CSC373 Summer '22
Assignment 4

Due Date: August 13, 2022, 11:59pm ET

Instructions

1. Typed assignments are preferred (e.g., PDFs created using LaTeX or Word), especially if your handwriting is possibly illegible or if you do not have access to a good quality scanner. Either way, you need to submit a single PDF named "hwk4.pdf" on MarkUS at `https://markus.teach.cs.toronto.edu/2022-05`

2. You will receive 20% of the points for a (sub)question when you leave it blank (or cross off any written solution) and write "I do not know how to approach this problem." If you leave it blank but do not write this or a similar statement, you will receive 10%. This does not apply to any bonus (sub)questions.

3. You may receive partial credit for the work that is clearly on the right track. But if your answer is largely irrelevant, you will receive 0 points.

Q1 [15 Points] Complement of an NP Problem

For a decision problem D_0, define $\overline{D_0}$ to be its complement problem i.e., a YES instance of D_0 is a NO instance of $\overline{D_0}$ and a NO instance of D_0 is a YES instance of $\overline{D_0}$. Below, "$\leq_K$" denotes a *Karp* reduction as defined in class (i.e., a polynomial time reduction that only produces a single instance and preserves the YES/NO value).

(a) [7.5 Points] Give a decision problem D_0 such that $D_0 \leq_K \overline{D_0}$.
Define D_0 precisely, describe the reduction $D_0 \leq_K \overline{D_0}$ in detail, and prove that your reduction is correct.

(b) [2.5 Points] Does your decision problem D_0 from part (a) belong to NP? Justify.

Solution to Q1

(a) Let D_0 = "On input $x \in \mathbb{N}$, output TRUE iff x is even."
Then, $\overline{D_0}$ = "On input $x \in \mathbb{N}$, output TRUE iff x is odd."
Also, $D_0 \leq_K \overline{D_0}$ with the reduction function $f(x) = x + 1$:

- clearly, $f(x)$ is computable in polytime;
- also, x is even iff $f(x) = x + 1$ is odd, i.e., $x \in D_0$ iff $f(x) \in \overline{D_0}$.

(b) Yes, $D_0 \in$ NP because $D_0 \in$ P $\subseteq$ NP.

Q2 [15 Points] Exponential Algorithms for NP Problems

Show that for every decision problem $D \in$ NP, there is some polynomial $p(n)$ and some algorithm A (both of which depend on D) such that A solves D in worst-case time $O\big(2^{p(n)}\big)$.
HINT: This involves mostly "unrolling" the definitions, so write up your answer carefully. In your answer, you must use the formal definition of NP in terms of polynomial-time *verifiers* – for this, you can refer to class slides or page 1064 of the text.

Solution to Q2

Let D be a decision problem in NP. For any input x, we write "$D(x)$" to denote the answer (TRUE or FALSE) of problem D on input x. Then by definition (see page 1064 of the text), there is a two-input polynomial-time algorithm A and constants c and d such that for all $x \in \{0,1\}^*$

$$D(x) \Leftrightarrow \text{there exists a certificate } y \text{ with } |y| \leq d|x|^c \text{ such that } A(x,y) = \textsc{True} \tag{1}$$

Thus for each input x of length n, the total number of possible strings that are potential certificates is the number of binary strings of length at most dn^c, which is at most 2^{dn^c}.
Here is an algorithm $A'(x)$ for solving decision problem $D(x)$:
Algorithm $A'(x)$:

$n \leftarrow |x|$
for every string y of length at most dn^c **do**
  if $A(x,y) = 1$ **then**
    return TRUE # and exit
  end if
end for
return FALSE

It is immediate from (1) that $A'(x) = D(x)$ for all inputs x.
We can upper bound the runtime of A' as follows. The number of iterations of the loop is at most the number of strings y of length at most dn^c, which is at most 2^{dn^c} (see above). The time required for each iteration is dominated by the time required to compute $A(x,y)$, where $|y| \leq dn^c$. Since the composition of two polynomials is again a polynomial, the time required to compute $A(x,y)$ is bounded by some polynomial $q(n)$. Thus the total time required for $A'(x)$ is $O\big(q(n)2^{dn^c}\big) = O\big(2^{p(n)}\big)$ for some polynomial $p(n)$, as required.

