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.

   We first prove the following claim.

   (*) Suppose there is a homework assignment with weight $w = 2^k$ and there is a set of homework assignment $Q = \{w_1, \ldots, w_r\}$ with total weight at least w satisfying $w_i \leq w$ for each $i = 1, \ldots, r$. There exists a subset $R \subseteq Q$ such that the total weight of R is exactly w.

   This can be proved by induction on k. Can you prove it?

   Now we prove the correctness of the greedy algorithm. Let S_t be the current set of S after t iterations, where $S_0 = \emptyset$. We will prove by induction that S_t is a subset of an optimal solution.

   The base step with S_0 is trivial.

   For the inductive step, suppose S_t is a subset of an optimal solution S^*. Suppose $(t+1)$ is the homework assignment selected in the $(t+1)$-th iteration of the algorithm. If $(t+1) \in S^*$, S_{t+1} is a subset of S^*, and we are done. If not, by the greedy nature of the

algorithm, we have $w_{t+1} \geq w_j$ for each $j \in S^* \setminus S_t$. Moreover, the total weight of $S^* \setminus S_t$ is at least w_{t+1}, for otherwise the total weight of S^* is less than $S_{t+1} = S_t \cup \{w_{t+1}\}$ and S^* cannot be an optimal solution. By (*), we can replace some assignments in S^* by the assignment $(t+1)$ while keeping the total weight of S^* unchanged. This means there exists an optimal solution S^{**} which contains S_{t+1} as a subset.

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.

   We can obviously assume without loss of generality that $w_i \leq T$ for each $i = 1, \ldots, n$. Let S be the output of the greedy algorithm. Firstly, $S \neq \emptyset$, as at least one assignment can be completed due to $w_i \leq T$. Let w be the weight of the assignment chosen by our algorithm in the first iteration. In addition, $w_i \leq w$ for each $i = 1, \ldots, n$ by the greedy nature of the algorithm.

   If $S = \{1, \ldots, n\}$, S is clearly optimal, and we are done. Suppose there exists $j \notin S$. We use $w(S)$ to denote the total weight of S, and we have $T - w(S) < w_j$ (otherwise, j should be included in S). On the other hand, $w(S) \geq w$, as the first assignment in S has weight w. Putting together, $T - w(S) < w_j \leq w \leq w(S)$, which implies $w(S) > 0.5 \cdot T$. Since T is clearly an upper bound to the value of the optimal solution, the algorithm is a 0.5-approximation.

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

   $w_1 = 0.5 + \delta$ (where $0.5 > \delta > 0$), $w_2 = 0.5$, $w_3 = 0.5$, and $T = 1$. The optimal solution chooses $\{2, 3\}$ and the algorithm chooses $\{1\}$. Taking $\delta \to 0$ shows that the greedy algorithm cannot guarantee $(0.5 + \varepsilon)$-approximation for any $\varepsilon > 0$.

   (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.

   Let S^* be the optimal solution and S be the solution output by the algorithm. If $S^* \subseteq S$, S is clearly optimal. Thus, we assume $S^* \setminus S \neq \emptyset$ from now on.

   If there exists $j \in S^* \setminus S$ such that $w_j \leq \frac{1}{3} w(S^*)$, the algorithm already achieves a $\frac{2}{3}$-approximation. This is for the similar reason as it is in Part (a): we have $T - w(S) < w_j \leq \frac{1}{3} w(S^*)$ for otherwise j should be further included in S by our algorithm; this implies $\frac{1}{3} w(S^*) > T - w(S) \geq w(S^*) - w(S)$, which further implies $w(S) \geq \frac{2}{3} w(S^*)$. Therefore, from now on, we assume $w_j > \frac{1}{3} w(S^*)$ for every $j \in S^* \setminus S$.

   If there exist two assignments $j_1, j_2 \in S^*$ with $w_{j_1} > \frac{1}{3} w(S^*)$ and $w_{j_2} > \frac{1}{3} w(S^*)$, the algorithm also achieves a $\frac{2}{3}$-approximation. To see this, we have $w_{j_1} + w_{j_2} >$

$\frac{2}{3}w(S^*)$. On the other hand, by the greedy nature of the algorithm, the one or two assignment(s) chosen in the first iteration must have total weight at least $w_{j_1}+w_{j_2} > \frac{2}{3}w(S^*)$, which implies a $\frac{2}{3}$-approximation. Therefore, the only unsettled case is that S^* contains only one assignment j with weight more than $\frac{1}{3}w(S^*)$, and $S^* \setminus S = \{j\}$.

