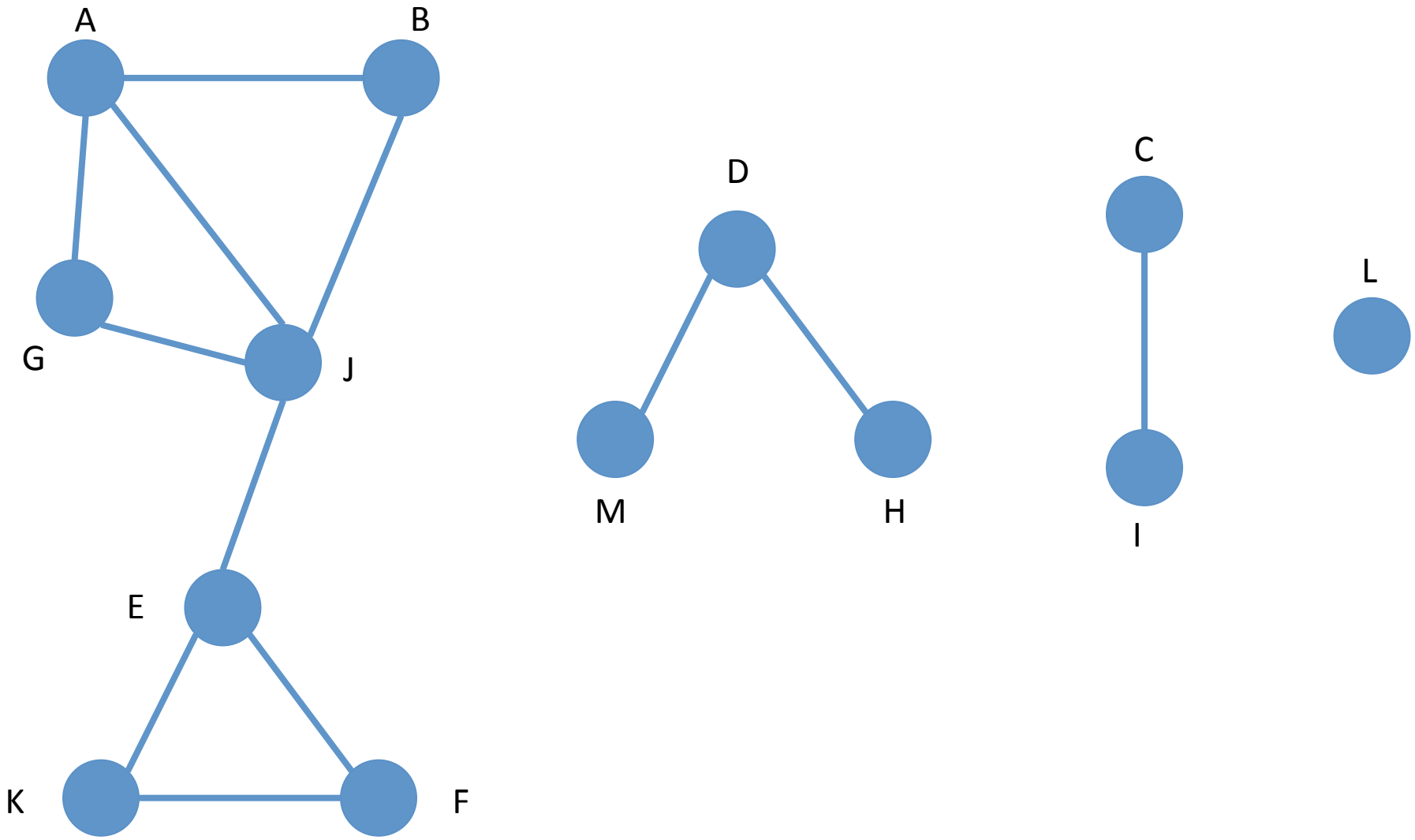
CS 341: Algorithms

Thursday, March 7th 2019

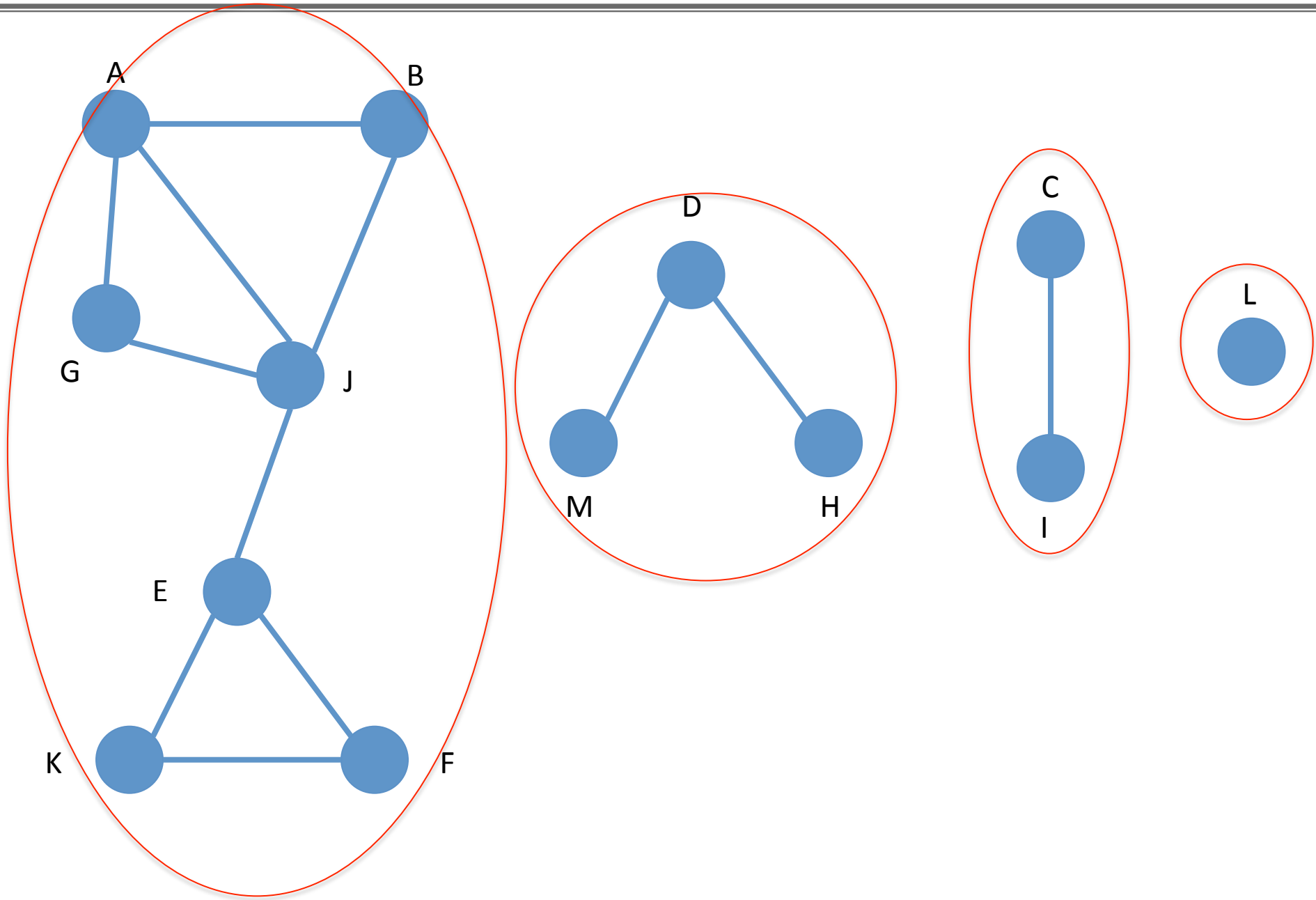
Outline For Today

1. Strongly Connected Components in Directed Graphs

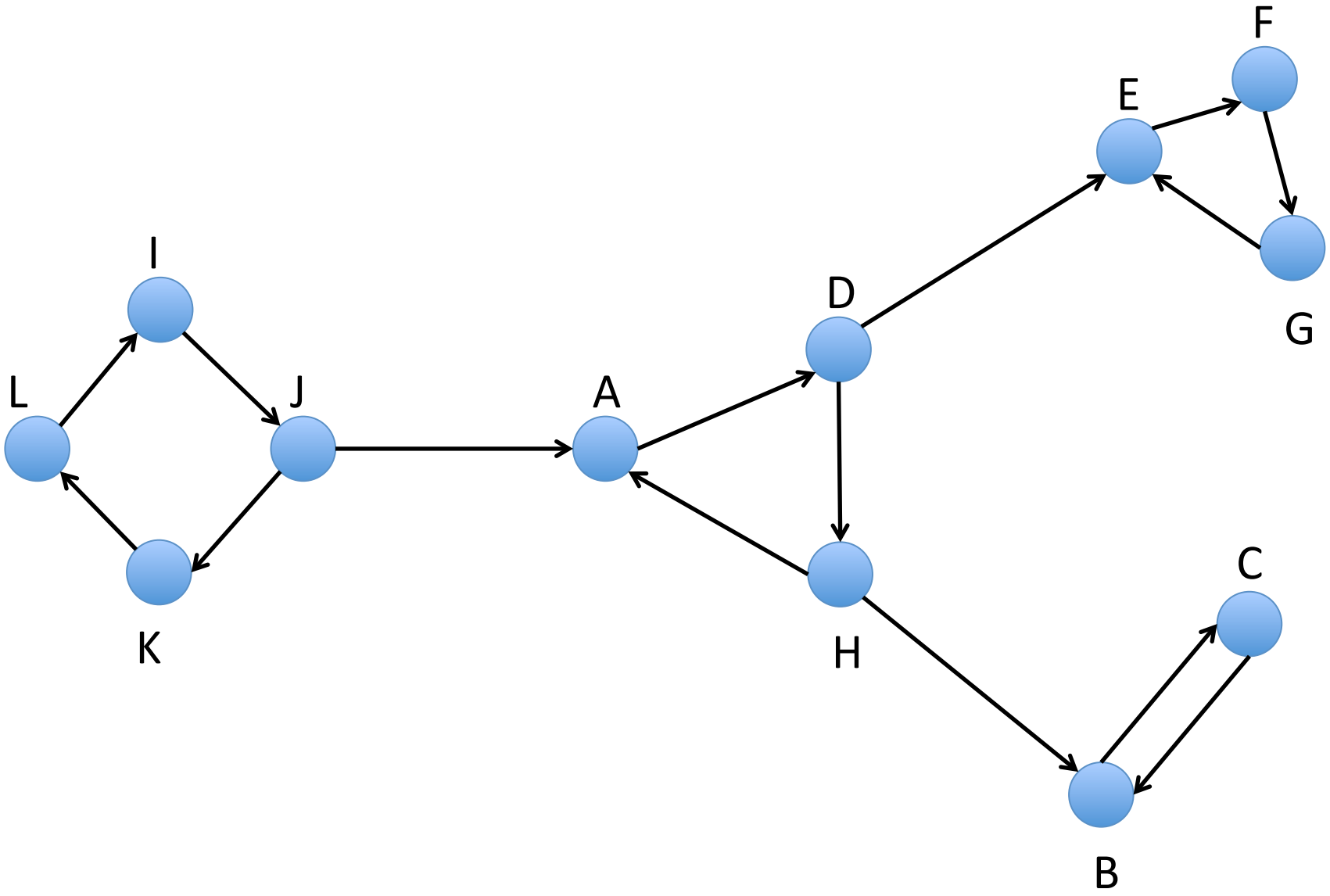
Recall: Connectivity in Undirected Graphs



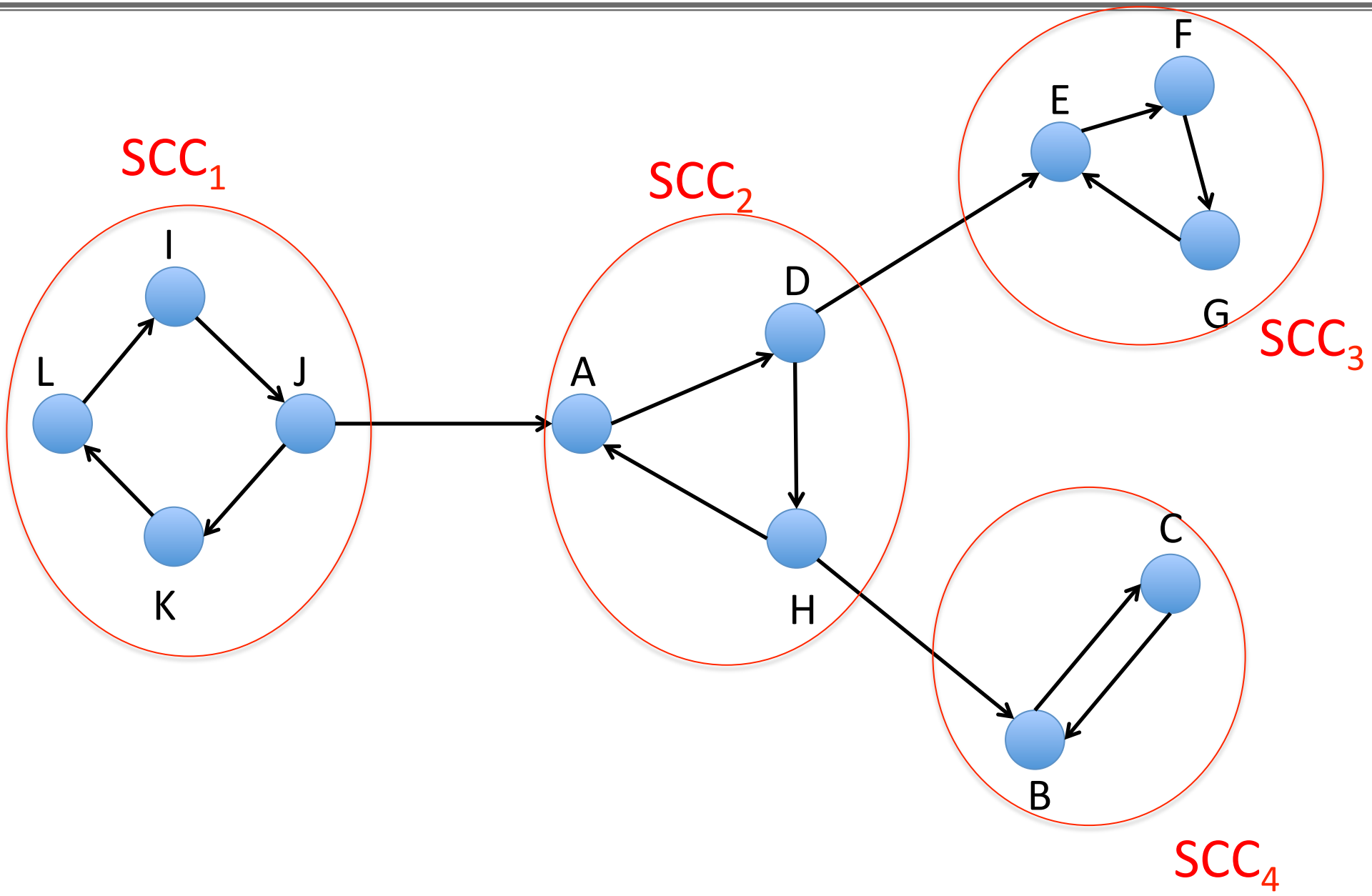
Recall: Connectivity in Undirected Graphs



Connectivity in Directed Graphs



“Strong Connectivity” in Directed Graphs



Formal Definition of SCCs

- Given a directed graph G(V, E)
- u and v are strongly connected iff:

  $u \rightsquigarrow v$ *AND* $v \rightsquigarrow u$

- *An SCC of G is a set* $C \subseteq V$ *s.t*

  1. $\forall (u, v) \in C$, *u and v are strongly connected*
  2. *C is non-empty and maximal, i.e.* $\forall v`$ *in V-C,*

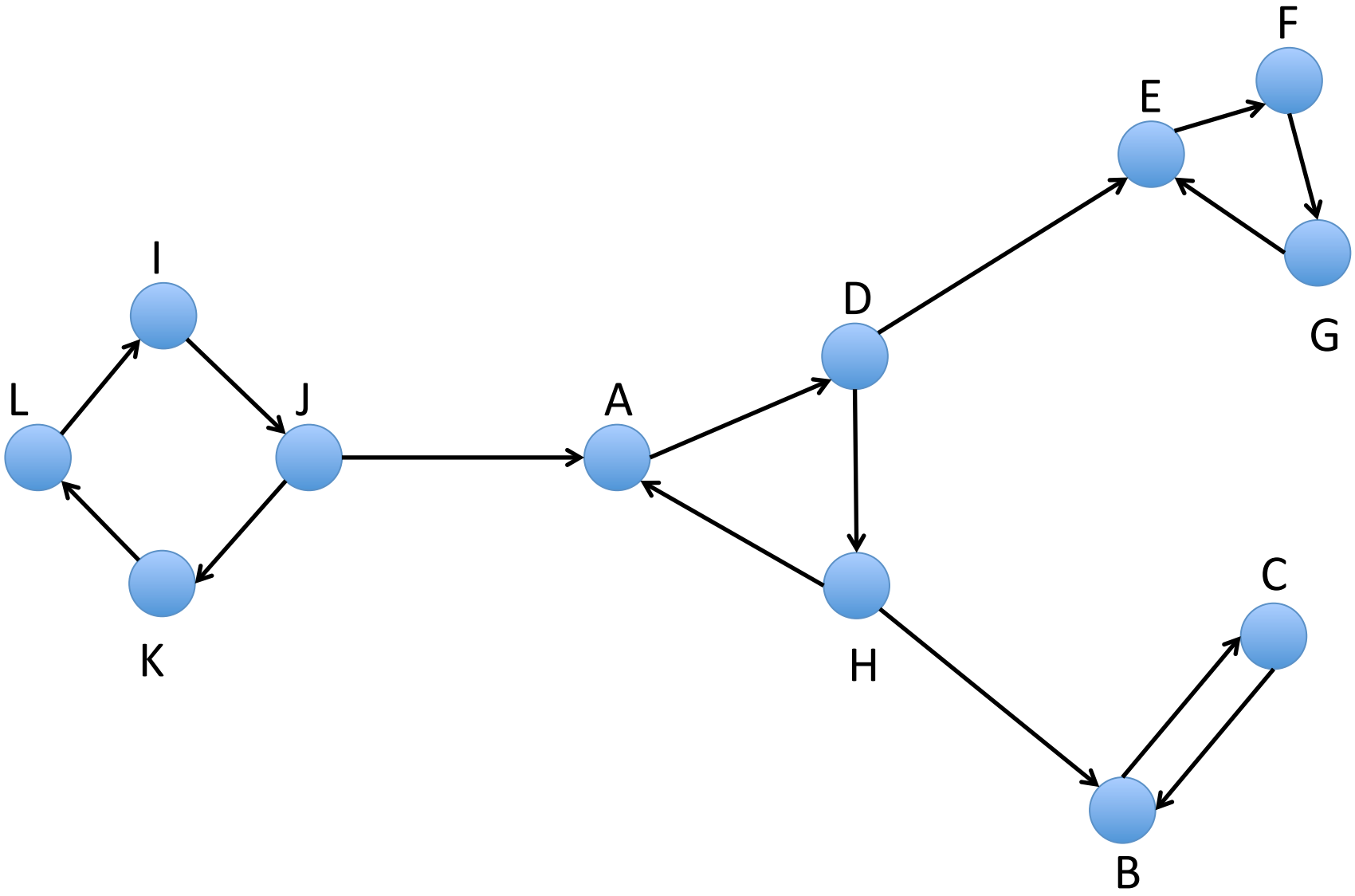
     v` is not strongly connected to any vertex in C

Two-tiered Structure

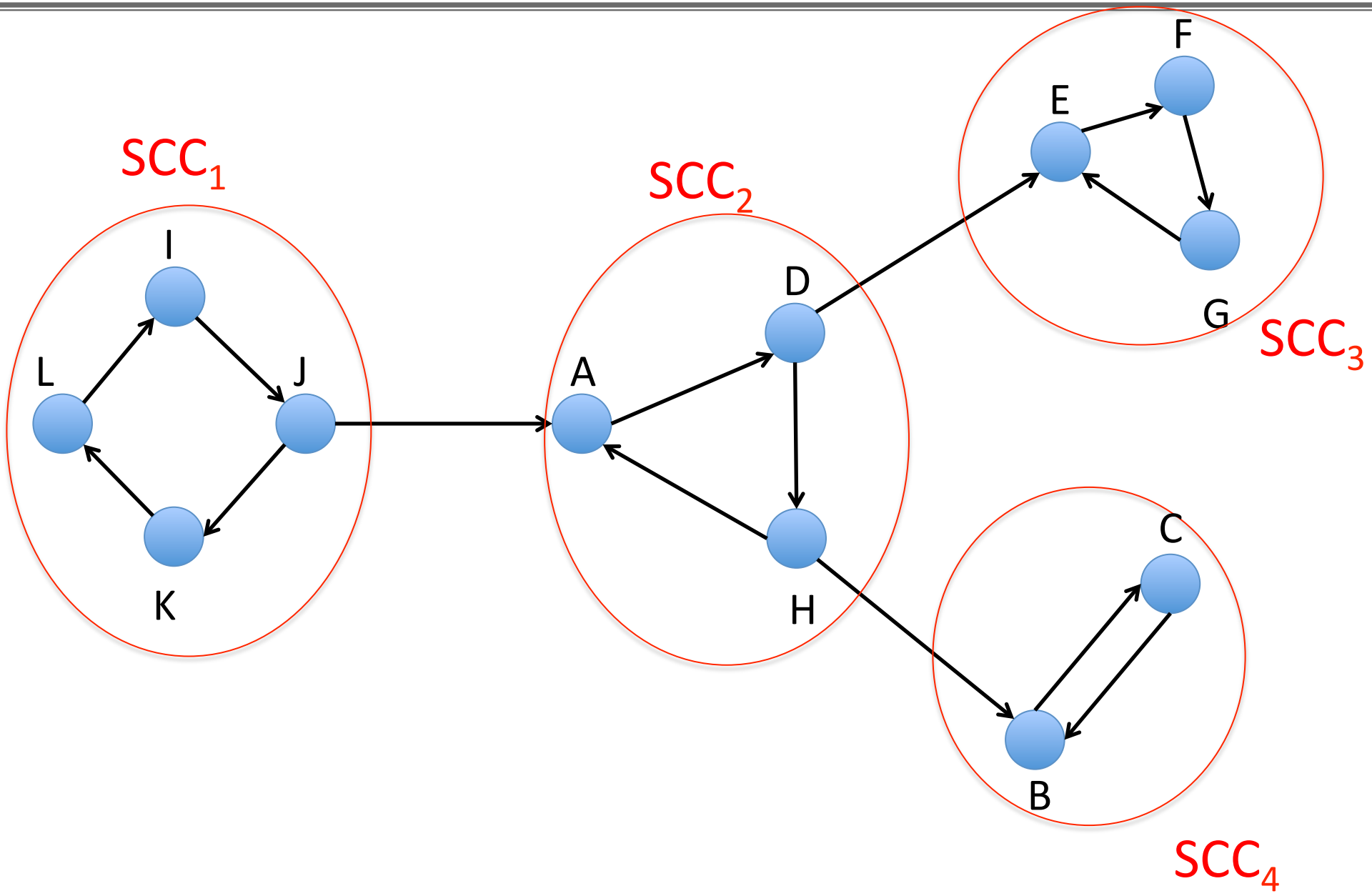
- Given a directed graph G(V, E)
- Consider G^{SCC}: the meta-graph of its SCCs
- We'll call this "SCC graph of G"
- *Claim: G^{SCC} is a DAG!*

Each directed graph has a second higher-level structure which is a DAG.

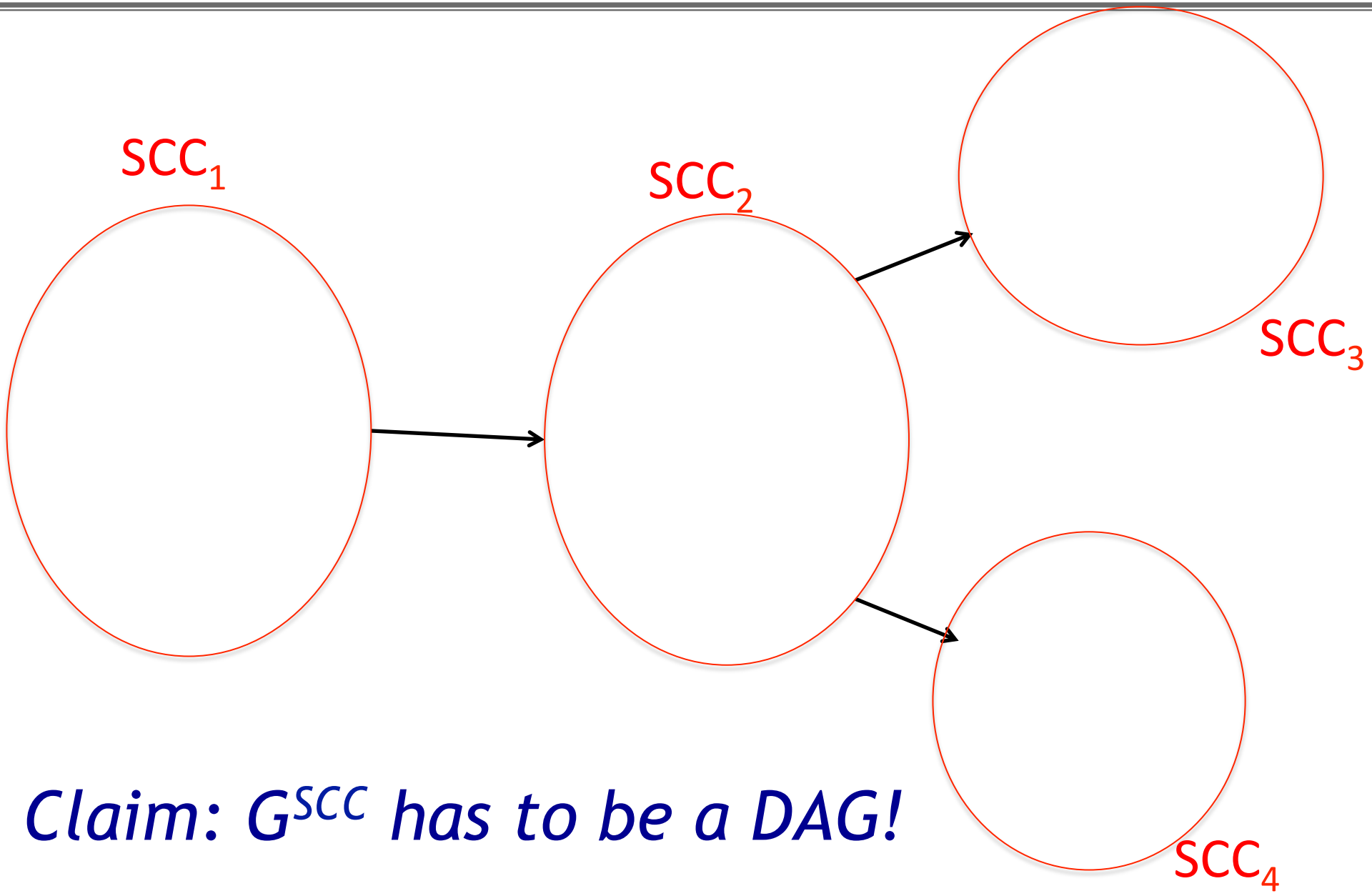
Example of Two-tiered Structure



Example of Two-tiered Structure

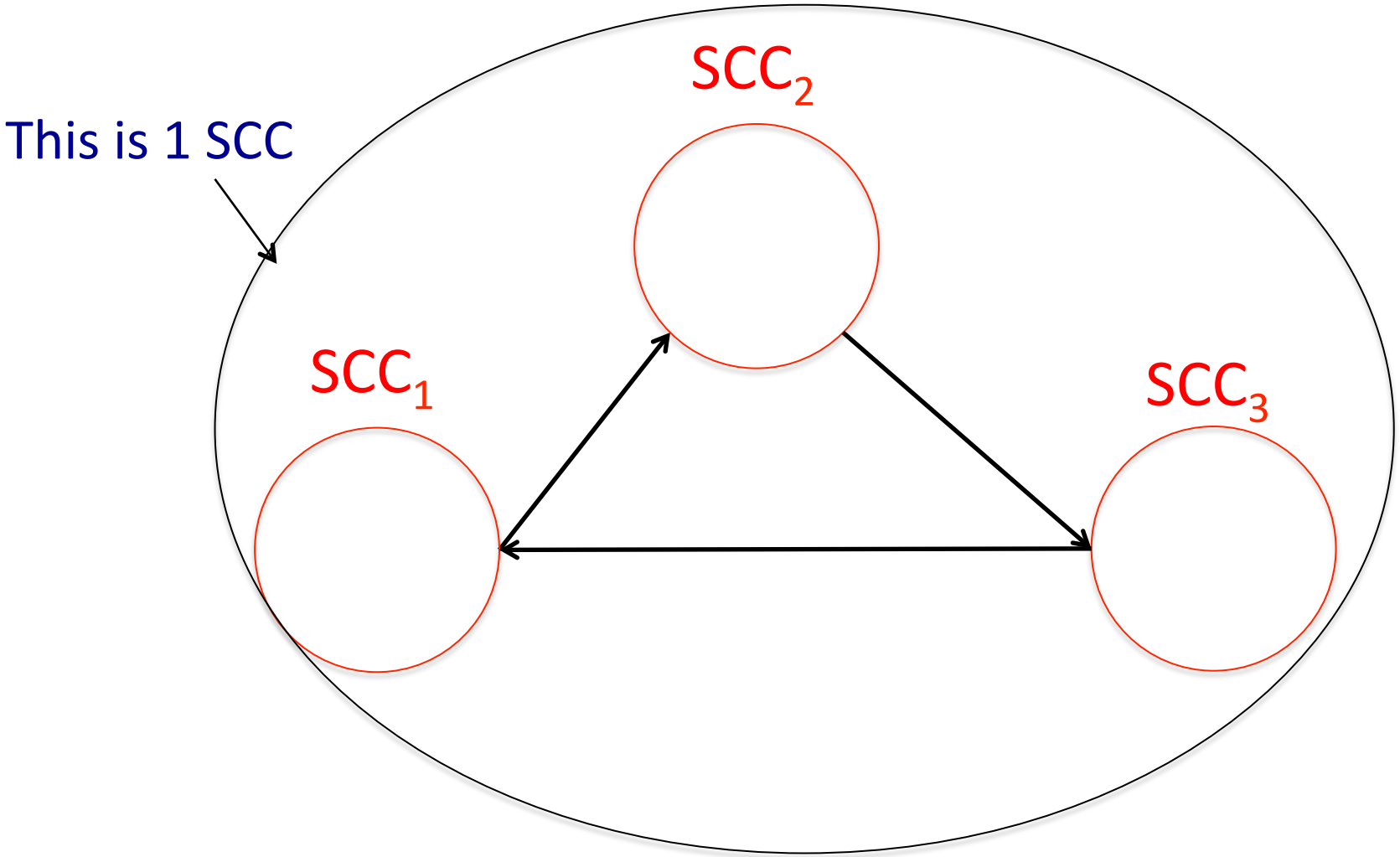


Example of Two-tiered Structure



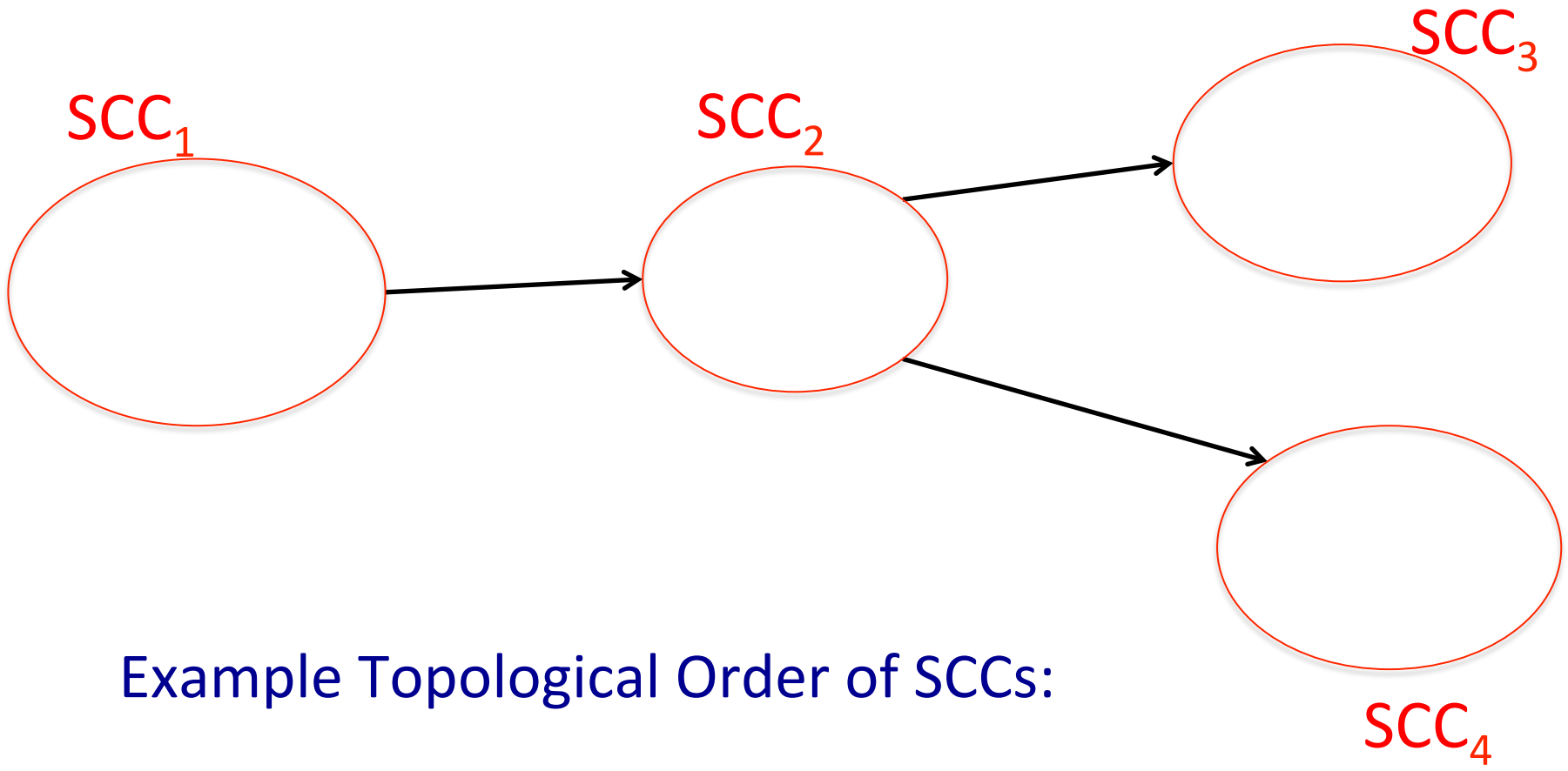
Claim: G^{SCC} has to be a DAG!

Proof Sketch of G^{SCC} being a DAG:



Therefore, can't have cycles in G^{SCC}

So We Can Topologically Order G^{SCC}

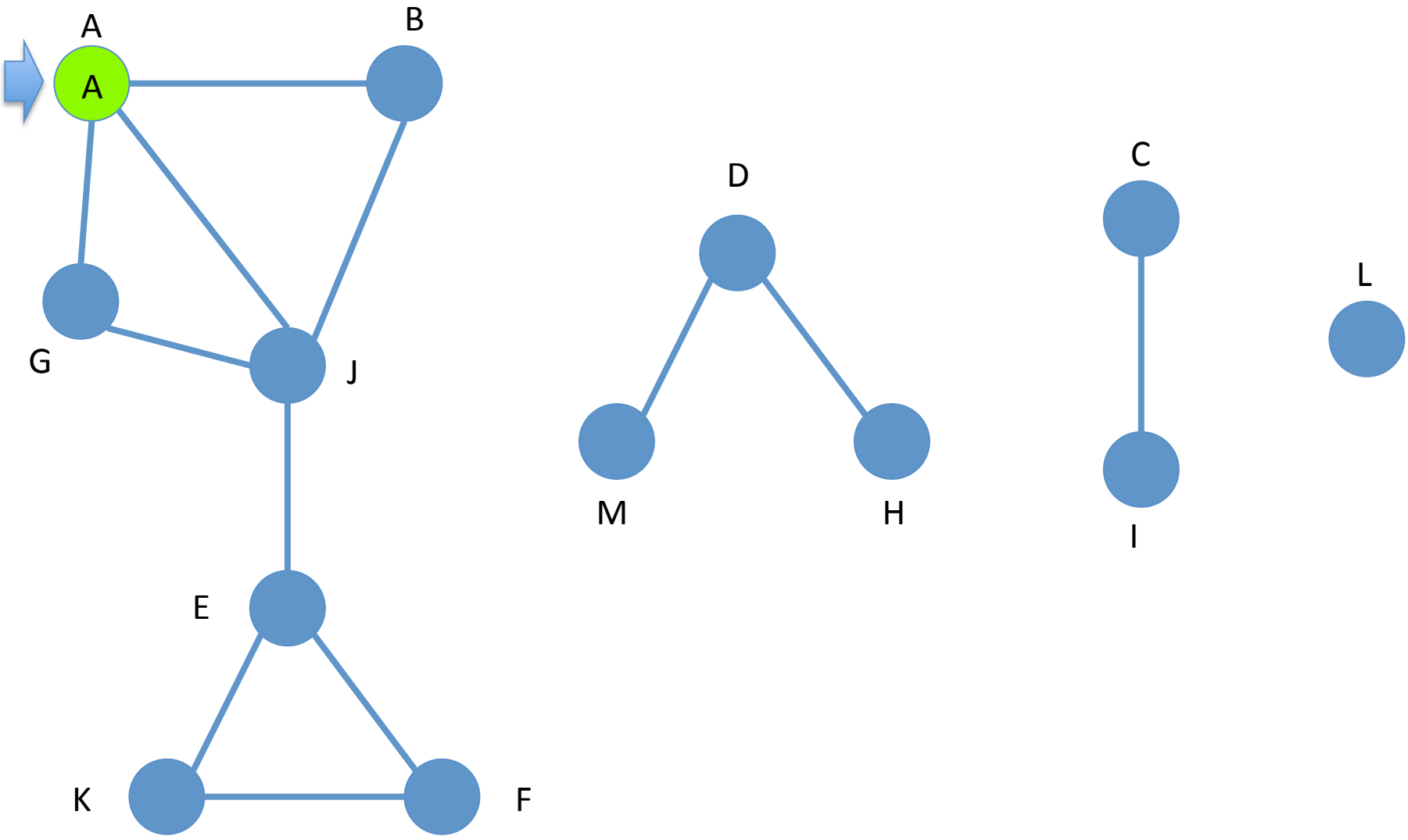


Example Topological Order of SCCs:

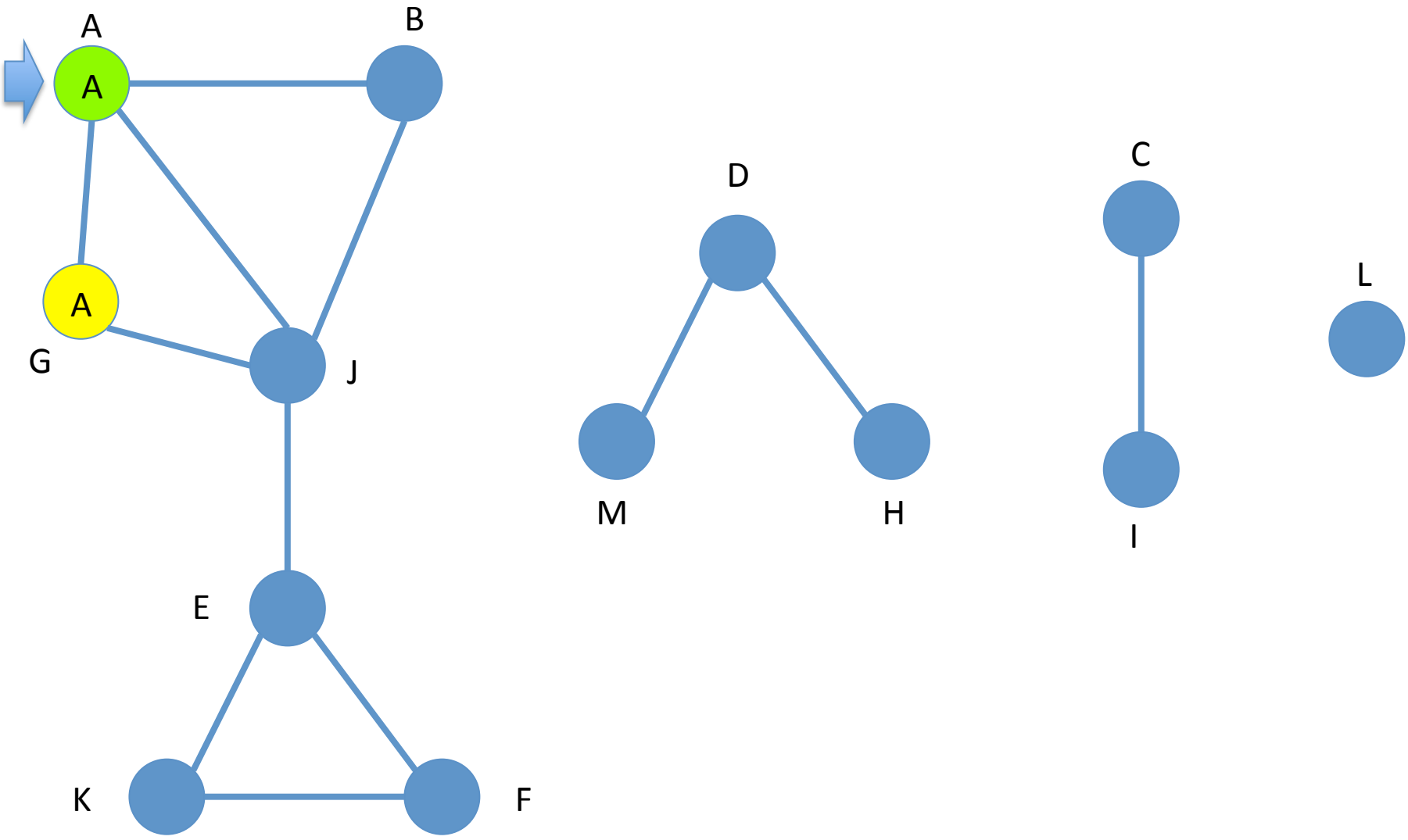
SCC_1 SCC_2 SCC_4 SCC_3

Will be important for Kosaraju's Algorithm!

