COMP9312 Reachability Queries

Outline

- Reachability
- Transitive closure
- Tree cover
- Implementation

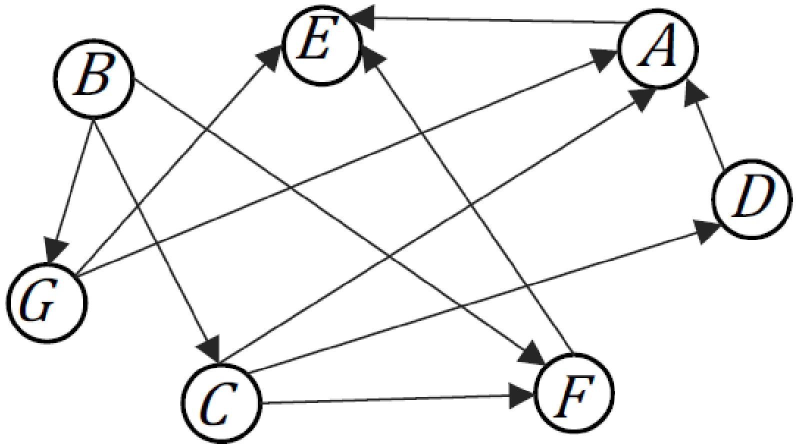
Reachability

Main idea

- Given a directed graph and two vertices u.and v, a reachability query asks for if there exists a path from u to v.

Reachability

- Use BFS to answer the following queries:
  - Can G reach B?
  - Can C reach A?
  - Can E reach F?

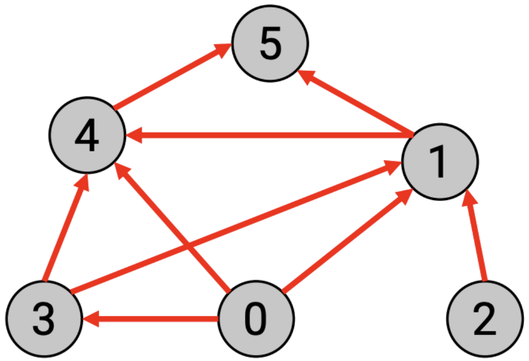


What is the time complexity for query processing?

Transitive closure

Definition

- A transitive closure is a Boolean matrix storing the answers of all possible reachability queries. The size of the matrix is $O(n^2)$.



The original graph G

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

The transitive closure of G

Transitive closure

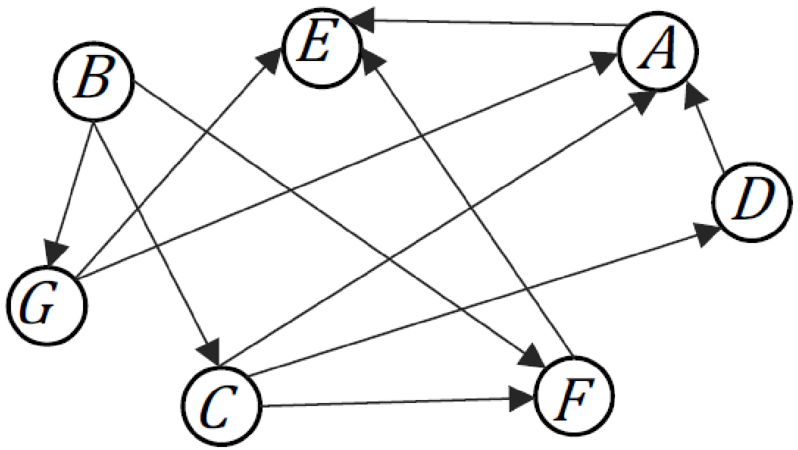
- Floyd-Warshall Algorithm

```
bool tc[num_vertices][num_vertices];

// Initialize the matrix tc:  O(n^2)
tc[i][j] = 1 if there is an edge from i to j, or i == j;

// Run Floyd-Warshall
for ( int k = 0; k < num_vertices; ++k ) {
    for ( int i = 0; i < num_vertices; ++i ) {
        for ( int j = 0; j < num_vertices; ++j ) {
            tc[i][j] = tc[i][j] || (tc[i][k] && tc[k][j]);
        }
    }
}
```

Idea: After the iteration k, find the reachability pairs(i,j) where the reachability path is formed by $\{v_0, v_1, \cdots, v_k\}$

