This homework is due by the start of class at 1PM on Thursday, Apr 22th. You must submit the homework via the course page on T-Square.

Homework Policies:

- Your work will be graded on correctness and clarity. Write complete and precise answers.
- You may collaborate with up to *three* other classmates on this problem set. However, you must *write your own solutions* in your own words and *list your collaborators*.
- You may portions of our two textbooks that we have covered. The purpose of these homeworks is NOT only to teach you the algorithms that we cover, but also to teach you how to problem-solve. Therefore using *ANY* outside resources like the internet, etc. is *not* allowed!
- **Optional** questions are for you to practice and learn the basics before you advance. They may ask you to follow the steps of an algorithm covered in class, or review material from an earlier class. Solutions to these problems will not be graded, but you must be are able to do them, and you are responsible for material that is addressed by these optional questions.
- **Basic** questions *must be solved and written up alone*; you may not collaborate with others. These will help you build the framework needed to solve the harder questions. Please do these first, before meeting in groups, for both you and your group's benefit.
- **Group** questions may be solved in collaboration with up to ***three*** other classmates. However, you must still *write up your own solutions* alone and in your own words. These will form the bulk of your homework questions.
- **Bonus** questions are like group questions, but they will be more challenging and will be graded more rigorously. Bonus points are tallied separately from your normal points, and will be applied to your grade *after* the curve. Bonus points are worth more than normal points, and can have a significant positive affect on your grade. Most importantly, they are are fun!

Directions for Homework 8: You may assume that the following problems (both the search and decision versions) are NP-complete when doing your reductions, as well as any that you prove NP-complete in this or your previous homework.

- SAT : Given a boolean formula F in CNF form with n variables and m clauses, determine if there is an assignment of the n variables that satisfies F.
- $3-$SAT : A special case of SAT where every clause has at most 3 literals.
- IntegerLinearProgramming : Given a set of n variables and m linear constraints, and a linear objective function, find an *integer* solution to the constraints that optimizes the objective.
- Independent Set : Given graph G and $k \in \mathbb{Z}^+$, determine if G has an indep. set of size $\geq k$.
- Vertex Cover : Given graph G and $k \in \mathbb{Z}^+$, determine if G has a vertex cover of size $\leq k$.
- Clique : Given graph G and $k \in \mathbb{Z}^+$, determine if G has a clique of size $\geq k$.
- Subset Sum : Given a list of integers A and a target T, determine if some sublist of A adds up to T.
- Hamiltonian Cycle (Rudrata): Given graph G, determine if G has a cycle that visits every vertex exactly once.

Remember that to show a problem A is NP-Complete, you need to do three things.

- Show that problem A is in NP. (Describe a verification algorithm for A).
- Reduce A **from** a previously established NP-hard problem B.
  - First, describe an algorithm to solve the NP-hard problem B that uses a solution for A as a subroutine, and does polynomial amounts of work, and calls A polynomially many times on polynomial-sized inputs.
  - Prove that your reduction is correct. You need to prove that your proposed algorithm for B will "accept" an input IF AND ONLY IF the input was a valid "yes" instance. This usually involves some analysis on how you transformed your input to problem B into the input to problem A in your reduction.

If problem A is a generalization of a known NP-complete problem, then the reduction step is much easier - just explain how to set the parameters of the more general problem to deal with the special case.

You do not have to write full pseudocode when describing your reductions, but you may if you feel that it will help you communicate your ideas. We expect your writing to be as clear as if you were writing pseudocode. A programmer should be able to read your description and implement your algorithm unambiguously.

1. (20 points) (basic) We define the problem $\mathsf{Clique3}(G, k)$. You are tasked with determining if G has a clique of size k, on cases where the max degree of any vertex of G is 3. In otherwords, this is the $\mathsf{Clique}(G, k)$ problem restricted to graphs in which every vertex has degree at most 3.

   (a) Prove that Clique3 is in NP.

   > **Solution:** Clique3 is a special case of Clique, and we can use the same certificate to verify Clique3 instances as Clique instances. (Let the certificate be a $k - clique$. Verify that there are k vertices in $O(k)$ time, and that they form a clique in $O(k^2)$ time by checking each pair of vertices for an edge.

   (b) I claim that Clique3 is NP-complete.

