HW 09 textbook problems and hints

4.4 (1 – 5)

10. *Chocolate bar puzzle* Given an n-by-m chocolate bar, you need to break it into nm 1-by-1 pieces. You can break a bar only in a straight line, and only one bar can be broken at a time. Design an algorithm that solves the problem with the minimum number of bar breaks. What is this minimum number? Justify your answer by using properties of a binary tree.

10. Breaking the chocolate bar can be represented by a binary tree.

5.3 (2 - 9)

9. A digraph is called ***strongly connected*** if for any pair of two distinct vertices u and v, there exists a directed path from u to v and a directed path from v to u. In general, a digraph's vertices can be partitioned into disjoint maximal subsets of vertices that are mutually accessible via directed paths of the digraph; these subsets are called ***strongly connected components***. There are two DFS-based algorithms for identifying strongly connected components. Here is the simpler (but somewhat less efficient) one of the two:

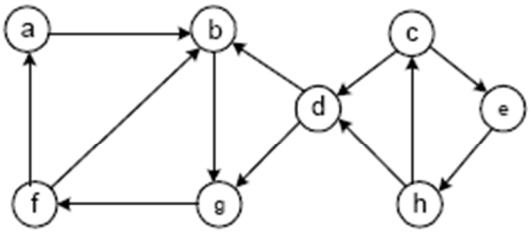
   Step 1. Do a DFS traversal of the digraph given and number its vertices in the order that they become dead ends.

   Step 2. Reverse the directions of all the edges of the digraph.

   Step 3. Do a DFS traversal of the new digraph by starting (and, if necessary, restarting) the traversal at the highest numbered vertex among still unvisited vertices.

   The strongly connected components are exactly the subsets of vertices in each DFS tree obtained during the last traversal.

   a. Apply this algorithm to the following digraph to determine its strongly connected components.



   b. What is the time efficiency class of this algorithm? Give separate answers for the adjacency matrix representation and adjacency list representation of an input graph.

   c. How many strongly connected components does a dag have?

9. a. Trace the algorithm on the input given by following the steps of the algorithm as indicated.

   b. Determine the efficiency for each of the three principal steps of the algorithm and then determine the overall efficiency. Of course, the answers will depend on whether a graph is represented by its adjacency matrix or by its adjacency lists.

5.4 (3- 9, 4 - 5, 5 - 5)

2. Generate all permutations of $\{1, 2, 3, 4\}$ by

   a. the bottom-up minimal-change algorithm.

   b. the Johnson-Trotter algorithm.

   c. the lexicographic–order algorithm.

2. We traced the algorithms on smaller instances in the section.

10. ▶ Design a decrease-and-conquer algorithm for generating all combinations of k items chosen from n, i.e., all k-element subsets of a given n-element set. Is your algorithm a minimal-change algorithm?

10. There are several decrease-and–conquer algorithms for this problem. They are more subtle than one might expect. Generating combinations in a predefined order (increasing, decreasing, lexicographic) helps with both a design and a correctness proof. The following simple property is very helpful. Assuming with no loss of generality that the underlying set is $\{1, 2, ..., n\}$, there are $\binom{n-i}{k-1}$ k-subsets whose smallest element is i, $i = 1, 2, ..., n-k+1$.

11. *Gray code and the Tower of Hanoi*

   (a) ▷ Show that the disk moves made in the classic recursive algorithm for the Tower-of-Hanoi puzzle can be used for generating the binary reflected Gray code.

   (b) ▶ Show how the binary reflected Gray code can be used for solving the Tower-of-Hanoi puzzle.

11. Represent the disk movements by flipping bits in a binary n-tuple.

5.5 (6 - 6)

2. Consider ***ternary search***—the following algorithm for searching in a sorted array $A[0..n-1]$: if $n = 1$, simply compare the search key K with the single element of the array; otherwise, search recursively by comparing K with $A[\lfloor n/3 \rfloor]$, and if K is larger, compare it with $A[\lfloor 2n/3 \rfloor]$ to determine in which third of the array to continue the search.

   a. What design technique is this algorithm based on?

   b. Set up a recurrence relation for the number of key comparisons in the worst case. (You may assume that $n = 3^k$.)

   c. Solve the recurrence for $n = 3^k$.

   d. Compare this algorithm's efficiency with that of binary search.

2. The algorithm is quite similar to binary search, of course. In the worst case, how many key comparisons does it make on each iteration and what fraction of the array remains to be processed?