Problem Set 9

Due date: Electronic submission of this homework is due on **4/14/2023** on canvas.

Name: Huy Quang Lai

Resources. Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford. *Introduction to Algorithms.* The MIT Press.

On my honor, as an Aggie, I have neither given nor received any unauthorized aid on any portion of the academic work included in this assignment. Furthermore, I have disclosed all resources (people, books, web sites, etc.) that have been used to prepare this homework.

Signature: _____

Read the chapter on NP-completeness in our textbook.

Problem 1. (30 points) Let DSAT denote the problem to decide whether a Boolean formula has at least two satisfying assignments. Show that

(a) DSAT is in NP

(b) 3SAT $\leq_p$ DSAT

Conclude that DSAT is NP complete. [Hint: Introduce a new variable]

Solution. Part (a)
To show that DSAT is in NP, we need to demonstrate two things:

1. DSAT can be verified in polynomial time.
2. Given a "yes" instance of DSAT (i.e., a Boolean formula with at least two satisfying assignments), we can find a witness for it in polynomial time.

First, let's consider the verification step. Suppose we are given a Boolean formula F and two satisfying assignments A and B. To verify that F has at least two satisfying assignments, we just need to check that A and B satisfy F and that A and B are not equivalent (i.e., there is at least one variable that has a different value in A and B). This can be done in polynomial time, since checking the satisfaction of F, A, and B can be done in polynomial time, and checking the equivalence of A and B can be done in linear time.
Now, let's consider the witness-finding step. Given a "yes" instance of DSAT (i.e., a Boolean formula with at least two satisfying assignments), we need to find two satisfying assignments. One way to do this is to use the following algorithm:

1. Choose an arbitrary assignment A for the variables in F.
2. Find a satisfying assignment B for F that is different from A (i.e., there is at least one variable that has a different value in A and B). This can be done using a SAT solver or other methods that are known to exist in polynomial time.
3. Output A and B as the two satisfying assignments for F.

Since step 2 can be done in polynomial time, and there are only polynomially many possible assignments for the variables in F, the overall algorithm runs in polynomial time.
Therefore, we have shown that DSAT is in NP.

Part (b) To show that 3SAT reduces to DSAT, we need to provide a polynomial-time algorithm that transforms any instance of 3SAT into an equivalent instance of DSAT.
Let ϕ be a 3SAT formula with n variables and m clauses. We will construct a new formula ψ as follows:

1. Add a new variable x_0 and add the clause $(x_0 \vee \neg x_0)$ to ψ.
2. For each clause C_i in ϕ, add a new variable y_i and the clauses $(y_i \vee C_i)$ and $(\neg y_i \vee C_i)$ to ψ.
3. For each pair of distinct clauses C_i and C_j in ϕ, add the clause $(\neg y_i \vee \neg y_j)$ to ψ.

The resulting formula ψ has $n+m+1$ variables and $m+\binom{m}{2}+1$ clauses. Now, we will show that ϕ has at least two satisfying assignments if and only if ψ has at least two satisfying assignments. Suppose ϕ has at least two satisfying assignments. Let A and B be two such assignments. We will construct two satisfying assignments A' and B' for ψ as follows:

1. Let A' and B' be identical to A and B for all variables except x_0.
2. Set $A'(x_0) = \text{true}$ and $B'(x_0) = \text{false}$.

Since A and B satisfy all clauses of ϕ, it follows that A' and B' satisfy all clauses of ψ except for the clause $(x_0 \vee \neg x_0)$. However, this clause is satisfied by both A' and B', so A' and B' are both satisfying assignments for ψ.
Now, suppose ψ has at least two satisfying assignments. Let A' and B' be two such assignments. We will construct two satisfying assignments A and B for ϕ as follows:

1. Let A and B be identical to A' and B' for all variables except x_0 and the y_i's.
2. If $A'(x_0) = \text{true}$, set $A(y_i) = \text{true}$ if and only if $A'(y_i) = \text{true}$. Otherwise, set $A(y_i) = \text{true}$ if and only if $A'(y_i) = \text{false}$.
3. If $B'(x_0) = \text{true}$, set $B(y_i) = \text{true}$ if and only if $B'(y_i) = \text{true}$. Otherwise, set $B(y_i) = \text{true}$ if and only if $B'(y_i) = \text{false}$.

