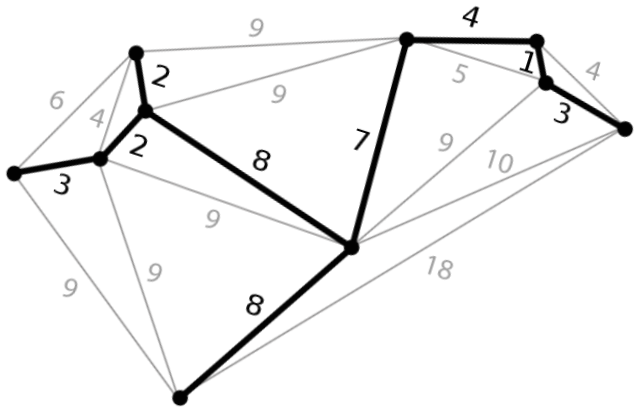
COMP261
Algorithms and Data Structures
2024 Tri 1

Jyoti Sahni
jyoti.sahni@ecs.vuw.ac.nz
Office Hours (COMP261): AM414, Thursday 10:00 – 12:00

Spanning Trees

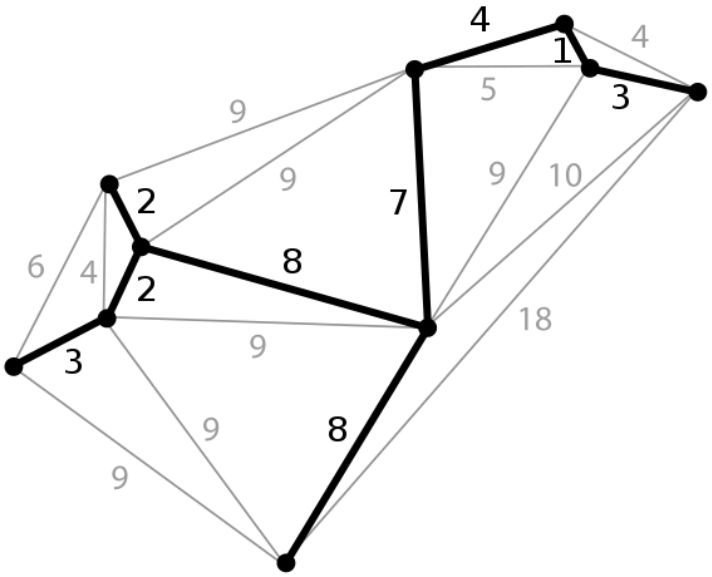
Given a connected, undirected, weighted graph, a spanning tree is a subgraph that contains all the nodes but has no cycles (is a tree)

A spanning tree is defined only for a connected graph, because a tree is always connected, and in a disconnected graph of n vertices we cannot find a connected subgraph with n vertices

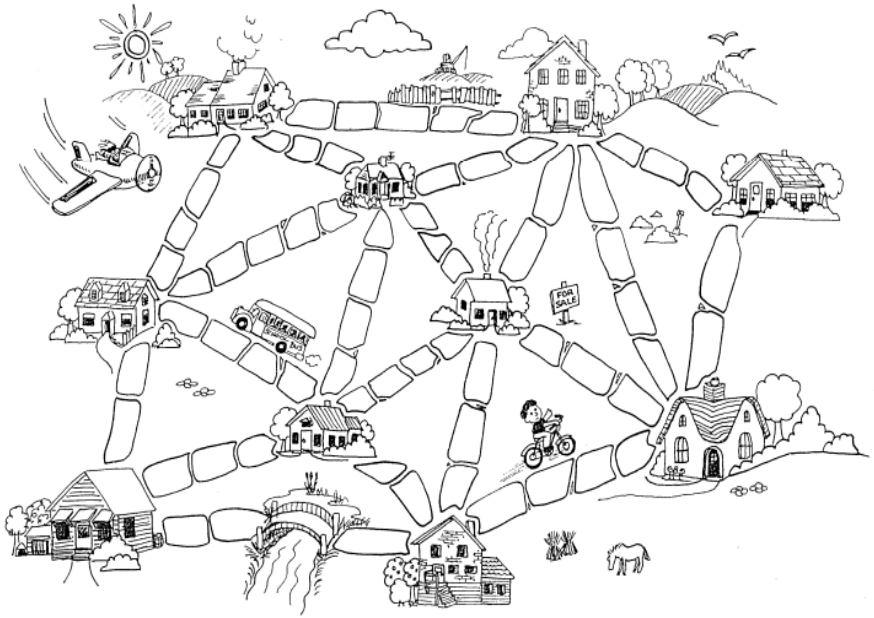


Spanning Tree

- Many applications
  - Design of networks (road, telecommunication, transportation etc.) - cover the network with minimum cost
  - Object detection, handwriting recognition
  - Cluster analysis (e.g. gene expression clustering)



Muddy City Problem



*The number of paving stones between each house represents the cost of paving that route

The best route is that connects all the houses, but uses as few paving stones as possible

- Consider a city with no concrete roads.
- Getting around the city is
- difficult especially after rainstorms
- We need to identify the streets that must be paved, but we don't want to spend more money than necessary.

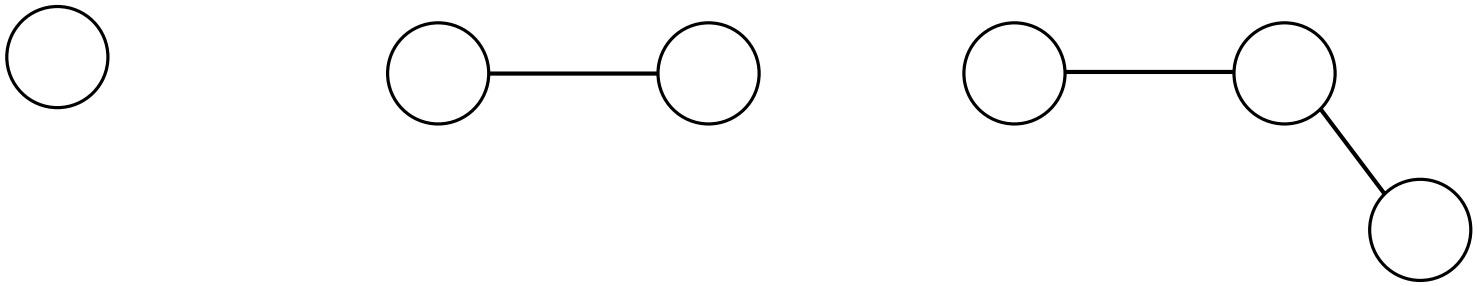
Image source: https://personal.utdallas.edu/~besp/teaching/mst-applications.pdf

Some points to note

- Every connected graph has at least one spanning tree.

- Given a connected graph of n vertices, what would be the number of edges in the corresponding spanning tree ?

- Theorem: Any tree with n vertices has $n - 1$ edges.
    - Proof (By induction)

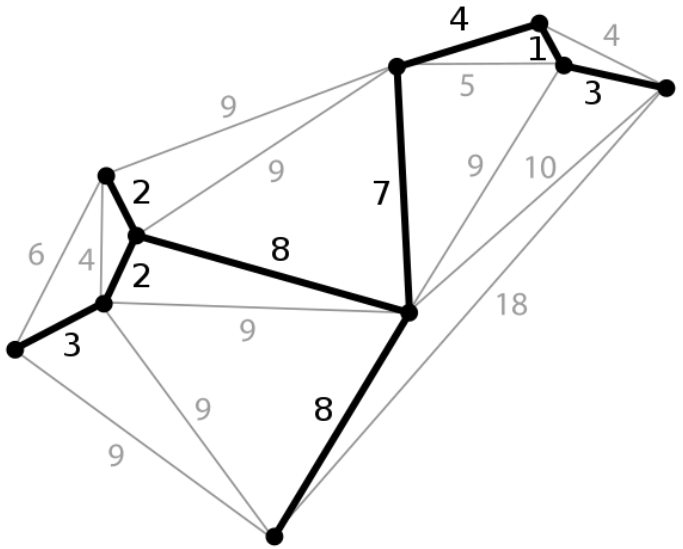


Some points to note

In an unweighted graph, we don't care about which edges are there in our tree : as long as the tree contains all the edges.