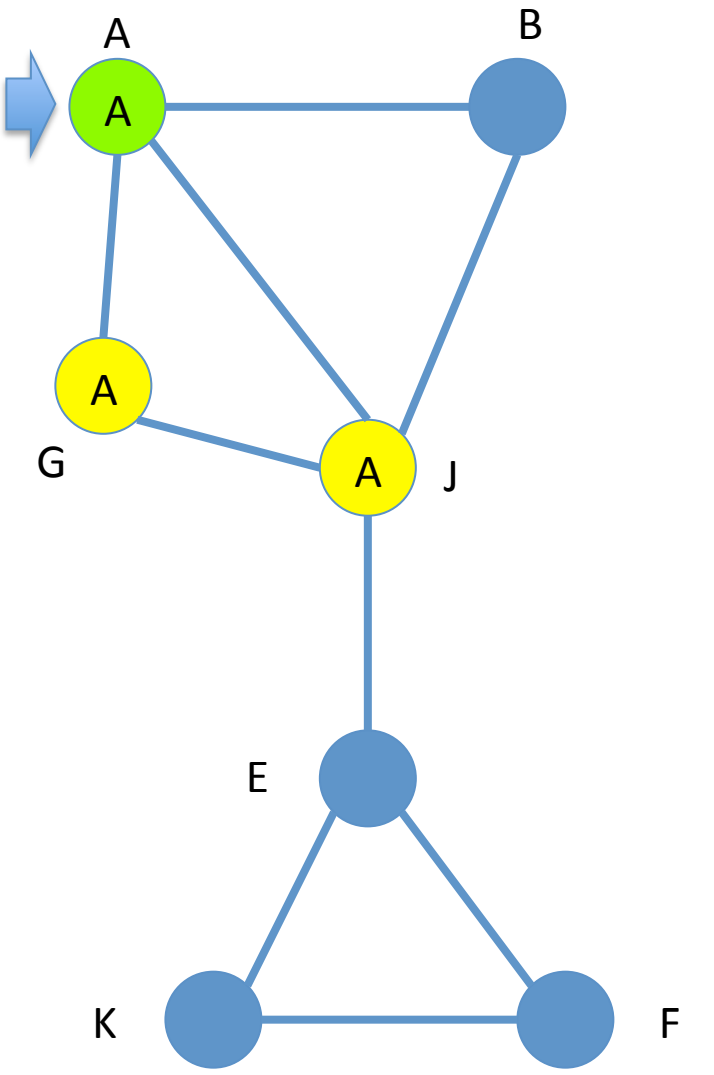
Recall Algorithm for Undirected Graphs

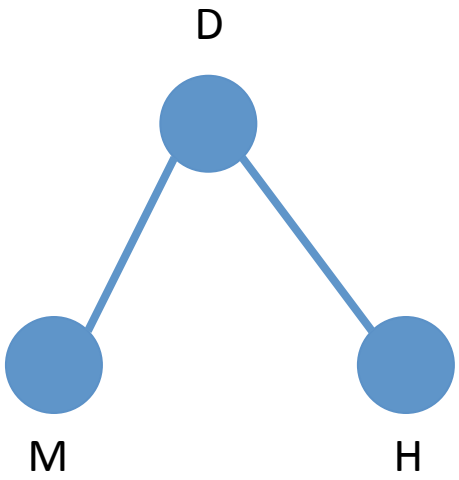


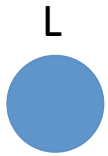
Recall Algorithm for Undirected Graphs



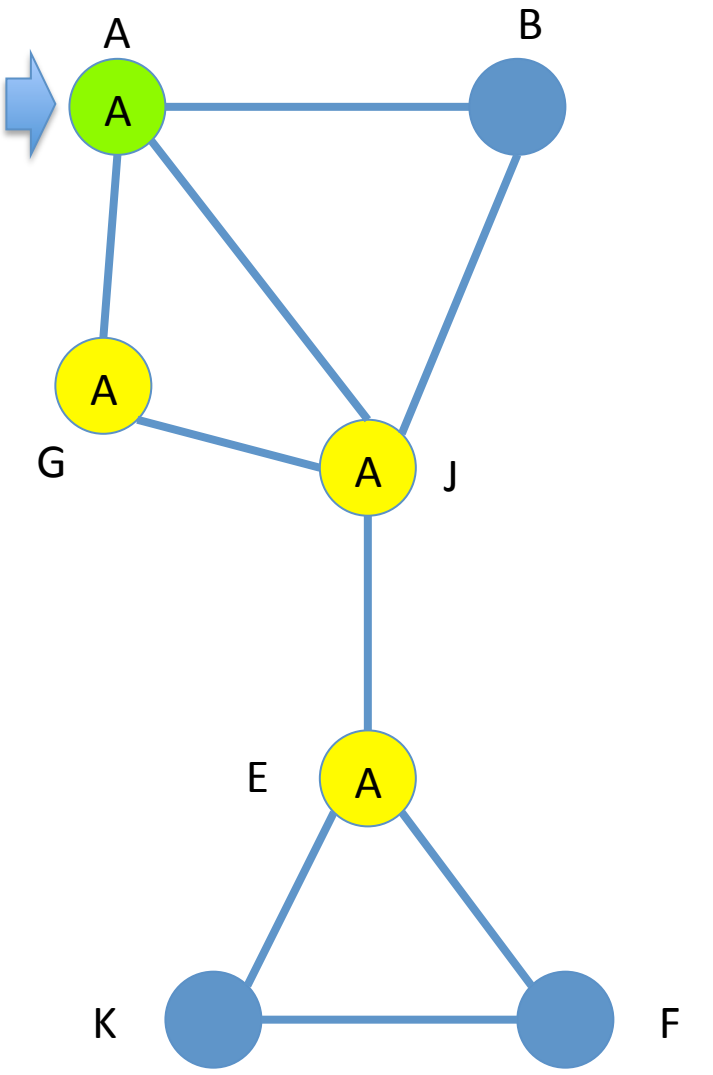
Recall Algorithm for Undirected Graphs

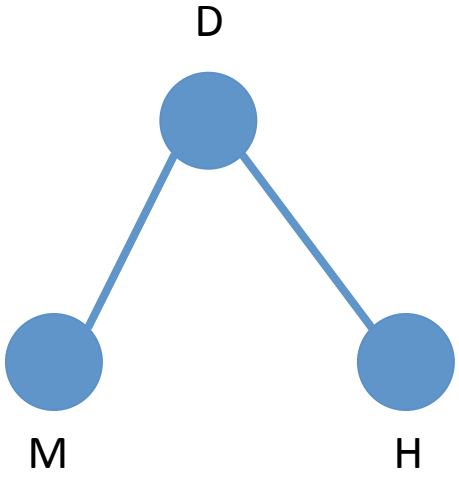


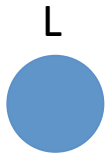




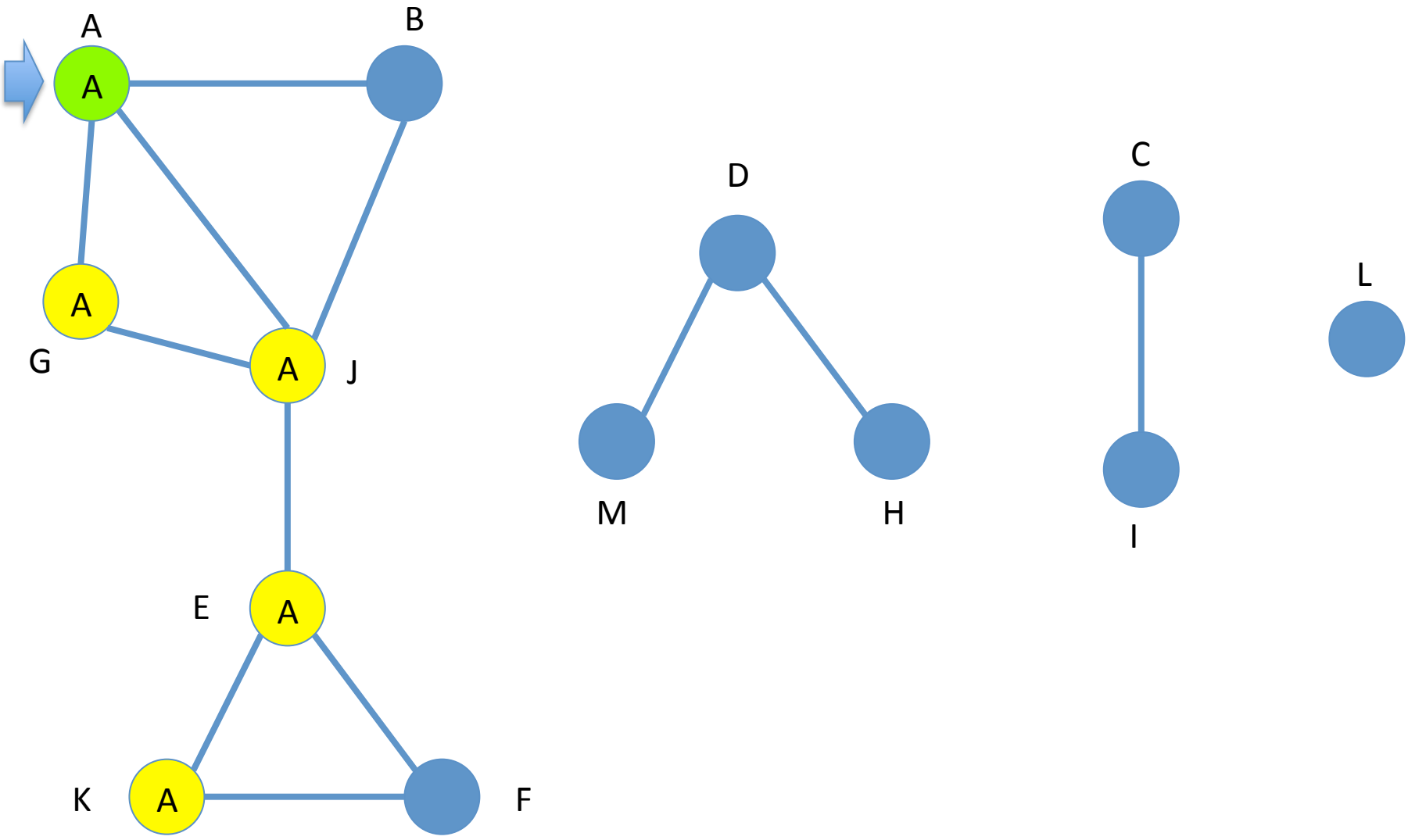
Recall Algorithm for Undirected Graphs



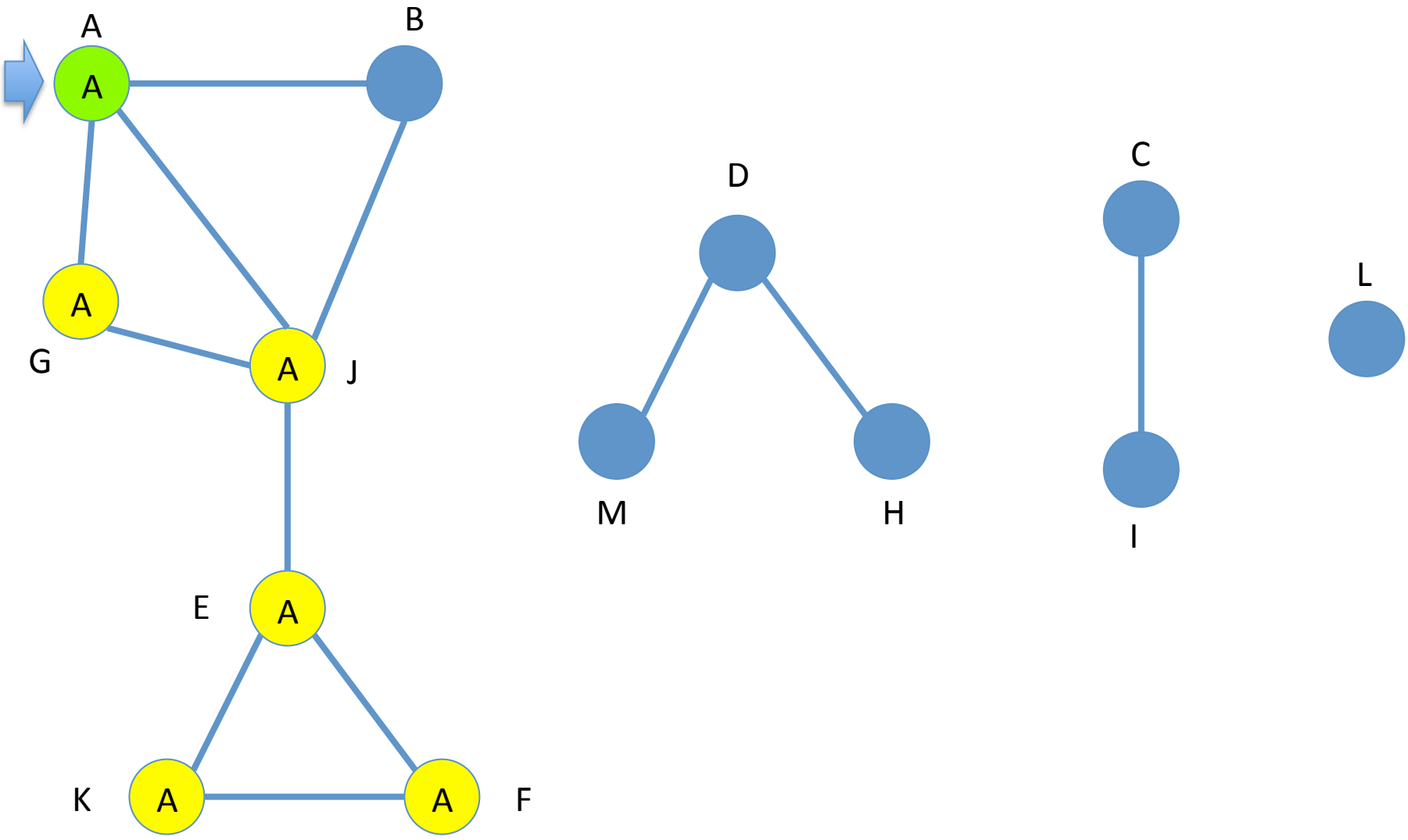




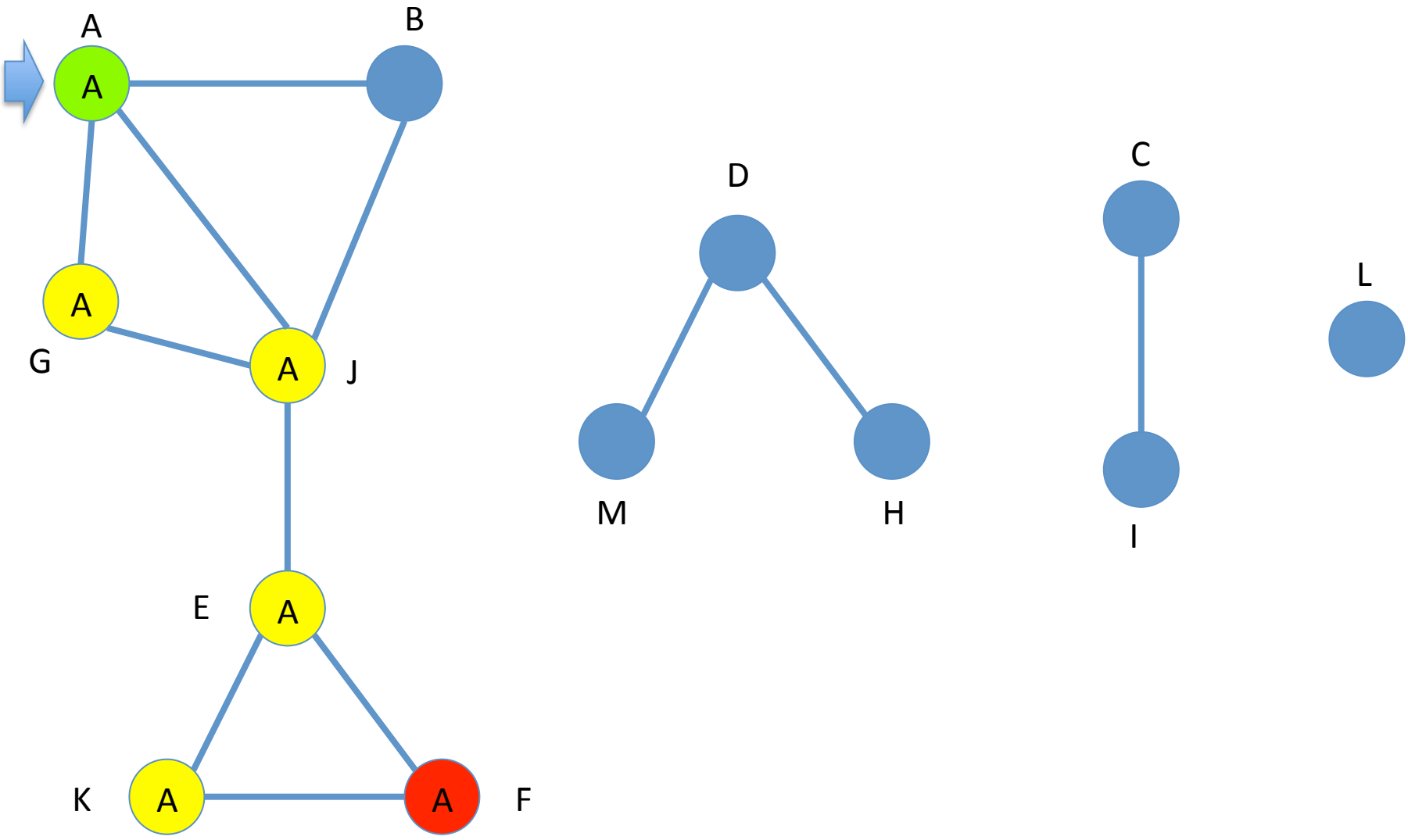
Recall Algorithm for Undirected Graphs



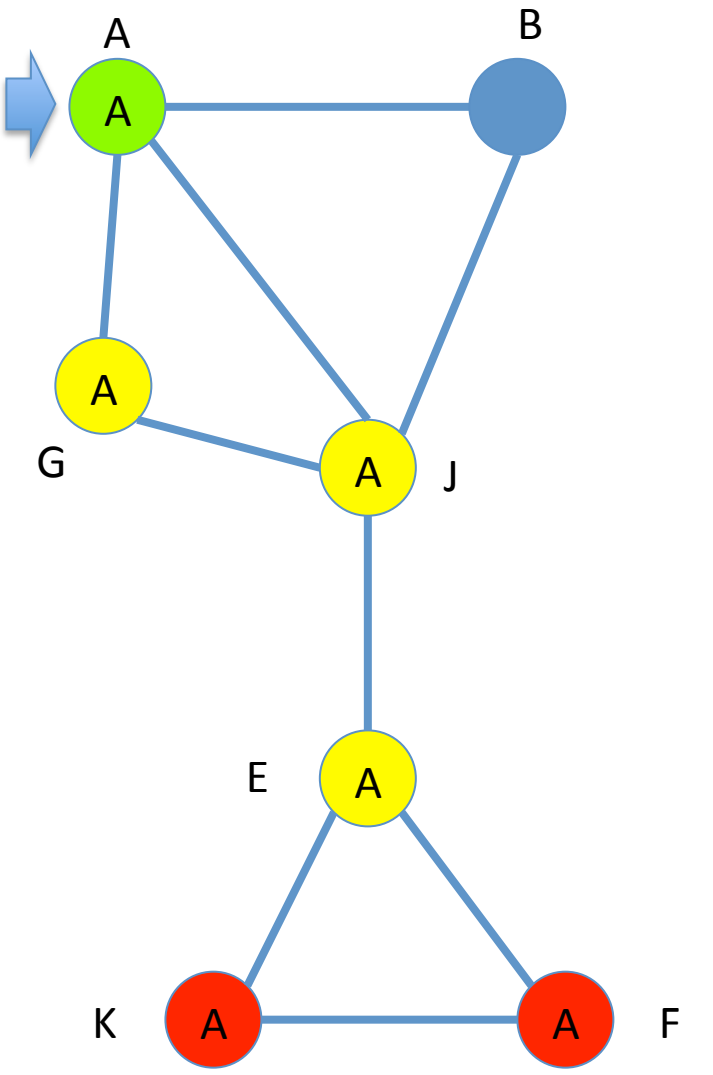
Recall Algorithm for Undirected Graphs

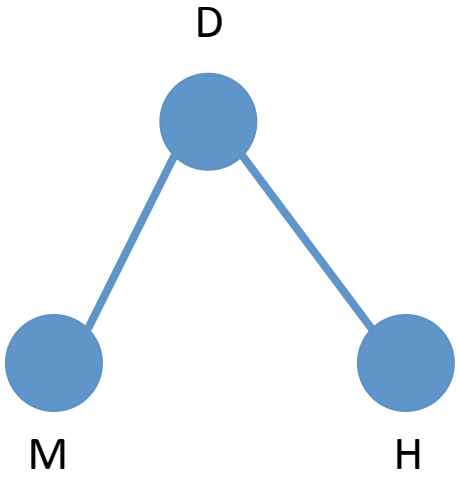


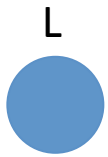
Recall Algorithm for Undirected Graphs



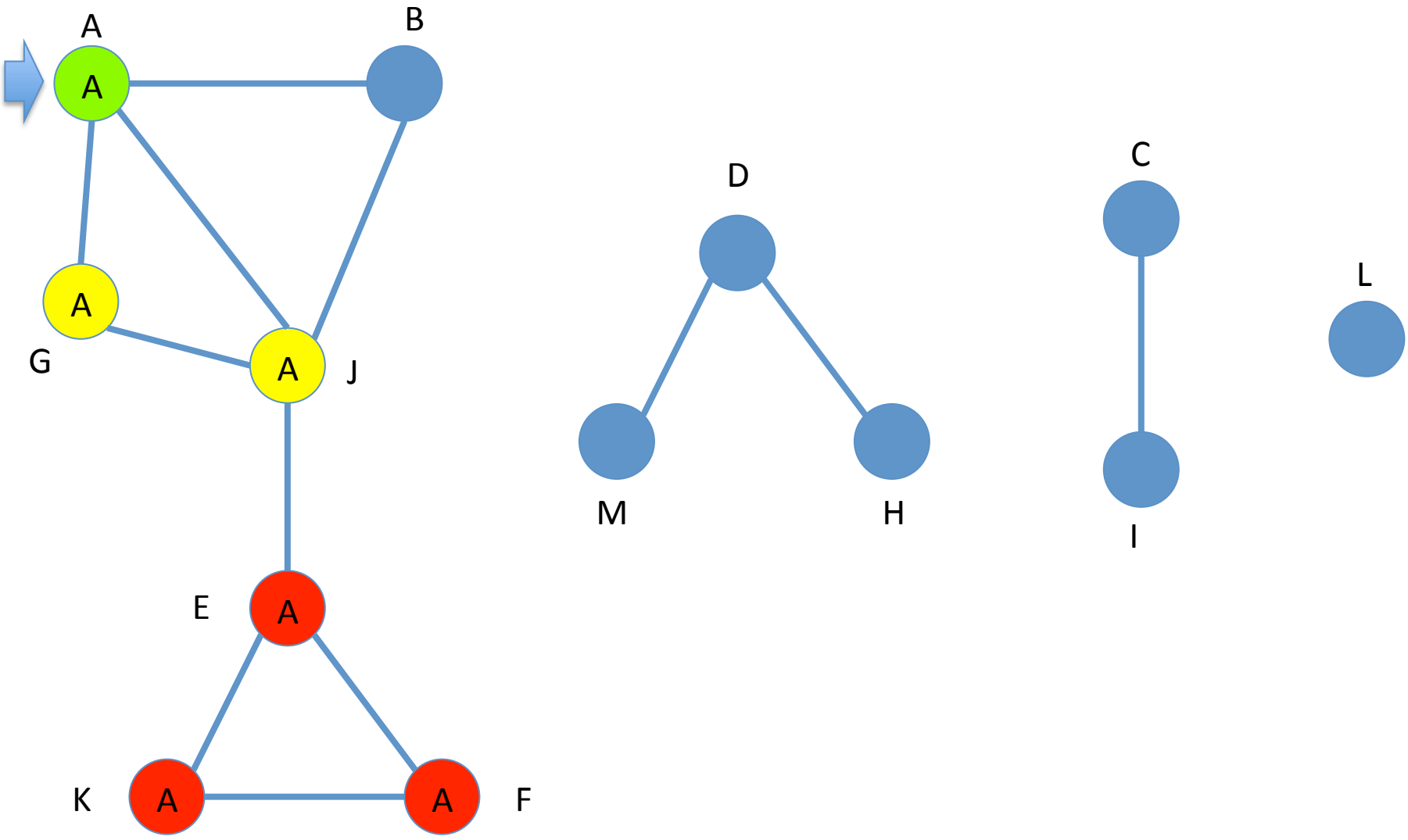
Recall Algorithm for Undirected Graphs



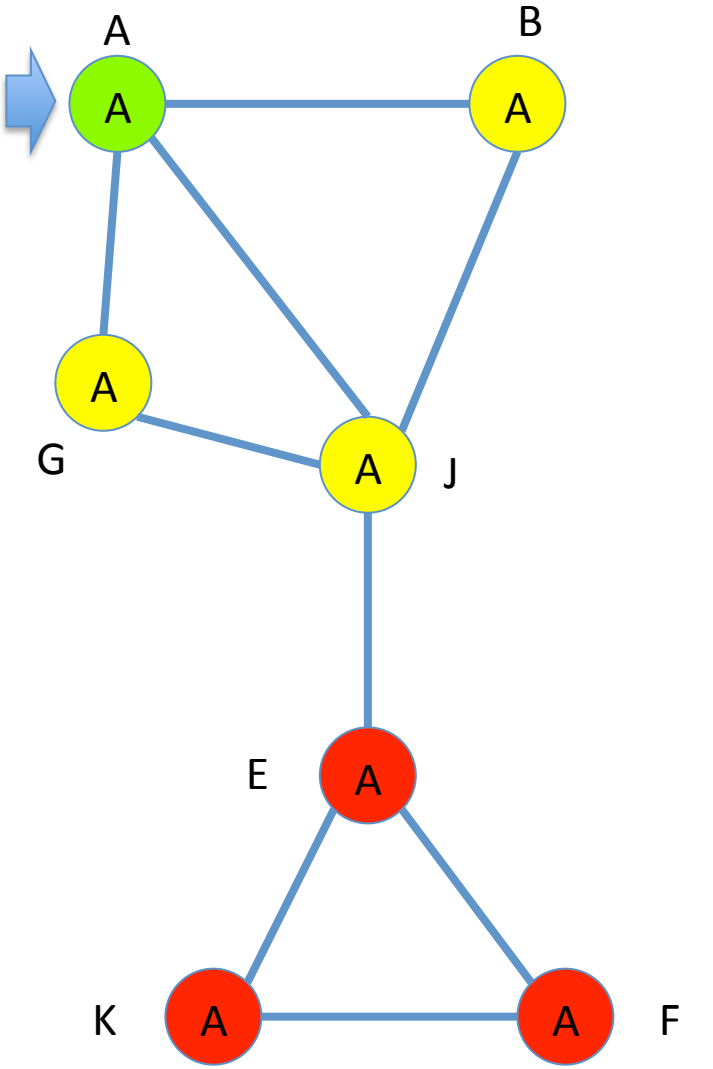


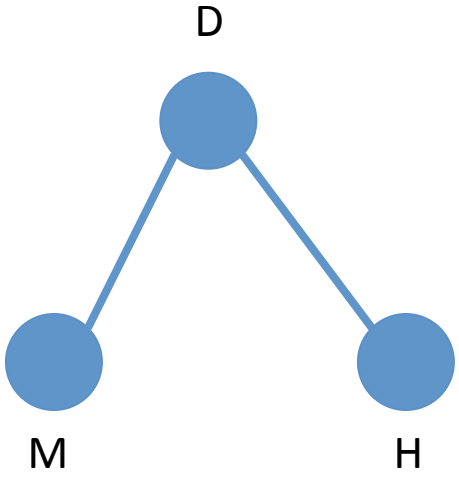


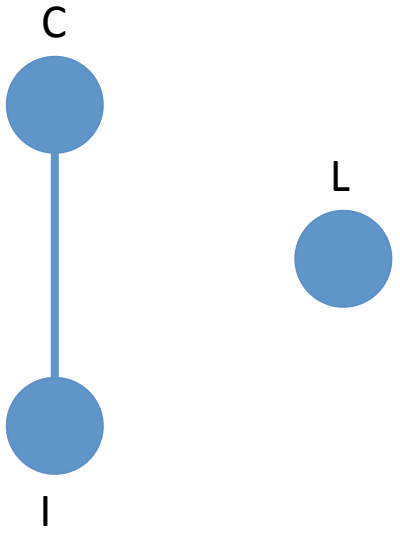
Recall Algorithm for Undirected Graphs



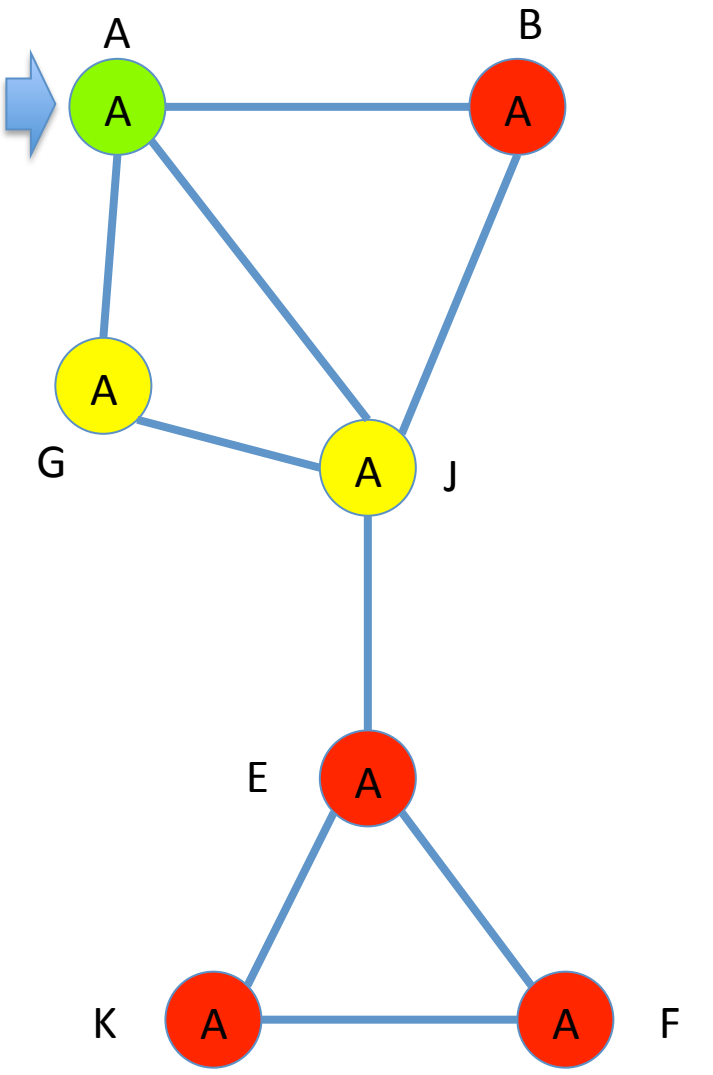
Recall Algorithm for Undirected Graphs

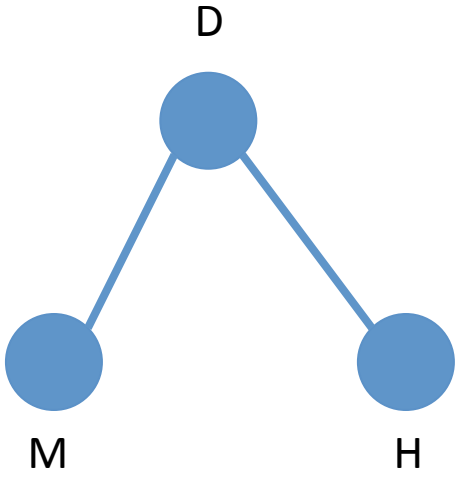


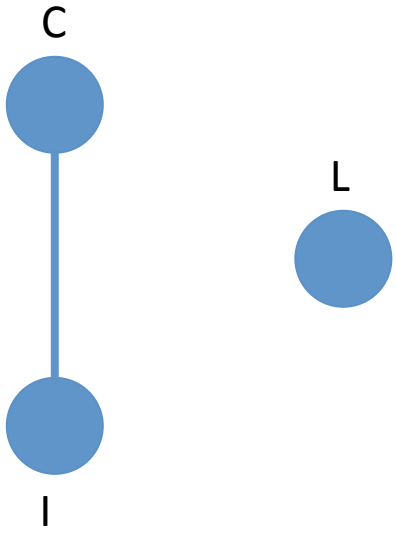




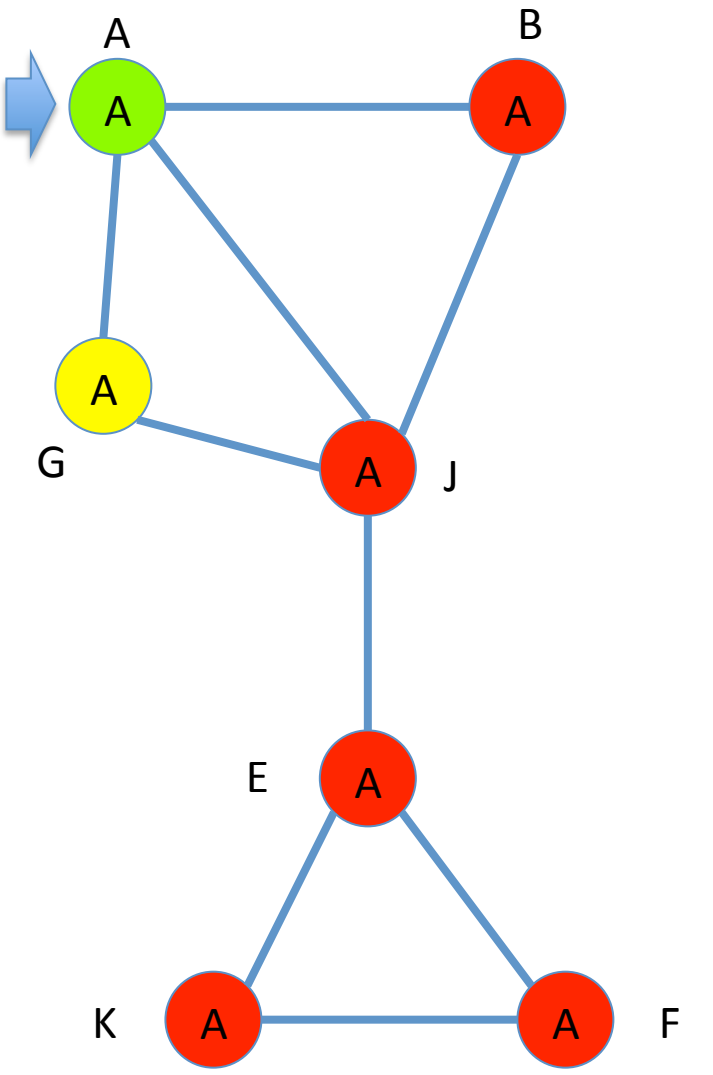
Recall Algorithm for Undirected Graphs

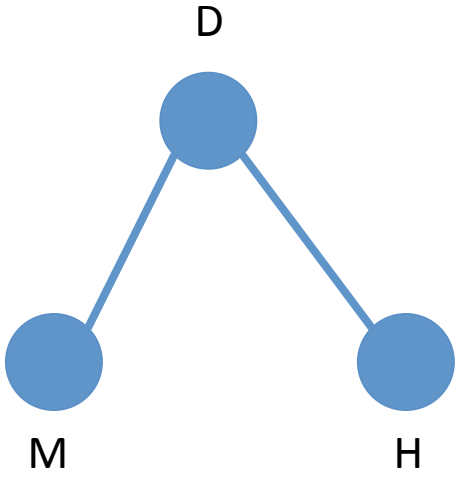


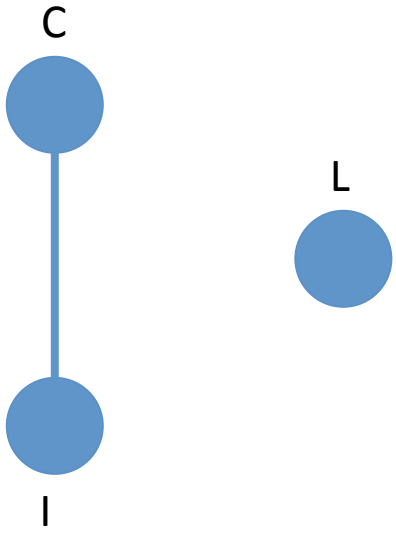




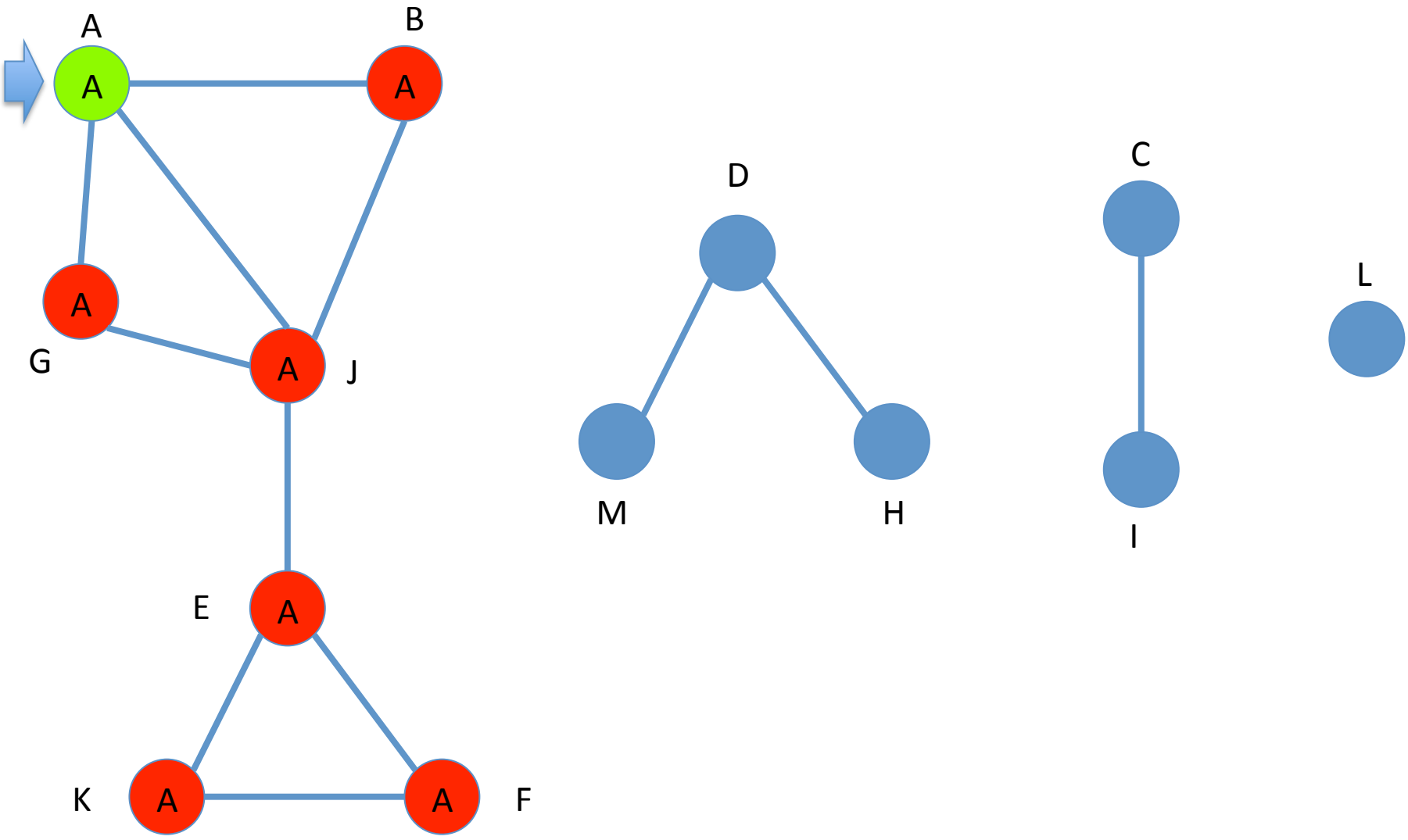
Recall Algorithm for Undirected Graphs



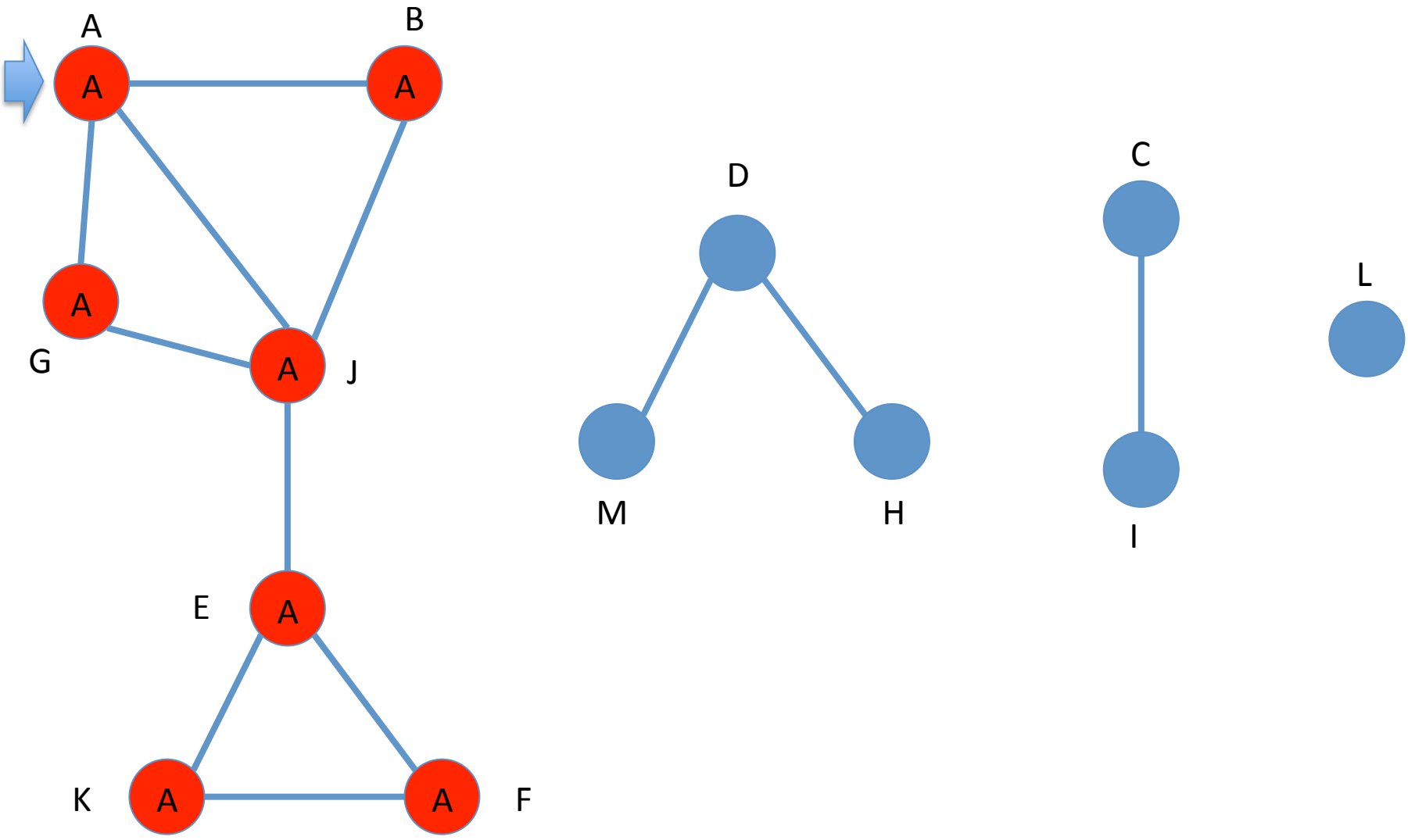




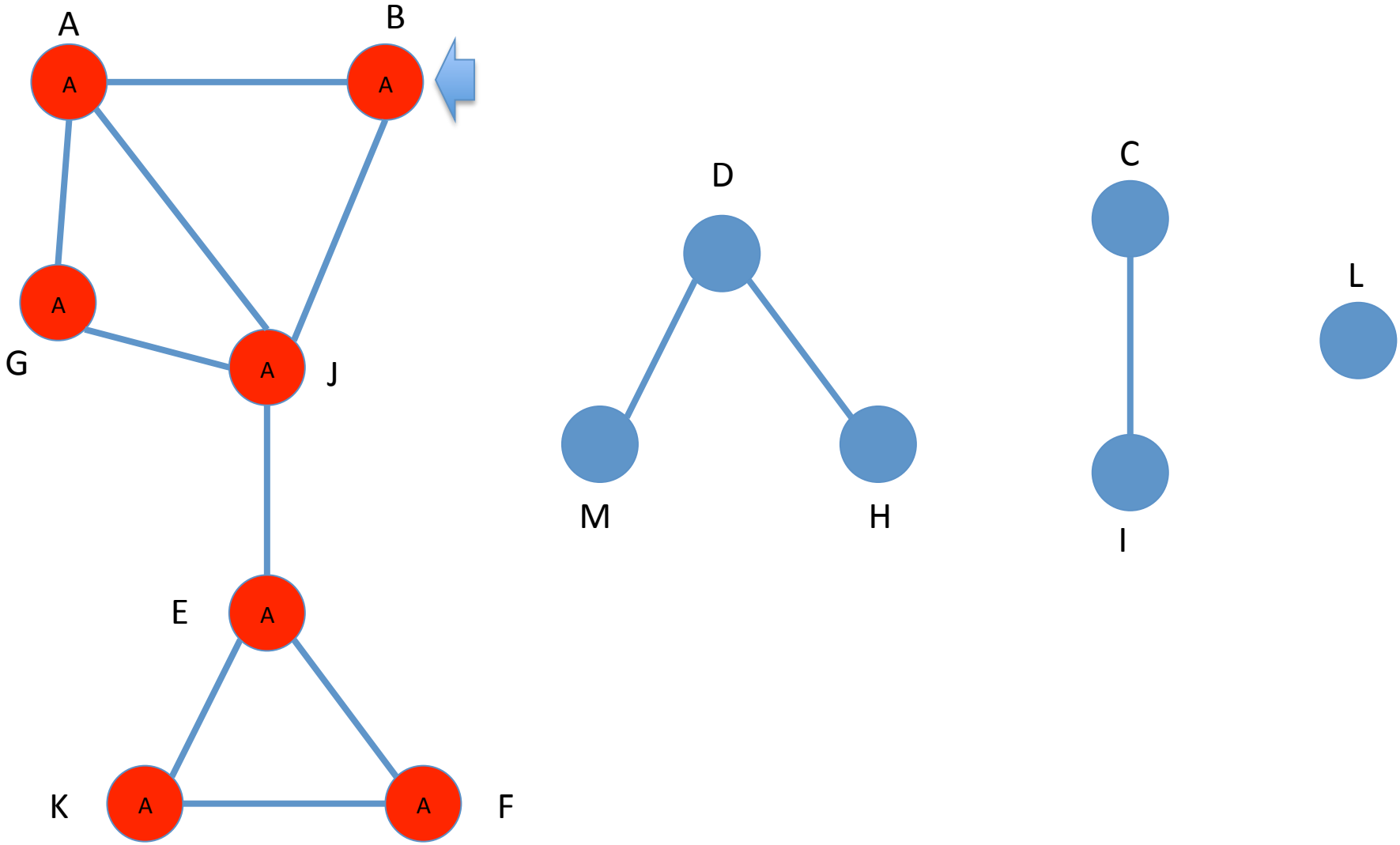
Recall Algorithm for Undirected Graphs



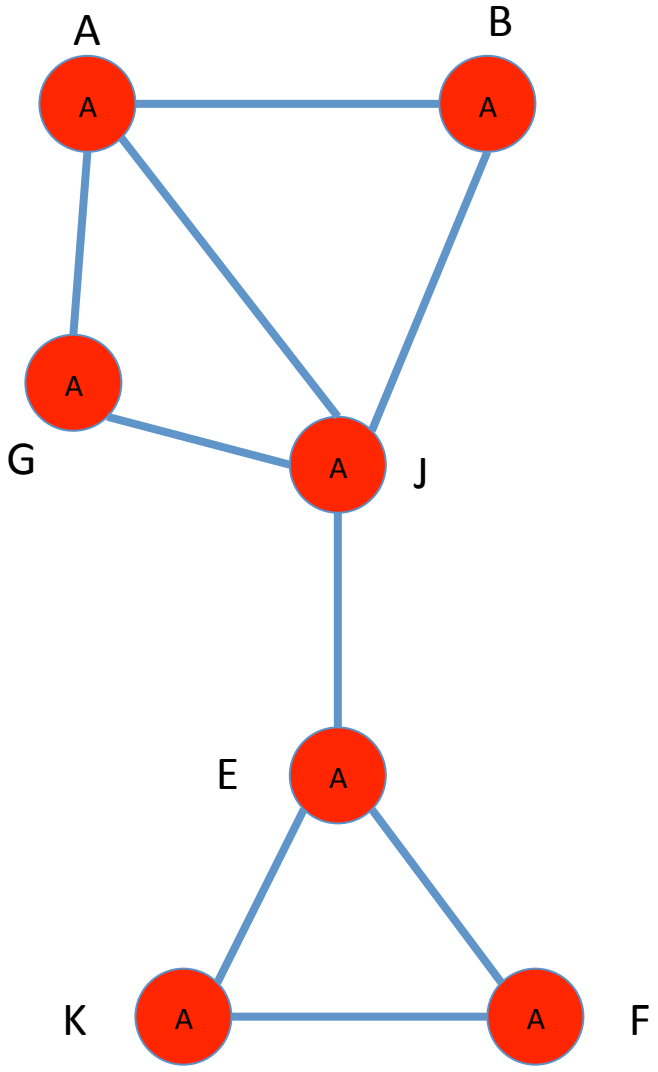
Recall Algorithm for Undirected Graphs

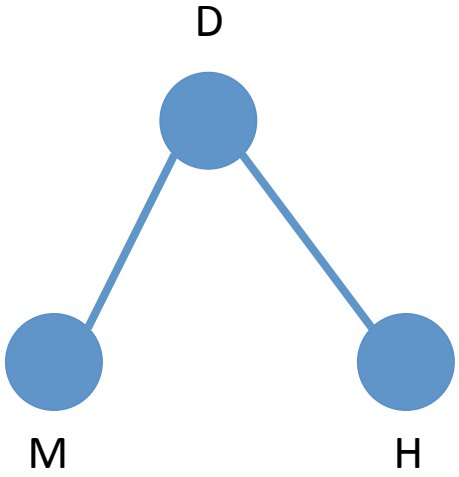


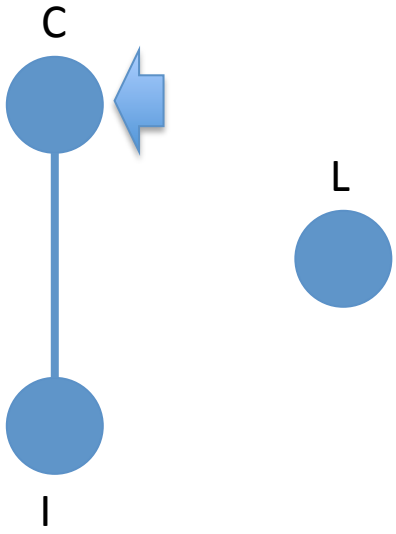
Recall Algorithm for Undirected Graphs



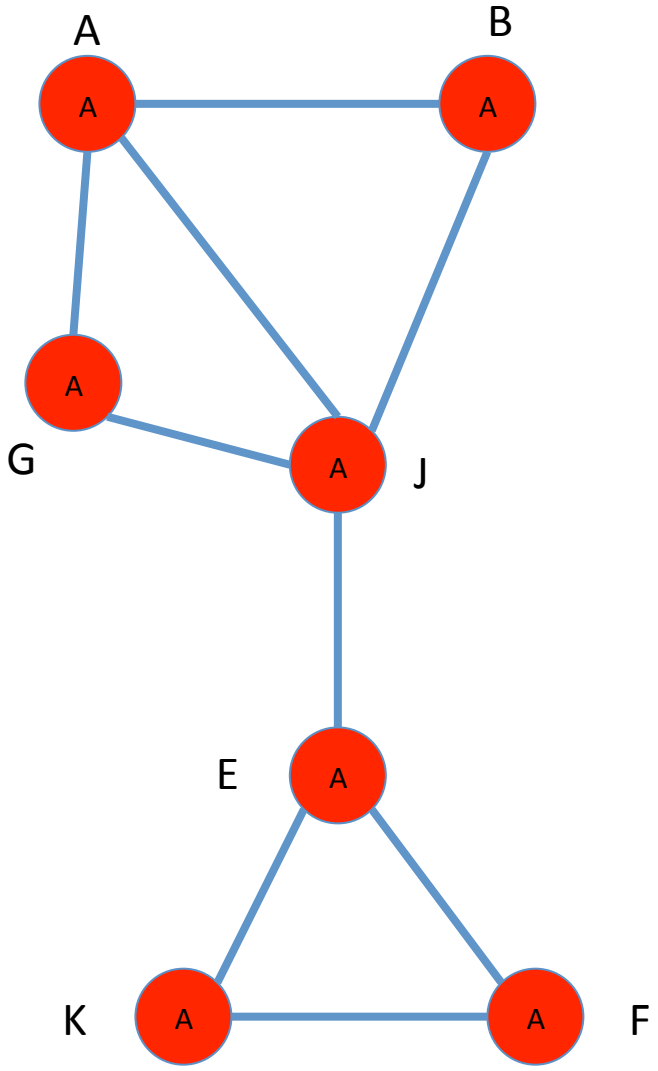
Recall Algorithm for Undirected Graphs

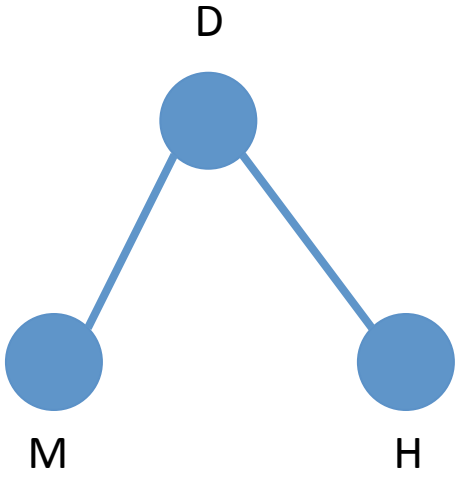


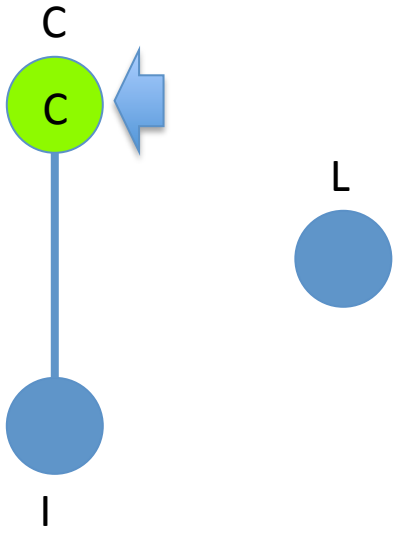




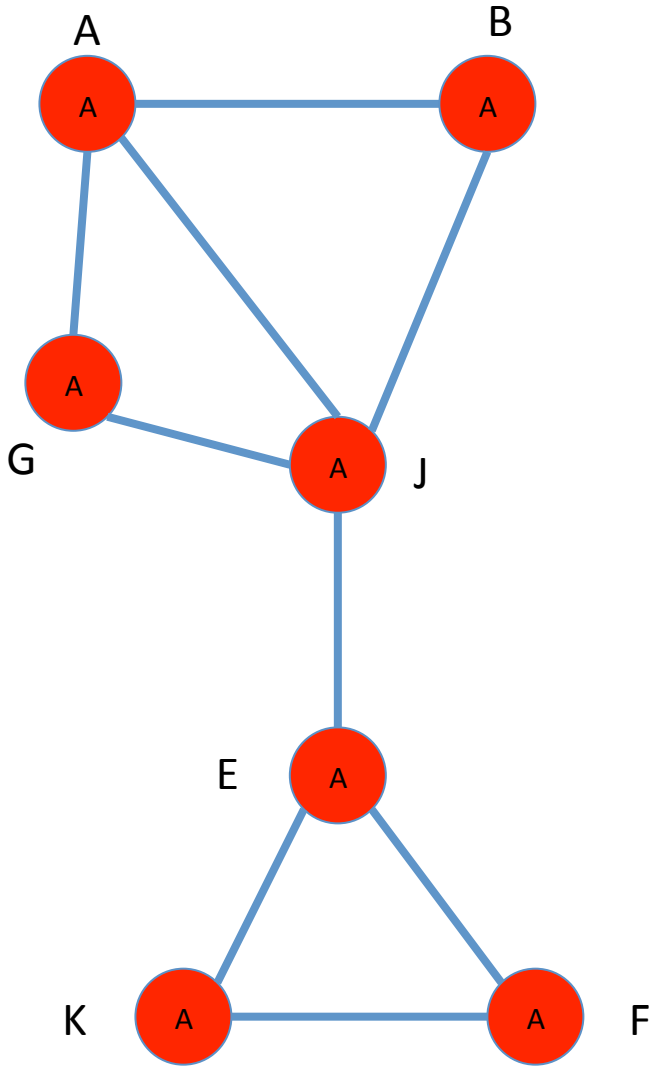
Recall Algorithm for Undirected Graphs

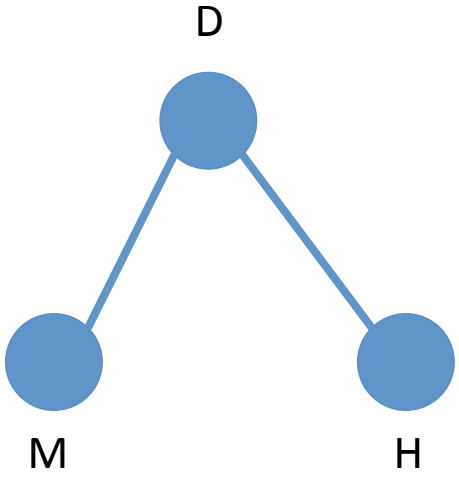


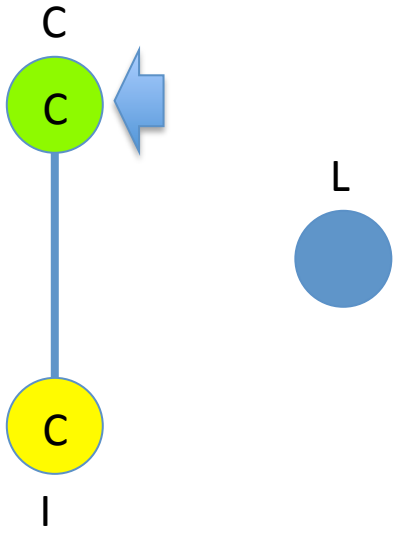




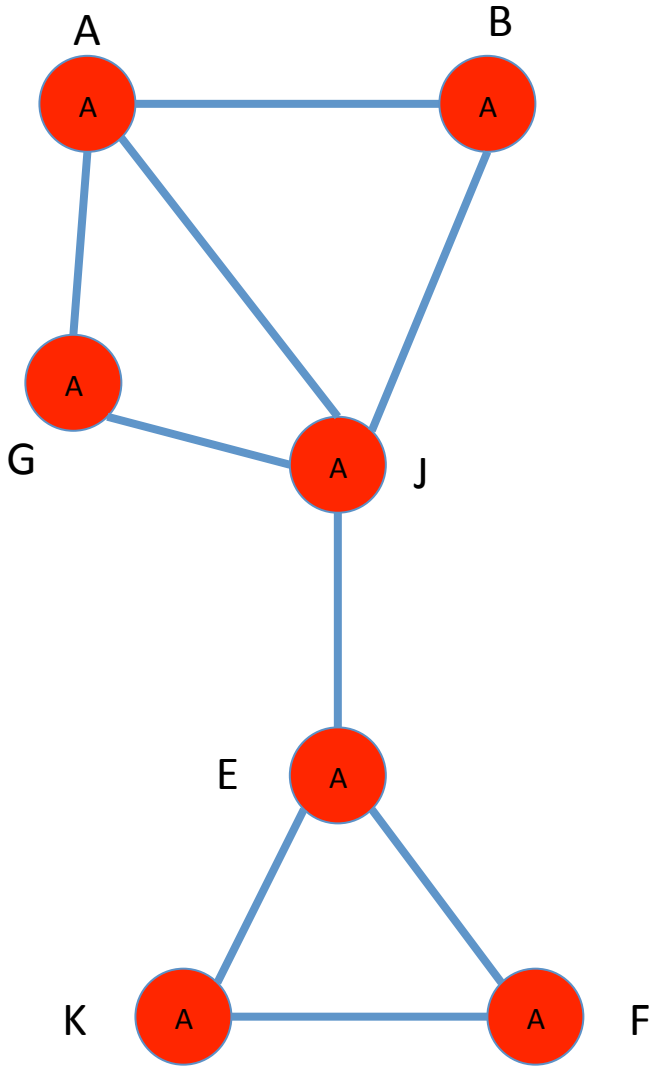
Recall Algorithm for Undirected Graphs

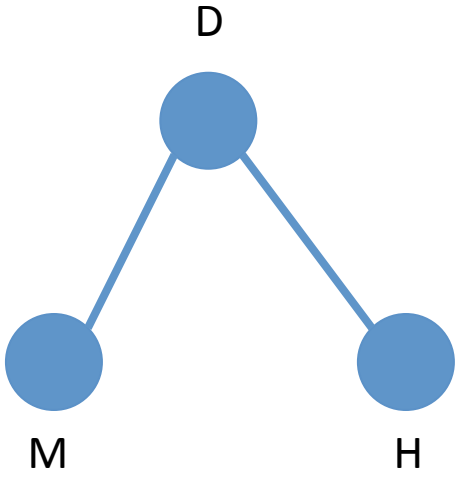


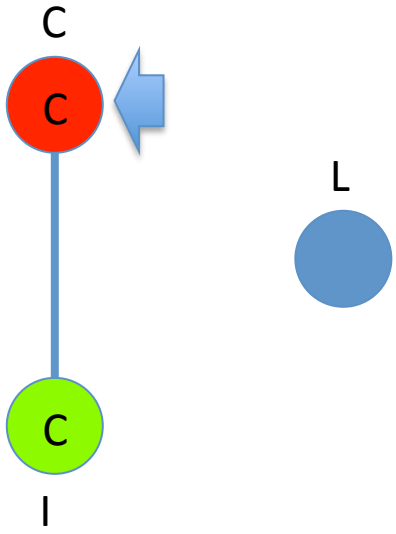




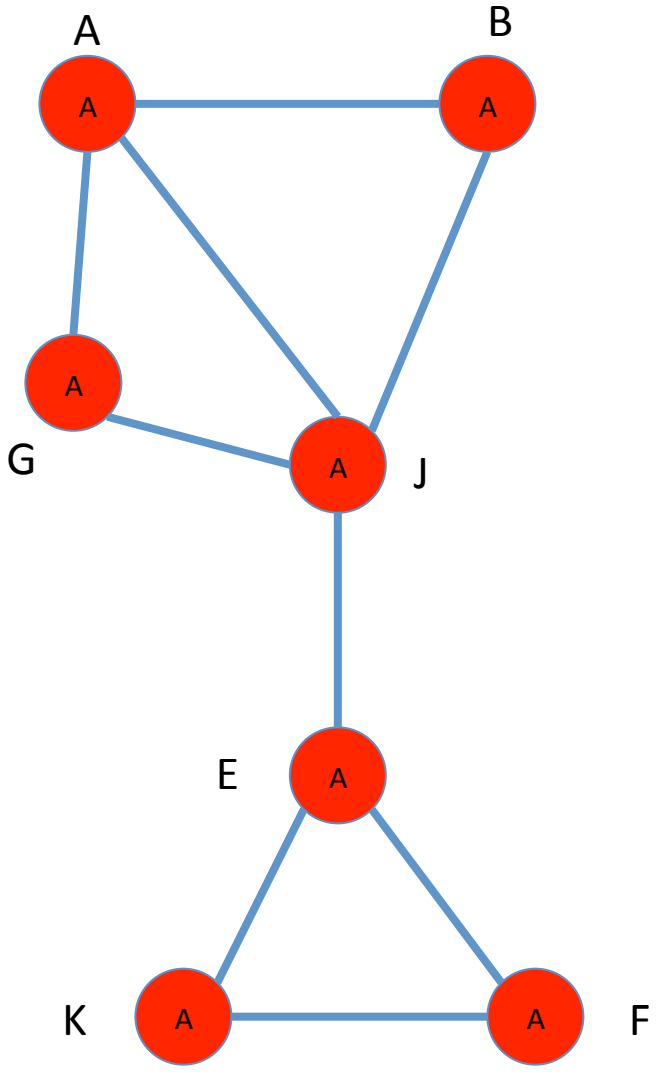
Recall Algorithm for Undirected Graphs

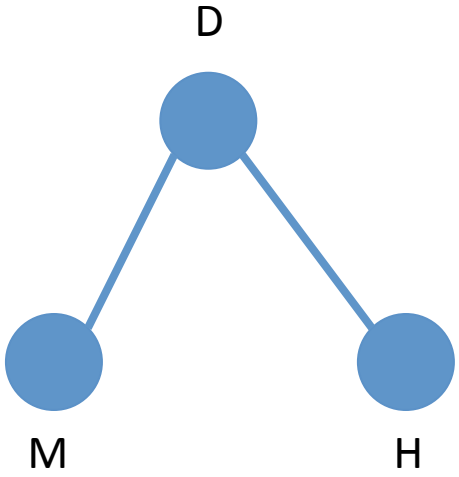


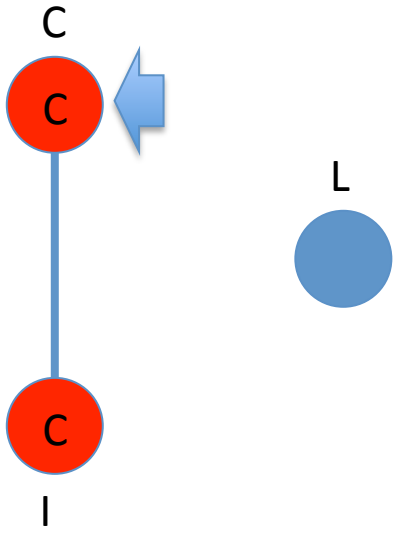




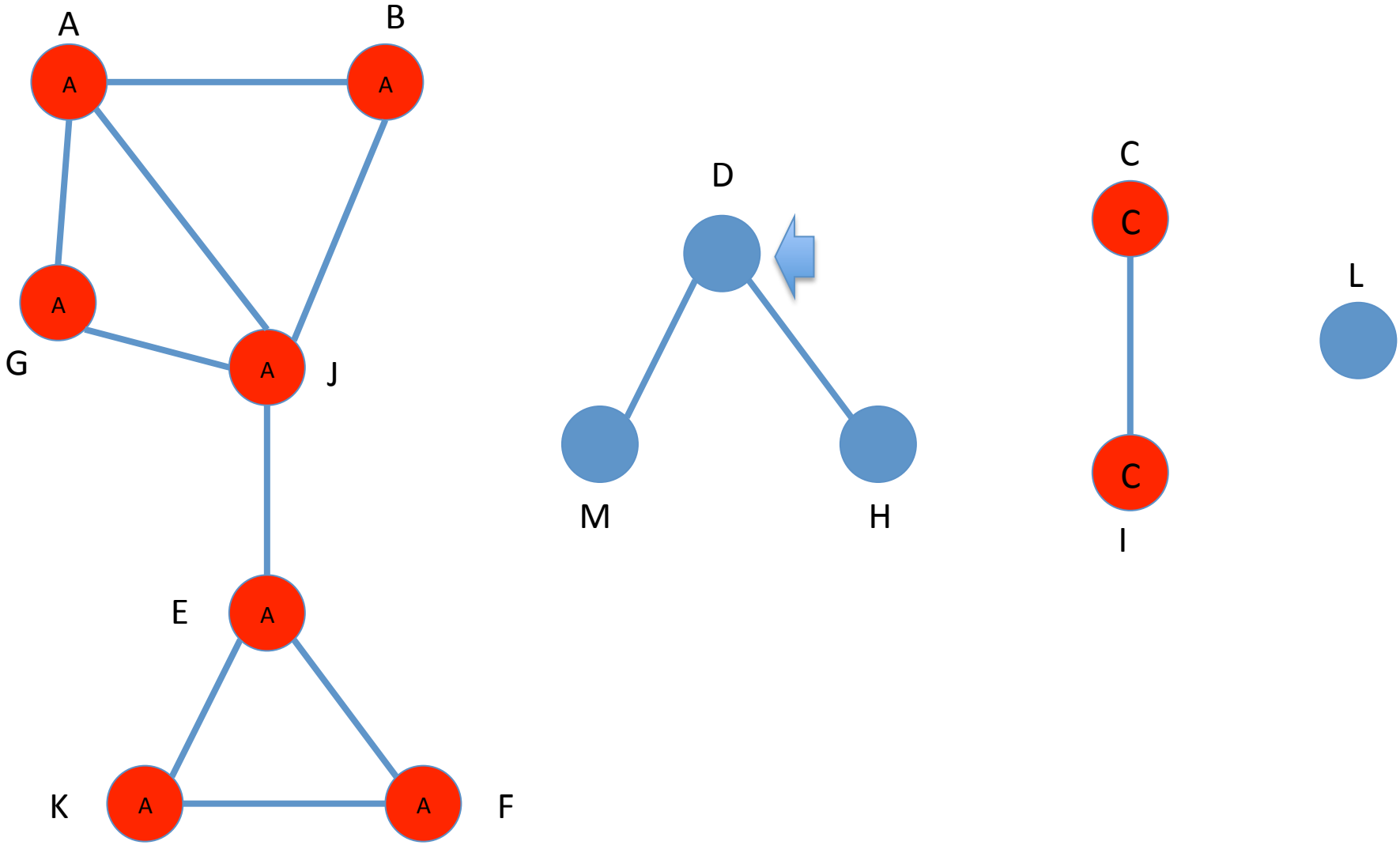
Recall Algorithm for Undirected Graphs



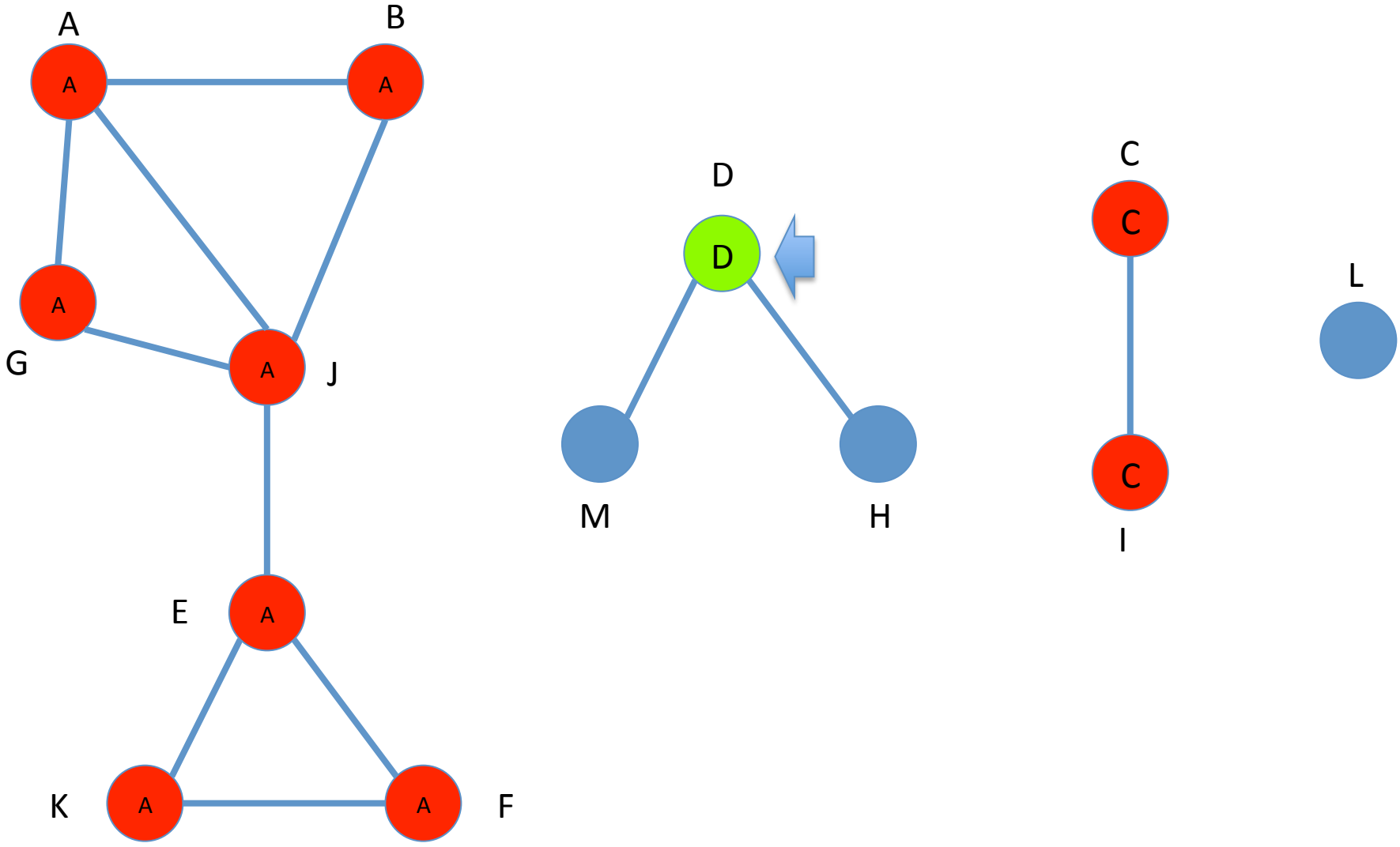




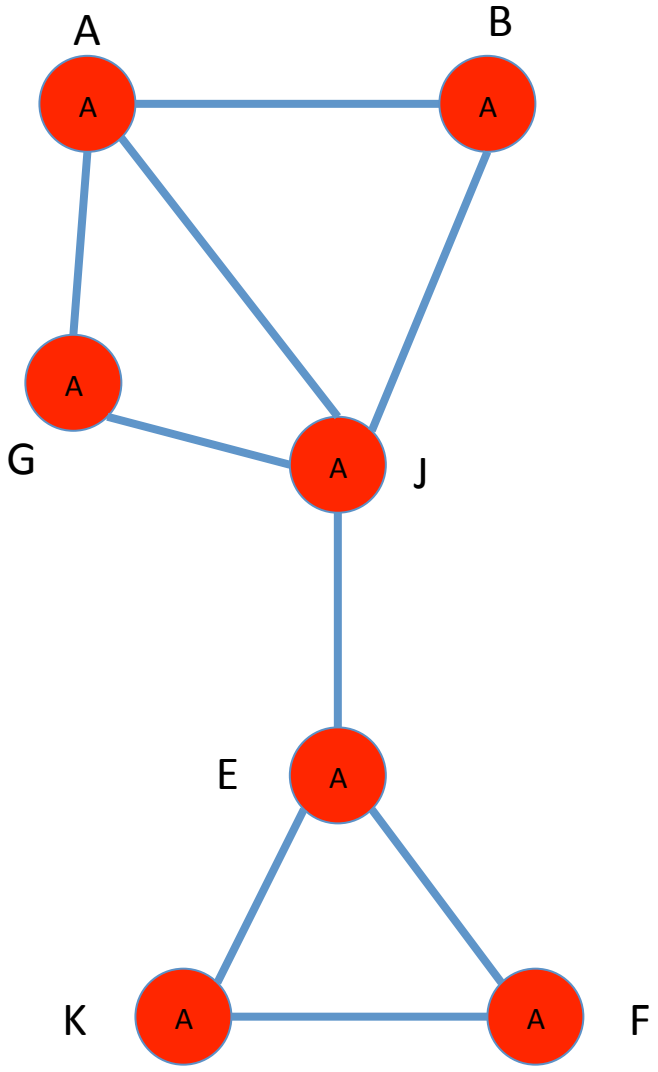
Recall Algorithm for Undirected Graphs

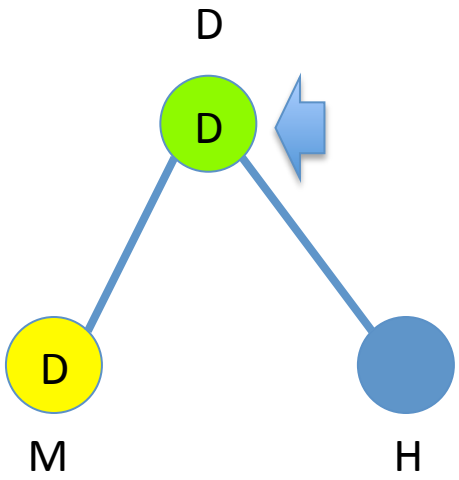


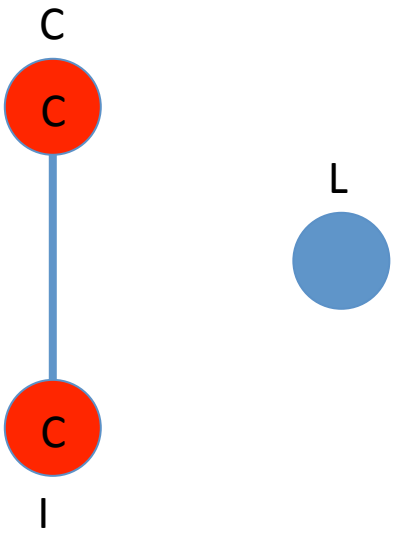
Recall Algorithm for Undirected Graphs



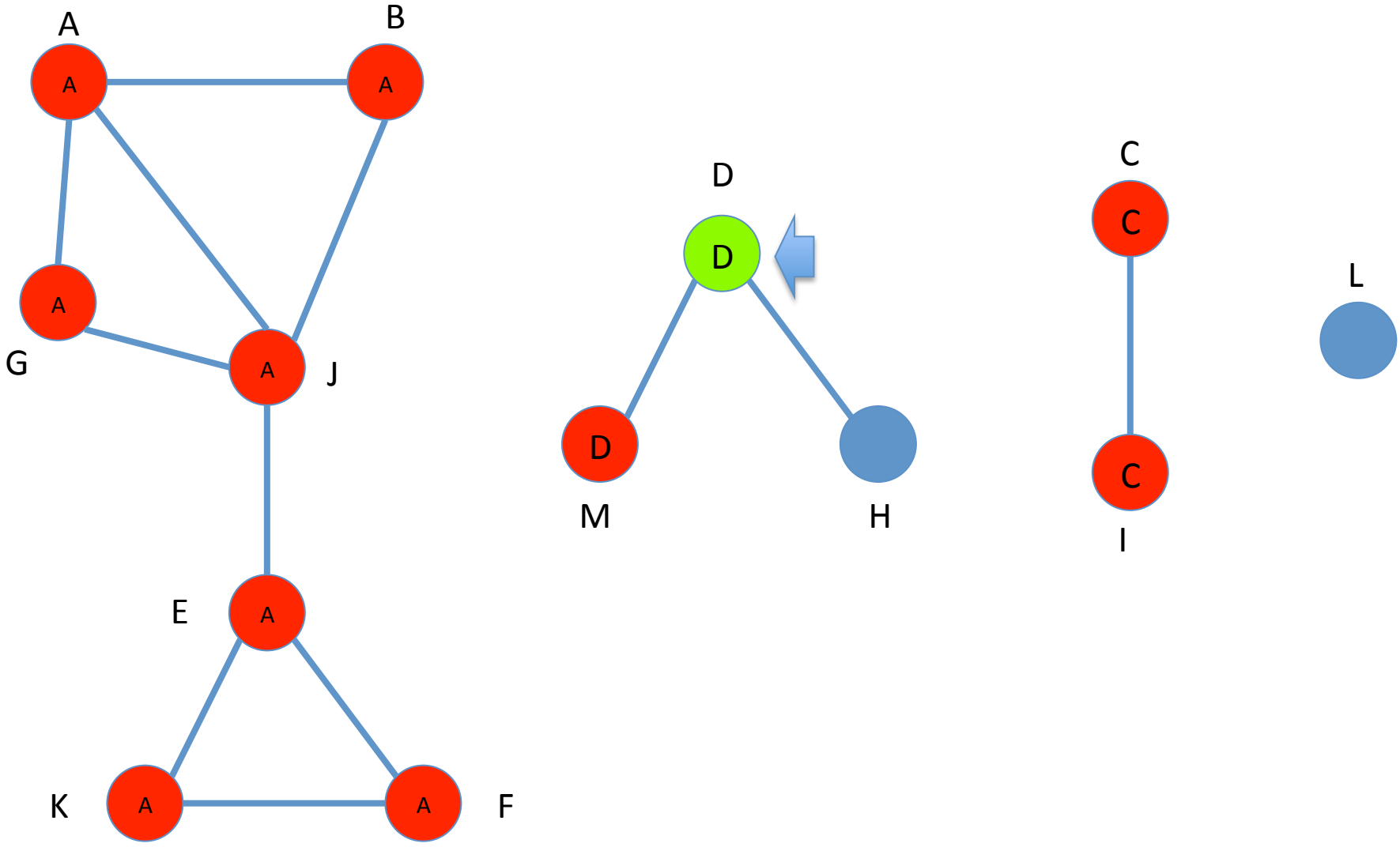
Recall Algorithm for Undirected Graphs



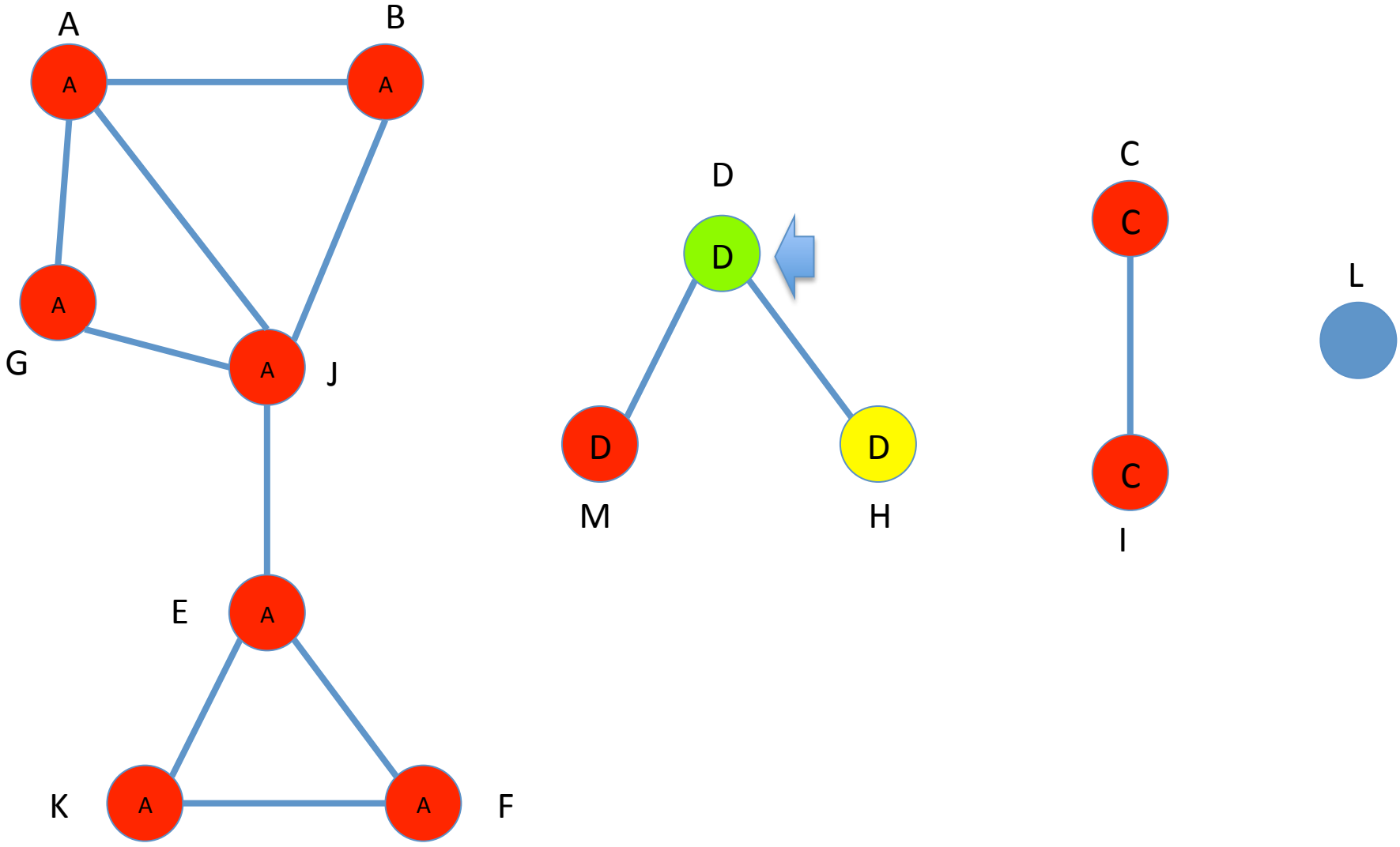




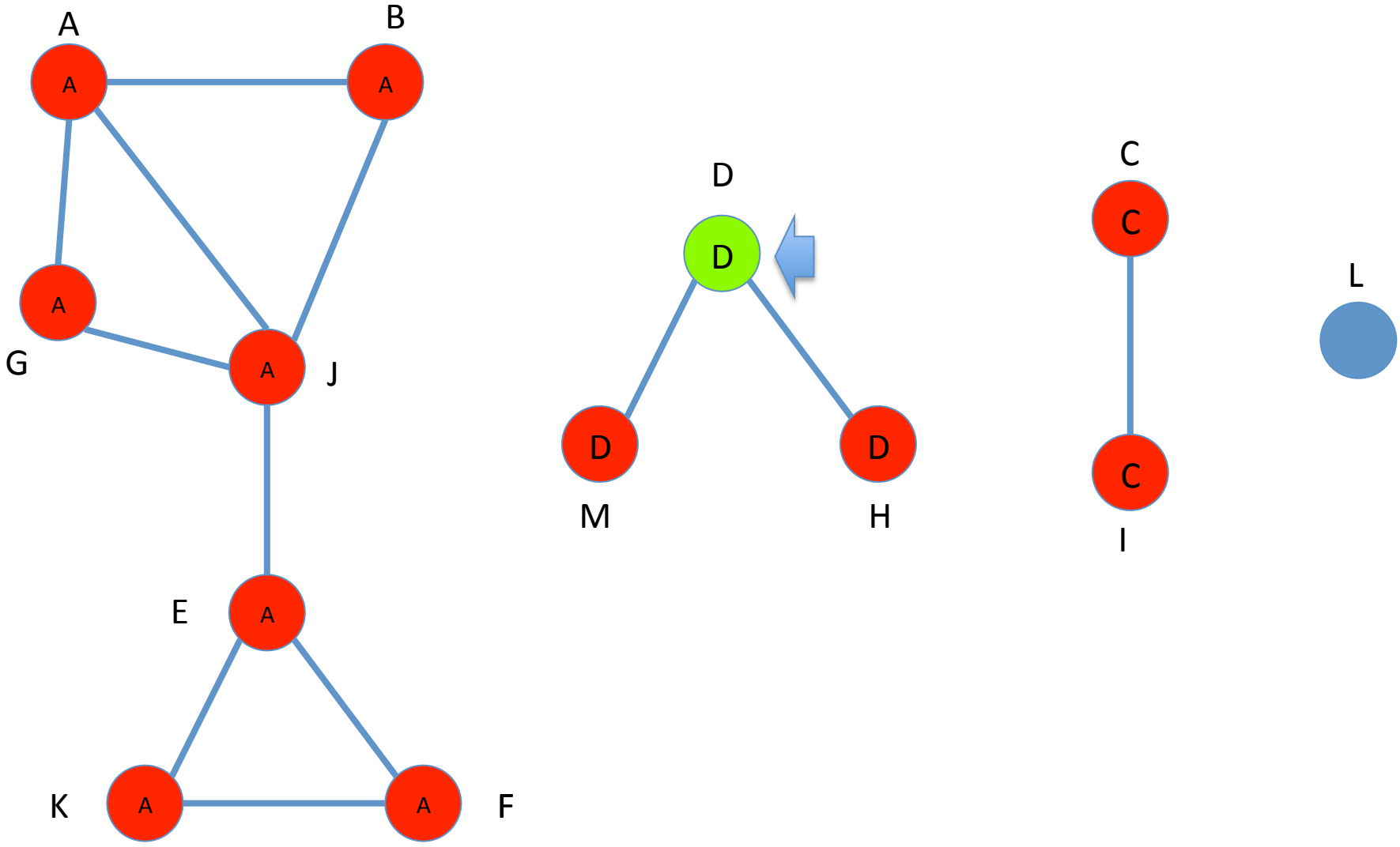
Recall Algorithm for Undirected Graphs



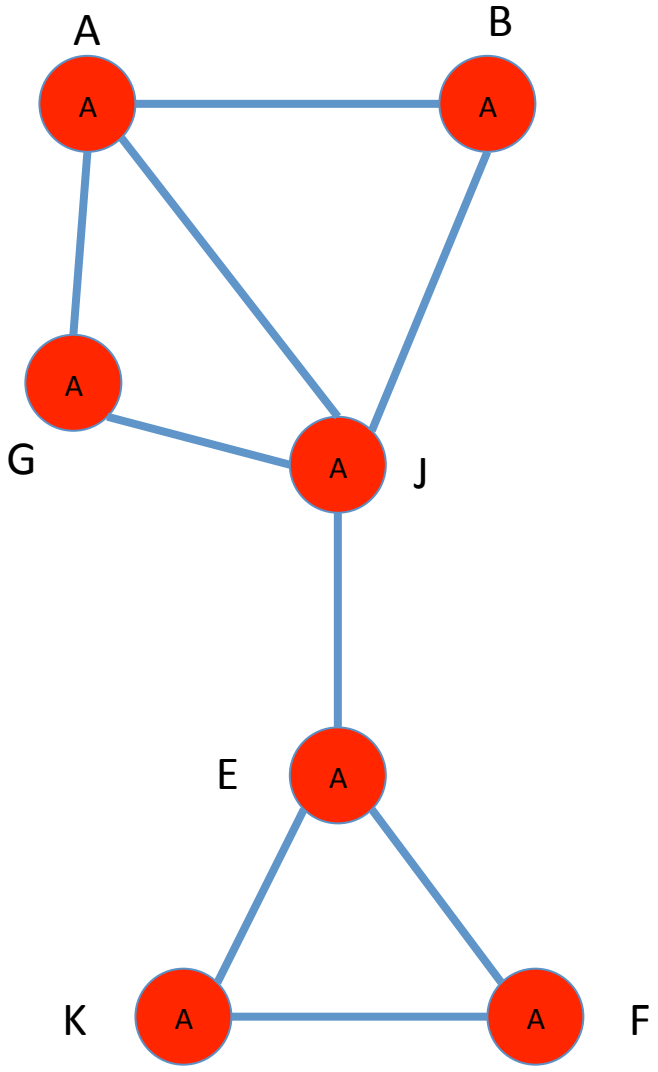
Recall Algorithm for Undirected Graphs

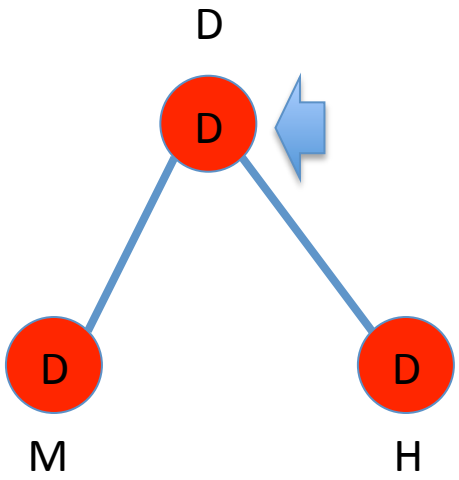


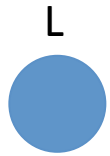
Recall Algorithm for Undirected Graphs



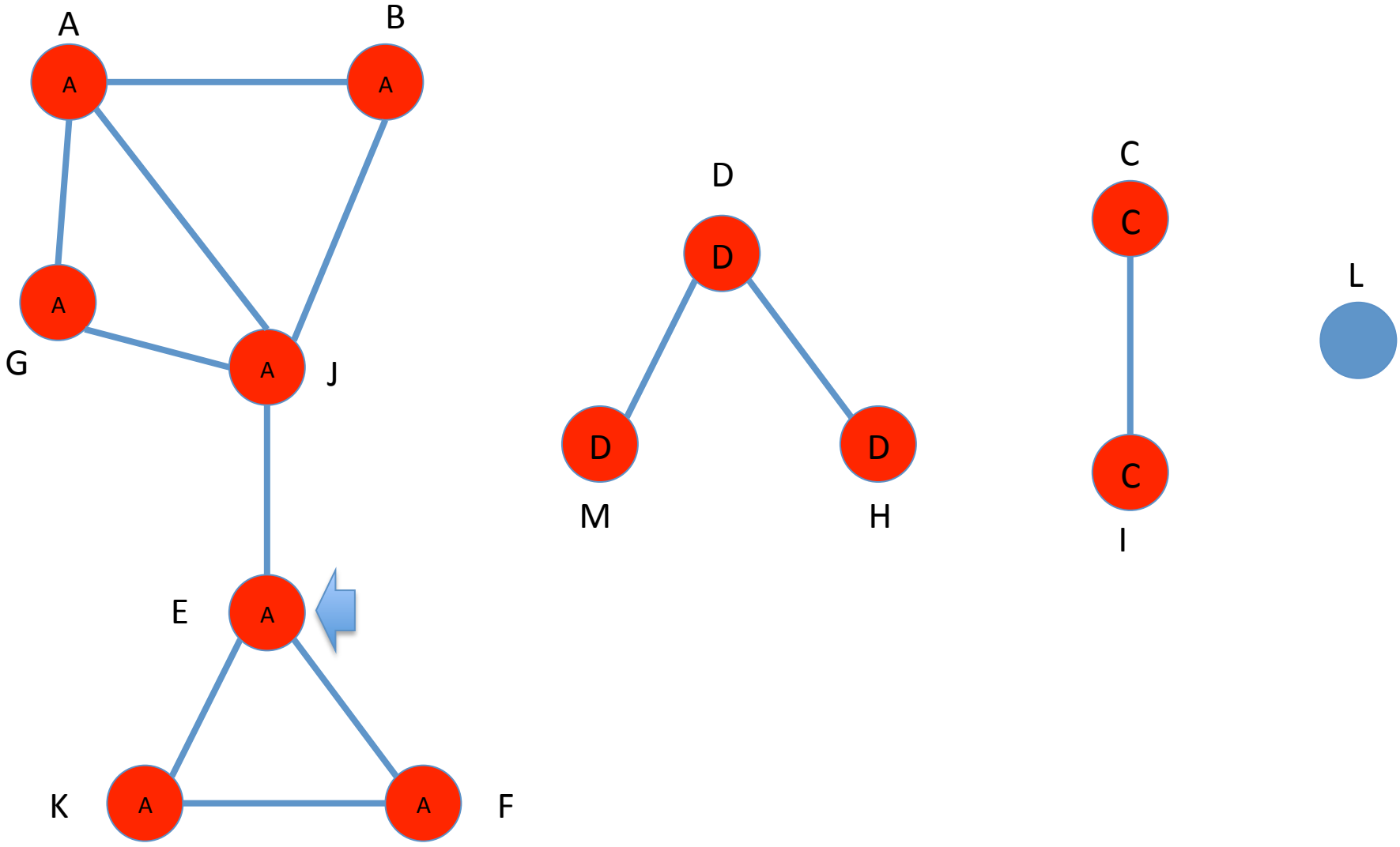
Recall Algorithm for Undirected Graphs



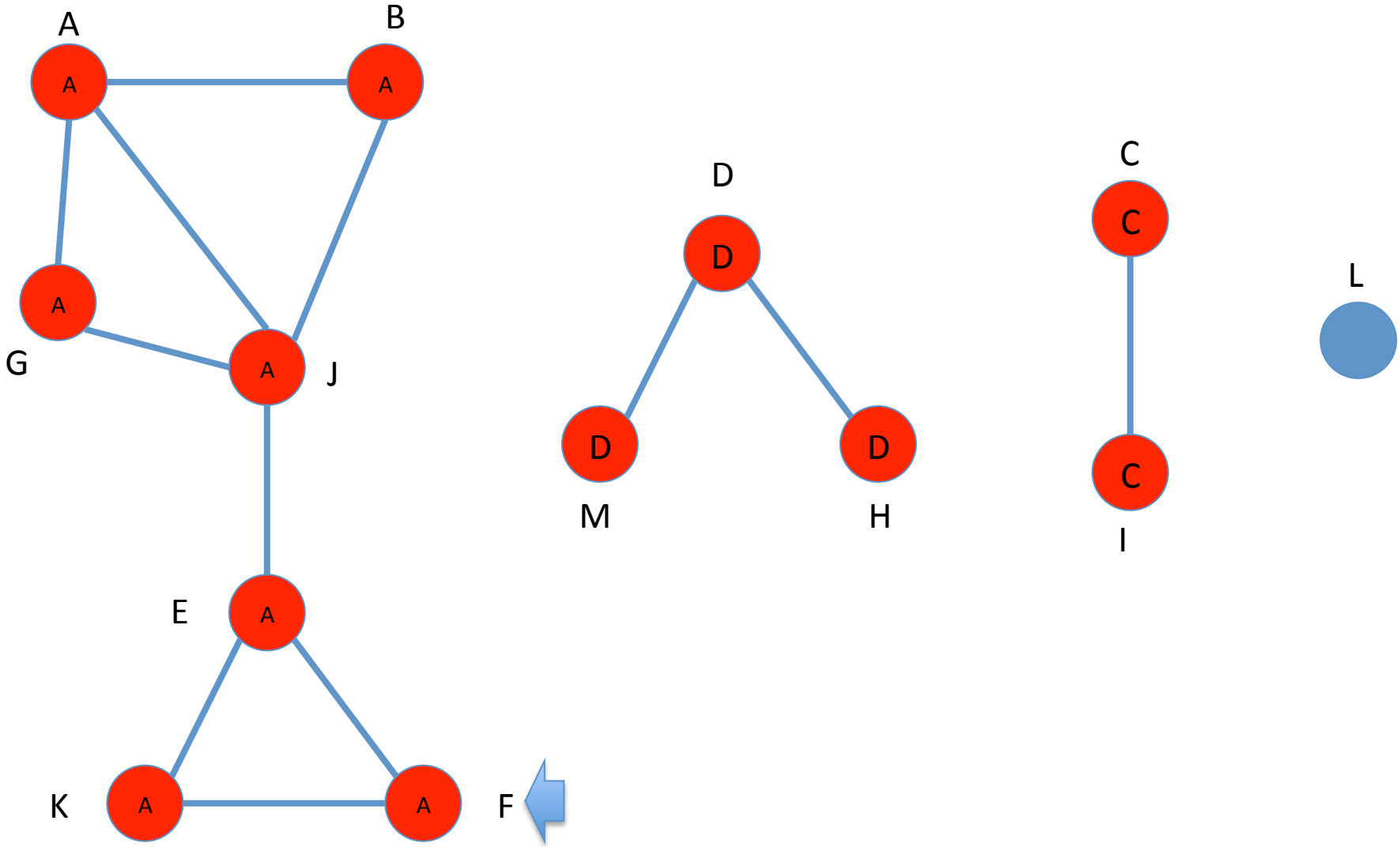




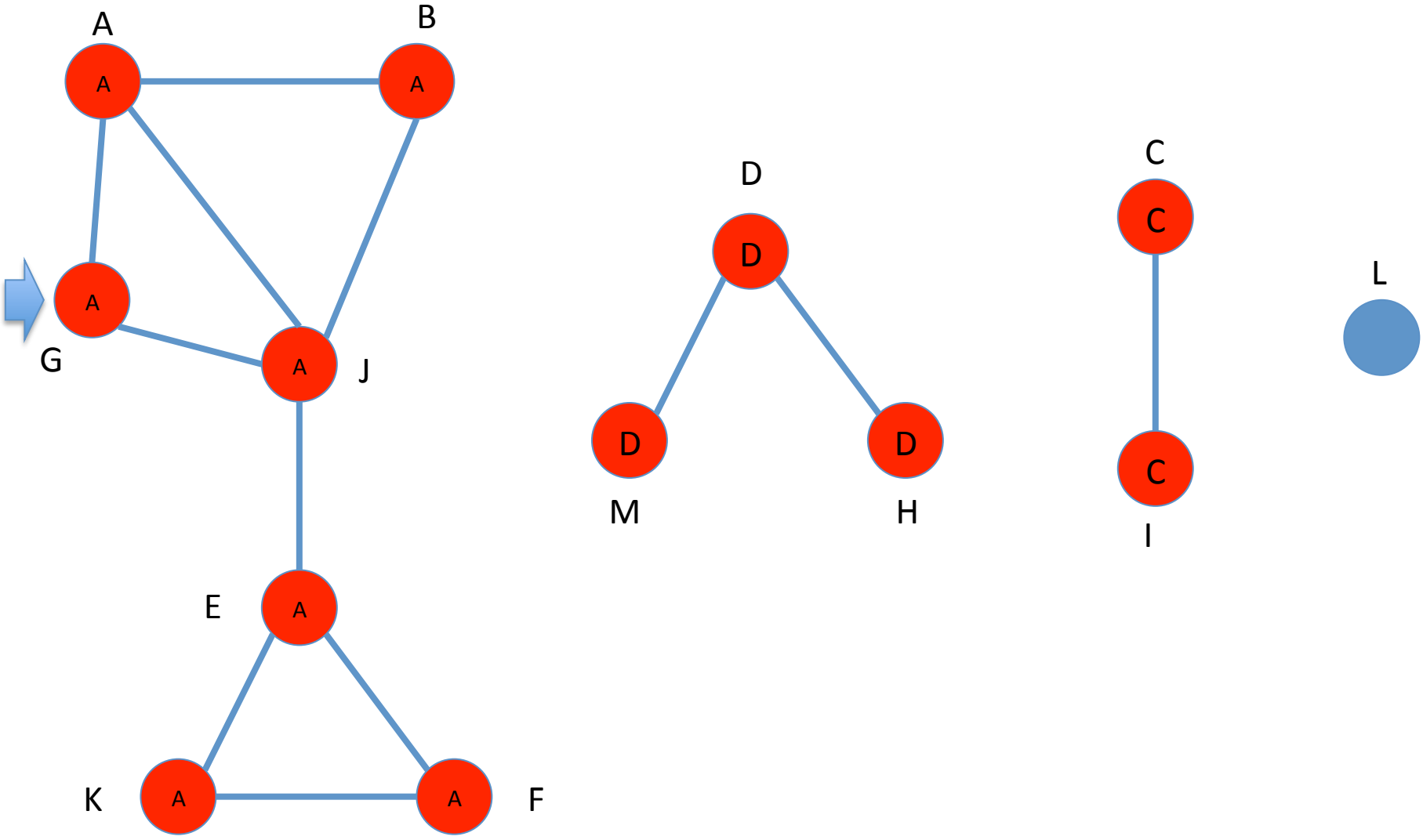
Recall Algorithm for Undirected Graphs



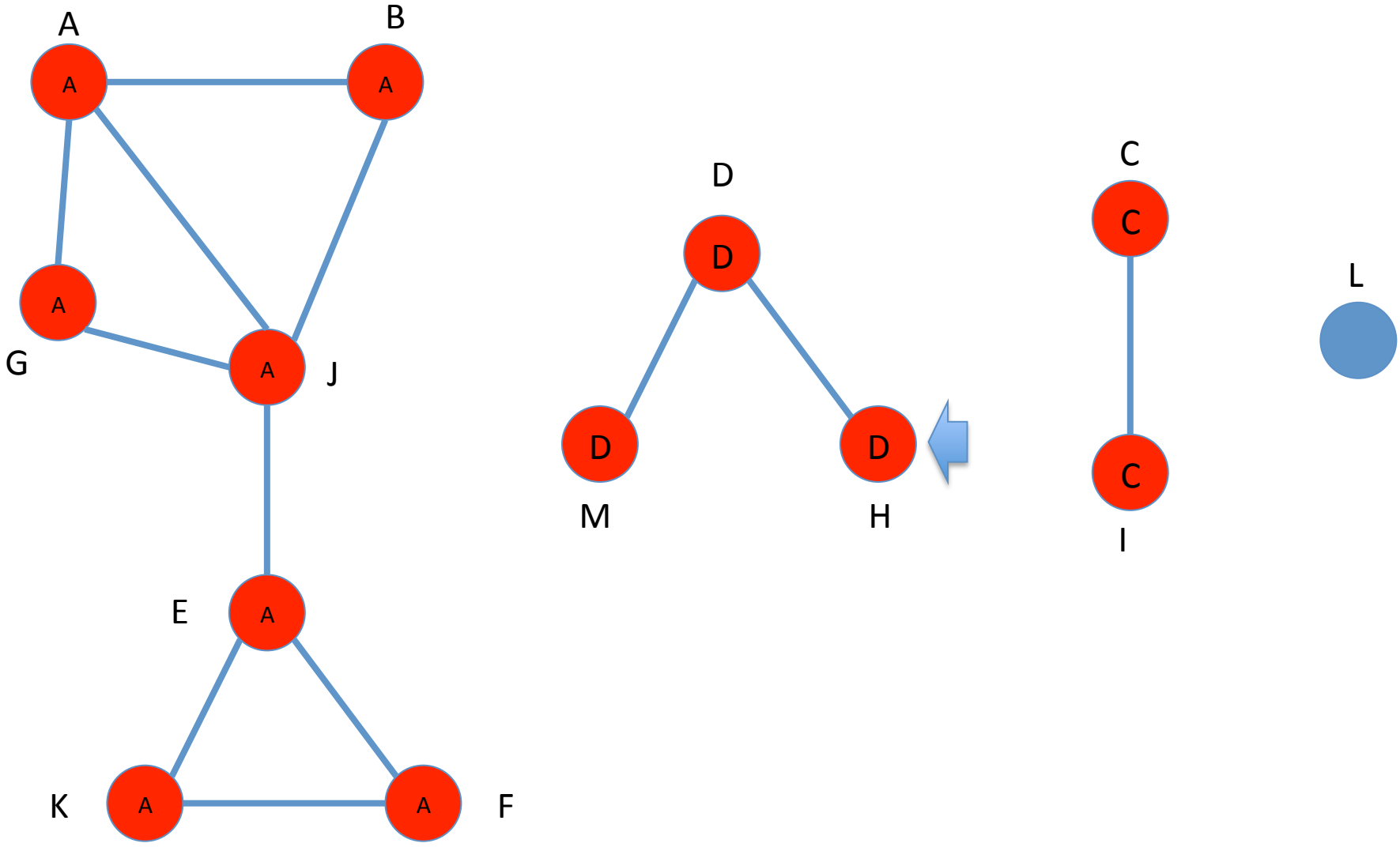
Recall Algorithm for Undirected Graphs



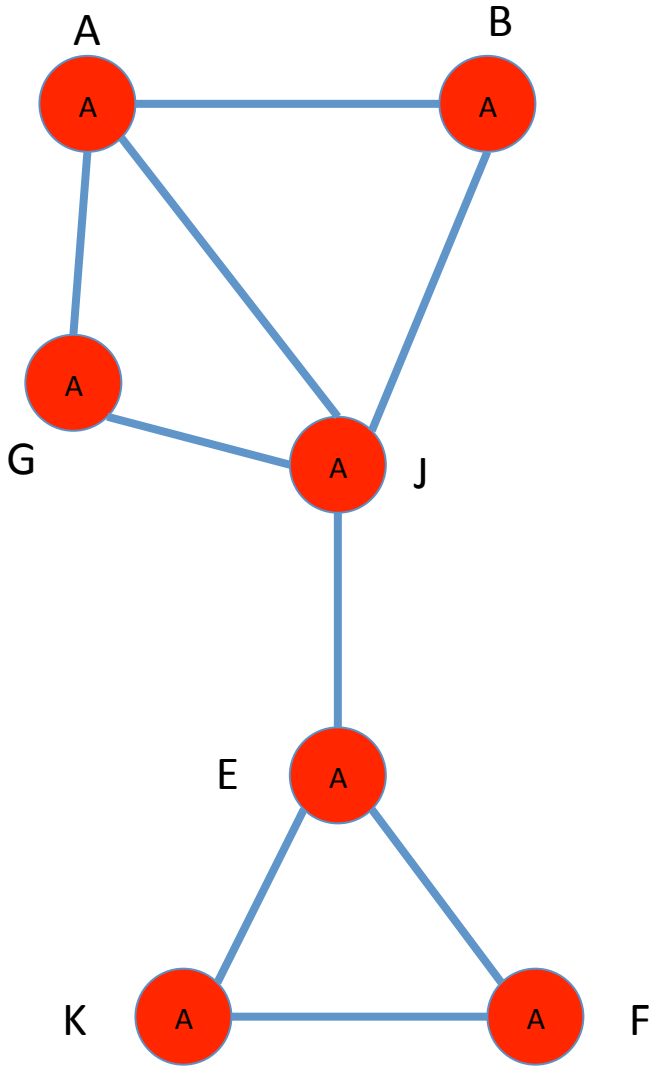
Recall Algorithm for Undirected Graphs

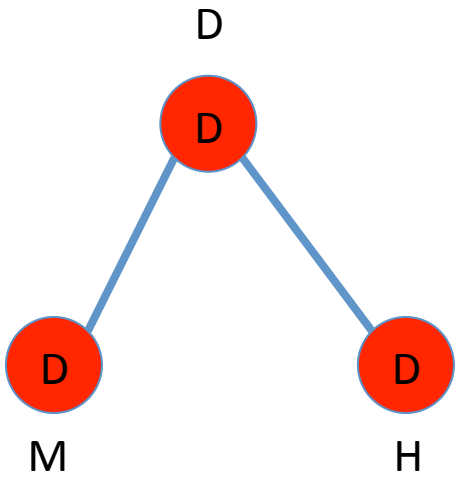


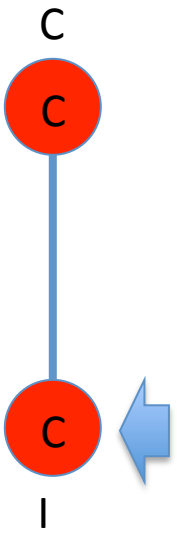
Recall Algorithm for Undirected Graphs

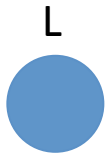


Recall Algorithm for Undirected Graphs

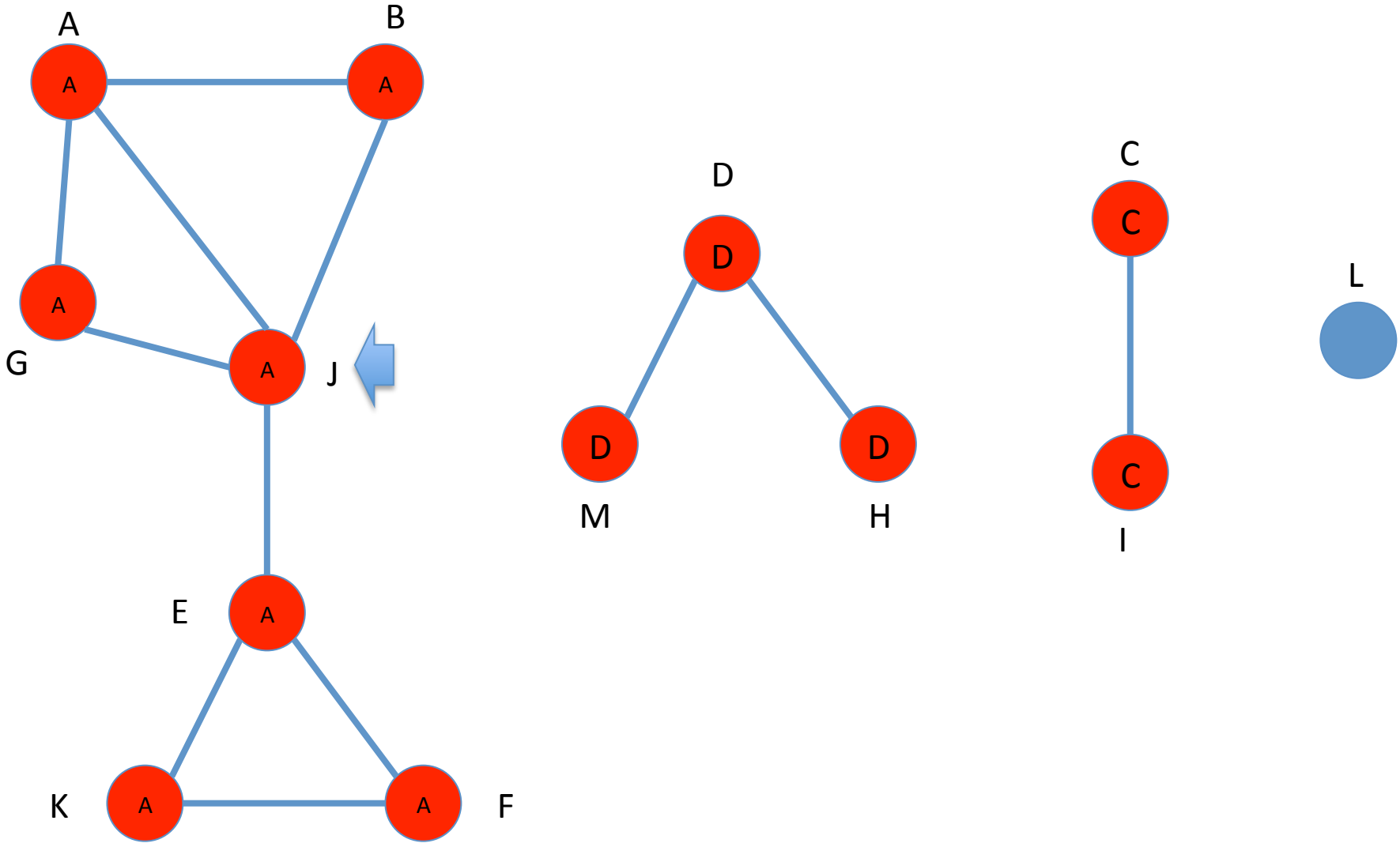




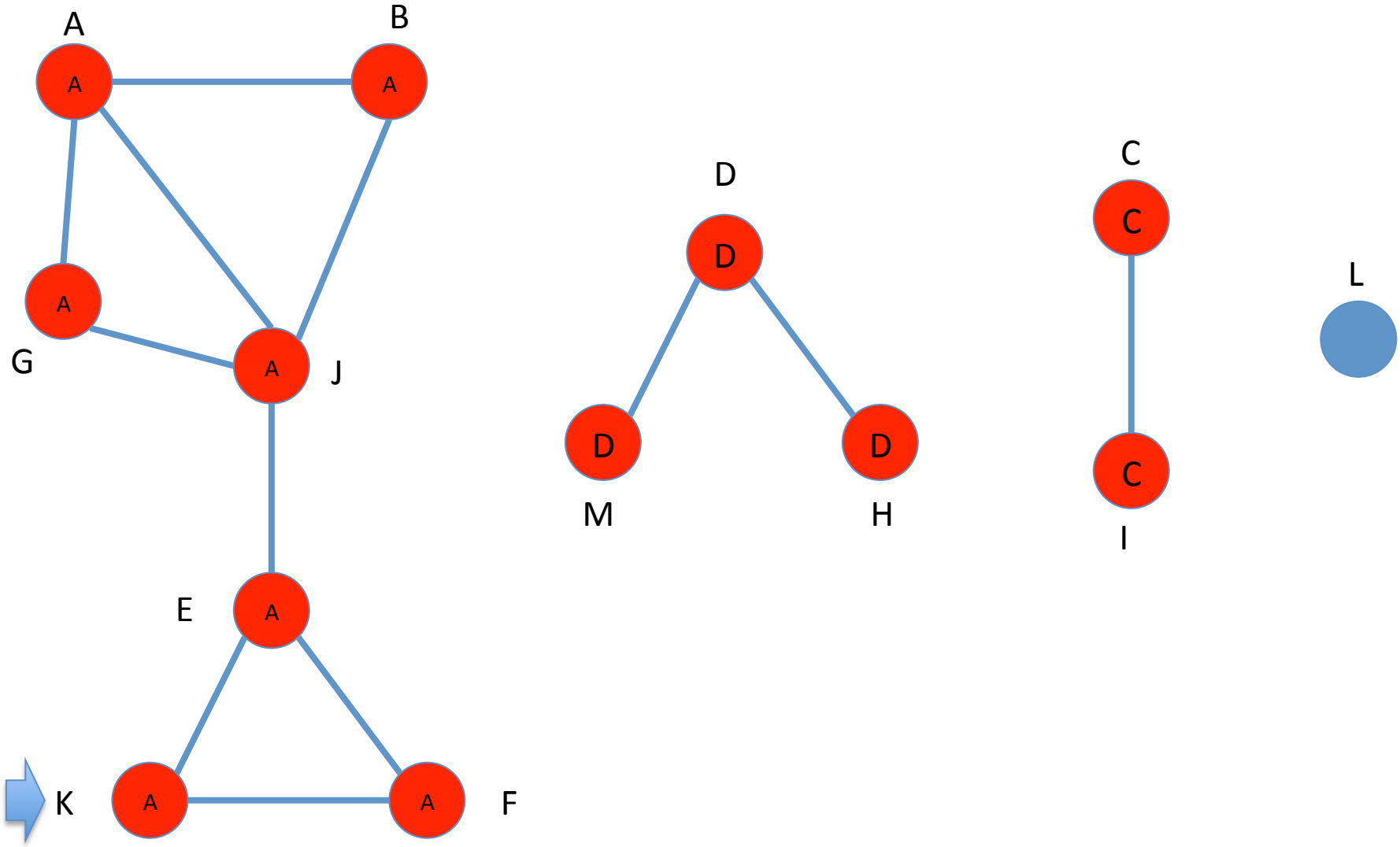




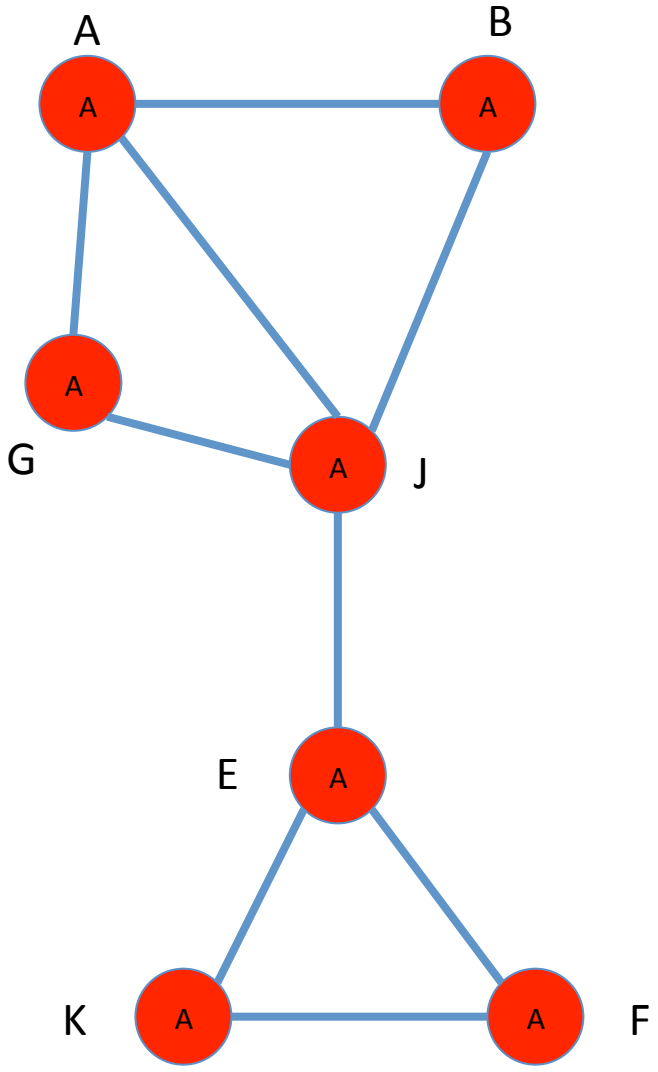
Recall Algorithm for Undirected Graphs

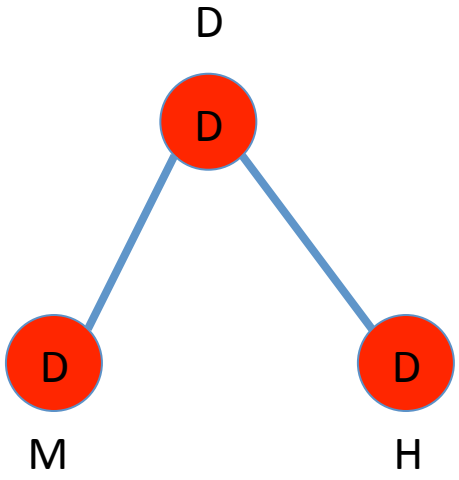


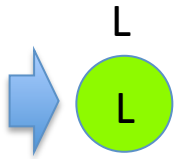
Recall Algorithm for Undirected Graphs



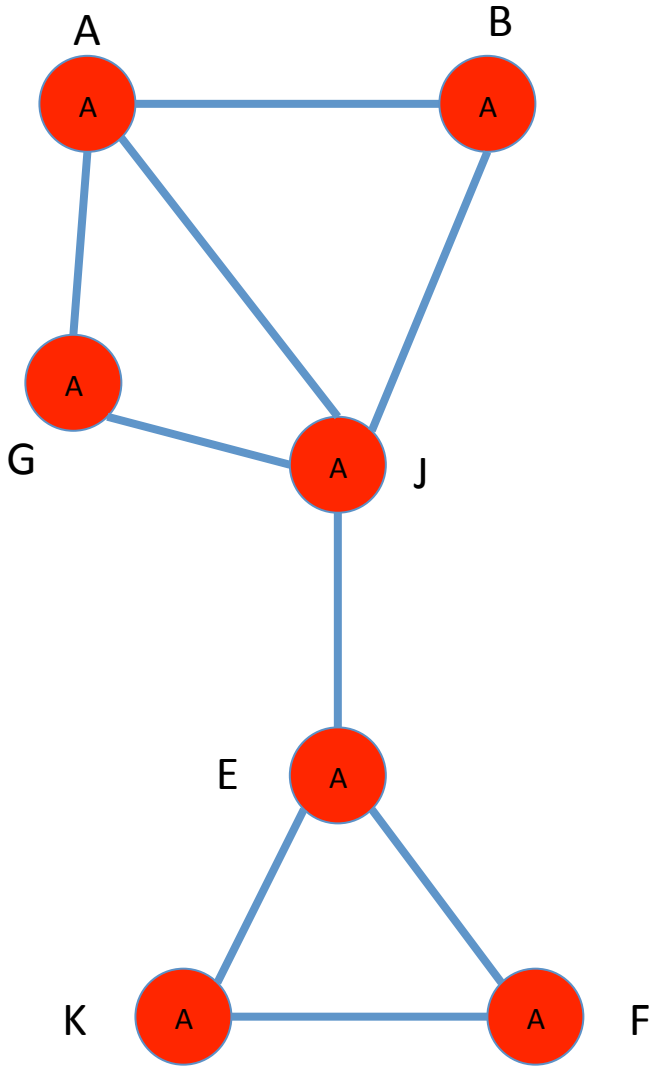
Recall Algorithm for Undirected Graphs

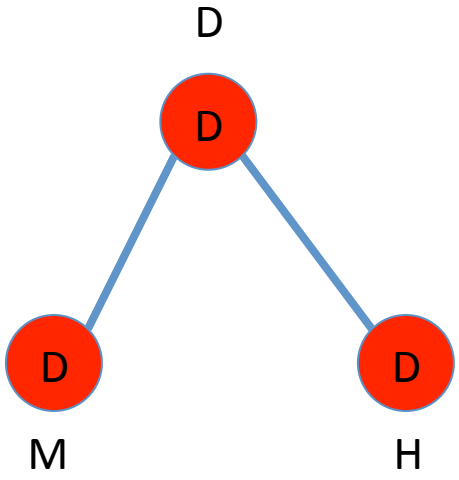


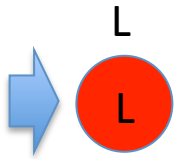




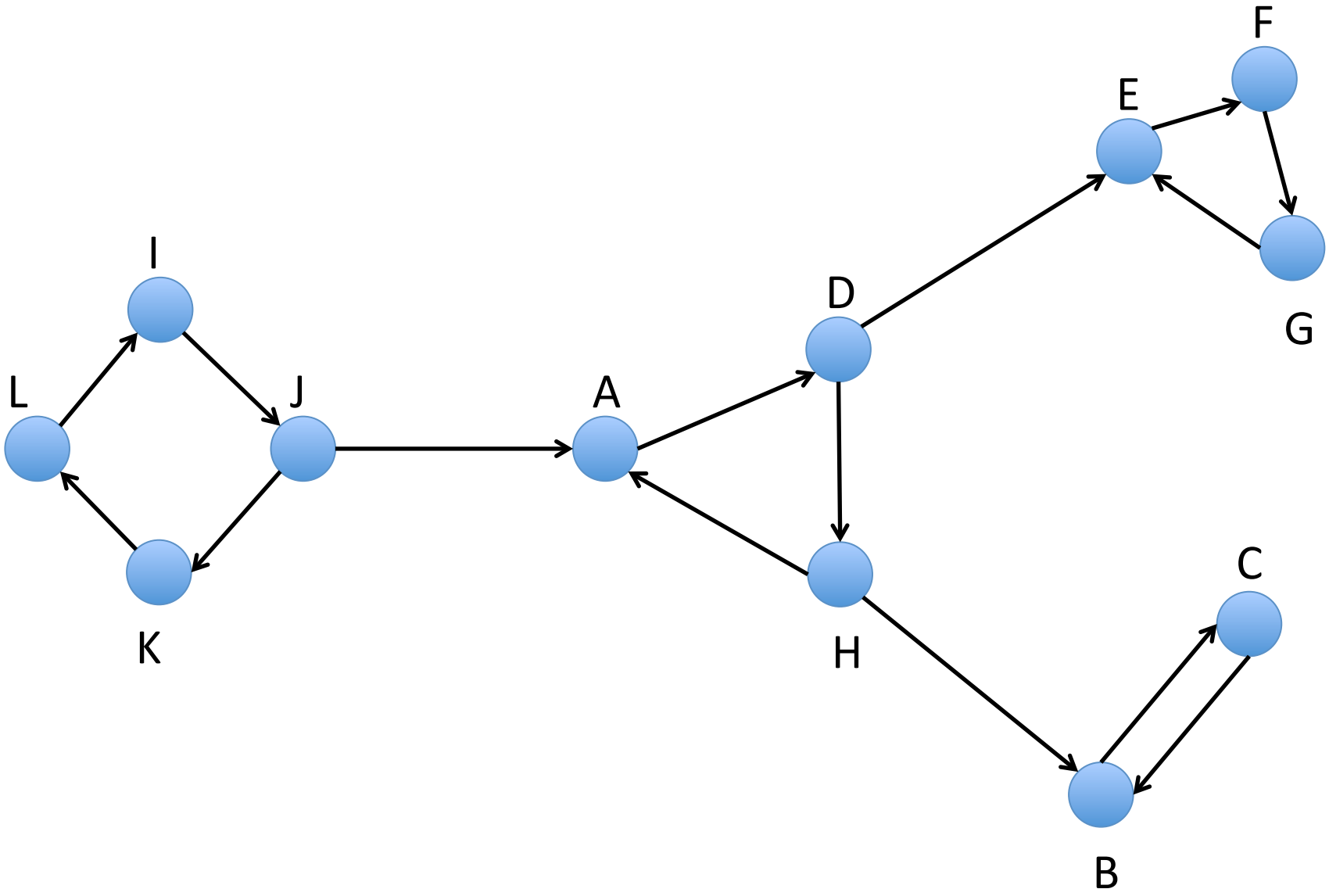
Recall Algorithm for Undirected Graphs



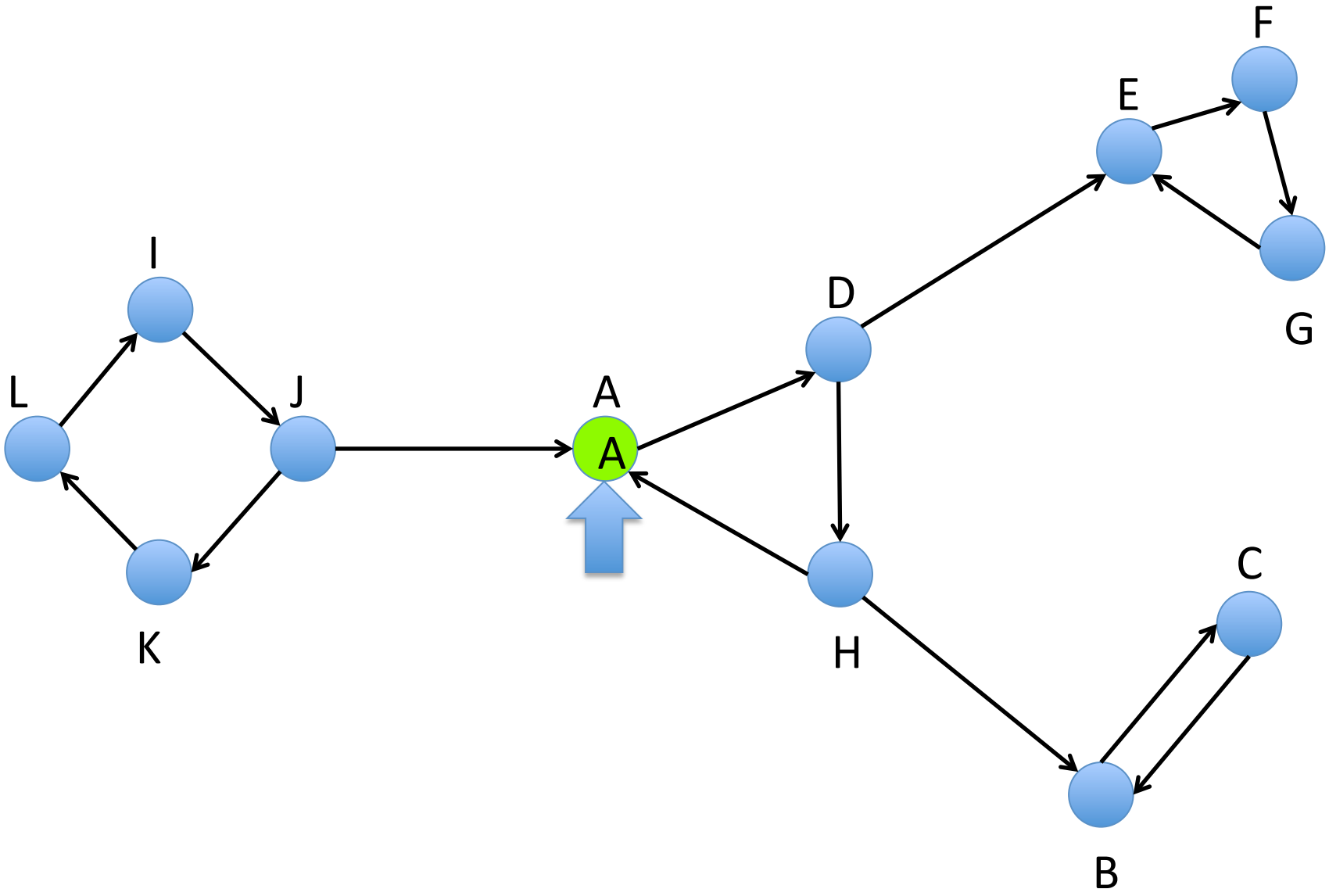




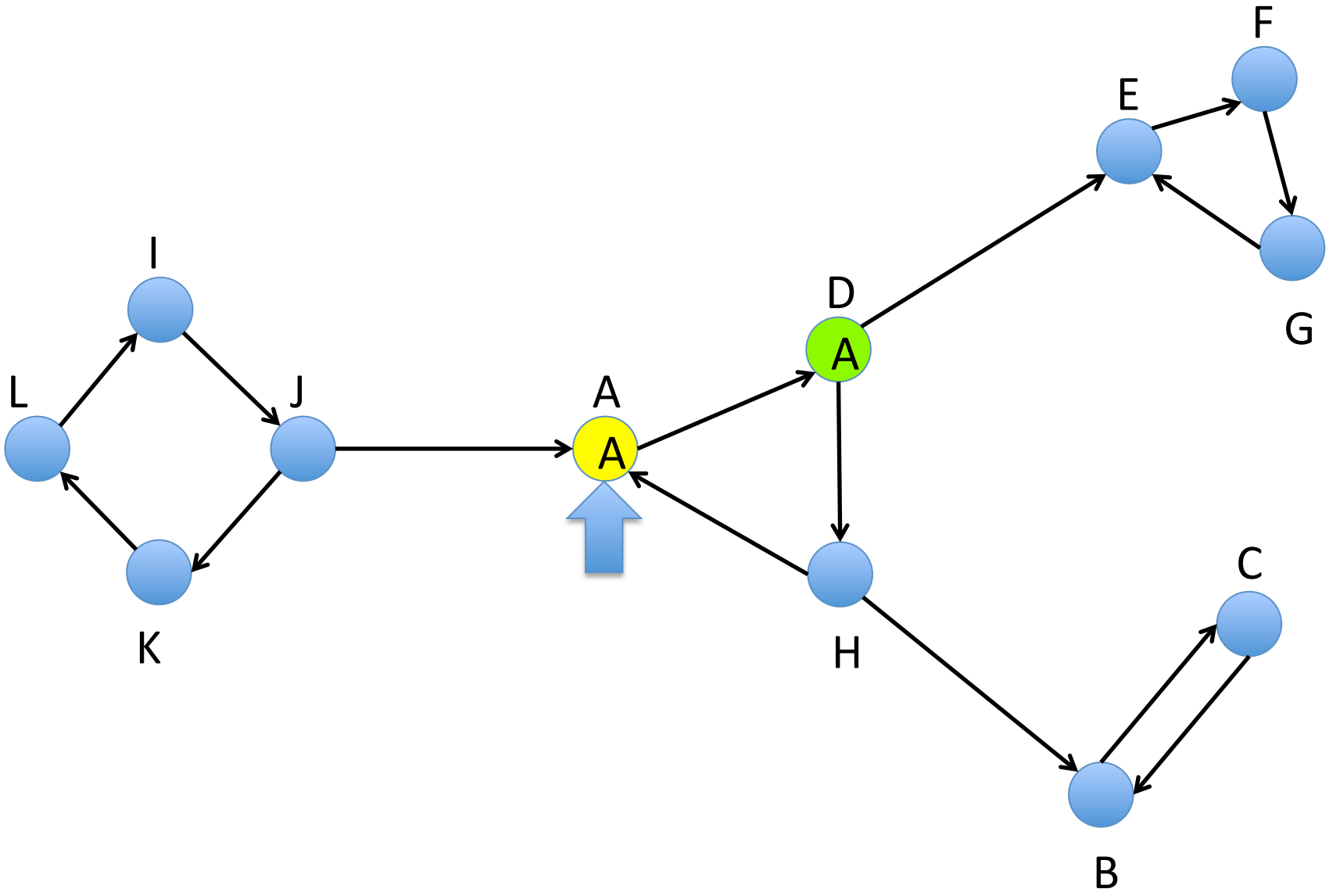
Can DFS/BFS Also Identify SCCs?



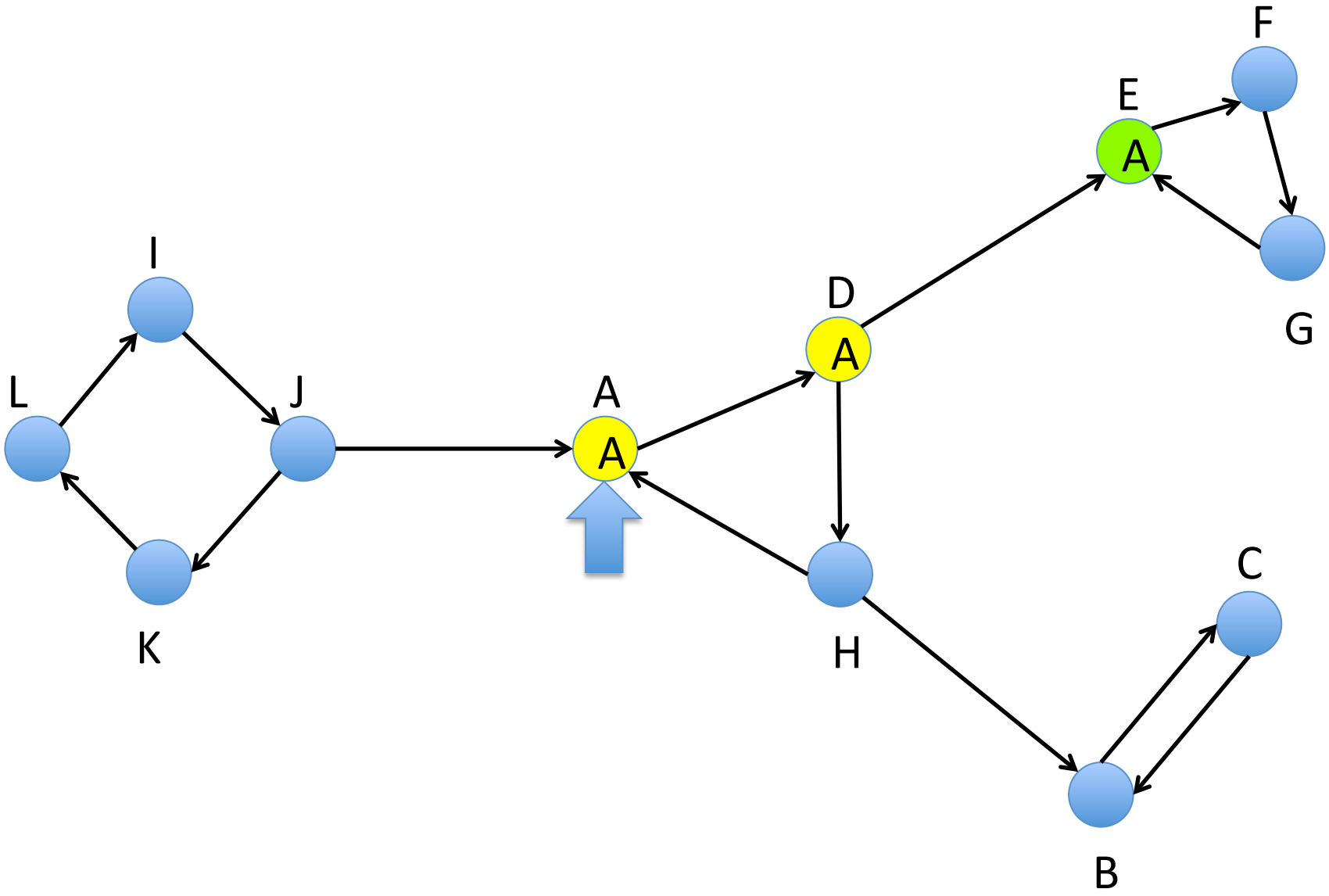
Can DFS/BFS Also Identify SCCs?



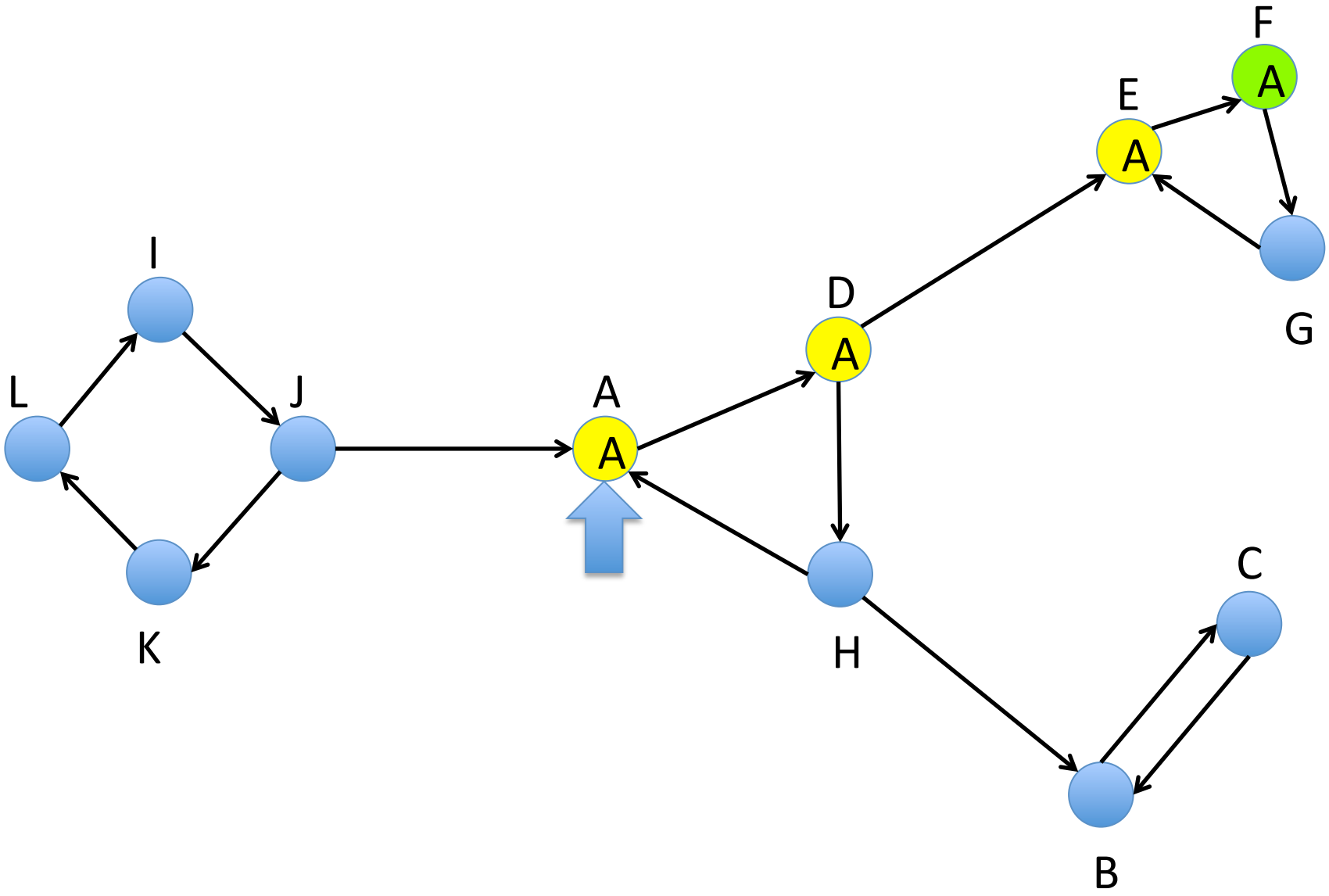
Can DFS/BFS Also Identify SCCs?



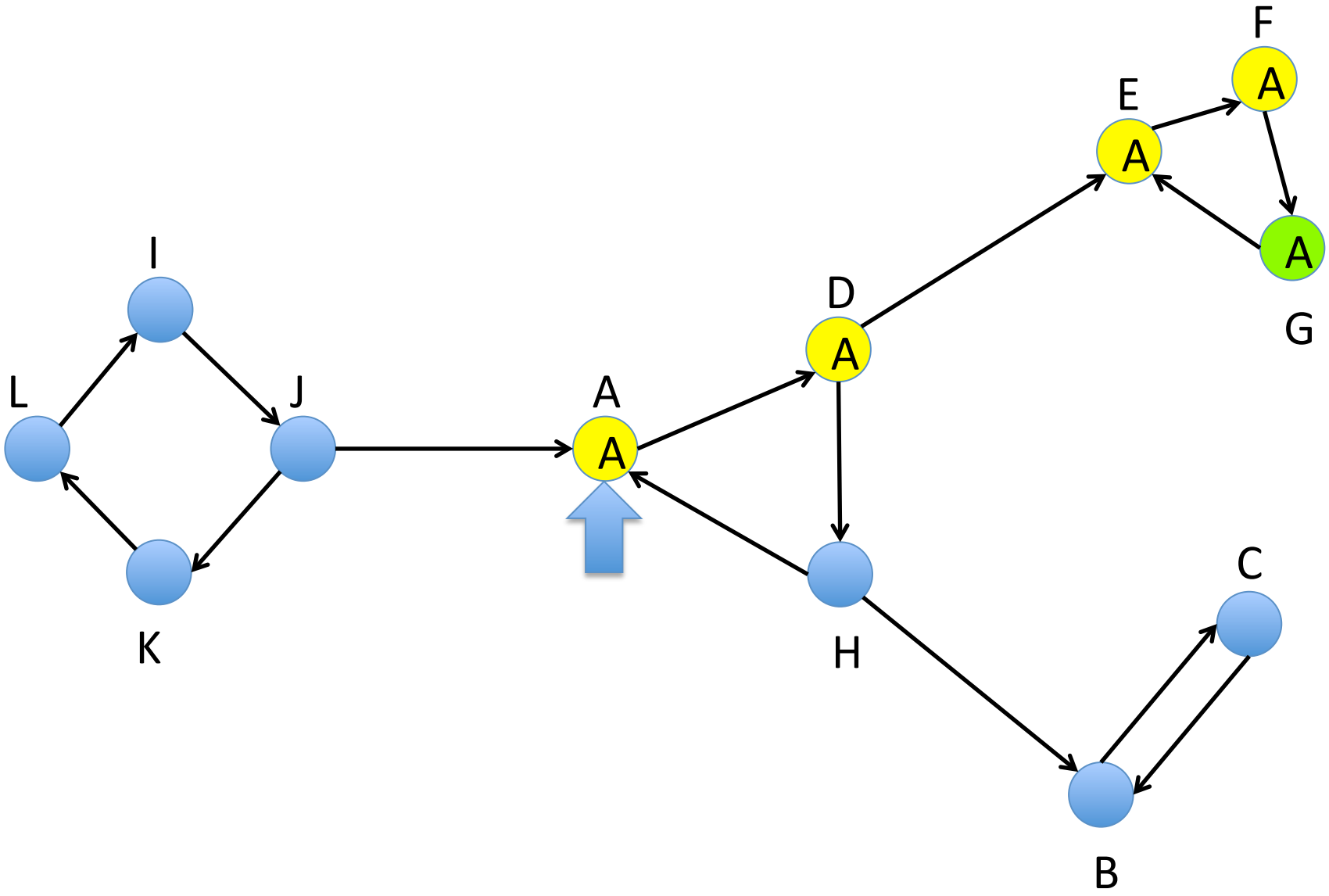
Can DFS/BFS Also Identify SCCs?



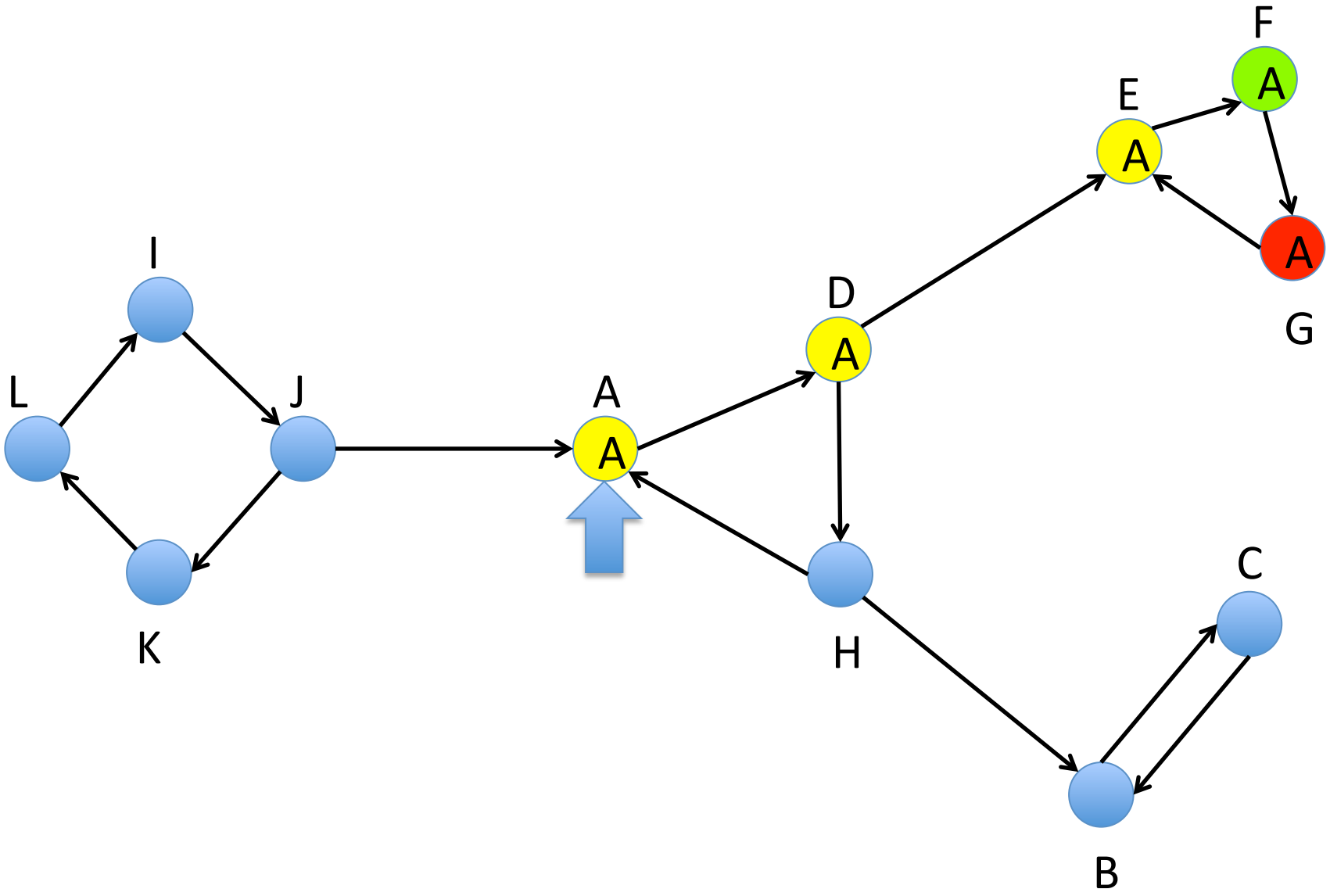
Can DFS/BFS Also Identify SCCs?



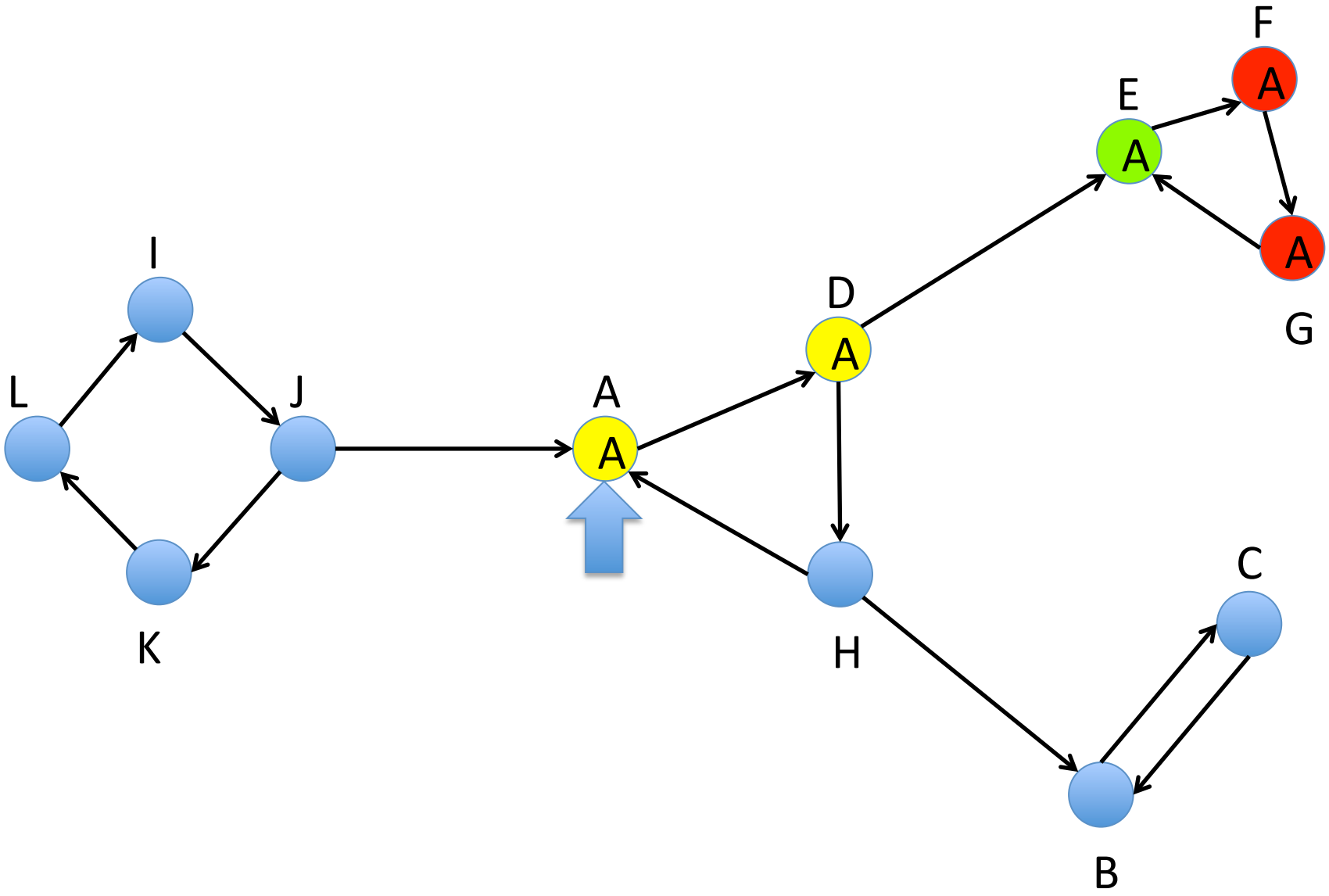
Can DFS/BFS Also Identify SCCs?



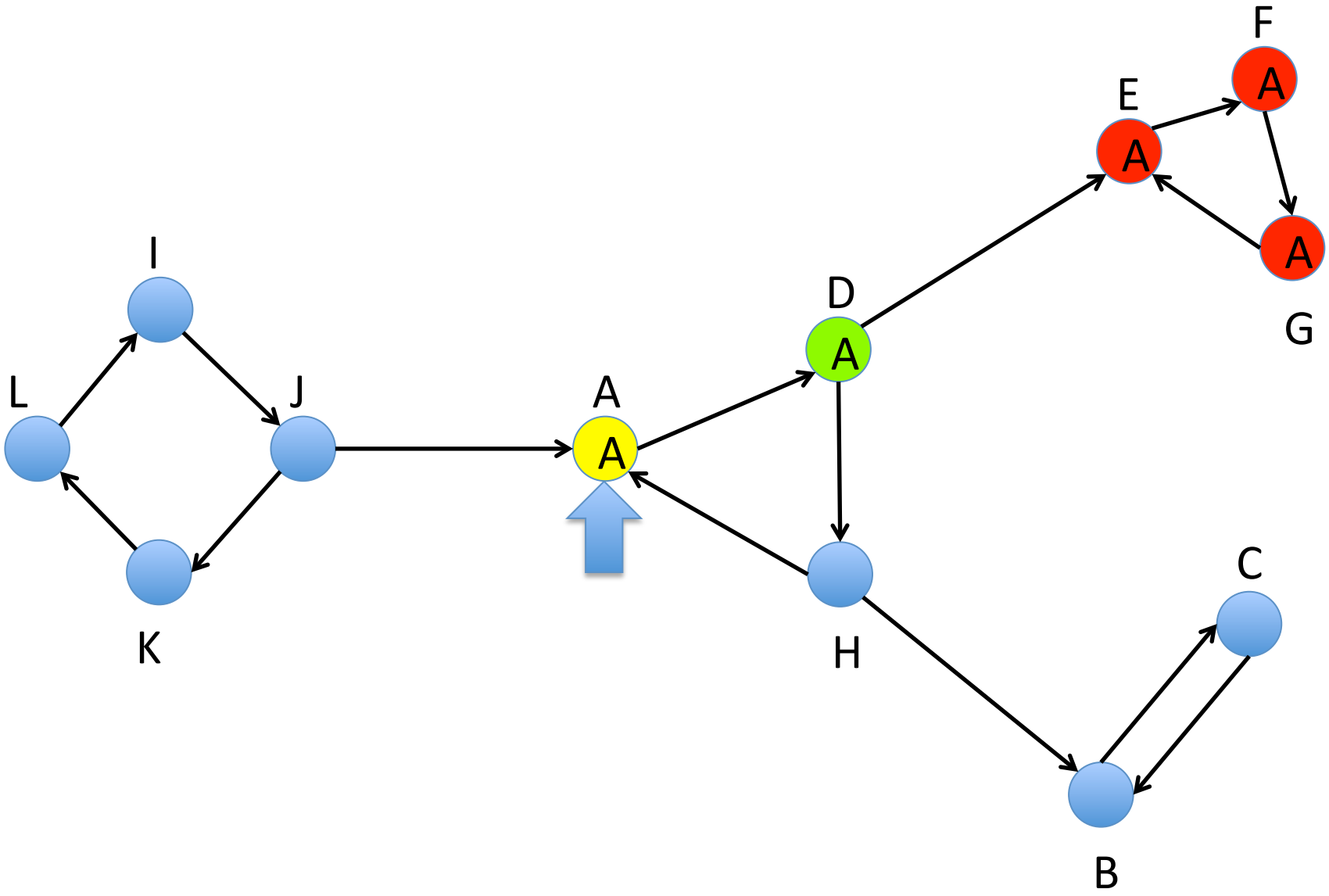
Can DFS/BFS Also Identify SCCs?



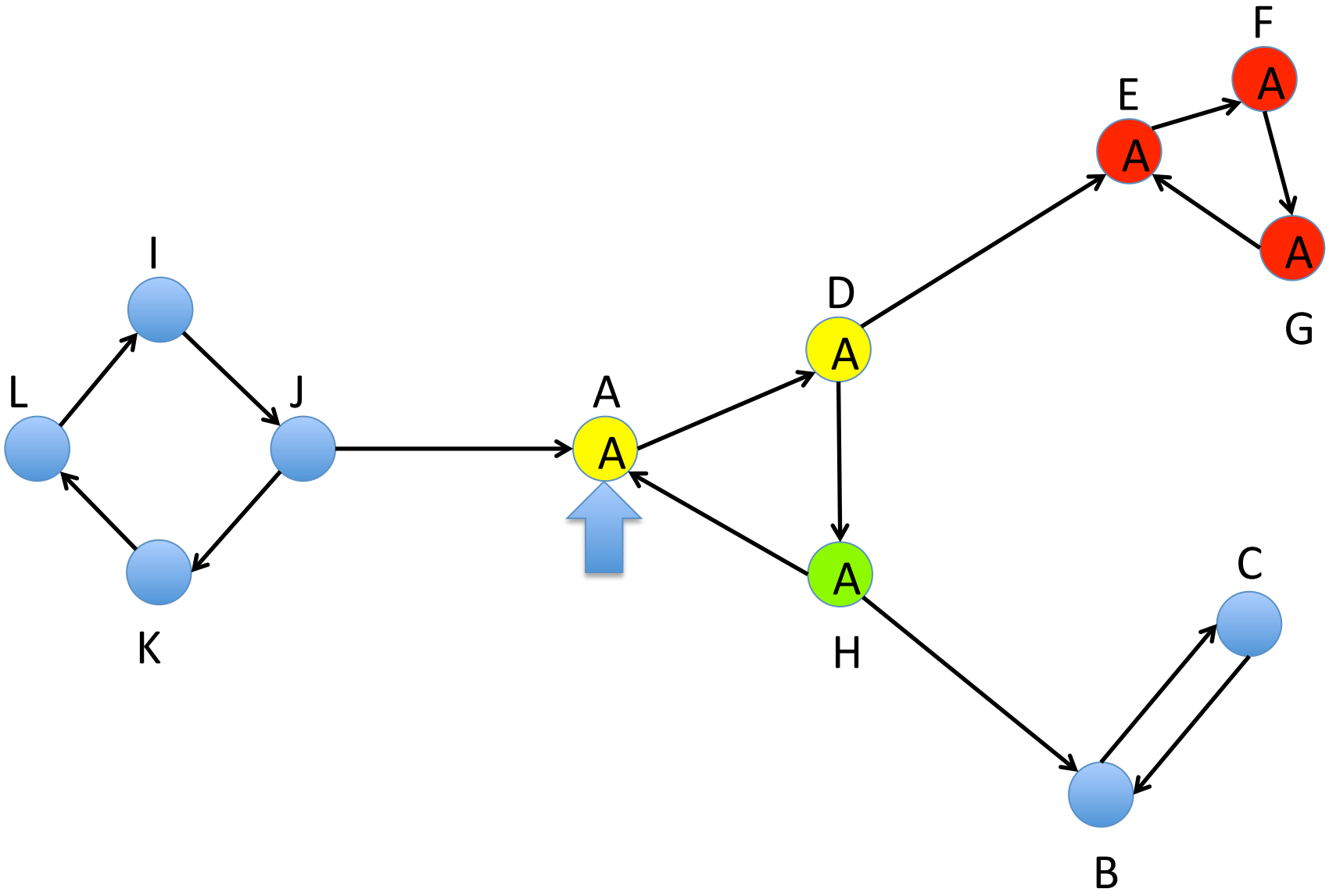
Can DFS/BFS Also Identify SCCs?



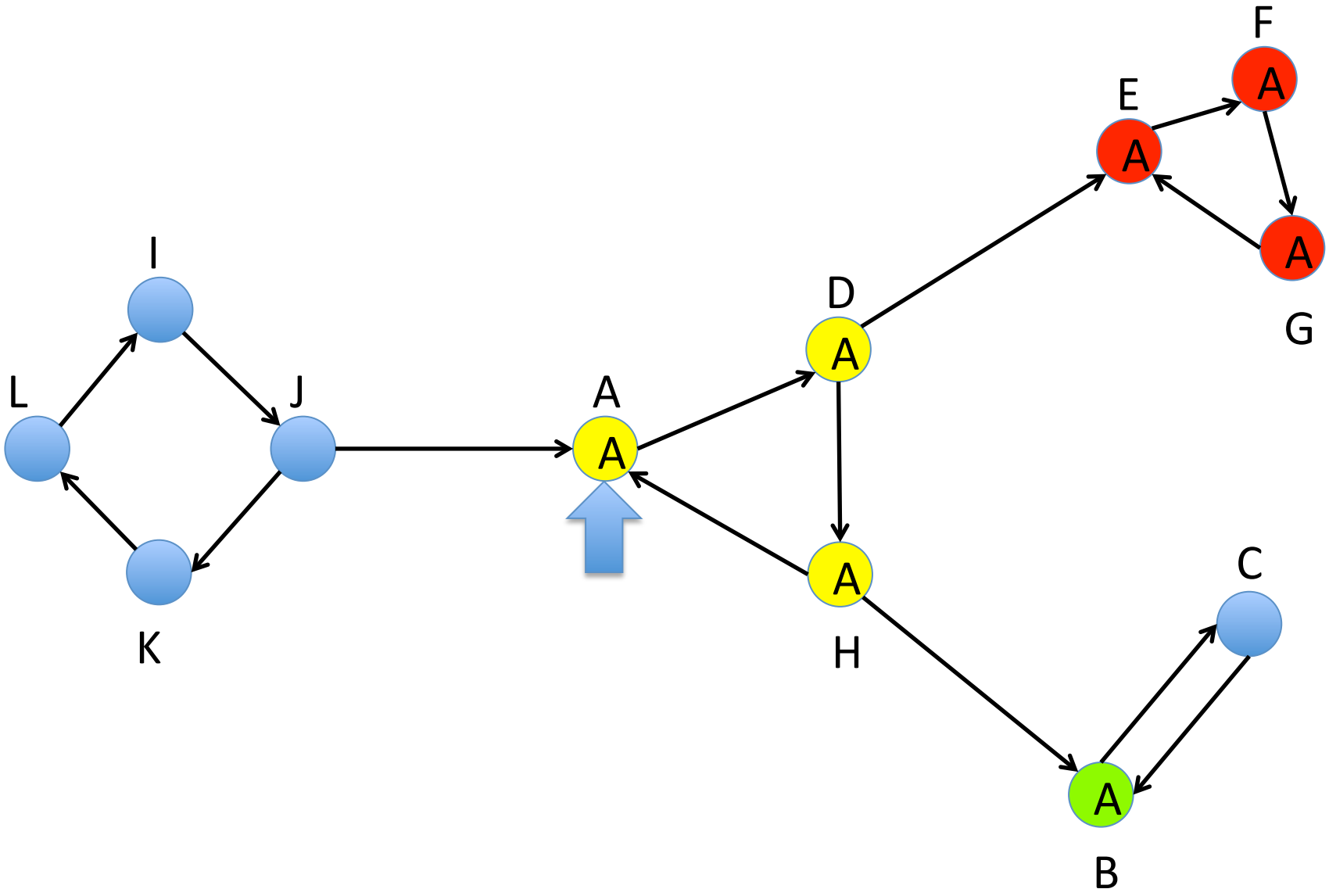
Can DFS/BFS Also Identify SCCs?



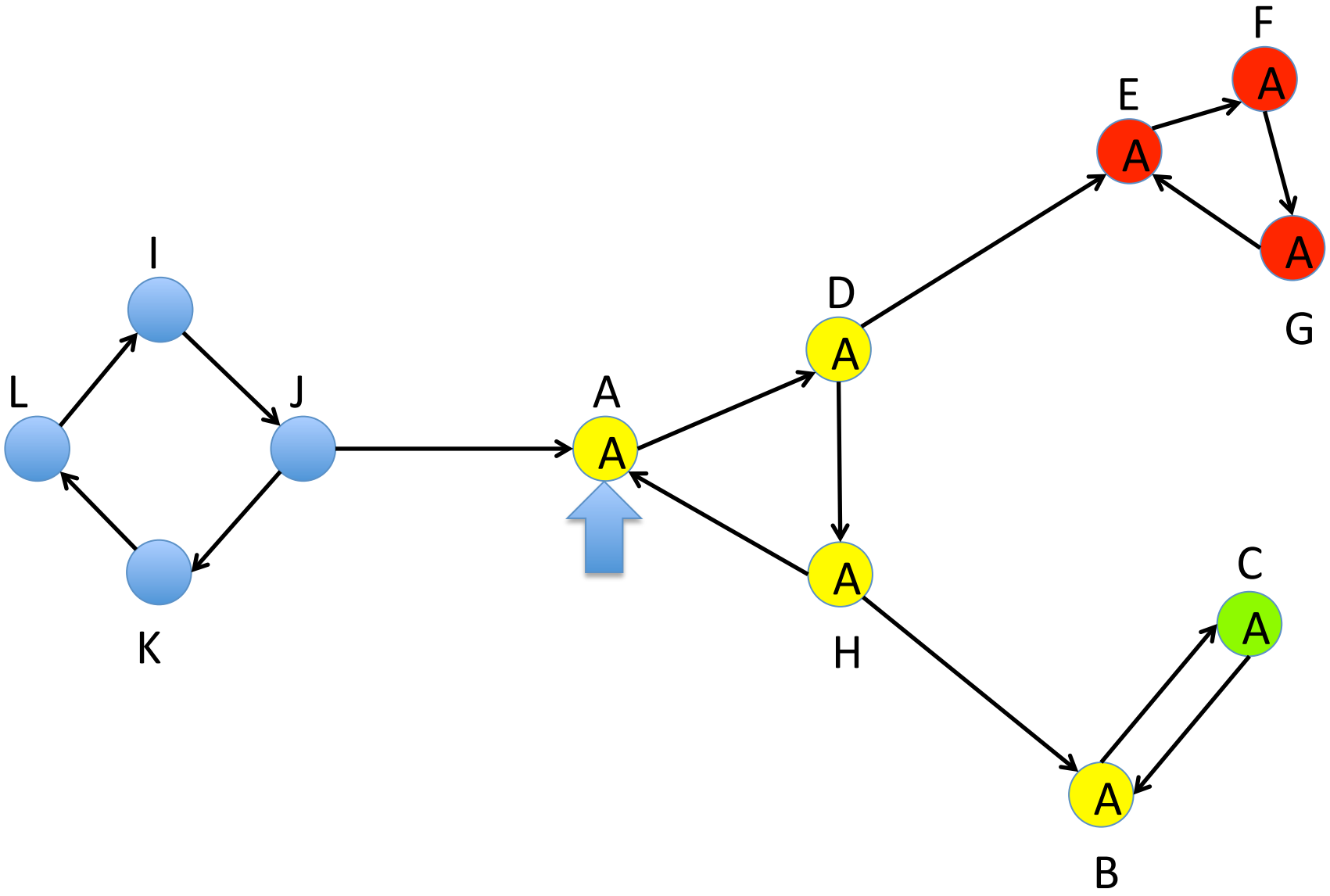
Can DFS/BFS Also Identify SCCs?



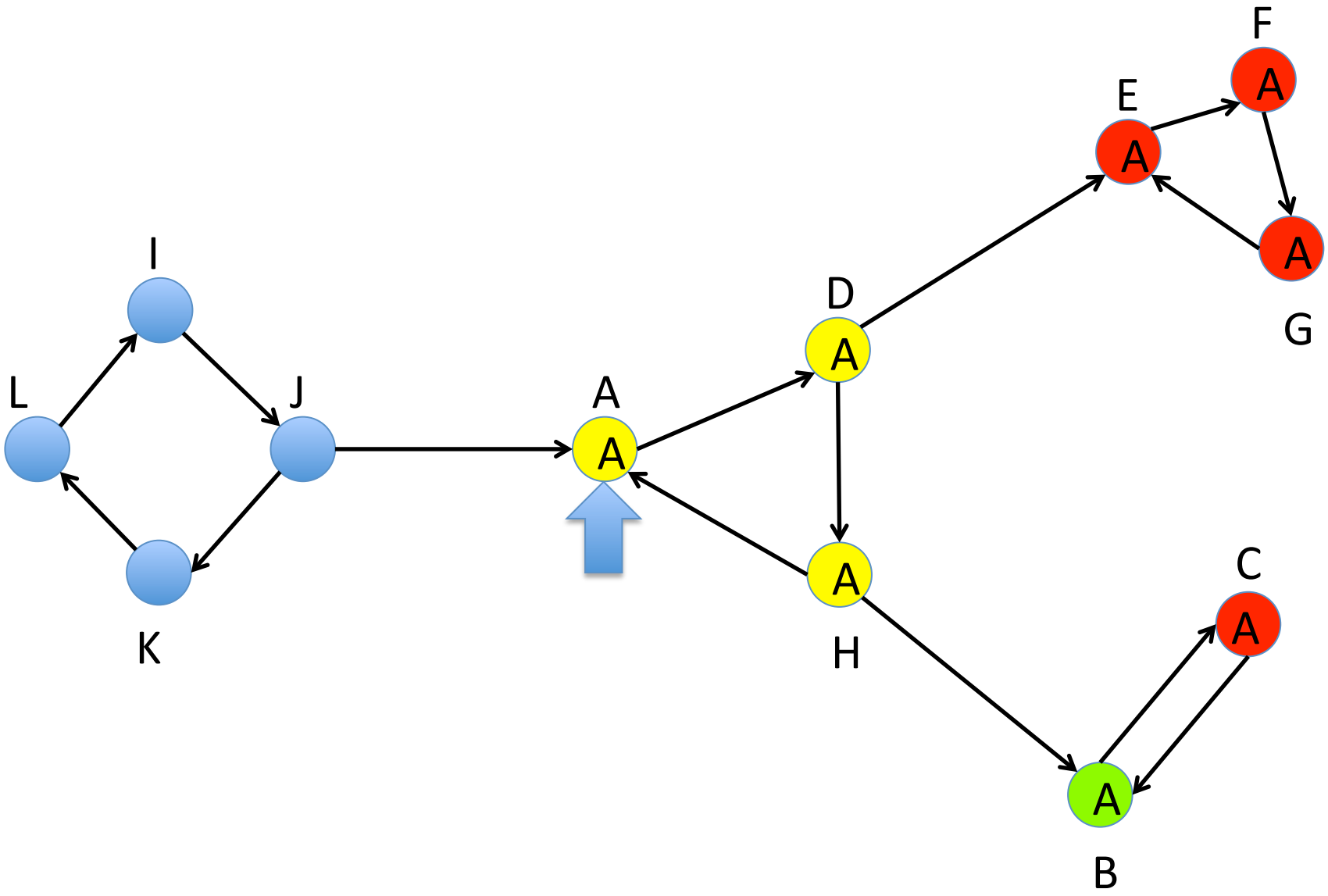
Can DFS/BFS Also Identify SCCs?



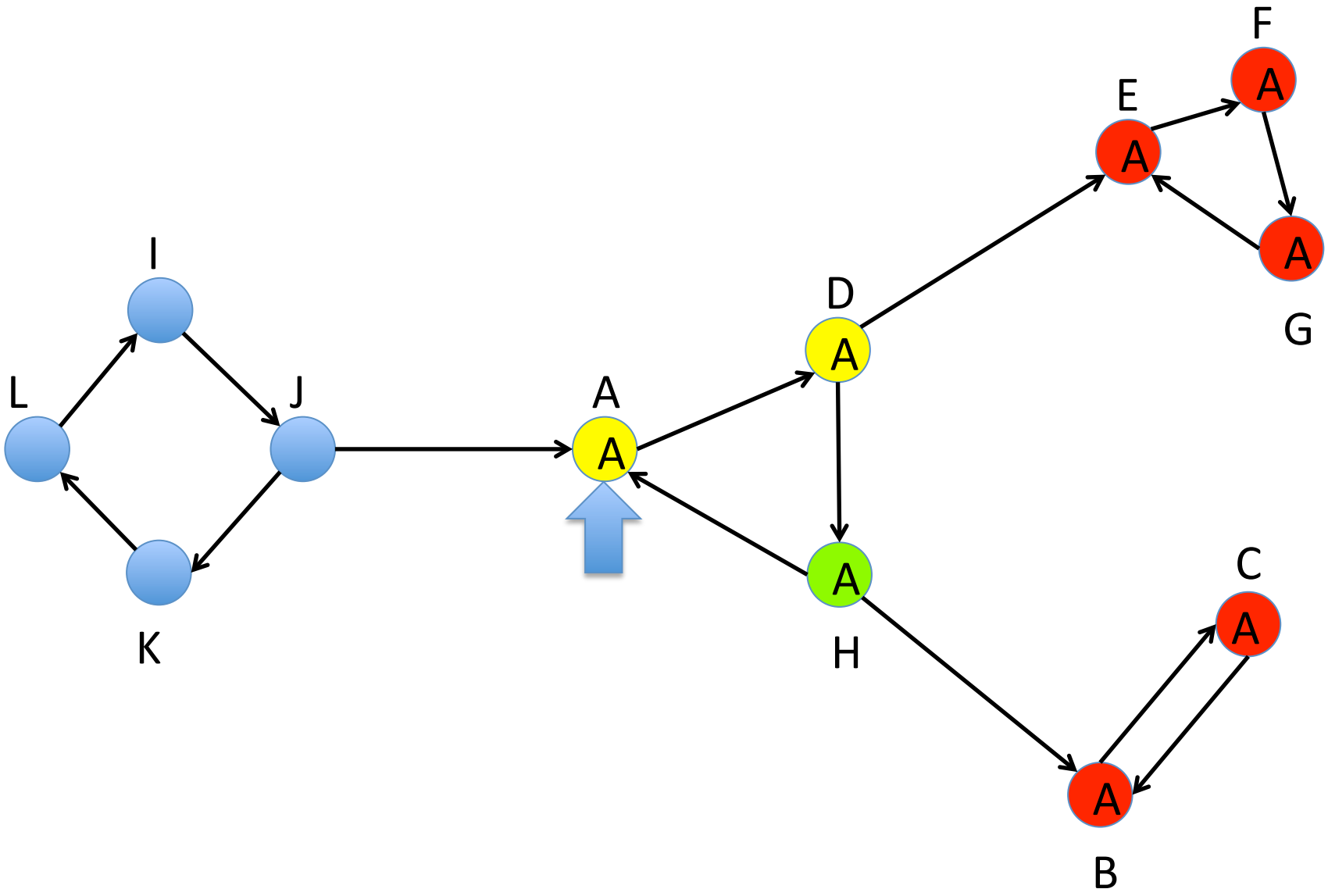
Can DFS/BFS Also Identify SCCs?



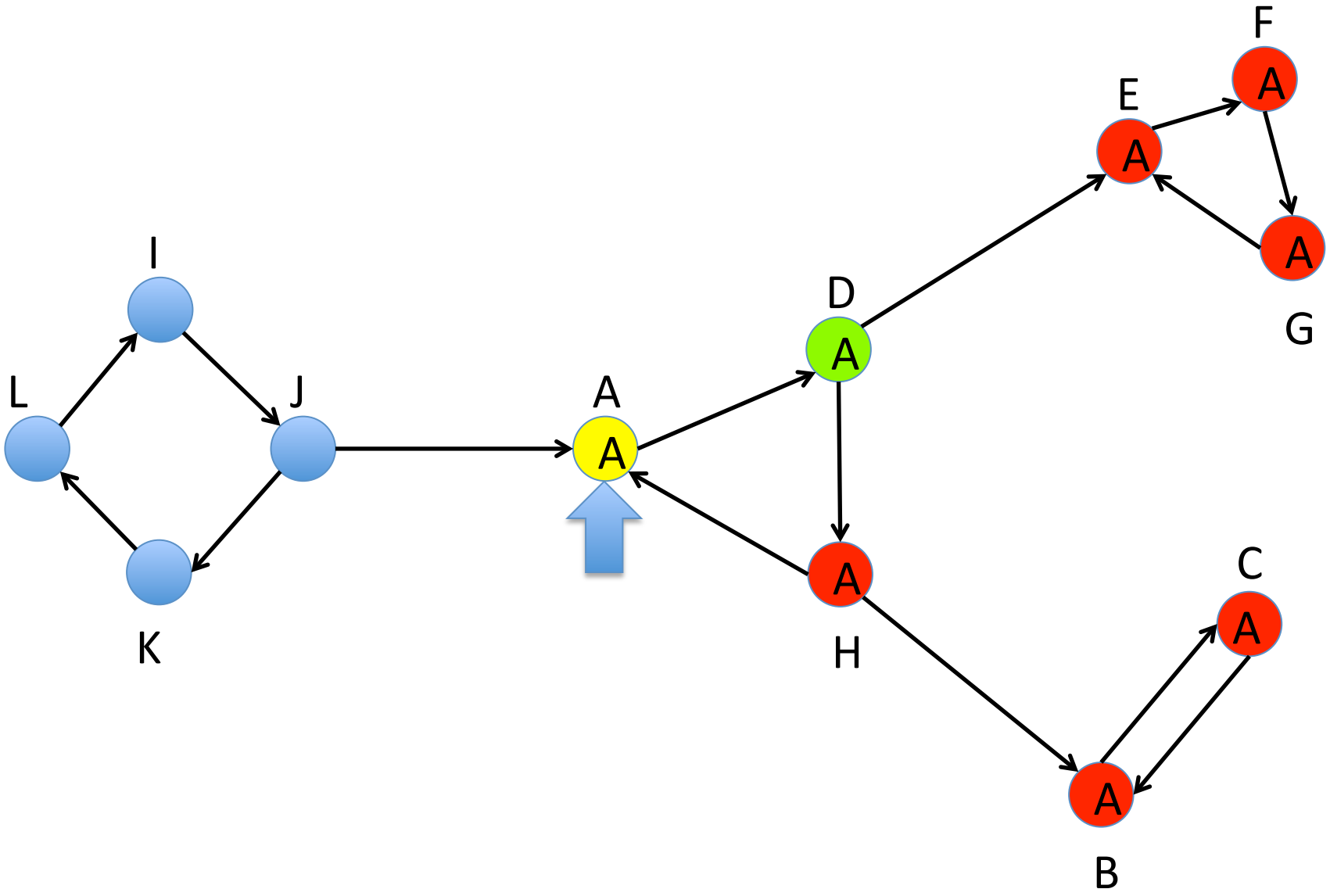
Can DFS/BFS Also Identify SCCs?



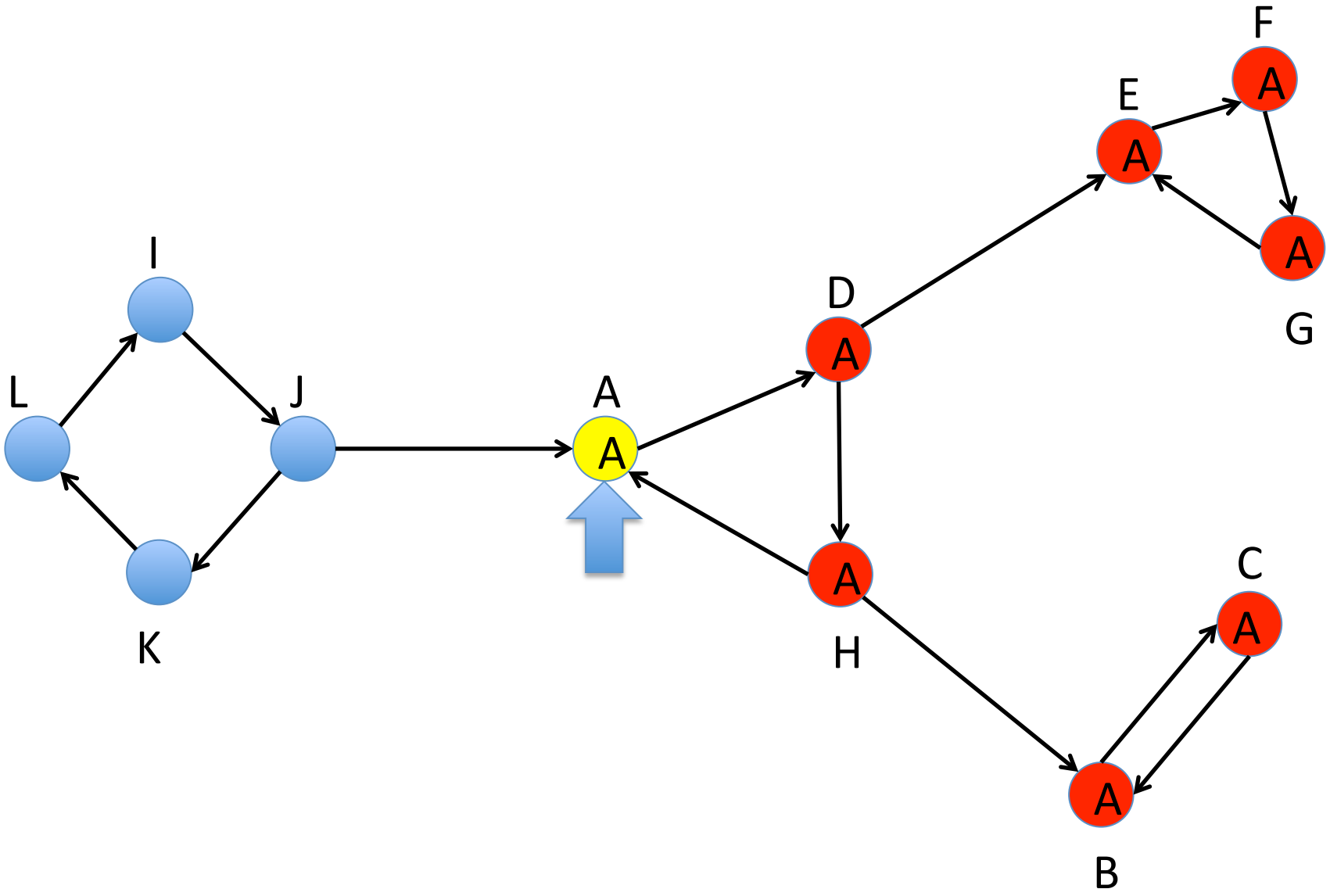
Can DFS/BFS Also Identify SCCs?



