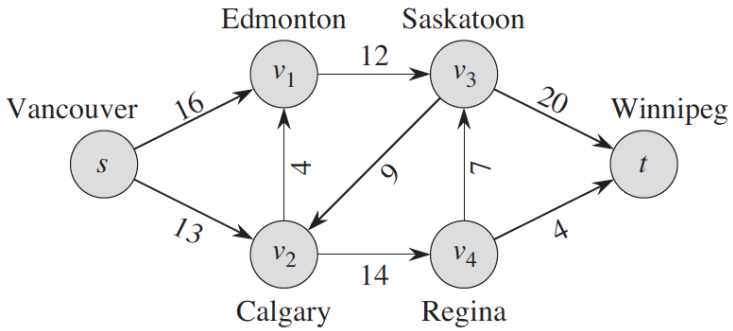
COMP 761: Lecture 26 – Graph Algorithms III

David Rolnick

November 4, 2020

Problem

A company would like to transport some goods from Vancouver to Winnipeg, but the maximum amount they can ship between any two cities en route is limited by the capacity of their trucks, as shown in the graph. How many of these items can they ship?



(Please don't post your ideas in the chat just yet, we'll discuss the problem soon in class.)

Course Announcements

Course Announcements

- Office hours: Vincent 10:30 am Thu, David 10:00 am Fri

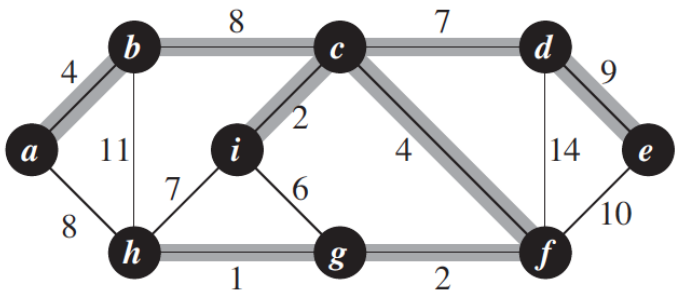
Spanning trees

Spanning trees

- Here we will be working with undirected, weighted graphs.

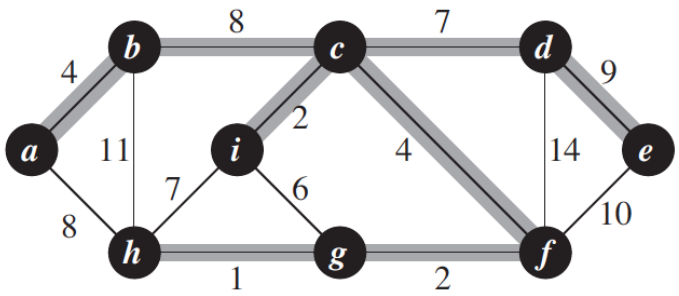
Spanning trees

- Here we will be working with undirected, weighted graphs.
- A *spanning tree* in a graph G is a subset of the edges which (i) cover all the vertices of the graph, (ii) form a tree (i.e. no cycles).



Spanning trees

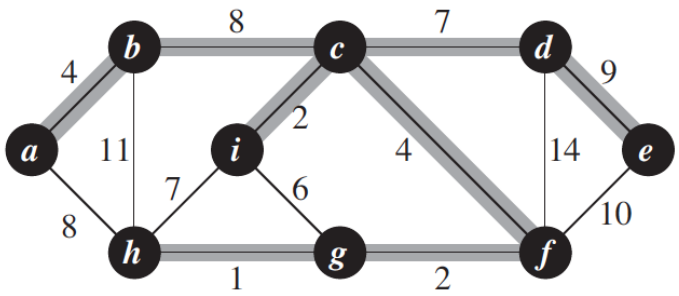
- Here we will be working with undirected, weighted graphs.
- A *spanning tree* in a graph G is a subset of the edges which (i) cover all the vertices of the graph, (ii) form a tree (i.e. no cycles).



- Every connected graph has at least one spanning tree – can just delete edges one at a time, making sure not to make the graph disconnected.

Spanning trees

- Here we will be working with undirected, weighted graphs.
- A *spanning tree* in a graph G is a subset of the edges which (i) cover all the vertices of the graph, (ii) form a tree (i.e. no cycles).



- Every connected graph has at least one spanning tree – can just delete edges one at a time, making sure not to make the graph disconnected.
- Adding any new edge to a spanning tree must make it have a cycle.

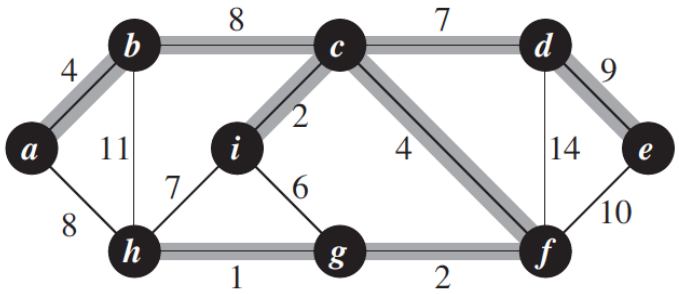
Minimal spanning trees

Minimal spanning trees

- The weight of a spanning tree is the sum of the edge weights.

