Topic 10: Minimum Spanning Tree
- Edge-weighted un-directed graph $G = (V, E)$ and edge weight function $w: E \rightarrow \mathbb{R}$

- E.g., road network, where each node is a city, and each edge is a road, and weight is the length of this road.
A Tree

- An undirected graph $G = (V, E)$ is a tree if:
  - $G$ is connected, and there is no cycle

- Equivalently, an undirected graph $G = (V, E)$ is a tree if:
  - $G$ is connected, and $E = V - 1$

- One key property:
  - Given a tree $T$ on nodes $V$, if we add any edge $e$ to $T$, it will create a cycle.
  - That is, $T \cup \{e\}$ contains a cycle for any $e \notin T$. 

CSE 2331 / 5331
A Spanning Tree

- A *spanning tree* $T$ of an undirected graph $G = (V, E)$
  - is a tree that includes all vertices $V$ of $G$
  - $T \subseteq E$

- **Weight** of a spanning tree $T$:
  - $w(T) = \sum_{(u,v) \in T} w(u, v)$
  - where $w : E \rightarrow R$ is the weight function on edges of $G$

- A *minimum spanning tree* of $G$
  - is a spanning tree with smallest weight.

If $G$ is not connected, then no spanning tree exists.
Examples

Is minimum spanning tree unique?
MST

- MST for given G may not be unique

- Since MST is a spanning tree:
  - # edges : |V| - 1

- If the graph is unweighted:
  - All spanning trees have same weight
Key Property of MST

- Given a MST $T$ of $G = (V, E)$, let $e \in E$ be any edge in $E$ but not in $T$. The following then holds:
  - There is a unique cycle $C$ containing $e$ in $T \cup \{e\}$.
  - $e$ is an edge with largest weight in $C$.

- Sketch of proof:
  - Why unique?
  - If $e$ does not have largest weight, let $e' \in C$ be an edge with largest weight in $C$.
    - $T' = T - \{e'\} + \{e\}$ is a spanning tree of $G$
    - $weight(T') \leq weight(T) \Rightarrow T$ cannot be MST.
    - Contradiction $\Rightarrow e$ must have largest weight in $C$. 

CSE 2331 / 5331
Construct MST

- **Input:**
  - An undirected graph $G = (V, E)$ with weight $w: E \rightarrow R$

- **Output:**
  - A minimum spanning tree MST($G$) of $G$.

- A MST $T$ is $V-1$ number of edges that connect all nodes, with no cycle.

- Intuitively, we will choose “safe” edges to incrementally build $T$
How to choose the first edge that is “safe”, namely, it must be in *some* MST?

- Note, it is *some*, not *all*!
Intuition

- **Input:**
  - An undirected graph $G = (V, E)$ with weight $w: E \to \mathbb{R}$

- **Output:**
  - A minimum spanning tree $\text{MST}(G)$ of $G.$

- **Intuitively,**
  - Incrementally grow a sub-tree $T(S) \subseteq E$ connecting a subset of nodes $S \subset V$
  - At the beginning of each iteration, $T(S)$ is a sub-tree of some MST of $G$
  - At each iteration, grow $T(S')$ to include one more vertex $S' = S \cup \{u\}$
    - Such that $T(S')$ is still a sub-tree of some MST of $G$
Prim’s MST Algorithm: High-level

```
procedure PrimMST(G)
1     U ← V(G) − {v₁};   /* V(G) = set of vertices of graph G */
2     v₁.parent ← NULL;
3     while (U ≠ ∅) and (∃ edge from (V(G) − U) to U) do
4         (vᵢ, vⱼ) ← minimum weight edge from V(G) − U to U;
5         vⱼ.parent ← vᵢ;
6         U ← U − {vⱼ};
7     end
```

- **U**: unconnected vertices
- **S = V − U**: vertices connected by current partial tree
Example

- Greedy algorithm
- Break ties arbitrarily
MST Theorem:

Let $T$ be a sub-tree of a minimum spanning tree.
If $e$ is a minimum weight edge connecting $T$ to some vertex
not in $T$, then $T \cup \{e\}$ is a subtree of a minimum spanning tree.

- Key to the correctness of PrimMST algorithm.
  - Invariance:
    each time PrimMST() algorithm grows the partial tree (i.e., adds
    another edge to it), the invariance is that the new tree is still a subtree
    of some minimum spanning tree of input graph $G$.
  - Termination:
    when all nodes are connected, we obtain a MST of $G$. 
Proof of MST Theorem

- By the theorem’s hypothesis, \( T \) is a subtree of some MST \( A \) of \( G \).
- If \( e \) is not an edge of \( A \), then \( A \cup \{e\} \) contains a cycle.
- Let \( C \) be this cycle. There must exists some edge \( e' \in C \) from \( T \) to a vertex not in \( T \).
- Since \( e \) is a minimum weight edge from \( T \) to vertices not in \( T \), \( \text{weight}(e) \leq \text{weight}(e') \).
- Replacing \( e' \in A \) by \( e \) gives a new tree \( B = A - \{e'\} + \{e\} \) such that \( \text{weight}(B) \leq \text{weight}(A) \).
- \( T \cup \{e\} \subseteq B \). So \( T \cup \{e\} \) is also a subtree of some MST.
- Done.
Naive Implementation of Prim’s MST

procedure PrimMST(G)
1 \( U \leftarrow V(G) - \{v_1\} \); /* \( V(G) \) = set of vertices of graph \( G \) */
2 \( v_1.parent \leftarrow \text{NULL}; \)
3 while (\( U \neq \emptyset \) and (\( \exists \) edge from \( (V(G) - U) \) to \( U \) )) do
4 \( (v_i, v_j) \leftarrow \text{minimum weight edge from } V(G) - U \text{ to } U; \)
5 \( v_j.parent \leftarrow v_i; \)
6 \( U \leftarrow U - \{v_j\}; \)
7 end

