Minimum Spanning Tree

- Minimum Spanning Tree
- Prim's Algorithm for MST
- Cuts in Graphs
- Correctness and Optimality of Prim's Algorithm
- Runtime
  - Basic Implementation
  - Vertex-Centric Implementation
  - Heap Based Implementation

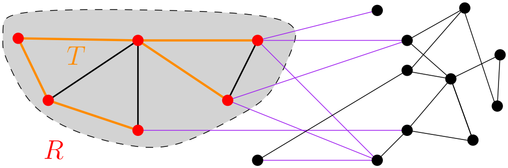
Imdad ullah Khan

Prim's Algorithm

Input: A weighted graph $G = (V, E, w)$, $w : E \to \mathbb{R}$
Output: A spanning tree of G with minimum total weight

Algorithm Prim's Algorithm for MST in $G = (V, E, w)$

$R \leftarrow \{s\}$ ▷ $s \in V$ an arbitrary vertex
$T \leftarrow \emptyset$ ▷ Begin with an empty tree
while $R \neq V$ **do**
  Get $e = (u, v)$, $u \in R, v \notin R$ with minimum $w(uv)$
  $T \leftarrow T \cup \{e\}$
  $R \leftarrow R \cup \{v\}$

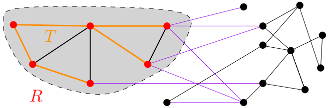


Prim's Algorithm: Naive Implementation

Algorithm Prim's Algorithm, $G = (V, E, w)$

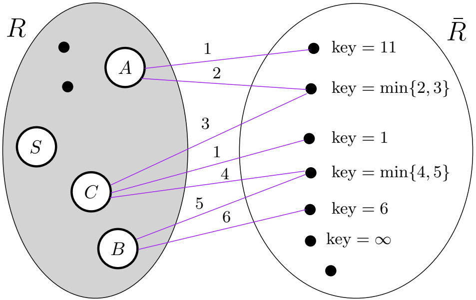
$R \leftarrow \{s\}$ ▷ $s \in V$ an arbitrary vertex
$T \leftarrow \emptyset$ ▷ Begin with an empty tree
while $R \neq V$ **do**
  Get $e = (u, v)$, $u \in R, v \notin R$ with minimum $w(uv)$
  $T \leftarrow T \cup \{e\}$
  $R \leftarrow R \cup \{v\}$

- While loop runs for $O(n)$ iterations
- Find min crossing edge takes $O(m)$
- Total runtime $O(nm)$
- Repeatedly finding minimum is expensive



Prim's Algorithm: Vertex-Centric Implementation

- Store information at vertices (target of many edges)
- Key at vertices is weight of lightest edge incident on it
- Find smallest vertex by key
- Keys are easy to update, just traverse neighbors of new vertex in R

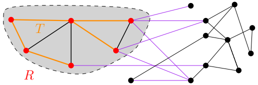


Prim's Algorithm: Vertex Centric Implementaiton

Algorithm Prim's Algorithm, $G = (V, E, w)$

$key[1 \dots n] \leftarrow [\infty \dots \infty]$
$key[s] \leftarrow 0$
$prev(v) \leftarrow null$ ▷ keeps the other end of min crossing edge incident on v
while $R \neq V$ **do**
  Select $v \in \overline{R}$ with minimum $key[v]$
  $R \leftarrow R \cup \{v\}$
  $T \leftarrow T \cup \{(prev[v], v)\}$
  for each $z \in N(v) \cap \overline{R}$ **do**
    if $key[z] > w(vz)$ **then**
      $key[z] \leftarrow w(vz)$
      $prev[z] \leftarrow v$

- While loop runs for $O(n)$ iterations
- Find minimum score **vertex** takes $O(n)$ time
- Need to update only neighbors of added vertex
- Total runtime $O(n^2 + m)$
- Better than last one, esp. for dense graphs
- Repeatedly finding minimum key is expensive

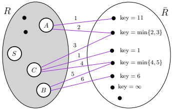


Min-Heap

- A rooted-tree structure satisfying the *heap property*
- If u is parent of v, then $key(u) < key(v)$
- Uses a complete binary tree (binary heap)
- Every node has a key smaller than both its children
- Root always contains the smallest element
- Operations:
  - $\mathcal{H} \leftarrow$ INITIALIZE()
  - INSERT$(\mathcal{H}, v, k)$
  - $v \leftarrow$ EXTRACT-MIN$(\mathcal{H})$
  - DELETE$(\mathcal{H}, v)$
  - DECREASE-KEY$(\mathcal{H}, v, k')$

Prim's Algorithm: Heap based Implementation

- Store information at vertices (target of many edges)
- Key at vertices is weight of lightest edge incident on it
- Find smallest vertex by key
- Easy to update keys, traverse neighbors of new vertex in R



- Store all vertices in $\overline{R}$ in a heap $\mathcal{H}$ with keys
- Initialize $\mathcal{H}$ with V, key of s is 0 for others ∞
- Save pointers (location in heap) to each vertex
- $v \leftarrow \text{EXTRACT-MIN}(\mathcal{H})$ to add to R
- Traverse $N(v)$ to update keys of neighbors in $\overline{R}$

Prim's Algorithm: Heap based Implementation

Algorithm Prim's Algorithm, $G = (V, E, w)$

```
R ← s,  T ← ∅
for v ∈ V do
    v.key ← ∞
    prev(v) ← null        ▷ keeps the other end of min crossing edge incident on v
H ← INITIALIZE(V, keys)
DECREASE-KEY(H, s, 0)
while R ≠ V do
    v ← EXTRACT-MIN(H)
    T ← T ∪ {(v, prev(v))}
    R ← R ∪ {v}
    for z ∈ N(v) do
        if z.key > w(vz) then
            DECREASE-KEY(H, z, w(vz))
            prev(z) ← v
```

Prim's Algorithm: Heap based Implementation

- In total there are n EXTRACT-MIN operations
- On extracting v, there are $O(deg(v))$ DECREASE-KEY operations
- Each EXTRACT-MIN takes $O(\log n)$ time
- Each DECREASE-KEY takes $O(\log n)$ time
- Total runtime $n \log n + m \log n = (n+m) \log n$