**cs473: Algorithms** Assigned: *Tue., Sep. 10, 2019*

# Problem Set #2

Prof. Michael A. Forbes
Prof. Chandra Chekuri

Due: *Wed., Sep. 18, 2019 (10:00am)*

Some reminders about logistics.

- **Submission Policy:** See the course webpage for how to submit your pset via gradescope.
- **Collaboration Policy:** For this problem set you are allowed to work in groups of up to three. Only one copy should be submitted per group on gradescope. See the course webpage for more details.
- **Late Policy:** Late psets are not accepted. Instead, we will drop several of your lowest pset problem scores; see the course webpage for more details.

All (non-optional) problems are of equal value.

1. Suppose you have $k$ dollars to invest in $n$ investment options. Investing $a$ dollars in option $i$ will fetch you a profit of $f_i(a)$ dollars — these are complicated investments and hence the actual return is not a well-defined function but rather given as a table entry. Investments can be only be done in full dollar amounts. How do you spread your $k$ dollars among the $n$ options to maximize your profit?

   Mathematically we formalize it as follows. We are given $n$ functions $f_1, f_2, \ldots, f_n$ which can be accessed as sub-routines. Given an integer $a$ we can obtain the value $f_i(a)$ in constant time. We wish to solve the following problem: given integer $k \geq 0$ find integers $k_1, k_2, \ldots, k_n \geq 0$ to maximize $\sum_{i=1}^{n} f_i(k_i)$ subject to the condition that $\sum_{i=1}^{n} k_i \leq k$ (not all $k$ dollars need to be invested if it is not profitable to do so).

   (a) Describe an algorithm for this problem that runs in time polynomial in $k$ and $n$.

   (b) Describe how to implement your algorithm so that it uses $O(k)$ space.

2. We have seen an algorithm to solve the maximum weight independent set problem in a given node-weighted tree. Consider the following generalization. The input now consists of a tree $T = (V, E)$ with non-negative integer weights $w : V \to \mathbb{Z}_{\geq 0}$ and also an integer $k$. Describe an efficient algorithm that computes the maximum weight independent set with $\leq k$ nodes.

   *Hint:* Consider using the algorithm from problem (1).

3. Let $G = (V, E)$ be a directed graph and let $k$ be an integer. Describe an efficient algorithm that given two nodes $s, t \in V$ checks whether there is an $(s, t)$-*walk* in $G$ that contains $\geq k$ distinct nodes. Note that it is important that we ask for a walk and not a (simple) path for otherwise the problem would be NP-complete.

   (a) Develop an algorithm when $G$ is a directed acyclic graph (DAG).

   (b) Develop an algorithm for the general case using the meta-graph DAG of strongly-connected components of a directed graph. (If you are unfamiliar with this concept, then see for example Chapter 3 of Dasgupta-Papadimitriou-Vazirani).
   *Hint:* What is the answer if $G$ is strongly connected?

4. (**optional**, *not* for submission) Given an undirected graph $G = (V, E)$ we defined its *square*, denoted by square($G$) as the graph $G' = (V, E')$ where $(u, v) \in E'$ iff there is a path of length $\leq 2$ between $u$ and $v$ in $G$. That is, $(u, v) \in E'$ if $(u, v) \in E$ or if there is a node $w$ such that $(u, w), (w, v) \in E$. In class we saw an algorithm for the maximum weight independent set problem in a tree $T = (V, E)$. Design and analyze an algorithm for the maximum weight independent set problem for the square of a given tree $T = (V, E)$.

5. (**optional**, *not* for submission) Given a tree $T = (V, E)$ describe an efficient algorithm to count the number of (*all*, not necessarily maximum) independent sets in $T$. Also for counting the number of (*all*, not necessarily minimal) dominating sets in $T$.

6. (**optional**, *not* for submission) Consider the following multi-processor scheduling problem. The input consists of $n$ jobs $J_1, J_2, \ldots, J_n$ and $m$ identical machines $M_1, M_2, \ldots, M_m$. Each job $J_i$ has a non-negative size $s_i$. The goal is to assign the jobs to the machines to minimize the maximum load over all machines. The *load* of a machine is the sum of the sizes of the jobs assigned to it. This is an NP-hard problem in general.

   However, here we will consider the setting where there are only 3 distinct job sizes $\{a, b, c\}$. That is, $s_i \in \{a, b, c\}$ for $1 \leq i \leq n$. In this case it suffices to specify the job instance by $m$, the sizes $a, b, c$, and 3 integers $n_a, n_b, n_c$ which indicate the number of jobs of each size. Describe an algorithm that runs in $(n + m)^{O(1)}$ time for this problem where $n = n_a + n_b + n_c$ is the total number of jobs.

   *Hint:* The number 3 is not important. In fact one can devise an $(n + m)^{O(k)}$-time algorithm if the jobs have only $k$ distinct job sizes.