

**Instructions:** You are welcome to form small groups (up to 4 people total) to work through the homework, but you **must** write up all solutions by yourself. List your study partners for homework on the first page, or “none” if you had no partners.

If using LaTeX (which we recommend), you may use the homework template linked on this [Piazza post](#) to get started.

Begin each problem on a new page. Clearly label where each problem and subproblem begin. The problems must be submitted in order (all of P1 must be before P2, etc). For questions asking you to give an algorithm, respond in what we will refer to as the *four-part format* for algorithms: main idea, pseudocode, proof of correctness, and running time analysis.

Read the [Homework FAQ Piazza post](#) on Piazza before doing the homework for more explanation on the four-part format and other clarifications for our homework expectations.

No late homeworks will be accepted. No exceptions. This is not out of a desire to be harsh, but rather out of fairness to all students in this large course. Out of a total of approximately 12 homework assignments, the lowest two scores will be dropped.

### Special Questions:

- *Shortcut questions:* Short questions are usually easy questions that give you opportunities to practice basic materials. However, we understand that some of you are very familiar with the topics and do not want to spend too much time on easy questions. Therefore, we design shortcut questions for this purpose. A shortcut question usually has multiple parts that build upon each other and are ordered by their difficulty level. You can work on those in order or start from wherever you like. However you only need to submit the last part you are able to solve. For example, if a question has 5 parts (a, b, c, d, e), you are confident about part e, you should submit part e without any of the previous four parts. If you are confident about d but not sure about e, you should submit d for grading purposes. Please clearly indicate in your submission which part you are submitting.
- *Redemption questions:* It is important that you carefully read the posted solutions, even for problems you got right. To encourage this, you have the option of submitting a redemption file, a few paragraphs in which you explain, for each problem you choose to cover, what you did wrong and what the right idea was in your own words (not cutting and pasting from the solution!), and appending it to your homework. For example, suppose that as you review your solutions to HW1, you realize you had misunderstood question 3 and answered it incorrectly. You would write down what you just learned, and then submit it in your HW2 assignment the following week. Because these are mainly for your benefit, feel free to format them however is most useful for you.
- *Extra credit questions:* We might have some extra credit questions in the homework for people who really enjoy the materials. However, please note that you should do the extra credit problems only if you really enjoy working on these problems and want an extra challenge. It is likely not the most efficient manner in which to maximize your score.

Due Wednesday, November 15, at 4:59pm

**0. Who did you work with?**

List all your collaborators on this homework. If you have no collaborators, please list “none”.

**Solution:** N/A

## 1. (★ level) Basic Complexity Concepts

- a. Suppose we reduce a problem  $A$  to another problem  $B$ . This means that if we have an algorithm that solves \_\_\_\_, we immediately have an algorithm that solves \_\_\_\_.

**Solution:** B, A.

- b. Again, suppose we show a (polynomial-time) reduction from  $A$  to  $B$ . This implies that up to polynomial factors, \_\_\_\_ cannot be any easier than \_\_\_\_.

**Solution:** B, A.

- c. Define the class **NP** in one sentence. *There are several correct characterizations. One of them is the most intuitive, and the definition we use in this class. Give this definition.*

**Solution:** The class of problems for which we can verify any proposed solution in polynomial time.  
Another acceptable answer: "The class of search problems."

- d. Define the class **NP-hard** in one sentence.

**Solution:** The class of problems to which all problems in **NP** reduce.

- e. Define the class **NP-complete** in one sentence.

**Solution:** The set of problems that are **NP-hard** and also in **NP**.

"The set of search problems to which all search problems reduce" is also acceptable.

- f. Show that for any problem  $\Pi$  in **NP**, there is an algorithm which solves  $\Pi$  in time  $O(2^{p(n)})$ , where  $n$  is the size of the input instance and  $p(n)$  is a polynomial (which may depend on  $\Pi$ ).

**Solution:** Since  $\Pi \in \mathbf{NP}$ ,  $\exists$  a verifier  $V$  that can verify solutions in  $O(q(n))$  time for some polynomial  $q(\cdot)$ . Since  $V$  must read a solution to verify it,  $V$  can read at most  $q(n)$  input bits. Therefore, we can simply run  $V$  on all binary strings of size  $2^{q(n)}$ . If the instance of  $\Pi$  has a solution, then one of these inputs is a solution. This solves any instance of  $\Pi$  in  $O(q(n)2^{q(n)}) = O(2^{2q(n)}) = O(2^{p(n)})$ , where  $p(n) = 2q(n)$ .

