Analysis of Disjoint Sets ADT with rank and path compression

Link(i, j)
{ if (Rank[i] > Rank[j])
    swap i and j;
    else if (Rank[i] == Rank[j])
        Rank[j]++;
    Boss[i] = j;
}

Union(i, j)
{ Link(Find(i),Find(j)) }

Find(i)
{ k = j = i;
    while ( j == Parent[i], i ≠ j )
        i = j;
    // compression loop
    /* compression loop */
    while ( j == Parent[k], i ≠ j )
    Parent[k] = i,
    k = j;
    return i;
}

Measuring the running time

We want to estimate the running time of an arbitrary sequence of \(m\) Find and Link operations. Because our Union consists of two Finds and one Link, this will characterize any sequence of operations on Disjoint Sets.

We will define the cost of operations as follows: Link(i, j) has cost 1, and Find(i) has cost equal to the number of visited nodes. In these note we are investigating the following question: for what range of \(n\) we can guarantee that every sequence of \(m\) operations has the sum of costs smaller than \(3m\).
Properties of Rank

For the sake of analysis, we introduce notation that treats nodes as pointers to structures with the fields \( Rnk, Prt, 0, 1, \cdots \) that have the following meaning:

\[
i \rightarrow Rnk \quad \text{Rank}[i] \\
i \rightarrow Prt \quad \text{Parent}[i]
\]

while the numbered fields are set during \texttt{Link} that can be rewritten as follows:

\[
\text{Link}(i, j) \\
\{ \\
\quad \text{if } (i \rightarrow Rnk > j \rightarrow Rnk) \\
\quad \quad \text{swap } i \text{ and } j; \\
\quad \text{else if } (i \rightarrow rank == j \rightarrow rank) \\
\quad \quad k = j \rightarrow Rnk; \\
\quad \quad j \rightarrow k = i, \\
\quad \quad (j \rightarrow Rnk)++; \\
\quad i \rightarrow Prt = j; \\
\}
\]

We extend notation \( i \rightarrow s \) from integers to sets of integers; \( i \rightarrow \emptyset \) equals \( i \), moreover, if \( S = \{s\} \cup T \) and \( s \) is larger than all elements of \( T \) then \( i \rightarrow S = i \rightarrow s \rightarrow T \).

We need several simple observations.

(a) \( i \rightarrow s \) is defined for integers such that \( s \in \{0, \cdots, (i \rightarrow Rnk) - 1\} \);

(b) \( i \rightarrow S \) is defined for integer sets such that \( S \subset \{0, \cdots, (i \rightarrow Rnk) - 1\} \);

(c) if \( S \neq T \) and \( i \rightarrow S \) and \( i \rightarrow T \) are both defined, than these are two different elements of the set where \( i \) belongs.

From (c) we may conclude that the set of \( i \) contains at least \( 2^{i \rightarrow Rnk} \) elements.

We will refer to \( i \rightarrow Rnk \) as the rank of \( i \).

We will characterize function \( B(r) \) with the following property: any sequence of \( m \texttt{Find} \) and \( \texttt{Link} \) operations has the total cost at most \( B(r)m \), provided that the the largest rank of a node is at most \( r \).
**B(1) and B(2)**

It is easy to see that \( B(1) = 0 \): if the maximum rank is 1, we can follow \( \text{Link}(0, 1) \) with \( \text{Find}(0) \) and the cost of these 2 operations is 3 > 2.

Similarly, one can show that \( B(2) = 2 \). The following sequence of 10 \( \text{Links} \), each with cost 1, and 11 \( \text{Finds} \), each with cost 3 shows that \( B(2) < 3 \).

\[
\begin{align*}
\text{Link}(1, 0), & \quad \text{Link}(3, 2), \quad \text{Link}(2, 0), \\
\text{Find}(3) & \\
\text{Link}(7, 6), & \quad \text{Link}(8, 6), \quad \text{Link}(9, 6), \quad \text{Link}(10, 6), \quad \text{Link}(11, 6), \\
\text{Link}(4, 5), & \quad \text{Link}(6, 5), \\
\text{Find}(7), & \quad \text{Find}(8), \quad \text{Find}(9), \quad \text{Find}(10), \quad \text{Find}(11), \\
\text{Link}(5, 4), & \quad \text{Link}(6, 4), \quad \text{Link}(4, 0), \\
\text{Find}(5), & \\
\text{Find}(7), & \quad \text{Find}(8), \quad \text{Find}(9), \quad \text{Find}(10), \quad \text{Find}(11),
\end{align*}
\]
Lower bound for $B(3)$

In the remainder of this section we will show that $B(3) \geq 89$. Because of properties of rank, this means that a sequence of $m$ \texttt{Link} and \texttt{Find} operations has cost at most $3m$ if the number of nodes is smaller than $2^{90}$, and thus for all proactical applications of Disjoint Sets ADT. Therefore the case of $r > 4$ is of interest only because its mathematics is so unusual in the context of algorithm analysis.

**Class Society**

Our analysis is using a version of *amortized analysis*. It means that the application the invokes an operation must pay the *price* of this operation, while we, maintainers of the data structure, must either pocket the surplus of the price minus cost, or pay, out of the pocket, the deficit i.e. the cost minus the price. If we can show that we can start with no money and yet we will never have a debit balance, then we prove that the sum of costs is bounded by the sum of prices. In our case, the prices of both \texttt{Link} and \texttt{Find} are equal to 3.

