1. (a) The IP dual is

$$\begin{array}{lrl} \text{Minimize} & y^T b & \\ \text{Subject to} & y^T A & \geq c^T \\ & y & \geq 0 \\ & y & \in \mathbb{Z}^n \end{array}$$

Claim 0.1 (Principle of weak duality for IP)**.** *If $\widehat{x} \in \mathbb{Z}^n$ is a solution to the primal, and $\widehat{y} \in \mathbb{Z}^n$ is a solution to the dual, then $c^T\widehat{x} \leq \widehat{y}^T b$.*

Proof.

$$\widehat{y}^T b \geq \widehat{y}^T(A\widehat{x}) = (\widehat{y}^T A)\widehat{x} \geq c^T\widehat{x}.$$

□

(b) Consider the following

Primal			Dual		
max	y		min	3β	
subj. to	$y - \frac{3}{2}x$	≤ 0	subj. to	$\alpha + \beta$	≥ 1
	$y + \frac{3}{2}x$	≤ 3		$-\frac{3}{2}\alpha + \frac{3}{2}\beta$	≥ 0
	y, x	≥ 0		α, β	≥ 0
	y, x	$\in \mathbb{Z}$		α, β	$\in \mathbb{Z}$

Primal of this IP has optimal value 1 (draw the feasible region), while the dual has optimal value 3 (achieved by setting $\alpha = \beta = 1$, note that β cannot be 0, because then $\alpha \geq 1$ and this violates the second inequality in the dual). Also, note that if the integrality constraint is dropped then both primal and dual have value 3/2 (achieved for $x = 1, y = 3/2$ and $\alpha = \beta = 1/2$).

(c) $L_{\text{IP-feas}} = \{\langle A, b\rangle \mid A \in \mathbb{Z}^{m\times n}, b \in \mathbb{Z}^m, \exists x \in \{0,1\}^n Ax \leq b\}$.

Claim 0.2. *$L_{IP\text{-}feas}$ is NP-complete.*

Proof. (1) $L_{\text{IP-feas}} \in$ NP: given A, b and a proposed solution $x \in \{0,1\}^n$, we can verify in polynomial time whether $Ax \leq b$. This is because matrix-vector multiplication takes polynomial time.

(2) We will show that $L_{\text{3-SAT}} \leq_p L_{\text{IP-feas}}$. We are given a 3-CNF $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$, where the C_i are clauses, i.e, $C_i = \ell_i^1 \vee \ell_i^2 \vee \ell_i^3$. We will encode each clause as a linear inequality. Let $pos(C_i)$ denote the variables that appear positively in C_i, and $neg(C_i)$ be the variables appear negated in C_i. Then the inequality corresponding to clause C_i is $\sum_{x_j \in pos(C_i)} x_j + \sum_{x_j \in neg(C_i)}(1 - x_j) \geq 1$. For example, if clause $C_1 = x_1 \vee x_2 \vee \neg x_3$, then the inequality we get from it is $x_1 + x_2 + (1 - x_3) \geq 1$. Note that $x \in \{0,1\}^n$ satisfies clause i if and only if x satisfies the derived inequality. We can transform these inequalities into the form $a_i^T x \leq b_i$ by collecting constant terms on the LHS and brining them to the RHS and multiplying the inequality by -1 to change $\geq$ into $\leq$. Putting all such inequalities together we end up with an IP-feasibility instance $Ax \leq b$. By the above observation, it follows that $x \in \{0,1\}^n$ satisfies *all clauses* of ϕ if and only if the same x satisfies all inequalities $a_i^T x \leq b_i$. This shows that $\langle\phi\rangle \in L_{\text{3-SAT}} \iff \langle A, b\rangle \in L_{\text{IP-feas}}$. Clearly, construction of A and b can be carried out in polynomial time. □

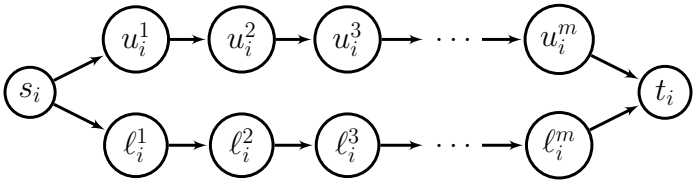
2. (20 pts)

(a) $L_{\text{NODE-DISJ}} = \{\langle G, k, s_1, \ldots, s_k, t_1, \ldots, t_k\rangle \mid \exists k \text{ node-disjoint paths } s_i \rightsquigarrow t_i\}$.