Alternatively, we can reduce  $\Pi$  to an **NP** complete problem that has an obvious exponential solution (i.e. CircuitSAT, 3-SAT, SAT). Because reductions are polynomial time, the CircuitSAT instance in the reduction has size  $q(n)$ , for some polynomial  $q$  where  $n = |\Pi|$ . Therefore, it suffices to show that there exists an algorithm that solves CircuitSAT in time  $O(2^{p_1(n)})$ , where  $p_1$  is a polynomial. If this holds, then through the reduction we will get an algorithm to solve  $\Pi$  in  $O(2^{p(n)})$  time, where  $p(n)$  is the polynomial  $p_1(q(n))$ . Consider any CircuitSAT instance  $I$ . The instance  $I$  can have only a finite number  $k$  of input wires, which are either 0 or 1, so there are  $2^k$  possible inputs. Therefore, we can try all of the  $2^k$  possible inputs to see if the circuit is satisfiable. Let  $n = |I|$  be the size of the circuit, so we know that  $k \leq n$ . Let  $p_2(n)$  be the time it takes to evaluate the circuit on a given input. Trying all possible inputs will take  $O(p_2(n)2^k)$  time. However, since  $p_2(n)$  is a polynomial, and  $k \leq n$ ,  $O(p_2(n)2^k)$  is  $O(2^{2n})$ . Therefore, the brute force algorithm solves CircuitSAT in  $O(2^{2n})$  time, so this gives us an algorithm to solve  $\Pi$  in  $O(2^{2q(n)})$  time.

## 2. (★★ level) Proving NP-completeness by generalization.

Proving **NP**-completeness by generalization. For each of the problems below, prove that it is **NP**-complete by showing that it is a generalization of some **NP**-complete problem we have seen in the lecture or in the book. You only need to show the reduction part in this problem (there is no need to show that the problem is in **NP**).

- a. **SUBGRAPH ISOMORPHISM**: Given as input two undirected graphs  $G$  and  $H$ , determine whether  $G$  is a subgraph of  $H$  (that is, whether by deleting certain vertices and edges of  $H$  we obtain a graph that is, up to renaming of vertices, identical to  $G$ ), and if so, return the corresponding mapping of  $V(G)$  into  $V(H)$ . (If you are confused by the meaning of 'identical up to renaming of vertices', take a look at this: [https://en.wikipedia.org/wiki/Up\\_to](https://en.wikipedia.org/wiki/Up_to))

**Solution:** We can view this as a generalization of the **CLIQUE** problem. Given an input  $(G, k)$  for **CLIQUE**, let  $H$  be a graph consisting of  $k$  vertices with every pair connected by an edge (i.e. a clique of size  $k$ ). Then  $G$  contains a clique of size  $k$  if and only if  $H$  is a subgraph of  $G$ .

- b. **LONGEST PATH**: Given a weighted graph  $G$  and an integer  $g$ , find in  $G$  a simple path of length  $g$ .

**Solution:** This is a generalization of **RUDRATA-PATH**. Given a graph  $G$  with  $n$  vertices, let  $g = n - 1$ . Then  $(G, n - 1)$  is an instance of **LONGEST PATH**. However, a simple path of length  $n - 1$  must contain  $n$  vertices and hence must be a Rudrata path. Also, any Rudrata path is of length  $n - 1$ . Hence for any graph with  $n$  vertices, **LONGEST-PATH** $(G, n - 1)$  is precisely the asking for the Rudrata path.

- c. **MAX SAT**: Given a CNF formula and an integer  $g$ , find a truth assignment that satisfies at least  $g$  clauses.

**Solution:** Given a formula  $\phi$  with  $m$  clauses, setting  $g = m$  gives **SAT** as a special case of **MAX-SAT**.

- d. **DENSE SUBGRAPH**: Given a graph and two integers  $n$  and  $m$ , find a set of  $n$  vertices of  $G$  such that there are at least  $m$  edges between them.

**Solution:** This also a generalization of **CLIQUE**. Given an instance  $(G, k)$  let  $n = k$ ,  $m = k(k - 1)/2$ . Any subgraph of  $G$  with  $k$  vertices and containing  $k(k - 1)/2$  edges, must have an edge between every possible pair out of these  $k$  vertices and hence must be a clique. Similarly, a clique on  $k$  vertices must contain  $k(k - 1)/2$  edges.

- e. **SPARSE SUBGRAPH**: Given a graph and two integers  $a$  and  $b$ , find a set of  $a$  vertices of  $G$  such that there are at most  $b$  edges between them.

