Problem 1 (10 points).

Show that the geography problem introduced in class is in PSPACE.

Solution: Similar to QSAT problem introduced in class, we can design an algorithm for geography problem, then show it only needs polynomial space. Define *Geography*(G,u) as if there is a winning strategy to start from vertex u on graph G.

```
function Geography $(G = (V,E), u)$
    $S$ = a set of vertices $v$ such that $v \in V$ and $(u,v) \in E$;
    If $S = \varnothing$
        return false;
    For $v$ in $S$
        $G'$ = remove $v$ from $G$
        If Geography $(G', v)$ = false
            return true;
    return false;
end function;
```

As the depth of the search tree is most $|V|$, and for each layer, it only use $|V|+|E|$ space. The overall space usage of above algorithm is polynomial.

Problem 2 (10 points).

You are given a set $P = \{p_1,...,p_n\}$ of n points in the plane given by their integer coordinates, a set $Q = \{q_1,...,q_m\}$ of m curves given by m well defined equations. Suppose you can determine whether a point p_i lies on curve q_i or not in $O(1)$ time. A curve cover X of P is a subset of Q which satisfies that every point of P lies on at least one curve of X.

(1) Your task is to find a curve cover X_0 with minimized size. Can you (always) complete this task in polynomial time? If yes, design a polynomial-time algorithm to complete this task. Otherwise, explain why (e.g. prove it is NPC and claim there is no polynomial-time solver for NPC problems yet).

Solution: Curve cover problem is NP-hard so we believe it is not solvable in polynomial time. We prove it by reducing vertex cover to a curve cover in polynomial time. In the vertex cover problem, we are given $G = (V,E)$ where $|V| = m$, $|E| = n$. For each $e_i \in E$, we make a point p_i. For each $v_i \in V$ which is incidental to $\{e_a, e_b, e_c, ...\}$, we make a curve q_i threading the corresponding points $\{p_a, p_b, p_c, \cdots\}$. (Fact: Given the coordinates of n points, we can compute the equation of a curve threading all of them in polynomial time, for example using Lagrange polynomial. You can use this fact without proving.) Hence, finding a vertex cover V_1 covering all edges in E is equivalent to finding a curve cover X covering all points P.
Complexity: Making m points takes $O(m)$ time. Making n curves takes $O(n) \times O(poly)$ where $O(poly)$ is the time for making each curve. Hence, the total time used to reduce vertex cover to curve cover is polynomial.

(2) You are informed that there is one curve cover of P with size of positive integer k ($0 < k \ll n$). You only know its size k, but don't know what curves are in the curve cover. Can you find size-minimized curve cover X_0 in polynomial time with respect to n? Please note that size of X_0 could possibly be k or smaller. If yes, design a polynomial-time algorithm to complete this task. Otherwise, explain why (e.g. prove it is NPC). You may consider k is a small positive integer constant.

Solution: We can use a brutal force to choose a subset of size l (up to k) out of m curves and it is still in polynomial time. We know we can always find the smallest curve cover with a subset of size less or equal to k. Then the total number of chosen subsets of Q is $\sum_{l=1}^{k}\binom{m}{l}$. Since k is a small constant, we have:

$$\sum_{l=1}^{k}\binom{m}{l} \leq k \times \binom{m}{k} \leq k \times \frac{m^k}{k!}$$

For each chosen subset, we can judge whether all points P are covered by those l points or not in $n \times l \times O(1) = O(nl) \leq O(nk)$. Hence the total time used is less than $k \times \frac{m^k}{k!} \times O(nk) = O(nk\frac{m^k}{(k-1)!})$ which is polynomial to both n and m.

Problem 3 (10 points).

You are given an undirected graph $G = (V,E)$ and an integer k. We call a subset vertices $V_1 \subset V$ *closed*, if for any two vertices $u,v \in V_1$, there is no edge (u,v) in E, and there also does exist another vertex $w \in V$ such that both $(u,w) \in E$ and $(w,v) \in E$. The Closed Sub-Set Problem is to decide whether G has a closed subset of size at least k. Prove that the Closed Sub-Set Problem is NP-complete.

