University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
S. Faour, M. Fuchs, A. Malyusz

Algorithm Theory
Sample Solution Exercise Sheet 15

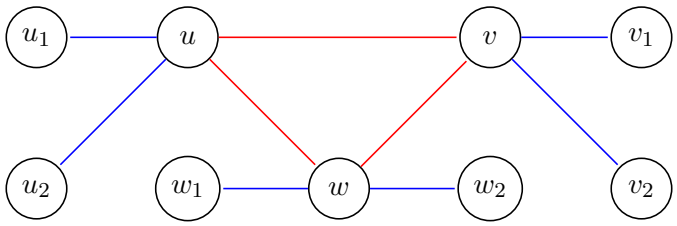
Due: Friday, 16th of February 2024, 10:00 am

Exercise 1: Maximum Sparse Subgraph *(8 Bonus Points)*

Let $G = (V, E)$ be a simple graph, we define a 2-sparse edge set $F \subseteq E$ to be the subset of edges such that the subgraph $H = (V, F)$ induced by the edges has maximum degree at most 2. We call F a maximum 2-sparse edge set if there is no 2-sparse edge set F' of larger size.
Assume that we are given the edges in online fashion. Observe the greedy algorithm that always adds the next incoming edge $e = \{u, v\}$ to F as long as this choice does not increase the degree of u or v to more than 2. Show that this algorithm has competitive ratio 2. Further, show that this is tight, i.e., there is a graph such that if the edges come in some specific order, the algorithm has exactly half as many edges as to optimal solution.

Sample Solution

Let F_{ALG} be the set of edges constructed by our online algorithm and let F_{OPT} be the set of edges in the optimal solution. We show that there is a mapping function $f : F_{OPT} \to F_{ALG}$ with the property that for all $e \in F_{ALG}$ we have $|f^{-1}(e)| \leq 2$. Note that this property directly implies that $|F_{OPT}| \leq 2 \cdot |F_{ALG}|$. The construct the mapping as follows:
First we map each edge $e \in F_{OPT} \cap F_{ALG}$ to itself i.e., $f(e) = e$. Now consider the remaining edges in $F_{OPT} \setminus F_{ALG}$. Clearly, for each such edge $e = \{u, v\}$ we have that u is incident to 2 edges from F_{ALG} or v is (or both), cause otherwise ALG would have added e to F_{ALG}. W.l.o.g., let u be an endpoint of e that has degree 2 in the subgraph (V, F_{ALG}) and let us call these edges e_1 and e_2. We set $f(e) = e_1$ if we do not already map another edge $e' \in F_{OPT}$ that also has u as endpoint to e_1, else we set $f(e) = e_2$. Note that since at most 2 edges from F_{OPT} have u as endpoint, it is always possible to map e either to e_1 or to e_2. Thus, by that construction we can map all edges from F_{OPT} to some edge in F_{ALG}. It remains to show that for all $e' \in F_{ALG}$ we have $|f^{-1}(e')| \leq 2$. By our construction it is clear that for each endpoint of e' at most one edge from F_{OPT} can be mapped to e'. Hence, the statement is true because each edge has at most 2 endpoints.
To see why this is a tight result, take a look at the following graph. An optimal solution will pick the blue edges, while our greedy algorithm may decide on the set of red edges if they come in a "bad" order.



Exercise 2: Parallel Parentheses *(12 Bonus Points)*

You are given a string S consisting of opening and closing parentheses. The expression S is supposed to be a string with **balanced parentheses** if each opening parenthesis has a corresponding closing one and the pairs of parentheses are properly nested. For example, consider the expressions (()()) as a correctly balanced string of parentheses and ())(as an incorrect one.

a) First, provide a (sequential) linear-time algorithm to determine whether S is balanced. *(4 Points)*

b) Now devise a parallel algorithm to check if the string S is balanced. Like in the lecture assume that we are given p processors. What is the asymptotic runtime T_p? *(8 Points)*

Sample Solution

a) We use a stack, that is empty in the beginning and process the string S from left to right. Whenever there is an opening parenthesis, we push it on that stack as a signal that a closing symbol needs to appear later. If we meet a closing parenthesis pop the latest symbol from the stack. As long as it is possible to pop the stack to match every closing symbol, the parentheses remain balanced. If at any time there is no opening symbol on the stack to match a closing symbol, the string is not balanced properly. At the end of the string, when all symbols have been processed successfully, the stack is empty if and only if the string is balanced. Since a single pass over the string S is sufficient, the runtime is as required.

   Note that there are easier algorithms for that problem, but this version can easily be generalized if there are different kinds of brackets like $(,), \{, \}$ and $[,]$.

b) We are given p processors and a string S of size $|S| = n$. We substitute the opening parenthesis with 1 and the closing parenthesis with -1 (this can be done in time $O(n/p)$ by splitting the string into p equally sized substrings). Afterwards, we can compute the **prefix sum** from the lecture on this substituted string (where addition is the operation of the prefix-sum). Our original string S is properly balanced if each prefix-sum value

   - is greater or equal to zero and
   - the last prefix sum equals zero.

   By the first condition we make sure that there are always more or equally many opening parentheses and by the second condition we make sure that in S these numbers are equal. These conditions can easily be checked in time $O(n/p)$. As we know from the lecture that prefix sums can be computed in time $O(n/p+\log n)$ it takes time $T_p = O(n/p+\log n)$ to check if S is balanced using p processors.