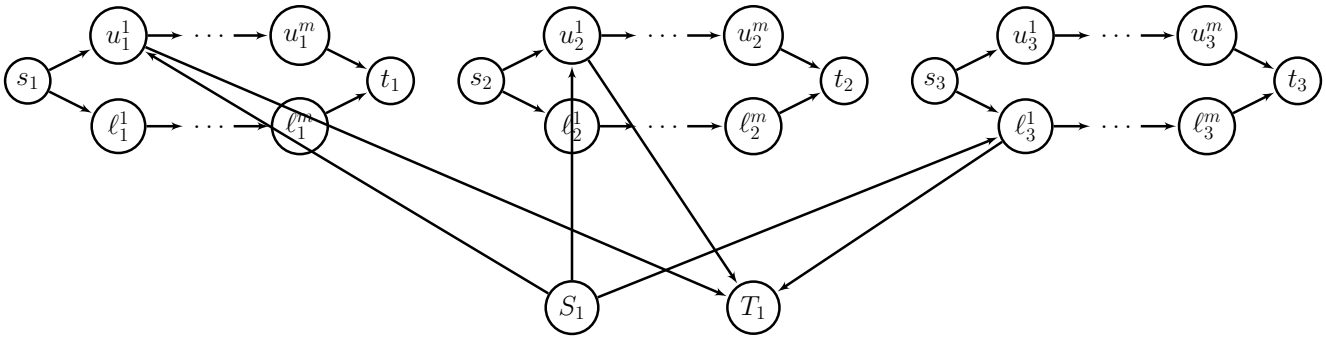
(b)

Claim 0.3. *$L_{3\text{-SAT}} \leq_p L_{NODE\text{-}DISJ}$*

Proof. Let ϕ be a given 3-CNF with n variables and m clauses. We show how to construct an instance of $L_{\text{NODE-DISJ}}$ such that the instance is in the language if and only if ϕ is satisfiable. For each variable x_i we introduce s_i and t_i, which are connected by two node-disjoint paths with m internal nodes: $s_i \to u_i^1 \to u_i^2 \to \cdots \to u_i^m \to t_i$, and $s_i \to \ell_i^1 \to \cdots \to \ell_i^m \to t_i$. These two paths are referred to as upper and lower. See the figure below:



For each clause C_j introduce a new starting node S_j and a new finishing node T_j (note we reserve capital letters for starting and finishing nodes corresponding to clauses, and small letters for starting and finishing nodes corresponding to variables). If clause C_j contains literal x_i add edges (S_j, u_i^j) and (u_i^j, T_j). If clause C_j contains literal $\neg x_i$ add edges (S_j, ℓ_i^j) and (ℓ_i^j, T_j). For example, if $C_1 = x_1 \vee x_2 \vee \neg x_3$ our construction gives the following:



Add such edges for each literal appearing in each clause. Thus, we get a resulting graph G and we are looking for $n + m$ disjoint-node paths connecting s_i to t_i (we have n of these) and S_j to T_j (we have m of these). Suppose ϕ is satisfiable by assignment x, then we can construct $n+m$ disjoint-node paths as follows. For each i such that $x_i = 0$, connect s_i to t_i via an upper path, and for each i such that $x_i = 1$ connect s_i to t_i via a lower path. As for the clauses, if C_j contains literal x_i and is satisfied by it, we can connect S_j to T_j via a path $S_j \to u_i^j \to T_j$. If C_j contains literal $\neg x_i$ and is satisfied by it, we can connect S_j to T_j via a path $S_j \to \ell_i^j \to T_j$. Since each clause is satisfied by some literal, we can connect all S_j to T_j. Conversely, if we have $n + m$ disjoint paths, then we can derive a satisfying assignment out of it. Note that each s_i can be connected to t_i only via an upper or a lower path. If s_i is connected to t_i via an upper path set $x_i = 0$, otherwise set $x_i = 1$. Since paths are node-disjoint, S_j has to connect to T_j either through a lower or an upper path (whichever one is unoccupied), but then

it is easy to see that clause C_j is satisfied by the corresponding literal. Lastly observe that the construction can be done in polynomial time. □

(c) To conclude $L_{\text{NODE-DISJ}}$ is NP-complete, it suffices to show that $L_{\text{NODE-DISJ}} \in$ NP. The certificate for $L_{\text{NODE-DISJ}}$ is a collection of k paths. This certificate is clearly of polynomial size in the size of the input. It can be verified in polynomial time that the collection of paths is node-disjoint, and that for each i s_i is connected to t_i via a path from the collection.

(d) Edge-disjoint paths problem is also NP-complete: (1) it is easy to see by a similar argument to the previous part that it is in NP, and (2) a reduction similar to the one in part (b) works for edge-disjoint paths. Expand each node u_i^j into $u_{i,in}^j$ and $u_{i,out}^j$ with a single edge $(u_{i,in}^j, u_{i,out}^j)$ between the two newly introduced nodes. Every edge pointing into u_i^j, now points to $u_{i,in}^j$, and every edge pointing out of u_i^j, now points out of $u_{i,out}^j$. Perform a similar modification on ℓ_i^j nodes — it is easy to see that this modification still results in a polynomial time reduction and ϕ is satisfiable if and only if we have $n + m$ edge-disjoint paths in the newly constructed graph.

