University of Copenhagen

# Union-Find

Author: William Henrich Due

# 1 Theory

## 1.1 Forests

**Definition 1.1** (Reachability)**.** A node $v$ is *reachable* from a node $u$ in a directed graph $G = (V, E)$ if there exists a sequence of directed edges $e_1, e_2, \ldots, e_m \in E$ where $m \geq 1$ and $e_i = (v_{i-1}, v_i)$ for $1 \leq i \leq m$, such that $v_0 = u$ and $v_m = v$. We denote this by $u \rightsquigarrow v$.

**Definition 1.2** (Cycle)**.** A cycle in a directed graph $G = (V, E)$ has a cycle if there exists $v \in V$ such that $v \rightsquigarrow v$.

**Definition 1.3** (Forest)**.** A forest is a directed graph $F = (V, E)$ where $V$ is a set of vertices and $E \subseteq V \times V$ is a set of directed edges such that:

1. There are no cycles $v \not\rightsquigarrow v$ for all $v \in V$, and
2. each node has at most one parent i.e. for all $(u, v_1), (u, v_2) \in E$ it holds that $v_1 = v_2$.

**Definition 1.4** (Root)**.** A node $v \in V$ in a forest $F = (V, E)$ is a root if it has no parent. This is defined as the predicate:

$$\mathcal{R}_F(v) : v \not\rightsquigarrow u \text{ for all } u \in V$$

**Definition 1.5** (Tree)**.** A tree is a forest $T = (V, E)$ where there exists a unique root $r \in V$ such that $v \rightsquigarrow r$ for all $v \in V \backslash \{r\}$.

**Proposition 1.1** (Forest Root Count)**.** A forest $F = (V, E)$ where $|V| = n$ and $|E| = n - k$ has $k$ roots.

*Proof.* Let $F = (V, E)$ be a forest where $|V| = n$ and $|E| = n - k$. By the second property of a forest then $n - k$ vertices must have a parent. Since there are $n$ vertices in total it follows that there are exactly $k$ vertices $r_1, r_2, \ldots, r_k \in V$ that has no parent. Hence there are exactly $k$ roots in $F$. □

**Proposition 1.2** (Roots Path Exist)**.** In a forest $F = (V, E)$ for each element $v \in V$ there exists at least one root $r \in V$ such that $\mathcal{R}_F(r)$ and either $v \rightsquigarrow r$ or $v = r$.

*Proof.* Let $F = (V, E)$ be a forest and let $v \in V$ be an arbitrary element in $V$. By proposition 1.1 there exists at least one root $r \in V$ such that $\mathcal{R}_F(r)$. This can be shown by structural induction on a vertex $v \in V$ that any $(v, p) \in E$ then either $p = r$ or $p$ has a path to some root $r \in V$.

- If $p = r$ then $p$ is a root and $v$ has a path to a root $r$ by $(v, r) \in E$.
- By induction hypothesis $p$ has a path to a root $r \in V$ such that $\mathcal{R}_F(r)$. Since $(v, p) \in E$ it follows that $v \rightsquigarrow p \rightsquigarrow r$ so .

□

**Proposition 1.3** (Forest Edge Limit)**.** A forest $F = (V, E)$ where $|V| = n$ and $|E| > n - 1$ is not a forest.

*Proof.* Let $F = (V, E)$ be a forest where $|V| = n$ and $|E| > n - 1$. By proposition 1.1 a forest with $n - 1$ has exactly one root, so it is a tree. By adding one more edge to the tree it must make one vertex have two parents. Or since every vertex $v \in V$ has a path to the root $r \in V$ it must create a cycle $v \rightsquigarrow v$ for some $v \in V$. In both cases it contradicts the properties of a forest. □

## 1.2 Union-Find Structure

**Definition 1.6** (Union-Find Structure)**.** The union-find structure $U$ is a forest $U = (V, E)$ where the vertices $V = S$ for some set of elements $S$. The edges $E \subseteq V \times V$ represent parent relations between elements in $S$ such that $(u, v) \in E$ means that $u$ has parent $v$.

