Advanced Data Structures and Algorithms
Exercise Sheet 3

Stuart Golodetz

February 14, 2006

1. Using the disjoint-set forest representation with union by rank and path compression, find the data structure that results from executing the following program:

```
for i ← 1 to 16
  do MAKE-SET(x_i)
for i ← 1 to 15 by 2
  do UNION(x_i, x_{i+1})
for i ← 1 to 13 by 4
  do UNION(x_i, x_{i+2})
UNION(x_1, x_5)
UNION(x_11, x_13)
UNION(x_1, x_10)
FIND-SET(x_2)
FIND-SET(x_9)
```

Answer

See separate sheet.

2. Describe how you would implement a non-recursive version of FIND-SET with path compression. Your solution should use a (small!) constant amount of auxiliary storage.

Answer

Listing 1 Non-recursive FIND-SET

```
FIND-ROOT(x)
  y ← x
  while y ≠ p[y]
    y ← p[y]
  return y

FIND-SET(x)
  root ← FIND-ROOT(x)
  while x ≠ p[x]
    y ← p[x]
    p[x] ← root
    x ← y
  return root
```

3. Give a sequence of m MAKE-SET, FIND-SET and LINK operations, n of which are MAKE-SET operations, that takes time $\Omega(m \lg n)$ using the disjoint-set forest with union by rank, but not path compression.

Answer

Let MS $\equiv$ MAKE-SET, U $\equiv$ UNION and FS $\equiv$ FIND-SET.

Consider the sequence

$$\langle MS(1), ..., MS(n), U(1,2), U(2,3), ..., U(n-1,n)\rangle$$

This sequence has length $n + 3 \times (n-1) = 4n - 3$, since UNION involves two FIND-SET operations and a LINK.

After this, we end up with a tree of height bounded above by $\lfloor \lg n \rfloor$, since we're using union by rank. Let's assume we're in the worst-case scenario, i.e. that the height is actually $\lg n$. Let k be a lowest leaf node of the tree. Then FIND-SET(k) takes $\Omega(\lg n)$ work each time. So if we just use the sequence

$$\left\langle MS(1), ..., MS(n), U(1,2), U(2,3), ..., U(n-1,n), \underbrace{FS(k), ..., FS(k)}_{(m-(4n-3))\ times} \right\rangle$$

then this takes $\Theta(n) + \Omega(n) + (m - (4n-3))\Omega(\lg n)$ work. Well this is $\Omega((m - (4n-3)) \lg n)$ work. Now considering that we can choose $m \gg n$, this is just $\Omega(m \lg n)$ work.

(Strictly speaking, this is a sequence of MAKE-SET, FIND-SET and UNION operations, rather than LINK operations, but since UNION just involves doing FIND-SET and LINK, the distinction isn't important.)

4. TODO: I couldn't do this one, sorry.

5. Consider a version of the division method in which $h(k) = k \bmod m$, where $m = 2^p - 1$ and k is a character string interpreted in radix 2^p. Show that if string x can be derived from string y by permuting its characters, then x and y hash to the same value. Give an application where this would be undesirable in a hash function. [Hint: any permutation of a string can be decomposed into a sequence of interchanges of pairs of characters.]

Answer

(a) Suppose the character string is $[c_n, ..., c_0]$, then its interpretation k in radix $r = 2^p$ is:

$$\sum_{i=0}^{n} c_i r^i$$

Now, consider:

$$\begin{array}{ll} & h(k) \\ = & [\sum_{i=0}^{n} c_i r^i] \bmod (r-1) \\ = & [\sum_{i=0}^{n} c_i (r \bmod (r-1))^i] \bmod (r-1) \\ = & [\sum_{i=0}^{n} c_i . 1^i] \bmod (r-1) \\ = & [\sum_{i=0}^{n} c_i] \bmod (r-1) \end{array}$$

Now this is clearly independent of the order of the characters, since the summation is (addition of integers is commutative), so we can straightforwardly deduce our result.

(b) One (slightly pathological) application where this would be undesirable would be as follows: suppose you're given a set of characters $\{c_1, ..., c_n\}$ and told to find all anagrams of them which are English words.[1] To do this, we decide (in a moment of intense stupidity, since we generally don't need to keep all the permutations around simulataneously) to create a hash table of type $String \rightarrow Boolean$, add all possible permutations of the characters to it and iterate through them in order, checking whether or not they're in a list

[1] Since what we mean by 'words' here could be considered slightly hazy, I'll define it to mean 'words which you could use in a game of Scrabble', thus leaving the semantic quibbling to others.

of acceptable words and marking them one way or the other in the hash table. But since they all have the same characters, they all get hashed to the same slot and every lookup on the hash table is $\Theta(n!)$ (where note that n here is the number of characters: obviously it would only be linear in the number of permutations, but there are a lot of permutations).

(It should probably be noted that one way or another this method is going to require a LOT of memory. As noted above, it's not a good way of going about it, it's just to illustrate the point.)

6. Suppose we use a hash function h to hash n distinct keys into an array T of length m. Assuming simple uniform hashing, what is the expected number of collisions? More precisely, what is the expected cardinality of $\{\{k, l\} : k \neq l \text{ and } h(k) = h(l)\}$?

Answer

Using simple uniform hashing, the expected length of the list in any slot is $\frac{n}{m}$. The expected number of collisions in a given slot is the number of ways of choosing a pair of numbers from the list, i.e. $\binom{n/m}{2}$. So the expected number of collisions over all the slots is $m\binom{n/m}{2}$.