19 Feb 2021 More on Minimum Spanning Tree.

## Announcements

1. Late homework policy.
   You have a budget of 9 late days over the length of the semester.
   Use at most 3 per assignment.
   Advice: save them for when you need them.
2. Blank solution policy.
   Starting with Problem Set 2, blank solutions get 20% of the points.
3. Collaboration policy: everything you turn in (words and code) must be written by you.
4. Optional recommended reading:
   Chapter 4.8, on data compression.

## Returning to Minimum Spanning Tree

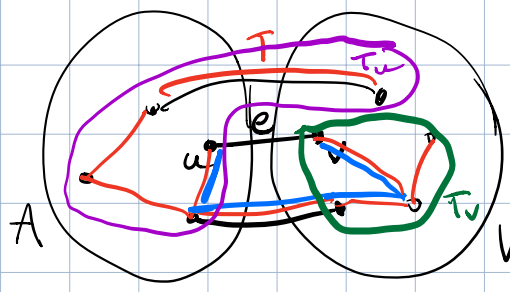
Recall: undirected connected graph $G = (V, E)$,
cost function $cost: E \to \mathbb{R}$

Goal: output an edge set that forms a spanning tree (meaning connected subgraph with no cycles) and has min total cost.

For now: Assume all edges have distinct costs.

**Cut Lemma.** If $A \subsetneq V$, $A \neq \emptyset$, then every MST contains the min cost edge from $A$ to $V-A$.

**Proof.**




Assume e is min-cost edge from A to V-A.

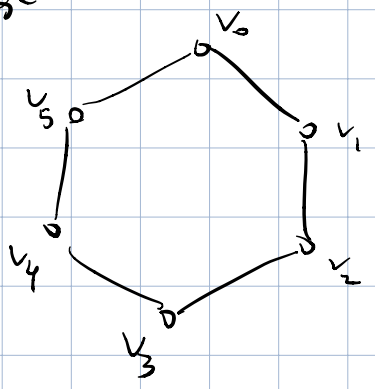
If $e \notin T$ then $T$ contains a path $P$ from $u$ to $v$.
Delete from $P$ an edge that crosses from $A$ to $V-A$, say $e'$.
This splits $T$ into a forest with 2 components, $T_u$ and $T_v$, with $u \in T_u$ and $v \in T_v$.
Rejoining $T_u$ to $T_v$ using edge $e$, we get a different spanning tree $T'$

$$\text{cost}(T') = \text{cost}(T) - \text{cost}(e') + \text{cost}(e)$$
$$\leq \text{cost}(T).$$

So $T$ is not a MST.

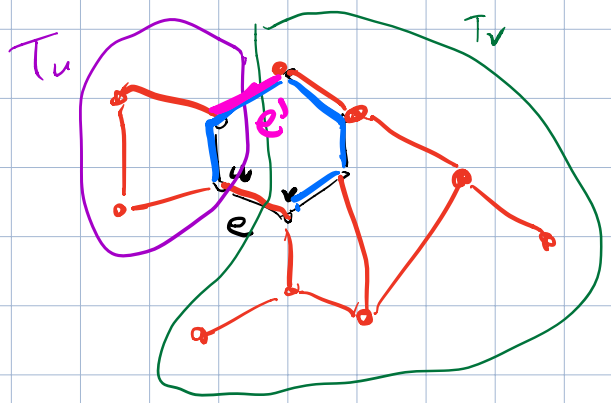
**Def.** A cycle in a graph is a sequence of distinct vertices $v_0, v_1, v_2, \ldots, v_{k-1}$ such that $(v_{i-1}, v_i)$ is an edge for each $i = 1, \ldots, k-1$, and $(v_{k-1}, v_0)$ is also an edge.

edge set of the cycle




**Cycle Lemma.** If $C$ is any cycle, and $e$ is the maximum-cost edge in its edge set, then $e$ does not belong to any MST.

**Proof.** Let $T$ be any tree containing $e$.



Delete $e$ from $T$, breaking it into components $T_u$, $T_v$ with $u \in T_u$, $v \in T_v$.

