ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute of Theoretical Computer Science
Mohsen Ghaffari, Angelika Steger, David Steurer, Emo Welzl, Peter Widmayer

Algorithms, Probability, and Computing — Solutions KW50 — HS18

Solution 1

Assume that we break all elements into $n/2$ disjoint pairs of consecutive elements. Label these pairs from 1 to $n/2$ arbitrarily. Define the random variable X_i for $1 \le i \le n/2$ to be 1 if the first element in the i-th pair is head and the second element, its successor, is tail (this immediately implies that the first element is in I), and 0 otherwise. Furthermore, let $X := \sum_{i=1}^{n/2} X_i$. By definition, $X \le |I|$. Thus, it suffices to prove that $X \ge n/16$ with probability $1 - \mathcal{O}(\frac{1}{n^{10}})$. We know that $\mathbb{E}[X_i] = 1/4$ and then $\mathbb{E}[X] = n/8$ by linearity of expectation. Now, by applying the Chernoff bound for $\delta = 1/2$, we get

$$\Pr[X \le (1 - \frac{1}{2})\mathbb{E}[X]] \le e^{-\frac{(1/2)^2 n/8}{2}} = \mathcal{O}(\frac{1}{n^{10}}).$$

Finally, note that $(1 - \frac{1}{2})\mathbb{E}[X] = n/16$.

Solution 2

As our input we have a graph on the node set $\{1, \cdots, n\}$ whose edge set is given in the form of an $n \times n$ binary adjacency matrix, where the entry at location (i, j) is 1 if the i-th and j-th nodes are adjacent, and 0 otherwise. We want to devise a parallel algorithm with $\mathcal{O}(\log n)$ depth and $\mathcal{O}(n^2)$ work that transforms this adjacency matrix to linked lists. In the linked lists, for each node $v \in V$, the nodes adjacent to v are given in a linked list $L[v] =< u_0, u_2, \cdots, u_{d(v)-1} >$, where $d(v)$ is the degree of the node v.

Since we can handle each row independently, it suffices to show that for a row we can construct the adjacency list for the corresponding node in $\mathcal{O}(\log n)$ depth and $\mathcal{O}(n)$ work. If we have two linked lists L and L' and their starting and ending, we can concatenate L and L' in c time-steps for some constant $c > 0$. Simply, set the successor of the last element of L to be the first element of L'. Now, we have a linked list L'', where the starting is the same as the starting of L and the ending is identical to the ending of L'. To each entry a_{ij} in row a_i for $1 \le j \le n$, we initially assign a linked list of one element which has the value j if $a_{ij} = 1$ and an empty linked list if $a_{ij} = 0$. Now, we divide these linked lists into $n/2$ consecutive disjoint pairs and concatenate each pair. In the next round, we divide the $n/2$ newly created linked lists into $n/4$ consecutive disjoint pairs and concatenate each pair. After $\log n$ steps we will be left with a linked list $L[i]$ which includes all nodes adjacent to node i. Thus, the depth is $\mathcal{O}(\log n)$. In the t-th time-step, the work done is $c \cdot \frac{n}{2^t}$ since we have to concatenate $n/2^t$ pairs and for each we need c time-steps. Therefore, the overall work for a row is

$$\sum_{t=1}^{\log n} c\frac{n}{2^t} = \mathcal{O}(n).$$

Solution 3

Consider a Depth First Search traversal of the nodes (according to the adjacency lists, which is the same as how the Eulerian path is defined in the lecture notes). Our objective is to compute a post-oder numbering $post : V \to \{0, \dots, n-1\}$ of the nodes. That is, in this numbering, for each node v, first a post-order numbering of the subtree rooted in the first child of v appears, then a post-order numbering of the subtree rooted the second child of v, an so on, and finally v appears.

