

Assignment 9. Sets and graphs

Given April 11, due April 18.

1. Suppose we have a family of disjoint sets represented by trees with parent pointers only. This is the data structure created by the `makeset`, `find`, and `union` methods. Suppose that there are n elements present. That is, that n `makeset` operations have been performed. Make no assumptions on how `union` works, so that the tree for a set may have height as large as the number of elements in the set. Show that m `find` operations with path compression take $O(n + m)$ work at most. Note that without path compression, each `find` operation could take n work, so the total would be $n \times m$ instead of $n + m$.
2. The *binomial tree* (related to binomial heap, CLR, chapter 20) of height $h + 1$, T_h , is defined inductively. T_0 is a single element. We have been saying that a tree with a single element has height 1, although one could make a case for saying height 0. T_{h+1} is made from two copies of T_h , call them T'_h with root x' , and T''_h with root x'' . In T_{h+1} , none of the pointers is changes except that the parent pointer of x'' is set to point to x' . There are no child pointers.
 - a. Show that T_h has height $h + 1$ and $n = 2^h$ nodes.
 - b. The *binomial coefficients*, $\binom{h}{k}$, satisfy the relation $\binom{h}{k} = \binom{h-1}{k} + \binom{h-1}{k-1}$. From this, find a formula for the number of nodes at depth k in T_h .
 - c. Show that T_h has the fewest elements among all trees of height h that can be made using `make_set` and `union` operations, assuming that we are using the union by rank heuristic. Hint: Let S_h be the sparsest tree of height h that can be made. Show that S_{h+1} is made by joining two copies of S_h , because any other join would either not increase the height, or have more elements than necessary.
3. In an undirected graph, an edge, `e`, connects `e.head` to `e.tail`. A graph, G' , is a subgraph of G if every vertex of G' is a vertex of G , and every edge of G' is an edge of G . A tree is a connected subgraph with no cycles. A spanning tree is a subgraph containing all the vertices of G that is a tree. A spanning forest is a collection of subgraphs, one spanning tree for each connected component. For this problem, the data structure is that there is a list of vertices, `V`, and a list of edges, `E`. An edge, `e` has two data elements, the head and tail (`e.head` and `e.tail` respectively). Although this is an undirected graph, we have not necessarily removed redundant edges, so it is possible that `e1.head = e2.tail` and `e2.head = e1.tail`. The following algorithm finds connected components:

```
for ( x in V ) { makeset(x); }
for ( e in E ) {
    if ( ( x = find(e.head)) != ( y = find(e.tail)) ) union(x,y); }
```

- a. Add a line or two to make a list of edges that forms a spanning tree if G is connected and a spanning forest otherwise. The work for this should be $O(e + v)$, where e is the number of edges (counting redundant edges if there are any) and v is the number of vertices.
- b. Write code to make this spanning tree into a tree sense we used before. That is, there should be a root, parent pointers, and child lists. For each vertex in the graph, create a tree element. Set parent and child pointers in the tree. Note that the head of an edge may be the parent or the child. You may construct the tree as you traverse it, but then you need some data structure to keep track of the edges you have selected so that you can find the edges in the spanning tree that touch a given vertex in $O(1)$ time. The total work for this should be $O(v)$.