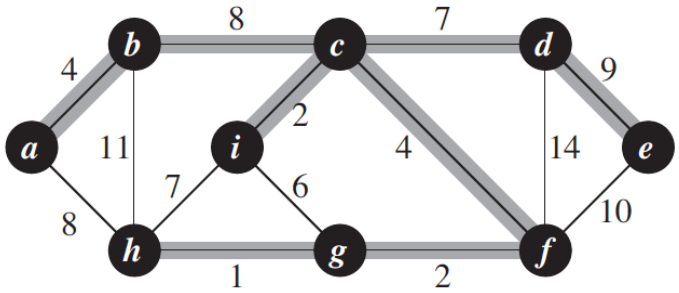
Minimal spanning trees

- The weight of a spanning tree is the sum of the edge weights.
- Problem of finding the *minimal spanning tree* in a weighted, undirected graph that is connected:



Minimal spanning trees

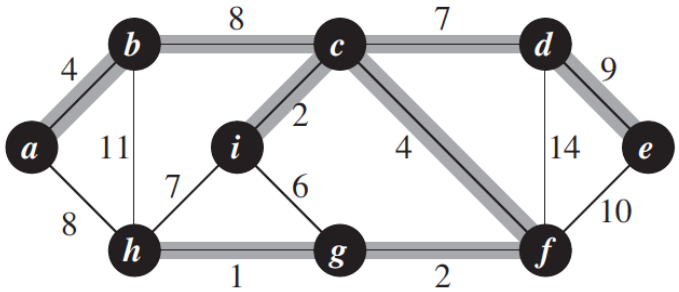
- The weight of a spanning tree is the sum of the edge weights.
- Problem of finding the *minimal spanning tree* in a weighted, undirected graph that is connected:



- Applications include designing circuits, building road networks, finding taxonomic trees, etc.

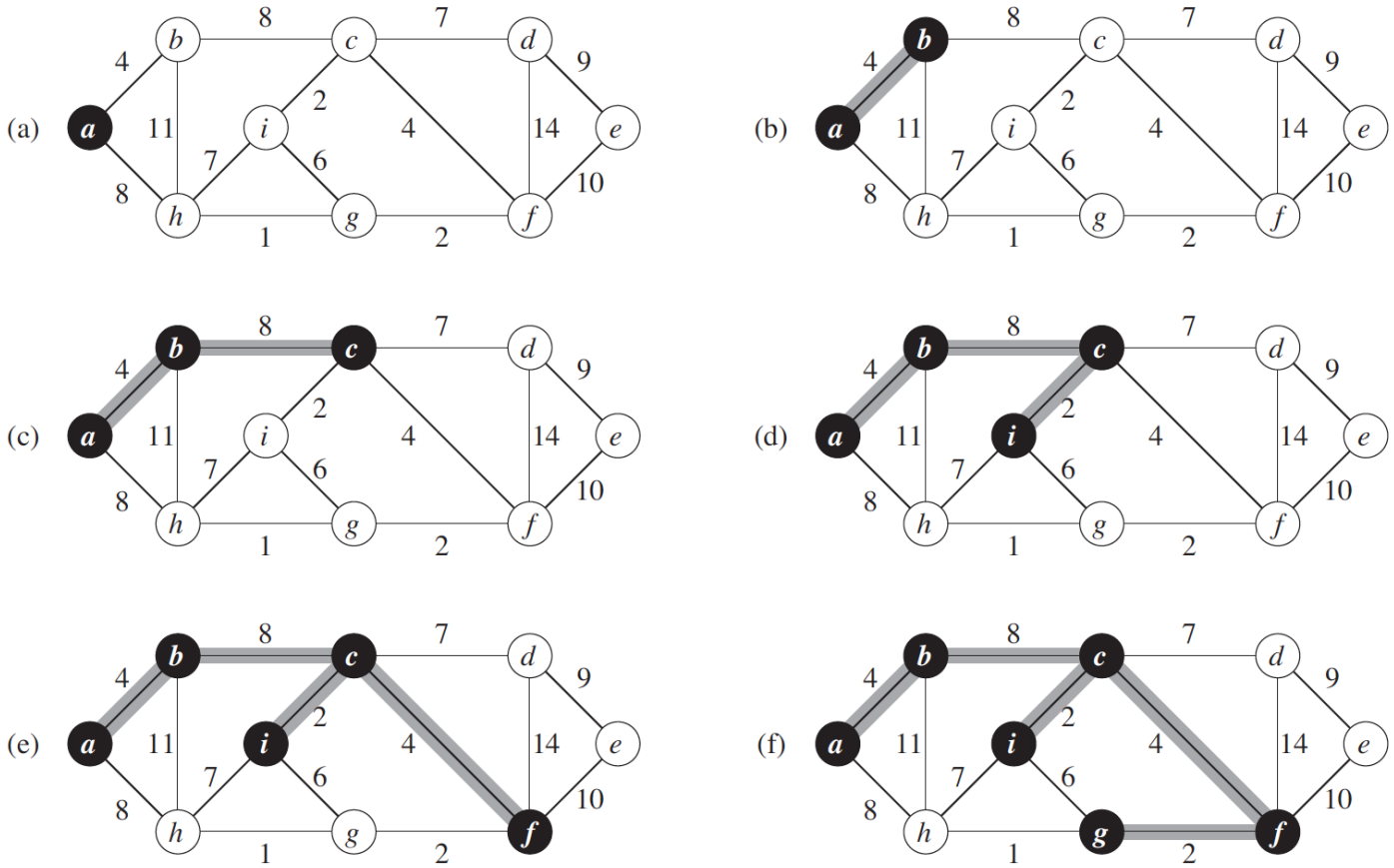
Minimal spanning trees

- The weight of a spanning tree is the sum of the edge weights.
- Problem of finding the *minimal spanning tree* in a weighted, undirected graph that is connected:

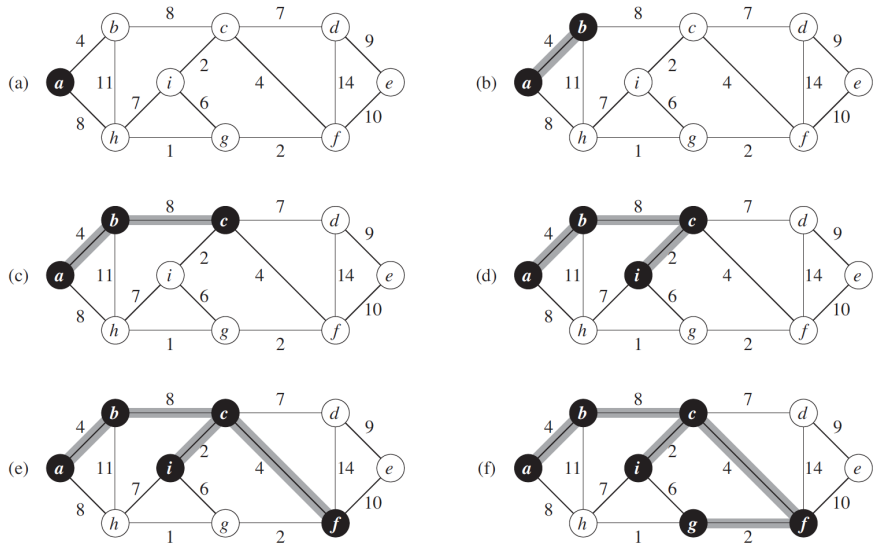


- Applications include designing circuits, building road networks, finding taxonomic trees, etc.
- Biological solution to a similar problem by slime molds:

