# Kruskal's Algorithm for MSTs

Idea:

- build up a forest (collection of trees) that is a subset of some MST
- Add edge of minimum weight $(u, v)$ connecting two of the trees This must be safe.

  [**Proof:** Suppose $(u, v)$ is minimum-weight edge joining $T_1$ and $T_2$. Let $S$ be vertices in $T_1$, $A$ the list of edges currently in forest. $(S, V - S)$ respects $A$; $(u, v)$ is a light edge.]
- Use UNION-FIND to keep track of trees
  - UNION the two trees joined by $(u, v)$.

MST-KRUSKAL$(G, w)$ [$w$ is a weight function]

```
1  A ← ∅
2  for each vertex u ∈ V[G]
3       do MAKE-SET(v)
4  sort the edges in E by nondecreasing weight
5  for each edge (u, v) ∈ E, in order of weight [least first]
6       do if FIND(u) ≠ FIND(v)
7             then A ← A ∪ {(u, v)}
8                  UNION(FIND(u),FIND(v))
9  return A
```

Analysis:

- Sorting takes time $O(|E| \lg |E|)$
- $|V|$ MAKE-SETs + $2|E|$ FINDs + (at most) $|E|$ UNIONs takes time $O((|V| + |E|) \lg^*(|V|) + |E|)$ (using path compression)
- For a connected graph $|V| = O(|E|)$
  - actually, $|E| \geq |V| - 1$
- Thus, $O((|V|+|E|) \lg^* |V|+|E| \lg |E|) = O(|E| \lg |E|)$

# Prim's Algorithm

Idea:

- The edges of $A$ now form a single tree
- If $S$ is the set of vertices in $A$, add a minimum-weight edge $(u, v)$ crossing $(S, V - S)$
- $(S, V - S)$ respects $A$, so $(u, v)$ must be light
- Thus, $(u, v)$ safe for $A$
- Need data structure that makes it easy to select the next edge
  - Fix a vertex $r$ to be root of tree
  - Use priority queue $Q$ to hold all vertices not currently in $A$ based on key
    - Remember priority queues have the heap property
    - Put minimum element on top (not max)
  - $key[u]$ is minimum weight of edge connecting $u$ to vertex in $A$
  - $key[u] = \infty$ if there is no edge from $u$ to $A$
  - $\pi[u]$ is the parent of $u$ in $A$
  - $A = \{(v, \pi[v]) : v \in V - \{r\} - Q\}$

MST-PRIM$(G, w, r)$ $\quad$ [$r$ is the root of the MST]

```
1   Q ← V[G]
2   for each vertex u ∈ Q
3       do key[u] ← ∞
4   key[r] = 0    [initially A = {r}]
5   π[r] = NIL
6   while Q ≠ ∅
7       do u ← EXTRACT-MIN
8           for each v ∈ Adj[u]
9               do if v ∈ Q and w(u, v) < key[v]
10                  then π[v] < u
11                       key[v] ← w(u, v)
```

Some invariants:

- if $key[v] \neq \infty$ and $v \neq r$, then
  - $\pi[v] \neq$ NIL
  - $key[v] = w(v, \pi[v])$
  - $(v, \pi[v])$ is a light edge connecting $v$ to a vertex in the tree
- $\pi[v] =$ NIL iff there is no edge from $v$ to the tree

**Analysis:** Running time depends on how we implement priority queue.

If we use binary heap (Chapter 7), sorted by key:

- Initialization (lines 1-3) takes time $O(|V|)$
- Go through outer loop (lines 6-11) $|V|$ times
- EXTRACT-MIN takes $O(\lg |V|)$ each time
- Go through inner loop (lines 8-11) $O(|E|)$ times
- Can implement line 9 in constant time
  - Keep a bit to test for membership in $Q$
- Line 11 involves a DECREASE-KEY
  - May need to move item up in the priority queue when we decrease its key
  - This can be implemented in time $O(\lg |V|)$
- Running time:

