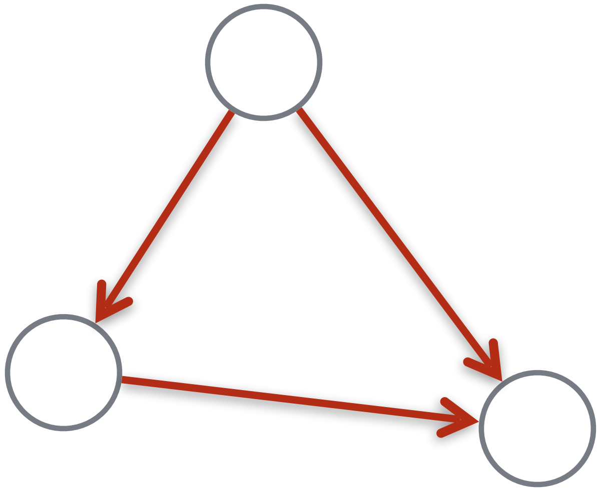
Find Strongly Connected Components Using Kosaraju's Algorithm

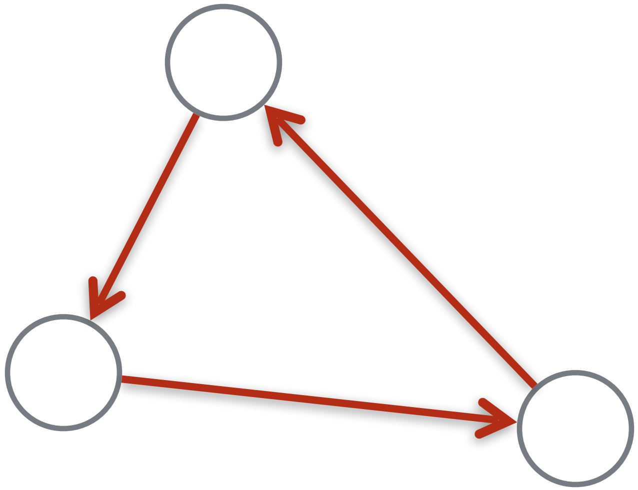
Outline

- What is strongly connected components (SCC) ?
- 2 properties of SCC.
- How Kosaraju's Algorithm works.
- Complexity of Kosaraju's Algorithm.
- How 's Tarjan's Algorithm works.
- Complexity of Tarjan's Algorithm.

What is strongly connected components?

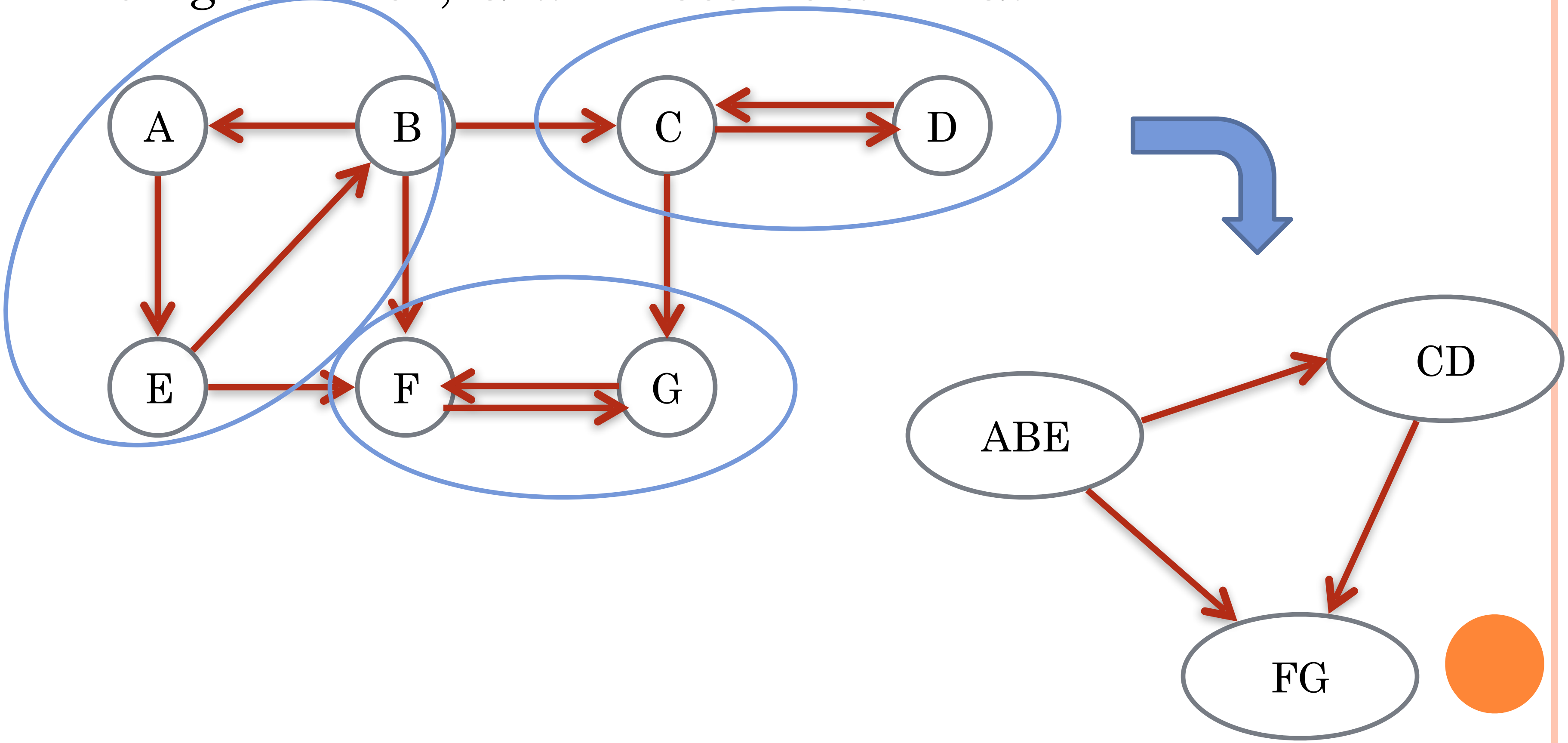
- Definition: A strongly connected components (SCC) of a directed graph G = (V, E) is a maximal set of vertices $C \subseteq V$, such that every vertices in C is **reachable from each other.**
- Not strongly connected: Strongly connected:





Property(I) of strongly connected components

- If we compress each SCC set in the graph G to a single vertex, G will become a DAG.



