Homework 3

Fundamental Algorithms, Spring 2003, Professor Yap

Due: Wed March 5, in class

INSTRUCTIONS:

- REMINDER: Our Midterm Exam will be held in the Week of March 10th. The midterm will be held either on Monday or Wednesday.
- Please read the questions carefully. When in doubt, please ask.

---

1. (20 Points)
Let us see the details for implementing insertion in (a, b)-trees. Assume each node u stores an array of alternating "key/references" elements:
$$u = (r_0, k_1, r_1, k_2, r_2, \ldots, r_{m-1}, k_m, r_m)$$
where r_i's are references (or pointers) to nodes, and k_j's are keys, with
$$k_1 < k_2 < \cdots < k_m.$$
So this node has m keys, but $m + 1$ references. We have the constraint $a \leq m \leq b$. In case u is the root, then we only have $2 \leq m \leq b$. Recall the method described in the Lecture Notes, where a, b satisfy the inequality
$$a \leq \frac{2b + 1}{3}.$$
(a) Suppose we insert a new key k into u. If the resulting node has $\leq b$ keys, we are done. So assume that u now has $b + 1$ keys. Suppose u is the root. Describe how to split u and form a new root.
(b) Continuing from (a), assume that u is not the root. Then we look at the siblings of u (note that u has either one or two siblings). If any sibling has less than b keys, then we can move one of our keys to this sibling. Describe this "move" in detail. HINT: keys and references in the parent of u is affected as well.
(c) Suppose that (b) fails. Then u has $b + 1$ keys and it also has a sibling v with b keys. Describe how to split u and v into three nodes, and modify the (a, b)-tree. HINT: Be sure to describe how this affects the common parent w of u and v. Furthermore, you must clearly describe the recursive nature of this process.

   NOTE on writeup: we prefer that you describe (a) and (b) without using a specific programming language, but in precise (mathematical) terms. If you must using a programming language, use it ONLY to supplement your English prose. Drawing pictures is very useful, but pictures are silent – you must describe explicitly how to interpret your pictures.

2. (20 Points)
Describe an efficient algorithm which, given two closed paths $p = (v_0, v_1, \ldots, v_k)$ and $q = (u_0, u_1, \ldots, u_\ell)$, determine whether they represent the same cycle, i.e., they are equivalent. What is the complexity of your algorithm? Make explicit any assumptions you need about representation of paths and vertices.

3. (20 Points)
(a) Describe an efficient algorithm for computing the reduced graph G^c for any input digraph G. Assume the digraph has an adjacency list representation, and that its vertex set is $V = \{1, 2, \ldots, n\}$. State other assumptions you need. If the reduced graph G^c has m vertices, assume that the vertex set of G^c are indexed by $\{1, 2, \ldots, m\}$.
(b) Analyze the running time of your algorithm in (a). This can be fairly brief.

4. (20 Points)
In Lecture IV, §7, we defined what it means for a permutation array $per[1..n]$ to be a "topological ranking" of a DAG G.
(a) Modify the Simple DFS Algorithm in the Lecture Notes to compute such a permutation. The idea is this: *when you reach a leaf in the DFS Tree, you know that you can safely put this leaf at the end of the permutation.* HINT: you will need a Driver Program for the Simple DFS Algorithm. You need only to specify the "Application Specific" subroutines such as INIT, VISIT and POSTVISIT used in the Simple DFS Algorithm.
(b) Briefly argue why your algorithm is correct.