Due 23:59 Nov 16 (Sunday). There are 100 points in this assignment.
Submit your answers **(must be typed)** in pdf file to CourSys
`https://coursys.sfu.ca/2025fa-cmpt-705-x1/`.
Submissions received after 23:59 will get penalty of reducing points: 20 and 50 points deductions for submissions received at $[00:00, 00:10]$ and $(00:10, 00:30]$ of Nov 17, respectively; no points will be given to submissions after $00:30$ of Nov 17.

1. (Chapter 11 Problem 1 of the text book) 15 points

   There are n containers $1, .., n$ of weights $w_1, .., w_n$ and some trucks, each truck can hold at most K units of weight (w_i and K, $w_i \leq K$, are positive integers). Multiple containers can be put on a truck subject to the weight restriction K. The problem is to use a minimum number of trucks to carry all containers. A greedy algorithm for this problem is as follows: start with an empty truck and put containers $1, .., j$ on the truck to have this truck loaded (i.e., $\sum_{1\leq i\leq j} w_i \leq K$ and $\sum_{1\leq i\leq j+1} w_i > K$); then put containers $j+1, j+2, ...$ on a new empty truck to have this truck loaded; continue this process until all containers are carried.

   (a) Give an example of a set of containers and a value K to show that the greedy algorithm above does not give an optimal solution.

   Solution:
   Consider this example $k = 5$, w_1, to w_4 equal to 3, 4 ,5 ,2 respectively. The greedy algorithm returns 4 trucks, (each container in one truck) but the if items with weights 2 and three put in one truck, the total number of trucks are 3. Thus, the greedy algorithm does not always return the optimal solution.

   (b) Prove the greedy algorithm is a 2-approximation algorithm.

   Solution:
   Let m be the total number of trucks from the Greedy algorithm and the optimal number of trucks be *opt*. We have:

   $$\sum_{i=1}^{n} w_i > n \times opt$$

   let L_j be the load of jth truck from the solution of the Greedy Algorithm. We have:

   $$\sum_{j=1}^{m} L_j = \sum_{i=1}^{n} w_i$$

   Let w_j be the first container in the sequence that could not be loaded into truck j. We have:

   $$K < L_j + w_j, \quad \sum_{j=1}^{m} w_j < \sum_{i=1}^{n} w_i$$

Sum the first inequality over all m trucks to get:

$$K \times m < \sum_{j=1}^{m} L_j + \sum_{j=1}^{m} w_j = \sum_{i=1}^{n} w_i + \sum_{j=1}^{m} w_j < \sum_{i=1}^{n} w_i + \sum_{i=1}^{n} w_i = 2 \times \sum_{i=1}^{n} w_i \leq 2k \times opt$$

Thus: $m < 2opt$

2. (Chapter 11 Problem 2 of the text book) 10 points

Given a set S of strings, for any two strings p and q in S, a distance $d(p,q) \geq 0$ is defined. Given a similarity threshold value $\Delta \geq 0$, two strings p and q are called similar if $d(p,q) \leq \Delta$. A subset R of S is called a representative set of S if for any string $p \in S$, there is a string $q \in R$ with $d(p,q) \leq \Delta$. The minimum representative set problem is to find a representative set of minimum size.

Give a polynomial time $O(\log n)$-approximate algorithm for the problem and analyze the algorithm.

Solution:
Run the following algorithm to find S_q for each q in S. This algorithm runs in $O(n^2)$ time.

Input: S and $d(p,q)$, for each p,q in S.
Output: $S_q = \{p | d(p,q) \leq \Delta\} \cup \{q\}$

```
for each string q in S:
    S_q = {q}
    for each string p in s:
        if d(p, q) ≤ Δ
            S_q = S_q ∪ {p}
        End if
    End for
End for
```

S_q, for each $q \in S$ is a set cover for S as S_q contains at least q, thus $\cup_q S_q = S$

Apply the Greedy Set Cover algorithm introduced in lecture notes. This algorithm is a $(1 + lnn)$-approximation. Prove is provided in the lecture notes. The first step runs in $O(n^2)$ and the greedy algorithm runs in poly time. Thus, the algorithm runs in poly time.