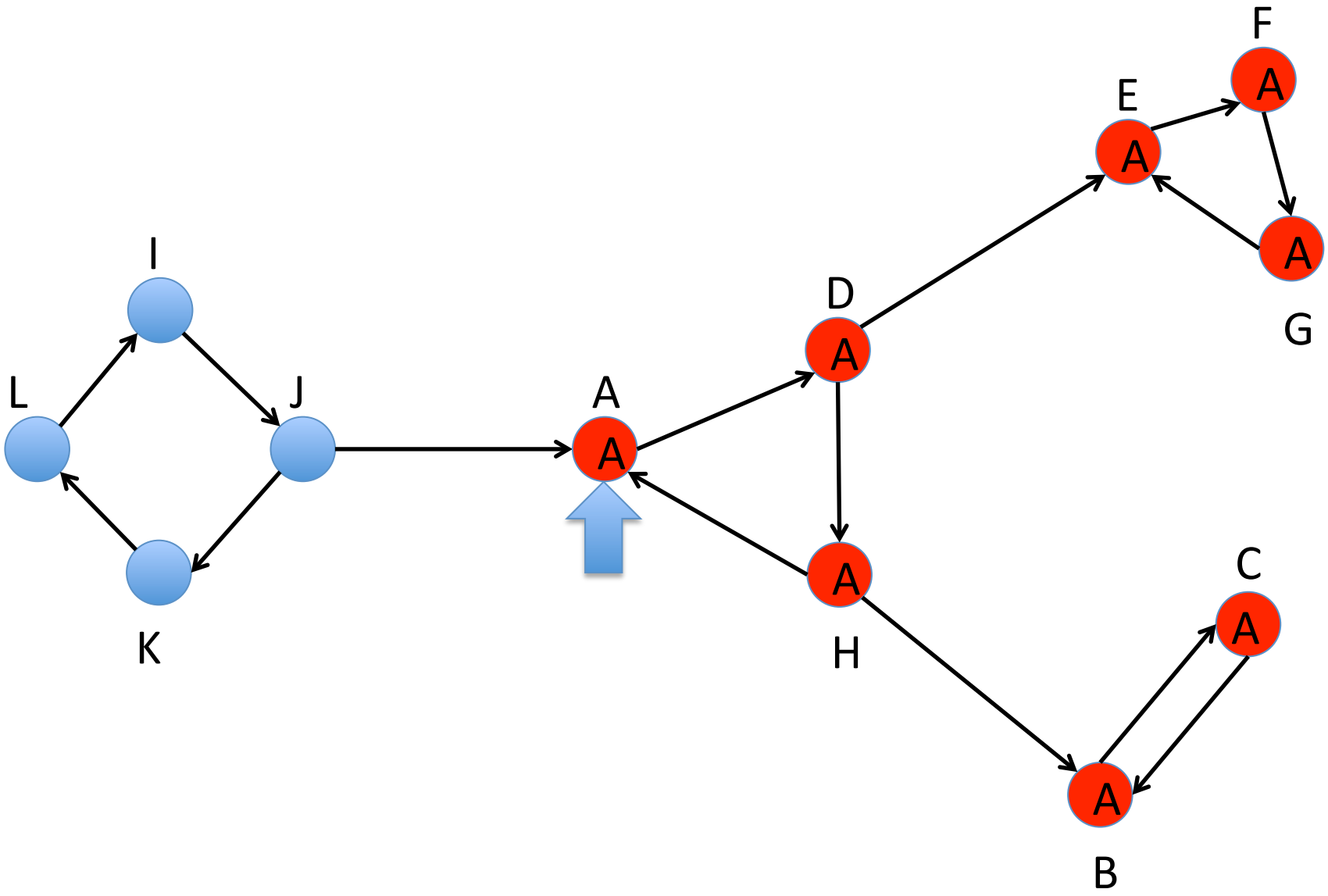
Can DFS/BFS Also Identify SCCs?



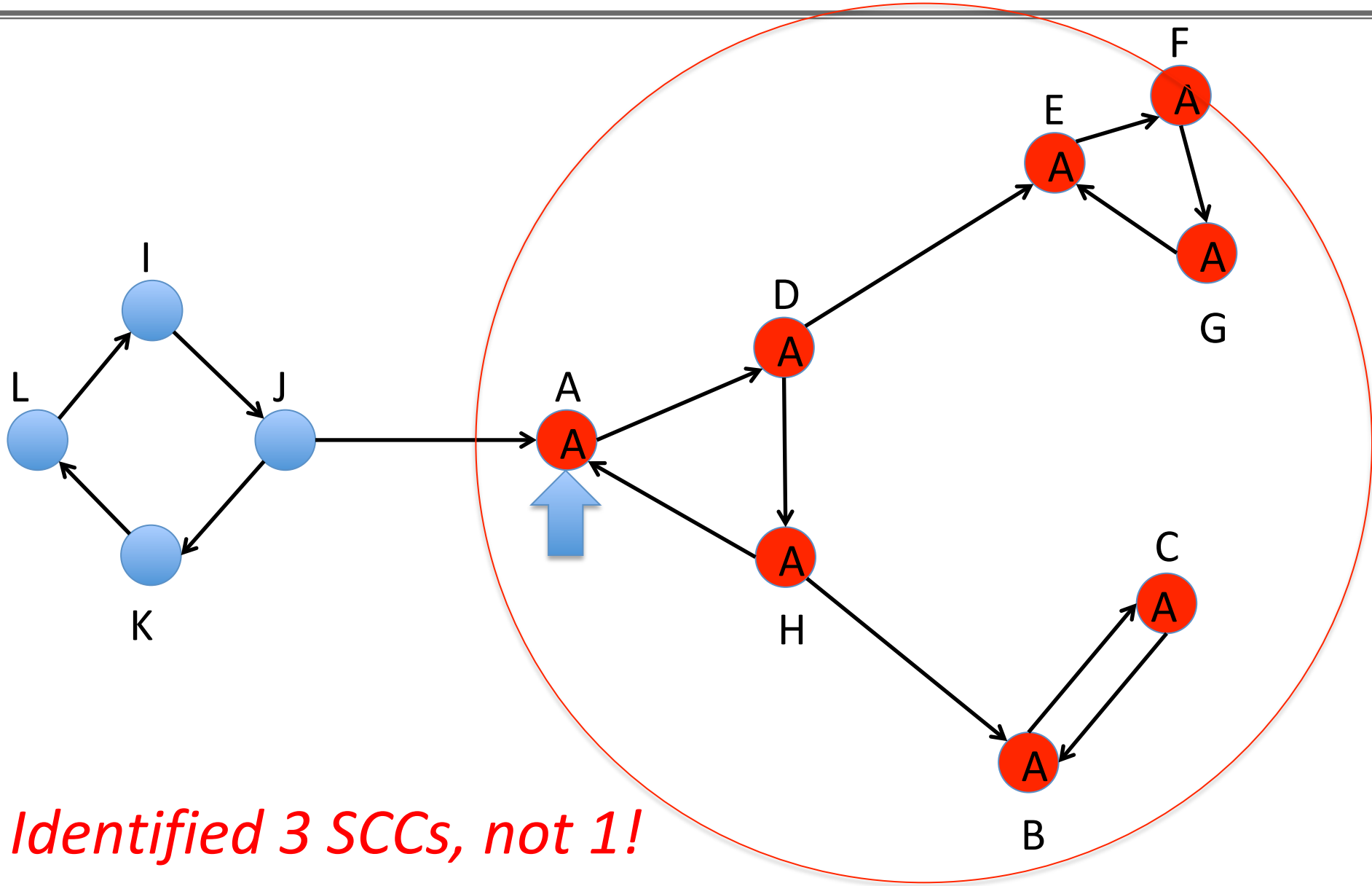
Can DFS/BFS Also Identify SCCs?



Can DFS/BFS Also Identify SCCs?

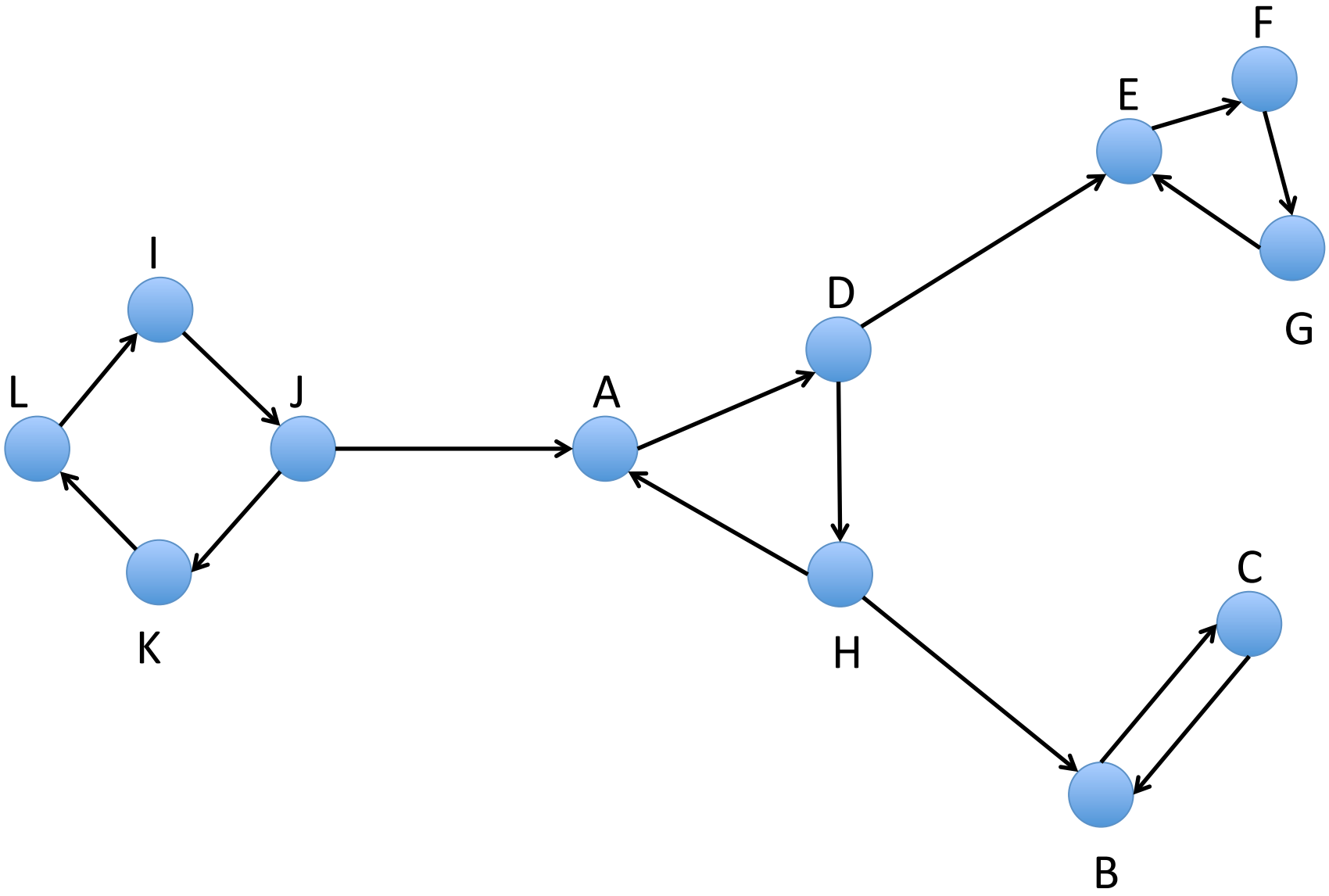


Can DFS/BFS Also Identify SCCs?

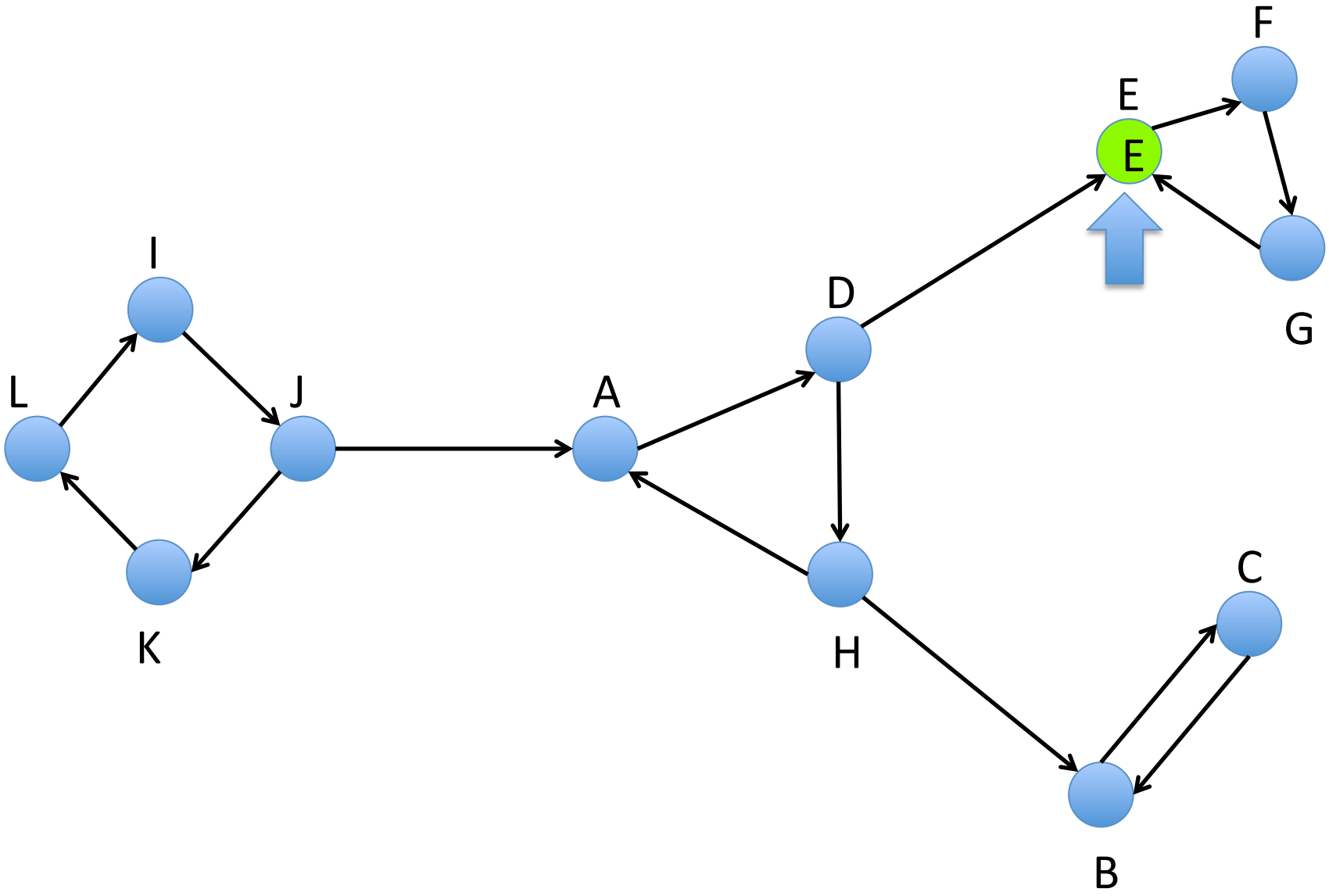


Identified 3 SCCs, not 1!

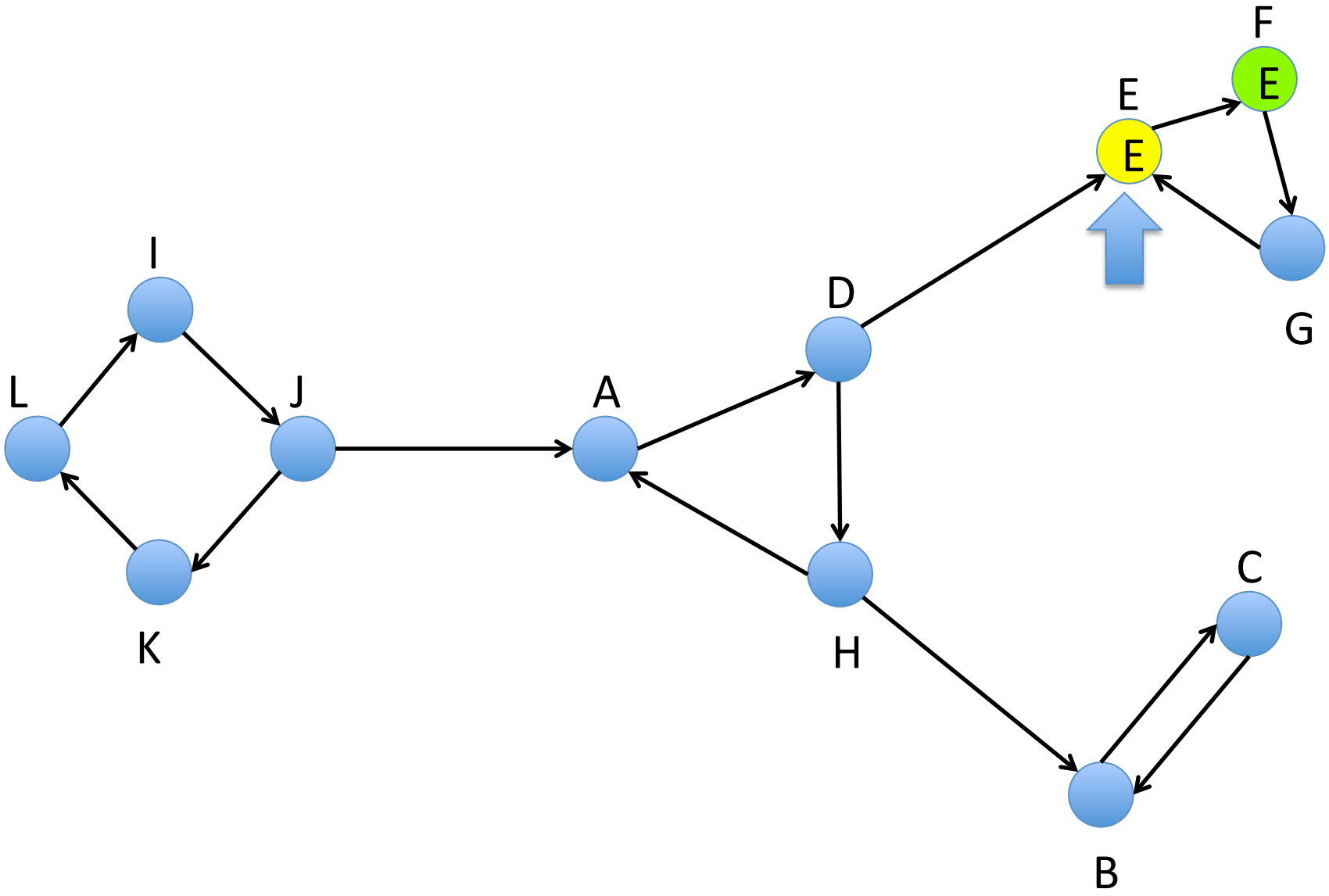
What If We Got Very Lucky?



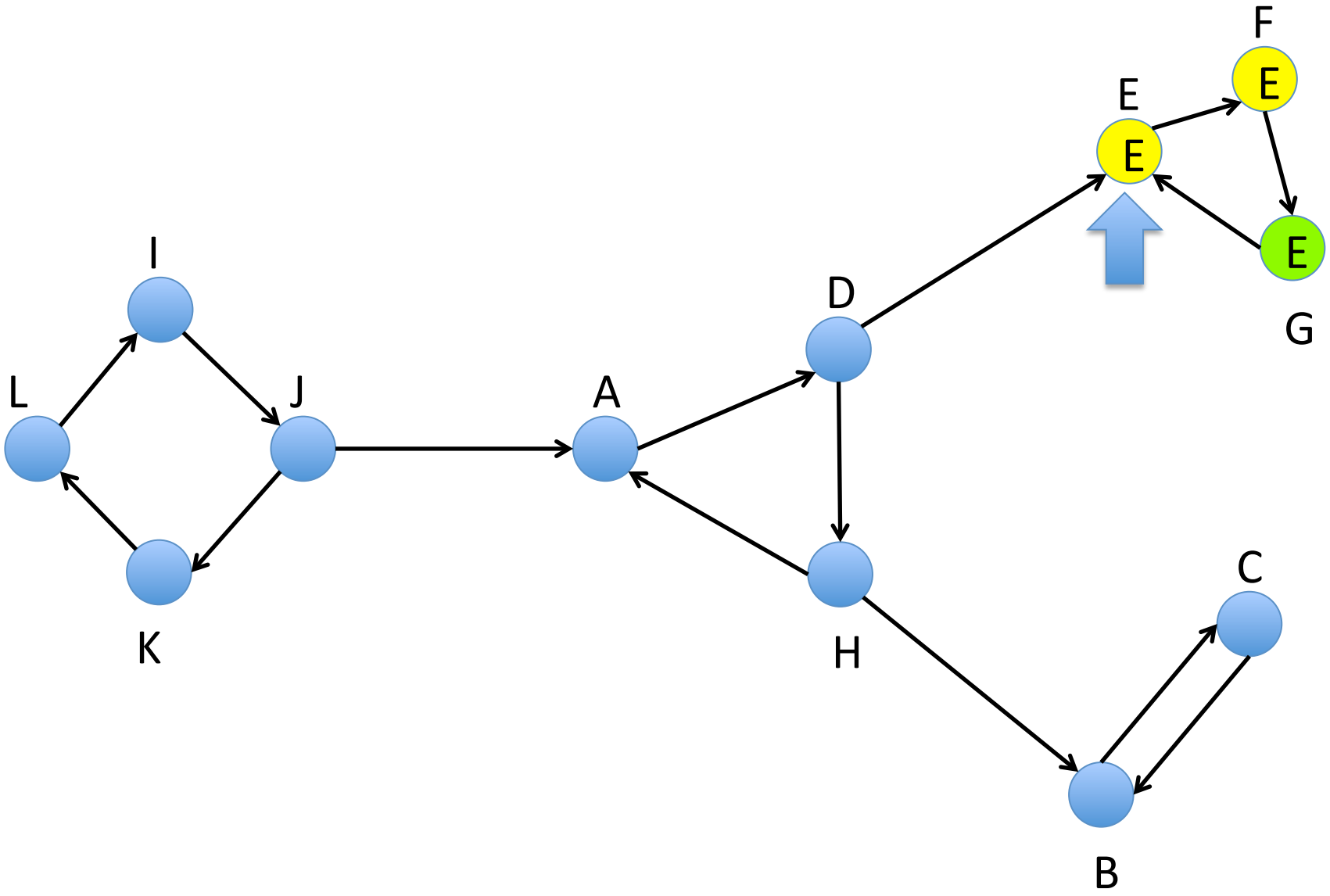
What If We Got Very Lucky?



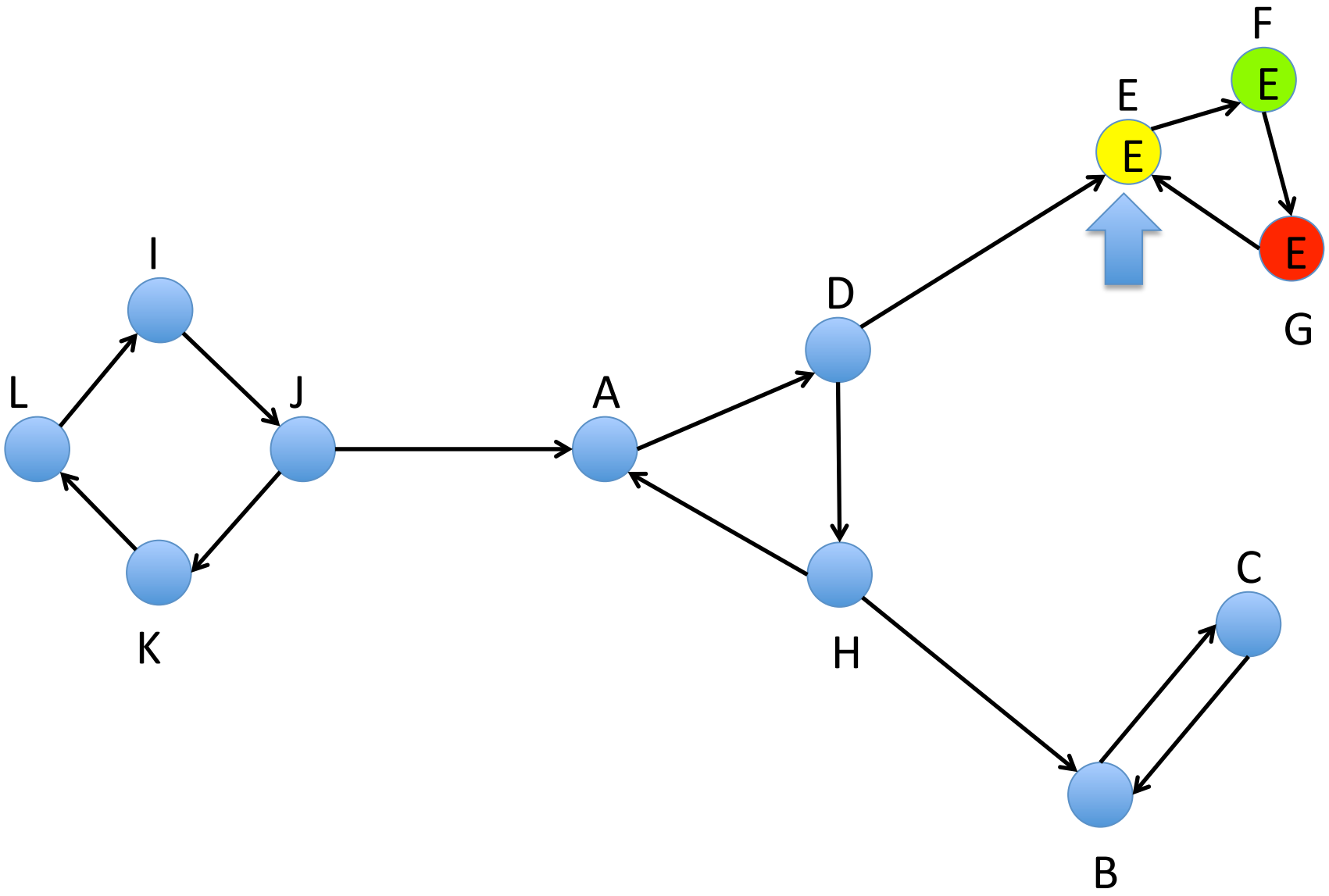
What If We Got Very Lucky?



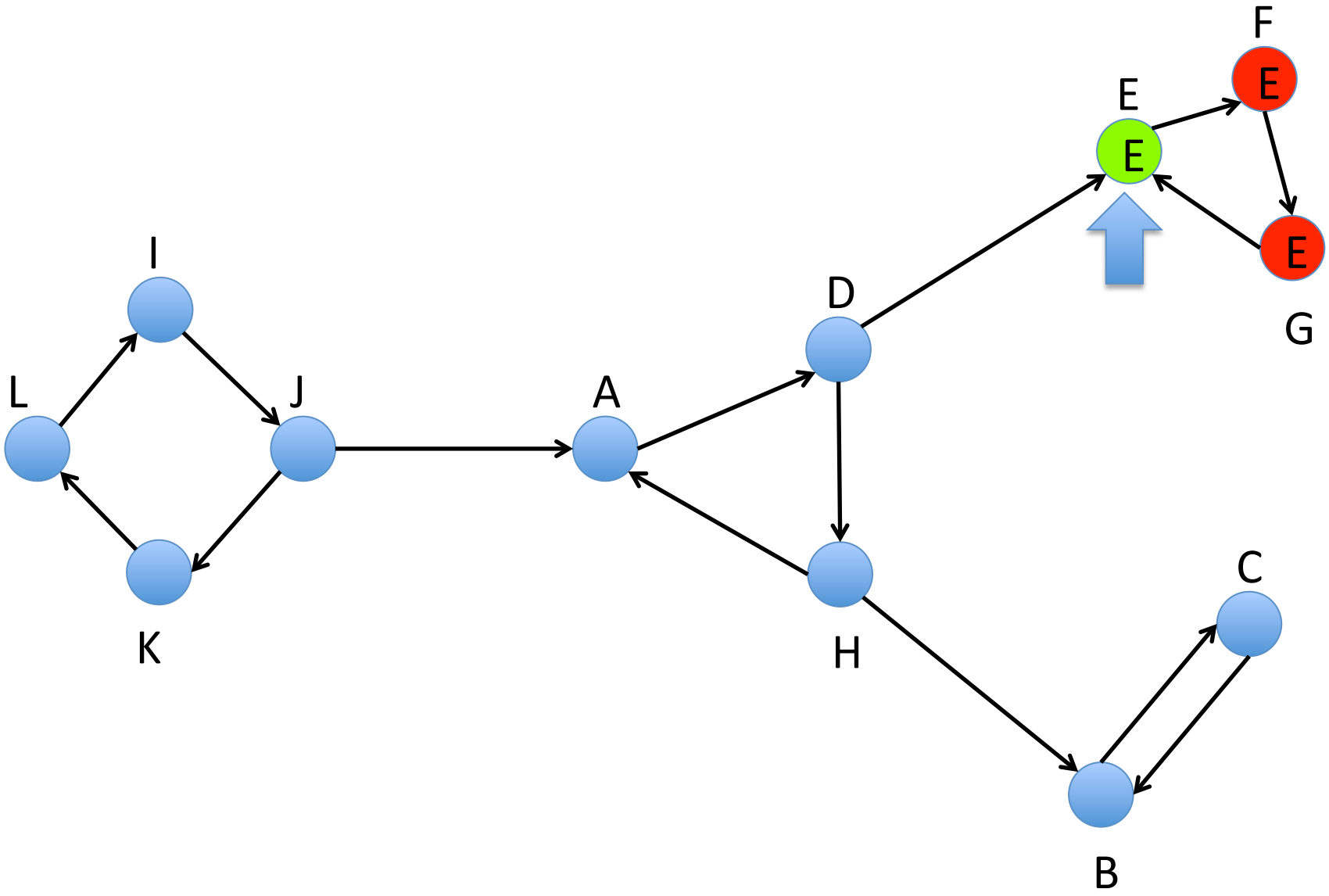
What If We Got Very Lucky?



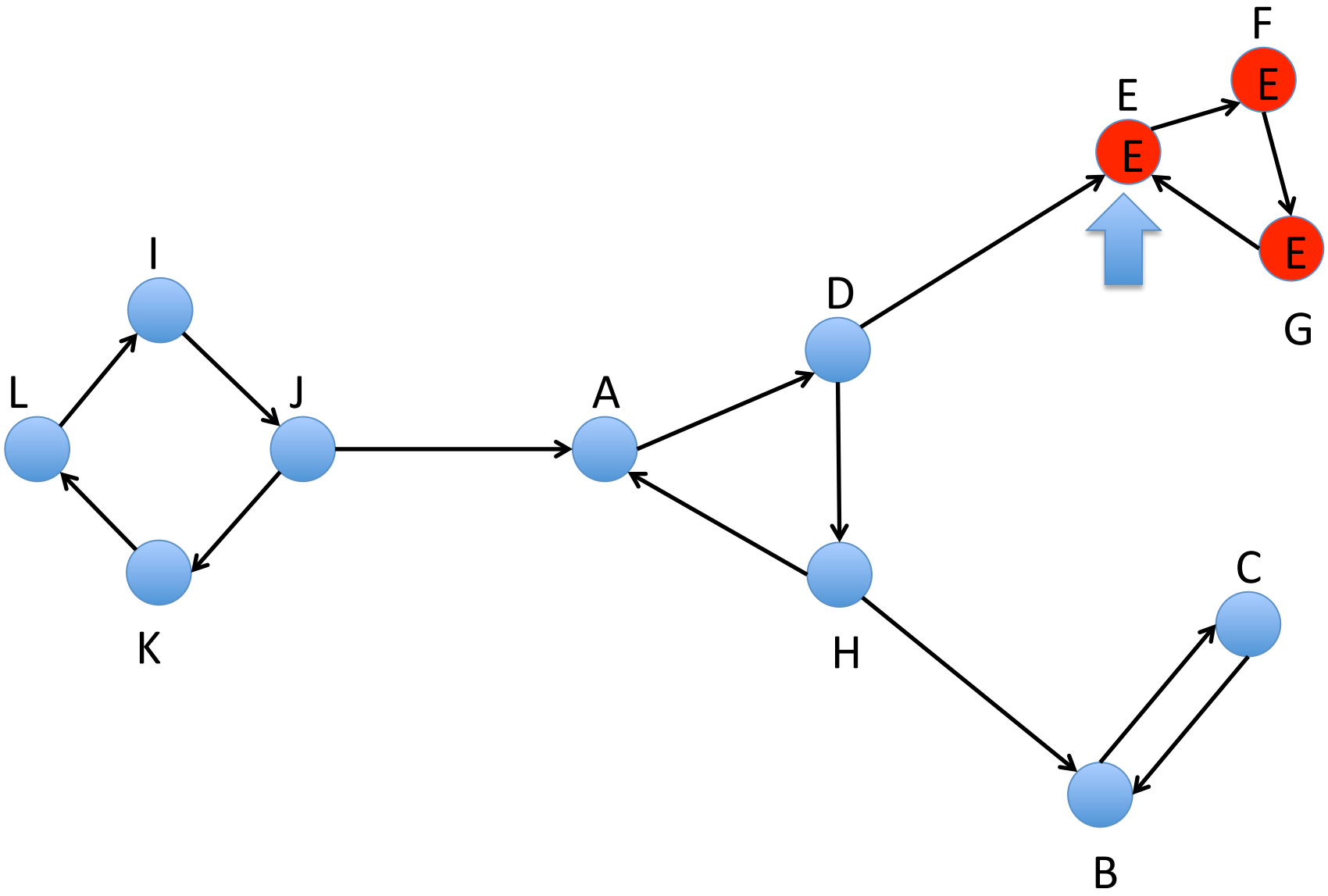
What If We Got Very Lucky?



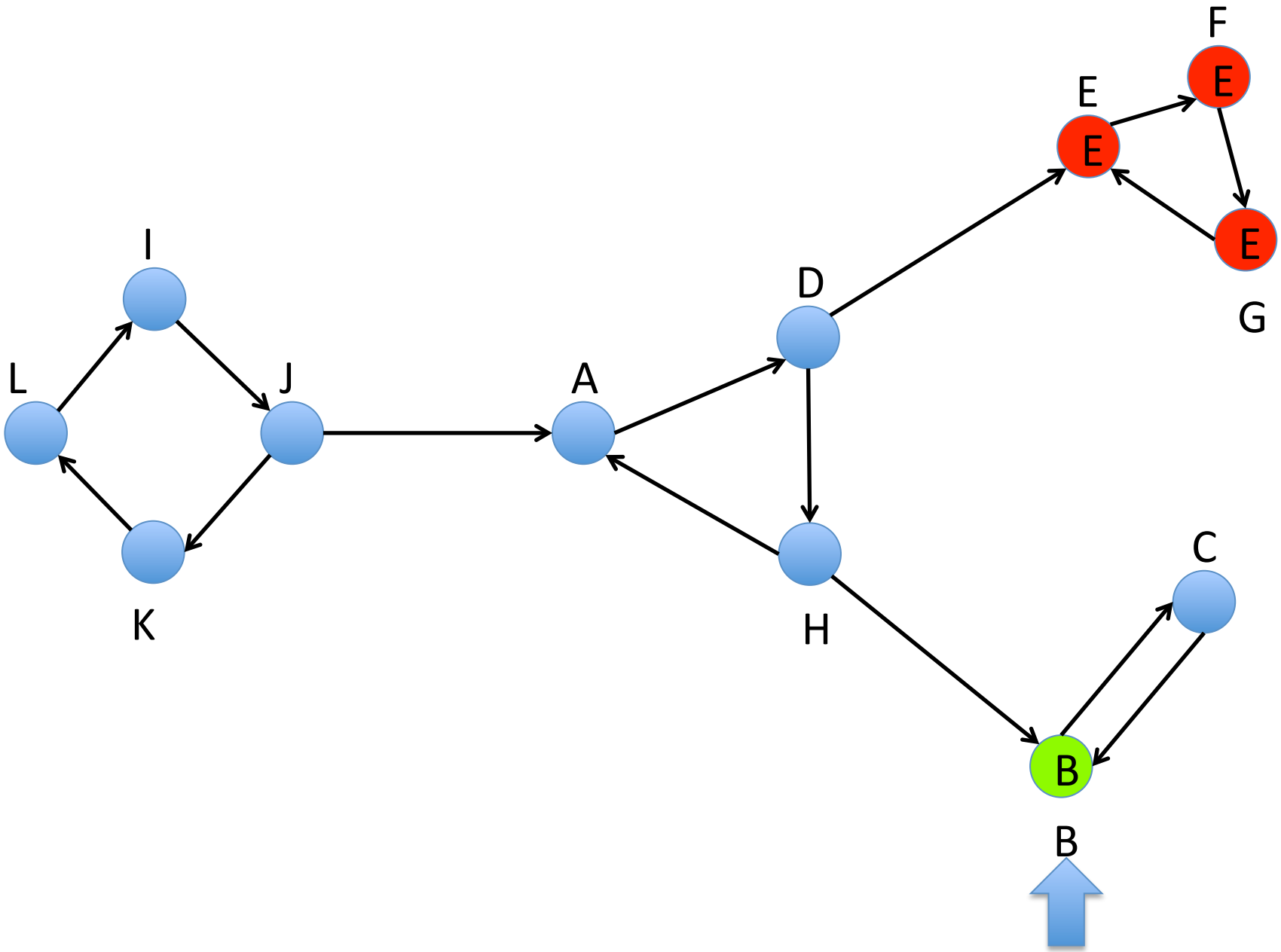
What If We Got Very Lucky?



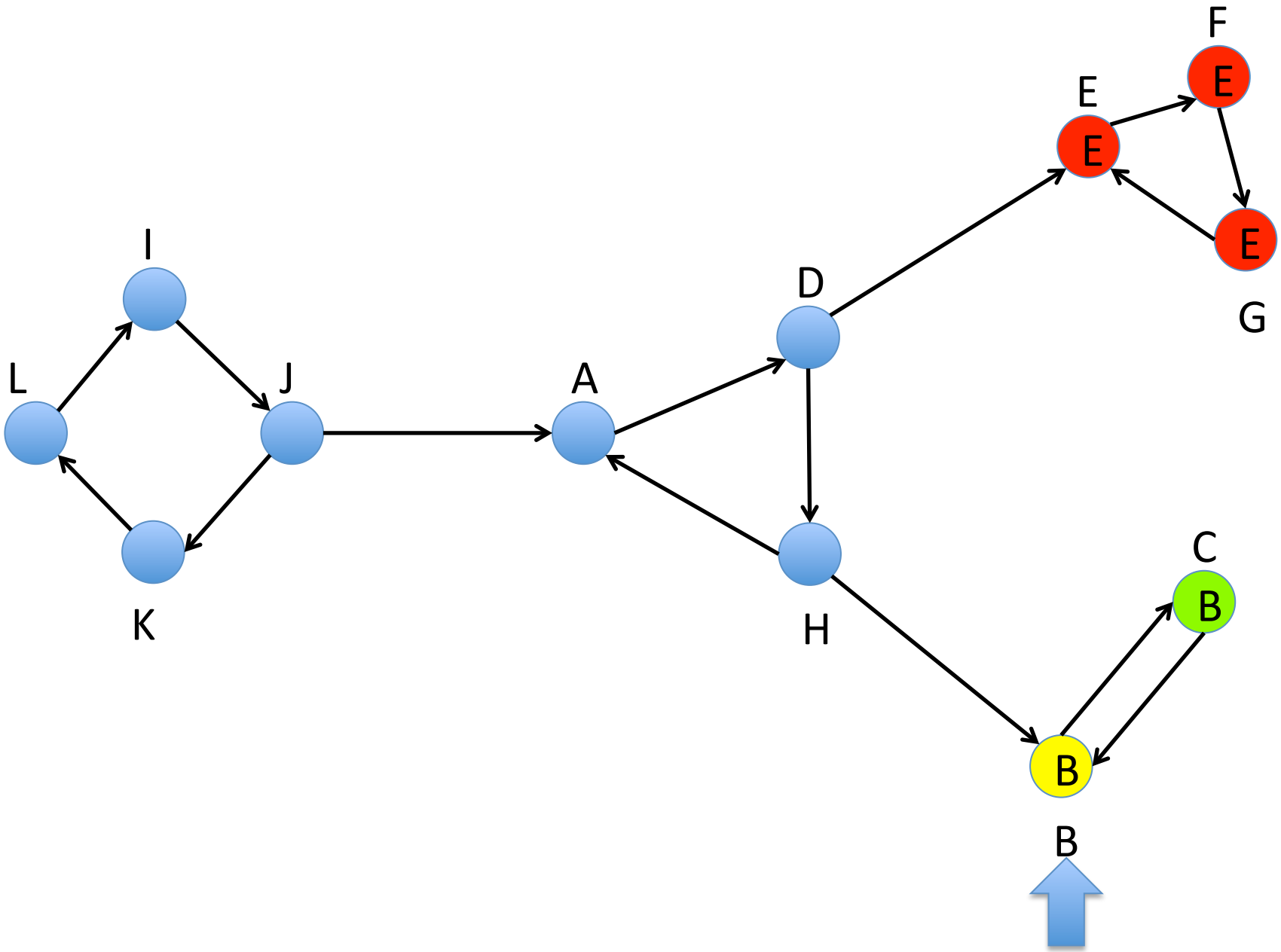
What If We Got Very Lucky?



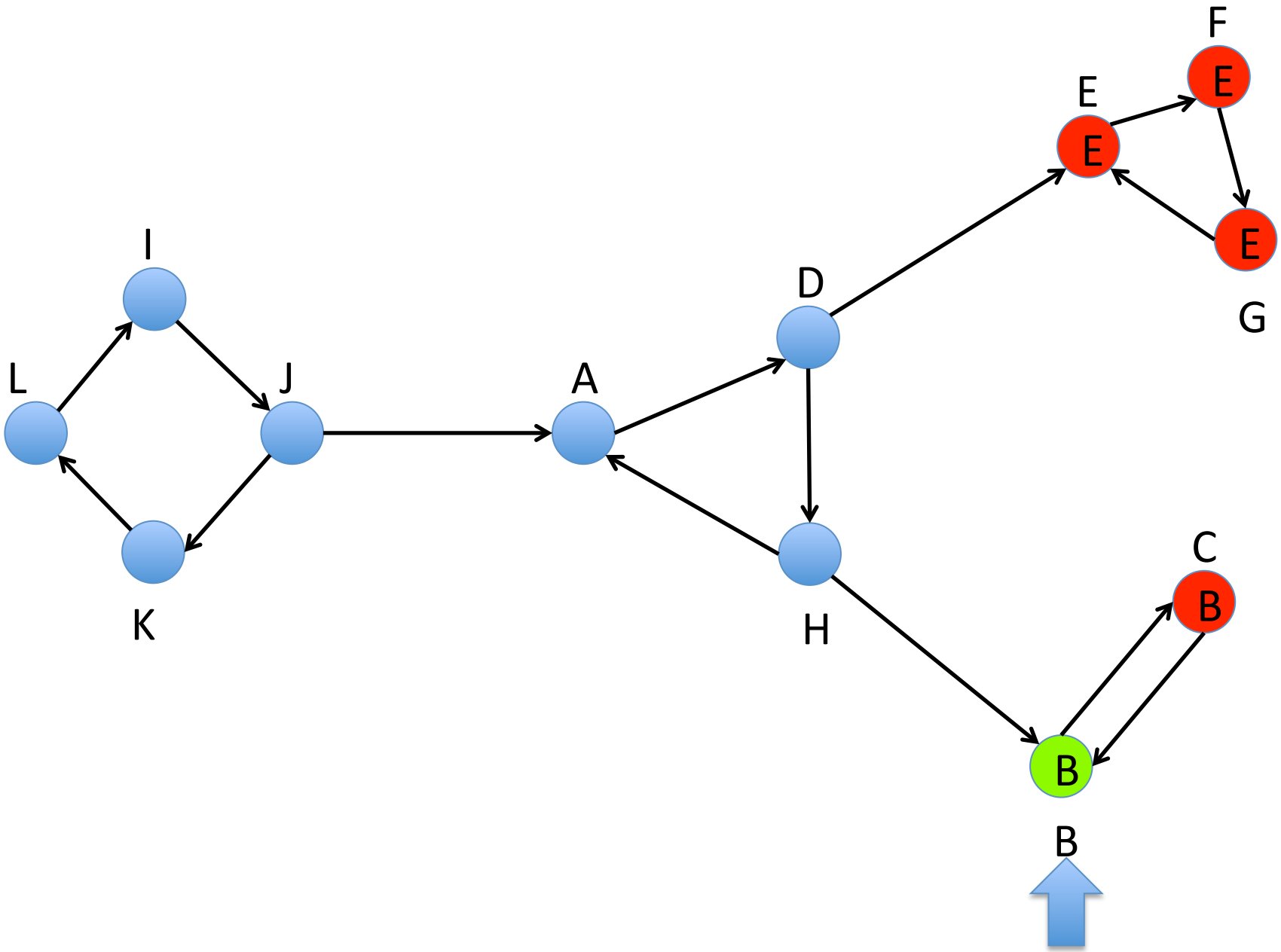
What If We Got Very Lucky?



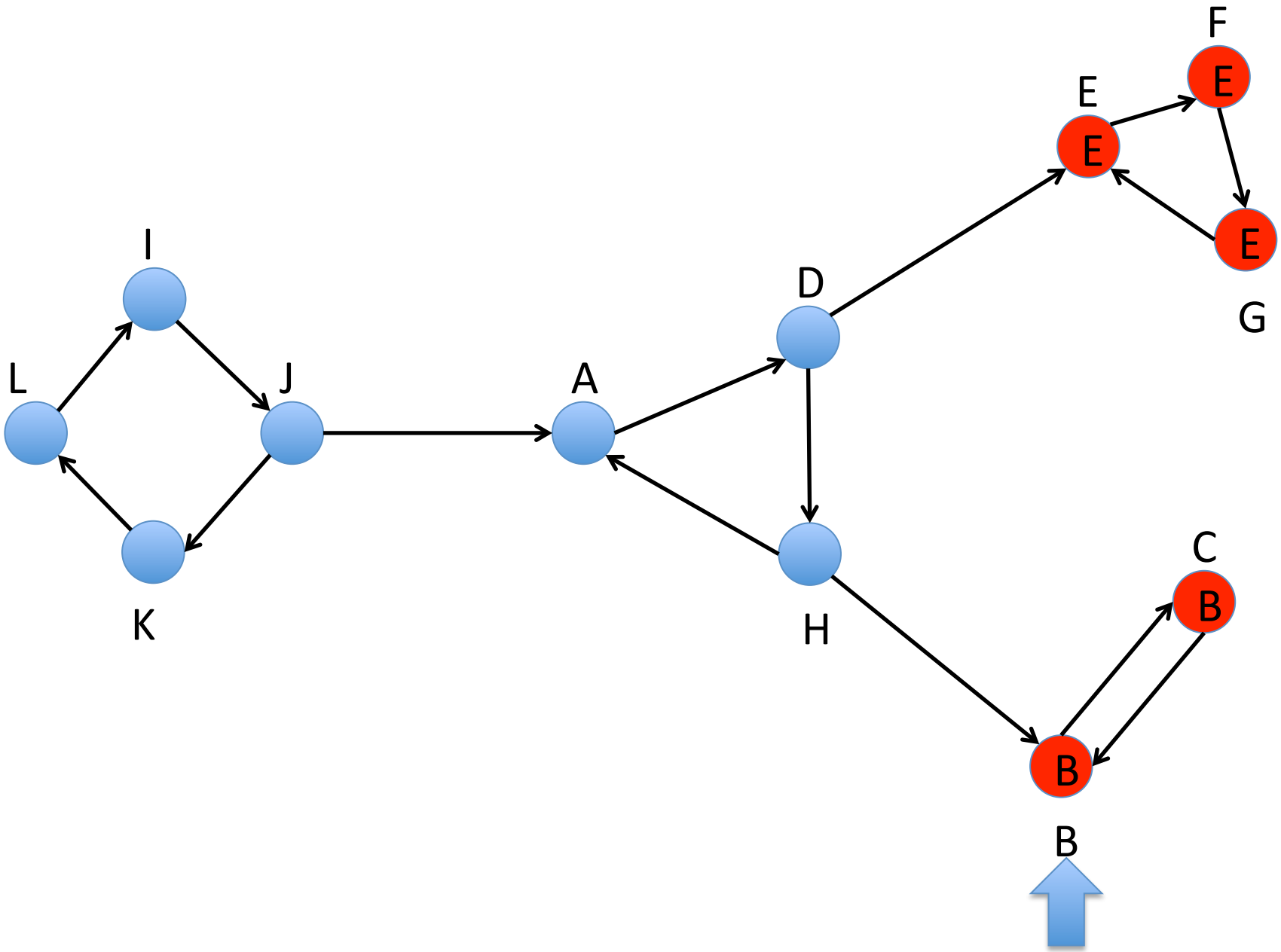
What If We Got Very Lucky?



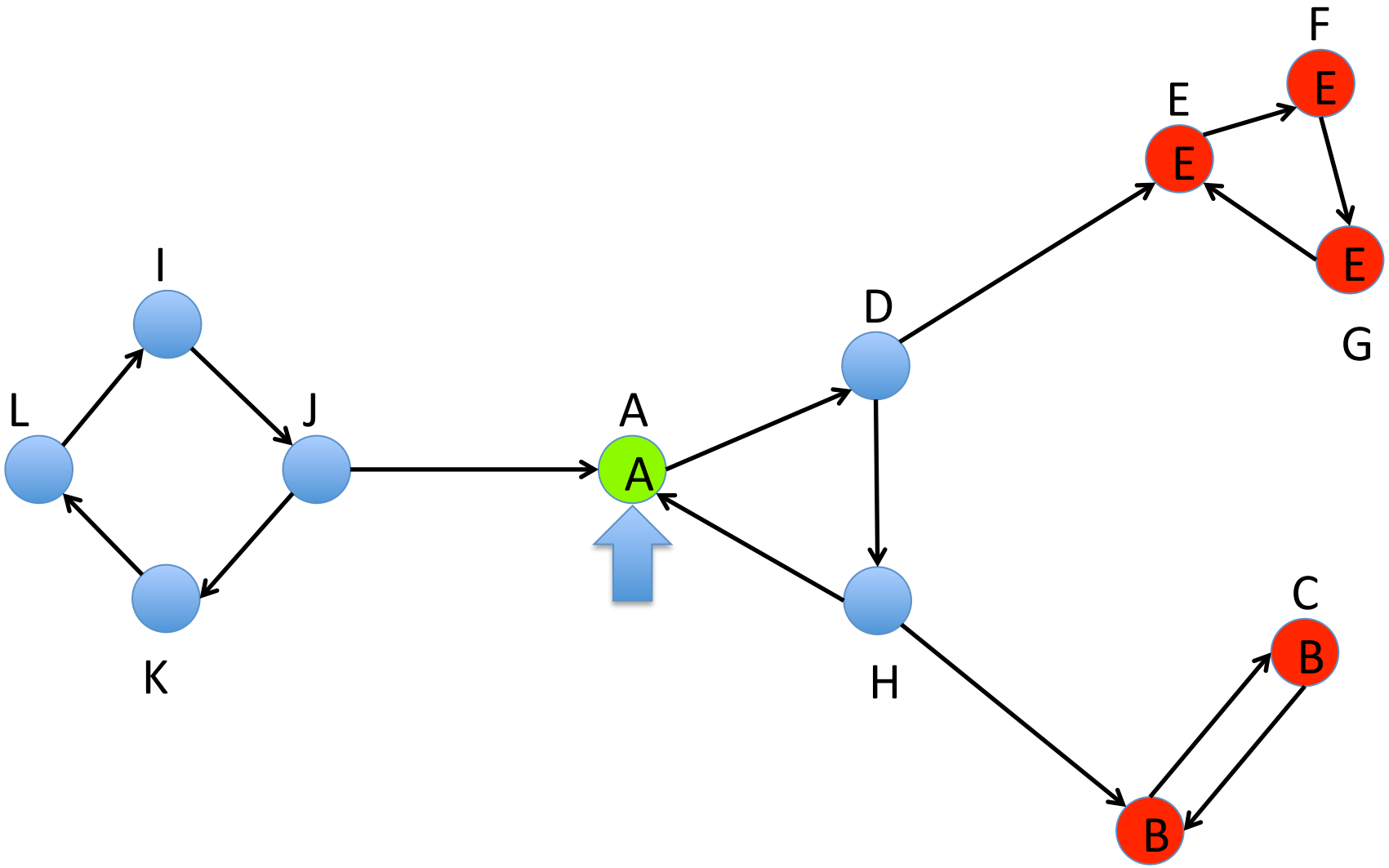
What If We Got Very Lucky?



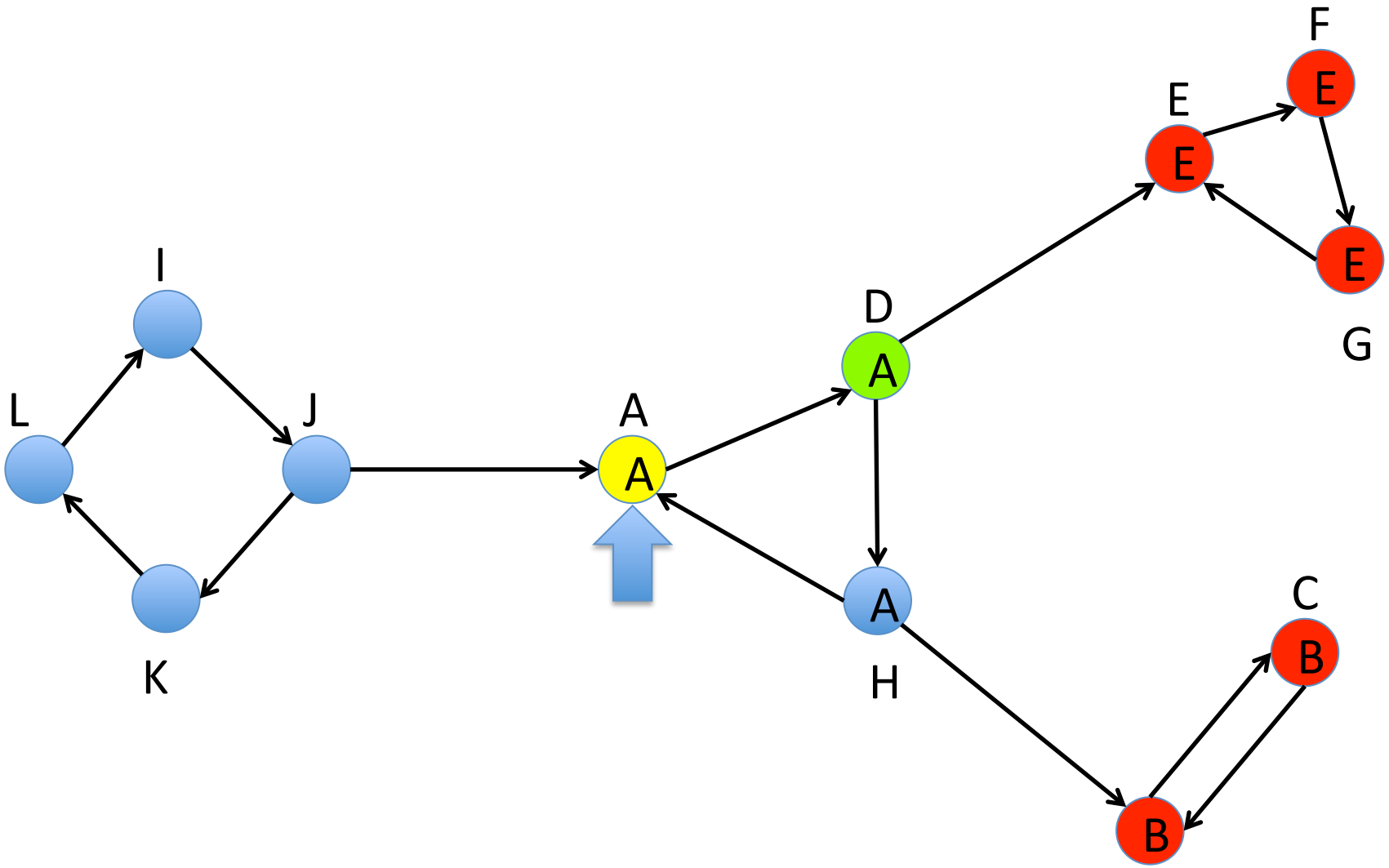
What If We Got Very Lucky?



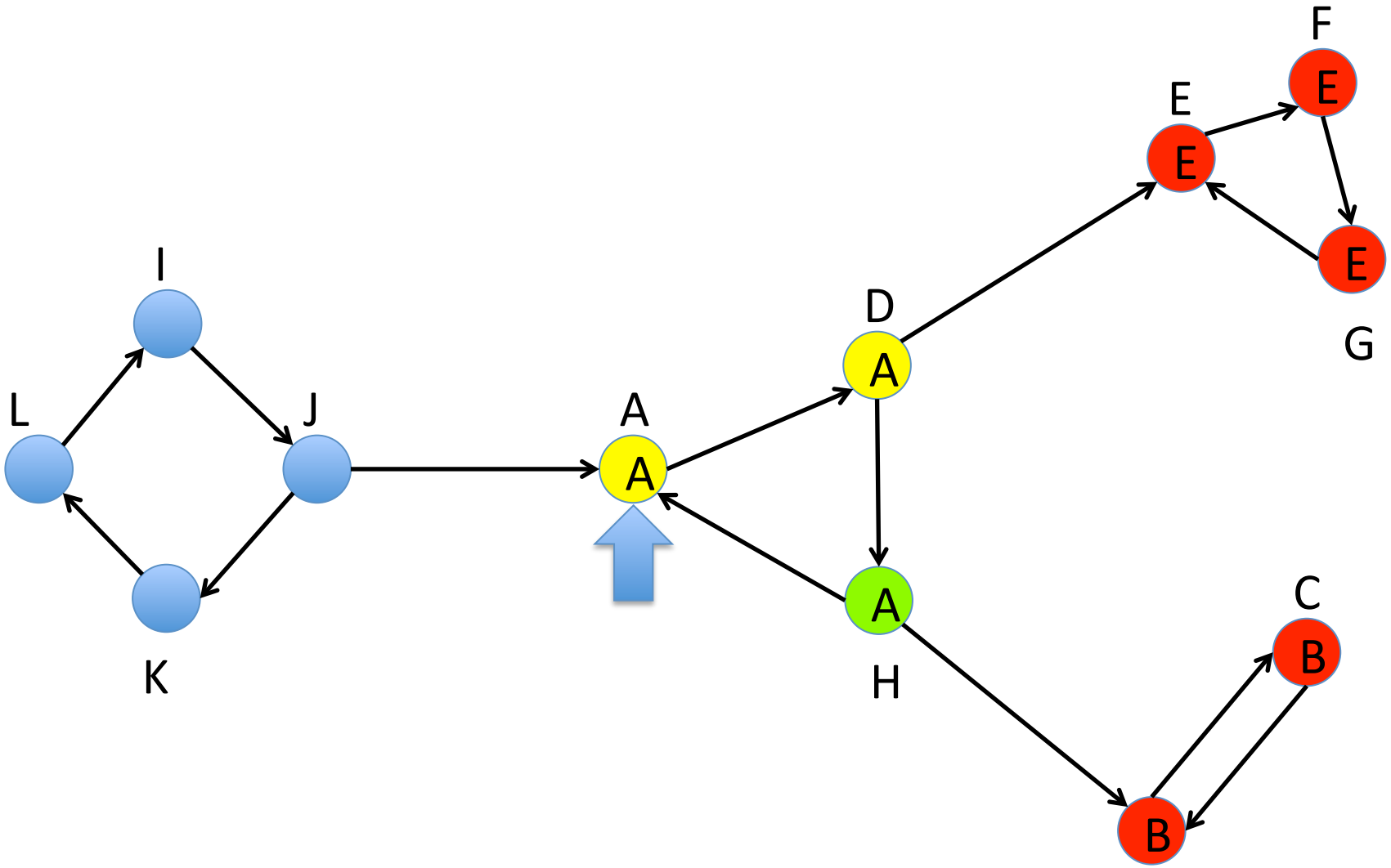
What If We Got Very Lucky?



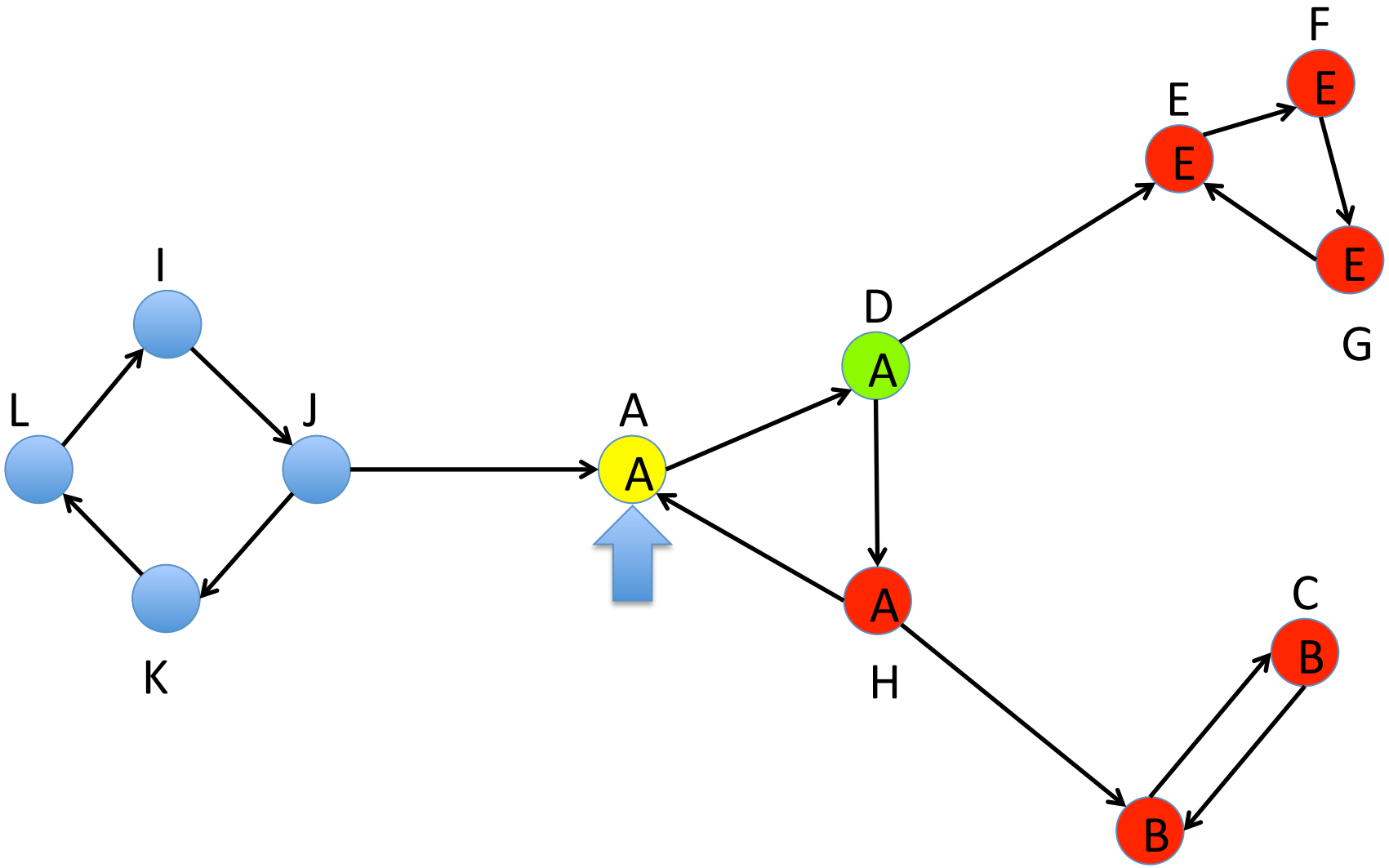
What If We Got Very Lucky?



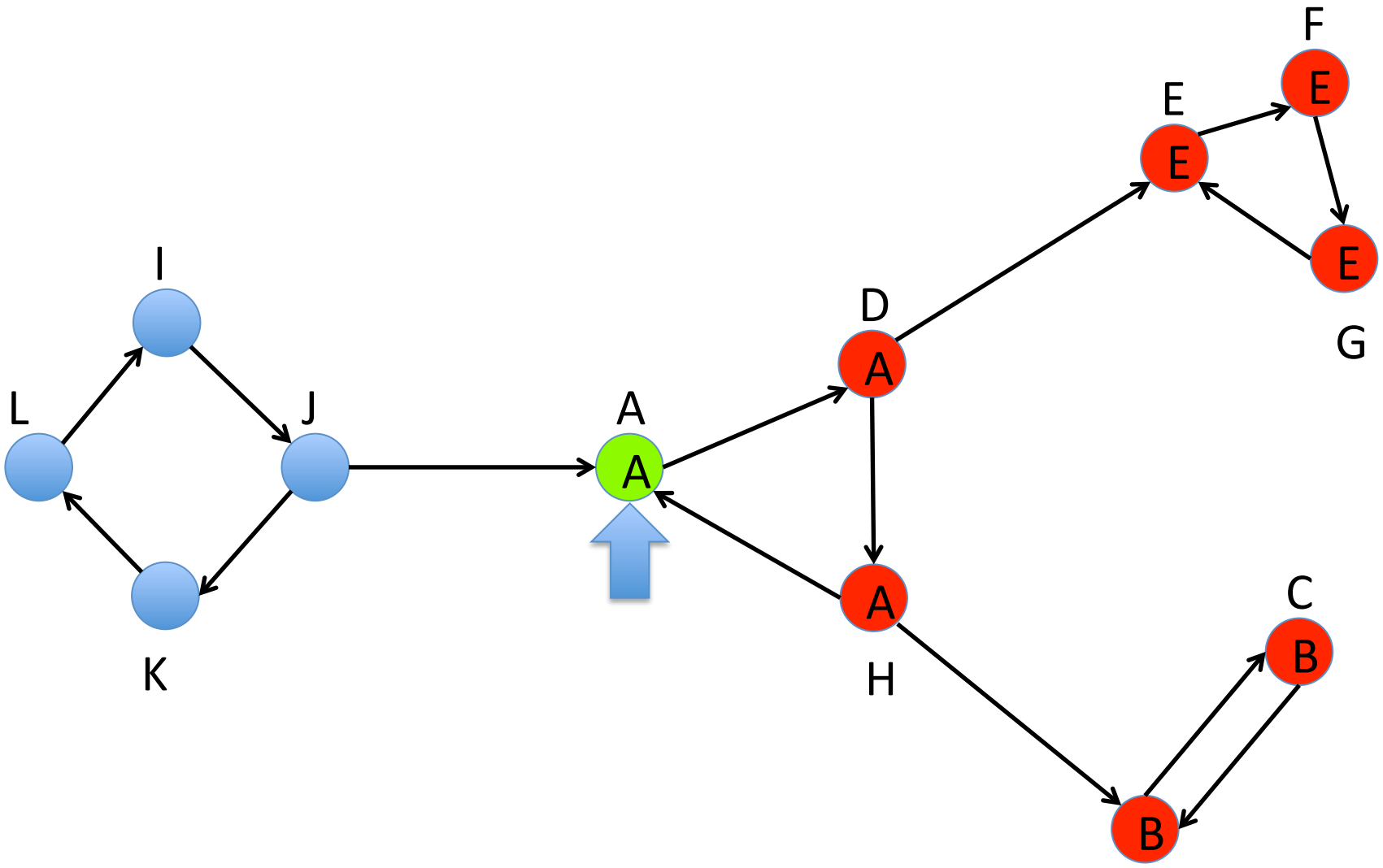
What If We Got Very Lucky?



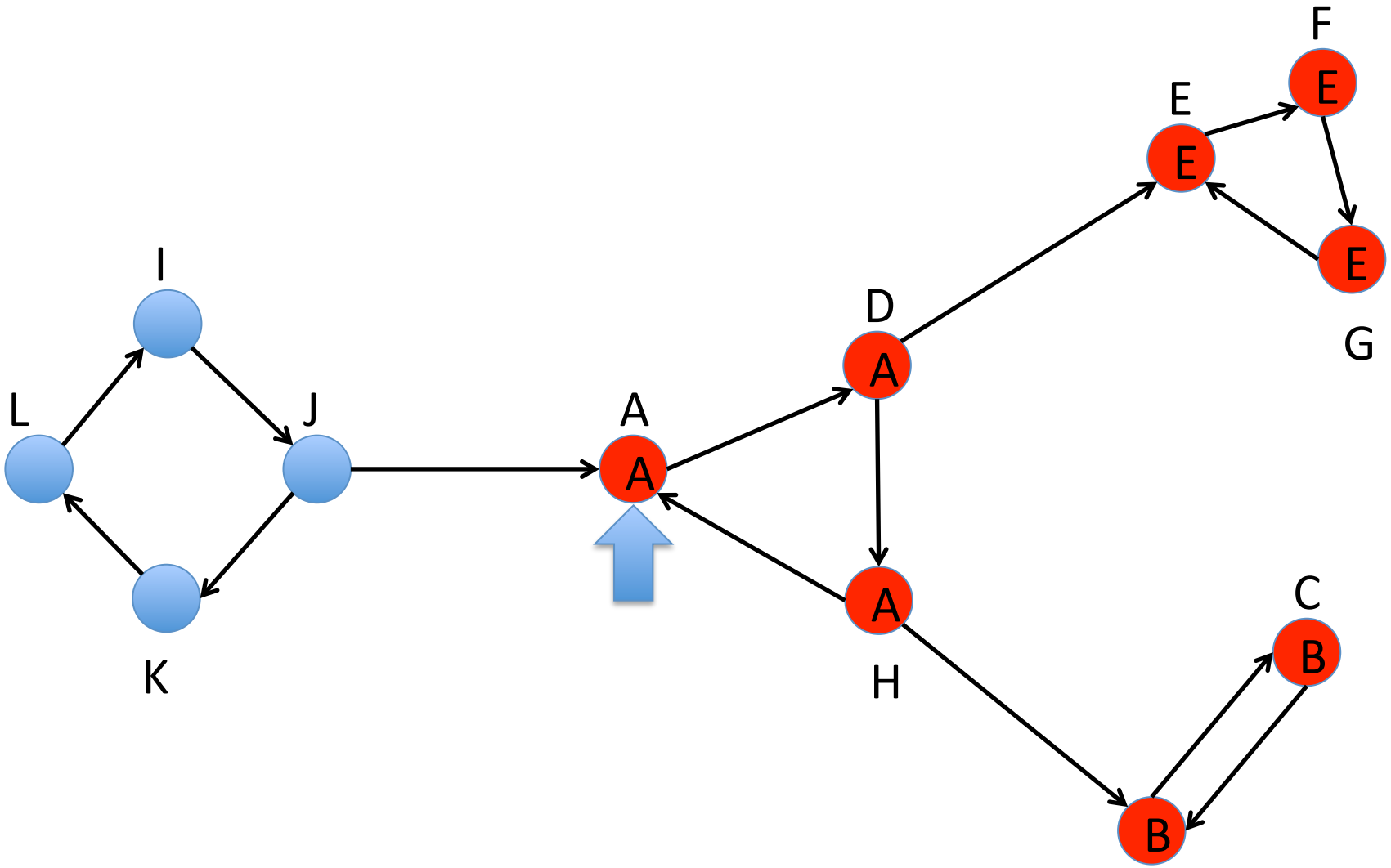
What If We Got Very Lucky?



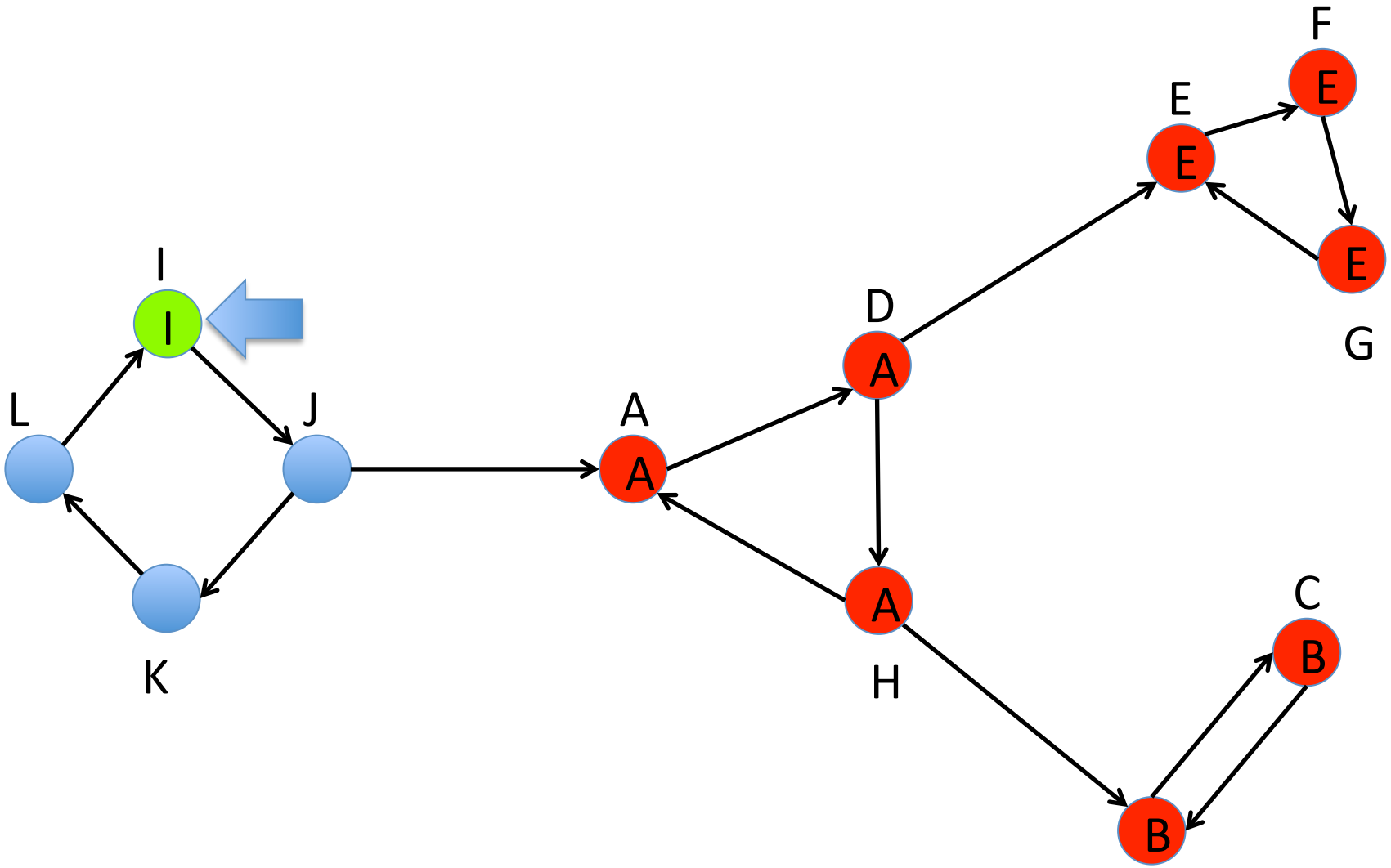
What If We Got Very Lucky?



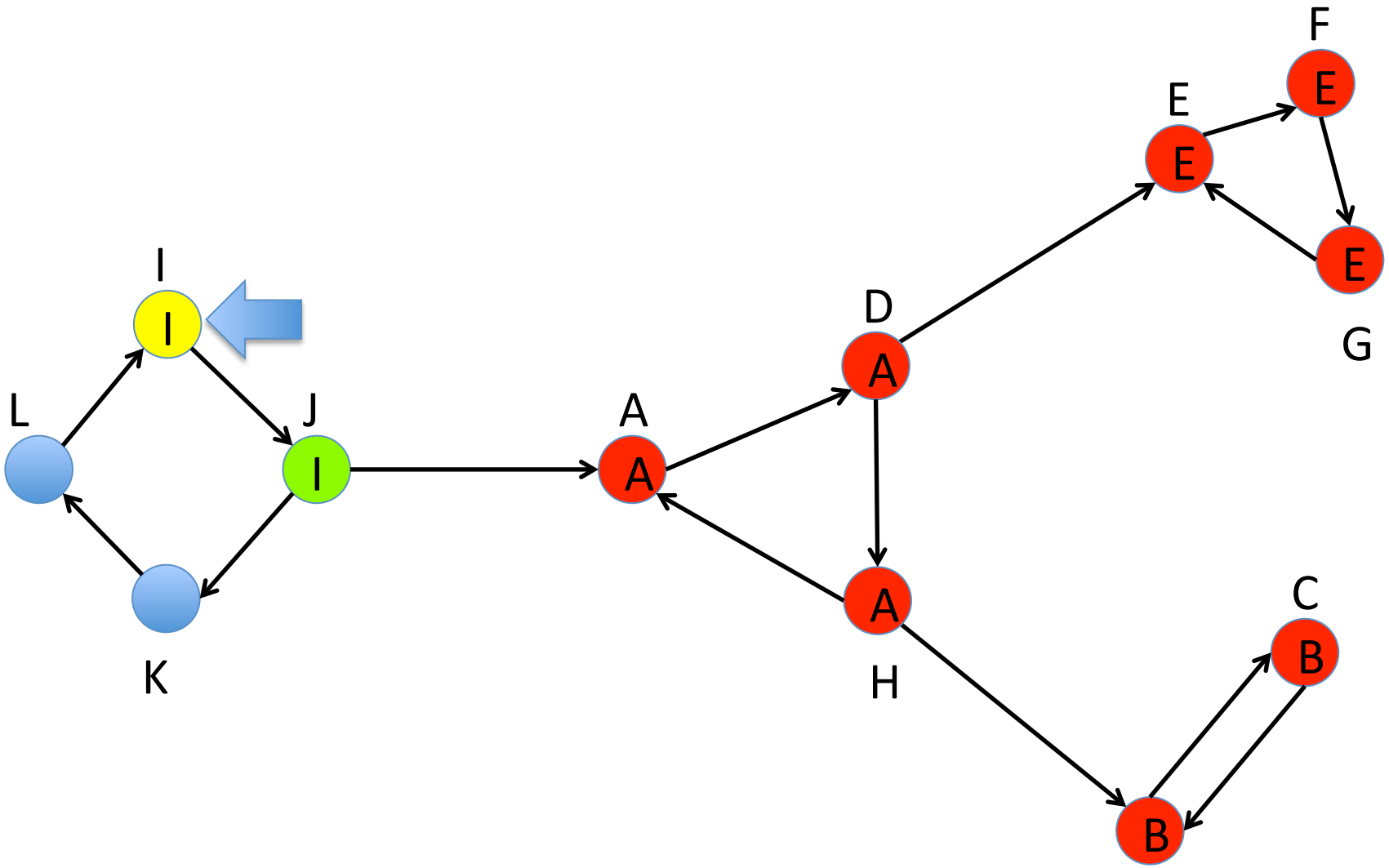
What If We Got Very Lucky?



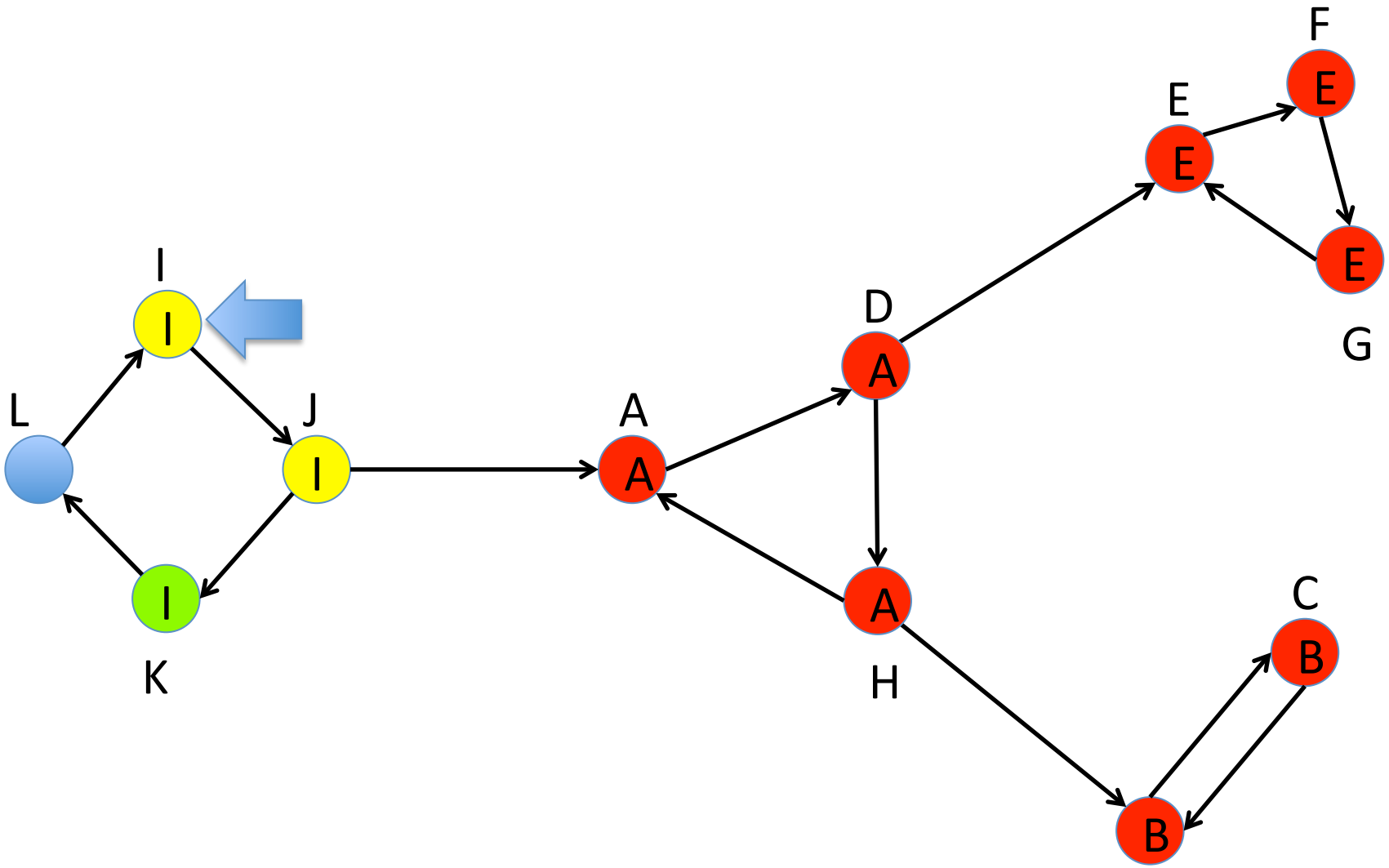
What If We Got Very Lucky?



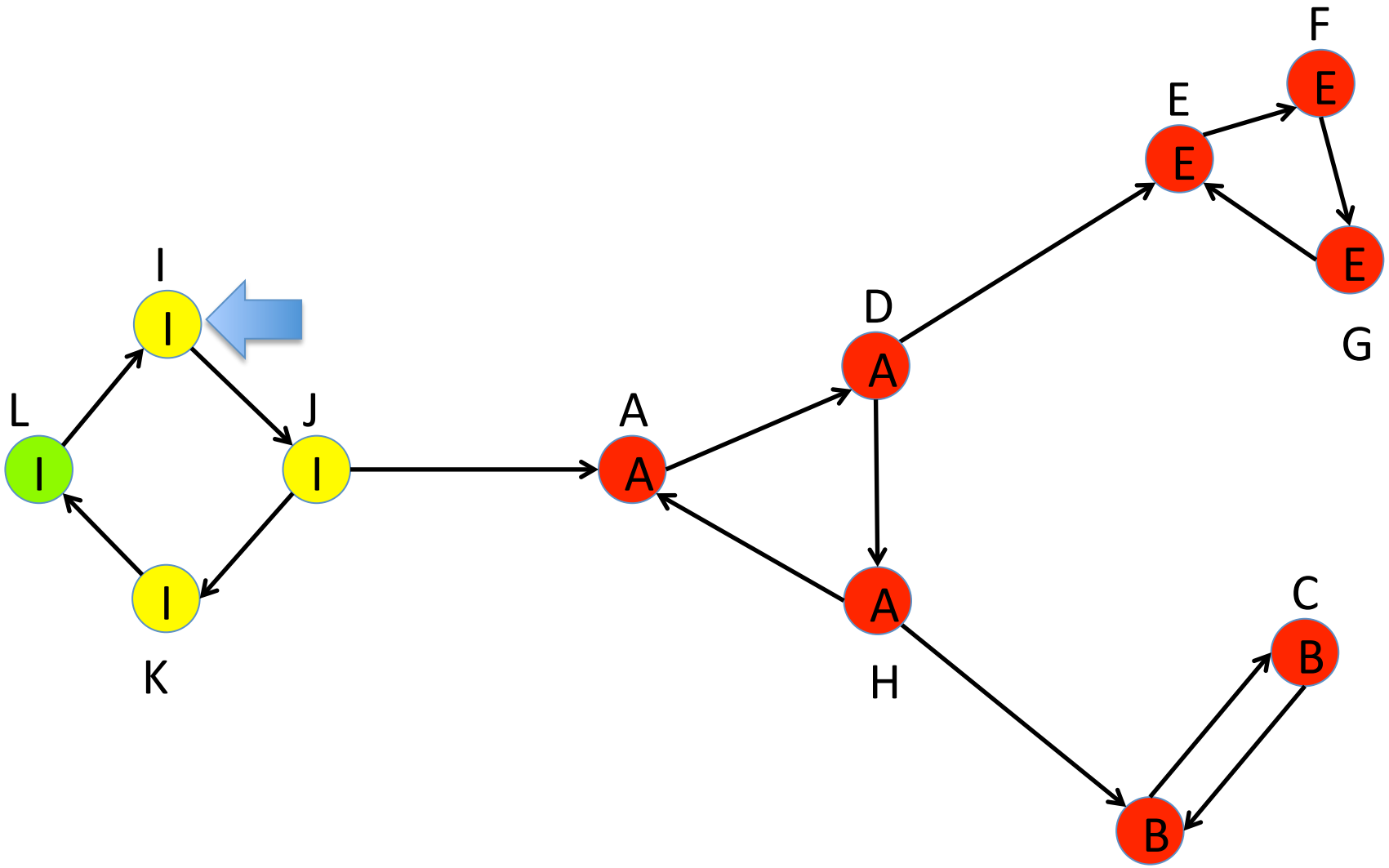
What If We Got Very Lucky?



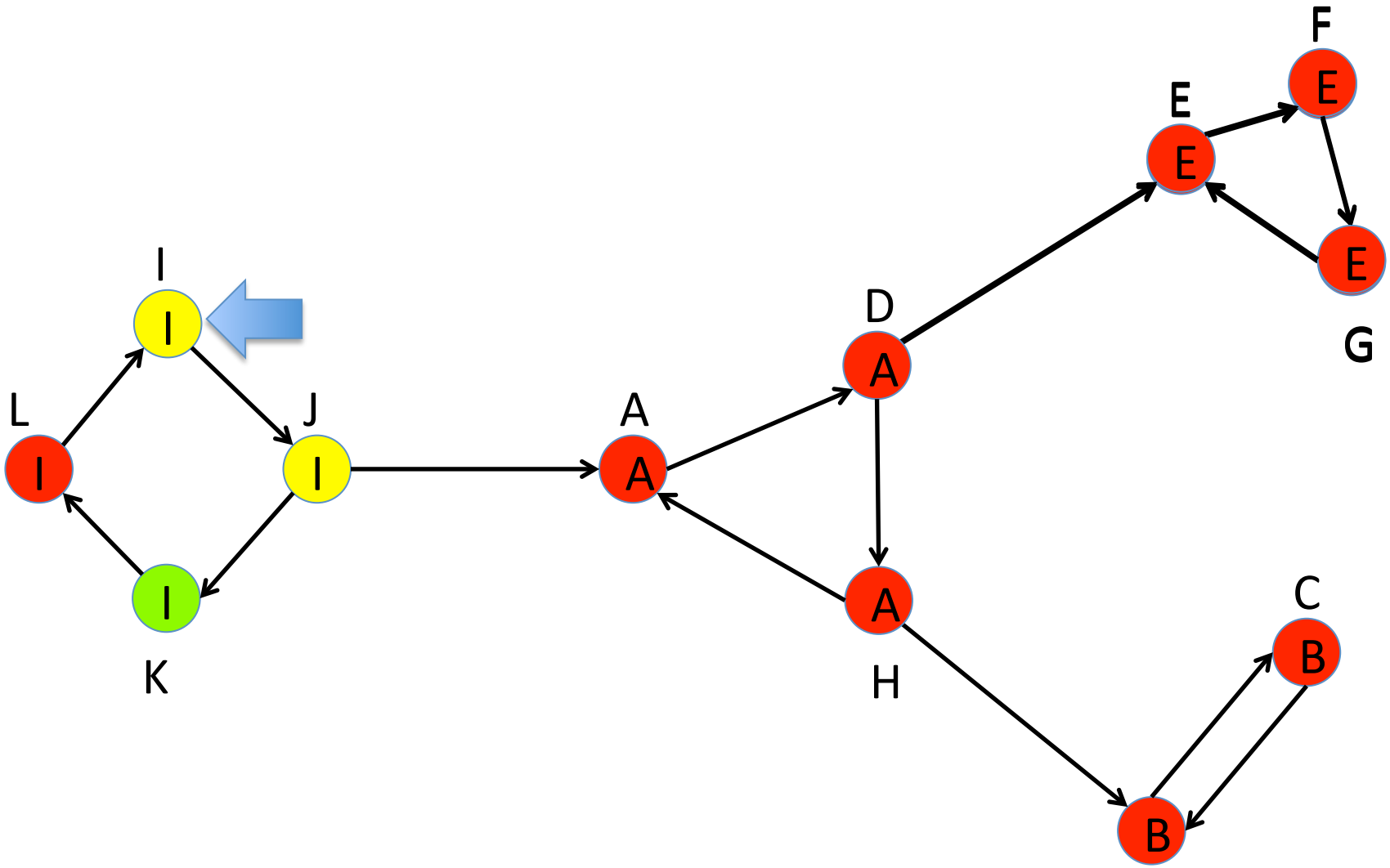
What If We Got Very Lucky?



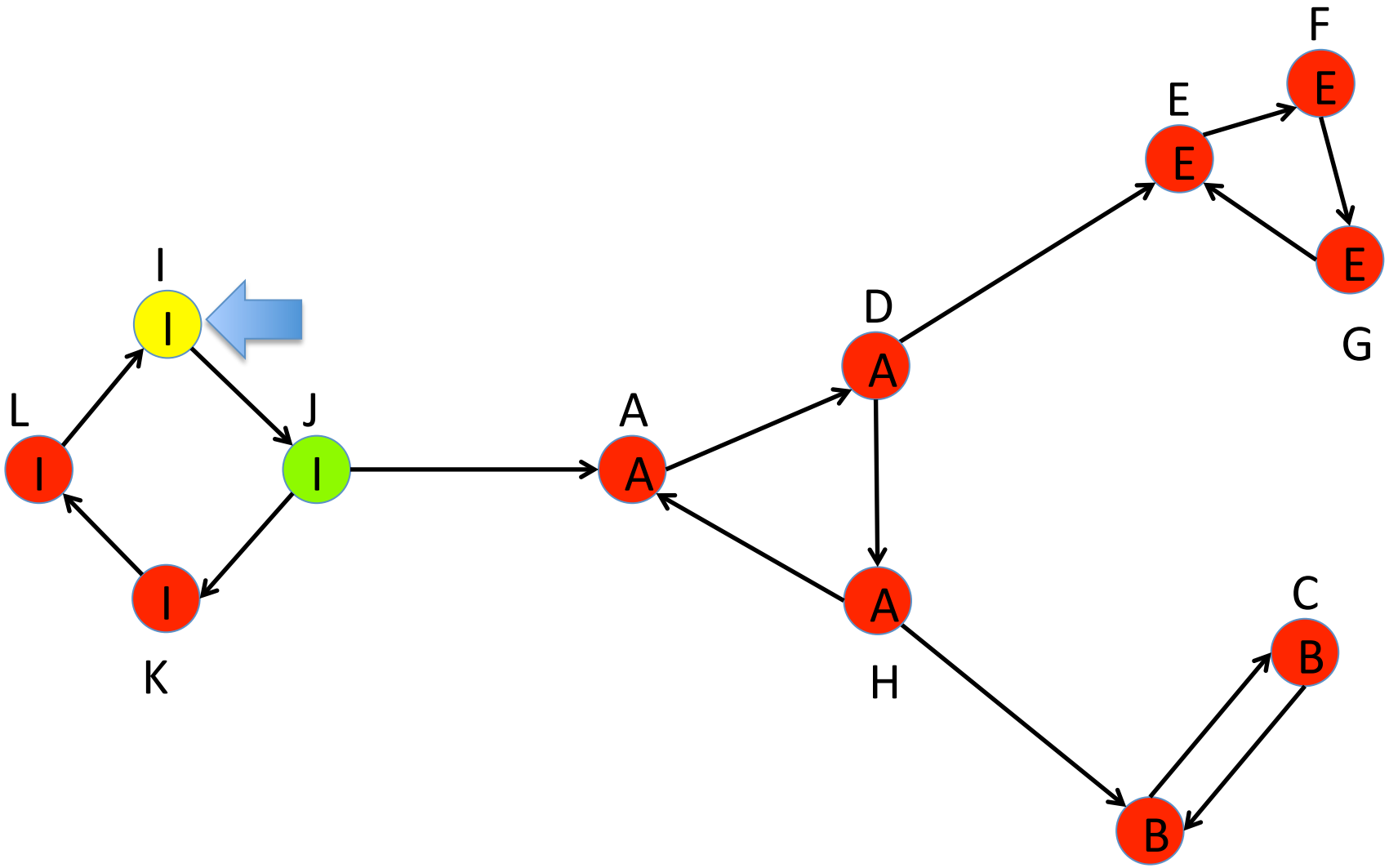
What If We Got Very Lucky?



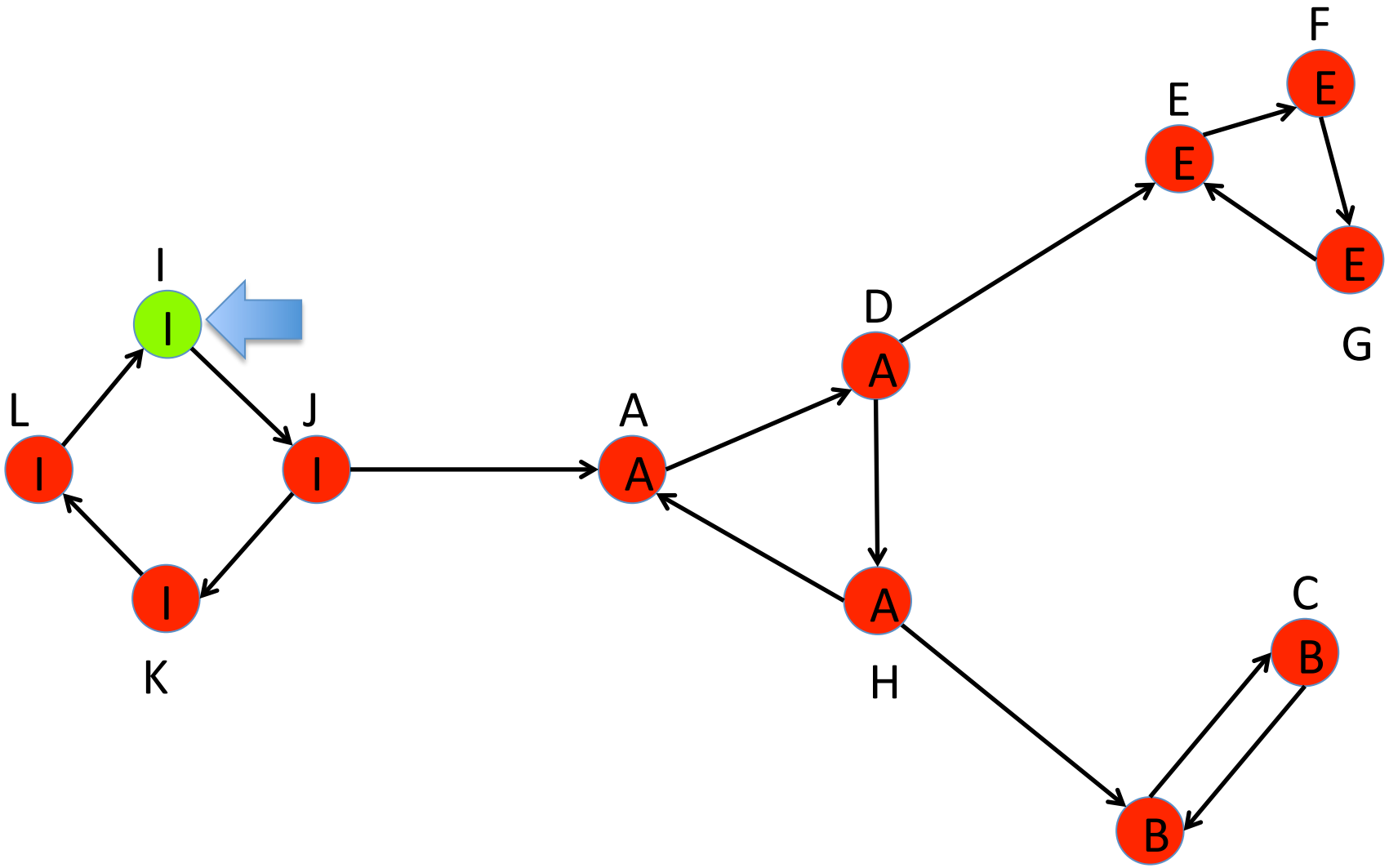
What If We Got Very Lucky?



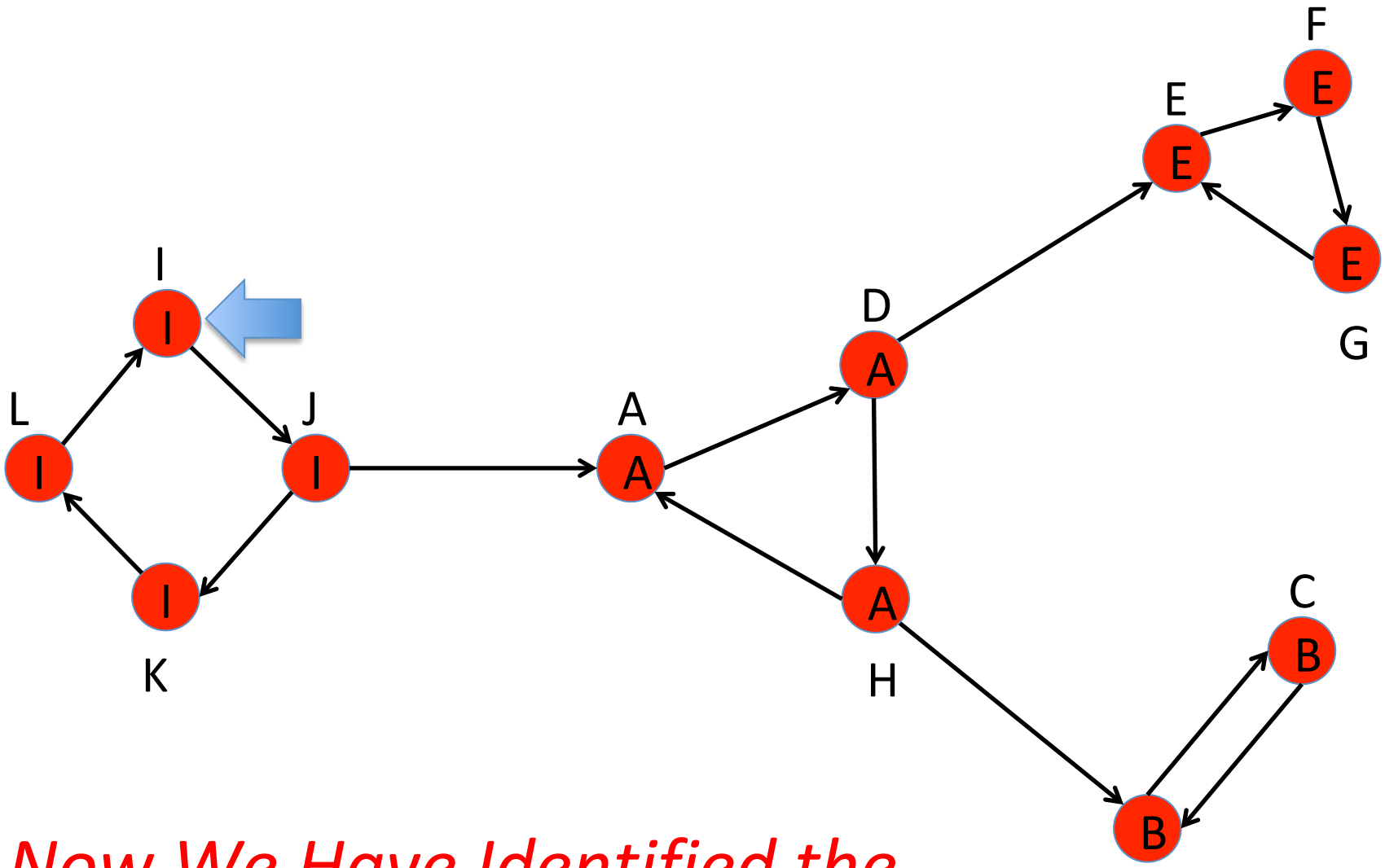
What If We Got Very Lucky?



What If We Got Very Lucky?

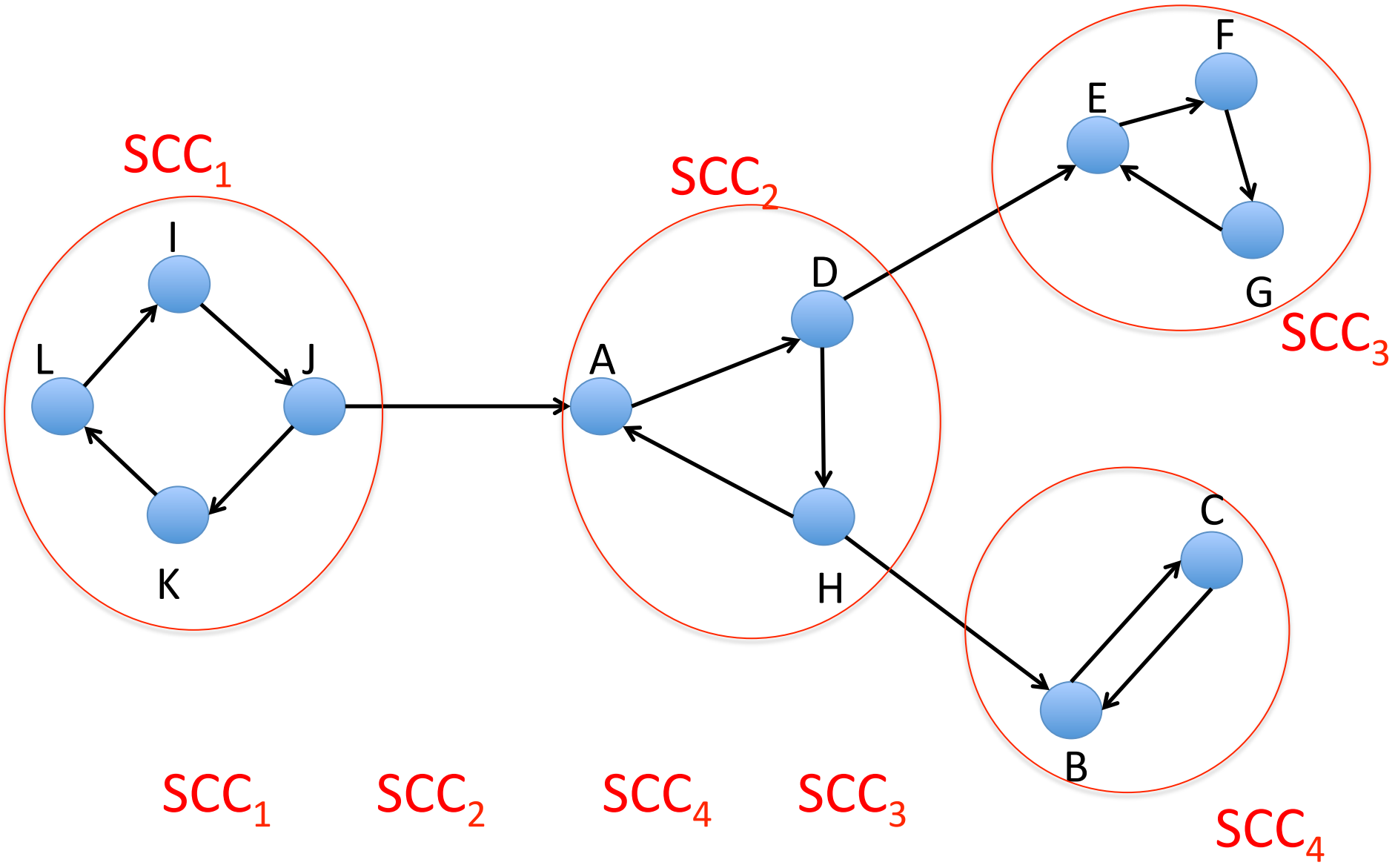


What If We Got Very Lucky?



Now We Have Identified the SCCs Correctly!

Need to start BFS/DFS from a vertex in reverse topological order of G^{SCC}



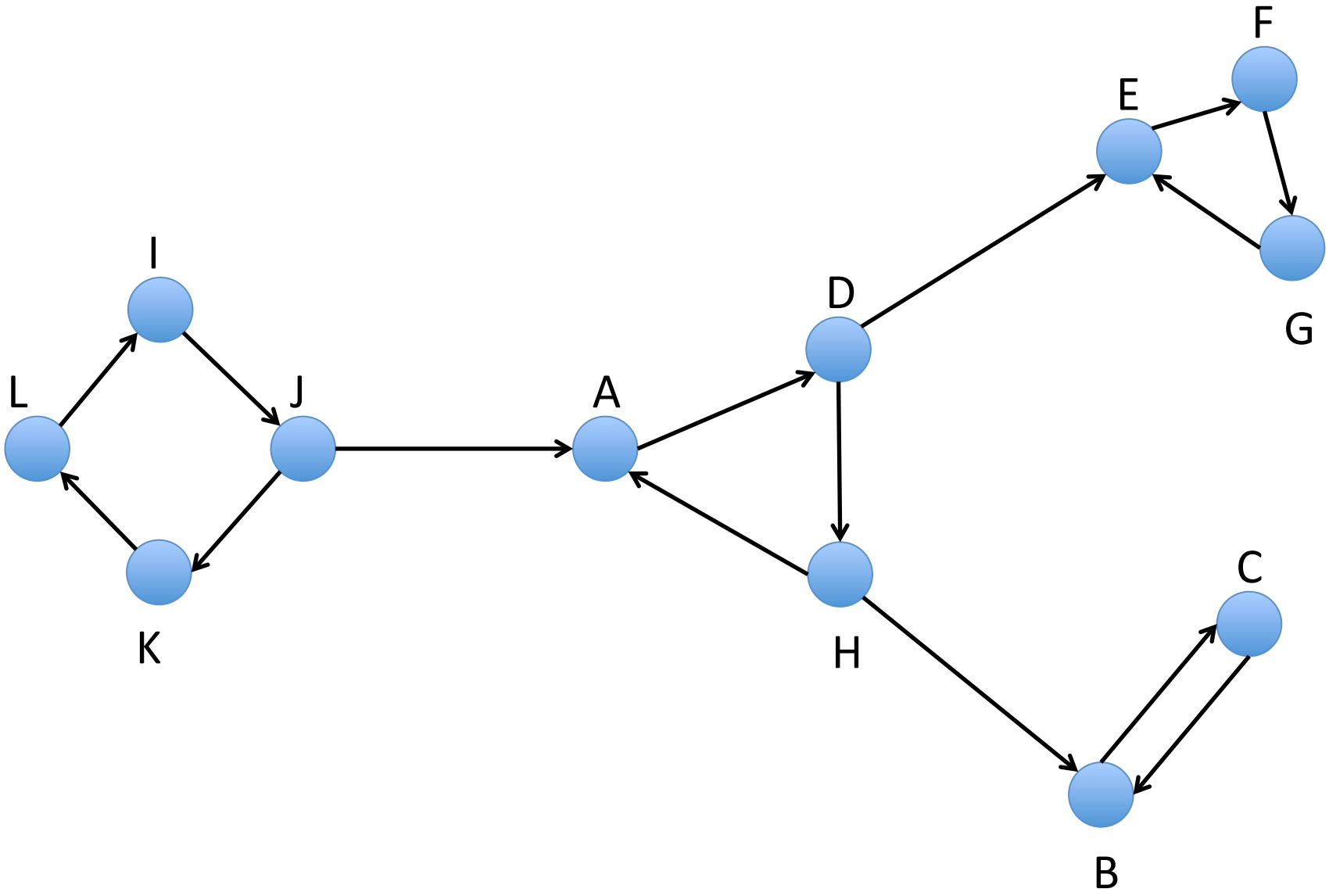
Key Idea in Kosaraju's Algorithm

- If we can start a BFS/DFS from vertices in reverse topological order of G^{SCC}
- In other words, if we can always pick a start vertex from a sink SCC
- We can use BFS/DFS to identify the components correctly!

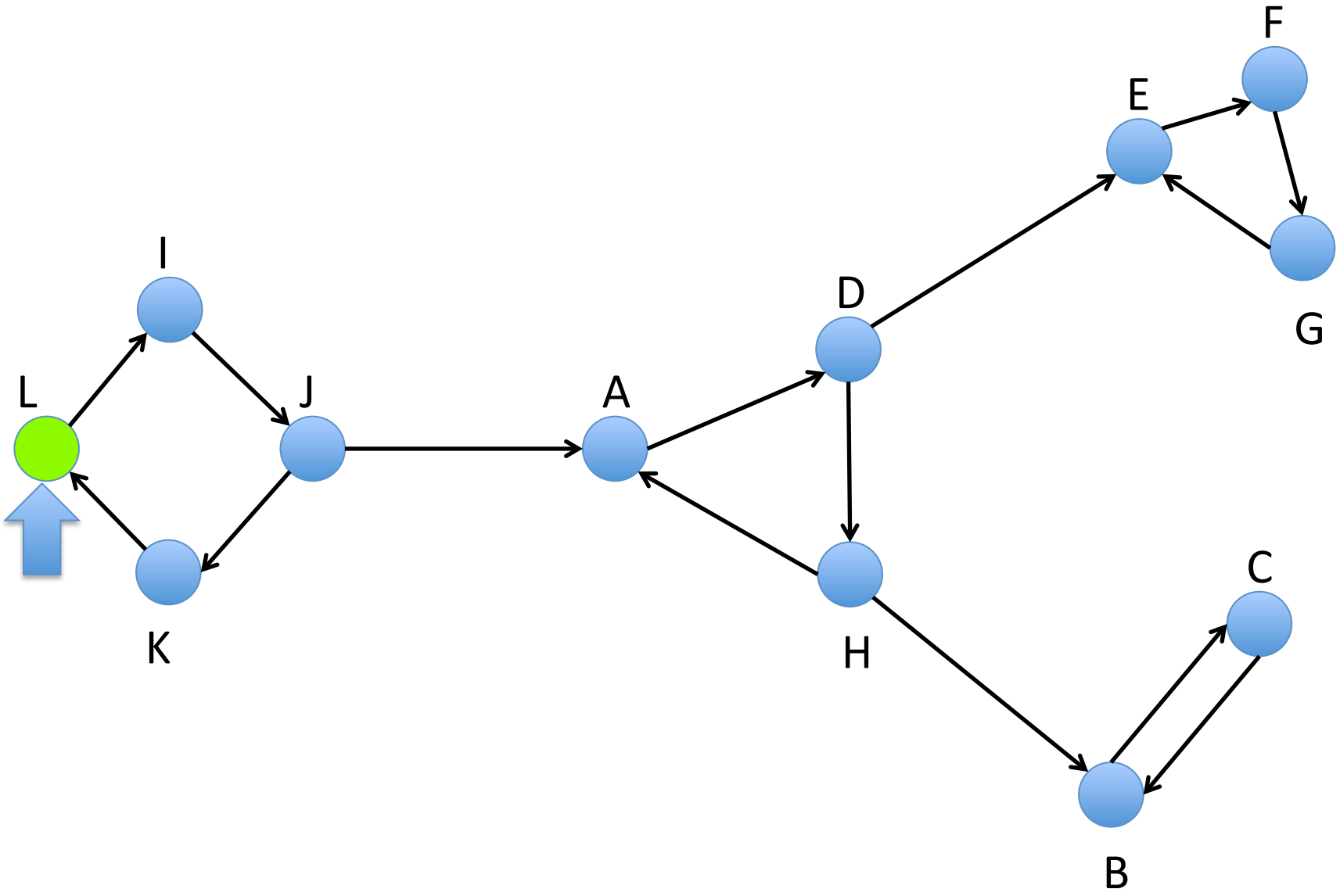
Goal: Do some preprocessing to guarantee that we always start a traversal from vertex from a sink SCC!

How?

