1. **Set Packing.** The set packing decision problem is defined as follows:

**SetPack:** Given a universe set $U$, a set of subsets $F = \{S_j \mid S_j \subseteq U, 1 \leq j \leq m\}$, and an integer $k$, does there exist $C \subseteq F$ with $|C| \geq k$ such that no two distinct elements $S_i, S_j \in C$ intersect (i.e., for all $S_i, S_j$ in $C$ with $S_i \neq S_j$ we have $S_i \cap S_j = \emptyset$)?

(a) Denote the independent set decision problem by $\text{IndepSet}$. Show $\text{IndepSet} \leq_p \text{SetPack}$.

(b) Define $\text{searchSetPack}$ to be the search problem for set packing. That is, given $U$ and $F$ as in the Set-Packing decision problem, find a subset $C \subseteq F$ such that $|C|$ is the maximum possible and no two distinct elements in $C$ intersect.

Prove that $\text{searchSetPack} \leq_p \text{SetPack}$.

**Hint 1:** You need to first find $k^*$, the maximum possible size $|C|$. Then find the elements of $C$.

**Hint 2:** In the second stage of the algorithm, where you build up a solution $C$, it is useful to write a loop invariant stating that the current solution “is promising” (i.e., in the same sense that we used for proving the correctness of greedy algorithms).

---

**Solution for Q1:**

1a **Soln.** We will show that the set packing problem is (more or less) a generalization of the independent set problem.

The input provided to $\text{IndepSet}$ is an undirected graph $G = (V, E)$ and an integer $k$. We will measure the size of this input to be $|s| = |V| + |E|$ (students can define anything reasonable here, but it is critical to specify $|s|$ in order to make the argument that various algorithms are poly-time in $|s|$).

The independent set problem concerns the existence of a subset of the vertices $C \subset V$ such that $|C| \geq k$ and no two vertices in $C$ share an edge. Note that we can express this latter constraint in terms of the sets of edges covered by each vertex, i.e.,

$$S_v = \{e = (u, v) \mid \text{for some } u \in V \text{ and } e \in E\}. \quad (1)$$

This observation forms the basis of the poly-time reduction.

One difficulty we might anticipate in using these $S_v$’s as the individual sets in a packing is that there can be multiple empty sets in this construction. Indeed $S_v = \emptyset$ for any vertex $v$ that is not an endpoint for any edges in $E$. Moreover, multiple such vertices can participate in a vertex cover. For example, if we define $F$ to be the set of all such $S_v$ then, in the case of a graph with no edges, we have $F = \{\emptyset\}$ and so $|F| = 1$.

Meanwhile, the maximum vertex cover for this case consists of all $V$. Another possible difficulty is that for cases in which $u, v \in V$ with $u$ and $v$ endpoints of only the one edge $e = (u, v)$, then $S_u = S_v = \{e\}$. Again, the sets $S_v$ would not be in one to one correspondence with the set of vertices.

It is therefore convenient to include specific vertices in each of the subsets to be used for packing. That is, we define

$$F_v = \{v\} \cup S_v, \text{ and } F = \{F_v \mid v \in V\}. \quad (2)$$

Here there is exactly one element of $F$ for every $v$, so $|F| = |V|$. And we define the universe set to be $U = V \cup E$.

**Claim:** For the above construction of $U$ and $F$ we have $\text{IndepSet}(G, k)$ iff $\text{SetPack}(U, F, k)$. Note we use the same $k$ for both problems.
Assuming this Claim for the moment note that, given the input \((G, k)\) of the \text{IndepSet} problem, we can construct \(U\) and \(F\) in at most \(O(|V||E|)\) time. By our definition of \(|s|\) above this is bounded by \(O(|s|^2)\) time. In addition, according to the claim, we require only one call to \text{SetPack}(U, F, k) to determine the answer for \text{IndepSet}(G, k). Therefore, by the definition of a poly-time reduction, we have \text{IndepSet}(G, k) \leq_p \text{SetPack}(U, F, k), as desired.