The cycle $C$ contains a path $P$ joining $u$ to $v$ without using $e$. Since $P$ starts in $T_u$, ends in $T_v$, it contains at least one edge $e'$ from $T_u$ to $T_v$.

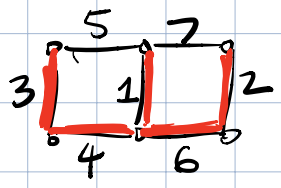
$T' = T - \{e\} \cup \{e'\}$ is a spanning tree.

$$\text{cost}(T') = \text{cost}(T) - \text{cost}(e) + \text{cost}(e')$$
$$< \text{cost}(T).$$

Other algorithms to solve MST.

① Kruskal's Algorithm.

- Sort all edges of $G$ in order of increasing cost.
- Iterate through the sorted list, selecting each edge that doesn't form a cycle with previously selected edges.



Why correct? Every time we skip an edge, it forms a cycle with previously selected (hence cheaper) edges. By Cycle Lemma, every edge that Kruskal skips has to be omitted from every MST.

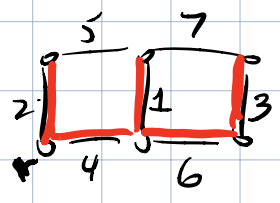
To finish up, why do the non-omitted edges form a spanning tree?

They form a subgraph with no cycles.

It is connected because for any partition of $V$ into non-empty $A$, $V-A$, the min-cost edge from $A$ to $V-A$ was inserted by Kruskal's alg.

(2) Prim's Algorithm

- start with an arbitrary root vertex, $r$, and no edges. $T = \{r\}$
- while $T$ is not yet a spanning tree:
  find min-cost edge $e=(u,v)$ from $V(T)$ to $V-V(T)$
  insert $e$ into $T$



Why correct? Every time it inserts an edge into $T$, the insertion is justified by the cut lemma.
(applied to $A = V(T)$.)

Implementing MST algorithms to run in $O(m \log n)$ time.

$m$ = # of edges

$n$ = # of vertices

For Boruvka: one phase of Boruvka's decreases # components by a factor of 2 or more.

$\Longrightarrow$ at most $\lceil \log_2(n) \rceil$ phases.

$\Longrightarrow$ Implementing one phase in $O(m)$ using DFS-inspired alg; see Ed Discussions post about Boruvka.

For Prim: to get $O(m \log n)$ use a data structure that implements a priority queue with $O(\log n)$ time per operation.

Priority Queue: a set of distinct keys with numerical values.

insert, delete,

decreaseKey(k, v): decreases the value of key k down to v.

findMin: returns key with min value.

One such implementation: balanced minHeap.

Idea: Store vertices in complement of T that have at least one edge joining them to T. Those vertices are the keys. Their values are cost of cheapest edge joining them to T.

One iteration of Prim:

find min cost edge leaving T. (one Find Min.) — 1 per iter — $O(\log n)$

insert that edge into T. — 1/iter — $O(1)$

update priority queue.

delete v — 1/iter — $O(\log n)$

insert new neighbors of v — $O(\log n)$

decreaseKey for previously discovered neighbors of v. — $O(\log n)$

$d_v$ = # of neighbors of v that was added to T

$d_v \leq n-1$ per iteration

n iterations, $O(n \log n)$ time per iteration

$\Rightarrow O(n^2 \log n)$.

In an iteration where $v$ is inserted into $T$, the time complexity of that iteration is $O(d_v \log n)$ where $d_v$ = # of neighbors of $v$.

In total, Prim's alg runs in time

$$O\left(n \log n + \left(\sum_v d_v\right) \cdot \log n\right)$$

Fact: In any graph, $\sum_v d_v = 2 \cdot (\#\text{ edges})$.

$$\therefore O(n \log n + (\sum_v d_v) \log n)$$
$$= O(n \log n + m \log n)$$
$$= O(m \log n)$$

For $O(m \log n)$ implementation of Kruskal, use Union-Find data structure in Chapter 4.6.