Solution: We will reduce an existing NP-complete problem the *Clique* problem into above problem. Given a graph $G = (V,E)$ and integer k, the Clique problem seeks if there exists a subset $V_1 \subset V$ such that for any two vertices $u,v \in V_1$ there is an edge $(u,v) \in E$. Such subset is called a clique of size k. We first show that the Clique problem is NP-complete. In fact, the Clique problem and the independent set problem are complementary: G contains an clique of size k if and only if its complementary graph G' contains an independent set of size k. The complementary graph $G' = (V,E')$ of $G = (V,E)$ is defined from as $(u,v) \notin E'$ if and only if $(u,v) \in E$. Clearly the Clique problem is in NP (the actual clique is the certificate). Additionally, the independent set is polynomial-time reducible to the Clique problem; so the Clique problem is NP-complete.

We now show that the Clique problem is polynomial-time reducible to above Closed Sub-set problem. Let $G = (V,E)$ and k be an instance of the Clique Sub-set problem. We create a new graph by modifying $G = (V,E)$: we add a new vertex w in the middle of every edge (u,v) and add two new edges (u,w) and (w,v). Then for every pair for new vertices w_i, w_j, we add an new edge (w_i,w_j). At last, we add one more new vertex w_0, and connect it to all the newly added vertices w_i.

Let G' be the resulting new graph. We now prove that, G has a clique of size k if and only if G' contains a closed subset of size $k+1$. On one hand, clearly, if G has a clique V_1 of size k, then in G' the same V_1 plus one extra vertex w_0 form a closed subset of size $k+1$. On the other hand, if G' has a closed subset V_1 of size $k+1$, then we can prove that G will contain a clique of size k. Notice that there is at most one newly added vertex can be in V_1 because newly added vertices are all pair-wisely connected with each other. Therefore, except for the newly added vertex, the other k vertices in V_1 must also be in G (i.e., not newly added ones). For any two vertices $u,v \in V_1$ and $u,v \in G$, there must exist a vertex w also in G‘ such that there are two edges (u,w) and (w,v). This w, must be the one that is newly added one that is between edge (u,v) when constructing G'. Thus, there must be an edge $(u,v) \in G$, and these k vertices must form a Clique in G. This concludes that the clique problem is polynomial-time reducible to the closed subset problem. And hence, the closed subset problem is NP-complete.

Problem 4 (0 points).

Prove that the 2SAT problem is in P.

Solution. A clause with 2 literals represents two implication relationship. For example, consider clause

$\overline{x_1} \vee x_2$. In order to satisfy it, if $\overline{x_1}$ is false then x_2 must be true, and if x_2 is false then $\overline{x_1}$ must be true. Formally, clause $A \vee B$ is equivalent to $\overline{A} \Rightarrow B$ and $\overline{B} \Rightarrow A$. Here A or B represents any literal (i.e., either a variable or the negation of a variable), and $\Rightarrow$ represents logic "implication".

We can then use a directed graph $G = (V, E)$, called implication graph, to represent all such implications of the given n clauses (with m variables). The graph contains $2m$ vertices, corresponding to all possible literals, i.e., $V = \{x_1, \overline{x_1}, x_2, \overline{x_2}, \cdots, x_n, \overline{x_n}\}$. And the graph contains $2n$ edges: each clause $A \vee B$ corresponds to 2 edges in G, which are $(\overline{A}, B)$ and $(\overline{B}, A)$. Intuitively, an edge $(u, v) \in E$ represents, if literal u is true then literal v must be true.