**Proof of Claim:**

\(\implies\) Suppose \text{IndepSet}(G, k) is true. Let \(C \subseteq V\) be an independent set of size \(|C| \geq k\). Define \(F_C = \{v \cup S_v \mid v \in C\}\). Then by (2) we have \(|F_C| = |C| \geq k\). Let \(F_a\) and \(F_b\) be any two distinct elements in \(F_C\). Then \(a \neq b\) and, since \(a, b \in C\) and \(C\) is an independent set, there are no edges in \(E\) which have both \(a\) and \(b\) as endpoints. Therefore \(S_a \cap S_b = \emptyset\). It now follows from (2) that \(F_a \cap F_b = \emptyset\). Therefore \(F_C\) is a set packing, and we’ve already shown \(|F_C| = |C| \geq k\), so \text{SetPack}(U, F, k) is true.

\(\Leftarrow\) Suppose \text{SetPack}(U, F, k) is true. Then there exists a set-packing of size at least \(k\). Since each element of \(F\) is uniquely identified with one element of \(V\), we can without loss of generality write such a set packing as \(F_C = \{v \cup S_v \mid v \in C\}\) where \(C \subseteq V\) and \(|C| = |F_C| \geq k\). Given any two distinct vertices \(a\) and \(b\) in \(C\), \(F_a\) and \(F_b\) are distinct elements in \(F\) and, by the definition of set packing, we know \(F_a \cap F_b = \emptyset\). Now by the definition of \(F_v\), this implies \(S_a \cap S_b = \emptyset\). That is, \(a\) and \(b\) cannot be endpoints of the same edge in \(E\). Since this is true for any such \(a \neq b\) in \(C\), we conclude that \(C\) is an independent set. Recall \(|C| = |F_C| \geq k\), so \(C\) is an independent set of \(G\) of size at least \(k\). Therefore \text{IndepSet}(G, k) is true.

1b Soln. We wish to have \text{searchSetPack}(U, F) return a maximum size subset \(M \subseteq F\) such that, for each distinct pair of elements \(M_i\) and \(M_j\) in \(M\), we have \(M_i \cap M_j = \emptyset\).

Define the size of the input to be \(|s| = |U| + |F|\) (again, make any reasonable choice, but make that choice clear). By using binary search on \(k\) and calling \text{SetPack}(U, F, k) at most \(O(\log(|F|)) \subset O(\log(|s|))\) times, determine the maximum size \(k^* = |M|\) for a set-packing \(M\) (details omitted). Given this \(k^*\) we then execute the following:

- Suppose \(F = \{F_1, F_2, \ldots, F_m\}\), in any order, where \(m = |F|\).
- Initialize \(M \leftarrow \emptyset\) and \(k \leftarrow k^*\)
  - for \(j = 1:\ldots m:\)
    - # Loop Invariant: LI\((j)\): There exists a maximum set-packing \(T \subset F\), with \(|T| = k^*\)
    - # such that, \(F_i \in T\) iff \(F_i \in M\) for \(1 \leq i < j\) (i.e., \(M\) is promising).
    - # Moreover \(|M| = k^* - k\).
    - if \(k == 0:\)
      - break # No more elements to find.
    - \(C \leftarrow (\bigcup_{M_i \in M} M_i)\) # Elements covered so far.
    - if \(F_j \cap C = \emptyset:\)
      - # Determine whether there exists a solution \(T' = M \cup F_j \cup T'_{j+1}\) of size \(k^*\),
      - # with \(T'_{j+1} \subseteq \{F_{j+1}, \ldots, F_m\}\) and \(|T'_{j+1}| = k^* - |M| - 1 = k - 1\).
      - \(C' \leftarrow C \cup F_j\)
      - \(F' = \{F_i \mid j < i \leq m\text{ and }F_i \cap C' = \emptyset\}\).
      - if \text{SetPack}(U, F', k - 1):
        - \(k \leftarrow k - 1\)
        - \(M \leftarrow M \cup F_j\)
  - return \(M\)

Here we only sketch the remainder of the proof. We use induction to prove the loop invariant \(LI\((j)\) for \(j = 0, 1, \ldots, J\), where \(J\) is the index \(j\) for which the break statement is executed or, if the break is not executed, then \(J = m + 1\). In the latter case, we define \(LI\((m + 1)\) as the loop invariant after the \(m^{th}\) execution of the loop.
The proof of the loop invariant follows an “is promising” style proof (as discussed in the lectures on greedy algorithms). In the situations where you need to switch from the optimal solutions $T$ in the loop invariant $LI(j)$ to a new optimal solution (i.e., “switch horses” in mid-proof) use the $T'$ described in the algorithm above.

