This homework has a total of 3 problems on 4 pages. Solutions should be submitted to GradeScope before 3:00pm on Monday Dec 4.

The problem set is marked out of 20, you can earn up to $21 = 1 + 8 + 7 + 5$ points.

If you choose not to submit a typed write-up, please write neat and legibly.

Collaboration is allowed/encouraged on problems, however each student must independently complete their own write-up, and list all collaborators. No credit will be given to solutions obtained verbatim from the Internet or other sources.

Modifications since version 0

1. 1 (changed in version 3): $\neg x$ (instead of $\bar{x}$) is now used to represent the negation of $x$.

2. 1c: (changed in version 2) rewored to showing NP-hardness.

3. 2b: (changed in version 1) added a hint.

4. 3b: (changed in version 1) added comment about the two loops’ $u$ variables being different due to scoping.

0. [1 point, only if all parts are completed]

(a) Submit your homework to Gradescope.

(b) Student id is the same as on T-square: it’s the one with alphabet + digits, NOT the 9 digit number from student card.

(c) Pages for each question are separated correctly.

(d) Words on the scan are clearly readable.

1. (8 points) NP-Completeness

Recall that the SAT problem, or the Boolean Satisfiability problem, is defined as follows:

- Input: A CNF formula $F$ having $m$ clauses in $n$ variables $x_1, x_2, \ldots, x_n$. There is no restriction on the number of variables in each clause.

- Output: YES if there is an assignment to the variables which satisfies all $m$ clauses, and NO otherwise.

Now consider the ALMOST-SAT problem which is defined below:

\begin{itemize}
\item \text{Input: } A \text{ CNF formula } F \text{ having } m \text{ clauses in } n \text{ variables } x_1, x_2, \ldots, x_n. \text{ There is no restriction on the number of variables in each clause.}
\item \text{Output: } \text{YES} \text{ if there is an assignment to the variables which satisfies all } m \text{ clauses, and } \text{NO} \text{ otherwise.}
\end{itemize}
• Input: A CNF formula \( F \) having \( m \) clauses in \( n \) variables \( x_1, x_2, \ldots, x_n \). There is no restriction on the number of variables in each clause.
• Output: YES if there is an assignment to the variables which satisfies \( \text{exactly } m - 1 \) clauses, and NO otherwise.

For example, for the formula
\[
F = (x_1 \lor x_2) \land (x_1 \lor \neg x_3 \lor \neg x_4 \lor x_5) \land (\neg x_2) \land (x_2 \lor x_5),
\]
one assignment to the variables which satisfies all four clauses is
\[
x_1 = T, x_2 = F, x_3 = F, x_4 = T, x_5 = T,
\]
so the expected output of SAT on this example will be YES.
This formula also has an assignment which satisfies exactly 3 of the 4 clauses
\[
x_1 = T, x_2 = T, x_3 = F, x_4 = T, x_5 = T.
\]
So the output of Almost-SAT on \( F \) should also be YES.

The goal of this problem is to show that Almost-SAT is NP-complete.

(a) (2 points) We will first prove that Almost-SAT is in NP. A problem is said to be in NP if a solution to the problem can be verified in polynomial time. Give a polynomial time algorithm which takes as input an assignment to each variable, and verifies whether it is a solution to Almost-SAT or not.

(b) (3 points) Let \( F \) be a boolean formula having \( m \) clauses in variables \( x_1, x_2, \ldots, x_n \). Construct a new formula \( F' \) by taking the conjunction of \( F \) with two new clauses (i.e. \( F' \) should have \( m + 2 \) clauses) such that an assignment of \( x_1, \ldots, x_n \) which satisfies all \( m \) clauses in \( F \) will satisfy exactly \( m + 1 \) clauses in \( F' \). Explain what your two new clauses should be. There is no restriction on the length of the two clauses, and you can use one or more of the variables from \( F \) to construct them.

