Algorithm Design and Analysis
Assignment 3
Deadline: April 26, 2023

1. (25 points) We have n homework assignments with weights $w_1, \ldots, w_n \in \mathbb{Z}^+$. Each homework assignment i requires w_i units of time to finish, and it will give you w_i points as reward. Suppose the semester starts at time 0 and ends at time T (for some $T \in \mathbb{Z}^+$). Suppose all the homework assignments are released at the beginning of the semester. Each homework assignment i needs to be completed before the end of the semester in order to receive the w_i points reward. Ideally, you would like to complete all the homework assignments in the time interval $[0, T]$, but this is impossible if $\sum_{i=1}^{n} w_i > T$. Therefore, your objective is to choose a subset of homework assignments that maximizes the reward.

   Consider the following greedy algorithm.

   1. initialize $S \leftarrow \emptyset$;
   2. let $\Gamma(S)$ be the set of homework assignments i satisfying 1) $i \notin S$ and 2) the weight of $S \cup \{i\}$ does not exceed T; find the homework assignment in $\Gamma(S)$ with the largest weight and include it to S;
   3. keep doing the second step until $\Gamma(S) = \emptyset$;
   4. return S.

   Suppose each w_i is an integer power of 2. That is, for each i, there exists $k_i \in \mathbb{Z}_{\geq 0}$ such that $w_i = 2^{k_i}$. Prove that the greedy algorithm correctly outputs S with the maximum reward.

2. (25 points) Consider the same problem in Question 1. In this question, we no longer assume each w_i is an integer power of 2.

   (a) (10 points) Prove that the greedy algorithm in Question 1 is a 0.5-approximation algorithm.

   (b) (5 points) Provide a tight example to show that the greedy algorithm cannot do better than a 0.5-approximation.

   (c) (10 points) Consider a variant of the greedy algorithm where one or two assignments can be added to S in each iteration. In particular, at the second step, we consider all subsets A with $1 \leq |A| \leq 2$ and $S \cap A = \emptyset$ such that the weight of $S \cup A$ does not exceed T, and we update $S \leftarrow S \cup A$ for A with the maximum weight that satisfies our requirements. We keep doing this until no more update is possible.

   Prove that this is a $\frac{2}{3}$-approximation algorithm.

3. (30 points) In the class, we learned Kruskal's algorithm to find a minimum spanning tree (MST). The strategy is simple and intuitive: pick the best legal edge in each step. The philosophy here is that local optimal choices will yield a global optimal. In this problem, we will try to understand to what extent this simple strategy works. To this end, we study a more abstract algorithmic problem of which MST is a special case.

   Consider a pair $M = (U, \mathcal{I})$ where U is a finite set and $\mathcal{I} \subseteq \{0,1\}^U$ is a collection of subsets of U. We say M is a *matroid* if it satisfies

   元素包含的子集在其中

   - (**hereditary property**) $\mathcal{I}$ is nonempty and for every $A \in \mathcal{I}$ and $B \subseteq A$, it holds that $B \in \mathcal{I}$.
   - (**exchange property**) For any $A, B \in \mathcal{I}$ with $|A| < |B|$, there exists some $x \in B \setminus A$ such that $A \cup \{x\} \in \mathcal{I}$.

   Each set $A \in \mathcal{I}$ is called an *independent set*.

   (a) (6 points) Let $M = (U, \mathcal{I})$ be a matroid. Prove that maximal independent sets are of the same size. (A set $A \in \mathcal{I}$ is called *maximal* if there is *no* $B \in \mathcal{I}$ such that $A \subsetneq B$.) A是B的子集。

   (b) (6 points) Let $G = (V, E)$ be a simple undirected graph. Let $M = (E, \mathcal{S})$ where $\mathcal{S} = \{F \subseteq E \mid F \text{ does not contain a cycle}\}$. Prove that M is a matroid. What are the maximal sets of this matroid?

   (c) (6 points) Let $M = (U, \mathcal{I})$ be a matroid. We associate each element $x \in U$ with a nonnegative weight $w(x)$. For every set of elements $S \subseteq U$, the weight of S is defined as $w(S) = \sum_{x \in S} w(x)$. Now we want to find a maximal independent set with maximum weight. Consider the following greedy algorithm.

Algorithm 1 Find a maximal independent set with maximum weight

Input: A matroid $M = (U, \mathcal{I})$ and a weight function $w : U \to \mathbb{R}_{\geq 0}$.

Output: A maximal independent set $S \in \mathcal{I}$ with maximum $w(S)$.

```
1: $S \leftarrow \emptyset$
2: Sort $U$ into decreasing order by weight $w$
3: for $x \in U$ in decreasing order of $w$:
4:     if $S \cup \{x\} \in \mathcal{I}$:
5:         $S \leftarrow S \cup \{x\}$
6:     endif
7: endfor
8: return $S$
```

Now we consider the first element x the algorithm added to S. Prove that there must be a maximal independent set $S' \in \mathcal{I}$ with maximum weight containing x.

(d) (6 points) Prove that the greedy algorithm returns a maximal independent set with maximum weight. (Hint: Can you see that Algorithm 1 is just a generalization of Kruskal's algorithm?)

(e) (6 points) Let $U \subseteq \mathbb{R}^n$ be a finite collection of n-dimensional vectors. Assume $m = |U|$ and we associate each vector $\mathbf{x} \in U$ a positive weight $w(\mathbf{x})$. For any set of vectors $S \subseteq U$, the weight of S is defined as $w(S) = \sum_{\mathbf{x} \in S} w(\mathbf{x})$. Design an efficient algorithm to find a set of vectors $S \subseteq U$ with maximum weight and all vectors in S are linearly independent.

4. (20 points) **Amortized Cost of ADD.** Let us consider the following situation. An integer (initially zero) is stored in binary. We have an operation called ADD that adds one to the integer. The cost of ADD depends on how many bit operations we need to do. (one bit operation can flip 0 to 1 or flip 1 to 0.) The cost can be high when the integer becomes large. Use amortized analysis to show the amortized cost of ADD is $O(1)$. You should define the potential function in your analysis.

5. How long does it take you to finish the assignment (including thinking and discussing)? Give a score (1,2,3,4,5) to the difficulty. Do you have any collaborators? Write down their names here.