- In weighted graph, in most of the cases we do require to consider the weights – as we want to minimize the cost involved

- Minimum Cost Spanning tree



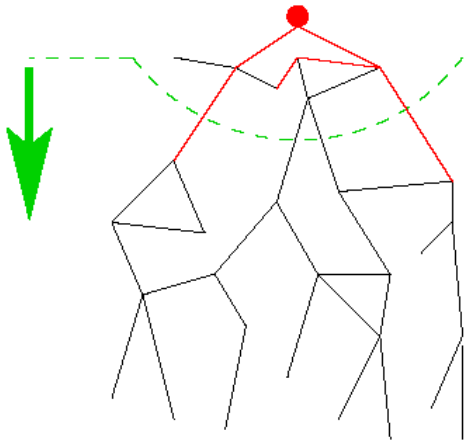
Finding Unweighted Spanning Trees

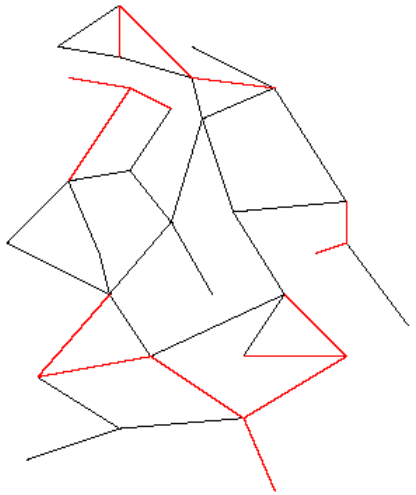
Different approaches to the spanning-tree problem:

- Do a graph traversal (e.g., depth-first search, but any traversal will do) and keep track of edges that form a tree

OR

- Iterate through edges and add to output any edge that doesn't create a cycle





Spanning Tree via DFS

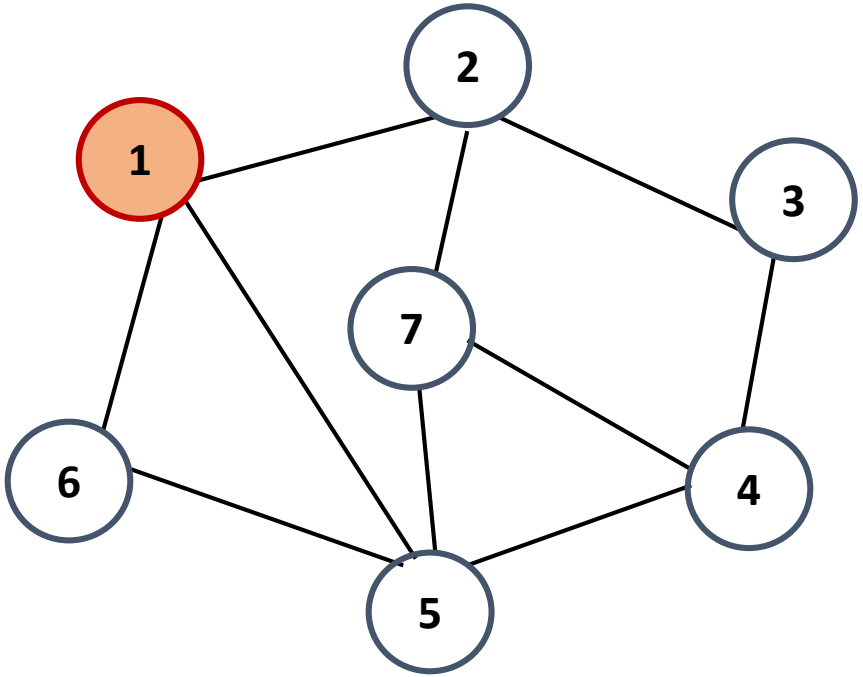
DFS reaches each node. We add one edge to connect an unvisited node to the already visited nodes. Order affects result.

```
spanning_tree_dfs(Graph G) {
  for each node i: i.marked = false
  for some node i: f(i)
}

f(Node i) {
  i.marked = true
  for each j adjacent to i:
      if(!j.marked) {
         add(i,j) to output
         f(j) // DFS
    }
}
```

Example

DFS calls

f(1)



1	→ 2, 5, 6
2	→ 1, 3, 7
3	→ 2, 4
4	→ 3, 5, 7
5	→ 1, 4, 6, 7
6	→ 1, 5
7	→ 2, 4, 5

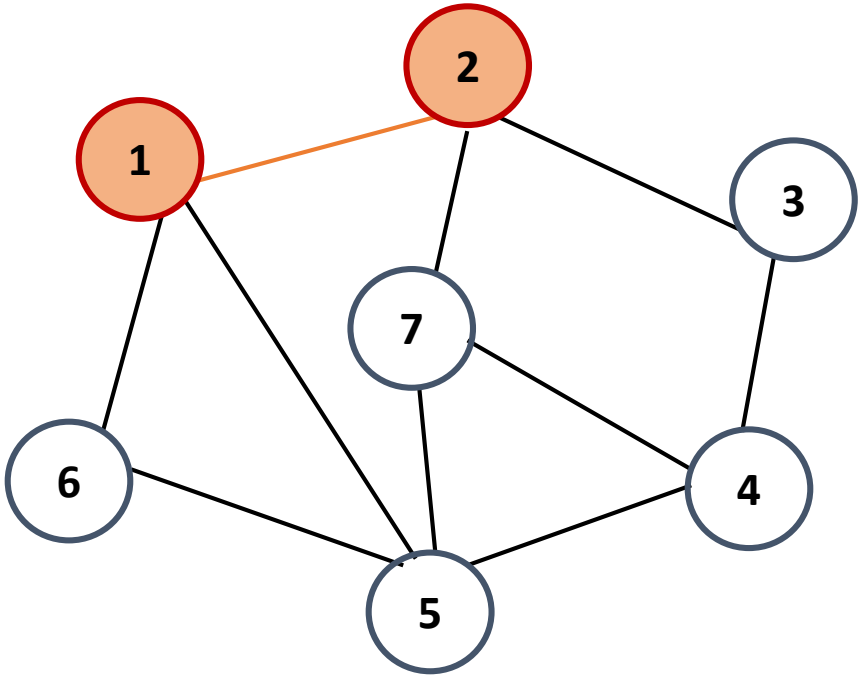
Output:

Example

DFS calls

f(1)

f(2)



1	→ 2, 5, 6
2	→ 1, 3, 7
3	→ 2, 4
4	→ 3, 5, 7
5	→ 1, 4, 6, 7
6	→ 1, 5
7	→ 2, 4, 5

Output: (1,2)

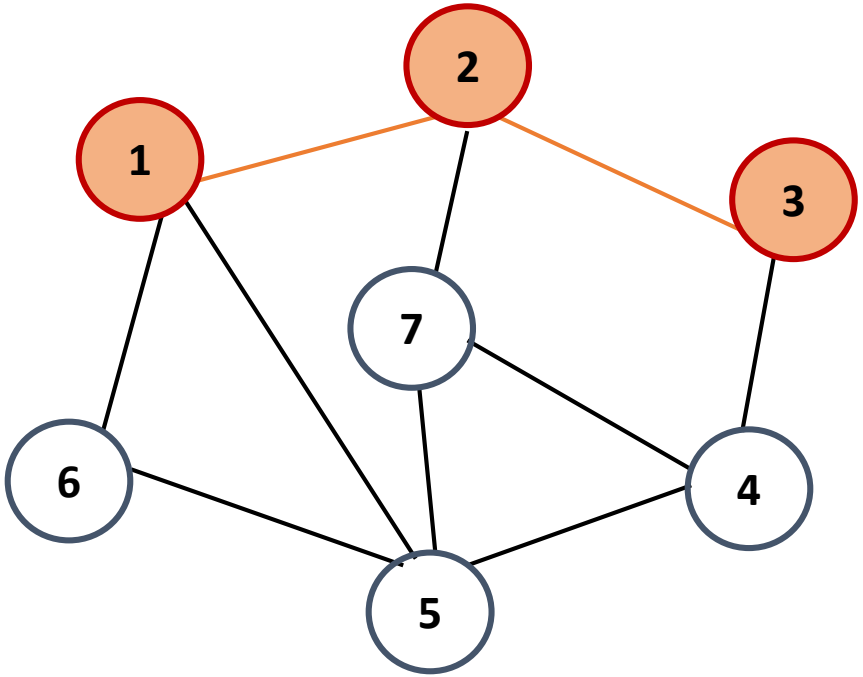
Example

DFS calls

f(1)

f(2)

f(3)



Output: (1,2), (2,3)

