CSE 4190.101: Discrete Mathematics Fall 2021

Problem Set 2 Solution

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.

Overall Grading Policy

1. No submissions or late submissions. (0 points received.)
2. Cheating detected. (0 points received.)
3. Not a single file. (including multiple files compressed into a single ZIP file) (10 points deducted.)
4. Not in a PDF format. (10 points deducted.)

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.

Suppose that the number of Ulam numbers is finite. Let A and B be the largest and the second largest Ulam number. By adding A and B, we can find a new number C. Since C is the sum of the two largest Ulam numbers, the sum of any other two Ulam numbers is less than C and thus C can be written uniquely as a sum of two Ulam numbers. However, there is a contradiction that C is not the largest Ulam number. Therefore, 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 $k \geq 0$).

Algorithm 1 Ternary Search Algorithm

Input: $a_1, a_2, \cdots, a_n$: increasing numbers, x: number to find location
$i := 1$ ▷ start index of list
$j := n$ ▷ end index of list
while $i < j$ **do**
  $m_1 := \lfloor \frac{2i+j}{3} \rfloor$
  $m_2 := \lfloor \frac{i+2j}{3} \rfloor$
  if $x > a_{m_2}$ **then** ▷ Search x in $a_{m_2+1}, \cdots, a_j$
    $i \leftarrow m_2 + 1$
  else if $x < a_{m_1}$ **then** ▷ Search x in $a_i, \cdots, a_{m_1-1}$
    $j \leftarrow m_1 - 1$
  else ▷ Search x in $a_{m_1}, \cdots, a_{m_2}$
    $i \leftarrow m_1$
    $j \leftarrow m_2$
  end if
end while
if $x = a_i$ **then return** i
else return 0 ▷ Not found
end if

For every iteration, the length of list is reduced by 1/3 and the algorithm ends when the length becomes 1 in the worst case. Let k be the number of iteration, Then the length after k iteration $l = \frac{n}{3^k} = 1$. Then $k = \log_3 n = \Theta(\log n)$ is the worst case time complexity.

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.

Let $f(x) = 2x, g(x) = x$, then clearly $f(x) = O(g(x))$, but there is no C, k such that $\left|2^{f(x)}\right| = \left|2^{2x}\right| \leq C \cdot \left|2^{g(x)}\right| = C\left|2^x\right|$ for all $x > k$. So $2^{f(x)} = O\left(2^{g(x)}\right)$ is not true.

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

First, show that $\log n! = O(n \log n)$: for all $n \in \mathbb{N}$,

$$\log n! = \sum_{k=1}^{n} \log k \leq \sum_{k=1}^{n} \log n = n \log n$$

Then, show that $\log n! = \Omega(n \log n)$: for all $n \geq 4$,

$$\log n! = \sum_{k=1}^{n} \log k \geq \sum_{k=1}^{\lfloor n/2 \rfloor} 0 + \sum_{k=\lfloor n/2 \rfloor + 1}^{n} \log \left\lceil \frac{n}{2} \right\rceil \geq \frac{n}{2} \log \frac{n}{2} = \frac{n}{2} \log n - \frac{n}{2} \log 2 \geq \frac{1}{4} n \log n$$

Therefore, $\log n! = \Theta(n \log n)$.

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.

Let men be M_i, women be W_i $(1 \leq i \leq n)$.

Algorithm 2 Gale-Shapley Algorithm

```
for each unengaged men do
    M_i proposes to the woman he prefers most
end for
for each woman who has received proposals do
    if W_j prefers M_i the most among the men who have proposed to W_j then
        M_i ← engaged
    end if
end for
while there exists unengaged man do
    for each unengaged man do
        M_i proposes to the most-preferred woman to whom he has not yet proposed
    end for
    for each woman who has received proposals do
        if W_j prefers M_i the most among the men who have proposed to W_j then
            if W_j currently has no partner then
                M_i ← engaged
            else if W_j prefers M_i over her current provisional partner M_k then
                Her current provisional partner M_k ← unengaged
                M_i ← engaged
            end if
        end if
    end for
end while
```

Worst-case complexity: $O(n^2)$

Solution 1

If all men have same preferences, all unengaged men will propose to same woman at each round. Since only one man can be engaged at each round, the number of rounds is equal to the number of men. From the first round to the last round, 1) the number of proposals is $n + (n-1) + (n-2) + ... + 1 = \frac{n(n+1)}{2}$, 2) the number of accepts is n, and 3) the number of rejects is $(n-1) + (n-2) + ... + 1 = \frac{n(n-1)}{2}$. Since 1) + 2) + 3) $\leq C|n^2|$, the worst-case complexity is $O(n^2)$.

Solution 2
For each step, each man proposes to woman that he didn't proposed yet, so, each man can propose at most n woman and there is n man, then the worst case time complexity is at most $n \times n = O(n^2)$ (time complexity of each woman selects from received proposals is same as the number of proposals made by men)

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

Everyone gets married
Suppose that there are unengaged man M_i and woman W_j. It means that M_i proposed to all women but rejected. However, if W_j receives a proposal, W_j has to choose and engage with the man who proposed to W_j. Therefore, this is a contradiction.