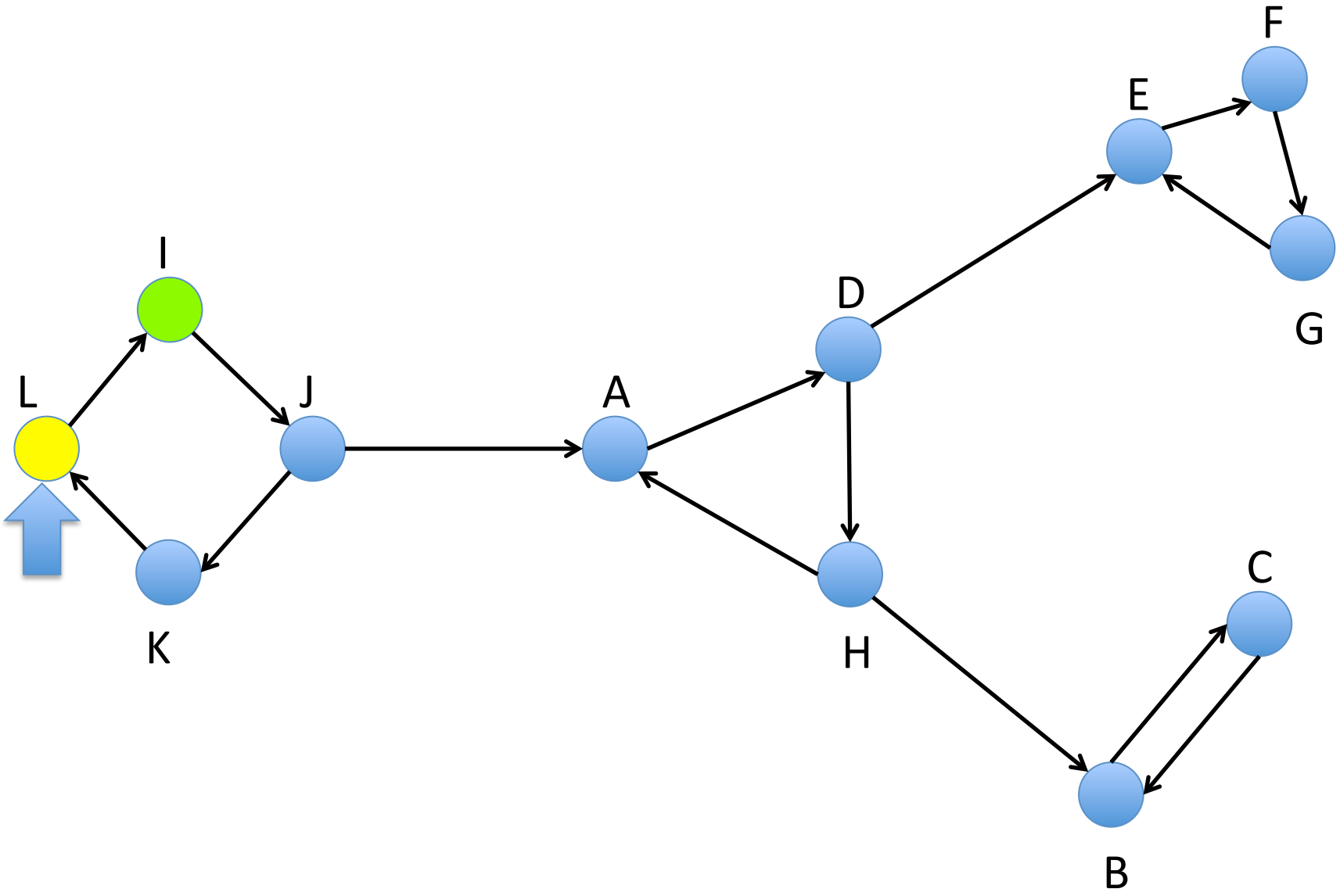
Let's Try DFS & Finishing Times (1)



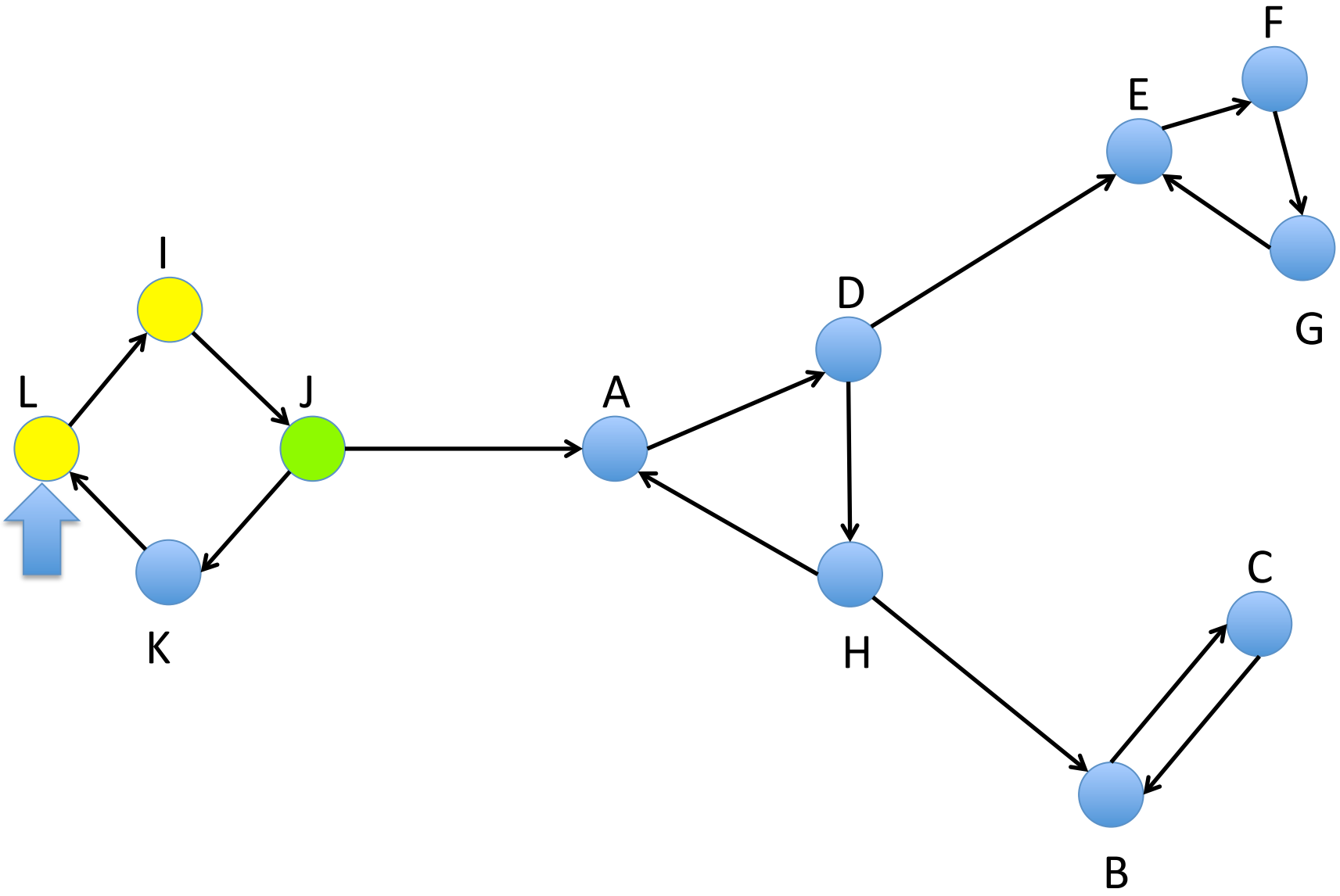
Let's Try DFS & Finishing Times (1)



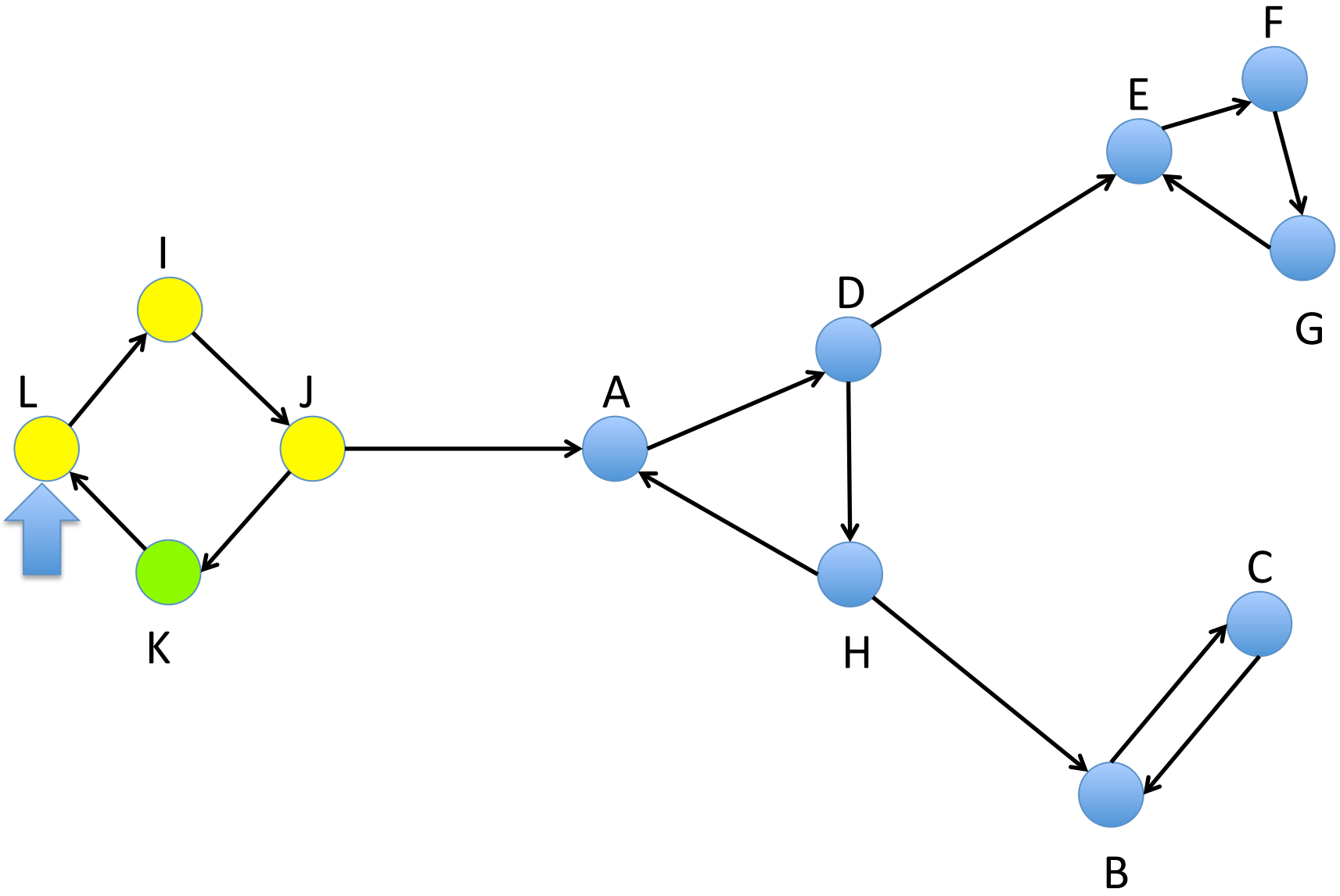
Let's Try DFS & Finishing Times (1)



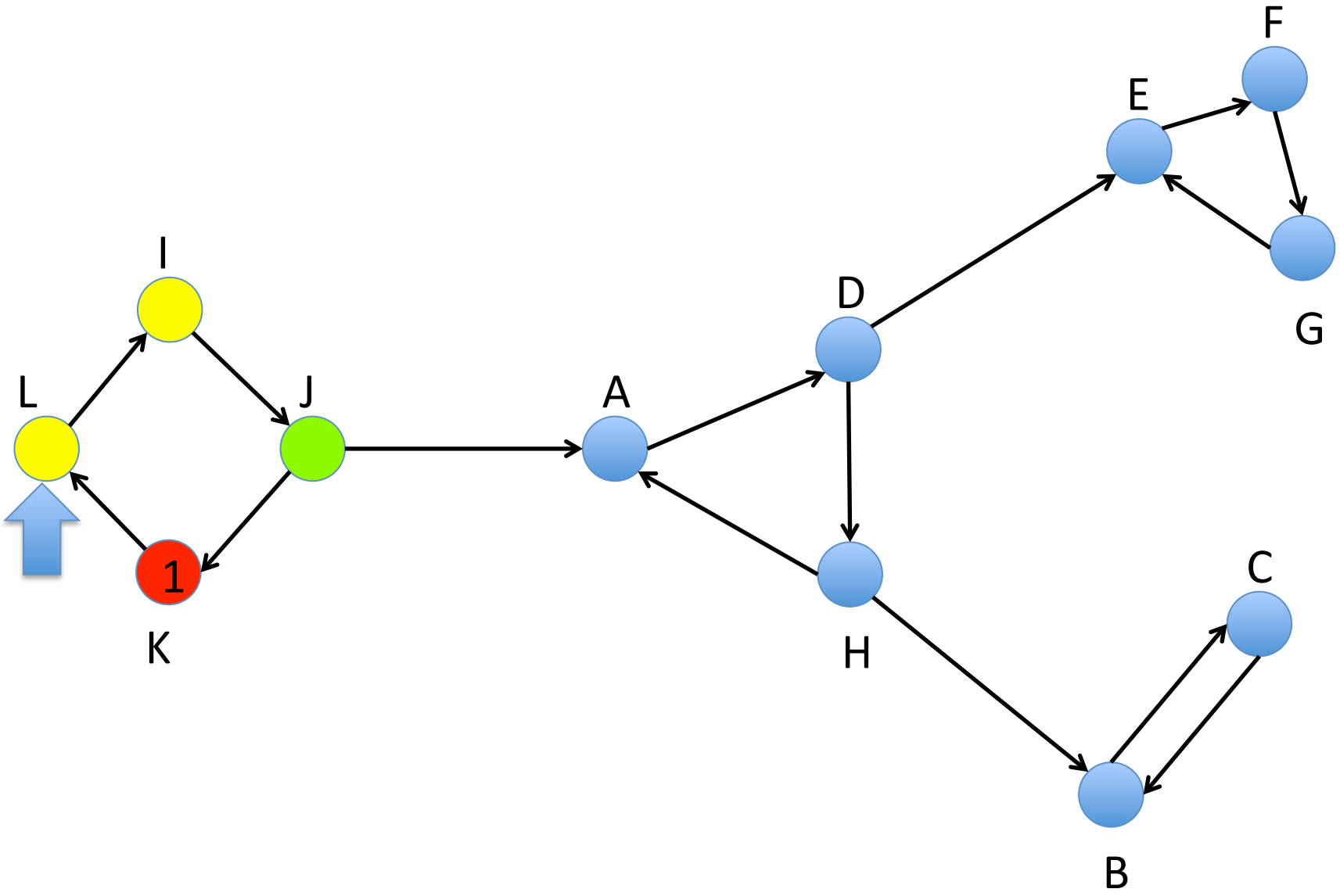
Let's Try DFS & Finishing Times (1)



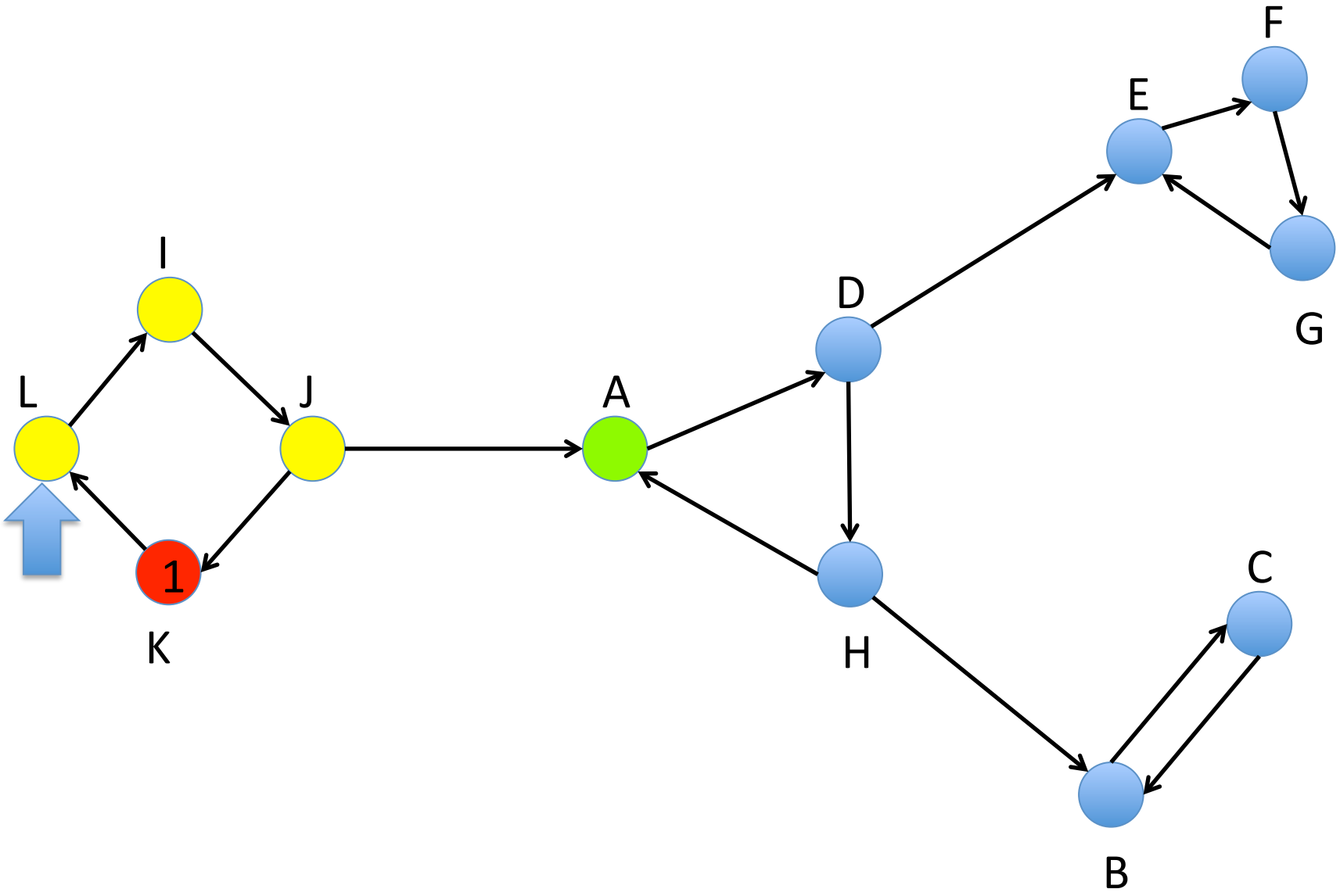
Let's Try DFS & Finishing Times (1)



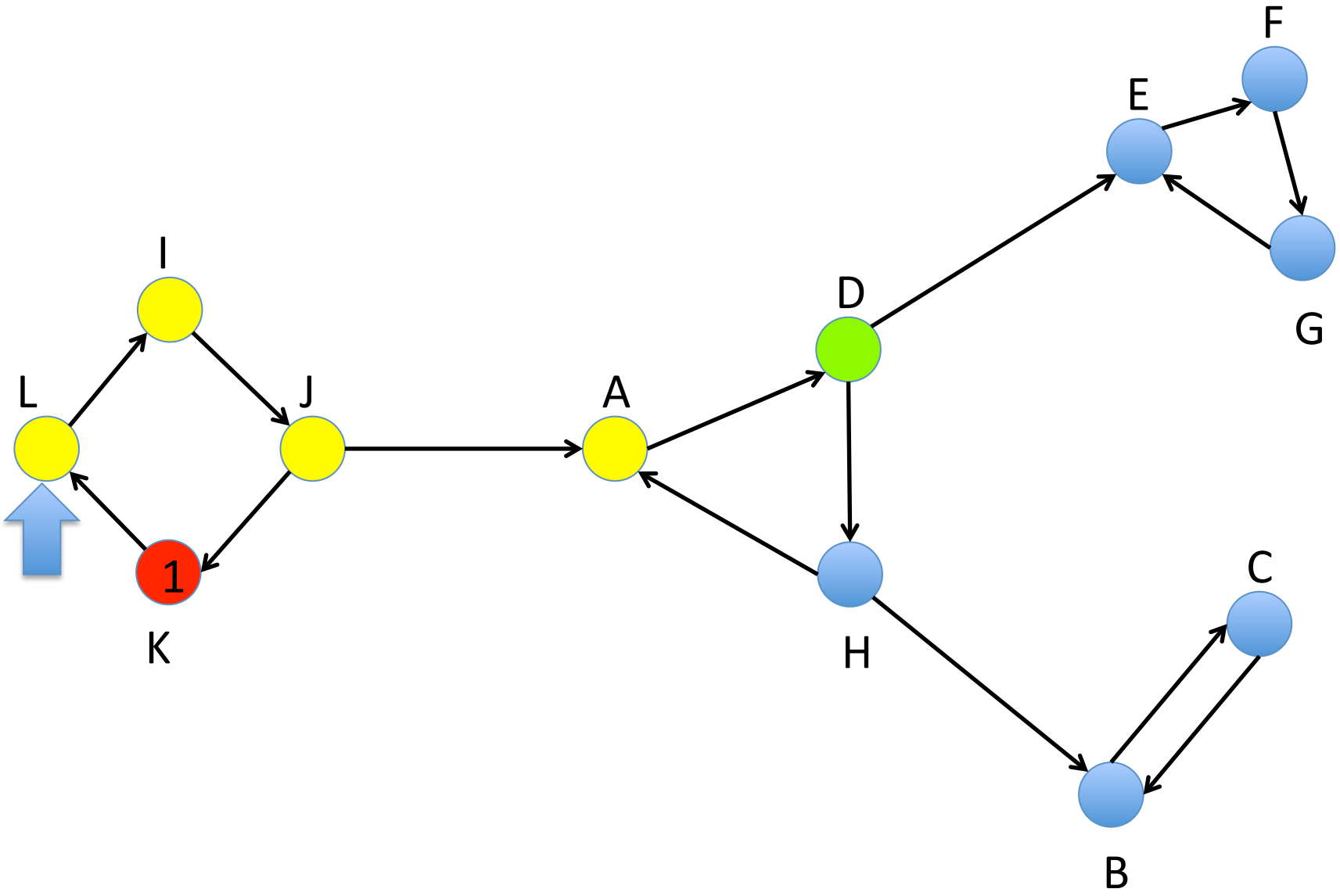
Let's Try DFS & Finishing Times (1)



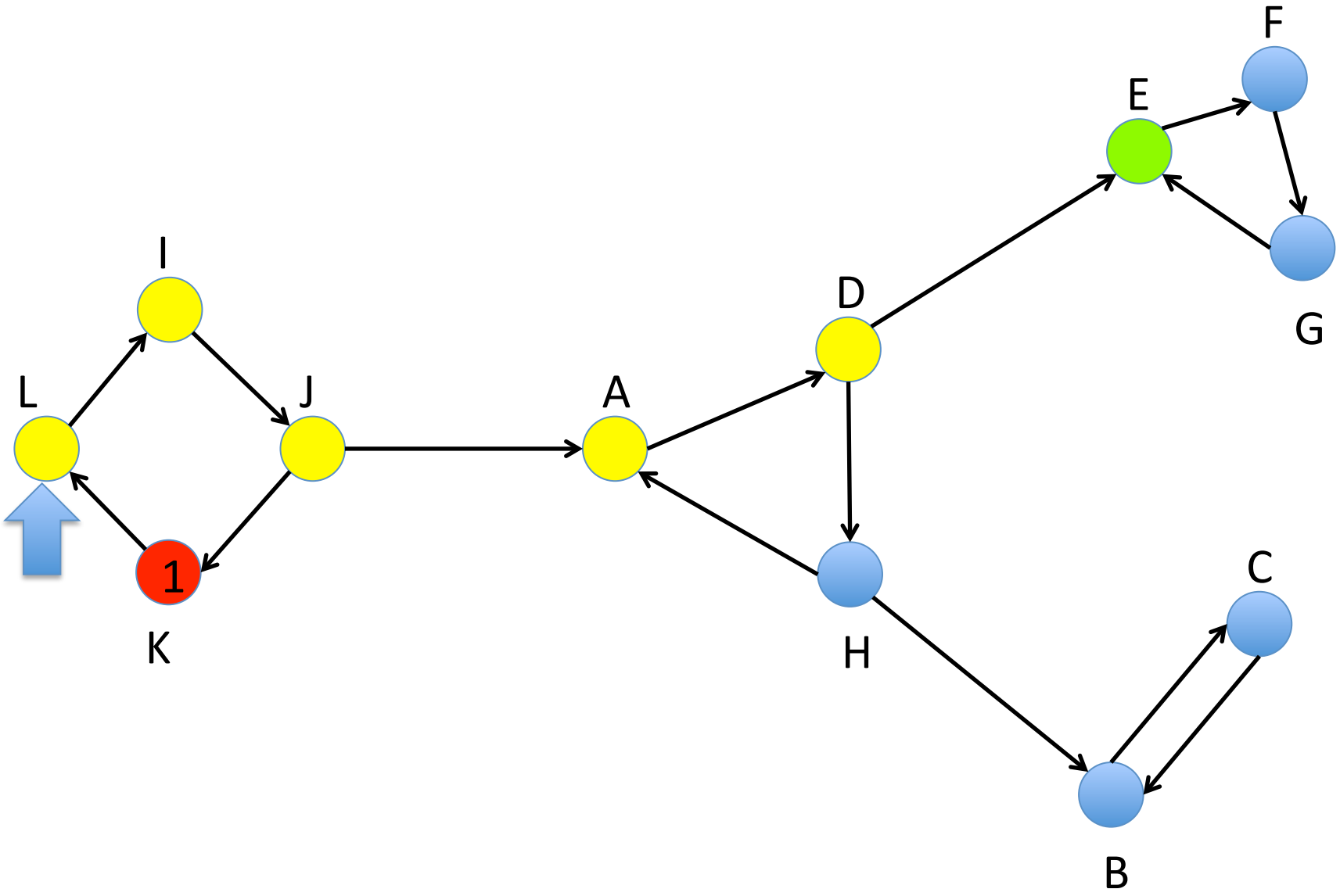
Let's Try DFS & Finishing Times (1)



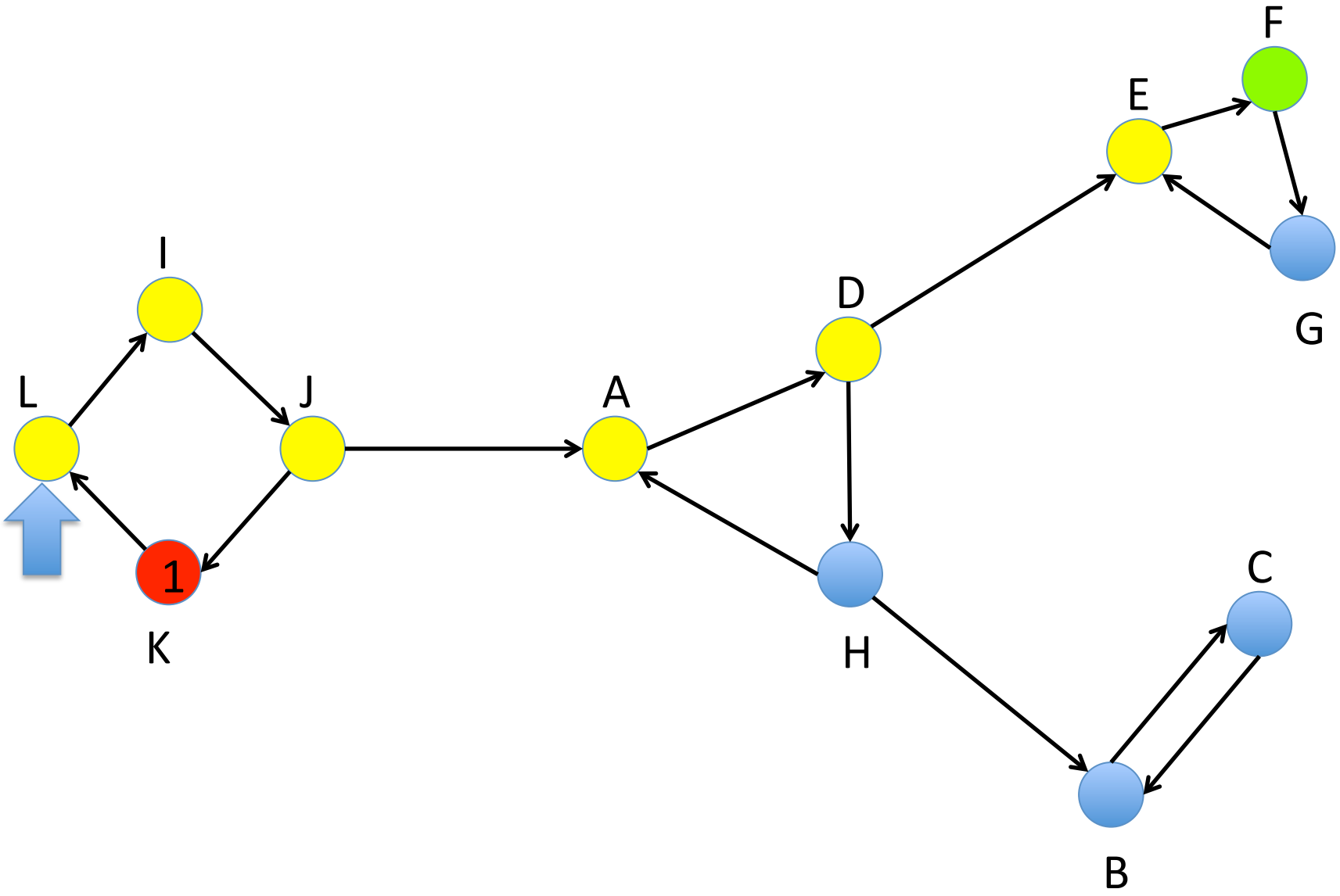
Let's Try DFS & Finishing Times (1)



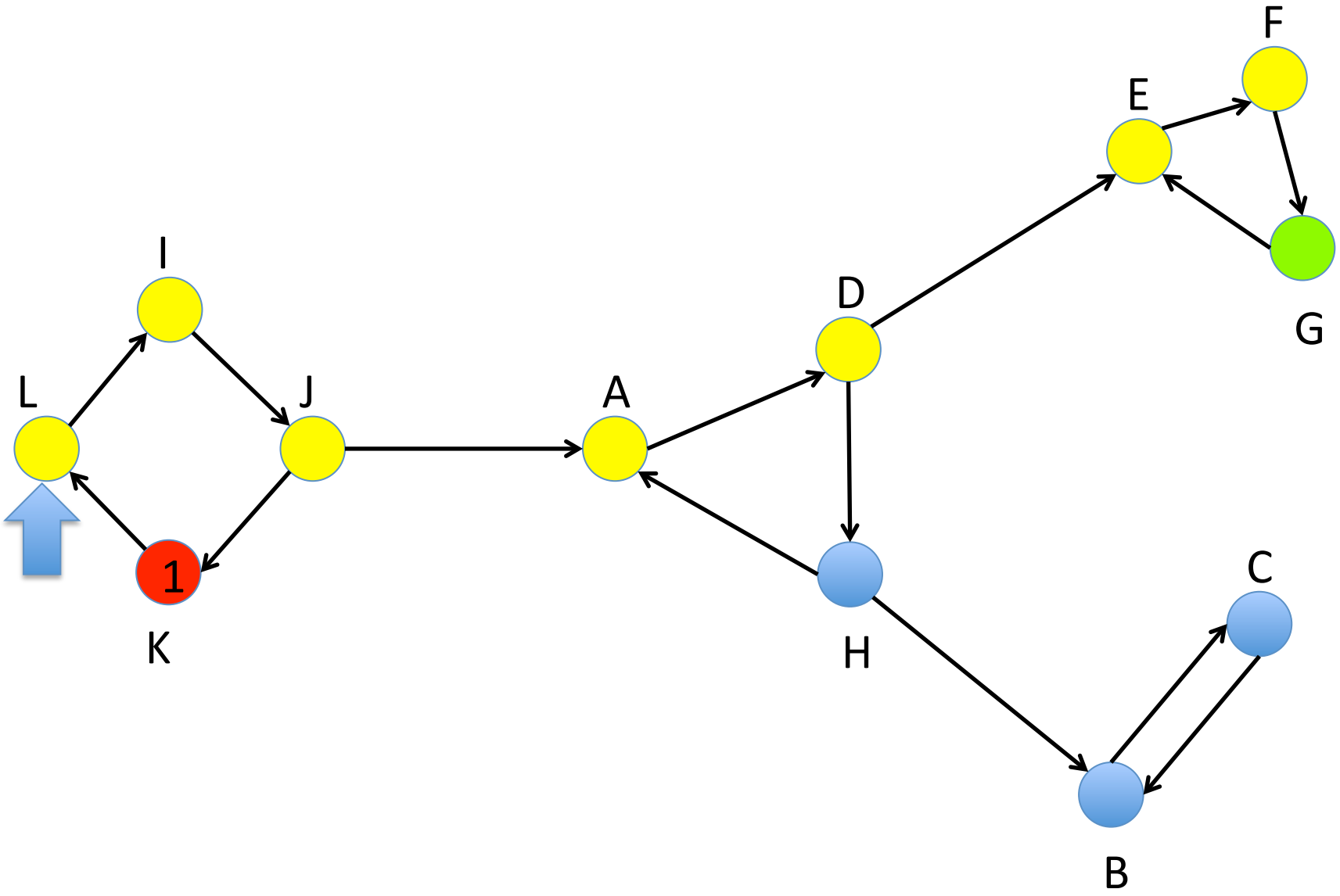
Let's Try DFS & Finishing Times (1)



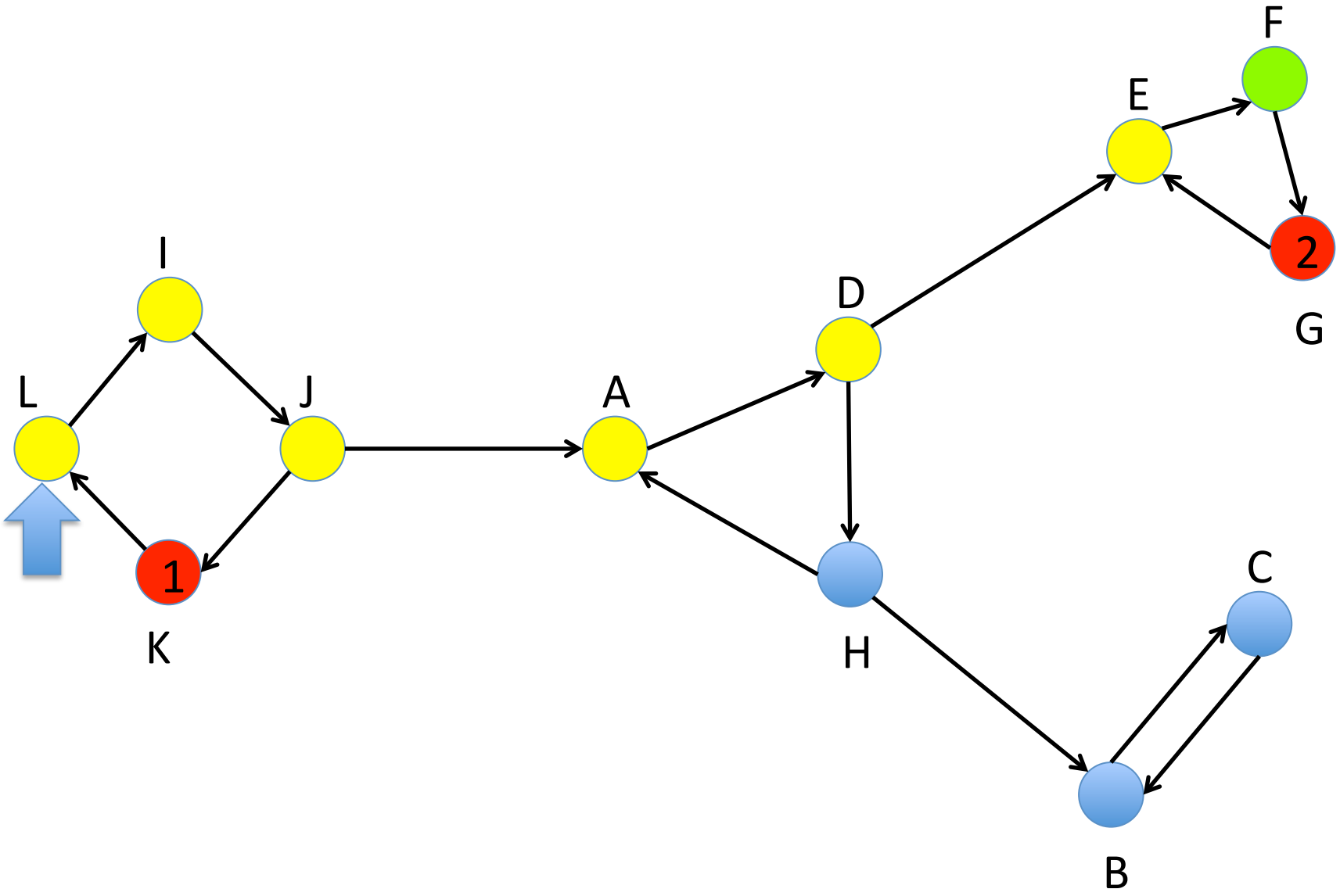
Let's Try DFS & Finishing Times (1)



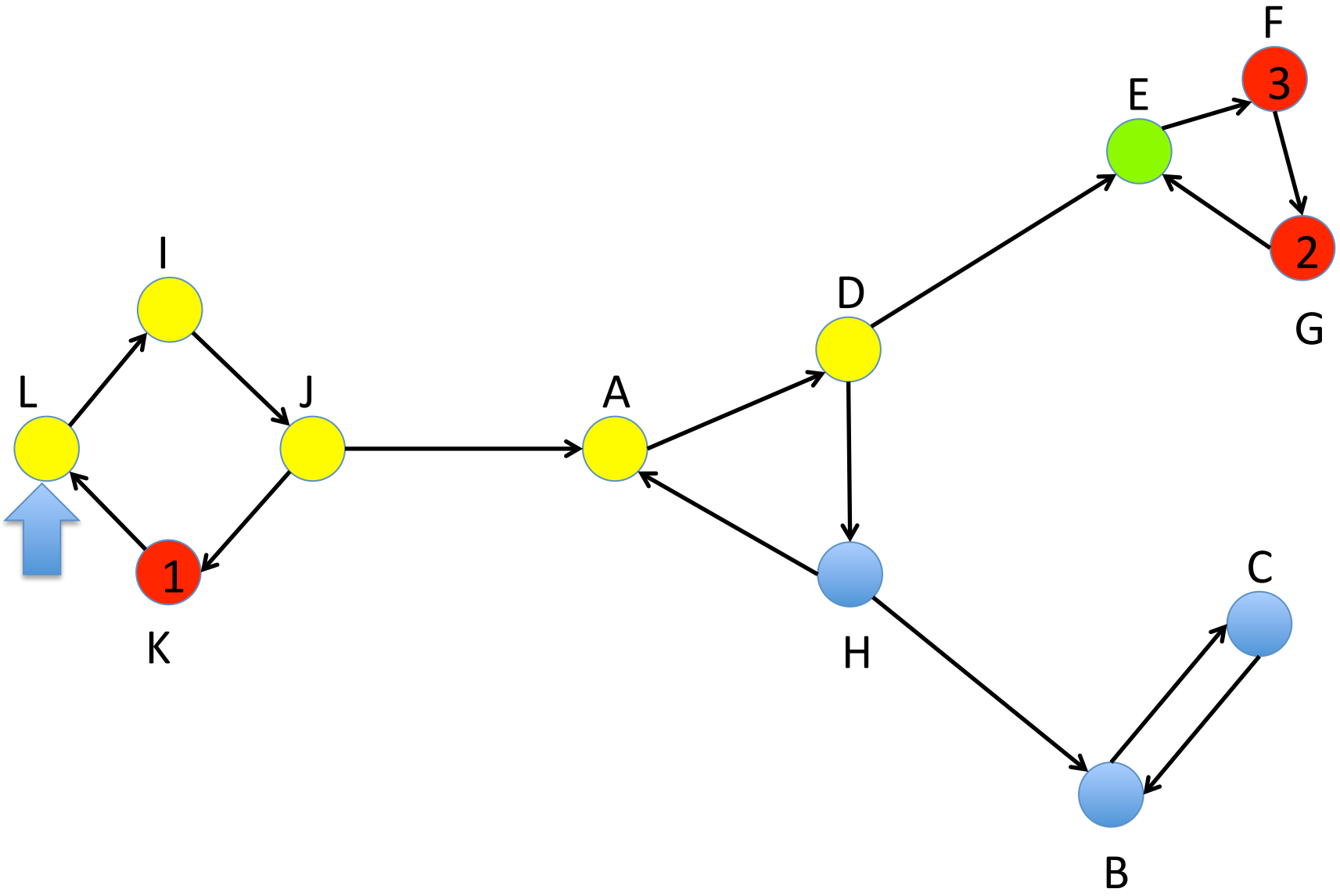
Let's Try DFS & Finishing Times (1)



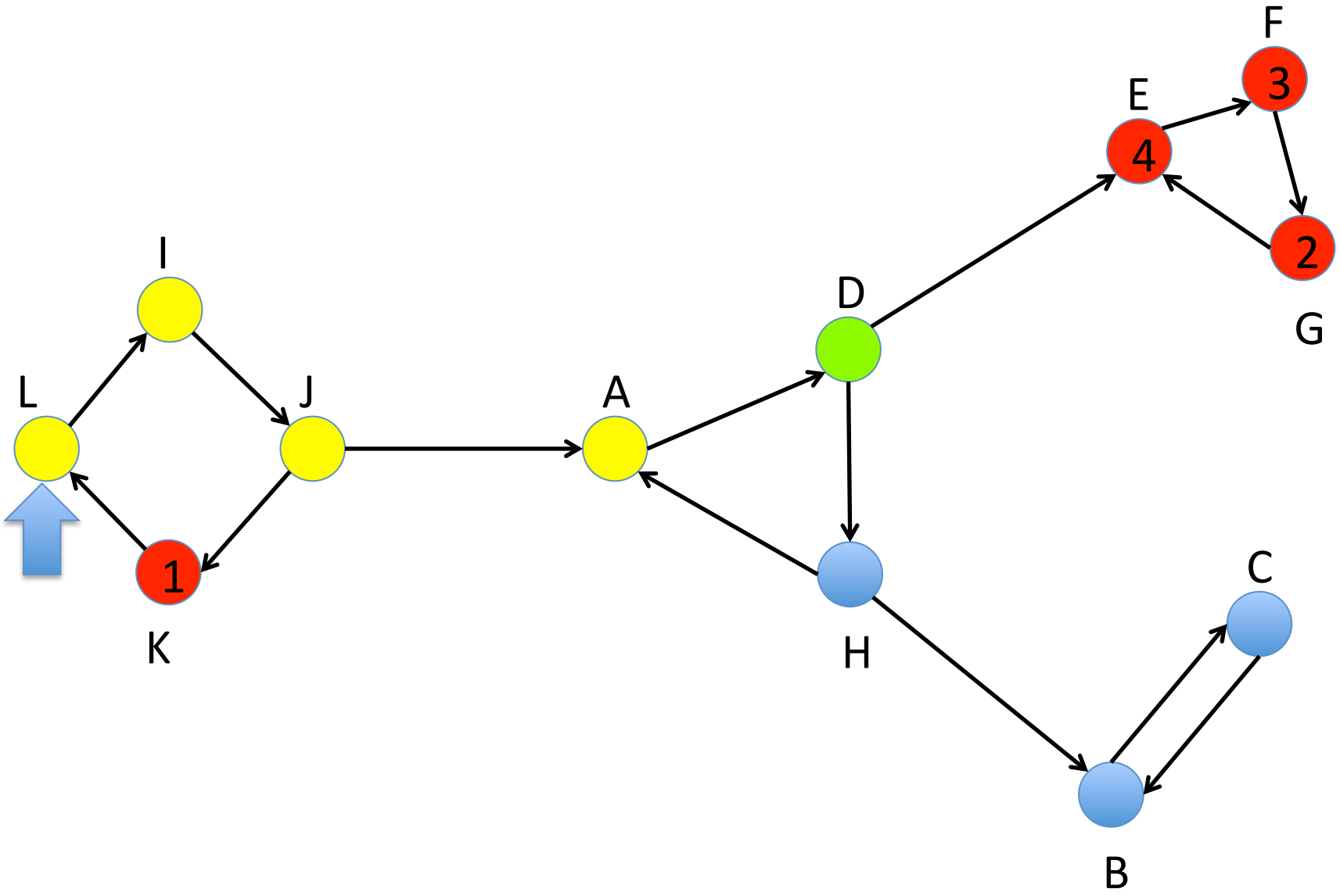
Let's Try DFS & Finishing Times (1)



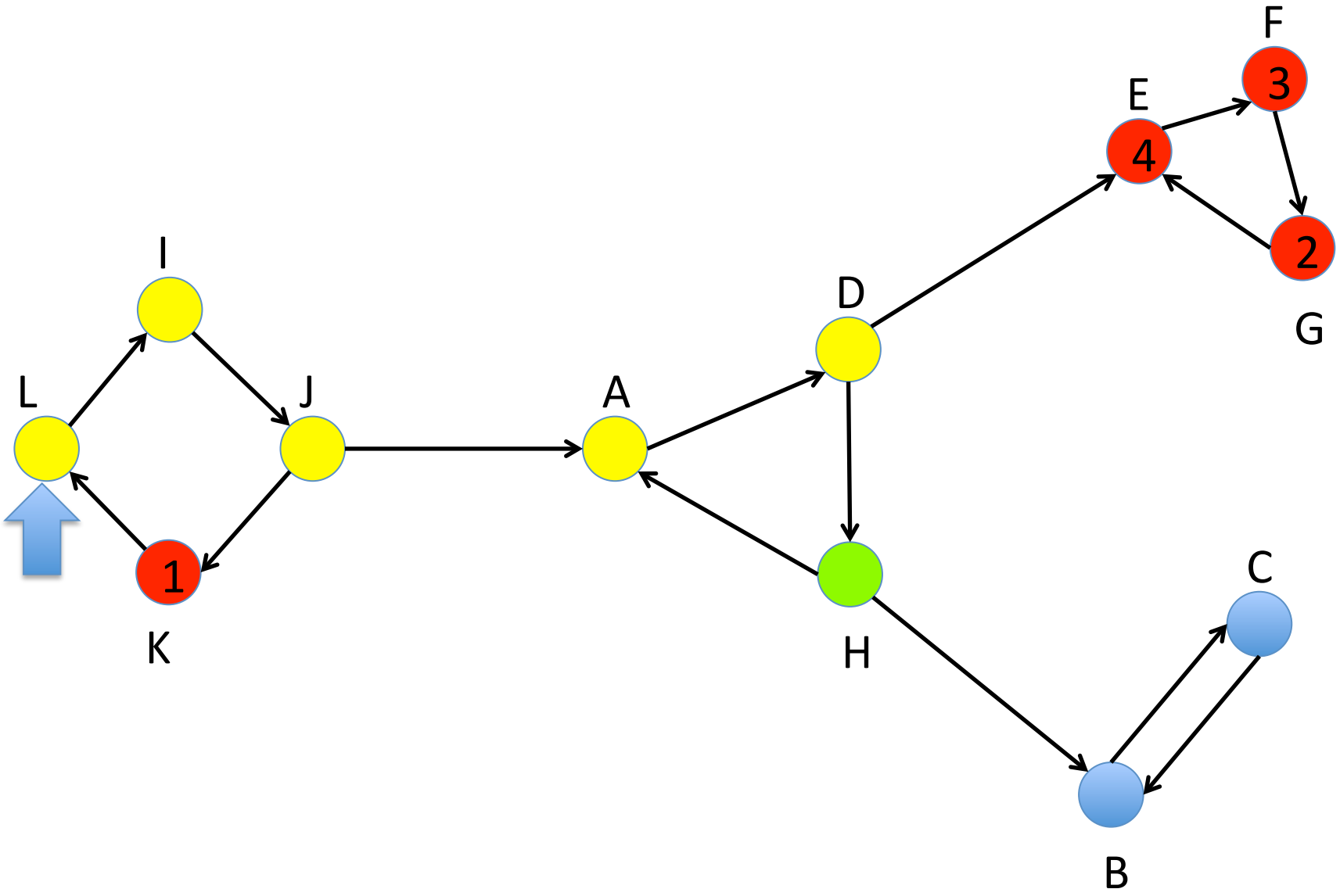
Let's Try DFS & Finishing Times (1)



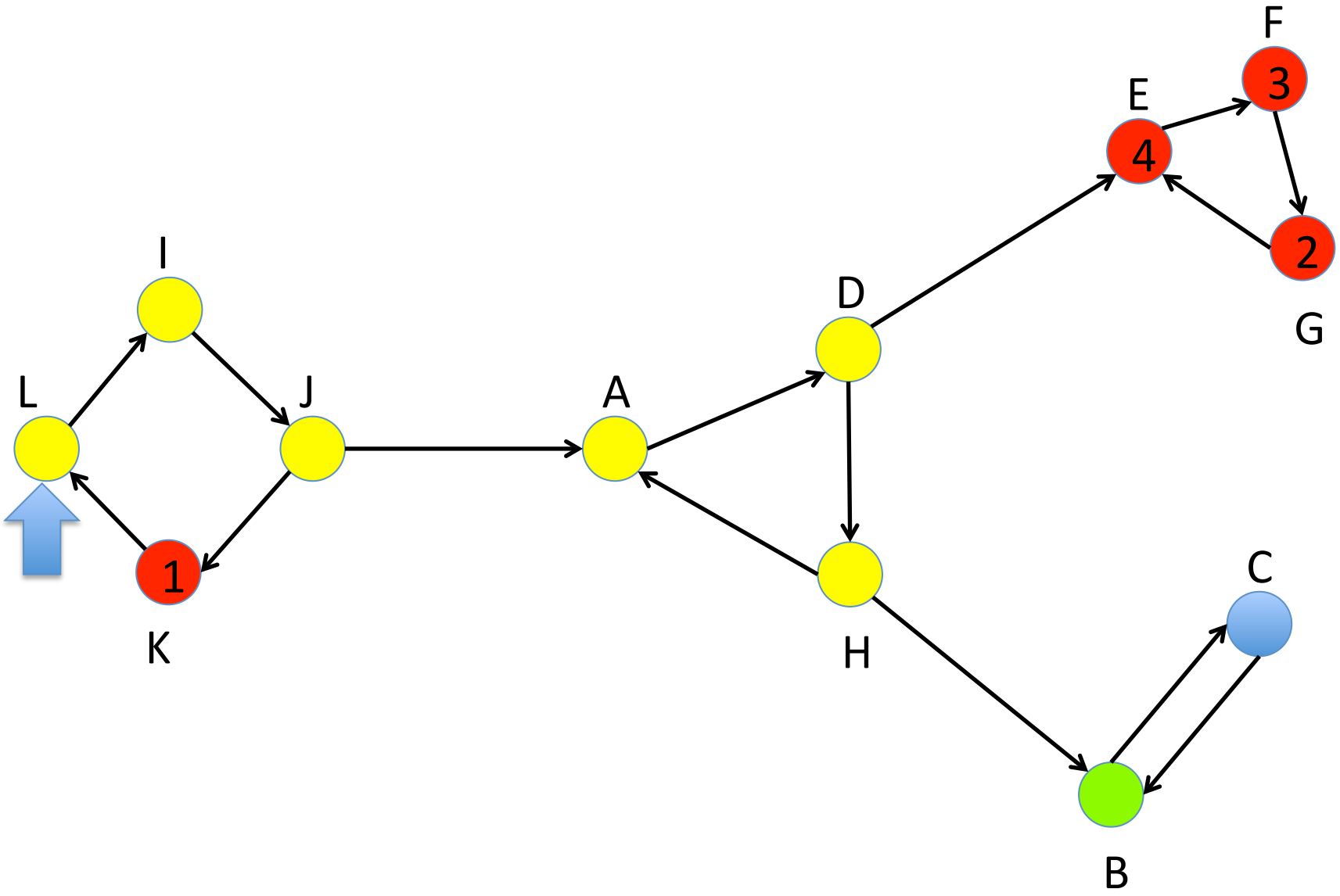
Let's Try DFS & Finishing Times (1)



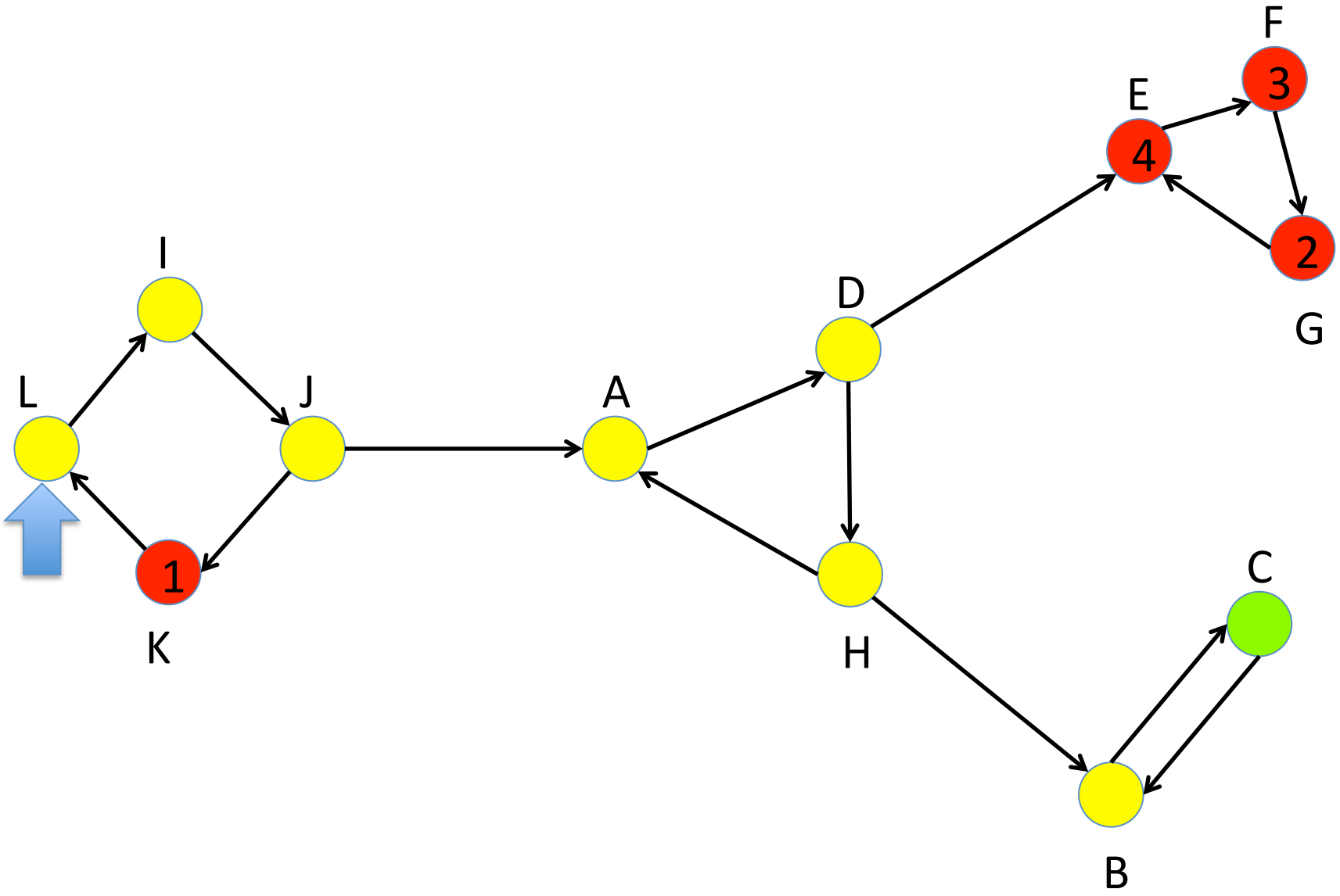
Let's Try DFS & Finishing Times (1)



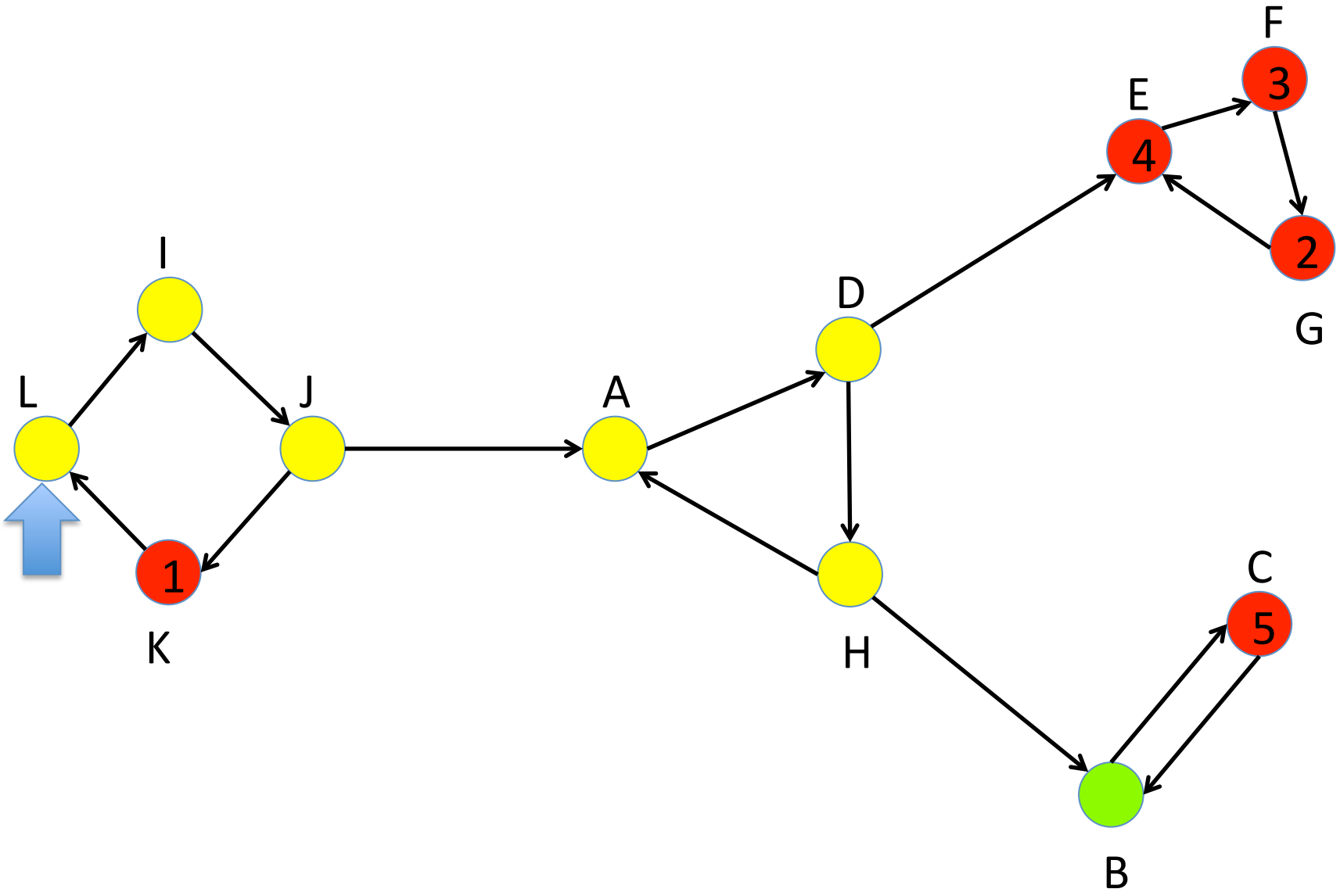
Let's Try DFS & Finishing Times (1)



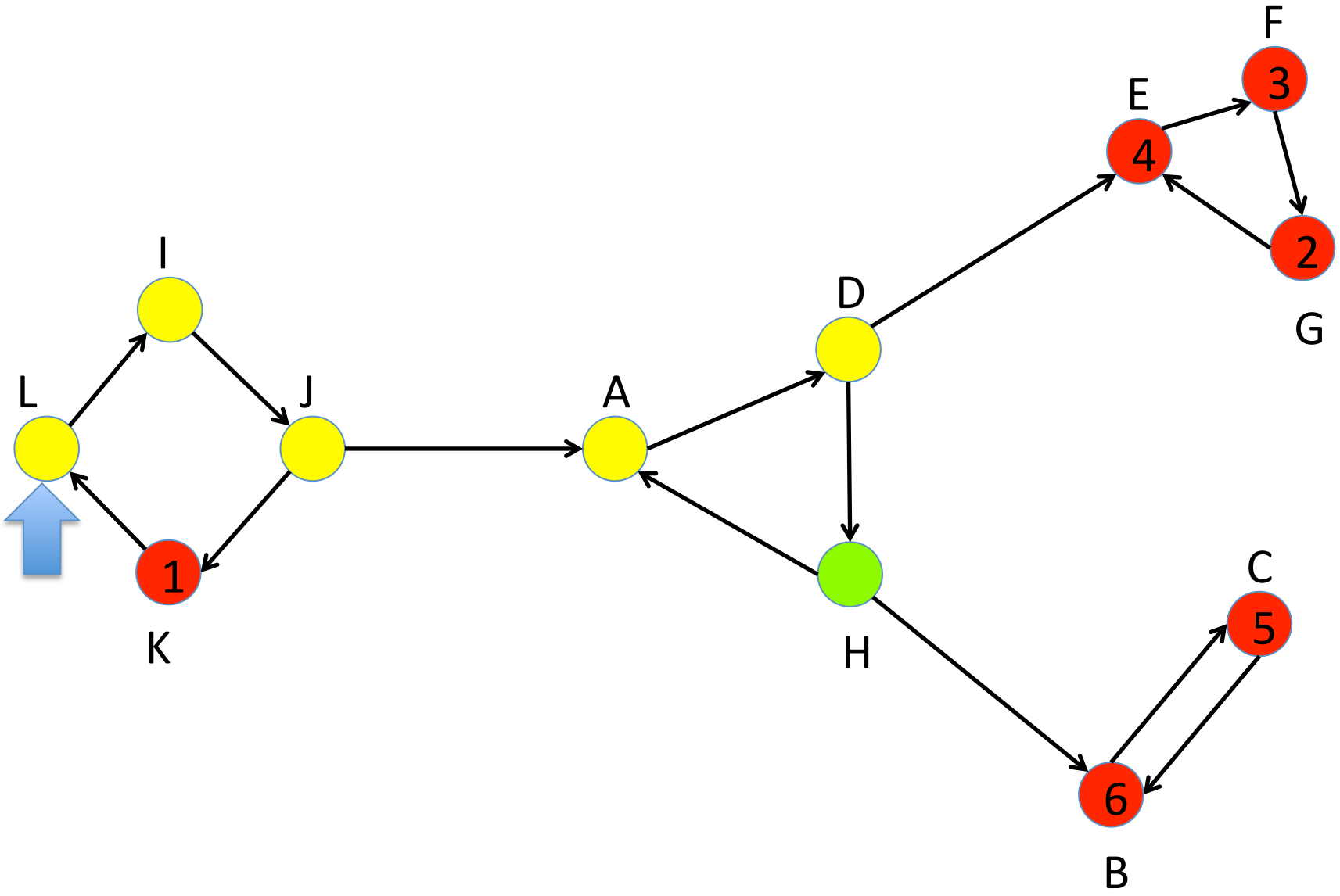
Let's Try DFS & Finishing Times (1)



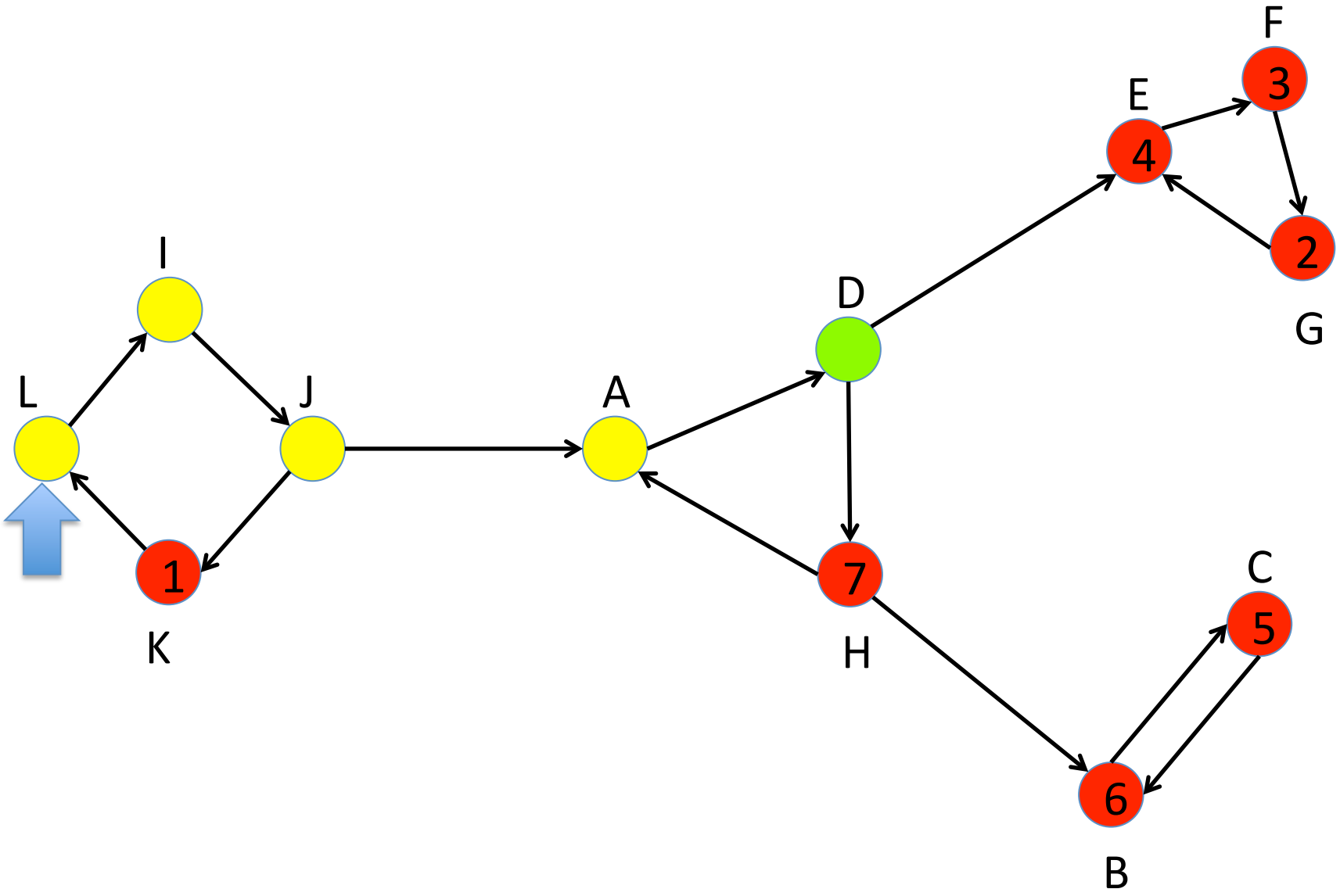
Let's Try DFS & Finishing Times (1)



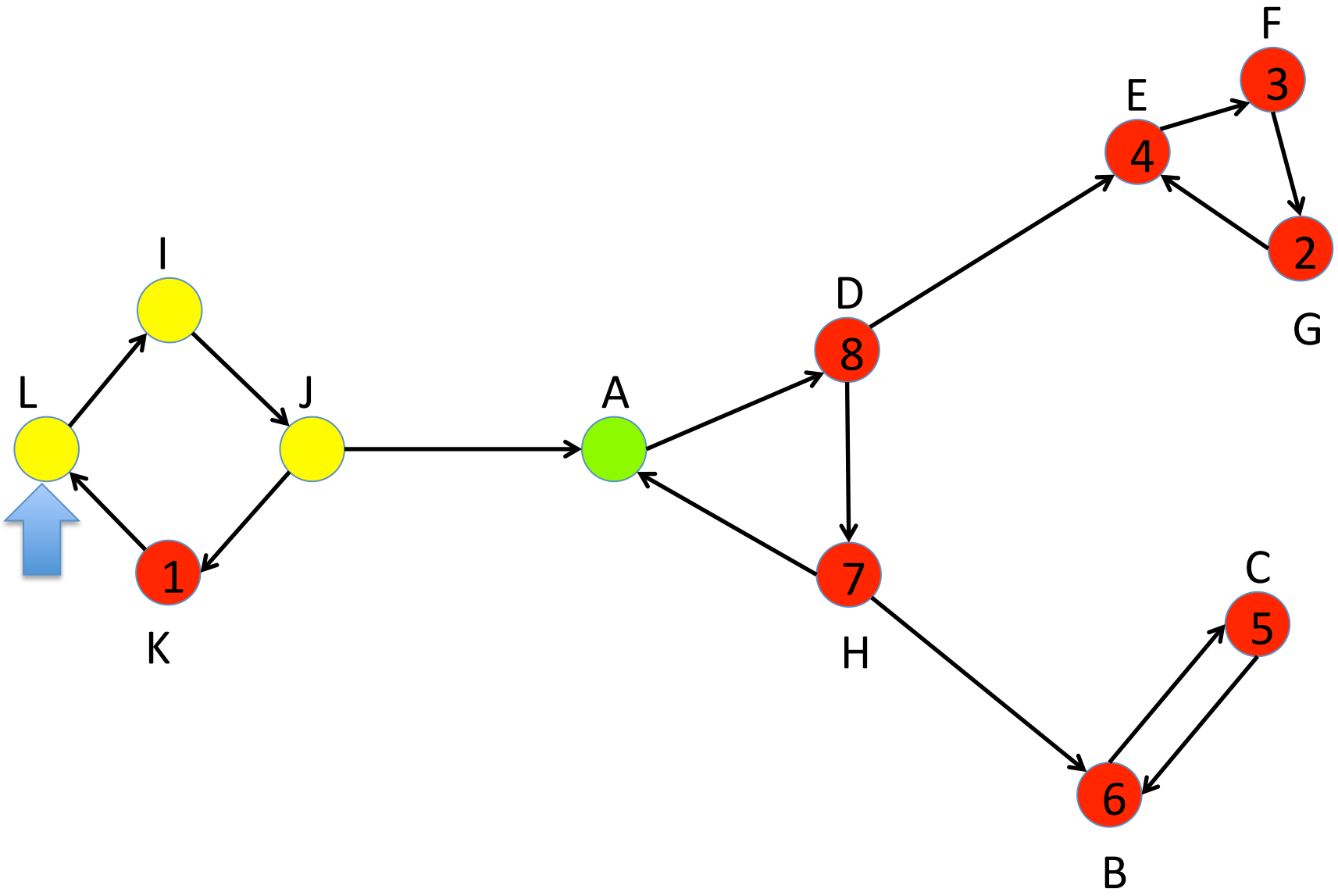
Let's Try DFS & Finishing Times (1)



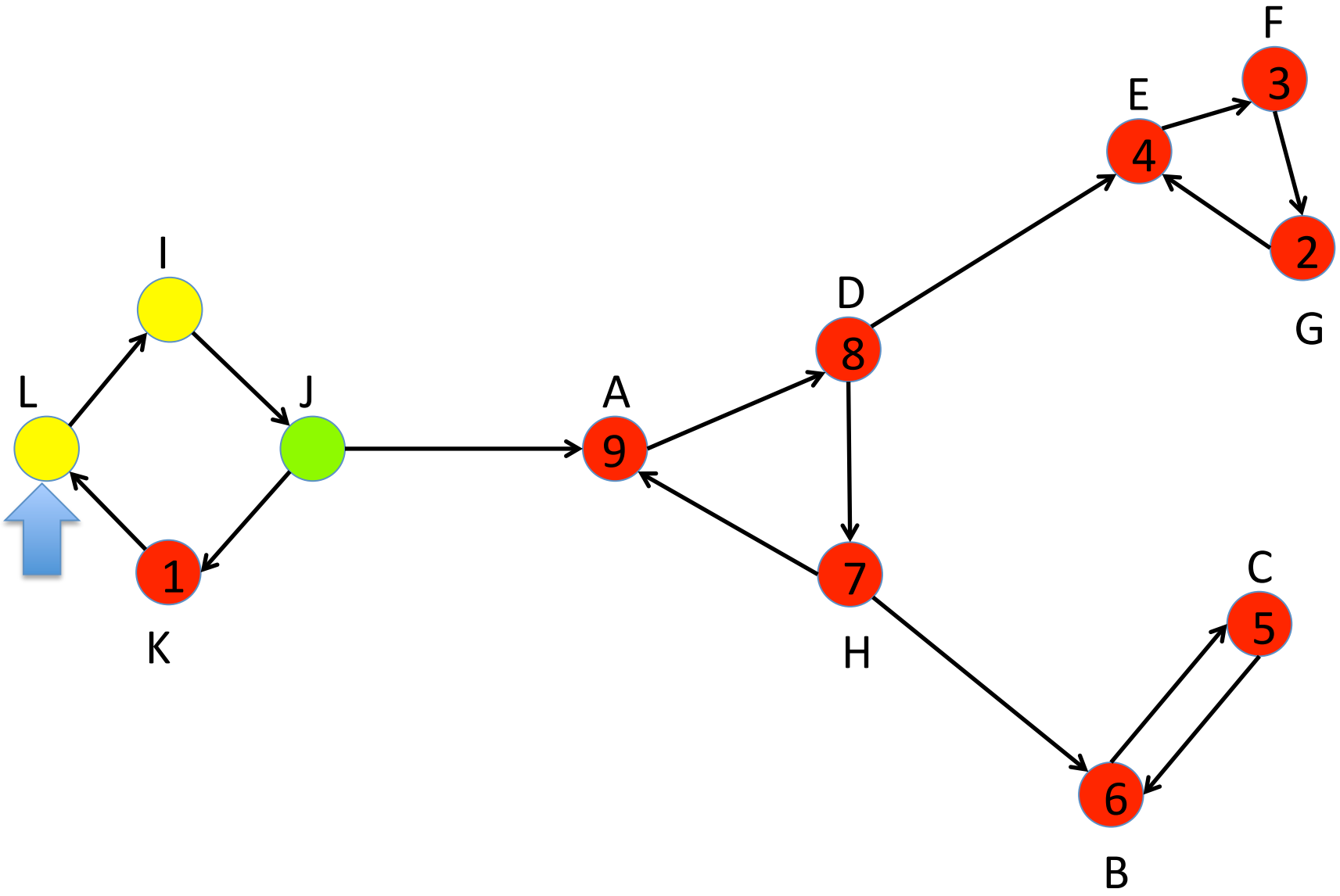
Let's Try DFS & Finishing Times (1)



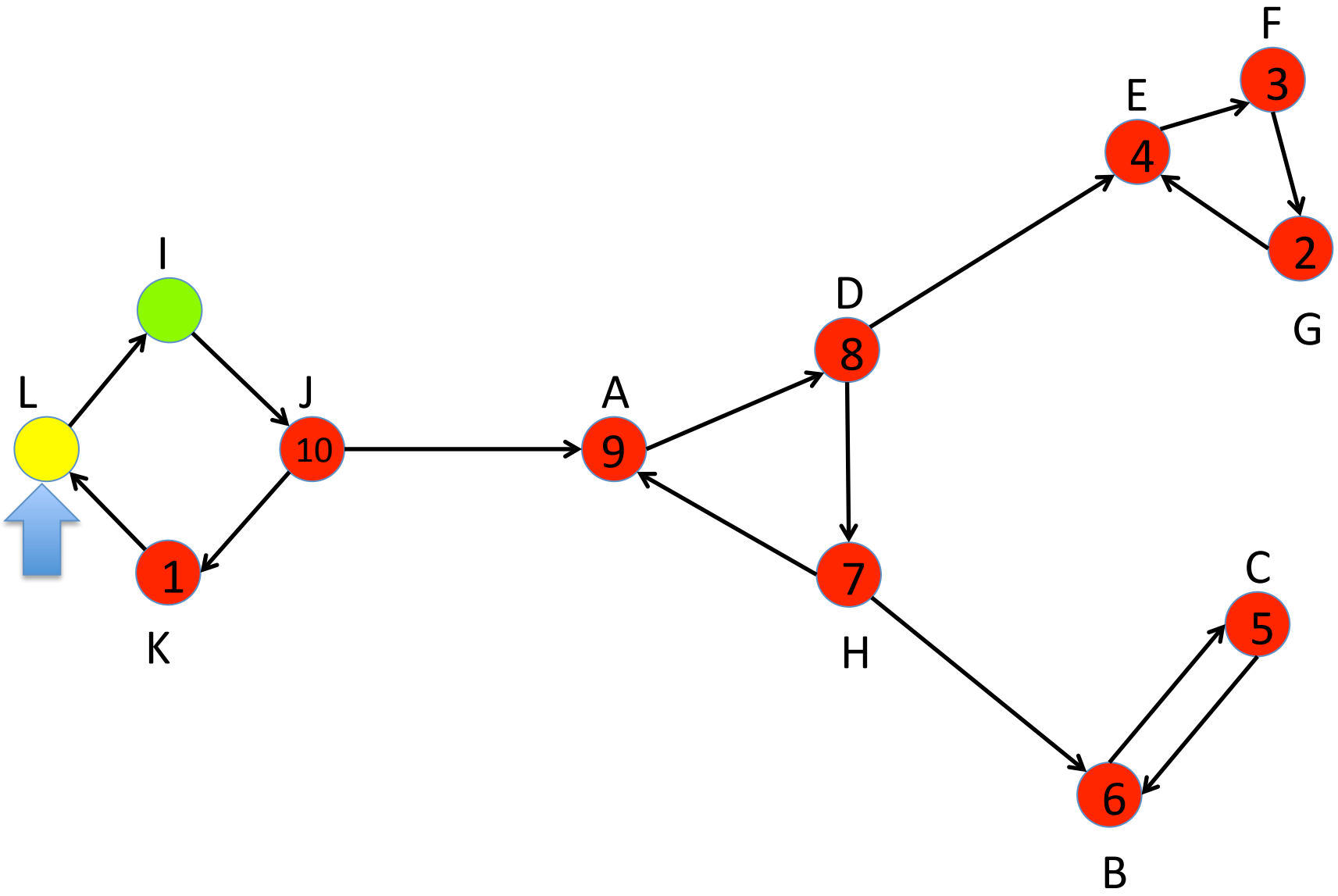
Let's Try DFS & Finishing Times (1)



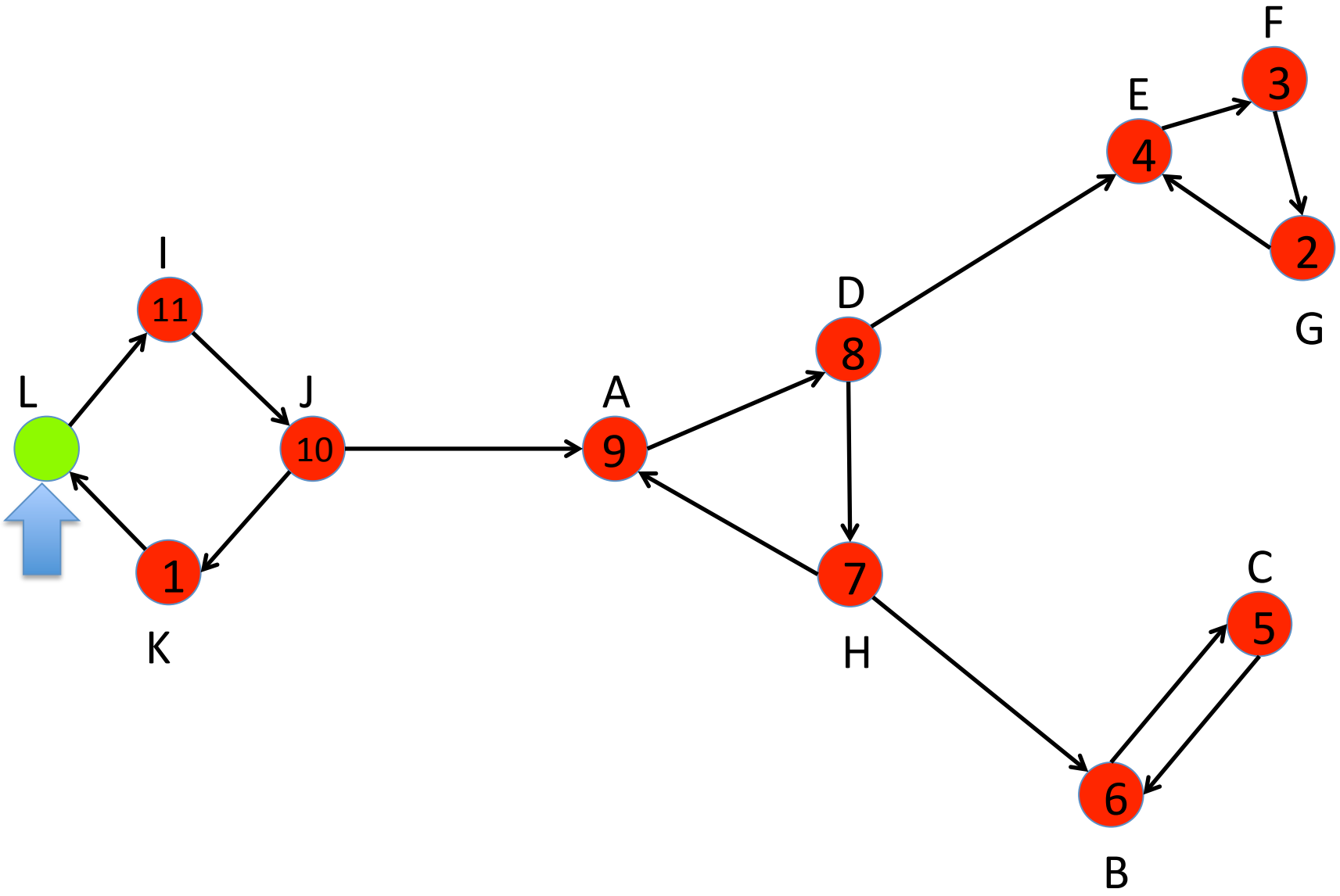
Let's Try DFS & Finishing Times (1)



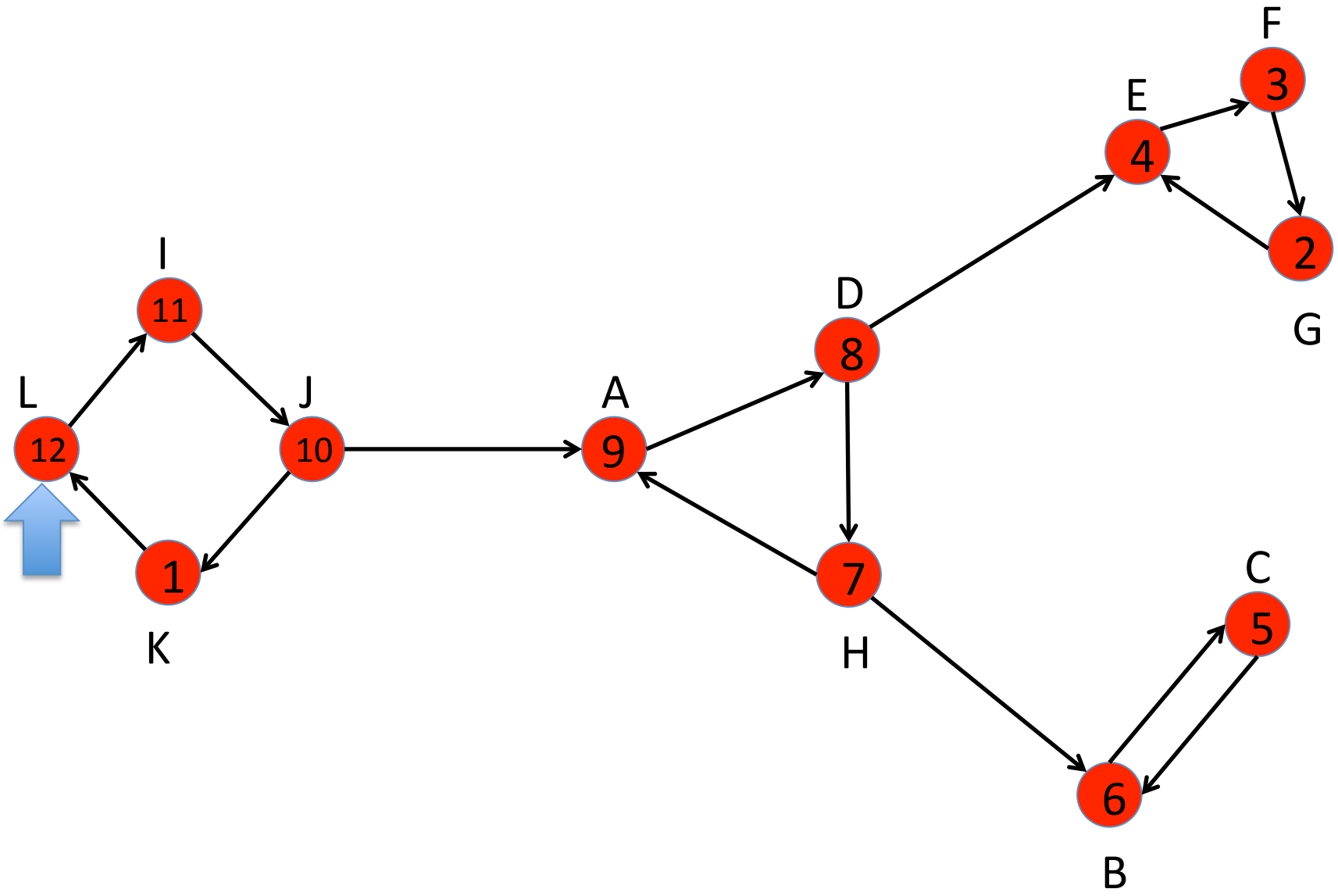
Let's Try DFS & Finishing Times (1)



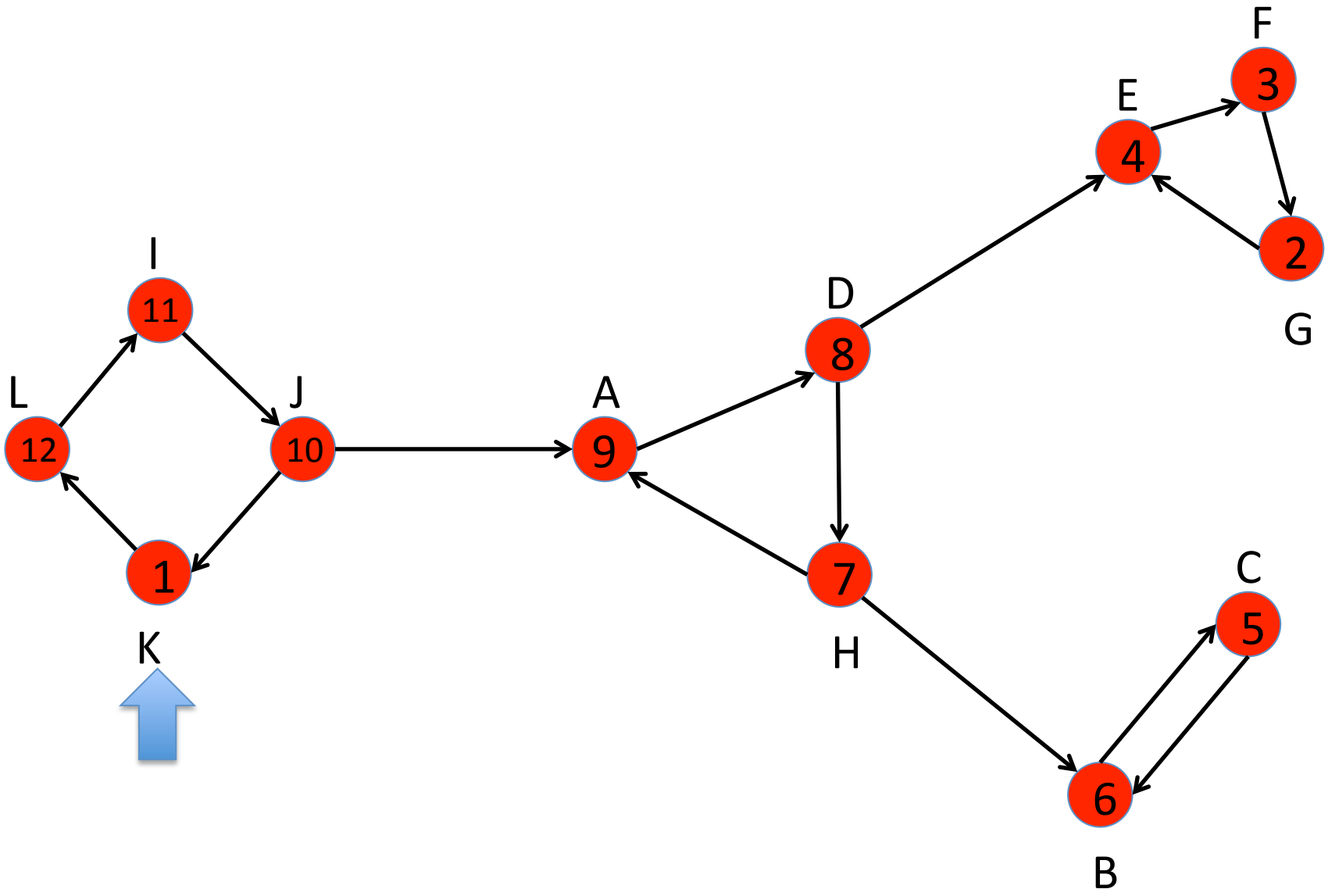
Let's Try DFS & Finishing Times (1)



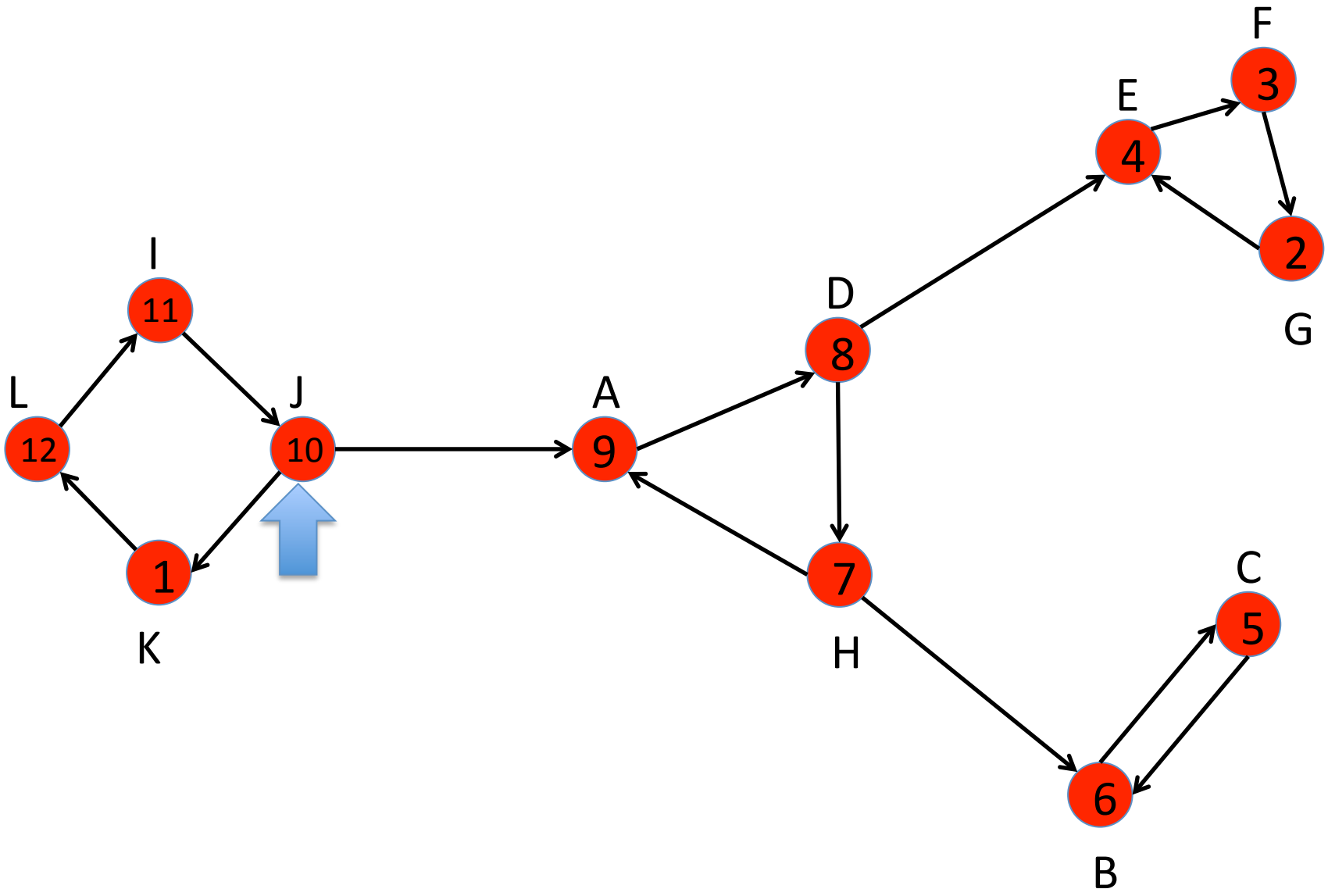
Let's Try DFS & Finishing Times (1)



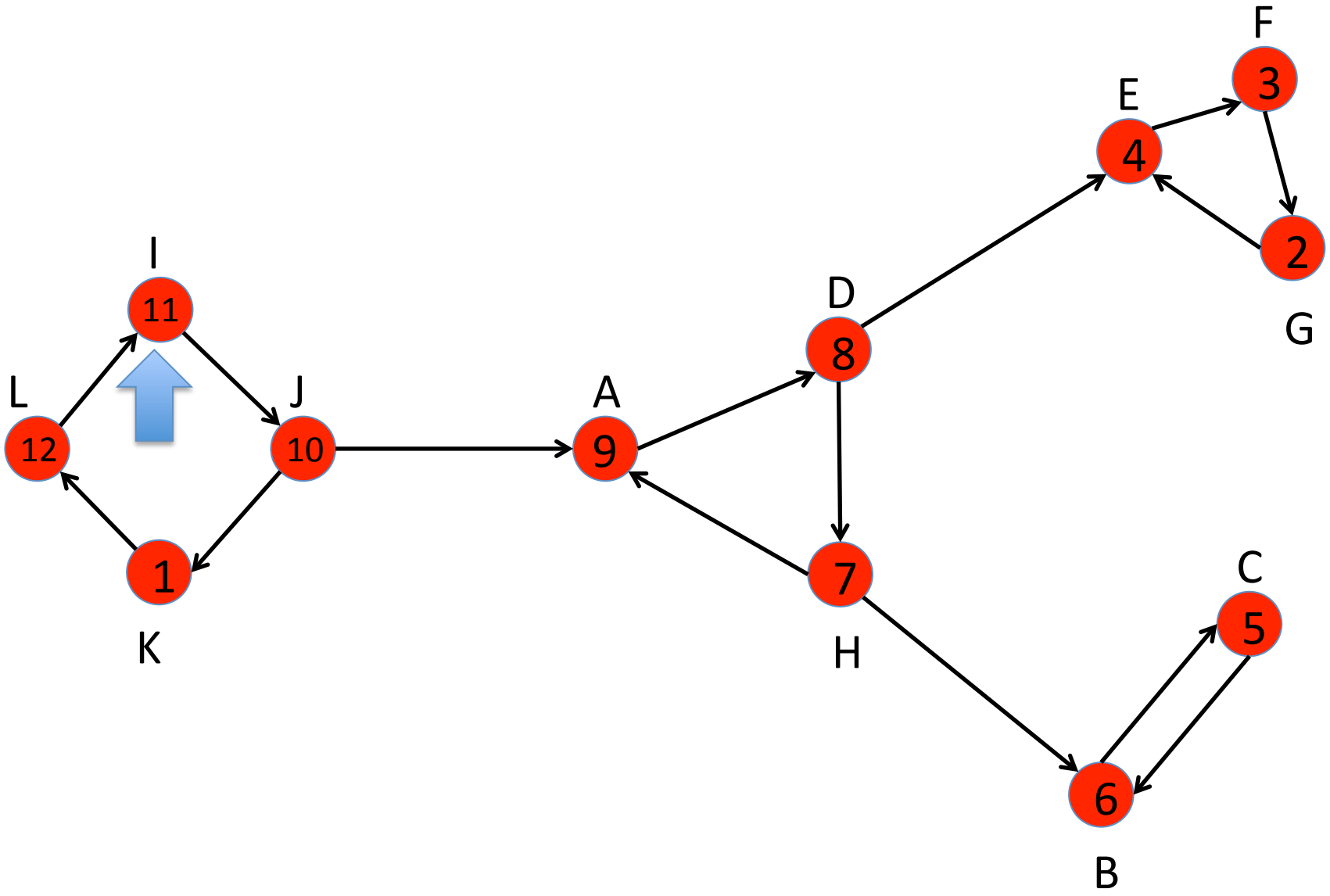
Let's Try DFS & Finishing Times (1)



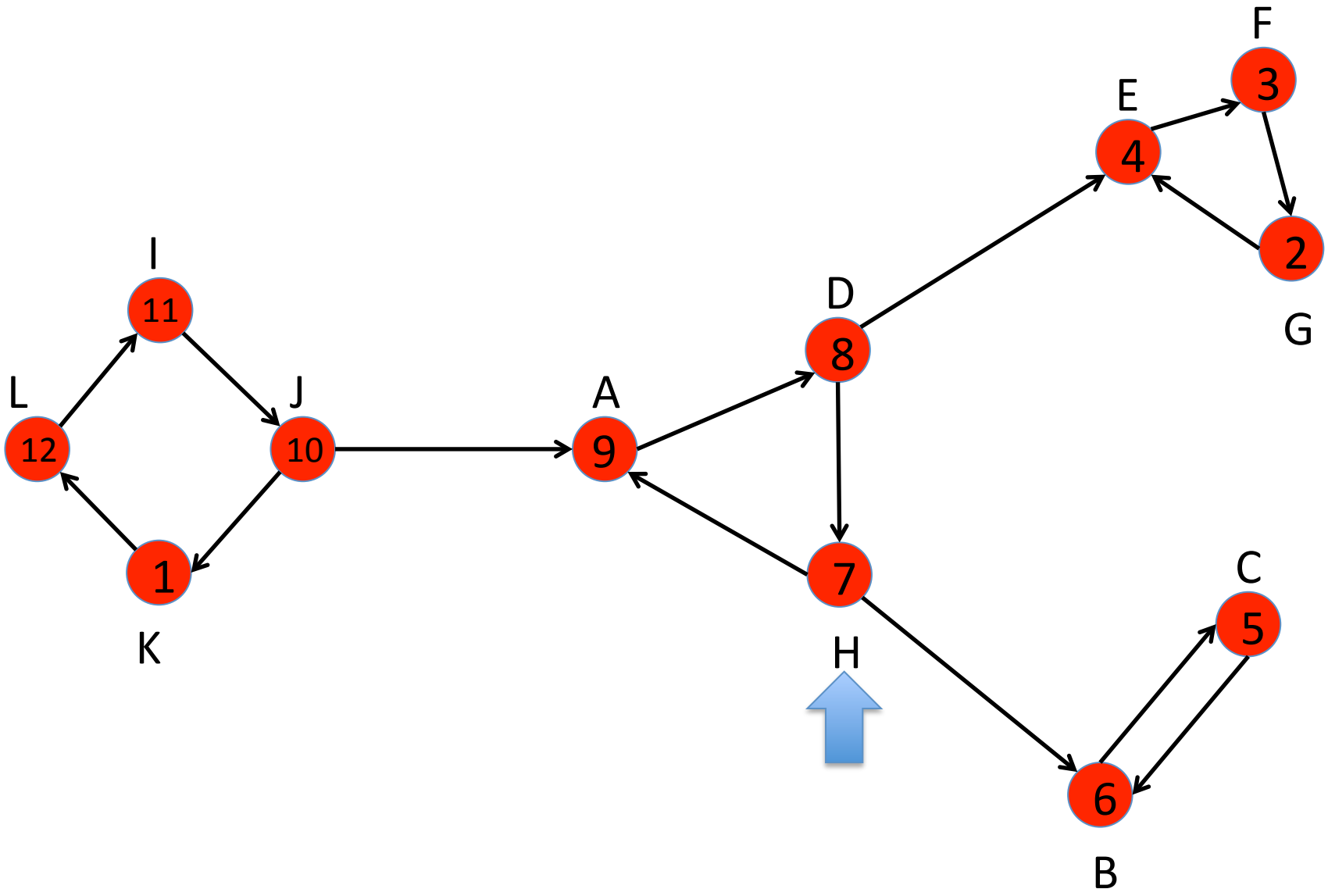
Let's Try DFS & Finishing Times (1)



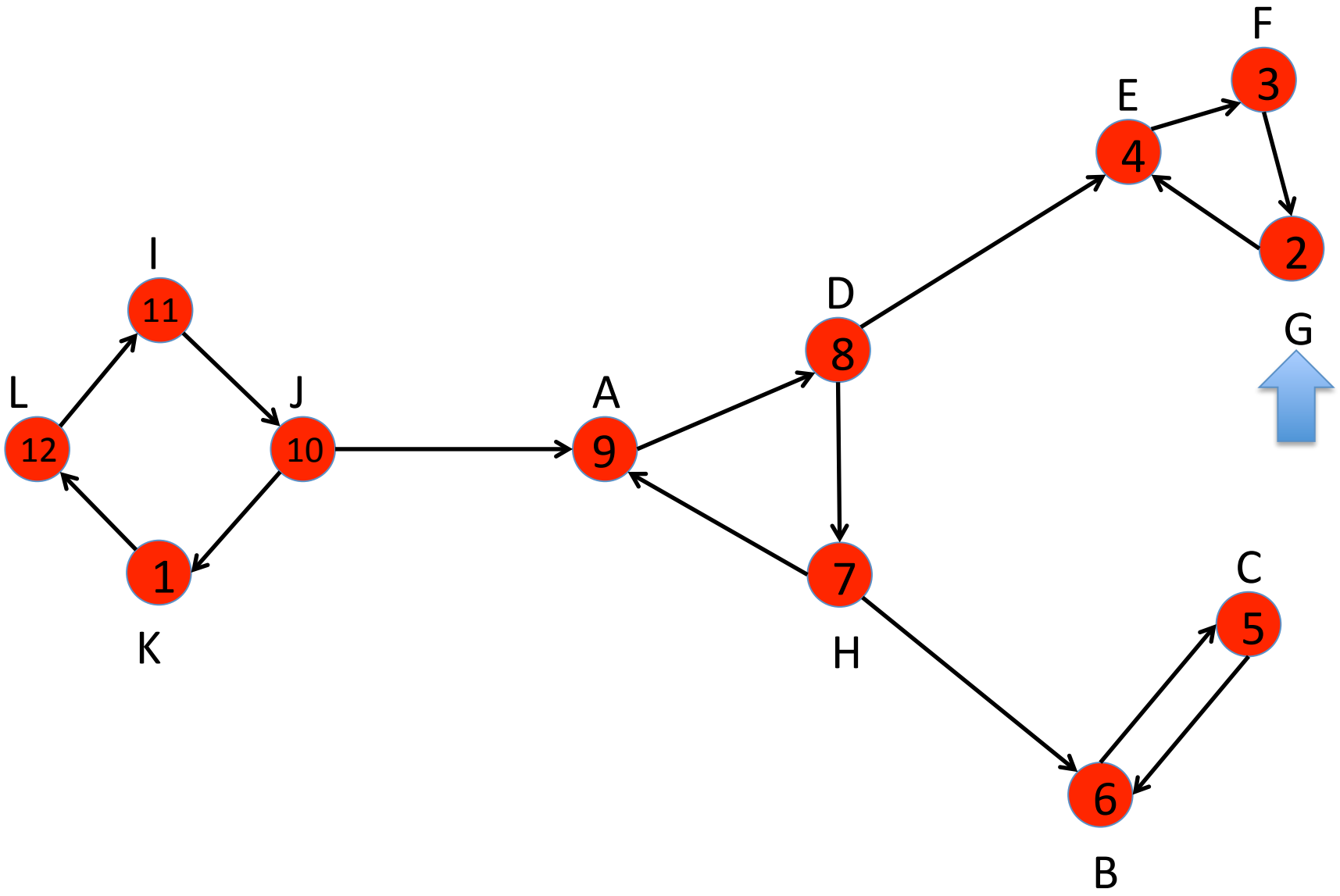
Let's Try DFS & Finishing Times (1)



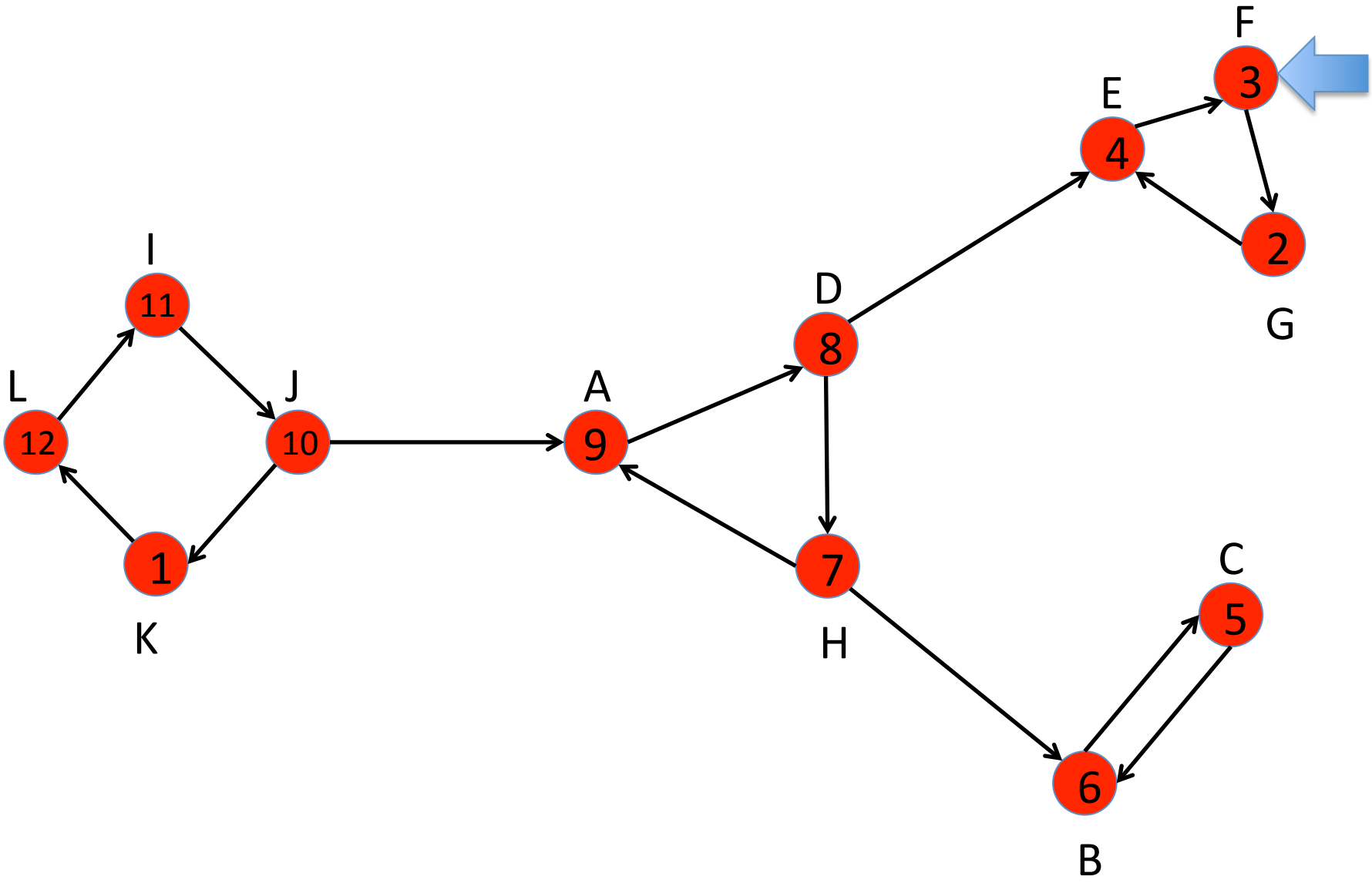
Let's Try DFS & Finishing Times (1)



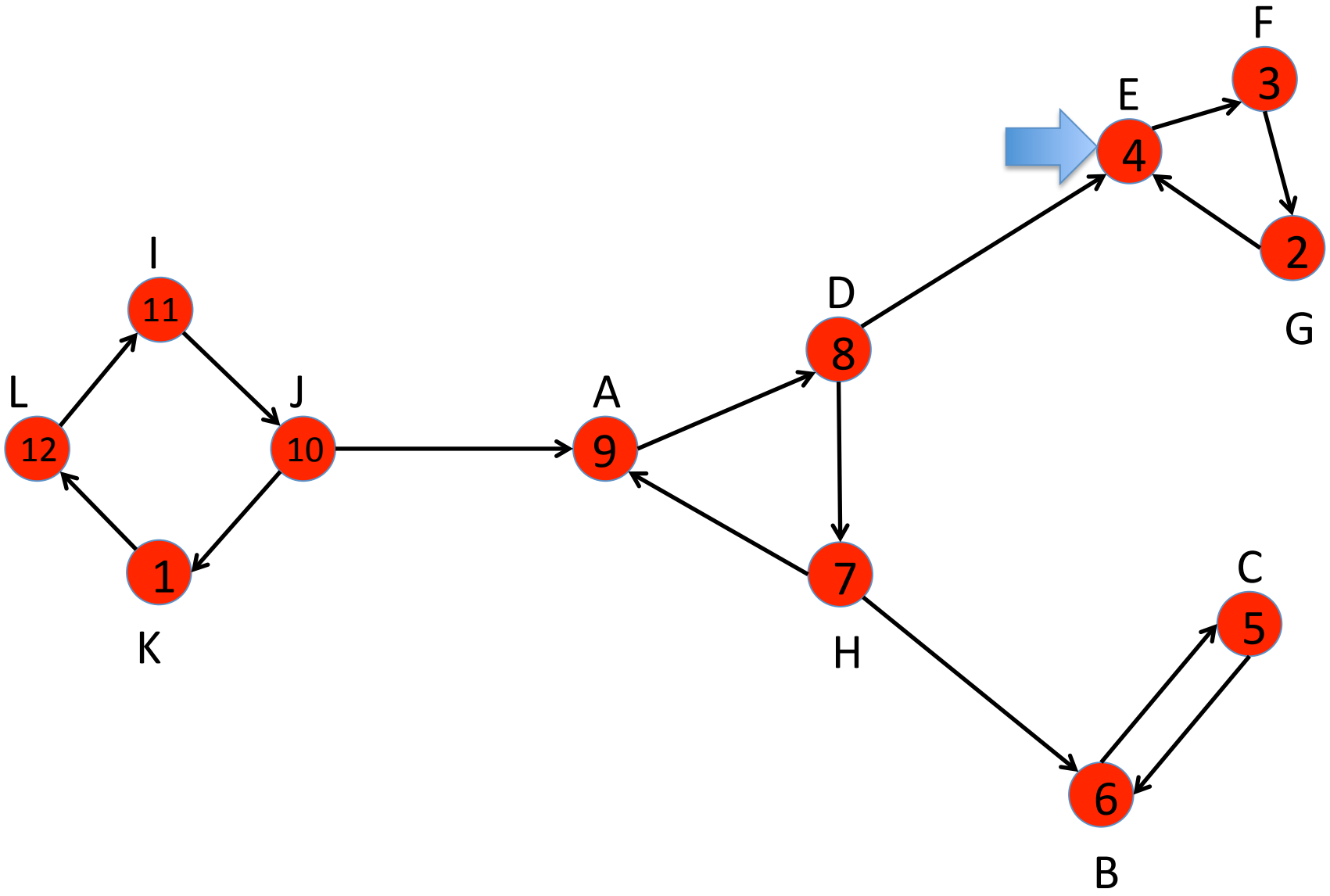
Let's Try DFS & Finishing Times (1)



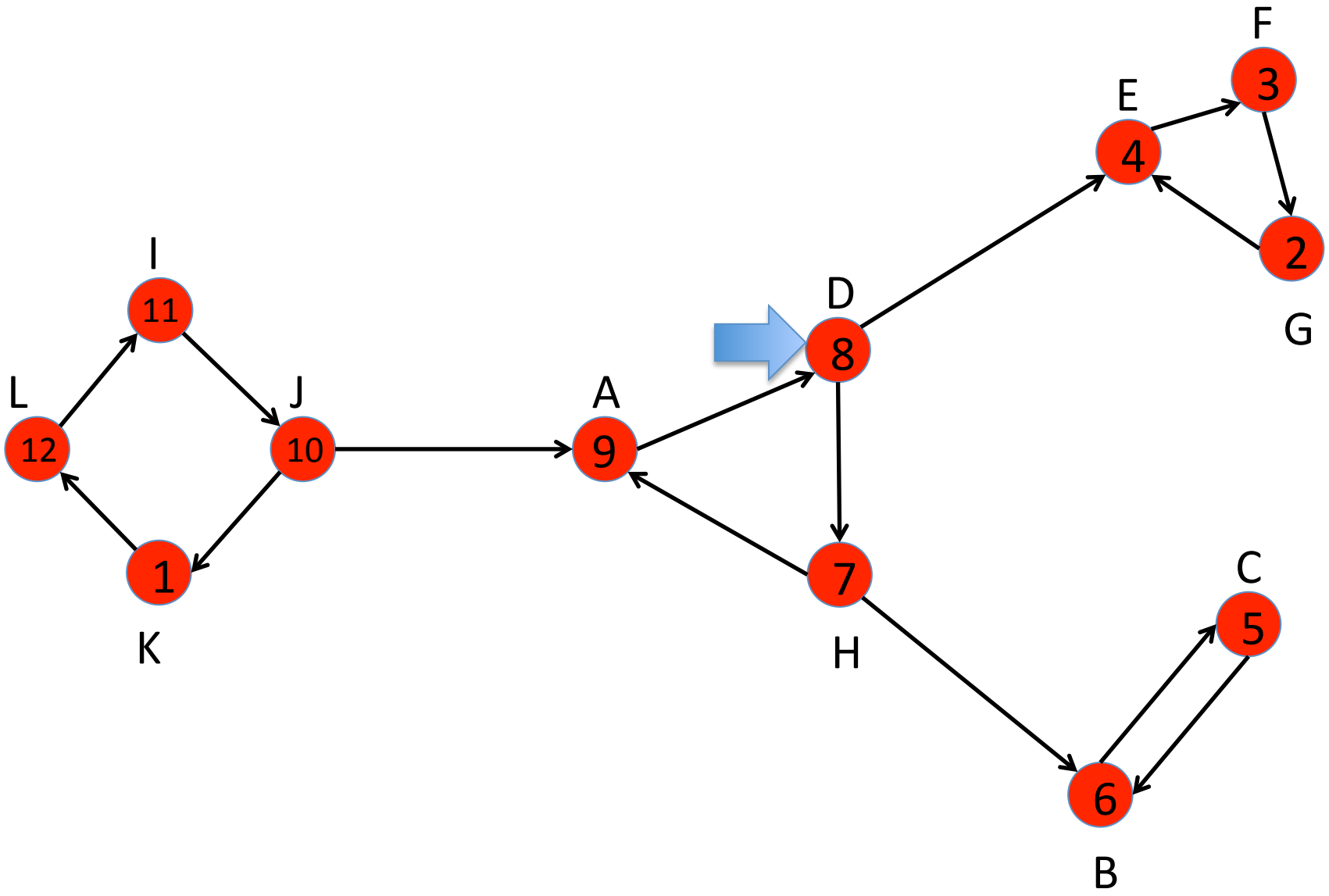
Let's Try DFS & Finishing Times (1)



