Answer the questions in the boxes provided on the question sheets. If you run out of room for an answer, add a page to the end of the document.

Name: Riya Kore Wisc id: rykore

1. *Kleinberg, Jon. Algorithm Design (p. 512, q. 14)* We've seen the Interval Scheduling Problem several times now, in different variations. Here we'll consider a computationally much harder version we'll call *Multiple Interval Scheduling*. As before, you have a processor that is available to run jobs over some period of time.

   People submit jobs to run on the processor. The processor can only work on one job at any single point in time. Jobs in this model, however, are more complicated than we've seen before. Each job requires a *set* of intervals of time during which it needs to use the processor. For example, a single job could require the processor from 10am to 11am and again from 2pm to 3pm. If you accept the job, it ties up your processor during those two hours, but you could still accept jobs that need time between 11am and 2pm.

   You are given a set of $n$ jobs, each specified by a set of time intervals. For a given number $k$, is it possible to accept at least $k$ of the jobs so that no two accepted jobs overlap in time?

   Show that Multiple Interval Scheduling is NP-Complete.

**Solution:**

1. Multiple Interval Scheduling is in the class NP.
Given a collection of jobs $J_1, ...., J_k$, for each job $J_i$, examine the intervals within the job and mark the processor as occupied during each interval. If an attempt is made to mark the processor during an interval that is already flagged, the solution is deemed invalid. Validity of the certificate is established if all k jobs are iterated through without attempting to mark the same time interval twice. This process takes, at most, time equal to the number of available time intervals, as each interval is flagged at most once in the worst case.

2. The Multiple Interval Scheduling (MIS) problem is NP-Hard. (Independent Set $\leq_p$ MIS)
An arbitrary instance of the Independent Set is defined by a graph G.

- For each edge $e_j$ in G, generate a corresponding time interval $t_j$.
- For each node $v_i$ in G, create a job $J_j$ that utilizes the intervals $t_j$ corresponding to edges adjacent to $v_i$.

If there exists an independent set of size k, we can select k nodes from G such that no edge is adjacent to more than one node in our set. Let $J_j$ be part of our candidate set of non-conflicting jobs if and only if $v_i$ is in this independent set. Due to the construction, each time interval $t_j$ corresponds to an edge in G, and only one of its adjacent nodes in G could have been selected. Consequently, only the jobs corresponding to those nodes require $t_j$. As we only selected one of those, our candidate set of jobs is non-conflicting.
If there exists a set of non-conflicting jobs of size k, then by the same logic as above, the set of nodes $v_i$ corresponding to the jobs $J_i$ forms an independent set in G. We have established a polynomial time mapping from Independent Set to Multiple Interval Scheduling, such that we have a positive instance of Independent Set if and only if we have a positive instance of MIS. Since Independent Set is NP-complete, Multiple Interval Scheduling is NP-Hard.

2. *Kleinberg, Jon. Algorithm Design (p. 519, q. 28)* Consider this version of the Independent Set Problem. You are given an undirected graph $G$ and an integer $k$. We will call a set of nodes $I$ "strongly independent" if, for any two nodes $v, u \in I$, the edge $(v, u)$ is not present in $G$, and neither is there a path of two edges from $u$ to $v$, that is, there is no node $w$ such that both $(v, w)$ and $(u, w)$ are present in $G$. The Strongly Independent Set problem is to decide whether $G$ has a strongly independent set of size at least $k$.

   Show that the Strongly Independent Set Problem is NP-Complete.

**Solution:**

1. Strongly Independent Set is in the class NP.
Given a set of nodes I, we can efficiently verify if any nodes $v, u \in I$ are in close proximity. For each $u \in I$, a polynomial time BFS can determine if any other nodes are at a distance of 1 or 2 edges away. This process takes a total time of O(nm).

