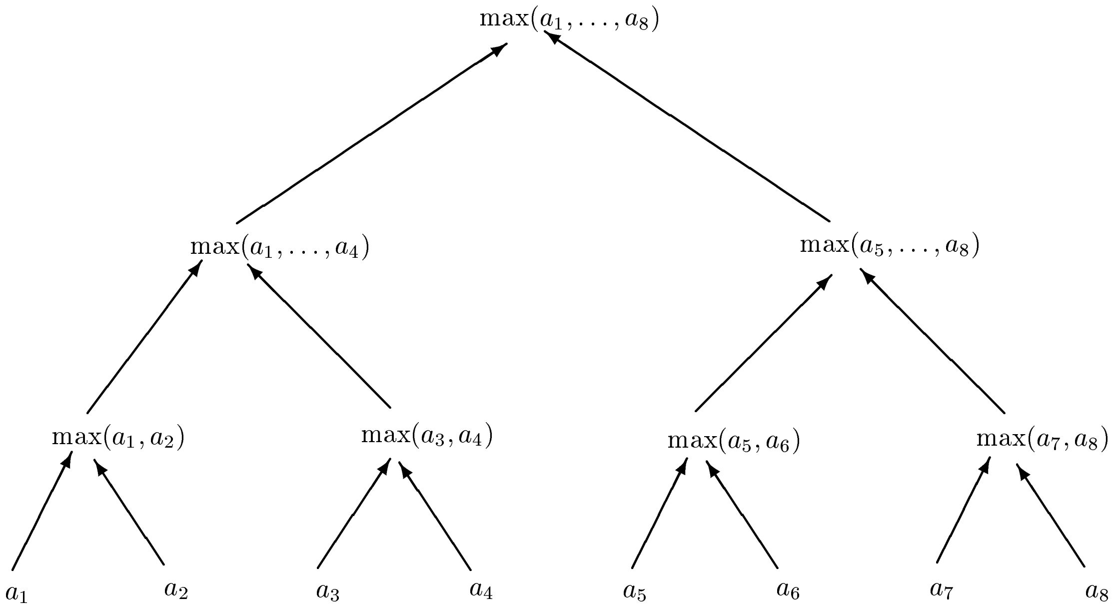
BU CLA CS 530: Analysis of Algorithms
Solutions for End-of-Term Test (8 May 1996)

Problem 1

To find the maximum value is equivalent to a tournament. The players (i.e. the entries in the array $A[1\,..\,n]$) are divided into pairs for the first round of the tournament, then all the winners of the first round are again divided into pairs, and so on until the final round. (At any round, if the number of players is odd, one of the players sits out.) The number of rounds is $\lceil \log n \rceil$. For the case $n = 8$ for example, we can describe the tournament in the form of a binary tree:



We obtain a parallel EREW algorithm by assigning a processor to every game in the tournament. However, each processor must know the two competing numbers, and for this we can put the winner of the game in the larger indexed position of the two players, i.e. if the game is played between a_i and a_j such that $i < j$, then $\max(a_i, a_j)$ is put in position j. Hence, in the first round, processor P_i compares a_{2i-1} and a_{2i}, for $i = 1, \ldots, \lfloor n/2 \rfloor$, and exchanges them if necessary. In the second round, P_i compares a_{4i-2} and a_{4i}, for $i = 1, \ldots, \lfloor n/4 \rfloor$, and so on. As each entry in A is involved in only one game at a time, the EREW model is adequate. The running time is $\mathcal{O}(\log n)$ and the number of prosessors utilized is $\mathcal{O}(n)$ (actually $\lfloor n/2 \rfloor$, so that the work performed is $\mathcal{O}(n \log n)$. Hence, this algorithm is *not* work-efficient. (You get 5 points if you reach this far.)

We can adjust the preceding algorithm to make it work-efficient. For this, we use only $\lceil n/ \log n \rceil$ processors. We divide the entries in A into $\lceil n/ \log n \rceil$ groups, with at most $\log n$ elements in each group and one processor assigned to each group. In a preliminary phase, each processor finds the maximum of its group using a sequential algorithm that makes no more than $\log n$ comparisons.

In a second phase, we use the previously described EREW algorithm to find the maximum of the $\lceil n/\log n \rceil$ maximums. The number of comparisons carried out by this two-phase algorithm does not exceed $\log n + \log(n/\log n) < 2 \cdot \log n$, and the number of processors utilized is $\lceil n/\log n \rceil$. Hence, the work performed is $\mathcal{O}(n)$, and the algorithm is work-efficient. (You get another 5 points for this second part.)