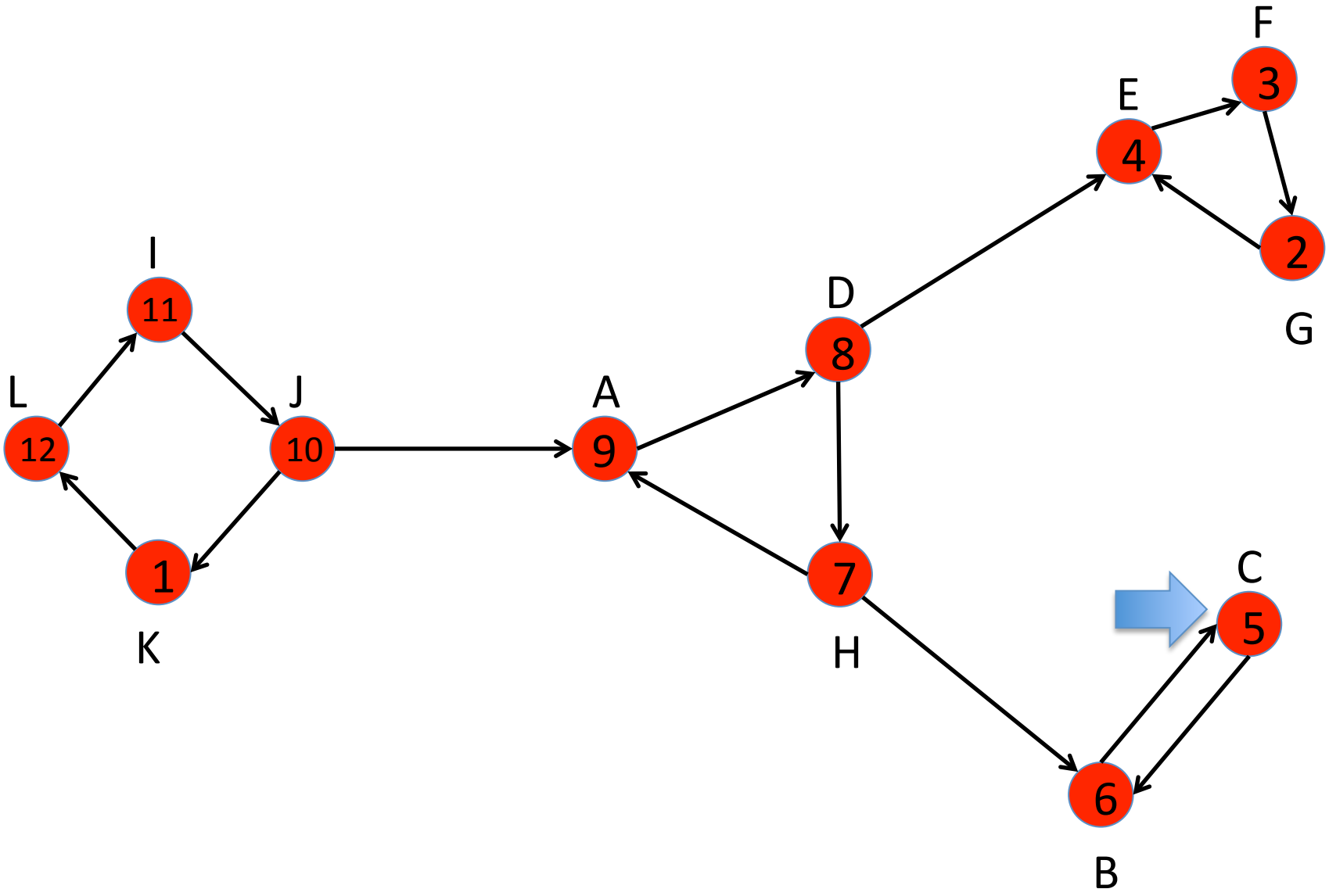
Let's Try DFS & Finishing Times (1)



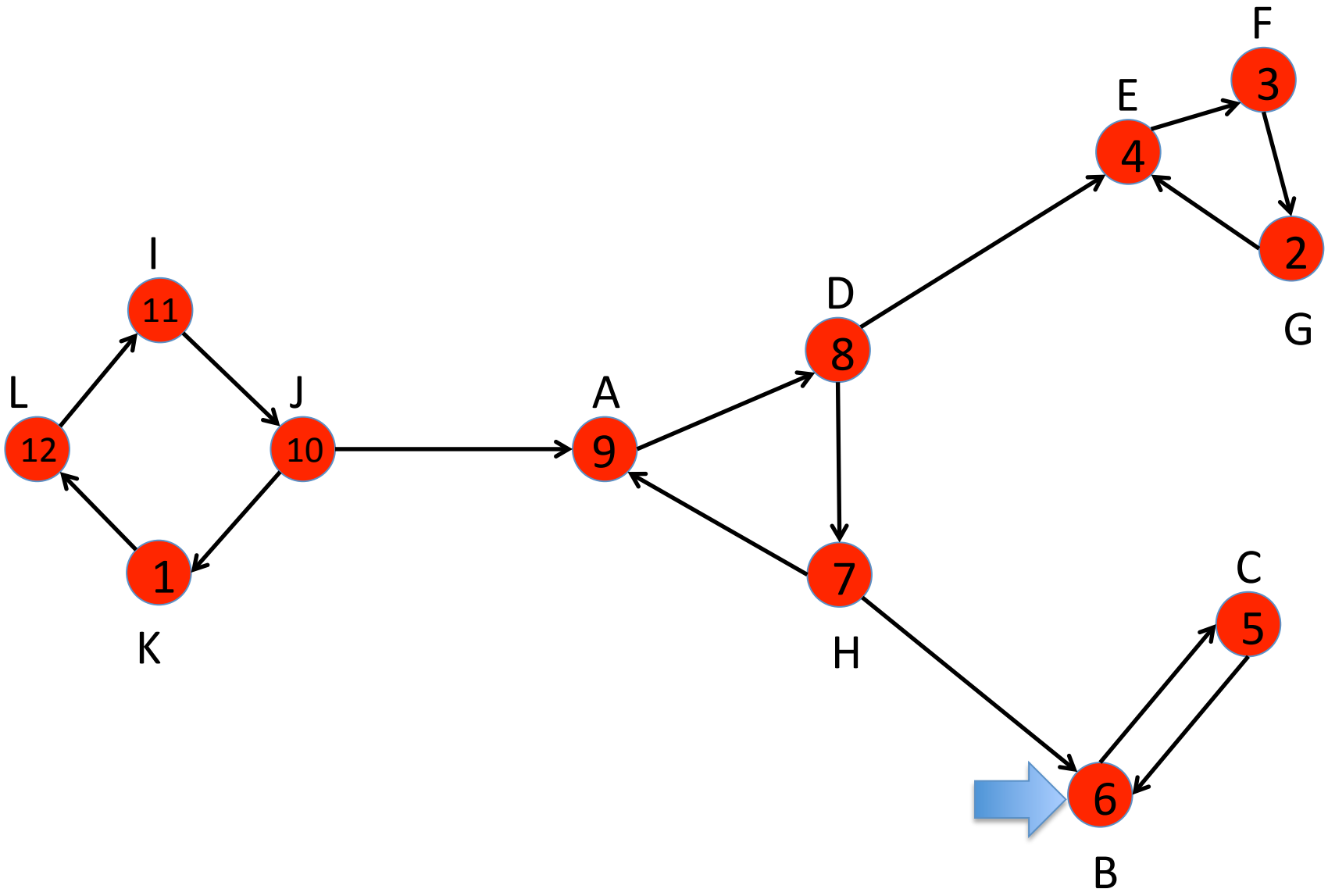
Let's Try DFS & Finishing Times (1)



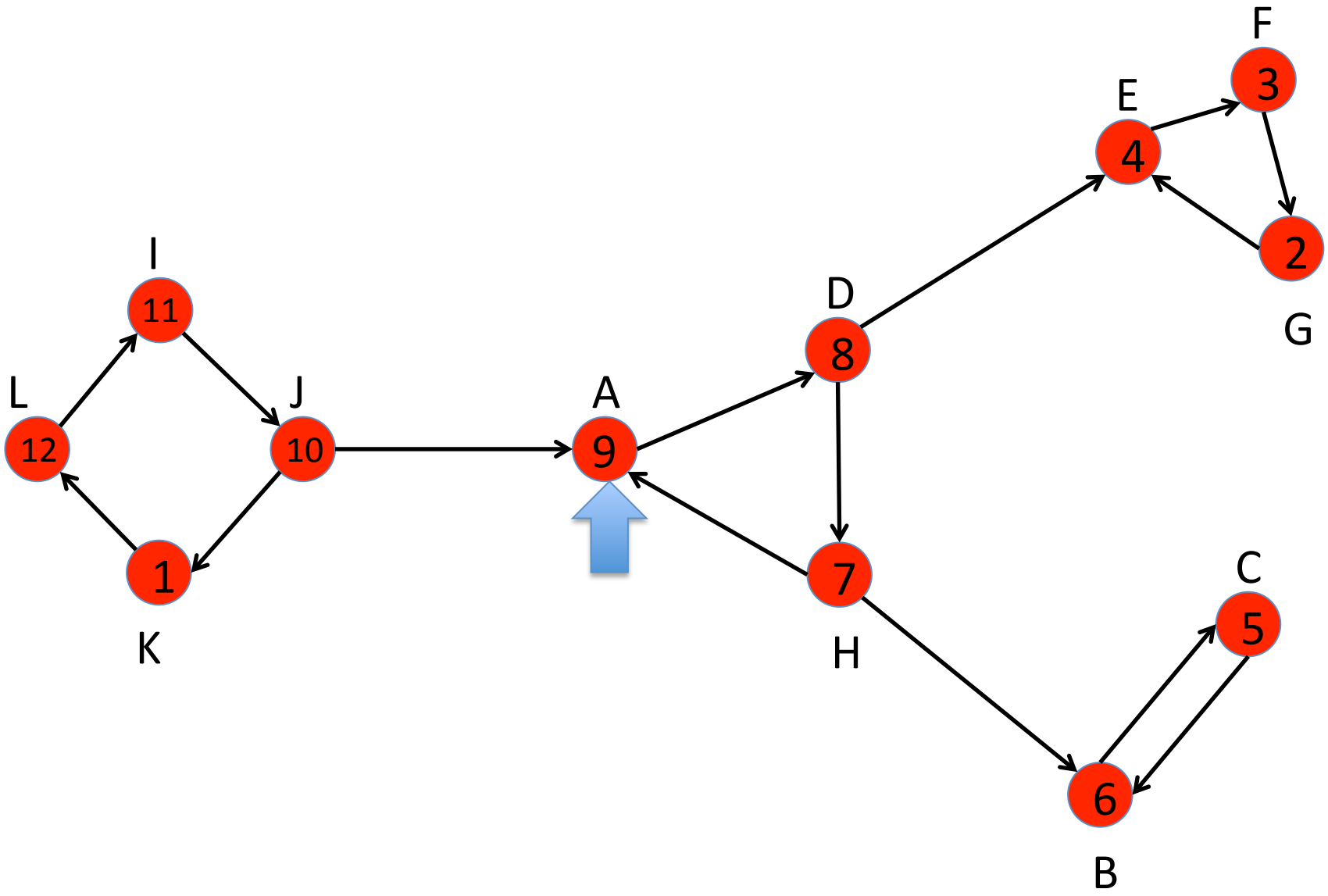
Let's Try DFS & Finishing Times (1)



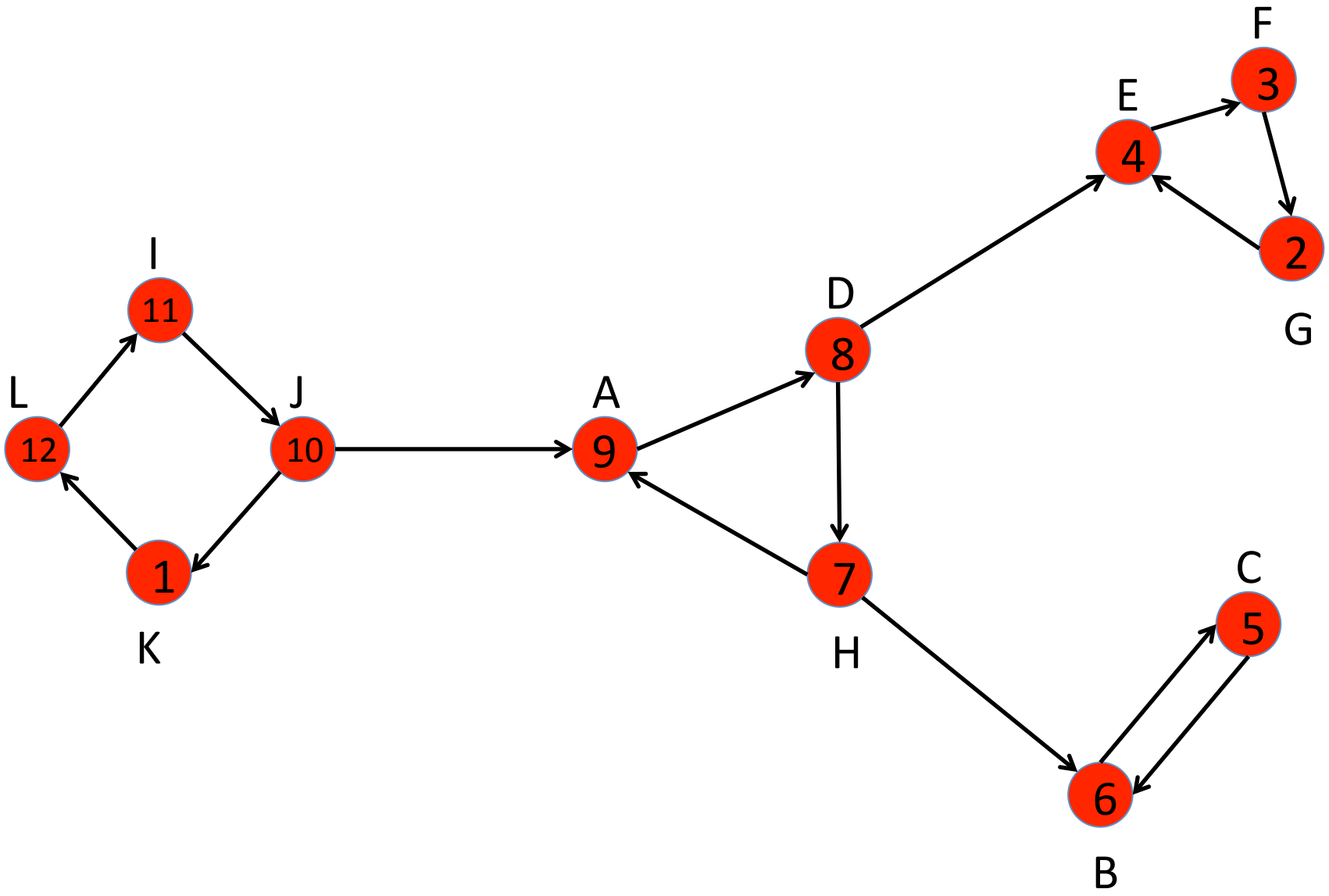
Let's Try DFS & Finishing Times (1)



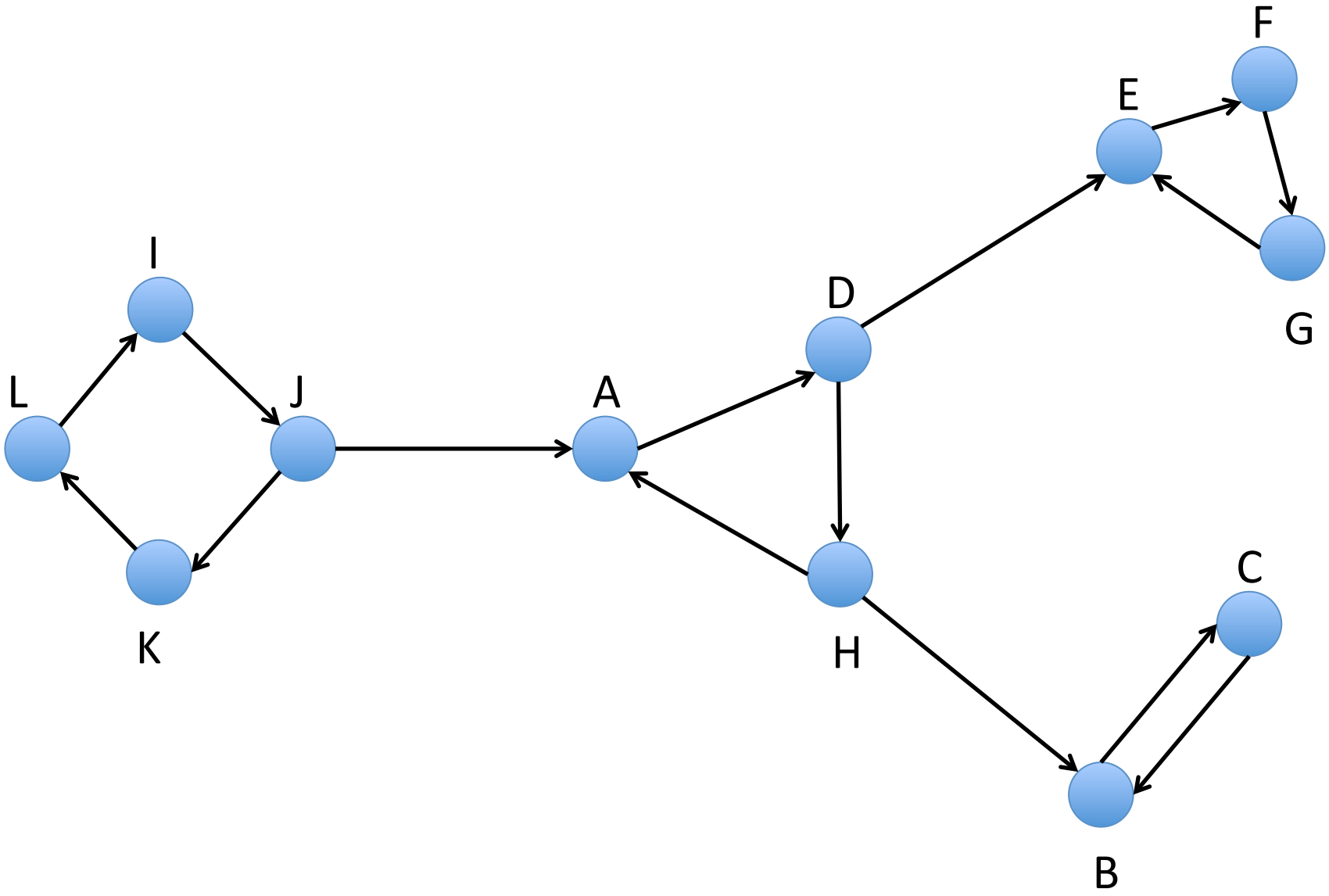
Let's Try DFS & Finishing Times (1)



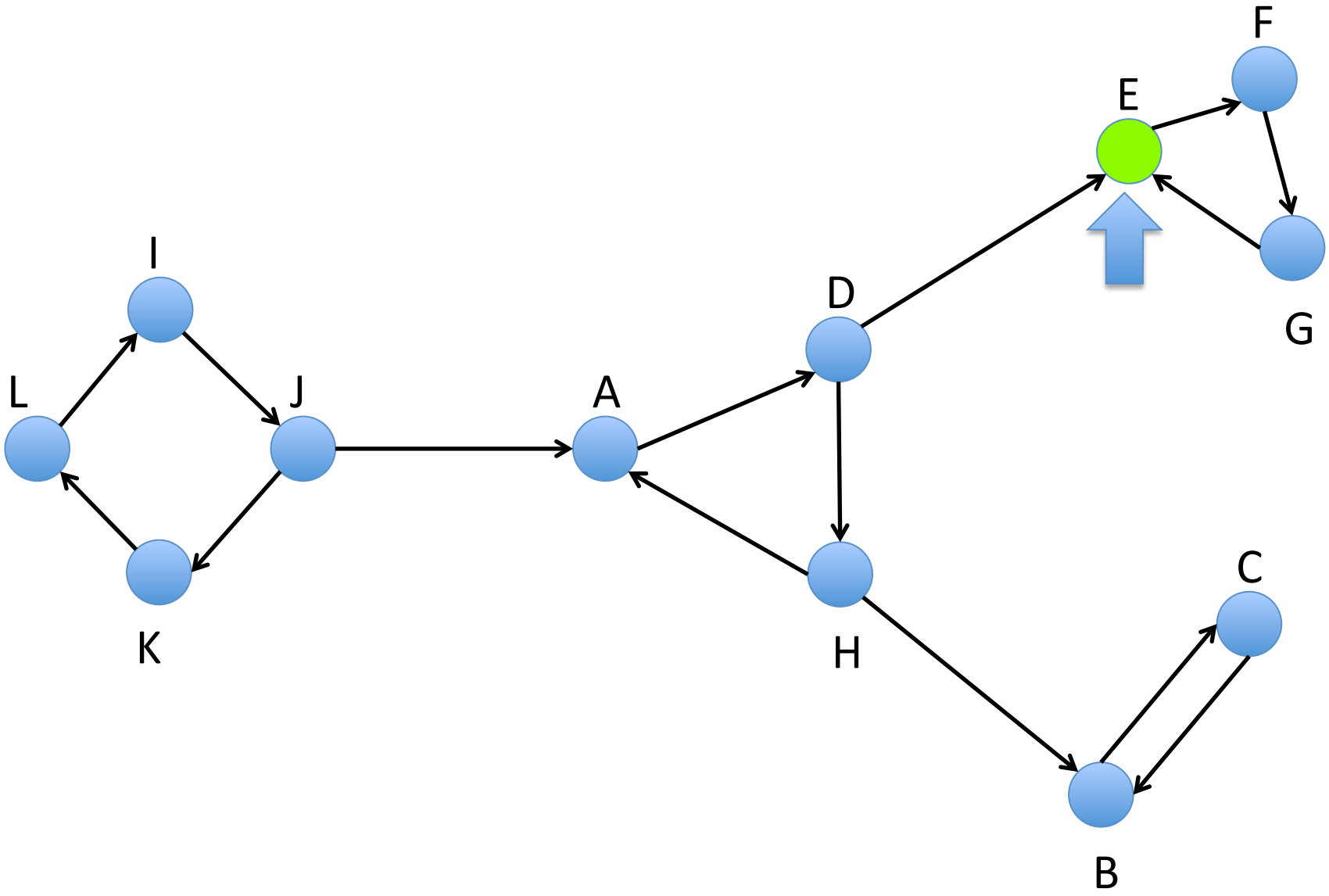
Let's Try DFS & Finishing Times (1)



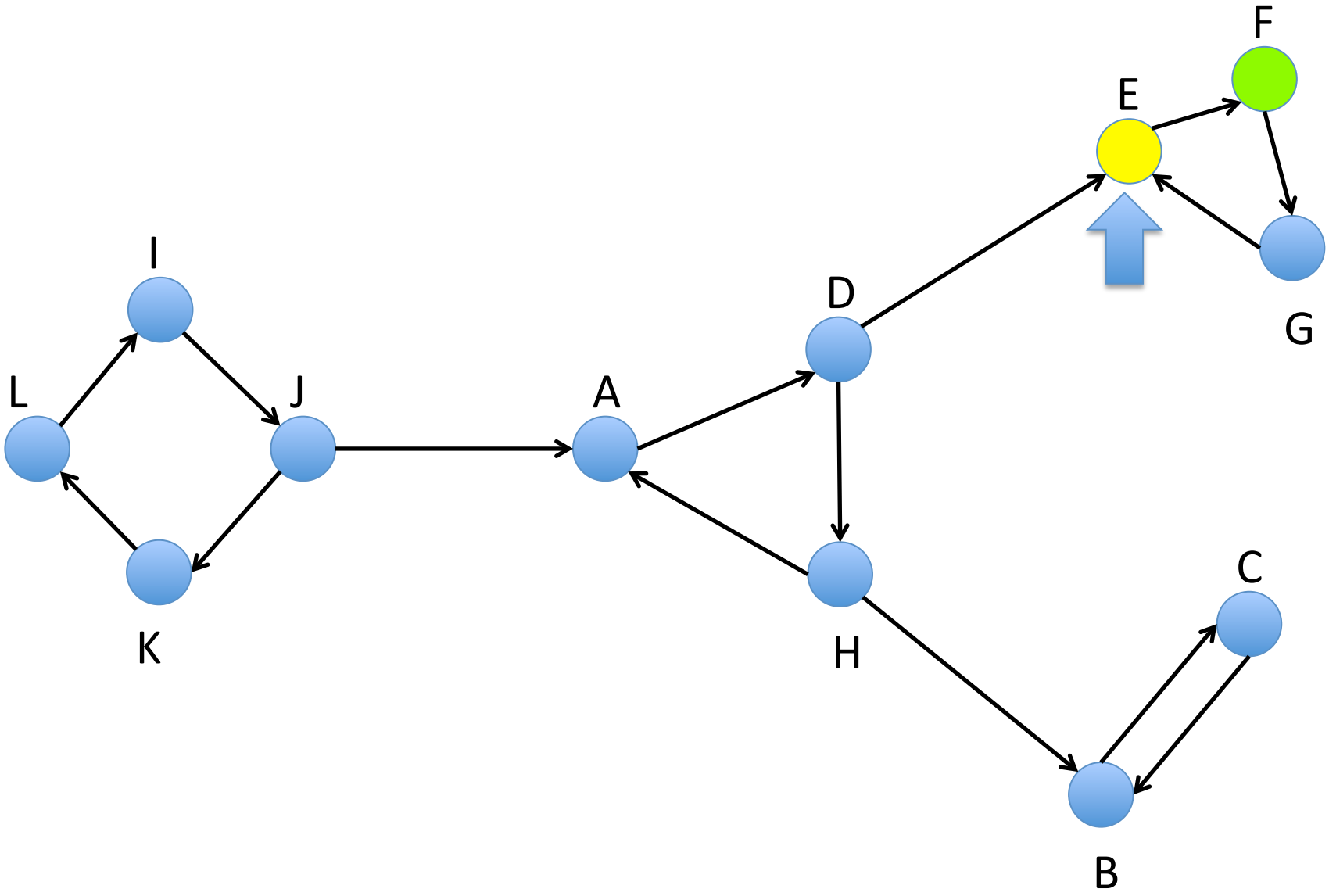
DFS & Finishing Times (2)



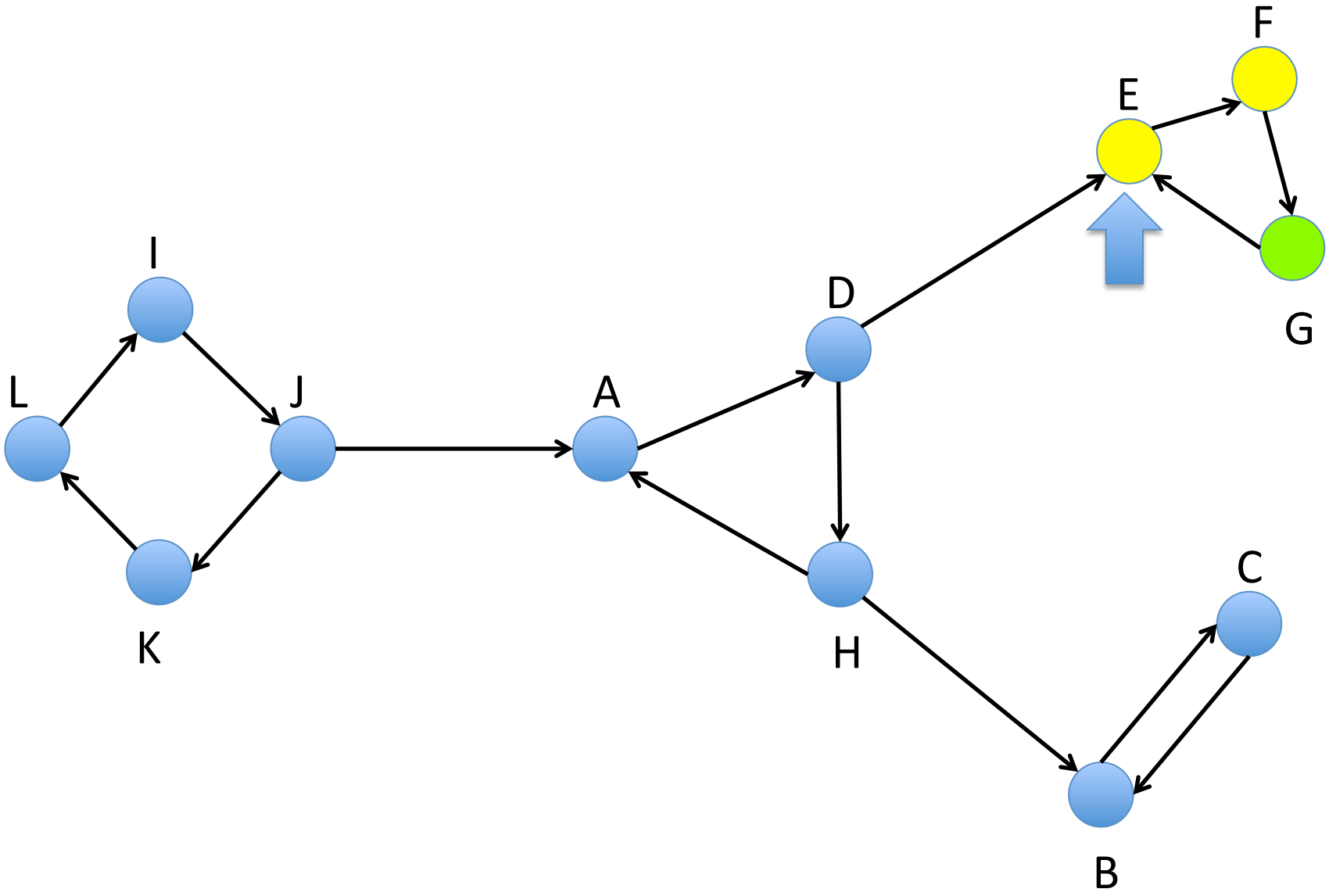
DFS & Finishing Times (2)



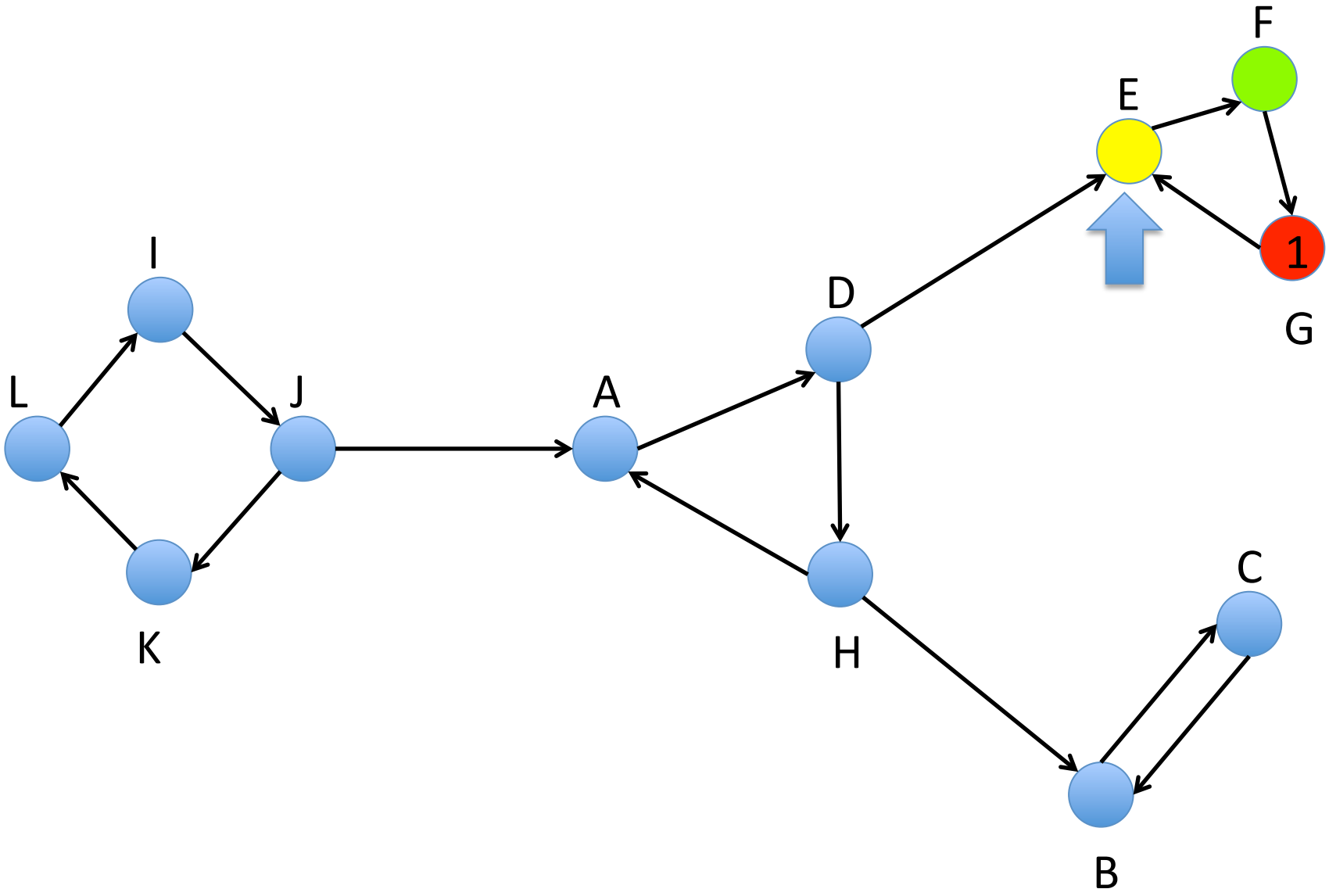
DFS & Finishing Times (2)



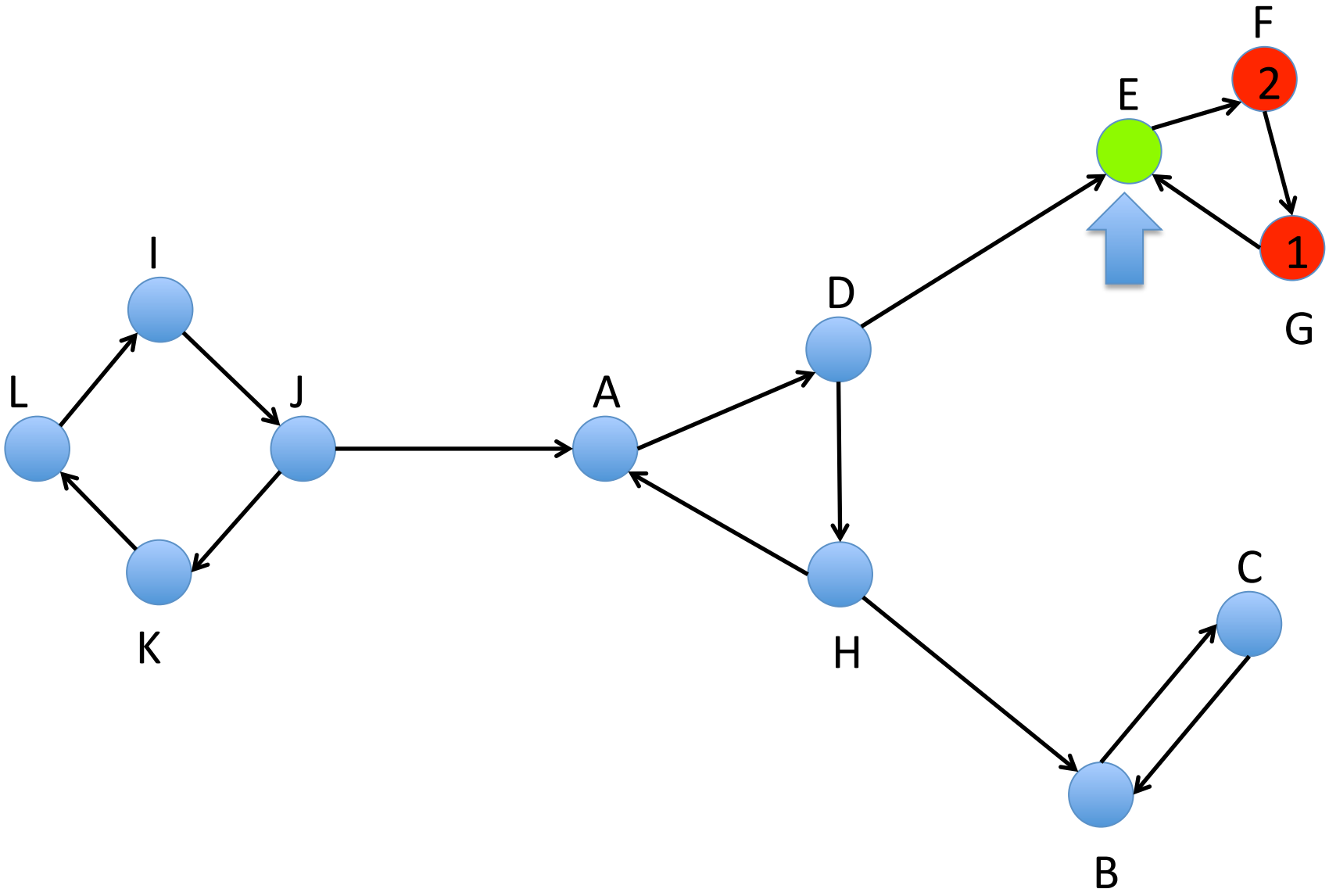
DFS & Finishing Times (2)



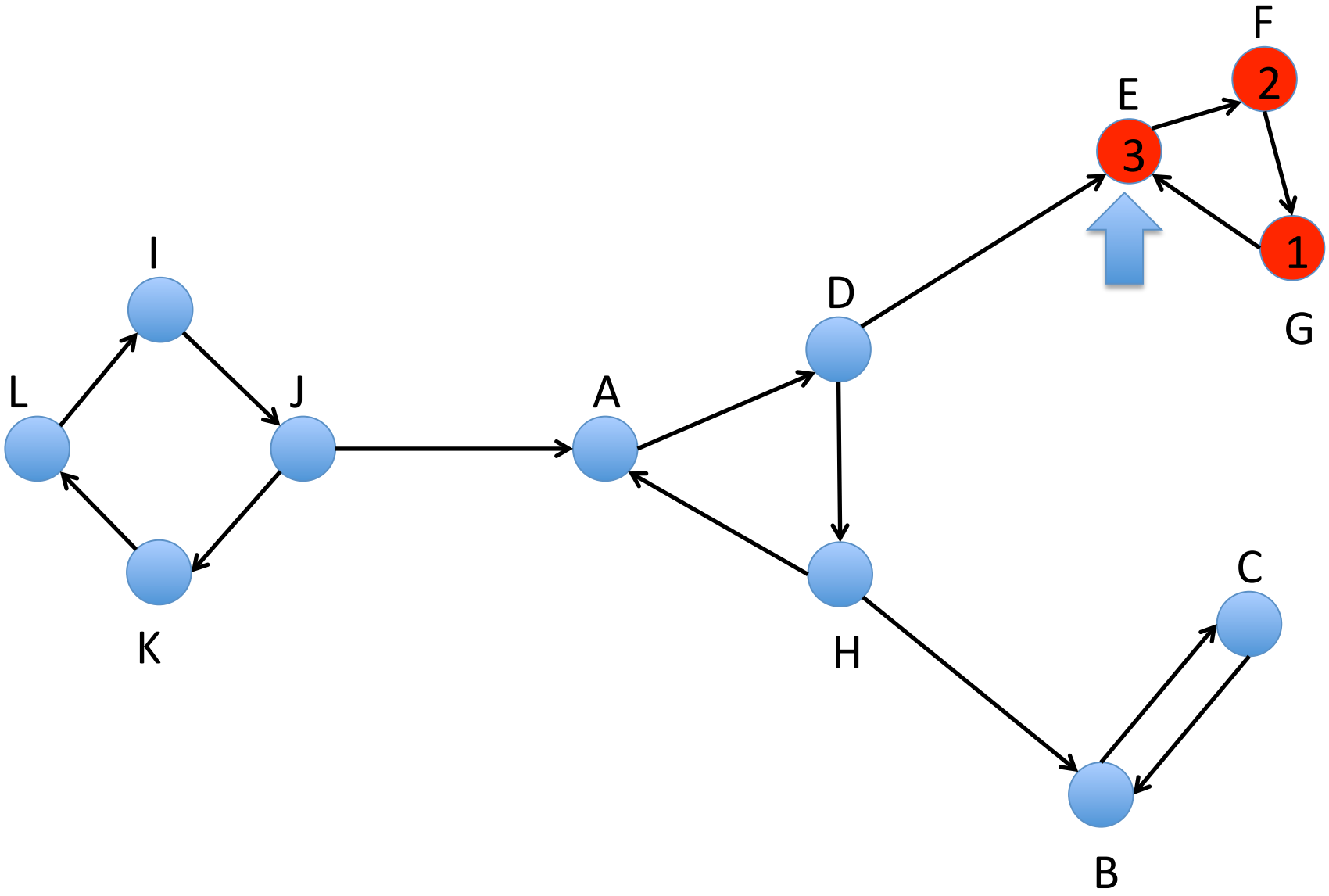
DFS & Finishing Times (2)



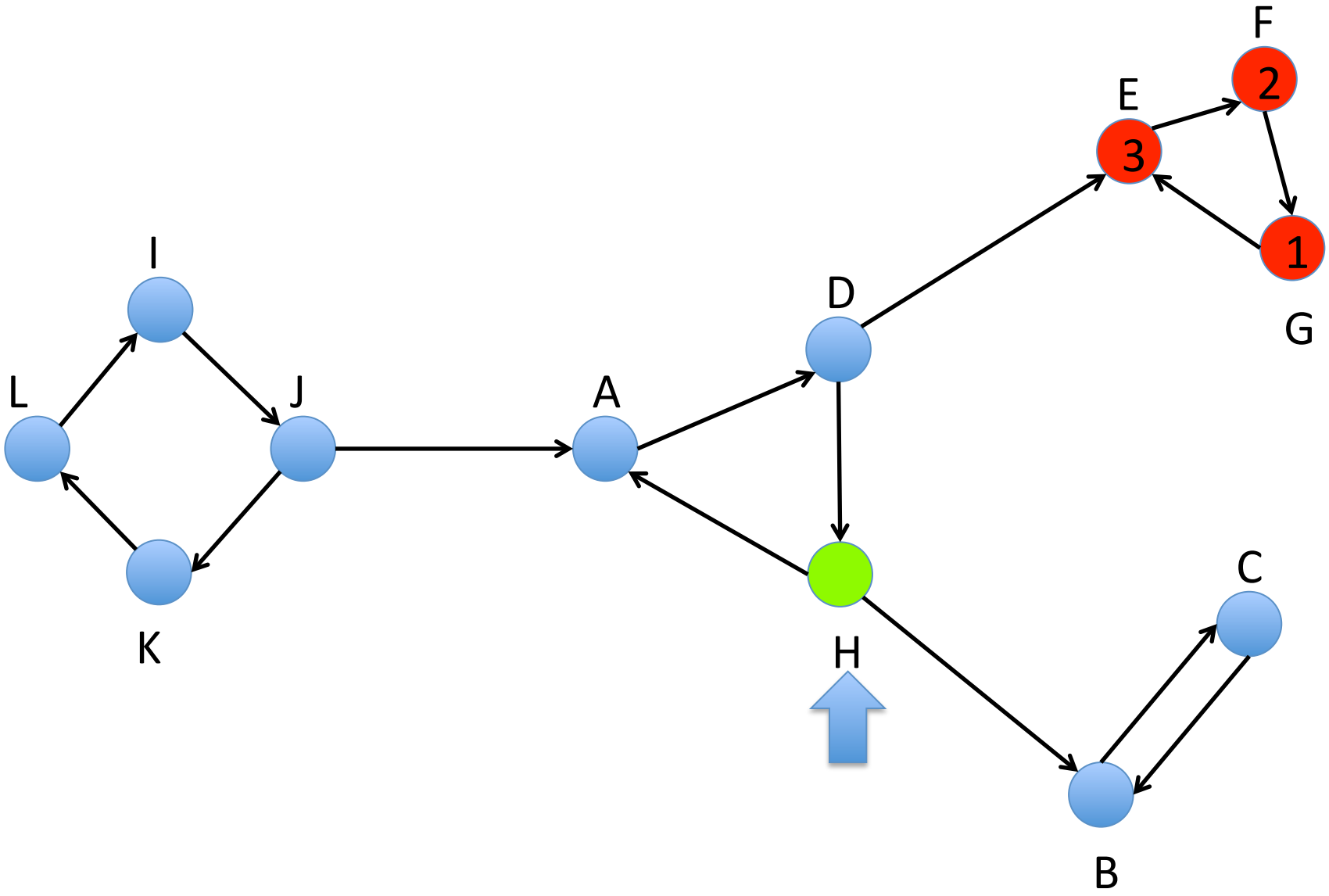
DFS & Finishing Times (2)



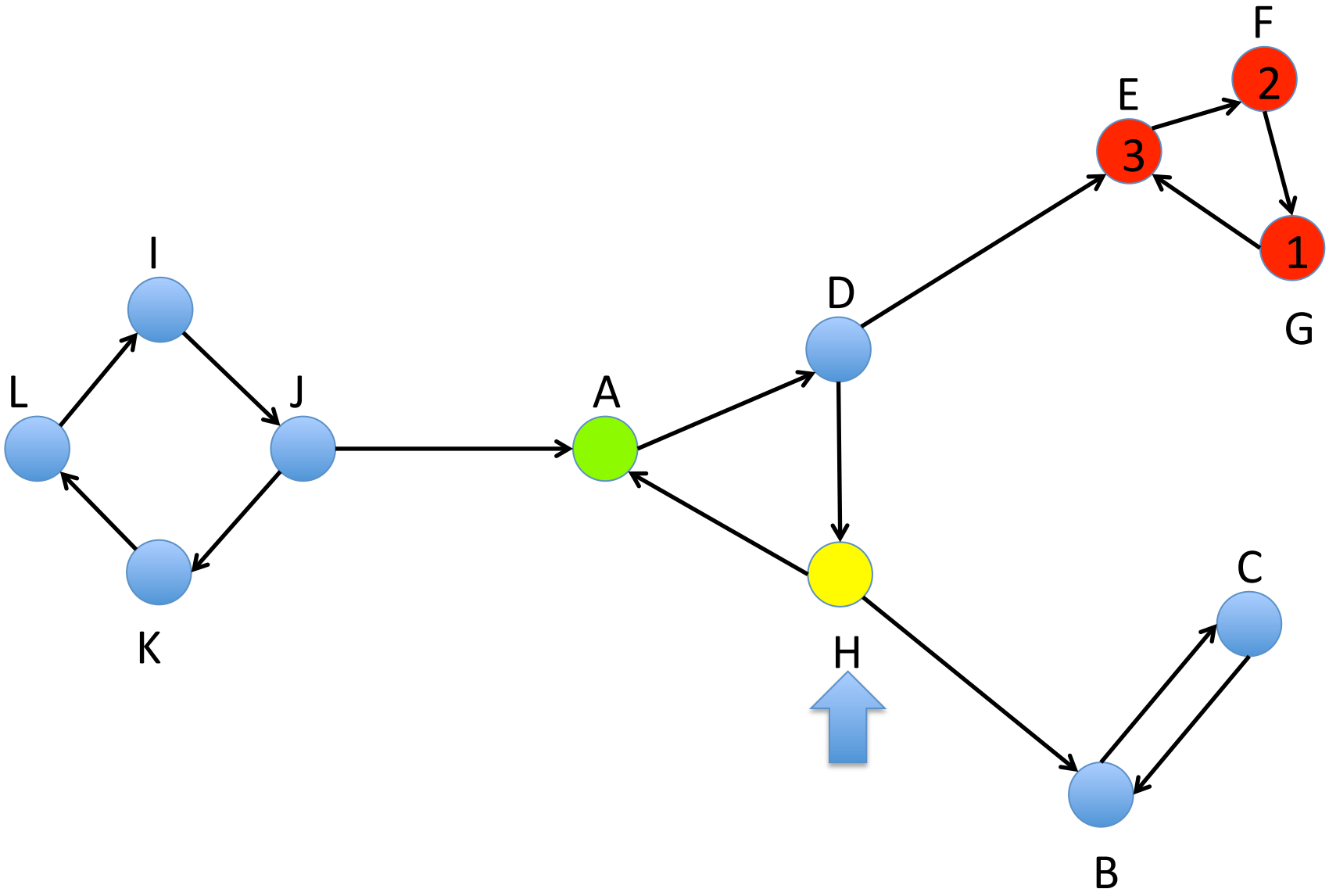
DFS & Finishing Times (2)



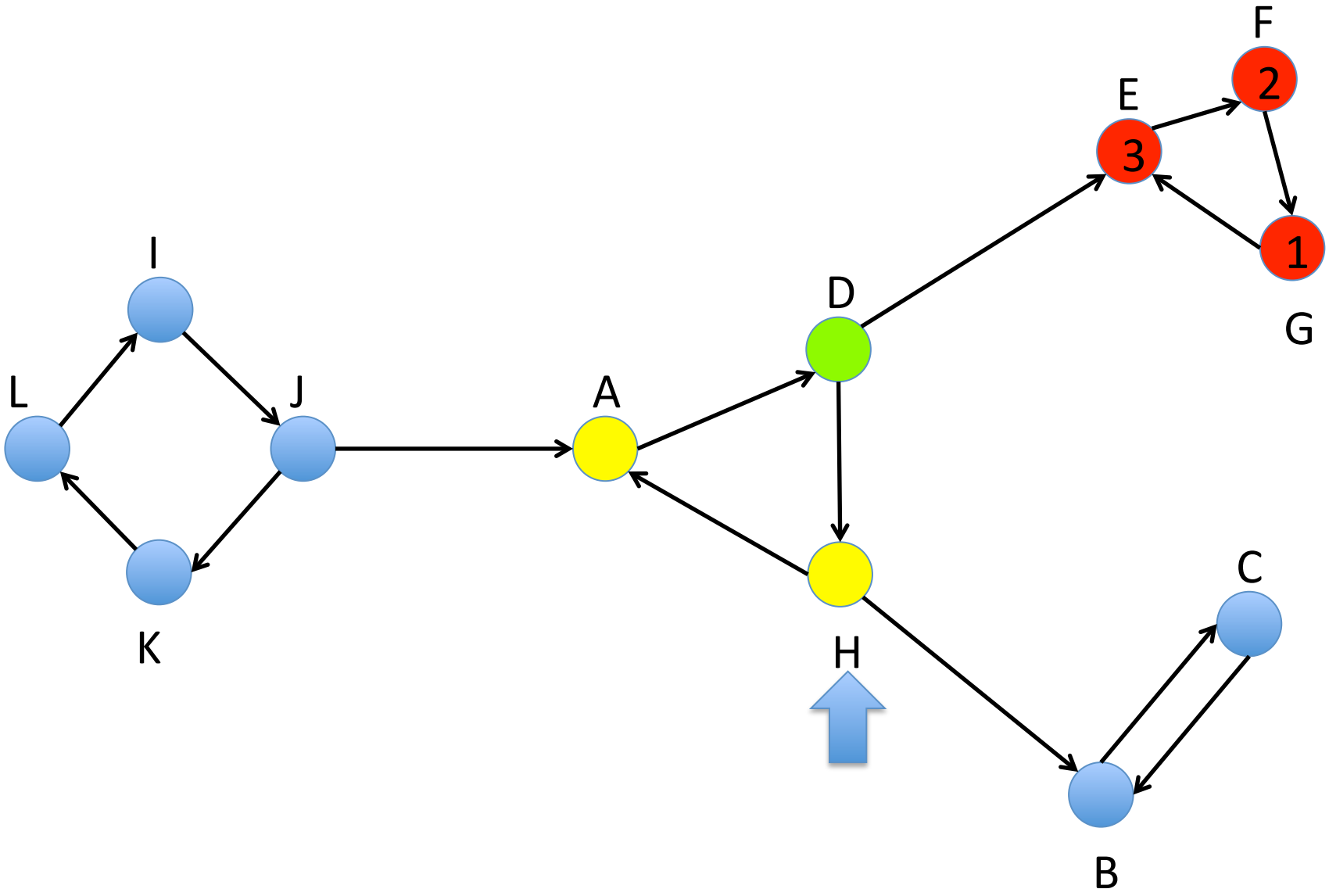
DFS & Finishing Times (2)



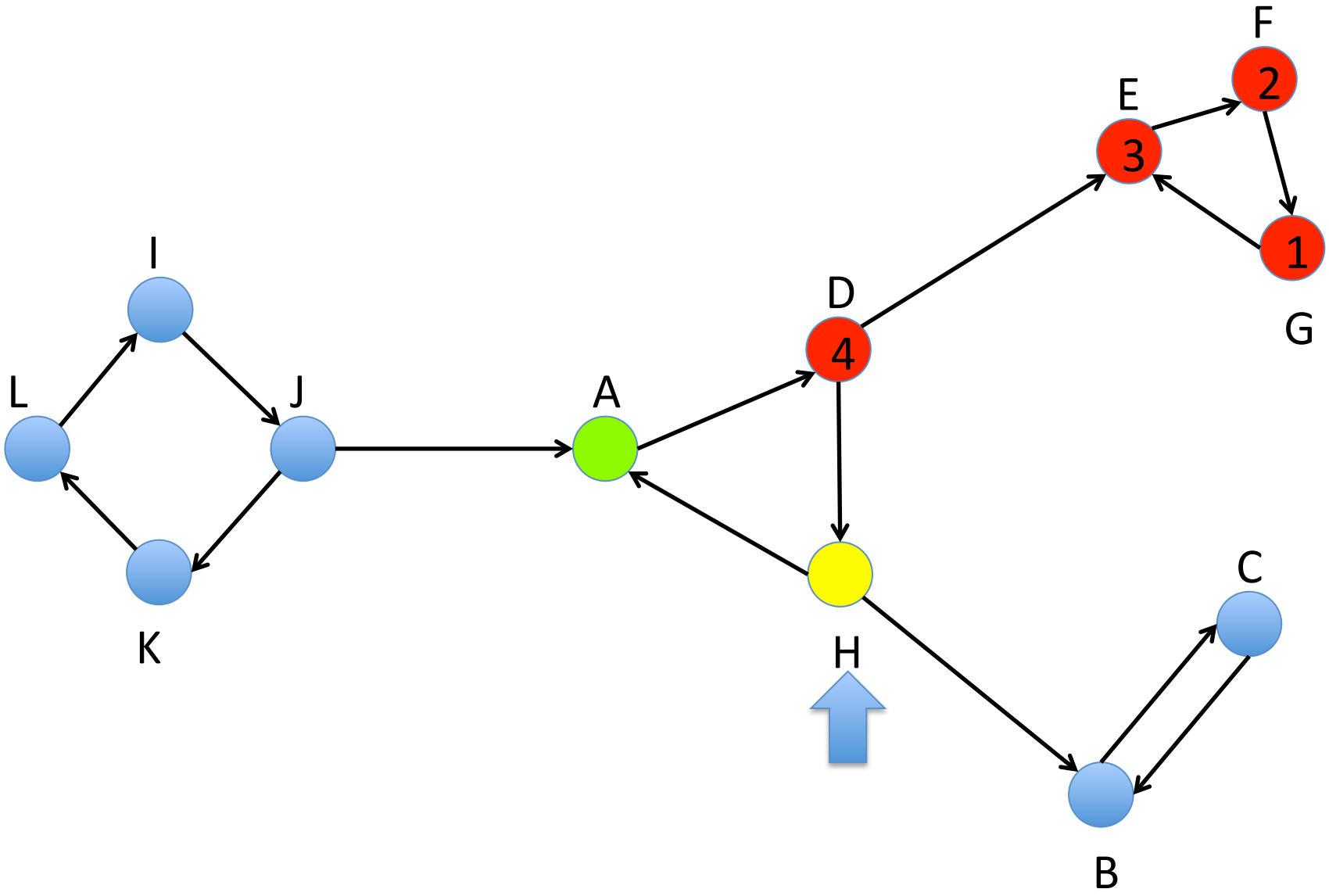
DFS & Finishing Times (2)



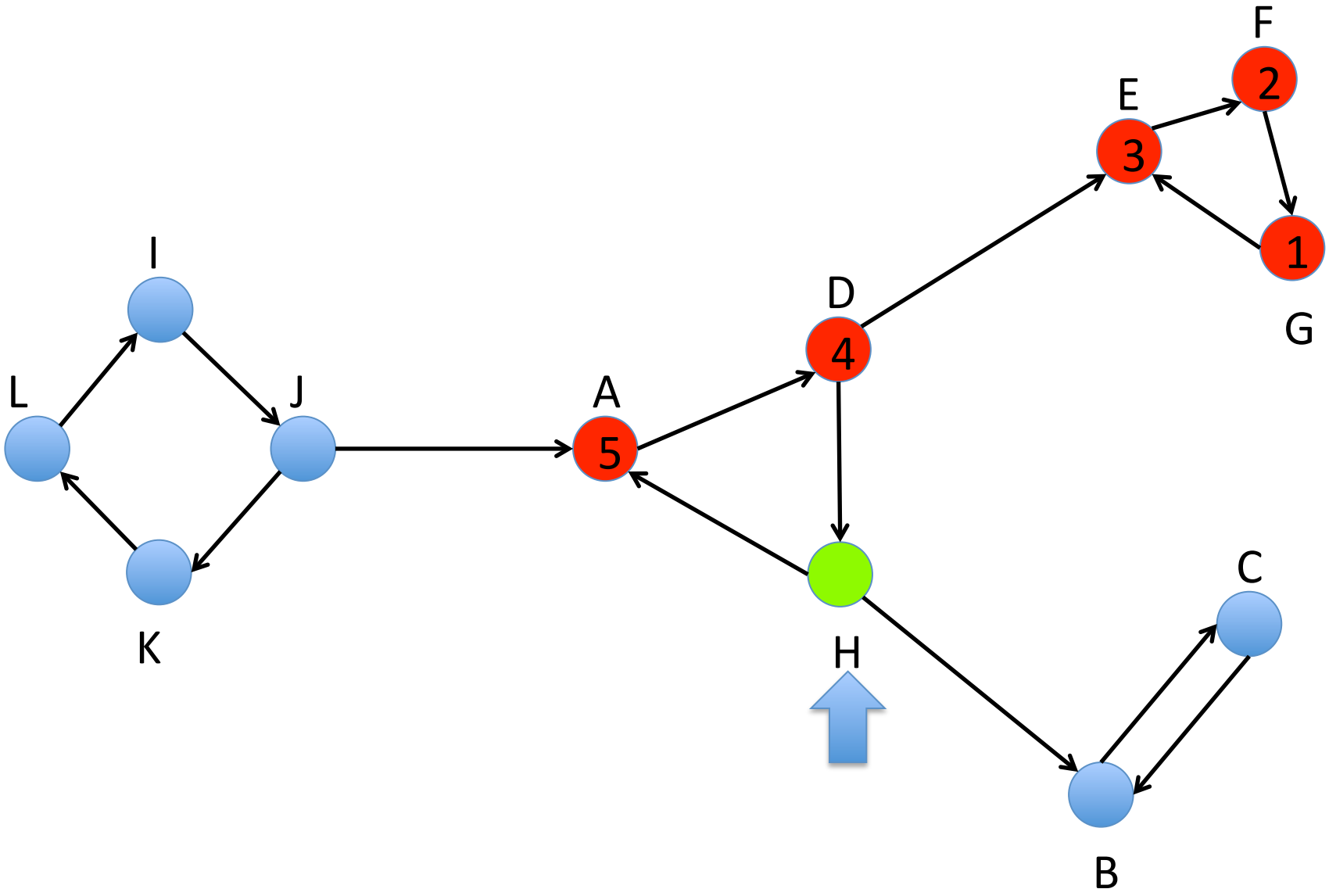
DFS & Finishing Times (2)



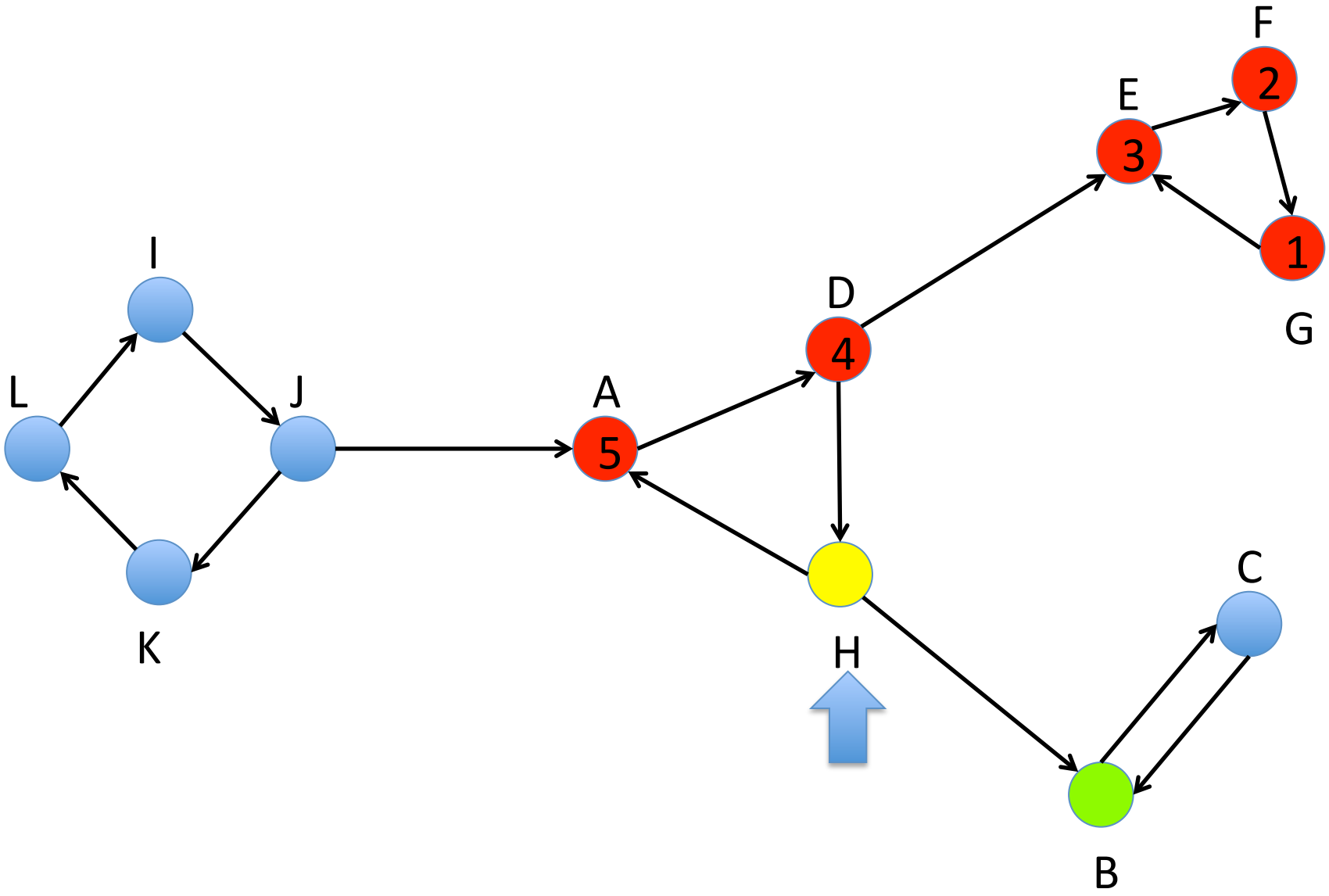
DFS & Finishing Times (2)



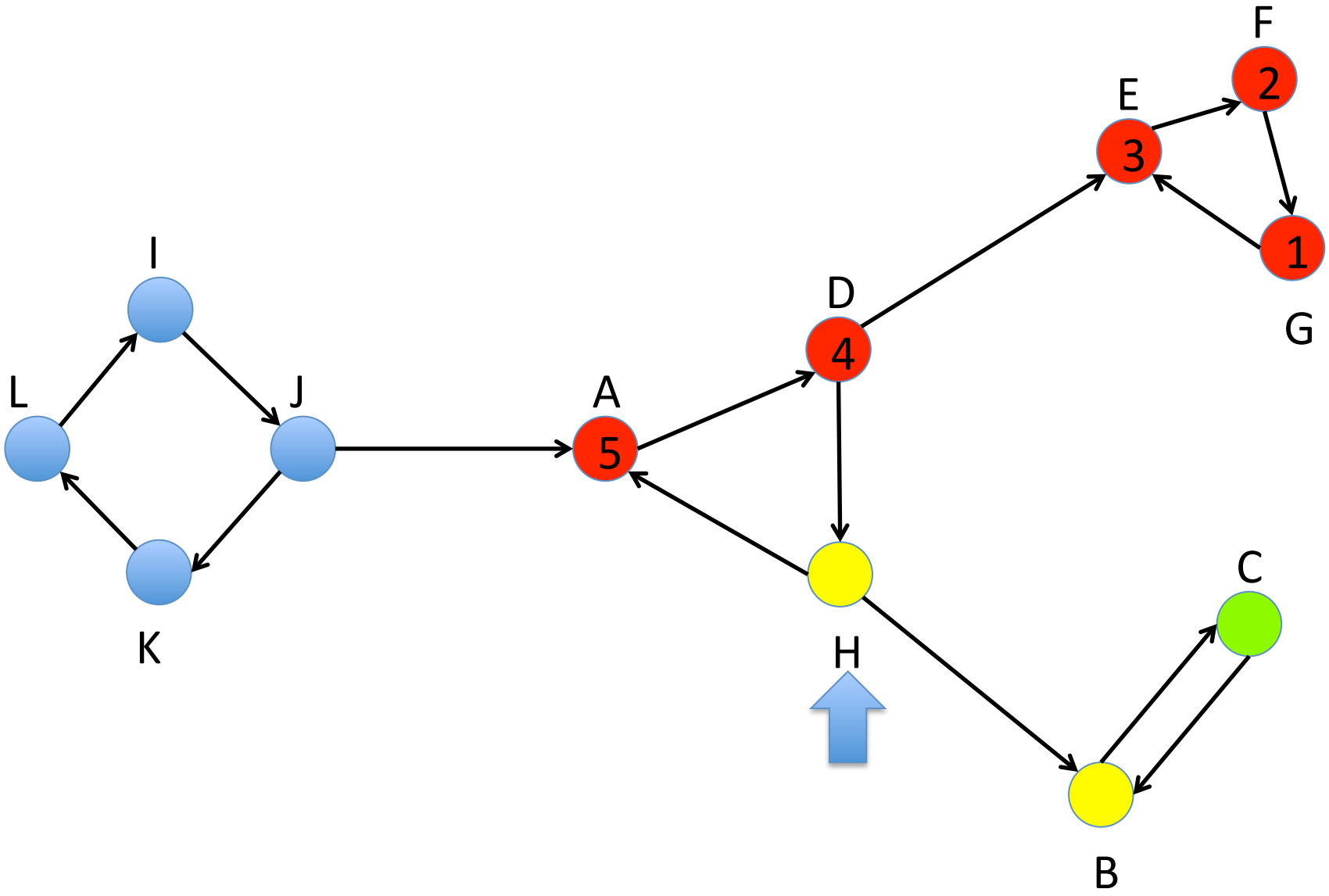
DFS & Finishing Times (2)



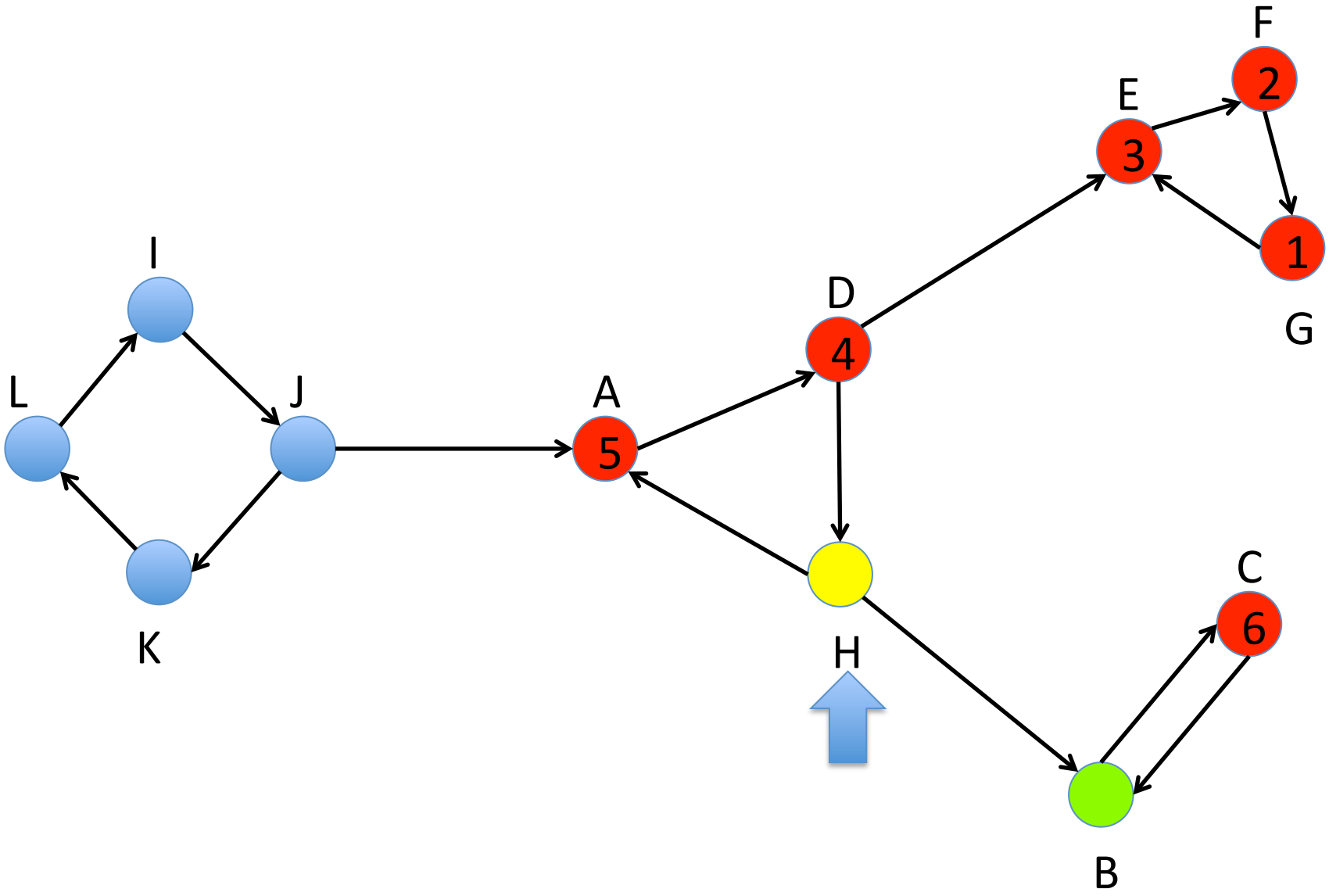
DFS & Finishing Times (2)



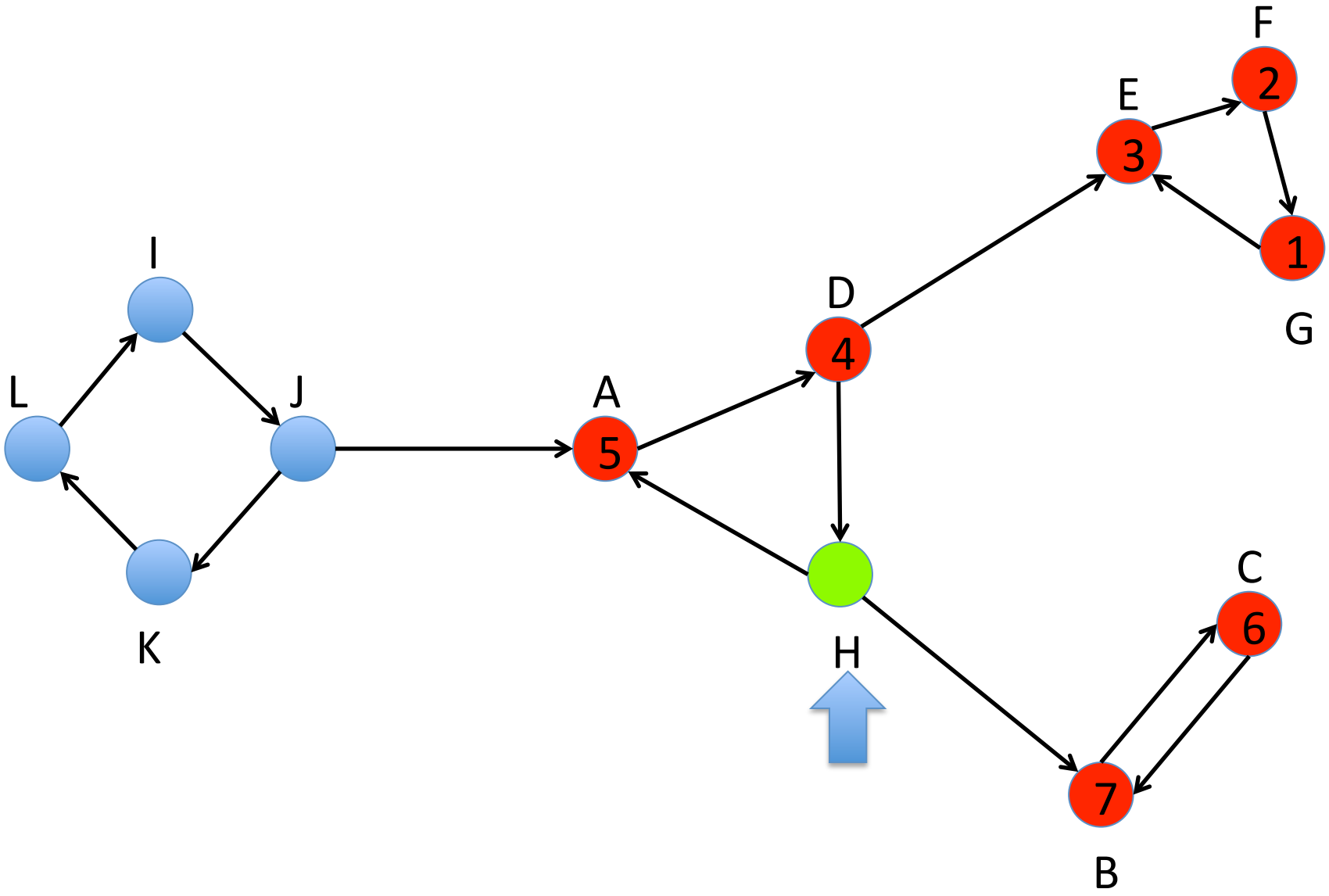
DFS & Finishing Times (2)



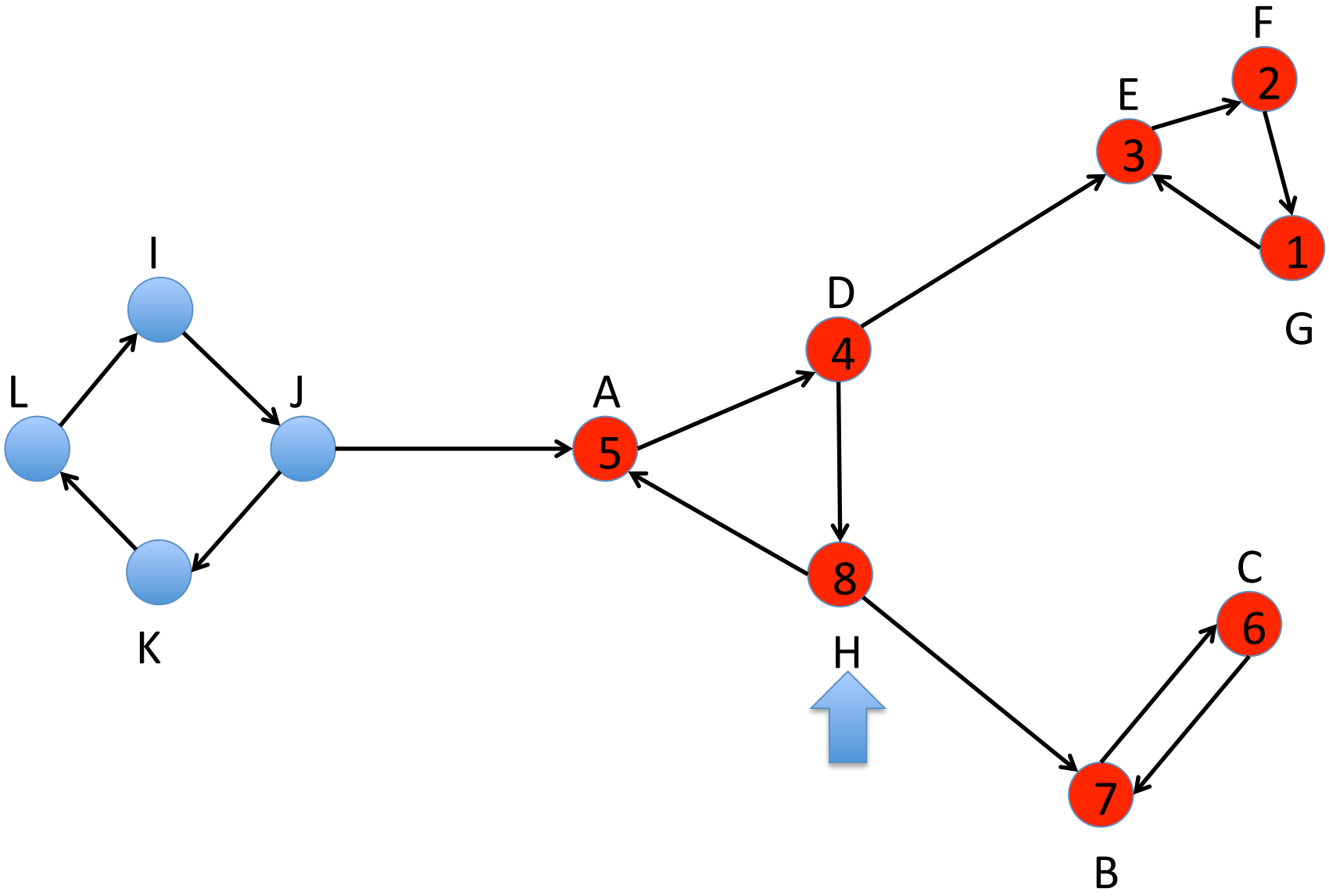
DFS & Finishing Times (2)



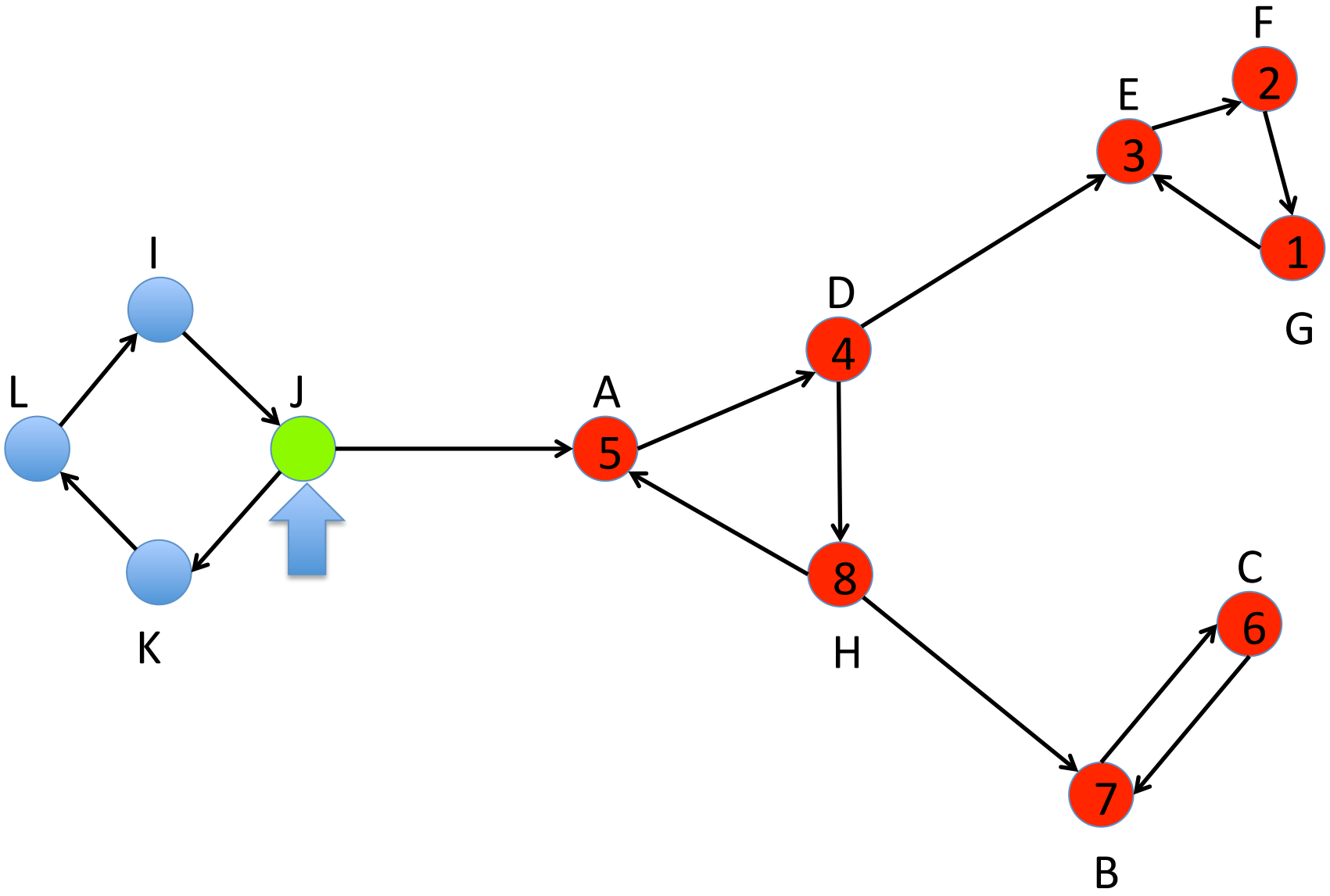
DFS & Finishing Times (2)



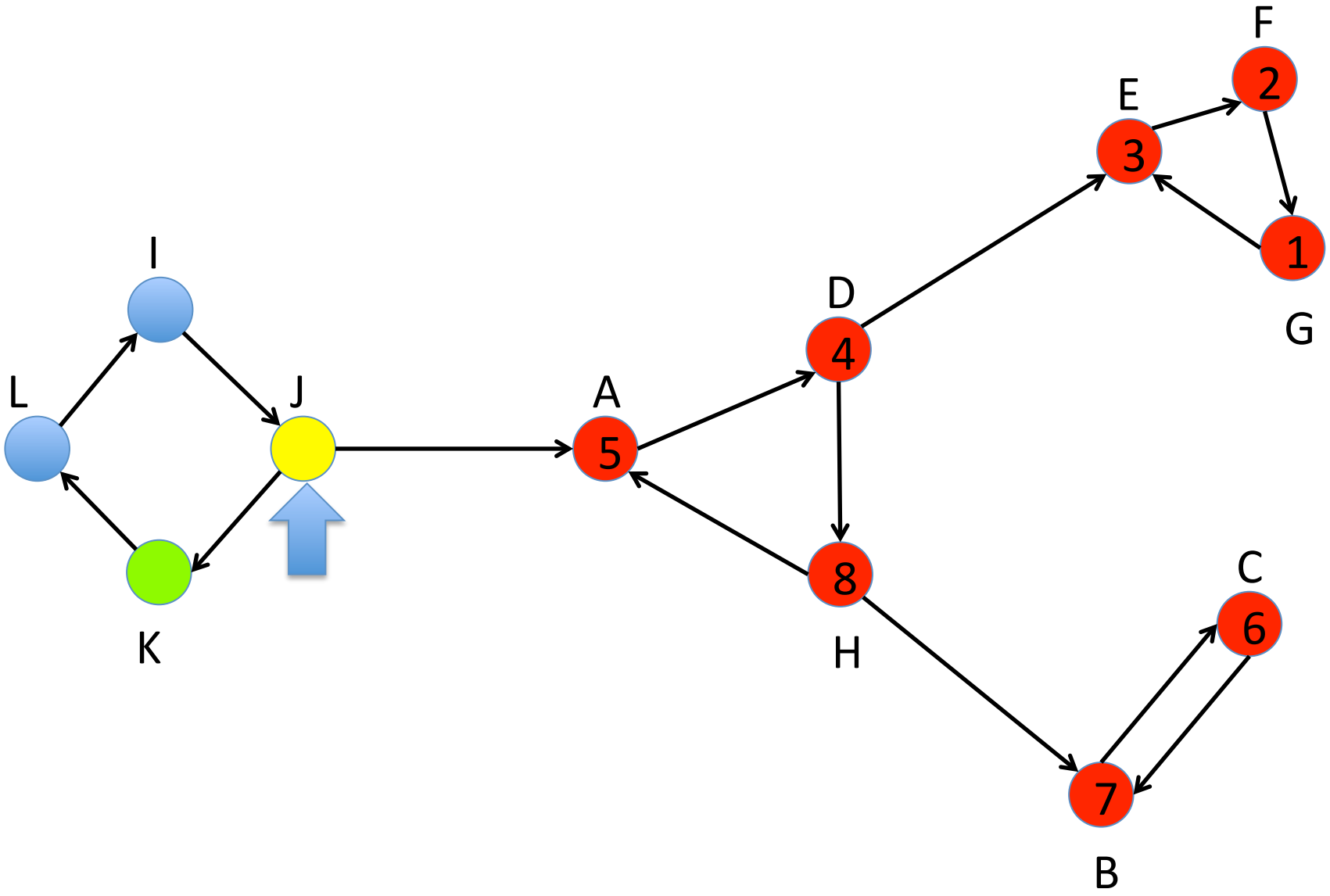
DFS & Finishing Times (2)



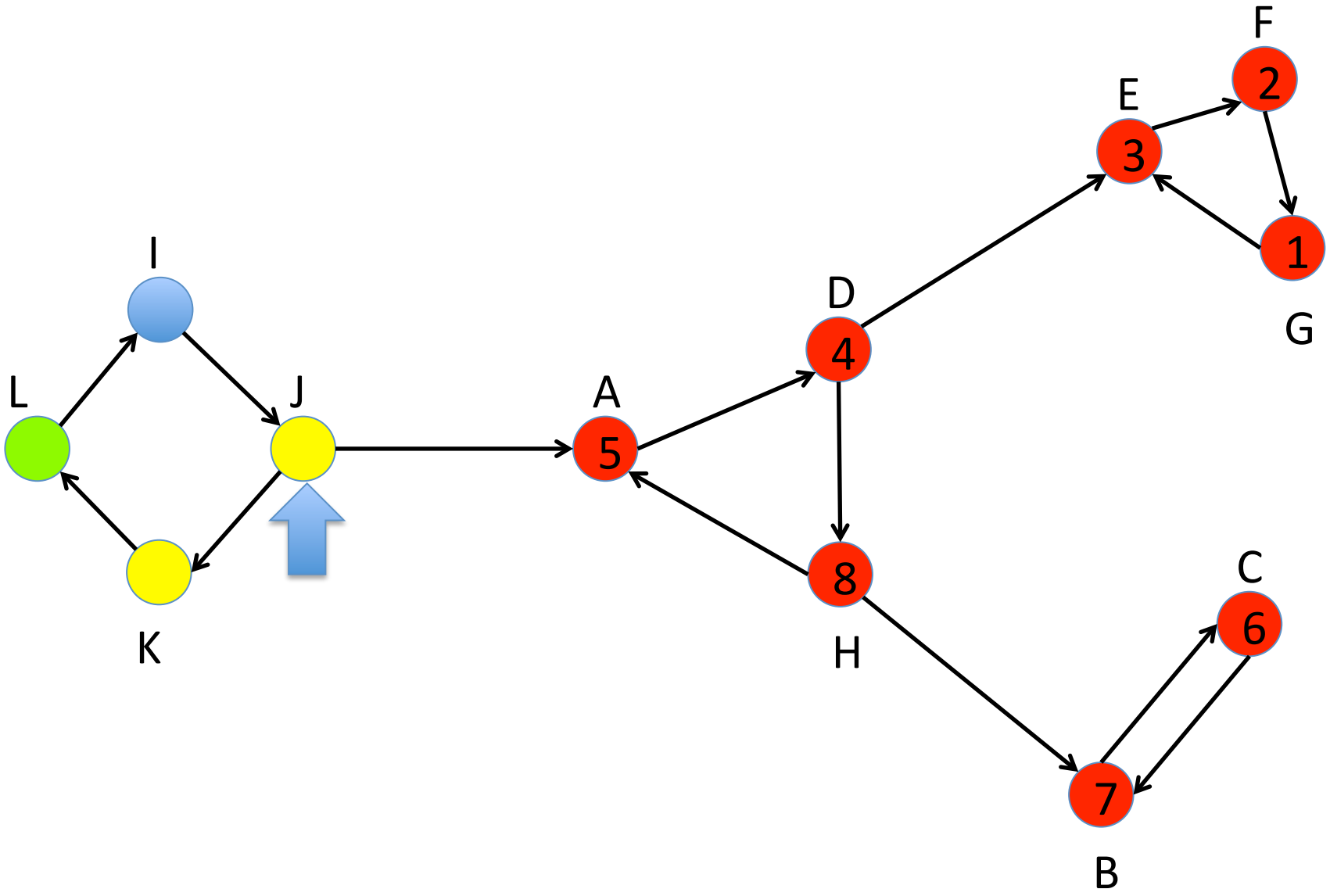
DFS & Finishing Times (2)



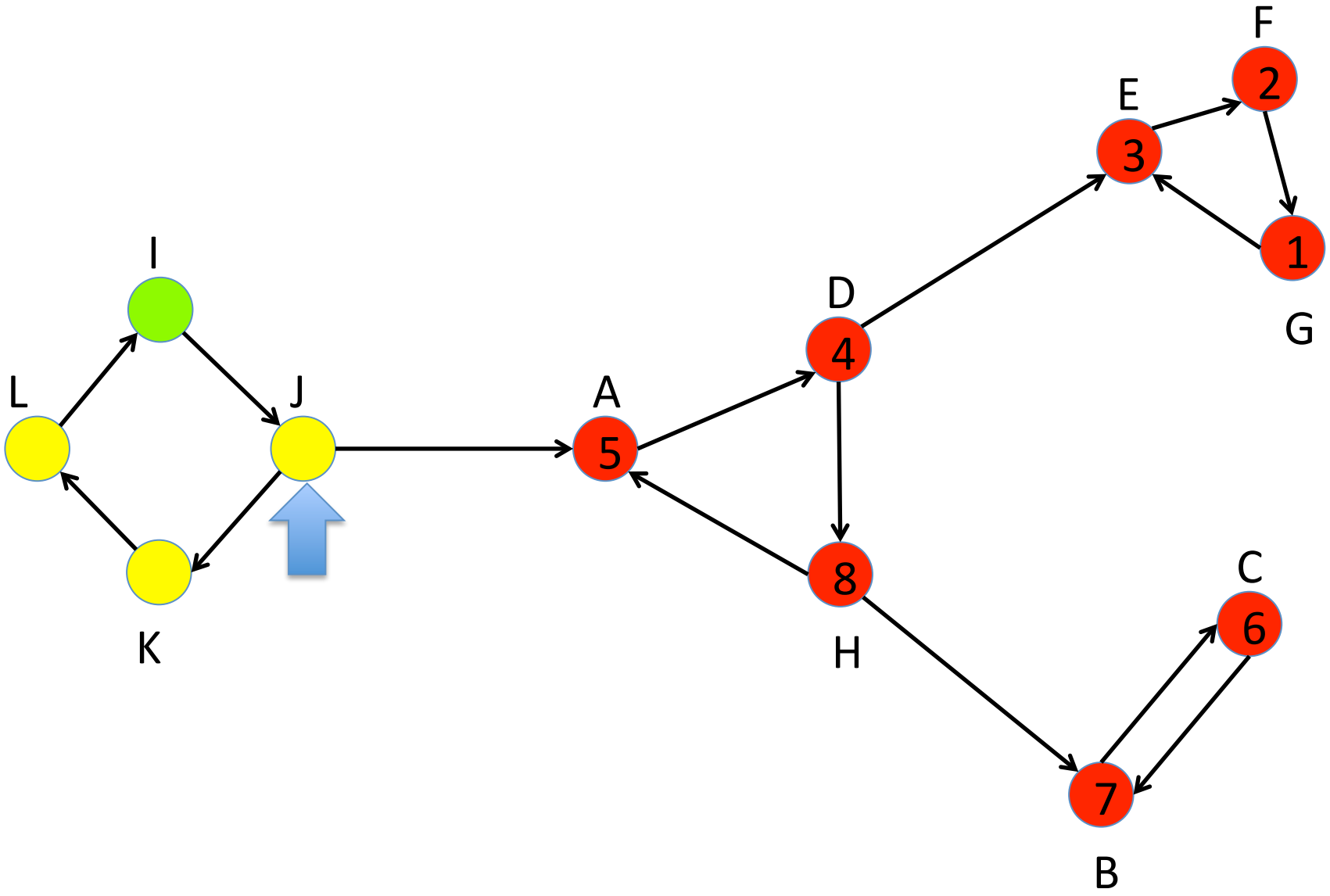
DFS & Finishing Times (2)



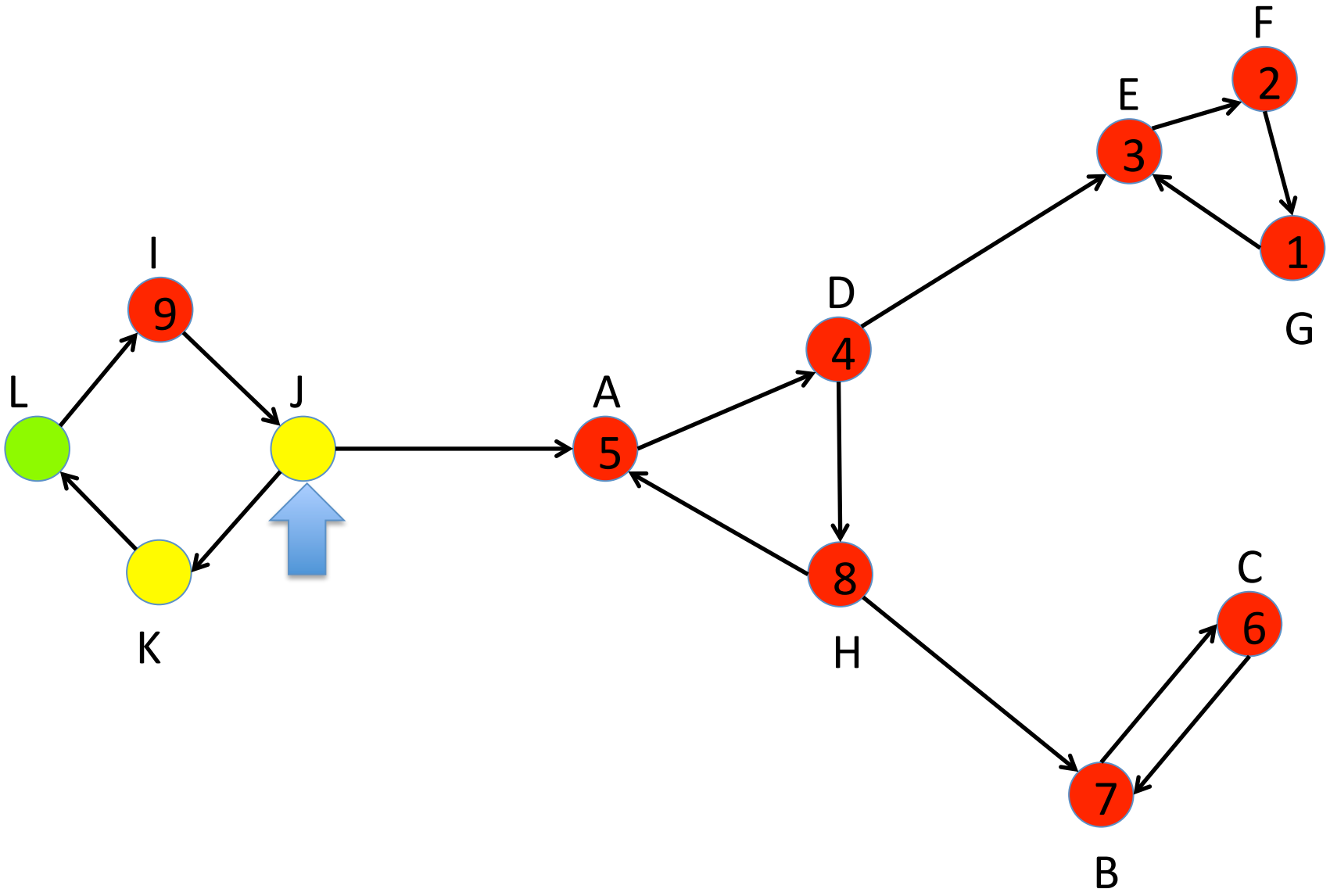
DFS & Finishing Times (2)



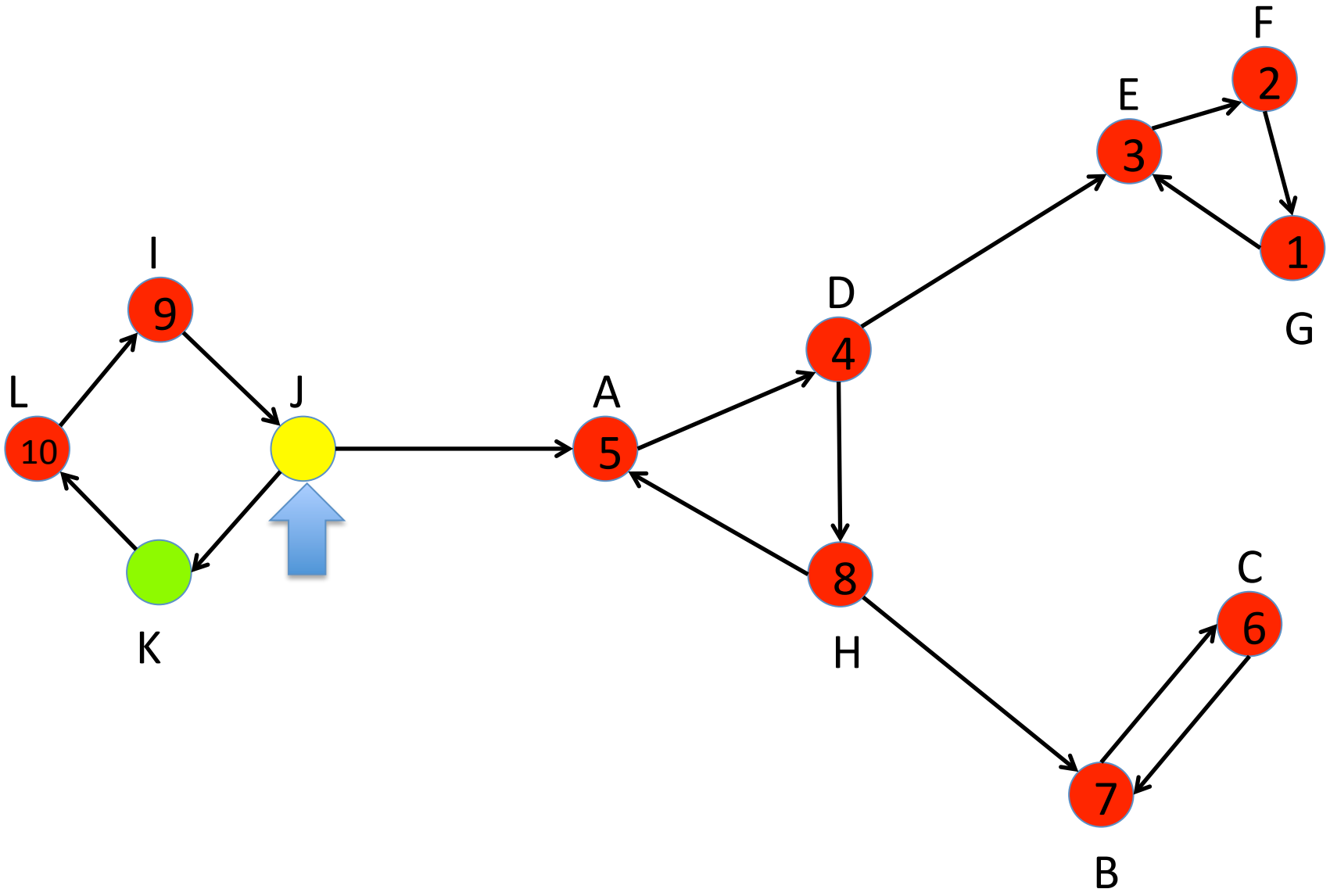
DFS & Finishing Times (2)



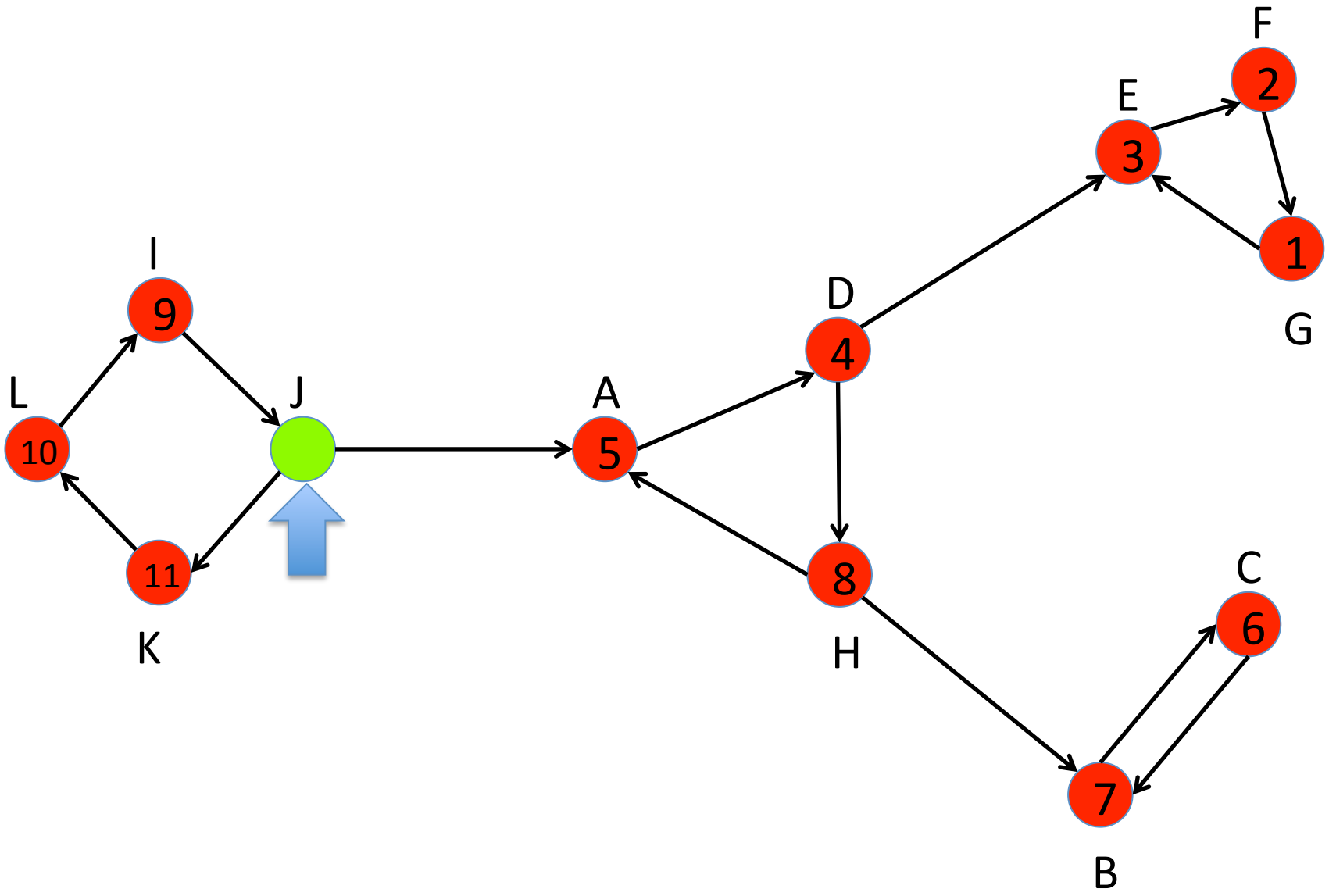
DFS & Finishing Times (2)



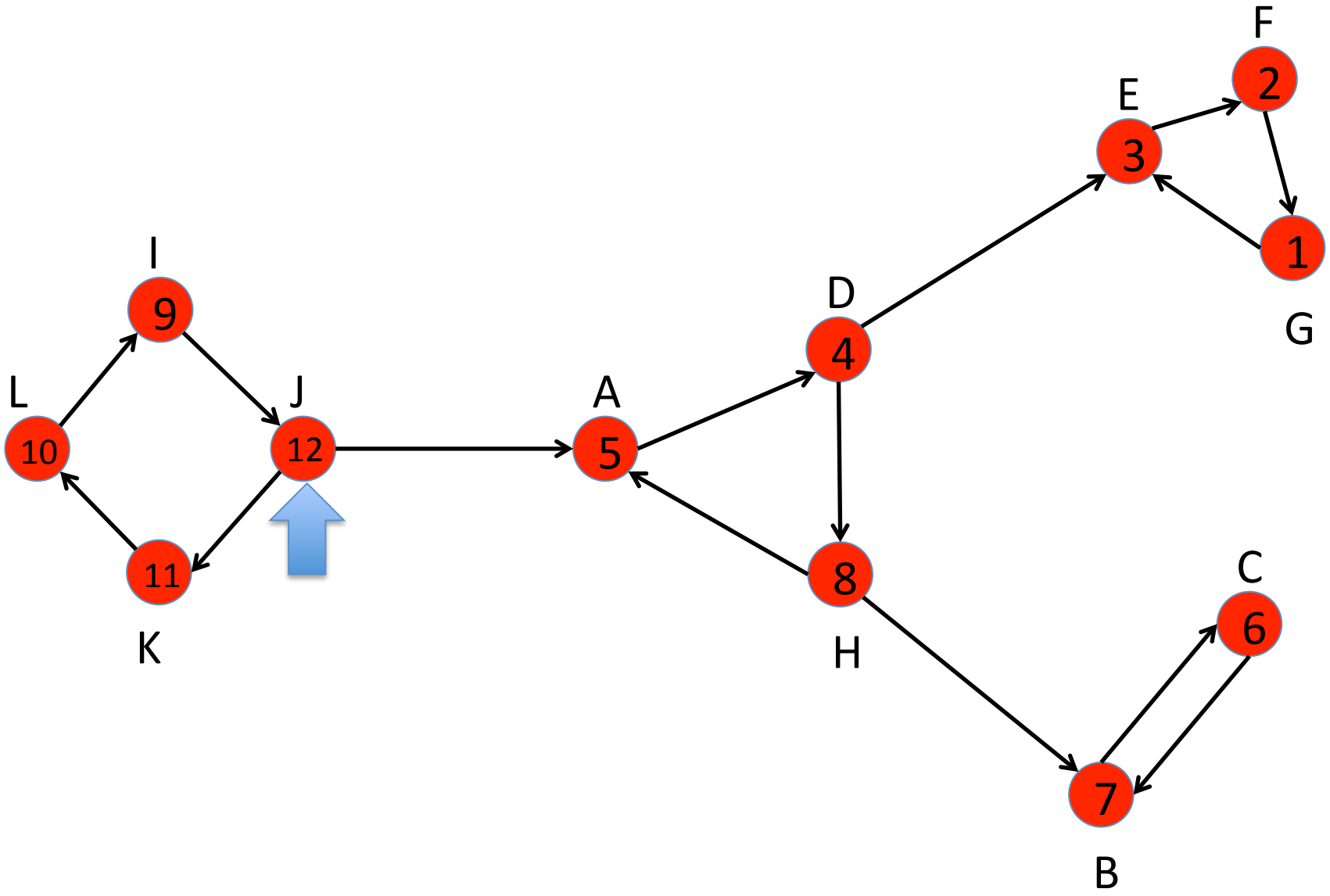
DFS & Finishing Times (2)



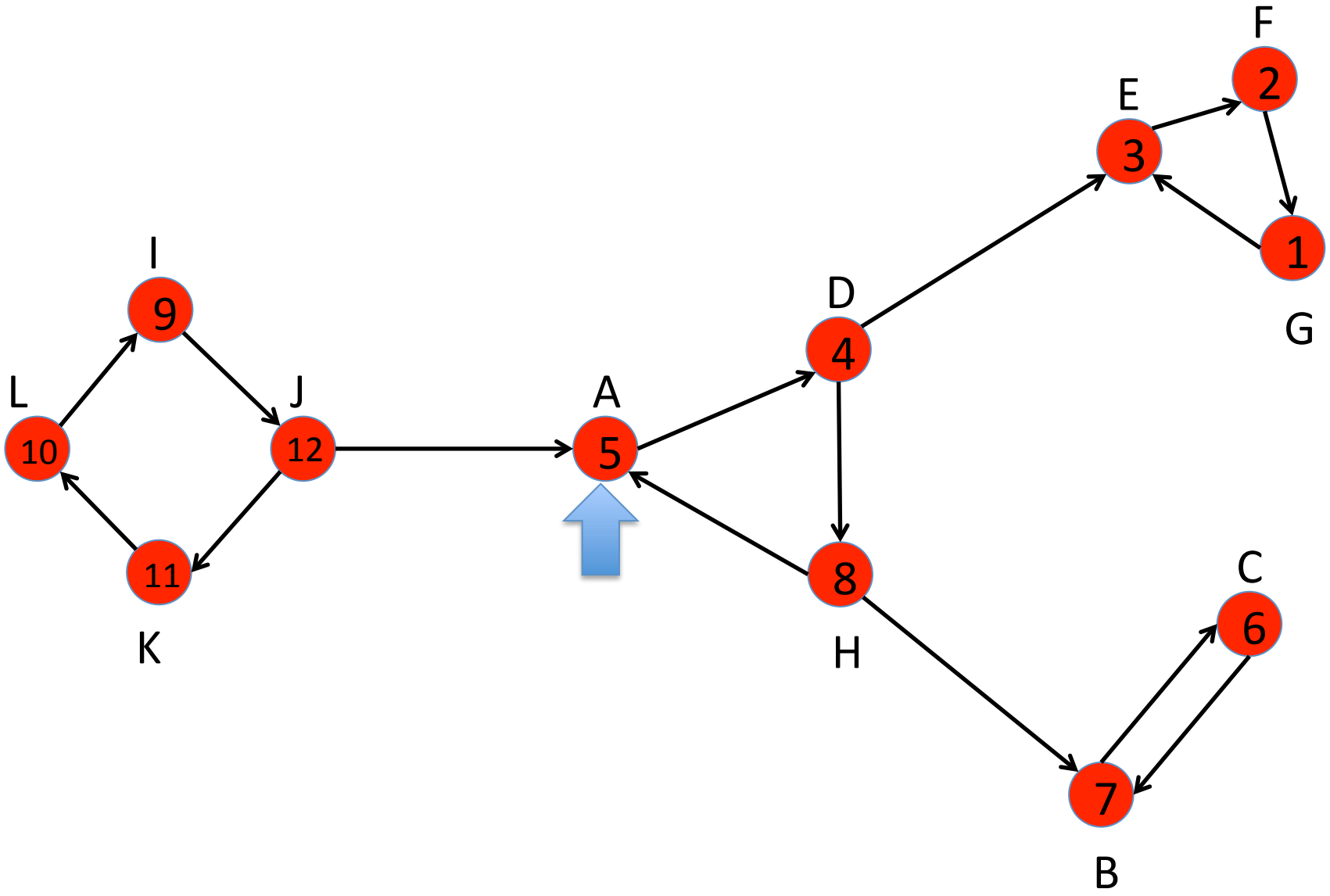
DFS & Finishing Times (2)



