Minimum Spanning Trees
(CLRS 23)

• Problem: Given connected, undirected graph \( G = (V,E) \) where each edge \((u,v)\) has weight \( w(u,v) \). Find acyclic set \( T \subseteq E \) connecting all vertices in \( V \) with minimal weight 
  \[ w(T) = \sum_{(u,v) \in T} w(u,v). \]

• An acyclic set connecting all vertices is called a spanning tree. We want to find a spanning tree of minimal weight. We use minimum spanning tree as short for minimum weight spanning tree.

• MST problem has many applications
  – For example, think about connecting cities with minimal amount of wire or roads (cities are vertices, weight of edges are distances between city pairs).

• Example:

1 PRIM’s algorithm

  – Greedy algorithm for computing MST:
    * Start with spanning tree containing arbitrary vertex \( r \) and no edges
    * Grow spanning tree by repeatedly adding minimal weight edge connecting vertex in current spanning tree with a vertex not in the tree
  – Implementation:
    * To find minimal edge connected to current tree we maintain a priority queue on vertices not in the tree. The key/priority of a vertex is the weight of minimal weight edge connecting it to the tree. (We maintain pointer from adjacency list entry of \( v \) to \( v \) in the priority queue).
    * For each node \( u \) maintain \( visit(u) \) (\((u, visit(u))\) is the currently best edge connecting it to the tree.)
PRIM(r)

For each $v \in V$ DO

  INSERT($PQ, v, \infty$)

DECREASE-KEY($PQ, r, 0$)

WHILE $PQ$ not empty DO

  $u = \text{DELETEMIN}(PQ)$

  (output edge $(u, \text{visit}(u))$ as part of MST)

  For each $(u, v) \in E$ DO

    IF $v \in PQ$ and $w(u, v) < \text{key}(v)$ THEN

    visit[$v$] = $u$

    DECREASE-KEY($PQ, v, w(u, v)$)

On the example graph, the greedy algorithm would work as follows (starting at vertex $a$):

a)  b)  c)  d)  e)  f)  g)  h)
Analysis:
* While loop runs $|V|$ times ⇒ we perform $|V|$ DELETEMIN’s
  ↓
  $O((|V| + |E|)\log|V|) = O(|E|\log|V|)$ running time.

Correctness:
* When designing a greedy algorithm the hard part is to prove that it works correctly.
  * We will prove a Theorem that allows us to prove the correctness of a general class of greedy MST algorithms:
    Some definitions
    - A cut $(S, V \setminus S)$ is a partition of $V$ into sets $S$ and $V \setminus S$
    - A edge $(u, v)$ crosses a cut $S$ if $u \in S$ and $v \in V \setminus S$ or $v \in S$ and $u \in V \setminus S$
    - A cut $S$ respects a set $T \subseteq E$ if no edge in $T$ crosses the cut
  Example: Cut $S$ respects $T$

  \[ \text{"cut"} \]
  \[ S \rightarrow \bullet \]
  \[ \forall \land = T \]
  \[ V \setminus S \]

Theorem: If $G = (V, E)$ is a graph such that $T \subseteq E$ is subset of some MST of $G$, and $S$ is a cut respecting $T$ then there is a MST for $G$ containing $T$ and the minimum weight edge $e = (u, v)$ crossing $S$.

Note: Correctness of Prim’s algorithm follows from the Theorem by induction—cut consist of current spanning tree.

Proof:
* Let $T^*$ be MST containing $T$
  * If $e \in T^*$ we are done
  * If $e \notin T^*$:
    - There must be (at least) one other edge $(x, y) \in T^*$ crossing the cut $S$ such that there is a unique path from $u$ to $v$ in $T^*$ ($T^*$ is spanning tree)
· This path together with $e$ forms a cycle
· If we remove edge $(x, y)$ from $T^*$ and add $e$ instead, we still have spanning tree
· New spanning tree must have same weight as $T^*$ since $w(u, v) \leq w(x, y)$

\[ \Downarrow \]

There is a MST containing $T$ and $e$.