   > PROOF: We showed that Clique3 is in NP in part a.
   >
   > We show a reduction from Clique3 to Clique , a known NP-hard problem. Clique is a generalizaion of Clique3, since Clique3 consists of the special case of Clique where the input graph has max degree 3. Therefore we can solve $\mathsf{Clique3}(G, k)$ by returning $\mathsf{Clique}(G, k)$ on exactly the input to Clique3 unchanged.
   >
   > This proves the correctness of the reduction and, therefore the NP-completeness of Clique-3. ■

   What is wrong with the above proof?

   > **Solution:** The reduction from Clique-3 to Clique is a valid reduction (you are indeed using Clique to solve Clique3), but this doesn't imply that Clique-3 is NP-Hard. To show that Clique3 is NP hard, you would need to reduce Clique to Clique-3 (the other direction).

(c) Here's a true fact that you can use without proof.

Theorem 1.1: Independent Set3(G, k), the special case of Independent Set(G, k) restricted to graphs of degree at most 3, is NP-complete.

Here's another fact that we learned in lecture when showing that Clique was NP-complete.

Theorem 1.2: A subset $X \subseteq V$ is an independent set in G if and only if X is a clique in the "opposite" graph $G' = (V, E' = \binom{v}{2} \setminus E)$. Therefore G has a independent set of size $\geq k$ if and only G' has a clique of size $\geq k$.

Now lets try again.
Claim: Clique3 is NP-complete.

PROOF: Clique3 is in NP as proved in part a.
We present a reduction from Independent Set3 to Clique3.
To solve Independent Set3(G, k), we first create G' by switching all the edges of G, and returning Clique3(G', k).
Proof of correctness: By the second fact, G has an independent set of size k if and only if G' has a clique of size k. This proves the correctness of the reduction.
By the first fact, Independent Set3 is NP-complete, and therefore Clique3 is NP-complete.■

What's wrong with the proof this time?

Solution: When we switch all the edges of the graph, then the graph won't have max degree 3 anymore; in fact every vertex will have degree at least $|V| - 4$, which could be much larger than 3. Therefore, our algorithm for Clique3 won't be able to be used.

(d) Describe any poly-time algorithm for Clique3. Hint: How big can a clique get?

Solution: The max size of a clique, if every vertex has degree 3, is 4. So what we do is: if $k > 4$, return false. Otherwise, check all n^k (up to n^4) possible subsets of $k \leq 4$ vertices to see if there is a Clique. Return $TRUE$ iff you find a clique of size k.
This is polytime, since we always terminate in at most $O(n^4)$ time.

2. (20 points) (group) Hamiltonian Path : Show that the following problem is NP-complete by reducing from Hamiltonian Cycle .

Given graph G, determine if G has a simple **path** (no repeated vertices) that visits every vertex exactly once. Here the start and end vertices of the path do not need to be the same (or adjacent).

Solution:

- To show this problem is in NP, let the certificate for a graph with a Hamiltonian path be the vertices of the path listed in order. Our verifier would check that every vertex was contained in this path, and that every proposed edge was in the graph. The verifier accepts iff the certificate is a Hamiltonian Path, which exists iff a Hamiltonian path exists.

- There are multiple constructions that you could have used to do the reduction.

  Most importantly, it is important to remember that you are trying to solve Hamiltonian Cycle using an algorithm that can solve Hamiltonian Path , and not the otherway around. There are valid ways to use Hamiltonian Cycle to solve Hamiltonian Path, but this won't show that Hamiltonian Path is NP-complete.

  Here's our reduction. Given graph G, we're going to construct G' as follows. Pick any vertex v, and make an extra copy of it called v', and add all the same edge connections as v. Now construct another new vertex s and connect it only to v, and another new vertex t, and connect it only to v'. Now, we return Hamiltonian Path(G').

  Here's the proof of correctness.