3. (Chapter 11 Problem 5 of the text book) 15 points

(a) Consider a load balancing problem instance of 10 machines and n jobs $S = \{1, .., n\}$ with $1 \leq t_i \leq 50$ for every i and $\sum_{i=1}^{n} t_i \geq 3000$. Prove that the greedy algorithm discussed in class finds a solution of makespan T with $T \leq (1.2)(\sum_{1 \leq i \leq n} t_i)/10$ for this instance.

Solution:
let L be the solution of the greedy algorithm:

We have:

$$L \geq \frac{\sum t_i}{m} = \frac{3000}{100} = 300$$

Thus, the load on the machine with longest run time is at least 300. Since each job runs in at most 50, thus there are at least 6 jobs on the machine with the longest running time.

Let L_i be the machine with the longest running time and t_l be the last job assigned to this machine. Since t_l is the last assigned job to L_i, it is shorter than all other jobs assigned to L_i and at least 5 other jobs are assigned to L_i. Thus, $\sum_{j=1}^{5} t_j \leq L_i$ when t_1 to t_5 be those 5 other jobs. We have:

$$t_l \leq \frac{\sum_{j=1}^{5} t_j}{5} \leq \frac{L_i}{5} \leq \frac{\sum t_i}{5m}$$

By the algorithm, at the time this job was assigned machine i has the least load. Thus, $L_i - t_l \leq L_k$, for all machine k. Sum this inequality over all m machines, we have:

$$m \times L_i - m \times t_l \leq \sum_{k}^{m} L_k = \sum_{i}^{n} t_i$$

thus:

$$l_i \leq \frac{\sum_i^n t_i}{m} + t_l \leq \frac{\sum_i^n t_i}{m} + \frac{\sum_i^n t_i}{5m} = 1.2 \frac{\sum_i^n t_i}{m}$$

Put $m = 10$ and $T = L_i$ we get the requested upper bound.

(b) Implement the greedy algorithm to find a solution for the load balancing problem instance in (a) and compare the solution with the lower bound $(\sum_{1 \leq i \leq n} t_i)/10$ on the makespan of the instance (each t_i can be generated randomly). Report your results for T, $(\sum_{i=1}^{n} t_i)/10$ and $T/((\sum_{i=1}^{n} t_i)/10)$ with $n = 100$.

Solutions:
Number of jobs n : 100
Number of machines m : 10
Makespan T : 308
$(\sum_{i=1}^{n} t_i)/10$: 306.90
$T/((\sum_{i=1}^{n} t_i)/10)$: 1.0036

4. (Chapter 11 Problem 7 of text book) 15 points

Given a set of customers $\{1, 2, .., n\}$, each customer has a value v_i and is shown to one of the advertisements (ads) $A_1, .., A_m$. A selection of ads to customers is to assign one ad to each customer. For a set C_j of customers assigned ad A_j, the total value of C_j is $v(C_j) = \sum_{i \in C_j} v_i$. The spread of the selection is $\min_{1 \le j \le m}\{v(C_j)\}$. The problem of finding the maximum spread is NP-hard. Give a (1/2)-approximation algorithm for finding the maximum spread for any input instance with $v_i \le (\sum_{k=1}^{n} v_k)/(2m)$ for $1 \le i \le n$, and analyze the algorithm.

Solution:
Use an algorithm similar to that of load balancing:

$LPT - (m, n, v_1, v_2, .., v_n)$
Input: m ads and n customers with value v_i.
Output: A allocation of customers to adds.
  Sort customers so that $v_1 \ge v_2 \ge .. \ge v_n$;
  for $j = 1$ to m do
    $v(C_j) = 0$; /* load on machine i */
    $C_j = \varnothing$; /* jobs assigned to machine i */
  end for
  for i = 1 to n do
    $j = argmin_{1 \le k \le m}$ min $v(C_k)$; /* ad j has the smallest value */
    $C_j = C_j \cup \{i\}$; /* assign job j to machine i */
    $v(C_j) = v(C_j) + t_i$; /* update load of machine i */
  end for
  Output $v(C_j)$ and C_j for $1 \le i \le m$

Running time:
Implementation takes $O(n \log n)$ time using a priority queue and merge-sort

Prove:
Let *opt* be the optimal spread. Thus:

$$\frac{\sum_{i=1}^{n} v_i}{m} \ge opt$$

V be the solution of the greedy algorithm., and v_l be the last customer assigned to the ad j which has the largest value on the solution of the greedy algorithm.