1	→ 2, 5, 6
2	→ 1, 3, 7
3	→ 2, 4
4	→ 3, 5, 7
5	→ 1, 4, 6, 7
6	→ 1, 5
7	→ 2, 4, 5

Example

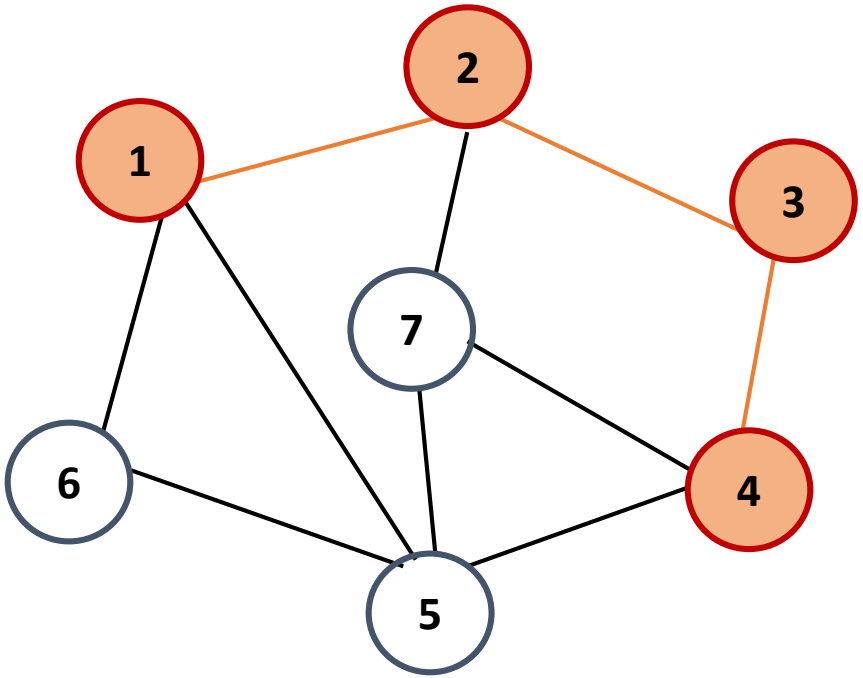
DFS calls

f(1)

f(2)

f(3)

f(4)



Output: (1,2), (2,3), (3,4)

1	→ 2, 5, 6
2	→ 1, 3, 7
3	→ 2, 4
4	→ 3, 5, 7
5	→ 1, 4, 6, 7
6	→ 1, 5
7	→ 2, 4, 5

Example

DFS calls

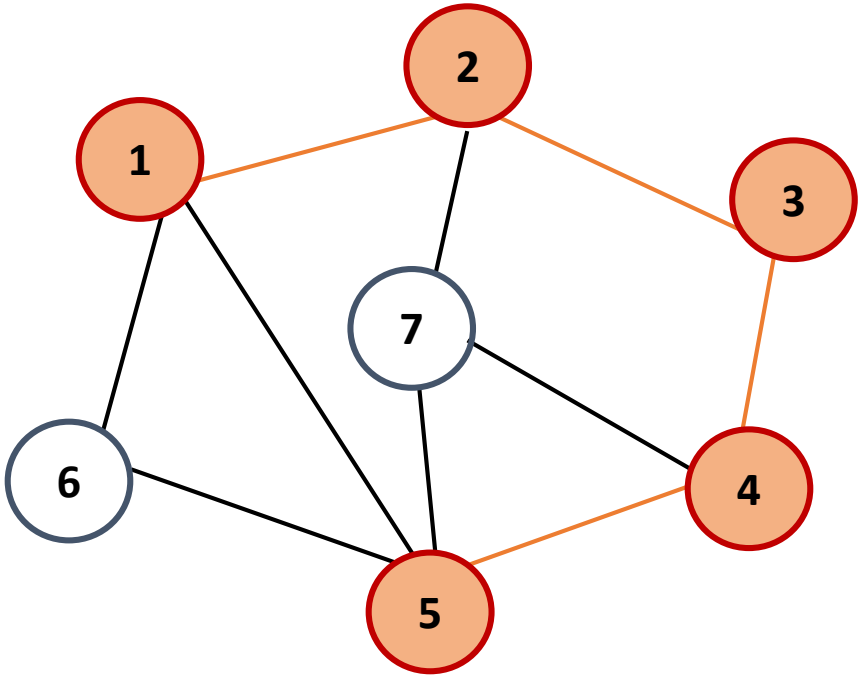
f(1)

f(2)

f(3)

f(4)

f(5)



1	2, 5, 6
2	1, 3, 7
3	2, 4
4	3, 5, 7
5	1, 4, 6, 7
6	1, 5
7	2, 4, 5

Output: (1,2), (2,3), (3,4), (4,5)

Example

DFS calls

f(1)

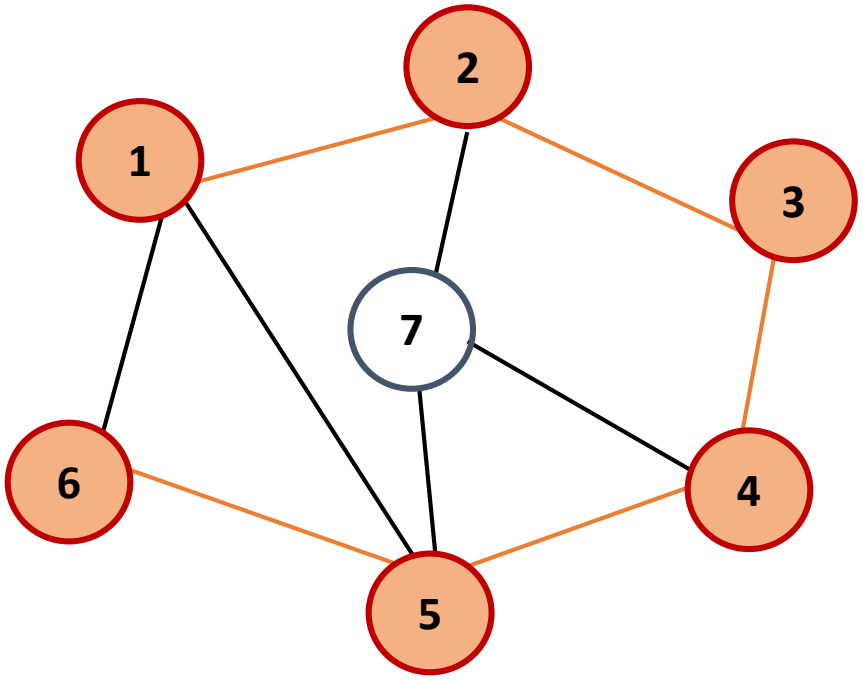
f(2)

f(3)

f(4)

f(5)

f(6)



1	→ 2, 5, 6
2	→ 1, 3, 7
3	→ 2, 4
4	→ 3, 5, 7
5	→ 1, 4, 6, 7
6	→ 1, 5
7	→ 2, 4, 5

Output: (1,2), (2,3), (3,4), (4,5), (5,6)

Example

DFS calls

f(1)

f(2)

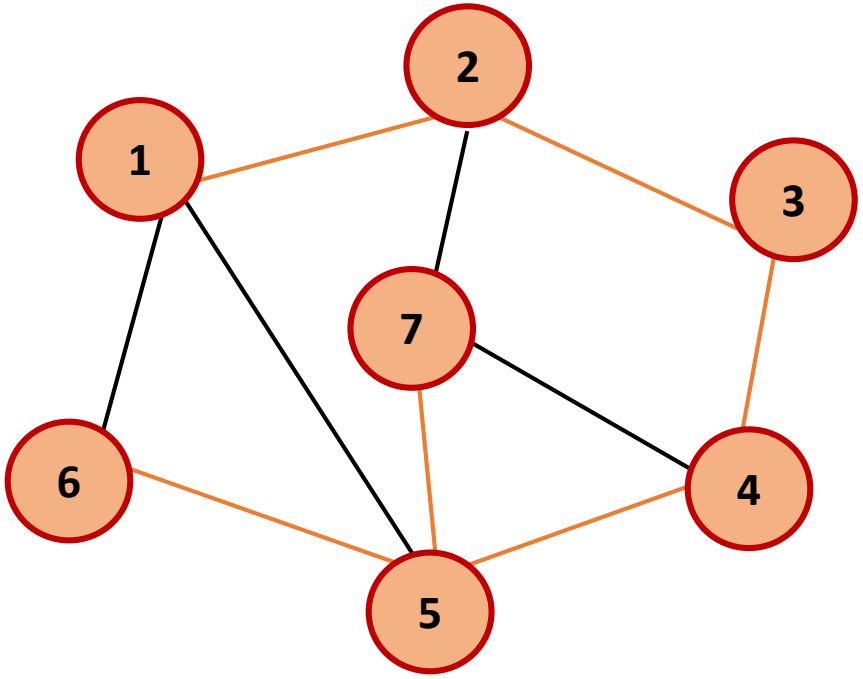
f(3)

f(4)

f(5)

f(6)

f(7)