2. SIS is NP-Hard. (Independent Set $\leq_p$ SIS).
An arbitrary instance of Independent Set is represented by a graph G. To construct a graph G" as a Strongly Independent Set instance:

- For each node $v_i$ in G, include $v_i$ in G".
- For each edge $e_j = (u, v)$ in G, introduce a node $w_j$ in G" and establish edges $(u, w_j)$ and $(w_j, v)$.
- Finally, introduce an edge between each pair of added $w$ nodes.

If there exists a size k independent set, then the same set of nodes forms a size k strongly independent set in G'. As each edge in G is divided into two edges in G', if two nodes were not adjacent in G (and both had at least one adjacent edge), the distance between them is now 3 $(u - w_i - w_j - v)$.
If there exists a size k strongly independent set in G' that does not use any of the added $w$ nodes, it follows from construction that they constitute an independent set in G. It remains to show that no nodes w can be part of a strongly independent set. Without loss of generality, consider a specific such node $w$. Since $w$ is adjacent to all other $w$ nodes, it can only be part of a strongly independent set if no other $w$ node is, and additionally, no other node adjacent to a $w$ node is. In other words, $w$ can only be part of a strongly independent set if it is the only node in the set (excluding nodes with no adjacent edges at all). Note that if there exists a strongly independent set that includes $w$, there must be a second one that includes $w$ nodes by replacing $w^*$ with any non-$w$ node that has at least one adjacent edge.
We have a polynomial time mapping from Independent Set to Strongly Independent Set, where we have a positive instance of IS if and only if we have a positive instance of SIS. Since Independent Set is NP-complete, Strongly Independent Set is NP-Hard.

3. *Kleinberg, Jon. Algorithm Design (p. 527, q. 39)* The *Directed Disjoint Paths Problem* is defined as follows: We are given a directed graph $G$ and $k$ pairs of nodes $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$. The problem is to decide whether there exist node-disjoint paths $P_1, \ldots, P_k$ so that $P_i$ goes from $s_i$ to $t_i$.

   Show that Directed Disjoint Paths is NP-Complete.

**Solution:** 1. Directed Disjoint Paths is in the class NP.
Given a set of paths $P_1, .., P_k$, for each path $P_i$, traverse the path and mark each node. If a node is encountered that is already marked, the solution is deemed invalid. Validity of the certificate is established if all k paths are traversed without attempting to mark the same node twice. This process takes at most time O(|V|), as each node in G is flagged at most once in the worst case.

2. DDP is NP-Hard (3-SAT $\leq_p$ DDP)
An arbitrary instance of 3-SAT is defined with variables $x_1, .., x_m$ and k clauses. This reduction is based on the 3D Matching reduction from 3-SAT discussed in class.
For each variable $x_i$, create a gadget. This gadget consists of a core with "start" nodes $s_i^1, ..., s_i^k$, "target-adjacent" nodes $u_i^1, ..., u_i^k$, and "target" nodes $t_i^1, ..., t_i^k$, totaling 3k nodes. Refer to the point nodes as $p_i^1, ..., p_i^{2k}$. Establish paths $s_i^a \longrightarrow p_i^{2a-1} \longrightarrow u_i^a \longrightarrow t_i^a$ and $s_i^a \longrightarrow p_i^{2a-2} \longrightarrow u_i^{a-1} \longrightarrow t_i^a$ for each $a \in 1, ..., k$.
For each clause $C_j$, create a start node $s_j$ and a target node $t_j$. Then, for each $x_i$ in the clause, establish a path $s_j \longrightarrow p_i^{2j-1} \longrightarrow t_j$. If $\bar{x_i}$ is in the clause, create a path $s_j \longrightarrow p_i^{2j} \longrightarrow t_j$. Given a 3-SAT instance with m variables and k clauses, this transformation generates an instance of DDP with k(2m + 1) paths and |V| = k(5m + 2) in polynomial time.
Suppose there is a satisfying assignment. If a variable $x_i$ is assigned True, let its gadget core use the "even" paths, which are inherently disjoint. If $x_i$ is in a clause, that clause has a path from start to target through an odd point in the $x_i$ gadget, making it also disjoint. If $x_i$ is assigned False, the gadget core can use the "odd" paths with the same logic. Because the assignment is satisfying, at least one variable in each clause must match, ensuring each clause gadget includes at least one disjoint $s \longrightarrow t$ path from the variable gadget it touches. Thus, there exists a set of disjoint paths covering all s $\rightarrow$ t pairs in the graph.
If there are k node-disjoint paths through the graph, then for each clause gadget, there is a path selecting one of the variables in that clause to hold. Consider each of these variable assignments as part of our candidate satisfying assignment. By construction, each addition satisfies its corresponding clause. It remains to show that we never pick both $x_i$ and $\bar{x_i}$. Because the $x_i$ gadget core requires selecting "odds" or "evens," we can never have clause paths that include both $x_i$ and $\bar{x_i}$ point nodes. Therefore, our candidate satisfying assignment is indeed satisfying, and any unassigned variables can be completed arbitrarily.
We have a polynomial time mapping from 3-SAT to DDP, where we have a positive instance of 3-SAT if and only if we have a positive instance of DDP. Since 3-SAT is NP-complete, DDP is NP-Hard.

