BU CLA CS 530: Analysis of Algorithms
8 May 1996
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: Parallel algorithms (10 points)

Consider the problem of finding the maximum value in an array $A[1\,..\,n]$ of $n \geqslant 1$ integers. An efficient sequential algorithm for this problem makes $(n-1)$ comparisons. Design a *work-efficient* EREW algorithm for this problem that runs in $\mathcal{O}(\log n)$ time. If you write the pseudo-code, explain briefly how your algorithm works. Explain why it is work-efficient.

Problem 2: Polynomial-time reductions (10 points)

The **clique** problem is the decision problem defined in [CLR] and in lecture: Given an undirected graph $G = (V, E)$ and a natural number k, decide whether there is a clique of size k in G.

The **maximum-clique** problem is an optimization problem defined by: Given an undirected graph $G = (V, E)$, determine a set of vertices $U \subseteq V$ which form a clique of maximum size in G.

The **maximum-clique-size** problem is another optimization problem defined as follows: Given an undirected graph $G = (V, E)$, determine the size of a maximum clique in G.

Prove the following sequence of polynomial reductions (3 of them):

$$\textbf{clique} \leq_P \textbf{maximum-clique} \leq_P \textbf{maximum-clique-size} \leq_P \textbf{clique}.$$

Problem 3: Matrix multiplication and randomization (15 points)

A, B, and C are $n \times n$ matrices. For simplicity, assume all entries in A, B, and C are integers.

1. Design an efficient algorithm (sequential, non-randomized, exact) to decide whether $AB = C$. We measure the running time by counting the number of individual integer multiplications.

Consider the following Monte Carlo algorithm to decide whether $AB = C$:

```
TEST-PRODUCT(A, B, C, n)
    for i ← 1 to n do X[i] ← RANDOM({−1, 1})
    if ABX = CX
    then return "AB = C"
    else  return "AB ≠ C"
```

The array $X[1\,..\,n]$ is used as a column vector, i.e. as a $n \times 1$ matrix. Instruction RANDOM($\{-1, 1\}$) returns uniformly at random one of two values: -1 or 1.

2. Show that TEST-PRODUCT can be implemented to run in $\mathcal{O}(n^2)$ time.

3. TEST-PRODUCT returns the correct answer if $AB = C$. On the other hand, it may return the wrong answer if $AB \neq C$. Prove that if $AB \neq C$, TEST-PRODUCT nevertheless returns the correct answer with probability $\geqslant 1/2$.
   Hint: If $Y[1\,..\,n]$ is a row in the matrix $AB - C$, show that at least half the ways of adding and subtracting entries in Y return a non-zero value, provided that at least one of the entries is non-zero.

4. Design a Monte Carlo algorithm to decide whether $AB = C$, which runs in $\mathcal{O}(n^2)$ time, with error probability $\leqslant 2^{-30}$.

Problem 4: NP-completeness and approximation algorithms (15 points)

The **bin-packing** optimization problem is this: Given a set S of $n \geqslant 1$ objects, where object i has size x_i (some positive integer) for $i = 1, \ldots, n$, and given a bin size b (a positive integer), we want to determine the minimum number of bins into which we can pack all the objects. Put differently, we want to partition $S = \{x_1, \ldots, x_n\}$ into as few subsets as possible such that the sum of integers in each subset $\leqslant b$.

The decision version of **bin-packing**, call it **bin-packing***, is this: Given S and b as above, and given $k \geqslant 1$, decide whether we can pack all the objects into k or fewer bins.

1. Show that the decision problem **bin-packing*** is NP-complete.
   Hint 1: Polynomially transform an instance (S, t) of **subset-sum** into an instance $(\widetilde{S}, b, 2)$ of **bin-packing***.
   Hint 2: Don't take $\widetilde{S}$ to be S, nor b to be t.

2. Show that the optimization problem **bin-packing** is NP-hard.

The **first-fit** heuristic takes each object in S in turn and places it into the first bin that has room for it. The **first-fit** heuristic is used to define an approximation algorithm for **bin-packing**.

3. Give a simple example (with $n = 3$ or 4) showing that the **first-fit** heuristic is not an exact algorithm for **bin-packing**.

4. Prove a ratio bound of 2 for the **first-fit** heuristic.
   Hint: No two bins can be left half full or less than half full.

Created: 96.05.07