  $O(|V| \lg |V| + |E| \lg |V|) = O(|E| \lg |V|) = O(|E| \lg |E|)$

  - $|V| - 1 \leq |E| \leq |V|^2$, so $O(\lg(|V|)) = O(\lg(|E|))$
- This is the same as Kruskal's algorithm

We can instead implement priority queues using *Fibonacci heaps* (Chapter 21)

- $|E|$ DECREASE-KEYs take time $O(|E|)$
  - amortized complexity of DECREASE-KEY is $O(1)$
- Total running time: $O(|V| \lg |V| + |E|)$

# Flow Networks: Motivation

Sony is trying to get Playstation 2s from the factory where they are manufactured in Shanghai to the warehouse in New York where they will be stored. Playstations can be shipped by boat or plane to the US, and from there by truck or train to New York. Customer demand is high, so Sony would like to ship as many as possible.

- There are many routes between Shanghai and New York
- The trucks/planes/boats/trains that go along each route have different capacities; the frequency that a route is traveled also varies.
    - May be only be 1 flight/day from Shanghai to SF, 2 flights/day to LA.

What's the best way for Sony to ship the Playstations so as to get as many as possible to the warehouse in New York?

- Assume that the factory in Shanghai can produce as many as necessary.

This is an instance of the *maximum flow* problem.

# Flow Networks: Basic Idea

Imagine materials "flowing" through a network.

- **Example:** Water flowing through a network of pipes.

Each edge pipe has some capacity.

- The capacity is the maximum rate at which water can flow through the pipe (e.g., 200 gallons/hour)

The *maximum flow* is the greatest rate at which material (e.g. water) can flow from a source to a sink.

Network flow problems arise everywhere:

- parts "flowing" through an assembly line
- current flowing through electrical grids
- information flowing through a network

# Flow Networks: Basic Definitions

A *flow network* $G(V, E)$ is a directed graph.

- Each edge $(u, v)$ has a capacity $c(u, v) \geq 0$.
- (If $(u, v) \notin E$, then $c(u, v) = 0$.)
- Two nodes in $G$ are distinguished:
  - the *source* $s$ and the *sink* $t$.
- Assume every node is on some path from $s$ to $t$.
  - for all $v$, there is a path $s \rightarrow v \rightarrow t$.

A *flow* is a real-valued $f : V \times V \rightarrow \mathbb{R}$ such that

- $f(u, v) \leq c(u, v)$
  - a channel can't carry more than its capacity
- $f(u, v) = -f(v, u)$
  - If $k$ units are flowing from $u$ to $v$, then $-k$ units are flowing from $v$ to $u$
  - This means that $f(u, u) = -f(u, u) = 0$.
- $\Sigma_{v \in V} f(u, v) = 0$ for all $u \in V$ other than $s, t$
  - amount flowing into $u$ = amount flowing out
  - conservation of flow: Kirchoff's Current Law

$f(u, v)$ is the *net flow* from $u$ to $v$.
The *value* of a flow $f$ is the amount flowing out of $s$:

$$|f| = \Sigma_{v \in V} f(s, v).$$

- (Note that $|f|$ is *not* the absolute value.)
- This is also the amount that will reach $t$

We are interested in finding the flow $f$ whose value is maximum.

- That's the *maximum flow* problem.

# Back to the Example

Sony is trying to get Playstation 2s from the factory where they are manufactured in Shanghai to the warehouse in New York where they will be stored.

- $s$ (source) is Shanghai
- $t$ (sink) is New York
- nodes are cities on some route from Shanghai to New York.
- edges represent transportation links
- capacity of a link = most Playstations that can be shipped each day along that link
    - This takes into account type of transport (train vs. truck) and frequency.

Suppose Sony uses a particular policy for shipping Playstations.

- This is represented by a flow $f$

What is the flow according to that policy?

- Suppose that 1000 Playstations/day can be shipped from Omaha to Chicago, and 800 can be shipped from Chicago to Omaha.
- On a given day, 800 Playstations are shipped from Omaha to Chicago and none are shipped in the opposite direction, then $f(O, C) = 800$ and $f(C, O) = -800$.
- If (for some silly reason) 300 are shipped back from Chicago to Omaha, $f(O, C) = 500$ (that's the net flow) and $f(C, O) = -500$.

