LECTURE 9

Announcements:

→ HW2 Grades are up

→ Exam 1 grades are up on Gradescope:

→ You have until next Wednesday to Submit regrade requests.

→ Solutions have been posted on CCLE.

→ Mean: 22.03 ; Median: 22.58

#Students between (90-100)% ~ 54 (34%)

(80-89) ~ 56 (35%

Letter grades:

Rough guidelines:

10% → A+
10% → A
10% → A-
} 30% is A's

10% → B+
} 30% is Bs

.

→ HW3 will be up today! (Due Feb 15th)

→ Quiz 5 will be up today.

Last class: Minimum Spanning Trees.

INPUT: Graph $G = (V, E)$ weighted undirected graph.
$\forall$ edge $\{u, v\}$ → cost/weight of the edge given as $c_{\{u,v\}}$.

OUTPUT: Compute a MST in G.
(Assume that G is connected!).

Many natural greedy algorithms work!

① Kruskal's Algorithm:

→ Start with empty graph

→ Repeat until you get a spanning tree:

→ Add the least weight edge that does not create cycle.

Kruskal's Algorithm:

① Set $T \leftarrow \phi$. Let $R = E$

② While $R \neq \phi$:

① Pick an edge e of least weight from R.

② Check if adding e to T creates a cycle

If NOT, ADD e to T.

③ Remove e from R.

You can improve this

HW2 Problem 4

$O(|E|)$ time!

How long does this take? $|E|$ iterations and each iteration takes $O(|E|)$ time.

Run-time: $O(|E| \cdot |E| + |E| \cdot \log|E|$

→ Can speed up Kruskal's algorithm to run in time $O(|E| \cdot \log |V|)$.

(We are not covering this :( ).

Prim's Algorithm:

(Problem 1.8 in the Exam).

① Start with some vertex s.

Add a vertex with least "attachment cost"

Full algorithm:

① Set $T = \phi$. $S = \{s\}$. $c'(s) = 0$, $c'(v) = \infty \; \forall v \neq s$.

② While $S \neq V$:

(i) For each $v \notin S$, compute

$$c'(v) = \min_{u : u \in S} c_{\{u,v\}}$$

(ii) Pick the vertex v with the least attachment cost c'. Let u be the vertex achieving the minimum in the definition of c'.

(iii) Add $\{u,v\}$ to T. Add v to S.

Prim's Algorithm:

(Problem 1.8 in the Exam).

① Start with some vertex S.

Add a vertex with least **"attachment cost"**

Full algorithm:

① Set $T = \phi$. $S = \{s\}$. $c'(s) = 0$, $c'(v) = \infty \ \forall v \neq s$.

② While $S \neq V$:

(i) For each $v \notin S$, compute $c'(v) = \min_{u : u \in S} c_{\{u,v\}}$ ↗ $\min_{u \in S} d(u) + c_{\{u,v\}}$

(ii) Pick the vertex v with the least attachment cost c'. Let u be the vertex achieving the minimum in the definition of c'.

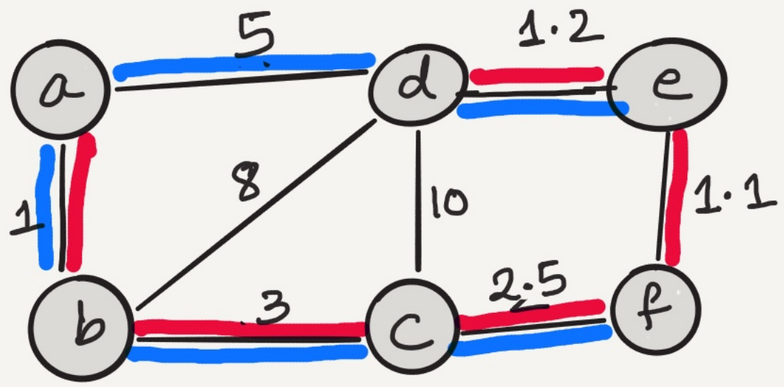
(iii) Add $\{u,v\}$ to T. Add v to S.