Our currency will be *tickets*.

In this section we will present an analysis which is valid under the assumption that $n < 2^{90}$. While this is a bounded range, it contains all values that may be encountered in practice, and quite a bit more. By the final observation of the previous section, our assumption implies that $\text{Rank}[i] < 90$. We divide the ranks into 3 *classes*:

- ranks 0 to 1 workers,
- ranks 2 to 6 managers,
- ranks 7 to 89 corporate officers, officers for short.

For example, if $i \rightarrow Rnk = 4$, then $i \rightarrow Cls = \text{manager}$.

During a \texttt{Link}(i, j) we receive 3 tickets and spend 1, the remaining 2 tickets are given to this argument of \texttt{Link} that ceases to be a boss.

During a \texttt{Find}(i), every node $j$ visited pays one ticket, with some exceptions which are paid with the tickets obtained as the price of \texttt{Find}. These exceptions are as follows:

(i) $i \rightarrow Prt \rightarrow Rnk$ remain unchanged after the compression;
(ii) $i \rightarrow Prt \rightarrow Cls$ remain unchanged and $i \rightarrow Prt \rightarrow Cls \neq i \rightarrow Cls$.

Observe that in the sequence of nodes visited by a \texttt{Find} the ranks are increasing, and only the last two nodes have their parents unchanged. Thus we can divide this sequence into two parts: part (A), nodes from the class lower than the boss, and part (B), nodes from the class of the boss.
The nodes exempt from paying are: the last node of (A) — exemption (i), and the last two nodes of (B) — exemption (ii).

To keep a better track of expenses, we will require that a node pays with a properly completed ticket. Assume that \( b \) is the boss, \( b \rightarrow Rnk = R \) and \( b \rightarrow Cls = C \). Then nodes from (A) pay with the ticket

\[
my \rightarrow Prt \rightarrow Cls \text{ becomes } C
\]

while

\[
my \rightarrow Prt \rightarrow Rnk \text{ becomes } R
\]

is the ticket used by the nodes from (B). A crucial observation is that a node never needs to use the same ticket twice, because \( i \rightarrow Prt \rightarrow Rnk \) and \( i \rightarrow Prt \rightarrow Cls \) can only increase.

We will analyze ow different classes balance income and expenses.
The balance of a worker

Workers = class 0 = ranks: 0, 1.
A worker can pay only these tickets:
\[ my \rightarrow Prt \rightarrow Cls \] becomes 1
\[ my \rightarrow Prt \rightarrow Cls \] becomes 2

When a node becomes a worker in a Link it gets these two tickets, and this increases the amortized cost of Link from 1 to 3.

The balance of a manager

Managers = class 1 = ranks: 2, 3, 4, 5, 6
A manager can pay only these tickets:
\[ my \rightarrow Prt \rightarrow Rnk \] becomes 4
\[ my \rightarrow Prt \rightarrow Rnk \] becomes 5
\[ my \rightarrow Prt \rightarrow Rnk \] becomes 6
\[ my \rightarrow Prt \rightarrow Cls \] becomes 2

When a node \( i \) becomes a manager in a Link it gets two tickets. Moreover, \( i \rightarrow 0 \) and \( i \rightarrow 1 \) do not need anymore tickets
\[ my \rightarrow Prt \rightarrow Cls \] becomes 1

so \( i \) takes these tickets away and rewrites them according to its needs.
The balance of an officer

officers = class 2 = ranks: 7, 8, 9, ...

An officer can pay only these tickets:

\[ my \rightarrow Prt \rightarrow Rnk \text{ becomes 9} \]
\[ my \rightarrow Prt \rightarrow Rnk \text{ becomes 10} \]
\[ my \rightarrow Prt \rightarrow Rnk \text{ becomes 11} \]

etc. So our schema works if

(a) an officer always starts with \( t \) tickets,
(b) the highest rank is at most \( t + 8 \). an officer always starts with \( t \) tickets,

When a manager \( i \) becomes an officer, it has rank at least 7, and gets 2 tickets, and moreover it can take

2 tickets from \( i \rightarrow 0 \) and \( i \rightarrow 1 \) (1 ticket taken back when \( i \) became manager, 1 ticket now)
1 ticket from \( i \rightarrow 1 \rightarrow 0 \),
4 tickets from \( i \rightarrow 2, i \rightarrow 3, i \rightarrow 4, i \rightarrow 5, i \rightarrow 6 \),
3 tickets from

\[ i \rightarrow 6 \rightarrow 5, i \rightarrow 6 \rightarrow 4, i \rightarrow 6 \rightarrow 3, i \rightarrow 6 \rightarrow 2, \]
\[ i \rightarrow 5 \rightarrow 4, i \rightarrow 5 \rightarrow 3, i \rightarrow 5 \rightarrow 2, \]
\[ i \rightarrow 4 \rightarrow 3, i \rightarrow 4 \rightarrow 2, \]
\[ i \rightarrow 3 \rightarrow 2, \]

2 tickets from another 10 nodes and 1 ticket from another 5 nodes.

Thus an officer starts with at least 82 tickets (or is it 81?)

Summarising, we can have 89 classes, which means that as long as \( n < 2^{90} \) the average cost of operations is at most 3.