**Definition 1.7** (Representative)**.** The representative of an element $v \in V$ in a forest $F = (V, E)$ is the root $r \in V$ such that there is a path from $v$ to $r$. This is defined as the function:

$$\rho_F(v) := r \text{ where } r \in V \text{ such that } \mathcal{R}_F(r) \wedge (v \rightsquigarrow r \vee v = r)$$

**Proposition 1.4** (Unique Representative)**.** In a forest $F = (V, E)$ each element $v \in V$ has a unique representative $\rho_F(v)$.

*Proof.* Let $F = (V, E)$ be a union-find structure and let $v \in V$ be an arbitrary element in $V$. By proposition 1.2 there exists at least one root $r \in V$ such that $\mathcal{R}_F(r)$ and $v \rightsquigarrow r$. Now assume that there exists another root $r' \in V$ such that $\mathcal{R}_F(r')$ and $v \rightsquigarrow r'$. Since $F$ is a forest it follows by the second property of a forest that $r = r'$ hence the representative is unique. □

**Definition 1.8** (Tree Set)**.** The set of vertices of the same tree $\mathcal{E}_F(v)$ in a forest $F = (V, E)$ is defined as:

$$\mathcal{E}_F(v) := \{u : u \in V \text{ where } \rho_F(u) = \rho_F(v))\}$$

**Definition 1.9** (Partition)**.** The set $P \subseteq \mathbb{P}(S)$ is a partition of a set $S$ if:

1. $a \neq \emptyset$ for all $a \in P$
2. $a \cap b = \emptyset$ for all $a, b \in P$ where $a \neq b$
3. $\bigcup_{a \in P} a = S$

**Proposition 1.5** (Forest Partition)**.** A forest $F = (V, E)$ is a partition of $V$ for the following set:

$$\{\mathcal{E}_F(v) : v \in V\}$$

*Proof.* Let $F = (V, E)$ be a forest. We will show that the set in the proposition is a partition of $V$ by showing that it satisfies the three properties in definition 1.9.

1. By definition of $\mathcal{E}_F(v)$ it can not be empty since for $\mathcal{E}_F(v)$ then $\rho_F(v) = \rho_F(v)$. Hence $\mathcal{E}_F(v) \neq \emptyset$ for all $v \in V$.
2. Let $a$ and $b$ be two arbitrary elements in the set such that $a \neq b$. By definition of $a$ and $b$ there exists $v_1, v_2 \in V$ such that $a = \{u : u \in V \wedge \rho_F(u) = \rho_F(v_1)\}$ and $b = \{u : u \in V \wedge \rho_F(u) = \rho_F(v_2)\}$. Since $a \neq b$ it follows that $\rho_F(v_1) \neq \rho_F(v_2)$ since otherwise $a = b$, hence $a \cap b = \emptyset$.
3. Let $v$ be an arbitrary element in $V$. By proposition 1.2 there exists a root $r \in V$ such that $\mathcal{R}_F(r)$ and $v \rightsquigarrow r$ or $v = r$. By definition of the representative it follows that $\rho_F(v) = r$. Now let $a = \{u : u \in V \wedge \rho_F(u) = \rho_F(v)\}$. By definition of $a$ it follows that $v \in a$. Since $v$ was arbitrary it follows that $\bigcup_{a \in P} a = V$.

□

**Definition 1.10** (Same Tree Relation)**.** The relation $\sim_F$ on a forest $F$ is defined as:

$$u \sim_F v : \iff u \in \mathcal{E}_F(v)$$

**Corollary 1.1** (Same Tree Relation is an Equivalence Relation)**.** The relation $\sim_F$ on a forest $F$ is an equivalence relation due to $\{\mathcal{E}_F(v) : v \in V\}$ being a partition of $V$. by proposition 1.5.

**Definition 1.11** (Forests with Equivalent Tree Sets)**.** Two forests $F = (V, E)$ and $F' = (V', E')$ have equivalent tree sets $F \cong F'$ if:

- Vertices are the same $V = V'$.

