BU CLA CS 530: Analysis of Algorithms
9 May 1995
End-of-Term Test

Reminder: This is a closed-book test. You are allowed one 8.5 by 11 crib sheet which you have to hand in with your test book.

Read carefully all questions before you start, organize your time, start with what you know best.

Problem 1 (5 points).

Let $G = (V, E)$ be an undirected weighted graph. Let T be a *minimum spanning tree* (MST) of G. Suppose now that all the weights in G are increased by the same constant $c \geq 1$. Is T still a MST? If not, how would you modify T into a MST? Justify briefly.

Problem 2 (10 points).

Consider the search problem in which a given ordered list of $n \geq 1$ items (with possible duplicates) is to be searched for the index of an item equal to a given value X, if any such index exists. Binary search solves this problem in sequential time $\mathcal{O}(\log n)$.

(a) Design a time-efficient CREW parallel algorithm for this problem, using as many processors as you need.

(b) What is the parallel running time of your algorithm? Is your algorithm work-efficient? Justify.

Problem 3 (10 points).

Let G be a directed graph with positive and negative edge weights. A path is said to be *simple* if it has no repeated vertices. Let s and t be distinct vertices of G. Show that the problem of determining whether G contains a simple path from s to t of negative weight is:

(a) in NP, and

(b) NP-hard. (Hint: Reduction from the directed-hamiltonian-cycle prolem.)

Problem 4 (10 points).

Consider the following Monte Carlo algorithm:

MAJOR($A[1 \,..\, n]$)
$i \leftarrow$ RANDOM$(1, n)$
$x \leftarrow A[i]$
$k \leftarrow 0$
for $j \leftarrow 1$ **to** n
  do if $A[j] = x$ **then** $k \leftarrow k + 1$
if $k > n/2$ **then return** *yes*
  else return *no*

Algorithm MAJOR decides whether the array $A[1..n]$ contains a majority element, with some error probability p. RANDOM is a random-number generator; a call to RANDOM$(1, n)$ returns an integer between 1 and n, inclusive, with each such integer being equally likely.

(a) Determine a tight upper bound on the error probability p. (Hint: Consider separately two cases, when $A[1..n]$ contains a majority element and when $A[1..n]$ does not contain one.)

(b) Consider another algorithm MAJOR′ which calls MAJOR as a subroutine:

```
MAJOR'(A[1 . . n])
1.   if MAJOR(A[1 . . n]) = yes
2.      then return yes
3.      else  return MAJOR(A[1 . . n])
```

Determine a tight upper bound on the resulting error probability p'. (Hint: Same as for part (a).)

Problem 5 (15 points).

Recall the **bin-packing** problem: Given a set S of $n \geq 1$ objects, where the size s_i of the ith object satisfies $0 < s_i < 1$, we want to determine the minimum number of unit-size bins into which S can be packed. Consider the following variation on **bin-packing**, which we call k-**bin-packing** where $k \geq 1$: The problem now is to determine a largest subset $T \subseteq S$ that can be packed into k unit-size bins (what is "largest" here is the number of items in T and not the sum of their sizes).

(a) Design a $\mathcal{O}(n \log n)$ algorithm for 1-**bin-packing**. (Hint: Sort the input set S in a preprocessing phase.)

(b) Show that 2-**bin-packing** is NP-hard. (Hint: Reduction from **partition**, where every instance is a finite set of positive integers.) Generalize your argument to show that k-**bin-packing** is NP-hard for every $k \geq 3$.

(c) Design a $\mathcal{O}(n \log n)$ approximation algorithm for 2-**bin-packing** — call it 2-BIN-PACK — such that if T^* is an optimal solution and T = 2-BIN-PACK(S), then $|T^*| - |T| \leq 1$. (Hint: Generalize your algorithm in part (a).) Extend your algorithm to k-**bin-packing**, so that if T^* is an optimal solution and $T = k$-BIN-PACK(S), then $|T^*| - |T| \leq k - 1$.

Assaf Kfoury
Created: 95.05.08