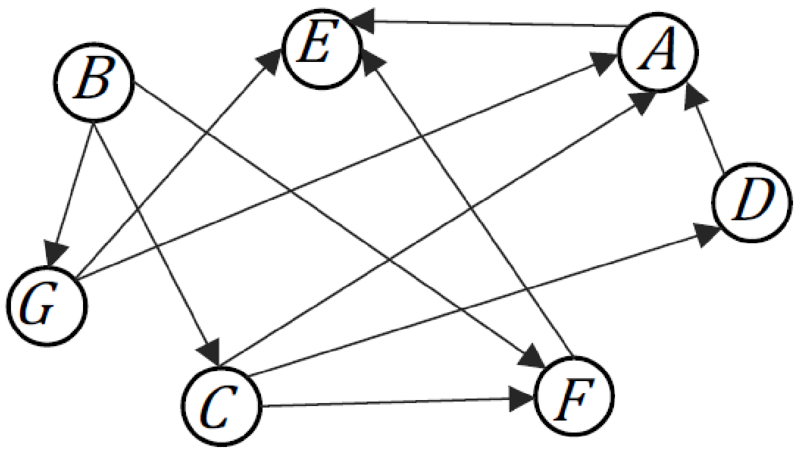
Transitive closure

- Construct the transitive closure for graph G.
- Use it to answer the reachability in Ex1.2
- What is the time/space complexity for answering queries in this case?



Now, you have 3 minutes to do Ex2.

Transitive closure

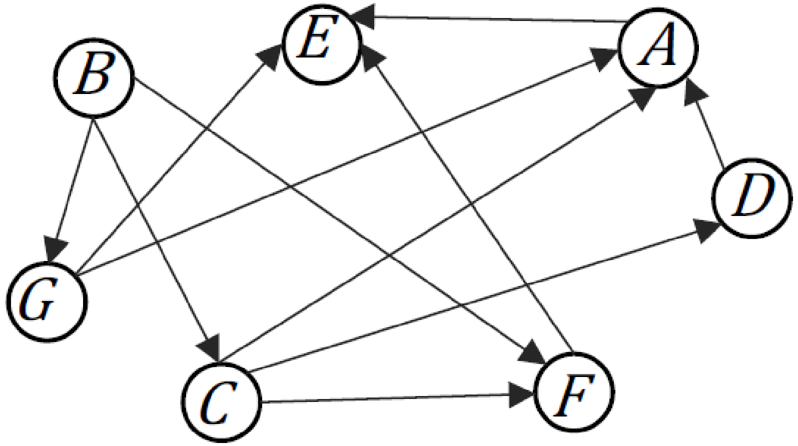


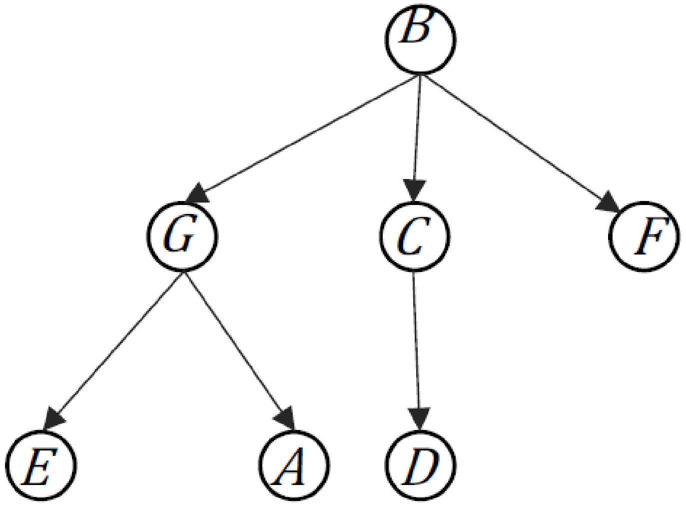
	A	B	C	D	E	F	G
A	1	0	0	0	1	0	0
B	1	1	1	1	1	1	1
C	1	0	1	1	1	1	0
D	1	0	0	1	1	0	0
E	0	0	0	0	1	0	0
F	0	0	0	0	1	1	0
G	1	0	0	0	1	0	1