by definition:

$$v(C_j) \ge \frac{\sum_{k=1}^{m} v(C_k)}{m} = \frac{\sum_{i=1}^{n} v_i}{m}$$

At the time v_l assigned to ad j, it has the least value, even less than the spread. Thus,

$$V \ge v(C_j) - v_l \ge \frac{\sum_{i=1}^{n} v_i}{m} - v_l$$

from the question, we have: $v_i \le (\sum_{k=1}^{n} v_k)/(2m)$. Thus:

$$V \ge \frac{\sum_{i=1}^{n} v_i}{m} - v_l \ge \frac{\sum_{i=1}^{n} v_i}{m} - \frac{\sum_{i=1}^{n} v_i}{2m} = \frac{\sum_{i=1}^{n} v_i}{2m} \ge \frac{opt}{2}$$

Thus, the algorithm is 1/2 approximate.

5. (Chapter 11 Problem 8 of text book) 15 points

   For every instance of the load balancing problem discussed in class, there exists an order of the jobs so that when Greedy-Balance processes the jobs in this order, it produces an assignment of jobs to machines with the minimum possible makespan.

   Solution:
   Yes. We can construct this order for every input instance.

   First, find the minimum possible makespan time allocation. If there are more than one solution with the minimum makespan, pick a solution for which the minimum of run time of all machines is maximized. Let S_j be the jobs assigned to machine j in this optimal solution.

   Make sequence R as follow: Put the first jobs in S_1 to S_m as the first m elements of sequence R. The greedy algorithm assigns each job to the associated machine.

   After that, if machine j has the lowest run time, put one job from S_j in the sequence. If two machine have the minimum run time, choose a job from S_j with lowest j, as this is the tie breaking rule in the greedy algorithm.

   If jobs are fed to the greedy algorithm in order of sequence R, the greedy algorithm returns optimal solution.

   Prove:
   By contradiction assume we cannot follow the algorithm. That is at the end of the algorithm, machine j has never had the lowest run time, but there is a job a_m in S_j that is not assigned to sequence R.

   if $L_j - a_m$ is larger than all run times, then can we move job a_m from machine j and another machine and decrease the makespan, which is a contradiction with the fact that it was optimal.

   Let machine k has the smallest run time. thus: $L_k < L_j - a_m$

   Hence if we move job a_m from machine j and put it on machine k the run time of machine k will increase. That means L_k could be larger while makespan the same, which is a contradiction with the fact that the machine with solution has the maximum, of minimum run time for all machines.

6. (Chapter 10 Problem 2 of text book) 15 points

   Given a 3-SAT instance Φ of n variables, a truth assignment $f : \{x_1, .., x_n\} \to \{0, 1\}$ and an integer $d \geq 0$, the procedure Explore(Φ, f, d) below answers whether there is a truth assignment $f' : \{x_1, .., x_n\} \to \{0, 1\}$ such that $d(f, f') = |\{i|f(x_i) \neq f'(x_i)\}| \leq d$ and f' satisfies Φ.
   Explore(Φ, f, d)
       if f satisfies Φ then return YES
       else if $d = 0$ then return NO

else
let $C = (l_1 \vee l_2 \vee l_3)$ be a clause of Φ that is not satisfied by f;
let f_i $(i = 1, 2, 3)$ be the truth assignment obtained from
f by inverting the assigned value to l_i;
if Explore($\Phi, f_1, d-1$)=YES then return YES;
if Explore($\Phi, f_2, d-1$)=YES then return YES;
if Explore($\Phi, f_3, d-1$)=YES then return YES;
return NO

(a) Prove Explore(Φ, f, d) returns YES iff there is a satisfying assignment f' with $d(f, f') \leq d$. Analyze the running time of Explore(Φ, f, d) as a function of n and d.

Solution:
If there is another assignment f' with $d(f, f') \leq d$, then if invert at ost d variables in f, it should be satisfiable. at each loop the algorithm changes one of them and try all possible variables in the clauses that are not satisfied. Thus, after at most d loop, it will reach to a satisfiable assignment and return yes

Proof by contradiction. Assume if return YES then there is no satisfying assignment f' with $d(f, f') \leq d$. The algorithm will invert at most d variables and get a satisfiable solution. Thus, there is a satisfiable solution with distance less than d. Which is a contradiction.