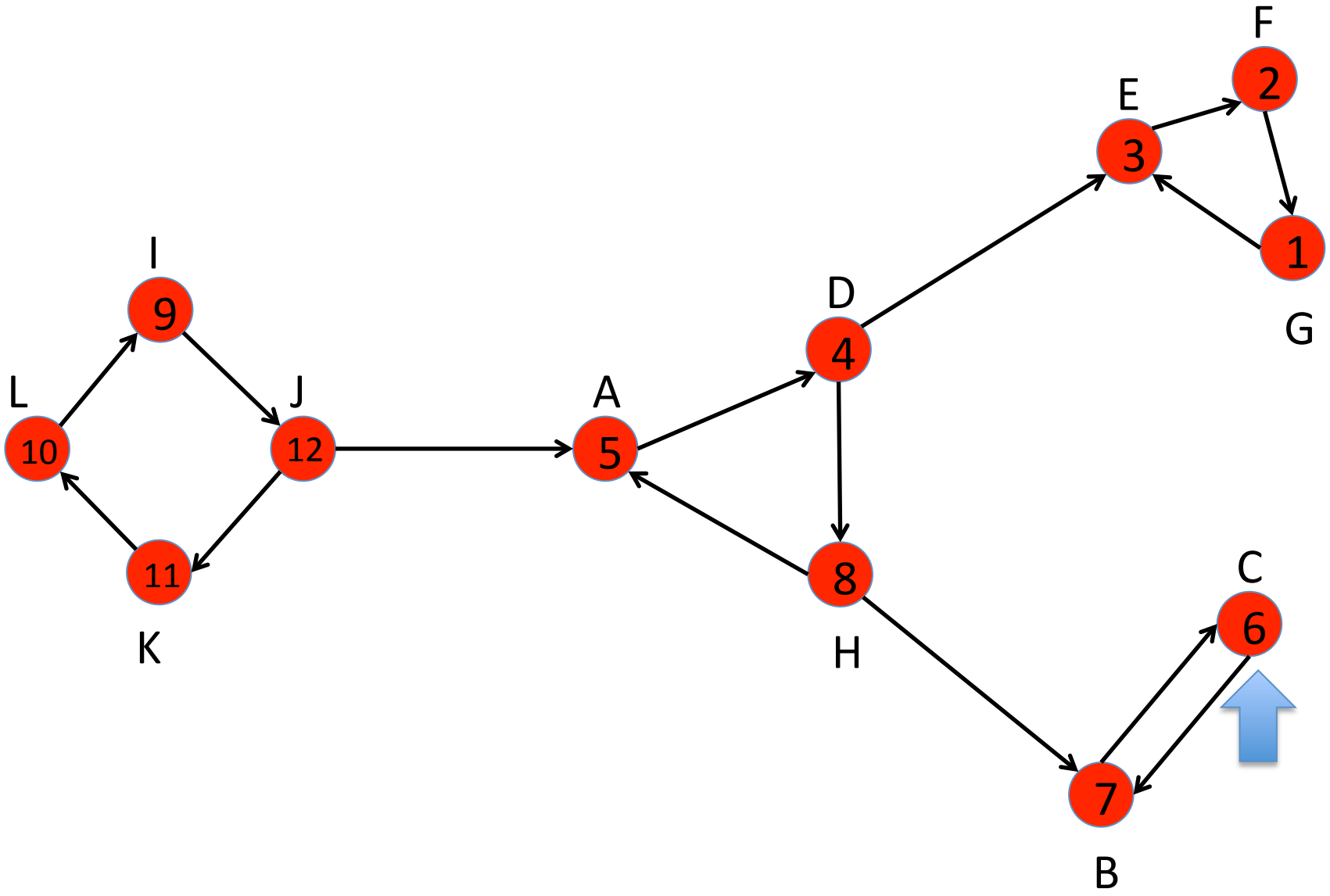
DFS & Finishing Times (2)



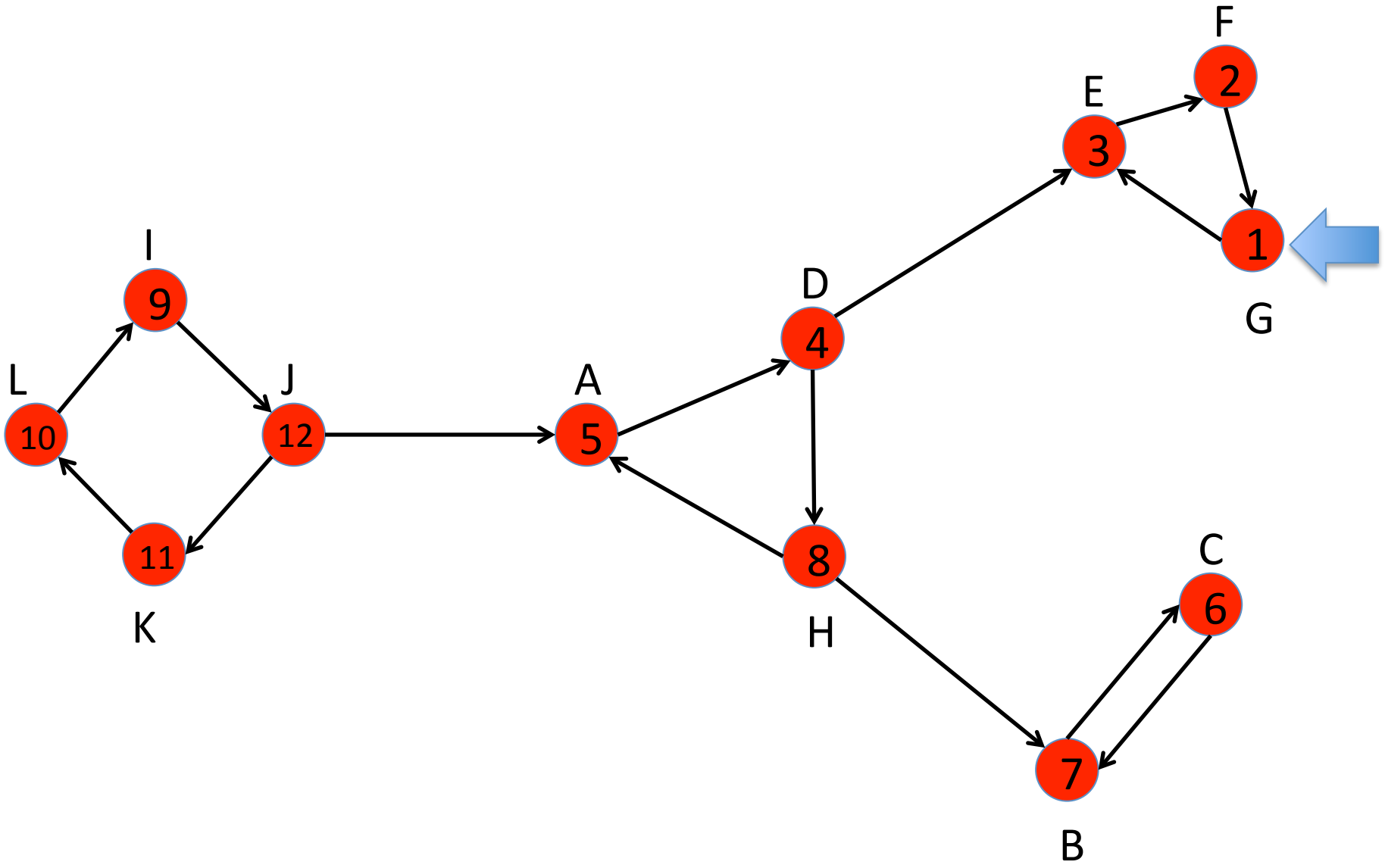
DFS & Finishing Times (2)



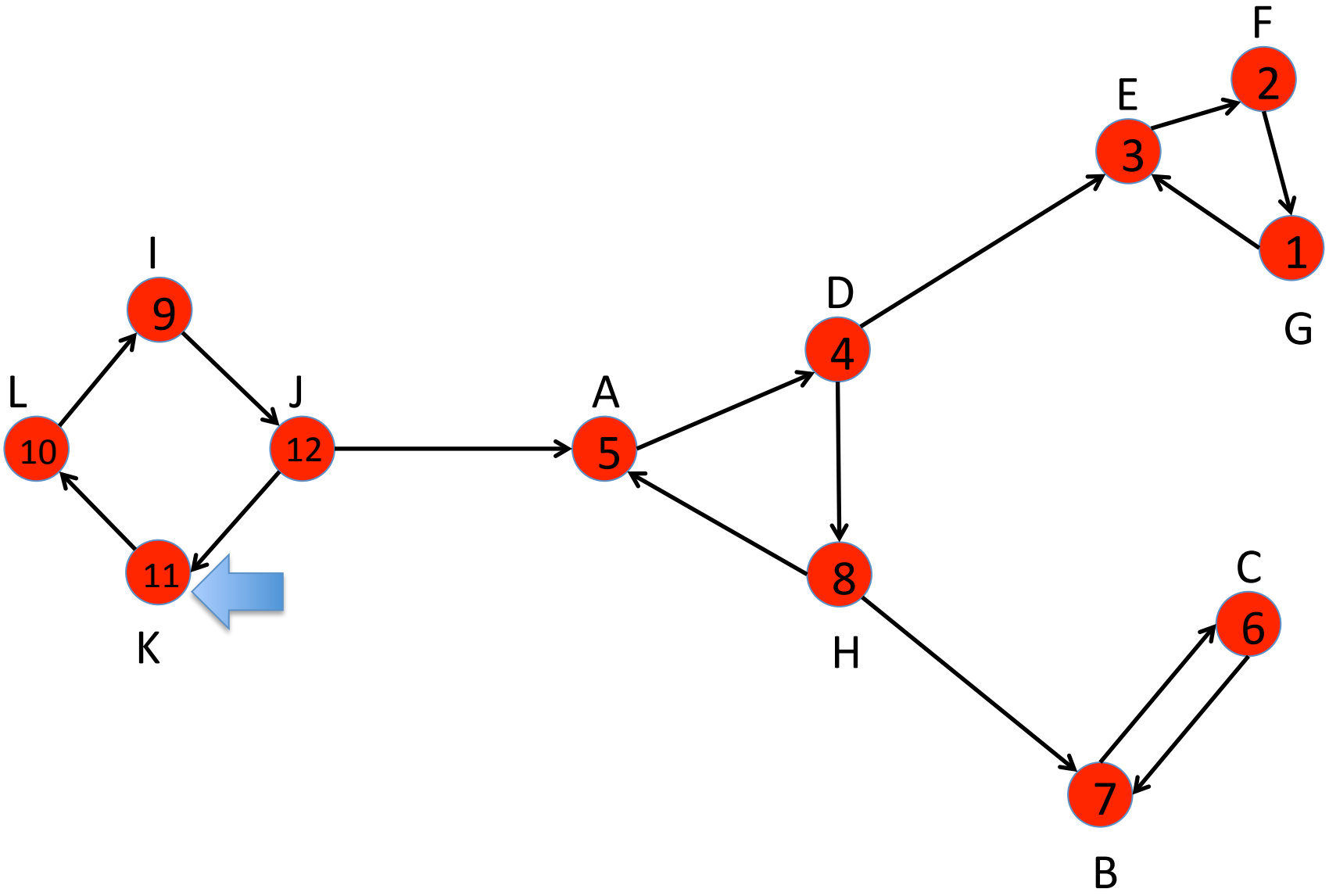
DFS & Finishing Times (2)



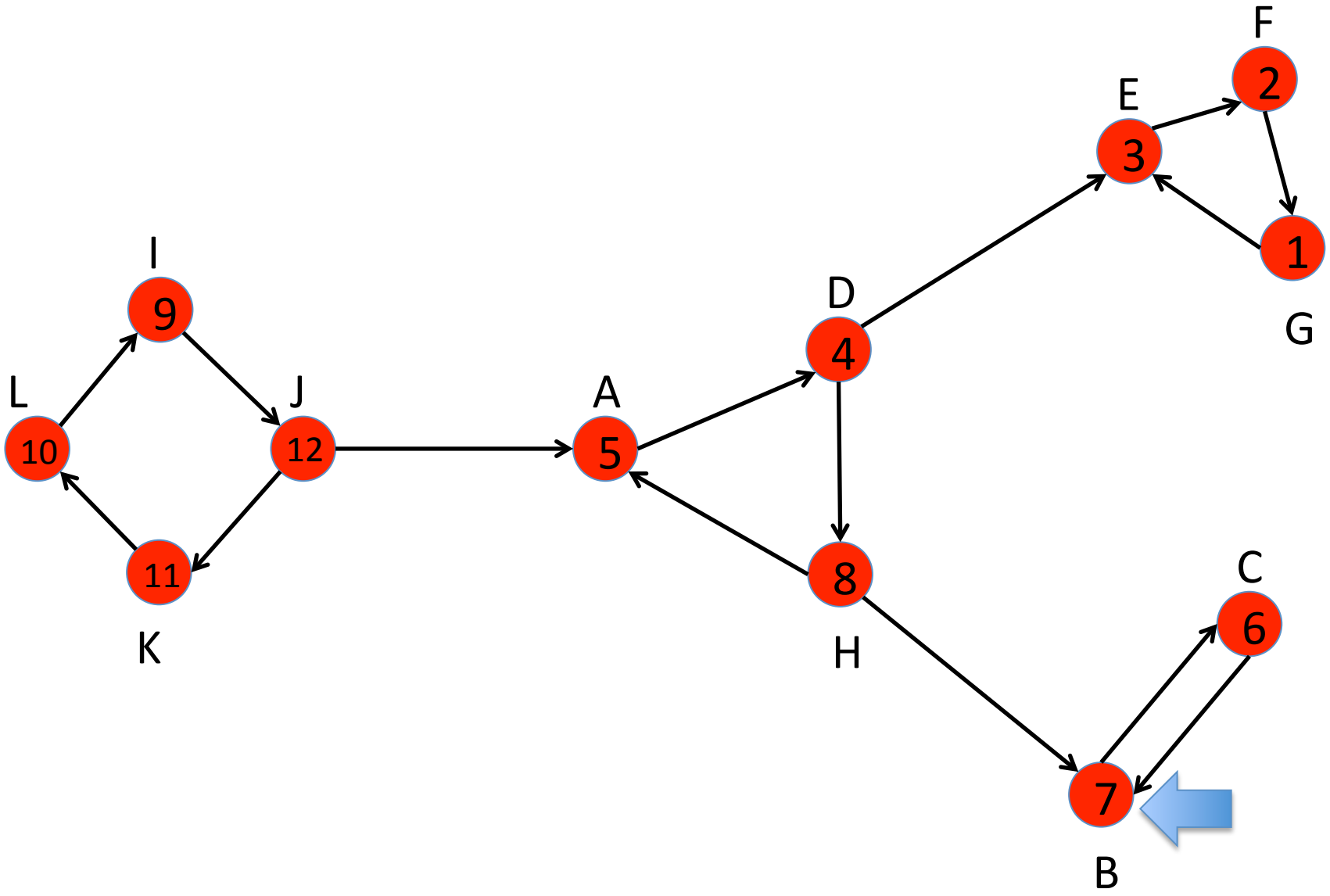
DFS & Finishing Times (2)



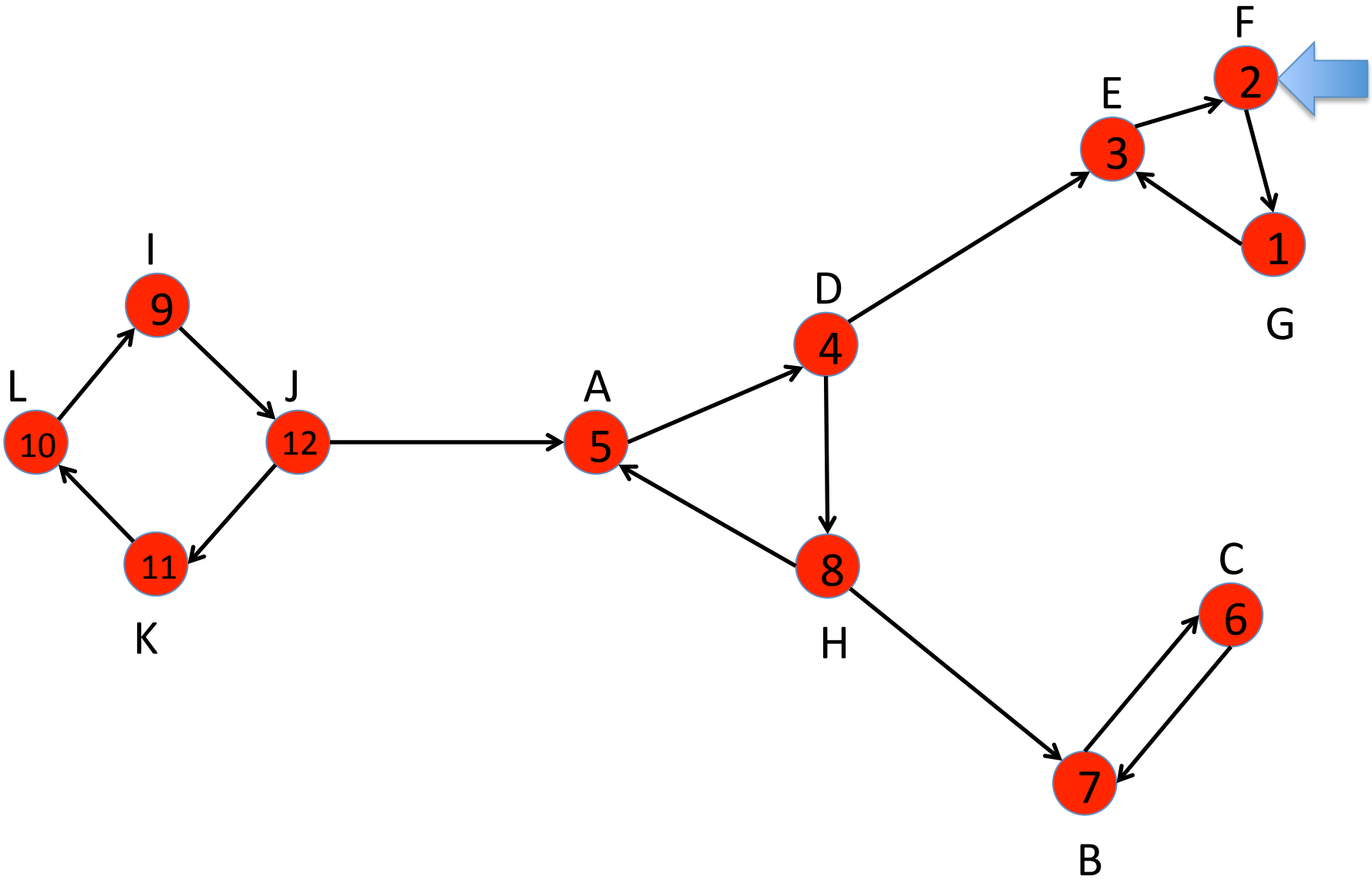
DFS & Finishing Times (2)



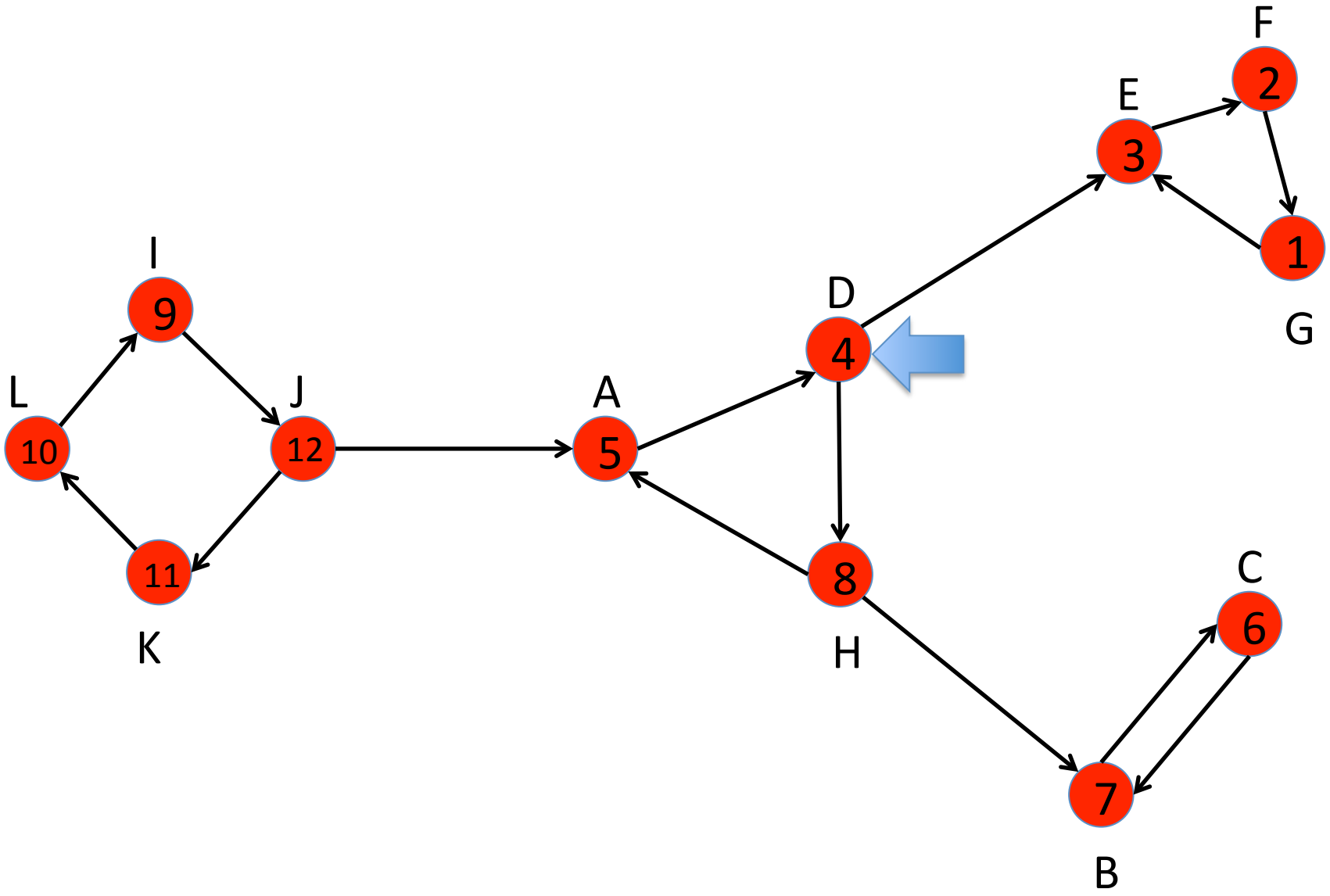
DFS & Finishing Times (2)



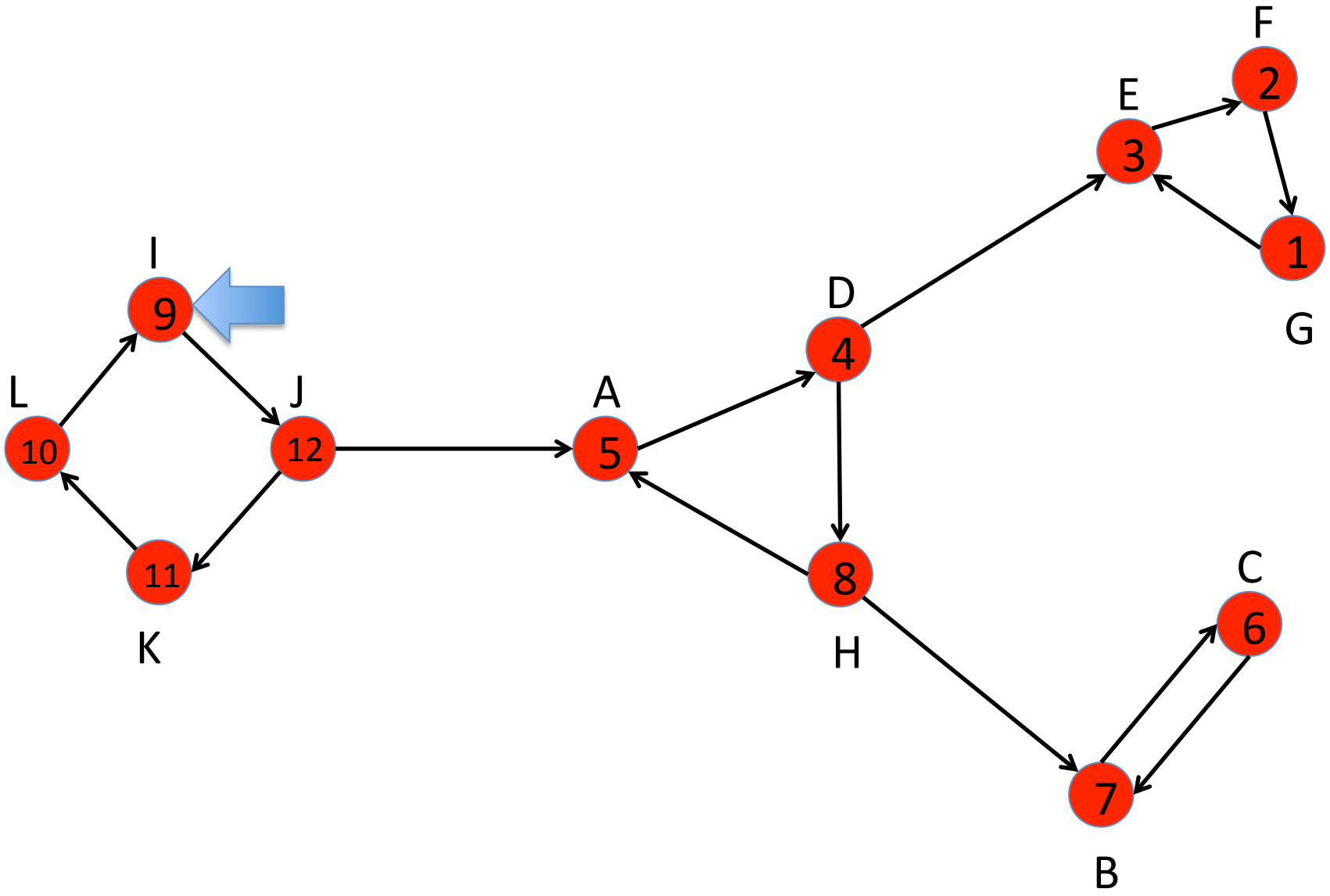
DFS & Finishing Times (2)



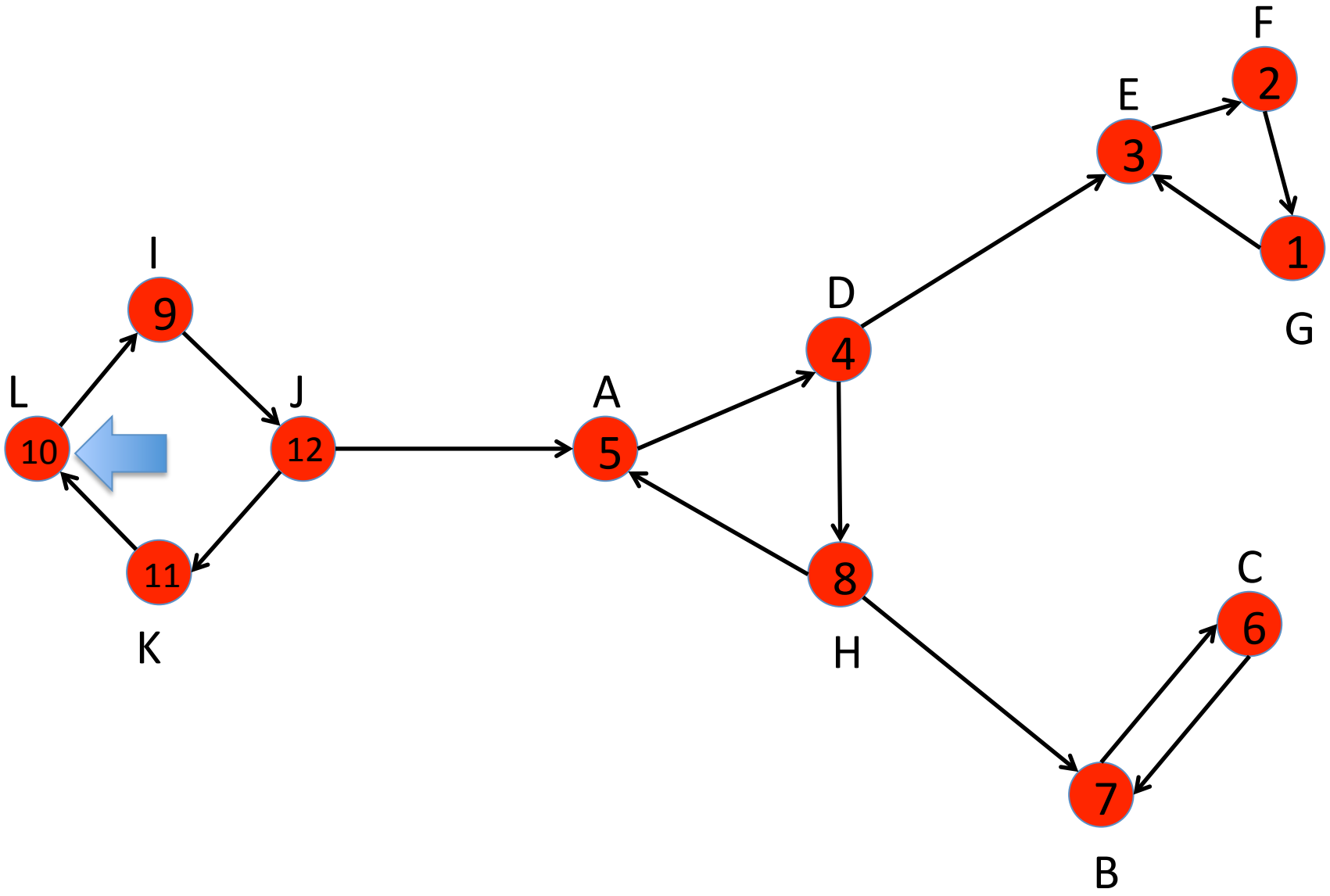
DFS & Finishing Times (2)



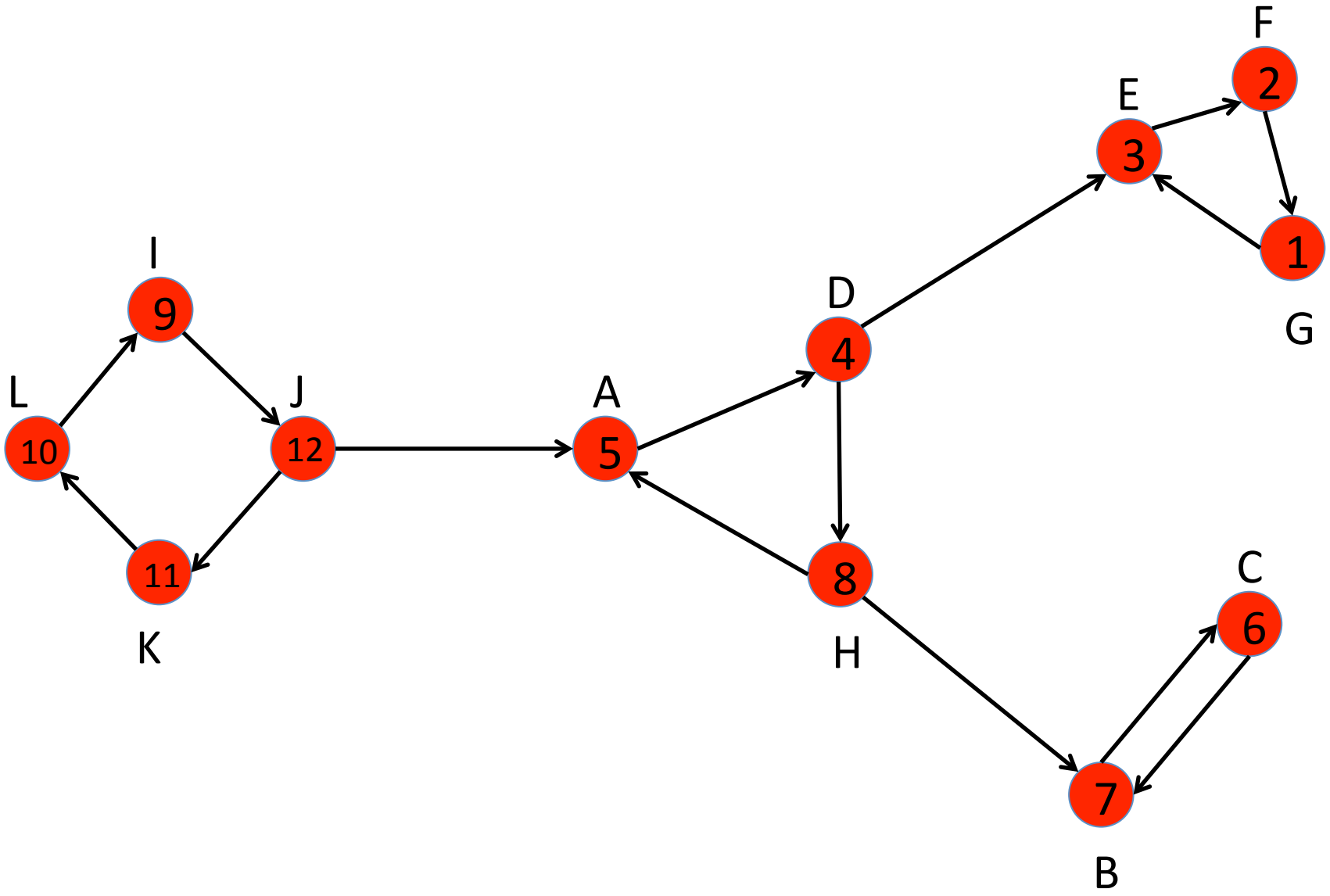
DFS & Finishing Times (2)



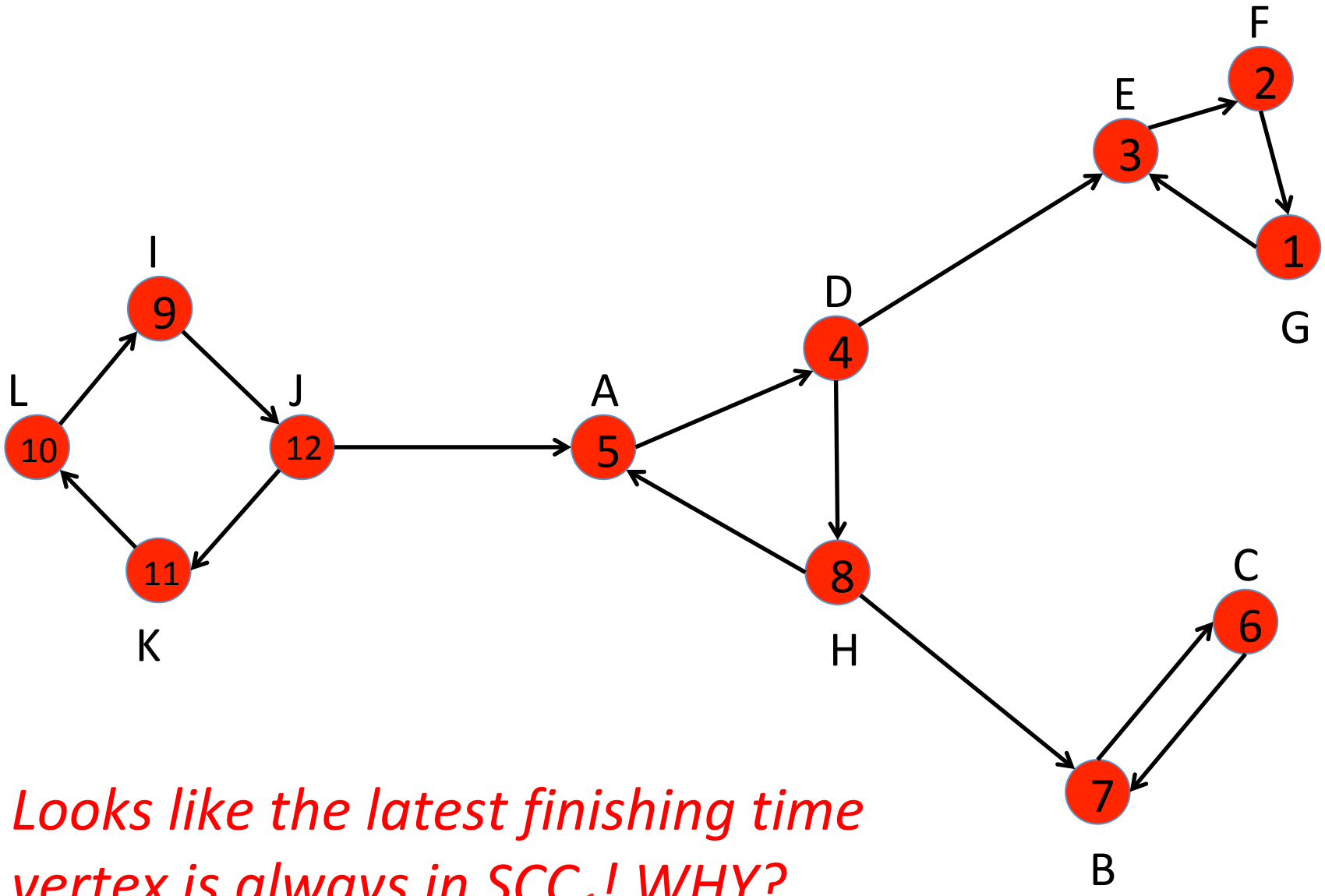
DFS & Finishing Times (2)



DFS & Finishing Times (2)



DFS & Finishing Times (2)



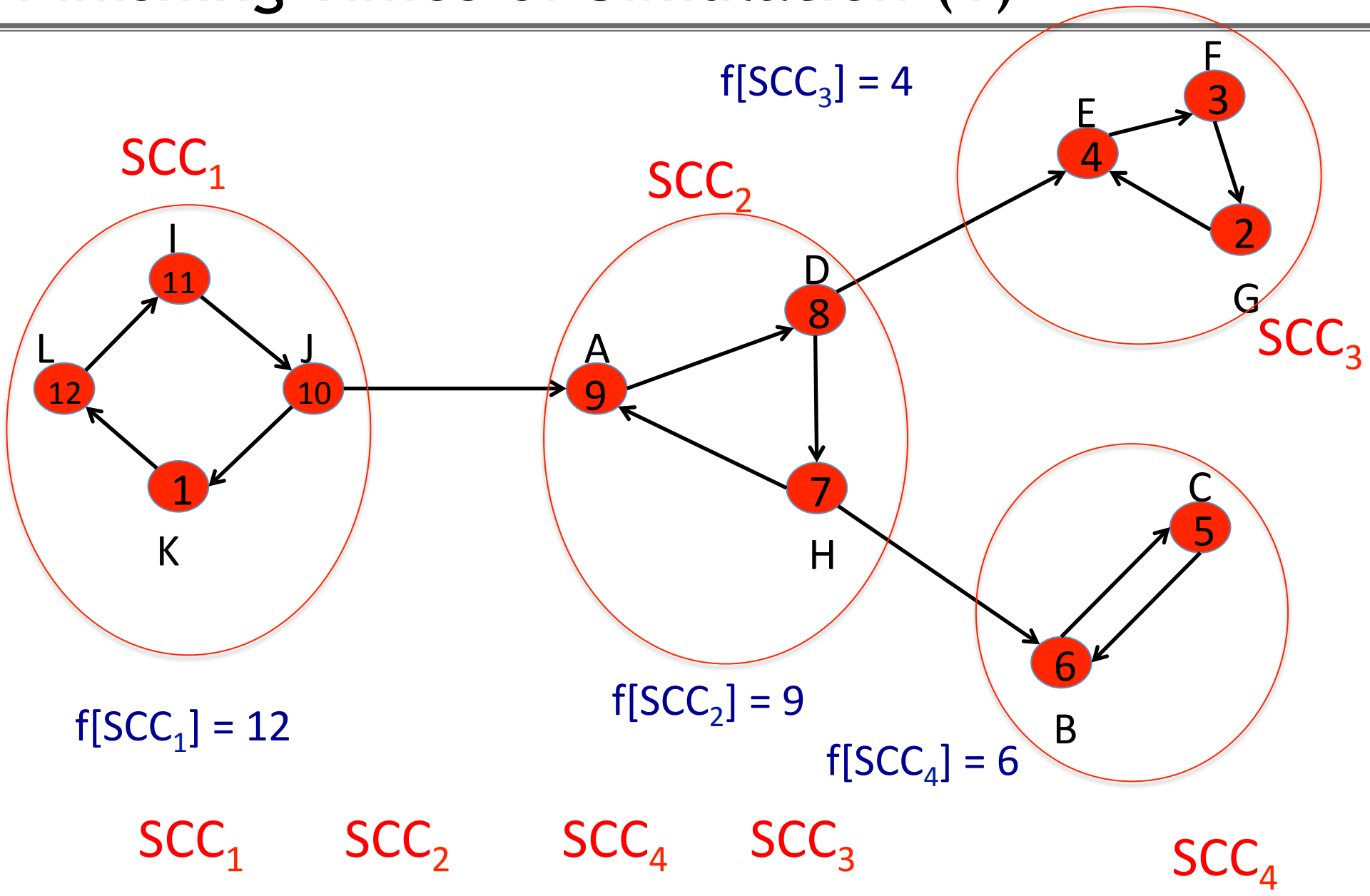
Looks like the latest finishing time vertex is always in SCC_1! WHY?

Definition: Finishing Time of an SCC

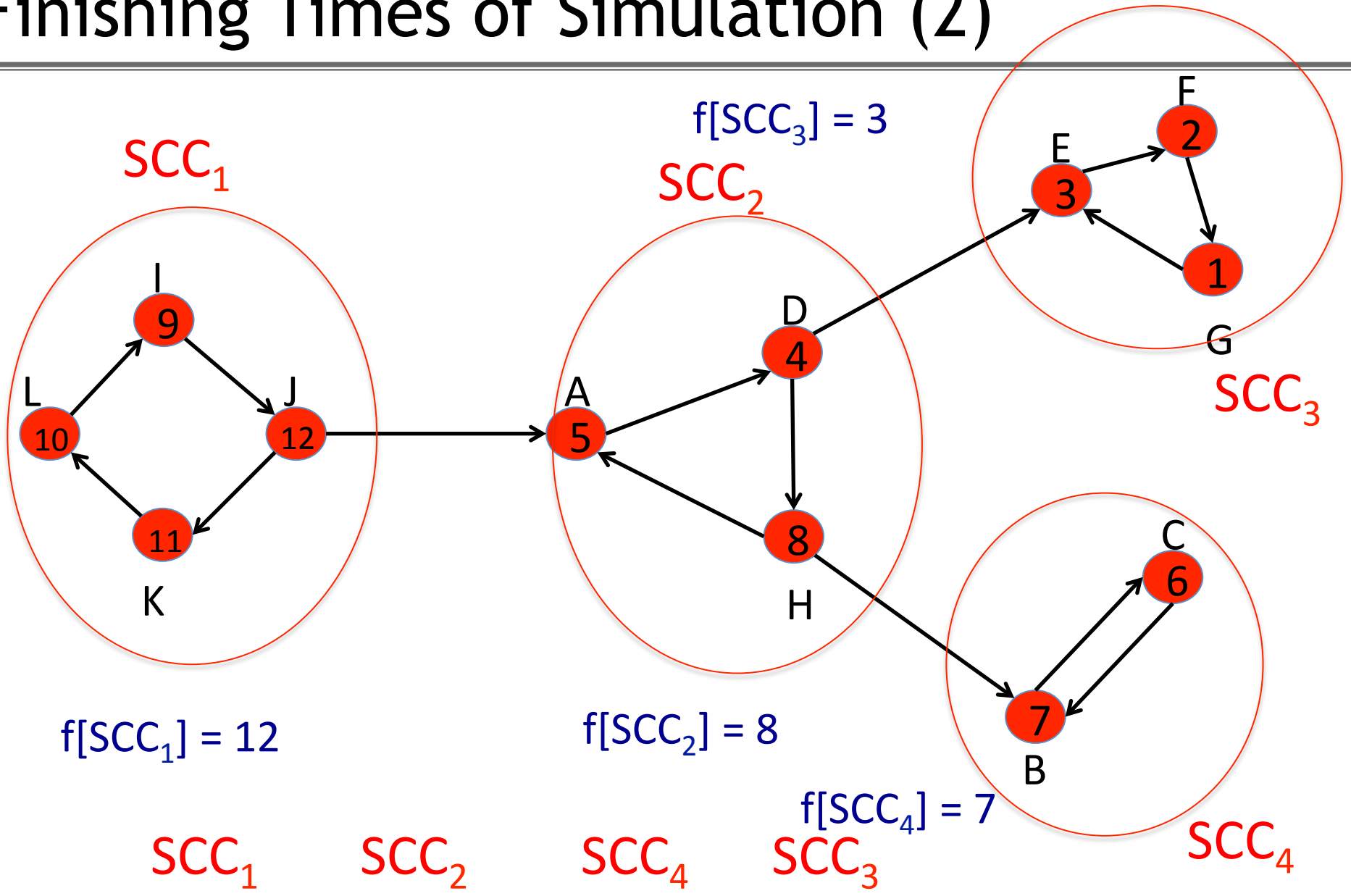
- Let "finishing time of an SCC_i" in DFS traversal be the maximum finishing time of the vertices in SCC_i

$$f[SCC_i] = \max_{v \in SCC_i} f[v]$$

Finishing Times of Simulation (1)



Finishing Times of Simulation (2)



Key Lemma about finishing times of SCCs

In G^{SCC} if $SCC_i \rightarrow SCC_j$, then
$***f[SCC_i] > f[SCC_j]***$

Will prove this later

Exstended Key Lemma

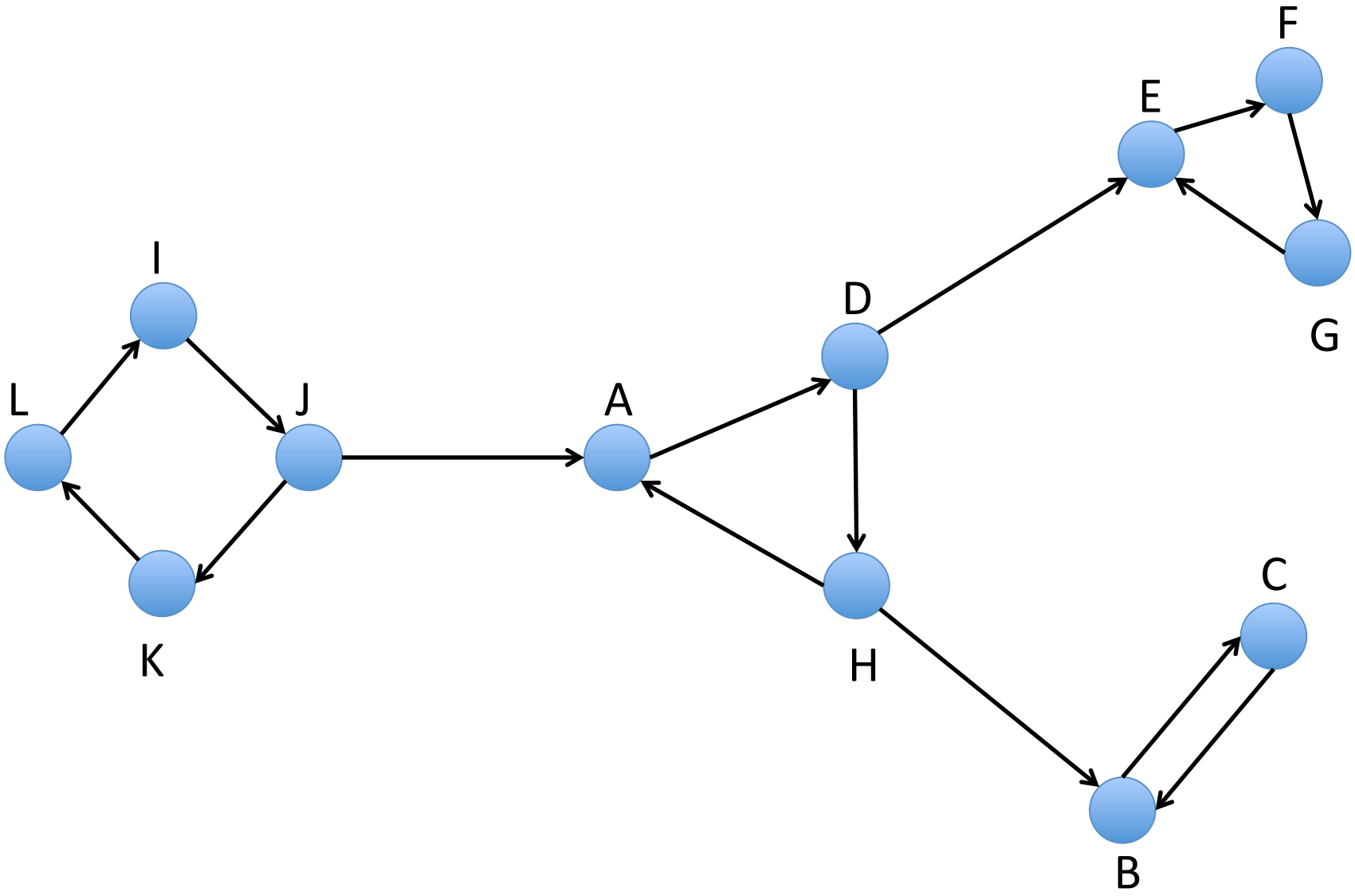
In G^{SCC} if $SCC_i \rightsquigarrow SCC_j$, then
****$f[SCC_i] > f[SCC_j]$****

Can prove by induction on the length of the paths using Key Lemma.

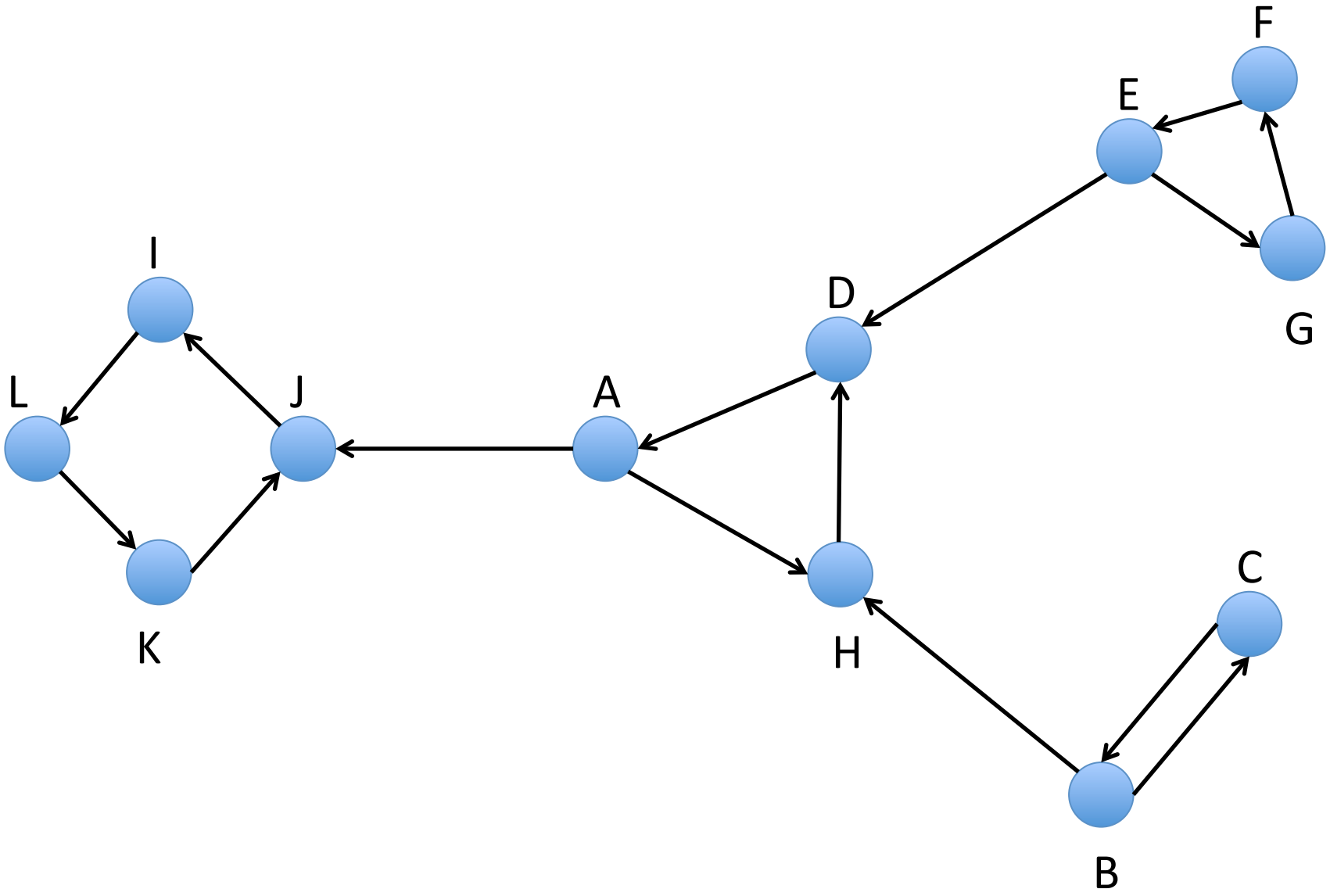
Final Tricks

- Recall we want to identify sinks NOT sources!
- So far it looks like we can identify sources.
- What if we took the reverse of G => G_{rev}
- And ran DFS on G_{rev} and studied the finishing times

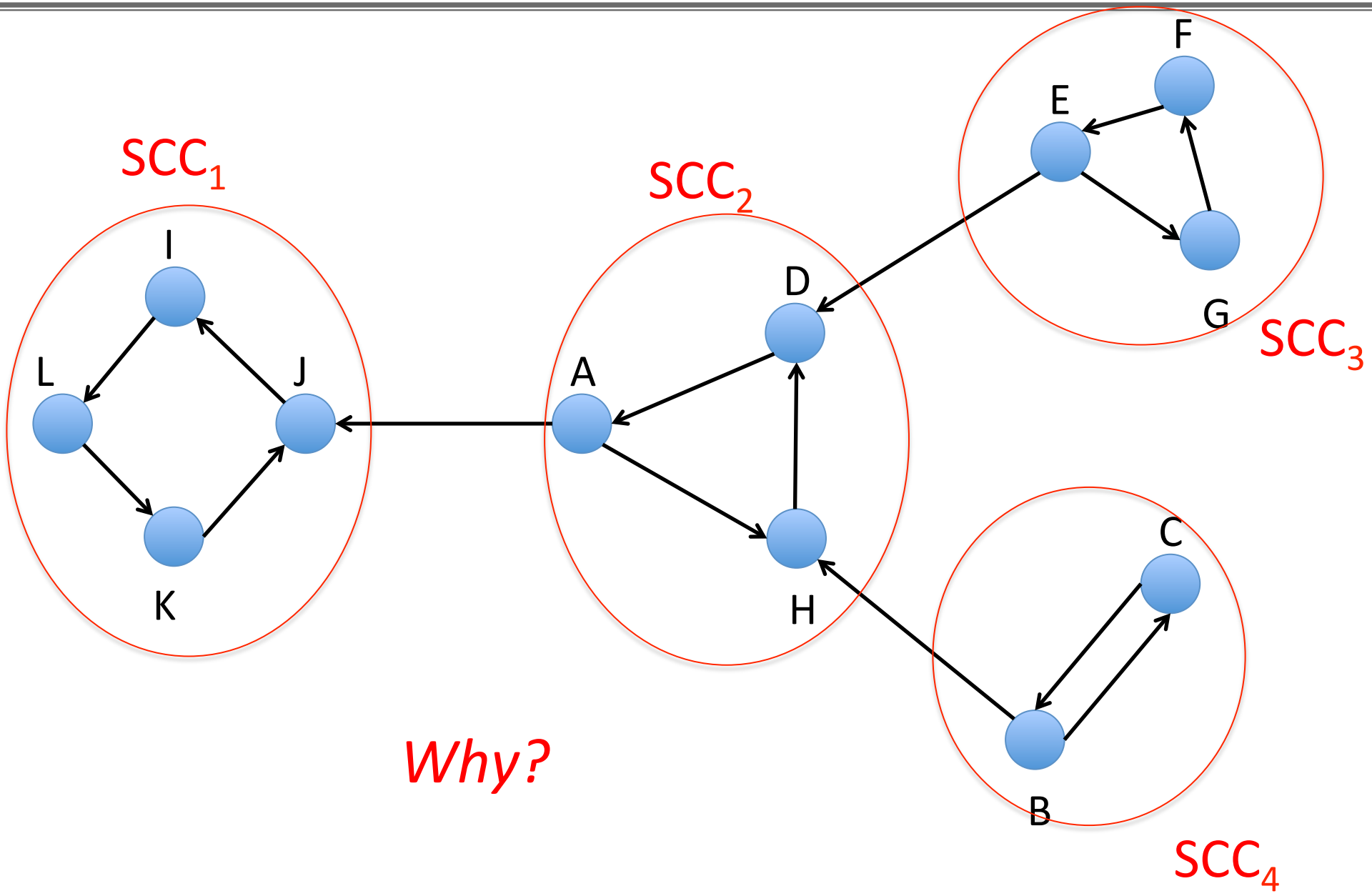
Original Graph G



G_{rev}



G_{rev} has the exact same SCCs as G!

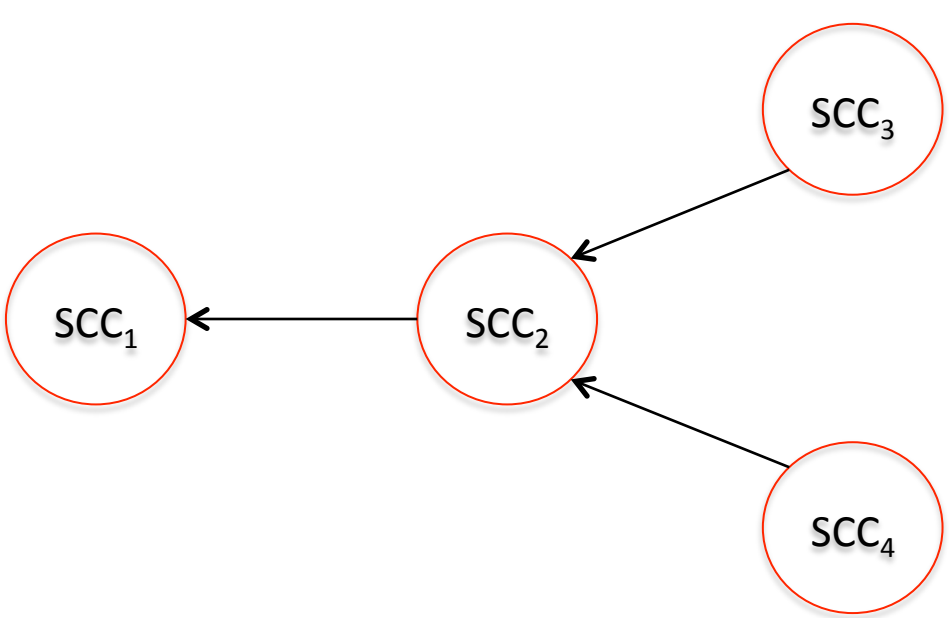


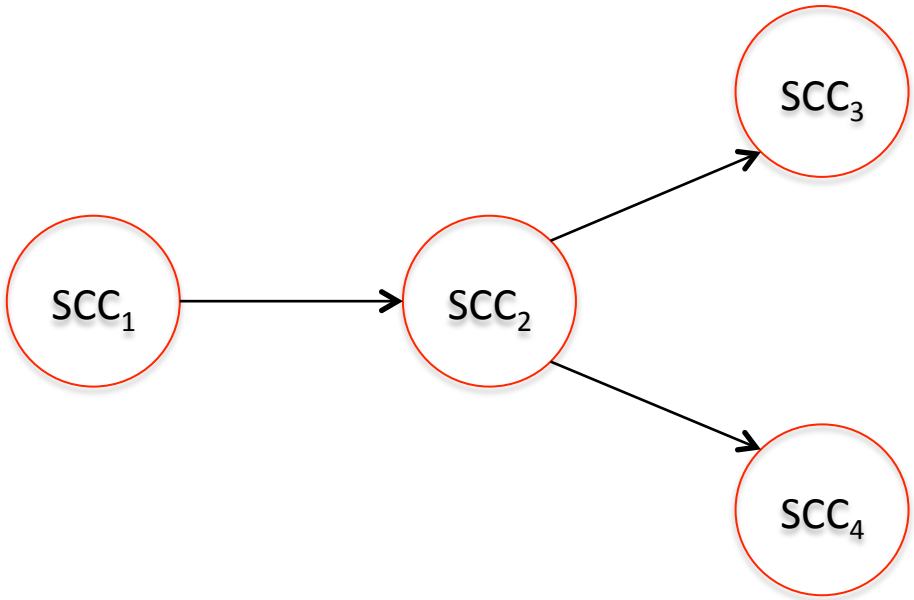
Properties of G_{rev}

1. G_{rev} and G have the same SCCs
2. The $(G_{rev})^{SCC}$ is the reverse of G^{SCC}

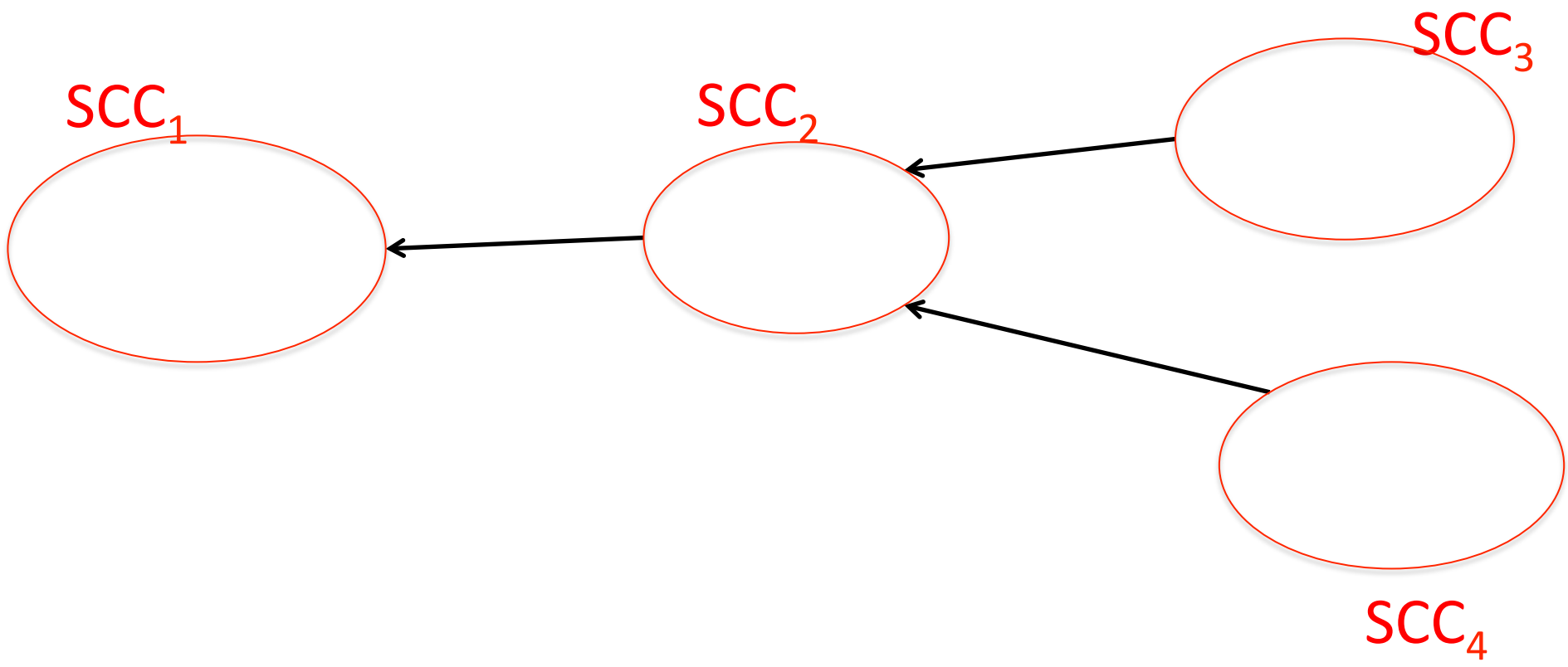
SCC meta-graph of G_{rev}

SCC meta-graph of G





Running DFS on G_{rev}

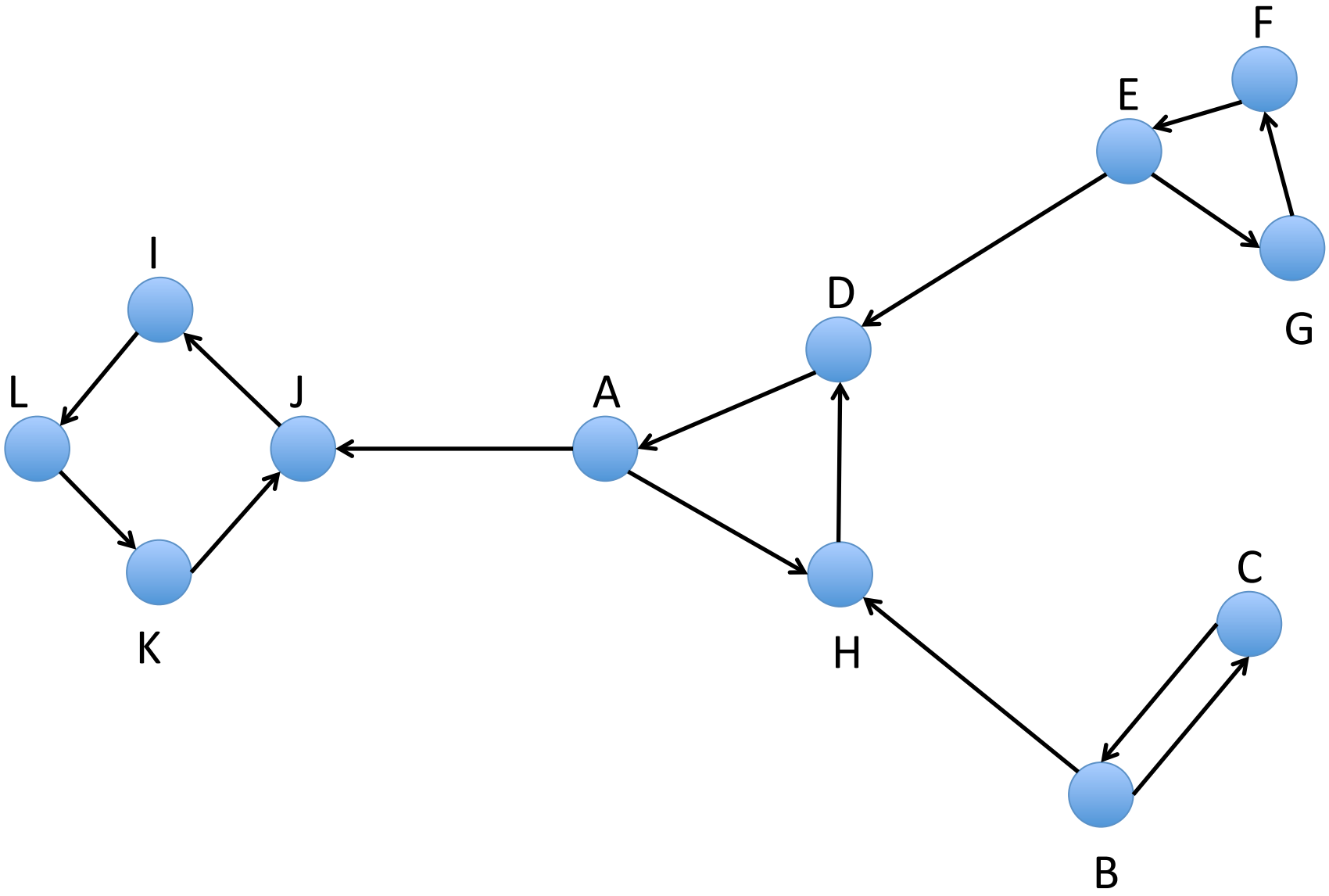


By Key Lemma we should get:

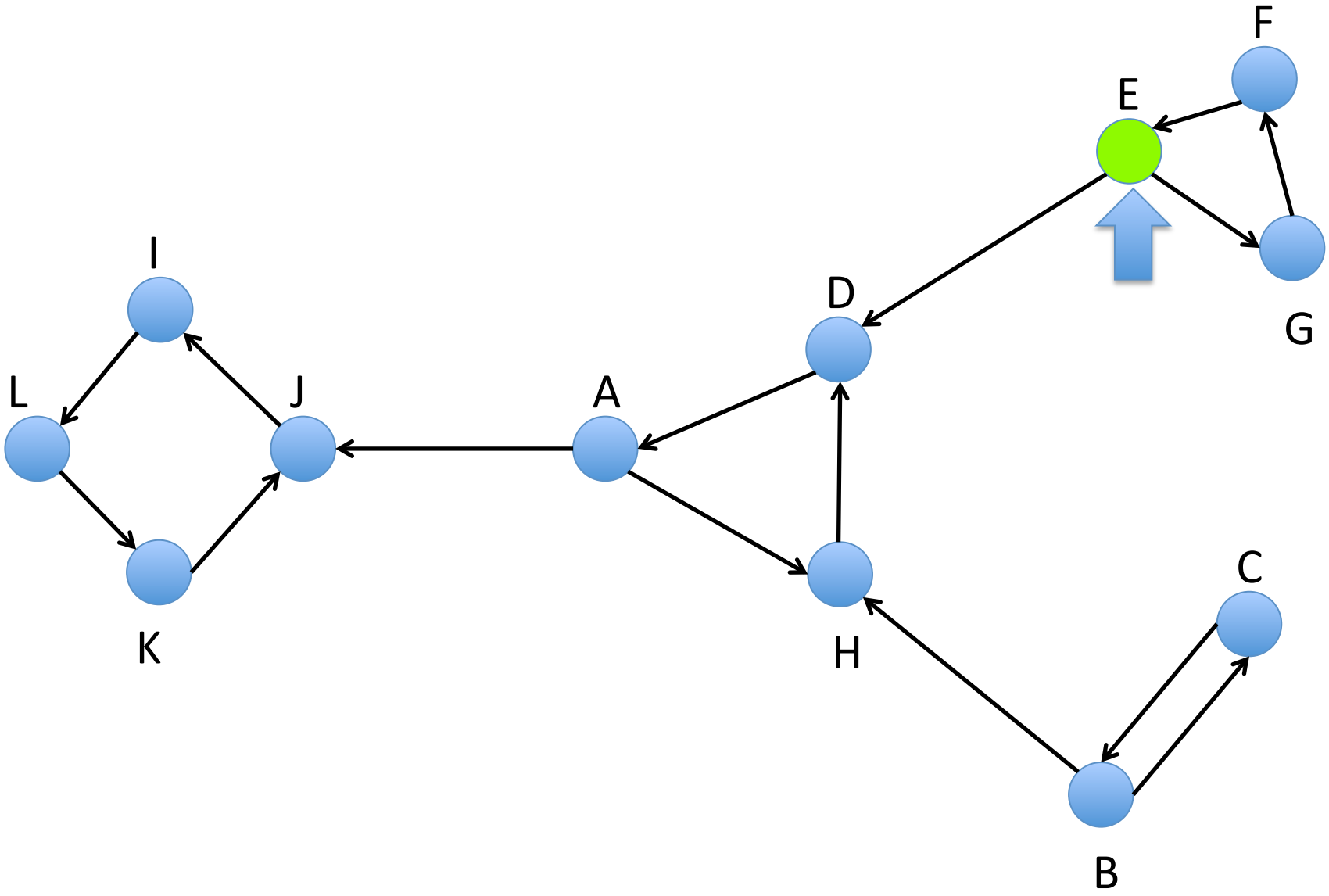
$f[SCC_3] > f[SCC_4] > f[SCC_2] > f[SCC_1]$ OR

