Algorithms and Data Structures

COMP261
Tutorial Week 5

Yi Mei

yi.mei@ecs.vuw.ac.nz

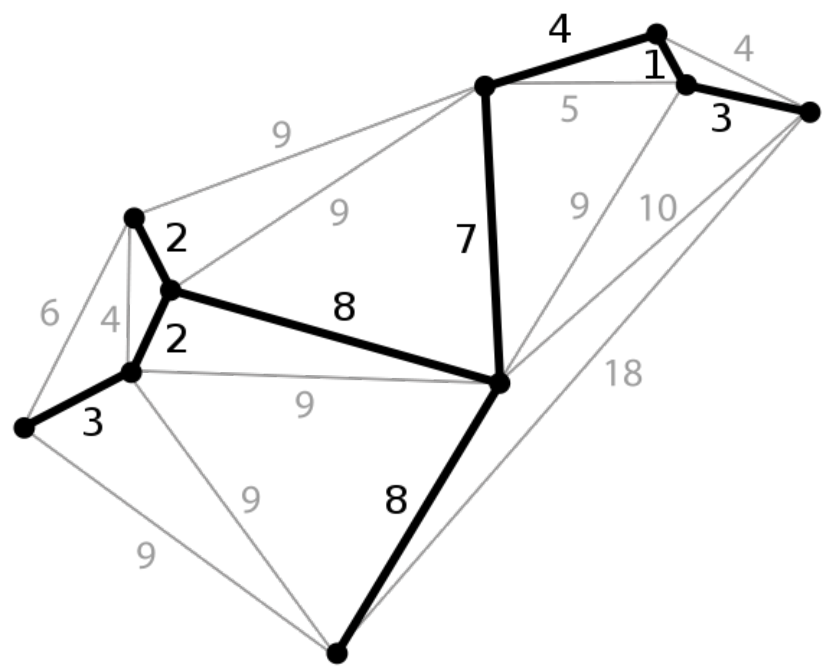
Outline

- Minimum Spanning Tree
  - Prim’s Algorithm
  - Kruskal’s Algorithm

- Disjoint set for Kruskal’s Algorithm

Minimum Spanning Tree

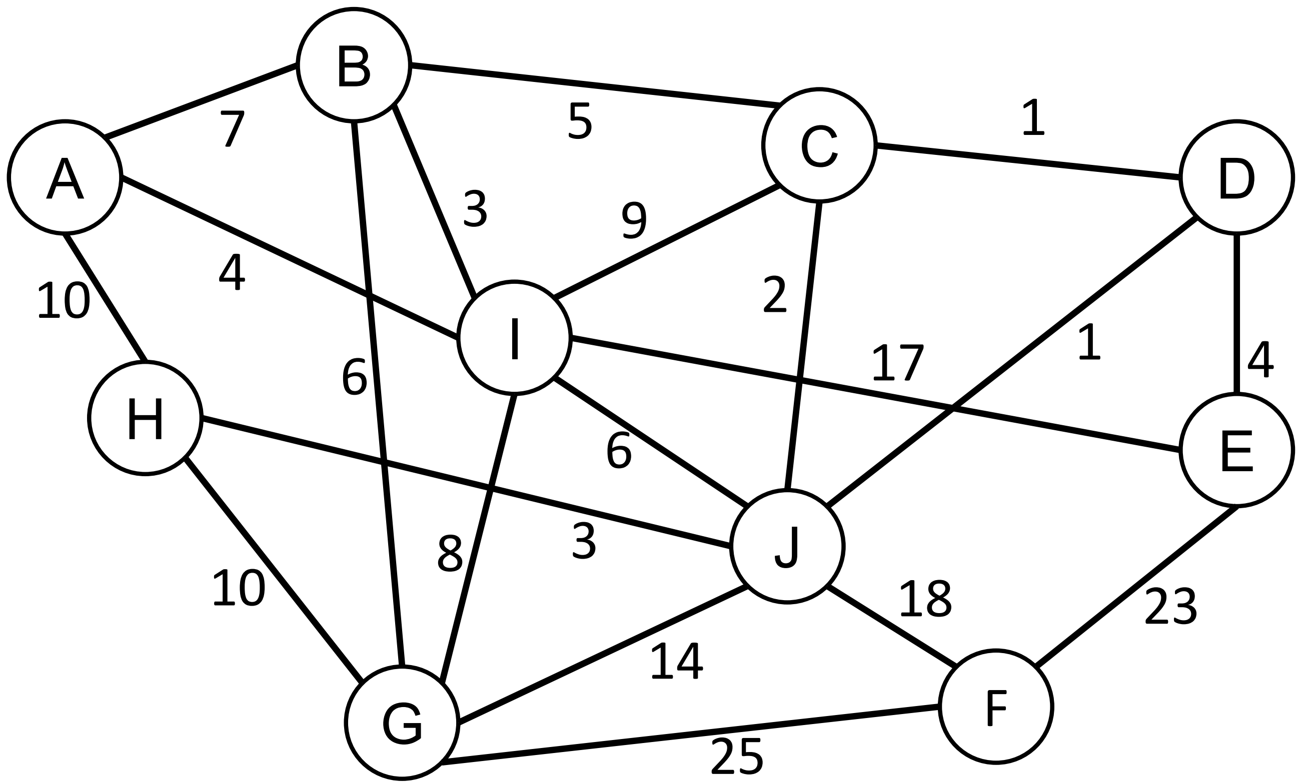
- Given a connected, undirected, weighted graph
- A **spanning tree** is a subgraph that contains all the nodes but is a tree (no cycle)
  - A spanning tree does not require a weighted graph
- A **minimum spanning tree (MST)** is a spanning tree with the minimum total weight among all the spanning trees, i.e. its total weight is no greater than the total weight of any other spanning tree