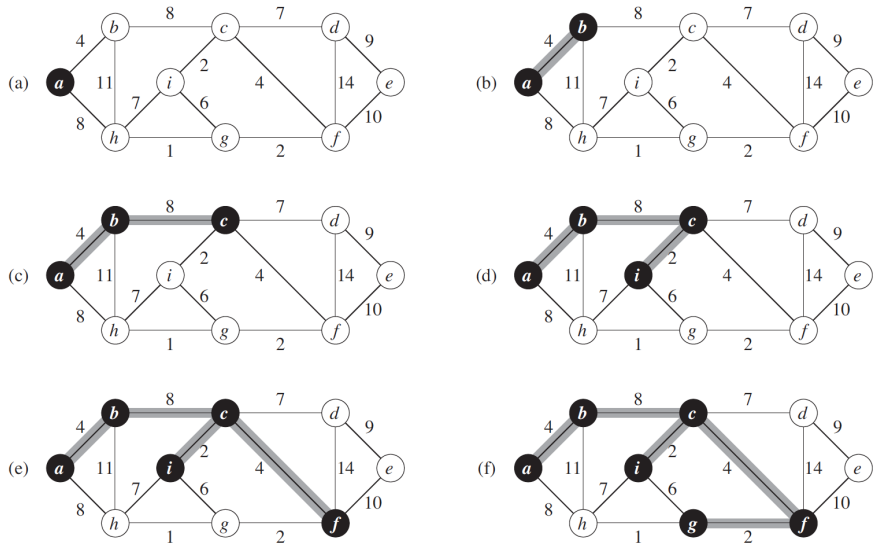
Prim's algorithm



Prim's algorithm

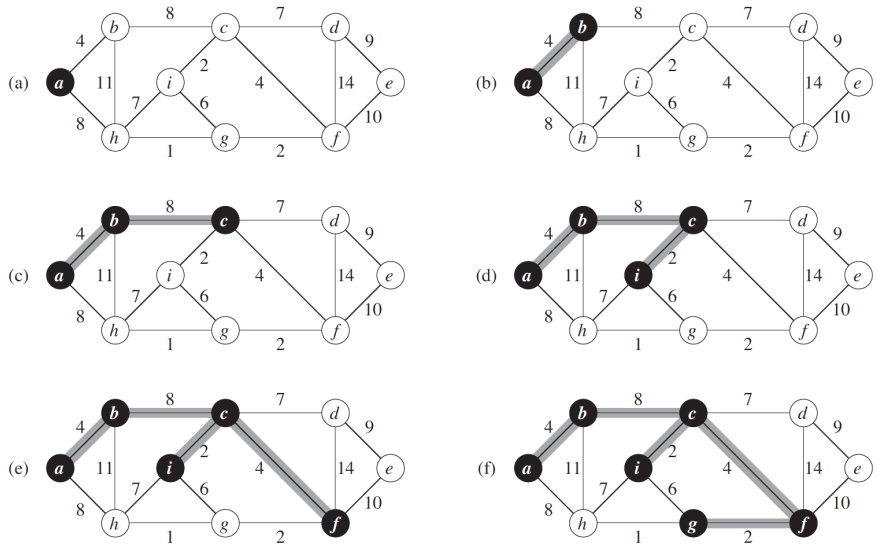


Prim's algorithm



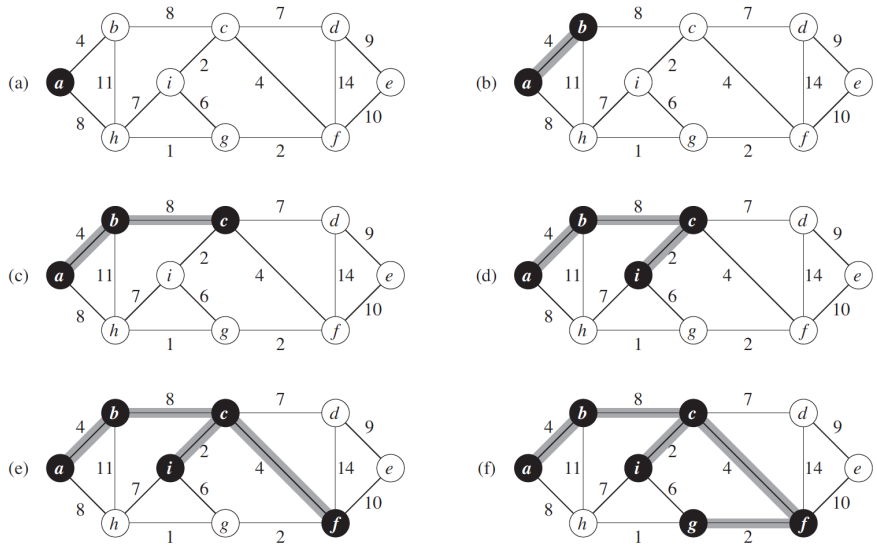
- Grow a subtree T_i, one vertex at a time.

Prim's algorithm



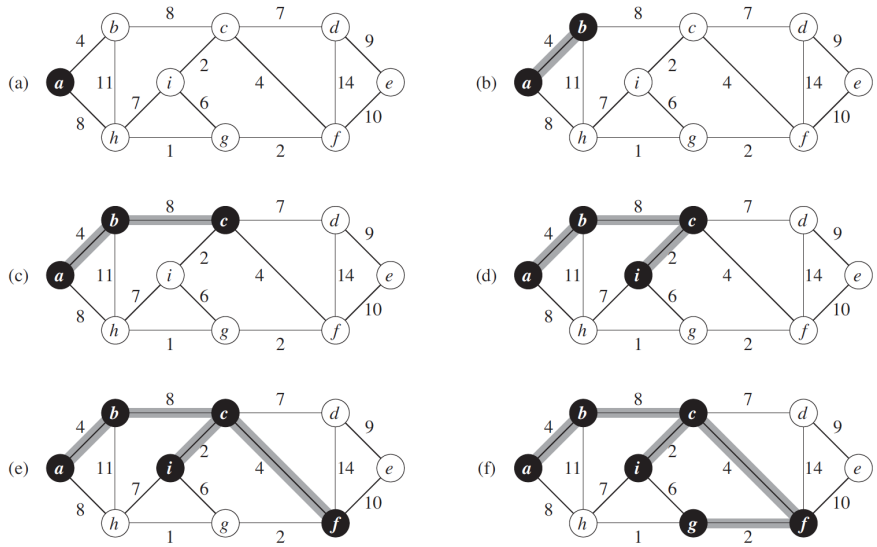
- Grow a subtree T_i, one vertex at a time.
- At step i, consider edges from the vertices we already have to new vertices.

Prim's algorithm



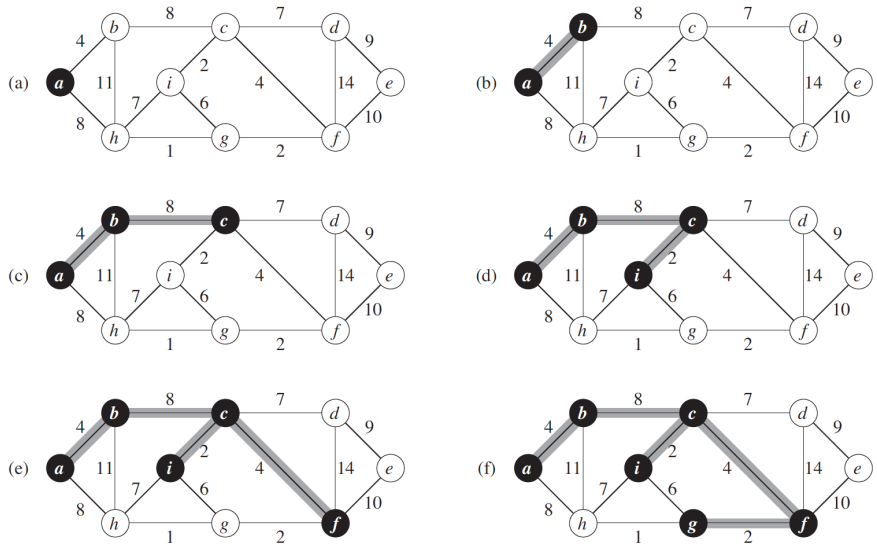
- Grow a subtree T_i, one vertex at a time.
- At step i, consider edges from the vertices we already have to new vertices.
- Take the edge with lowest weight.

Prim's algorithm



- Grow a subtree T_i, one vertex at a time.
- At step i, consider edges from the vertices we already have to new vertices.
- Take the edge with lowest weight.
- Add that edge and new vertex to T_i to get T_{i+1}.

Prim's algorithm



- Grow a subtree T_i, one vertex at a time.
- At step i, consider edges from the vertices we already have to new vertices.
- Take the edge with lowest weight.
- Add that edge and new vertex to T_i to get T_{i+1}.
- Stop when no more vertices to add.

Proof of correctness

Proof of correctness

- First, how do we know that this gives us a spanning tree? (not necessarily minimal)

Proof of correctness

- First, how do we know that this gives us a spanning tree? (not necessarily minimal)
- We add a new edge only if the vertex is new too, so no cycles = it's a tree.

Proof of correctness

- First, how do we know that this gives us a spanning tree? (not necessarily minimal)
- We add a new edge only if the vertex is new too, so no cycles = it's a tree.
- It's spanning because we stop only when we have all vertices.

