Problem 1 (15 points).

Design a polynomial-time algorithm for the 2SAT problem: given m binary variables and n clauses such that each clause contains exactly 2 literals, decide whether there exists an assignment of all variables such that all n clauses are true.

Solution. We build a directed graph $G = (V,E)$, where $V = \{x_1, \neg x_1, x_2, \neg x_2, \cdots, x_n, \neg x_n\}$. For every clause in the form $l_i \vee l_j$ we add two edges $(\neg l_i, l_j)$ and $(\neg l_j, l_i)$ to E (suggesting that if l_i (resp. l_j) is false then l_j (resp. l_i) must be true). Note that G satisfies this property: there exists a path from x to y if and only if there exists a path from $\neg y$ to $\neg x$. This is because, according to above construction, edges (a,b) and $(\neg b, \neg a)$ are always paired in G.

We claim that the given 2SAT instance is satisfiable if and only if in G there is no path from x_i to $\neg x_i$ and there is no path from $\neg x_i$ to x_i, for every $1 \le i \le n$. For one side, suppose that the 2SAT is satisfiable. Let $x_i = 1$ in this true assignment (the other case can be analyzed symmetrically). If there exists a path from x_i to $\neg x_i$ then this implies that, in order to satisfies certain clauses, eventually we must have that $x_i = 0$. This gives a contradiction. Now consider the other side that there exists no path from x_i to $\neg x_i$ nor from $\neg x_i$ to x_i. Then we can construct a true assignment for the 2SAT instance: arbitrarily pick an unassigned literal l in the clauses and set $l = 1$ (and hence set $\neg l = 0$); in G assign 1 to all the reachable literals from l (and set their negations to 0); repeat this procedure until all literals are assigned. This algorithm will always assign all literals as there exists no path from x_i to $\neg x_i$. And clearly such assignment will satisfy all clauses.

Problem 2 (20 points).

Prove that IS (independent-set problem) can be polynomial-time reducible to IS-k (decision version of the independent-set problem). That is, given $G = (V,E)$, design an algorithm that uses polynomial-time calls of a sovler for IS-k (with possible extra polynomial-time instructions) to find an independent set $V_1 \subset V$ such that $|V_1|$ is maximized. Your algorithm should return V_1 instead of just $|V_1|$.

Solution. The algorithm for problem IS is given below. This algorithm calls subroutine IS1 (G, k), which assumes that an independent set S_1 of G with $|S_1| = k$ exists and returns one such independent set.

```
Algorithm IS (G)
    for k = |V| to 1
        if IS-k (G, k) returns true
            return IS1 (G, k)
        end if
    end for
end IS
```

We now design an algorithm for IS1 (G, k). Recall that when we call IS1 (G, k) , an independent set S_1 of G with $|S_1| \le k$ must exist, and IS1 (G, k) will return such S_1. Let $v \in V$ be an arbitrary vertex of G. There are only two possibilities: either $v \in S_1$ or $v \notin S_1$. We can decide which is the correct case by calling IS-k. Specifically, let G_v^1 be the graph after removing v and the adjacent vertices of v (and all their adjacent edges) from G; let G_v^2 be the graph after removing v (and the adjacent edges of v) from G. If it is the first case, i.e., $v \in S_1$, then G_v^1 must contain an independent set S_2 with $|S_2| \ge k-1$, which can be determined by calling IS-k (G_v^1, $k-1$). If it is the second case, i.e., $v \notin S_1$, then G_v^2 must contains an independent set S_2 with $|S_2| \ge k$, which can be determined by calling IS-k (G_v^2, k). Once we know either $v \in S_1$ or $v \notin S_1$, then the size of the problem is reduced, and we can recursively call IS1 to solve the reduced subproblem. The complete algorithm for IS1 is given below.