2. 3D Matching. We consider the following two types of 3D matching problems:

- **partial3DM**: Given three distinct sets $X$, $Y$, and $Z$, with $|X| = |Y| = |Z| = n$, a set of triples $T \subseteq X \times Y \times Z$, and an integer $k$, does there exist a subset of triples $C \subseteq T$ of size $|C| \geq k$ such that no two distinct elements $C_i, C_j \in C$ have any element in common (i.e., if $C_i = (C_{i,1}, C_{i,2}, C_{i,3})$ and $C_j = (C_{j,1}, C_{j,2}, C_{j,3})$ are distinct triples in $C$ then, for each $p = 1, 2, 3$, we have $C_{i,p} \neq C_{j,p}$)?

- **perfect3DM**: The input is similar to partial3DM except no integer $k$ is input for this problem. The question here is whether there exists a subset of triples $C \subseteq T$ such that $|C| = n$ and no two distinct elements $C_i, C_j \in C$ have any of their three elements in common? (That is, the matching is perfect in the sense that each element of $X$, $Y$, or $Z$ is covered by exactly one triple in $C$.)

Note that Wikipedia defines the “3D matching problem” to be the problem partial3DM above, while the Kleinberg and Tardos text defines it to be perfect3DM. Moreover, in your answers below you can use the fact that 3-SAT $\leq_p$ perfect3DM, which is proved in the Kleinberg and Tardos text.

(a) Show perfect3DM $\leq_p$ partial3DM.

(b) Show partial3DM $\leq_p$ SAT by using an encoding of the constraints for 3D matching in terms of a CNF formula. Use the binary variables $x_i$, where $x_i$ is true iff the $i^{th}$ triple in $T$ is to be included in the set $C$. (Note that we are asking simply for a reduction to SAT, not to 3-SAT.)

(c) Given the above results can you conclude that partial3DM or perfect3DM is NP-complete? Explain.

Solution for Q2:

2a Soln. In all subproblems below define $|s|$, the size of the input, to be $|s| = |X| + |T| + |U|$ (specifically, we are assuming the universe set is finite).

Note the value of perfect3DM$(X, Y, Z, T)$ is the same as partial3DM$(X, Y, Z, T, k)$ with $k = |X| = n$. Since $U$ is finite, we can compute $|X|$ in poly-time, and this shows the desired result.

2b Soln. Consider the problem partial3DM$(X, Y, Z, T, k)$, and the binary variables $x_i$, where $x_i$ is true iff the $i^{th}$ triple in $T$ is to be included in the set $C$. WLOG we assume that $k \leq |T| = m$.

We next encode the constraint that, for $i \neq j$, $x_i$ and $x_j$ cannot both be true if the two triples $T_i$ and $T_j$ intersect (that is, for some $p \in \{1, 2, 3\}$, we have $T_{i,p} = T_{j,p}$). This is done by including the constraint $\neg(x_i \land x_j)$, which is logically the same as the conjunctive clause $(\bar{x}_i \lor \bar{x}_j)$. The eventual SAT formula will include such a clause for every pair of $T_i$ and $T_j$ which intersect. There are at most $O(|T|^2)$ disjunctive clauses, and each such clause can be computed in constant time.

The conjunction of all the clauses computed so far will guarantee that the set of triples selected according to $x_i$ being true correspond be a set $C$ of triples which satisfy the pairwise non-intersection property of the 3D matching problem. The remaining constraint we need to encode is that $|C| \geq k$. That is, for $i = 1, 2, \ldots, m$, at least $k$ of the logical variables $x_i$ must be true.