- The tree sets are equivalent $\mathcal{E}_F(v) = \mathcal{E}'_F(v)$ for all $v \in V$.

**Definition 1.12** (Tree Union)**.** The tree union of two elements $v$ and $u$ for a forest $F = (V, E)$ is such that $v \sim_{F'} u$ in a new forest $F' = (V', E')$ and $F'$ satify the following properties:

1. $\mathcal{E}_{F'}(v) = \mathcal{E}_{F'}(u) = \mathcal{E}_F(v) \cup \mathcal{E}_F(u)$ and
2. $\mathcal{E}_{F'}(w) = \mathcal{E}_F(w)$ for all $w \in V \backslash (\mathcal{E}_F(v) \cup \mathcal{E}_F(u))$.

**Proposition 1.6** (Tree Union)**.** Let forest $F = (V, E)$, $p = \rho_F(u)$ be the representative of $u$ and let $q = \rho_F(v)$ be the representative of $v$ where $q \neq p$. Then defined $F'$ as:

$$F' := (V, E \cup \{(q, p)\})$$

Then $u \sim_{F'} v$ in $F'$ and $F'$ will satisfy the properties of a tree union.

*Proof.* Let $F = (V, E)$, $p = \rho_F(u)$ be the representative of $u$ and let $q = \rho_F(v)$ be the representative of $v$. By definition $q$ will have parent $p$ in $F'$ and since $q$ is a root it has no parent then $F'$ is a forest. Now for all $w \in \mathcal{E}_F(q)$ it holds that $w \rightsquigarrow q$ or $q = w$ and since $q \rightsquigarrow p$ it follows that $w \rightsquigarrow p$. Hence $w \in \mathcal{E}_{F'}(p)$ for all $w \in \mathcal{E}_F(q)$ and trivially $w \in \mathcal{E}_{F'}(p)$ for all $w \in \mathcal{E}_F(p)$ so it follows that $\mathcal{E}_{F'}(v) = \mathcal{E}_{F'}(u) = \mathcal{E}_F(v) \cup \mathcal{E}_F(u)$. Now let $w \in V \backslash (\mathcal{E}_F(v) \cup \mathcal{E}_F(u))$ be an arbitrary element. Since $w \not\rightsquigarrow p$, $w \neq p$, $w \not\rightsquigarrow q$, and $w \neq q$ it follows that $w$ has the same representative in $F'$ as in $F$ hence $\mathcal{E}_{F'}(w) = \mathcal{E}_F(w)$. □

## 1.3 Parallel Union-Find

**Definition 1.13** (Conflict-free Set)**.** Let $F$ be a forest, $X \subseteq \{(\rho_F(v), \rho_F(u)) : (v, u) \in V \times V\}$ be a set of root pairs. Then $X$ is a conflict-free set in $F$ if $(V, Y)$ is a forest.

**Proposition 1.7** (Conflict-free Forest Union)**.** Let forest $F = (V, E)$ be a forest and let $X \subseteq V \times V$ be a conflict-free set in $F$ where $|X| = n$. Then defining the following forests:

$$F_0 := F$$
$$F_i := (V, E_{i-1} \cup \{(v_i, u_i)\}) \text{ for } (v_i, u_i) \in X \text{ and } 1 \leq i \leq n$$

Then $F_n$ is a forest.

*Proof.* Let forest $F = (V, E)$ be a forest, $X \subseteq V \times V$ be a conflict-free set in $F$. We will show that $F_n$ is a forest by induction on $i$.

- Base case: If $i = 0$ then $F_i = F_0 = F$ which is a forest.
- Induction hypothesis: Assume that $F_{i-1}$ is a forest for all $1 \leq i < n$. Let $(v_i, u_i) \in X$, we know that $v_i \neq v_j$ for all $(v_j, u_j) \in Y \backslash \{(v_i, u_i)\}$ since otherwise $(V, X)$ would not be a forest and $X$ would not be a conflict-free set in $F$. So $v_i$ will only have one parent in $F_i$ since it only appears once as a child in $(V, X)$. By definition all of the edges in $X$ consists of roots in $F$, and since $(V, X)$ is a forest there are no cycles $y \not\rightarrow y$ for all $y \in V$ in $F_i$. Hence $F_i$ is a forest.

