Lecture 11, Oct 16, 2023

Disjoint Sets (Union/Find)

- Consider a situation where we have n distinct elements named $1, \cdots, n$; initially each element is in its own set, $S_1 = \{ 1 \}, \ldots, S_n = \{ n \}$
  - Each set is represented by some element x
- We want to support the operations:
  - UNION(S_x, S_y): create a new set $S = S_x \cup S_y$ and return the representative element of S
  - FIND(x): find the set containing the element x and return its representative element
- Using such a data structure we can test if two elements are in the same set by checking if FIND$(x) =$ FIND(y)
- Consider a sequence σ of m FIND operations and n UNION operations; we would like to analyze the complexity
- We can implement this using a disjoint-set forest:
  - Each set is represented by a tree, where the root node contains the set representative
  - Each node contains one element
  - Each non-root node points to its parent
  - Since we only need a parent pointer, this can be efficiently implemented using an array of n elements, with index i containing the parent of i
- The operations can be implemented as follows:
  - To find, we simply traverse up the tree until we reach the root
  - To merge, we find the root of both sets, and make one of them a child of the other
  - With this, in the worst case we can get a complexity of $O(mn)$ since merging sets can create a chain of m nodes
- To improve this, we can perform weighted union (WU) by size, i.e. every time we merge, we make the larger tree the parent
  - With this, any tree T created during the execution of σ has height at most $\log_2(n)$
    * Lemma: any tree T of height h created during the execution of σ has at least 2^h nodes
      - Base case: for $h = 0$, any tree of height 0 contains at least $2^0 = 1$ node
      - Inductive step: suppose the lemma holds for some h; we will show that it holds for $h + 1$
        - The tree must have been created by merging two trees, one of height h and one of height $h + 1$
        - By the inductive hypothesis the height h tree has at least 2^h elements
        - Since the smaller tree is the child, the height h tree must be the child, so the height $h + 1$ tree must be bigger and has more than 2^h nodes
        - Therefore overall the tree has at least $2^h + 2^h = 2^{h+1}$ nodes
    * Since $2^h \leq |T| \leq n$, we have $h \leq \log_2 n$
  - Therefore the worst case cost is $O(m \log n)$
- Another more effective technique is path compression (PC): after a FIND(x) operation, before returning, the parent of x is set to be the representative element of the set containing x, so that future FIND operations take a shorter path
  - This increases the cost of FIND, but makes future operations cheaper
  - This is called *amortization*

Important

Differences between our data structure and the one described in CLRS:

- We assume that x and y in the UNION operation are the representatives of their respective sets (as opposed to CLRS which does not require this).
- In our analysis it is assumed that we have n elements and m FIND operations (as opposed to m total FIND and UNION operations in CLRS)
- In our disjoint set forest, we are using a weighted union heuristic, i.e. union-by-size (as opposed to union-by-rank in CLRS)