If an assignment i with weight at least w_j is chosen in the first iteration, S must be an optimal solution: since $w_i \geq w_j > \frac{1}{3}w(S^*)$ and j is the only assignment in S^* with weight larger than $\frac{1}{3}w(S^*)$, we have $i \notin S^*$; since $S^* \setminus S = \{j\}$ and $i \in S \setminus S^*$, we have $w(S) \geq w(S^* \cup \{i\} \setminus \{j\}) = w(S^*) - w_j + w_i \geq w(S^*)$.

It remains to analyze the case where all the assignments chosen in the first iteration have weights strictly less than w_j. In this case, we must have chosen two assignments (instead of only one) in the first iteration since the total weight for the assignments chosen in the first iteration must be at least w_j by the greedy nature. Let 1 and 2 with weights w_1 and w_2 be the two assignments chosen in the first iteration. We must have $w_1 + w_j > T$ and $w_2 + w_j > T$, for otherwise $\{1, j\}$ or $\{2, j\}$ will be a better pick than $\{1, 2\}$ for the first iteration. Moreover, we must also have $w_1 + w_2 \geq w_j$, for otherwise $\{j\}$ is a better pick than $\{1, 2\}$. Putting the three inequalities together, $w_1+w_j > T$ and $w_2+w_j > T$ imply $w_1+w_2+2w_j > 2T$, and substituting $w_1 + w_2 \geq w_j$ yields $w_1 + w_2 + 2(w_1 + w_2) > 2T$, which implies $w_1 + w_2 > \frac{2}{3}T \geq \frac{2}{3}w(S^*)$. The first two assignments pick by the algorithm already give us a $\frac{2}{3}$-approximation.

As a remark, this algorithm cannot do better than a $\frac{2}{3}$-approximation. Consider the following tight example with five assignments: $w_1 = \frac{1}{3}$, $w_2 = \frac{1}{3}$, $w_3 = \frac{1}{3}$, $w_4 = \frac{1}{3}+\delta$, $w_5 = \frac{1}{3} + \delta$, and $T = 1$.

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

   - (**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 $A \subsetneq B$.)

   We prove it by contradiction. Assume that there are two maximal independent sets A_1 and A_2, $|A_1| < |A_2|$. For A_1, there is no $B \in \mathcal{I}$ such that $A_1 \subsetneq B$. According to the **exchange property**, there exists some $x \in A_2 \backslash A_1$ such that $A_1 \cup \{x\} = B \supsetneq A_1$, which leads to contradiction. Hence all maximal independent sets are of the same size.

   (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?

   We first prove M is a matroid as follows:

   - **Hereditary**: Firstly for any single edge $e \in E$, $\{e\} \in \mathcal{S}$. So $\mathcal{S}$ is nonempty. For every $A \in \mathcal{S}$ and $B \subseteq A$, B must be acyclic as well since no new edges are introduced.
   - **Exchange**: $A, B \in \mathcal{S}$ with $|A| < |B|$. Denote the vertices of A, B as V_A, V_B. Denote the forest consisted of A's vertices and edges as F_A, and each subtree comprises a_i vertices. Consider the conditions of edges in $B \backslash A$:
     1. At least one vertex is not in F_A;
     2. Both vertices are in F_A but belong to different substrees;
     3. Both vertices are in F_A and belong to the same substree.

     1-type and 2-type edges won't make F_A cyclic, but 3-type edges will. For each subtree in F_A, there's at most $a_i - 1$ 3-type edges in B. Totally there can be

at most $\sum_i (a_i - 1) = |A|$ 3-type edges in B. From the assumption we know that $|B| > |A|$, so B contains at least one 1-type or 2-type edge e. Hence, $A \cup \{e\} \in \mathcal{S}$.

Therefore M is a matroid.

The maximal sets of M is the sets of spanning trees of G. The maximal property holds because any new edge will make it cyclic.

(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 ← ∅
2: Sort U into decreasing order by weight w
3: for x ∈ U in decreasing order of w:
4:     if S ∪ {x} ∈ I:
5:         S ← S ∪ {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.

