CSE 4190.101: Discrete Mathematics Fall 2021

Problem Set 2

Instructor: Yongsoo Song **Due on:** Oct 22, 2021

Please submit your answer through eTL. It should be a single PDF file (not jpg), either typed or scanned. **Please include your student ID and name (e.g. 2020-12345 YongsooSong.pdf).** You may discuss with other students the general approach to solve the problems, but the answers should be written in your own words. You should cite any reference that you used, and mention what you used it for. You should follow the academic integrity rules that are described in the course information.

Problem 1 (10 points)

We define the Ulam numbers by setting $u_1 = 1$ and $u_2 = 2$. Furthermore, after determining whether the integers less than n are Ulam numbers, we set n equal to the next Ulam number if it can be written uniquely as the sum of two different Ulam numbers. Note that $u_3 = 3, u_4 = 4, u_5 = 6$, and $u_6 = 8$. Prove that there are infinitely many Ulam numbers.

Problem 2 (10 points)

The ternary search algorithm locates an element in a list of increasing integers by successively splitting the list into three sublists of equal (or as close to equal as possible) size, and restricting the search to the appropriate piece. Describe the algorithm using pseudocode and determine the worst-case complexity (assume that the length of the input list is 3^k for some $k \geq 0$).

Problem 3 (10 points)

(a) Suppose that $f(x) = O(g(x))$. Does it follow that $2^{f(x)} = O(2^{g(x)})$? Justify your answer.

(b) Determine whether $\log(n!)$ is $\Theta(n \log n)$. Justify your answer.

Problem 4 (25 points)

The stable matching problem, in its most basic form, takes as input equal numbers of two types of participants (n men and n women, for example), and an ordering for each participant giving their preference for whom to be matched to among the participants of the other type. A matching is called **stable** if there is no match between a man and a woman where both prefer someone else to their current partner. The **Gale–Shapley algorithm** can be used to solve the stable matching problem. It involves a number of "rounds":

1. In the first round, each unengaged man proposes to the woman he prefers most, then each woman replies "maybe" to her suitor she most prefers and "no" to all other suitors. She is then provisionally "engaged" to the suitor she most prefers so far, and that suitor is likewise provisionally engaged to her.
2. In each subsequent round, first each unengaged man proposes to the most-preferred woman to whom he has not yet proposed (regardless of whether the woman is already engaged), and then each woman replies "maybe" if she is currently not engaged or if she prefers this man over her current provisional partner (in this case, she rejects her current provisional partner who becomes unengaged). The provisional nature of engagements preserves the right of an already-engaged woman to "trade up" (and, in the process, to "jilt" her until-then partner). This process is repeated until everyone is engaged.

(a) Use pseudocode to describe the algorithm and determine the worst-case complexity.

(b) Show this algorithm guarantees that everyone gets married and the matching is stable.

(c) Show by giving an example that each woman may be able to misrepresent her preferences and get a better match (so the algorithm is non-truthful for the women).

An approximation algorithm for an optimization problem produces a solution guaranteed to be close to an optimal solution. More precisely, suppose that the optimization problem asks for an input S that maximizes (or minimizes) $F(X)$ where F is some function of the input X. If an algorithm always finds an input T with $F(T) \geq c \cdot F(S)$ (or $F(T) \leq c \cdot F(S)$, respectively), where c is a fixed positive real number, the algorithm is called a c-approximation algorithm for the problem.

Problem 5 (20 points)

The **knapsack problem**: Suppose that we have a knapsack with total capacity W. We also have n items where item j has value $v_j > 0$ and weight $0 < w_j \leq W$. The **knapsack problem** asks for a subset of these n items such that the total weight is $\leq W$ and the total value is as large as possible.

Consider the following greedy algorithm: order all items in decreasing value/weight ratio (and relabel) such that $v_1/w_1 \geq v_2/w_2 \geq \cdots \geq v_n/w_n$, and pack items greedily as long as possible (i.e., take the first k items where $k \leq n$ is the maximum integer such that $\sum_{j=1}^{k} w_j \leq W$).

(a) Show that this algorithm may be arbitrarily bad. In other words, the greedy algorithm is not a c-approximation algorithm for the knapsack problem for any constant $c > 0$.

(b) Consider the following modified algorithm: compute the greedy solution as before and find the item of maximum value $v_i = \max_j v_j$. Output the maximum of the greedy algorithm and v_i. Show that this new algorithm gives a 1/2-approximation algorithm.

Problem 6 (25 points)

In computing, load balancing refers to the process of distributing a set of tasks over a set of resources (computing units), with the aim of making their overall processing more efficient. Suppose that we are given a collection of m machines. There are n jobs, t_j is the time required to run job j, each job runs without interruption on a single machine until finished, and a machine can run only one job at a time. The **load** L_k of machine k is the sum over all jobs assigned to machine k of the times required to run these jobs. The **makespan** is the maximum load over all the m machines (i.e., $\max_{1 \leq k \leq m} L_k$). Our goal is to find an assignment of jobs to machines to minimize the makespan.

(a) Suppose that L^* is the minimum makespan. Show that $L^* \geq \max_{1 \leq j \leq n} t_j$ and $L^* \geq \frac{1}{m} \sum_{1 \leq j \leq n} t_j$.

(b) Write out in pseudocode the greedy algorithm that goes through the jobs in order and assigns each job to the processor with the smallest load at that point in the algorithm. Prove that the algorithm is a 2-approximation algorithm.

(c) Take a modification on the first step: order the jobs in decreasing order of processing time. Prove that the new algorithm is a 3/2-approximation algorithm (Hint: Let L be the makespan and let k be a machine with load L in the output. Consider the last job assign to machine k. Fact: This is actually a 4/3-approximation algorithm).