Problem sheet 4

COL106 (Data Structures), Semester I, 2018-19, IIT Delhi

Amitabha Bagchi
Department of CS&E, IIT Delhi

Thanks are due to Rajbir Malik for keeping a record of various problems arising during lecture.

Questions with a * may be difficult

November 4, 2018

Exercise 1
Recall that in the definition of the Binary Search Tree we said that for any node containing key x all the nodes in the left subtree have keys $\leq x$ (i.e. can contain x itself) while all the nodes in the right subtree have keys $> x$ (i.e. keys that are *strictly* greater than x. Let us call this the *strict BST*. In class some students claimed that this strict BST invariant is not compatible with the AVL tree property (height balance property). They gave me the following example: 10, 11, 11.

1. Convince yourself that if we try to create an AVL tree with the three keys 10, 11 and 11 we cannot maintain the height balance property.
2. The implication of this example is this: either (a) we do not allow duplicates in AVL tree or (b) we relax the strict condition on the right subtree, i.e., keys in the right subtree of the node containing key x are allowed to be $\leq x$. Suppose we implement (b): How does the find algorithm change? Does its running time increase? Does its running time increase asymptotically?
3. (*) Suppose we implement (a), i.e., no duplicates allowed, prove that it is always possible to construct a AVL tree that maintains the strict BST invariant.

Exercise 2 (*)
In class I said that the problem with AVL trees is that each node has to maintain its height and that leads to the AVL tree node not remaining a fixed-size node. At this one of your classmates suggested that we maintain only the height imbalance at each node, i.e., at any node we maintain the quantity "height of left subtree - height of right subtree." Clearly, if the AVL tree height balance invariant is maintained this quantity can only take three values: -1, 0 and 1. And so we can store it in 2 bits. The question is: If this is the only information we maintain can we still do insertion and deletion in $\theta(\log n)$ time? Either show that we can or argue why this information is insufficient. Note that we still want $\theta(\log n)$ time. If we had as much time as we needed we can address height imbalanace by just rebuilding the AVL tree from scratch (time is just $\theta(n)$ if we use the correct method).

Exercise 3
We encountered a deterministic version of the skip list, called *1-2-3 skip lists* in the exam. Recall the structural invariant:

> In a 1-2-3 skip list S, given 2 keys $x < y \in S$, if $h = \min\{h(x), h(y)\} \geq 1$ there must be at least one and at most 3 keys $z \in S$ such that $x < z < y$ and $h(z) = h - 1$.

The lists are numbered from below starting with 0. The height of an element x, $h(x)$, is the number of the highest list to which x has been promoted. Assume there are *no duplicates* in S. Note that we will always assume there are two *sentinel nodes* with keys $-\infty$ and $+\infty$ with height one more than the height of the highest key in the 1-2-3 skip list so that the invariant is not trivially satisfied by a single linked list.

Study the deletion operation of the 2-4 tree and try to work out the deletion operation of the 1-2-3 skip list in analogy with that.

Exercise 4
Given an array A with n integers in it, one way of measuring the distance of the array from a sorted array is by counting *inversions.* A pair of indices $i, j \in \{0, \ldots, n-1\}$ is called an inversion if $i < j$ and $A[i] > A[j]$. Clearly a completely sorted array has 0 inversions. Now answer the following questions.

- What is the maximum number of inversions possible in an array with n elements? What sequence achieves this number?
- If the number of inversions is k is it possible to sort the array in $\theta(n+k)$ steps? Is there a standard sorting algorithm that can do this. Choose one out of insertion sort, selection sort, heap sort, bubble sort, merge sort and quicksort.
- Characterize the worst case running time of all the remaining algorithms out of the list given above in terms of n and k.

Exercise 5
Given a set of key pairs $\{(x_i, y_i) : 1 \leq i \leq n\}$ we say a sorting algorithm is stable if when we apply it to sort the pairs considering *only* the first coordinate of the pairs then the duplicates are in the same order as in the input, e.g., if $x_1 = x_{20}$ then we are guaranteed that (x_1, y_1) appears *before* (x_{20}, y_{20}) in the sorted order.

In class we showed that a natural implementation of insertion sort and selection that is stable. Either show the same about merge sort, quick sort and heap sort or show with examples that it is not easy to implement these in a stable way.

Exercise 6
Given a simple undirected graph with no self-loops, $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, an $n \times n$ matrix A is said to be the *adjacency matrix* of G if $A_{i,j}$ is 1 if $(i, j) \in E$ and 0 otherwise.

Now, suppose we define a special kind of matrix product for matrices whose entries are only 0 or 1. In this special matrix product we replace addition by OR and multiplication by AND. With this being our definition of the matrix product answer the following questions

1. If $B_2 = (A+I)^2$ then argue that $B_2 = C_2 + I$ where C_2 is the adjacency matrix of the graph $G' = (V, E')$ with the same vertex set as G. We say that $(i, j) \in E'$ if $(i, j) \in E$ or there exists a $k \in V$ such that $(i, k), (k, j) \in E$.
2. * Clearly the time taken to compute C_2 by the above method is $\theta(n^2)$. Is it possible to compute C_2 (by some other way perhaps) in time which is $o(n^2)$? Or is there some example on which the time taken to compute C_2 has to be $\Omega(n^2)$?
3. For any $\ell \geq 2$, we say that there is a path of length ℓ between vertices i and j of V in G if there exists a sequence $k_0, k_1, \ldots, k_\ell$ such that $(k_i, k_{i+1}) \in E$ for $0 \leq i \leq \ell - 1$ and $k_0 = i, k_\ell = j$. Prove that for any $\ell \geq 2$ if $B_\ell = (A+I)^\ell$ then $B_\ell = C_\ell + I$ where C is the adjacency matrix of the graph $G' = (V, E')$ with the same vertex set as G whose edge set has the property that $(i, j) \in E'$ if either $(i, j) \in E$ or there is path of length at most ℓ between i and j in G.

4. Two vertices $i, j \in V$ are said to be *connected* in G if $(i, j) \in E$ or there is a path of length ℓ between them in G for some $2 \leq \ell \leq n$. If this is true we denote it by $i \leftrightarrow' j$. We now augment the relation $\leftrightarrow'$ by adding in the pairs $\{(i, i) : 1 \leq i \leq n\}$. We denote this augmented relation $\leftrightarrow$.

   (a) Show that $\leftrightarrow$ is an equivalence relation, i.e., that it is reflexive, symmetric and transitive.

   (b) The equivalence classes of $\leftrightarrow$ are called the *connected components* of G.

   (c) Compute C_n as described above by computing $B_n = (A + I)^n$. If $U \subseteq V$ is a connected component of V then prove that $A_{ij} = 1$ for all $i, j \in U$ such that $i \neq j$.

   (d) Under what condition is there an $\ell < n$ such that C_ℓ has the same property as C_n that we proved in the previous question?