(c) (3 points) To complete the proof of NP-completeness, we need to show that Almost-SAT is NP-hard.
To do this, we take SAT, a known NP-hard problem, and reduce it to an instance of Almost-SAT. Let \( F \) be the boolean formula input to SAT. Let \( F' \) be the formula obtained from \( F \) using the construction in Part (b). We will use \( F' \) as our input to Almost-SAT. Specifically, show that if \( F \) has an assignment which satisfies SAT, \( F' \) has an assignment which satisfies Almost-SAT; and if \( F \) has no assignment which satisfies SAT, \( F' \) has no assignment which satisfies Almost-SAT. Argue that the construction of \( F' \) takes polynomial time.
2. (7 points) Hardness and Approximation of Maximum 2-SAT

Consider a 2-SAT instance: there are \( n \) Boolean variables \( x_1 \ldots x_n \) and \( m \) clauses, each of the form of

\[
l_{ij} \lor l_{i2},
\]

where \( l_{ij} \) could be \( x_r \) or \( \neg x_r \) for some variable \( x_r \).

The goal of maximum 2-SAT is to find an assignment to the \( x_r \)s that satisfies the maximum number of these clauses.

(a) (1 points) Give a decision version of maximum 2-SAT.

(b) (3 points) We will show that maximum 2-SAT is NP-complete by showing that

\[
\text{Max-Cut} \rightarrow \text{Max-2-SAT}.
\]

Here \( \text{Max-2-SAT} \) is the decision version of the maximum 2-SAT problem that you defined in part a). The definition of the deterministic version of \( \text{Max-Cut} \) (which you may assume is NP-complete) is

**Definition 0.1** (Decision Version of \( \text{Max-Cut} \)). Given an undirected unweighted graph \( G \) and a parameter \( k \), answer whether there is a cut \( S \subseteq V(G) \) so that the number of edges leaving \( S \) is at least \( k \).

**HINT**: first figure out the following reduction involving a single edge: for two Boolean variables \( x \) and \( y \), give two SAT clauses over them so that they are both satisfied when \( x \neq y \), and only one of them is satisfied when \( x = y \).

(c) (1 points) Conclude from parts a) and b) that \( \text{Max-2-SAT} \) is NP-Complete.

(d) (2 points) Give a 2-approximation algorithm for the optimization version of maximum 2-SAT. Your algorithm should run in \( O(m) \) time.

3. (5 points) Approximating Graph Diameter

Define the distance between a pair of vertices \( x \) and \( y \), \( \text{DIST}(x, y) \) to be the length of the shortest path between them. The diameter of a graph is then the maximum distance between a pair of vertices

\[
D = \max_{u,v} \text{DIST}(u, v).
\]

In this problem we will give an \( O(m) \) time 2-approximation algorithm to the diameter of an undirected, unit weight graph with \( n \) vertices and \( m \) edges. That is, we will try to produce a pair of vertices, \( x, y \), such that

\[
\text{DIST}(x, y) \geq \frac{1}{2} D.
\]
For the purpose of this problem, you may use the fact that on any weighted graph with positive edge weights, for any vertex \( s \), breadth first search (BFS) computes in \( O(m) \) time the values of \( \text{Dist}(s, u) \) for all vertices \( u \). (we did not cover BFS in this class, but you should only use it as a black-box here)

(a) (1 point) Show using the given running time of BFS that we can compute exactly the diameter of a graph in \( O(nm) \) time.

(b) (2 points) Use the fact that distances obey the triangle inequality, aka

\[
\text{Dist}(u, v) \leq \text{Dist}(u, w) + \text{Dist}(v, w)
\]

for any vertices \( u, v, w \) to show that for any vertex \( s \),

\[
D \leq 2 \cdot \max_u \text{Dist}(s, u).
\]

(note that ‘iteration-like’ operators such as max or summation work similar to for loops: the \( u \) on the RHS is different than the \( u \) in the definition of \( D \))

(c) (2 points) Use part b) to give an algorithm that on a connected graph with \( m \) edges, produces an 2-approximation of the diameter in \( O(m) \) time. Specifically, it should output in \( O(m) \) time a pair of vertices, \( x, y \) such that

\[
\text{Dist}(x, y) \geq \frac{1}{2} D.
\]