Proof of correctness

- First, how do we know that this gives us a spanning tree? (not necessarily minimal)
- We add a new edge only if the vertex is new too, so no cycles = it's a tree.
- It's spanning because we stop only when we have all vertices.
- Why is it minimal weight? What proof technique can we use?

Proof of correctness

- Let's use induction.

Proof of correctness

- Let's use induction.
- Inductive hypothesis: at each step, T_i is a subset of a minimal spanning tree for G.

Proof of correctness

- Let's use induction.
- Inductive hypothesis: at each step, T_i is a subset of a minimal spanning tree for G.
- Base case of 1 vertex is easy.

Proof of correctness

- Let's use induction.
- Inductive hypothesis: at each step, T_i is a subset of a minimal spanning tree for G.
- Base case of 1 vertex is easy.
- Suppose true for T_k, subset of a minimal spanning tree M.

Proof of correctness

- Let's use induction.
- Inductive hypothesis: at each step, T_i is a subset of a minimal spanning tree for G.
- Base case of 1 vertex is easy.
- Suppose true for T_k, subset of a minimal spanning tree M.
- Suppose we get T_{k+1} by adding the edge e.

Proof of correctness

- Let's use induction.
- Inductive hypothesis: at each step, T_i is a subset of a minimal spanning tree for G.
- Base case of 1 vertex is easy.
- Suppose true for T_k, subset of a minimal spanning tree M.
- Suppose we get T_{k+1} by adding the edge e.
- If e is in M, then we are done.

Proof of correctness

- Let's use induction.
- Inductive hypothesis: at each step, T_i is a subset of a minimal spanning tree for G.
- Base case of 1 vertex is easy.
- Suppose true for T_k, subset of a minimal spanning tree M.
- Suppose we get T_{k+1} by adding the edge e.
- If e is in M, then we are done.
- Otherwise, think about adding e to M – this must make a cycle since M is a spanning tree.

Proof of correctness

- Let's use induction.
- Inductive hypothesis: at each step, T_i is a subset of a minimal spanning tree for G.
- Base case of 1 vertex is easy.
- Suppose true for T_k, subset of a minimal spanning tree M.
- Suppose we get T_{k+1} by adding the edge e.
- If e is in M, then we are done.
- Otherwise, think about adding e to M – this must make a cycle since M is a spanning tree.
- There must be another edge e' in this cycle that we could have added instead of e.

Proof of correctness

- Let's use induction.
- Inductive hypothesis: at each step, T_i is a subset of a minimal spanning tree for G.
- Base case of 1 vertex is easy.
- Suppose true for T_k, subset of a minimal spanning tree M.
- Suppose we get T_{k+1} by adding the edge e.
- If e is in M, then we are done.
- Otherwise, think about adding e to M – this must make a cycle since M is a spanning tree.
- There must be another edge e' in this cycle that we could have added instead of e.
- Since we didn't add it, we have $w(e') \geq w(e)$.