  - G has a Hamiltonian Cycle $\rightarrow$ G' has a Hamiltonian Path

    PROOF: Say G has a Hamiltonian Cycle . Then since a cycle visits every vertex, including v, write the vertices of the cycle as $(v, x_1), (x_1, x_2), \ldots, (x_{n-1}, v)$. By construction, the edges $(v, x_1), (x_1, x_2), \ldots, (x_{n-1}, v')$ are in the graph G', as are (s, v) and (v', t). Since the only edges we added were v', s, t, this new set of edges are a Hamiltonian Path in G'. ■

  - G' has a Hamiltonian Path $\rightarrow$ G has a Hamiltonian Cycle

    PROOF: Say G' has a Hamiltonian Path . Because s and t both have degree 1, any Hamiltonian Path in G' must have s and t as its endpoints. Since s only connects to v and t to v', then the path must go from s to v to every other vertex to v' to t. The middle part of this path is a path from v to every other vertex back to v' in G'. The equivalent edges in G would form a path from v to every other vertex and back to v again, which are a Hamiltonian Cycle . ■

Another valid approach is to add t to a different neighbor of v, one at a time, until you find a Hamiltonian Path . This would require at most $O(n)$ calls to Hamiltonian Path , once for each neighbor of v, which is a valid polytime reduction.

3. (20 points) (group) Partition: Show that the following problem is NP-complete.

   Given a list of integers A, determine if the integers of A can be partitioned into two disjoint lists A_1, A_2 that have the same sum.

   For example, the list $A = [7, 3, 2, 9, 6, 13]$ can be partitioned into $[7, 13]$, $[2, 3, 6, 9]$ each with sum 20. The list $A = [4, 6, 8, 12]$ cannot be partitioned.

Solution: Here's an observation about Partition that will make this a lot easier - Partition is equivalent to the special case of Subset Sum where Partition(A) = Subset Sum$(A, \sum A/2)$]. This is because if you can find one subset of total sum $\sum(A)/2$, the other elements are automatically another subset of total $\sum(A) - \sum(A)/2 = \sum(A)/2$, and we have a partition!

- First we show that the problem is in NP. Since Partition is a special case of Subset Sum , we can just use the same certificate and verifier as Subset Sum (On a certificate

which is a set of indices I where $\sum i \in IA[i] = \sum(A)/2$, check to make sure the sum is correct.)

- Here's the reduction from Subset Sum to Partition. On instance A, T of Subset Sum , create the new array A' which is A with an extra element of value $2T - \sum(A)$. Return Partition(A').

  The idea behind this reduction is that we added a single element so that the total sum of A' is now $2T$. Now the partition problem on A' is equivalent to finding a subset of sum T in A'.

  Here's the proof of correctness.

  – A has a subset of sum $T \rightarrow A'$ has a partition.

  PROOF: Let X be a subset of A that sums to T. X is also present in A', thus X is a subset of elements in A' that sums to T. By construction, the sum of all elements in A' is $2T$, therefore the other elements in $A' \setminus X$ also sum to T. This is a partition of A'. ■

  – A' has a partition $\rightarrow A$ has a subset of sum T.

  PROOF: Say A' has a partition. This means it has two subsets of elements that sum to $\sum(A')/2 = T$. Only one of these subsets consists of the extra element $(2T - \sum(A))$ that we added, the other subset is entirely composed of elements of the original array A. Therefore this is a subset of A that sums to T. ■

4. (40 points) (group) Generalizations: Show that each of the following problems are NP-hard. (Don't worry about showing that these problems are in NP). Each reduction should be "easy." That is, each problem below is either a direct generalization of a known NP-hard problem (like from Independent Set to Weighted Independent Set), or solves a known NP-hard problem with a relatively simple transformation (like from Independent Set to Clique).

   (a) Given a set of n elements E, and a family of m subsets $S_i \subseteq E$, and "budget" b, determine if there are some b of these subsets $\{S_i\}$ such that their union is E.

   Solution: With a little work, we can see that this problem, known as Set Cover is a generalization of Vertex Cover .

   Given an input (G,k) to Vertex Cover , we construct a set cover problem as follows. Vertex cover is about covering every edges of a graph, so we will create an element for every edge in our graph. So we can treat the set of elements as the edges E (the lettering was a hint!). Each vertex covers those edges incident to it, so for every vertex we will create a set that consists of the edges incident to that vertex. Now the problem of vertex cover (choose up to k of these vertices so that they cover all the edges) is now equivalent to the problem (choose up to k of these vertex-neighbor sets whose union is all the edges). So we run Set Cover on this instance with budget $b = k$.

   (b) Longest s − t Path Given a graph G, two vertices s and t in V, and an integer k determine if G has a simple path (no repeat vertices) that uses k or more edges from s to t.

Solution: We reduce from Hamiltonian Path , which was proven to be NP-hard in a previous problem.