4. *Kleinberg, Jon. Algorithm Design (p. 508, q. 9)* The *Path Selection Problem* may look initially similar to the *Directed Disjoint Paths Problem*, but pay attention to the differences between them! Consider the following situation: You are managing a communications network, modeled by a directed graph $G$. There are $c$ users who are interested in making use of this network. User $i$ issues a "request" to reserve a specific path $P_i$ in $G$ on which to transmit data.

   You are interested in accepting as many path requests as possible, but if you accept both $P_i$ and $P_j$, the two paths cannot share any nodes.

   Thus, the Path Selection Problem asks, given a graph $G$ and a set of requested paths $P_1, \ldots, P_c$ (each of which must be a path in $G$), and given a number $k$, is it possible to select at least $k$ of the paths such that no two paths selected share any nodes?

   Show that Path Selection is also NP-Complete.

**Solution:** 1. Path Selection is within the class NP.
Given a set of paths $P_1, .., P_k$, for each path $P_i$, traverse the path and mark each node. If a node is encountered that is already marked, the solution is considered invalid. Validity of the certificate is established if all k paths are traversed without attempting to mark the same node twice. This process takes at most O(|V|) time, as each node in G is marked at most once in the worst case.

2. PS is NP-Hard. (Independent Set $\leq_p$ PS)
An arbitrary instance of Independent Set is represented by a graph G. We will construct a graph G' to run Path Selection on.

- For each edge in G, introduce a node in G'.
- For each node $v_i$ in G, create a path $P_i$ that visits each node in G' corresponding to an edge in G adjacent to $v_i$. In other words, if $e_{j1}, e_{j2}, ..., e_{jh(i)}$ are the edges adjacent to a node $v_i$ in G, then create a path $P_i$ with edges $e_{j1} \longrightarrow e_{j2} \longleftarrow ...e_{jh(i)}$ (order doesn't matter). (See Figure ?? for illustration).

G has an independent set of size k if and only if there are k paths that are node-disjoint in G'.
If there is a size k independent set, then we can select k nodes from G such that no edge is adjacent to more than one node in our set. Let $P_i$ be in our candidate set of disjoint paths if and only if $v_i$ is in this independent set. Due to construction, each node $e_j$ in G' corresponds to an edge in G, so only one of its adjacent nodes in G could have been chosen, and only the paths corresponding to those nodes include $e_j$ in G'. Since we only selected one of those, our candidate set of paths is indeed disjoint.
If there is a size k set of node-disjoint paths, then by the same logic as above, the set of nodes $v_i$ corresponding to the disjoint paths $P_i$ form an independent set in G.
We have a polynomial time mapping from Independent Set to Path Selection, where we have a positive instance of IS if and only if we have a positive instance of PS. Since Independent Set is NP-complete, Path Selection is NP-Hard.