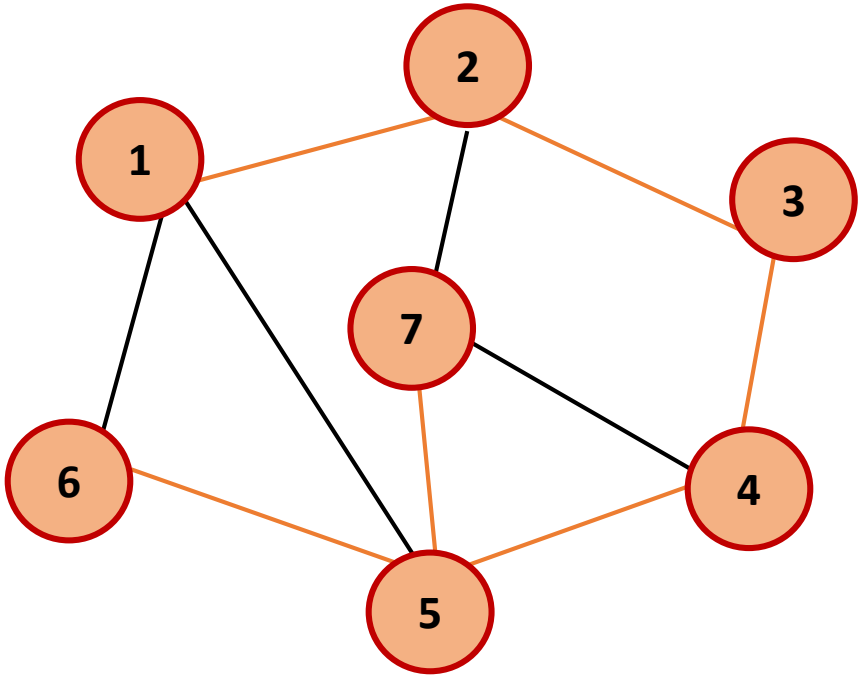
1	→ 2, 5, 6
2	→ 1, 3, 7
3	→ 2, 4
4	→ 3, 5, 7
5	→ 1, 4, 6, 7
6	→ 1, 5
7	→ 2, 4, 5

Output: (1,2), (2,3), (3,4), (4,5), (5,6), (5,7)

Example

DFS calls

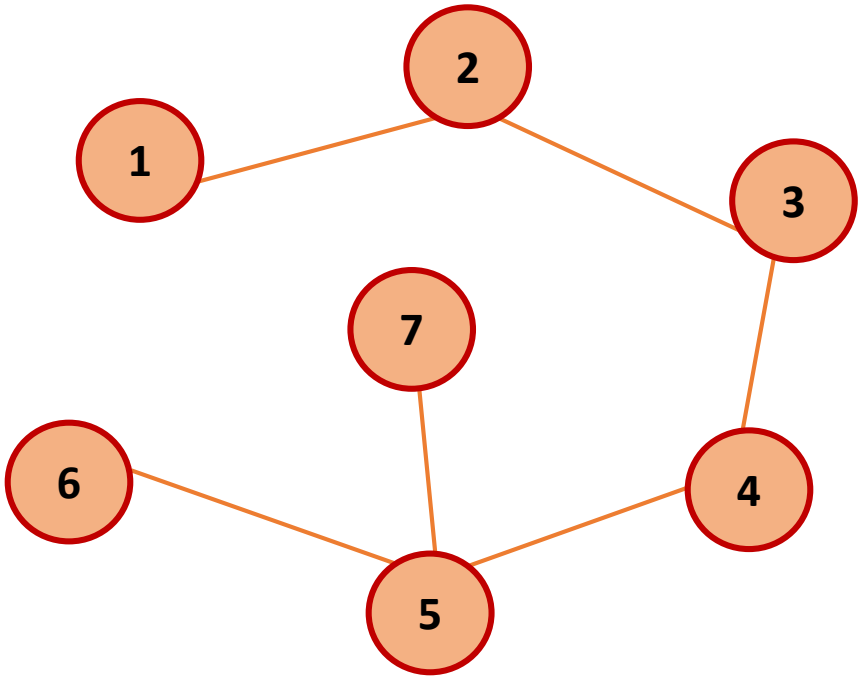
f(1)
f(2)
f(3)
f(4)
f(5)
f(6)
f(7)



1	→ 2, 5, 6
2	→ 1, 3, 7
3	→ 2, 4
4	→ 3, 5, 7
5	→ 1, 4, 6, 7
6	→ 1, 5
7	→ 2, 4, 5

Output: (1,2), (2,3), (3,4), (4,5), (5,6), (5,7)

Example



1	→ 2, 5, 6
2	→ 1, 3, 7
3	→ 2, 4
4	→ 3, 5, 7
5	→ 1, 4, 6, 7
6	→ 1, 5
7	→ 2, 4, 5

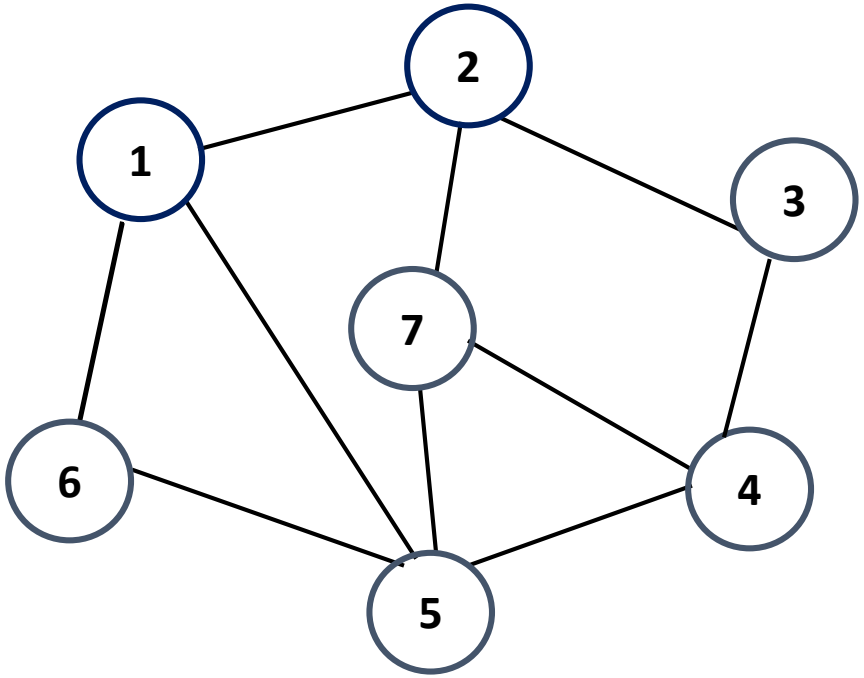
Output: (1,2), (2,3), (3,4), (4,5), (5,6), (5,7)

Another approach

Iterate through edges; output any edge that does not create a cycle

Consider edges in some arbitrary order:

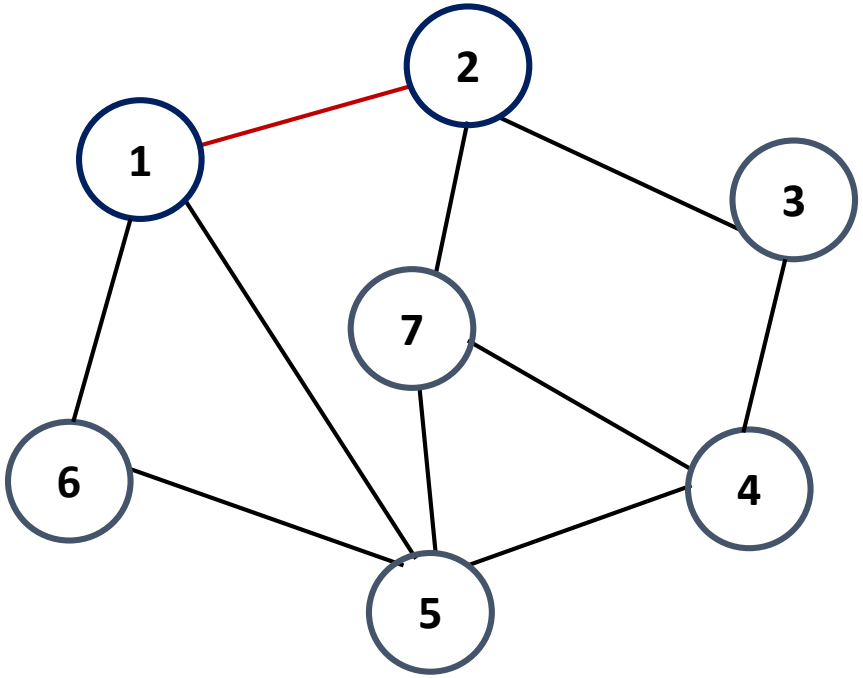
(1,2), (3,4), (5,6), (5,7),(1,5), (1,6), (2,7), (2,3), (4,5), (4,7)