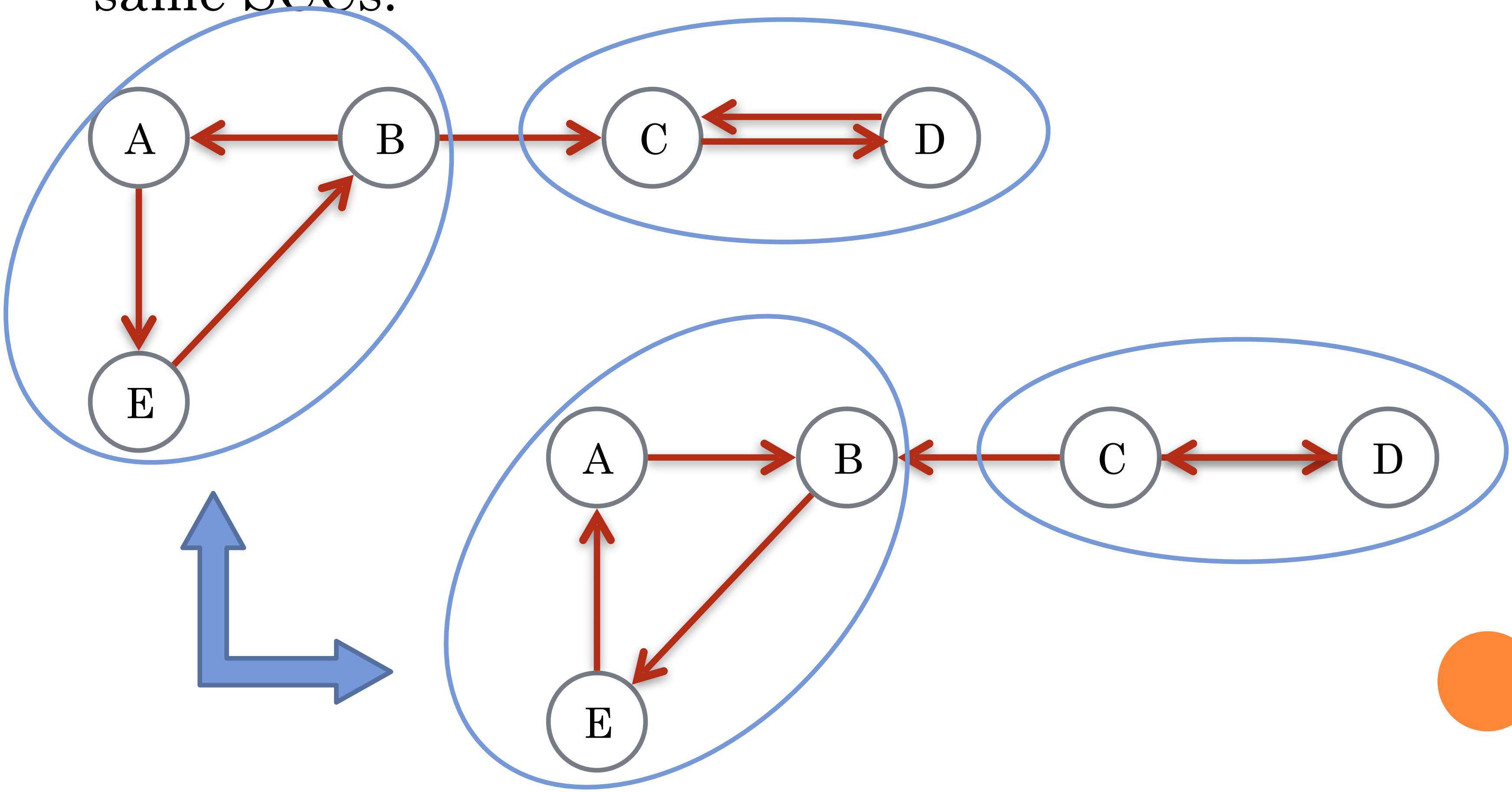
PROPERTY(I) OF STRONGLY CONNECTED COMPONENTS

- The compressed SCC graph is a DAG.
- Lemma 22.13: Let C and C' be distinct SCCs in directed graph G. Let u, v ∈ C; u', v' ∈ C'. If G contains a path (u, u'), then G cannot also contain a path (v', v).
- Proof:

1. If G contains both (u, u') and (v', v), then C and C' is reachable from each other.
2. Because vertices in a SCC set is reachable from each other, every vertex in C and C' will be reachable from each other.
3. Thus, C and C' are not distinct SCC => contradiction.

Property(II) of strongly connected components

- Graph G and its transpose G^T have exactly the same SCCs.



Property(II) of strongly connected components

- Informal proof.
- A SCC set → vertices within it construct a cycle.
- Reverse the edges of a cycle → still a cycle.
- According to property I, we can transform graph G in to a DAG.
- Reverse the edges of a DAG → still a DAG.
- So the SCC sets of G is the same as G^T's.

