Disjoint-Set Data Structures

Lecture outline

Pierre Senellart

15 November 2018

Call the roll. Preliminary questions?

1 (From Last Lecture) Red–black binary trees (13)

- Red–black trees. Properties of red–black trees:
  - The root is black.
  - If a node is red, its children are black.
  - Same number of black nodes on every path from one node to its descendants with 0 or 1 child.
- Example 13.4 (without the z node)
- Bound (proved last week): $h = O(n)$
- Left rotation, right rotation
- Insertion in a red–black tree:
  - Insertion of z at its natural position, red colored
  - Correction of red–black violation between z and its parent:
    * If z is the root, we color it in black.
    * If z's parent is black, nothing to do.
    * If z's parent is red (and thus its grandparent is black):
      · Deal with the case where z's parent is a left child, the other case is symmetric
      · If the uncle of z is red, it is colored in black together with z's parent, and the grandparent is colored in red. Recursively process the grandparent.
      · Else, if z is a right child, we left-rotate z and its parent and consider the new left child of z and move to the following case.
      · z is a left child, we color its parent in black and its grandparent in red, then right-rotate z's parent and z's grandparent. Done.
  - Complexity
- A word on deletion, no detail

2 Disjoint-set data structures (21)

- Data structure, basic operations: MakeSet, FindSet, Union
- Example application: connected components in a dynamic graph
- Data structure: forest with roots representative elements
- Sequence of m operations including n MakeSets
- First idea: shallow forest; complexity
- Better idea: weighted union heuristics; complexity
- Two heuristics:
  - Union by rank
  - Path compression
- Non-tight complexity analysis: proof of $O(m \log^* n)$:
  - Basic properties:
    * Rank of a node doesn't change when not root any more
    * Rank of a node < rank of its parent
    * At least 2^r nodes in a subtree with rank-r root at the time it got its rank
    * Maximum number of nodes of rank r: $\frac{n}{2^r}$
    * Number of nodes of rank in $[k, 2^k)$ at most $\frac{n}{2^{k-1}}$
  - Count the total complexity of finds by counting the number of child-parent pairs encountered:
    * that are in different buckets
    * that are in the same buckets (rank of a child remains constant!)
- Mention the tight complexity analysis: $O(m\alpha(n))$. Implications.
- How to get all elements in a set?