Theorem allows us to describe a very abstract greedy algorithm for MST:

\[
\begin{align*}
T &= \emptyset \\
\text{While } |T| &\leq |V| - 1 \text{ DO} \\
&\quad \text{Find cut } S \text{ respecting } T \\
&\quad \text{Find minimal edge } e \text{ crossing } S \\
&\quad T = T \cup \{e\}
\end{align*}
\]

* Prim’s algorithm follows this abstract algorithm.
* Kruskal’s algorithm is another implementation of the abstract algorithm.

2 Kruskal’s Algorithm

- Kruskal’s algorithm is another implementation of the abstract algorithm.
- Idea in Kruskal’s algorithm:
    * Start with $|V|$ trees (one for each vertex)
    * Consider edges $E$ in increasing order; add edge if it connects two trees
- Example:
We need (Union-Find) data structure that supports:

* **MAKE-SET(v)**: Create set consisting of \( v \)
* **UNION-SET(u,v)**: Unite set containing \( u \) and set containing \( v \)
* **FIND-SET(u)**: Return unique representative for set containing \( u \)
KRUSKAL

\[ T = \emptyset \]

FOR each vertex \( v \in V \) MAKE-SET(\( v \))

Sort edges of \( E \) in increasing order by weight

FOR each edge \( e = (u, v) \in E \) in order DO

\[ \text{IF } \text{FIND-SET}(u) \neq \text{FIND-SET}(v) \text{ THEN} \]

\[ T = T \cup \{e\} \]

\[ \text{UNION-SET}(u, v) \]

– Analysis:

* We use \( O(|E| \log |E|) \) time to sort edges and we perform \( |V| \) MAKE-SET, \( |V| - 1 \) UNION-set, and \( 2|E| \) FIND-Set operations.

* We will discuss a simple solution to the Union-Find problem such that MAKE-SET and FIND-SET take \( O(1) \) time and UNION-SET takes \( O(\log V) \) time amortized.

Kruskal’s algorithm runs in time \( O(|E| \log |E| + |V| \log |V|) = O((|E| + |V|) \log |E|) = O(|E| \log |V|) \) like Prim’s algorithm.

– Correctness

* follows from Theorem above: If minimal edge connects two trees then there exists a cut respecting the current set of edges (cut consisting of vertices in one of the trees)

### 3 Union-Find

– The Union-Find problem: Maintain a set system under:

* MAKE-SET(\( v \)): Create set consisting of \( v \)

* UNION-SET(\( u, v \)): Unite set containing \( u \) and set containing \( v \)

* FIND-SET(\( u \)): Return unique representative for set containing \( u \)

– Simple solution:

* Maintain elements in same set as a linked list with each element having a pointer to the first element in the list (unique representative)

Example:
Sets

```
1 2
10 3
6

5 4
12 8

9 11
7
```

Representation

```
3 2 1 10 6
8 5 4 12
11 9 7
```

* Make-Set\( (v) \): Make a list with one element ⇒ \( O(1) \) time
* Find-Set\( (u) \): Follow pointer and return unique representative ⇒ \( O(1) \) time
* Union-Set\( (u, v) \): Link first element in list with unique representative Find-Set\( (u) \) after last element in list with unique representative Find-Set\( (v) \) ⇒ \( O(|V|) \) time (as we have to update all unique representative pointers in list containing \( u \))

- With this simple solution the \( |V| - 1 \) Union-Set operations in Kruskal’s algorithm may take \( O(|V|^2) \) time.
- We can improve the performance of Union-Set with a very simple modification: Always link the smaller list after the longer list (⇒ update the pointers of the smaller list)
  * One Union-Set operation can still take \( O(|V|) \) time, but the \( |V| - 1 \) Union-Set operations takes \( O(|V| \log |V|) \) time altogether (one Union-Set takes \( O(\log |V|) \) time amortized):
    - Total time is proportional to number of unique representative pointer changes
    - Consider element \( u \):
      After pointer for \( u \) is updated, \( u \) belongs to a list of size at least double the size of the list it was in before
      \( \downarrow \)
      After \( k \) pointer changes, \( u \) is in list of size at least \( 2^k \)
      \( \downarrow \)
      Pointer can be changed at most \( \log |V| \) times.
- With improvement, Kruskal’s algorithm runs in time \( O(|E| \log |E| + |V| \log |V|) = O((|E| + |V|) \log |E|) = O(|E| \log |V|) \) like Prim’s algorithm.