Running time:
$T(n, d) \leq 3 \times T(n, d-1) + O(m+n)$

Thus, based on the master theorem the running time is $O(n^{O(1)} 3^d)$

(b) Using Explore(Φ, f, d) as a subroutine, give an algorithm which decides whether a 3-SAT instance is satisfiable or not in $O(n^{O(1)}(\sqrt{3})^n)$ time.

The solution should have polynomial many calls to explore with d= n/2. In that case, the running time is $O(n^{O(1)}(\sqrt{3})^n)$.

We need to find a polynomial many initial assignments that cover all possible solutions with distance less than or equal to n/2.

Try the following method:

For $j = 1$ to n For $i = j$ to n

$f_{ij} =$ variables j to i equal to 1, and the rest equal to zero $Explore(\phi, f_{ij}, n/2)$

I think all other possible assignments are within distance n/2 of one assignments above. There are n^2 calls to explore function with $d = n/2$. Thus the running time is $O(poly(n)(\sqrt{3})^(n/2))$

7. (Chapter 10 Problem 5 of the text book) 15 points

Given a node weighted graph G (each node v of G is assigned a positive weight $w(v)$, a minimum weight dominating set D of G is a subset of $V(G)$ such that for every node u of G, either $u \in D$ or u is adjacent to a node $v \in D$ and $\sum_{v \in D} w(v)$ is minimized. Give a polynomial time dynamic programming algorithm (optimal solution structure, Bellman equation, pseudo code, and running time) to find a minimum weight dominating set D and the weight of D in a tree.

Solution:
For each u which is the root of a sub-tree, there are two options.
1. Include it and choose the minimum weight sub-tree rooted at each of its children, no matter include or not includes each child v
2. Not include it and include at least one of its child and the optimal sub-tree of each of the children. The child to include should has the least difference between the optimal solution of sub-tree starting from that node when including v or not including it.

Bellman equations:
let $opt_{in}(u)$ be the weight of min-DS in sub-tree rooted at u, containing u. let C_u be the collection of all children of u.

$$opt_{in}(u) = w(u) + \sum_{v \in C_u} \min\{opt_{in}(v), opt_{out}(v)\}$$

$$opt_{out}(u) = opt_{in}(j) + \sum_{v \in C_u \setminus \{j\}} \min\{opt_{in}(v), opt_{out}(v)\}$$

$$j = argmin_{v \in C_u}\{opt_{in}(v) - opt_{out}(v)\}$$

Algorithm:
min-Weight-DS-Tree(T)
Input: A node weighted tree T.
Output: Weight of min-weight DS S in T.
for each node u of T in postorder do
  if u is a leaf then $M_{in}[u] = w(u); M_{out}[u] = 0$
  for each child v of U:
    $j = argmin_{v \in C_u}\{M_{in}(v) - M_{out}(v)\}$
    $M_{in}[u] = w(u)$
    $S_{in}[u] = \{u\}$
    $M_{out}[u] = M_{in}[j]$
    $S_{in}[u] = \{j\}$
    if $M_{in}(v) < M_{out}(v)$, then
      $M_{in}[u] = M_{in}[u] + M_{in}(v)$
      $S_{in}[u] = S_{in}[u] \cup \{v\}$
      if $v \neq j$ then:
        $M_{out}[u] = M_{out}[u] + M_{in}(u)$
        $S_{out}[u] = S_{in}[u] \cup \{v\}$
      if $M_{out}(v) < M_{in}(v)$, then
        $M_{in}[u] = M_{in}[u] + M_{out}(v)$
        $S_{in}[u] = S_{out}[u] \cup \{v\}$
        if $v \neq j$ then:

$M_{out}[u] = M_{out}[u] + M_{out}(u)$
$S_{out}[u] = S_{out}[u] \cup \{v\}$
end for
end for
if $M_{in}[r] < M_{out}[r]$
return $S_{in}[r]$
else $S_{out}[r]$
Return $max M_{in}[r], M_{out}[r]$;

Running time:
Two loops, one over each node an another over each child. Thus, running time is $O(n + m)$, but since this graph is a tree, the running time is $O(n)$.