Thus by induction $F_n$ is a forest. □

**Proposition 1.8** (Conflict-free Set Equivalence)**.** Let forest $F$ be a forest and let $X \subseteq V \times V$ be a conflict-free set in $F$ where $|X| = n$. Then defining the following forests:

$$
\begin{aligned}
F_0 &:= F \\
F_i &:= (V, E_{i-1} \cup \{(v_i, u_i)\}) \text{ for } (v_i, u_i) \in X \text{ and } 1 \leq i \leq n \\
G_0 &:= F \\
G_j &:= (V, E_{i-1} \cup \{(\rho_{G_{j-1}}(v_j), \rho_{G_{j-1}}(u_j))\}) \text{ for } (v_j, u_j) \in X \text{ and } 1 \leq j \leq n
\end{aligned}
$$

Then $F_n \cong G_n$.

*Proof.* Let forest $F$ be a forest, $X \subseteq V \times V$ be a conflict-free set in $F$. We will show that $F_n \cong G_n$. We know that for some $(v_i, u_i) \in X$ then $v_i \neq y$ for all $(y, w) \in X \backslash \{(v_i, u_i)\}$ since otherwise $(V, X)$ would not be a forest and $X$ would not be a conflict-free set in $F$. So all edge set unions will only give a root $v_i$ a new parent $u_i$ once. So $\rho_{F_n}(v_i) = \rho_{F_n}(u_i)$ and $\rho_{G_n}(u_i) = \rho_{G_n}(v_i)$ hence $v_i$ remains in the same tree in both $F_n$ and $G_n$. Since this holds for all $(v_i, u_i) \in X$ it follows that all elements in $V$ remains in the same tree in both $F_n$ and $G_n$. Hence $F_n \cong G_n$. □

**Proposition 1.9** (Ordered Edges Implies Acyclicity)**.** Let $G = (V, E)$ be a directed graph where for all $(v, u) \in E$ it holds that $v < u$ for some strict total order $(V, <)$. Then $G$ has no cycles.

*Proof.* Let $G = (V, E)$ be a directed graph where for all $(u, v) \in E$ it holds that $u < v$ for some total order $(V, <)$. Let edges $e_1, e_2, \ldots, e_m \in E$ where $m \geq 1$ and $e_i = (v_{i-1}, v_i)$ for $1 \leq i \leq m$ be some path in $G$. Since the edges are ordered it follows that:

$$v_0 < v_1 < v_2 < \cdots < v_{m-1} < v_m$$

Hence by transitivity of the total order it follows that $v_0 < v_m$. So $v_0 \neq v_m$ hence there are no cycles in $G$. □

**Proposition 1.10** (Inverted Acyclic Graph is Acyclic)**.** Let $G = (V, E)$ be a directed acyclic graph. Then the inverted graph $G' = (V, E')$ where $E' = \{(u, v) : (v, u) \in E\}$ is also acyclic.

*Proof.* Let $G = (V, E)$ be a directed acyclic graph and $G' = (V, E')$ where $E' = \{(u, v) : (v, u) \in E\}$ is the inverted graph. Let edges $e_1, e_2, \ldots, e_m \in E'$ where $m \geq 1$ and $e_i = (v_{i-1}, v_i)$ for $1 \leq i \leq m$ be some path in $G'$. By definition of $E'$ it follows that there exists edges $e'_1, e'_2, \ldots, e'_m \in E$ where $e'_i = (v_i, v_{i-1})$ for $1 \leq i \leq m$. If there was a cycle in $G'$ then it would hold that $v_0 = v_m$. But since $G$ is acyclic it follows that $v_0 \neq v_m$. Hence there are no cycles in $G'$. □

**Algorithm 1.1** (Left Maximal Union)**.** Let forest $F = (V, E)$ be a forest and let $Z \subseteq \{(\rho_F(v), \rho_F(u)) : (v, u) \in V \times V\}$ be a set of root pairs $F$ and $(V, Z)$ is an acyclic directed graph. The left maximal conflict-free set algorithm is defined as:

*LeftMaximalUnion*$(F, Z)$

1. $(V, E) \leftarrow F$
2. Let $X \subseteq Z$ where $\{v : (v, u) \in X\} = \{v : (v, u) \in Z\}$ and $|X| = |\{v : (v, u) \in Z\}|$
3. $E \leftarrow E \cup X$
4. `return` $((V, E), Z \backslash X)$

**Proposition 1.11** (Left Maximal Union Correctness)**.** Let forest $F = (V, E)$ be a forest and let $Z \subseteq \{(\rho_F(v), \rho_F(u)) : (v, u) \in V \times V\}$ be a set of root pairs $F$ and $(V, Z)$ is an acyclic directed graph. Then the left maximal conflict-free set algorithm returns a forest $F' = (V, E')$ and a conflict-free set $X \subseteq Z$ in $F$.

*Proof.* Let forest $F = (V, E)$ be a forest and let $Z \subseteq \{(\rho_F(v), \rho_F(u)) : (v, u) \in V \times V\}$ be a set of root pairs $F$ and $(V, Z)$ is an acyclic directed graph. Since $X$ is defined such that $\{v : (v, u) \in X\} = \{v : (v, u) \in Z\}$ and $|X| = |\{v : (v, u) \in Z\}|$ it follows that $X$ is a conflict-free set in $F$ since no vertex $v$ appears more than once as a child in $X$ i.e. $(V, X)$ is a forest. By adding the edges in $X$ to $E$ it follows by proposition 1.7 that $F' = (V, E')$ is a forest where $E' = E \cup X$. □

**Proposition 1.12** (Left Maximal Union Time Complexity)**.** Let forest $F = (V, E)$ be a forest and let $Z \subseteq \{(\rho_F(v), \rho_F(u)) : (v, u) \in V \times V\}$ be a set of root pairs $F$ and $(V, Z)$ is an acyclic directed graph. Then the left maximal conflict-free set algorithm runs in $O(|Z|)$ work and $O(\log |Z|)$ depth.

*Proof.* For step 1. it takes $O(1)$ work and $O(1)$ if we assume that $E$ is only used once in this function. Step 2. can be implemented by a parallel sort on the first element of each pair in $Z$ followed by a parallel filter that selects the first occurrence of each unique first element. This takes $O(|Z|)$ work using radix sort with a fixed key length and $O(\log |Z|)$ depth. Step 3. takes $O(|X|)$ work and $O(1)$ depth to add the edges in $X$ to $E$. Step 4. takes $O(|Z|)$ work and $O(\log |Z|)$ depth to compute the set difference $Z \backslash X$ by a filter. Hence the total work is $O(|Z|)$ and the total depth is $O(\log |Z|)$. Hence the left maximal conflict-free set algorithm runs in $O(|Z|)$ work and $O(\log |Z|)$ depth. □

**Proposition 1.13** (Acyclic Directed Graph Presevation)**.** Let $G = (V, E)$ be a directed acyclic graph and let $F = (V, X)$ be a forest where $X \subseteq E$. Then the directed graph $G' = (V, E')$ where $E' = \{(\rho_F(v), \rho_F(u)) : (v, u) \in E \backslash X \wedge \rho_F(v) \neq \rho_F(u)\}$ is also acyclic.

*Proof.* Let $G = (V, E)$ be a directed acyclic graph and let $F = (V, X)$ be a forest where $X \subseteq E$. Let $q$ be a path $e_1, e_2, \ldots, e_m \in E$ where $m \geq 1$ and $e_i = (v_{i-1}, v_i)$ for $1 \leq i \leq m$ be some path in $G$.

