CSE 421

Union Find DS
Dijkstra’s Algorithm,

Shayan Oveis Gharan
Union Find Data Structure

Each set is represented as a tree of pointers, where every vertex is labeled with longest path ending at the vertex.

To check whether $A, Q$ are in the same connected component, follow pointers and check if root is the same.

![Diagram of Union Find Data Structure]
Union Find Data Structure

**Merge**: To merge two connected components, make the root with the smaller label point to the root with the bigger label (adjusting labels if necessary). Runs in $O(1)$ time

At most one label must be adjusted
Claim: If the label of a root is k, there are at least $2^k$ elements in the set. Therefore the depth of any tree in algorithm is at most $\log n$

So, we can check if $u, v$ are in the same component in time $O(\log n)$
Depth vs Size: Correctness

Claim: If the label of a root is k, there are at least $2^k$ elements in the set.

Pf: By induction on k.
Base Case (k = 0): this is true. The set has size 1.
IH: Suppose the claim is true until some time t
IS: If we merge roots with labels $k_1 > k_2$, the number of vertices only increases while the label stays the same.
If $k_1 = k_2$, the merged tree has label $k_1 + 1$, and by induction, it has at least
$$2^{k_1} + 2^{k_2} = 2^{k_1+1}$$
elements.
Kruskal’s Algorithm with Union Find

Implementation. Use the `union-find` data structure.

- Build set $T$ of edges in the MST.
- Maintain a set for each connected component.
- $O(m \log n)$ for sorting and $O(m \log n)$ for union-find

```
Kruskal(G, c) {
    Sort edges weights so that $c_1 \leq c_2 \leq \ldots \leq c_m$.
    $T \leftarrow \emptyset$

    foreach ($u \in V$) make a set containing singleton \{u\}

    for $i = 1$ to $m$
        Let $(u, v) = e_i$
        if (u and v are in different sets) {
            $T \leftarrow T \cup \{e_i\}$
            merge the sets containing $u$ and $v$
        }

    return $T$
}
```
Removing weight Distinction Assumption

Suppose edge weights are not distinct, and Kruskal’s algorithm sorts edges so

\[ c_{e_1} \leq c_{e_2} \leq \cdots \leq c_{e_m} \]

Suppose Kruskal finds tree \( T \) of weight \( c(T) \), but the optimal solution \( T^* \) has cost \( c(T^*) < c(T) \).

Perturb each of the weights by a very small amount so that

\[ c'_{e_1} < c'_{e_2} < \cdots < c'_{e_m} \]

where \( c'_{e_i} = c_{e_i} + i.\epsilon \)

If \( \epsilon \) is small enough, \( c'(T^*) < c(T) \).

However, this contradicts the correctness of Kruskal’s algorithm, since the algorithm will still find \( T \), and Kruskal’s algorithm is correct if all weights are distinct.
Single Source Shortest Path

Given an (un)directed graph \( G=(V,E) \) with non-negative edge weights \( c_e \geq 0 \) and a start vertex \( s \)

Find length of shortest paths from \( s \) to each vertex in \( G \)
Dijkstra’s Algorithm

Maintain a set $S$ of vertices whose shortest paths are known
• initially $S=\{s\}$

Maintaining current best lengths of paths that only go
through $S$ to each of the vertices in $G$
• Path-lengths to elements of $S$ will be right, to $V-S$ they
  might not be right

Repeatedly add vertex $v$ to $S$ that has the shortest path-
length of any vertex in $V-S$
• Update path lengths based on new paths through $v$
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra's Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra's Algorithm

\[
\text{Dijkstra}(G, c, s) \{
\text{foreach } (v \in V) \quad d[v] \leftarrow \infty \quad //\text{This is the key of node } v \\
\quad d[s] \leftarrow 0 \\
\text{foreach } (v \in V) \text{ insert } v \text{ onto a priority queue } Q \\
\text{Initialize set of explored nodes } S \leftarrow \{s\} \\
\] \\
\text{while } (Q \text{ is not empty}) \{ \\
\quad u \leftarrow \text{delete min element from } Q \\
\quad S \leftarrow S \cup \{u\} \\
\quad \text{foreach } (\text{edge } e = (u, v) \text{ incident to } u) \\
\hspace{1em} \text{if } ((v \notin S) \text{ and } (d[u] + c_e < d[v])) \\
\hspace{2em} d[v] \leftarrow d[u] + c_e \\
\hspace{2em} \text{Decrease key of } v \text{ to } d[v]. \\
\quad \text{Parent}(v) \leftarrow u \\
\}
Disjkstra’s Algorithm: Correctness

Prove by induction that throughout the algorithm, for any \( u \in S \), the path \( P_u \) in the shortest from \( s \) to \( u \).

**Base Case:** This is always true when \( S = \{s\} \).

**IH:** Suppose \( |S| = k \) and the claim holds for \( S \).

**IS:** Say \( v \) is the \( k+1 \)-st vertex that we add to \( S \). Let \( \{u,v\} \) be last edge on \( P_v \).

If \( P_v \) is not the shortest path there is a path \( P \) to \( s \) which is shorter.

Consider the first time that \( P \) leaves \( S \) (with edge \( \{x,y\} \)).

\( S \rightarrow x \) has weight (at least) \( d(x) \)

So, \( c(P) \geq d(x) + c_{x,y} \geq d(v) = c(P_v) \).

A contradiction.