# NCM IST, Mathematics for Computer Science
# Problems on bipartite graphs, matchings, BFS/DFS

22 June, 2018

1. Consider the set cover problem defined as follows. Let $U$ be a set of elements and $\mathcal{F}$ be a collection of subsets of $U$ such that the union of the sets in $\mathcal{F}$ is $U$. The goal is to select a minimum sized sub-collection of $\mathcal{F}$ whose union is $U$. Show that the vertex cover problem transforms (reduces) to the set cover problem. Does it prove that the set cover problem is NP-Complete?

2. Consider the following greedy algorithm for computing a matching in a graph $G = (V, E)$. Let $M = \emptyset$. For every edge $e \in E$, if $M \cup \{e\}$ is a matching, then $M = M \cup \{e\}$. Show that $M$ is a *maximal* matching. Construct an example where $M$ is *not* a maximum sized matching. Argue that $|M|$ is at least $\frac{1}{2}$ the size of a maximum matching.

3. Consider the following algorithm for the vertex cover problem in a graph $G = (V, E)$. Construct the DFS tree $T$ of $G$. Let $S$ denote the non-leaf vertices of $T$. Show that $S$ is a vertex cover of $G$. Show that $|S|$ is at most two times the mininum sized vertex cover. **Hint:** Argue that $G$ has a matching of size $|S|/2$.

4. Consider the problem of checking if a number $n$ is prime. i.e. the language we are interested is the set of all numbers $n$ such that $n$ is prime. What is the size of the input to an algorithm which is deciding this language? Think of your favourite algorithm to check if a number is prime and analyze its running time. Is your algorithm a polynomial time algorithm?

5. Suppose you have an polynomial time algorithm to decide the following language $\mathcal{L} = \{< G, k, m > | G \textit{ is a graph with exactly } m \textit{ distinct vertex covers of size exactly } k\}$. How can you use this to solve the decision version of the vertex cover problem, for a given graph is there a vertex cover of size less than or equal to d? Is $\mathcal{L}$ in NP?

6. In the parallel-machine-scheduling problem, we are given $n$ jobs, $J_1, J_2, \cdots, J_n$ , where each job $J_k$ has an associated nonnegative processing time of $p_k$. We are also given $m$ identical machines, $M_1, M_2, \cdots, M_m$. Any job can run on any machine. A schedule specifies, for each job $J_k$, the machine on which it runs and the time period during which it runs. Each job $J_k$ must run on some machine $M_i$ for $p_k$ consecutive time units, and during that time period no other job may run on $M_i$. Let $C_k$ denote the completion time of job $J_k$ , that is, the time at which job $J_k$ completes processing. Given a schedule, we define $C_{\max} = \max_{1 \le j \le n} C_j$ to be the makespan of the schedule. The goal is to find a schedule whose makespan is minimum.

   Given a parallel-machine-scheduling problem, we let $C^*_{\max}$ denote the makespan, of an optimal schedule.

   In class, we have seen 2 lower bounds for $C^*_{\max}$.

   - The optimal makespan is at least as large as the greatest processing time, that is,

     $C^*_{\max} \ge \max_{1 \le k \le n} p_k$

- The optimal makespan is at least as large as the average machine load, that is,

$$C^*_{\max} \geq \frac{1}{m} \sum_{1 \leq k \leq n} p_k$$

Suppose that we use the following greedy algorithm for parallel machine scheduling: whenever a machine is idle, schedule any job that has not yet been scheduled.
For the schedule returned by the greedy algorithm, show that

$$C_{\max} \leq \frac{1}{m} \sum_{1 \leq k \leq n} p_k + \max_{1 \leq k \leq n} p_k$$

Conclude that this algorithm is a polynomial-time 2-approximation algorithm.