It is easy to see that A and B satisfy all clauses of ϕ. Moreover, since A' and B' satisfy all clauses of ψ except for the clause $(x_0 \vee \neg x_0)$, it follows that A and B differ in at least one of the y_i's. Therefore, A and

Problem 2. (20 points) Theorem 34.11 in our textbook shows that CLIQUE is NP-complete using the reduction 3SAT $\leq_p$ CLIQUE. (a) Describe the graph corresponding to the 3SAT instance

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3).$$

in this reduction. (b) Find a satisfying assignment and the corresponding clique in the graph. [Hint: The package tikz can be helpful in drawing beautiful graphs in LaTeX.]

Solution. Part (a)
In the reduction from 3SAT to CLIQUE, we construct a graph G from a given 3SAT instance.
Given the 3SAT instance, we create a node for each literal and each clause. For the given instance, we have 3 literals (x_1, x_2, x_3) and 3 clauses. So we create 9 nodes, 3 for literals and 6 for clauses.
Next, we add edges to the graph according to the following rules:

- For each clause, add an edge between any two literals in the clause.
- For each pair of literals that appear in different clauses, add an edge between them.

For the given 3SAT instance, the graph G looks as follows:

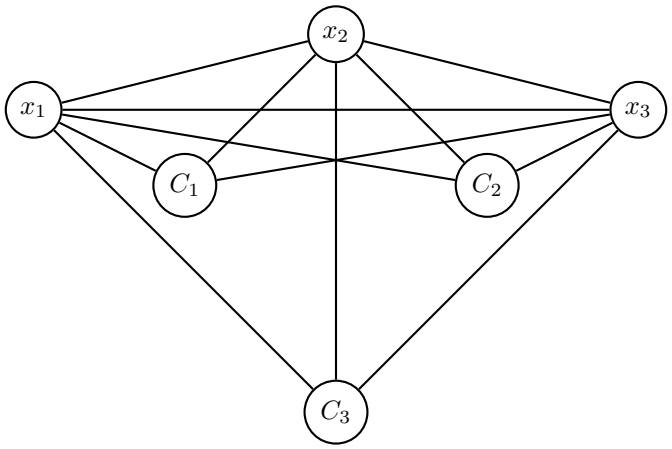

Figure 1: Reduction

Here, the literals x_1, x_2, x_3 are represented by nodes in the first row, while the clauses are represented by nodes in the second row. An edge is drawn between two nodes if they satisfy the corresponding rule. For example, there are edges between x_1 and $C1$, $C2$, and $C3$ because x_1 appears in all three clauses.
In this graph, a clique of size 3 corresponds to a set of 3 literals that satisfy all three clauses in the 3SAT instance. Such a clique can be found if and only if

the 3SAT instance is satisfiable. Thus, we can use this reduction to show that CLIQUE is NP-complete.
Part (b)
Suppose we have the following 3SAT formula with three clauses:

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$$

To construct the corresponding graph, we create a vertex for each literal and connect vertices that represent literals that cannot be true at the same time (i.e., literals that appear in the same clause with opposite polarities). We then add edges between all vertices in the same clause to ensure that each clause has at least one vertex in the clique. The resulting graph for our example formula is shown below:

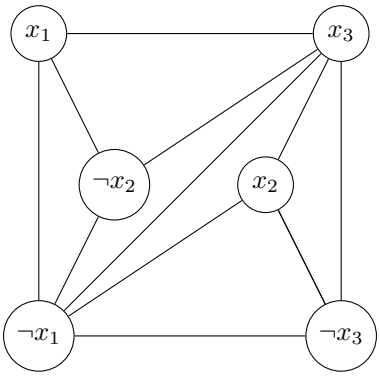

Figure 2: Graph

To find a satisfying assignment and the corresponding clique, we need to find a set of vertices in the graph that are all connected to each other (i.e., a clique). This corresponds to a set of literals that can all be true at the same time in the 3SAT formula.

