Algorithms and Data Structures: Minimum Spanning Trees (Kruskal)
Minimum Spanning Tree Problem

Given: *Undirected connected weighted graph* $\langle G, W \rangle$
Output: *An MST of* $G$

- We have already seen the PRIM algorithm, which runs in $O((m + n) \lg(n))$ 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 \text{Kruskal}(G, W)
\begin{enumerate}
  \item \(F \leftarrow \emptyset\)
  \item \text{for all} \ e \in E \ \text{in the order of increasing weight} \ \text{do}
  \item \text{if} \ \text{the endpoints of} \ e \ \text{are in different connected components of} \ (V, F) \ \text{then}
    \item \(F \leftarrow F \cup \{e\}\)
  \item \text{return} \ \text{tree with edge set} \ F
\end{enumerate}
Example

ADS: lecture 16 – slide 4 –
Correctness of Kruskal’s algorithm

1. Throughout the execution of \textup{Kruskal}, \((V, F)\) remains a spanning forest.
   \textit{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.
   \textit{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 \(G\) 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 \textup{Kruskal}, \((V, F)\) is contained in some MST of \(G\).
   \textit{Proof:} Similar to the proof of the corresponding statement for Prim’s algorithm.

\textit{ADS: lecture 16 – slide 5 –}
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 \textit{dynamic}, i.e., they may change over time.
- Each set \( S_i \) is identified by some \textit{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 \textsc{Kruskal}(G, W)

1. \( F \leftarrow 0 \)
2. \textbf{for all} vertices \( v \) of \( G \) \textbf{do}
3. \hspace{1em} \textsc{Make-Set}(v)
4. sort edges of \( G \) into non-decreasing order by weight
5. \textbf{for all} edges \((u, v)\) of \( G \) in non-decreasing order by weight \textbf{do}
6. \hspace{1em} \textbf{if} \ \textsc{Find-Set}(u) \neq \textsc{Find-Set}(v) \ \textbf{then}
7. \hspace{2em} \( F \leftarrow F \cup \{(u, v)\} \)
8. \hspace{2em} \textsc{Union}(u, v)
9. \textbf{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: \( \Theta\left(2m \cdot T_{\text{FIND-SET}}(n) + (n - 1) \cdot T_{\text{UNION}}(n)\right) \)
- Line 9: \( \Theta(1) \)

Overall:

\[
\Theta\left(n T_{\text{MAKE-SET}}(n) + (n - 1) T_{\text{UNION}}(n) + m(\lg m + 2 T_{\text{FIND-SET}}(n))\right)
\]
Analysis of Kruskal (overview)

\[ T(n, m) = \Theta\left( n T_{\text{MAKE-SET}}(n) + (n-1) T_{\text{UNION}}(n) + m(\lg m + 2 T_{\text{FIND-SET}}(n)) \right) \]

We will see that 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.
Linked List Implementation of Disjoint Sets

Each element represented by a pointer to a cell:

Use a linked list for each set.
Representative of the set is at the head of the list.
Each cell has a pointer direct to the representative (head of the list).
Example

Linked list representation of

\{a, f\}, \{b\}, \{g, c, e\}, \{d\}:

The "representatives" are a, b, g and d respectively.

last( ) pointers are in red.
Analysis of Linked List Implementation

**Make-Set:** constant (\(\Theta(1)\)) time.

**Find-Set:** constant (\(\Theta(1)\)) time.

**Union:** Naive implementation of

\[
\text{\textsc{Union}}(x, y)
\]

appends \(x\)'s list onto end of \(y\)'s list.

**Assumption:** Representative \(y\) of each set has attribute \(\text{last}[y]\): 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:

\(\Theta(\text{length of } x\text{'s list}).\)
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.

\textbf{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

\text{\textsc{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}.  

\textit{ADS: lecture 16 – slide 14 –}
Example (\textbf{UNION}(g, b)))

- b's "representative" pointer changes to point at g-cell
- e's "next" pointer changes to point at b-cell
- g's "last" pointer changes to point at b-cell

result of performing \textbf{UNION}(g, b)
Kruskal with Linked lists (weighted union)

The run-time for \texttt{Kruskal} (for \( G = (V, E) \) with \(|V| = n, |E| = m \)) is

\[
T(n, m) = \Theta(n T_{\text{MAKE-SET}}(n) + (n - 1) T_{\text{UNION}}(n) + m(\lg m + 2 T_{\text{FIND-SET}}(n)))
\]

In terms of the collection of “Disjoint-sets” operations, we have \( \hat{m} = 2n + 2m - 1 \) operations, \( \hat{n} = n \) which are \texttt{UNION}. So

\[
T(n, m) = \Theta(m \lg(m) + (2n + 2m - 1) + n \lg(n)) = \Theta(m \lg(m))
\]