- Can G reach B? False
- Can C reach A? True
- Can E reach F? False

Tree cover

- Find a spanning tree of example graph G

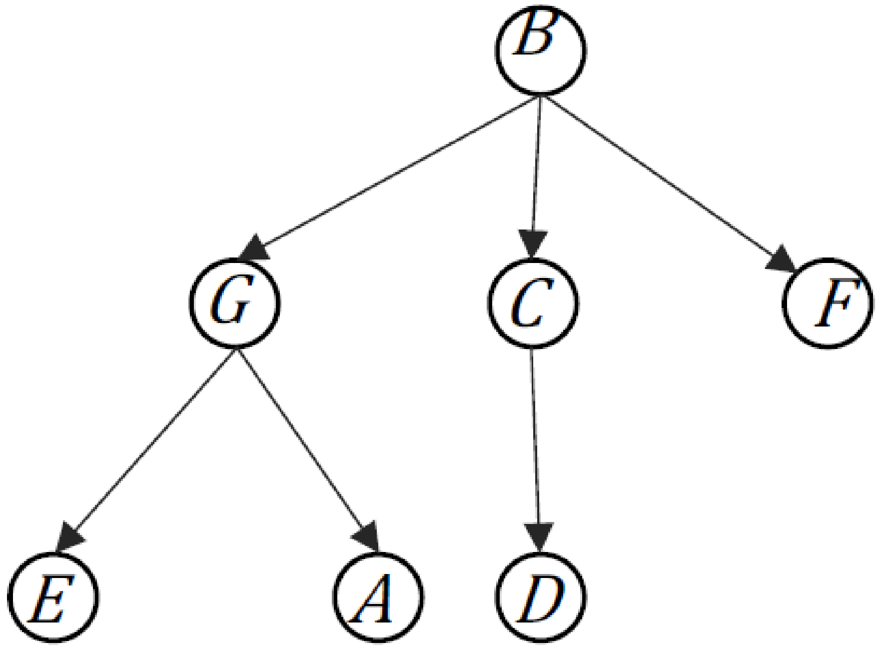




One possible tree cover

Tree cover

- **Algorithm1:**
  1. Find a spanning tree (tree cover) T of G
  2. Assign post-order numbers and indices as intervals to the nodes of T
  3. Go through vertices in reverse topological order. For each processed vertex q, consider all its in-edges (p, q). Add the intervals of q to the interval of p. If any interval is subsumed, discard it.



post-order-traversal(root):

for each v of root's children from left to right:

// traverse the subtree rooted at v

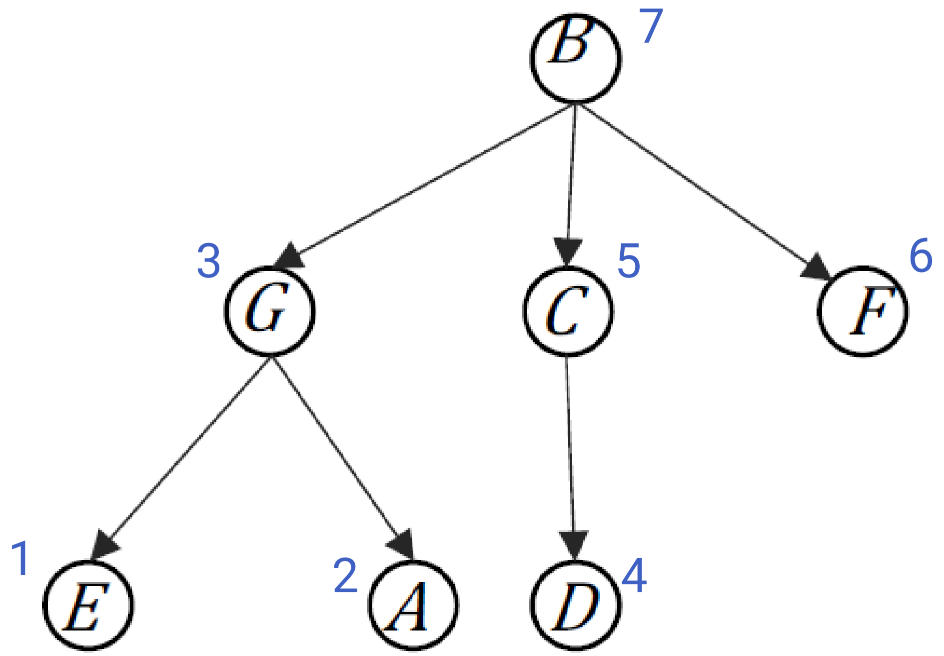
post-order-traversal(v)

visit root

Now, you have 5 minutes to do Ex3.2.