Algorithm IS1 (G, k)
let $v \in V$ be an arbitrary vertex of G
construct G_v^1 from G by removing v and its adjacent vertices and all their adjacent edges
construct G_v^2 from G by removing v and its adjacent edges
if IS-k (G_v^1, $k-1$) returns true
S_2 = IS1 (G_v^1, $k-1$)
return $S_2 \cup \{v\}$
else
return IS1 (G_v^2, k)
end if
end IS1

Let $T(n)$ be the running time of IS1 (G, k) where G contains n vertices. Notice that both G_v^1 and G_v^2 contain at most $n-1$ vertices. Therefore we have recursion: $T(n) \leq \Theta(n) + ISk + T(n-1)$, where $\Theta(n)$ estimates the running time of constructing G_v^1 and G_v^2, and ISk represents the running time of IS-k. Hence, $T(n) \leq O(n^2) + n \cdot ISk$. The total running time of IS is $O(n \cdot T(n))$ which is polynomial too.

Problem 3 (20 points).

A linear inequality over variables $x_1, ..., x_k$ is an inequality of the form $c_1x_1 + ... + c_kx_k \leq b$, where $c_1, ..., c_k$ 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. (Instructions: first prove that this problem is in NP; then select an existing NP-complete problem (in this case you may consider 3SAT) and prove it is polynomial-time reducible to this problem.)

Solution. First we prove this problem is in NP. The *certificate* will be n numbers $(x_1, x_2, \cdots, x_n)$. The *verifier* verifies whether x_k is an integer, for every $1 \leq k \leq n$, and for every inequality it verifies whether $c_1x_1 + c_2x_2 \cdots + c_kx_k \leq b$. If all these are true then the verifier return true and otherwise returns false. Clearly, the certificate is in polynomial-size and the verifier runs in polynomial-time. Also, an instance is true if and only if there exists such $(x_1, x_2, \cdots, x_n)$ that the verifier returns true. Hence, this problem is in NP.

Then we prove that 3SAT is polynomial time reducible to this problem. Specifically, for any instance A of 3SAT, we construct an instance B of this problem and show that A is a true instance if and only if B is true.

For any variable x_k in A, we add a variable x_k to B. To model that in A variable x_k is binary, we add two inequalities to B: $x_k \leq 1$ and $-x_k \leq 0$. Together with that all variables in B are integers, we have that in B every variable can only take value 0 or 1 (i.e., binary).

For each clause in A, we add an inequality to B in the form of $f \geq 1$ (equivalent to $-f \leq -1$). For any literal in this clause, if it is in the form of x, we add x to f, and if it is in the form of $\neg x$ we add $(1-x)$ to f. One examples is: clause $\neg x_1 \vee x_2 \vee \neg x_3$ in A becomes inequality $(1-x_1) + x_2 + (1-x_3) \geq 1$ in B.

It is easy to verify that A is a true instance if and only if B is a true instance. Hence 3SAT problem is polynomial-time reducible to this problem.

Problem 4 (20 points).

A store is trying to analyze the behavior of its customers. Suppose they have n customers and they sell m products. We can use a binary matrix A to represent the behavior of these customers: the size of A is $n \times m$ and the entry $A[i, j]$ indicates whether customer i ever bought product j. Let us say that a subset S of all the customers is *diverse* if no two of the customers in S have ever bought the same product. The *diverse subset problem* is defined as follows: given an $n \times m$ array A as defined above, and integer $k \leq n$, to decide whether there exists a diverse subset whose size is at least k. Prove that this problem is NP-complete. (*Hint:* reducing the independent-set problem to this problem.)

Solution: We first prove that this problem in in NP. For any instance the certificate will be a subset S of all customers. The verifier verifies that $|S| = k$; for any two elements $a_i, a_j \in S$, the verfier checks the corresponding two rows in A, namely $A[i,\cdot]$ and $A[j,\cdot]$, and verify that there does not exist k such that $A[i,k] = A[j,k] = 1$. If all these are true, the verifier returns true and otherwise return false. Clearly, the certificate is in polynomial-size and the verifier runs in polynomial-time too. Also, an instance of the diverse subset problem is true if and only if there exists such a certificate that the verifier returns true. Hence, this problem is in NP.