Using the Eulerian tour technique, we can solve the problem, as follows: After having identified the parents, we now define a new weight for the arcs. We set $w(< parent(v), v >) = 0$ and $w(< v, parent(v) >) = 1$. Notice that the former are forward arcs in the DFS and the latter are backward arcs. Then, we compute all the prefix sums of these weights, on the linked list provided by our Eulerian path (i.e., maintained by the successor pointers). Hence, each arc knows the number of backward arcs before it (and including itself), in the Eulerian path. Set $post(r) = n - 1$ for the root node r. For each node $v \neq r$, set $post(v)$ to be the prefix sum on the arc $< v, parent(v) >$ minus one, which is equivalent to the total number of backward arcs before this arc. This gives exactly our desired post-order numbering because each backward edge corresponds to a node which appears before node v in the post-order numbering.

Based on the lecture notes (similar to the computation of pre-order numbering), the aforementioned algorithm has $\mathcal{O}(\log n)$ depth and $\mathcal{O}(n)$ work. The only difference is that at the end we subtract the prefix sum on the arc $< v, parent(v) >$ by one for each node v in parallel to obtain $post(v)$, which is doable in $\mathcal{O}(1)$ depth and $\mathcal{O}(n)$ work.

Solution 4

We want the number of descendants $des(v)$ for each node v, which is the total number of nodes in the subtree rooted at node v.

We use the Eulerian tour technique again. After having identified the parents, we now define the weights for the arcs similar to pre-order numbering. We set $w(< parent(v), v >) = 1$ and $w(< v, parent(v) >) = 0$. In other words, we set the weight of all forward arcs in the DFS to 1 and the weight of all backward edges to 0. Then, we compute all the prefix sums of these weights, on the linked list provided by our Eulerian path. Hence, each arc knows the number of forward arcs before it (and including itself), in the Eulerian path. Set $des(r) = n$ for the root node r. For each node $v \neq r$, set $des(v)$ to be the prefix sum on the arc $< v, parent(v) >$ minus the prefix sum on the arc $< parent(v), v >$, plus one. The prefix sum on an arc is equivalent to the total number of forward arcs before this arc (including itself). Furthermore, each forward arc corresponds to visiting an unvisited node in the DFS. Therefore, the prefix sum on the arc $< v, parent(v) >$ minus the prefix sum on the arc $< parent(v), v >$ is equal to the total number of forward arcs in the subtree rooted at v. We add one to this value since it does not include the forward arc corresponding to node v itself.

The depth and the work required by this algorithm is similar to the one for computing the pre-order numbering, except at the end we compute $des(v)$ for each node v in parallel, which can be done in depth $\mathcal{O}(1)$ and work $\mathcal{O}(n)$. Therefore, this algorithm performs in depth $\mathcal{O}(\log n)$ and work $\mathcal{O}(n)$.

Solution 5

Each leaf node u in the tree $T = (V, E)$ is given a number $b(u)$. We want to devise a parallel algorithm with $O(\log n)$ depth and $O(n)$ total computation that computes for each node v the value $sl(v)$, the summations of the $b(u)$ values over all the leaves u that are descendants of v.

We use the Eulerian tour technique again. After having identified the parents, we now define new weights for the arcs. We set $w(< parent(v), v >) = w(< v, parent(v) >) = 0$ if v is not a leaf and $w(< v, parent(v) >) = b(v)$ if v is a leaf. In other words, all arcs have weight zero, except the backward edges from the leaves. Now, we compute all the prefix sums of these weights, on the linked list provided by our Eulerian path. For each node v, the difference between the prefix sum on the arc $< v, parent(v) >$ and $< parent(v), v >$ is equivalent to the sum of all arcs in the subtree rooted in v, plus $w(< v, parent(v) >)$. This is clearly the same as the summations of the $b(u)$ values over all the leaves u that are descendants of v (including itself). Thus, we simply set $sl(v)$ to be the prefix sum on the arc $< v, parent(v) >$ minus the prefix sum on the arc $< parent(v), v >$.

Again, the depth and total computations of the algorithm is asymptotically equivalent to the algorithm for computing the pre-order numbering, i.e., $\mathcal{O}(\log n)$ depth and $\mathcal{O}(n)$ work. The only difference is that at the end we subtract two prefix sum to compute $sl(v)$ for each node v in parallel, which does not change the depth and total work value asymptotically.