

Lecture 9

Data Structures (DAT037)

Ramona Enache

(with slides from Nick Smallbone and
Nils Anders Danielsson)

Trees (as graphs)

Again

A **tree** is

+ acyclic connected graph



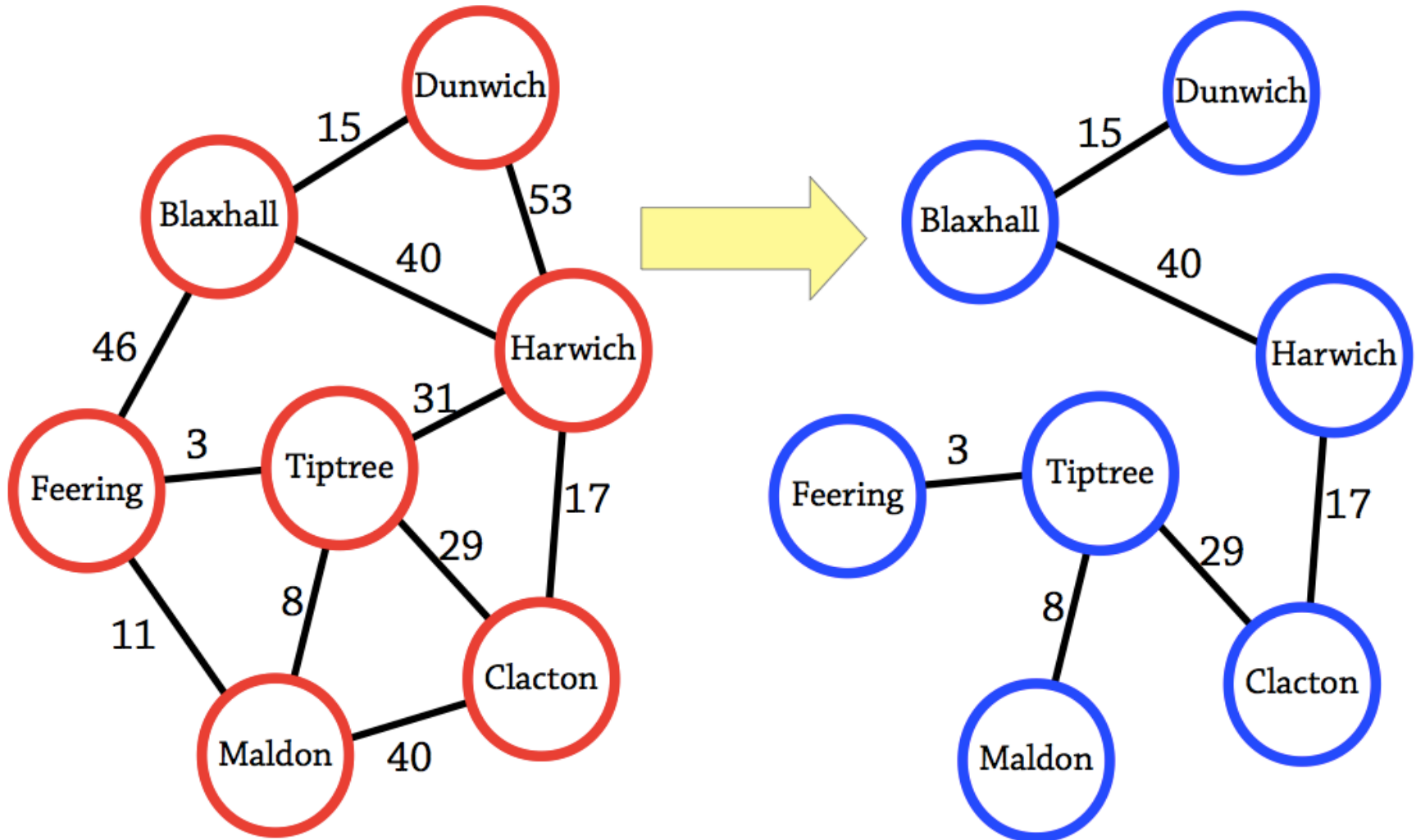
Minimum Spanning Tree (MST)

A **spanning tree** of a graph is a subgraph (obtained by deleting some of the edges) which:

- is acyclic
- is connected

A **minimum spanning tree** is one where the total weight of the edges is as low as possible

