Algorithms and Data Structures: Minimum Spanning Trees (Kruskal)

Minimum Spanning Tree Problem

Given: *Undirected connected weighted graph* $(\mathcal{G}, W)$
Output: *An MST of* $\mathcal{G}$

- 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.

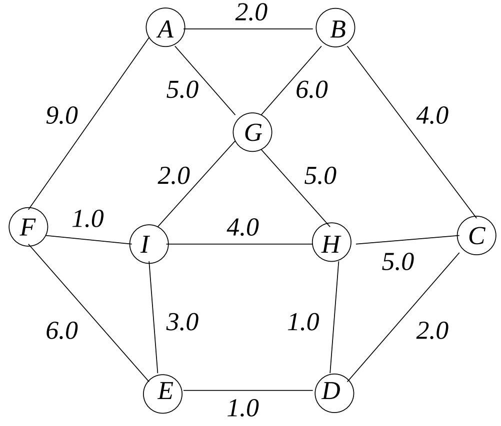
Idea

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 KRUSKAL$(\mathcal{G}, W)$

$F \leftarrow \emptyset$
for all $e \in E$ in the order of increasing weight **do**
  if the endpoints of e are in different connected components of (V, F) **then**
    $F \leftarrow F \cup \{e\}$
return tree with edge set F

Example



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 $\mathcal{G}$ 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 KRUSKAL, (V, F) is contained in some MST of $\mathcal{G}$.
   Proof: Similar to the proof of the corresponding statement for Prim's algorithm.

Data Structures for Disjoint Sets

- A disjoint set data structure maintains a collection $\mathcal{S} = \{S_1, \ldots, S_k\}$ 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($\mathcal{G}, W$)

```
1. F ← 0
2. for all vertices v of G do
3.       MAKE-SET(v)
4. sort edges of G into non-decreasing order by weight
5. for all edges (u, v) of G in non-decreasing order by weight do
6.       if FIND-SET(u) ≠ FIND-SET(v) then
7.             F ← F ∪ {(u, v)}
8.             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: $\Theta\Big(2m \cdot T_{\text{FIND-SET}}(n) + (n-1) \cdot T_{\text{UNION}}(n)\Big)$.
- Line 9: $\Theta(1)$

Overall:

$$\Theta\Big(n T_{\text{MAKE-SET}}(n) + (n-1) T_{\text{UNION}}(n) + m\big(\lg m + 2 T_{\text{FIND-SET}}(n)\big)\Big)$$

Analysis of Kruskal (overview)

$$T(n,m) = \Theta\Big(nT_{\text{MAKE-SET}}(n) + (n-1)\,T_{\text{UNION}}(n) + m\big(\lg m + 2T_{\text{FIND-SET}}(n)\big)\Big)$$

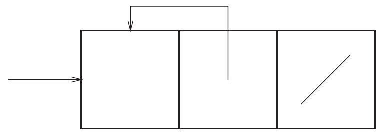
We will see that with standard efficient implementations of disjoint sets this amounts to

$$T(n,m) = \Theta(m \lg(m)).$$

- *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:

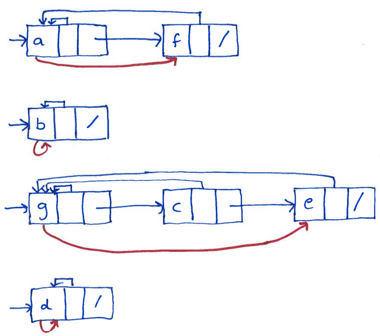


Use a linked list for each set.
Representative of the set is at the head of the list.
Each cell has a pointer direct to the representative (head of the list).

Example

Linked list representation of

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



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

$$\text{UNION}(x, y)$$

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:

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

Notation for Analysis

Express running time in terms of:

- $\widehat{n}$: the number of MAKE-SET operations,
- $\widehat{m}$: the number of MAKE-SET, UNION and FIND-SET operations overall.

Note

1. After $\widehat{n}-1$ UNION operations only one set remains.
2. $\widehat{m} \geq \widehat{n}$.

Weighted-Union Heuristic

Idea

Maintain a "length" field for each list. To execute

$$\text{UNION}(x, y)$$

append shorter list to longer one (breaking ties arbitrarily).

Theorem 1

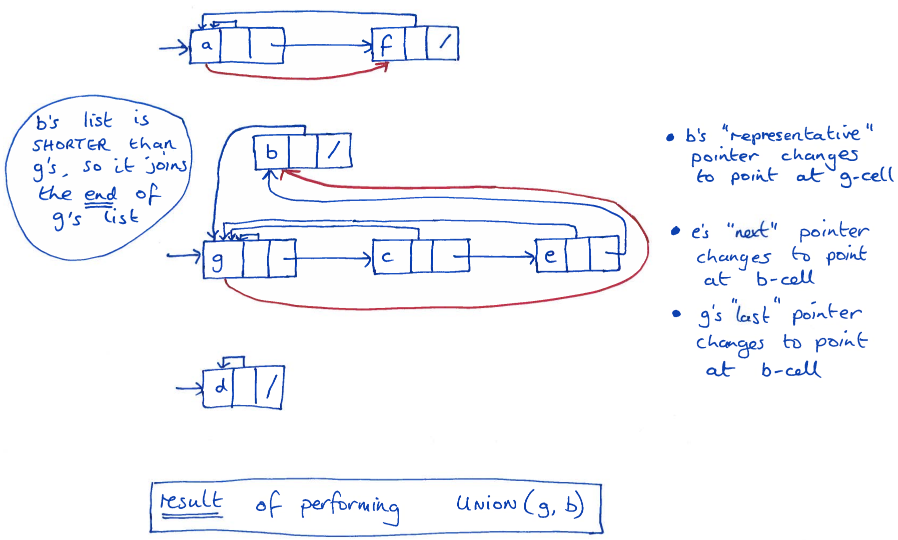
Using the linked-list representation of disjoint sets and the weighted-union heuristic, a sequence of $\widehat{m}$ MAKE-SET, UNION & FIND-SET operations, $\widehat{n}$ of which are MAKE-SET operations, takes

$$O(\widehat{m} + \widehat{n} \lg \widehat{n})$$

time.

"Proof": Each element appears at most $\lg \widehat{n}$ times in the short list of a UNION.

Example (UNION(g, b))



KRUSKAL with Linked lists (weighted union)

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

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

In terms of the collection of "Disjoint-sets" operations, we have $\widehat{m} = 2n + 2m - 1$ operations, $\widehat{n} = n$ which are UNION. So

$$\begin{aligned} T(n, m) &= \Theta(m \lg(m) + (2n + 2m - 1) + n \lg(n)) \\ &= \Theta(m \lg(m)) \end{aligned}$$