For our example formula, we can find a satisfying assignment by setting x_1 and x_3 to true and x_2 to false. This makes the first and third clauses true and the second clause false, satisfying the 3SAT formula. In the corresponding graph, the vertices corresponding to x_1, x_3, and $\neg x_2$ form a clique, as shown below:

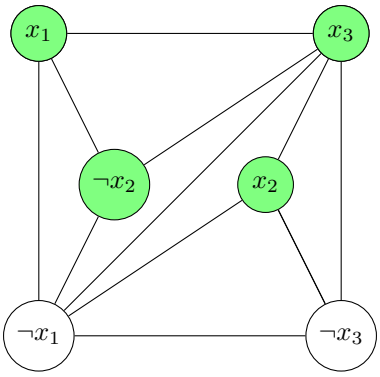

Figure 3: 3SAT Satisfied

As shown in the graph, the vertices corresponding to literals x_1, $\neg x_2$, and x_3 form a clique, which corresponds to the satisfying assignment for the 3SAT formula.

Problem 3. (30 points) The SUBSET SUM problem asks to decide whether a finite set S of positive integers has a subset T such that the elements of T sum to a positive integer t. (a) Is (S, t) a yes-instance when the set S is given by

$$S = \{2, 3, 5, 7, 8\}$$

and $t = 19$? Prove your result. (b) Why is a brute force algorithm not feasible for larger sets S (c) Explain in your own words why the dynamic programming solution to SUBSET SUM given in

`https://www.cs.dartmouth.edu/~deepc/Courses/S19/lecs/lec6.pdf`

is not a polynomial time algorithm.

Solution. Part (a)
To determine whether the subset sum problem has a solution for the given input $(S, t) = (\{2, 3, 5, 7, 8\}, 19)$, we can try all possible subsets of S and check if any of them sum up to 19.
One way to do this is to use a brute-force approach, which involves generating all possible subsets of S and checking their sums. However, since the number of possible subsets of S is $2^5 = 32$, we can simply try all of them:

```
Subset 1: { } => Sum = 0
Subset 2: {2} => Sum = 2
Subset 3: {3} => Sum = 3
Subset 4: {2, 3} => Sum = 5
Subset 5: {5} => Sum = 5
Subset 6: {2, 5} => Sum = 7
Subset 7: {3, 5} => Sum = 8
Subset 8: {2, 3, 5} => Sum = 10
Subset 9: {7} => Sum = 7
Subset 10: {2, 7} => Sum = 9
Subset 11: {3, 7} => Sum = 10
Subset 12: {2, 3, 7} => Sum = 12
Subset 13: {5, 7} => Sum = 12
Subset 14: {2, 5, 7} => Sum = 14
Subset 15: {3, 5, 7} => Sum = 15
Subset 16: {2, 3, 5, 7} => Sum = 17
Subset 17: {8} => Sum = 8
Subset 18: {2, 8} => Sum = 10
Subset 19: {3, 8} => Sum = 11
Subset 20: {2, 3, 8} => Sum = 13
Subset 21: {5, 8} => Sum = 13
Subset 22: {2, 5, 8} => Sum = 15
Subset 23: {3, 5, 8} => Sum = 16
Subset 24: {2, 3, 5, 8} => Sum = 18
Subset 25: {7, 8} => Sum = 15
Subset 26: {2, 7, 8} => Sum = 17
```

```
Subset 27: {3, 7, 8} => Sum = 18
Subset 28: {2, 3, 7, 8} => Sum = 20
Subset 29: {5, 7, 8} => Sum = 20
Subset 30: {2, 5, 7, 8} => Sum = 22
Subset 31: {3, 5, 7, 8} => Sum = 23
Subset 32: {2, 3, 5, 7, 8} => Sum = 25
```

Since a subset T of S such that the elements of T sum up to $t = 19$ does not exist, we can conclude that (S, t) is a no-instance of the subset sum problem.

