HW4: Brainpower

Due: October 6

Problem 1

a. Given an instance $< G, k >$ of VertexCover, $G = (V, E)$, create a corresponding instance $< T, Q, k >$ of SubsetUnion. Assign a distinct integer (a **name**) to each edge in G, and let the set of all these names be T. For each vertex v in G, let Q_v be the names of the edges adjacent to v and let $Q = \{Q_v \mid v \in V\}$. Let k have the same value for both instances.

b. We show that this translation of instances of VertexCover to instances of SubsetUnion maps "Yes" instances of the former to "Yes" instances of the latter and vice versa. As a consequence, "No" instances of the former correspond to "No" instances of the latter.

   Given a "Yes" instance of VertexCover, there is set S of k vertices such that every edge touches a vertex in S. The corresponding instance of SubsetUnion is a "Yes" instance because T is the set of names of all edges and it can be formed by taking the union of k sets each of which is the set of edges touching one edge. The reverse also holds. Given a "Yes" instance of SubsetUnion, there is a set of k sets in Q whose union is the set of all edges. But the sets in Q that are combined to form T correspond exactly to the set S of k vertices of G such that the edges touching vertices in S is the set of all edges. Thus, a "Yes" instance of SubsetUnion corresponds directly to a "Yes" instance of VertexCover.

c. The input is a graph $G = (V, E)$ plus a binary integer k. Therefore, the input has size $|V| \log_2 |V| + 2|E| \log_2 |V| + \log_2 k$ because each vertex requires $\log_2 |V|$ bits and an edge is represented by two vertices.

   Our reduction assigns to each edge a number, which takes $|E| \log_2 |E|$ time because each edge requires $\log_2 |E|$ bits. It must also place each edge into an element of Q twice, once for each vertex it is connected to (or just once if it is a self-loop). This takes time $O(|E| \log_2 |E|)$ as well. k is unchanged. Thus, the total time is $O(|E| \log_2 |E|$, which is polynomial in the size of the input.

d. If VertexCover is NP-Complete, (a) it is in NP and (b) it is NP-hard; that is, every problem in NP can be reduced to it in polynomial time. By the above reduction, VertexCover can be reduced to SubsetUnion in polynomial time, so every problem in NP can also be reduced to SubsetUnion in polynomial time (by first reducing to VertexCover).

   To show that SubsetUnion is in NP, given an instance of $< T, Q, k >$ of the problem, let a choice agent choose k sets from Q. Determining whether or not the union of this collection of sets is the set T can be done in time $O(|T| \log_2 |T|)$, which is polynomial in the description of the instance $< T, Q, k >$.

Problem 2

Given an instance $< x, s >$ of Substring, x is a substring of s if x starts at either the first, second or $(|s| - |x| + 1)st$ position in s. To test for this, extract the c string in s of length $|x|$ beginning in the jth position, $1 \leq j \leq (|s| - |x| + 1)$, form the string cx^R, and pass this string to the recognizer for Palindrome. If the resulting string is a palindrome, $< x, s >$ is a "Yes" instance of Substring.

At each step of the loop, the algorithm only takes a substring, appends another string to it, and runs M. This happens at most $|s|$ times, which is polynomial in the size of the input (which is $|x| + |s|$).

Lab Problem 1

a. Although the algorithm is polynomial in the *value* of the input, in order to be polynomial time, it must be polynomial in the *size* of the input. The size of the input is the size of the inputs a, b, c, and p. Since these integers are binary integers, the size of the input is $\log_2(a) + \log_2(b) + \log_2(c) + \log_2(p)$. This algorithm runs its inner loop b times, which is $2^{\log_2(b)}$, and thus exponential in the size of the input.

b. The algorithm MOD-EXP shown below computes $a^b \bmod p$ in polynomial time. To decide Mod-Exp, return true if this algorithm's result equals $c \bmod p$, which is one extra operation.

```
MOD-EXP($a, b, p$):
    If $b$ is even:
        Set $r \leftarrow$ MOD-EXP($a, b/2, p$)
        Return $r \times r \bmod p$
    If $b$ is odd:
        Set $r \leftarrow$ MOD-EXP($a, (b-1)/2, p$)
        Return $a \times r \times r \bmod p$
```

This algorithm takes polynomial time. To see why, note that each time it recurses, the value of b is (at least) halved. b can be halved only $\log_2(b)$ times (which is polynomial in the size of the input), and the non-recursive part of the algorithm takes polynomial time.

