Minimum Spanning Tree Problem

Given: Undirected connected weighted graph \((G, W)\)
Output: An MST of \(G\)

- We have already seen the Prim algorithm, which runs in \(O((m + n) \log(n))\) time (standard Heap implementation) for graphs with \(n\) vertices and \(m\) edges.
- In this lecture we will see Kruskal’s algorithm, a different approach to constructing a MST.

Kruskal’s Algorithm

A forest is a graph whose connected components are trees.

Idea
Starting from a spanning forest with no edges, repeatedly add edges of minimum weight (never creating a cycle) until the forest becomes a tree.

Algorithm \textsc{Kruskal}(G, W)

1. \(F \leftarrow \emptyset\)
2. \textbf{for all} \(e \in E\) in the order of increasing weight \textbf{do}
3. \quad \textbf{if} the endpoints of \(e\) are in different connected components of \((V, F)\) \textbf{then}
4. \quad \quad \(F \leftarrow F \cup \{e\}\)
5. \quad \textbf{return} tree with edge set \(F\)
Correctness of Kruskal’s algorithm

1. Throughout the execution of **Kruskal**, \((V, F)\) remains a spanning forest.
   *Proof:* \((V, F)\) is a spanning subgraph because the vertex set is \(V\). It always remains a forest because edges with endpoints in different connected components never induce a cycle.

2. Eventually, \((V, F)\) will be connected and thus a spanning tree.
   *Proof:* Suppose that after the complete execution of the loop, \((V, F)\) has a connected component \((V_1, F_1)\) with \(V_1 \neq V\). Since \(S\) is connected, there is an edge \(e \in E\) with exactly one endpoint in \(V_1\). This edge would have been added to \(F\) when being processed in the loop, so this can never happen.

3. Throughout the execution of **Kruskal**, \((V, F)\) is contained in some MST of \(G\).
   *Proof:* Similar to the proof of the corresponding statement for **Prim’s algorithm**. Will prove in week 9 Tutorial.

### Data Structures for Disjoint Sets

- A disjoint set data structure maintains a collection \(S = \{S_1, \ldots, S_k\}\) of disjoint sets.
- The sets are dynamic, i.e., they may change over time.
- Each set \(S_i\) is identified by some representative, which is some member of that set.

**Operations:**

- **Make-Set**(x): Creates new set whose only member is \(x\). The representative is \(x\).
- **Union**(x, y): Unites set \(S_x\) containing \(x\) and set \(S_y\) containing \(y\) into a new set \(S\) and removes \(S_x\) and \(S_y\) from the collection.
- **Find-Set**(x): Returns representative of the set holding \(x\).

### Implementation of Kruskal's Algorithm

**Algorithm Kruskal(G, W)**

1. \(F \leftarrow 0\)
2. for all vertices \(v\) of \(G\) do
   3. **Make-Set**(\(v\))
   4. ... + \(T_{\text{Union}}(n)\)
   5. \(F \leftarrow F \cup \{(u, v)\}\)
   6. **Union**(\(u, v\))
9. return \(F\)

### Analysis of Kruskal

Let \(n\) be the number of vertices and \(m\) the number of edges of the input graph.

- Line 1: \(\Theta(1)\)
- Loop in Lines 2–3: \(\Theta(n \cdot T_{\text{Make-Set}}(n))\)
- Line 4: \(\Theta(m \lg m)\)
- Loop in Lines 5–8: \(\Theta(m \cdot T_{\text{Find-Set}}(n) + n \cdot T_{\text{Union}}(n))\).
- Line 9: \(\Theta(1)\)

Overall:

\[
\Theta\left(n \left(T_{\text{Make-Set}}(n) + T_{\text{Union}}(n)\right) + \frac{m}{\lg m} + T_{\text{Find-Set}}(n)\right)
\]
Analysis of **Kruskal** (overview)

\[ T(n, m) = \Theta \left( n \left( T_{\text{MAKE-SET}}(n) + T_{\text{UNION}}(n) \right) + m \left( \lg m + T_{\text{FIND-SET}}(n) \right) \right) \]

With standard efficient implementations of disjoint sets this amounts to

\[ T(n, m) = \Theta (m \lg m). \]

- **NOT** better than the standard Heap implementation of **Prim** for typical implementations of disjoint sets.
- Always have to sort the weights when using **Kruskal**:
  - \( \Theta (m \lg m) \) if the weights are arbitrarily large.

**Example**

Linked list representation of
\{a, f\}, \{b\}, \{g, c, e\}, \{d\}:

![Graph representation]

**Analysis of Linked List Implementation**

**MAKE-SET**: constant (\( \Theta(1) \)) time.

**FIND-SET**: constant (\( \Theta(1) \)) time.

**UNION**: Naive implementation of

\[ \text{UNION}(x, y) \]

appends x’s list onto end of y’s list.

**Assumption**: Representative y of each set has attribute last[y]: a pointer to last cell of y’s list.

**Snag**: have to update “representative pointer” in each cell of x’s list to point to the representative (head) of y’s list.

Cost is:

\( \Theta(\text{length of x’s list}) \).
Notation for Analysis

Express running time in terms of:
- \( \hat{n} \): the number of \texttt{Make-Set} operations,
- \( \hat{m} \): the number of \texttt{Make-Set}, \texttt{Union} and \texttt{Find-Set} operations overall.

Note
1. After \( \hat{n} - 1 \) \texttt{Union} operations only one set remains.
2. \( \hat{m} \geq \hat{n} \).

Weighted-Union Heuristic

Idea
Maintain a “length” field for each list. To execute
\[ \texttt{Union}(x,y) \]
append shorter list to longer one (breaking ties arbitrarily).

Theorem 1
Using the linked-list representation of disjoint sets and the weighted-union heuristic, a sequence of \( \hat{m} \) \texttt{Make-Set}, \texttt{Union} & \texttt{Find-Set} operations, \( \hat{n} \) of which are \texttt{Make-Set} operations, takes
\[ O(\hat{m} + \hat{n}\lg \hat{n}) \]
time.

“Proof”: Each element appears at most \( \lg \hat{n} \) times in the short list of a \texttt{Union}.

Kruskal with Linked lists (weighted union)

The run-time for \texttt{Kruskal} (for \( G = (V,E) \) with \( |V| = n, |E| = m \)) is
\[ T(n,m) = \Theta(n(T_{\text{Make-Set}}(n) + T_{\text{Union}}(n)) + m(\lg m + T_{\text{Find-Set}}(n))) \]
In terms of the collection of “Disjoint-sets” operations, we have \( \hat{m} = 2n + 2m \) operations, \( \hat{n} = n \) which are \texttt{Union}. So
\[
T(n,m) = \Theta(m\lg(m) + (2n + 2m) + n\lg(n)) = \Theta(m\lg(m))
\]