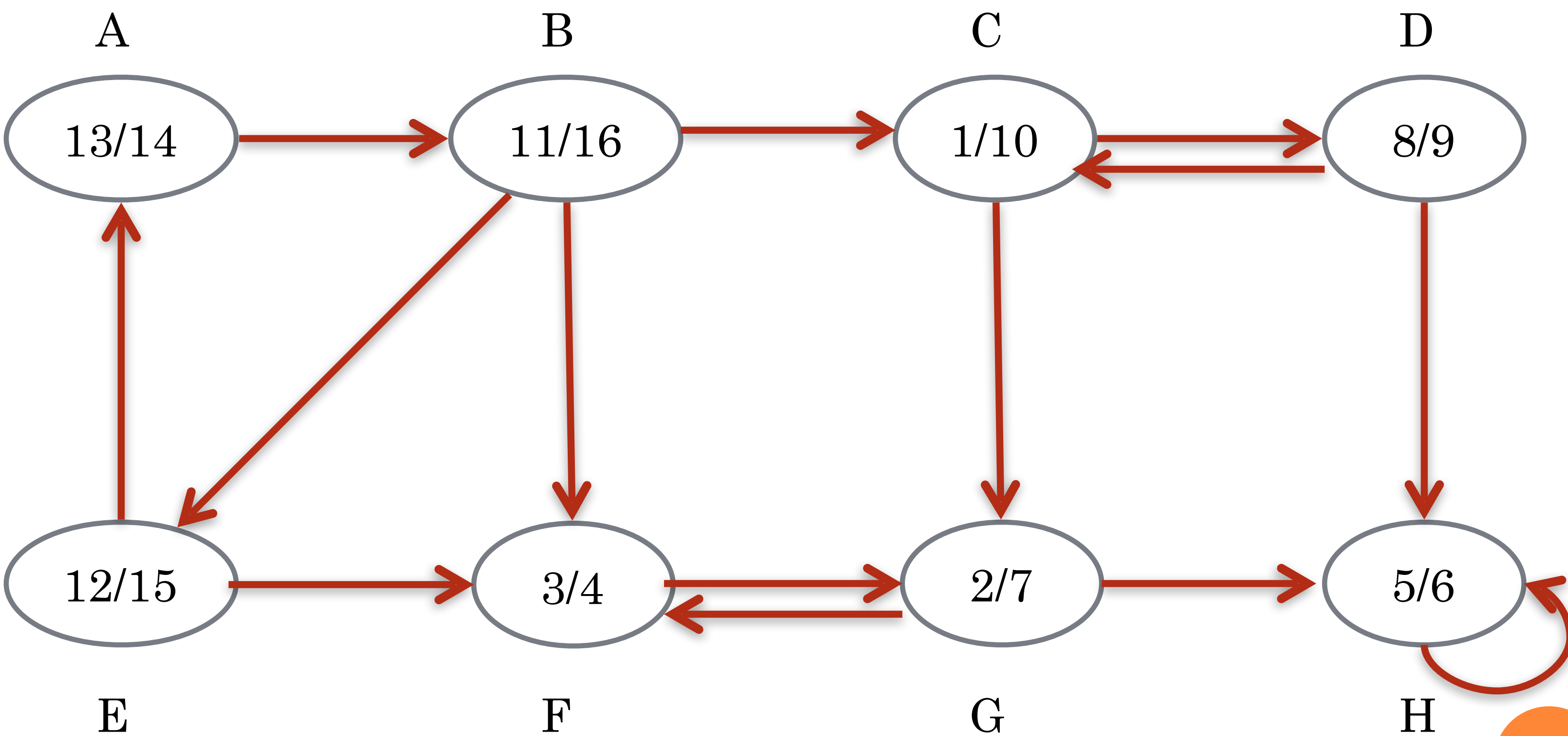
Kosaraju's Algorithm

- Make use of the second property of SCC.
- Kosaraju(G)

1. Run DFS(G) to compute **finishing times u.f** for each vertex u.
2. Compute G^T.
3. Run DFS(G^T), choose the vertices with the **decreasing order** of u.f calculated in step 1.
4. Put every vertex visited during each DFS iteration into current SCC set.
5. When current DFS iteration comes to an end, put current SCC set into SCC list, and start next DFS iteration from another unvisited vertex.
6. After all vertices are visited, we get a complete SCC list.

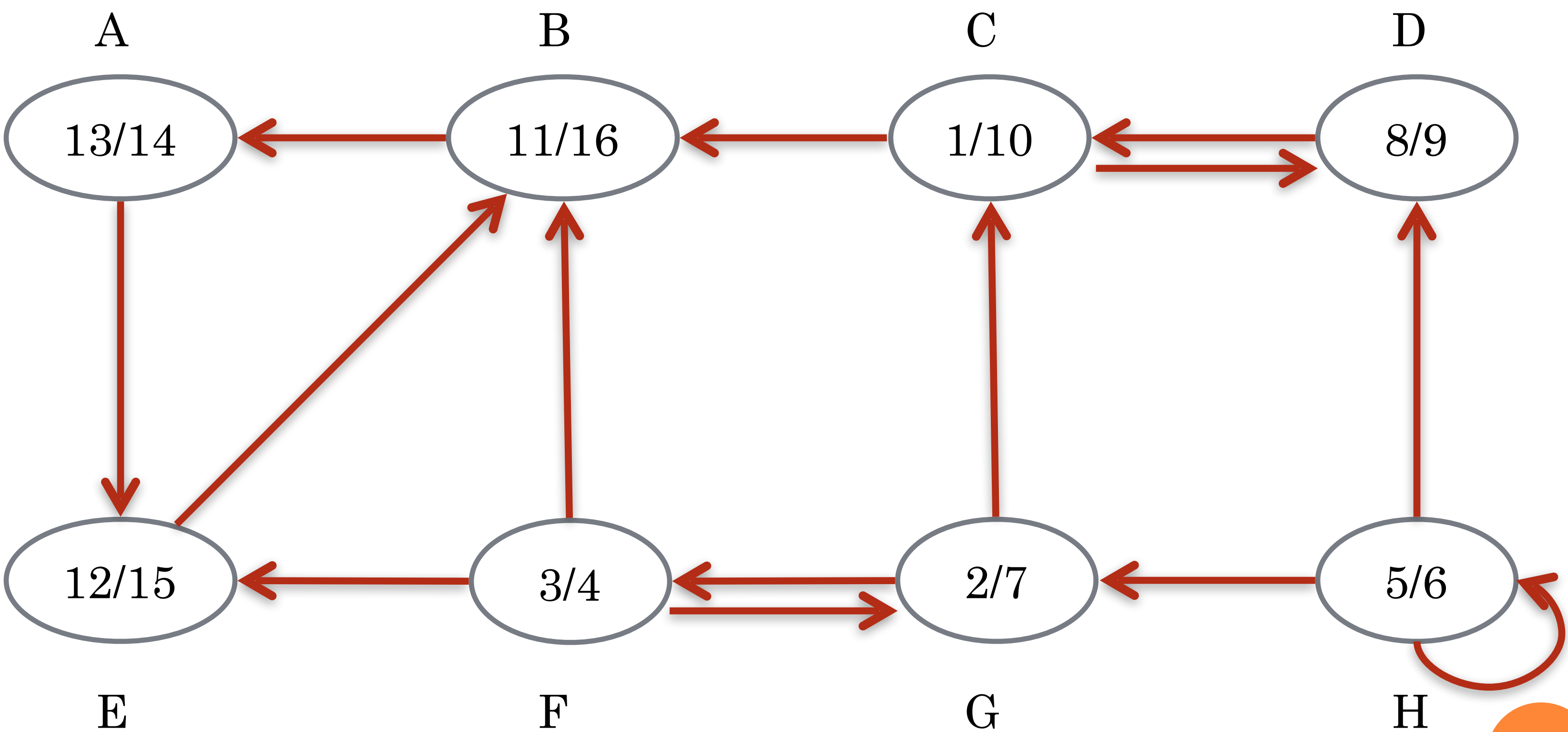
Kosaraju's Algorithm

- Run DFS(G).



Kosaraju's Algorithm

- Compute G^T.