Q3 [20 Points] Showing NP-Completeness I

A web server has a number of simultaneous "requests" to reply to. The server has to send its replies in "packets," each one of which has a fixed positive integer size limit L. Each packet can contain more than one reply (so multiple requests can be replied to in a single packet), but each individual reply has its own positive integer size s_i. We would like to use as few packets as possible to send all of the replies.
This problem can be formulated formally as a decision problem FEWPACKETS ("FP" for short), as follows:

Input: Positive integer *packet size limit* L, positive integers *reply sizes* $s_1, \ldots, s_n$, positive integer *number of packets* k.

Output: Is there some partition of $\{1, \ldots, n\}$ into packets $P_1, \ldots, P_k$, where each packet has size at most L– formally: $\exists P_1, \ldots, P_k, P_1 \cup \cdots \cup P_k = \{1, \ldots, n\} \wedge (\forall i, j, P_i \cap P_j = \emptyset) \wedge \forall i, \sum_{j \in P_i} s_j \leq L$?

For example, $(8, \{2, 5, 4, 5\}, 3) \in$ FP because we can use packets $P_1 = \{1, 3\}$ (with size $s_1 + s_3 = 2 + 4 \leq 8$), $P_2 = \{2\}$ (with size $s_2 = 5 \leq 8$), and $P_3 = \{4\}$ (with size $s_4 = 5 \leq 8$). However, $(8, \{2, 5, 4, 5\}, 2) \notin$ FP because there is no way to partition every reply into only 2 packets, each one of size ≤ 8 – even though the total size of all replies is only 16.
Write a *detailed* proof that FP is NP-complete: first show that FP $\in$ NP, and then use a reduction from Subset-Sum to show that FP is NP-hard. You may assume that the following "positive integer" version of Subset-Sum is NP-complete:

Input: Positive integer t (target sum) and a set S of n positive integers $\{x_1, \ldots, x_n\}$.

Output: Is there some subset $\emptyset \neq S' \subseteq S$ that sums to exactly t?

Solution to Q3

First we show that FP $\in NP$. Given a YES instance $(L, s_1, \ldots, s_n, k)$ of FP, a certificate is a partition $P_1, \ldots, P_k$ of $\{1, \ldots, n\}$ satisfying the conditions specified in the definition of FP. A verifier can easily check in polynomial time that the certificate satisfies all of these conditions.
To show that FP is NP-hard, we show that Subset-Sum $\leq_p$ FP. (Here, we let Subset-Sum refer to the version in the problem statement, which is given to be NP-hard.)
Our task is to give a poly-time algorithm which transforms a sequence $I = (x_1, \ldots, x_\ell, t)$ of positive integers, describing an instance of Subset-Sum, to an instance $I' = (L, s_1, \ldots, s_n, k)$ of FP such that I is a YES instance of Subset-Sum iff I' is a YES instance of FP.
Let $u = x_1 + x_2 + \cdots + x_\ell$.

Case 1: If $u = 2t$, then let $L = t, k = 2, n = \ell$, and $(s_1, \ldots, s_n) = (x_1, \ldots, x_\ell)$.

Case 2: If $u < 2t$, then let $L = t, k = 2, n = \ell + 1$, and $(s_1, \ldots, s_n) = (x_1, \ldots, x_\ell, x_{\ell+1})$, where $x_{\ell+1} = 2t - u$.

Case 3: If $u > 2t$, then let $L = u - t, k = 2, n = \ell + 1$, and $(s_1, \ldots, s_n) = (x_1, \ldots, x_\ell, x_{\ell+1})$, where $x_{\ell+1} = 2(u - t) - u = u - 2t$.

Clearly, $L, s_1, \ldots, s_n, k$ can be computed in poly-time from $x_1, \ldots, x_\ell, t$ because all arithmetic comparisons and operations can be performed in poly-time for binary integers.
Suppose I is a YES instance of Subset-Sum. Then there exists $S \subseteq \{1, \ldots, \ell\}$ such that $\sum_{i \in S} x_i = t$.

