W3U1: Minimum Spanning Trees Part 2

C Pandu Rangan

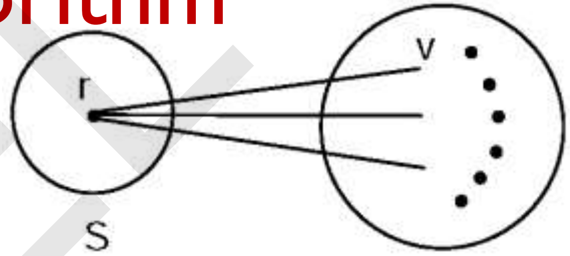
Recap

- MST Basics

Outcomes

- Design and development of Prim's Algorithm.

Development of Prim's Algorithm



- When A is empty, the simplest cut is just a vertex in S, say r, and rest of the vertices in $V - S$. The minimum cost crossing edge is an edge (r, v) such that $w(r, v) \leq w(r, x)\ \forall x \in Adj(r)\ in\ V - S$

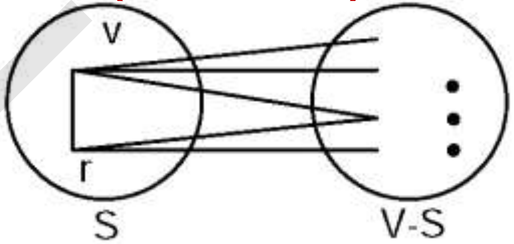
We now update

$A = A \cup \{r, v\} = \emptyset \cup \{r, v\} = \{(r, v)\}$

How do we update the current cut to respect the updated A?

Simple idea – move $v\ to\ S$!

Development of Prim's Algorithm (contd)

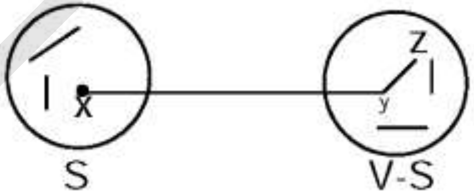


- This move will convert the crossing edge (r, v) to an internal edge with respect to the new cut and thus the new cut would be a cut respecting current A.

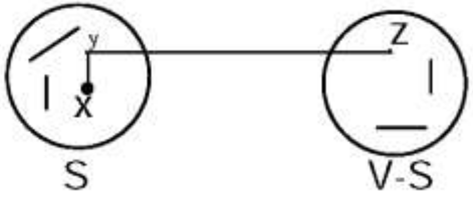
- Notice that the set of crossing edges has become bigger and it is a bit more complex to find the minimum weight crossing the edge now.

- Moreover, there is a minor issue in simply moving the vertex v from $V - S$ to S

Development of Prim's Algorithm (contd)

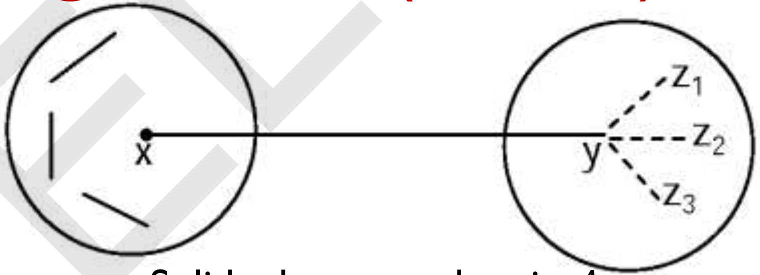


- Suppose current minimum weight crossing edge is (x, y). The edges of A are shown in the picture. If we move y to s, the edge (y, z) in A becomes a crossing edge and hence the new cut is NOT a respecting cut!
- Thus, while moving y may make (x, y) an internal edge, it may make some other edge of A a crossing one!
- Thus, simply moving y when (x, y) is a minimum weight crossing edge may not work!



Development of Prim's Algorithm (contd)

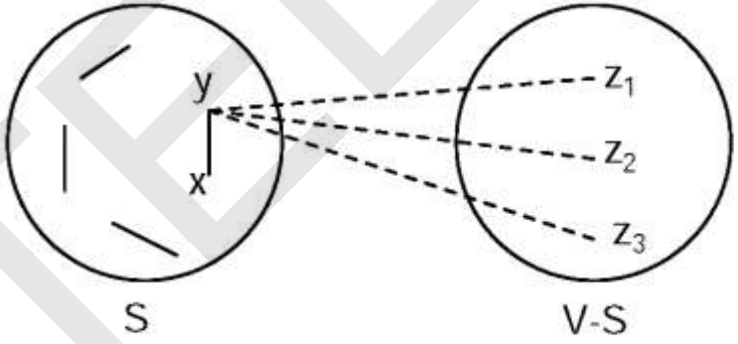
- Fortunately, there is a simple fix to this problem. We maintain all edges of A as internal to S and NO EDGE of A is in $V - s$.
- In this case, there is no problem in moving any vertex from $V - S$ to S.
- Now, moving $y\ to\ S$ makes (x, y) internal to s and new crossing edges are NOT in A. Hence, new cut respects A and has all edges of A internal to S.



