The University of British Columbia

Department of Computer Science

Computer Science 420—Advanced Algorithm Design and Analysis

Homework Assignment 8/9

Due: 2015 April 09

Please review the class policy on collaboration before starting this assignment. You are free to discuss problems in groups of size at most three. However, your actual homework submission must be prepared on your own. At the top of the first page of every homework submission you must clearly acknowledge all sources (including books, websites and discussions with fellow students) that you have used in the preparation of your submission.

1) Two natural generalizations of the maximum flow problem are as follows:

(i) the given network G has $a \geq 1$ source vertices $s_1, s_2, \ldots, s_a$ and $b \geq 1$ sink vertices $t_1, t_2, \ldots, t_b$, and our objective is to maximize the *total* flow, from all sources to all sinks; and

(ii) each vertex of the network G also has an associated capacity $c(v)$ that restricts the flow that can enter that vertex.

a) Describe an efficient reduction of generalization (i) to the standard maximum flow problem

b) Describe an efficient reduction of generalization (ii) to the standard maximum flow problem

2) An *independent set* in an undirected graph $G = (V, E)$ is a subset $I \subset V$ such that no edge in E joins two vertices in I. We want to show that finding a maximum size independent set in a *tree* T can be done efficiently.

The idea is to choose an arbitrary vertex r of T as the root, and direct all edges away from the root. Then observe that (i) the maximum size independent set that includes r (a) includes none of the children of r, and (b) can be assumed to include the maximum size independent sets in the subtrees rooted at the grandchildren of r, and (ii) the maximum size independent set that excludes r can be assumed to include the maximum size independent sets in the subtrees rooted at the children of r.

a) Describe an efficient algorithm that, given a tree T and a specified root vertex r, labels the vertices of T from 1 to n in such a way that the labels associated with the children of a vertex v (according to the rooting of T at r) are larger than the label associated with v.

b) Suppose the vertices of T have been labeled as in part (a). Give a recursive formula for MIS(i) that specifies the size of the maximum independent set of the subtree rooted at the vertex with label i.

c) What is the cost of determining MIS(1), the maximum independent set of the full tree?

d) Show how to output a maximum size independent set in T, given the values in MIS[$1:n$].

3) In the *knapsack problem*, we are given a knapsack of some integer capacity W, and a collection of n items. The i-th item has a weight w_i and a value v_i, both of which are positive integers. The objective is to pack items of maximum total value into the knapsack, subject to the constraint that their total weight cannot exceed the capacity W.

In the discrete (0-1) version of the problem items must either be chosen or not chosen. In the continuous (*fractional*) version, it is permitted to choose only a fraction (between 0 and 1) of an item, with the correspondingly reduced weight and value.

a) Prove that for the fractional version a solution (the optimal choice) includes a non-zero amount of item i only if it includes *all* of each item j with $v_j/w_j > v_i/w_i$.

b) Give an efficient reduction showing that that the $0-1$ version is at least as hard as the *subset-sum problem*: given a set $S = \{x_1, x_2, \ldots, x_n\}$ of positive integers and a positive integer t, determine whether there exists a subset of S whose elements add up to the target value t.

c) Describe an $O(nW)$-time dynamic-programming algorithm for the $0-1$ version, where n is the number of items and W is the knapsack (weight) capacity.

d) Use the result of part (c) to design an approximation algorithm for the $0 - 1$ version that guarantees a solution that is at least as large as the optimal using no more than $1 + \epsilon$ times the allowed capacity, for an arbitrary $\epsilon > 0$. Show that the cost of your algorithm grows only as some polynomial in $1/\epsilon$. (Hint: round the weight of each item up to the next integer multiple of $\epsilon W/n$.)

4) Let S be a set of n integers. We say that S is *degenerate* if there exist $a, b, c \in S$ such that $a+b+c = 0$. Let A, B and C be three sets containing a total of n integers. We say the sets are *sum-related* if there exist $a \in A$, $b \in B$, and $c \in C$ such that $a + b = c$.

a) Show that, for any $k \geq 1$, an $O(n^k)$-time algorithm for testing for the sum-relatedness of A, B and C can be used to test the degeneracy of S in $O(n^k)$ time.

b) Show that, for any $k \geq 1$, an $O(n^k)$-time algorithm for testing for the degeneracy of S can be used to test for the sum-relatedness of A, B and C in $O(n^k)$ time.

c) A given set P of n points in the plane is said to be in *general position* if no three distinct points in P are colinear (i.e. lie on a common line). Show that, for any $k \geq 1$, an $O(n^k)$-time algorithm for testing for the general position property for P can be used to test the degeneracy of S in $O(n^k)$ time. (Hint: try projecting points of S onto the curve $y = x^3$.)

5) [continuing from Question 1 on Assignment 5...] We return to study B-S arrays, this time including the DELETE operation. Note: as is common practice, it is assumed that if you wish to DELETE an element then you are given a pointer to that element (i.e. it is not necessary to search for the element being deleted.)

a) Describe an implementation of the DELETE operation for B-S arrays that reestablishes the B-S array invariant after every deletion. What is the worst case cost of DELETE operations in your implementation? Explain.

b) We can modify the B-S array invariant to permit *lazy* DELETE operations, by associating with each array location a delete-flag and maintaining the property that every non-empty block has less than half of its elements flagged for deletion. Show that with this change the amortized cost of DELETE operations is $O(1)$. How does this modification affect the cost of MEMBER and INSERT operations?