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: Zinnia Nie Wisc id: 908 319 4044

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 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: n processes, m resources total

1. Prove Resources Reservation is NP
   - given a set of processes with already allocated resources
   - if the size of the set is $\geq k$, then the certificate is a yes, else a no
2. Independent Set $\leq_p$ Resource Reservation
   - Processes are nodes, Resources are edges
   - Processes that need a resource are nodes connected by the edge.
   - We are looking for an independent set of size k so if it is possible to allocate resources to k processes
   - This reduction can be created in polynomial time $O(nm)$
   - Independent set is yes $\Leftrightarrow$ Resource Reservation is yes
     - if independent set has k sets then there are at least k processes that do not require the same resources because those k processes/sets are not connected by an edge / resource.
     - if resource reservation problem shows k processes can be active, then that means those processes don't share resources. As the resources are edges in Independent Set, this means only one process is chosen from a connected edge. This fulfills the definition of Independent set so there are k independent sets in the graph

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

Solution:

We can brute force test every possible pair combination of jobs and resources to see if there is a way to allocate resources

This is polynomial time to go thrugh all pairs in $O(n^2)$.

(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:

We can use bipartite matching to match each types of resource (ex. matching people to equipment). If there are k pairs, then it is possible to allocate resources to jobs.

We know bipartite matching can be solved in polynomial time

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

Solution:

If this is reduced to Independent Set, it is just an instance where the resource has processes as nodes on either side of the resource edge.

Therefore, this is just a special case of Resource Reservation and is also NP-complete.

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 $W \cup Y \cup X \cup Z$ appears 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. Prove 4-D Matching is NP
   - certificate is a set of 4-tuples
   - First check size of set, if $< n$, is a no instance
   - Then loop through all the 4-tuples and check that each w_i, x_j, y_k, z_ℓ is not a duplicate and is part of sets W, X, Y, Z respectively
   - This can be done in polynomial time
2. 3D Matching $\leq_p$ 4D Matching
   - have a group of n instances of 3D matching using (x_j, y_k, z_ℓ) groups with w_i being the i of the instance of 3D matching
   - If 3D Matching is a yes → 4D Matching is yes
     - 3D Matching being yes means (x_j, y_k, z_ℓ) are disjoint. Then w_i being equal to $i = 1..N$ means w_i is also unique for each 4-tuple and (w_i, x_j, y_k, z_ℓ) are disjoint
   - If 4D Match is yes → 3D match must be yes
     - 4D Match being yes means (w_i, x_j, y_k, z_ℓ) are disjoint, so the smaller group (x_j, y_k, z_ℓ) are also disjoint

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.

Solution:

1. Prove Monotone Satisfiability $\in$ NP
   - certificate is a set of assignments
   - first check the number of 1s, if $< k$, is a no instance
   - Then check that the assignment satisfies each clause
   - This can be done in polynomial time

2. Vertex Cover $\leq_p$ Monotone satisfiability
   - each node is a variable x_i.
   - each edge (i,j) becomes a clause $x_i \vee x_j$
   - edges become clauses so we want to know if a set of k variables being set to 1 will satisfy all clauses
   - If vertex cover is yes $\rightarrow$ MS is yes
     - vertex cover being yes means all edges are incident to the nodes in the set
     - Therefore, setting all the nodes in the set to 1 would mean each clause has at least one 1 and would be satisfied
   - If MS is yes $\rightarrow$ VC must have been yes
     - can satisfy all clauses with k variables set to 1.
     - That means, since each clause is an edge, each edge is incident to one of the nodes corresponding to the k variables
     - This creates a vertex cover

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. Prove Strategic Advertising $\in$ NP
   - certificate is set of Nodes to place advertisements
   - first check size of set, if $> k$, is a no instance
   - loop through all paths P_i and check that at least one node in the set is also in the path
   - this can be done in polynomial time

2. Vertex Cover $\leq_p$ Strategic Advertising
   - vertex cover is undirected, so arbitrarily direct all edges
   - let a path be an edge in the vertex cover graph
   - this can be done in polynomial time
   - If VC is yes $\rightarrow$ SA is yes
     - There is a set of k nodes that are incident to every edge in the Graph. since every edge is covered, any path would then go through at least one node in the set. So those nodes are the places to put ads.
   - If SA is yes $\rightarrow$ VC must have been yes
     - All the paths are just one edge, so if there is a set that advertises through all paths, each edge must be incident to a node in the set

(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:

```
If k=1 : check for a node in all paths
else:
  run S on G for k
  If return no : return "no such set"
  else:
    for each node v:
      delete v from G and any paths with v
      run S on G for k-1
      if yes : add to a set of nodes
    return set of nodes
```

- will call S n+1 times
- if v is part of the set of nodes, deleting it and any paths going through it should create a smaller instance of SA with k-1 nodes that are part of the set, as one of said nodes was deleted. Therefore, repeatingly searching for the node where this happens should create a set of nodes to place ads.