Solid edges are edges in A
dotted edges are NOT in A.

Development of Prim's Algorithm (contd)

- The new crossing edges created by moving y to S are NOT in A. The new cut respects A and has all edges of A internal to S.



Development of Prim's Algorithm (contd)

- The first cut, namely $(\{r\}, V - \{r\})$ has the property that all edges of A are internal to $\{r\}$ because $A = \emptyset$!
- Thus, we may identify the minimum weight crossing edge (u, v) where $u \in S, v \in V - S$ and simply move v from $V - S$ to S .
- Now we turn our attention to finding minimum weight crossing edge.
- In the absence of any other information, minimum weight crossing edge may be identified only by checking all crossing edges and this could take $O(m)$ time. Since we perform $(n - 1)$ iteration, the complexity of such a naïve algorithm will be $O(nm)$.

Development of Prim's Algorithm (contd)

- Suppose we maintain for every vertex v in $V - S$ a minimum weight crossing edge (v, v'). Then the overall minimum weight crossing edge can be identified by looking at only edges of the type (v, v') for $v \in V - S$.
- Thus, by examining only $|V - S|$ edges we identify the minimum weight crossing edge.
  Since $|V - S| \leq (n - 1)$ this is a significant improvement over $O(m)$ naïve approach.
- Let us introduce some notations to present our approach in a cleaner way.

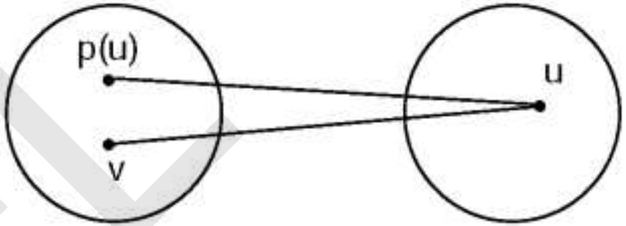
Development of Prim's Algorithm (contd)

- For every vertex v in $V - S$, let (v, v')be a minimum weight crossing edge, incident on v. Here $v' \in S$. That is $w(v, v') \leq w(v, u)$ for all crossing edge (v, u) incident on v.
- We call (v') as a partner of v in S and denote it by $p(v)$. The weight of the crossing edge $(v, p(v))$ is denoted by $c(v)$. Thus, for each vertex v, we maintain $p(v)$ and $c(v)$ and now, it is easy to determine the minimum weight crossing edge.
- In fact, let v be a vertex in $V - S$ such that $c(v) \leq c(u) \forall u \in V - S$.

Development of Prim's Algorithm (contd)

- Then, clearly, $(v, p(v'))$ is the minimum weight crossing edge. Hence, we add $(v, p(v))$ to A and move v to S.
- After moving v to S, the values $p(u)$ and $c(u)$ must be updated for $u \in V - S$.
- The entity $p(u)$ needs an update only if $(v, u) \in E$ and $w(v, u) < w(u, p(u))$.
- $p(u)$ is updated to v.

High level Pseudocode



$A = \emptyset, S = \{r\},$
$V - S = V - \{r\}$

For all $v \in V$,
$p(v) = \text{NULL}$
$c(v) = \infty$

For each $u \in V - S$
If $(n, u) \in E$
$p(u) = r;$
$c(u) = w(r, u)$

While $S \neq V$
Find a vertex v in $V - S$
with minimum $c(v)$ value
Move v to S and add
$(v, p(v))$ to A
Update $c(u)$ and $p(u)$
values for all $u \in V - S$

Pseudocode in more detail

Prim $\left(G = (V, E, w)\right)$

\\$r \in V$. r is arbitrary.

\\$T = (V, A)$, A is implicitly represented by $p(\ )$.

$A = \{(v, p(v)) \mid v \in V - r\}$

$S = \{r\}$ \\($S, V - S$ is the current cut)

$p(v) = \text{NULL}\ \forall\ v \in V$

$c(v) = \infty \quad \forall\ v \in V$

Pseudocode in more detail (contd)

For all $u \in V - S$

If $(u, r) \in E$

$p(u) = r$

$c(u) = w(u, r)$

while ($V - S$ is non-empty)

1) Find $v \in V - S \ni$

$C(V) \leq C(u) \forall u \in V - S$

2) Move $v\ to\ S$

3) For each $u \in V - S$

if $((u, v) \in E$ and $w(u, v) <$

W3U2

- Krushkal's Algorithm.

Thank You