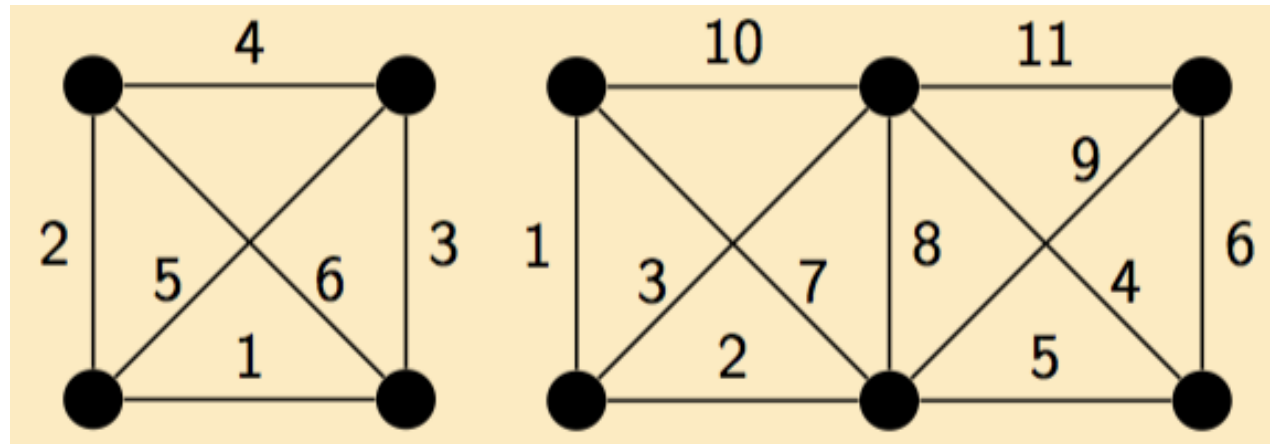
Minimum Spanning Tree (MST)



Question

What is the total weight of the MSTs corresponding to the following graphs ?

1. 19
2. 22
3. 24
4. other



govote.at
Code 380424

Prim's Algorithm

We will build a minimum spanning tree by starting with no edges and adding edges until the graph is connected

Keep a set S of all the nodes that are in the tree so far, initially containing one arbitrary node

While there is a node not in S :

- pick the *lowest-weight edge* between a node in S and a node not in S
- add that edge to the spanning tree, and add the node to S

Prim's Algorithm

Done = new set containing arbitrary node s

ToDo = $V \setminus \{s\}$

T = new empty set // MST for nodes from Done.

while ToDo is non-empty

do

if no edge connects Done and ToDo

then raise error: graph not connected

(u,v) = cheapest edge connecting Done and ToDo ($u \in \text{Done}$, $v \in \text{ToDo}$)

Done = Done $\cup \{v\}$

ToDo = ToDo $\setminus \{v\}$

$T = T \cup \{(u,v)\}$

return T

Prim's Algorithm

Done = new set containing arbitrary node s

ToDo = $V \setminus \{s\}$

T = new empty set // MST for nodes from Done.

while ToDo is non-empty

do

if no edge connects Done and ToDo

then raise error: graph not connected

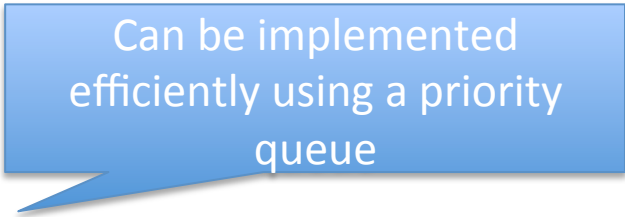
(u,v) = cheapest edge connecting Done and ToDo ($u \in \text{Done}$, $v \in \text{ToDo}$)