One (unsuitable) attempt to do this is as follows. Consider the conjunction of clauses that involve combinations of $m - k + 1$ distinct variables $x_i$. If we assume at least $k$ variables $x_i$ are true (over $i = 1, \ldots, m$) then, given any selection of $m - k + 1$ such variables, at least one of the variables in the selection must be true. This can be expressed as a simple disjunction over all the selected variables. Moreover, we can obtain a necessary and sufficient condition for at least $k$ variables to be true by taking the conjunction over all these disjunctive clauses. However there will be $\binom{m}{m-k+1}$ such disjunctive clauses or, equivalently, $\binom{m}{k-1}$ such clauses (see binomial coefficients).
While this approach does give a SAT formula that is satisfiable if and only if $\text{partial3DM}(X, Y, Z, T, k)$ should return true, this reduction algorithm does not necessarily run in poly-time. Indeed, the number of clauses in the SAT formula produced is not bounded by a polynomial in $|s|$. Specifically, $(2m^3/m^{1/2})$ grows like $\Omega(2^m/m^{1/2})$ for $k - 1 = m/2$ (look up lower bounds for binomial coefficients and Stirling’s formula). The number of clauses therefore can grow exponentially with $m$ (and exponentially with $|s|$), so this is not a poly-time reduction.

How can we more efficiently encode the condition that at least $k$ of the binary variables must be true? Consider a $k \times m$ assignment matrix $A$, where the $i,j$ element of $A$ is a logical variable $a_{i,j}$. The $i^{th}$ row of $A$ effectively specifies which of the $m$ logical variables is to be considered the $i^{th}$ logical variable $x_{(i)}$ that must be true (according to $A$). We require $A$ to have exactly one variable in each row that is true, and at most one variable in each column of $A$ is true. We will show how to write SAT constraints for $A$ further below.

For the moment, assume we can build any such $A$. Then, for each $j = 1, 2, \ldots, m$, we consider the implication $(a_{1,j} \lor \ldots \lor a_{k,j}) \implies x_j$. Such an implication forces $x_j$ to be true whenever the $j^{th}$ column of $A$ has at least one $a_{i,j}$ which is true. Moreover, the constraints on $A$ force $k$ different columns to have one true value, and therefore at least $k$ different $x_j$’s will be forced to be true. Finally, although this is not critical, by using the logical implication we will allow additional $x_j$’s to be true as well.

Note that $(a_{1,j} \lor \ldots \lor a_{k,j}) \implies x_j$ is equivalent to $x_j \lor (\neg(a_{1,j} \lor \ldots \lor a_{k,j}))$, which in turn is equivalent to the conjunction of the 2-clauses $(x_j \lor \bar{a}_{i,j})$ for all $1 \leq i \leq k$. Thus there are $m$ such implications, and each of these implications can be written as the conjunction of $k$ clauses of size 2, so there are $O(m^k)$ such clauses.

In order to write the constraints on $A$ note that we can enforce at least one value in the $i^{th}$ row of $A$ to be true by including the single clause $(a_{i,1} \lor \ldots \lor a_{i,m})$. Next, to enforce the constraint that exactly one variable in the $i^{th}$ row is true we only need to add the constraint that at most one variable in $\{a_{i,j}\}_{j=1}^{m}$ is true. This is equivalent to the condition that $\neg(a_{i,p} \land a_{i,q})$ is true or, equivalently, $(a_{i,p} \lor a_{i,q})$ is true, for all $p \neq q$ with $1 \leq p, q \leq m$. There are $O(m^2)$ such clauses for each of the $k$ rows of $A$. Finally, the constraint that at most one variable in each column of $A$ is true generates $O(k^2)$ more clauses of the form $(\bar{a}_{p,j} \lor \bar{a}_{q,j})$ for each column $j = 1, 2, \ldots, m$.

Since $k \leq m$, the total number of clauses is $O(m^3) \subset O(|s|^3)$, and these can all be generated in poly-time with respect to $|s|$.

We omit the argument that the resulting SAT problem is satisfiable iff the original decision problem $\text{partial3DM}(X, Y, Z, T, k)$ is true.

2c Soln. First, note that we still need to show that $\text{perfect3DM}$ and $\text{partial3DM}$ are in NP. This follows by using a solution to the problem as a certificate $t$, and then showing that: a) the certificate $t$ is bounded in size by a polynomial in $s$; and b) a certifier which checks the solution $t$ runs in poly-time with respect to $|s|$. We skip the details.

Now the facts that $\text{3-SAT}$ is NP-complete and $\text{3-SAT} \leq_P \text{perfect3DM}$, which is proved in the Kleinberg and Tardos text, together imply $\text{perfect3DM}$ is NP-complete. The result in (2a) above then shows that $\text{partial3DM}$ is also NP-complete.