- If all edges $e_i \notin X$ then the path $q$ exists in $G'$ and does not form a cycle since $G$ is acyclic.
- If $q$ has some subpath $e_j, e_{j+1}, \ldots, e_k \in X$ for $1 \leq j < k \leq m$ where $(v_{j-1}, v_j) \in X$ then $\rho_F(v_{j-1}) = \rho_F(v_j)$ so it will not be in $E \backslash X$. Let $(v, u) \in E$ where $u = v_{j-1}$ or $u = v_j$ then $\rho_F(u) = v_k$ so the edge in $G'$ will be $(\rho_F(v), v_k)$.
  - If $v \in \{w : (w, u) \in X\}$ then $\rho_F(v) = v_k$ so the edge is $(v_k, v_k)$ which is a trivial cycle and since $\rho_F(v) \neq \rho_F(u)$ it will not be in $E'$.
  - If $v \notin \{w : (w, u) \in X\}$ then $\rho_F(v) \neq v_k$ and $(\rho_F(v), v_k)$ will be an edge in $G'$. And since the path $v \rightsquigarrow v_k$ in $G$ does not form a cycle it follows that single edge from $(\rho_F(v), v_k) \in E'$ does not form a cycle either.

□

**Algorithm 1.2** (Parallel Tree Union)**.** Let forest $F = (V, E)$ be a tree and let $A \subseteq V \times V$ be a set of pairs of elements in $V$ that will be unioned in

parallel. The parallel tree union algorithm is defined as:

*ParallelTreeUnion*$(F, A)$

1. $Z_p \leftarrow \{(\rho_F(v), \rho_F(u)) : (v, u) \in A \wedge \rho_F(v) \neq \rho_F(u)\}$
2. $Z \leftarrow \{(\min\{v, u\}, \max\{v, u\}) : (v, u) \in Z_p\}$
3. `while` $|Z| > 0$ `do`
4. $\quad (F, Z_q) \leftarrow \mathit{LeftMaximalUnion}(F, Z)$
5. $\quad Z \leftarrow \{(\rho_F(u), \rho_F(v)) : (v, u) \in Z_q \wedge \rho_F(v) \neq \rho_F(u)\}$
6. `return` $F$

**Proposition 1.14** (Parallel Tree Union Correctness)**.** Let forest $F = (V, E)$ be a tree and let $A \subseteq V \times V$ be a set of pairs of elements in $V$ that will be unioned in parallel. Then the parallel tree union algorithm returns a forest $F' = (V, E')$ where for all $(v, u) \in A$ it holds that $v \sim_{F'} u$.

**Proposition 1.15** (Parallel Tree Union Time Complexity)**.** Let forest $F = (V, E)$ be a tree and let $A \subseteq V \times V$ be a set of pairs of elements in $V$ that will be unioned in parallel. Then the parallel tree union algorithm runs in $O(|A|^2)$ work and $O(|V|)$ span.

## 1.4 Union by Size and Rank

## 1.5 Parallel Union-Find Imrovements

# 2 Implementation

## 2.1 Interface

The interface of the union-find structure consists of a data-type which is the union-find structure itself. The elements in the union-find structure are represented as integers. These integers are called *handles* and are used to refer to the elements in the union-find structure. The union-find structure is initialized with a fixed number of elements $n$ where the handles are in the range $[0, n-1]$ so they can be used as indices in an array of size $n$. When exposing these elements to a user then they are abstract datatypes such that the the user cannot do unintened operations on the handles or give invalid handles to the union-find structures operations.

The operations supported are *find* and *union*. The find operation takes a handle and returns the representative which is also a handle. This can be used to check if two elements are in the same set or is "equivalent" by

checking if their representatives are the same. The union operation takes two handles such that they have the same representative and are therefore in the same set or to be considered "equivalent". These operations are done in bulk meaning that the find operation takes an array of handles and union can union multiple pairs of handles in parallel.

## 2.2 Union-find

The first and simplest implementation of the union-find structure is based on the basic parallel tree union algorithm 1.2. It will be extremely inefficient if we do not apply normalization of a left maximal union since you may produce long chains of elements pointing to each other which will make the find operation very expensive. This can be solves by compressing the paths during the find operation such that all elements point directly to their representative after a left maximal union. This will not optimize a sequence of union or find operations but assuming you only have to do one singular bulk union followed by multiple bulk find operations it will be efficient.