Assume that x does not belong to any maximal independent set $S' \in \mathcal{I}$ with maximum weight. According to the **exchange property**, there exists some $y_1 \in S' \backslash \{x\}$ such that $\{x, y_1\} \in \mathcal{I}$. Repeat adding new element to S until $S = \{x, y_1, \ldots, y_{n-1}\}$ and $|S| = |S'|$. Due to the algorithm, x has the maximum weight, so

$$\begin{aligned} w(S) &= \sum_{z \in S} w(z) \\ &= w(x) + \sum_{i=1}^{n-1} w(y_i) \\ &\geq \sum_{z \in S'} w(z) = w(S'), \end{aligned}$$

which leads to contradiction. Therefore 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?)

Simply using the settings in problem (2), we've proved that M is a matroid. Associate each edge $x \in U$ with a non-negative weight $w(x)$. The algorithm is exactly the same as Kruskal's algorithm.

To prove that the algorithm does ouput a maximal independent set with maximum weight, we can use the conclusion in (3). For every new element added to S, the property in (3) also holds (just replace $\{x\}$ with $\{x_1, \ldots, x_k\}$, $k = 1, \ldots, n$ in the proof). Hence, the final output S must be a maximal independent set with maximum weight.

(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.

Let $\mathcal{I}$ be the collection of subsets of U that within each subset the vectors are linearly independent. We first prove that $M = (U, \mathcal{I})$ is a matroid.

- **Hereditary**: $\mathcal{I}$ is not empty since for every $\boldsymbol{x} \in U$, $\{\boldsymbol{x}\} \in \mathcal{I}$. For every $A \in \mathcal{I}$ and $B \subseteq A$, B must be linearly independent because no new vectors are introduced.
- **Exchange**: $A, B \in \mathcal{I}$ with $|A| < |B|$. Since A is linearly independent, it can be considered as a basis of the subspace V_A with dimension $r_A = |A|$. Likewise B is a basis of the subspace V_B with dimension $r_B = |B|$. Given that $|A| < |B|$, there must exist at least one vector $\boldsymbol{x} \in B \backslash A$ such that $\boldsymbol{x} \notin V_A$, otherwise V_B can simply be expressed by linear combinations of vectors in V_A. Hence $A \cup \{\boldsymbol{x}\}$ is also linearly independent, meaning that $A \cup \{\boldsymbol{x}\} \in \mathcal{I}$.

Then we design an algorithm (shown in in Algorithm 2) with the same idea in (c).

Algorithm 2 Find a set of vectors with maximum weight

1: **Input:**A matroid $M = (U, \mathcal{I})$ and a weight function $w : U \to \mathbb{R}_{\geq 0}$.
2: **Output:**A set of vectors $S \in \mathcal{I}$ with maximum $w(S)$. $S \leftarrow \emptyset$
3: Sort U into decreasing order by weight w
4: **for** $\boldsymbol{x} \in U$ in decreasing order of w **do**
5: **if** S and $\boldsymbol{x}$ are linearly independent **then**
6: $S \leftarrow S \cup \{\boldsymbol{x}\}$
7: **end if**
8: **end for**
9: **return** S

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.

   The potential function is defined as follows:

$$\phi(n) = \sum_{i=0}^{k} a_i, \text{ where } n = \overline{(a_k ... a_1 a_0)_2}.$$

   When the lowest i bits of n are 1 and the $i+1$th -lowest bit is 0, we only need to flip the lowest $i+1$ bits, hence $C_i = i+1$. Meanwhile, $\Delta\phi = -(i-1)$ since the lowest i 1 are flipped into 0 and the $i+1$th 0 is flipped into 1 and the number of 1 decreases $i-1$. Therefore,

$$\hat{C}_i = C_i + \Delta\phi = i + 1 - (i-1) = 2$$
$$\Rightarrow \sum_{i=0}^{n} C_i \leq 2(n+1) + \max_{1 \leq j \leq n} \phi(j) - \phi(0)$$
$$\Rightarrow \sum_{i=0}^{n} C_i \leq 2(n+1) + log(n) = O(n)$$

   By amortized analysis, the amortized cost of ADD is $\frac{O(n)}{n} = O(1)$. □

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.