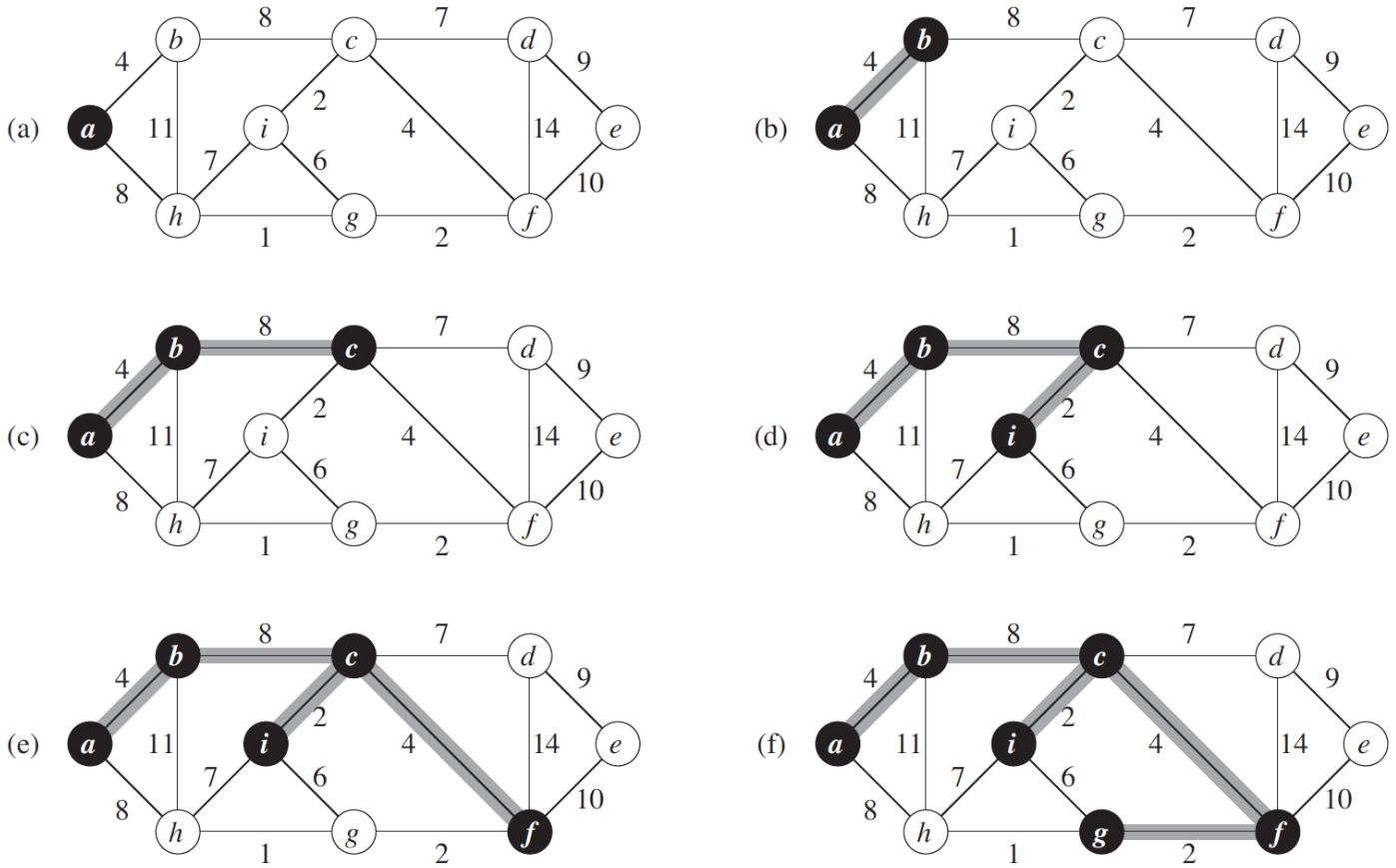
Proof of correctness

- Let's use induction.
- Inductive hypothesis: at each step, T_i is a subset of a minimal spanning tree for G.
- Base case of 1 vertex is easy.
- Suppose true for T_k, subset of a minimal spanning tree M.
- Suppose we get T_{k+1} by adding the edge e.
- If e is in M, then we are done.
- Otherwise, think about adding e to M – this must make a cycle since M is a spanning tree.
- There must be another edge e' in this cycle that we could have added instead of e.
- Since we didn't add it, we have $w(e') \geq w(e)$.
- Swapping e for e' in M gives a spanning tree M' with $w(M') \leq w(M)$.

Proof of correctness

- Let's use induction.
- Inductive hypothesis: at each step, T_i is a subset of a minimal spanning tree for G.
- Base case of 1 vertex is easy.
- Suppose true for T_k, subset of a minimal spanning tree M.
- Suppose we get T_{k+1} by adding the edge e.
- If e is in M, then we are done.
- Otherwise, think about adding e to M – this must make a cycle since M is a spanning tree.
- There must be another edge e' in this cycle that we could have added instead of e.
- Since we didn't add it, we have $w(e') \geq w(e)$.
- Swapping e for e' in M gives a spanning tree M' with $w(M') \leq w(M)$.
- This is a contradiction, so e is in M and the inductive step holds.

Prim's algorithm



Implementation

Implementation

- How shall we implement this?

Implementation

- How shall we implement this?
- Keep min-priority queue of vertices haven't added.

Implementation

- How shall we implement this?
- Keep min-priority queue of vertices haven't added.
- For each one, the key is the minimal edge to the vertices we have added.

Implementation

- How shall we implement this?
- Keep min-priority queue of vertices haven't added.
- For each one, the key is the minimal edge to the vertices we have added.
- New vertex to add: ExtractMin.

Implementation

- How shall we implement this?
- Keep min-priority queue of vertices haven't added.
- For each one, the key is the minimal edge to the vertices we have added.
- New vertex to add: ExtractMin.
- When add new vertex, DecreaseKey on any neighbors that are now closer.

Implementation

- How shall we implement this?
- Keep min-priority queue of vertices haven't added.
- For each one, the key is the minimal edge to the vertices we have added.
- New vertex to add: ExtractMin.
- When add new vertex, DecreaseKey on any neighbors that are now closer.
- How long does this take (using a min-heap as a min-priority queue)?

Implementation

- How shall we implement this?
- Keep min-priority queue of vertices haven't added.
- For each one, the key is the minimal edge to the vertices we have added.
- New vertex to add: ExtractMin.
- When add new vertex, DecreaseKey on any neighbors that are now closer.
- How long does this take (using a min-heap as a min-priority queue)?
- $O(|V|)$ to make the heap.

Implementation

- How shall we implement this?
- Keep min-priority queue of vertices haven't added.
- For each one, the key is the minimal edge to the vertices we have added.
- New vertex to add: ExtractMin.
- When add new vertex, DecreaseKey on any neighbors that are now closer.
- How long does this take (using a min-heap as a min-priority queue)?
- $O(|V|)$ to make the heap.
- $O(|V|)$ calls to ExtractMin, each $O(\log |V|)$.

Implementation

- How shall we implement this?
- Keep min-priority queue of vertices haven't added.
- For each one, the key is the minimal edge to the vertices we have added.
- New vertex to add: ExtractMin.
- When add new vertex, DecreaseKey on any neighbors that are now closer.
- How long does this take (using a min-heap as a min-priority queue)?
- $O(|V|)$ to make the heap.
- $O(|V|)$ calls to ExtractMin, each $O(\log |V|)$.
- $O(|E|)$ calls to DecreaseKey, each $O(\log |V|)$.

