Computer Science 221

Solutions to Practice Questions - Set 4
Data Structures and Graph Theory

1. The algorithms are straightforward; you just need to account for the complexities for the given calls (i.e., the data structure itself has been analyzed for you).

   For Algorithm A:

   - The open operation is O(1).
   - The while() loop is executed O(u) times.
     - Inserting a 6-tuple in the ROCK-tree can be done in $O(6 \lg n) \in O(\lg n)$ time
     - Other operations are O(1), and we can ignore them.

   Solution: Algorithm A runs in O(u) * O(lg n) => O(u lg n) time. You could also write O(u lg(n+u)) time, since the tree will slowly grow as you insert elements.

   For Algorithm B:

   - Searching the ROCK-tree takes O(lg n) time to find the band's tuple, PLUS $O(6d^2 + x) \in O(d^2 + x)$ time to return the addresses of all tuples in the hypersphere centered at the band's location. This sums to $O(\lg n + d^2 + x)$ time. Furthermore, regardless of whether or not the same call tells us the number of tuples in the hypersphere, an extra O(x) factor for adding them up adds nothing to the complexity. Note that we can't simplify the expression further since n, d, and x are all unknowns at this point. It is true that x could be n in the worst-case, but we're after the *expected* number of tuples, so we should leave in the x.
   - Printing out all x tuples in the results list, means using their addresses to get the data. Knowing their address, it's O(1) time; so, overall, it's O(x) time. (If the function had returned the band names instead of the tuples' addresses, it would be O(lg n) extra lookup time per band, or O(x lg n) time, instead.)

   Solution: Algorithm B runs in $O(\lg n + d^2 + x) + O(x)$ expected time, which is $O(\lg n + d^2 + x)$ time.

2. (a) `floor`(4096 / (30+10)) = 102 data entries per leaf page. (You can also say "4096-10" instead of "4096", but we can simply ignore any space for sibling pointers and overhead bytes.)

   (b) (i) `ceiling`(10000 data entries / 102 data entries/page)=99 leaf pages

   (ii) `ceiling`(99 leaf pages / 102 leaf pages per next-level-up page) = 1 next-level-up page

   When we reach "1", we stop; this represents the root page.

   Conclusion: We need 1 + 99 = 100 4K pages.

   (c) (a) `floor`(4096 / (30 + 5(4))) = 81 data entries per leaf page

   (b) (i) `ceiling`(10000 data entries / 81 data entries per leaf page) = 124 leaf pages needed

   (ii) An interior page can hold: `ceiling`(4096 / (30+10)) = 102 keys (and 103 pointers accompanying them). Therefore, we estimate: `ceiling`(124 leaf pages / 103 leaf pages per next-level-up page) = 2 next-level-up pages. Finally, we compute `ceiling`(2 level-one pages / 103 level-one pages per level-zero page) = 1 level-0 page. This is where we stop; it's the root.

   Conclusion: We need 1 + 2 + 124 = 127 4K pages.

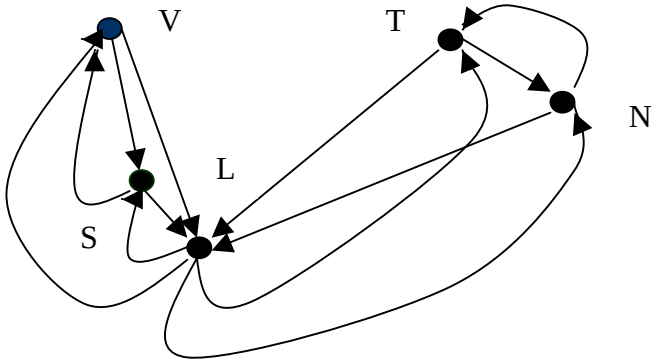
3. (a) If we assume that "visiting an airport" means a return trip, then the airports can be visited in 4! ways, with Vancouver being fixed at the start and the end of the trip. When in Vancouver, they can visit (permute) Vancouver, Burnaby, and Surrey in 3! ways. Furthermore, they can visit 0, 1, 2, or all 3 of these cities (Vancouver, Burnaby, Surrey) at the start (with the rest at the end). This is 4*4!*3! ways overall.

   (b) Visiting each vertex once, with the same start/end vertex is a Hamilton cycle (of the airport cities). A Hamilton cycle is possible since we can get from any city to any other. For example, one possibility is: Van-Tor-NY-LA-SF-Van.

   (c) There are at least 2 interpretations here. (i) If we assume that the band has to start its tour in Vancouver, then no Hamilton path is possible since we'd have to visit Vancouver twice during a path (e.g., V-S-B-V-Tor-NY-LA-SF or V-Tor-NY-LA-SF-V-S-B). (ii) If we assume the band starts its tour in the first city that it plays in, then we can visit Burnaby and Surrey by appending or prefixing them

to Vancouver. Thus, S-B-V-Tor-NY-LA-SF is a valid Hamilton path that visits all 7 cities.

4. We can take the adjacency matrix and turn it into a directed graph:

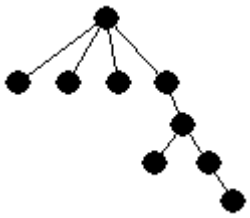


An Euler cycle requires that we visit each edge exactly once, and return to the same city that we started from. Notice that there are no vertices with odd degree, and that there are as many outgoing edges as ingoing edges for each vertex.

One Euler cycle is: V-S-L-N-T-L-T-N-L-V-L-S-V.

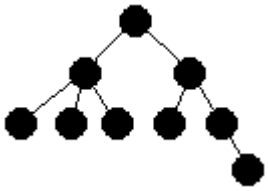
5. (a) The graph is a tree, because there is a unique simple path between any two vertices.

4-ary Rooted Tree



Note that any tree with a maximum of 4 branches per child can be defined as a 4-ary tree.

Balanced Tree



Note that a tree of height h is balanced if all leaves are at most one level apart (i.e., at levels x or x+1).

(b) The graph is not a tree, because there are two vertices with multiple simple paths.

Since the graph is not a tree, it can be neither a 4-ary rooted tree nor a balanced tree.