Problem 2

The first reduction (worth 1 point):

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

is trivial. Details omitted. The third reduction (3 points):

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

is also easy, if we are allowed to call an algorithm CLIQUE for **clique** a polynomial number of times (instead of just one call, as in [CLR]). Details omitted. The only non-trivial reduction (6 points) is the second one:

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

For this reduction we write an algorithm MAXIMUM-CLIQUE that calls an algorithm MAXIMUM-CLIQUE-SIZE a polynomial number of times (actually, at most $\mathcal{O}(n^2)$ times, where n is the number of vertices in the input graph G). First, we introduce some convenient notation. If $G = (V, E)$ is an undirected graph and $v \in V$, let $G_v = (V', E')$ be the subgraph of G consisting of v and all its adjacent vertices — formally:

$$V' = \{v\} \cup \{ w \mid (v, w) \in E \} \quad \text{and} \quad E' = E \cap (V' \times V')$$

Denote by $G - v$ the subgraph of G obtained by omitting v and all edges entering or leaving v. Here is the pseudo-code for MAXIMUM-CLIQUE (it is a recursive algorithm):

MAXIMUM-CLIQUE(G)
if G has no vertices
return $\varnothing$
else let v a vertex such that
MAXIMUM-CLIQUE-SIZE(G_v) = MAXIMUM-CLIQUE-SIZE(G)
return $\{v\} \cup$ MAXIMUM-CLIQUE($G_v - v$)

Problem 3

1. (2 points) The matrix product AB can be carried out efficiently using Strassen's algorithm in $\mathcal{O}(n^{2.81})$ time — or using one of the algorithms that asymptotically outperforms Strassen's in $\mathcal{O}(n^{2.376})$ time (mentioned but not analyzed in lecture).

2. (3 points) The run time of TEST-PRODUCT is dominated by the matrix product chain ABX. To minimize the number of individual integer multiplications, we carry out this product as $A(BX)$ which takes $\mathcal{O}(n^2)$ time, instead of $(AB)X$ which takes at least $\Omega(n^{2.376})$ time (using the asymptotically fastest known algorithm for $n \times n$ matrix multiplication).

3. (6 points) $AB \neq C$ iff $AB - C$ is the non-zero $n \times n$ matrix iff at least one row in the matrix $AB - C$ is non-zero. Consider one such non-zero row $Y[1..n]$ in $AB - C$, i.e. if

$$Z = \{z_1, \ldots, z_k\} = \{ Y[i] \mid i \in \{1, \ldots, n\} \text{ and } Y[i] \neq 0 \}$$

then $Z \neq \varnothing$. The result of carrying out the matrix product YX (Y as a $1 \times n$ matrix and X as a $n \times 1$ matrix) is one way of adding and subtracting entries in Y. All ways of adding and subtracting entries in $Y[1..n]$ that return non-zero values are exactly all ways of adding and subtracting integers in Z. By induction on $k \geqslant 1$, prove that at least half of these ways return a non-zero value. (We omit details of the straightforward induction.)

4. (4 points) Iterate TEST-PRODUCT 30 times or less, until it returns "$AB \neq C''$ for the first time (in which case the answer is correct with error probability 0), or until it returns "$AB = C$" consecutively 30 times (in which case the answer is correct with error probability $\leqslant (1/2)^{30}$).

Problem 4

1. (6 points) First, show that **bin-packing*** is in NP, i.e. there is a polynomial-time algorithm to verify whether a proposed solution is one. Second, show that **bin-packing*** is NP-hard by a reduction from **subset-sum**. Given $S = \{x_1, \ldots, x_n\}$ and t, define $\widetilde{S} = S \cup \{b - t, b - (s - t)\}$ where $s = x_1 + \cdots + x_n$ and b is a sufficiently large numbre (actually any number $> s$ is appropriate). If we try to pack all the objects of $\widetilde{S}$ into 2 bins, then $(b - t)$ and $(b - (s - t))$ are too large to fit into the same bin of size b. Hence, if we succeed at all, then $(b - t)$ and $(b - (s - t))$ must be placed in separate bins. And if we succeed, then the bin containing $(b - t)$ will also include a subset of $S = \{x_1, \ldots, x_n\}$ whose entries add up to t.

2. (2 points) Straightforward reduction. (Details omitted.)

3. (1 point) Take $x_1 = 5$, $x_2 = 4$, $x_3 = 6$, $x_4 = 5$, and $b = 10$. The **first-fit** heuristic places x_1 and x_2 in bin 1, x_3 in bin 2, and x_4 in bin 3 (for a total of 3 bins). An optimal solution places x_1 and x_4 in bin 1, and x_2 and x_3 in bin 2 (for a total of 2 bins).

4. (6 points) The **first-fit** heuristic does not leave two bins half full or less than half full, because all the items in two such bins can fit together in a single bin. Hence, the **first-fit** heuristic leaves at most one bin that is half full or less than half full. Let k be the number of bins used in a solution produced by the **first-fit** heuristic. Suppose the first $(k - 1)$ bins contain

$x_1, \ldots, x_\ell$, the last bin contains $x_{\ell+1}, \ldots, x_n$, and only the last bin is half full or less than half full. We then have:

$$(k-1) \cdot b < 2 \cdot (x_1 + \cdots + x_\ell) < 2 \cdot (x_1 + \cdots + x_n)$$

so that $k < 1 + 2 \cdot (x_1 + \cdots + x_n)/b$ or $k \leqslant 2 \cdot (x_1 + \cdots + x_n)/b$. On the other hand, if $\widetilde{k}$ is the number of bins used in an optimal solution, then $\widetilde{k} \cdot b \geqslant x_1 + \cdots + x_n$. Hence, $\widetilde{k} \geqslant (x_1 + \cdots + x_n)/b$. Hence, $(k/\widetilde{k}) \leqslant 2$.

Assaf Kfoury
Created: 96.05.08