- Two algorithms to find MST
  - Prim's Algorithm
  - Kruskal's Algorithm
  - Both can prove correctness

Prim's Algorithm

- Grow the tree from a root node
  - Randomly select a root node, initialise a single-node tree
  - 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
    - The added edge has the minimum weight



Prim's Algorithm

Given: a connected undirected weight graph

Initialise *fringe* to have a root node with *costToTree* = 0 and a dummy edge, 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'*) outgoing from node *n**) {

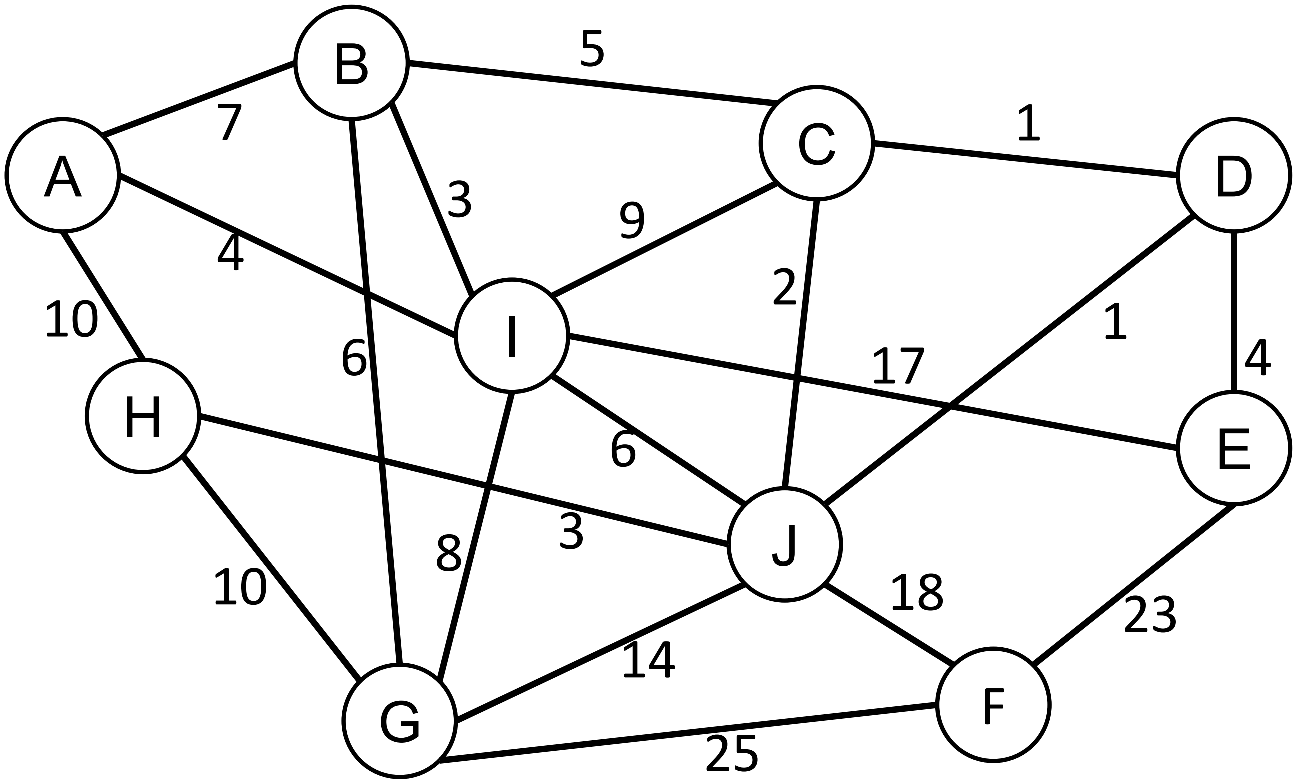
if (*n'* is not unvisited) **then** add *<n', (n*,n'), cost(n*,n')>* into the fringe;

}

}

Kruskal's Algorithm

- Merge trees
  - Initially, each node is a single-node tree
  - At each step, merge two trees into one
  - The merge cost is the minimum (min-cost edge)



Kruskal's Algorithm

Given: a connected undirected weight graph (*N* nodes, *M* edges)

Set *forest* as *N* node sets, each containing a node;

Set *fringe* as a priority queue of all the edges ⟨n1, n2, length⟩;

Set *tree* as an empty set of edges;