Implementation

- How shall we implement this?
- Keep min-priority queue of vertices haven't added.
- For each one, the key is the minimal edge to the vertices we have added.
- New vertex to add: ExtractMin.
- When add new vertex, DecreaseKey on any neighbors that are now closer.
- How long does this take (using a min-heap as a min-priority queue)?
- $O(|V|)$ to make the heap.
- $O(|V|)$ calls to ExtractMin, each $O(\log |V|)$.
- $O(|E|)$ calls to DecreaseKey, each $O(\log |V|)$.
- Total time:

$$O((|V| + |E|) \log |V|) = O(|E| \log |V|).$$

Implementation

- How shall we implement this?
- Keep min-priority queue of vertices haven't added.
- For each one, the key is the minimal edge to the vertices we have added.
- New vertex to add: ExtractMin.
- When add new vertex, DecreaseKey on any neighbors that are now closer.
- How long does this take (using a min-heap as a min-priority queue)?
- $O(|V|)$ to make the heap.
- $O(|V|)$ calls to ExtractMin, each $O(\log |V|)$.
- $O(|E|)$ calls to DecreaseKey, each $O(\log |V|)$.
- Total time:

$$O((|V| + |E|) \log |V|) = O(|E| \log |V|).$$

- As with Dijkstra, faster implementation exists: $O(|V| \log |V| + |E|)$.

Flow networks

Flow networks

- We have seen problems (shortest path and spanning tree) where weights on edges in a graph are used to represent lengths.

Flow networks

- We have seen problems (shortest path and spanning tree) where weights on edges in a graph are used to represent lengths.
- They can also represent the maximum capacity on edges.

Flow networks

- We have seen problems (shortest path and spanning tree) where weights on edges in a graph are used to represent lengths.
- They can also represent the maximum capacity on edges.
- A *flow network* is a directed graph where, for each edge (u, v), there is an associated capacity $c(u, v) > 0$.

Flow networks

- We have seen problems (shortest path and spanning tree) where weights on edges in a graph are used to represent lengths.
- They can also represent the maximum capacity on edges.
- A *flow network* is a directed graph where, for each edge (u, v), there is an associated capacity $c(u, v) > 0$.
- We assume there are no self-loops (u, u).

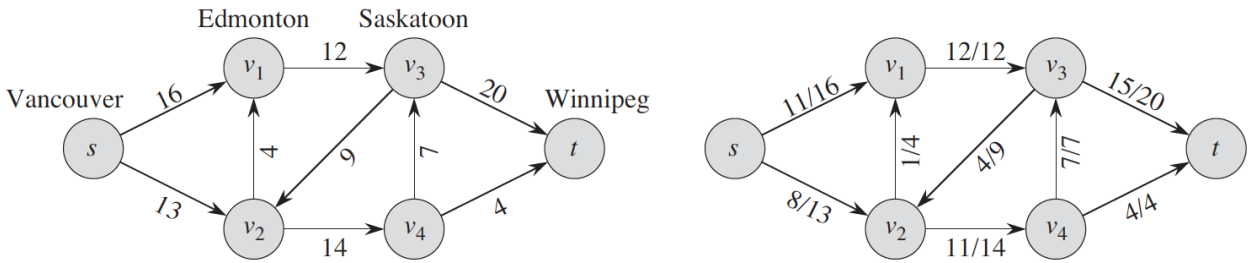
Flow networks

- We have seen problems (shortest path and spanning tree) where weights on edges in a graph are used to represent lengths.
- They can also represent the maximum capacity on edges.
- A *flow network* is a directed graph where, for each edge (u, v), there is an associated capacity $c(u, v) > 0$.
- We assume there are no self-loops (u, u).
- A *flow* from s to t is a function f where if (u, v) is an edge, then

$$f(u, v) \leq c(u, v)$$

and $f(u, v) = 0$ if there is no edge (u, v), and where for all u different from s and t,

$$\sum_v f(v, u) = \sum_v f(u, v).$$



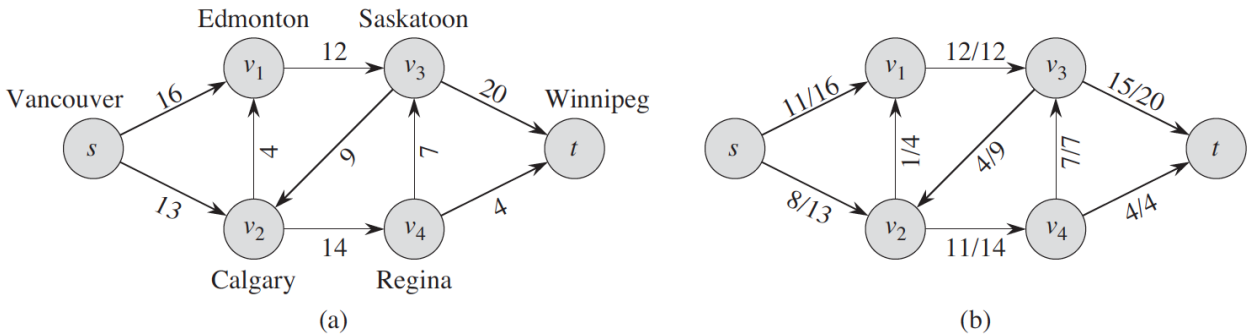
Flow networks

- A *flow* from s to t is a function f where if (u, v) is an edge, then

$$f(u, v) \leq c(u, v)$$

and $f(u, v) = 0$ if there is no edge (u, v), and where for all u different from s and t,