$f[SCC_4] > f[SCC_3] > f[SCC_2] > f[SCC_1]$

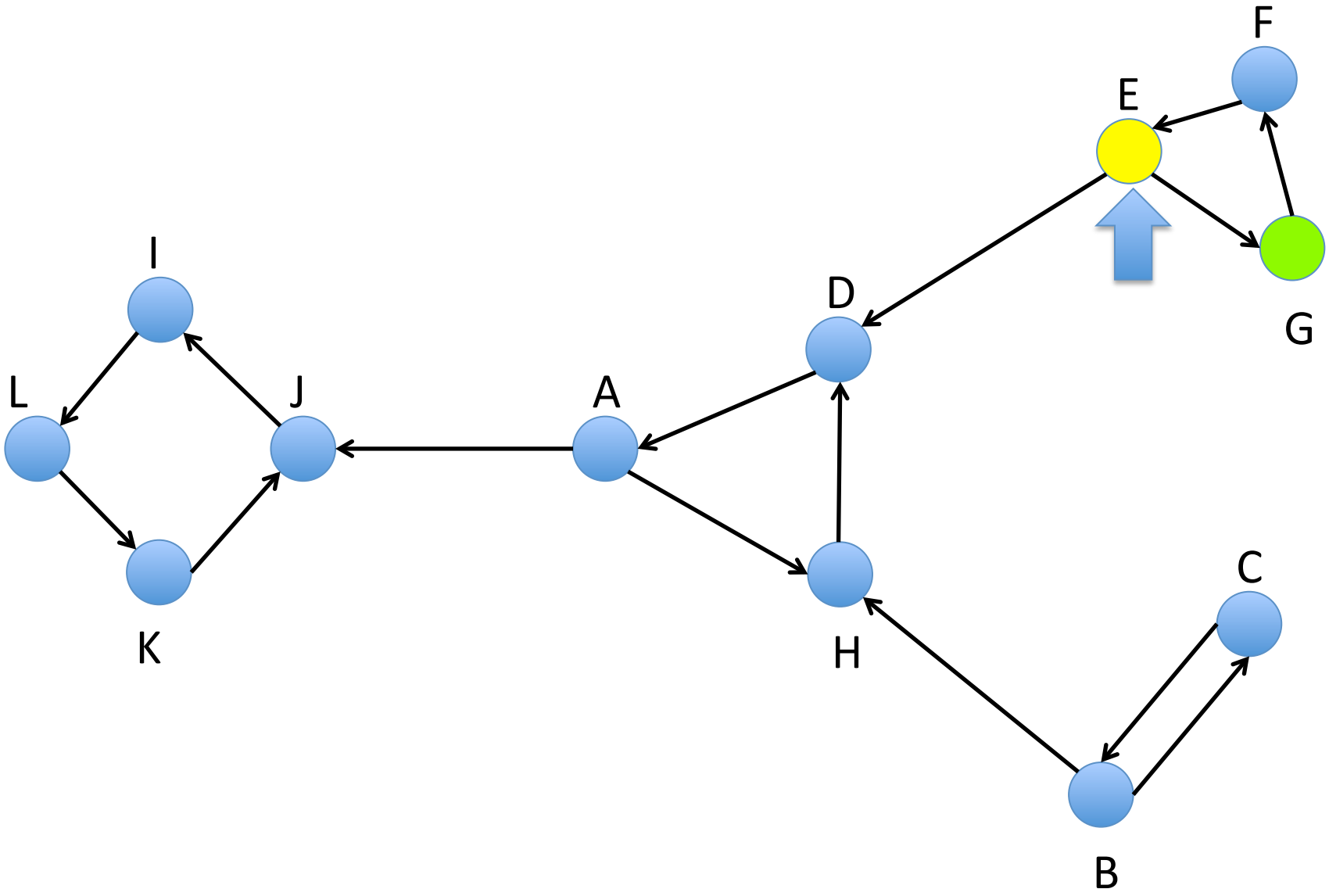
DFS & Finishing Times on Grev



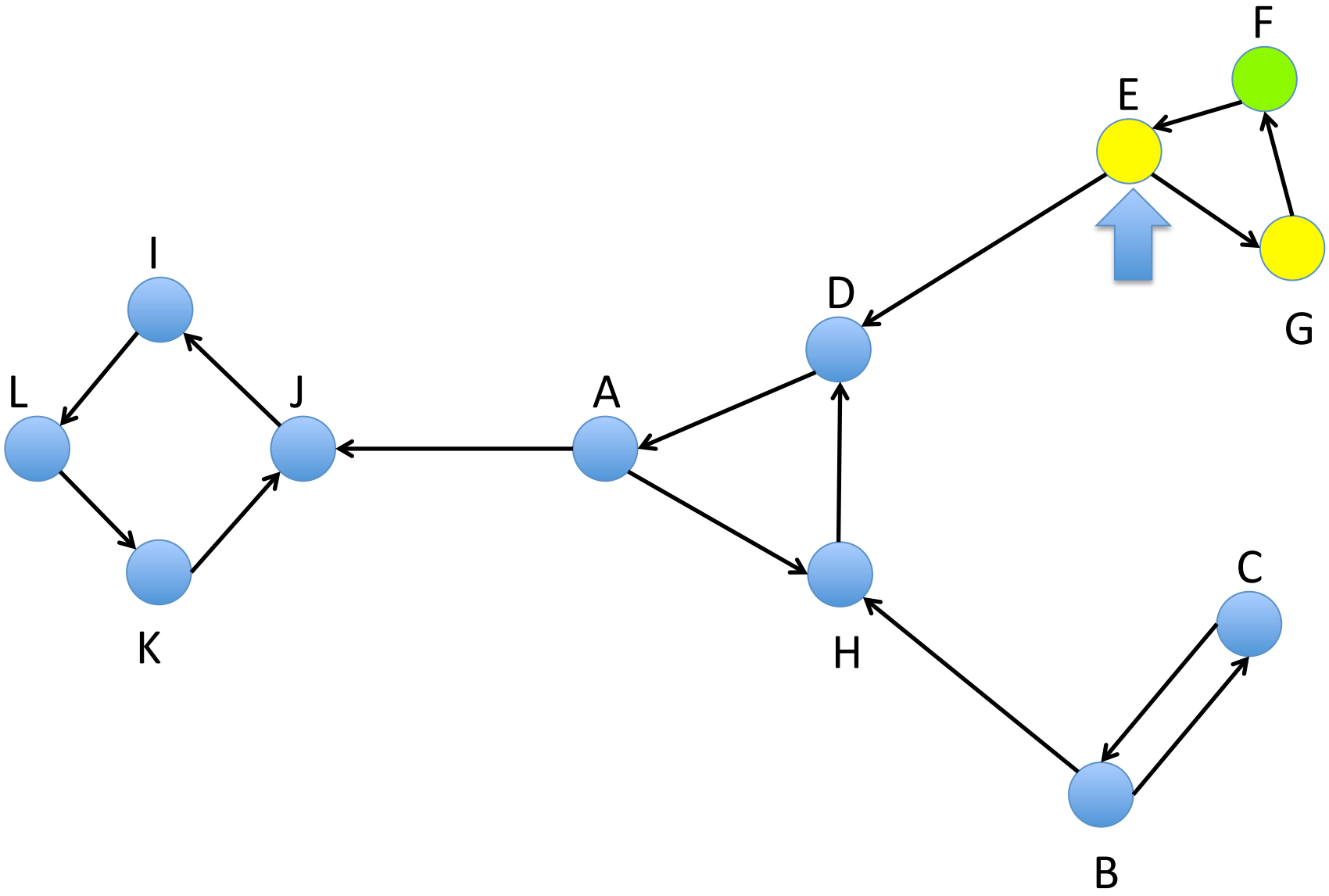
DFS & Finishing Times on Grev



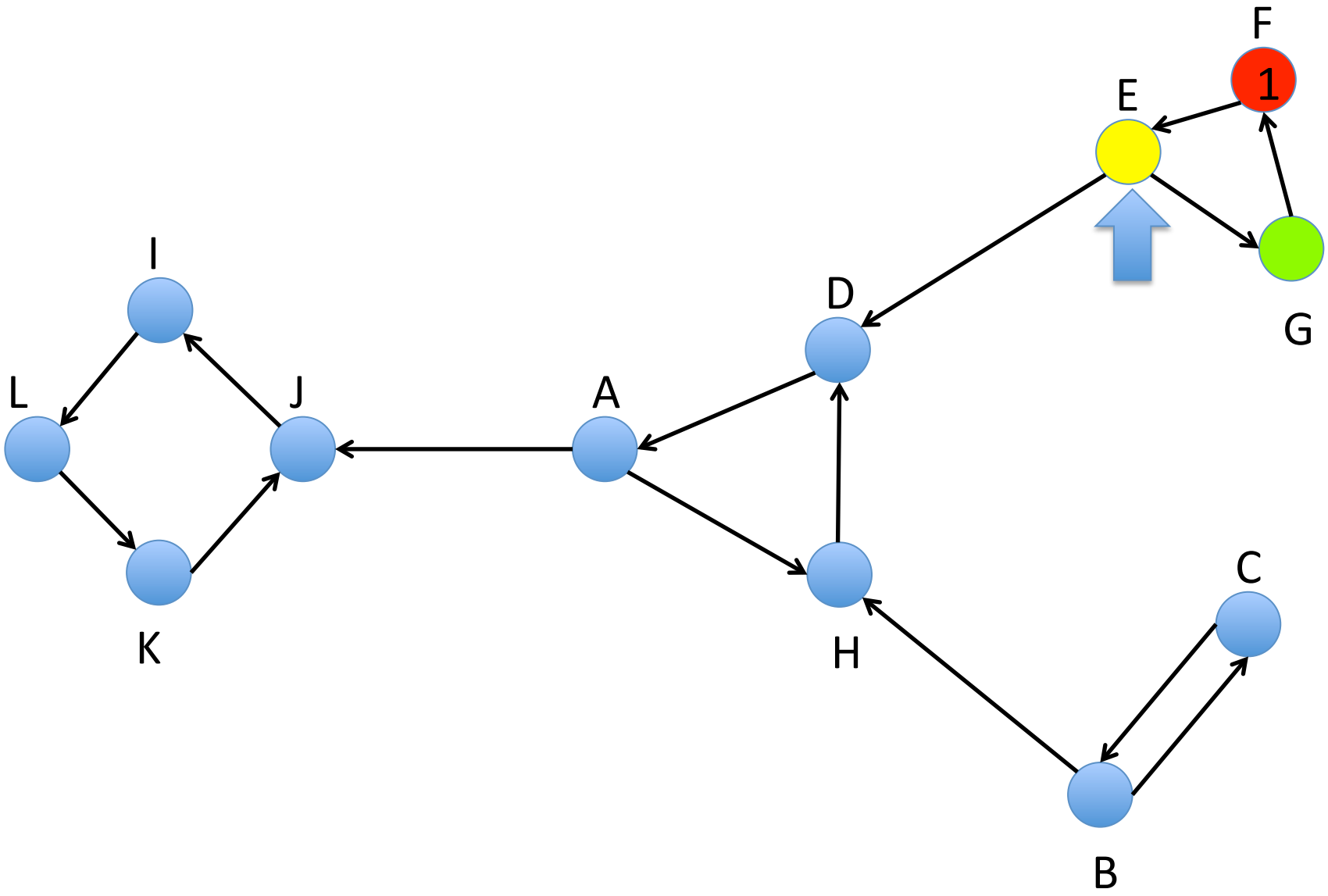
DFS & Finishing Times on Grev



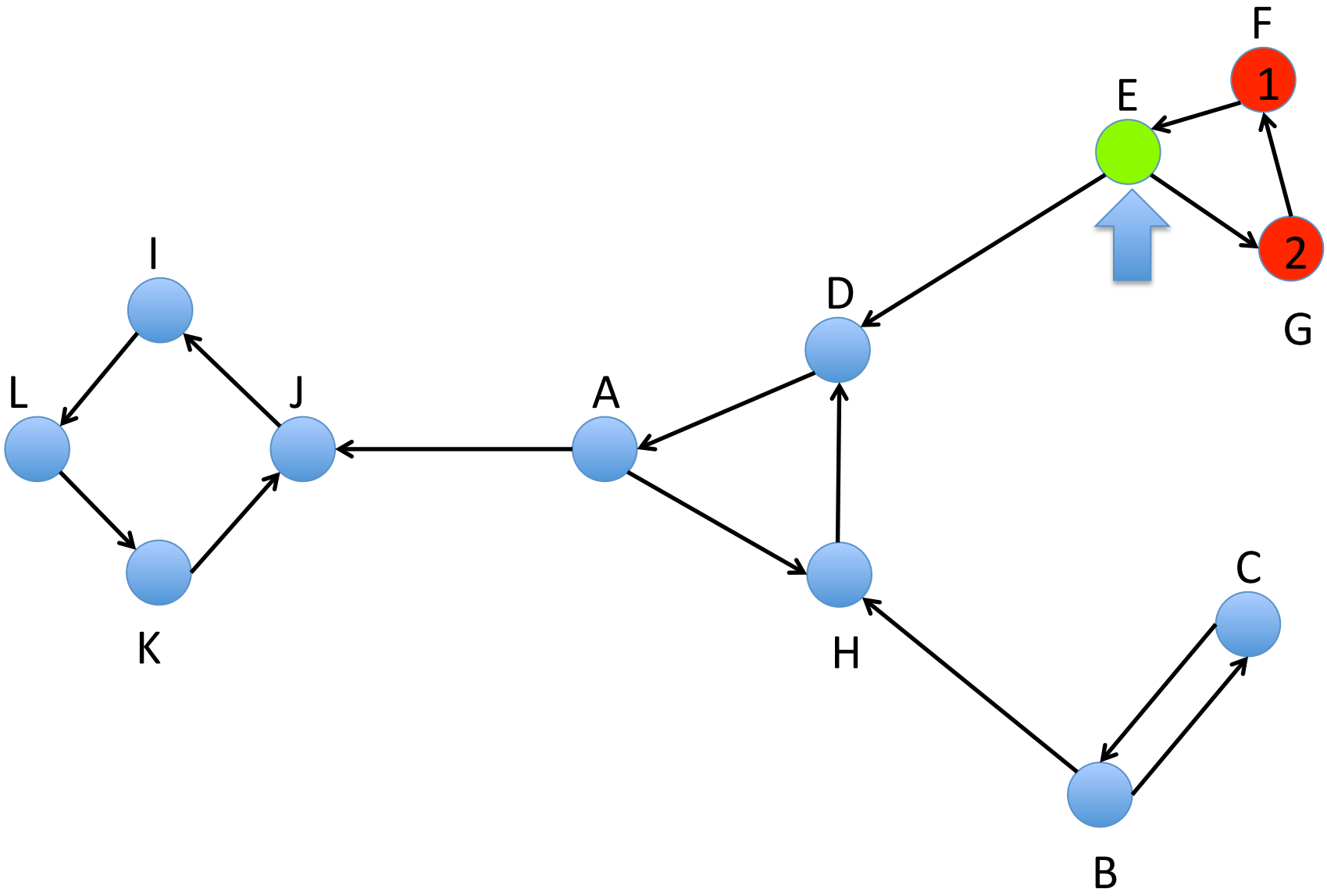
DFS & Finishing Times on Grev



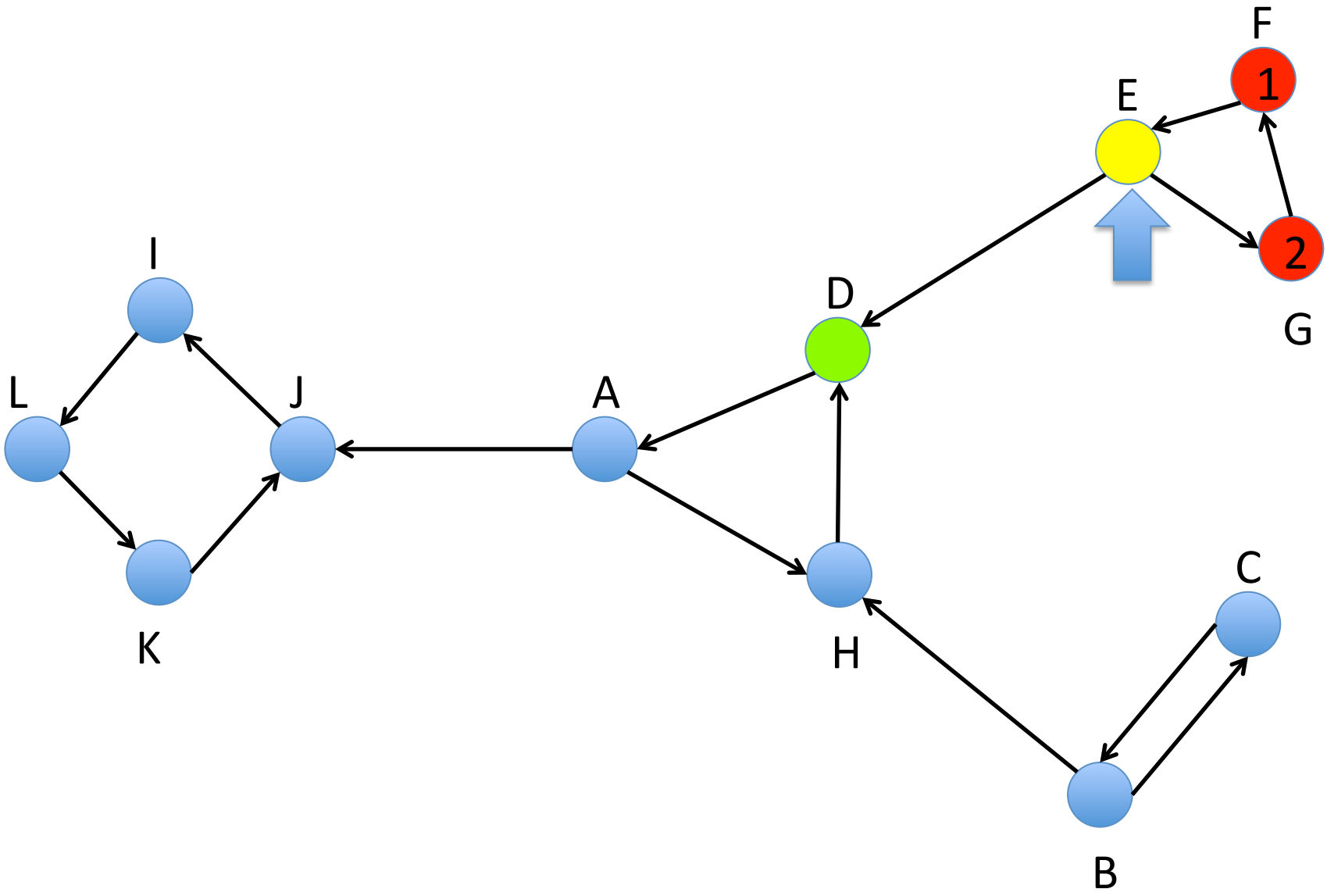
DFS & Finishing Times on Grev



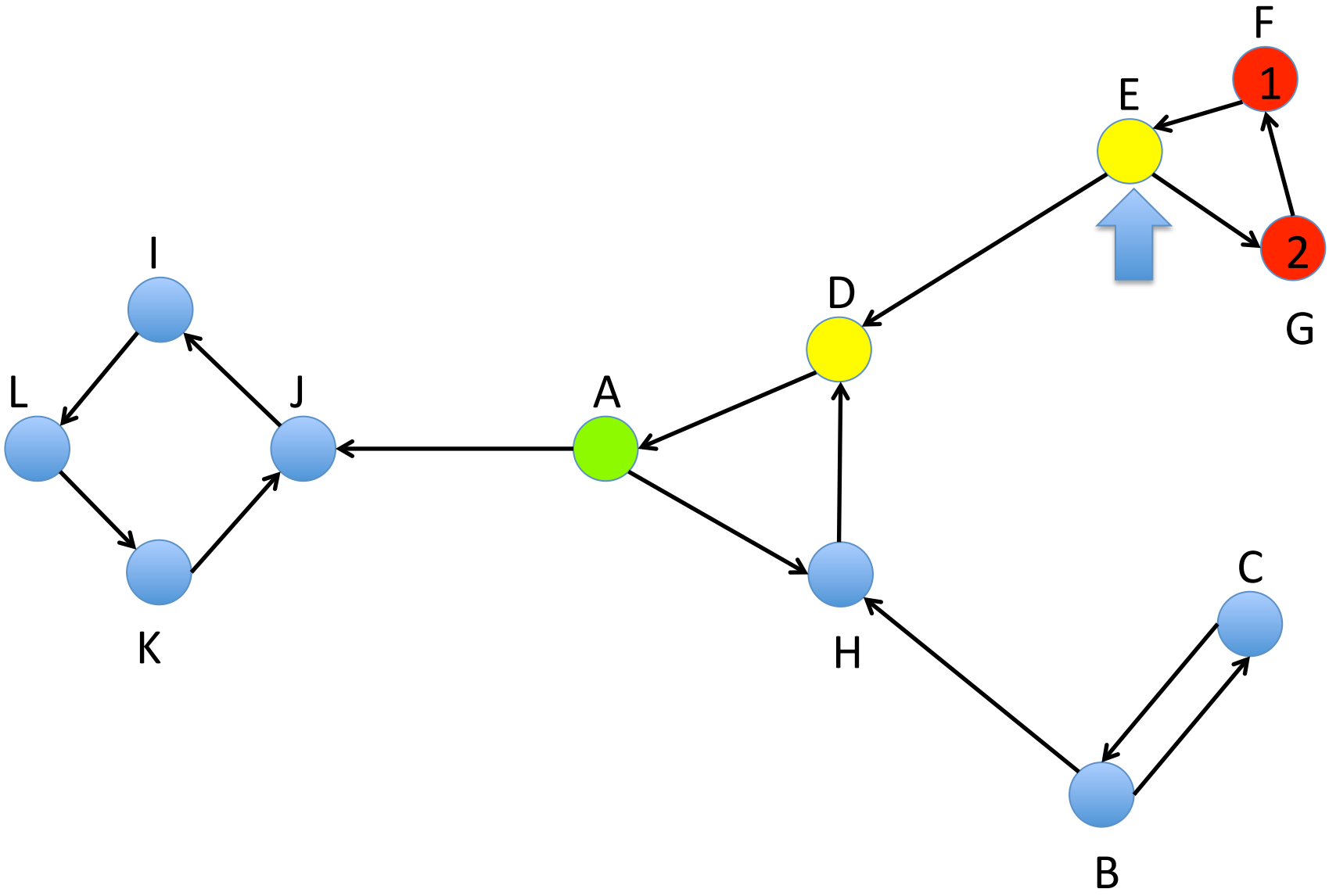
DFS & Finishing Times on Grev



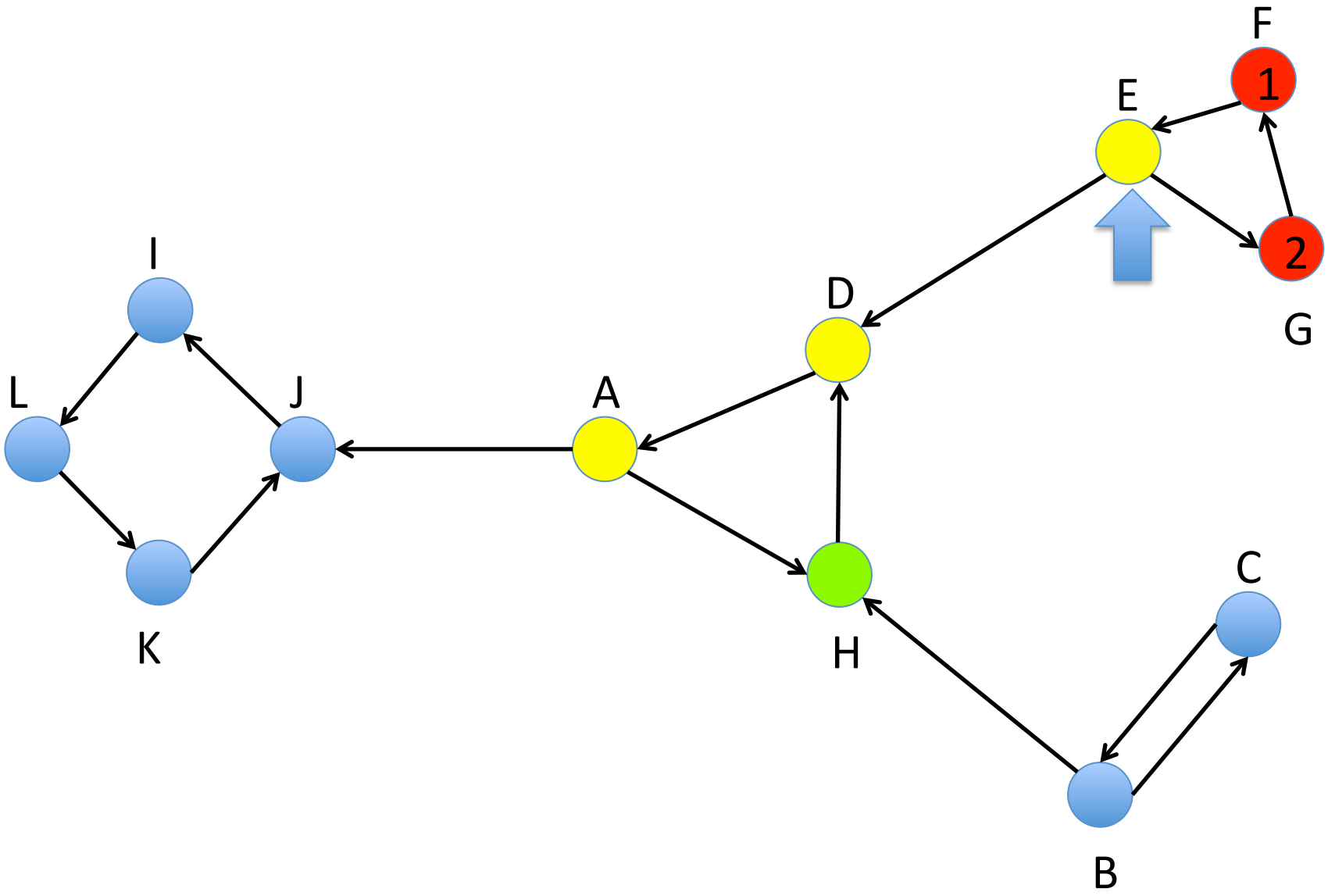
DFS & Finishing Times on Grev



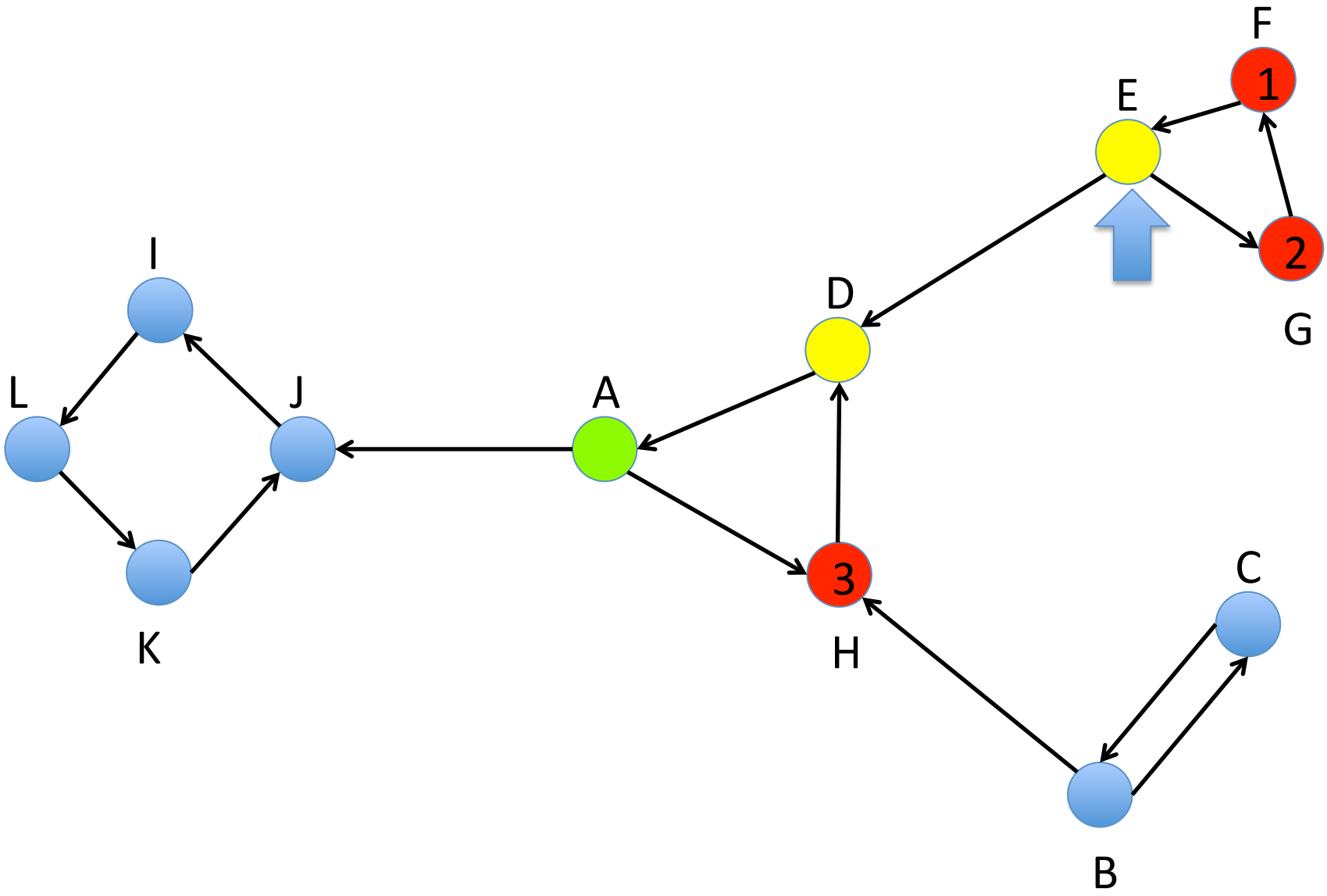
DFS & Finishing Times on Grev



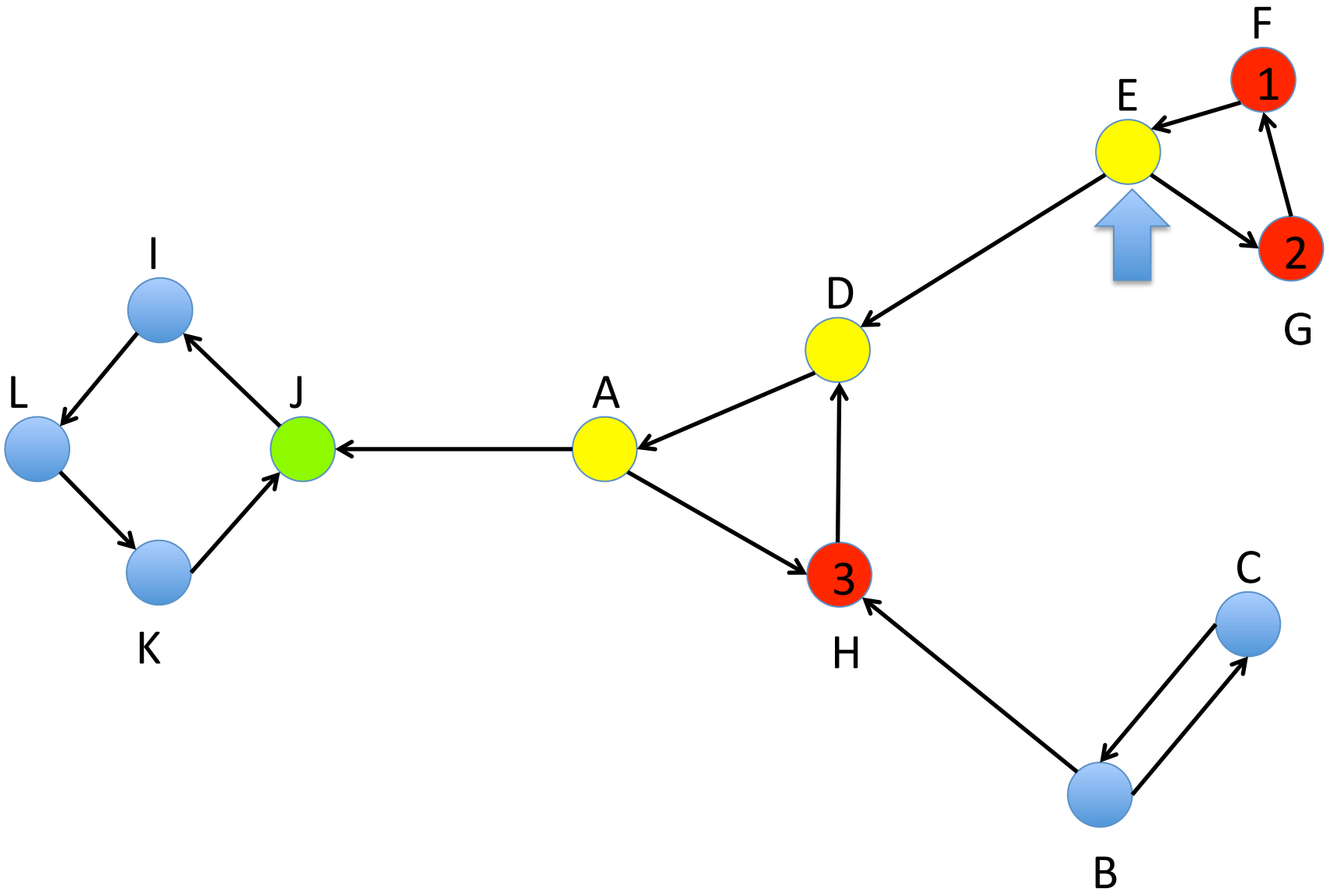
DFS & Finishing Times on Grev



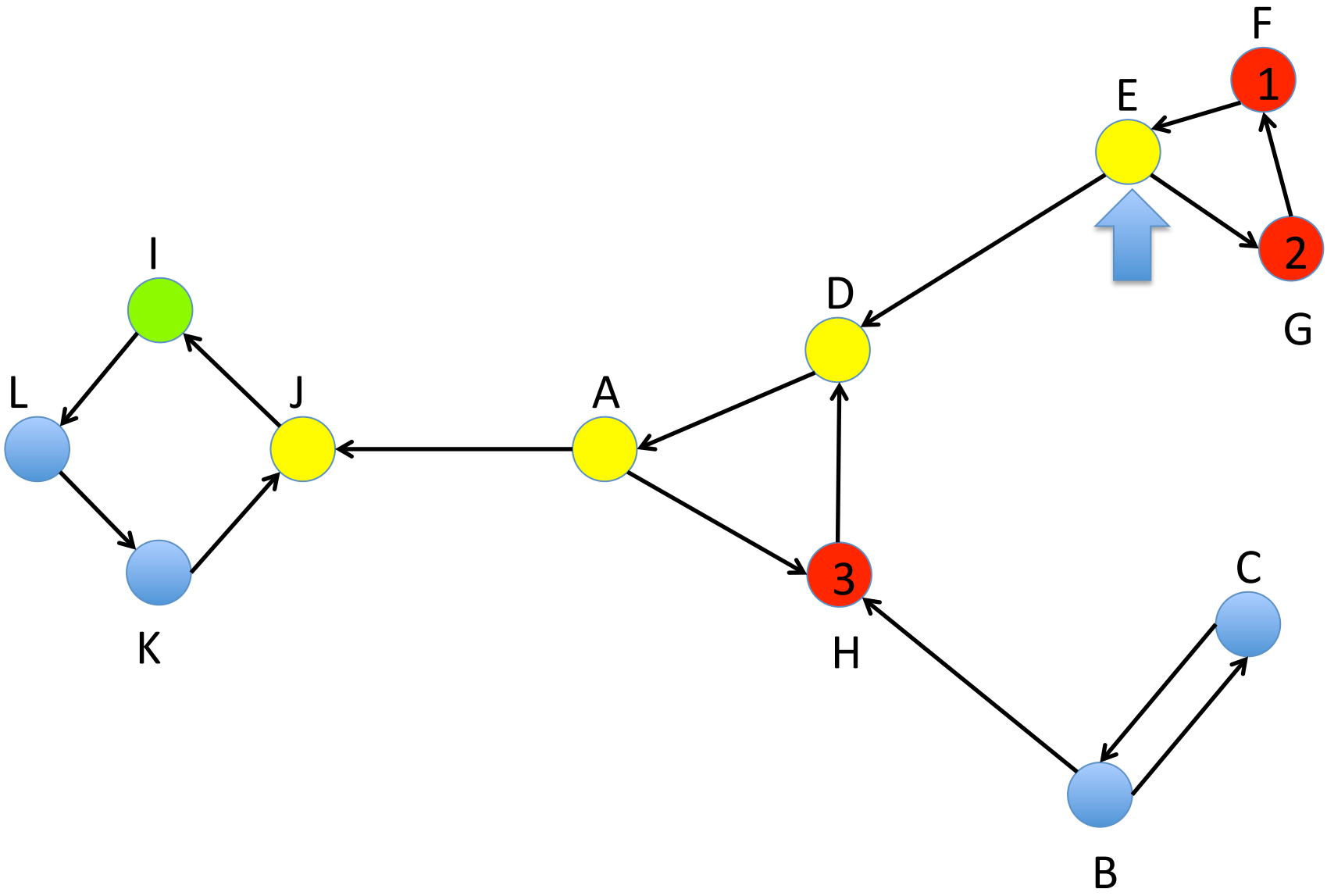
DFS & Finishing Times on Grev



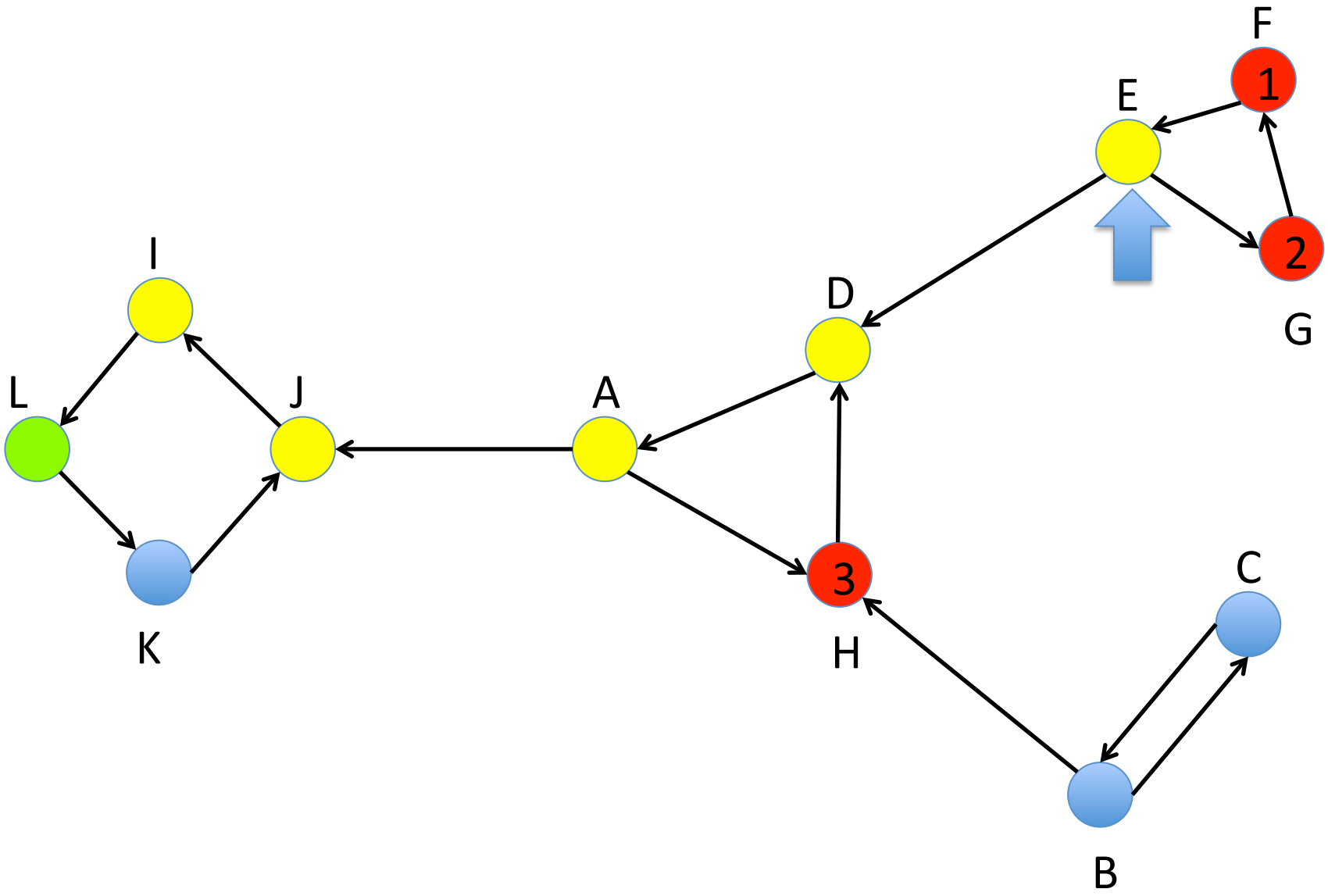
DFS & Finishing Times on Grev



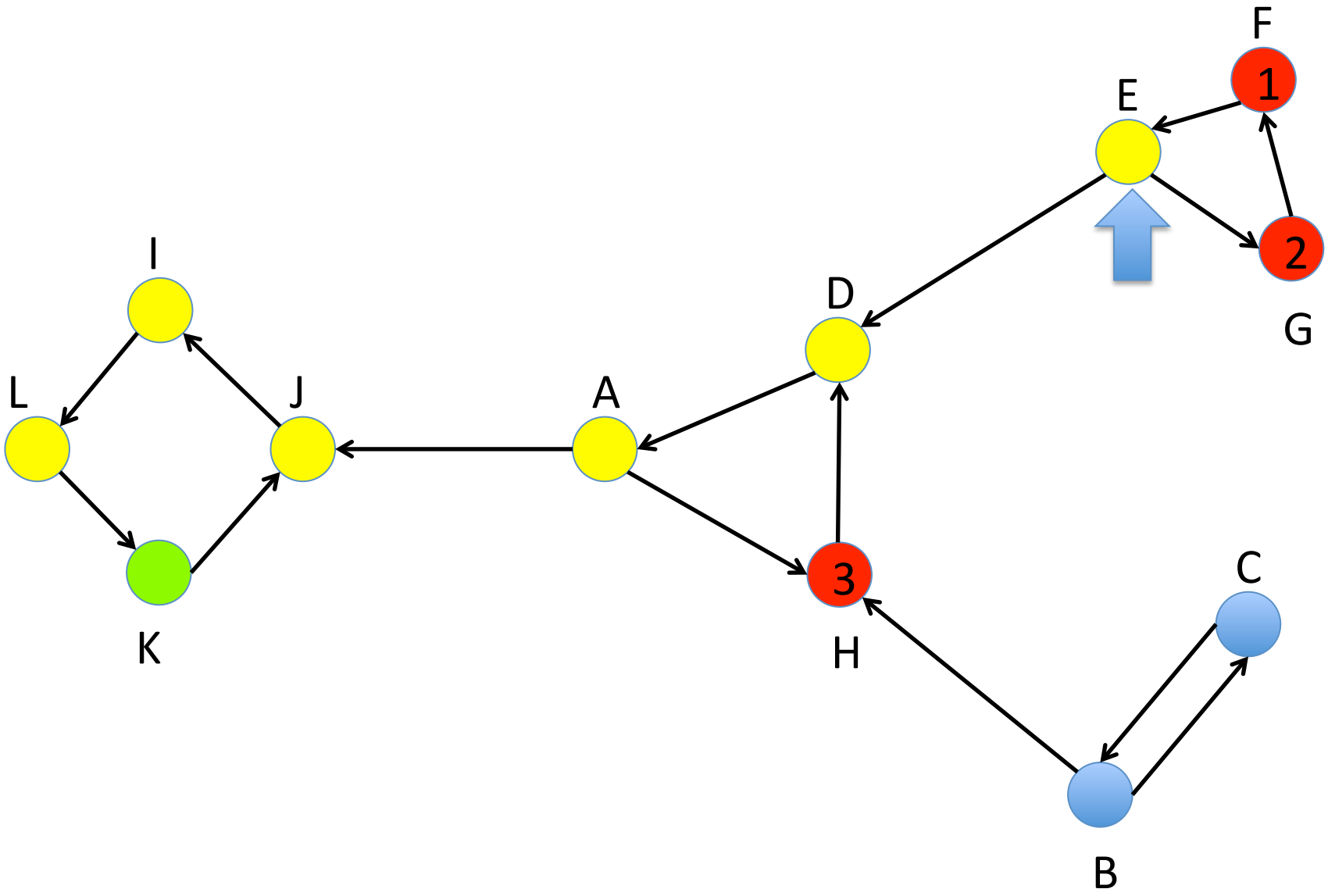
DFS & Finishing Times on Grev



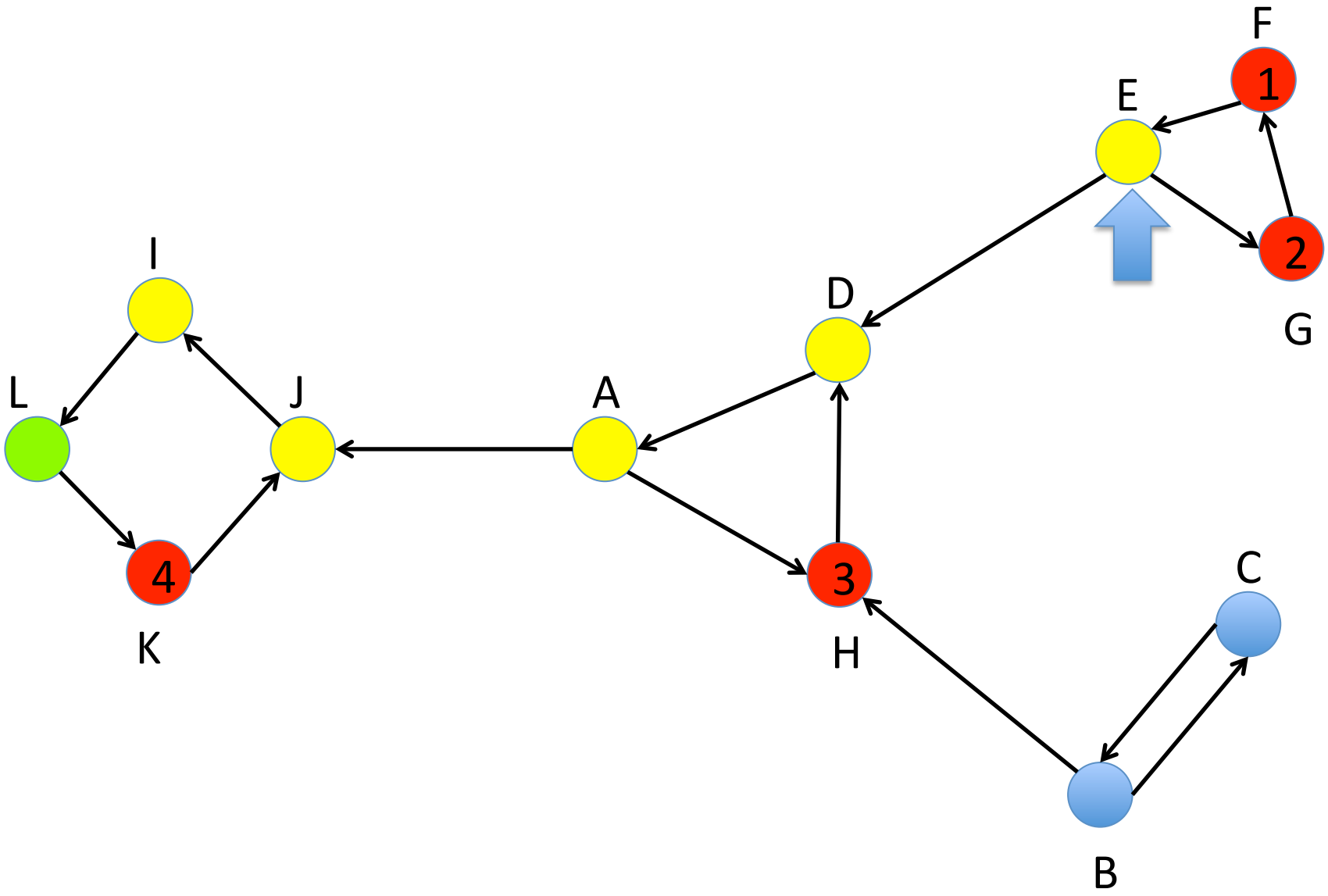
DFS & Finishing Times on Grev



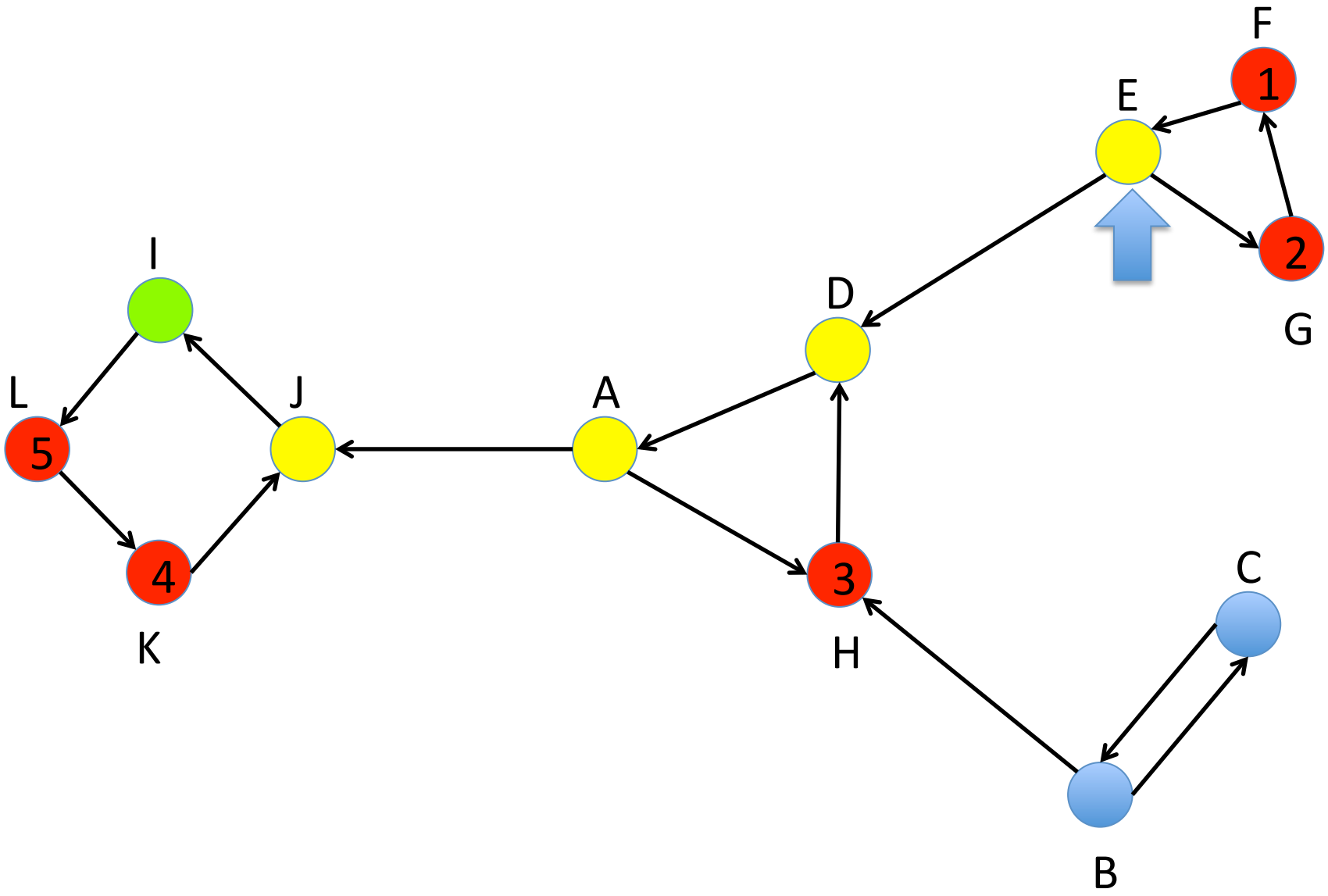
DFS & Finishing Times on Grev



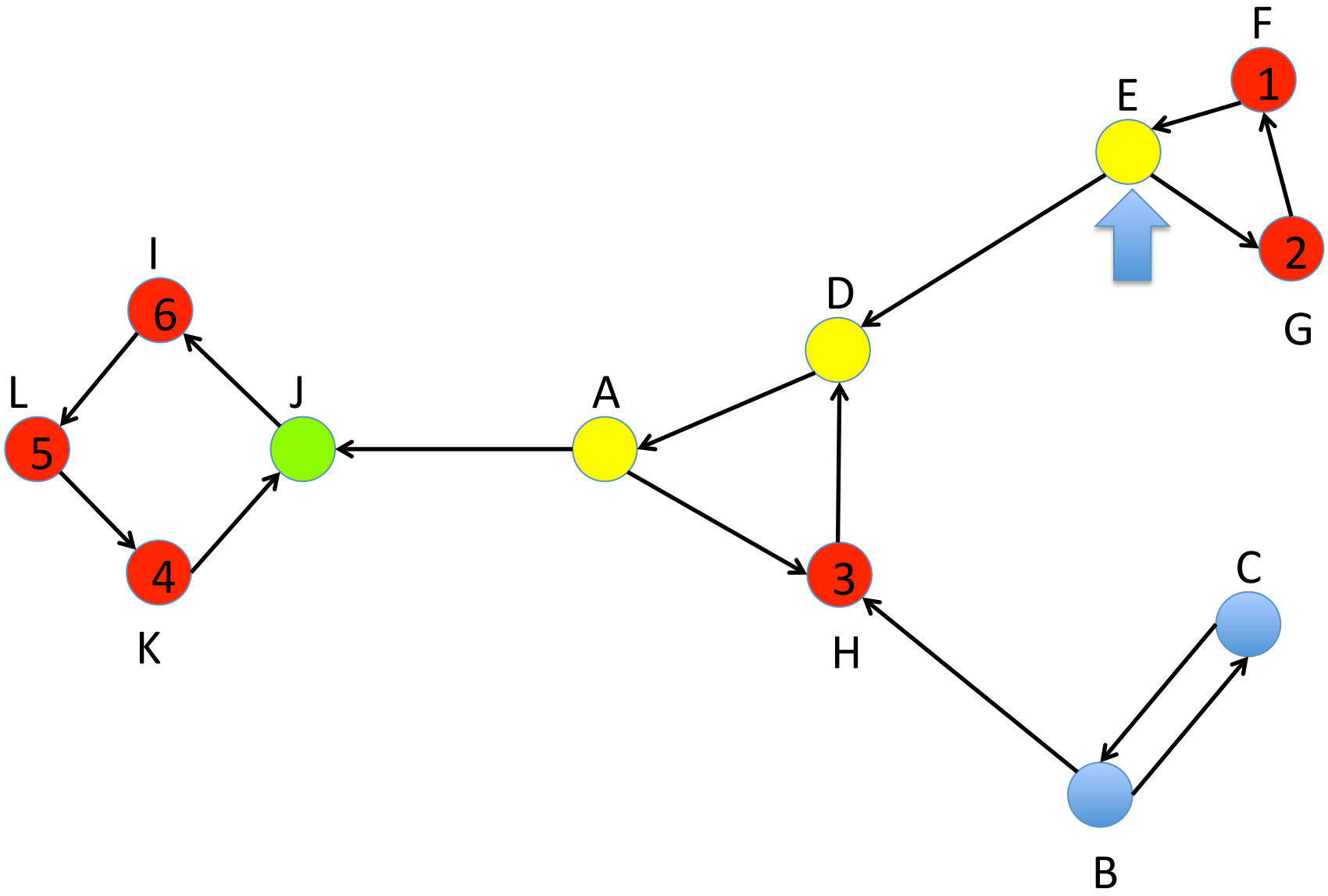
DFS & Finishing Times on Grev



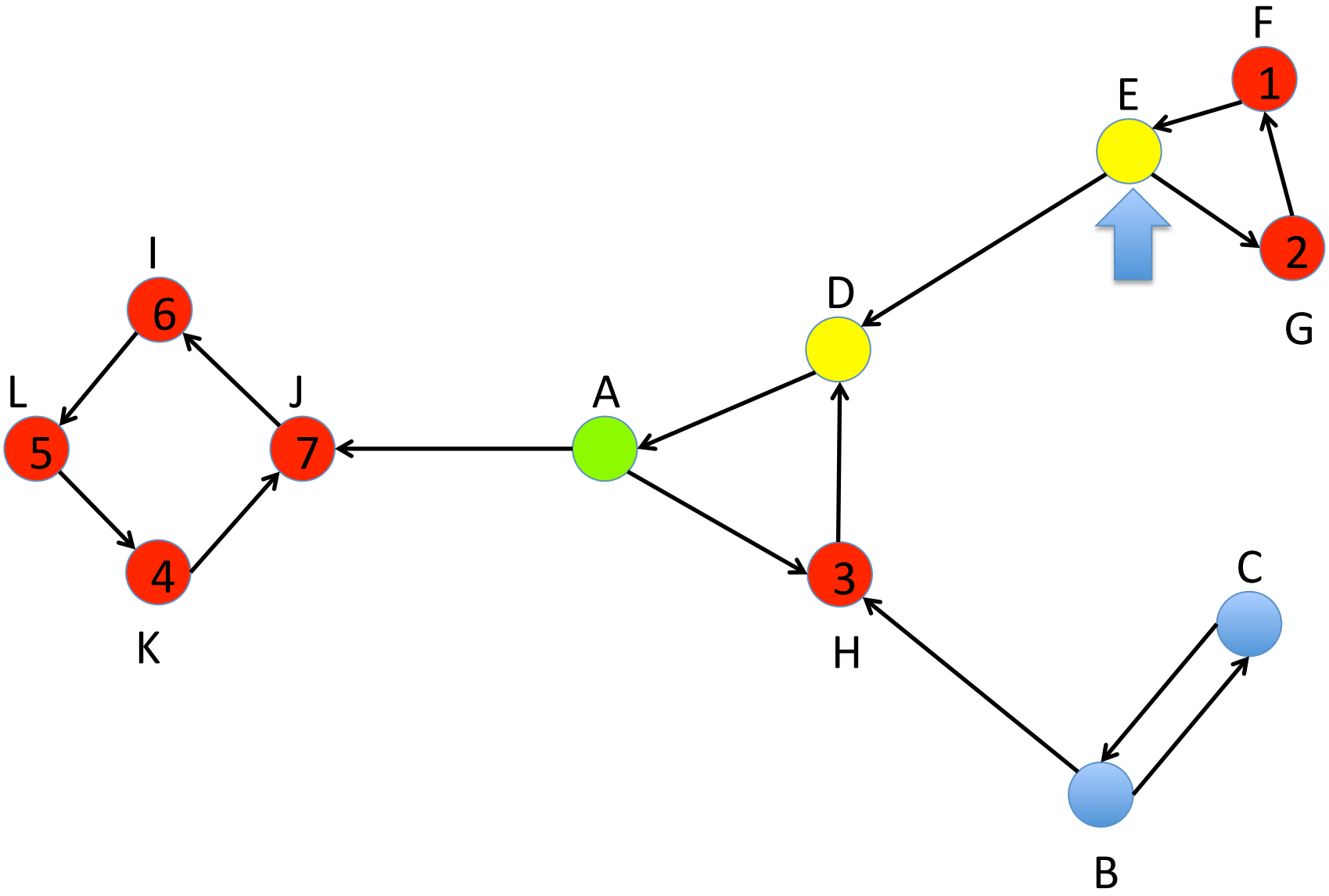
DFS & Finishing Times on Grev



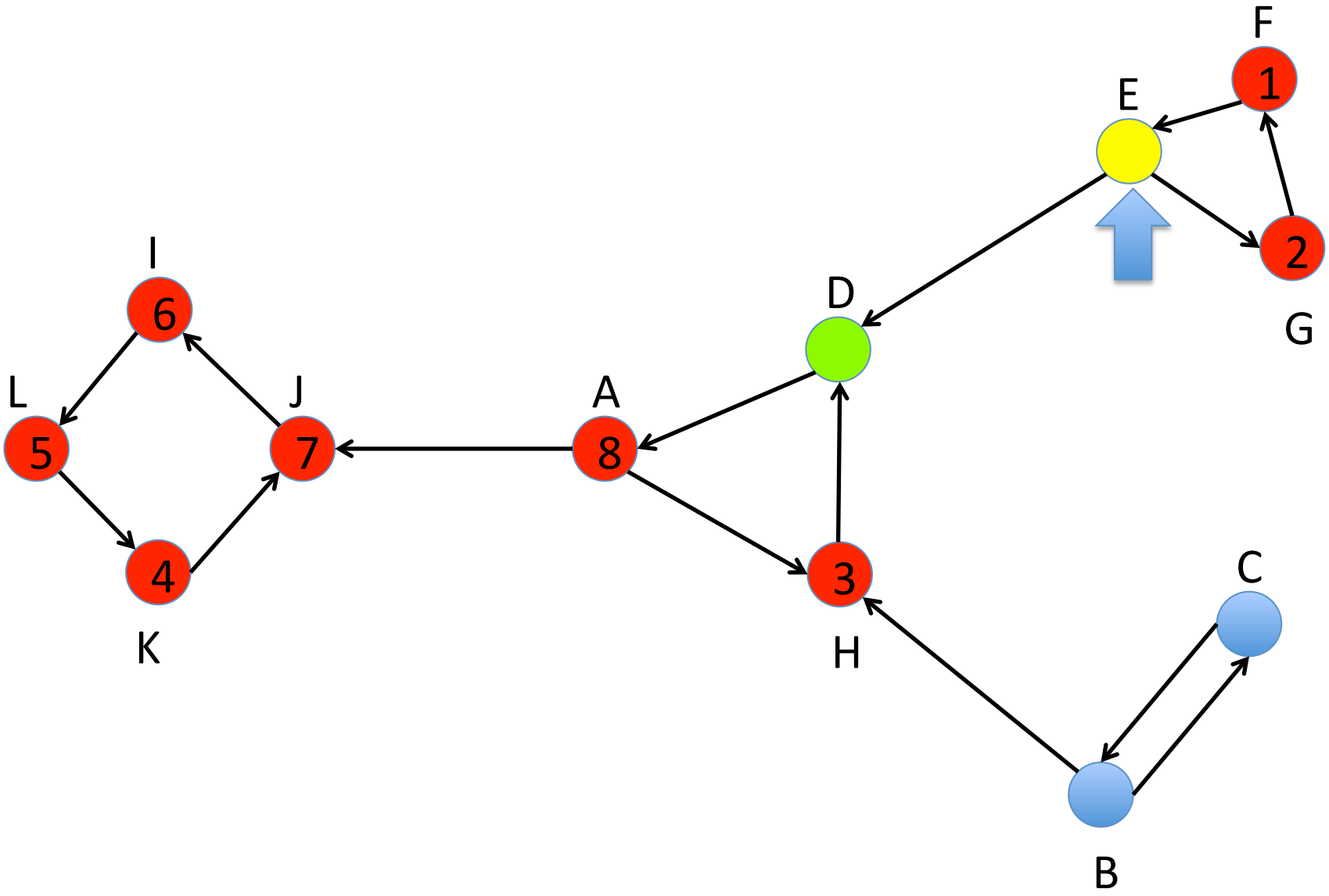
DFS & Finishing Times on Grev



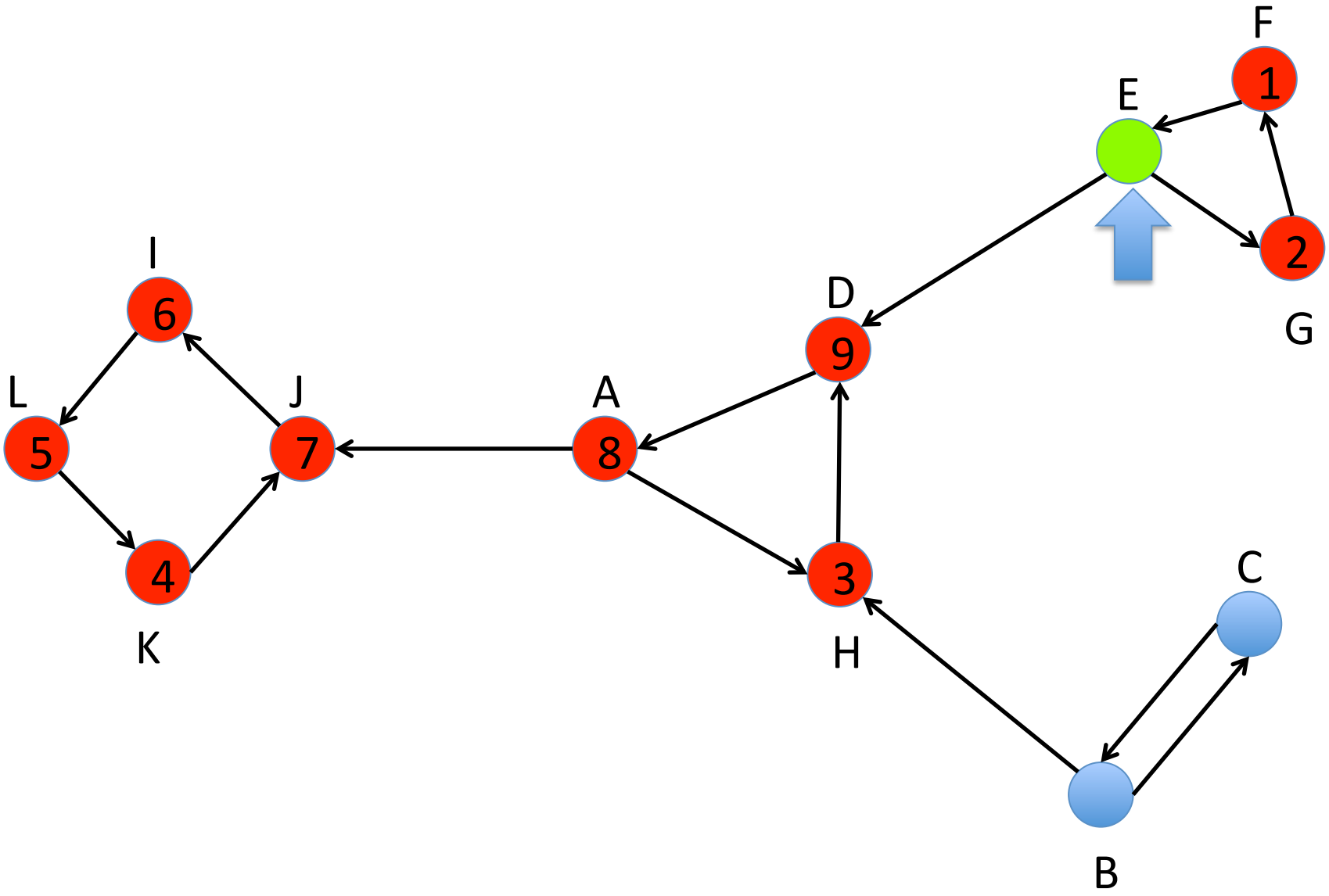
DFS & Finishing Times on Grev



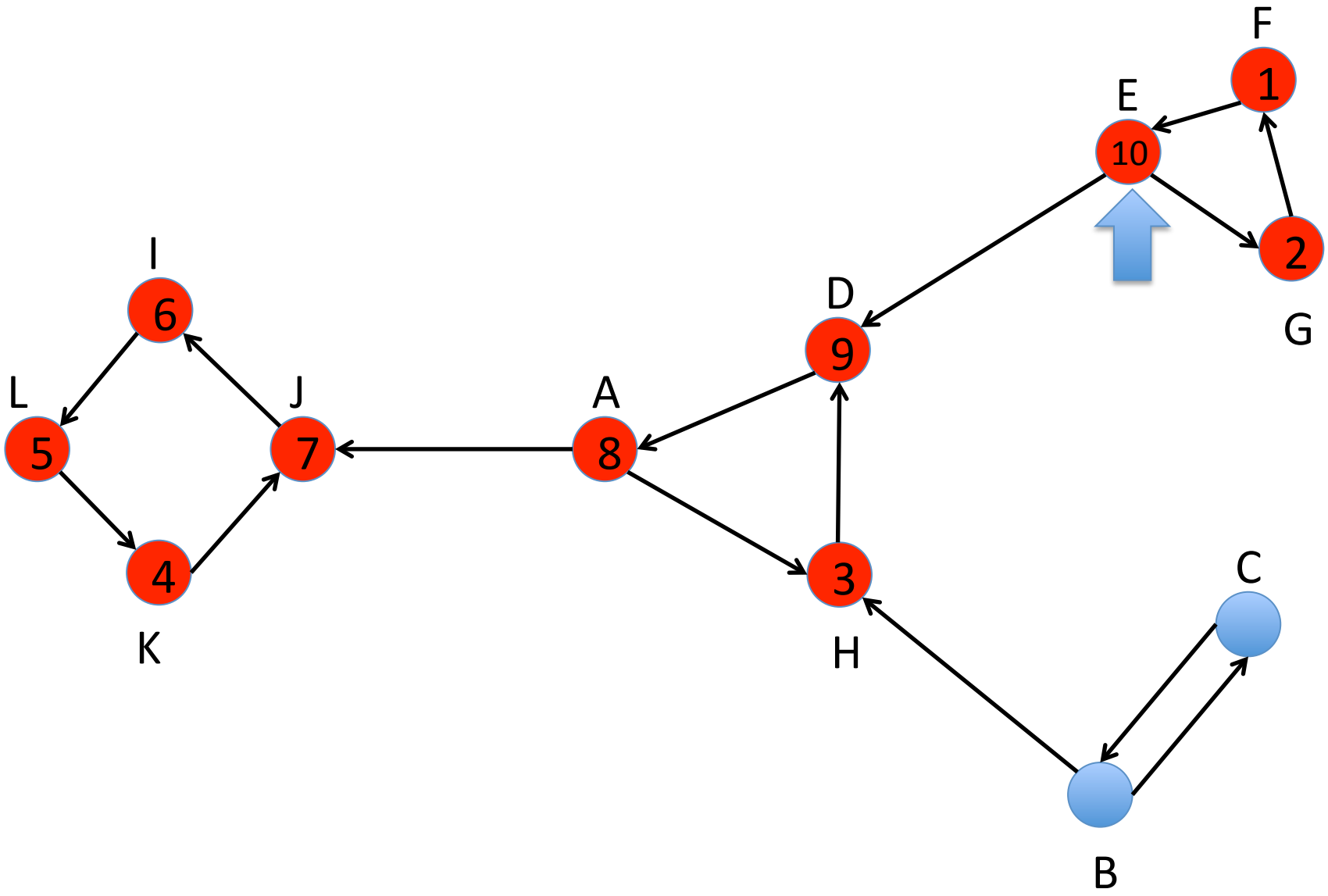
DFS & Finishing Times on Grev



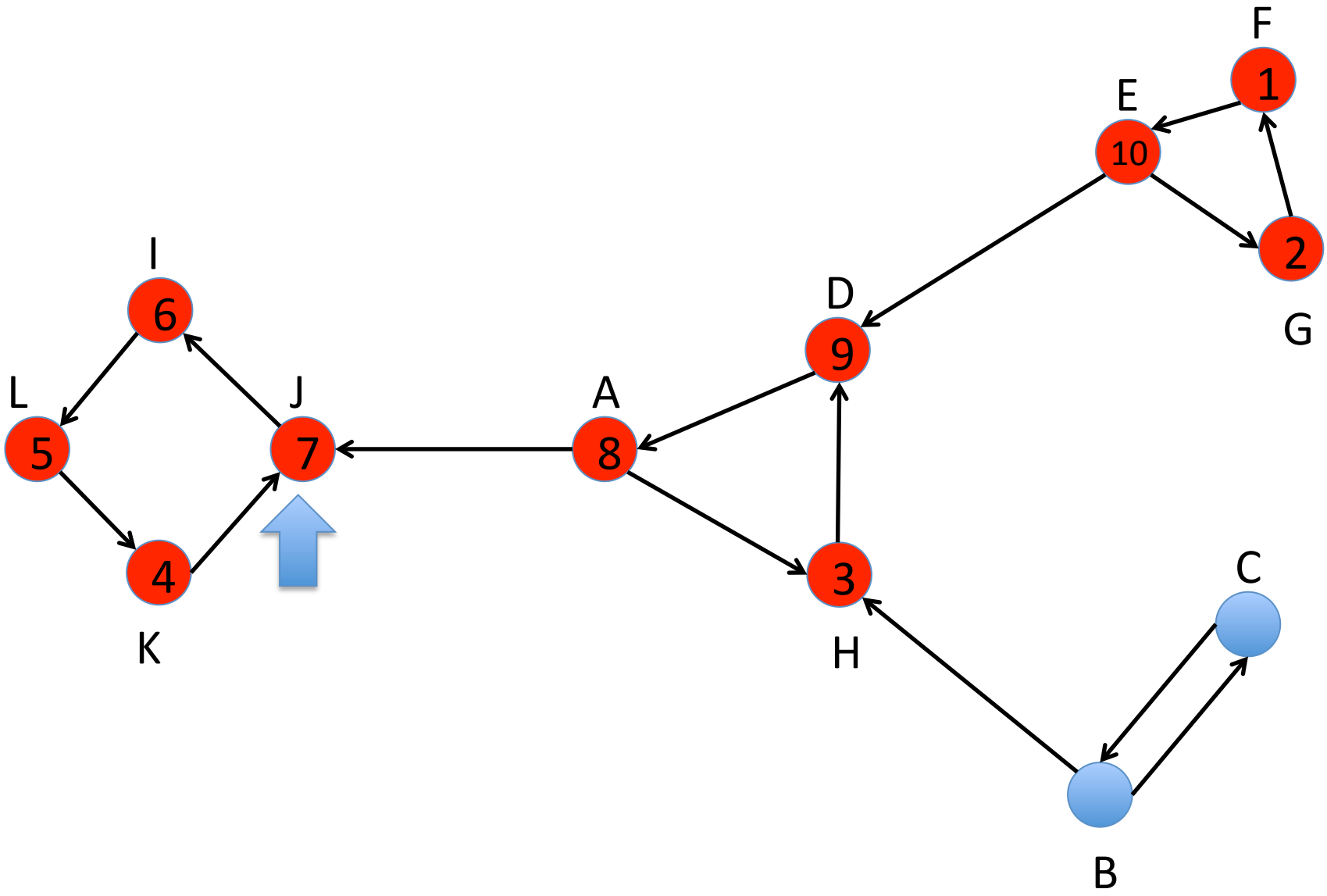
DFS & Finishing Times on Grev



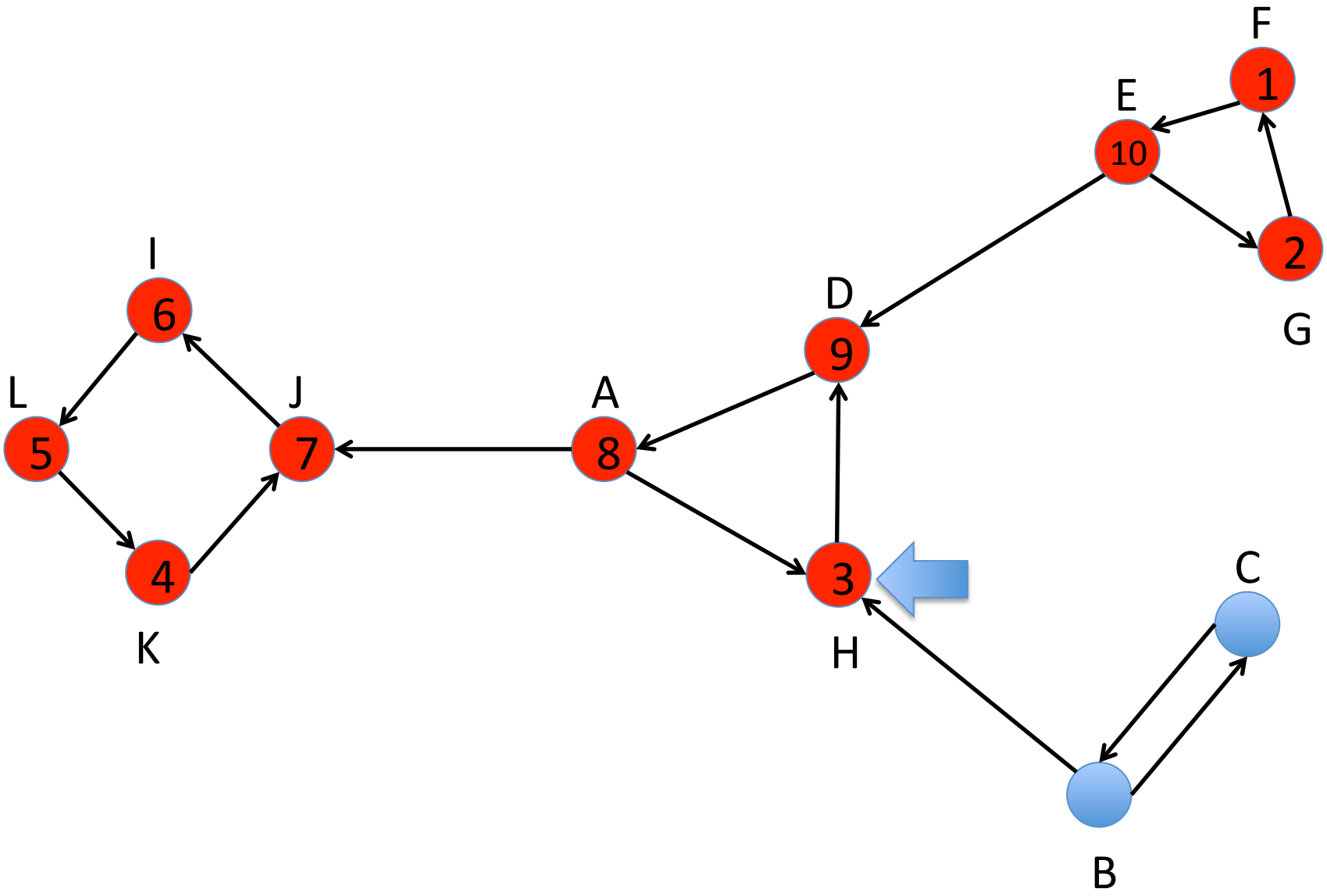
DFS & Finishing Times on Grev



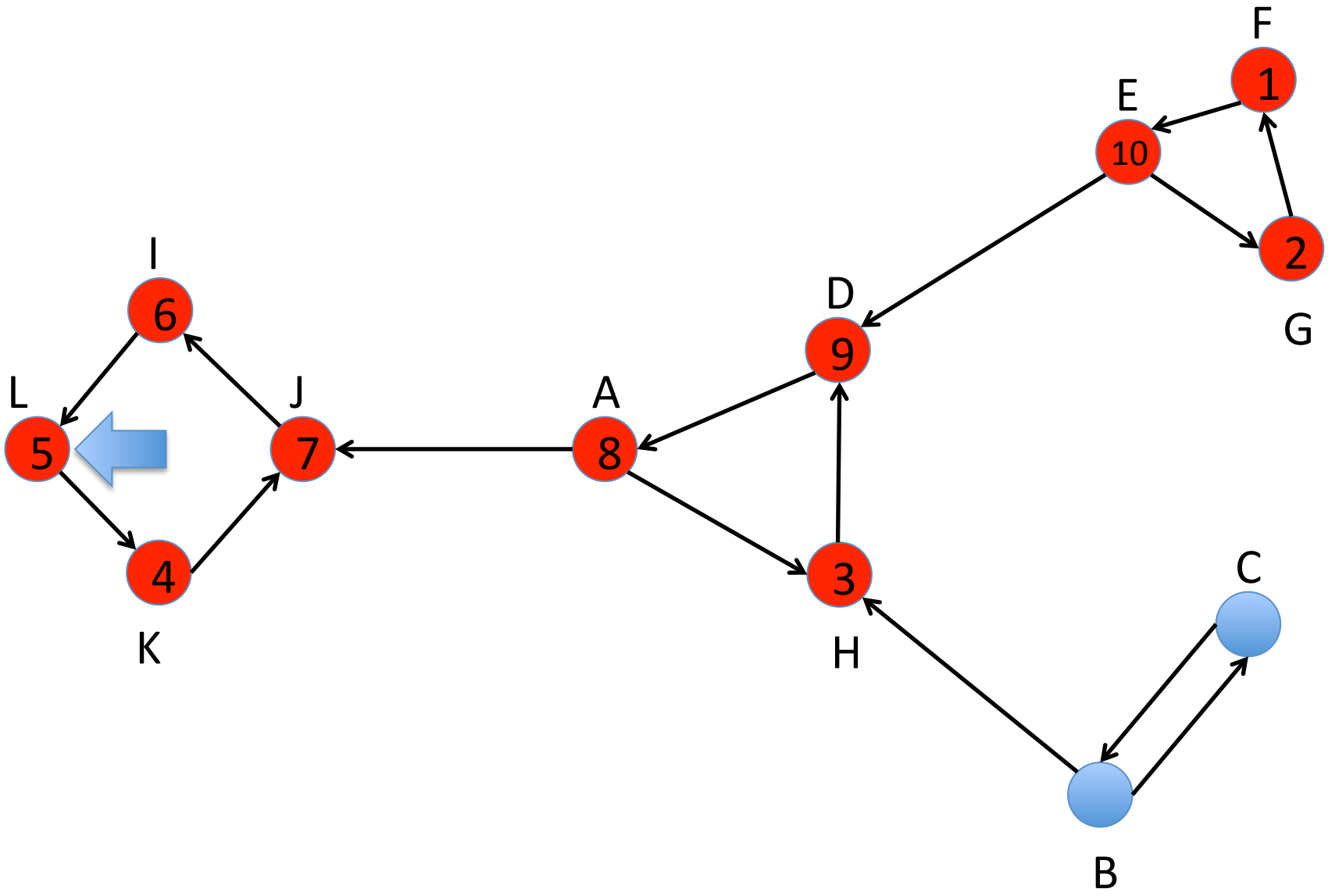
DFS & Finishing Times on Grev



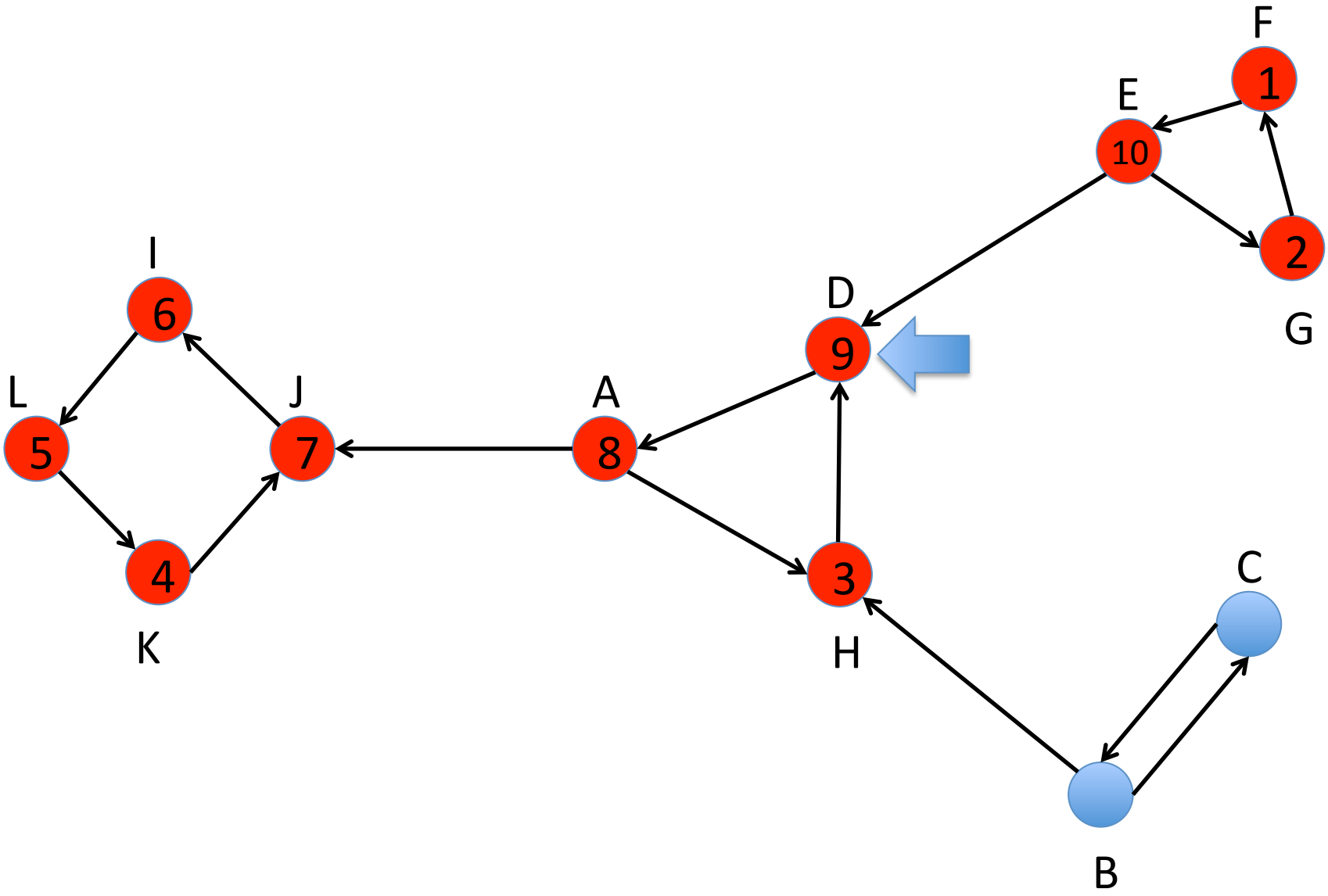
DFS & Finishing Times on Grev



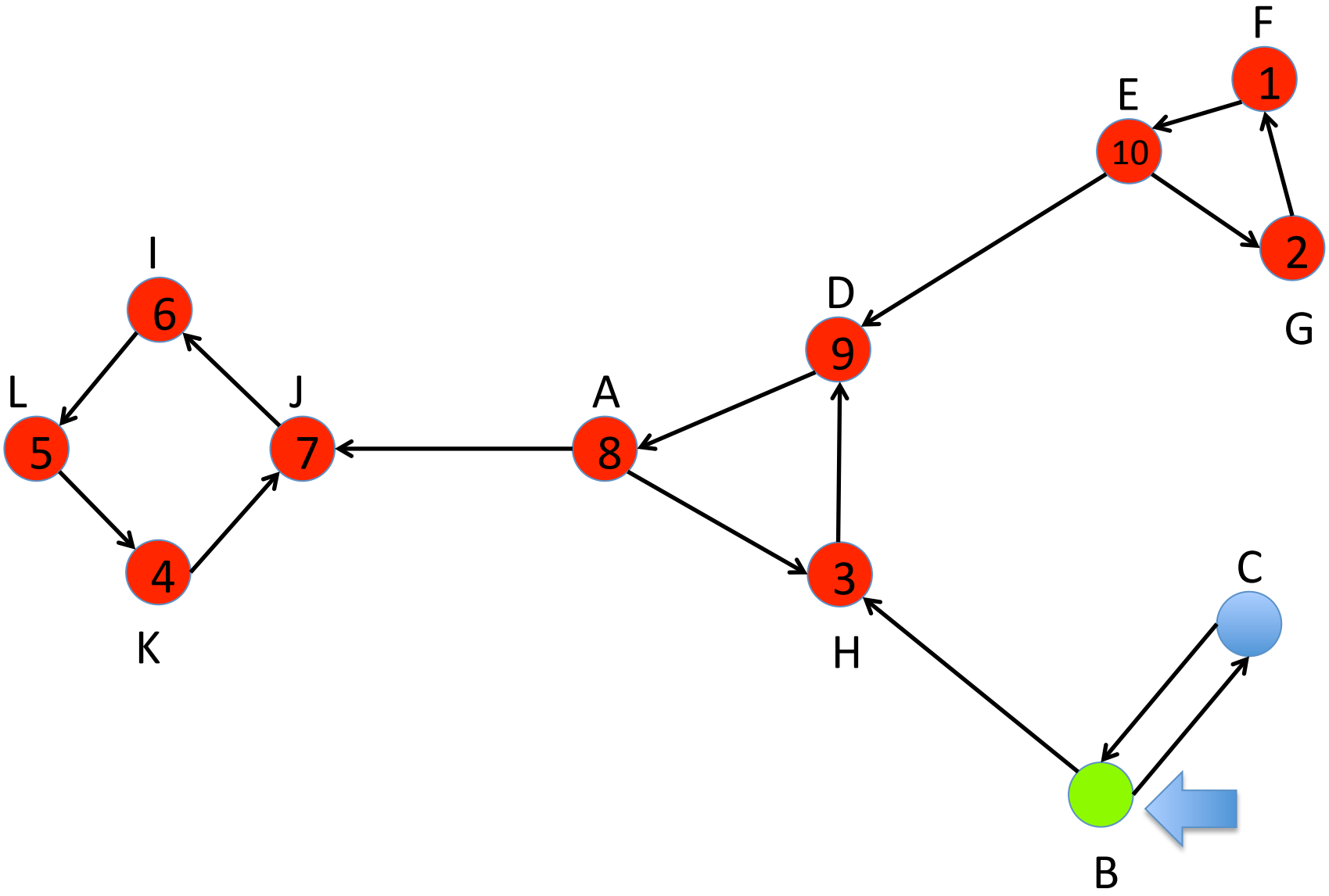
DFS & Finishing Times on Grev



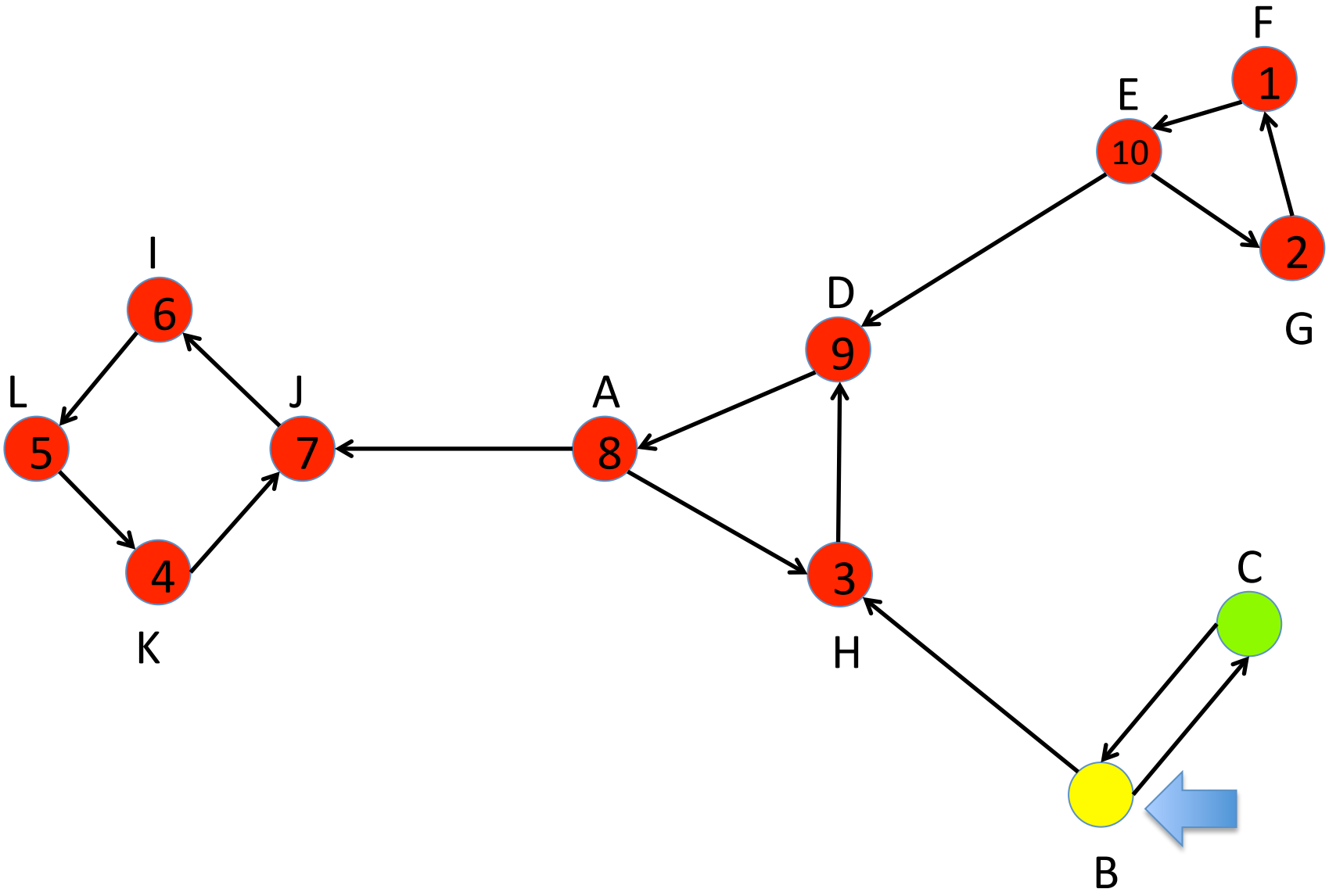
DFS & Finishing Times on Grev



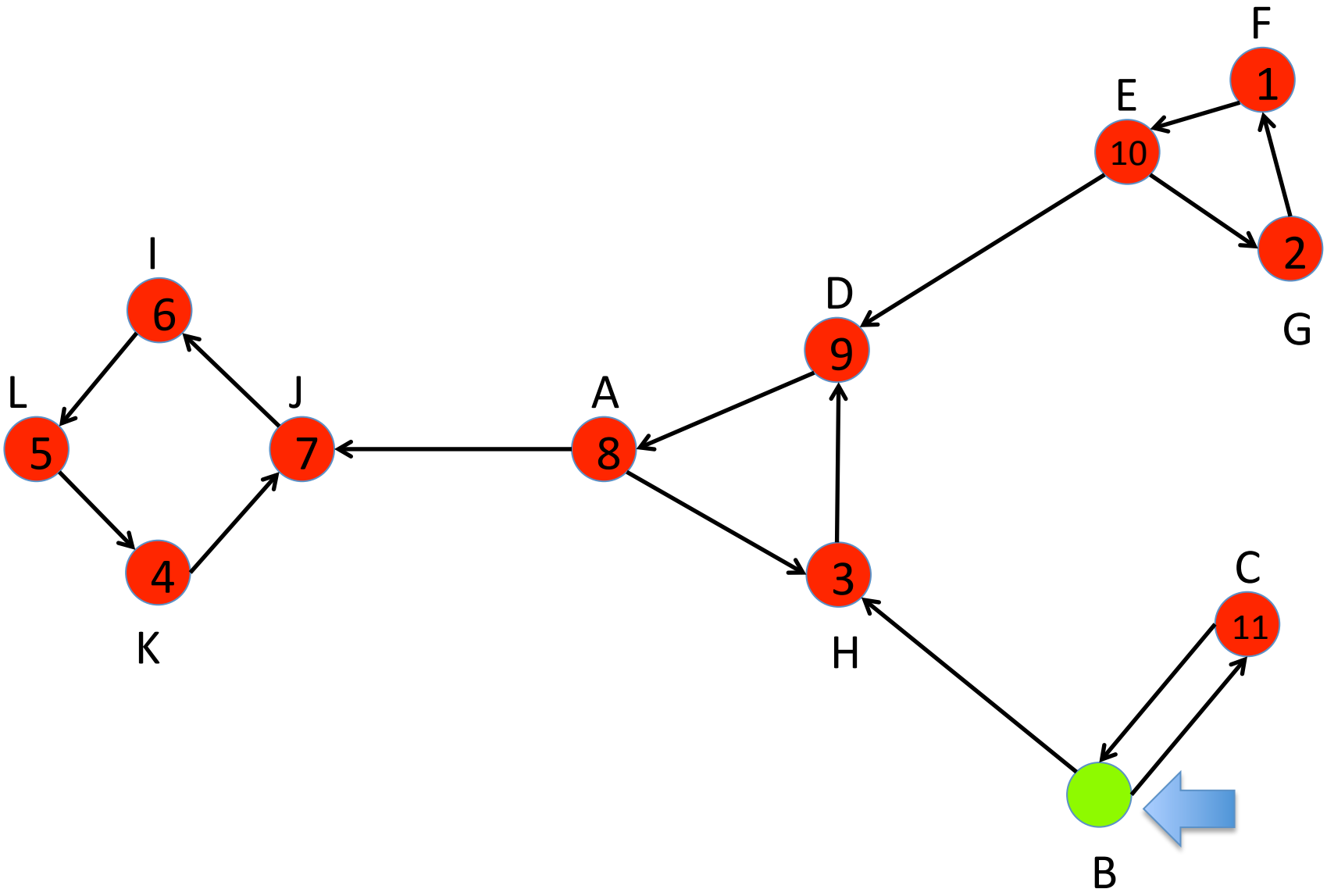
DFS & Finishing Times on Grev



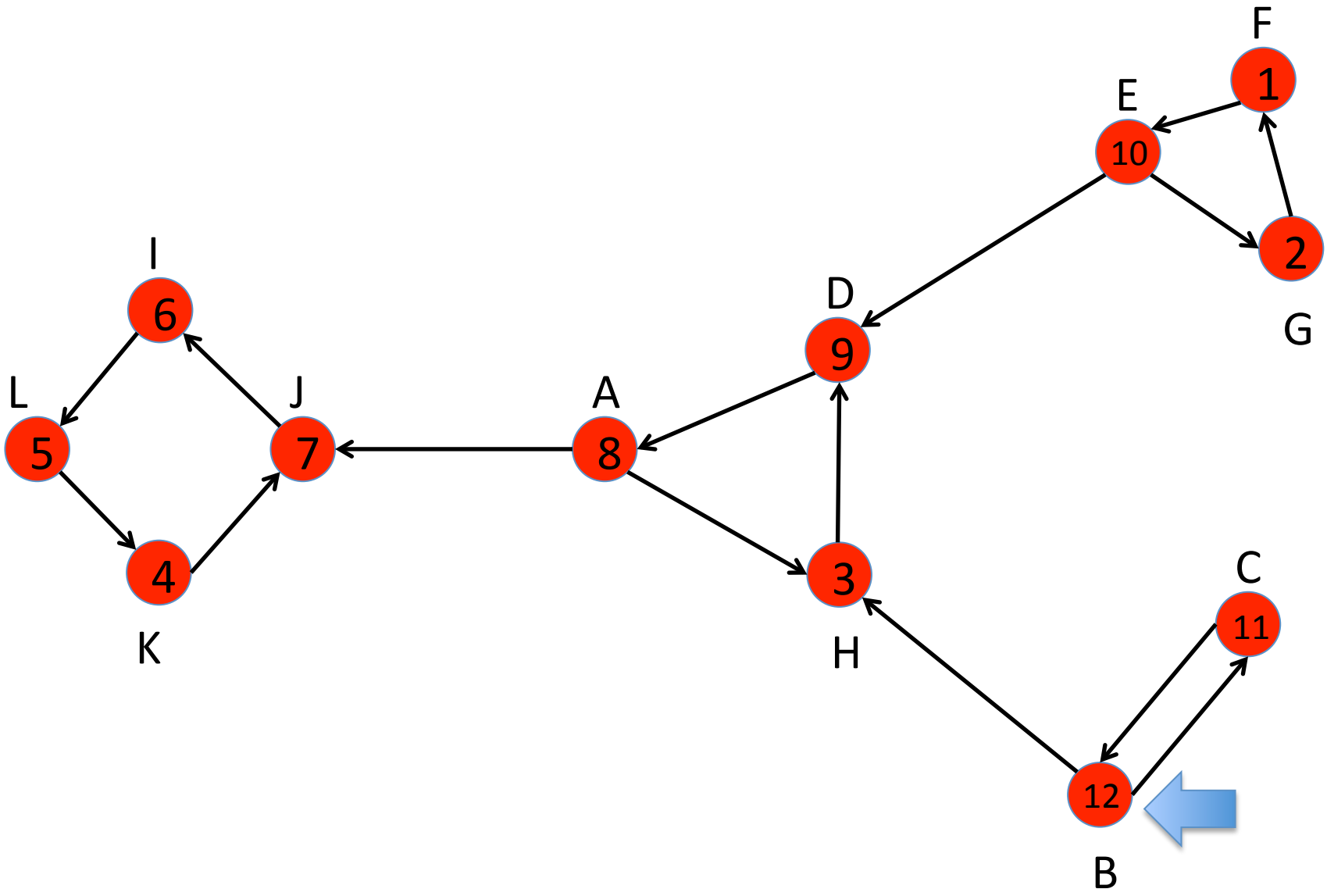
DFS & Finishing Times on Grev



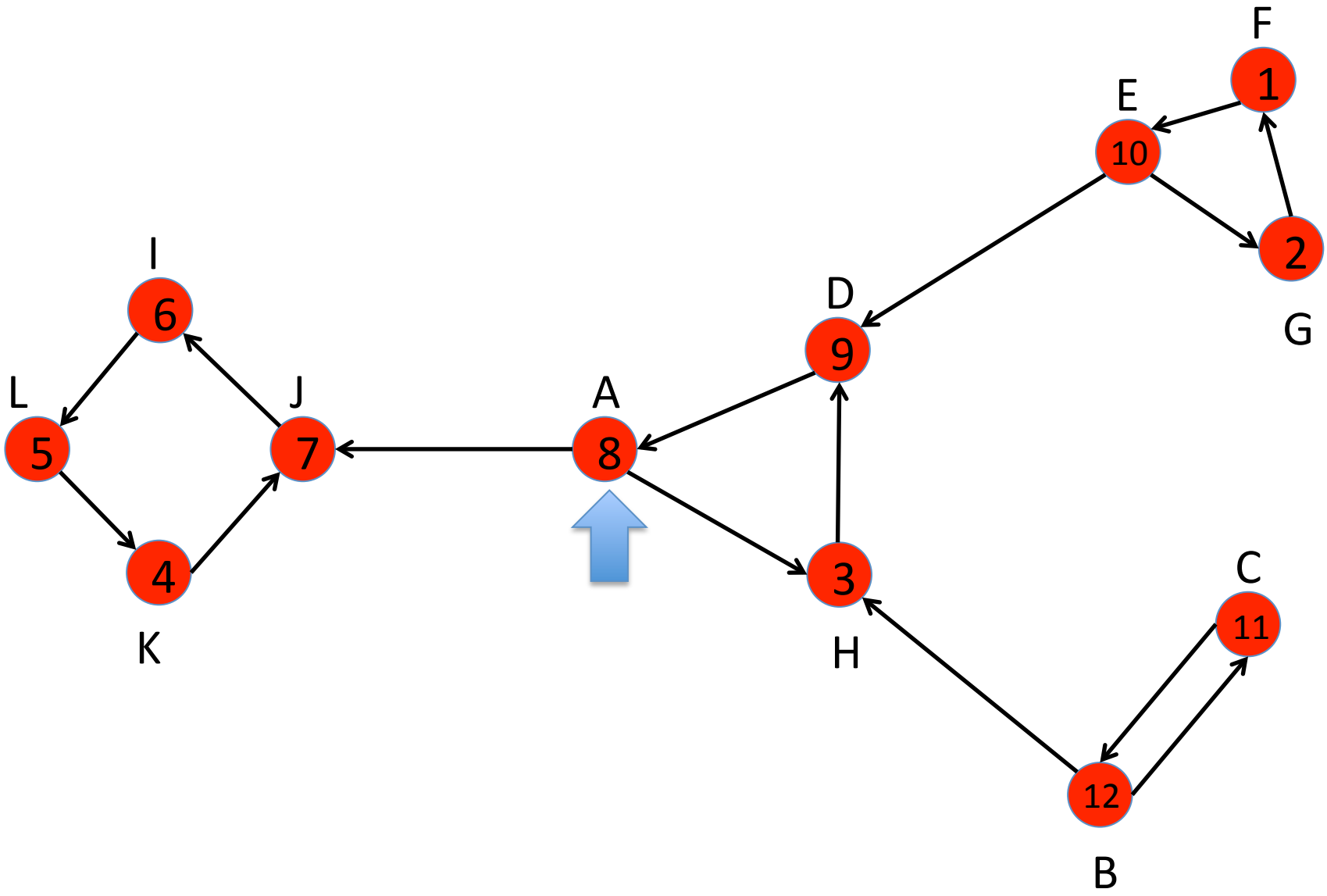
DFS & Finishing Times on Grev



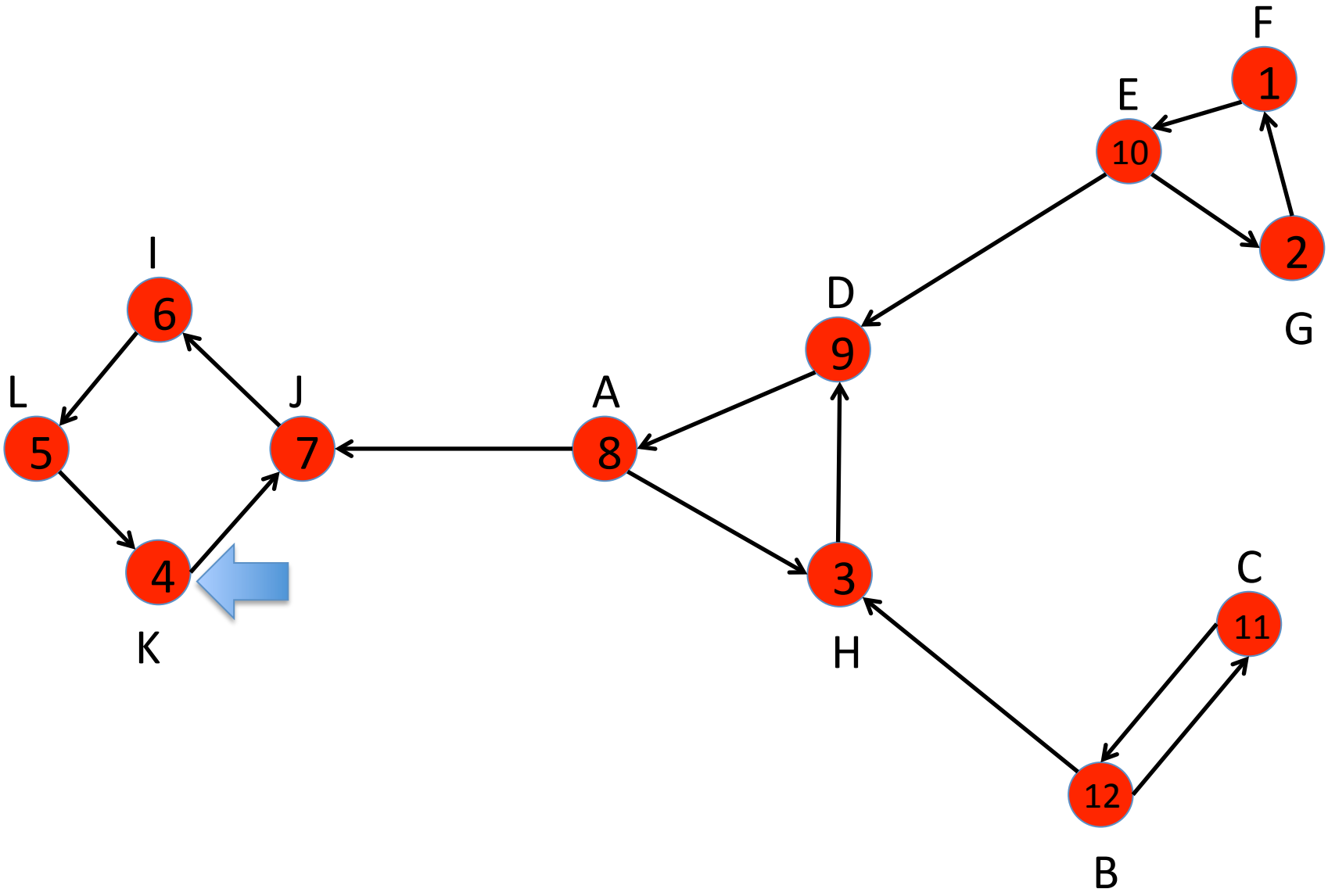
DFS & Finishing Times on Grev



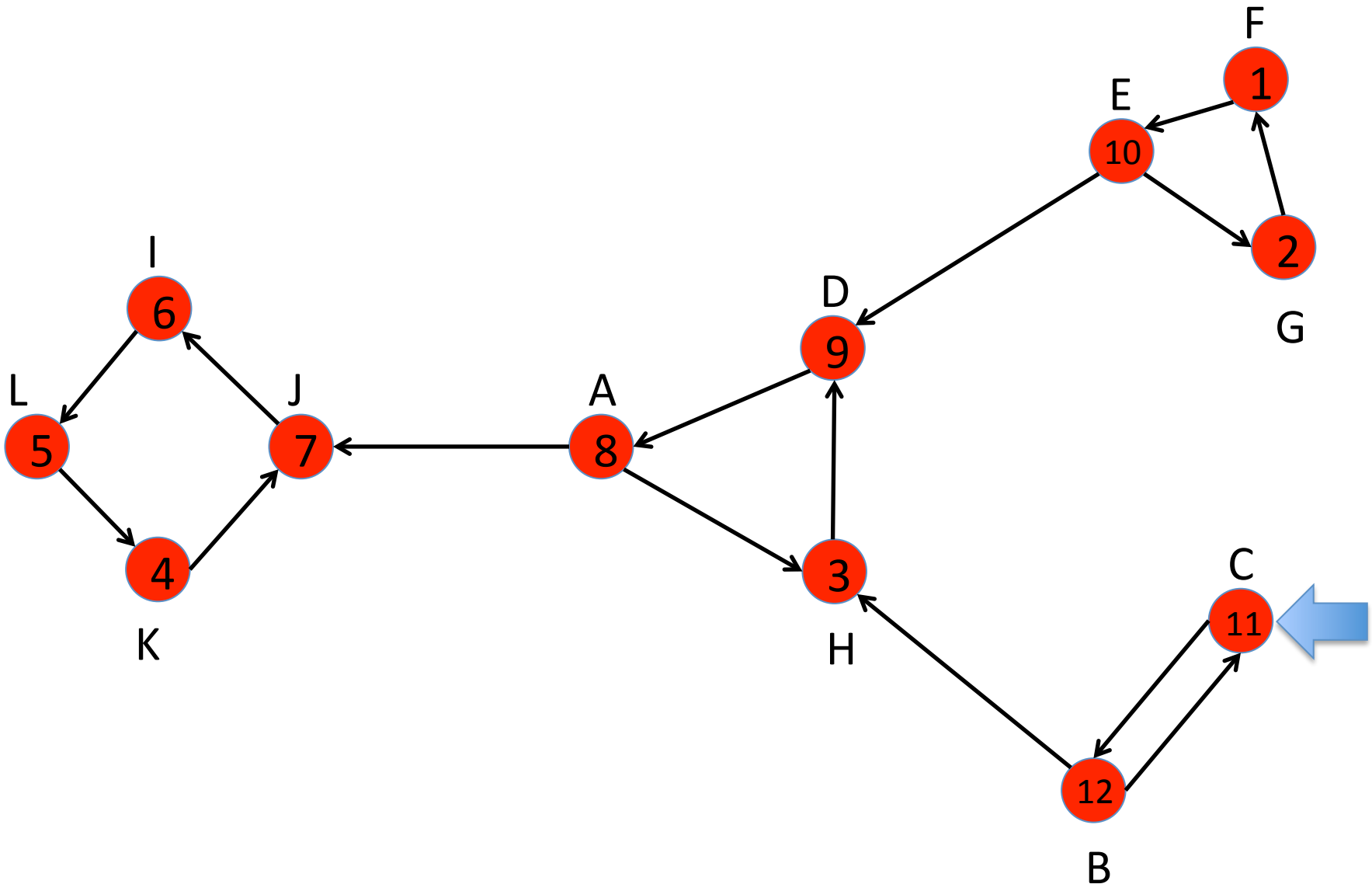
DFS & Finishing Times on Grev



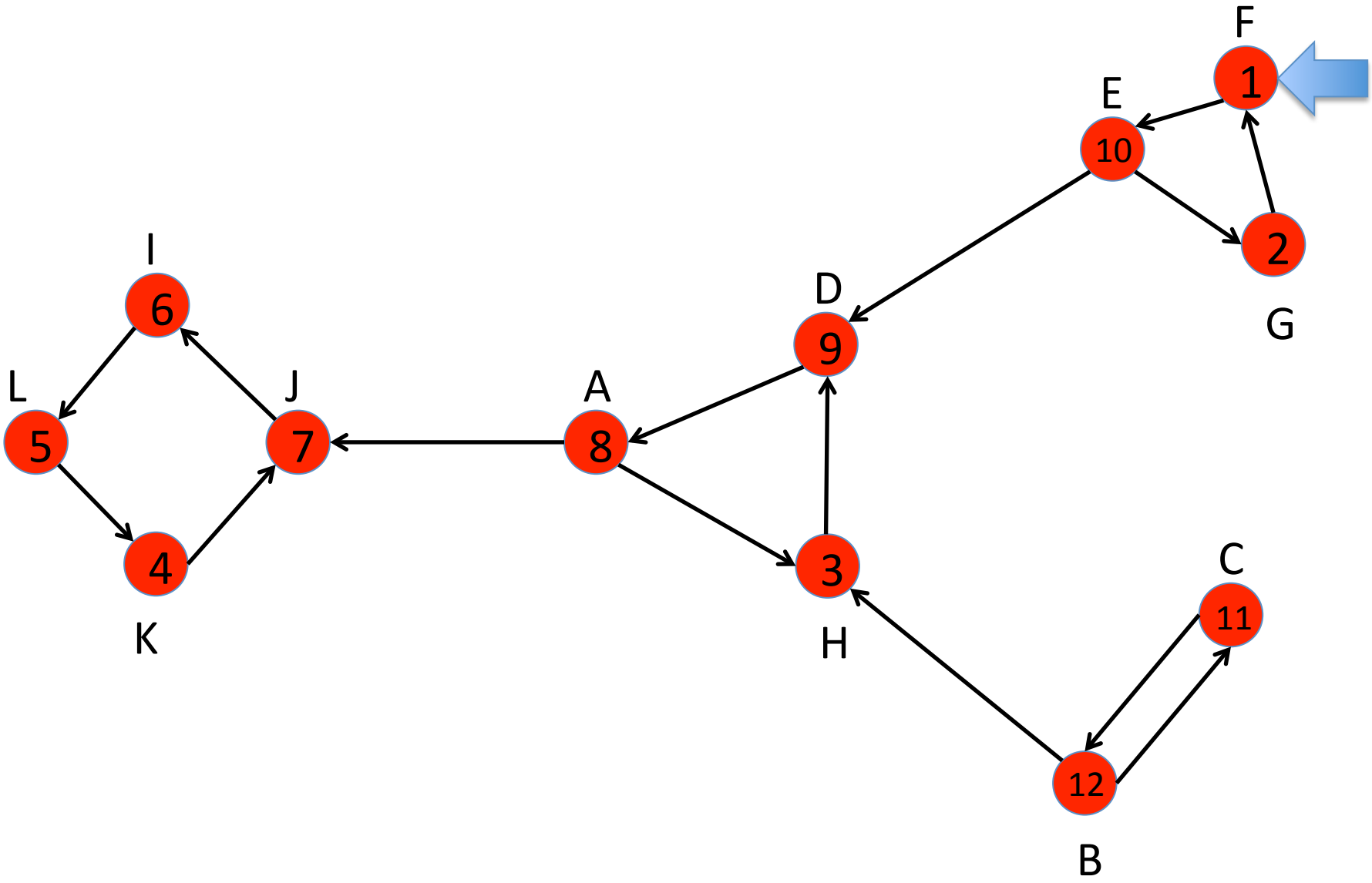
DFS & Finishing Times on Grev



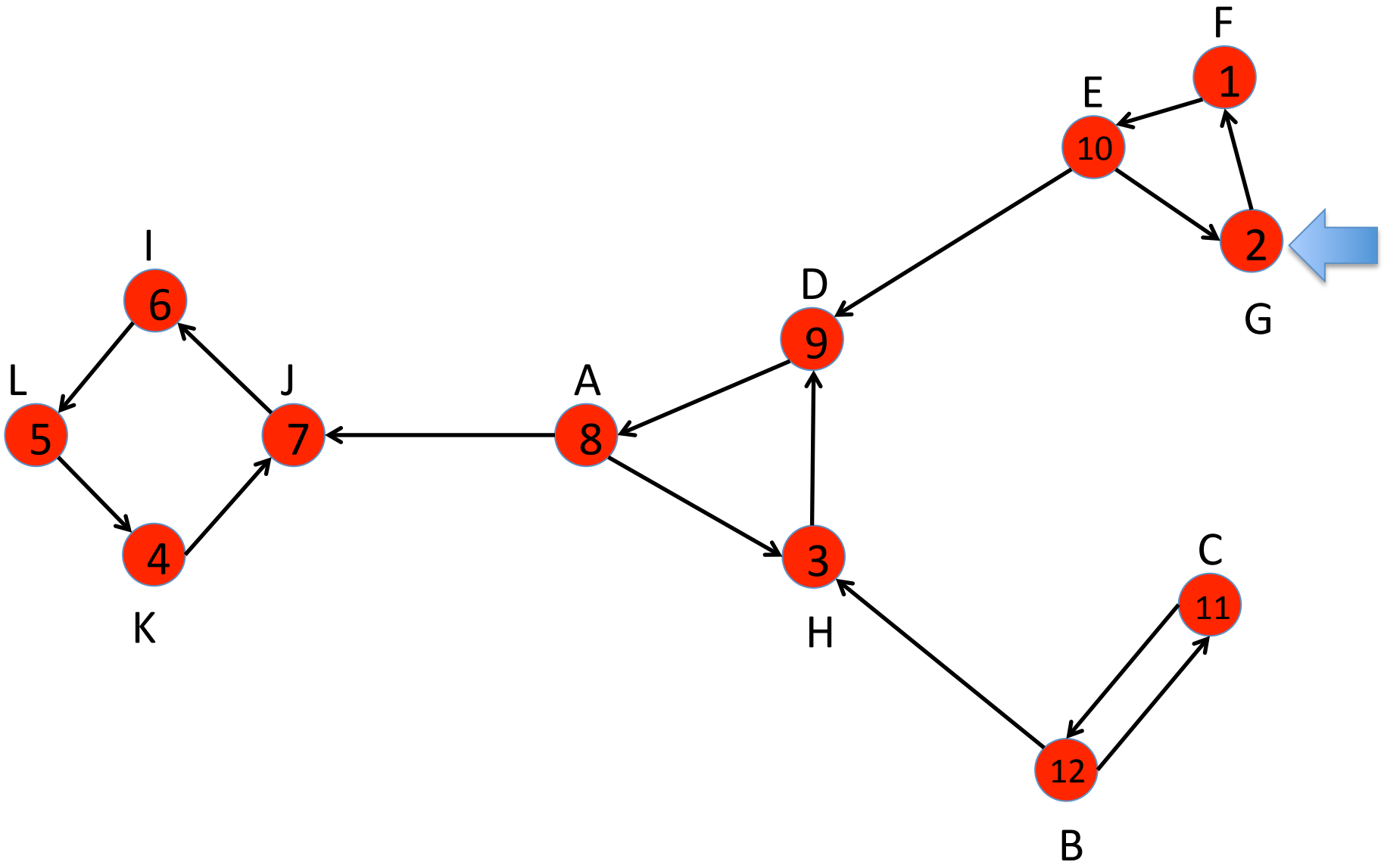
DFS & Finishing Times on Grev



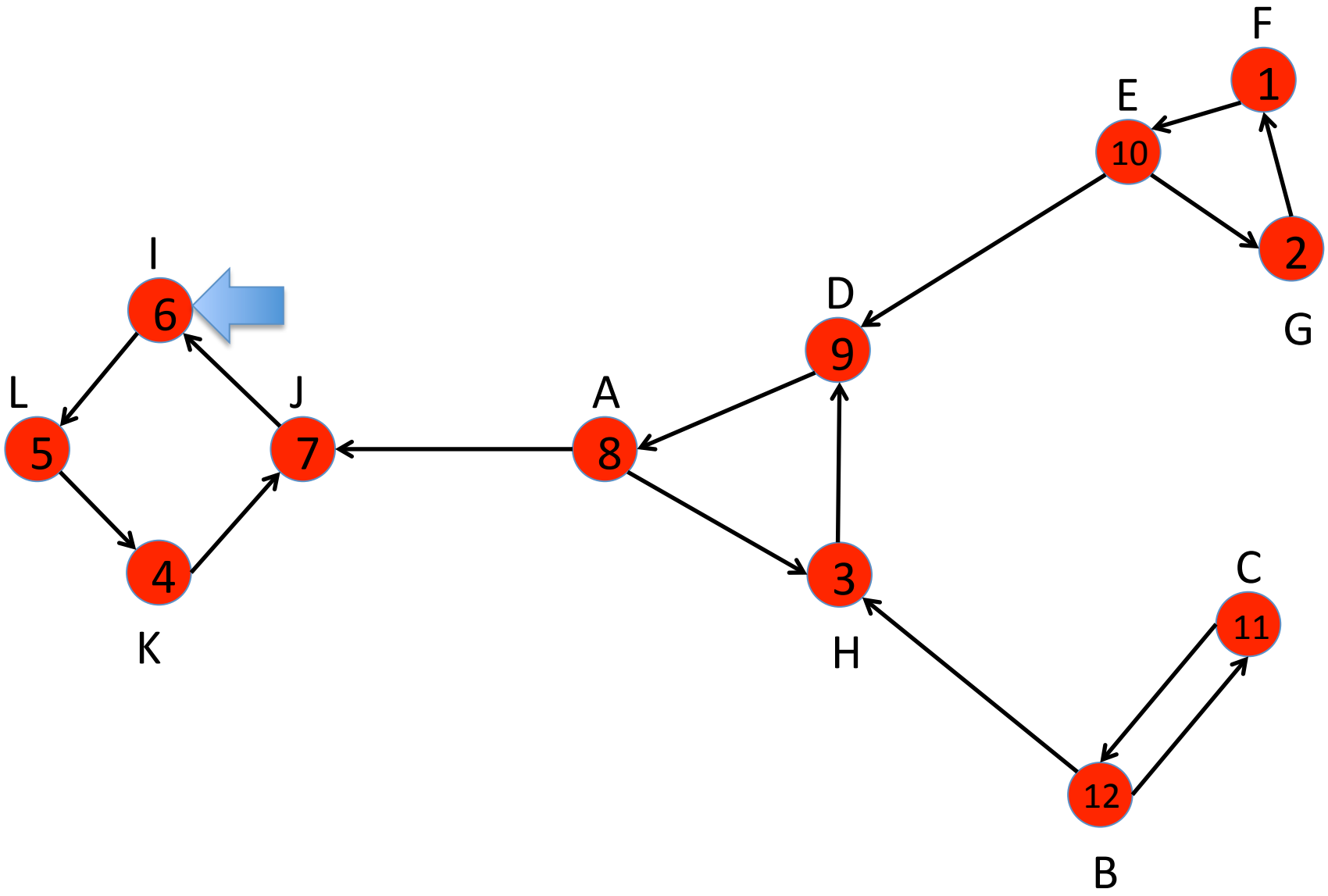
DFS & Finishing Times on Grev



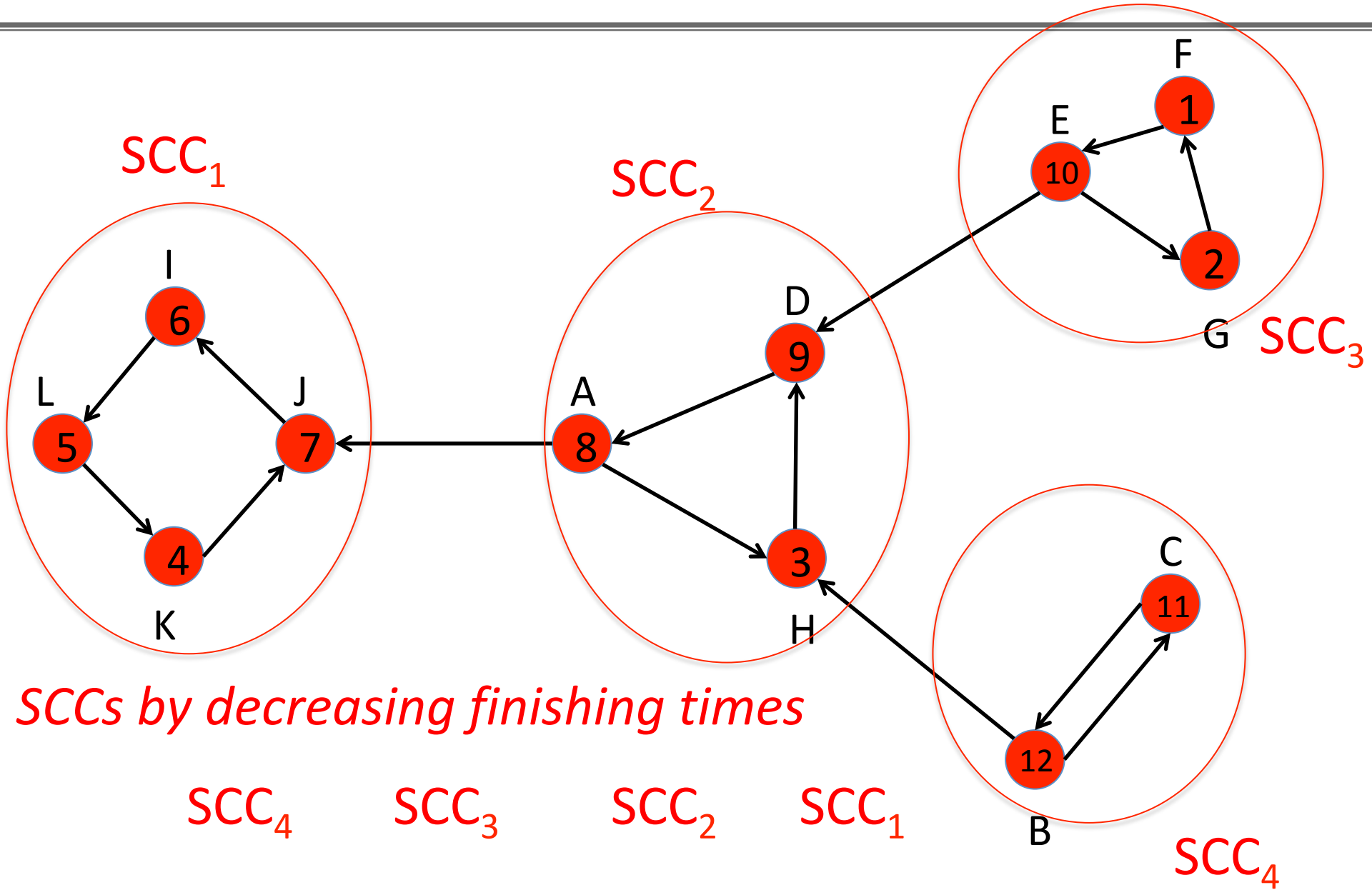
DFS & Finishing Times on Grev



DFS & Finishing Times on Grev



DFS & Finishing Times on Grev



SCCs by decreasing finishing times

SCC_4 SCC_3 SCC_2 SCC_1

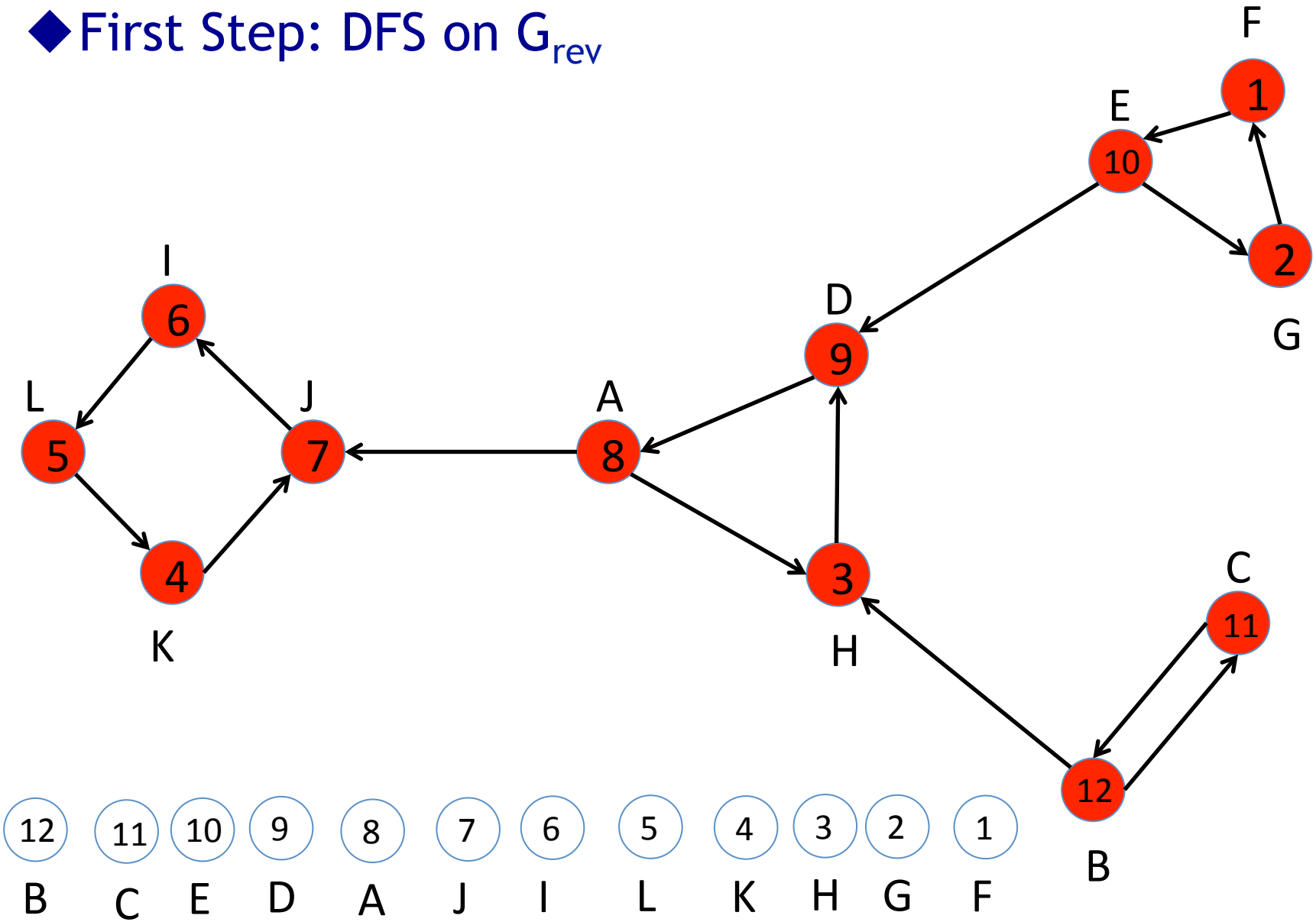
Finally: Kosaraju's Algorithm

```
procedure kosarajusSCC(DAG G):
    1. run DFS on G_rev and keep finishing times f
    2. run DFS/BFS in G:
        pick start vertices by decreasing f times
        propagate the ID of each new start vertex
        during traversal
```

Again, the labels will implicitly identify the SCCs!

Kosaraju's Algorithm Simulation

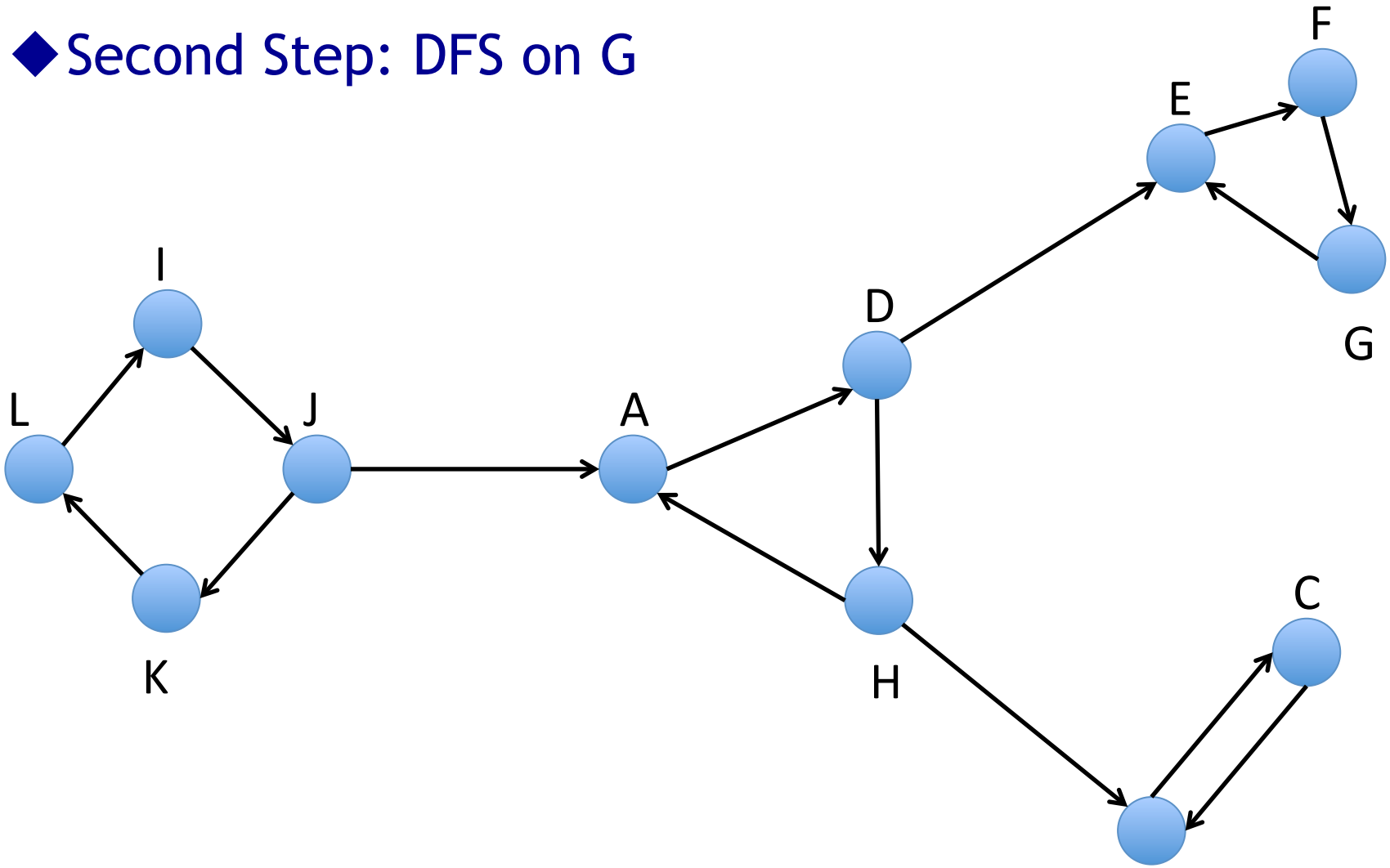
- First Step: DFS on G_{rev}



12	11	10	9	8	7	6	5	4	3	2	1
B	C	E	D	A	J	I	L	K	H	G	F

Kosaraju's Algorithm Simulation

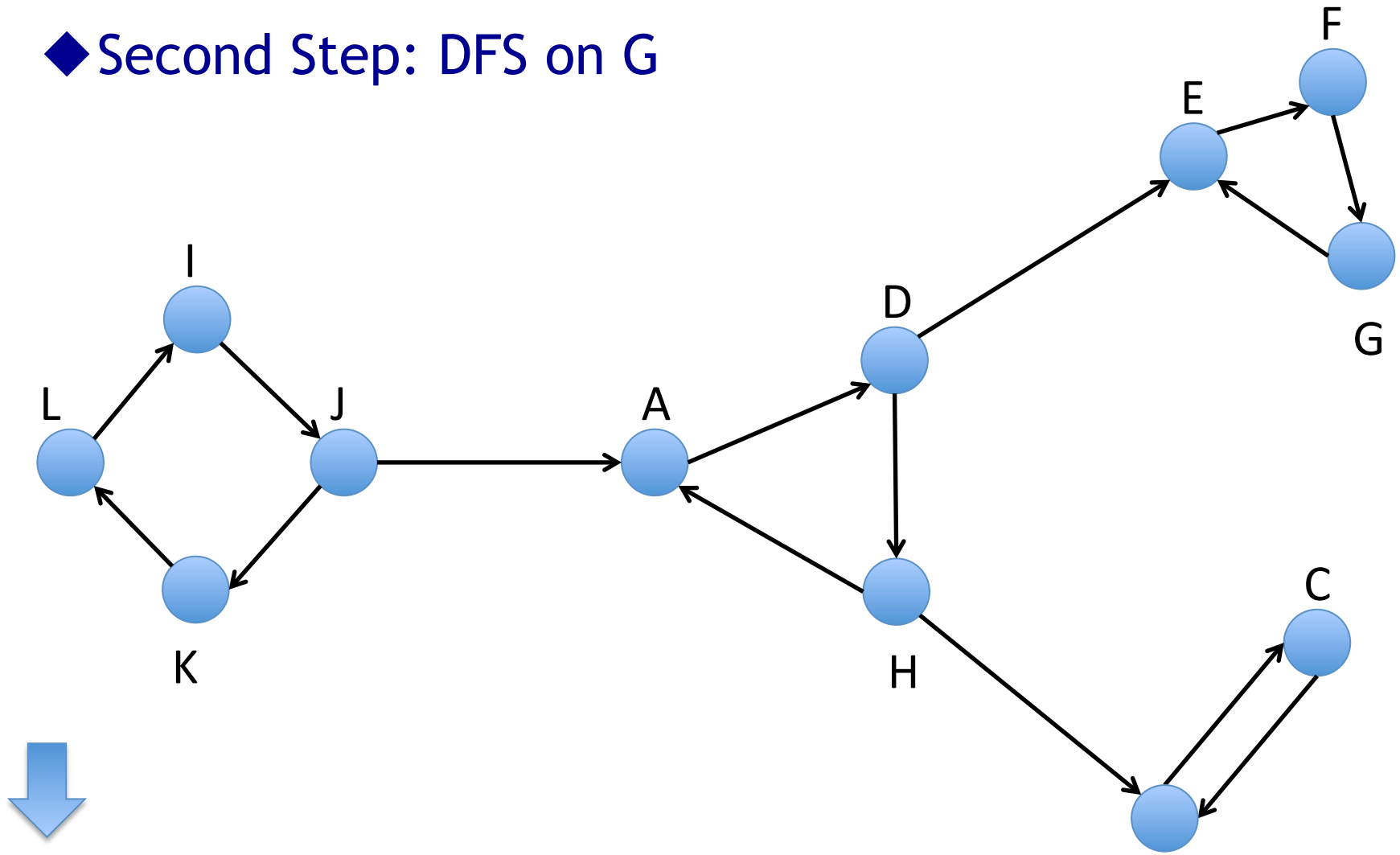
- Second Step: DFS on G



12	11	10	9	8	7	6	5	4	3	2	1
B	C	E	D	A	J	I	L	K	H	G	F

Kosaraju's Algorithm Simulation

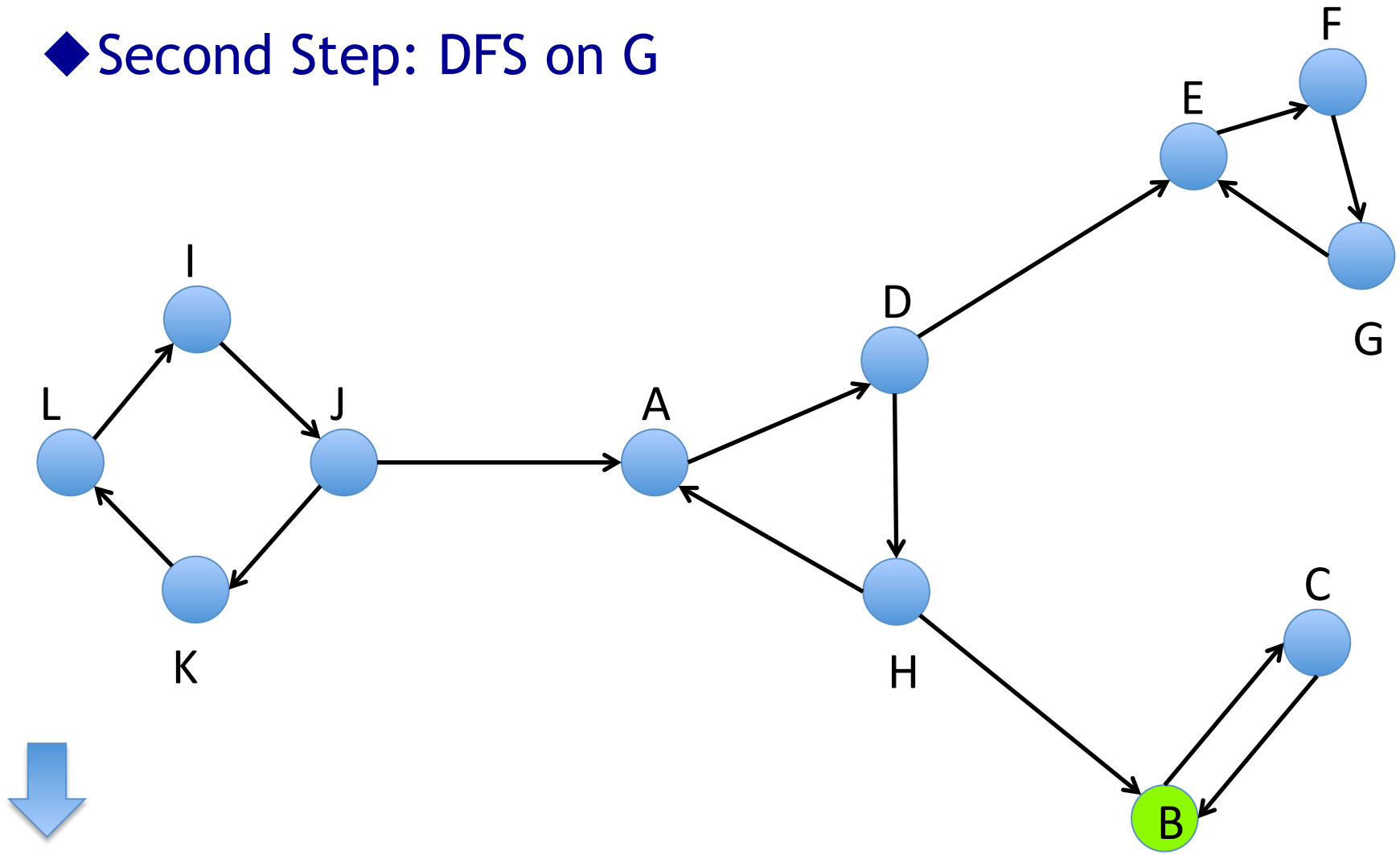
- Second Step: DFS on G



12	11	10	9	8	7	6	5	4	3	2	1
B	C	E	D	A	J	I	L	K	H	G	F

Kosaraju's Algorithm Simulation

- Second Step: DFS on G

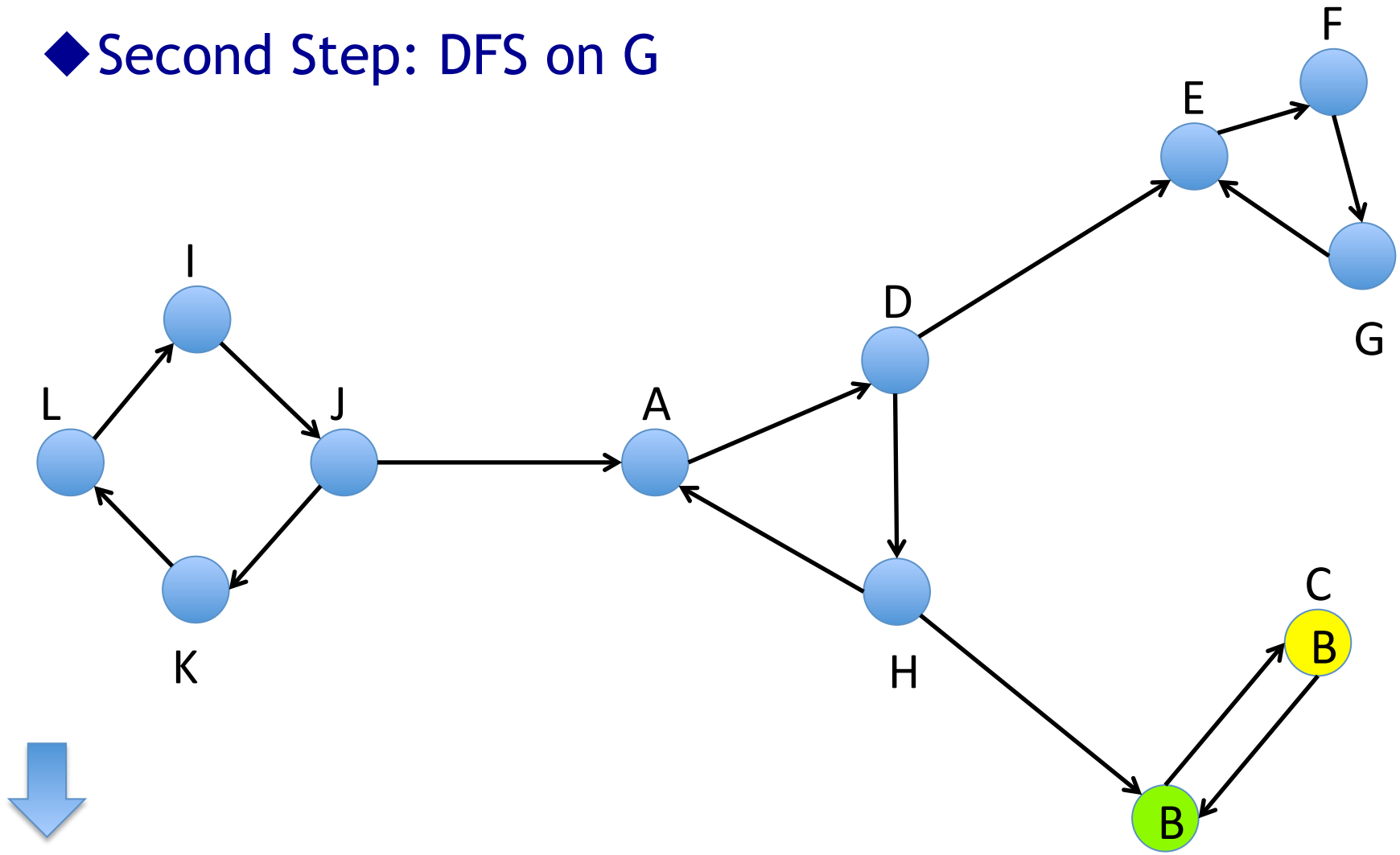


12	11	10	9	8	7	6	5	4	3	2	1
B	C	E	D	A	J	I	L	K	H	G	F

B

Kosaraju's Algorithm Simulation

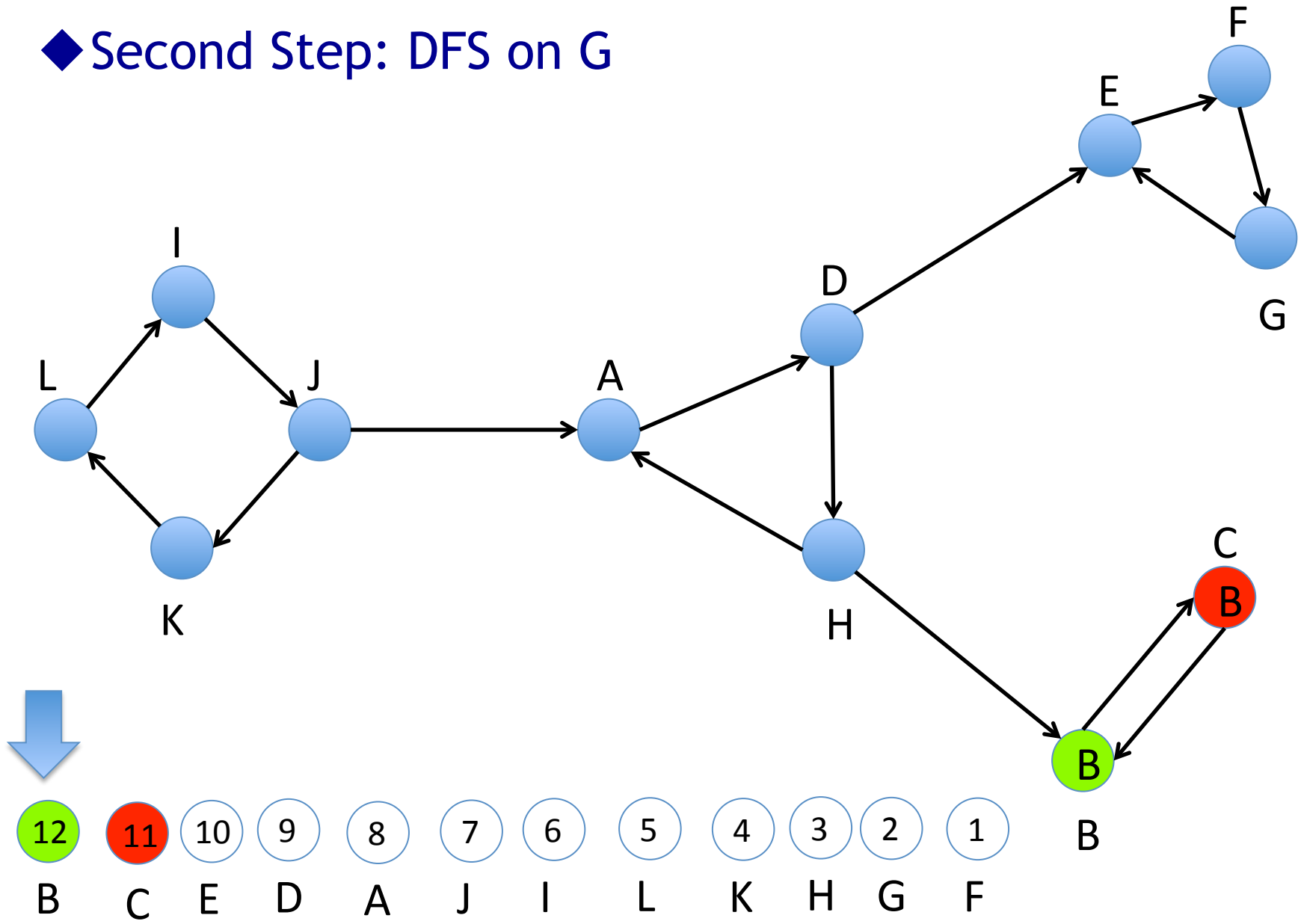
- Second Step: DFS on G



12	11	10	9	8	7	6	5	4	3	2	1
B	C	E	D	A	J	I	L	K	H	G	F

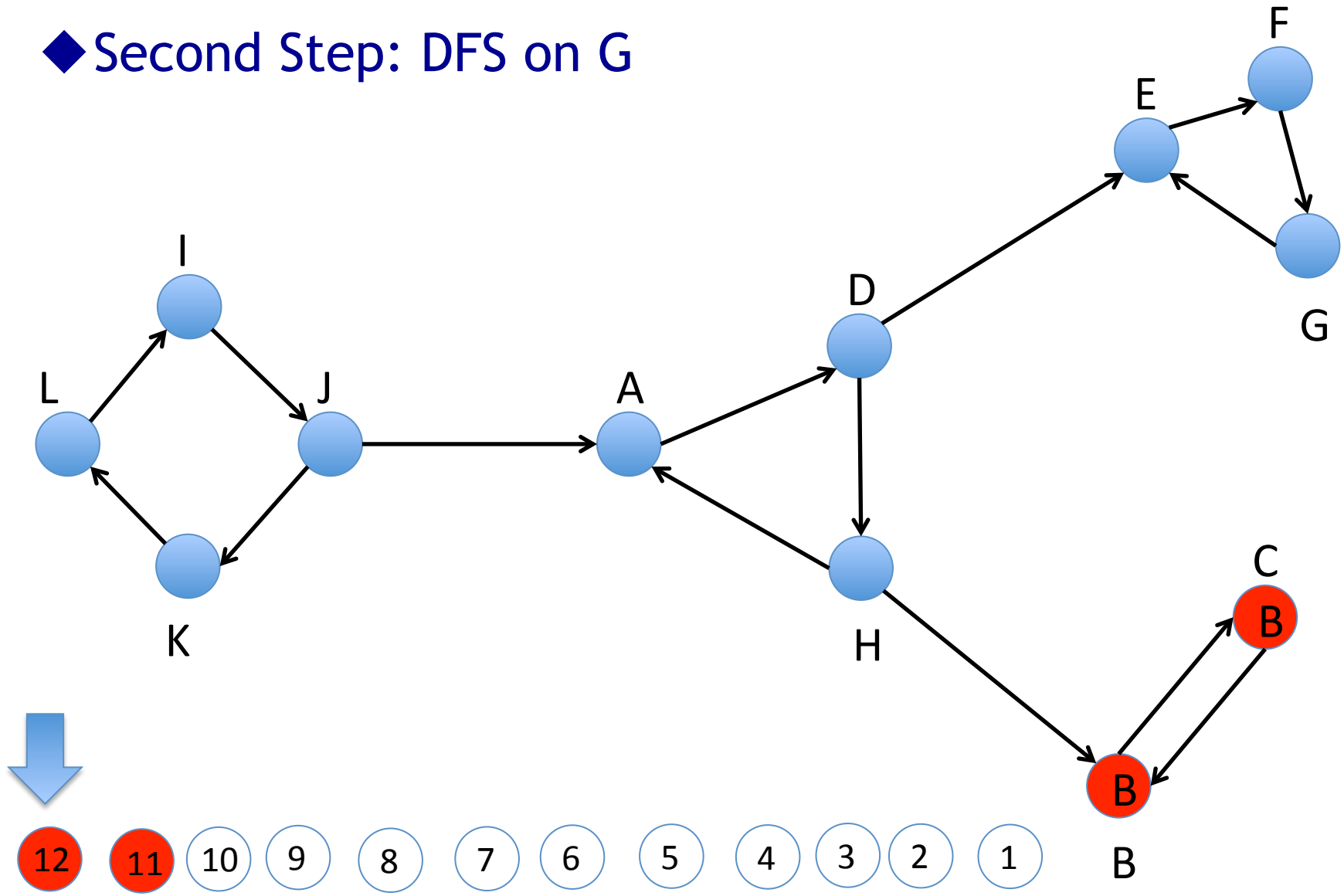
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