Time-Complexity : $O(|V| \cdot |E|)$.

You can also improve it to $O(|E| \cdot \log|V|)$

Prim's Vs Dijkstra's

Prim:	Dijkstra:
Add vertex $v \notin S$ least "attachment cost"	Add vertex $v \notin S$ That is "closest" to start



Start = {a}

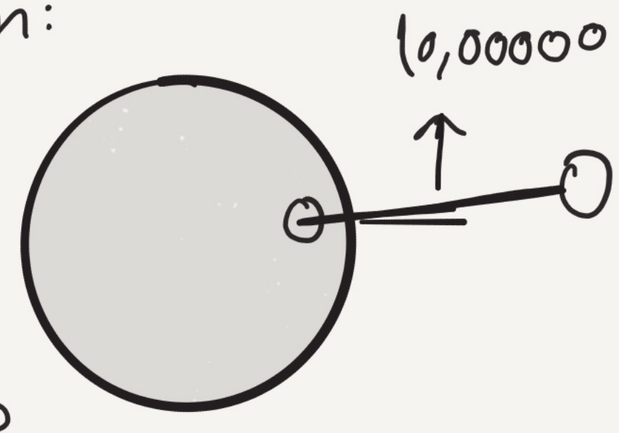
PRIM's Adds: (a, b, c, f, e, d) (the tree is shown in red).

DIJKSTRA's ADDS: (a, b, c, d, e, f) (the tree is shown in blue)

wt(PRIM's TREE) = 8.8

wt(DIJKSTRA's TREE) = 12.7

Reverse-Delete ("Reverse-Kruskal's") Algorithm:



① Set $T = (V, E)$, $R = E$

② While $R \neq \phi$:

(i) Pick the heaviest edge e in R

(ii) If deleting e from T does not disconnect T, remove e from T.

(iii) Remove e from R.

Can use BFS/DFS to check. ←

Analysis: Why do these algorithms work?

Simplifying assumption: All edges have distinct weights in the graph G_1.