Example

Edges in some arbitrary order:

(1,2), (3,4), (5,6), (5,7),(1,5), (1,6), (2,7), (2,3), (4,5), (4,7)

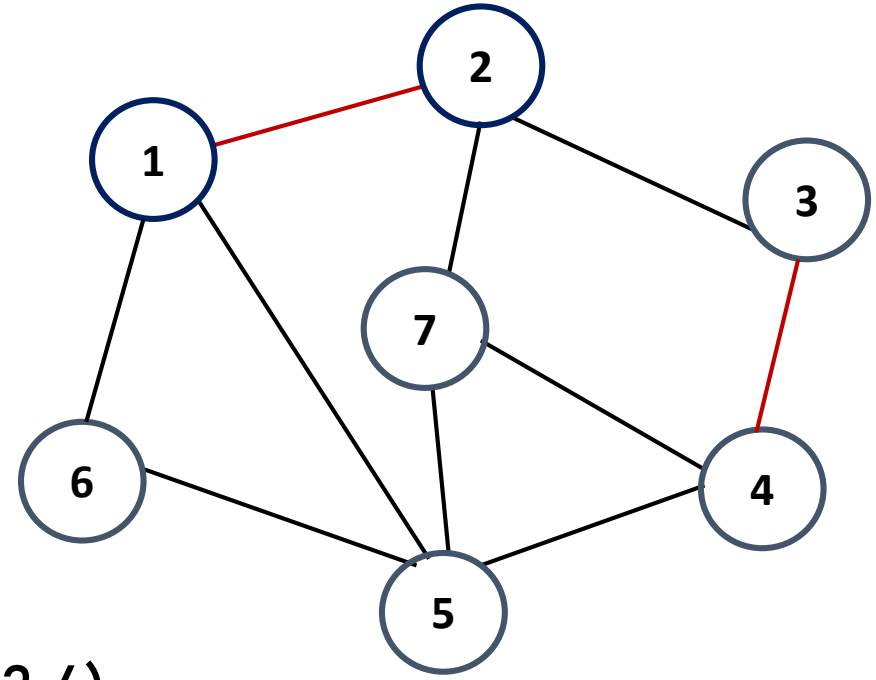


Output: (1,2)

Example

Edges in some arbitrary order:

(1,2), (3,4), (5,6), (5,7),(1,5), (1,6), (2,7), (2,3), (4,5), (4,7)

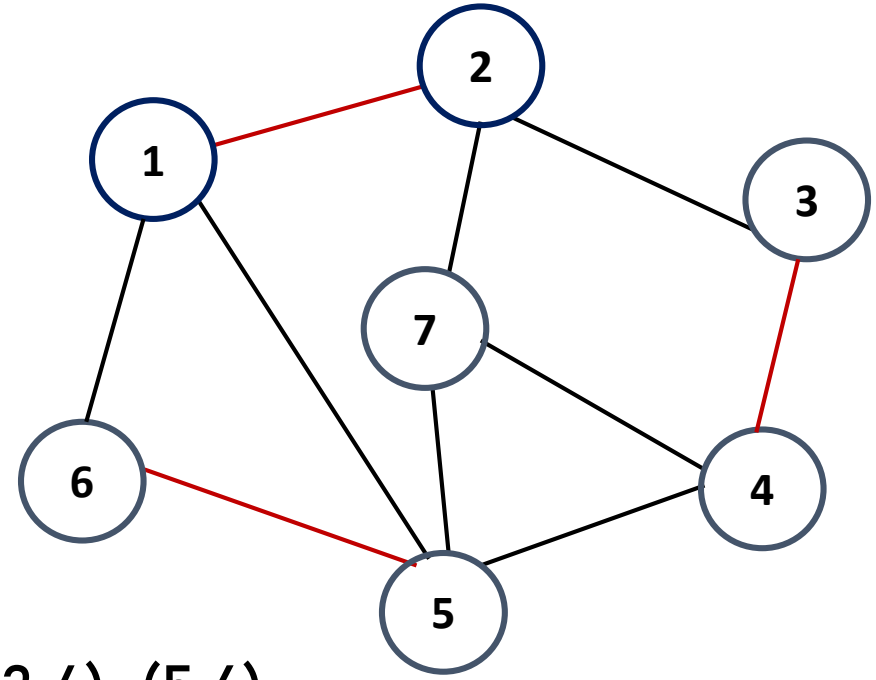


Output: (1,2), (3,4)

Example

Edges in some arbitrary order:

(1,2), (3,4), (5,6), (5,7),(1,5), (1,6), (2,7), (2,3), (4,5), (4,7)

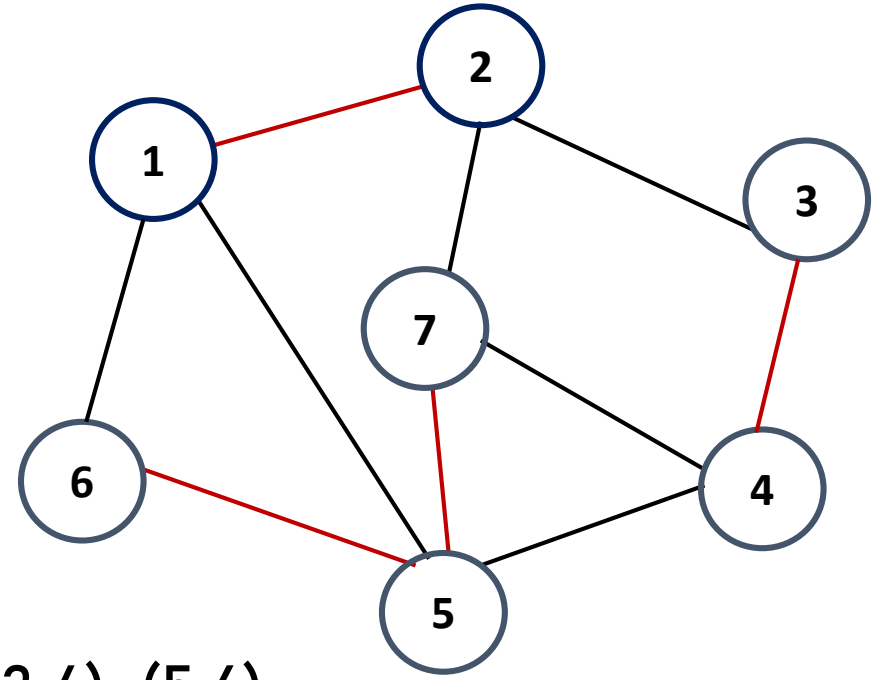


Output: (1,2), (3,4), (5,6)

Example

Edges in some arbitrary order:

(1,2), (3,4), (5,6), (5,7), (1,5), (1,6), (2,7), (2,3), (4,5), (4,7)

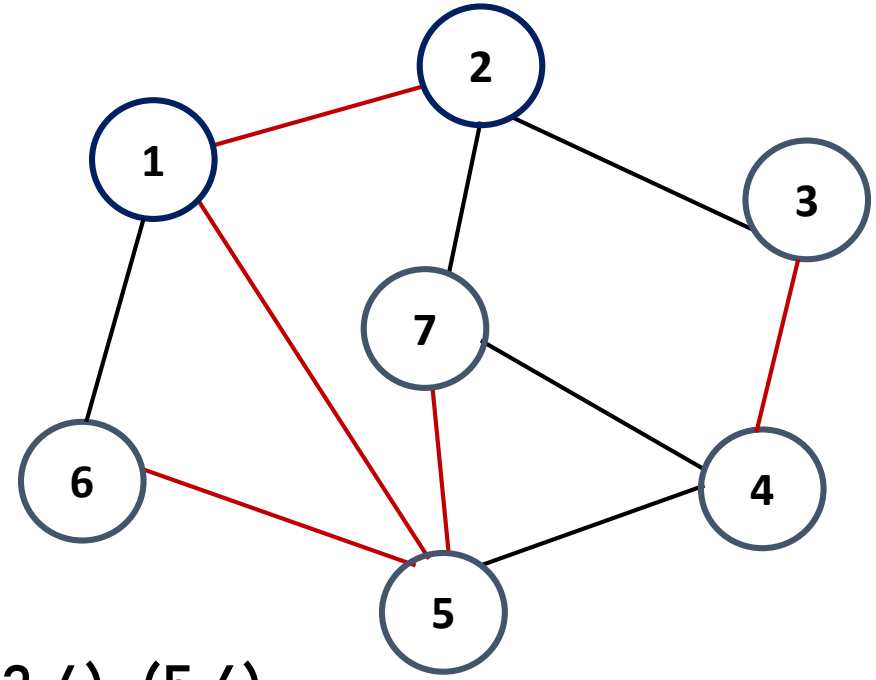


Output: (1,2), (3,4), (5,6), (5,7)