The implication can be carrying forward following any path in the implication graph. For example, path $u \to v \to w$ implies that if literal u is true then literal w must be true. What if there is path from u to $\overline{u}$? This means that: if literal u is true then $\overline{u}$ must be true, which is a contradiction. Hence, if such path exists then literal u can't be true. What if there is path from u to $\overline{u}$ *and* there is path from $\overline{u}$ to v? Then this means that u can't be true *and* $\overline{u}$ can't be true. In other words, the instance is not satisfiable. We summarize as the following statement: If in the implication graph there is a path from literal u to literal $\overline{u}$ and there is a path from literal $\overline{u}$ to literal u, then the instance is not satisfiable. Note that the condition that u can reach $\overline{u}$ and $\overline{u}$ can reach u is equivalent to that u and $\overline{u}$ are in the same connected component of G. Determine all connected components of G and constructing its meta-graph (a DAG in which each vertex corresponds to a componenet) takes $\Theta(|V| + |E|)$ time.

We now show that if such pair does not exist, then the 2SAT instance is satisfiable. The proof is constructive, i.e., when such pair does not exist, we can build an assignment that satisfies all clauses. The algorithm to find such an assignment is as follows. See example in Figure 1.

```
Algorithm build-assignment (components $C_1, C_2, \cdots, C_n$, organized as a DAG $G^M$ meta-graph)
    build a linearization $X$ of the meta-graph $G^M$ of the implication graph $G$;
    for each pair of literals $\{u, \overline{u}\}$
        let $C(u)$ and $C(\overline{u})$ be the components that include $u$ and $\overline{u}$, respectively;
        if $C(u)$ is before $C(\overline{u})$ in $X$: set literal $u$ to false and $\overline{u}$ to true;
        if $C(u)$ is after $C(\overline{u})$ in $X$: set literal $u$ to true and $\overline{u}$ to false;
    end for;
end algorithm;
```

$C_1 : \overline{x_1} \vee x_3$

$C_2 : \overline{x_2} \vee x_3$

$C_3 : x_2 \vee \overline{x_3}$

$C_4 : \overline{x_3} \vee \overline{x_4}$

$C_5 : \overline{x_3} \vee x_4$

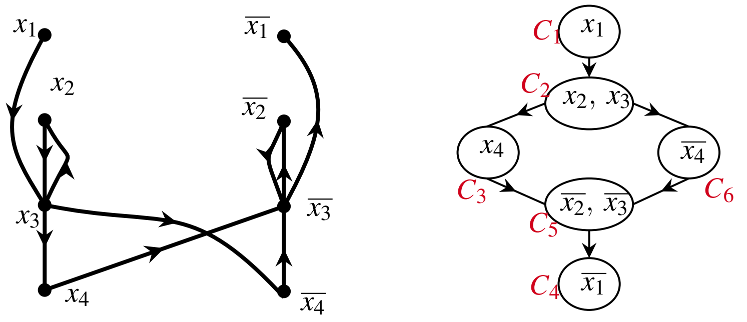

Figure 1: An instance of 2SAT, implication graph $G = (V, E)$, and the meta-graph G^M of G. A possible linearization of G^M is $X = (C_1, C_2, C_6, C_3, C_5, C_4)$. Using X above algorithm gives assignment $x_1 = F$, $x_2 = F$, $x_3 = F$, and $x_4 = T$.

We now prove that the assignment given in above algorithm satisfies all clauses. Suppose, by contradiction, that there exists a clause $u \vee v$ that is not satisfied, i.e., both literals u and v are set to false. According to above algorithm, we know that $C(u)$ is before $C(\overline{u})$ in X and $C(v)$ is before $C(\overline{v})$ in X. Note, $C(u) \neq C(\overline{u})$ and $C(v) \neq C(\overline{v})$. Consider the construction of the implication graph G: this clause $u \vee v$ adds to G two edges $(\overline{u}, v)$ and edge $(\overline{v}, u)$. Think about the components $C(\overline{u})$ and $C(v)$. The existence of edge $(\overline{u}, v)$ in G implies that either $C(\overline{u}) = C(v)$, i.e., $\overline{u}$ and v is in the same connected component, or there exists an edge from $C(\overline{u})$ to $C(v)$ in the meta-graph $G^M = (V^M, E^M)$. Hence, in any linearization X of G^M, $C(\overline{u})$ is equal to or before $C(v)$ in X. Similarly, edge $(\overline{v}, u)$ in G implies that either $C(\overline{v}) = C(u)$ or $(C(\overline{v}), C(u)) \in E^M$. Hence $C(\overline{v})$ is equal to or before $C(u)$ in linearization X. We have four ordering constraints for X: $C(u)$ is before $C(\overline{u})$, $C(\overline{u})$ is equal to or before $C(v)$, $C(v)$ is before $C(\overline{v})$, and $C(\overline{v})$ is equal to or before $C(u)$. It's not possible to have X to satisfy all of them. $\square$