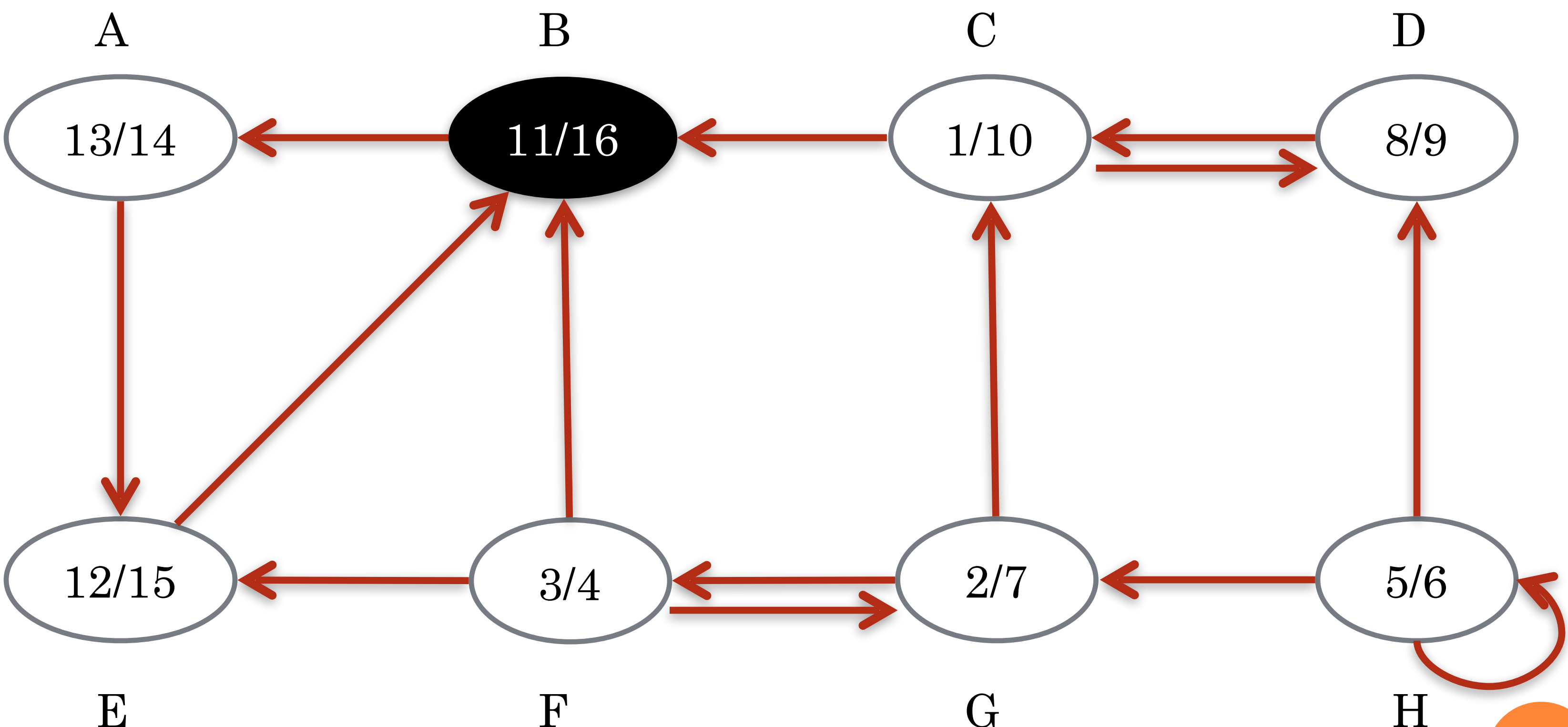


Kosaraju's Algorithm

- Run DFS(G^T).
- Start from B

Present SCC set: {B}

Present SCC list: NULL

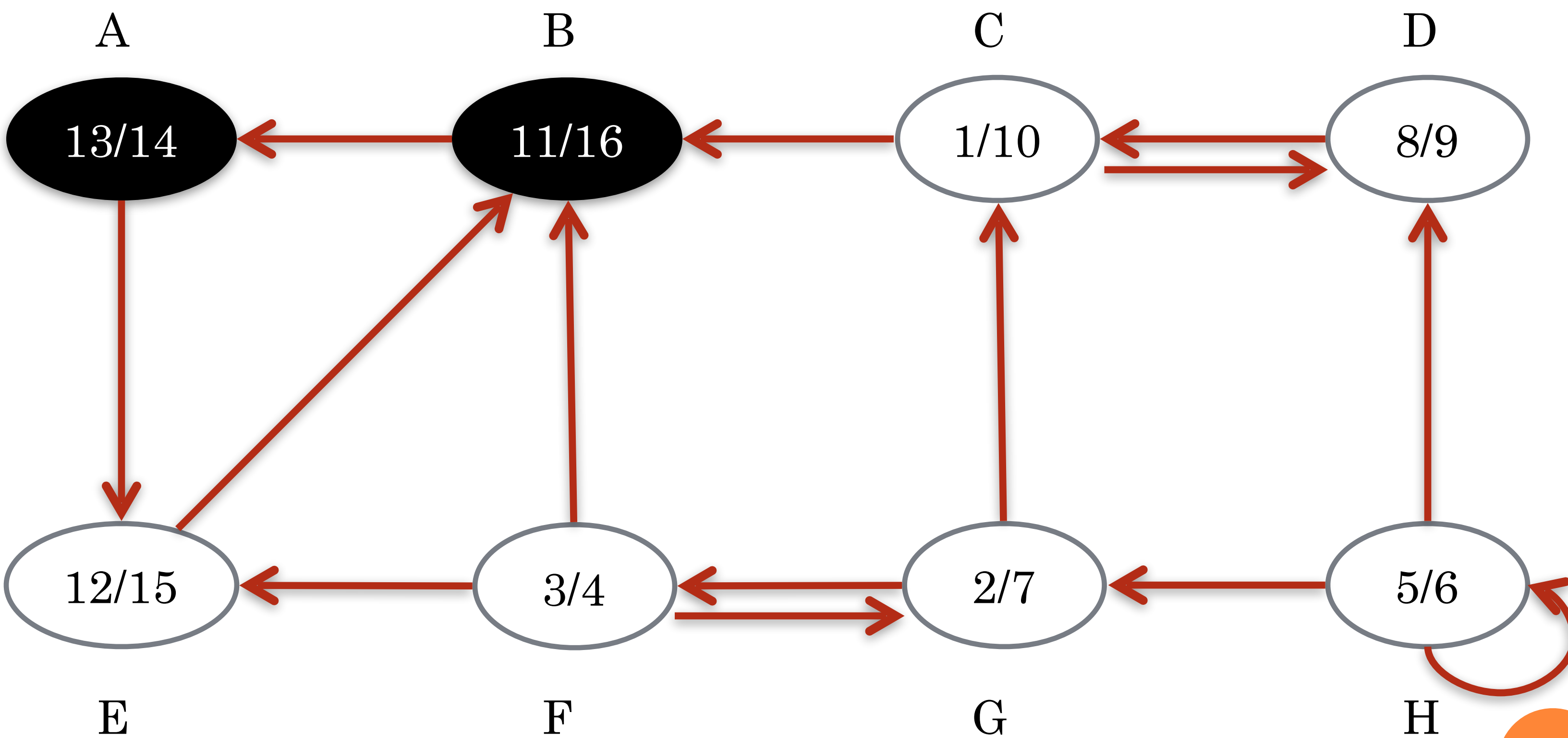


Kosaraju's Algorithm

- Run DFS(G^T).

