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: Mahir Wisc id: _____

Intractibility

1. *Kleinberg, Jon. Algorithm Design (p. 506, q. 4).* A system has a set of n processes and a set of m resources. Each process specifies a set of resources that it requests to use. Each resource might be requested by many processes; but it can only be used by a single process. If a process is allocated all the resources it requests, then it is active; otherwise it is blocked.

   Thus we phrase the Resource Reservation Problem as follows: Given a set of processes and resources, the set of requested resources for each process, and a number k, is it possible to allocate resources to processes so that at least k processes will be active?

   For the following problems, either give a polynomial-time algorithm or prove the problem is NP-complete.

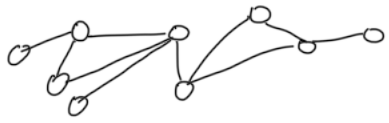
   (a) The general Resource Reservation Problem defined above.

Proving that RRP is NP: Given a set of k processes, we verify that there is no overlap in the m resources required by each process in poly time. Basically, for each resource, count how many processes request that resource. If the counts for all of these counts are 0 or 1, then the solution is valid, otherwise sol. is not valid, since our requirement is not met. Canting the resources and processes needing each resource should take $O(nm)$, and checking for valid counts takes $O(1)$. —> Polynomial validity is shown, thus the problem is NP!

Showing RRP is NP-Hard | will reduce Independent set to RRP, i.e.: $IS \leq_p RRP$

Reduction:

* Let G be an istance of IS, and k be the number of independent sets we want to find
* Let the nodes of G be the processes in RRP
* Let the edges of G be the set of resources
* A process requires a resource in the reduction if the node is adjacent to the edge in G.



↳ This reduction can be done in polynomial time, since we can form the sets of processes/resources directly from the Graph G. Determining the relationship between resources and processes is equal to nodes and their incident edges. Thus, IS is reduced to RRS.

Correctness of Reduction: Independent Set instance is true <=> RRP instance is true

=> If we have an independent set in G, then there are some nodes that have no edges in common, for a k amount of nodes. As per our reduction, this means that there are some processes that have no resources in common with k-1 amount of other processes, as per the reduction we did. Thus, if we have a yes instance for IS, we have a yes instance for RRP.

<= If we have k processes with no overlap between resources, then the k nodes that correspond to each process must have no edges in common. Thus, a yes for RRP shows a yes for IS.

↳ Thus, we have a polynomial time mapping between IS and RRP, and since IS is NP complete, the RRP must be NP-Complete too.

(b) The special case of the problem when $k = 2$.

Since $k=2$, this can be solved polynomially.
* For each pair of processes, check if they have any resources required in common. Which can be done in $O(m)$ each time, this is (mn^2) total
* If there is a pair of processes with the same resources needed, return a "no" instance.
* else, return a yes instance, i.e: no shared resources.

(c) The special case of the problem when there are two types of resources–say, people and equipment– and each process requests at most one resource of each type (In other words, each process requests at most one person and at most one piece of equipment.)

* Assume a set of processes which only require one resource, where each process requires a different resources
Discard any duplicate processes that need the same resource, so no duplicate resource usage.
* For each resource type $R1$ and $R2$, create a set of nodes. If there is a resource r requested by a process, remove that node.
* For each process P_i that requires $r_1 \in R_1$ or $r_2 \in R_2$, create edge from r_1 to r_2.
* Then, solve as bipartite matching problem, done in polynomial time, where max flow = k.

(d) The special case of the problem when each resource is requested by at most two processes.

* This case can be solved by the reduction above, just set $k = 2$

2. *Kleinberg, Jon. Algorithm Design (p. 506, q. 7).* The 3-Dimensional Matching Problem is an NP-complete problem defined as follows:

   Given disjoint sets X, Y, and Z, each of size n, and given a set $T \subseteq X \times Y \times Z$ of ordered triples, does there exist a set of n triples in T such that each element of $X \cup Y \cup Z$ is contained in exactly one of these triples?

   Since 3-Dimensional Matching is NP-complete, it is natural to expect that the 4-Dimensional Problem is at least as hard.

   Let us define 4-Dimensional Matching as follows. Given sets W, X, Y, and Z, each of size n, and a collection C of ordered 4-tuples of the form $(w, x, y, z) \in W \times X \times Y \times Z$, do there exist n 4-tuples from C such that each element of $W \cup X \cup Y \cup Z$ appears in exactly one of these 4-tuples?

   Prove that 4-Dimensional Matching is NP-complete. Hint: use a reduction from 3-Dimensional Matching.

4D matching is NP

Given that we have n 4-tuples in C, it can easily be verified in poly time that $W \cup X \cup Y \cup Z$ belongs to exactly one of the n-tuples, thus 4D matching is NP.