Part (b)

The brute force algorithm for the Subset Sum problem would involve generating all possible subsets of the set S, computing the sum of each subset, and checking if any of these sums equal the target value t. The time complexity of generating all possible subsets of a set of size n is $O(2^n)$, which means that the number of operations required by the algorithm grows exponentially with the size of the input set.

For larger sets S, the number of possible subsets can quickly become infeasibly large. For example, if S has 100 elements, there are 2^{100} possible subsets, which is a number so large that it would take an impractical amount of time even for the fastest computers to generate all subsets.

In fact, the Subset Sum problem is known to be NP-complete, which means that it is unlikely that there exists a polynomial-time algorithm for solving it. Thus, for larger sets S, more sophisticated algorithms, such as dynamic programming or approximation algorithms, are needed to find solutions to the Subset Sum problem efficiently.

Part (c)

The dynamic programming solution to SUBSET SUM involves constructing a table of size $(n + 1) \times (t + 1)$, where n is the size of the input set S and t is the target sum. The table is filled using a recursive formula that checks whether adding the ith element of S to a subset of the first $i - 1$ elements can produce each possible sum up to t. The final entry of the table determines whether there exists a subset of S that sums to t.

The time complexity of this dynamic programming algorithm is $O(nt)$ because each entry of the table requires $O(1)$ time to compute, and there are nt entries in total. Therefore, the algorithm is not polynomial time, since it depends on both the size of the input set S and the target sum t.

Furthermore, SUBSET SUM is known to be an NP-complete problem, which means that there is no known polynomial-time algorithm for solving it in the worst case. While the dynamic programming algorithm may be efficient for small values of n and t, it cannot solve all instances of the problem in polynomial time.

Problem 4. (20 points) Exercise 34.5-5 on page 1101 using the reduction SUBSET SUM $\leq_p$ SET PARTITION.

Solution. We start with an instance of the SUBSET SUM problem, which consists of a set of n integers $S = a_1, a_2, \ldots, a_n$ and a target integer t. We want to determine whether there is a subset $S' \subseteq S$ whose elements sum up to t. We will reduce this problem to the SET PARTITION problem, which is defined as follows:
Given a set of n integers $S = a_1, a_2, \ldots, a_n$, determine whether there is a partition of S into two sets S_1 and S_2 such that $\sum_{a_i \in S_1} a_i = \sum_{a_i \in S_2} a_i$.
We will show that if the SET PARTITION problem can be solved in polynomial time, then the SUBSET SUM problem can also be solved in polynomial time. We do this by reducing SUBSET SUM to SET PARTITION.
The reduction proceeds as follows:

1. Let S and t be the input to the SUBSET SUM problem.
2. Compute the sum S_{sum} of all the elements in S.
3. Create a new set S' by adding two elements to S: t and $S_{sum} - t$.
4. Run the SET PARTITION algorithm on the set S'. If the algorithm returns YES, then there is a partition of S' into two sets S_1 and S_2 such that $\sum_{a_i \in S_1} a_i = \sum_{a_i \in S_2} a_i$. Since t and $S_{sum} - t$ are in different sets, we can remove them and obtain a partition of S into two sets S_1 and S_2 such that $\sum_{a_i \in S_1} a_i = t$, which means that there is a subset of S whose elements sum up to t. Therefore, the SUBSET SUM problem has a YES answer.
5. If the SET PARTITION algorithm returns NO, then there is no partition of S' into two sets S_1 and S_2 such that $\sum_{a_i \in S_1} a_i = \sum_{a_i \in S_2} a_i$. Since t and $S_{sum} - t$ are in different sets, we know that $\sum_{a_i \in S} a_i \neq 2t$, which means that there is no subset of S whose elements sum up to t. Therefore, the SUBSET SUM problem has a NO answer.

The reduction can be performed in polynomial time, since computing S_{sum} takes $O(n)$ time and adding two elements to a set takes constant time. Therefore, if the SET PARTITION problem can be solved in polynomial time, then the SUBSET SUM problem can also be solved in polynomial time. Since the SET PARTITION problem is NP-complete, the SUBSET SUM problem is also NP-complete.