CSE 3500 Algorithms and Complexity

Fall 2016 Exam III – Solutions

1. **Note** :- Here also we can use either **BFS** or **DFS**.

   (a) The **BFS** solution (Simpler):-

   Since all the edge weights are the same, we do a simple BFS at every node, storing the distance from the source (in multiples of w). if one encounters a node again - ignore it (because the distance would be greater than or equal to older distance).

   (b) The **DFS** solution

   Here, it is possible that we find a better path (shorter path), at a later stage in the traversal. Thus, modify the DFS such that initially all distances are ∞. Once a node is traversed, its distance from the source is recorded in multiples of w. If the node is re-encountered (it is not avoided - as in the case of BFS algorithm), rather the new value of distance_from_source and the old value are compared and the minimum is retained.

2. Here is an algorithm that checks if X is sorted in nondecreasing order:

   Processor 1 writes 1 in *Result*;
   for $i = 1$ **to** $(n-1)$ **in parallel do**
       Processor i tries to write a zero in *Result* if $k_i > k_{i+1}$;

   The correctness of the algorithm is clear and the algorithm takes $O(1)$ time.

3. We can use the prefix computation algorithm to solve this problem. We use one prefix computation for each possible value that the keys can take. Let $n_0 = 0$ and $n_i = |\{q \in X : q = i\}|$, for $i = 1, 2, \ldots, 10$. More details of the algorithm follow.

   1) **for** $i = 1$ **to** 10 **do**
   2)   Initialize $A[1:n]$ to all zeros;
   3)   Set $A[j] = 1$ if $k_j = i$, for $1 \leq j \leq n$;
   4)   Perform a prefix sums computation on $A[1], A[2], \ldots, A[n]$
   5)   to get $B[1], B[2], \ldots, B[n]$;
   6)   If $k_j = i$ then write k_j in cell $n_{i-1} + B[j]$, for $1 \leq j \leq n$;
   7) $n_i = B[n]$;

   Run time analysis: We first write all the keys that have a value 1 in successive memory cells, starting from cell 1; Followed by this we write all the keys that have a value 2, and so on. The prefix sums computation done in step 4 gives us unique addresses that can be used to write keys with a value i in successive memory cells.

The **for** loop of step 1 is executed 10 times. Step 2 can be done in one unit of time using n processors. Using the slow-down lemma, this can also be done in $O(\log n)$ time using $\frac{n}{\log n}$ processors. Step 3 is similar to Step 2 and hence can be done in $O(\log n)$ time using $\frac{n}{\log n}$ processors. Step 4 takes $O(\log n)$ time using $\frac{n}{\log n}$ processors as was proven in class. Step 6 is similar to Steps 2 and 3 and hence can be completed in $O(\log n)$ time using $\frac{n}{\log n}$ processors. Step 7 takes one unit of time using 1 processor.

Thus the entire algorithm runs in $O(\log n)$ time using $\frac{n}{\log n}$ CREW PRAM processors.

4. To solve this problem, we can use the fact that we can find the maximum of n elements in $O(1)$ time using n^2 common CRCW PRAM processors. Here is a recursive algorithm:

   0) Partition X into $\sqrt{n}$ groups $X_1, X_2, \ldots, X_{\sqrt{n}}$ such that each group has $\sqrt{n}$ keys;
   1) **for** $1 \leq i \leq \sqrt{n}$ **in parallel do**
   2) Find the maximum M_i of X_i using $\sqrt{n}$ processors;
   3) Find and output the maximum M of $M_1, M_2, \ldots, M_{\sqrt{n}}$ using n procesors;

   Run time analysis: Let $T(n)$ be the run time of this algorithm on any input of size n using n processors. Step 2 takes $T\left(\sqrt{n}\right)$ time. Step 3 takes $O(1)$ time (since we only have $\sqrt{n}$ elements and n processors). Thus the recurrence relation for $T(n)$ is:

$$T(n) = T\left(\sqrt{n}\right) + O(1).$$

   Using repeated substitutions, we can solve for $T(n)$ to get: $T(n) = O(\log \log n)$.

5. We can solve this problem in polynomial time as follows. Let $x_1, x_2, \ldots, x_n$ be the variables involved in F. Let $F_{x_i=1}$ stand for the Boolean formula (on the variables $x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$) obtained from F by substituting $x_i = 1$. Similarly, let $F_{x_i=0}$ stand for the Boolean formula (on the variables $x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$) obtained from F by substituting $x_i = 0$.

   if !SATALG(F) **then** output "F is not satisfiable" and quit;
   for $i = 1$ **to** n **do**
       if SATALG($F_{x_i=1}$) **then**
           Output "$x_i = 1$"; $F = F_{x_i=1}$;
       else
           Output "$x_i = 0$"; $F = F_{x_i=0}$;

   Run time: Note that SATALG is called $(n+1)$ times in the above algorithm. If the run time of SATALG is $p(n)$ for some polynomial $p(.)$, then the run time of the above algorithm is $(n+1)p(n)$ which is also a polynomial in n.