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: _____ 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. At any given point in time, each process specifies a set of resources that it requests to use. Each resource might be requested by many processes at once; but it can only be used by a single process at a time. 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.

Solution:

1. *The general Resource Reservation Problem (RRP) is in NP.*
   Given a set of k processes, we can verify that there is no overlap in the resources required by the processes in polynomial time. For each resource, count how many selected processes have requested that resource. This can be done in time $O(km)$. If all counts are ≤ 1, the solution is valid, if not, the solution is invalid. Since a proposed solution can be confirmed or rejected in polynomial time, this problem is in NP.
2. *The general RRP is NP-Hard.* (*Independent Set* $\leq_p$ *RRP*)
   An arbitrary instance of Independent Set is defined on a graph G and an integer k.
   - Let the nodes of G be the set of processes
   - Let the edges of G be the set of resources
   - A process requires a resource if and only if the node is adjacent to the edge in G.

   This transformation can be done in polynomial time, since we can form the sets of processes and resources directly from the sets of vertices and edges in G, and determining which processes require which resources is a matter of finding edges adjacent to nodes, which is very easy with an adjacency list representation (but still polynomial with any "normal" graph representation). In this way, we transform an arbitrary instance of IS into an instance of RRP.

   ($\Rightarrow$) If we have a k nodes in G which correspond to an independent set, the selected nodes have no edges in common, which means that no processes have any required resources in common by the way we transformed the instance above. So a yes instance for independent set is a yes instance for the general resource reservation problem.

   ($\Leftarrow$) If we have k processes that have no overlap in required resources, then the k nodes that correspond to those processes must have no edges in common. So a yes instance for the general resource reservation is a yes instance for independent set.

   Thus, we have a polynomial time mapping from independent set to the general resource reservation problem, in which we have a yes instance for independent set if and only if we have a yes instance for the general resource reservation problem. Since Independent Set is NP-Complete, the general resource reservation problem is NP-Hard.

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

Solution:

Consider the following polynomial-time algorithm which solves the special case of $k = 2$:

- For each pair of processes, check all m resources to see if they have any requirements in common. ($O(mn^2)$)
- If there is a pair of processes which have no requirements in common, the answer is yes.
- If no such pair exists, the answer is no.

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

Solution:

- Let S denote the set of processes which only require one resource, where every request is disjoint in resource requirements. Discard all other processes which request a resource required by a process in S. (polynomial in number of processes)
- Create two sets of nodes, T_1 and T_2, for each type of resource. For each resource r requested by processes in S, remove the node for r. (polynomial in number of resoures)
- For each process that requires $t_1 \in T_1$ and $t_2 \in T_2$, where neither t_1 nor t_2 have been removed, create an edge from t_1 to t_2. (polynomial in number of processes)
- Now this can be solved as a bipartite matching problem (which can be done in polynomial time), where the desired number of matches is $k - |S|$.

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

Solution:

The reduction outlined in (a) corresponds to this special case, since the resource/edges of independent set are adjacent to exactly two process/nodes. Thus, this special case is still NP-Complete, by that reduction.

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 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_i, x_j, y_k, z_ℓ), do there exist n 4-tuples from C such that each element of $W \cup Y \cup X \cup Z$ is contained in exactly one of these 4-tuples?

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

Solution:

1. *4-Dimensional Matching is in NP.*
   Given n 4-tuples in C, it can easily be verified in polynomial time that each element in $W \cup X \cup Y \cup Z$ belongs to exactly one of the n tuples. So 4-Dimensional Matching is in NP.
2. *4-Dimensional Matching is NP-Hard.* (*3-D Matching* $\leq_p$ *4-D Matching*)
   An arbitrary instance of 3-Dimensional matching is defined on sets X, Y, and Z, each of size n, along with a set $T \subseteq X \times Y \times Z$ of ordered triples. We can transform this into an instance of 4D matching by padding the tuples. One possibility for this is to copy one of the sets and associated elements. We can let $T' \subseteq X \times X \times Y \times Z$ be the same ordered triples, but where we had $\langle x, y, z\rangle$ before, use $\langle x, x, y, z\rangle$. This is a polynomial-time transformation. In this way, we transform an arbitrary instance of 3-D Matching into an instance of 4-D Matching.

   ($\Rightarrow$) Observe that $X \cup X \cup Y \cup Z = X \cup Y \cup Z$, and that for any $a \in X \cup Y \cup Z$, we have $a \in \langle x, y, z\rangle \iff a \in \{x, y, z\} \iff a \in \langle x, x, y, z\rangle$. If there is a set of n triples that satisfy the instance of 3-Dimensional Matching, the corresponding transformed 4-tuples will also satisfy the transformed 4-Dimensional Matching instance, since the additional information is just a copy of what was supplied for the 3-Dimensional Matching instance.

   ($\Rightarrow$) If there is a set of n 4-tuples that satisfy the instance of 4-Dimensional Matching, the corresponding original triples will also satisfy the 3-Dimensional Matching instance, similarly to as described above.