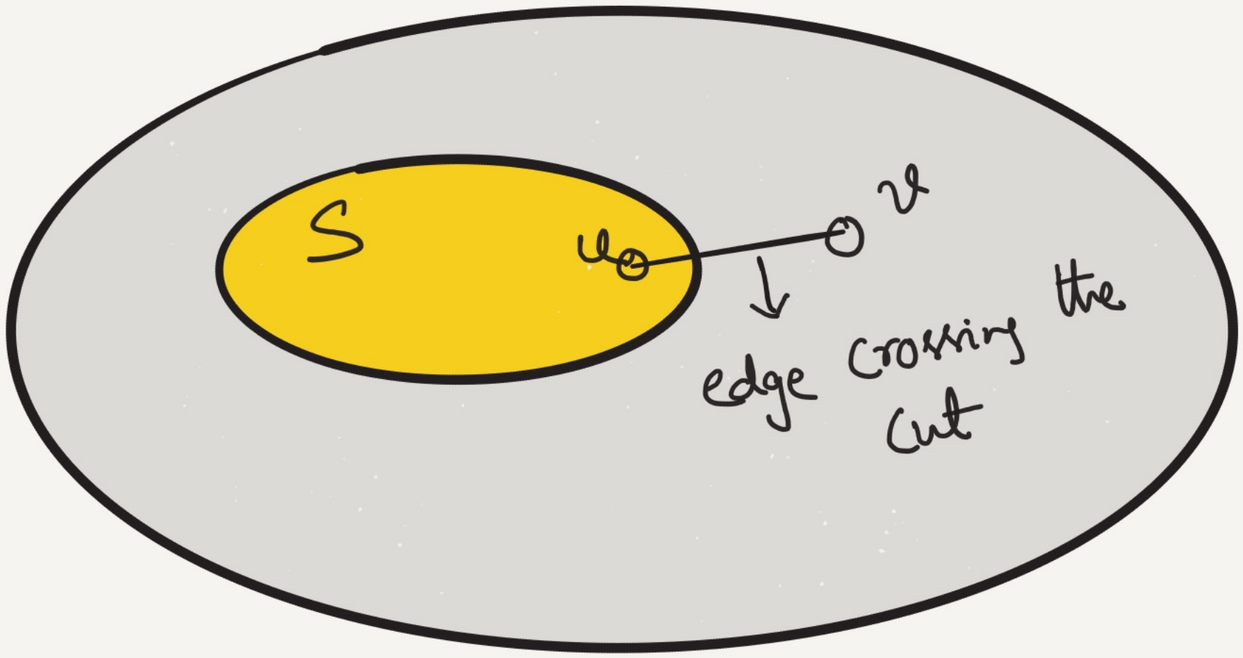
One magical property: **CUT PROPERTY**

→ "Exchange method"

→ "Safe choices" → we only make safe choices.

CUT PROPERTY:

CUT: Any subset $S \subseteq V$, $(S \neq \phi$ or $V)$.
An edge (u, v) with $u \in S$, $v \notin S$ is called an edge that <u>crosses</u> the cut.



CUT PROPERTY:

Take any cut $S \subseteq V$. Let e be the minimum weight edge that crosses the cut.

Then, e must be in the MST.

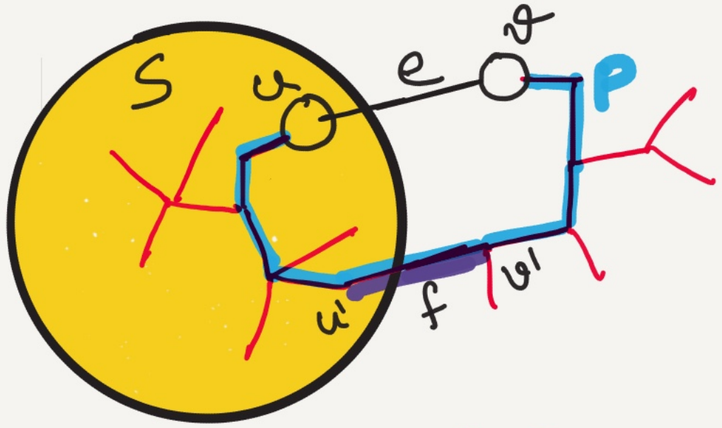
"This minimum weight edge crossing the cut is safe to add".

(Recall: All weights are distinct.)

Proof: Subtle "exchange argument.

Proof by Contradiction.

Suppose there exists some cut S where the minimum weight edge $e = \{u, v\}$ crossing the cut is not part of a MST T.

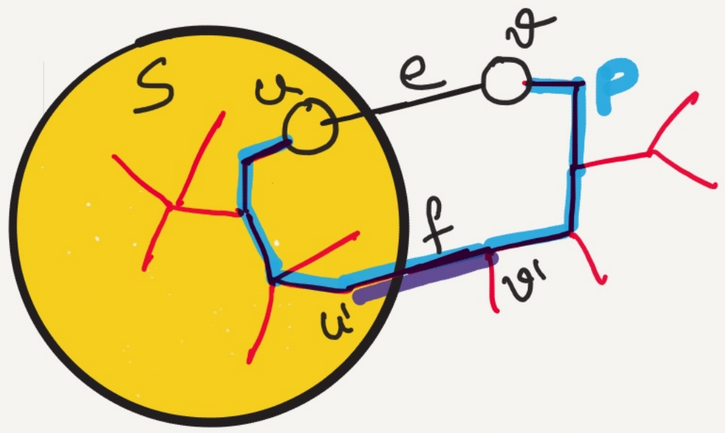


Edges of T in RED
P highlighted in BLUE
f shown in PURPLE

① Since T spans everything u, v should be part of the tree T.

$\Rightarrow$ ∃ some path P from u to v in T.

② ∃ some edge $f = \{u', v'\}$ on P crossing the cut.



Edges of T in RED

P highlighted in BLUE

Consider T' obtained by removing e and adding f:

$$T' = T \cup \{e\} \setminus \{f\}$$

Observation 1: $c_e < c_f$. (As e was the min. weight crossing edge.

