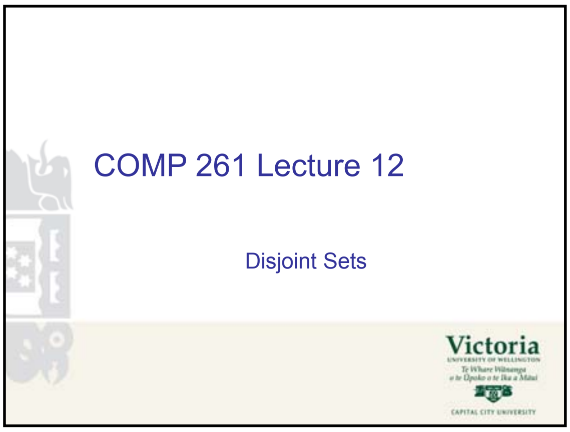
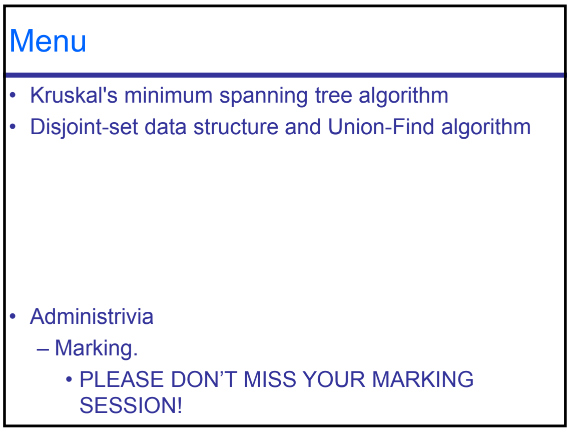


COMP 261 Lecture 12

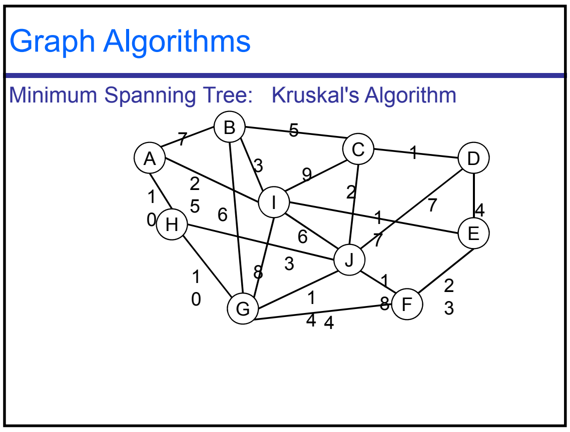
Disjoint Sets



Menu

- Kruskal's minimum spanning tree algorithm
- Disjoint-set data structure and Union-Find algorithm

- Administrivia
  - Marking.
    - PLEASE DON'T MISS YOUR MARKING SESSION!



Graph Algorithms

Minimum Spanning Tree: Kruskal's Algorithm

Refining Kruskal's algorithm

- Given: a graph with *N* nodes and *E* weighted edges

```
forest ← a set of N sets of nodes, each containing one node of the graph
edges ← a priority queue of all the edges: ⟨$n_1$, $n_2$, length⟩
spanningTree ← an empty set of edges
Repeat until forest contains only one tree or edges is empty:
    ⟨$n_1$, $n_2$, length⟩ ← dequeue(edges)
    If $n_1$ and $n_2$ are in different sets in forest then
        merge the two sets in forest
        Add edge to the spanningTree
return spanningTree
```

priority:
short edges first

What's the cost?

- Implementing *forest* :
  - set of sets with two operations:
    - findSet(n_1) =?= findSet(n_2) = "find" the set that *n* is in
    - merge(s_1, s_2) = replace s_1, s_2 by their "union"

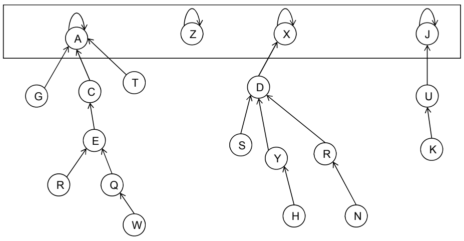
Implementing sets of sets

1: forest = set of sets of nodes:
- findSet(n_1)
  - iterate through all sets, calling s.contains(n_1)
- merge (s_1 , s_2)
  - add each element of s_1 to s_2 and remove s_1
- cost?

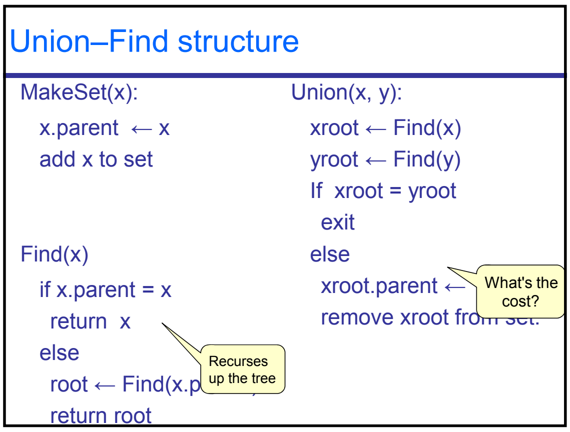
2: forest = mark each node with ID of its set
- findSet(n_1):
  - look up n_1.setID
- merge(s_1 , s_2)
  - iterate through all nodes, changing IDs of nodes in s_1
- cost?