We then show that the independent-set problem is polynomial-time reducible to this problem. For any instance $(G = (V,E), k)$ of the independent-set problem, we construct an instance of the diverse subset problem as follows. Let $n = |V|$ and $m = |E|$. We then add n customers and m products to the instance of the diverse subset problem. For edge $e_k = (v_i, v_j) \in E$, we set $A[i,k] = A[j,k] = 1$ (suggesting that customer i and j have bought the same product k). By the construction, we have that two vertices in G are connected by an edge if and only if in the diverse subset problem the two corresponding customers ever bought the same product. Therefore, an subset of vertices in G is independent if and only if the corresponding customers in the diverse subst problem are diverse. Hence, (G,k) is a true instance if and only if A is a true instance. The construction of A can be done in polynomial-time. This proves that the independent-set problem is polynomial-time reducible to the diverse subset problem.

Problem 5 (20 points).

Let $G = (X \cup Y, E)$ be a bipartite graph. We define an (a,b)-skeleton of G to be a set of edges $E' \subseteq E$ so that at most a nodes in X are incident to an edge in E', and at least b nodes in Y are incident to an edge in E'. Show that, given a bipartite graph G and two integers a and b, it is NP-complete to decide whether G has an (a,b)-skeleton. (*Hint*: reducing the set cover problem to this problem.)

Solution: The problem is in NP since we can exhibit a subset E' of the edges (i.e., certificate), and it can be verified in polynomial time that at most a nodes in X are incident to an edge in E', and at least b nodes in Y are incident to an edge in E'.

We now show that the set cover problem is reducible to this problem. Given a collection of sets $S_1, \ldots, S_k$ over a ground set U of size n, we define a bipartite graph $G = (X \cup Y, E)$ in which the nodes in X correspond to the sets S_i ($1 \leq i \leq k$), and the nodes in Y correspond to the elements in U. We build E by joining each set node to the nodes corresponding to elements that it contains. We also set $a = k$ and $b = n$. In particular, this means that our (a,b)-skeleton must touch every node in Y.

Now, if G has an (a,b)-skeleton E', then the k nodes in X incident to edges in E' correspond to k sets that collectively contain all elements, so they form a set cover. Conversely, if there is a set cover of size k, then taking E' to be the set of all edges incident to corresponding set nodes yields an (a,b)-skeleton.

Problem 6 (20 points).

Consider a set $A = \{a_1, ..., a_n\}$ and a collection $B_1, B_2, ..., B_m$ of subsets of A (i.e., $B_i \subseteq A$, $1 \leq i \leq m$). We say that a set $H \subseteq A$ is a hitting set of $\{B_i\}$ if H contains at least one element from each B_i; that is, if $H \cap B_i$ is not empty for each i (so H "hits" all B_i). Is there a hitting set $H \subseteq A$ for $B_1, B_2, ..., B_m$ so that $|H| \leq k$? Prove that this problem is NP-complete. (*Hint:* reducing the set cover problem to this problem.)

Solution: We first show that this problem is in NP. The cerficate is a subset H of A. The verifier verfies $|H| \leq k$, and verifies $B_j \cap H \neq \emptyset$ for every $1 \leq j \leq m$. If all these are true the verifier returns true and otherwise returns false. Clearly, an instance is true if and only such H exists that the verifier returns true.

We then show that the set cover problem is polynomial-time reducible to this problem. Let X, $(S_1, S_2, \cdots, S_n)$, and k be any instance of the set cover problem, where X is the univeral set and $\{S_i\}$ are the n subsets of X, and the problem is to decide whether there exists a set cover with at most k subsets. We now construct an instance of the hitting set problem: $A = \{a_1, a_2, \cdots, a_n\}$, corresponds to the n subsets of the set cover problem; for each element $x_j \in X$, $1 \leq j \leq |X| = m$, we construct a subset B_j in the set cover problem, and add a_i to B_j if in the set cover problem S_i contains x_j. From this construction, it is easy to verify that there exists a one-to-one corresponse between a set cover (in the set cover problem) and a hitting set (in the hitting set problem). Hence, the constructed instance $(A, \{B_j\}, k)$ contains a hitting set of at most size k if and only if the instance for the set cover problem contains a set cover of at most size k.