   Thus, 4-Dimensional Matching is NP-Hard.

3. *Kleinberg, Jon. Algorithm Design (p. 507, q. 6).* Consider an instance of the Satisfiability Problem, specified by clauses $C_1, ..., C_k$ 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.

Solution:

1. *Monotone Satisfiability with Few True Variables (MSFTV) in in NP.*
   Given k (or fewer) variables to set to 1, along with a boolean formula Φ, we can easily verify in polynomial time (1) if Φ is monotone and (2) if Φ is satisfied on the k identified variables set to 1 and all other variables set to 0. So Monotone Satisfiability with Few True Variables in in NP.
2. *MSFTV is NP-Hard.* (*Vertex Cover* $\leq_p$ *MSFTV*)
   An arbitrary instance of vertex cover is defined on a graph $G = (V, E)$ and a number k. For every $v \in V$, create a variable, x_v. For every $\{u, v\} \in E$, create a clause $(x_u \vee x_v)$. The formula Φ is the conjunction of all such clauses. This transformation is clearly polynomial, as we just need to process each vertex and edge in G. Additionally, this will always produce a monotone formula. In this way, we transform an arbitrary instance of vertex cover into an instance of MSFTV.

   ($\Rightarrow$) Suppose there is a vertex cover for G of size at most k. For each vertex in the vertex cover, setting the corresponding variable to 1 in Φ will result in every clause evaluating to true, which means Φ is satisfiable.

   ($\Leftarrow$) Suppose there is a satisfying assignment for Φ in which at most k variables are set to 1. Since Φ is monotone, Φ is satisfiable if and only if at least one variable in every clause is set to 1. If we take the vertices that correspond to the variables that are set to 1, we have at most k vertices and they form a vertex cover.

   Thus, Monotone Satisfiability with Few True Variables is NP-Hard.

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.

Solution:

1. *Strategic Advertising is in NP.*
   Consider an instance of the Strategic Advertising Problem on G, $\{P_i : i = 1, ..., t\}$, and k. If we are given k (or fewer) nodes on which to place advertisements, we can check the validity of this solution by checking all t paths for the presence of any of the k nodes ($O(kt|V|)$), rejecting if any path misses all selected nodes and accepting otherwise. So the Strategic Advertising Problem is in NP.
2. *Strategic Advertising is NP-Hard.* (*Vertex Cover* $\leq_p$ *Strategic Advertising*)
   An arbitrary instance of vertex cover is defined on a graph $G = (V, E)$ and a number k. Since G is undirected and Strategic Advertising is defined on a directed graph, arbitrarily direct all edges ($O(|E|)$) to form G'. For each new directed edge e_i, define a path P_i (which consists of the single edge), also $O(|E|)$. In this way, we can take an arbitrary instance of Vertex Cover and turn it into an instance of Strategic Advertising.

   ($\Rightarrow$) If G has a vertex cover of at most k vertices, then every path in G' includes as least of of those vertices, and thus there is a strategic advertisement.

   ($\Leftarrow$) If there is a strategic advertisement for G' of at most k vertices, then every path contains at least one of those vertices (by definition of the problem). If every path in G' contains at least one of the selected vertices, then every edge in G is indicent on one of the (at most) k vertices, and thus there is a vertex cover of size at most k.

   Thus, Strategic Advertising is NP-Hard.

(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 $\mathcal{S}$.

Solution:

1. Query $\mathcal{S}$ on the original instance of the problem.
   - If $\mathcal{S}$ returns no, output that no such set of at most k nodes exists.
   - If $\mathcal{S}$ returns yes, continue.
2. Create a graph $G' = (V', E')$ that is a copy of G, as well as a copy P' of the paths and a tracker k' which starts equal to k. Finally, we need a set A of vertices for advertising, initially the empty set.
3. Arbitrarily select a vertex $v \in V'$.
   - For each occurrence of v in E':
     - If $\{u, v\}$ is a directed edge and no edge $\{v, w\}$ exists, remove $\{u, v\}$ from E'.
     - If $\{v, w\}$ is a directed edge and no edge $\{u, v\}$ exists, remove $\{v, w\}$ from E'.
     - If $\{u, v\}$ and $\{v, w\}$ are both directed edges, form a new edge $\{u, w\}$ and remove $\{u, v\}$ and $\{v, w\}$ from E'.
   - The above steps also need to be taken on all paths P'_i that contain v, altogether this can be done in $O(t|E'|^2)$.
   - Remove v from V'.
4. Query $\mathcal{S}$ on G', P', and k'.
   - If $\mathcal{S}$ returns no:
     - Add v to A.
     - Remove all paths from P' that contained v.
     - Decrement k'.
     - If k' is 0 or $|A| = k$, output the contents of A.
     - If $k' > 0$, return to step 3.
   - If $\mathcal{S}$ returns yes:
     - Return to step 3.