CLICKERS OUT
The Union-Find Data Structure

Efficient way of maintaining partitions
Equivalence Relations

- An equivalence relation $E(x,y)$ over a domain $S$ is a boolean function that satisfies these properties for every $x,y,z$ in $S$
  - $E(x,x)$ is true \(\text{(reflexivity)}\)
  - If $E(x,y)$ is true, then $E(y,x)$ is true \(\text{(symmetry)}\)
  - If $E(x,y)$ and $E(y,z)$ are true, then $E(x,z)$ is true \(\text{(transitivity)}\)

- For example:
  - $E(x,y)$: Are cities $x$ and $y$ in the same country?

- Another example
  - $E(x,y)$: Are the integers $x$ and $y$ equal?
  - Then $E()$ is an equivalence relation over integers
Equivalence Classes

• An equivalence relation $E()$ over a set $S$ defines a system of equivalence classes within $S$
  • The equivalence class of some element $x \in S$ is that set of all $y \in S$ such that $E(x,y)$ is true
  • Note that every equivalence class defined this way is a subset of $S$
  • The equivalence classes are disjoint subsets: no element of $S$ is in two different equivalence classes
  • The equivalence classes are exhaustive: every element of $S$ is in some equivalence class

Q: What are the equivalence classes defined over all cities in Asia by the equivalence relation - $E(x,y)$: Are cities $x$ and $y$ in the same country?

A. The countries in Asia that share a border
B. The country of either city $x$ or city $y$
C. All the cities in Asia
D. All the countries in Asia
Equivalence Classes for Kruskal’s

For Kruskal’s algo we will partition all vertices of the graph into disjoint sets, based on the equivalence relation: Are two vertices connected?

Q: The above equivalence relation partitions the graph into which of the following equivalence classes?

A. Connected subgraphs
B. Fully-connected (Complete) subgrapahs
C. Disconnected subgraphs
Application of Union-Find to Kruskal’s MST

- Vertices that form a connected will be in the same group
- Connected components that are isolated from each other will be in different groups

Q1: How can we check if adding an edge to the graph creates a cycle?

Q2: What would we like to do if adding the edge does not create a cycle?
Implementation details: Eager simple Union-find

1. In each group, select one of the vertices to represent the leader
2. Augment each vertex to have a pointer to the leader vertex
3. How do we implement find(x)?

Ref: Tim Roughgarden (stanford)
Run time of find operation

1. Finding a path between two vertices
   => finding if they are both in the same group
   => checking if they both point to the same leader: $O(1)$
Implementation of union operation

union(group1, group2)
=> updating leader pointers of vertices in the group
Implementation of union operation

Q: In the worst case, how many leader pointer updates are needed when fusing two groups in the union method:

A. $O(1)$
B. $O(\log|V|)$
C. $O(|V|)$
D. $O(|V|^2)$

Ref: Tim Roughgarden (stanford)
Implementation of smart union operation

Idea 1: When two groups are merged, elect the leader of the larger group to be the leader of the new group and update the leader pointers of the smaller group.

Q: In the worst case, how many leader pointer updates are needed when fusing two groups in the union method:

A. \(O(1)\)
B. \(O(\log|V|)\)
C. \(O(|V|)\)
D. \(O(|V|^2)\)

Ref: Tim Roughgarden (stanford)
Implementation of smart union operation

Idea 1: When two groups are merged, elect the leader of the larger group to be the leader of the new group and update the leader pointers of the smaller group.

Q: (Revised question) In the worst case, how many leader pointer updates does a single vertex experience overall (until the MST is constructed by Kruskal’s)?

A. $O(1)$
B. $O(\log |V|)$
C. $O(|V|)$
D. $O(|V|^2)$

Ref: Tim Roughgarden (stanford)
Q: In the worst case, how many leader pointer updates does a single vertex experience overall (until the MST is constructed by Kruskal’s)?
A. $O(1)$
B. $O(\log|V|)$
C. $O(|V|)$
D. $O(|V|^2)$

Since the leader pointer of a node is updated only when it joins another group with a larger size i.e. the size of its group doubles or more, there can be at most $\log |V|$ such merge operations, resulting in a max of $O(|V|\log|V|)$ updates overall.