**Solution:** This is a generalization of **INDEPENDENT SET**. Given  $(G, k)$ , an instance for independent set, let  $a = k$ ,  $b = 0$ . Then  $(G, a, b)$  is an instance for **SPARSE SUBGRAPH**. Also, any subgraph of  $G$  with  $k$  vertices and 0 edges must be an independent set of size  $k$  and vice-versa.

### 3. (★★★ level) Reduction, Reduction, and Reduction

**Instruction:** In this problem, you will prove three problems are **NP**-Complete. Please be aware that to prove a problem is **NP**-Complete, you need to argue it is in **NP** and then briefly justify a reduction that shows it is **NP**-hard. Keep in mind that for a reduction from  $A$  to  $B$ , you are only allowed to use a black-box for  $B$  once in order to solve  $A$ . You cannot call it multiple times.

1. Alice and Bob go out on a date at a nice restaurant. At the end of the meal, Alice has eaten  $c_A$  dollars worth of food, and has in her wallet a set of bills  $A = \{a_1, a_2, \dots, a_n\}$ . Similarly, Bob owes the restaurant  $c_B$  dollars and has bills  $B = \{b_1, b_2, \dots, b_m\}$ . Now, Alice and Bob are very calculating people, so they agree that each of them should pay their fair share ( $c_A$  and  $c_B$ , respectively).

One thing they don't mind doing, however, is fairly trading bills. That is, Alice can exchange a subset  $A' \subseteq A$  of her bills for a subset  $B' \subseteq B$  of Bob's bills, so long as  $\sum_{a \in A'} a = \sum_{b \in B'} b$ . Under the above conditions, Alice and Bob wish to find, after trading as many times as desired, subsets of their bills  $A^*$ ,  $B^*$  such that  $\sum_{a \in A^*} a = c_A$  and  $\sum_{b \in B^*} b = c_B$ .

Show that FAIR DATE is **NP**-complete.

**Hint:** You may assume SUBSET SUM is **NP**-complete.

**Solution:** We can reduce SUBSET SUM to this problem. Given a weight capacity  $W$  and the item weights  $w_1, \dots, w_n$ , we convert to an instance of this problem as follows:

Alice has the set of paper bills  $\{w_1, \dots, w_n\}$  and owes  $W$  dollars. Her date is an imaginary Bob, who naturally has no bills and eats nothing.

Now Alice and Bob can't trade, and Alice must pay the exact amount  $W$  with  $\{w_1, \dots, w_n\}$ , which is precisely the SUBSET SUM problem.

2. Consider the following problem PUBLIC FUNDS:

You are looking to build a new fence for your mansion, to keep out pesky people protesting profligate purchases. You have  $m$  bank accounts at your disposal to use to pay for your fence; each account  $i$  has a balance of  $b_i$ . You must choose one of  $n$  options for your fence; each fence  $j$  costs  $c_j$  dollars. You would like to withdraw from at most  $k$  of the bank accounts to build the fence, and due to peculiar UC accounting rules, if you use a particular bank account, you must use the whole balance (all  $b_m$  dollars.)

Determine whether it is possible to exactly pay for some fence  $j$ ; that is, whether there is a  $j$  between 1 and  $n$  such that you can withdraw exactly  $c_j$  dollars given the bank account balances  $b_1, \dots, b_m$ , the fence costs  $c_1, \dots, c_n$ , and  $k$ , and if so return the corresponding choice of fence and set of bank accounts that you withdraw from.

Show that PUBLIC FUNDS is **NP**-Complete.

**Solution:** We can trivially verify a solution for PUBLIC FUNDS in polynomial time by verifying that the sum of the balances of the chosen bank accounts adds up exactly to the price of the fence, therefore it is in NP.

To show that it is in NP-Hard, we reduce from SUBSET SUM. Let each of the  $m$  elements in the set of numbers correspond to a bank account's balance  $b_m$ . Let the desired sum  $s$  correspond to the price of the only fence  $c_1$ . Also, set the maximum number of bank accounts used,  $k$ , to the number of elements in the set of numbers  $m$ . If an algorithm for PUBLIC FUNDS finds a solution, we can interpret the bank accounts chosen as a solution for SUBSET SUM. Furthermore, all instances of SUBSET SUM correspond to a valid instance of PUBLIC FUNDS by construction.

One common mistake was to reduce in the wrong direction. To show that PUBLIC FUNDS is NP-Hard, we need to show that it can be used to solve a known NP-Complete problem (with polynomial time modifications), not the other way around.

