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:Matej Popovski _____ Wisc id: _____

Reductions

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:

Problem Multiple Interval Scheduling is in NP since, given a set of working k jobs intervals, can be check that none of them overlap in polynomial time.
- We chose Independent Set problem as NP-complete, to show that Multiple Interval Scheduling is NP-complete via polynomial-time reduction from Independent Set.
Independent Set <=p Multiple Interval Scheduling
- We define a graph G = (V, E) that will correspond to the multiple interval scheduling of n jobs and find if it has an independent set of size at least k. Let m denote the number of edges in G, and label them e1,...,em. The time is divided into m disjoint slices, s1,...,sm, with slice si, corresponding to edge ei. For each vertex v, we define a job that requires precisely the slices si for which the edge ei has an endpoint equal to v. Two jobs can both be accepted if and only if they have no time slices in common; that is, if and only if they aren't the two endpoints of the some edge. Thus one can check that a set of jobs that can be simultaneously accepted corresponds to an independent set in the graph G.

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:
- The problem is in NP since we can exhibit a set of k nodes and check that distance between all pairs is at least 3.
- We chose the Independent Set problem as NP-complete to show that Strongly Independent Set is NP complete via polynomial-time reduction from Independent Set.
Independent Set <=p Strongly Independent Set
- Given a graph G and number k, we construct a new graph G' in which we replace each edge e=(u,v) by path of length 2 by adding new node we and two new edges (u, we) and (we ,u) we also include edges between all new nodes.
Now suppose graph G has an independent set of size k. Then all k nodes in new graph G' will be on the distance of each other at least 3 so graph G' is Strongly Independent Set of size k.
- Conversely, suppose G' has strongly independent set of size k. Each of the new added nodes is not in that set being within distance two to every other node in the graph. Thus, independent set consist of nodes present in graph G independent set. Any two new nodes have common edge so can't be neighbours in G.

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 $(s_1, t_1), \ldots, (s_k, t_k)$. 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:
- To show that Directed Disjoint Paths is NP-complete, we need to demonstrate two things: first, that it is in NP, and second, that it is NP-hard.
Directed Disjoint Paths is in NP, since in a polynomial-time we can verify a solution to the problem. Given a set of k node-disjoint paths P1, ..., Pk, the length of each path being at most the number of nodes in the graph, we simply check that each path Pi goes from si to ti and that no two paths share any nodes.
- To show that Directed Disjoint Paths is NP-complete it is reduced the NP-complete problem of 3SAT
- To construct instance of the Direct Disjoint Paths problem, we have graph with 2n directed paths, each of length k, one path Pi corresponding to variable xi and one path Pi' corresponding to ¬xi. We add n source sink pairs corresponding to n variables and connect source si to the first node of paths Pi and Pi', and connect the last nodes in paths Pi and Pi' to sink ti. So there are 2 directed paths connected si to ti, the path si,Pi,ti used to set xi to false and the path si,Pi',ti used to set xi to true.
- We add additional Sj, Tj source sink nods corresponding to clause Cj. We claim that there is a path from Sj to Tj disjoint from the pat selected to connect si-ti source sink pairs iff clause Cj is satisfied by corresponding assignment. Assume clause Cj contains the literal tj1, tj2 and tj3. We have a path Pi or Pi' corresponding to each of these variables or negated variables. The path s have n nodes each, and let vi1, vi2, vi3 denote jth node on the 3 corresponding paths. For each l=1,2,3 we add the edges (Sj,vjt) and (vjl, Tj).
- Resulting directed graph has node disjoint paths iff 3SAT instance is satisfiable.
- => If 3SAT is satisfiable, then select the path connecting si to ti corresponding to the satisfying assignment, as suggested. Then source sink pair Sj and Tj can be connected through the path using the true variable the clause.
- <= If disjunctive paths exist than 3SAT has satisfying assignment. Paths Pi and Pj are disjoint and graph has no edges connecting different paths. The only edges outside these paths in the graph are edges entering one of the sinks, or leaving source. As result the only paths in the graph connecting an si-ti pair are two paths si,Pi,ti and si,Pi',ti, and the only paths in G' connecting Sj-Tj pairs are three possible paths through each of 3 variable nodes in Cj. Hence, sets of disjoint paths connecting source-sink pairs, correspond to satisfying assignments.

4. *Kleinberg, Jon. Algorithm Design (p. 508, q. 9)* The *Path Selection Problem* may look initially similar to the problem from the question 3. 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:
- Path Selection is in NP. Given a set of paths P1, P2, ...,Pc, and number k, we can check in the polynomial-time, since the length of each path is at most the number of nodes in the graph, if each of the k paths P1, ..., Pk, is a requested path, and no two paths share any nodes.
To show that Path Selection is NP-complete, we will reduce the problem to NP-complete problem of Vertex Cover.
Vertex-cover<=p Path-Selection
- Given an undirected graph G = (V, E) and an integer k, we construct a directed graph G' as follows: For each vertex v belongs to V, we add two nodes sv and tv and two additional vertexes s and t in G'. For each edge (u, v) blongs to E, we add a directed path from sv to tv and directed path from s to each vertex sv and each vertex tv to t in G'. We claim that there exists a vertex cover of size k in G if and only if there exists a selection of k paths in G' such that no two paths share any nodes.
Suppose there exists a vertex cover V' of size k in G. We can construct a selection of k paths in G' as follows. For each vertex v in V', we select the path from sv to tv. We also select the paths from s to each of the vertices sv, and the paths from each of the vertices tv to t. Since V' is a vertex cover, every edge in E is incident to at least one vertex in V', so every directed path from sv to tv in G' contains at least one vertex in V'. Therefore, no two paths share any nodes in G'.
- Conversely, suppose there exists a selection of k paths in G' such that no two paths share any nodes. We can construct a vertex cover V' of size k in G as follows. For each vertex v, if the path from sv to tv is selected, we add v to V'. Since the paths from s to each of the vertices sv are selected, s is covered by V'. Similarly, since the paths from each of the vertices tv to t are selected, t is covered by V'. It remains to show that every edge in E is incident to at least one vertex in V'. Suppose for a contradiction that there exists an edge (u,v) in E that is not incident to any vertex in V'. Then the directed path from sv to tv is not selected, so it must share a node with some other selected path. But this contradicts the assumption that no two paths share any nodes. Therefore, every edge in E is incident to at least one vertex in V', and V' is a vertex cover of size k in G.