3. Max Degree 12 Spanning Tree. Show the following problem is NP-complete:

Degree12Tree: Given an undirected graph $G = (V, E)$, does there exist a subgraph $T = (V, F)$ of $G$ (i.e., with $F \subseteq E$) such that $T$ is a spanning tree of $G$ and the degree of every vertex in $T$ is at most 12?

Note, the degree of a vertex $v$ in the graph $(V, F)$ is defined to be the number of edges in $F$ that have $v$ as one of their endpoints.

Hint: Consider making use of the Hamiltonian path decision problem.

Solution for Q3:
Soln Q3, Step 1: Degree12Tree is in NP: Degree12Tree(G) (or D12T for short) is a decision problem. The input is an undirected graph \( G = (V, E) \). We define the input string \( s \) to be this graph \( G \) and we take the size of the input to be \( |s| = |V| + |E| \).

Define the certificate \( t \) to describe a spanning tree \( T = (V, F) \) of \( G \) with degree \( \leq 12 \). Such a certifier exists if \( s \in D12T \). Moreover, \( |t| = |V| + |F| \leq |s| \), and the conditions on \( t \) can be checked in \( O(|s|) \) time (say by doing a modified breadth first search on \( T \) to verify \( T \) spans \( V \) and \( T \) is acyclic). Therefore there exists a polytime certifier for the decision problem D12T, hence it is in \( NP \).

Soln Q3 Step 2: Choose a NP-complete problem \( X \): We will use the undirected Hamiltonian path problem (uHP), which is known to be NP-complete (see Wikipedia). We wish to show \( uHP(G) \approx_p D12T(G) \).

Soln Q3 Step 3: Show a poly-time reduction of \( X \) to Degree12Tree: Given the undirected graph \( G = (V, E) \) we create a new graph \( G' = (V', E') \) for the D12T problem. For every vertex \( u \in V \), add 10 additional vertices \( \{w_k(u) \mid k = 1, \ldots, 10\} \) along with 10 additional edges \( (u, w_k(u)) \), \( k = 1, \ldots, 10 \). These new edges are the only edges to each of the vertices \( w_k(u) \). That is, \( u \) is the root of a subtree of height one, which consists of the vertex \( u \) and these 10 new vertices \( \{w_k(u), k = 1, \ldots, 10\} \) as leaves. Define

\[
W = V \cup \{w_k(u) \mid u \in V \text{ and } 1 \leq k \leq 10\}
\]

Define the set of edges \( E' \) to be

\[
E' = E \cup \{(u, w_k(u)) \mid u \in V \text{ and } 1 \leq k \leq 10\}.
\]

This construction of \( G' \) from the original graph \( G \) requires \( O(|V|) \subseteq O(|s|) \) time.

Claim 1. For \( G' \) as defined above, \( D12T(G') \) is true iff \( uHC(G) \) is true.

Notice that the desired result, namely \( uHP \approx_p D12T \), will follow from Claim 1 since the construction of \( G' \) is poly-time and we then require only one call to \( D12T(G') \).

Proof of Claim 1. First assume \( uHP(G) \) is true, and \( P \) is a Hamiltonian path with endpoints \( v_1 \) and \( v_2 \) in \( V \). Note that \( P \) also forms a spanning tree of \( G \) with degree at most 2. Moreover, since it is a tree, \( P \) has exactly \( |V| - 1 \) edges.

Next we consider the implication that the existence of \( P \) has on the problem \( D12T(G') \). Consider the subgraph \( (V', T') \) of \( G' \) formed from the vertices and edges in \( P \) along with the \( 10|V| \) additional vertices \( w_k(u) \) and edges \( (u, w_k(u)) \), for each \( u \in V \) and each \( k = 1, \ldots, 10 \). By construction, the addition of these extra edges implies \( (V', T') \) has at most degree 12 (since \( P \) has at most degree 2). Also, the number of edges in \( T' \) is the sum of \( |V| - 1 \) (i.e., the number of edges in \( P \)) plus \( 10|V| \) (the number of added edges). That is, \( |T'| = 11|V| - 1 \). Similarly, the number of vertices in \( (V', T') \) is \( |V| + 10|V| = 11|V| \). Moreover, since every vertex \( w_k(u) \) is connected to \( u \in V \), and the path \( P \) spans \( V \), we conclude that \( (V', T') \) must be connected. Therefore \( (V', T') \) is a connected graph with \( |T'| = |V'| - 1 \) edges, and it follows that \( (V', T') \) must be a spanning tree of \( G' \). Therefore \( D12T(G') \) must be true.