Here's how this is indicated graphically; the first number is the flow, and the second is the capacity:

Typically we represent only positive flows.

# Multiple sources and sinks

In many applications, we have many sources and many sinks, rather than just one.

- Imagine that Sony has lots of factories and lots of warehouses.

Can easily reduce the multiple source/multiple sink problem to a single source/single sink problem.

- Just add a "virtual" supersource, connected to all the real sources by an edge with infinite capacity, and a "virtual supersink", with an edge of infinite capacity from each real sink to the supersink.
  - Any flow in one case is a flow in the other.
  - Figuring out the maximum flow in the new network will give the maximize the sum of the flows from the sources to the sinks in the original network.

# Ford-Fulkerson Method

Ford-Fulkerson is a general method for solving maximum-flow problems.

- There are different ways of implementing it (which affect the running time)

Main idea:

- Start by taking $f(u, v) = 0$ for all edges
- Try to increase the flow value by finding a path from $s$ to $t$ along which you can push more flow; this is an *augmenting path*.

FORD-FULKERSON METHOD$(G, s, t)$

```
1  initialize flow f to 0
2  while there exists an augmenting path p;
3        do augment flow f along p
4  return f
```

Problems:

- How do you find an augmenting path?
- How do you know that no augmenting path $\Rightarrow$ you have a maximum flow?

# Residual networks

Given a flow network and a flow $f$, the *residual network* describes how much more flow the network can accommodate.

- Replace the capacity $c$ by $c_f$, where

$$c_f(u, v) = c(u, v) - f(u, v).$$

- $c_f(u, v)$ must be nonnegative, since $c(u, v) \geq f(u, v)$.
- Given $G = (V, E)$ and $f$, the *residual network* is $G_f = (V, E_f)$ where $E_f$ consists of all $(u, v)$ such that $c_f(u, v) > 0$.
- $f$ can be negative, so can have $(u, v) \in E_f$ even though $(u, v) \notin E$.

**Claim:** $|E_f| \leq 2|E|$.

**Proof:** If $(u, v) \in E_f$, then either $(u, v) \in E$ or $(v, u) \in E$.

- If $c(u, v) = c(v, u) = 0$, then $f(u, v) \leq 0$ and $f(v, u) \leq 0$. Since $f(u, v) = -f(v, u)$, must have $f(u, v) = f(v, u) = 0$. But then $c_f(u, v) = 0$.
- Thus, if $(u, v) \in E_f$, then either $(u, v) \in E$ or $(v, u) \in E$, so $|E_f| \leq 2|E|$.

Given two flows $f$ and $f'$ in $G$, $f + f'$ may not be a flow in $G$.

- Problem: may have $f(u,v) + f'(u,v) > c(u,v)$.
- Note: $(f + f')(u,v) = f(u,v) + f'(u,v)$.

However, if $f$ is a flow in $G$ and $f'$ is a flow in $G_f$, then $f + f'$ is a flow in $G$. This is easy to check. E.g.:

$$\begin{aligned}(f + f')(u,v) &= f(u,v) + f'(u,v) \\ &\leq f(u,v) + c_f(u,v) \\ &= f(u,v) + c(u,v) - f(u,v) \\ &= c(u,v)\end{aligned}$$

Even easier to check that

$$(f + f')(u,v) = -(f + f')(v,u)$$
$$\Sigma_{v \in V}(f + f')(u,v) = 0 \text{ if } u \neq s,t.$$

Most important fact: $|f + f'| = |f| + |f'|$

**Proof:** Just use the definition:

$$\begin{aligned}&|f + f'| = \Sigma_{v \in V}(f + f')(s,v) \\ &= \Sigma_{v \in V} f(s,v) + \Sigma_{v \in V} f'(s,v) = |f| + |f'|.\end{aligned}$$