### 2.2.1 Data type

A simple union-find structure can be implemented using an array indices which where the index represents the element/handle and the value at that index is the parent of that element. If the value at that index is a special value which is the highest possible integer value then that element is a root and therefore its own representative. In pseudo code we denote this as an array of integers:

$parents : [n]\textbf{int}$

Initially all of these values are set to *none* which indicates that all elements are their own representative. Here *none* is defined as an integer $n \geq none$.

### 2.2.2 Find

To implement the find operation we can simple do a parallel map over all elements to find their representative by a simple loop that follows the parent pointers until a root is found. Since this operation does not modify the

structure we can simplify implement it as so:

```
def find_one (parents : [n]int) (handle : int) =
    if parents[handle] = none
    then handle
    else find parents (parents[handle])
```

Then the bulk find operation can be implemented as:

```
def find (parents : [n]int) (handles : [m]int) =
    map (find_one parents) handles
```

### 2.2.3 Union

The union operation can be implemented using the parallel tree union algorithm 1.2 but due to its abstract nature one must figure out how to make it work in the concrete implementation.

The initial step is to find the representatives of all handle pairs that will be unioned. This can be done by a map followed by a filter to remove pairs where both elements already have the same representative:

```
def find_pairs (parents : [n]int) (pairs : [m](int, int)) =
    let find_one_pair (v, u) =
      (find_one parents v, find_one parents u)
    in (filter (λ(v, u) → v ≠ u) ∘ map find_one_pair) pairs
```

The next step is we want to order all pairs such that it forms an acyclic directed graph. This can be done by simply ordering each pair such that the first element is always less than the second element:

```
def order (pairs : [m](int, int)) =
    map (λ(v, u) → if v < u then (v, u) else (u, v)) pairs
```

Now left maximal union can be performed on the ordered pairs, we do this simply by selecting one parent for each unique first element in the pairs. This can be done by a reduce by index operation using *min* as the reduction operator. Now we just have to partition the pairs into those that were successfully unioned and those that were not. The ones that were not unioned are those where the parent did not change.

There is just one problem with this approach and it is that this may construct one long chain of elements pointing to each other which will make

future find operations expensive. This can be solved by normalizing the structure after the left maximal union such that all elements that were given a new parent now point directly to their representative. This results in the following left maximal union implementation:

```
def left_maximal_union (parents : [n]int) (pairs : [m](int, int)) =
  let (l, r) = unzip pairs
  let parents' = reduce_by_index parents min none l r
  let (remaining, done) = partition (λ(i, p) → parents'[i] ≠ p) pairs
  let parents'' = normalize parents' (map (λ(x, y) → x) done)
  in (parents'', remaining)
```

The normalization step can be implemented by using the wyllie list ranking algorithm [1, p. 59] to compress the paths in the union-find structure. The modification is instead of checking if the parent does not exists we have to check if it is a root or the grandparent is a root then we are done.

```
def normalize_step (is : [m]int) (parents : [n]int) =
  let next (handle : int) =
    if parents[handle] = none
    then handle
    else if parents[parents[handle]] = none
    then parents[handle]
    else parents[parents[handle]]
  in scatter parents is (map next is)
```

Then we just have to repeat this process $\lceil \log_2 m \rceil$ times to ensure that all paths are compressed:

```
def normalize (parents : [n]int) (is : [m]int) =
  let loop (parents' : [n]int) (is' : [m]int) (count : int) =
    if count = 0
    then parents'
    else normalize_step is' parents'
    in loop parents' ps' (count − 1)
  in loop parents is ⌈log2 m⌉
```

The last step in the parallel tree union algorithm is to swap the pairs such that we can perform left maximal union again on the inverted pairs. This

takes the acyclic directed graph formed by the remaining pairs and inverts all edges:

```
def swap (pairs : [n](int, int)) =
    map (λ(v, u) → (u, v)) pairs
```

Now we have all the functions needed to implement the union operation:

```
def union (parents : [n]int) (pairs : [m](int, int)) : [n]int =
    let pairs =
```

## 2.3 Union by Rank

## 2.4 Union by Size