Example

Edges in some arbitrary order:

(1,2), (3,4), (5,6), (5,7),(1,5), (1,6), (2,7), (2,3), (4,5), (4,7)

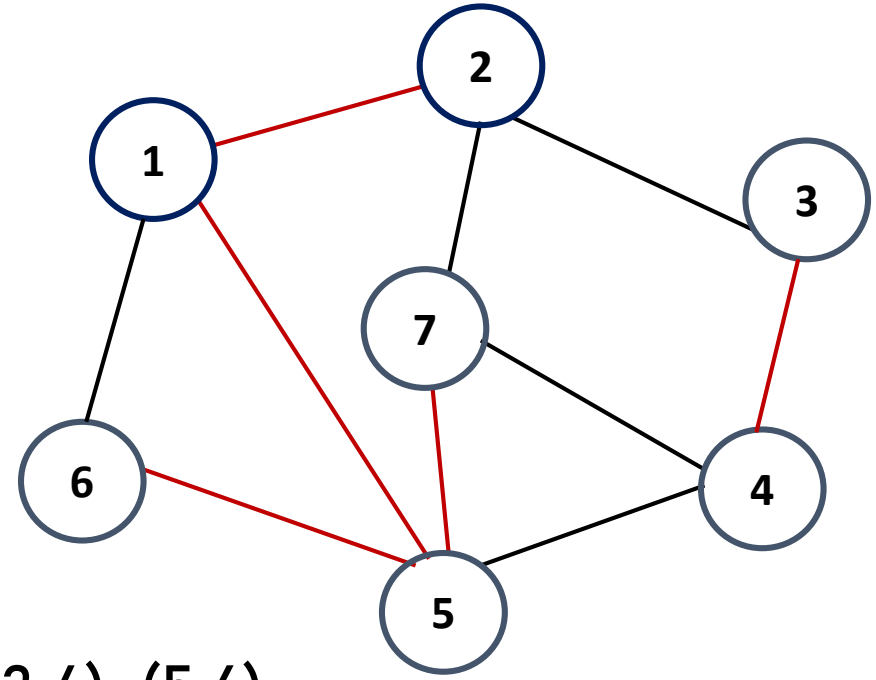


Output: (1,2), (3,4), (5,6),
(5,7), (1,5)

Example

Edges in some arbitrary order:

(1,2), (3,4), (5,6), (5,7),(1,5), (1,6), (2,7), (2,3), (4,5), (4,7)

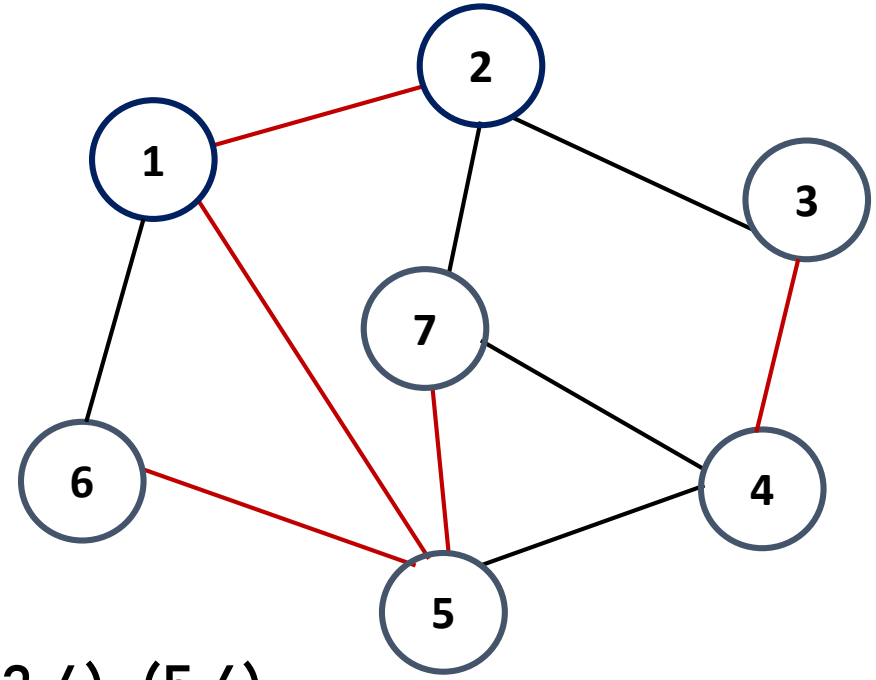


Output: (1,2), (3,4), (5,6),
(5,7), (1,5)

Example

Edges in some arbitrary order:

(1,2), (3,4), (5,6), (5,7),(1,5), (1,6), (2,7), (2,3), (4,5), (4,7)

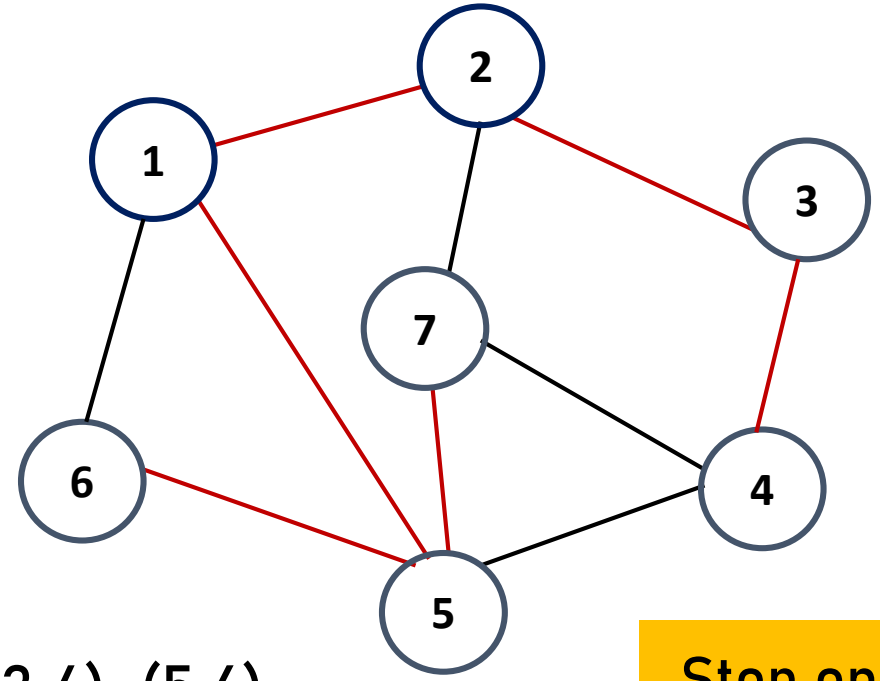


Output: (1,2), (3,4), (5,6),
(5,7), (1,5)

Example

Edges in some arbitrary order:

(1,2), (3,4), (5,6), (5,7),(1,5), (1,6), (2,7), (2,3), (4,5), (4,7)



Output: (1,2), (3,4), (5,6), (5,7), (1,5), (2,3)

Stop once we have |V|-1 edges

Complexity

- Spanning tree via DFS: Adjacency matrix $O(V^2)$,

Adjacency list: $O(V + E)$

- Iterating through edges: Involves deciding if addition of an edge could form a cycle.
  - Finding a cycle:
    - Adjacency matrix
      - $O(V^2)$
    - Adjacency list
      - $O(V + E)$
  - Overall:
  - Adjacency matrix
    - $O(V^3)$
  - Adjacency list
  - $O(V(V + E)) \approx O(VE)$ (if E>> V)

```
spanning_tree_edge(Graph G) {
  n =0;
  while n < |V|-1 :
      pick an edge (i,j)
       if(addition of (i,j) does not form a
          cycle in the output)
              add(i, j) to output
       n++
```