Tree cover

- **Algorithm1:**
  1. Find a spanning tree (tree cover) T of G
  2. **Assign post-order numbers and indices as intervals to the nodes of T**
  3. Go through vertices in reverse topological order. For each processed vertex q, consider all its in-edges (p, q). Add the intervals of q to the interval of p. If any interval is subsumed, discard it.



post-order-traversal(root):

for each v of root's children from left to right:

// traverse the subtree rooted at v

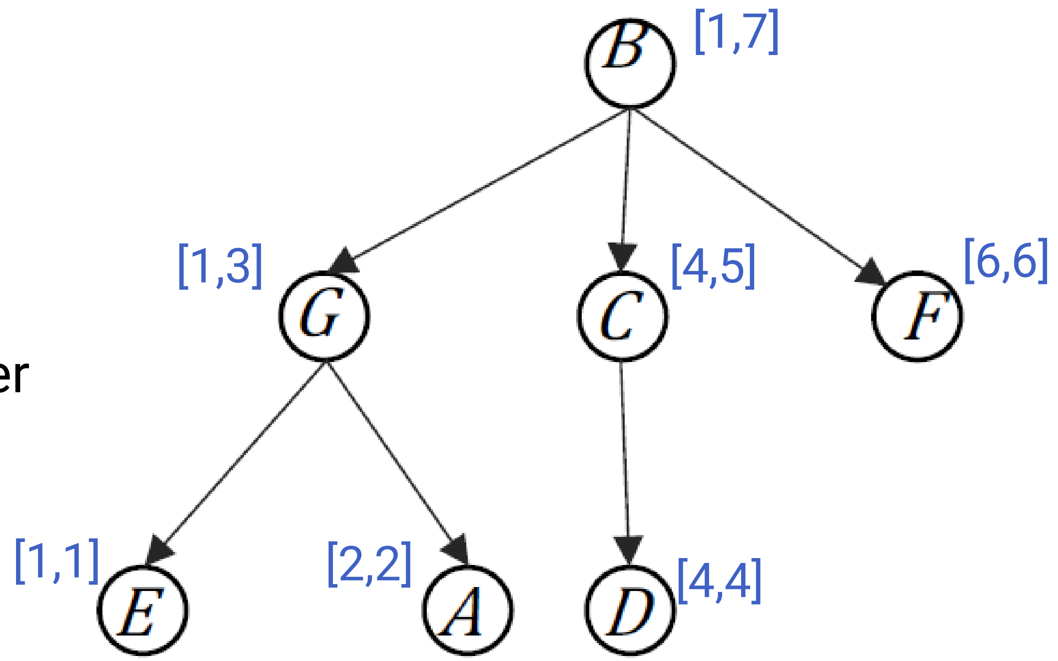
post-order-traversal(v)

visit root

A:2, B:7, C:5, D:4, E:1, F:6, G:3

Tree cover

- **Algorithm1:**
  1. Find a spanning tree (tree cover) T of G
  2. **Assign post-order numbers and indices as intervals to the nodes of T**
  3. Go through vertices in reverse topological order. For each processed vertex q, consider all its in-edges (p, q). Add the intervals of q to the interval of p. If any interval is subsumed, discard it.