Union–Find structure

3: forest: set of inverted trees of nodes:
- Each set represented by a linked tree with links pointing ***towards*** the root (= "shared linked list structure")



- The nodes in these trees *are* the nodes of the graph!



Union–Find structure

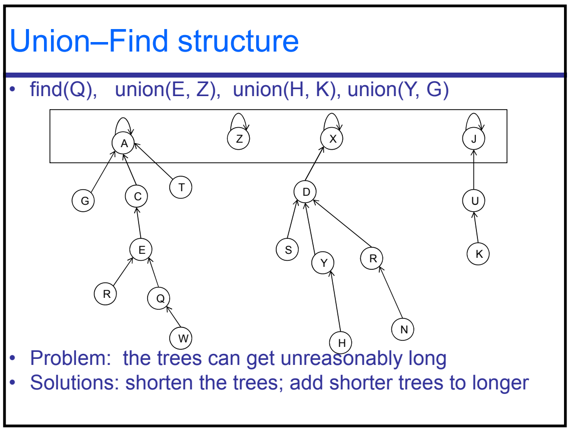
```
MakeSet(x):
  x.parent ← x
  add x to set

Find(x)
  if x.parent = x
    return x
  else
    root ← Find(x.p
    return root

Union(x, y):
  xroot ← Find(x)
  yroot ← Find(y)
  If xroot = yroot
    exit
  else
    xroot.parent ←
    remove xroot from set.
```

Recurses up the tree

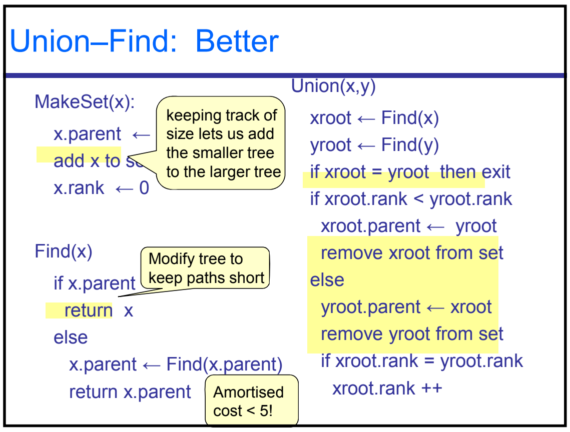
What's the cost?



Union–Find structure

- find(Q), union(E, Z), union(H, K), union(Y, G)

- Problem: the trees can get unreasonably long
- Solutions: shorten the trees; add shorter trees to longer



Union–Find: Better

```
MakeSet(x):
  x.parent ←
  add x to s
  x.rank ← 0

Find(x)
  if x.parent
    return x
  else
    x.parent ← Find(x.parent)
    return x.parent

Union(x,y)
  xroot ← Find(x)
  yroot ← Find(y)
  if xroot = yroot then exit
  if xroot.rank < yroot.rank
    xroot.parent ← yroot
    remove xroot from set
  else
    yroot.parent ← xroot
    remove yroot from set
    if xroot.rank = yroot.rank
      xroot.rank ++
```

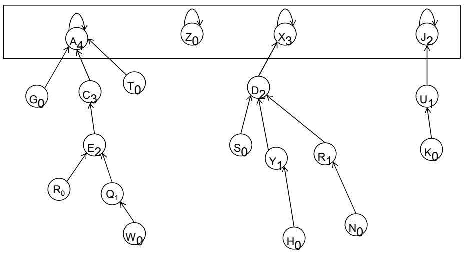
keeping track of size lets us add the smaller tree to the larger tree

Modify tree to keep paths short

Amortised cost < 5!

Union–Find structure

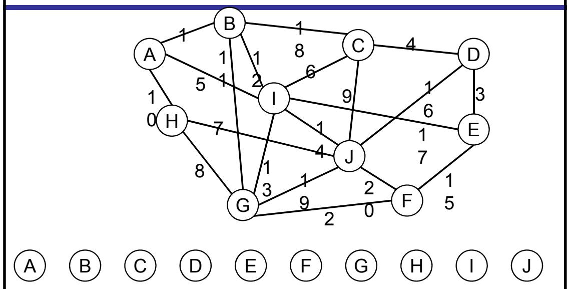
- find(Q), union(E, Z), union(H, K), union(Y, G)



Applications

- Union-Find is very efficient for a collection of sets if
  - Have an explicit collection of possible members
  - All sets are disjoint.
  - Only asking for same set membership and merging two sets.
- Inefficient for
  - enumerating the elements of a set
  - removing an element from a set
- Doesn't work
  - if the sets could share elements.

Exercise:


A
B
C
D
E
F
G
H
I
J
1
18
4
11
12
6
5
11
10
9
3
16
7
14
1
17
8
13
19
2
20
15
A
B
C
D
E
F
G
H
I
J