$$\sum_v f(v, u) = \sum_v f(u, v).$$



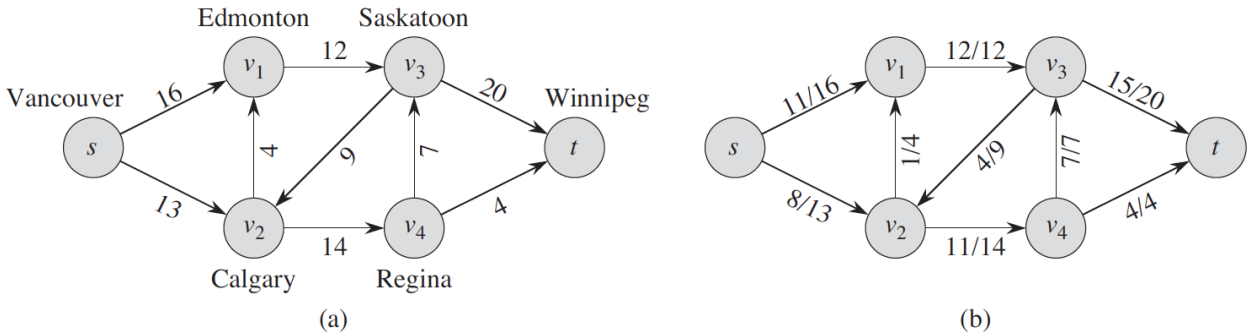
Flow networks

- A *flow* from s to t is a function f where if (u, v) is an edge, then

$$f(u, v) \leq c(u, v)$$

and $f(u, v) = 0$ if there is no edge (u, v), and where for all u different from s and t,

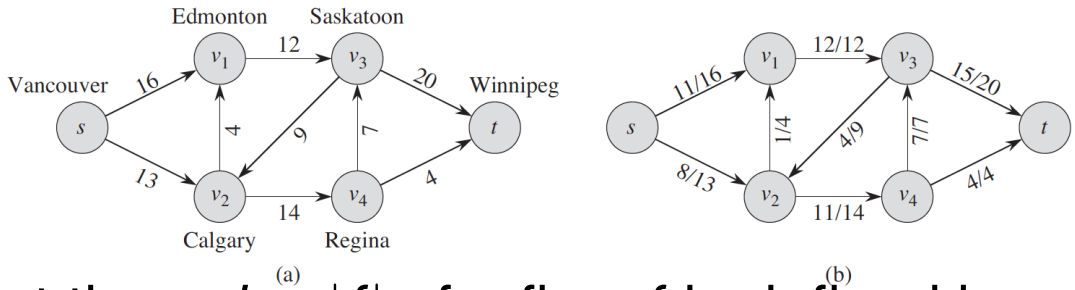
$$\sum_v f(v, u) = \sum_v f(u, v).$$



- We say that the *value* $|f|$ of a flow f is defined by:

$$|f| = \sum_v f(s, v) - \sum_v f(v, s).$$

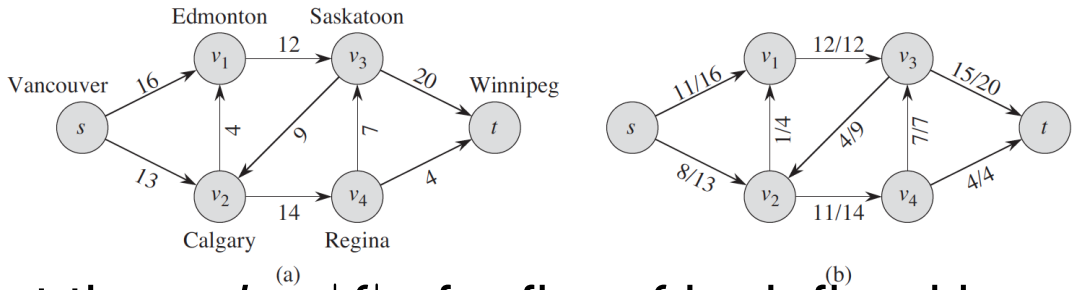
Flow networks



- We say that the *value* $|f|$ of a flow f is defined by:

$$|f| = \sum_v f(s, v) - \sum_v f(v, s).$$

Flow networks



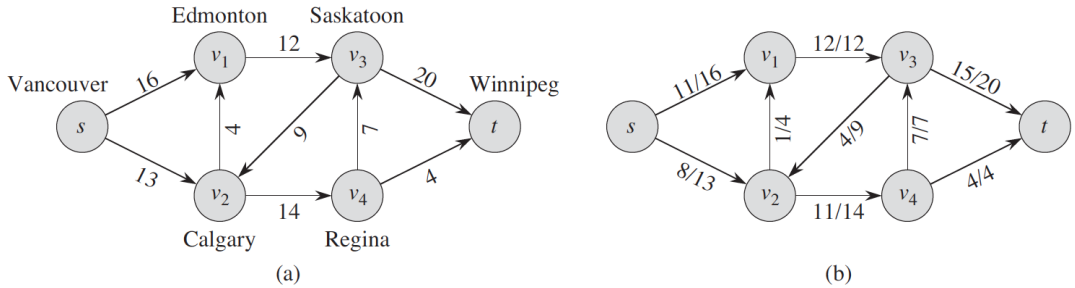
- We say that the *value* $|f|$ of a flow f is defined by:

$$|f| = \sum_v f(s, v) - \sum_v f(v, s).$$

- Why is it also true that

$$|f| = \sum_v f(v, t) - \sum_v f(t, v)?$$

Flow networks



- We have

$$0 = \sum_u \left(\sum_v f(u,v) - \sum_v f(v,u) \right)$$

$$= \left(\sum_v f(s,v) - \sum_v f(v,s) \right) + \left(\sum_v f(t,v) - \sum_v f(v,t) \right)$$

$$+ \sum_{u \neq s,t} \left(\sum_v f(u,v) - \sum_v f(v,u) \right) =$$

$$= |f| + \left(\sum_v f(t,v) - \sum_v f(v,t) \right) + \sum_{u \neq s,t} 0,$$

so $\sum_v f(t,v) - \sum_v f(v,t) = -|f|$.

Next time!

Max-flow and min-cut