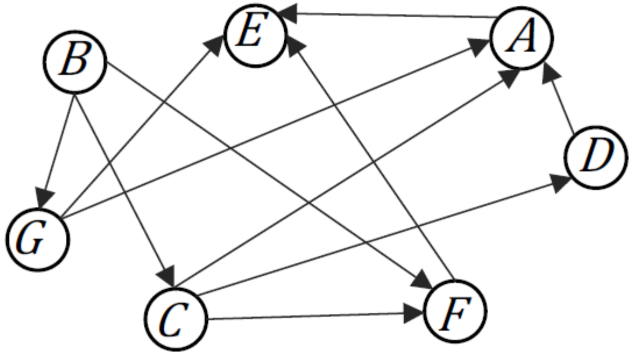


For each vertex, compute the minimum post-order number of its subtree

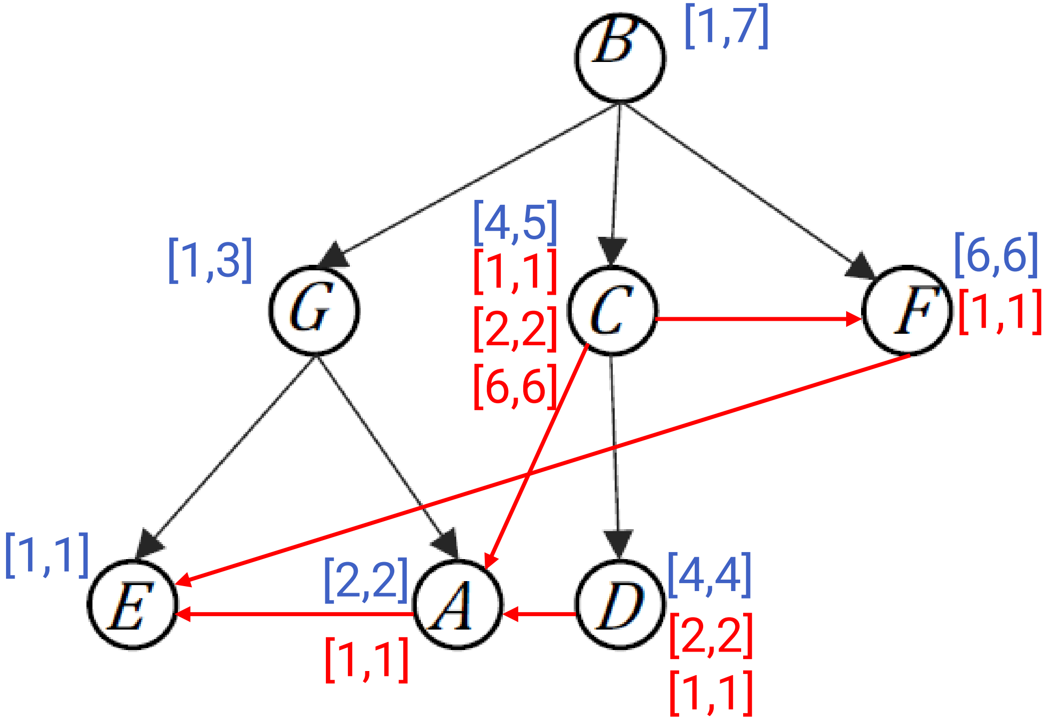
A:2, B:7, C:5, D:4, E:1, F:6, G:3
A:2, B:1, C:4, D:4, E:1, F:6, G:1

Tree cover

- **Algorithm1:**
    1. Find a spanning tree (tree cover) T of G
    2. Assign post-order numbers and indices as intervals to the nodes of T
    3. **Go through vertices in reverse topological order. For each processed vertex q, consider all its in-edges (p, q). Add the intervals of q to the interval of p. If any interval is subsumed, discard it.**