Kosaraju's Algorithm Simulation

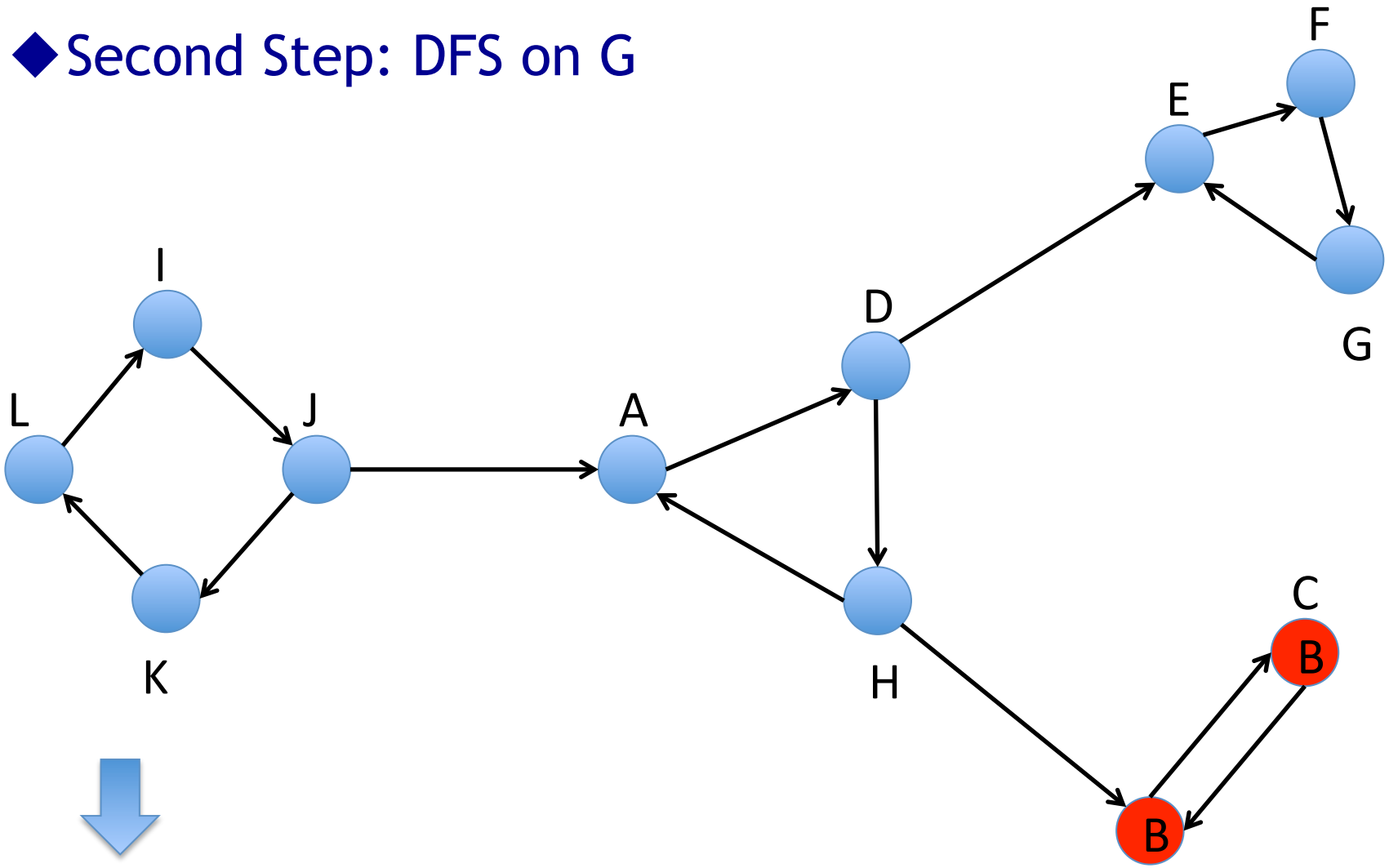
- Second Step: DFS on G



12	11	10	9	8	7	6	5	4	3	2	1
B	C	E	D	A	J	I	L	K	H	G	F

Kosaraju's Algorithm Simulation

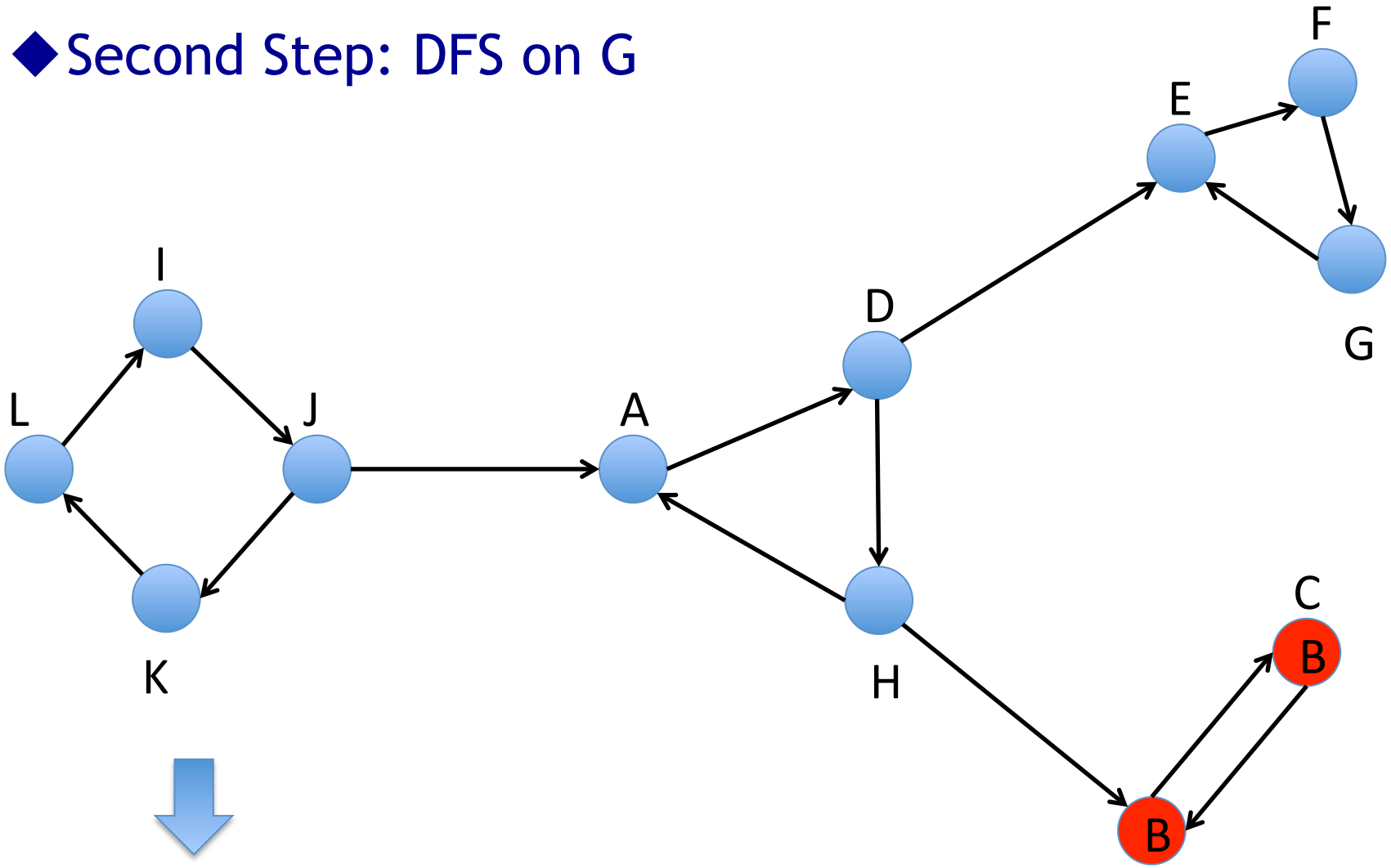
- Second Step: DFS on G



12	11	10	9	8	7	6	5	4	3	2	1
B	C	E	D	A	J	I	L	K	H	G	F

Kosaraju's Algorithm Simulation

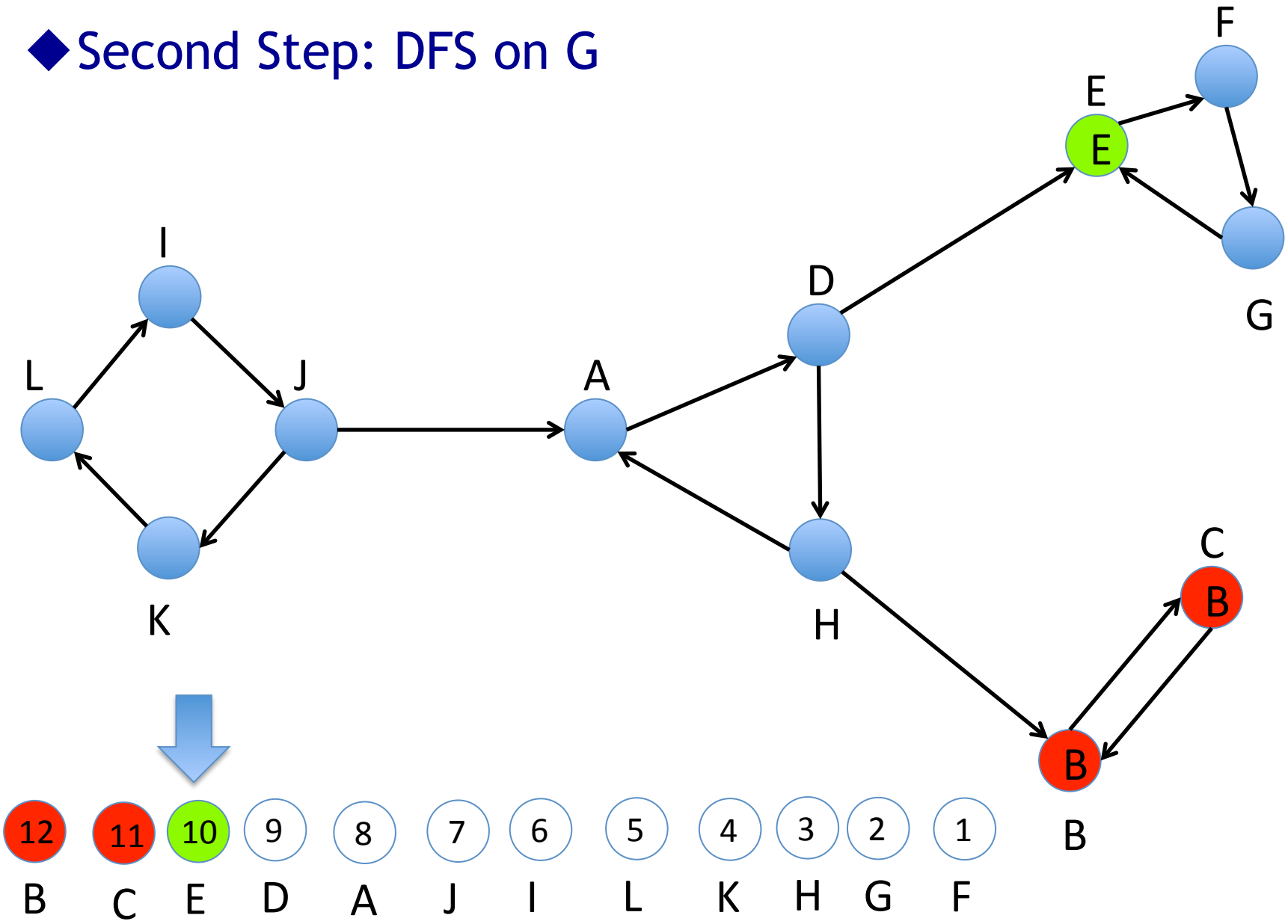
- Second Step: DFS on G



12	11	10	9	8	7	6	5	4	3	2	1
B	C	E	D	A	J	I	L	K	H	G	F

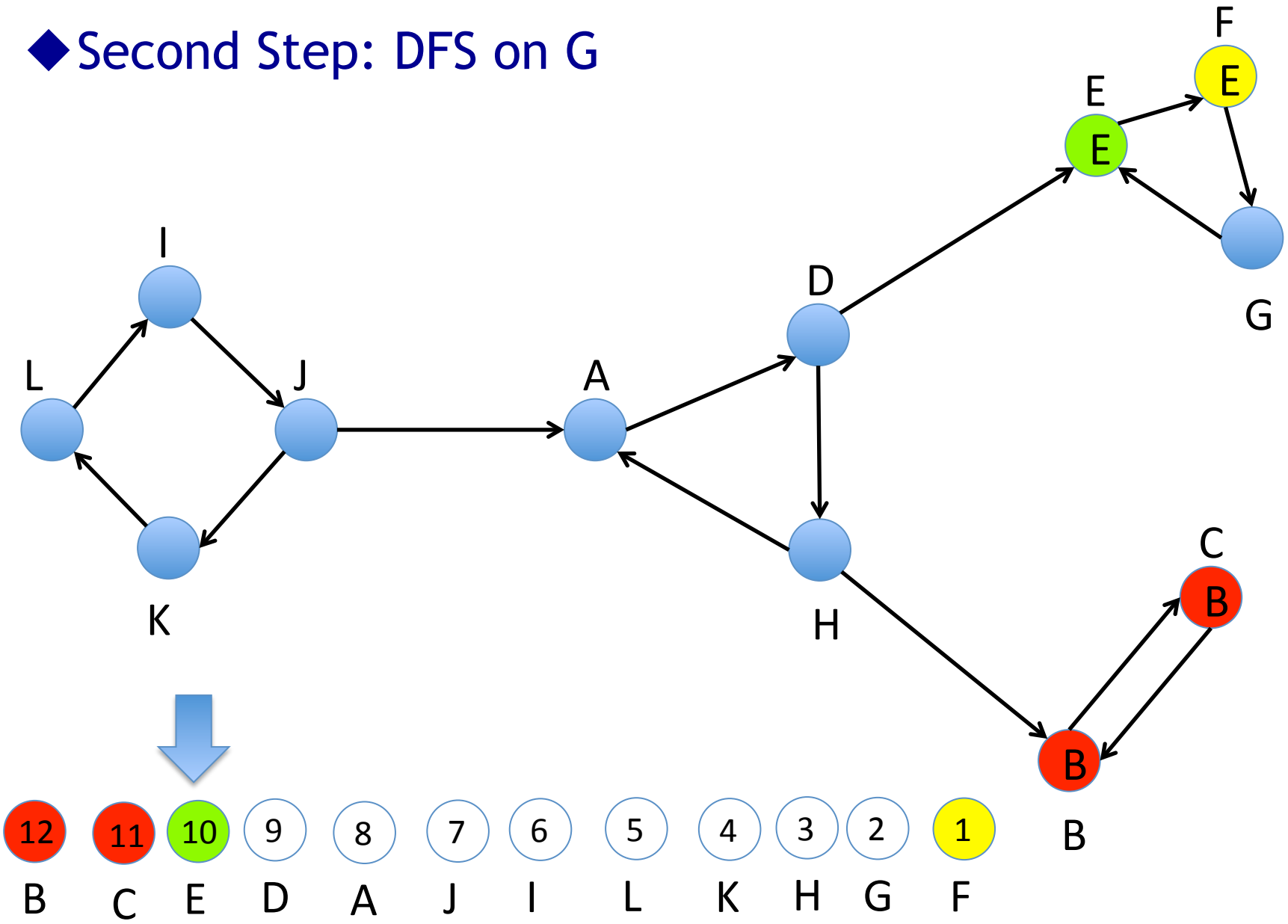
Kosaraju's Algorithm Simulation

◆Second Step: DFS on G



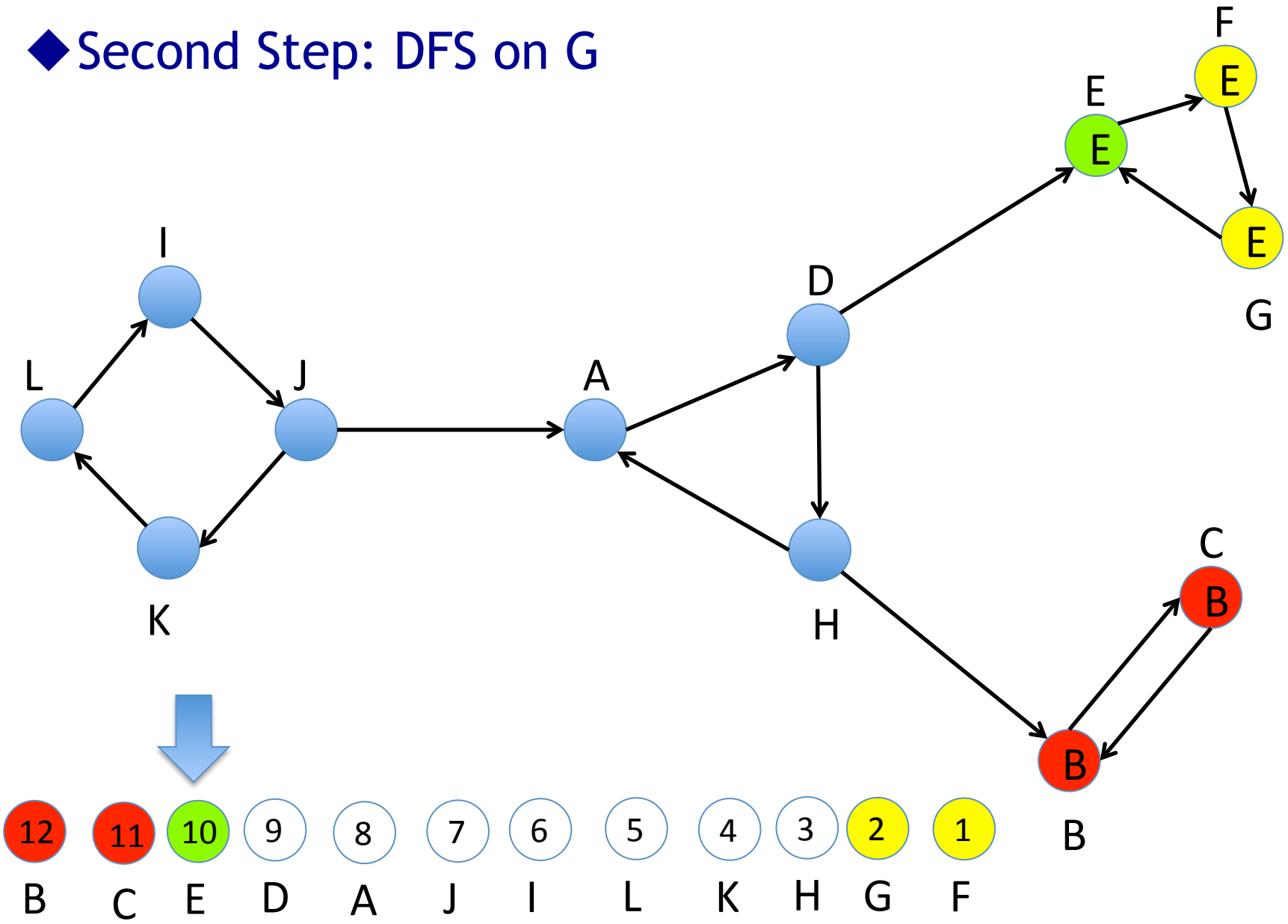
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



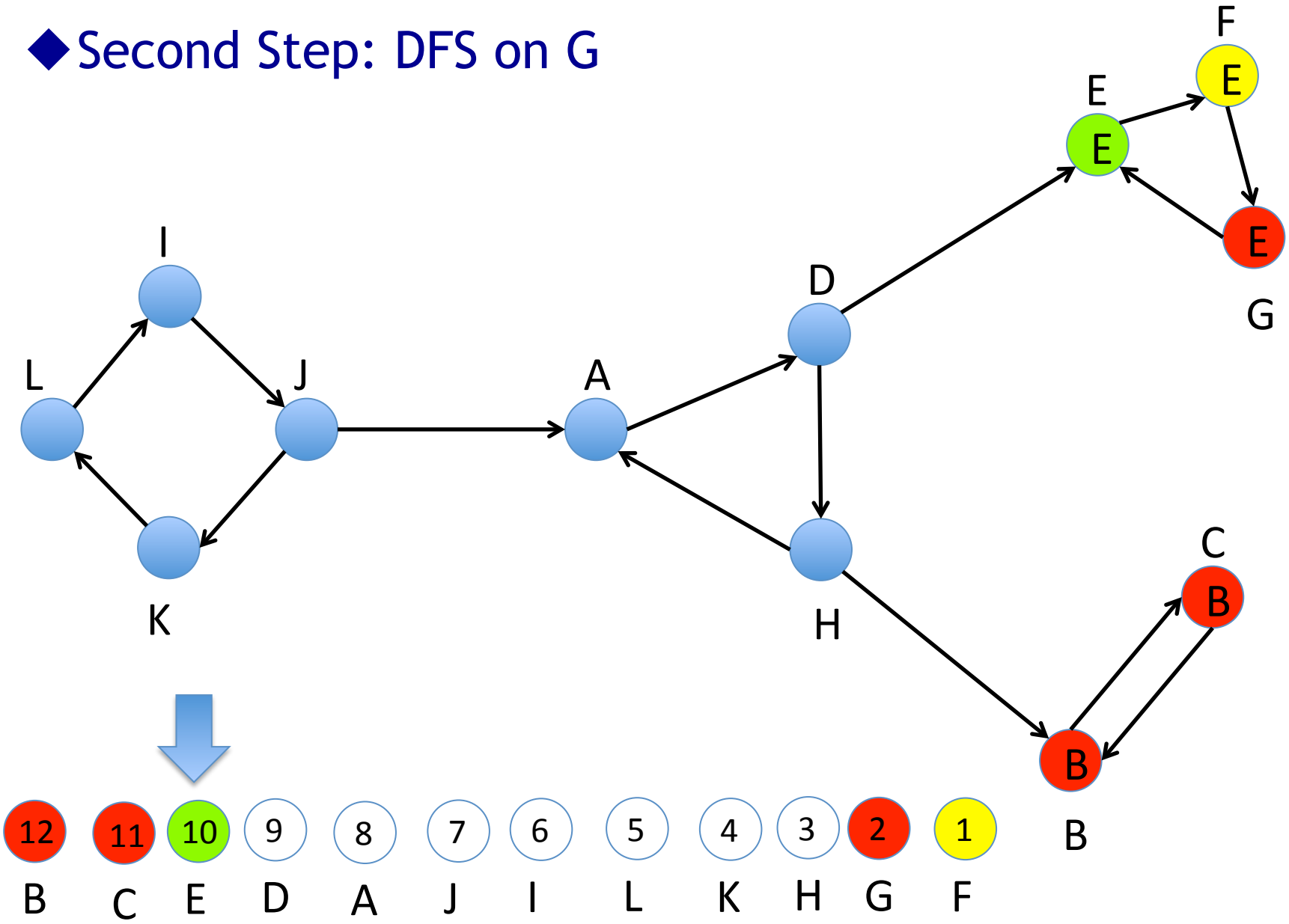
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



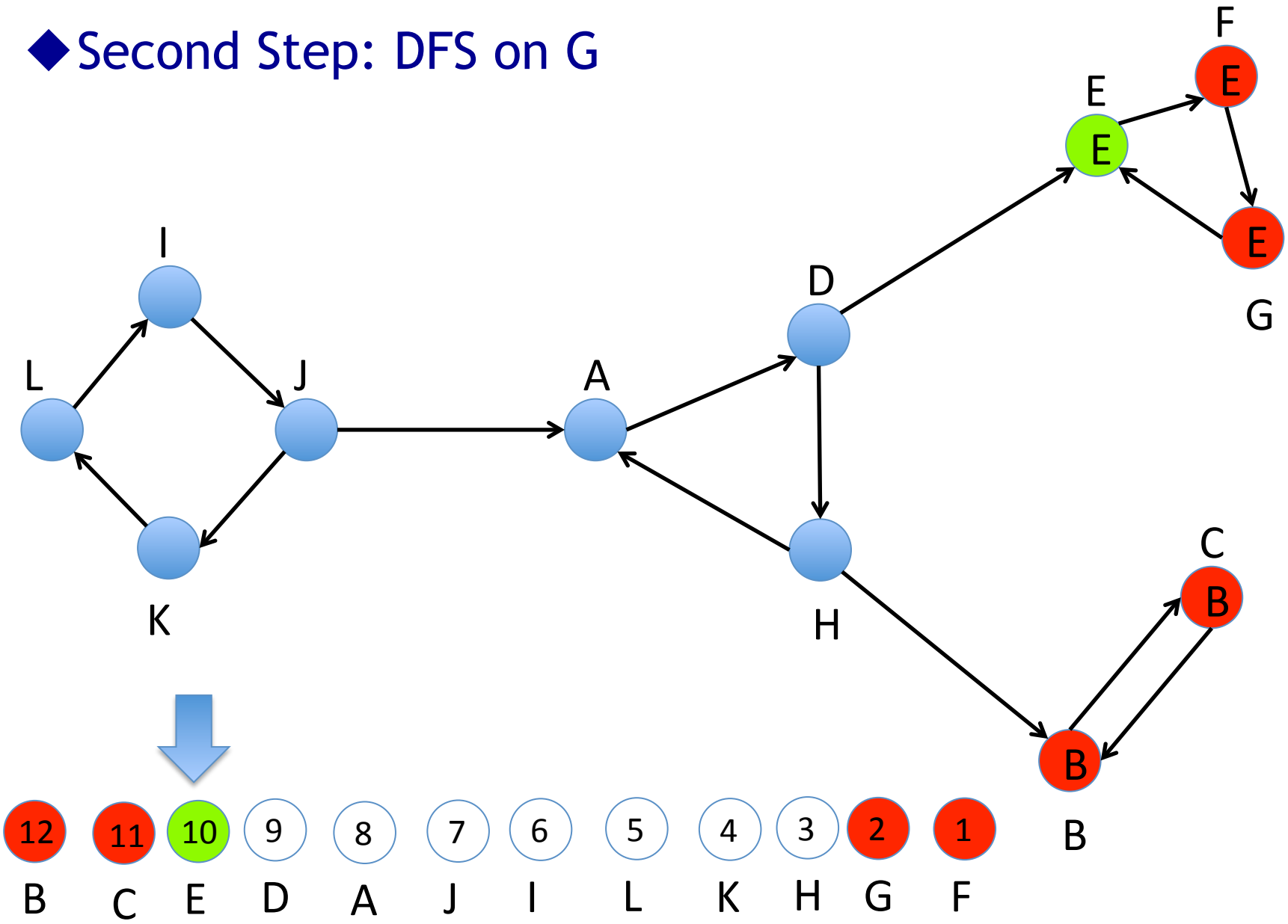
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



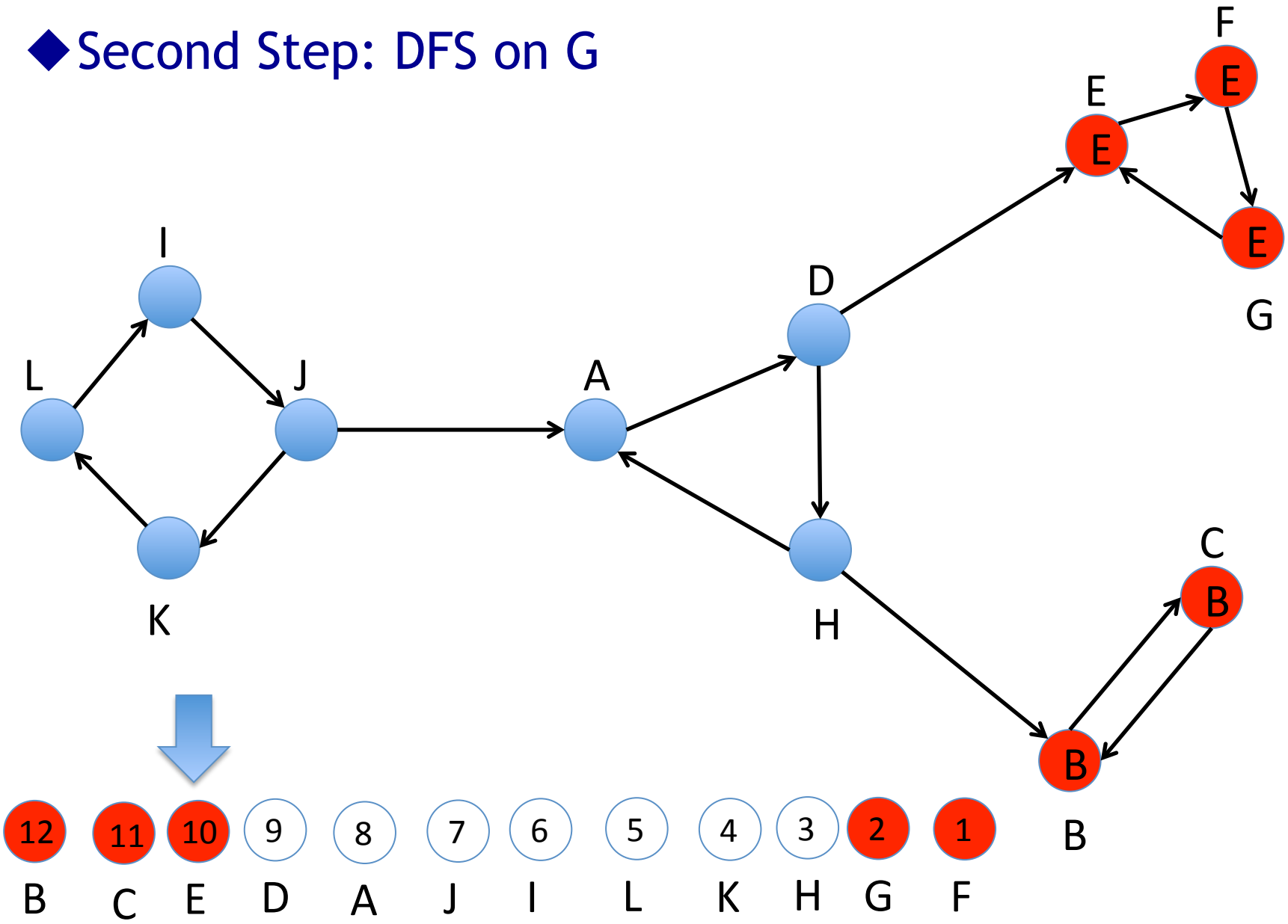
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



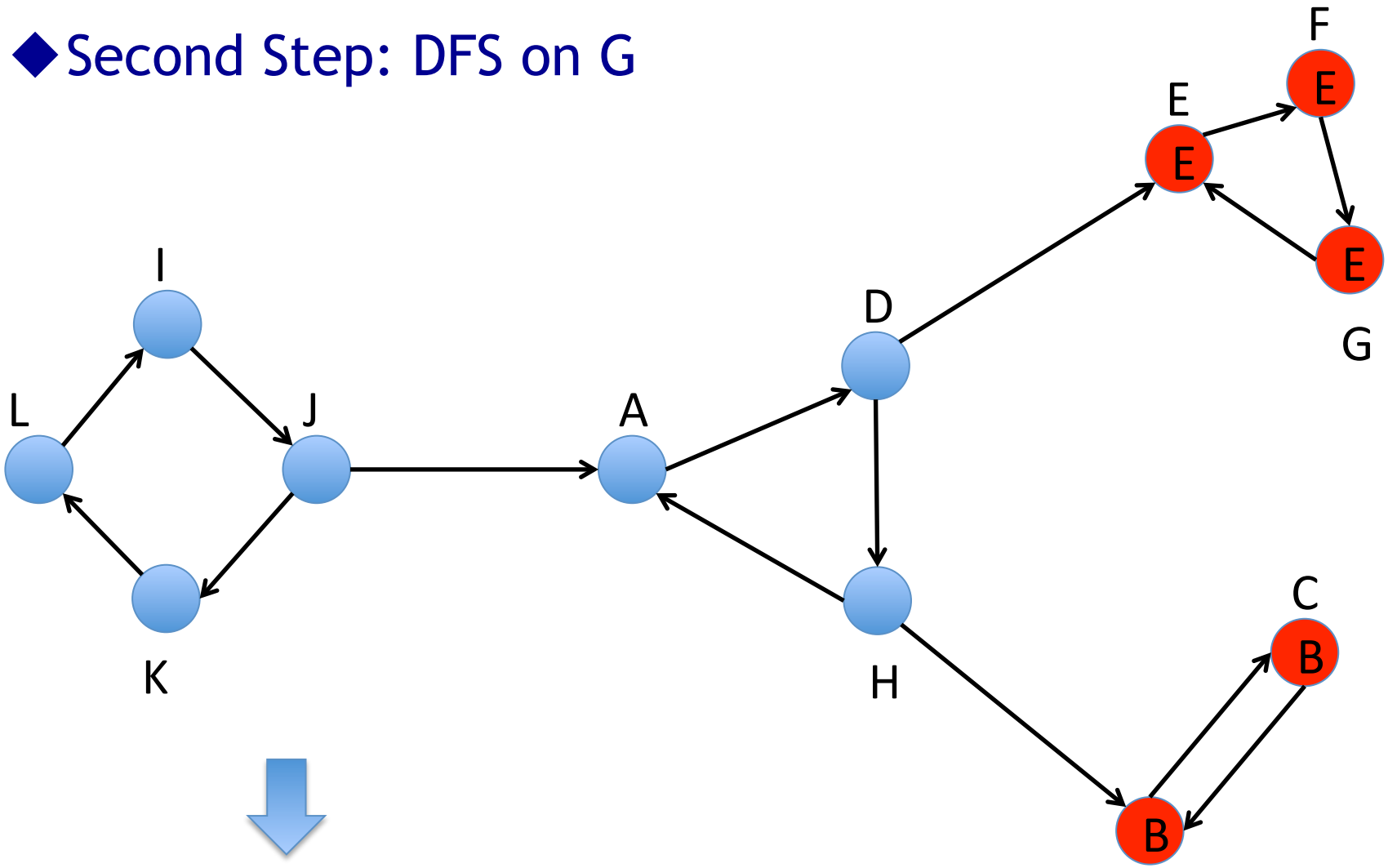
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



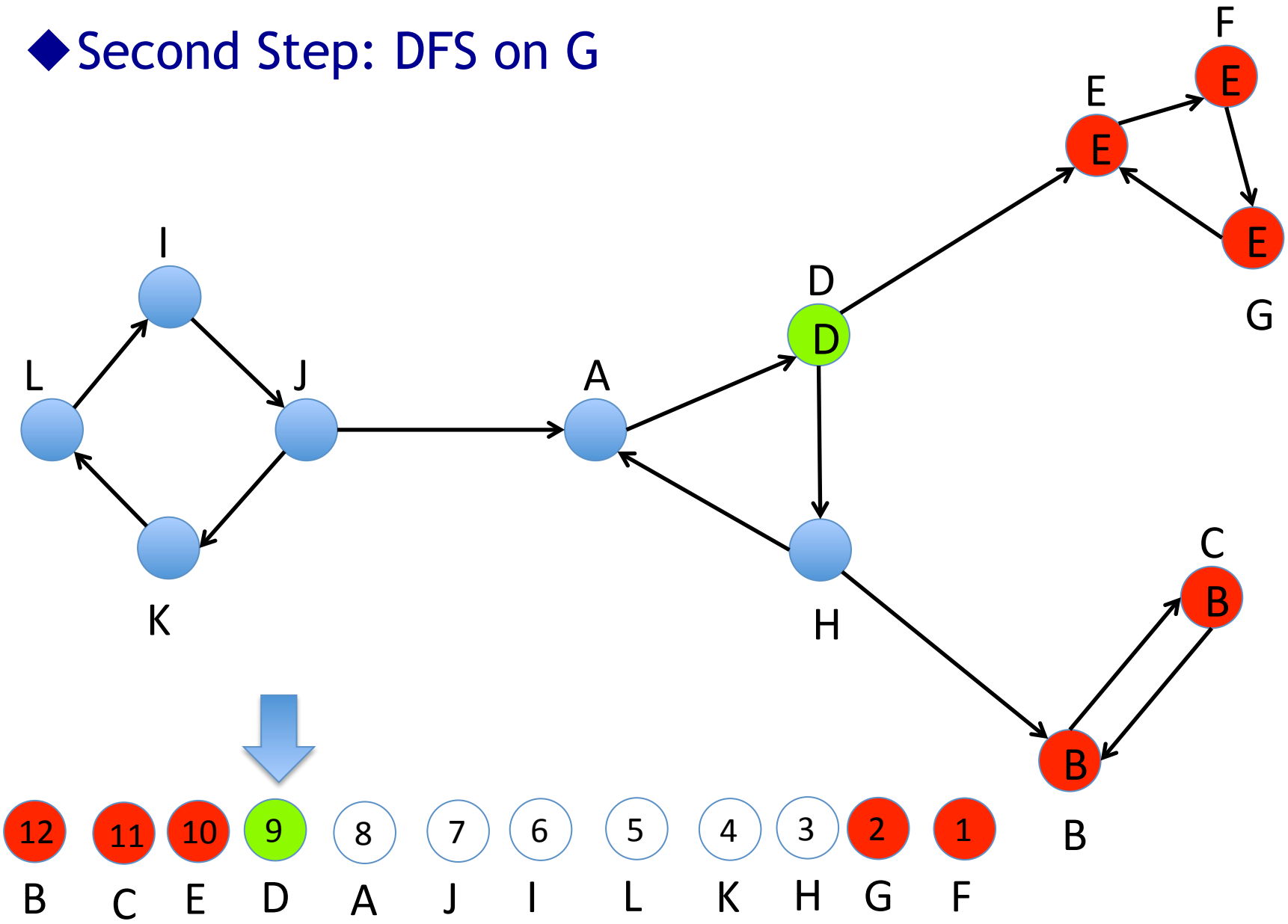
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



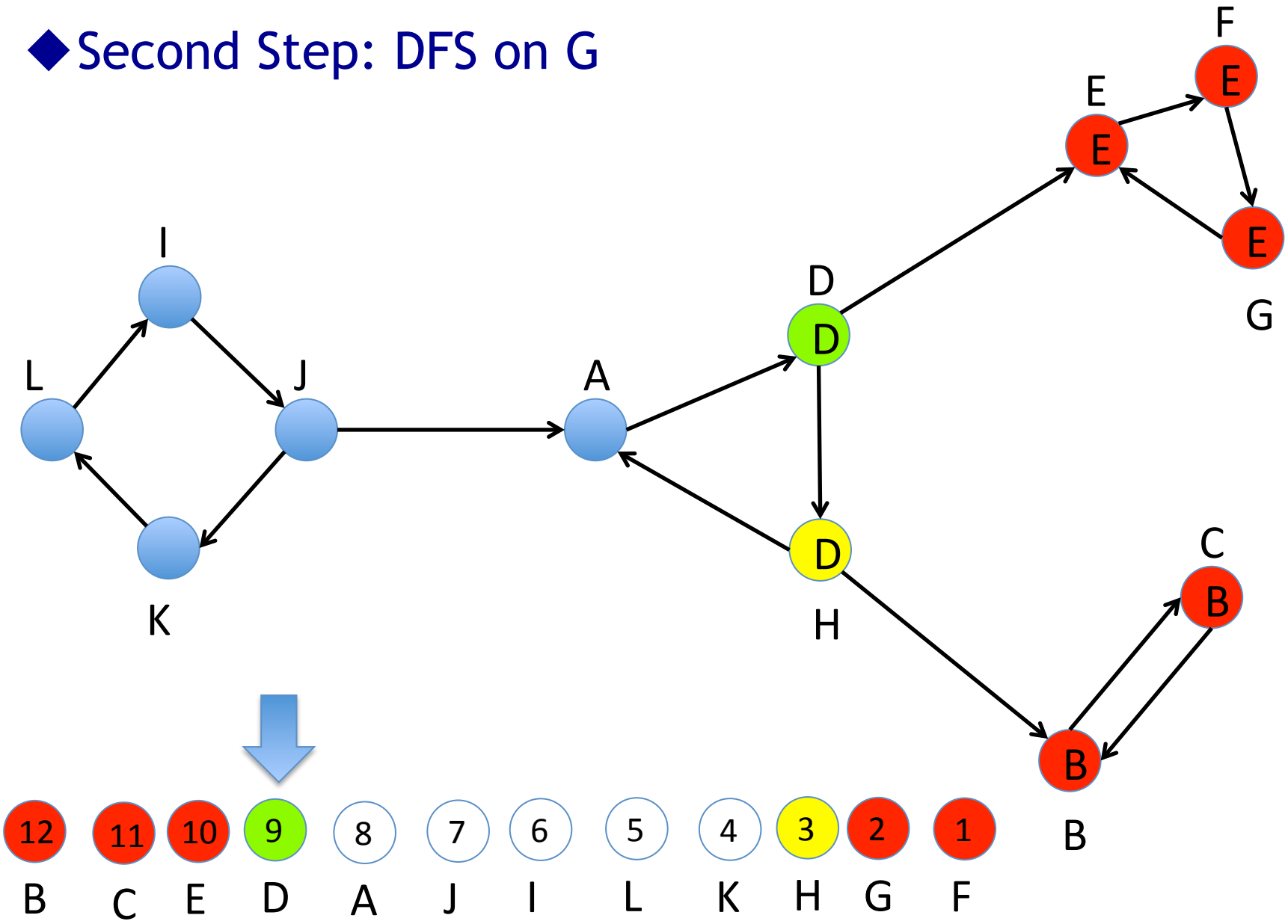
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



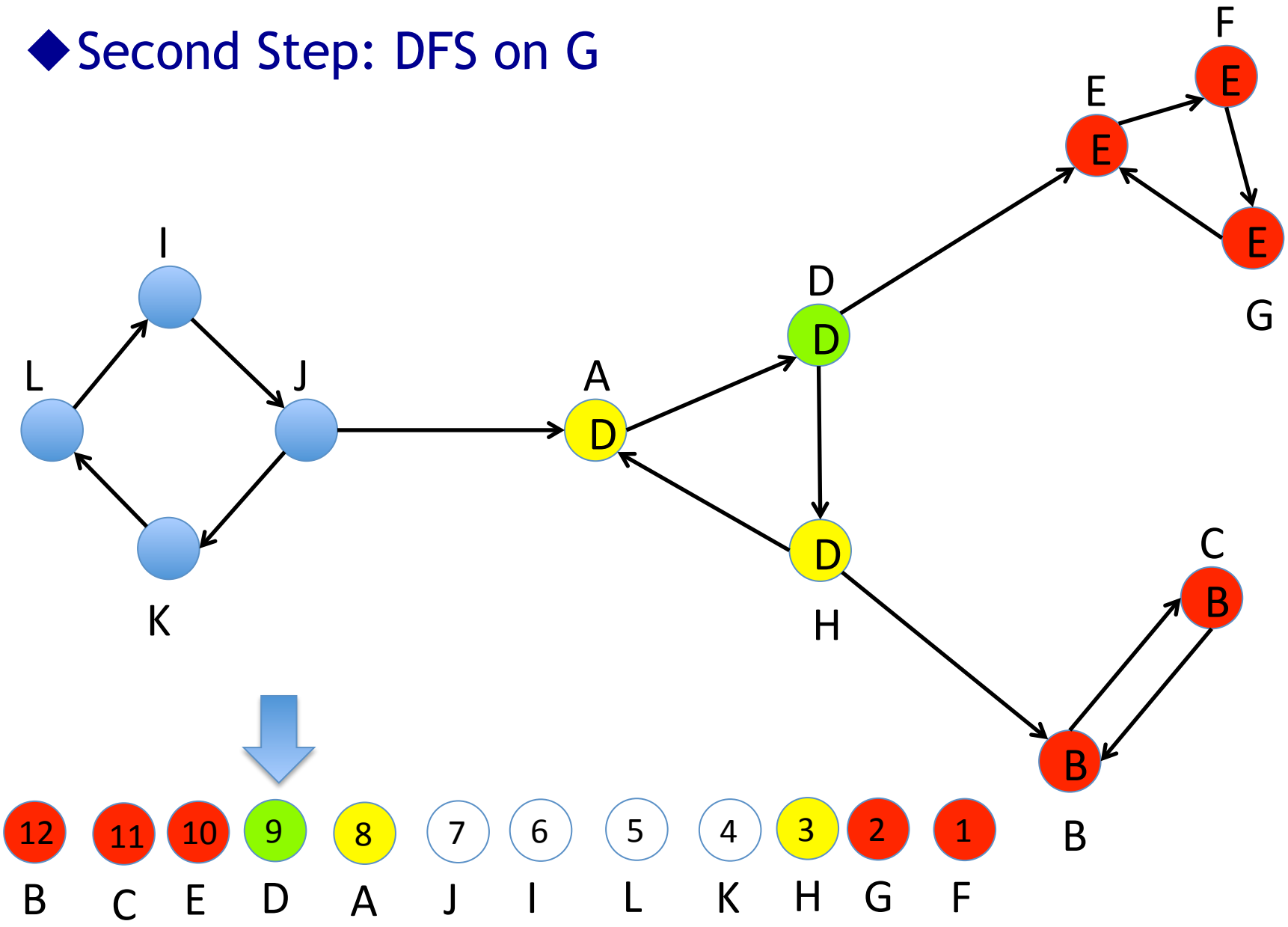
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



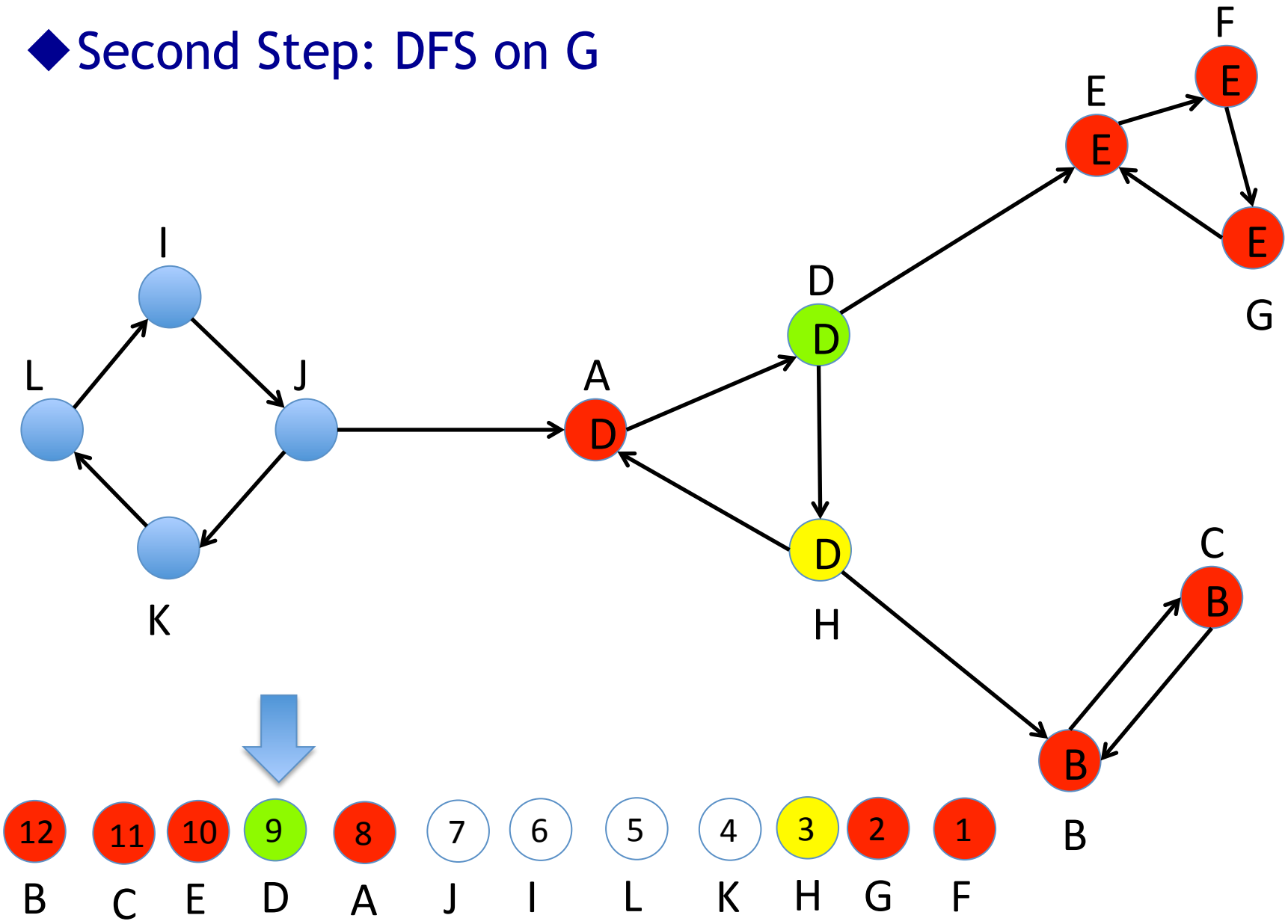
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



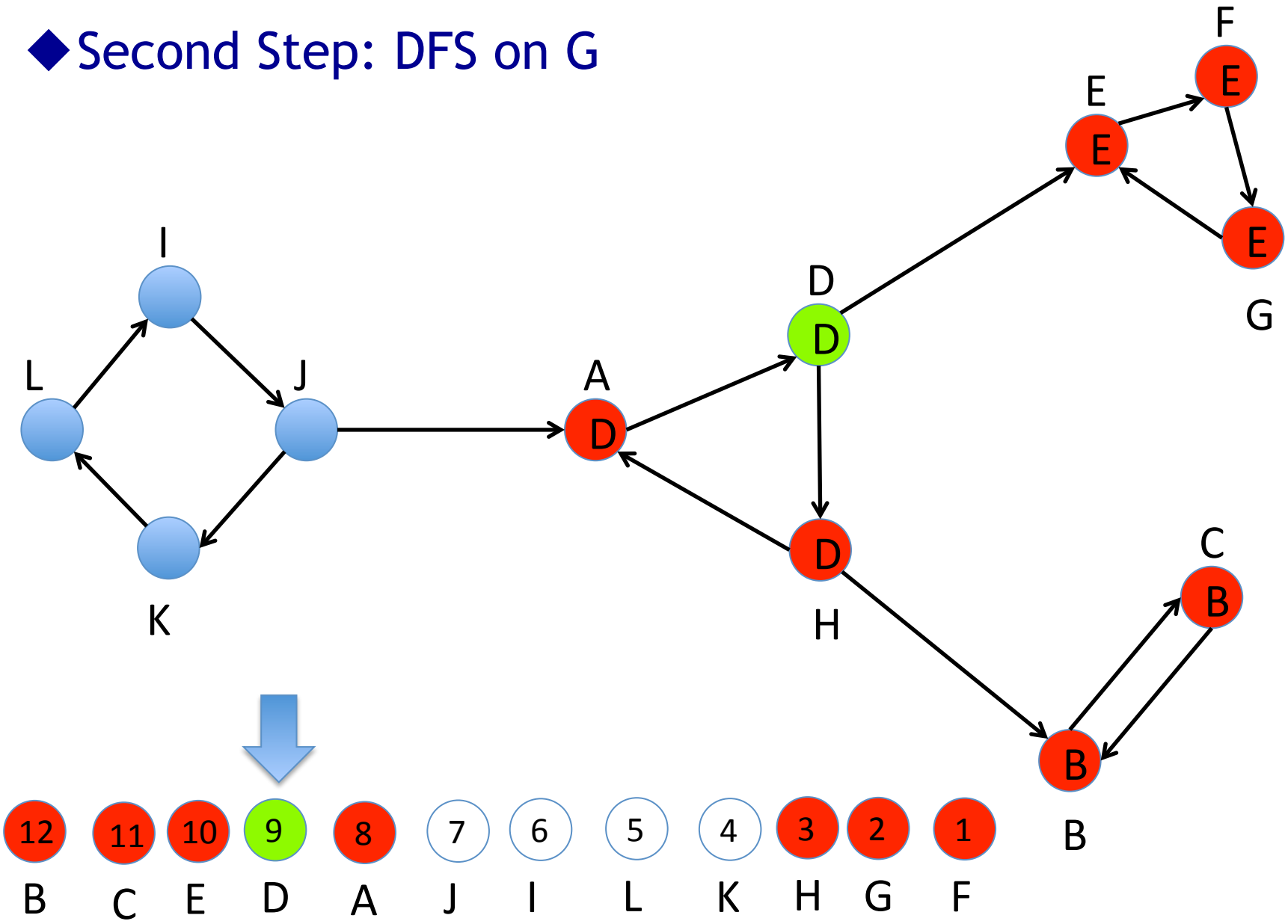
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



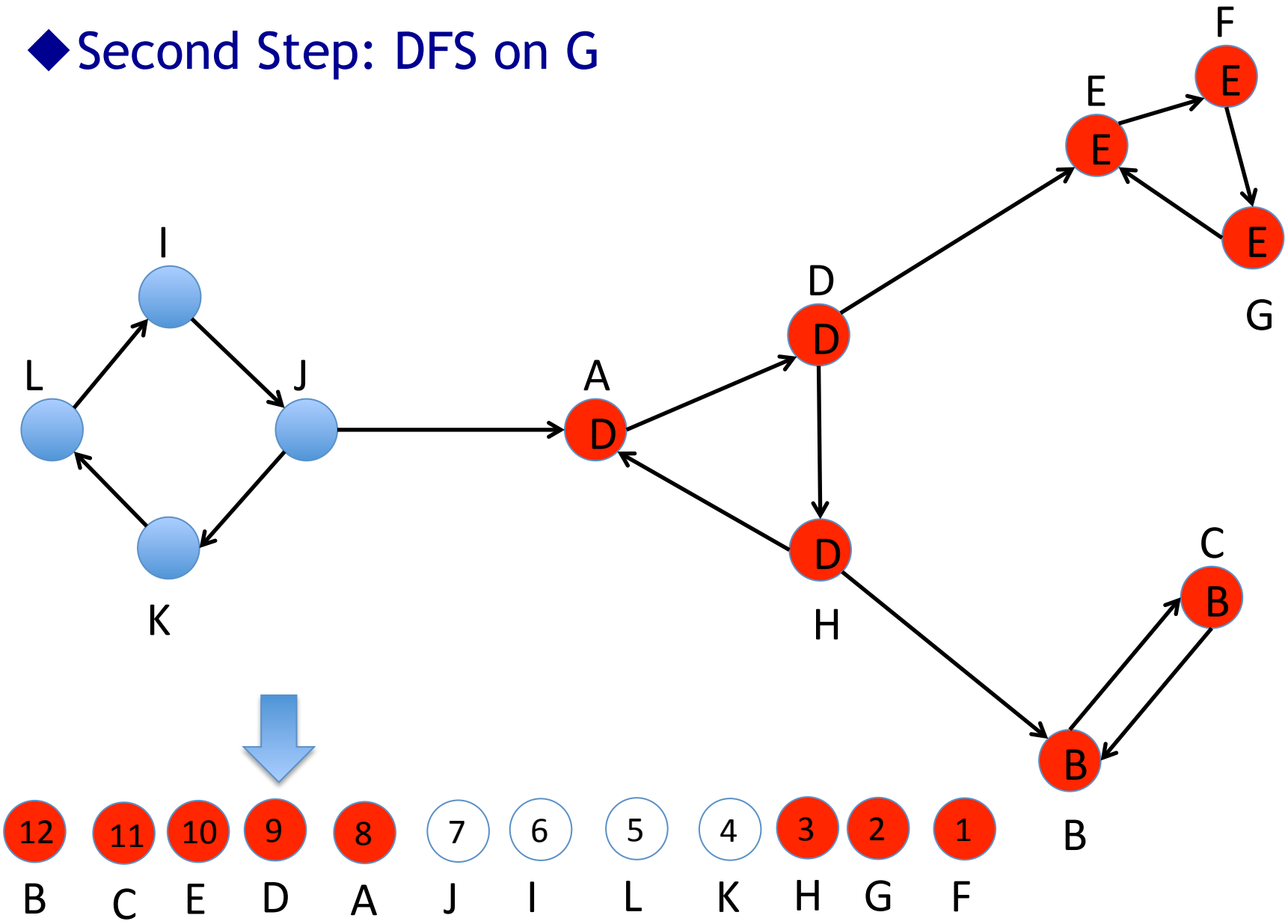
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



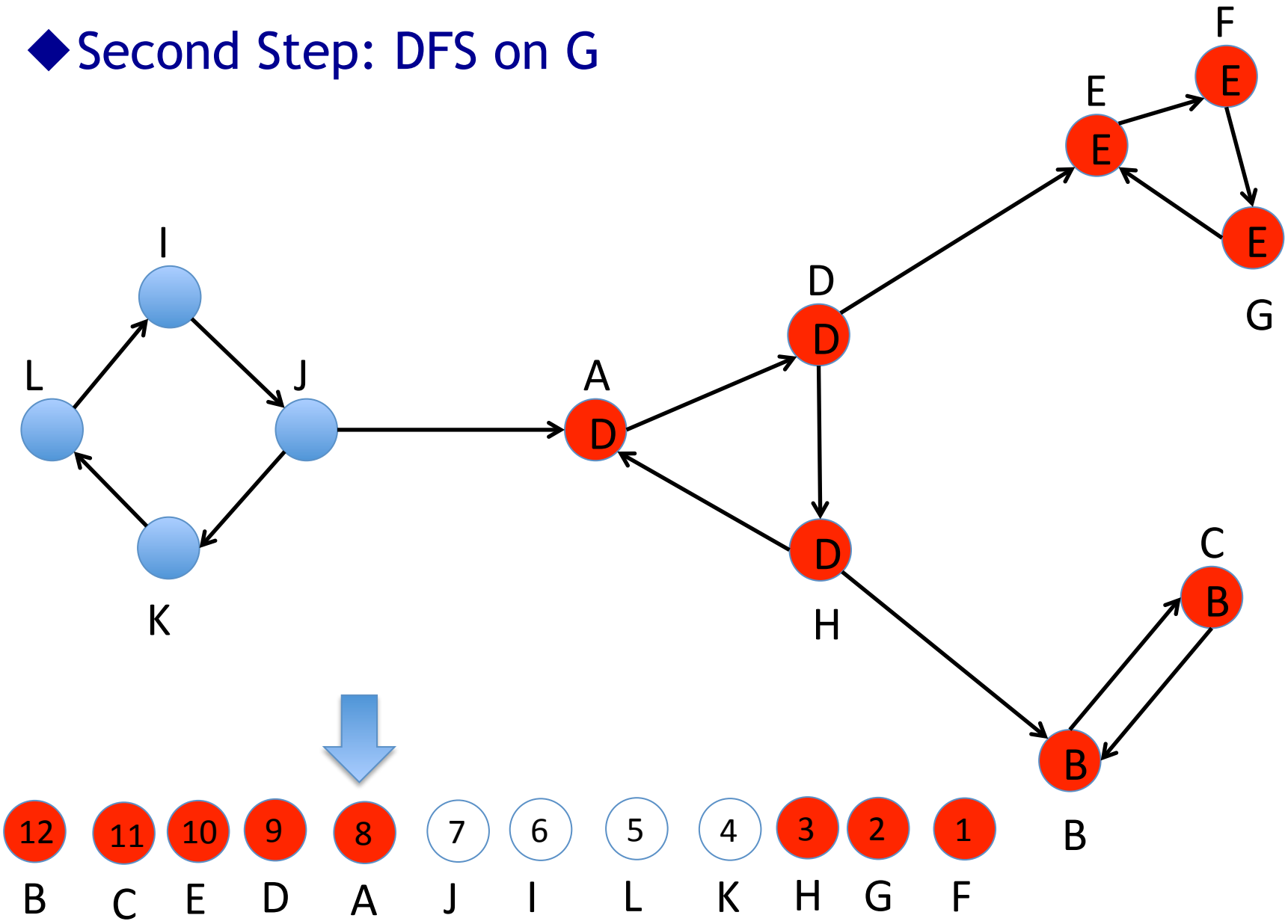
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



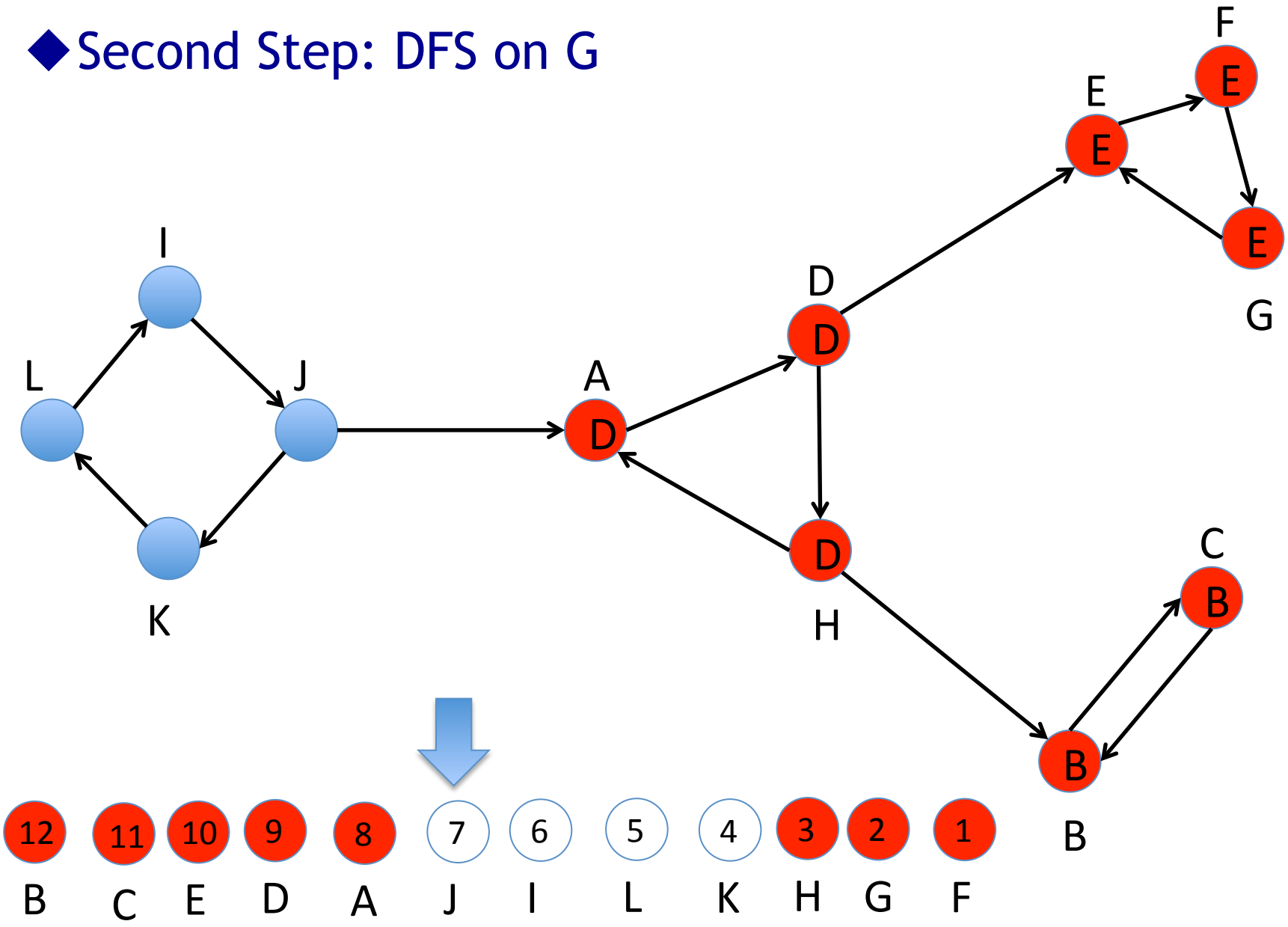
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



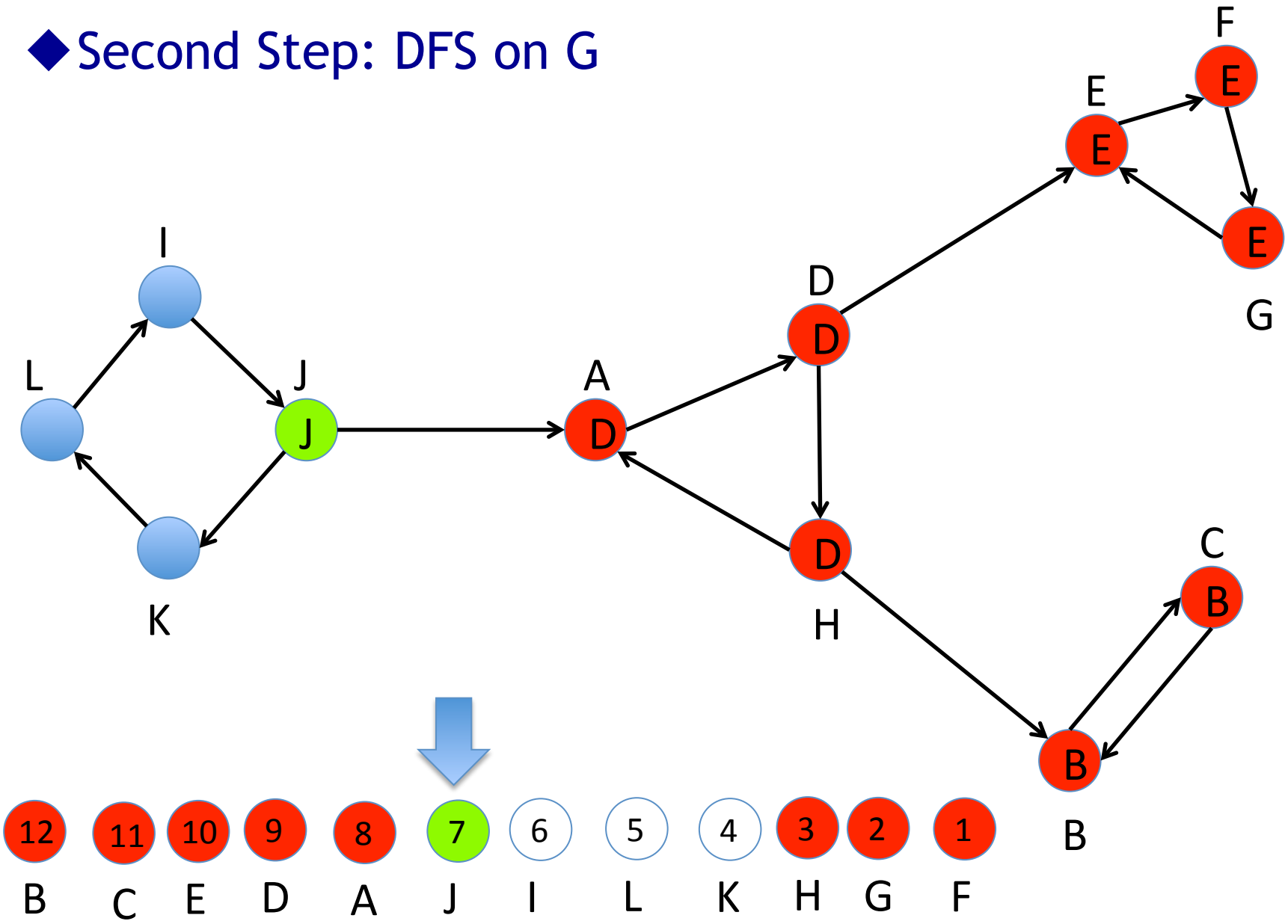
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



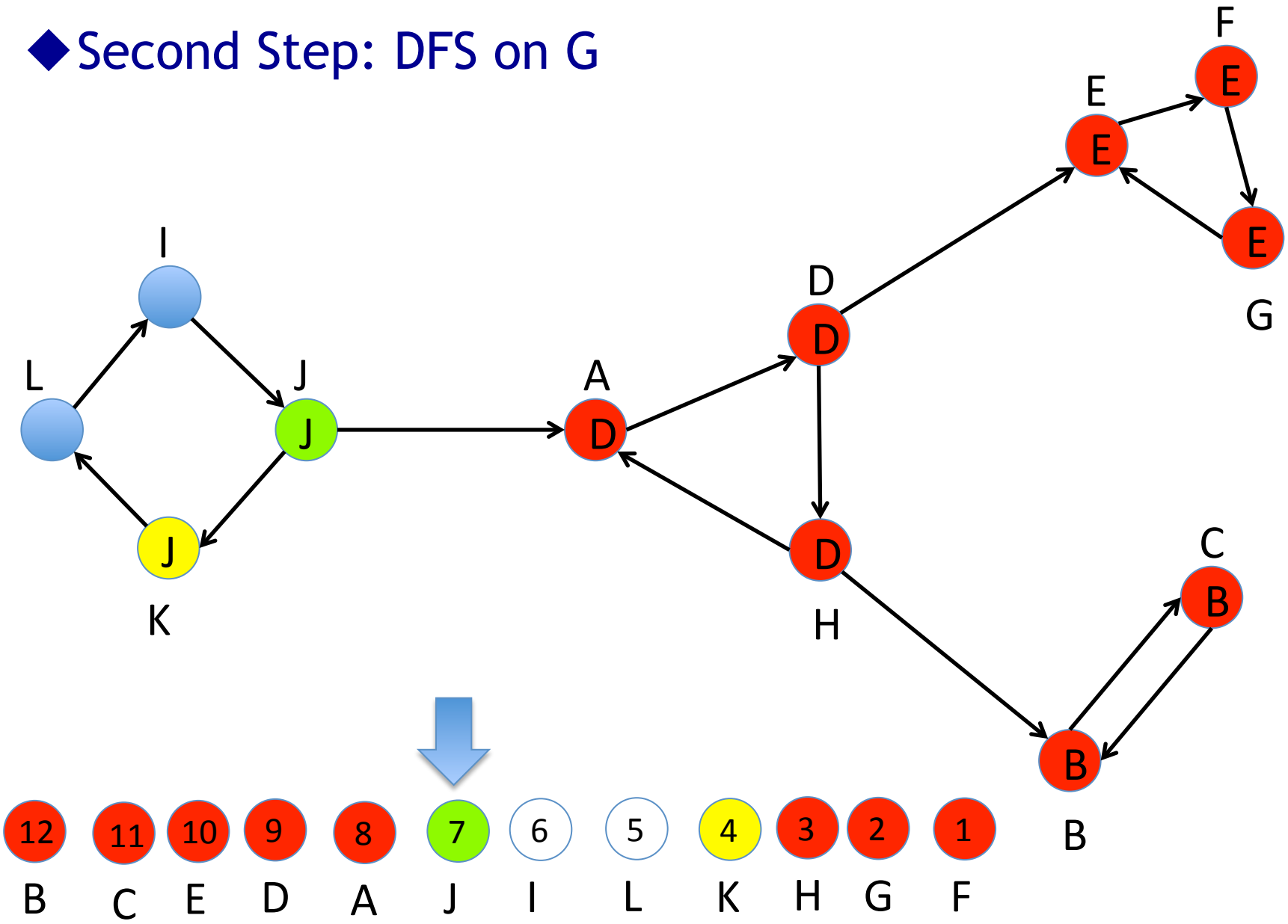
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



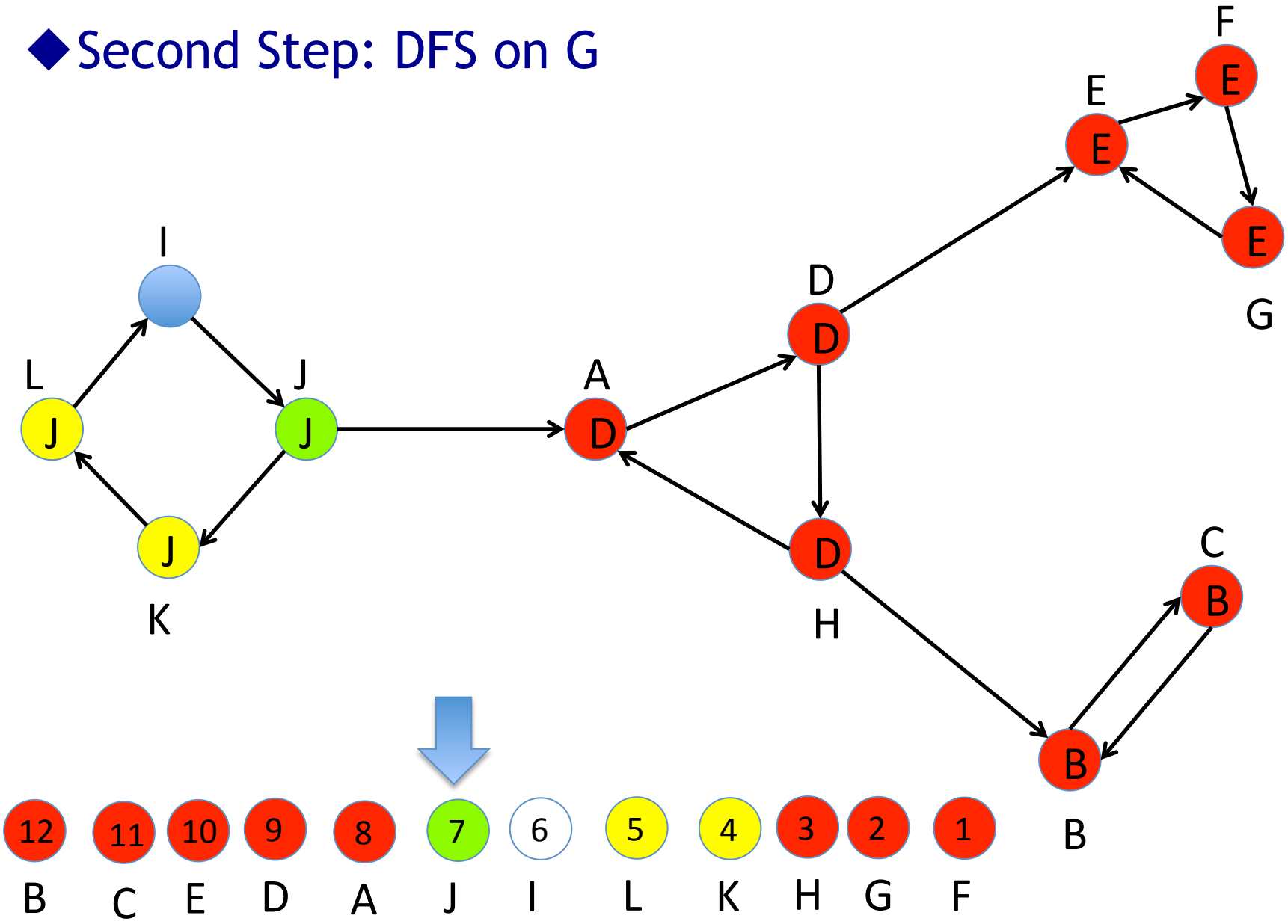
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



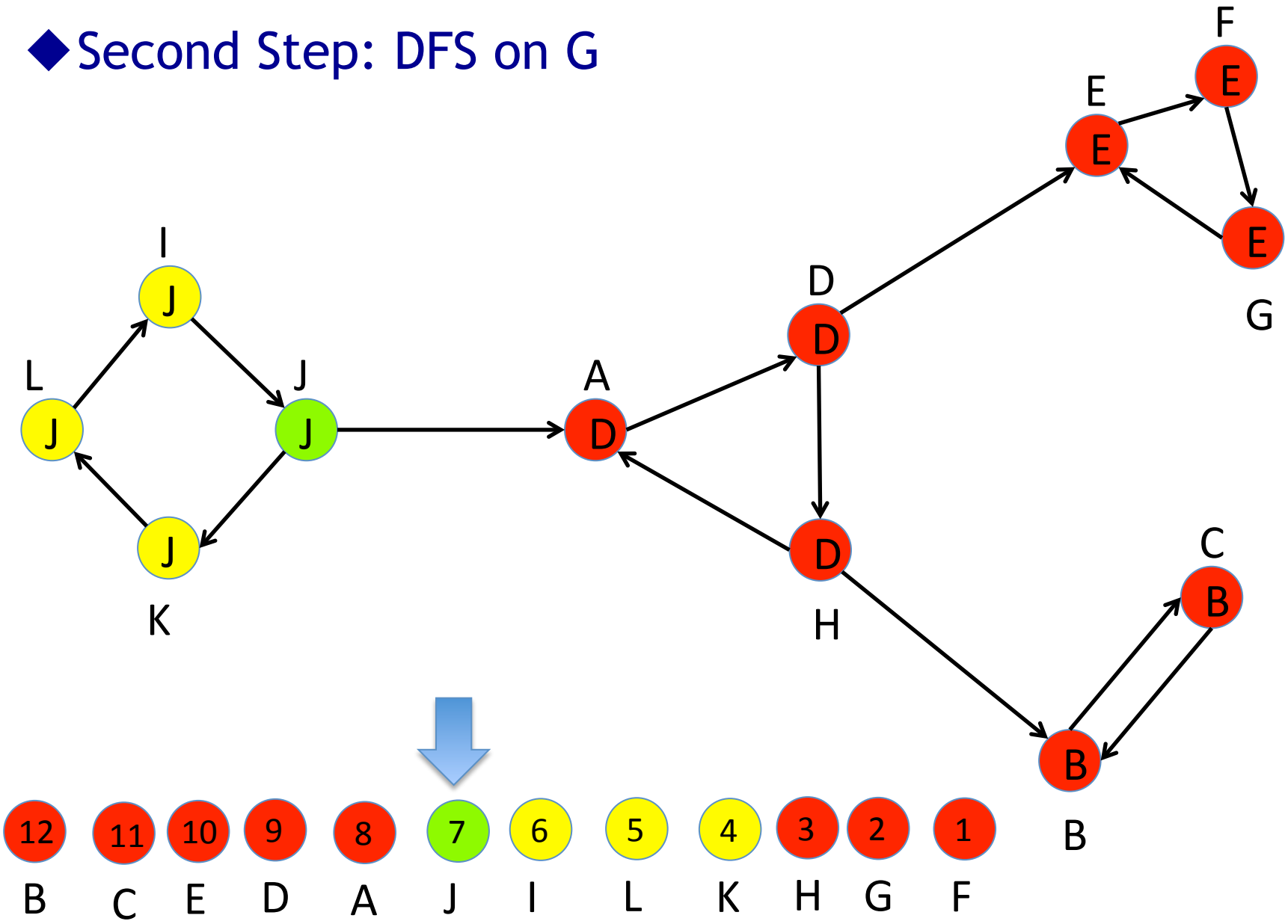
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



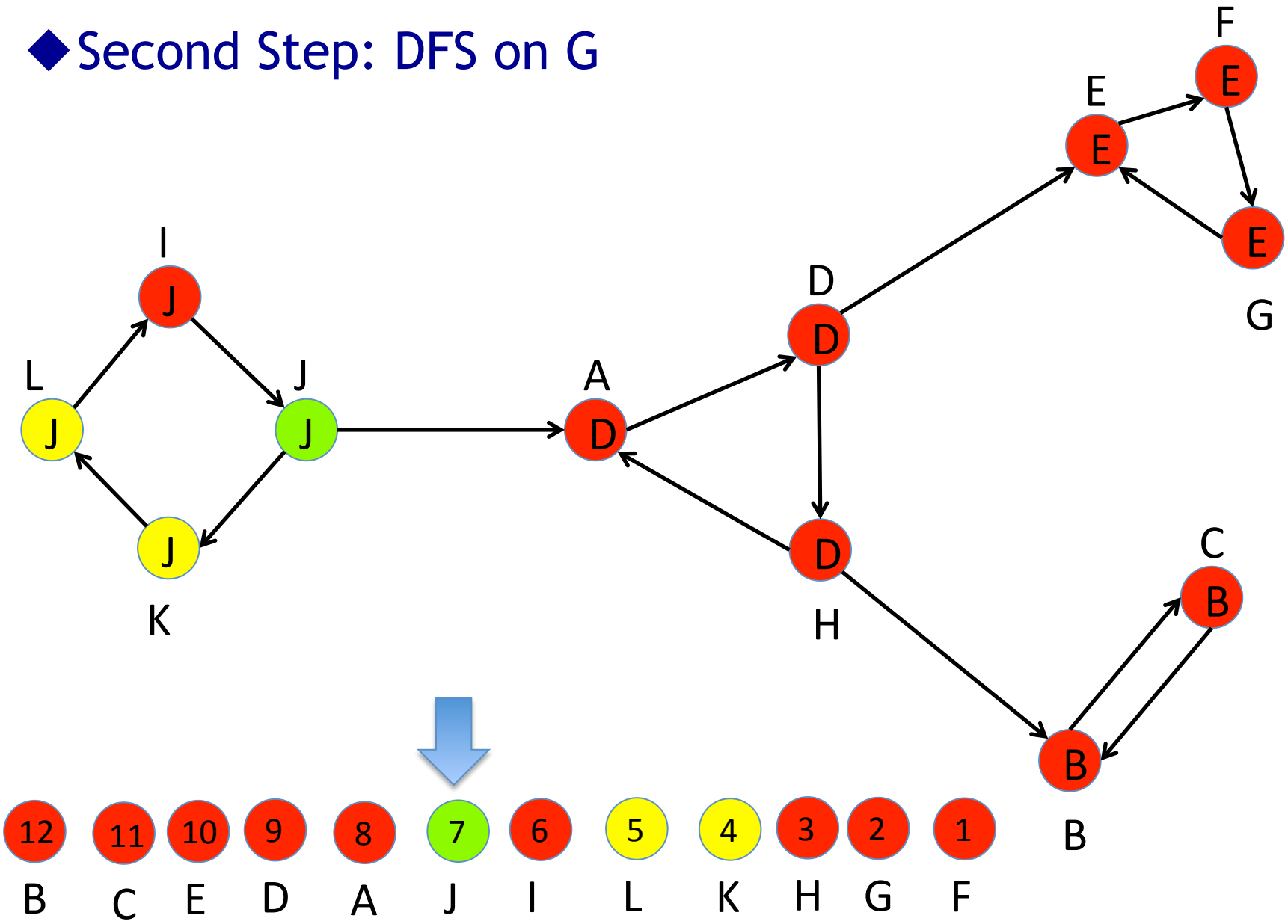
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



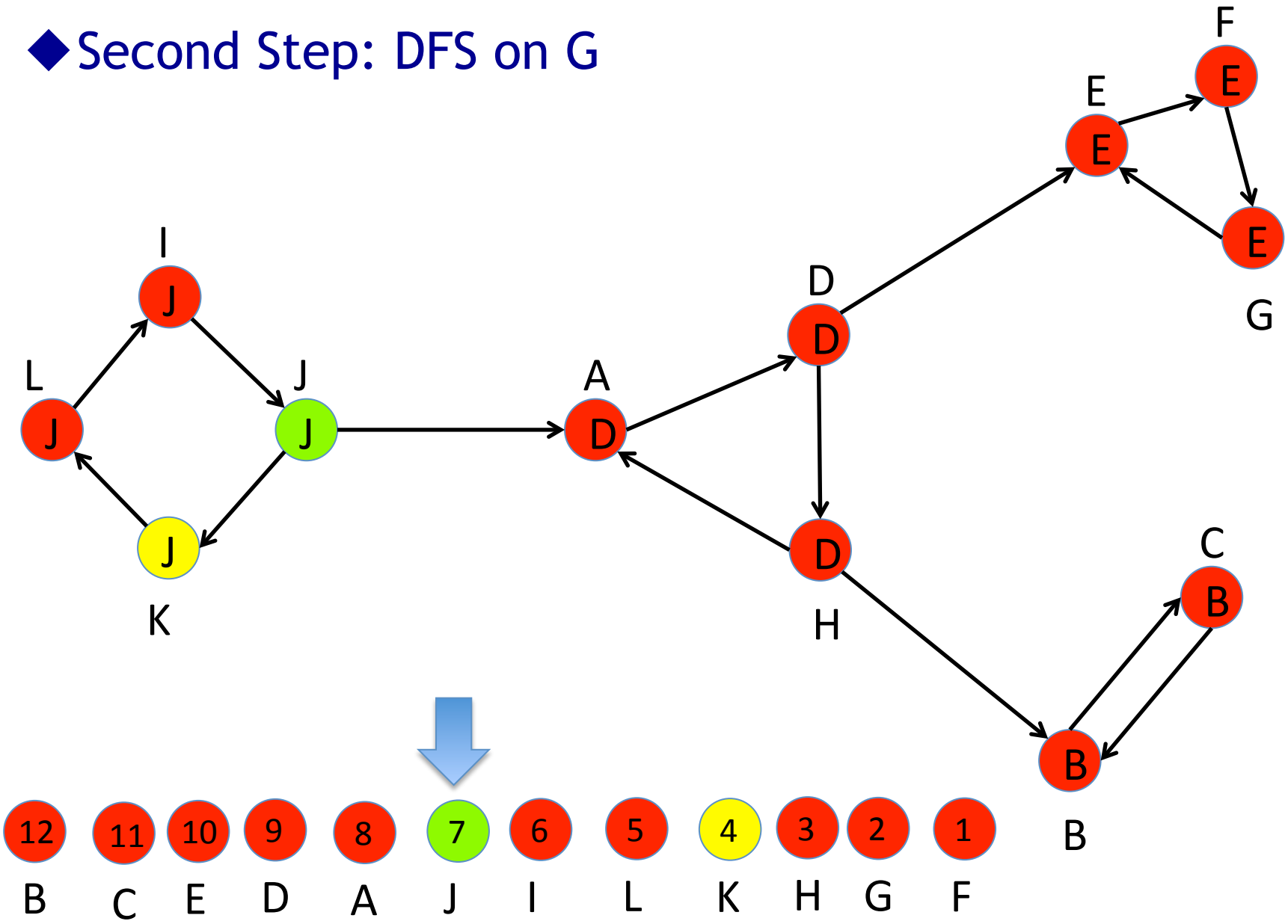
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



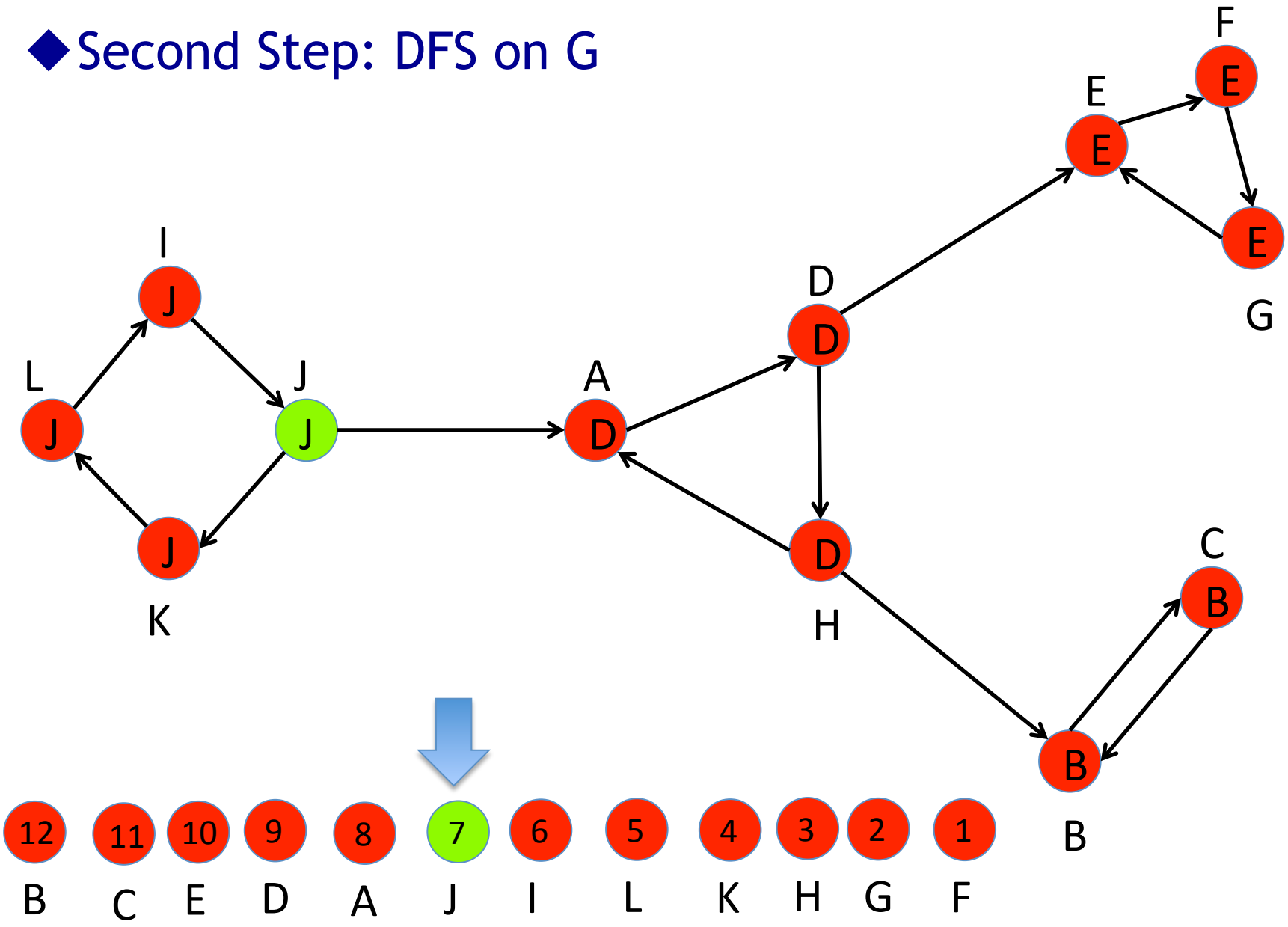
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



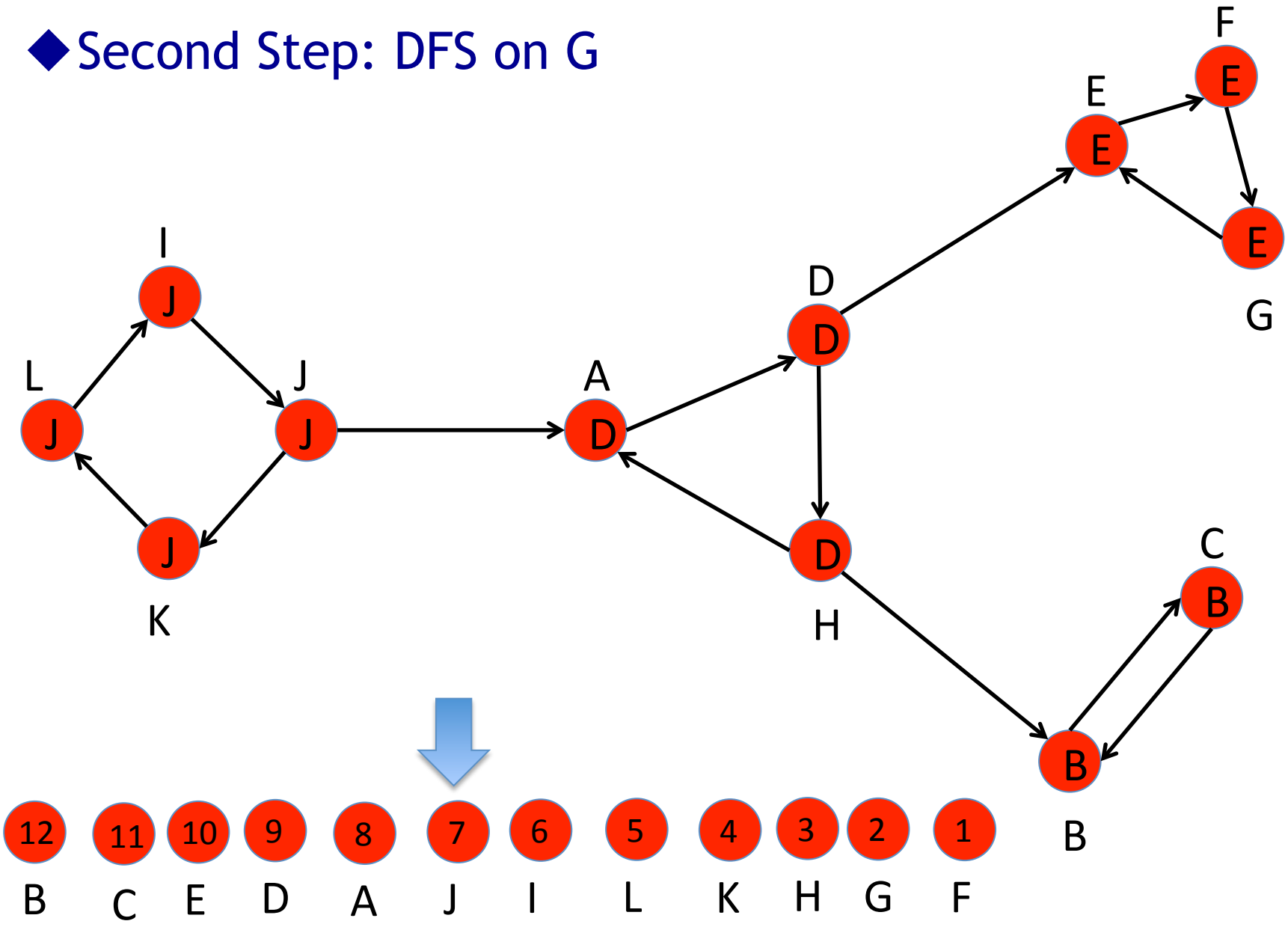
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



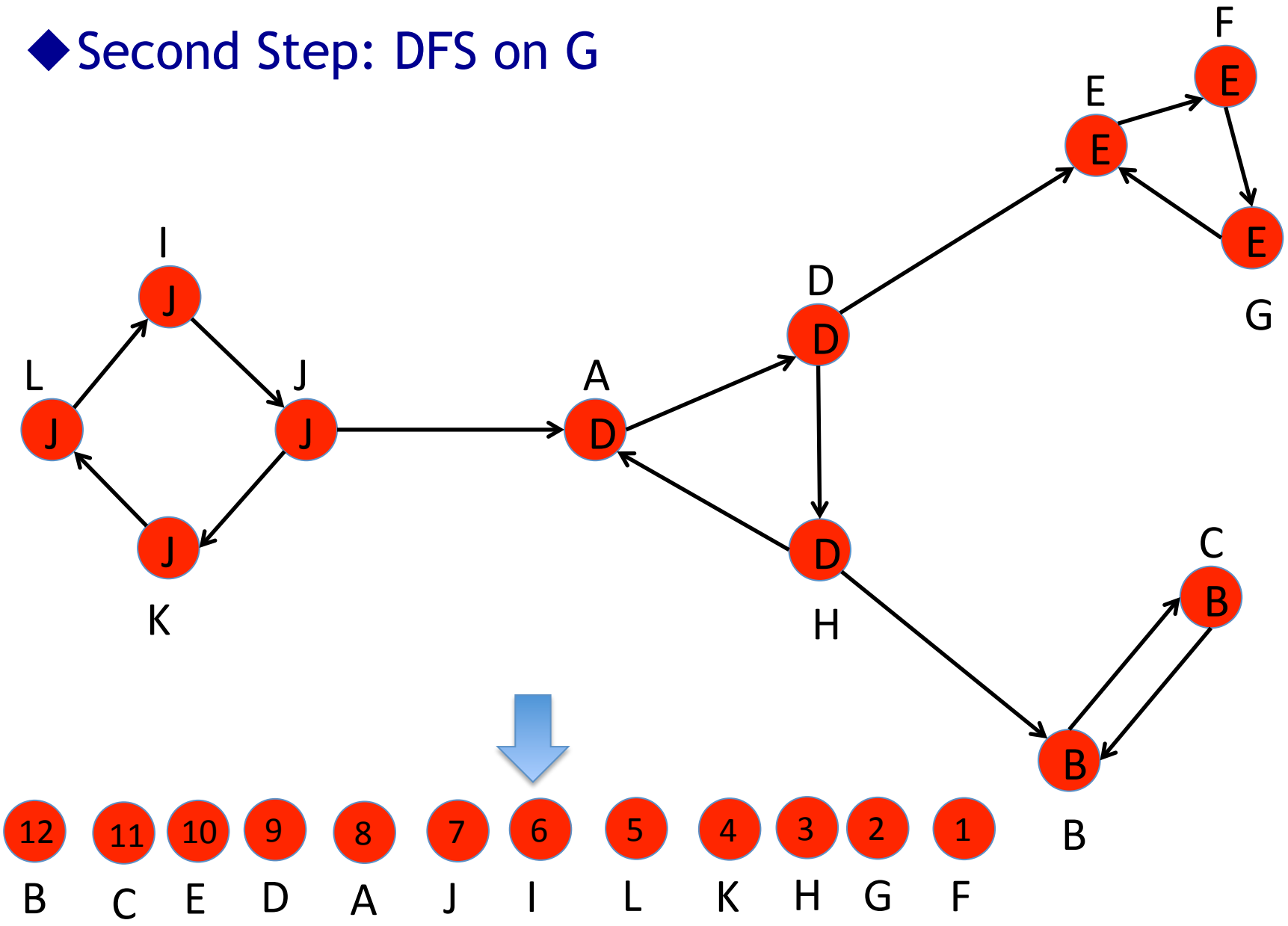
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



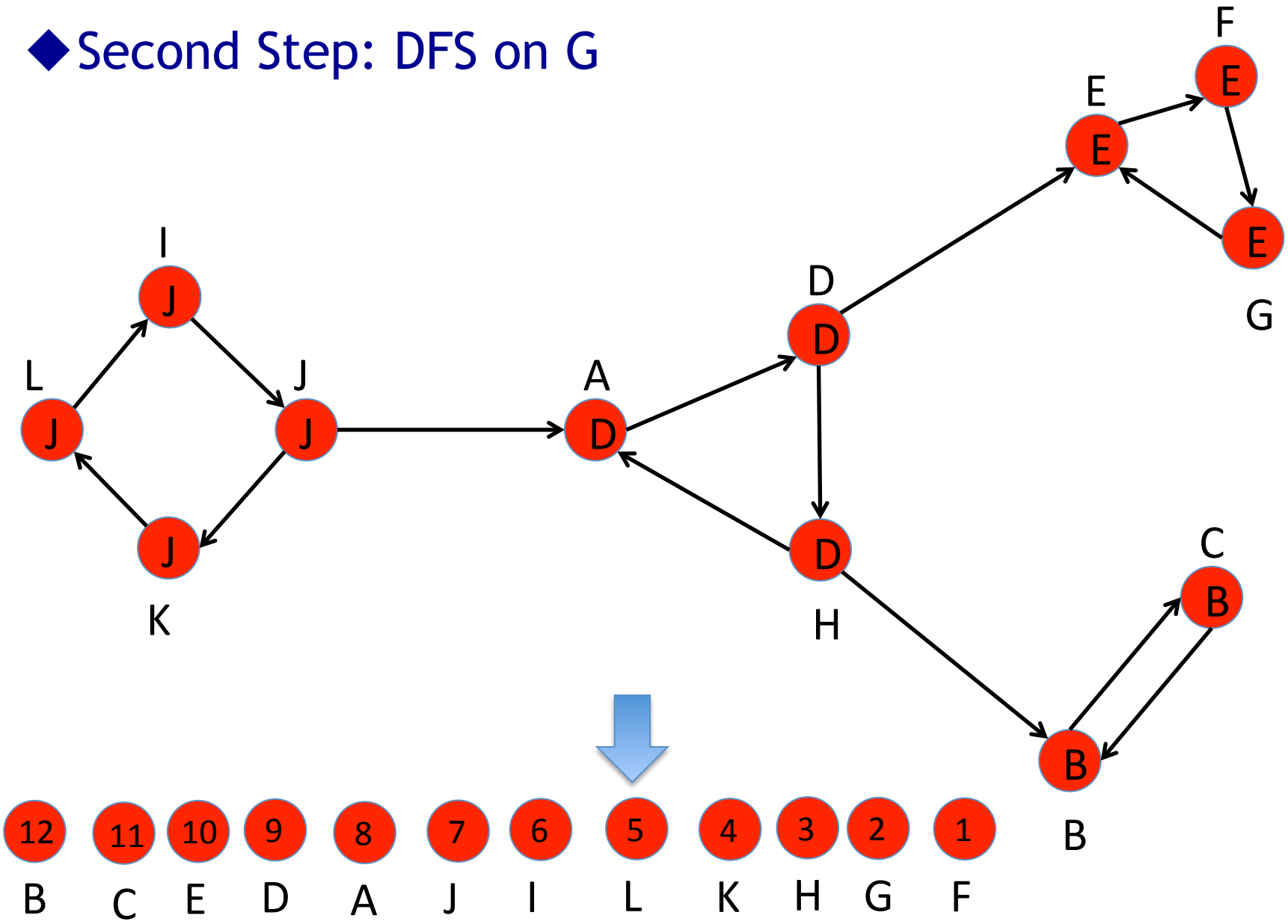
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



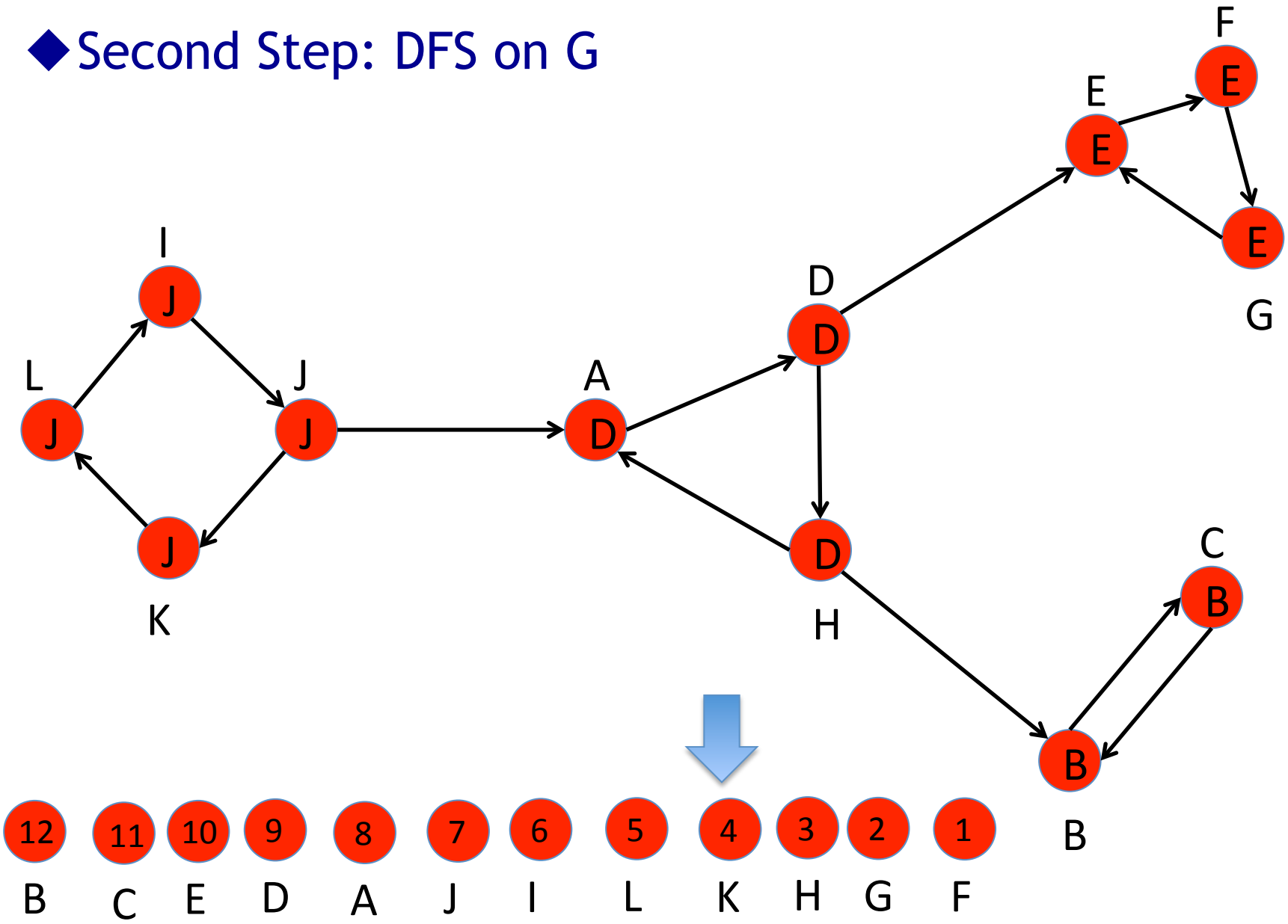
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



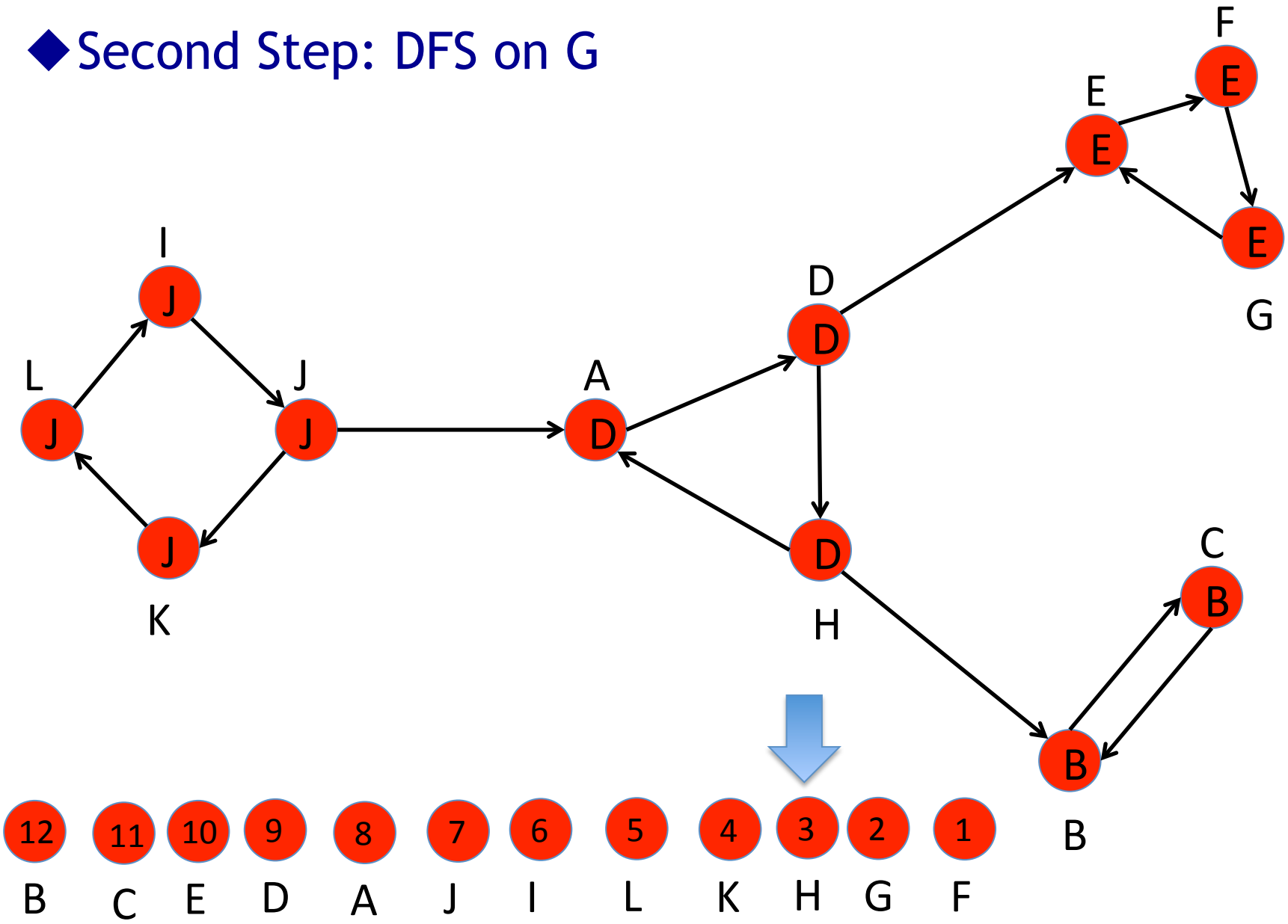
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



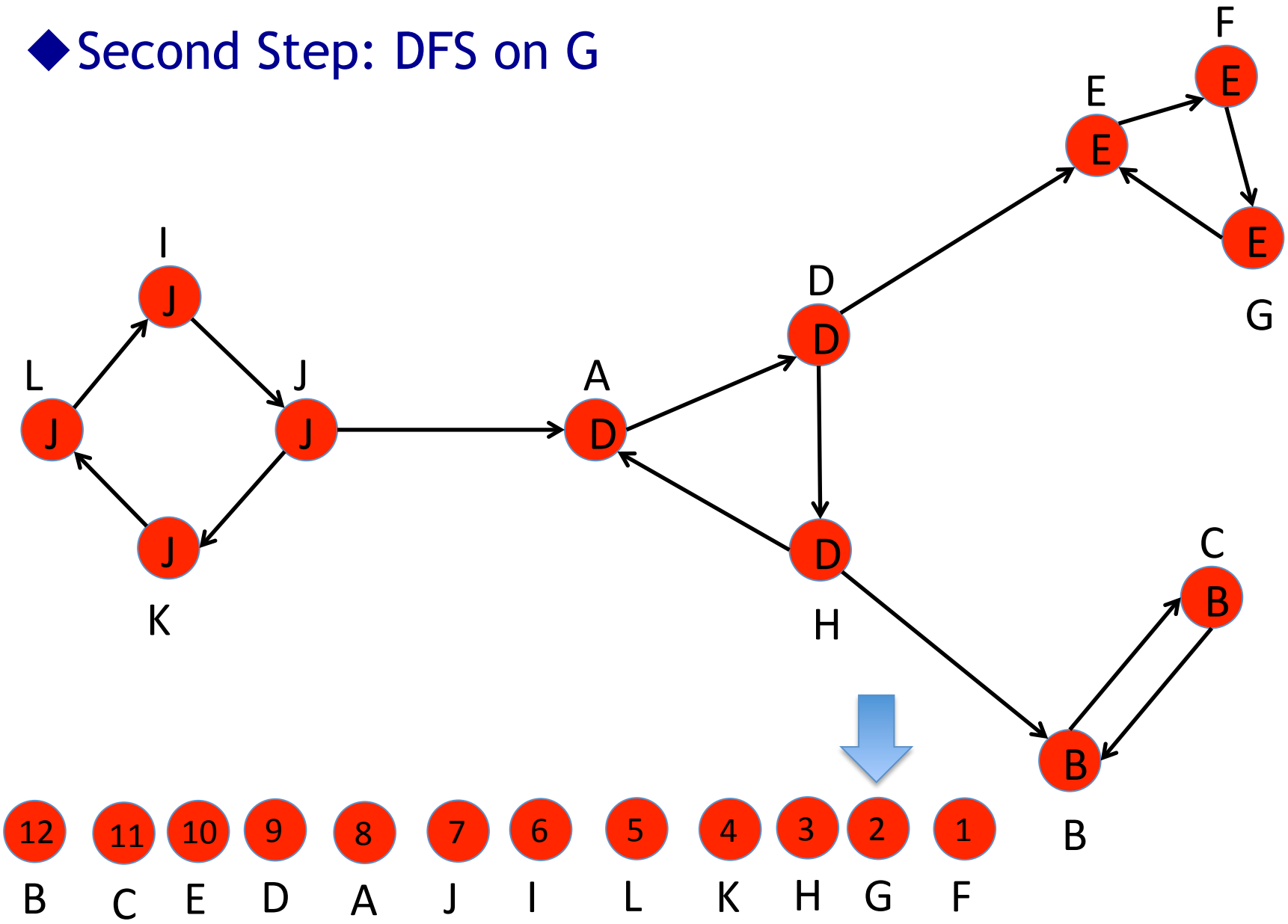
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



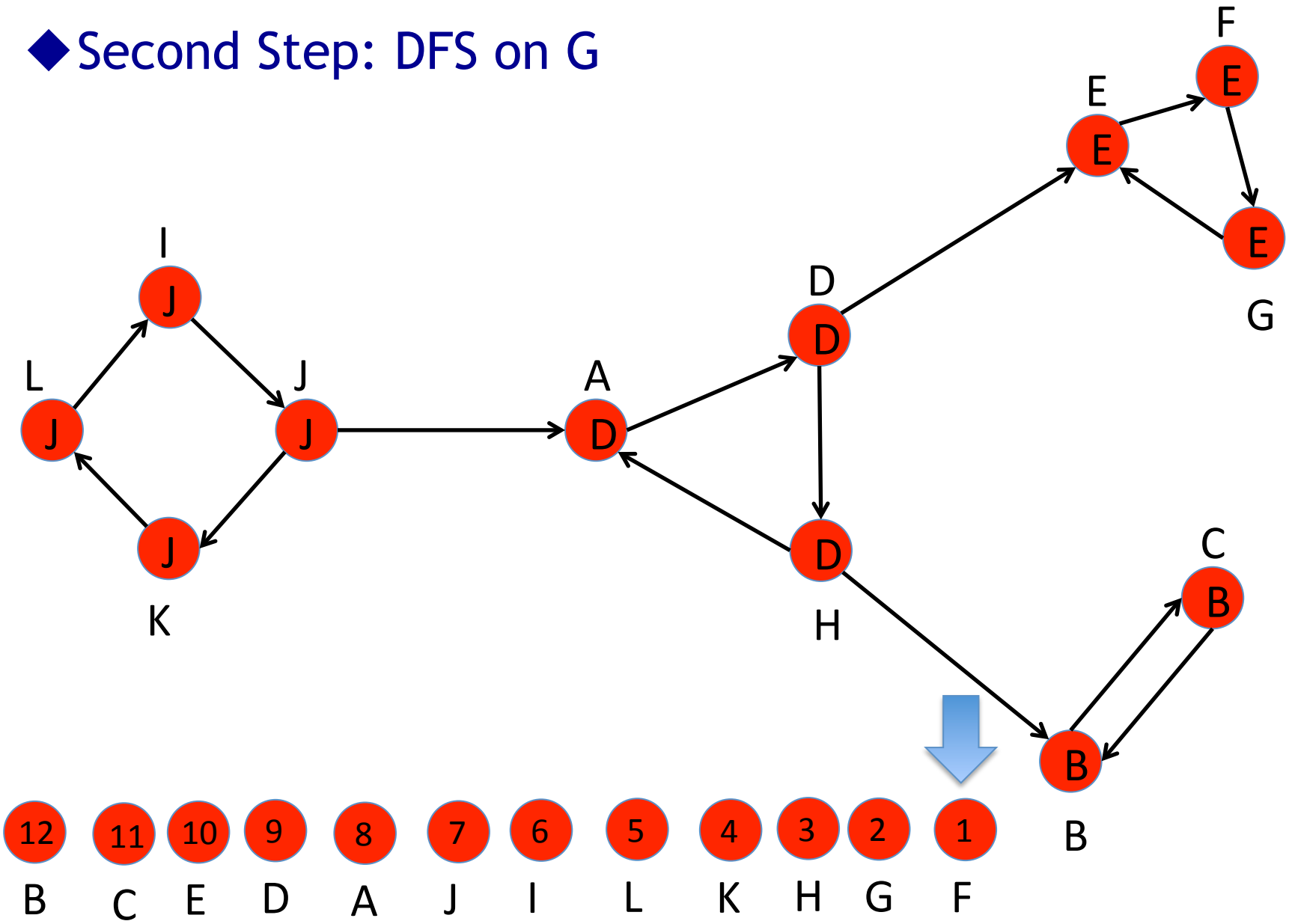
Kosaraju's Algorithm Simulation

- Second Step: DFS on G



Kosaraju's Algorithm Simulation

- Second Step: DFS on G



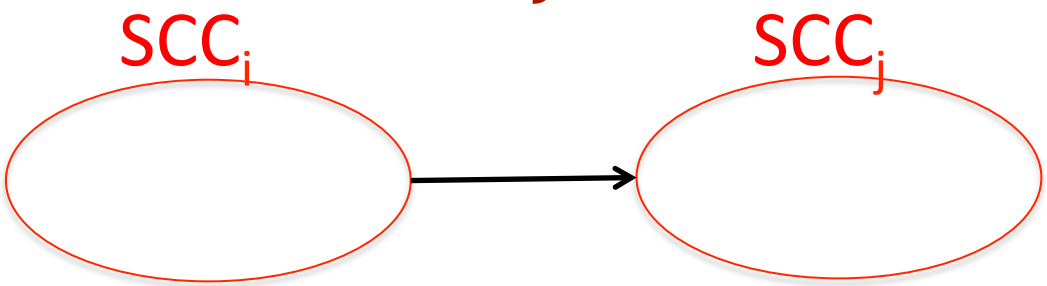
Kosaraju's Algorithm Analysis

- *Run-time: O(n + m)!*
- *Correctness Intuition:*

Recall Key Lemma: $SCC_i \rightarrow SCC_j$, then
$***f[SCC_i] > f[SCC_j]***$
Corallary: In G_{rev}, the highest f[v] will be in a source SCC => In G v will be in a sink SCC
⇒ A BFS/DFS from v will only identify v's SCC
⇒ Once v's SCC is "peeled-off" next highest f[v`] also has to be a sink by induction
⇒ A BFS/DFS from v` will only identify v`s SCC
⇒ Inductively finding all SCCs

Proof of Key Lemma:

In G^{SCC} if $SCC_i \rightarrow SCC_j$, then $f[SCC_i] > f[SCC_j]$



Let u be the first vertex visited in SCC_i or SCC_j

Again, break into 2 cases by whether u is in SCC_i or SCC_j

Case 1: u is in SCC_i. Then:

by extended key lemma 1 (EKL1) of Topological Sort

f[u] will be greater than any vertex in SCC_j

Case 2: u is in SCC_j. Then by (EKL1):

$f[u] > f[y]$ for any vertex in SCC_j

$f[u] < f[x]$ for any vertex in SCC_i

$\Rightarrow f[SCC_i] > f[SCC_j]$