CSE 6512 Randomization in Computing. Fall 2011

Exam #1 Solutions

---

1. Let $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_n\}$. Construct two polynomials $f(x) = \Pi_{i=1}^{n}(x - a_i)$ and $g(x) = \Pi_{i=1}^{n}(x - b_i)$. The problem of checking if A and B are identical can be reduced to the problem of checking if $f(x)$ and $g(x)$ are identical. We can use fingerprinting to do this in $O(n)$ time as follows. Let $\mathcal{S}$ be the set of integers in the range $[1, n^{\alpha+1}]$. Pick a random integer r from $\mathcal{S}$, evaluate $f(r)$ and $g(r)$, and check if $f(r) = g(r)$. If $f(r) = g(r)$, then output: "A and B are identical"; else output: "A and B are not identical".

   Clearly, if $f(r) \neq g(r)$, then A and B are not identical. If A and B are identical, then the above algorithm will never give an incorrect answer. If A and B are not identical, what is the probability that $f(r) = g(r)$? Note that the polynomial $h(x) = f(x) - g(x)$ has at most n distinct zeros. Therefore, $Prob.[f(r) = g(r)] \leq \frac{n}{n^{\alpha+1}} = n^{-\alpha}$.

   Note: If $f(r)$ and $g(r)$ are very large numbers, we can use a random prime p and check if $f(r) \bmod p = g(r) \bmod p$ instead of checking if $f(r) = g(r)$. If p is chosen from a large enough range, the overall probability of an incorrect answer can be ensured to be $\leq n^{-\alpha}$.

2. Note that the diameter of G is $2\sqrt{n} - 1$. As a result, the resistance of G, $R(G)$ is $\leq 2\sqrt{n} - 1$. The number of edges in G is $O(n\sqrt{n})$. Therefore, $C(G) = O(|E|\ R(G) \log n) = O(n^2 \log n)$.

3. Consider a random assignment to the n variables, where each variable is assigned the value T with probability 1/2 and it is assigned the value F with the same probability. Consider any clause C_i and let the number of variables in this clause be $k \geq 1$. Probability that C_i is not satisfied is $\leq 2^{-k}$. Thus, probability that C_i is satisfied is $\geq 1/2$. As a result, the expected value of the total weight of all the satisfied clauses is $\sum_{i=1}^{m} w_i \times$ Prob.[C_i is satisfied] $\geq \frac{\sum_{i=1}^{m} w_i}{2}$. This implies that there exists an assignment under which the sum of weights of all the satisfied clauses is $\geq \frac{\sum_{i=1}^{m} w_i}{2}$.

4. Let $X = k_1, k_2, \ldots, k_n$. Assume without loss of generality that the keys are distinct. Note that the right neighbor of any input key k_i is nothing but the minimum among all the input keys that are greater than k_i. Key k_i is assigned a group G_i of n processors, $1 \leq i \leq n$. The processors associated with k_i use an array $A_i[1:n]$. This array is initialized with all ∞'s. Processor j of group G_i writes k_j in $A_i[j]$ if $k_j > k_i$. After this write step that takes one parallel step, processors in G_i find the minimum of $A_i[1], A_i[2], \ldots, A_i[n]$ in $\widetilde{O}(1)$ time. This minimum is the right neighbor of k_i.

5. We will show that we can stably sort n integers in the range $[1, \sqrt{n}]$ in $O(\sqrt{n})$ time using $\sqrt{n}$ CREW PRAM processors. Using the idea of radix sorting it will follow that we can sort n integers in the range $[1, n^c]$ (for any constant c) in $O(\sqrt{n})$ time using $\sqrt{n}$ processors.

   Let $X = k_1, k_1, \ldots, k_n$ be the input sequence. Assign $\sqrt{n}$ keys per processor. In particular, the first processor gets the keys $k_1, k_2, \ldots, k_{\sqrt{n}}$; the second processor gets the keys $k_{\sqrt{n}+1}, k_{\sqrt{n}+2}, \ldots, k_{2\sqrt{n}}$; and so on.

(a) Each processor sorts its keys using bucket sorting. This takes $O(\sqrt{n})$ time. Let $N_{i,j}$ be the number of keys of value j that processor i has, for $1 \leq i, j \leq \sqrt{n}$.

(b) All the $\sqrt{n}$ processors perform a prefix sums computation on $N_{1,1}, N_{2,1}, \ldots, N_{\sqrt{n},1}$, $N_{1,2}, N_{2,2}, \ldots, N_{\sqrt{n},2}$, $\cdots$, $N_{1,\sqrt{n}}, N_{2,\sqrt{n}}, \ldots, N_{\sqrt{n},\sqrt{n}}$.

(c) Each processor now uses these prefix sums values to output its keys in the sorted order.

Since each of the above three steps takes $O(\sqrt{n})$ time, the run time of the algorithm is $O(\sqrt{n})$.

6. Assume that A and B are in common memory in successive cells. In particular, assume that A is in $M[1:n]$ and B is in $M[n+1:m+n]$.

(a) Sort B, i.e., sort $M[n+1:n+m]$. This can be done in $\widetilde{O}(\log m)$ time using m arbitrary CRCW PRAM processors.

(b) Assign one processor per element of A. Processor i performs a binary search in $B[n+1:n+m]$ to check if $M[i]$ is in B, for $1 \leq i \leq n$. This binary search takes $O(\log m)$ time.

(c) In this step, we'll use an array $Q[1:2m]$. Each element of A that is also in B will be placed in a unique cell of Q. Each element of A is assigned one processor. If an element of A is in $A \cap B$, the corresponding processor will try to place the element in Q. If an element of A is not in $A \cap B$, the corresponding processor goes to sleep. If a processor π has an element that has to be placed in Q, π proceeds in rounds. It takes as many rounds as needed to successfully place its key.

In a round, π picks a random cell in Q; If this cell is occupied, it waits for the next round; If this cell is empty, it tries to write its key in the cell; Processor π reads from this cell to check if its key is there; If so, the processor goes to sleep; If not, it moves to the next round.

Probability that π succeeds in any round is $\geq 1/2$. Thus the number of rounds needed to place π'key successfully in Q is $\widetilde{O}(\log m)$, for any processor π.

(d) Use a prefix computation to compress the array $Q[1:2m]$ (and get rid of the empty cells). This can be done in $O(\log m)$ time using $\frac{2m}{\log m} \leq n$ processors.

The compressed array Q is $A \cap B$.

We could do steps (c) and (d) in a different way as follows. We use an array $Q[1:m]$ initialized to all zeros. Each element of A is assigned a processor. Processor i goes to sleep if k_i is not in $A \cap B$, $1 \leq i \leq n$. Otherwise, processor i writes a 1 in $Q[j]$ if $M[i] = M[n+j]$. After this parallel write step, we assign one processor per element of B. These processors empty the cells of B that are not in $A \cap B$. A prefix sums computation is done on Q in $O(\log m)$ time using $\frac{m}{\log m}$ processors. These prefix sums are used to write the elements of $A \cap B$ in successive cells in common memory.