Conversely, suppose \( D12T(G') \) is true. Let \( (V', T') \) be any spanning tree of \( G' \) of degree less than or equal to 12. By the nature of the construction of \( G' \), we know \( (V', T') \) must contain each of the subtrees rooted at \( u \in V \) and also all the edges \( \{(u, w_k(u)) \mid u \in V \text{ and } 1 \leq k \leq 10\} \) that we introduced above. Define \( (V, F) \) to be the subgraph of the original graph \( G \) formed by deleting from \( (V', T') \) all the new vertices in \( V \setminus V \) and all the edges \( \{(u, w_k(u))\} \). We will show that \( (V, F) \) is a Hamiltonian path.

From \( (V', T') \) and the above deletion process it follows that the degree of every vertex in \( (V, F) \) is at most 2. Moreover, since \( (V', T') \) is connected, it follows that for every \( u, v \in V \) there exist a (unique) simple path \( Q \) from \( u \) to \( v \) in \( (V', T') \). Such a path cannot contain any one of the added vertices \( w_k(z) \), since these are leaf nodes in \( (V', T') \). Thus \( Q \) is a simple path in \( (V, F) \). Moreover, since we began with an arbitrary pair of points \( u, v \in V \), it follows that \( (V, F) \) is connected.
Next we consider the number of edges in \((V, F)\). Note that \(|F| = |T'| - 10|V|\), since we removed the edge to each \(w_k(z)\). But, from above, \(|T'| = 11|V| - 1\) so it follows that \(|F| = |V| - 1\). And, since we know \((V, F)\) is connected with \(|V| - 1\) edges, it must be a spanning tree.

In summary, we have shown that \((V, F)\) is a spanning tree of \(G\) with a maximum degree of 2. By counting edges and noting that connectivity implies that there are no degree zero vertices in \((V, F)\), it follows that there must be exactly two vertices \(u_1\) and \(u_2\) in \((V, F)\) with degree 1, and all the other vertices must have degree equal to 2. (Here we are assuming \(|V| \geq 2\).) Moreover, by connectivity, there must be a unique simple path \(P\) in \((V, F)\) between \(u_1\) and \(u_2\).

Next we show that the path \(P\) constructed in the previous paragraph must visit each vertex \(v \in V\) exactly once. Since the \(P\) constructed above is edge-simple, and the degree of each vertex in \((V, F)\) is at most 2, \(P\) must also be vertex-simple. Therefore \(P\) visits each vertex on \(P\) at most once.

But does \(P\) need to visit every vertex in \(V\)? Suppose \(v \in V\) and \(v\) is not on the path \(P\). Since \((V, F)\) is a spanning tree, there exists a unique simple path, say \(Q\), from \(u_1\) to \(v\) in \((V, F)\). Since we are assuming \(v \notin P\), there must be a first edge at which the \(u_1\) to \(u_2\) path \(P\), and the \(u_1\) to \(v\) path \(Q\), deviate. Since the degree of \(u_1\) is one, the paths \(P\) and \(Q\) must be the same over this first edge out of \(u_1\). Moreover, \(Q\) cannot contain the vertex \(u_2\) since the degree of \(u_2\) is also one (and \(Q\) is simple). Let \(w\) be first vertex on \(P\) after which the path \(Q\) chooses a different edge (such a vertex must exist assuming \(v\) is not on \(P\)). But in this case the degree of \(w\) must be at least three (an “incoming” edge on the path from \(u_1\) and two “outgoing” edges, one in the direction of \(u_2\) along \(P\) and another in the direction of \(v\) along \(Q\)). This contradicts \((V, F)\) having degree at most two. Therefore every vertex \(v \in V\) must be on \(P\).

We conclude that \((V, F)\) must be a Hamiltonian path for \(G\).

This completes the proof of Claim 1 and also Step 3.