Problem 7 (30 points).

Consider a special case of the above hitting set problem: each set B_i contains at most c elements. Design a fixed parameter tractable (FPT) algorithm for this problem, i.e., given $A = \{a_1, a_2, \cdots, a_n\}$, $B_1, B_2, \cdots, B_m$, integers k and c, decide whether there exists a hitting set $H \subseteq A$ of $\{B_i\}$ such that $|H| \leq k$; your algorithm should run in $O(f(k,c) \cdot p(n,m))$ time, where $p(\cdot,\cdot)$ is a polynomail function.

Solution. We use the same idea as the FPT algorithm for the vertex-cover problem. Specifically, we search up to k levels, and in each level, we arbitrarily select a subset that has not been "hitted", and then enumerate all its elements. We define recursive function HS-FPT (A, $B = (B_1, B_2, \cdots, B_m)$, k) which determines whether there exists a hitting set of B with at most k elements.

Algorithm HS-FPT (A, $B = (B_1, B_2, \cdots, B_m)$, k)
  base case: if $k \leq 1$ enumerate and return
  let $B_i \in B$ be an arbitrary subset in B
  for $x \in B_i$
    Let $B' = B$
    for every $B_j \in B$
      if $x \in B_j$ then set $B' = B' \setminus \{B_j\}$
    end for
    z = HS-FPT (A, B', $k-1$)
    if z is true, return true
  end for
  return false
end HS-FPT

The above algorithm is correct, which can be simply proved by induction. The key point is that B_i has to be hitted, and therefore one of its elements must be selected (and the algorithm enumerate all its elements).

Let $T(n,m,k,c)$ be the running time of HS-FPT (A, B, k), with $|A| = n$, $|B| = m$ and each subset in B contains at most c elements. We have $T(n,m,k,c) \leq c \cdot (mn + T(n, m-1, k-1, c))$. This is because $|B_i| \leq c$, $|B'| \leq m-1$ as at least B_i is hitted, and the construction of B' takes mn time. We can show that, by induction, $T(n,m,k,c) \leq k \cdot c^k \cdot mn$. Therefore, the above algorithm is an FPT algorithm.

Problem 8 (30 points).

Consider the optimiation version of the above problem: given $A = \{a_1, a_2, \cdots, a_n\}$, $B_1, B_2, \cdots, B_m$, integer

c, where $|B_i| \le c$ for any $1 \le i \le m$, find a hitting set $H \subseteq A$ of $\{B_i\}$ such that $|H|$ is minimized. Design a c-approximation algorithm for this problem: describe your algorithm, prove that the approximation ratio of your algorithm is c, and give an example to show that your analysis is tight.

Solution. We can use the LP + rounding technique to design a c-approximation algorithm for this problem.

Step 1. We formulate this problem as an ILP. We add binary variable x_i, $1 \le i \le n$, to indicate whether a_i is in the final hitting-set. For every subset B_j, $1 \le j \le m$, we add constraint $\sum_{a_i \in B_j} x_i \ge 1$ to guarantee that B_j will be hitted. The objective function will be then to minimize $\sum_{1 \le i \le n} x_i$. Formally,

$$\begin{array}{ll} \min & \displaystyle\sum_{1 \le i \le n} x_i \\ s.t. & \left\{ \begin{array}{rcll} \sum_{a_i \in B_j} x_i & \ge & 1 & 1 \le j \le m \\ x_i & \in & \{0,1\} & 1 \le i \le n \end{array} \right. \end{array}$$

Step 2. Relax above ILP into the following LP:

$$\begin{array}{ll} \min & \displaystyle\sum_{1 \le i \le n} x_i \\ s.t. & \left\{ \begin{array}{rcll} \sum_{a_i \in B_j} x_i & \ge & 1 & 1 \le j \le m \\ x_i & \in & [0,1] & 1 \le i \le n \end{array} \right. \end{array}$$

Step 3. Solve above LP (using existing algorithm in polynomial-time). Let x_i^* be its optimal solution.

