CSC263 - Assignment 2

Cristyn Howard

Monday, January 29, 2018

Question 1

Part a)

- Let $\alpha(n)$ denote the number of 1's in the binary representation of n.
- Lemma: based on the structural definition of binary forests, a binary forest with n nodes contains $\alpha(n)$ trees.
- Lemma: a tree with n nodes contains $n-1$ edges.
- Let the trees in a binary heap with n nodes be labelled G_i, where $1 \leq i \leq \alpha(n)$, and let k_i denote the number of nodes in tree G_i.
- We know that $\sum_{i=1}^{\alpha(n)} k_i = n$, so then $\sum_{i=1}^{\alpha(n)} (ki-1) = n - \alpha(n)$.
- Thus we can conclude that a binary heap with n nodes contains $n - \alpha(n)$ edges.

Part b)

- Binomial heap H has n nodes and $n - \alpha(n)$ edges before any insertions.
- After k insertions, H has $n+k$ nodes and $(n+k) - \alpha(n+k)$ edges.
- The number of edges created by k insertions is thus $[(n+k)-\alpha(n+k)]-[n-\alpha(n)] = n+k-\alpha(n+k)-n+\alpha(n) = k + \alpha(n) - \alpha(n+k)$.
  - Note that $\alpha(n+k) \geq 0$, and so $k + \alpha(n) - \alpha(n+k) \leq k + \alpha(n)$.
  - Further, note that $\alpha(n) \leq \lfloor \log_2 n \rfloor + 1$ (by the definition of $\alpha(n)$).
  - So all together we have $k + \alpha(n) - \alpha(n+k) \leq k + \lfloor \log_2 n \rfloor + 1$.
- Pairwise comparisons between node values (which occur in constant time) take place whenever an insertion of an element into H creates a new edge, as the newly connected nodes must be compared and potentially swapped to preserve the heap order property.
- The dominant factor affecting the worst case running time of inserting k elements into H is the number of comparisons that must occur, or the number of edges that must be created, which we know is bounded above by $k + \lfloor \log_2 n \rfloor + 1$.
- When $k < \log_2 n$, the dominant term in $[k + \lfloor \log_2 n \rfloor + 1]$ is $\log_2 n$, so the worst case running cost of inserting k elements into H is $O(\log_2 n)$.
- However when $k > \log_2 n$, the dominant term in $[k + \lfloor \log_2 n \rfloor + 1]$ is k, so the worst case running cost of inserting k elements into H is $O(k)$, and the average cost of inserting 1 element is $O(k/k)$, or $O(1)$.

Question 2

```
OUTER(u)
    return check-bal(u, TRUE)


check-bal(x, isRoot)
    if (x == NIL) return NIL

    dist-long = 0, dist-short = 0

    if (!(x.lchild == NIL && x.rchild == NIL))

        if (x.lchild != NIL && x.rchild != NIL)

            ldist = check-bal(x.lchild, FALSE)
            if (ldist == FALSE) return FALSE

            rdist = check-bal(x.rchild, FALSE)
            if (rdist == FALSE) return FALSE

            dist-long = max(ldist[0], rdist[0]) + 1
            dist-short = min(ldist[1], rdist[1]) + 1

        else
            child-dist = NIL;
            if (x.lchild != NIL && x.rchild == NIL)
                child-dist = check-bal(x.lchild, FALSE)
            else if (x.rchild != NIL && x.lchild == NIL)
                child-dist = check-bal(x.rchild, FALSE)

            if (child-dist == FALSE) return FALSE

            dist-long = child-dist[0] + 1
            dist-short = child-dist[1] + 1

        if (dist-long > 2 * dist-short) return FALSE

    if isRoot
        return TRUE

    return [dist-long, dist-short]
```

- The wrapper function contains only constant time operations (besides the call to check-bal).
- Base case inputs to check-bal that trigger no recursive calls (x = leaf, x = NIL) contain only constant time operations.
- In non-base cases (when x has at least one child), all operations that are not recursive calls (evaluating if conditions, doing arithmetic operations, executing return statements) occur in constant time.
- Then the dominant factor in the running time is the number of recursive calls.
- Whenever x has one child, check-bal is called on that child, and whenever x has two children, check-bal is structured to call on both of them. So check-bal is structured to call recursively on all nodes of the subtree rooted at x.
- When x is the root of a 2-balanced BST (no recursive calls to check-bal return FALSE), we know that check-bal is called recursively <u>on all nodes</u> of the BST, thus there exists an input $\forall n \in \mathbb{N}$ where check-bal is called n times. So $T \in \Omega(n)$.
- Check-bal is called <u>at most</u> once per node, therefore the maximum number of check-bal calls for a BST of size $n \in \mathbb{N}$ is n. So $T \in O(n)$.
- Thus the worst case running time of this algorithm is $\Theta(n)$, where n is the $\#$ of nodes in the BST rooted at u.