An equivalent way to compute MOD-EXP is to write $b = b_k 2^k + b_{k-1} 2^{k-1} + \cdots + b_0 2^0$. Then, $a^b = a^{b_k 2^k} a^{b_{k-1} 2^{k-1}} \cdots a^{b_0 2^0}$. Since the b_j's are 0 or 1, we can compute a to powers of 2 up to 2^k by multiplying the previous power of two with itself, for example, $a^{2^3} = a^{2^2} a^{2^2}$. Do these multiplications modulo p and all the k powers of 2 are generated. This takes $k = O(\log_2 b)$ steps. Then choose the powers of 2 corresponding to j such that $b_j = 1$ and multiply them together modulo p. This takes $O(k)$ time. Thus, in polynomial time in the length of the input MOD-EXP is computed.

Lab Problem 2

1. Remember from that $\overline{p} \vee q$ is equivalent to $p \Rightarrow q$. Therefore, to translate an instance of 2SAT to an instance of ConSAT, we construct a new boolean formula by negating the first literal in each clause of the 2SAT formula, and changing all the "or"s to implications. The resultant boolean formula is logically equivalent to the old one, so the reduction will preserve satisfiability.

2. Construct an implication graph G as follows. Let the G contain two vertices for each literal mentioned in the ConSAT formula; one for the literal itself, and one for its negation. For each clause $(p \Rightarrow q)$ in the formula, add two directed edges to G: one from p to q, and one from $\overline{q}$ to $\overline{p}$. This second edge represents the contrapositive of the clause; information which would be lost if only the forward edge were included. Note that the construction of this graph takes polynomial time, since it adds, at most, four vertices and two edges per clause.

Suppose we have a machine M which decides Path for an input $< G, p, q >$. If M returns true on input $< G, p, q >$, then we know that the boolean formula implies that $p \Rightarrow q$, even if this is not included explicitly as a clause in the formula. We complete our reduction by making 2 calls to M for each literal in the ConSAT formula (a polynomial number of calls). Specifically, for each literal p, we run M on the input $< G, p, \overline{p} >$ and on the input $< G, \overline{p}, p >$. If, for any boolean literal p, these calls both return true, then we have $p \Leftrightarrow \overline{p}$, a contradiction; thus, we return false if this happens. If this never happens for any literal, then there are no contradictions in the formula, and it is satisfiable; thus we return true.

3. Yes, 2SAT is in P. Given a 2SAT formula, we can reduce it to a ConSAT problem in polynomial time, and reduce this to a Path problem in polynomial time. If Path were solvable in polynomial time, then we could solve it and have our answer. Fortunately, the depth-first search algorithm (among others) finds paths in a graph in polynomial time; thus Path and everything which reduces to it (including 2SAT) is in P.

Lab Problem 3

Suppose, without loss of generality, that the target sum s for SubsetSum is at least half of the total sum of the values in the set, which we will call A. (If this is not the case, then we can consider the SubsetSum problem where the target value is $A - s$.)

To reduce SubsetSum to GuitarPacking, we will construct a GuitarPacking problem from any given SubsetSum problem. A SubsetSum input consists of a set of integers which sum up to A, and a integer target s. To transform this into an instance of GuitarPacking, we will set the number k of guitar-holding bins equal to 2, and we will set the heights of the bins to be s. We will have one guitar for each integer in the set, with the same height as that integer. We will also have an extra guitar, of height $2s - A$. We now have a set of guitars, a number of bins (2), and heights for the bins (s). So, our reduction is done.

We now need to prove that our reduction works, and that it is polynomial time. It is easy to see that this is a polynomial time reduction; we are only transforming the input one at a time, and adding a single element. To see why the reduction works, notice that after the extra guitar of height $2s - A$ has been added, the total height of all guitars is $2s$. Thus, if a subset adds to s, it completely fills one bin. Because the sum of the heights of all guitars is $2s$, the other bin is also completely full. Thus, a "Yes" instance of SubsetSum corresponds to a "Yes" instance of GuitarPacking. If there is no solution to SubsetSum, then more than $k = 2$ bins is needed. Suppose that we have a "Yes" instance of GuitarPacking. Then, both bins are completely full and one of the two bins contains the extra guitar. The heights in the other bin correspond exactly to the integers in the instance of SubsetSum.

We have reduced SubsetSum to GuitarPacking in polynomial time. Therefore, since SubsetSum is NP-hard, it must also be that GuitarPacking is NP-hard. In order to complete our proof that GuitarPacking is NP-Complete, we just need to show that it is in NP. This is simple: Given a nondeterministic input which is an assignment of guitars to bins, we can check in polynomial time that the total number of bins used is less than the limit k, and we can add in polynomial time the heights of all guitars for each bin, and check that it is no more than the bin size b. Therefore, GuitarPacking is in NP (and NP-Complete).