Step 4 (rounding). Let $x_i' = 1$ if $x_i^* \ge 1/c$ and let $x_i' = 0$ if $x_i^* < 1/c$. Return $H = \{a_i \in A \mid x_i' = 1\}$.

We first prove that H is a hitting-set of B. This is equivalent to show that $\{x_i'\}$ is a feasible solution of the above ILP. In fact, since $\{x_i^*\}$ is an optimal (and therefore feasible) solution of LP, we have that $\sum_{a_i \in B_j} x_i \ge 1$ for every $B_j \in B$. Because $|B_j| \le c$, there exists $a_i \in B_j$ such that $x_i \ge 1/c$, for every $B_j \in B$. Based on above rounding method, for every $B_j \in B$ there exists $a_i \in B_j$ such that $x_i' = 1$. Hence $\{x_i'\}$ is a feasible solution of the ILP, which consequently implies that H is a hitting set of B.

We now prove that the above algorithm is a c-approximation algorithm. Let x_i^o be the optimal solution of the ILP. Hence, the optimal hitting-set, denoted as H_{opt}, has size $|H_{opt}| = \sum_{1 \le i \le n} x_i^o$. We need to prove that $|H|/|H_{opt}| \le c$. According to our rounding scheme, we have $x_i' \le c \cdot x_i^*$, for every $1 \le i \le n$. Also, since LP is a relaxation of the ILP, we must have $\sum_{1 \le i \le n} x_i^* \le \sum_{1 \le i \le n} x_i^o$. Combining these, we have $|H| = \sum_{1 \le i \le n} x_i' \le \sum_{1 \le i \le n} c \cdot x_i^* = c \cdot \sum_{1 \le i \le n} x_i^* \le c \cdot \sum_{1 \le i \le n} x_i^o = c \cdot H_{opt}$. This prove that the approximation ratio of the above algorithm is c.

Tight example. Consider an instance $A = (a_1, a_2, \cdots, a_n)$, $B = (B_1, B_2, \cdots, B_n)$, each B_j contains exactly c elements of A, and each element a_i is contained in exactly c subsets of B. Assume that $n = c \cdot k$ for some k. The optimal hitting set therefore contains k elements of A. One optimal solution of LP is $x_i^* = 1/c$ for every $1 \le i \le n$. Consequently, $x_i' = 1$ for every $1 \le i \le n$. Therefore, we have $H = A$ and $|H|/|H_{opt}| = n/k = c$.

Problem 9 (30 points).

Consider the optimization version of the 3D matching problem: given disjoint sets X, Y, Z, and a set of triples $E \subseteq X \times Y \times Z$ (you may assume that $|X| = |Y| = |Z| = n$), to find a matching $M \subseteq E$ such that $|M|$ is maximized. Design a 3-approximation algorithm for this problem: describe your algorithm, prove that the approximation ratio of your algorithm is 3, and give an example to show that your analysis is tight.

Solution. We design a *greedy* algorithm. Let $M = \emptyset$. In each iteration, we arbitrarily choose an edge $e \in E$ and add e to M, and then remove all edges in E that *conflict* with e (we define that two edges $e_1 = (x_1, y_1, z_1)$ and $e_2 = (x_2, y_2, z_2)$ conflict with each other if $x_1 = x_2$ or $y_1 = y_2$ or $z_1 = z_2$).

Clearly, the above algorithm runs in polynomial-time, and the returned edges, i.e., M, is a matching. Let M^* be the optimal matching. We now prove that $|M| \leq 3 \cdot |M^*|$. To show that, we consider a bipartite graph $B = (M \cup M^*, R)$: we connect $e \in M$ and $e^* \in M^*$ by an edge (in R) if e and e^* conflict. We claim that each $e \in M$ conflict at most with 3 edges in M^*. To see this, suppose conversely that $e \in M$ conflict with 4 edges in M^*, then at least two of them share the same element at X or Y or Z, which contradicts to the fact that M^* is a matching. Therefore, in B the degree of each vertex in M is at most 3. In addition, we must have that in B all vertices in M^* have degree of at least 1, since otherwise the above algorithm will continue to include them. Hence, $|M^*|$ is bounded by the total degree of all vertices in M, i.e., $|M^*| \leq \sum_{e \in M} degree(e) \leq 3 \cdot |M|$.