Case 1: If $u = 2t$, then $L = t$. Let $P_1 = S$ and $P_2 = \{1, \ldots, n\} - S$. Then $\sum_{i \in P_1} s_i = \sum_{i \in S} x_i = t \leq L$ and $\sum_{i \in P_2} s_i = \sum_{i=1}^{\ell} x_i - \sum_{i \in S} x_i = u - t = t \leq L$.

Case 2: If $u < 2t$, then $L = t$. Let $P_1 = S$ and $P_2 = \{1, \ldots, n\} - S$. Then $\sum_{i \in P_1} s_i = \sum_{i \in S} x_i = t \leq L$ and $\sum_{i \in P_2} s_i = \sum_{i=1}^{\ell+1} x_i - \sum_{i \in S} x_i = x_{\ell+1} + u - t = (2t - u) + u - t = t \leq L$.

Case 3: If $u > 2t$, then $L = u - t$. Let $P_1 = \{1, \ldots, \ell\} - S$ and $P_2 = S \cup \{\ell + 1\}$. Then $\sum_{i \in P_1} x_i = u - t \leq L$ and $\sum_{i \in P_2} x_i = \left(\sum_{i \in S} x_1\right) + x_{\ell+1} = t + (u - 2t) = u - t \leq L$.

In all cases, I' is a YES instance of FP, as required.
Conversely, suppose that I' is a YES instance of FP. Then there exists a partition P_1, P_2 of $\{1, \ldots, n\}$ such that $\sum_{i \in P_1} x_i \leq L$ and $\sum_{i \in P_2} x_i \leq L$.

Case 1: If $u = 2t$, then $L = t$ and $\sum_{i \in P_1} x_i + \sum_{i \in P_2} x_i = x_1 + \cdots + x_\ell = u = 2t = 2L$, so both P_1 and P_2 must have sum *exactly* equal to L. Let $S = P_1$. Then $\sum_{i \in S} x_i = \sum_{i \in P_1} x_i = t$.

Case 2: If $u < 2t$, then $L = t$ and $\sum_{i \in P_1} x_i + \sum_{i \in P_2} x_i = x_1 + \cdots + x_\ell + x_{\ell+1} = u + (2t - u) = 2t = 2L$, so both P_1 and P_2 must have sum *exactly* equal to L. Let $S = P_1$ or $S = P_2$, whichever set does *not* contain $x_{\ell+1}$. Then $\sum_{i \in S} x_i = L = t$.

Case 3: If $u > 2t$, then $L = u - t$ and $\sum_{i \in P_1} x_i + \sum_{i \in P_2} x_i = x_1 + \cdots + x_\ell + x_{\ell+1} = u + (u - 2t) = 2(u - t) = 2L$, so both P_1 and P_2 must have sum *exactly* equal to L. Let $S = P_1 - \{x_{\ell+1}\}$ or $S = P_2 - \{x_{\ell+1}\}$ (depending on which packet contains $x_{\ell+1}$). Then $\sum_{i \in S} x_i = L - (u - 2t) = (u - t) - (u - 2t) = t$.

In all cases, I is a YES instance of SUBSET-SUM, as required.
(NOTE: Many other reductions were possible. For example, we could set $L = 2u$, $s_1, \ldots, s_n = x_1, \ldots, x_\ell, (u + t), (2u - t)$, and $k = 2$.)

Q4 [20 Points] Showing NP-Completeness II

Imagine that we have a network of processors, each one of which needs access to a common database in order to carry out its work (for example, a network of banking machines accessing client information). To simplify, assume all operations are queries (no operation changes the data). Our problem is to determine which processor(s) should store the database (these processors will be called "servers") in order to have reasonably fast access time (including the load on each server), as well as having as little duplication of the database as possible.
At one extreme, each processor could store its own copy of the database. This would ensure the fastest possible access time, but at the cost of replicating the database many times. At the other extreme, the database could be stored in a single server (all other processors would connect to the server to access the data). This would ensure the smallest amount of duplication, but at the cost of longer access times and server load. In either case, there is a significant cost if the network is large.
A reasonable middle ground between the two extremes would be to fix a distance parameter d and to store the database on as few servers as possible, chosen so that every processor is within distance d of at least one server (the distance is measured as the minimum number of links required to send communications from one processor to the next). Unfortunately, it seems impossible to solve this problem efficiently.
Formally, show that the following SERVERLOCATION problem is NP-complete. You may use any NP-complete problem from class for your reduction.