Present SCC set: {B, A}

Present SCC list: NULL

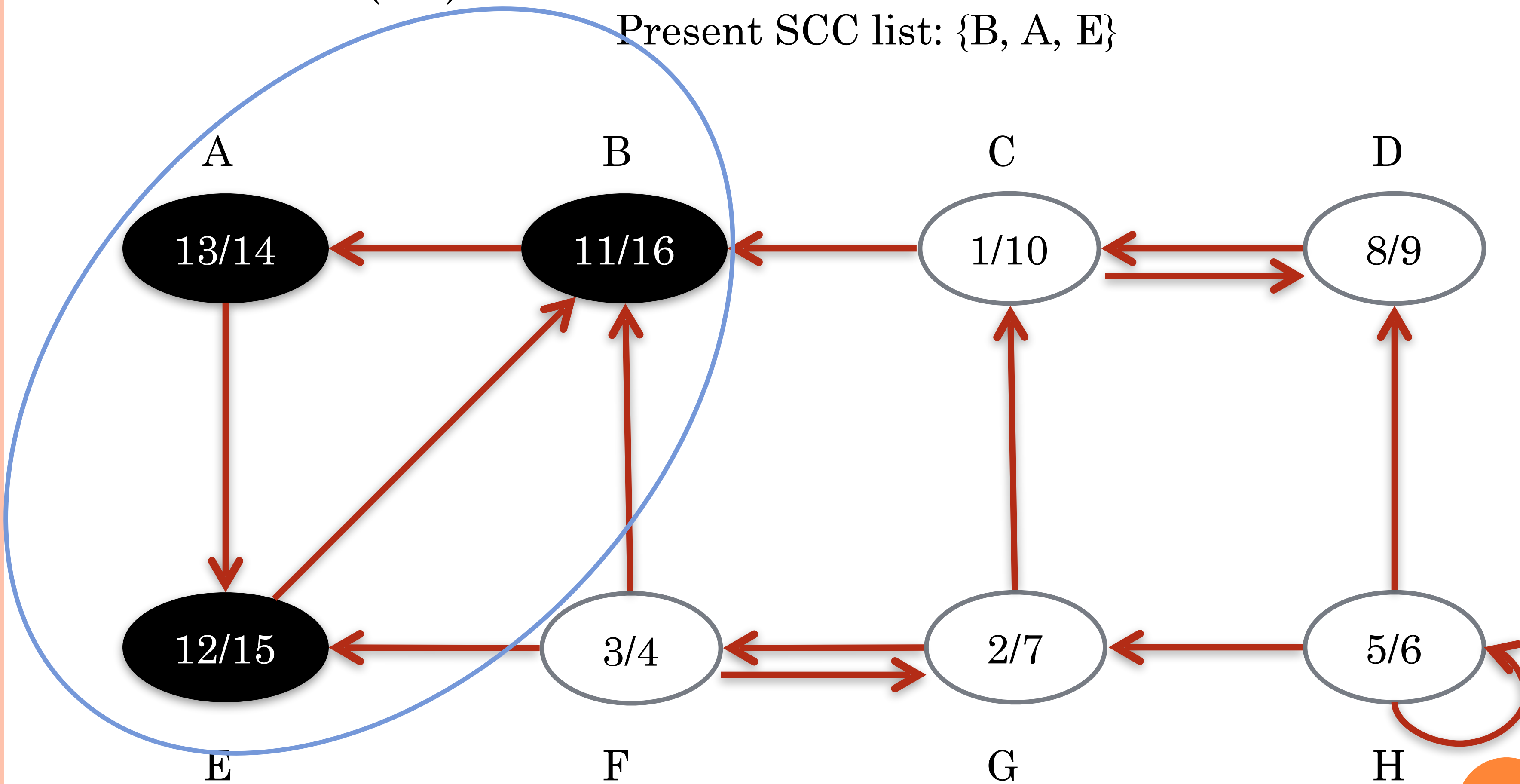


KOSARAJU'S ALGORITHM

- Run DFS(G^T).