3. (20 pts) Given an undirected graph $G = (V, E)$, a k-coloring of G is a function $c : E \to [k]$ such that for every edge $\{u, v\} \in E$ we have $c(u) \neq c(v)$. If G has a k-coloring, G is called k-colorable.

(a) Since G is 2-colorable if and only if each connected component of G is 2-colorable, it suffices to design an algorithm for the case when G is connected (for general graphs, we can just run our algorithm on each connected component). Hence, we assume G is connected. Pick arbitrary vertex s. Run BFS to compute shortest unweighted paths from s. Label vertex u with the parity of unweighted distance from s to u, i.e., label of u is $dist(s, u) \mod 2$. For each edge $\{u, v\} \in E$ check that u and v have different labels. If so, output 2-colorable, otherwise output NOT 2-colorable.

Clearly, if the algorithm outputs 2-colorable, it is correct, since it actually constructs 2-coloring. So we only need to argue that if the algorithm says not 2-colorable then it is correct as well. Note that the algorithm outputs not 2-colorable if and only if there exists two vertices that are at the same parity distance from s. Call these vertices u_1 and u_2. Then consider the following cycle in G: $s \rightsquigarrow u_1 \to u_2 \rightsquigarrow s$, where $\rightsquigarrow$ refers to the path from s to a given vertex in the BFS tree, and $\to$ refers to an edge. Note that this cycle is of odd length, and no cycle of odd length can be 2-colored. This finishes the proof of correctness.

The runtime is $O(|V| + |E|)$ since BFS takes linear time, and checking the validity of coloring takes linear time as well.

(b) $L_{\text{SCHEDULE}} = \{\langle F_1, \ldots, F_k, S_1, \ldots, S_\ell, h\rangle \mid \exists \text{valid schedule with } \leq h \text{ time slots}\}$

Claim 0.4. *$L_{SCHEDULE}$ is NP-complete.*

Proof. (1) L_{SCHEDULE} is in NP — the certificate is the schedule mapping each exam to a time slot in $\{1, \ldots, h\}$. Note this certificate is of size polynomial in the size of the input, and it can be verified in polynomial time — we need to check that no two exams corresponding to each student are scheduled at the same time.

(2) We show how to reduce L_{COL} to L_{SCHEDULE} in polynomial time. Given an instance $\langle G, k\rangle$ we create an instance of L_{SCHEDULE} as follows. Let $G = (V, E)$ where $V = \{1, \ldots, n\}$ and $m = |E|$. We create n exams $F_1, \ldots, F_n$ and m students $S_1, \ldots, S_m$. Thus, each exam corresponds to a node in G and each student i corresponds to an edge $\{u_i, v_i\} \in E$. Each student has exactly two exams. Let student i have exams F_{u_i} and F_{v_i}. We claim that $\langle F_1, \ldots, F_n, S_1, \ldots, S_m, k\rangle$ is in L_{SCHEDULE} if and only if $\langle G, k\rangle$ is in L_{COL}. Note that schedule mapping $\{F_1, \ldots, F_n\}$ to $\{1, \ldots, k\}$ can be interpreted as a coloring of nodes $\{1, \ldots, n\}$ with colors $\{1, \ldots, k\}$, and vice versa. The condition that no student has a conflict is equivalent to the condition that coloring is valid, i.e., no edge is colored with the same color. Clearly, constructing the instance $\langle F_1, \ldots, F_n, S_1, \ldots, S_m, k\rangle$ takes polynomial time. □

4. If you squint hard enough you can recognize this problem to be the dictionary problem from term test 1. Now, the dictionary is implemented with algorithm A, and string w is in $L^\dagger$ precisely when it can be split into the words from the dictionary.

   (a)
   $$D[i] = \begin{cases} 1 & \text{if } w[1..i] \in L^\dagger, \\ 0 & \text{otherwise.} \end{cases}$$
   The answer is in $D[n]$.

   (b)
   $$D[i] = \bigvee_{k=0}^{i-1} D[k] \wedge A(w[k+1..i])$$
   $$D[0] = \text{TRUE}$$

   (c) Note $w[1..i]$ belongs to $L^\dagger$ if and only if there is some last word from L appearing as a substring of $w[1..i]$. This means that $w[1..k] \in L^\dagger$ and $w[k+1..i] \in L$ for some $k \in \{0, \ldots, i-1\}$. This justifies the first line of the computational array. If $i = 0$ then $w[1..i]$ is the empty string, which always belongs to $L^\dagger$. This justifies the second line of the computational array.

   (d) Let $T(n)$ denote the runtime of A on inputs of length $\leq n$. Note that A is invoked $O(n^2)$ times on inputs of length $\leq n$ when we fill out the above DP table. Thus, the overall runtime is $O(n^2 T(n))$.