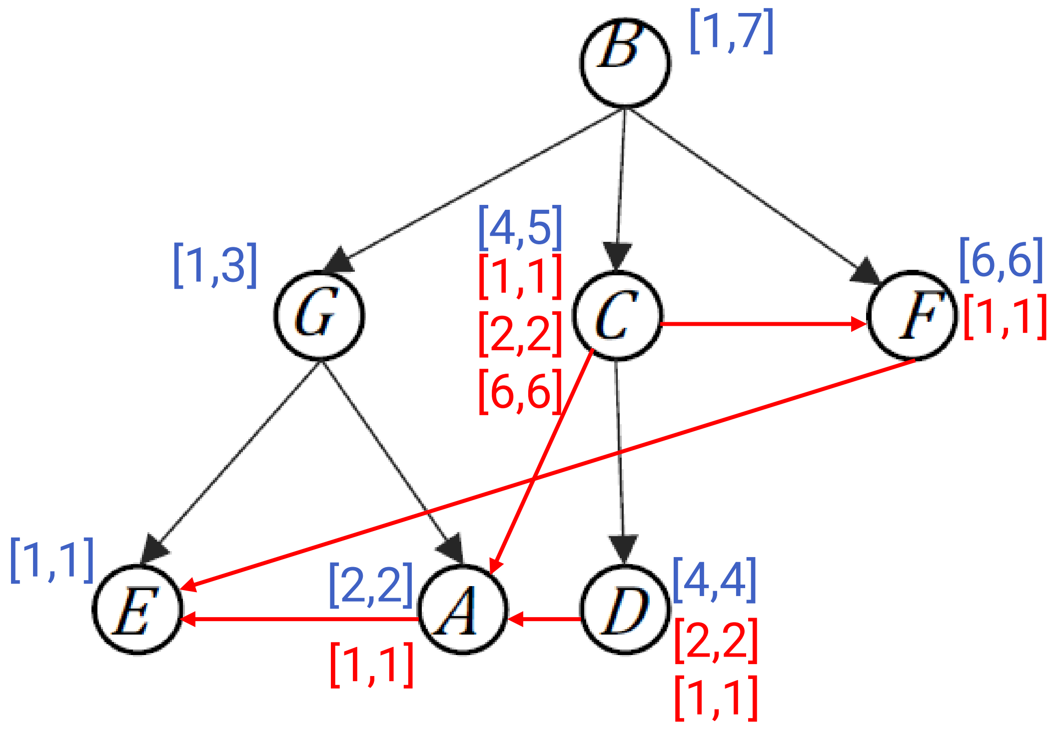
Topological order:
{B,G,C,F,D,A,E}



Number of intervals: 14

Tree cover

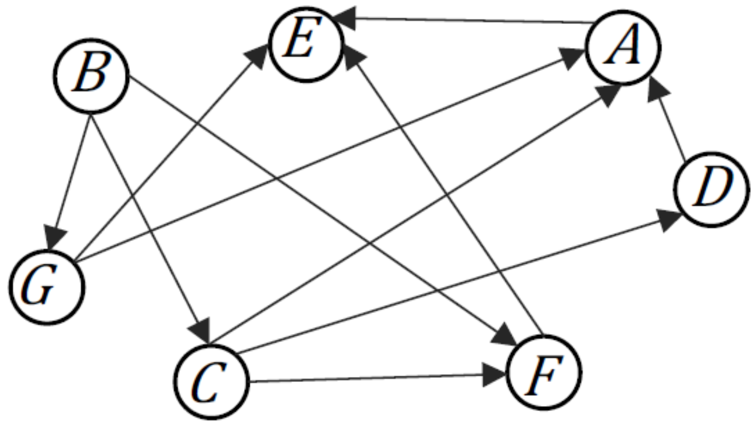
- Query Processing: $?\,(u \rightarrow v\,) \Rightarrow$ if the interval of v is within the updated interval of u.
  - $?\,G \rightarrow B \Rightarrow False$
  - $?\,C \rightarrow A \Rightarrow True$
  - $?\,E \rightarrow F \Rightarrow False$

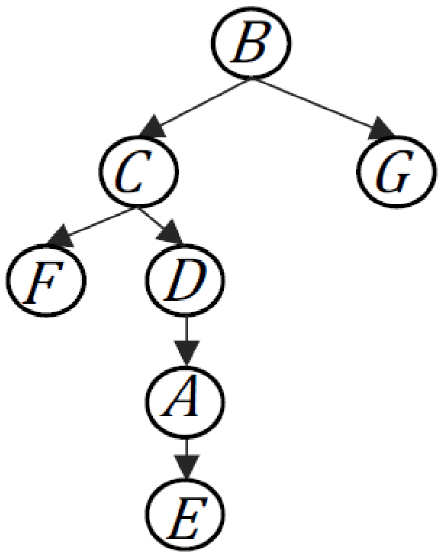


Number of intervals: 14

Tree cover

- **Optimal tree cover**
  The tree cover with the minimum number of intervals in the resulting compression scheme.