Problem 5 (0 points).

Prove that the geography problem is in P if the underlying graph is acyclic.

Solution: If the underlying graph is acyclic, we can design a dynamic programming algorithm to solve the geography problem in polynomial time. First, use a topological sort to sort all the vertices in G. A topological order of a directed graph is for every directed edge (u, v), u always comes before v in the ordering. Here we want to use the reverse order of topological order to do the dynamic programming. Let $DP[v]$ be if there is a winning strategy to pick the next vertex from v.

Initially, for any vertex v without any outgoing edges, $DP[v] =$ false. Then we calculate the value of DP for all the vertices following the reverse topological order. When it comes to vertex u, we must have gotten the values of $DP[v]$ such that $(u, v) \in E$. Assume there are k out edges of u, then we have:

$$DP[u] = \neg(DP[v_1] \wedge DP[v_2] \cdots \wedge DP[v_k]) \quad ((u, v_i) \in E, 1 \leq i \leq k)$$

The answer will be the value of source vertex. As topological sort takes $O(|V| + |E|)$ time, and the dynamic programming also takes $O(|V| + |E|)$. The total running time is $O(|V| + |E|)$.

Problem 6 (0 points).

A linear inequality over variables $x_1, \cdots, x_k$ is an inequality of the form $c_1 x_1 + \cdots + c_k x_k \leq b$, where $c_1, \cdots, c_k$ and b are integers. Given a set of such inequalities, the problem is to decide whether it has an integeral solution, i.e., whether one can assign integeral values to all variables in such a way that all inequalities are satisfied. Prove that this problem is NP-complete.

Solution: First we want to prove this problem is in NP. Given $c_1, \cdots, c_k$, b and a certificate $C = x_1, \cdots, x_k$, we can verify it as:

```
$Verifier$ (inequalities, $C$)
    If $\exists x_i \in C$ is not integer
        return "No";
    For $c_1, \cdots, c_k$, $b$ in inequalities
        If $c_1 x_1 + \cdots + c_k x_k > b$
            return "No";
    return "Yes";
end function;
```

Now we will prove 3-SAT is polynomial time reducible to this problem. Let the variables in the 3SAT formula be $z_1, z_2, \cdots, z_k$. We can have corresponding variables $x_1, x_2, \cdots, x_k$ where $x_i = 1$ when z_i is true and $x_i = 0$when z_i is false. Apparently we need to restrict each variable from 0 to 1: $0 \leq x_i \leq 1$ by add inequalities $x_i \leq 1$ and $-x_i \leq 0$.

For each clause, we can convert it into an inequality. For example, if the clause is $z_1 \vee \neg z_2 \vee z_3$, the corresponding inequality will be:

$$\begin{aligned} &x_1 + (1 - x_2) + x_3 > 0 \\ \iff &-x_1 + x_2 - x_3 \leq 0 \end{aligned}$$

To satisfy this inequality, we must either set $x_1 = 1$ or $x_2 = 0$ or $x_3 = 1$. Then the origin clause can also be true. It takes polynomial time to get the inequalities for the new problem. So 3-SAT is polynomial time reducible to this problem. This problem is NP-complete.