Ref: Tim Roughgarden (stanford)
Running Time of Kruskal’s algorithm with union find data structure:

1. Sort edges in increasing order of cost

2. Set of edges in MST, \( T = \{ \} \)

3. For \( i = 1 \) to \( |E| \)

   If \( T \cup \{e_i = u,v\} \) has no cycles i.e. \( \text{find}(u) == \text{find}(v) \) {

   Add \( e_i \) to \( T \)

   union(\text{find}(u), \text{find}(v))

}
Implementation of Union-Find

• The implementation of the Eager (smart) union-find is in the text. You can use that for the PA
Eager Unions vs. Lazy Unions

Perform these operations:

Find(4) =
Find(3) =
Union(1,0) =
Find(4) =

Eager Union

Lazy Union
Array representation of Up-trees

- A compact and elegant implementation
- Each entry is the up index
- -1 for the roots
- Write the forest of trees, showing parent pointers and node labels, represented by this array

Find(4)
Performing a union operation

Union(6,7)

Fill in 6, 7 and 8 in the array

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>-1</td>
<td>-1</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Disjoint subsets using up-trees (Lazy simple Union and Find)

Start with 4 items: 0, 1, 2, 3

Suppose $\text{Union}(i, j)$ makes the node $\text{Find}(i)$ the parent of the node $\text{Find}(j)$

Perform these operations:

$\text{Union}(2, 3)$
$\text{Union}(1, 2)$
$\text{Find}(0) =$
$\text{Find}(3) =$
$\text{Union}(0, 1)$
$\text{Find}(1) =$
$\text{Find}(0) =$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Run time of Lazy simple union and find (using up-trees)

If we have no rule in place for performing the union of two up trees, what is the worst case run time of find and union operations in terms of the number of elements N?

A. find: O(N), union: O(1)  
B. find: O(1), union: O(N)  
C. find: O(log₂N), union: O(1)  
D. find: O(1), union: O(log₂N)
Improving Union operations

• In the Union operation, a tree becomes like a linked list if, when joining two trees, the root of the smaller tree (e.g. a single node) always becomes the root of the new tree

• We can avoid this by making sure that in a Union operation, the *larger* of the two trees’ roots becomes the root of the new tree (ties are broken arbitrarily)

Henry Kautz, U. Washington
Smarter Union operations

- Avoid this problem by having the *larger* of the 2 trees become the root
  - **Union-by-size**
    - Each root stores the size (# nodes) of its respective tree
    - The root with the larger size becomes the parent
    - Update its size = sum of its former size and the size of its new child

- **Union-by-height (also called union by rank)**
  - Each root stores the height of its respective tree
  - If one root has a greater height than the other, it becomes the parent. Its stored height doesn’t need to be updated
  - If the roots show equal height, pick either one as the parent
  - Its stored height should be increased by one
  - Break ties arbitrarily
Disjoint subsets using trees
(Union-by-height and simple Find)

Start with 4 items: 0, 1, 2, 3

Assume that $\text{Union}(i,j)$ makes the root of the shorter tree a child of the root of the taller tree.

Perform these operations:

- Union(2,3) Union(1,2)
- Find(0) = 
- Find(3) = 
- Find(1) = 
- Find(0) =
Cost of disjoint subsets operations with smarter Union and simple Find

Either union-by-size or union-by-height will guarantee that the height of any tree is no more than $\log_2 N$, where $N$ is the total number of nodes in all trees.

Q: What is the worst case run time of find and union operations in terms of the number of elements $N$?

A. find: $O(N)$, union: $O(1)$
B. find: $O(1)$, union: $O(N)$
C. find: $O(\log_2 N)$, union: $O(1)$
D. find: $O(1)$, union: $O(\log_2 N)$
Q: What is the worst case run time of find and union operations in terms of the number of elements N?

A. find: O(N), union: O(1)
B. find: O(1), union: O(N)
C. find: O(log₂N), union: O(1)
D. find: O(1), union: O(log₂N)

Therefore, doing N-1 union operations (the maximum possible) and M find operations takes time O(N + M log₂N) worst case

This is a big improvement; but we can do still better, by a slight change to the Find operation: adding path compression