Present SCC set: {B, A, E}

Present SCC list: {B, A, E}

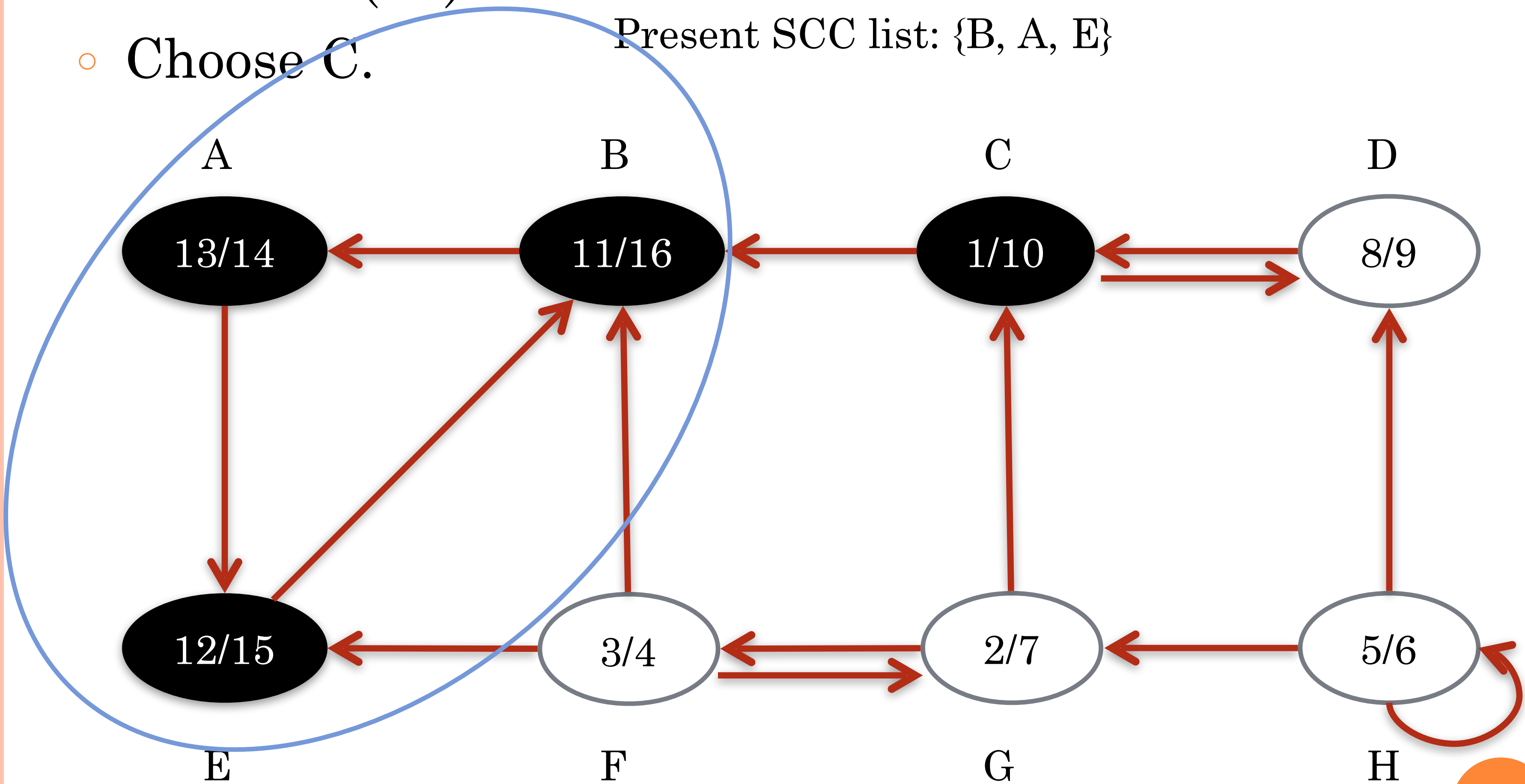


Kosaraju's Algorithm

- Run DFS(G^T).
- Choose C.

Present SCC set: {C}

Present SCC list: {B, A, E}

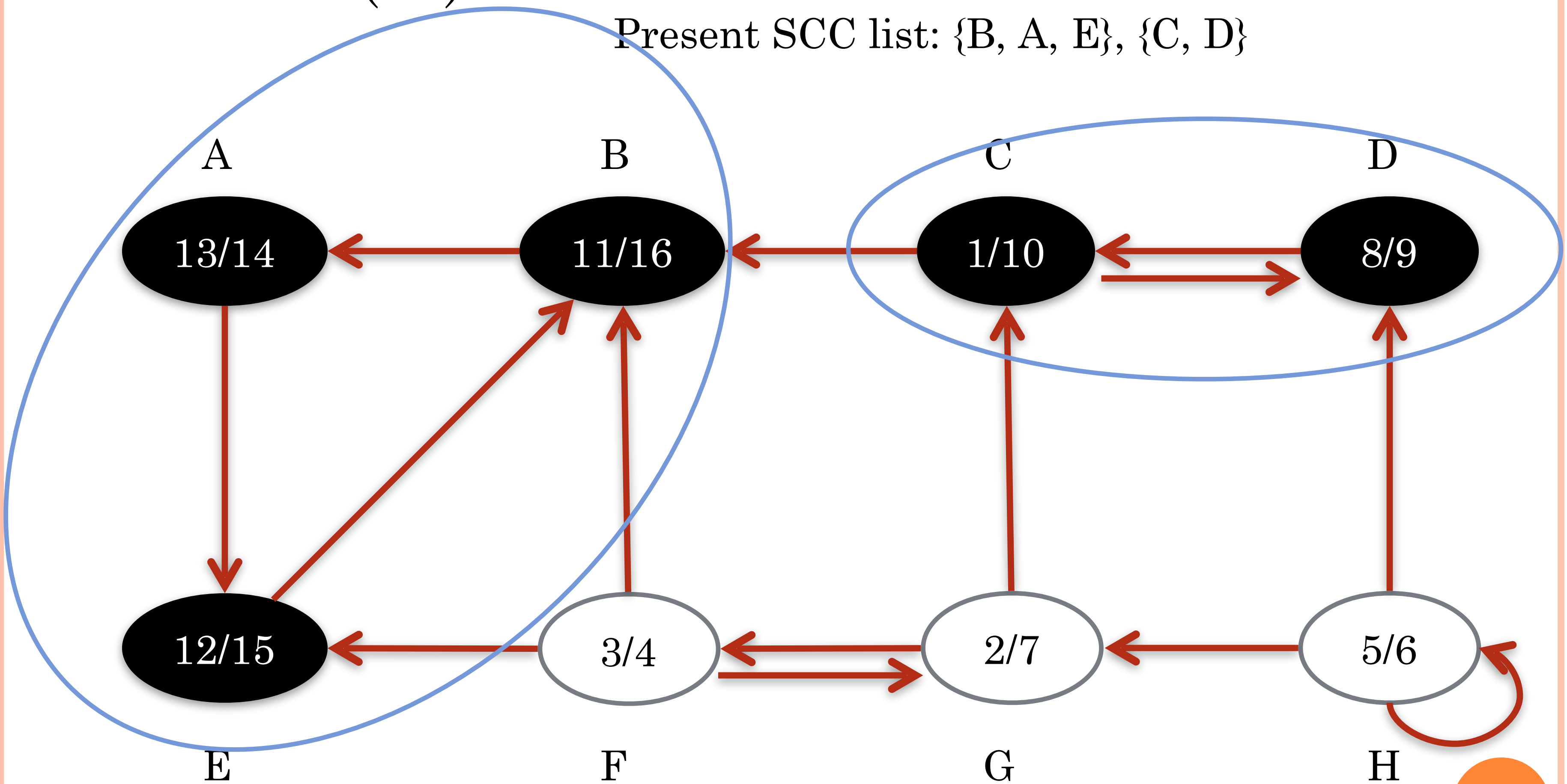


Kosaraju's Algorithm

- Run DFS(G^T).