Done = Done $\cup \{v\}$

ToDo = ToDo $\setminus \{v\}$

$T = T \cup \{(u,v)\}$

return T

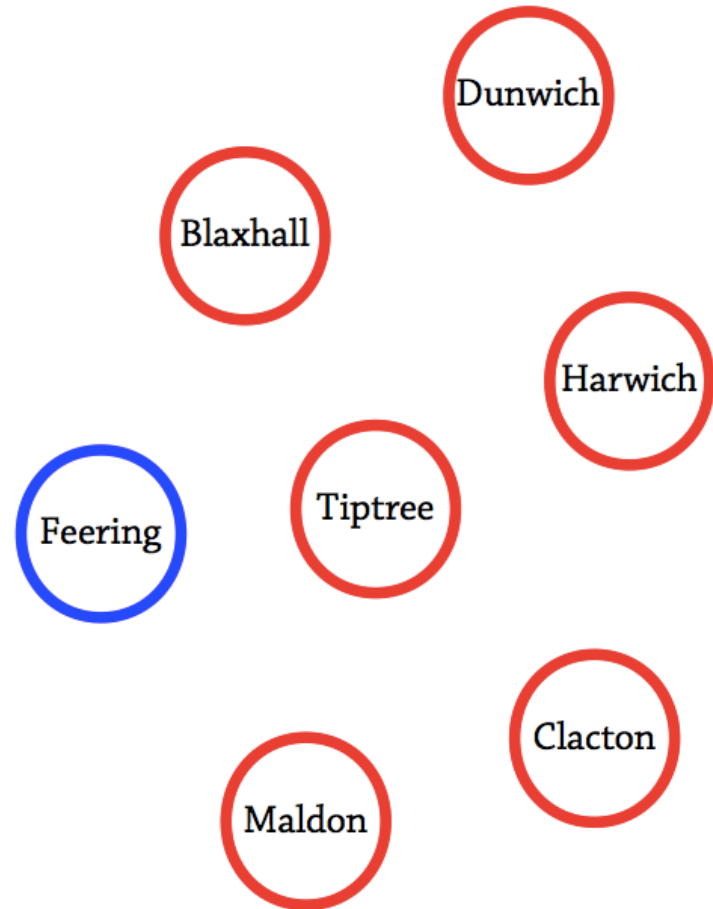
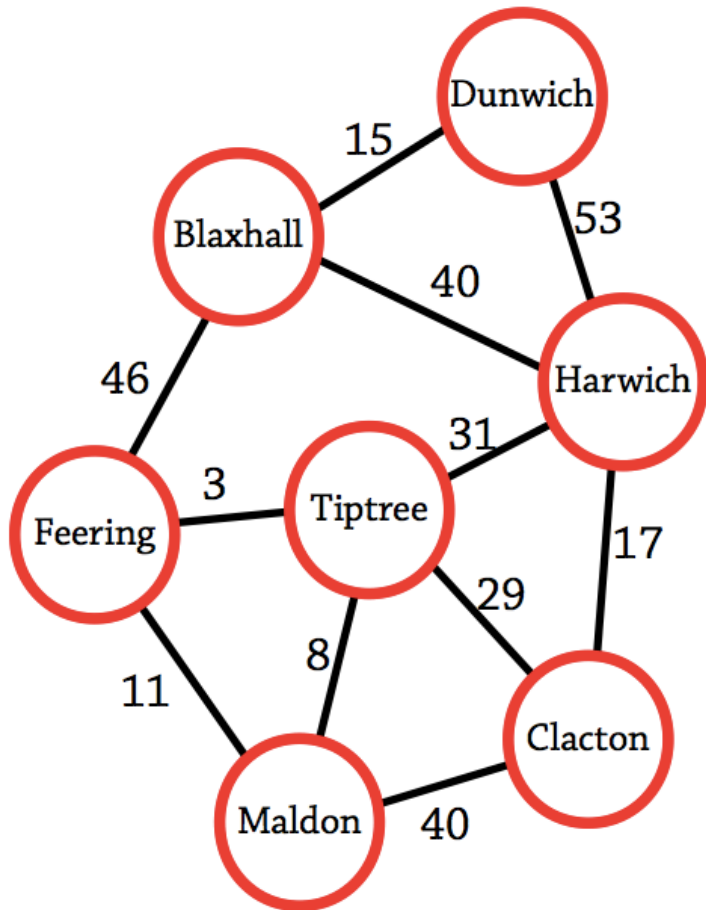


Can be implemented
efficiently using a priority
queue

Prim's Algorithm

- The operation of picking the lowest-weight edge between a node in Done and ToDo takes $O(|V|)$ time if we're not careful! Then Prim's algorithm will be $O(|V|^2)$
- To implement Prim's algorithm, use a **priority queue** containing all edges between Done and ToDo
- Whenever you add a node to Done, add all of its edges to nodes in ToDo to a priority queue
- To find the lowest-weight edge, just find the minimum element of the priority queue
- Just like in Dijkstra's algorithm, the priority queue might return an edge between two elements that are now in Done: ignore it
- New time: $O(|V| \log |E|)$, assuming $|E| = O(|V|)$