Repeat until *forest* contains only one tree or edges is empty {

Get and remove *⟨ n1*, n2*, length*⟩* as the edge with minimum length from fringe;

if (*n1** and *n2** are in different sets in forest) {

Merge the two sets in *forest*;

Add the edge to *tree*;

}

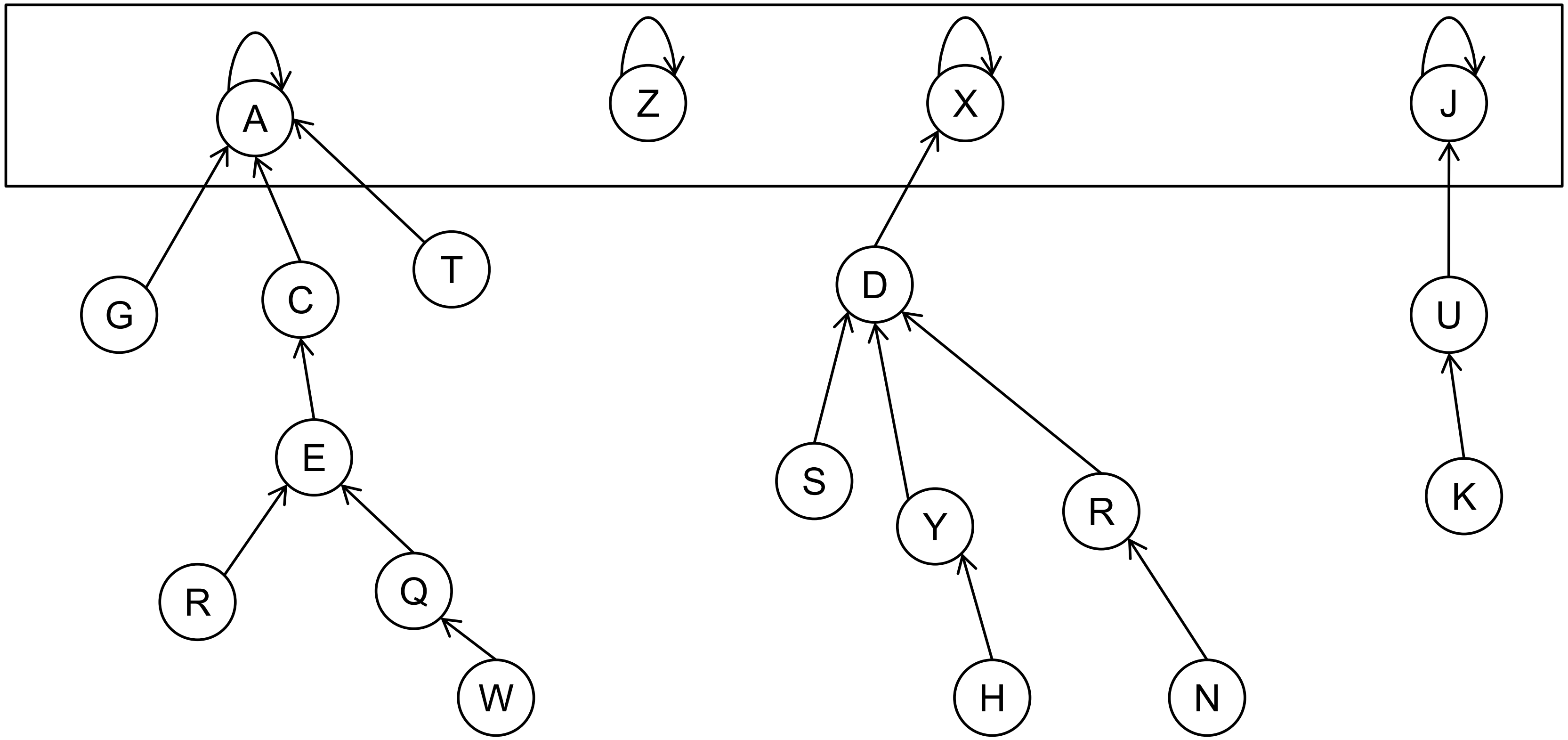
}

return *tree*;

Most time consuming steps

Disjoint Set

- Disjoint-set (union-find) data structure
  - Set of inverted trees
  - Each set is represented by a linked tree with links pointing **towards** the root
  - **Forest** = set of root nodes



Disjoint Set

- To reduce complexity, always merge shorter trees into deeper ones

```
MakeSet(x) {
    x.parent = x;
    x.depth = 0;
    add x to forest;
}

Find(x) {
    if (x.parent == x) {
        return x;
    } else {
        root = Find(x.parent);
        return root;
    }
}
```

```
Union(x, y) {
    xroot = Find(x);
    yroot = Find(y);
    if (xroot == yroot) {
        return;
    } else {
        if (xroot.depth < yroot.depth) {
            xroot.parent = yroot;
            remove xroot from forest;
        } else {
            yroot.parent = xroot;
            remove yroot from forest;
            if (xroot.depth == yroot.depth)
                xroot.depth ++;
        }
}
```

Example