Present SCC set: {C, D}

Present SCC list: {B, A, E}, {C, D}

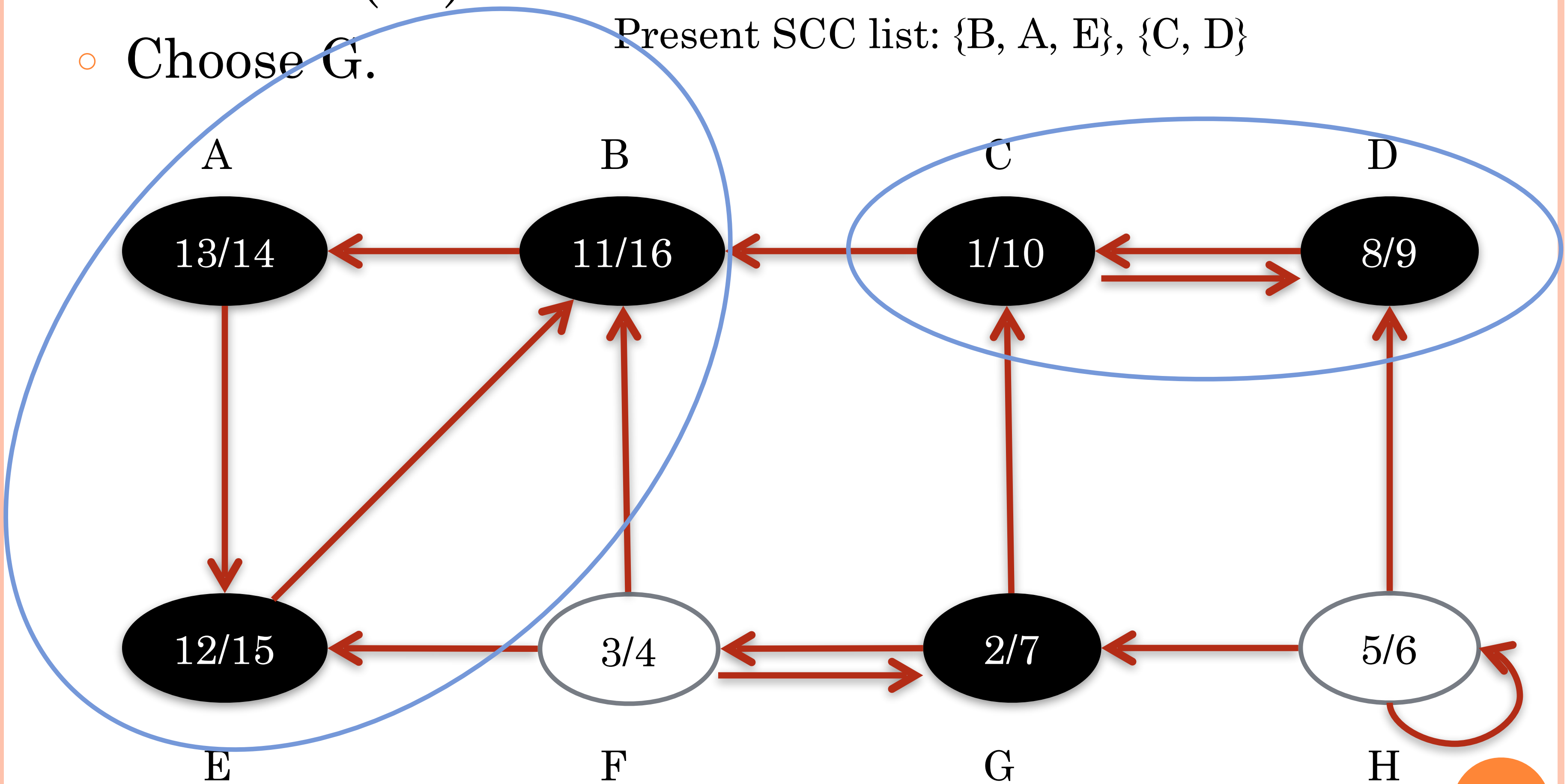


Kosaraju's Algorithm

- Run DFS(G^T).
- Choose G.