Prim's Algorithm - Example

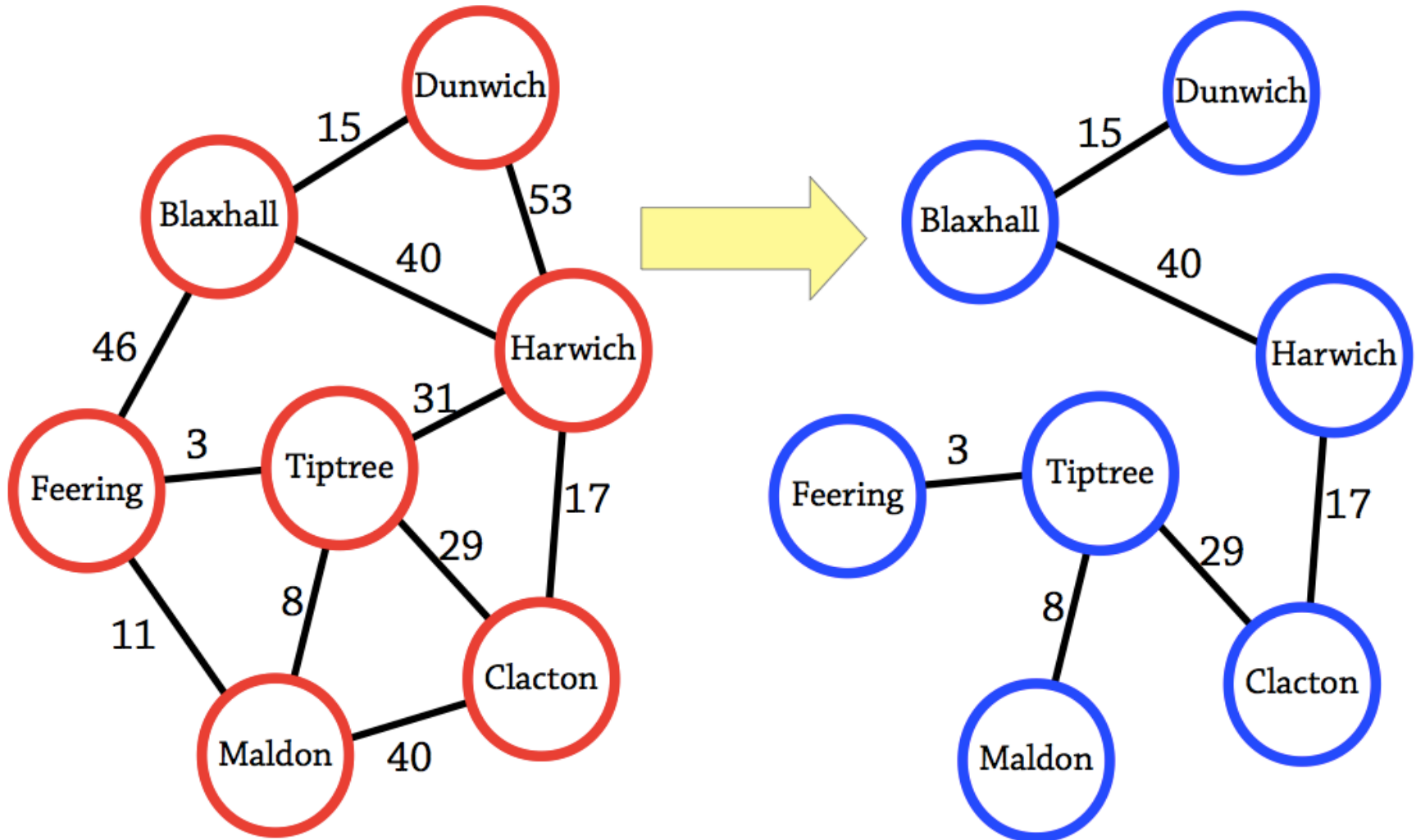


Prim's Algorithm - Example

Fast forward to the end
(demonstration on the board)



Prim's Algorithm - Example



Question

- Consider the following statements:
 - A connected graph always has an MST
 - By adding an edge to an MST we always get exactly one cycle
 - There is exactly one MST for each graph
 - In Prim's algorithm, the total cost of the result doesn't depend on the choice of the starting node

How many statements are true ?

1. 1
2. 2
3. 3
4. 4.

govote.at
Code 486103

Prim's Algorithm

- Chooses the edge with lowest cost at each step **greedy algorithm**
 - usually most efficient
 - need correctness proof

Prim's Algorithm - Correctness

- Lemma 1

The result of Prim's algorithm is a spanning tree.

