Maths 3467 quizzes
4: Thursday 21/11/13 due Thu 28/11

Please be sure you understand what is wanted. If in doubt, send e-mail. Collaboration is permissible.

(1) In analysing the UNION-FIND algorithm, there are other ways of defining 'rank' which, incorporated into the `union()` routine, and combined with path compression, also yield overall time $O((m+n)\log^* n)$. Here is an alternative definition of rank. We replace the array `size` by an array `rank`, initially all zero; and the `union` routine is altered as follows.

```
void union ( int x_1, int x_2 )
{
  if ( rank ( x_1 ) == rank ( x_2 ) )
  {
    parent (x_2) = x_1; rank (x_1) = 1 + rank(x_2);
  }
  else if ( rank (x_1) > rank (x_2) )
  {
    parent (x_2) = x_1;
  }
  else
  {
    parent (x_1) = x_2;
  }
}
```

As before, the (transient) rank of a node x increases until x acquires a parent, after which it remains fixed: the 'rank' of x is the final value stored in `rank(`x`).`

What is the significance of rank, as defined here, in terms of the reference forest? Justify your answer.

Answer. It is the height of nodes in the reference forest. This is true initially. Assume that rank of roots x_1 and x_2 of trees T_1 and T_2 equals the height, h_1 and h_2, of these trees before `union(x_1, x_2)`, and T with root x is the tree after the union.

Case (1): $h_1 = h_2$. After the union, all paths from the root x_1 of T to leaves in T_1 have length $\leq h_1$, and the longest path from x_1 to a leaf in T_2 has length $1 + h_2$, which is the new rank of x_1.

Case (2): $h_1 > h_2$. In T, h_1 is the maximum depth of leaves of T_1, and all leaves in T_2 have depth at most h_2 in T_2 and $1 + h_2$, which is $\leq h_1$, in T. So T has height h_1, equal to the rank of x_1.

Case (3): $h_1 < h_2$. As (2) except x_1 and x_2 exchange rôles. ∎

(2) Show that with this definition of rank, and this version of `union()`, every node of rank r has at least 2^r descendants in the reference forest.

Answer. By induction; true initially. Let T_1 and T_2 have roots x_1 and x_2 and heights h_1 and h_2 before `union(x_1, x_2)`, and let T be the tree, and h the rank of its root, after the union. By induction, $|T_i| \geq 2^{h_i}$, $i = 1, 2$. If $h_1 = h_2$, $|T| = |T_1| + |T_2| \geq 2^{h_1} + 2^{h_2} = 2^h$. If $h_1 > h_2$ then $|T| \geq |T_1| \geq 2^{h_1} = 2^h$, and if $h_1 < h_2$ then $|T| \geq 2^{h_2} = 2^h$. ∎

(3) Show that with this version of `union`, and path compression, the usual $O((m+n)\log^* n)$ bound holds. (Hint. The amortised analysis depends on three simple facts. First, rank increases strictly from node to parent. Hence, second, when a node acquires a new parent, its rank is greater than the previous parent's. Third, there are at most $n/2^r$ nodes of rank r.)

Answer. Since every time a node is involved in a FIND path, except for the root and its child on the path, the node acquires a parent of higher rank, a node in rank group r is involved in at most F_r FINDs before it acquires a parent in a higher rank group.

Since there are at most $n/2^r$ nodes of rank r, there are at most n/F_r nodes in the r-th rank group, so each rank group acquires a total charge of at most n in this way. Altogether, the total is $O(n \log^* n)$.

The $\log * n$ or so charged to the individual FINDs is the same as before, and UNIONs cost $O(n)$, so we get $O((m+n)\log^* n)$.

(4) Other path-compression schemes have been proposed. For instance, 'splitting.' It is somehow more elegant than the original form of path-compression because it traverses the path from node to root once, not twice.

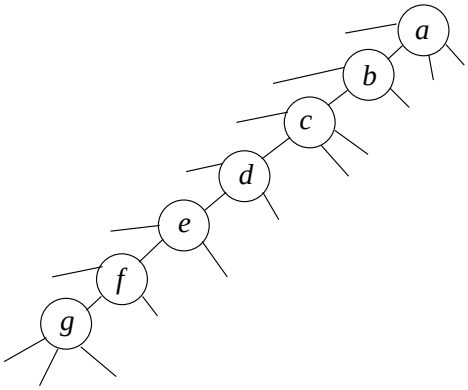
```
find ( int x )
{
  u = x;
  v = parent (x);  // parent (x) is -1 when x is a root

  if ( v < 0 ) // i.e., x is a root
    return x;

  while ( v >= 0 )
  {
    w = parent (v);
    if ( w < 0 )
      return v;
    else
    {
      parent (u) = w;
      u = v; v = w;
    }
  }
```

```
}
```

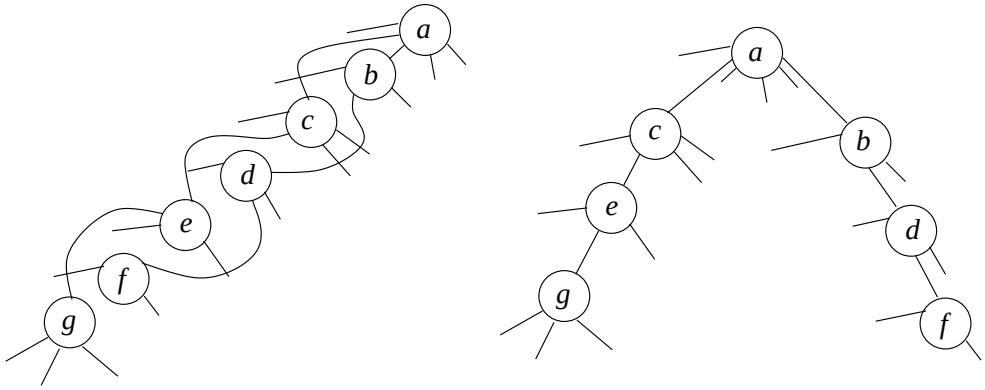
Simulate this on the path illustrated, that is, `find(`g`)` (a is the root.)



Answer.

x	u	v	w	parent
g				
	g			
		f		
			e	
				parent[g]=e
	f			
		e		
			d	
				parent[f]=d
	e			
		d		
			c	
				parent[e]=c
	d			
		c		
			b	
				parent[d]=b
	c			
		b		
			a	
				parent[c]=a
	b			
		a		
			-1	

```
return a
```



(5) Show that with this form of path compression, and either definition of rank, — you may choose either version of rank — the usual $O((m+n)\log^* n)$ bound holds. Same hints as in (3).

Answer. Again, there are at most r/F_r nodes in rank group r. Again, when a node x is involved in a FIND path then it acquires a parent of higher rank (except the last two on the path). This is enough for the $O((m+n)\log^* n)$ estimate. ∎