Present SCC set: {G}

Present SCC list: {B, A, E}, {C, D}

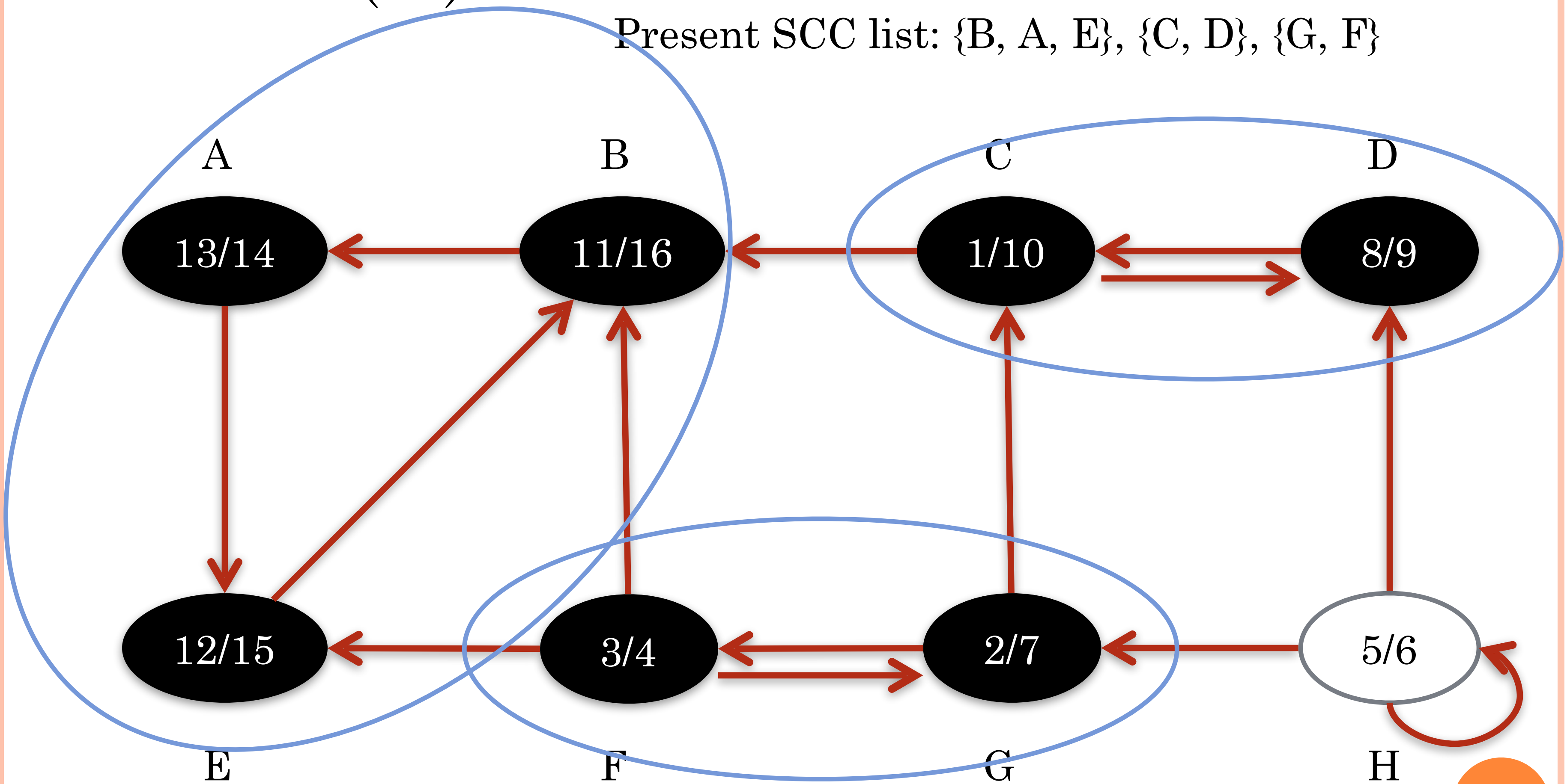


Kosaraju's Algorithm

- Run DFS(G^T).

Present SCC set: {G, F}

Present SCC list: {B, A, E}, {C, D}, {G, F}

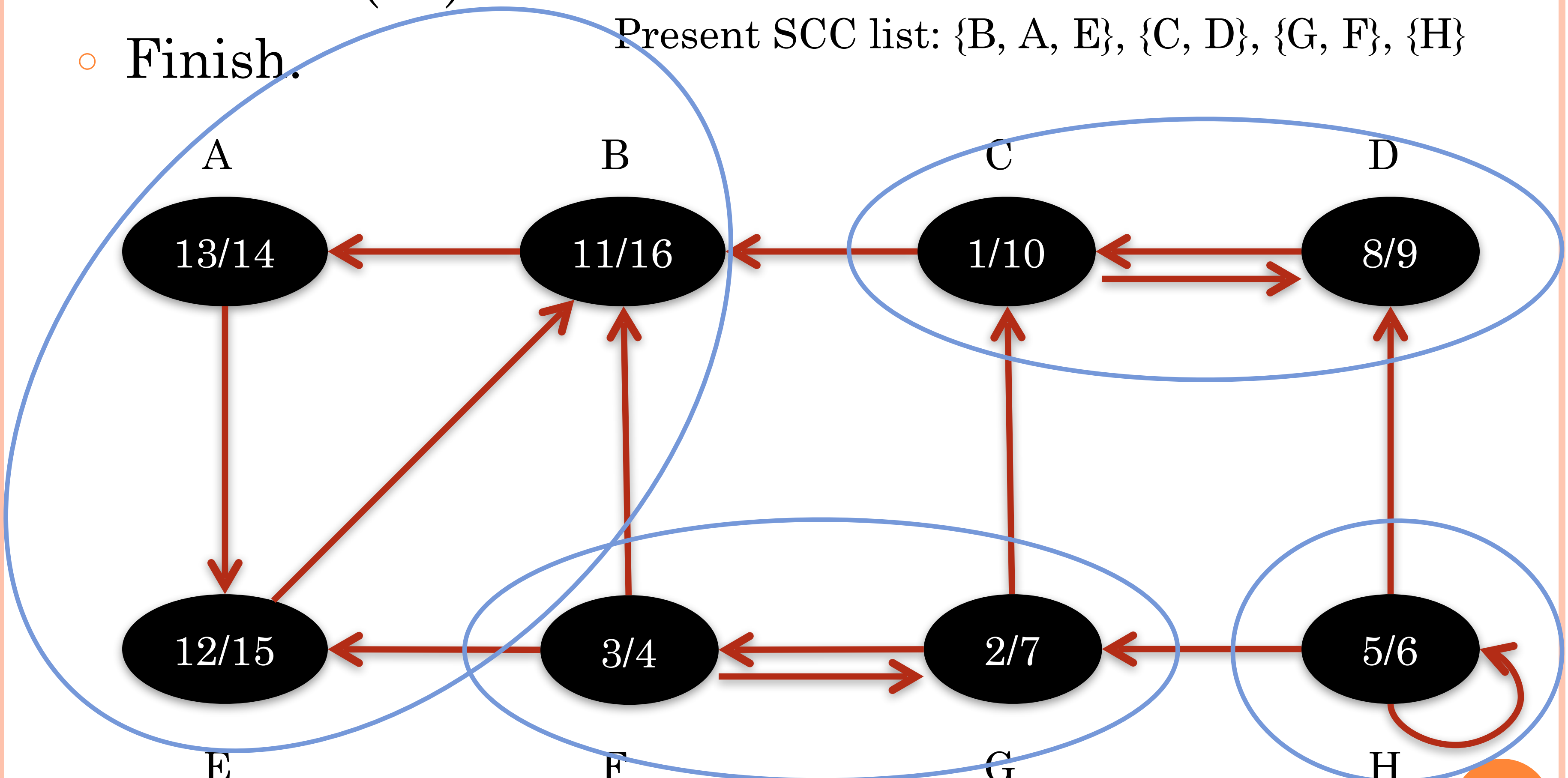


Kosaraju's Algorithm

- Run DFS(G^T).
- Finish.

Present SCC set: {H}

Present SCC list: {B, A, E}, {C, D}, {G, F}, {H}



Why Kosaraju's Algorithm Works?

- According to Corollary 22.15: Each edge in the transpose of G that goes between different SCC goes **from a vertex with an earlier finishing time to one with a later finishing time**.
- According to **property I**, we can compress the graph into a **DAG**.
- So, if we choose the vertex in the transpose of G to start DFS in the order of decreasing finishing time, we are actually visiting the DAG of SCC in the **reverse topological order**.

Why Kosaraju's Algorithm Works?

- Because there is a two-way path between any pair of vertices in a SCC.
- We can traverse all the vertices in a SCC by DFS in any order.
- Because we visit the DAG of SCC in the reverse topological order.
- No matter how we traverse all vertices within a SCC by DFS, we won't accidentally go into other SCC. The vertices we can visit by one time of DFS are constrained to the same SCC set.
- Remember to delete the whole SCC set from the graph once it is visited.

Complexity of Kosaraju's Algorithm

- DFS x 2, G transpose x 1
- Analysis:

1. Do DFS(G) first time. => $\Theta(V+E)$ / $\Theta(V^2)$
2. Transpose G. => $\Theta(E)$ / 0
3. Do DFS(G^T) and produce SCC sets. => $\Theta(V+E)$ / $\Theta(V^2)$

- Using adjacency list: $\Theta(V+E)$
- Using adjacency matrix: $\Theta(V^2)$