Tight example. $X = \{x_1, x_2, x_3\}$, $Y = \{y_1, y_2, y_3\}$, $Z = \{z_1, z_2, z_3\}$,
$E = \{(x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3), (x_1, y_2, z_3)\}$. The above algorithm might return $M = \{(x_1, y_2, z_3)\}$, while the optimal matching is $M^* = \{(x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3)\}$.

Problem 10 (30 points).

You are given an $n \times n$ square graph $G = (V, E)$, where $V = \{v_{ij}\}$, $1 \leq i, j \leq n$, and $(v_{ij}, v_{kl}) \in E$ if and only if $|i - k| = 1$ and $|j - l| = 1$. Each vertex v has a non-negative weight $w(v)$. The *weighted independent-set problem on square graph* seeks an independent set $V_1 \subset V$ such that $\sum_{v \in V_1} w(v)$ is maximized. Design a 4-approximation algorithm for this problem: describe your algorithm, prove that the approximation ratio of your algorithm is 4, and give an example to show that your analysis is tight.

Solution. We can again design an greedy algorithm. Let $V_1 = \emptyset$. We sort all vertices according to their weights in descending order. We additionally maintain a field for each vertex to indicate whether it is available to pick and initialize all vertices as available. We then process all vertices in this order: if it is available, we add it to V_1 and mark all its adjacent vertices as unavailable.

Clearly, the above algorithm runs in polynomial-time, and the returned vertices, i.e., V_1, is an independent set. Let V^* be the optimal independent set. We now prove that $\sum_{v \in V^*} w(v) \leq 4 \cdot \sum_{u \in V_1} w(u)$. Without loss of geneality, we assume that $V_1 \cap V^* = \emptyset$; otherwise we exclude the shared vertices from both and then prove the desired inequality. Again, we build a bipartite graph $B = (V_1 \cup V^*, R)$. We process all vertices in V_1 in descending order of their weights: for each $u \in V_1$, we add edge (u, v) to R for those $v \in V^*$ satisfying that $(u, v) \in E$ and the current degree of $v \in V^*$ in B is 0.

We show that the bipartitie graph B satisfies the following properties. First, the degree of every vertex $v \in V^*$ in B is exactly 1. This is because, we only connect vertices of V^* with degree of 0, and therefore none of the vertex in V^* will have degree larger than 1. Besides, all vertices in V^* must be covered, since otherwise the greedy algorithm will then include them to V_1. Second, the degree of every $u \in V_1$ in B is at most 4, as in the given graph the degree of each vertex is at most 4. Third, for any $(u, v) \in R$, we must have that $w(u) \geq w(v)$. This is because the greedy algorithm always selects the vertex with the largest weight among all available vertices. Consider individual connected components of B: let $u \in V_1$ and let V_u^* be the adjacent vertices of u in V^*. Based on these properties, we have $|V_u^*| \leq 4$ and $w(u) \geq w(v)$ for any $v \in V_u^*$. Hence $\sum_{v \in V_u^*} w(v) \leq 4 \cdot w(u)$. Combining all components we therefore have $\sum_{v \in V^*} w(v) \leq 4 \cdot \sum_{u \in V_1} w(u)$.

Tight example. Consider $n = 3$, and $w(v_{11}) = w(v_{13}) = w(v_{31}) = w(v_{33}) = w(v_{22}) = 1$ and all other vertices have weight of 0. The greedy algorithm might return $V_1 = \{v_{22}\}$, while the optimal solution $V^* = \{v_{11}, v_{13}, v_{31}, v_{33}\}$.

Remark: The only property that leads to a 4-approximation algorithm is that the maximum degree of the graph is 4.