CS294-5 Great Algorithms **Spring 2006**

Lecture 8: 2.13.06

Lecturer: Richard Karp *Scribe: Cheng Tien Ee*

8.1 Union-Find Algorithms

A UNION-FIND data structure maintains a partition of a set X of size n. For each set A in the partition it maintains a representative $\mathcal{S}(A)$ contained in A. The initial partition has each element in a set by itself. Operations include

UNION$(\mathcal{S}(A), \mathcal{S}(B))$, where $\mathcal{S}(A) \neq \mathcal{S}(B)$, replaces A and B by $A \cup B$, and specifies a representative for $A \cup B$.

FIND(x), where $x \in X$, returns $\mathcal{S}(A_x)$, the representative of the set containing x.

Applications of UNION-FIND**:** Various minimum spanning-tree algorithms maintain a set of forests, and merge two forests by adding an edge connecting them.

We consider a UNION-FIND data structure in which each set A is represented by a forest where nodes are the elements of A, with $\mathcal{S}(A)$ at the root and an edge from each node to its parent. The entire data structure is a forest $\mathcal{F}$ with a tree for each set in the partition. The parent pointers are stored in an array, as shown in Figure 8.1.

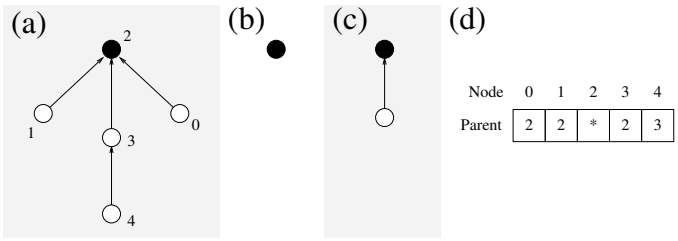

Figure 8.1: *(a)-(c) Examples of trees, with the black nodes representing roots of the corresponding trees. (d) Representation of parent pointers in array form for tree shown in (a); the root node is represented using a* $*$.

UNION$(\mathcal{S}(A), \mathcal{S}(B))$ is easy to execute, just add a pointer from $\mathcal{S}(A)$ to $\mathcal{S}(B)$, or from $\mathcal{S}(B)$ to $\mathcal{S}(A)$.

FIND(x) is executed by following pointers from x up to $\mathcal{S}(x)$.

The *cost* of UNION$(\mathcal{S}(A), \mathcal{S}(B))$ is 0 and that of FIND(x) is the number of pointers changed. At most $n-1$ UNION operations can be executed.

We consider an implementation of FIND involving two stages:

1. following pointers from x to $\mathcal{S}(A_x)$,
2. retracing the path, replacing the parent pointer of each node (except the last) by a pointer to $\mathcal{S}(A_x)$.

This implementation of FIND(x) is called *path compression*. Figure 8.2 gives an example.

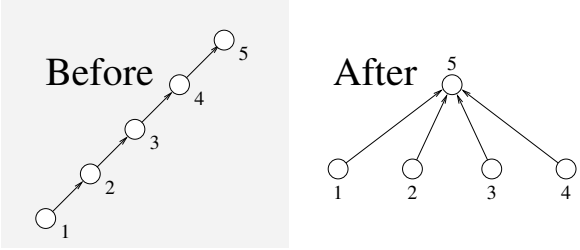

Figure 8.2: *Path compression: rather than point to the immediate parent, each node except the root is made to point to the root instead.*

In general, UNION operations are interspersed with FIND operations. However, given any sequence of operations, we can define another in which the UNIONs are all performed first, and then the same paths as in the original algorithm are compressed. However, these paths no longer must go all the way to the root. This separation of UNIONs from FINDs simplifies the analysis.

Lemma 8.1 *Given any sequence of* UNION *and path compression operations, there is a sequence with the same number of* UNIONs *and the same cost, in which the* UNIONs *come first, followed by partial path compressions in 1-1 correspondence with the original path compressions.*

A partial path compression takes a path p ending at a vertex y, which is not the root, and causes every vertex in p to point to the parent of y.

We now begin our analysis of sequences of partial path compressions, starting with a given forest $\mathcal{F}$. A partition of X into disjoint sets X_b and X_t is called a *dissection* of X if X_t is *ancestor-closed*, i.e. if $x \in X_t$ then every ancestor of x in $\mathcal{F}$ is in X_t. Figure 8.3 shows an example of X_t in $\mathcal{F}$.

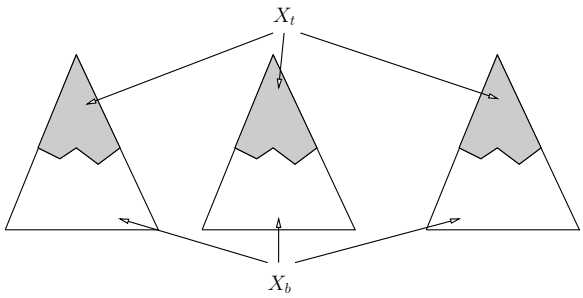

Figure 8.3: *Dissection of a forest $\mathcal{F}$ into X_t, the upper shaded portions, and X_b, the lower unshaded ones.*

Abbreviation: We refer to partial path compression operations as *compressions*.