Proof:

Invariant: At every step, T is a spanning tree for D_{done}

In the beginning: T contains $\{s\}$ and no edge – trivially true

After adding each new node: Assuming that T was a spanning tree for D_{done} , by adding one new node (not connected by any edge before) and one edge connecting it to D_{done} we get a tree.

In the end: T is a spanning tree for $D_{\text{done}} = V$

Prim's Algorithm - Correctness

- T , built by Prim's algorithm is an MST

Proof:

Invariant: T built at each step is a part of an MST

In the beginning: T contains $\{s\}$ and no edge – trivially true

Prim's Algorithm - Correctness

- T , built by Prim's algorithm is an MST

Proof:

After adding each new node:

We assume that T is a subtree of an MST, M

We add edge $k = (u,v)$, where v not in Done.

If k not in M , then there is another edge k' in M , such as $T \cup \{k'\}$ has a cycle C .

Then $(T \setminus \{k\}) \cup \{k'\}$ is a subtree of M .

So we can build $M' = M \setminus \{k'\} \cup \{k\}$ – also a tree, but with lower cost, because $k \leq k'$ and M' is a spanning tree also.

In the end: T is an MST

Kruskal's Algorithm

- Start with the set of all nodes and no edges
- At each point choose an edge with the lowest cost which doesn't create a cycle

Kruskal's Algorithm

- More intuitive 😊
- Harder to implement 😞
 - how should we represent the set of edges efficiently ?
 - how do we know which edges create a cycle ?

Extra material: Disjoint Set ADT

- Chapter 8 (8.1-8.3) – course book
- Operations:
 - $\text{find}(i)$ – the set where i belongs
 - $\text{union}(x,y)$ – union of two disjoint sets

Amortised $O(1)$ complexity for both

Kruskal's Algorithm

if $|V| \leq 1$ then return empty set

Partition = new disjoint-set($|V|$)

Edges = new priority queue containing E, prioritised by edge weight

T = new empty set

while Edges is non-empty

do

(u,v) = Edges.delete-min()

if Partition.find(u) \neq Partition.find(v) **then**

 Partition.union(u,v)

$T = T \cup \{(u,v)\}$

if $|T| == |V| - 1$ **then return** T

raise error: graph not connected

Question

What is the time complexity of Kruskal's algorithm ?

1. $O(|E| \log |V|)$
2. $O(|V|^2)$
3. $O(|E| \log |E|)$
4. $O(|V| \log |E|)$

govote.at
Code 81697

Kruskal's Algorithm

if $|V| \leq 1$ then return empty set

Partition = new disjoint-set($|V|$) $O(|V|)$

Edges = new priority queue containing E , prioritised by edge weight $O(|E|)$

T = new empty set

while Edges is non-empty $O(|E|)$

do

(u,v) = Edges.delete-min() $O(\log |E|)$

if Partition.find(u) \neq Partition.find(v) **then**

 Partition.union(u,v)

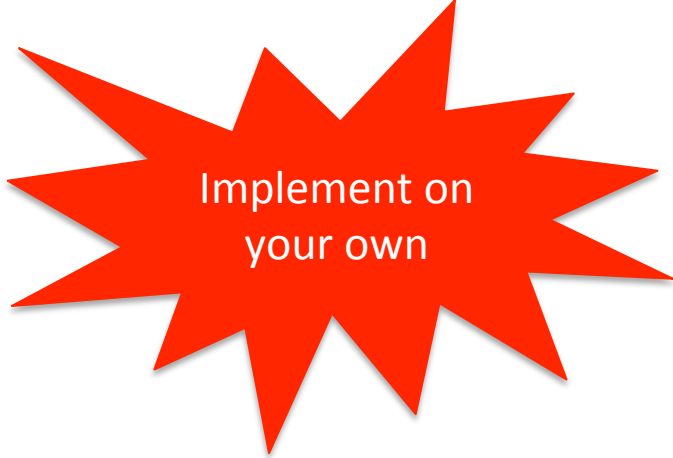
$T = T \cup \{(u,v)\}$

if $|T| == |V| - 1$ **then return** T

raise error: graph not connected

Kruskal's Algorithm - Correctness

- Similar to Prim's algorithm
- Same lemma
- Same invariant



Implement on
your own

To Do

Read from the book:

+ 9.5

Implement:

+ Prim's and Kruskal's algorithm in your favourite programming language

Dugga review:

+ here (HB3), after the class

Coming up:

- + advanced data structures
 - balanced trees
 - more on sorting