5. (20 pts) **Minesweeper on Graphs**

   Let G be an undirected graph. Consider the following version of the game Minesweeper. Each node in G is either empty or contains a single hidden mine. If the player clicks on a node with a mine, the player loses the game. If the player clicks on a node without a mine, the node is labeled with the number of mines contained in the adjacent (neighboring) nodes. The regular Minesweeper game is a special case played on the grid graph.

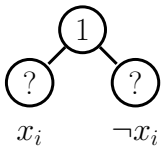
   Now, consider *mine consistency problem*. You are given a graph G, in which some nodes are labeled with numbers and other nodes are unlabeled. The goal is to decide if it is possible to place mines on some of the unlabeled nodes such that all labels are correct, that is if node v is labeled with number k then it has exactly k neighbors with mines.

(a) $L_{\text{MINE}} = \{\langle G, \ell\rangle \mid \exists$ a placement of mines consistent with labels $\ell\}$.

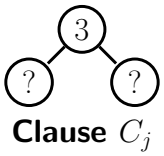
(b)

Claim 0.5. *$L_{3\text{-}SAT} \leq_p L_{MINE}$*

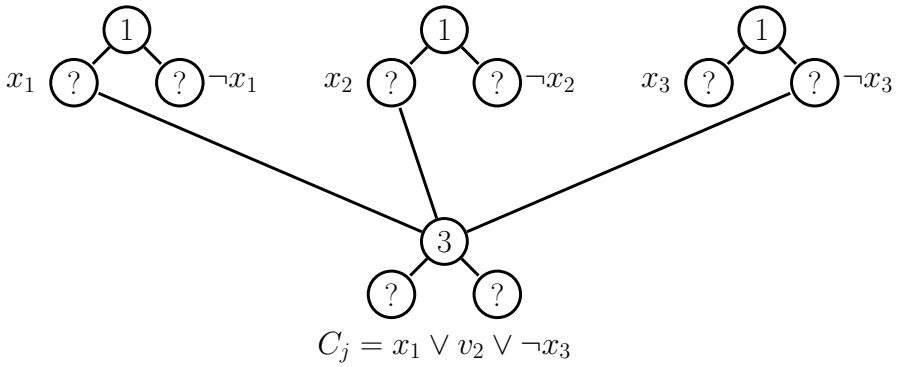
Proof. Given ϕ — a 3-CNF with n variables and m clauses, we show how to construct a labeled graph (instance of L_{MINE}) in polynomial time such that there is a placement of mines consistent with the labels if and only if the original formula ϕ is satisfiable. For each variable x_i introduce a variable gadget – three nodes such that two are unlabeled (we think of them as literals x_i and $\neg x_i$) connected to the third node labeled with 1. Thus, we can think of this gadget as follows: labeled node forces exactly one of its neighbors to contain a mine. The mine will be placed on the node x_i in case $x_i = 1$ in the satisfying assignment, and it will be placed on the node $\neg x_i$ if $x_i = 0$ in the satisfying assignment.



For each clause C_j we introduce a similar gadget, but the top node is labeled with 3 now:



We connect the top node of the gadget corresponding to C_j to the bottom nodes of the corresponding literals in the variable gadgets. For example, if $C_j = x_1 \vee x_2 \vee \neg x_3$ we get the following subgraph:



Clearly, this construction can be done in polynomial time. It is left to show that this instance has consistent placement of mines if and only if ϕ is satisfiable. Suppose that ϕ is satisfiable by assignment x. If $x_i = 0$ place a mine in the variable gadget i at the node corresponding to $\neg x_i$, otherwise place a mine at the node corresponding to x_i. Clause C_j can be satisfied by 1,2, or 3 literals. If clause C_j is satisfied by 3 literals, then placement of mines is already consistent with the clause gadget. If C_j is satisfied by 2 literals, then we place one more mine at one of the bottom nodes of the clause gadget to satisfy the label of the top node in the clause gadget. Lastly, if C_j is satisfied by 1 literal, then we place two mines at the unlabeled nodes of the clause gadget. This way we constructed a placement of mines consistent with all labels. It is now easy to see the other direction. Suppose that there is a placement of mines consistent with the labels. Then we can define as assignment x by setting x_i to 1 if and only if node corresponding to x_i has a mine in the variable gadget. This assignment satisfies every clause C_j, because the top node of the gadget corresponding to this clause has to have a neighboring mine from one of the variable clauses, i.e., assignment x satisfies clause C_j.

□

(c) It is left to show that L_{MINE} is in NP. The certificate is the placement of mines. Note that this certificate is polynomial in the size of the input, and it can be verified in polynomial time – for each labeled node we simply need to count the number of mines placed in the adjacent nodes and check that the label matches this number.