Lemma 8.2 *Let C be a sequence of $|C|$ compressions in a forest $\mathcal{F}$ with node set X. Let X_b,X_t be a dissection of $\mathcal{F}$, and let the forests induced respectively by X_b and X_t be denoted $\mathcal{F}(X_b)$ and $\mathcal{F}(X_t)$. Let C be a sequence of $|C|$ compressions in $\mathcal{F}$. Then there exists a sequence C_b of compressions in $\mathcal{F}(X_b)$ and a sequence C_t of compressions in $\mathcal{F}(X_t)$ such that*

$$|C_b| + |C_t| = |C| \tag{8.1}$$

and

$$cost(C) \leq cost(C_b) + cost(C_t) + |X_b| + |C_t| \tag{8.2}$$

Proof: For each compression in C, we define a corresponding compression in C_b or C_t, and charge the cost of the compression to one of four "accounts". We then complete the proof by summing the amounts charged to the different accounts. Associated with each compression in C is a path p in $\mathcal{F}$, ending at node y.

Case 1: All nodes of p lie in X_b and the parent of y lies in X_b. Assign p to C_b and charge $cost(p)$ to account 1.

Case 2: All nodes of p lie in X_t. Assign p to C_t and charge its cost to account 2.

Case 3: Some nodes of p lie in X_b and some lie in X_t. Let p_2 be the part of p containing nodes in X_t. Assign p_2 to C_t and charge $cost(p_2)$ to account 2. For each node in $p \cap X_b$ whose parent is in X_b, charge 1 to account 3, and charge 1 to account 4. Then $|C| = |C_b| + |C_t|$ since each compression assigns a path to C_b or to C_t but not both. The total amount charged to all accounts is $cost(C)$, and one can verify that the amounts charged are

Account 1: $cost(C_b)$

Account 2: $cost(C_t)$

Account 3: at most $|X_b|$, since each element of X_b causes a charge of at most 1

Account 4: at most $|C_t|$, since the compression of each path in C_t causes a charge of at most one to this account

■

8.1.1 Union by Rank

Recall that, in executing UNION($\mathcal{S}(A), \mathcal{S}(B)$), one can either create a pointer from $\mathcal{S}(A)$ to $\mathcal{S}(B)$, or a pointer from $\mathcal{S}(B)$ to $\mathcal{S}(A)$. The UNION-BY-RANK rule specifies a way of making this choice which tends to keep the trees shallow. Associate with each node x a rank $rank(x)$ which is initially zero. When a UNION requires choosing either a pointer from x to y or y to x, the pointer should run from the node of lower rank to the one of higher. If the ranks are equal, then the choice is arbitrary, but the node to which the new pointer is directed has its rank incremented by 1. Figure 8.4 shows an example.

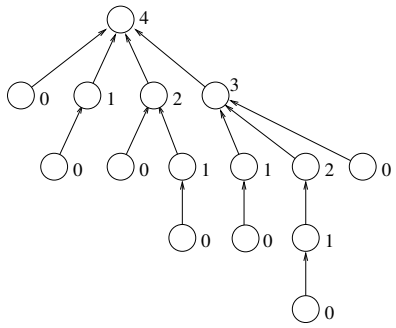

Figure 8.4: *The smallest tree whose root is of rank 4.*

In the scenario where all UNIONs precede all partial path compressions, the forest created by the UNIONs has the following properties:

(1) a node of rank k has at least one child of each rank $< k$,

(2) the rank of any node is less than the rank of its parent, and

(3) for any s, the number of nodes of rank greater than s is at most $n/2^{s+1}$, where $n = |X|$.

A forest produced using the UNION-BY-RANK operation, and therefore respecting (1), (2) and (3), is called a *rank forest*. Let $f(m, n, r)$ be the maximum cost of a sequence of m compressions applied to a rank forest of size n, in which the root is of rank r. We shall derive amazingly low upper bounds on $f(n, m, r)$. For example, $f(m, n, r) \leq \min(m + 4n, 2m + 2n)$ when $r \leq 65$.[1]

Let $g : N \to N$ be a function with $g(n) < n$ for $n > 0$. Define $g^0 : N \to N$ by

$$g^0(n) = \begin{cases} g(n) & \text{if } g(n) \leq 1 \\ 1 + g^0(\lceil \lg g(n) \rceil) & \text{if } g(n) > 1 \end{cases}$$

Our main result is:

Theorem 8.3 *If $f(m, n, r) \leq km + 2ng(n)$ for all m, n and r, then $f(m, n, r) \leq (k+1)m + 2ng^0(n)$ for all m, n and r.*

The proof will be given in the next lecture.

References

[1] R. Seidel and M. Shamin, *"Top-Down Analysis of Path Compression"*, SIAM J. Comput. Vol. 34, No. 3, pp. 515 – 525 (2005)

[1] Note that $n < 2^{66} \Rightarrow r \leq 65$, so that the upper bounds on f holds whenever $n < 2^{66}$.