Observation 2: $\text{weight}(T') < \text{weight}(T)$.

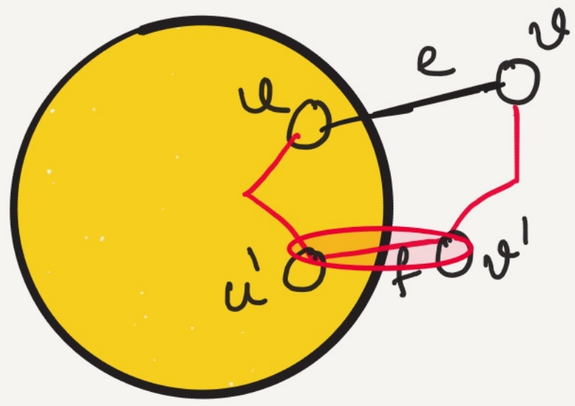
Proof: As we only added e & removed f,

$$\text{weight}(T') = \text{weight}(T) + c_e - c_f$$

$$< \text{weight}(T).$$

Observation 3: T' is a spanning tree.

→ It is Connected: Any path that used the edge f can now be rerouted the "long way" via the edge e.



→ clearly, it covers all the vertices.

⇒ T' is also a spanning tree.

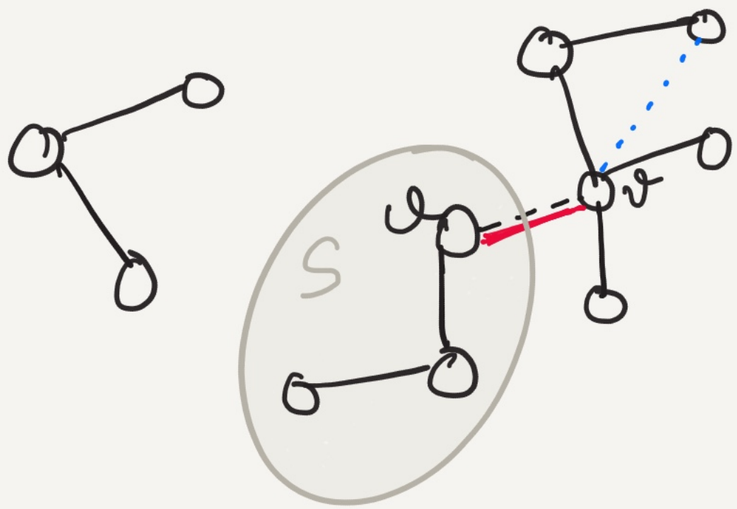
Observations (1), (2) and (3) lead to a contradiction.

Hence, e must be part of the MST.

Analysis of Kruskal's Algorithm:

Claim: Every edge added in the algorithm is part of the MST.

Proof: Suppose the algorithm in an iteration adds an edge (u, v).



(Black edges = edges picked so far.)

Let S = all vertices reachable from u in the graph constructed by the algorithm so far.

Observation 1: $v \notin S$. (v is not reachable from u in the current graph).

(If not adding (u,v) would create a cycle.)

Observation 2: Every edge crossing the cut S does not create a cycle.

Proof: Suppose some edge (v',w') crossing S creates a cycle. Now, $v' \in S$ and $w' \notin S$. So, v' is reachable from S and w' is not reachable from S. But if (v',w') creates a cycle, then there must be a path from v' to w' (in the graph constructed so far).

$\Rightarrow$ w' is also reachable from u; this contradicts $w' \notin S$

Observation 1: $v \notin S$. (v is not reachable from u in the current graph).

(If not adding $\{u,v\}$ would create a cycle.)

Observation 2: Every edge crossing the cut S does not create a cycle.

Observation 3: $e=\{u,v\}$ is the min-weight edge crossing the cut S.

Proof: Follows from (2) and from the fact that the algorithm picks the least weight edge that does not create a cycle

---

①, ②, ③ + CUT PROPERTY $\Rightarrow$ e is part of the MST.

(proves the claim).