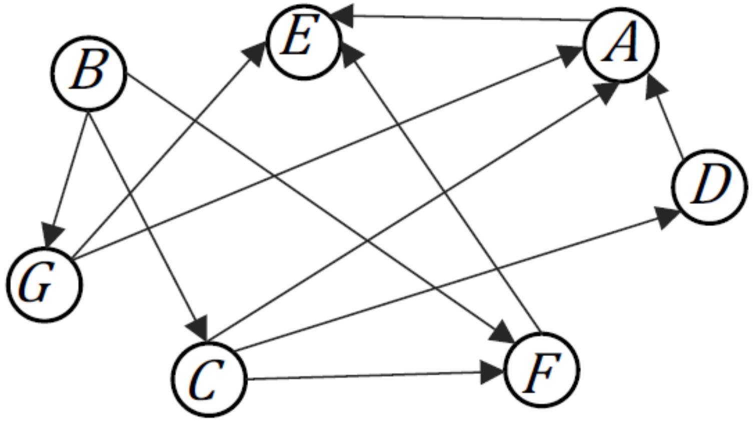


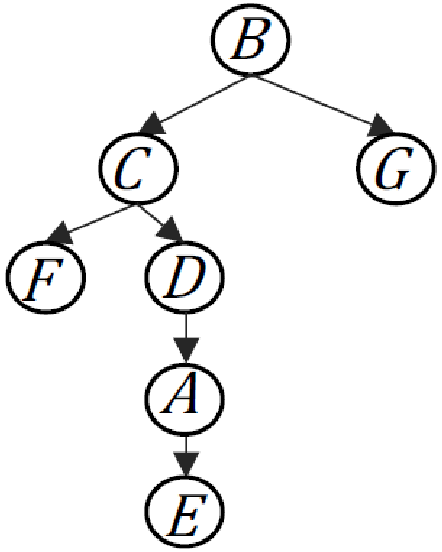


An optimal tree cover

Tree cover

- Repeat the above process for the given optimal tree cover.



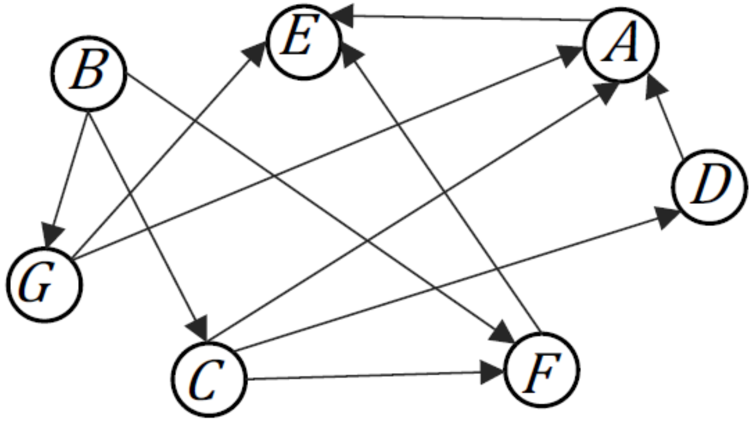


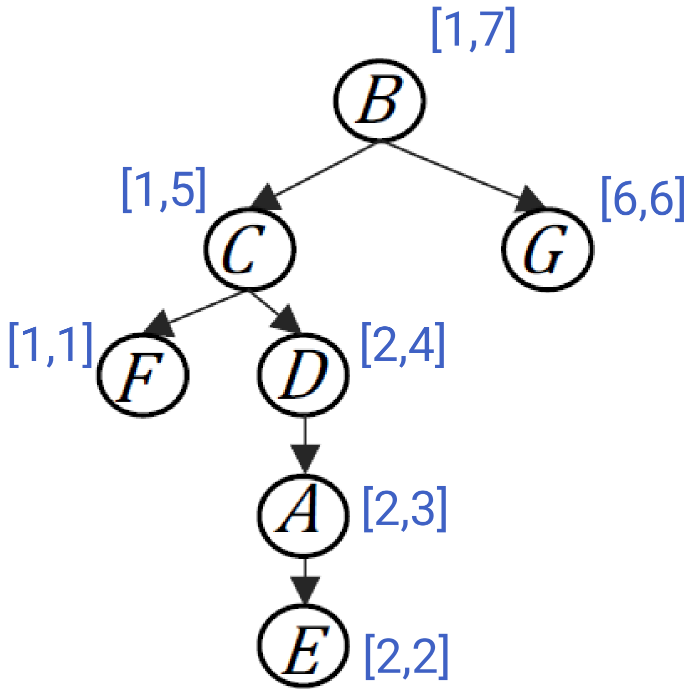
An optimal tree cover

Now, you have 5 minutes to do Ex3.4.