The matching is stable
Suppose that M_i and W_j prefer each other over the others but not their current partner. It means that M_i proposed to W_j but rejected, and this is a contradiction.

(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).

Suppose that there are three men (M_1, M_2, and M_3), three women (W_1, W_2, and W_3), and preferences.
1) Preference order of W_1: $M_1 \rightarrow M_2 \rightarrow M_3$.
2) Preference order of W_2: $M_2 \rightarrow M_1 \rightarrow M_3$.
3) Preference order of W_3: $M_3 \rightarrow M_1 \rightarrow M_2$.
4) Preference order of M_1: $W_2 \rightarrow W_1 \rightarrow W_3$.
4) Preference order of M_2: $W_1 \rightarrow W_2 \rightarrow W_3$.
4) Preference order of M_3: $W_2 \rightarrow W_1 \rightarrow W_3$.
If W_1 tells her preferences correctly.
- 1st round
  - (M_1, W_2), (M_2, W_1)
- 2nd round
  - (M_1, W_2), (M_2, W_1)
- 3rd round
  - (M_1, W_2), (M_2, W_1), (M_3, W_3)

If W_1 misrepresents her preferences. (Preference order of W_1: $M_1 \rightarrow M_3 \rightarrow M_2$)
- 1st round
  - (M_1, W_2), (M_2, W_1)
- 2nd round
  - (M_1, W_2), (M_3, W_1)
- 3rd round
  - (M_2, W_2), (M_3, W_1)
- 4th round
  - (M_2, W_2), (M_1, W_1)
- 5th round
  - (M_2, W_2), (M_1, W_1), (M_3, W_3)

If W_1 misrepresents her preferences, W_1 is engaged with the most preferred man.

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 $c > 0$.

Suppose that there are two items, $(v_i,\ w_i) = (2,\ 1)$ and $(W,\ W)$ where $W \geq 2$. Since $v_1/w_1 > v_2/w_2$ the greedy algorithm will choose the first item, but the optimal solution is to choose the second item. Then, $F(T) = 2$ and $F(S) = W$, but there is no fixed positive real number c value satisfying $F(T) = 2 \geq cF(S) = cW$.

(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.

Let S be the optimal solution, S_g be the solution obtained by using the original greedy algorithm, and S_m be the solution obtained by using the new greedy algorithm. Then, $S_g + v_i \leq S_m + S_m$. In addition, since there is no weight limitation on $S_g + v_i$, $S_{optimal} \leq S_g + v_i$. Therefore, $S_{optimal} \leq S_g + v_i \leq S_m + S_m$, i.e. $\frac{1}{2} S_{optimal} \leq S_m$.

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 $L^* \geq \frac{1}{m} \sum_{1 \leq j \leq n} t_j$.

$L^* \geq \max\limits_{1 \leq j \leq n} t_j$
Since every job must be assigned to machines, there is one machine m_i which processes the largest job. Therefore, $\max\limits_{1 \leq j \leq n} t_j \leq L_i \leq L^*$.
$L^* \geq \dfrac{1}{m} \sum\limits_{1 \leq j \leq n} t_j$

$\iff mL^* \geq \sum_{1 \leq j \leq n} t_j$

Since $\sum_{1 \leq j \leq n} t_j = \sum_{1 \leq i \leq m} L_i$ and $\sum_{1 \leq i \leq m} L_i \leq mL^*$, $\sum_{1 \leq j \leq n} t_j = \sum_{1 \leq i \leq m} L_i \leq mL^*$

(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.

Algorithm 3 Greedy Algorithm for Load Balancing

```
for i ← 1 to n do
    j = arg min_k L_k
    L_k += t_i
end for
```

Let m_k have the maximum load L_k, j be the last job on m_k. When j was assigned to m_k, m_k had the minimum load.
Therefore, $(L_k - t_j) + t_j \leq \frac{1}{m} \sum_{1 \leq i \leq m} L_i + \max_{1 \leq j \leq n} t_j \leq L^* + L^* = 2L^*$

(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).

If $n \leq m$
Since each machine has only one or no job, it is the optimal solution. If $n > m$
By pigeonhole principle, there is a machine which has two jobs of the first $m + 1$ jobs. and t_{m+1} is the minimum of $\{t_1, \cdots, t_{m+1}\}$, so $2t_{m+1} \leq L^*$.
Let j be the last job on m_k. If $j \leq m$, it implies that m_k has only one job because first m jobs should be distributed to each m machines. then $L_k = t_j \leq L^* \leq \frac{3}{2} L^*$

If $j > m$, then $2t_j \leq 2t_{m+1} \leq L^* \implies t_j \leq \frac{1}{2} L^*$
Therefore, $(L_k - t_j) + t_j \leq L^* + \frac{1}{2} L^* = \frac{3}{2} L^*$