Faster method with Disjoint sets
Next topic

Spanning Trees in Weighted Graphs

Spanning trees: Weighted graphs

The spanning-tree problem

- Add nodes to partial tree
- Add acyclic edges

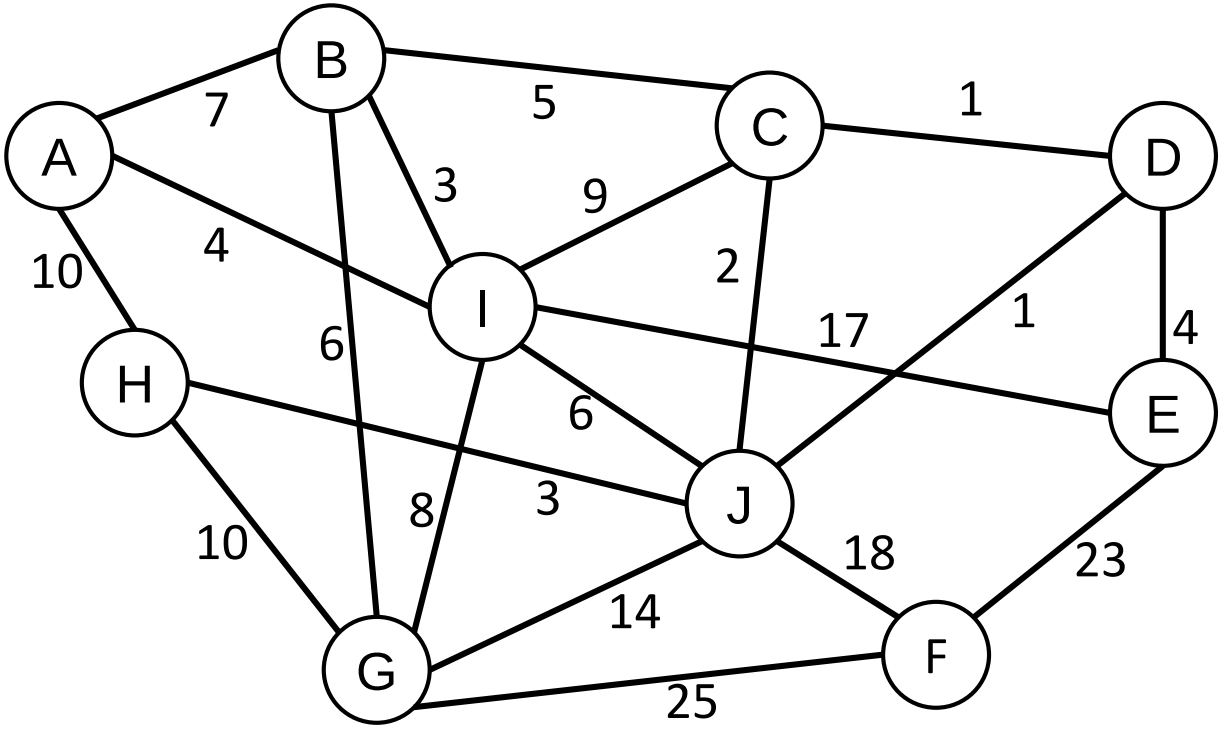
Minimum-cost-spanning-tree problem

- Given a connected, weighted, undirected graph, find a spanning tree of minimum weight
- The above approaches suffice with minor changes:
  - Add nodes to partial tree approach: Prim's Algorithm
  - Add acyclic edges approach : Kruskal's algorithm

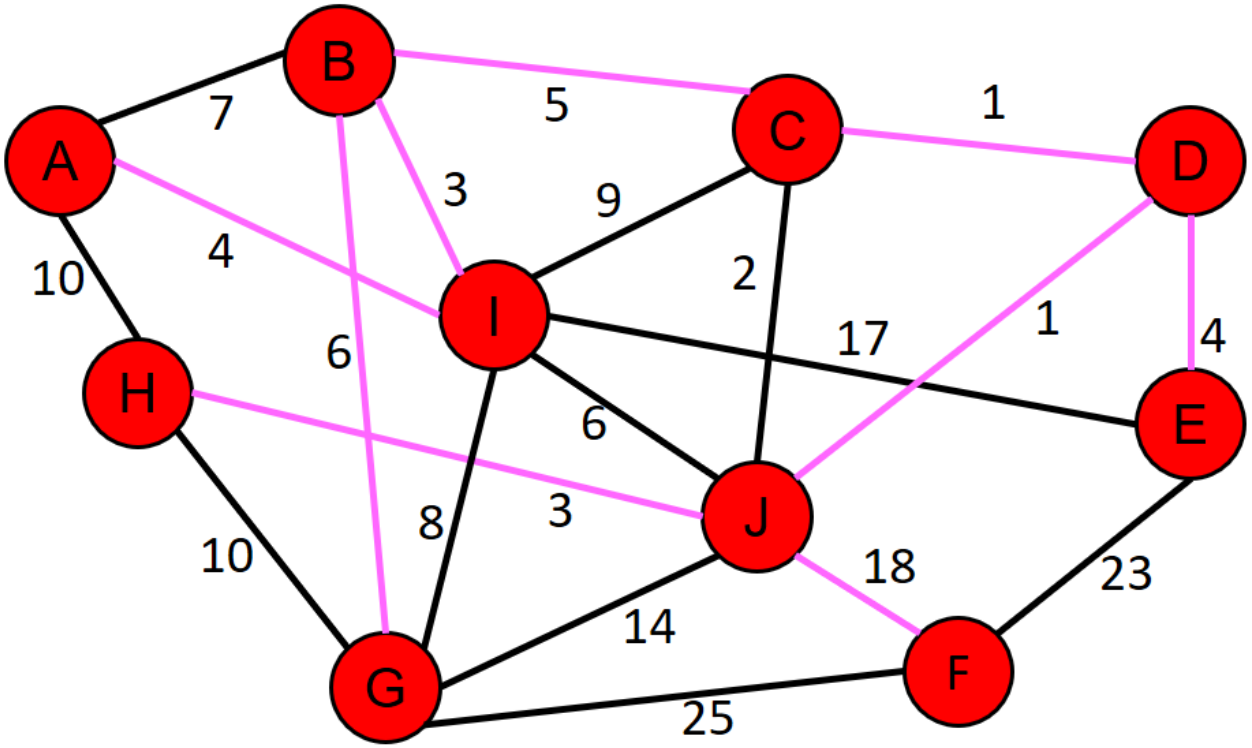
Prim's Algorithm

- Randomly select a root node, initialize a single-node tree
- Grow the tree from a root node
- Repeatedly add one node outside the tree into the tree until all the nodes are in the tree
- Add a new edge: one node in the tree, the other outside the tree such that the added edge has the minimum weight

Prim's Algorithm



Prim's Algorithm

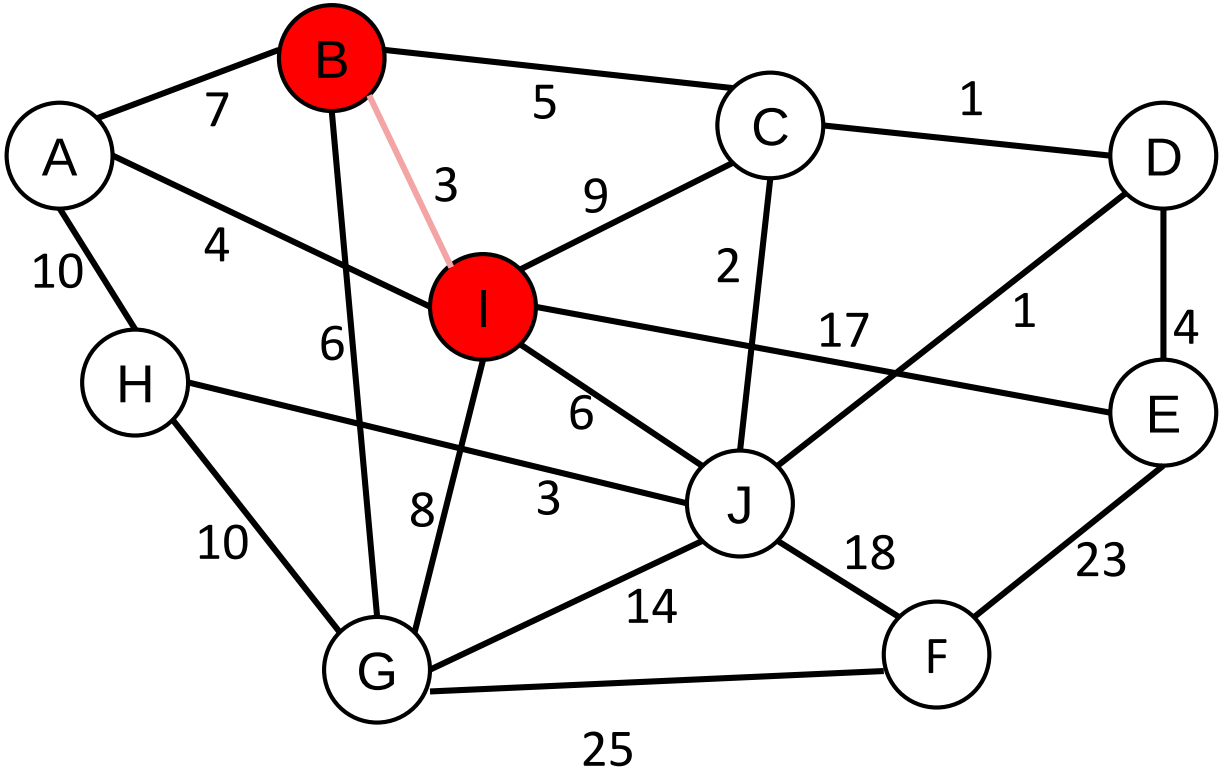


Prim's Algorithm

Given: a connected undirected weight graph

```
Initialize fringe to have a root node with costToTree = 0

all nodes are unvisited;

Repeat until all nodes are visited {
   Choose from fringe the unvisited node (n*) with minimum costToTree;

  Add the corresponding edge to the spanning tree, set n* as visited

  for each (edge (n*, n') with one end-node n*) {
      if (n' is not visited) then add <n', (n*,n'), cost(n*,n')> into the fringe;
   }
}
```