Another common mistake was to reduce from KNAPSACK WITHOUT REPETITIONS. All NP-Complete problems can theoretically be reduced to each other (through a chain of reductions), but direct reductions from KNAPSACK WITHOUT REPETITIONS don't have a way to handle the weights of the items. Furthermore, you're not allowed to modify problem you're reducing from; some people attempted to modify knapsack to force the weights to be equivalent to the values of the item.

3. Consider the search problem MAX-ACYCLIC-INDUCED-SUBGRAPH:

INPUT: A *directed* graph  $G = (V, E)$ , and a positive integer  $k$ .

OUTPUT: A subset  $S \subseteq V$  of size  $k$  such that the graph  $G_S$  obtained from  $G$  by keeping exactly those edges both whose endpoints are in  $S$  is a DAG.

Show that MAX-ACYCLIC-INDUCED-SUBGRAPH is **NP**-complete.

**Solution:** We reduce from MAX-INDEPENDENT-SET. Let  $(H, k)$  be an instance of MAX-INDEPENDENT-SET, where  $H$  is an undirected graph. Suppose  $H = (V, E)$ . We construct a directed graph  $G = (V, E')$  as follows. For each edge  $\{u, v\}$  in  $H$ , we include in  $E'$  the pair of directed edges  $(u, v)$  and  $(v, u)$ . We then return the instance  $(G, k)$  of MAX-ACYCLIC-INDUCED-SUBGRAPH. Given a solution  $S$  to this instance, we simply return  $S$  as a solution to the original MAX-INDEPENDENT-SET instance. Thus, our reduction is clearly polynomial time.

We now consider correctness. Suppose there was an independent set  $I$  of size  $k$  in  $H$ . Then, there are no edges between vertices in  $I$  in  $H$ , and hence, by construction, the induced subgraph on  $I$  in  $G$  is completely disconnected and trivially a DAG. Thus, there is an induced DAG of size  $k$  in  $G$ .

Now suppose that  $G$  has an induced DAG  $G_S$  of size  $k$ . If  $u \in S$  and  $v \in S$  such that  $\{u, v\} \in E$ , then both the edges  $(u, v)$  and  $(v, u)$  are present in  $G_S$ , and hence it is not a DAG. Thus, we see that  $S$  is an independent set in  $H$ .

4. (★★★ level) **Rad Reduction** You are the captain of a ship, tasked with taking animals across the river. However, not all animals get along – some animals will attack each other. If the  $i$ th animal gets attacked, you have to pay the owner a fee of  $f_i$ . If the  $i$ th animal makes it across the river without getting attacked, you receive a reward of  $r_i$ .

In other words, for each animal  $i$ , your payoff is as follows:

- 0 if you don't take the animal
- $r_i$  if you take the animal and it doesn't get attacked
- $-f_i$  if you take the animal and it gets attacked

You want to know if there's a subset of animals you can take such that your net profit (the sum of the rewards you get minus the sum of the fees you pay) is at least  $k$ .

**Input:**  $[f_1, \dots, f_n]; [r_1, \dots, r_n]; k; E = \{(i, j) : \text{animal } i \text{ attacks animal } j\}$ .

Prove that this problem is NP-complete.

**Main idea:**

**Solution:**

- Reduce from Independent Set; this shows problem is NP-hard.
- In NP: enumerate over animals on ship; for animal  $i$  subtract  $f_i$  if it is attacked by another animal on ship, and add  $r_i$  otherwise.

**Reduction:**

**Solution:** Given an independent sets instance  $(G, k)$ , construct an animal crossing instance with  $|V|$  animals as follows:

$$\begin{aligned} f_i &= \infty \\ r_i &= 1 \\ k' &= k \\ E' &= E \end{aligned} \quad (\text{turn undirected edge } (u, v) \text{ into 2 directed edges } (u \rightarrow v) \text{ and } (v \rightarrow u))$$

Remark: instead of  $f_i = \infty$  one could set  $f_i$  to any non-negative value.

**Proof:**

**Solution:**

- If there's an independent sets solution  $S$  of size  $\geq k$ , take those same animals. You have  $\geq k$  of them, so your reward is at least  $k$ . None attack each other because there are no edges between them.
- If there's an animal crossing solution  $S$ , it can't contain any animals that attack each other because otherwise the total profit would be  $-\infty$ . At least  $k$  animals must cross because all the rewards are 1. Thus, we have  $\geq k$  animals that don't attack each other, so those same  $\geq k$  vertices have no edges between them.

**5. (??? level) (Optional) Redemption for Homework 8**

Submit your *redemption file* for Homework 8 on Gradescope. If you looked at the solutions and took notes on what you learned or what you got wrong, include them in the redemption file.

**Solution:** N/A