Assignment

1. RULES

Purpose of this list: this list of tasks is addressed to people that want to get ECTS points for my lecture.

1.1. General Rules

- there are 5 tasks
- pick 3 tasks to solve and write down your solutions:
  — either on paper, and send the scans
  — or in latex (or whatever that can be made into pdf), and send the pdf
  on my address (knowicki@cs.uni.wroc.pl) before the 14th of May, 23:59 AoE

1.2. Grading rules

Grading a single task: for each task you can get:

- α points for a partial solution / solution with significant gaps
- β points for solutions with minor errors / gaps
- γ points for good solutions

where (α, β, γ) is the triplet after the task number.

Grading scale:

- 6 to 8 $\rightarrow$ 3.0
- 9 to 11 $\rightarrow$ 3.5
- 12 to 14 $\rightarrow$ 4.0
- 15 to 17 $\rightarrow$ 4.5
- 18+ $\rightarrow$ 5.0

1.3. Expected level of detail

Whenever I ask you to provide an algorithm, I don't expect all the low level details. Assume that some basic algorithms are given, i.e.

- sorting can be done in $\mathcal{O}(1)$ rounds,
- vertex centric computation (i.e. computation performed by vertices, which communicate between each other) can be easily simulated in MPC(n), and partially simulated in MPC(n^α) (where a vertex can broadcast a message to all neighbours, and aggregate messages from all neighbours using some cumulative and commutative function)
- you can also use any other algorithm from that can be found in *Sorting, Searching, and Simulation in the MapReduce Framework* by Michael T. Goodrich, Nodari Sitchinava, Qin Zhang [ISAAC 2011]

Instead of going into technical details of implementation, focus on the graph properties and reductions that make your solution work, i.e. I expect that you will prove that the algorithm is correct and that it has claimed round / memory complexity, however it is not necessary to go into all the implementation details.

Denotations and usual assumptions: we work with unweighted, undirected, n-vertex, m-edge input graphs. Whenever we work with MPC(n^α), you may assume that $\alpha = 1/2$ (if it simplifies your algorithms).

Application of the Spanning Forest Algorithm

In the two following tasks you can use the $\mathcal{O}(1)$ round deterministic algorithm for Connected Components.
Task 1; (2,5,6): Propose a deterministic algorithm that in $\mathcal{O}(1)$ rounds of MPC(n) decides whether the graph is bipartite.
Task 2; (4,10,12): Propose a deterministic algorithm that for a given integer k, and input graph G in $\mathcal{O}(k)$ rounds of MPC(kn) verifies whether G is k-edge connected.

Algorithms for trees in MPC(n^α)

Task 3; (2,5,6): Propose an $\mathcal{O}(1)$ round randomized algorithm for MPC(n^α) that partitions a set of edges E of a tree $T = (V, E)$ into two disjoint sets E_1, E_2 such that maximum diameter of a connected component of (V, E_1) and (V, E_2) is $\mathcal{O}(\log n)$, with high probability. You can use Chernoff / union bound to show high probability of success.
Task 4; (4,10,12): You are given a MPC(n^α) algorithm that finds connected components (i.e. assigns to all members of the same connected components the same identifier; the assignment is such that no two vertices from different components get the same identifier) of the input graph G in $f(D)$ rounds, where D is the maximum diameter of a connected component in G.

Propose $\mathcal{O}(f(\log n))$ round randomized algorithm for MPC(n^α) that finds a maximal independent set of a given tree. You can use an algorithm from Task 3 as a building block, even if you didn't solve it. Hint: some solutions to Task 1 might be useful here.

Random greedy algorithms

Task 5; (3,8,9): Show that the MPC(n) algorithm for the MIS problem presented on day 2 can be used to find a maximal matching in $\mathcal{O}(1/\varepsilon)$ rounds of MPC($n^{1+\varepsilon}$).