- Naïve implementation: linear scan all edges to identify min-weight edge \((v_i, v_j)\) at each iteration
- Total time complexity: \(\Theta(VE)\)
First Improvement of Implementation

- Storing costs at vertices
  - Each unvisited vertex in $U$ maintain the smallest weighted edge to visited vertices
First Improvement: Cost at Vertices

procedure PrimMST(G)
1. \( U \leftarrow V(G) \); /* \( V(G) \) = set of vertices of graph \( G \) */
2. foreach \( v_i \in V(G) - \{v_1\} \) do \( v_i\.cost \leftarrow \infty \);
3. \( v_1\.cost \leftarrow 0 \);
4. \( v_1\.parent \leftarrow \text{NULL} \);
5. while \( (U \neq \emptyset) \) and \( (v_i\.cost < \infty \text{ for some } v_i \in U) \) do
   6. \( v_j \leftarrow v_i \in U \) with minimum \( v_i\.cost \);
   7. \( U \leftarrow U - \{v_j\} \); /* Remove \( v_j \) from \( U \) */
   8. /* \((v_j, v_j\.parent)\) is an MST edge */
   9. foreach edge \((v_j, v_k)\) incident on \( v_j \) do
      10. if \((v_k \text{ is in } U \text{ and } \text{weight}(v_j, v_k) < v_k\.cost)\) then
          11. \( v_k\.parent \leftarrow v_j \);
          12. \( v_k\.cost \leftarrow \text{weight}(v_j, v_k) \);
      end
   end
13. end
14. end
Analysis

- Time to update cost at each vertex at each iteration (in the While-loop):
  - Line 6: $\Theta(V)$
  - Lines 8—13: $\Theta(\deg(v_j))$

- Total time complexity:
  - $\Theta \left( V^2 + \sum_{v_j \in V} \deg(v_j) \right)$
  - $= \Theta(V^2 + E)$
Second Improvement: Efficient Update

- Consider Line 6 of previous implementation
  - We spend linear time to identify the vertex \( v_j \) with smallest cost from \( U \)
- Can we do better?
- What operations do we need to support?
  - extracting the smallest value from a set,
  - updating the cost of some vertices
    - The cost can only decrease!

Use a min-priority queue!
Second Improvement: Priority Queue

```
procedure PrimMST(G)
1 foreach $v_i \in V(G) - \{v_1\}$ do $Q$.Insert($v_i, \infty$);
2 $Q$.Insert($v_1, 0$); /* $Q$ is a priority queue of vertices */
3 $v_1$.parent $\leftarrow$ NULL;
4 while $Q$.IsNotEmpty() and ($Q$.MinKey() $\neq \infty$) do
5     $v_j \leftarrow Q$.DeleteMin(); /* ($v_j, v_j$.parent) is an MST edge */
6     foreach edge ($v_j, v_k$) incident on $v_j$ do
7         if ($Q$.Contains($v_k$) and $Q$.Key($v_k$) $>$ weight($v_j, v_k$))
8             then
9                 $v_k$.parent $\leftarrow v_j$;
10                $Q$.DecreaseKey($v_k$, weight($v_j, v_k$));
11             end
12         end
13     end
14 end
```
Analysis:

- We use min-heap to implement the priority queue
- The maximum size of $Q$ is $V$
- # iterations of While-loop?
  - $V$
- # iterations of each call of the inner for-loop?
  - $\deg(v_j)$
- Total #times lines 7—10 are executed:
  - $\sum_{v_j \in V} \deg(v_j)$
Analysis cont.

- **Q.insert:**
  - Total #: \( V \)
  - Total cost: \( \Theta(V \lg V) \)

- **Q.IsNotEmpty and Q.MinKey:**
  - Total #: \( V \)
  - Total cost: \( \Theta(V) \)

- **Q.DeleteMin**
  - Total #: \( V \)
  - Total cost: \( \Theta(V \lg V) \)

- **Q.DecreaseKey**
  - Total #: \( 2E \)
  - Total cost: \( \Theta(E \lg V) \)

- **Q.Contains and Q.Key**
  - Total #: \( 2E \)
  - Total cost: \( \Theta(E) \)

Total time complexity:
\[ \Theta((V + E) \lg V) = \Theta(E \lg V) \]
Remarks

- Prim’s algorithm is a greedy algorithm
  - The choice of the minimum-weight edge at each iteration
- One can use even more efficient Priority-Queue implementation, based on the Fibonacci Heap
  - $\Theta(V \lg V + E)$

- There are many possible MST algorithms
  - Another popular one is Kruskal Algorithm, which also runs in $\Theta(E \lg V)$ time