Example: First iteration



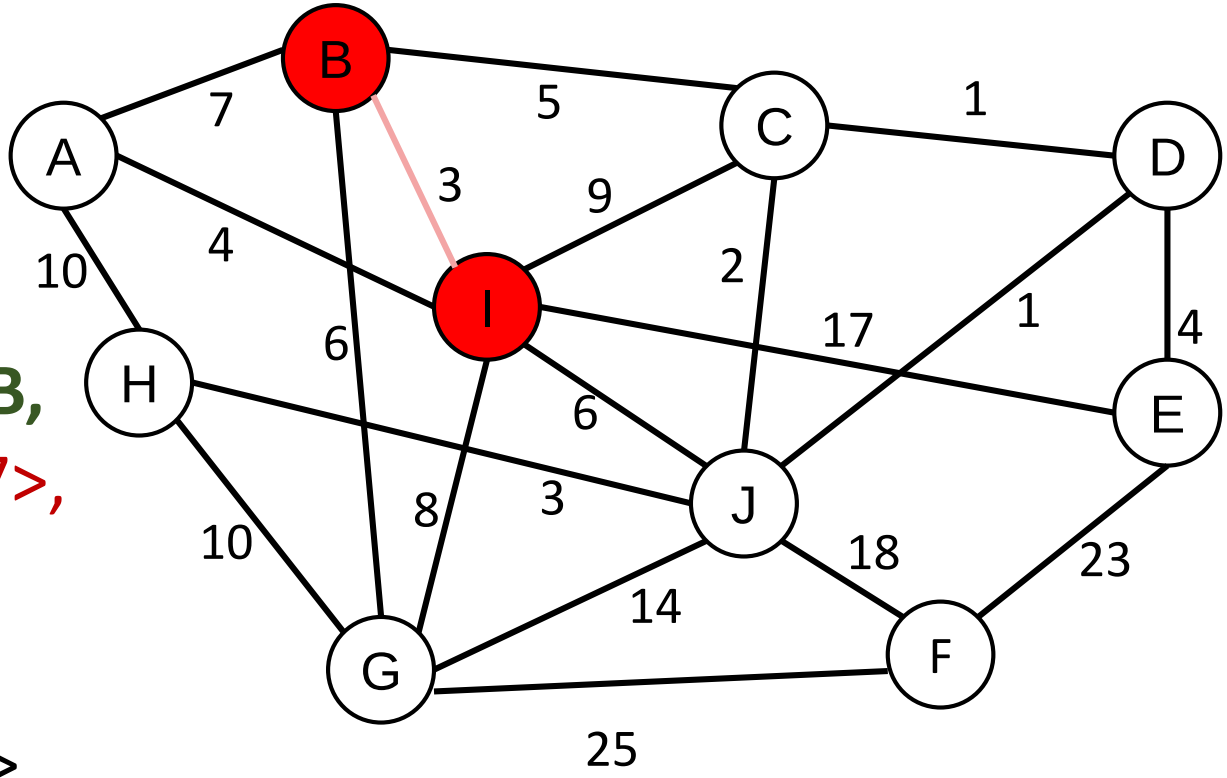
Fringe: <I, null 0>

Spanning Tree:

Visited: I

Fringe: <I, A, 4>, <I, B, 3>, <I, C, 9>, <I, E, 17>,
<I, J, 6>, <I, G, 8>

Example: Second iteration



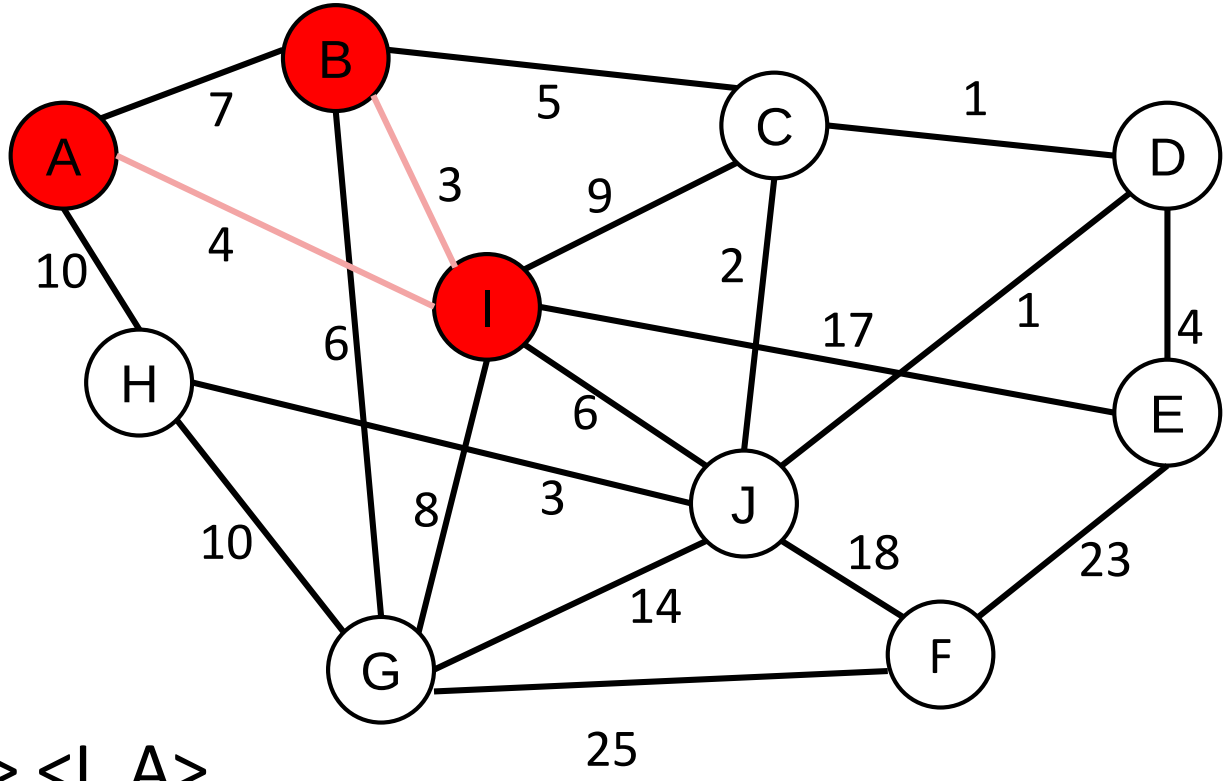
Fringe: <I, A, 4>, **<I, B, 3>**, <I, C, 9>, <I, E, 17>, <I, J, 6>, <I, G, 8>

Spanning Tree: <I,B>

Visited: I, B

Fringe: <I, A, 4>, ~~<I, B, 3>~~, <I, C, 9>, <I, E, 17>, <I, J, 6>, <I, G, 8>, **<B, A, 7>, <B, C, 5>, <B, G, 6>**

Example: Third iteration



Fringe: <I, A, 4>
<I, C, 9>, <I, E, 17>,
<I, J, 6>, <I, G, 8>,
<B, A, 7>, <B, C, 5>,
<B, G, 6>

Spanning Tree: <I,B> <I, A>

Visited: I, B, A

Fringe: ~~<I, A, 4>~~, <I, C, 9>, <I, E, 17>,
<I, J, 6>, <I, G, 8>, <B, A, 7>, <B, C, 5>, <B, G, 6>,
<A, H, 10>

Next Lecture

Spanning Trees (cont.)