ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute of Theoretical Computer Science
Rasmus Kyng, Bernd Gärtner, David Steurer, Vera Traub

Algorithms, Probability, and Computing **Exercises KW51** **HS25**

General rules for solving exercises

- When handing in your solutions, please write your exercise group on the front sheet:

  Group A: Wed 14–16 CAB G 56

  Group B: Wed 14–16 CAB G 57

  Group C: Wed 16–18 CAB G 56

  Group D: Wed 16–18 CAB G 57

- This is a theory course, which means: if an exercise does not explicitly say "you do not need to prove your answer", then a formal proof is **always** required.

The following exercises will be discussed in the exercise classes on December 17, 2024. You can hand in your solution via Moodle, no later than 2 pm at December 16.

Exercise 1

(Exercise 6.7 from the lecture notes)

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

Exercise 2

(Exercise 6.8 from the lecture notes)

Suppose that we are given an array of length n and each element in it is tagged with a number in $\{1, 2, \ldots, \sqrt{n}\}$, which indicates the index of its subproblem. Devise an algorithm that in $O(\log n)$ depth and using $O(n \log n)$ work, creates one array A_i for each of the subproblems $i \in \{1, \ldots, \sqrt{n}\}$, holding all the elements tagged with number i. Each element should be in exactly one array and the summation of the lengths of the arrays should be n.

Hint: First, think about creating linked lists instead of arrays. In particular, imagine a balanced binary tree on the n elements, where each node keeps a number of linked lists, one for each of the subproblems present in its descendants. Then, when you pass up the linked lists, from the children of one node v to the node v itself, you can merge these linked lists in $O(1)$ depth.

Exercise 3

(Exercise 6.15 from the lecture notes)

Explain how we can identify the minimum neighbors as desired in Section 6.5.1 of the lecture notes, simultaneously for all the fragments, using $O(\log n)$ depth and $O(m + n \log n)$ work. You can do this in three steps, as outlined below:

(A) First, create an array for the edges incident on the nodes of each fragment, in a manner that the sum of the lengths of these arrays over all fragments is at most $2m$. This should be doable using $O(\log n)$ depth and $O(m + n \log n)$ work (how? hint: think about the idea in Exercise 1).

(B) Then, explain how to discard the entries of the array of each fragment that do not connect to other fragments. Moreover, for entries that are not discarded, add to them the identifier of the root node of the other endpoint's fragment. These two operations should be doable using $O(1)$ depth and $O(m)$ work.

(C) Finally, explain how we can compute the minimum among the entries that are not discarded, simultaneously for all fragments, using $O(\log n)$ depth and $O(m)$ work.

Exercise 4

(Exercise 6.16 from the lecture notes)

Prove that at the end of the algorithm presented in Section 6.5.1, each component will have its identifier equal to the minimum identifier among the nodes of that component.

Exercise 5

(Exercise 6.9 from the lecture notes)

Devise a randomized parallel algorithm that given an array of length n, finds the k^{th} smallest elements of this array, with probability at least $1 - O(1/n^5)$. Ideally, your algorithm should have $O(\log n)$ depth and $O(n)$ work.

Hint: You might apply the idea of picking $\sqrt{n}$ pivots at random to decrease the number of elements to $O(\frac{n}{\log n})$. Now, you can simply sort the remaining elements in $O(\log n)$ depth and $O(n)$ work.