Our reduction: Create vertices s and t and connect them both to every vertex in G. Return Longest $s-t$ Path$(G, s, t, |V|+1)$.

Proof: An $s-t$ path, if you cut off the edges from s and t is a path entirely among the edges of G. Thus an $s-t$ path that visits $|V|+1$ edges in this graph exists iff a path in G with $|V|-1$ edges exists, which is another word for a Hamiltonian path.

By the same logic, your reduction could also have done:

Given an instance (G) of Hamiltonian Path, run Longest $s-t$ Path$(G, s, t, |V|)$ for every pair s, t. Accept iff this returns true for some s, t.

(c) Traveling Salesman Problem Given a weighted graph G and bound B, determine if there is a cycle in G that visits every vertex exactly once and has the total sum of edge weights $\leq B$.

Solution: This is a generalization of Hamiltonian Cycle . Given an instance G of Hamiltonian Cycle , construct a weighted graph where every edge of G was given weight 1. Now return Traveling Salesman Problem$(G, |V|)$.

By construction, a set of edges in G has weight $|V|$ iff there are $|V|$ edges, which means that it is a Hamiltonian Cycle .

(d) Degree Restricted MST: Given an undirected graph $G = (V, E)$ and an integer k, find a spanning tree T of G such that each vertex of the tree has maximum degree k, if such a tree exists.

Solution: What type of tree has max degree 2? A path!

Thus a spanning tree of max degree 2 is Degree Restricted MST$(G, 2)$ is exactly the same as Hamiltonian Path . No more work needed!

5. (10 points) (bonus) Consider the following Job Assignment problem. You are a hiring manager at a consulting firm. You have a set of n available workers W and a set of m tasks T that your clients want you to do. Each worker w knows how to complete some subset $T[w] \subseteq T$ of the tasks, and requires a certain salary to be hired, $S[w]$. Each worker, once hired, can complete all the tasks that they know how to do. Each completed task earns the company a revenue $R[t]$ per task t.

   You goal is to maximize your profit, the sum of all revenues earned minus the sum of all salaries paid. Not all tasks have to be assigned; sometimes the revenue for completing some tasks may not be worth hiring a worker to do it.

   The decision version of this question asks: Given all this information, $n, m, T[w], S[w], R[t]$, can you achieve a profit of at least k? Show that this problem is NP-complete.

Solution:

- We first show that this is in NP. A certificate could be just a list of the set of workers to hire. From there, we can compute the total salary that these workers are getting

paid, and iterate through all the tasks to see which ones can be done by some worker in our list, and then add that to the total revenue. Finally, we check that the profit is at least k. Every step of this can be done in polynomial time.

- We'll reduce from Vertex Cover. We're givin an input (G, b) to Vertex cover. (Here we're using b as the input parameter to avoid confusion with the parameter k for the Job Assignment problem.)

  We'll create a task for every edge, and a worker for every vertex. Each vertex-worker can complete all the edge-tasks that are incident to it.

  Our goal is to make each edge-task so lucrative and the target k high enough that any solution absolutely needs to complete every task (i.e. cover every edge and be a Vertex Cover). Let's say that the revenue from each edge was large, say $2|V|$ per edge. The cost to hire each worker will be just 1.

  What our reduction will do is create this Job Assignment problem as described above, and then run our Job Assignment algorithm with parameter $k = 2|V||E| - b$. We return whatever this returns.

  Here's the proof of correctness.

  – Vertex Cover of size $b \rightarrow$ Profit of $2|V||E| - b$.

    PROOF: If there was a Vertex cover of size b, then we could get a profit of $2|V||E| - b$ for the total revenue from the $|E|$ edges minus the total salaries of the b workers. ∎

  – Profit of $2|V||E| - b \rightarrow$ Vertex Cover of size b.

    PROOF: If we could acheive profit $2|V||E| - b$, then choosing even one fewer edge than all $|E|$ of them would result in a maximum possible profit of $2|V|(|E| - 1) = 2|V||E| - 2|V| < 2|V||E| - b$, as $b \leq |V|$. So we must have completed every edge-task, which means we must have hired a vertex-worker cover. Since we can get profit $2|V||E| - b$, knowing that the revenue must have been $2|V||E|$, that means that we must have hired at most b workers. In otherwords, we had a vertex cover of size at most b. ∎