Input: An undirected graph G (the network of processors) and non-negative integers k and d.

Output: Is there some subset S of no more than k vertices (the "servers") such that every processor is within distance d of some server in S? (The "distance" between vertices u, v is measured

as the number of edges on a shortest path from u to v, or infinity (∞) if there is no path from u to v.)

Solution to Q4

ServerLocation $\in NP$: Given $(G = (V, E), k, d)$, a certificate (or advice) can simply be a subset $C \subseteq V$: then it takes poly-time to verify that $|C| \leq k$, and that every vertex in G is within distance d of some element of C (by running BFS from each vertex in G).

ServerLocation is NP-hard: We show VertexCover $\leq_p$ ServerLocation.

Main Idea: Create G' by taking $G = (V, E)$ and adding an extra vertex for each edge $e \in E$, connected to both endpoints of e — effectively turning each edge of G into a triangle. Keep k the same and set $d = 1$.

This **almost** works, but not quite! There are two technical details that require special handling:

- If G contains isolated vertices, they may or may not belong to a vertex cover of size k, but they must be servers.

  We'll handle this by actually removing all isolated vertices from G'.
- If any of the new, extra vertices belong to a server set of size k in G', how do we know there is a corresponding vertex cover that contains only "old" vertices (those of G)?

  We'll handle this by showing that old vertices can be swapped for new ones in any solution — see below for the details.

Details:

On input (G, k), where $G = (V, E)$:

- Let $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$. Furthermore, let $V_0 \subseteq V$ be the set of every isolated vertex of G, i.e., the vertices of G that have no edge at all attached to them.
- Let $V' = V - V_0 \cup \{x_1, \ldots, x_m\}$, i.e., remove all isolated vertices and add one new vertex for every edge in G.
- Let $E' = E \cup \{(x_i, v_i), (x_i, u_i) : e_i = (u_i, v_i) \in E\}$, i.e., for each edge e_i of G, add edges from each endpoint of e_i to the new vertex x_i (turning e_i into a triangle in G').
- If $|V - V_0| < k$, then output $(G' = (V', E'), |V - V_0|, 1)$; else output $(G' = (V', E'), k, 1)$.

Clearly, this algorithm runs in polytime (as a function of n, m).

Moreover, if G contains fewer than k non-isolated vertices (those in $V - V_0$), then those vertices alone form a vertex cover in G and a server set of size $|V - V_0|$ in G'. Otherwise, if G contains a vertex cover C of size k, then every vertex of G' (including the new vertices x_i) is within distance 1 of a vertex in C — because C covers every edge of G and G' contains

no isolated vertex. Starting with $S = C$ and replacing every isolated vertex in S by some non-isolated vertex outside S yields a server set of size k in G'.

Finally, if G' contains a server set of size $|V - V_0|$, then G contains fewer than k non-isolated vertices (in $V - V_0$) and, together with a sufficient number of isolated vertices, these form a vertex cover in G. Otherwise, if G' contains a server set S of size k, then there is some server set S' of size k such that S' contains no "new" vertex x_i: simply replace each x_i in S with u_i or v_i (one of the endpoints of edge e_i). If both u_i and v_i already belong to S', then replace x_i with any other vertex outside S'. But then, S' is a vertex cover in G: since every x_i is outside S', it must be connected to some vertex in S', which means every edge e_i of G has at least one endpoint in S'.

Alternate reduction: It was also possible to show that Exact 3SAT $\leq_p$ ServerLocation.

Main Idea: From an Exact 3CNF A, create a graph G with one node C_j for each clause C_j in A and one triangle $x_i — x_i' — \neg x_i$ for each variable x_i of A. Then, connect each C_j to all three literals that appear in C_j, and set $d = 1$ and $k =$ number of variables of A.

- This can be done in poly-time.
- Every satisfying assignment of values to the variables of A yields a server set that consists of the nodes that correspond to true literals.
- Every server set in G must include at least one node from each triangle (otherwise the node x_i' would neither be a server nor connected to one), so every server set of size k contains exactly one node from each triangle. If variables are set according to the nodes in the server set (picking values arbitrarily when x_i' is in the server set), then F will be satisfied because each clause is "covered".