1 Graphs

- Last time we defined (weighted) graphs (undirected/directed) and introduced basic graph vocabulary (vertex, edge, degree, path, connected components, ...).
- We also discussed adjacency list and adjacency matrix representation.
  - We will use adjacency list representation unless stated otherwise ($O(|V| + |E|)$ space).
- We discussed $O(|V| + |E|)$ breadth-first (BFS) and depth-first search (DFS) algorithms and how they can be used to compute e.g. connected components, shortest path distances in unweighted graphs, and solve the topological sorting problem.
- We will now start discussing more complicated problems/algorithms on weighted graphs.

2 Minimum Spanning tree (MST)

- 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 $w(T) = \sum_{(u,v) \in T} w(u,v)$.
- Note: Problem is to find a spanning tree (acyclic set connecting all vertices) 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 (cities are vertices, weight of edges are distances between city pairs).
- Example:

  - Weight of MST is $4 + 8 + 7 + 9 + 2 + 4 + 1 + 2 = 37$
  - MST is not unique: e.g. $(b, c)$ can be exchanged with $(a, h)$
2.1 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

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

  ![Diagram](image1.png)

  ![Diagram](image2.png)

  ![Diagram](image3.png)

  ![Diagram](image4.png)

  ![Diagram](image5.png)

- **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).
PRIM(r)

For each \( v \in V \) DO
\[ \text{INSERT}(Q, v, \infty) \]
OD

CHANGE\((Q, r, 0)\)

WHILE \( Q \) not empty DO
\[ u = \text{DELETEMIN}(Q) \]
For each \((u, v) \in E\) DO
\[ \text{IF } v \in Q \text{ and } w(u, v) < \text{key}(v) \text{ THEN} \]
\[ \text{visit}[v] = u \]
\[ \text{CHANGE}(Q, v, w(u, v)) \]
FI
OD
OD

- **Analysis:**
  - While loop runs \(|V|\) times ⇒ we perform \(|V|\) \text{DELETEMIN}'s
  - We perform at most one \text{CHANGE} for each of the \(|E|\) edges
  \[ \Downarrow \]
  \[ O((|V| + |E|) \log |V|) = O(|E| \log |V|) \] running time.

- **Correctness:**
  - As discussed previously, when designing a greedy algorithm the hard part is often 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 \( S \) 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 \( T \)

- **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 got to 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 $e$.

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

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

– Prim’s algorithm follows this abstract algorithm.

3 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:

- Correctness of Kruskal’s algorithm follows from Theorem: If minimal edge connects two trees then a cut respecting the current set of edges exists (cut consisting of vertices in one of the trees)
Implementation:

```plaintext
KRUSKAL

T = ∅
FOR each vertex v ∈ V DO
    MAKE-SET(v)
OD

Sort edges of E in increasing order by weight
FOR each edge e = (u, v) ∈ E in order DO
    IF FIND-SET(u) ≠ FIND-SET(v) THEN
        T = T ∪ {e}
        UNION-SET(u, v)
    FI
OD
```

- 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
- 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.
- Next time we will discuss a simple solution to the Union-Find problem (maintain set system under FIND-SET and UNION-SET) 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.

Note:

- Prim’s algorithm can be improved to $O(|V| \log |V| + |E|)$ using another heap (Fibonacci heap)
- Very recently an $O(|V| + |E|)$ randomized minimum spanning tree algorithm has been developed.