4D matching is NP Hard

* Assume arbitrary matching of the 3D matching, defined as sets X, Y, Z, each of size n, with $T \subseteq X \cup Y \cup Z$ triples

↳ Transform this into 4D by duplicating one of the sets from 3D instance. This should take polynomial time to do. Thus, we have reduced a 3D matching problem into a 4D matching problem.

Proof

=> If there is a set of n triples that satisfy the 3D matching, ie: X, Y, Z, then the reduced 4 tuple will also match the 4D matching instance since one set is duplicated, i.e: X, X, Y, Z, thus yes instance of 3D => yes instance of 4D

<= If an instance of a set of n 4-tuples satisfying the instance of 4D matching, the corresponding original 3-tple will also satisfy the 3D matching instance.

3. *Kleinberg, Jon. Algorithm Design (p. 507, q. 6).* Consider an instance of the Satisfiability Problem, specified by clauses $C_1, ..., C_m$ over a set of Boolean variables $x_1, ..., x_n$. We say that the instance is monotone if each term in each clause consists of a nonnegated variable; that is, each term is equal to x_i, for some i, rather than $\overline{x_i}$. Monotone instances of Satisfiability are very easy to solve: They are always satisfiable, by setting each variable equal to 1.

   For example, suppose we have the three clauses

   $$(x_1 \vee x_2), (x_1 \vee x_3), (x_2 \vee x_3).$$

   This is monotone, and the assignment that sets all three variables to 1 satisfies all the clauses. But we can observe that this is not the only satisfying assignment; we could also have set x_1 and x_2 to 1, and x_3 to 0. Indeed, for any monotone instance, it is natural to ask how few variables we need to set to 1 in order to satisfy it.

   Given a monotone instance of Satisfiability, together with a number k, the problem of *Monotone Satisfiability with Few True Variables* asks: Is there a satisfying assignment for the instance in which at most k variables are set to 1? Prove this problem is NP-complete.

1) Problem is NP

Given k or fewer variables are set to 1, along with a boolean formula, we can verify in polynomial time if the formula is monotone, and also if the formula is satisfied on the k variables set to 1, and all other variables set to 0. Thus, the problem is NP

2) NP-Hard

I will reduce vertex cover to problem, i.e: $VC \leq_P X$.

An arbitrary instance of vertex cover is defined on a graph $G = (V, E)$, and a number k. For every node, create a literal x_i. For every edge (u, v), create a clause (x_i, x_j). The boolean formula is the conjunction of all clauses. The transformation is polynomial, as we just need to go through each node and edge in G. Thus, we have reduced VC into the problem.

=> Suppose there is a VC for G of size at most k. For each node in the cover, setting the corresponding literal x_i to 1 will result in every clause evaluating to true, which means that the boolean formula is satisfiable

<= Same idea, other way. Map all parts of the SAT problem back into VC based on the above reduction.

4. *Kleinberg, Jon. Algorithm Design (p. 509, q. 10).* Your friends at WebExodus have recently been doing some consulting work for companies that maintain large, publicly accessible Web sites and they've come across the following Strategic Advertising Problem.

   A company comes to them with the map of a Web site, which we'll model as a directed graph $G = (V, E)$. The company also provides a set of t trails typically followed by users of the site; we'll model these trails as directed paths $P_1, P_2, ..., P_t$ in the graph G (i.e., each P_i is a path in G).

   The company wants WebExodus to answer the following question for them: Given G, the paths $\{P_i\}$, and a number k, is it possible to place advertisements on at most k of the nodes in G, so that each path P_i includes at least one node containing an advertisement? We'll call this the Strategic Advertising Problem, with input $G, \{P_i : i = 1, ..., t\}$, and k. Your friends figure that a good algorithm for this will make them all rich; unfortunately, things are never quite this simple.

   (a) Prove that Strategic Advertising is NP-Complete.

NP

Consider that we were given k nodes to place ads on, and we can check the validity by checking all t paths for the presence of the k nodes, rejecting if any path misses the selected nodes, and rejects accordingly.

NP - Hard

I have no idea how to reduce and prove this part, I tried my best

(b) Your friends at WebExodus forge ahead and write a pretty fast algorithm $\mathcal{S}$ that produces yes/no answers to arbitrary instances of the Strategic Advertising Problem. You may assume that the algorithm $\mathcal{S}$ is always correct.

Using the algorithm $\mathcal{S}$ as a black box, design an algorithm that takes input $G, \{P_i : i = 1, ..., t\}$, and k as in part (a), and does one of the following two things:

- Outputs a set of at most k nodes in G so that each path P_i includes at least one of these nodes.
- Outputs (correctly) that no such set of at most k nodes exists.

Your algorithm should use at most polynomial number of steps, together with at most polynomial number of calls to the algorithm $\mathcal{S}$.

Very confusing, see above. Need more practice to get this.