Tree cover

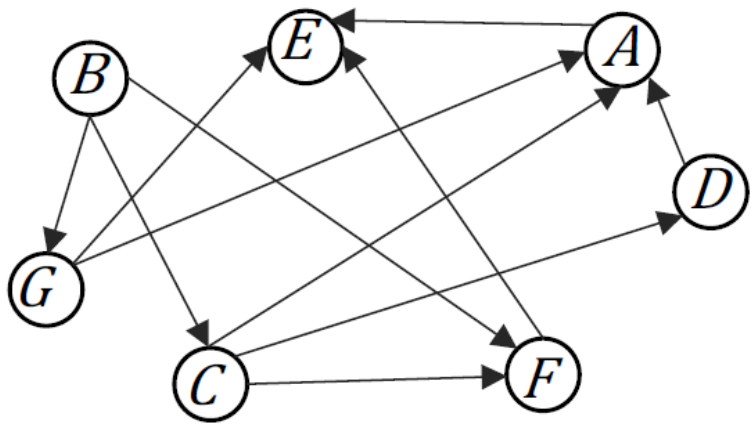
- Repeat the above process for the given optimal tree cover.

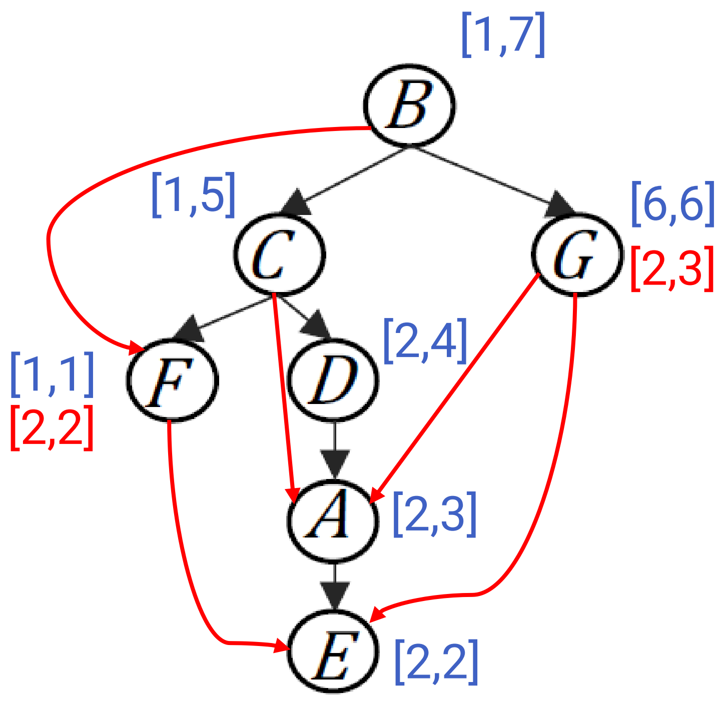




Tree cover

- Repeat the above process for the given optimal tree cover.

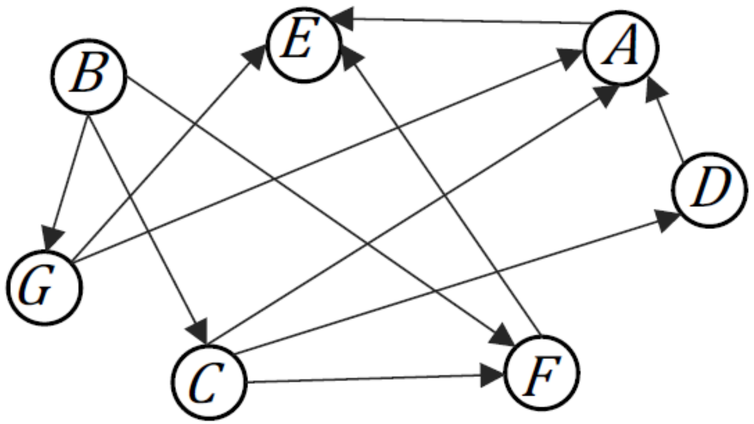




Number of intervals: 9

Implementation

- Load the example graph G via the class ‘DirectedGraph’ in tutorial_5.py.
- Implement the class ‘Reachability’ which inputs the example graph G and answer the queries in Exercise 1.2.
- You can choose one of the algorithms mentioned in lecture.



Now, you have 15 minutes to do Ex4.

Q & A