Homework 10

Winston (Hanting) Zhang

Friday, December 6, by 23:00

Late Submission. Regardless of how many late days you have remaining, this assignment cannot be submitted more than three days late, i.e., the latest possible submission date is December 9 at 23:00. This is so we can release sample solutions before the weekend, and you can use them to study.

General Instructions. The following assignment is meant to be challenging, and we anticipate that it will take most of you at least 10–15 hours to complete, so please allow yourself plenty of time to work on it. We highly recommend reading the entire assignment right away — you never know when inspiration will strike. Please provide a formal mathematical proof for all your claims, and present runtime guarantees for your algorithms using asymptotic (big-$O/\Omega/\Theta$) notation, unless stated otherwise. You may assume that all basic arithmetic operations (multiplication, subtraction, division, comparison, etc.) take constant time. In problems involving graphs, we typically use n and m to denote the number of nodes and edges, respectively, unless otherwise stated.

Collaboration. Please carefully check the collaboration policy on the course website. When in doubt, ask an instructor.

Consulting outside sources. Please carefully check the policy regarding outside sources on the course website. Again, when in doubt, ask an instructor.

Submission. Homework submission will be through the Gradescope system. Instructions and links have been provided through the course website and Piazza. The only accepted format is PDF. Only typed solutions will be accepted, and we highly recommend producing your solutions using LaTeX (the text markup language we are also using for this assignment).

Recommended practice problems (do not hand in): KT Problems 8.3, 8.4, 8.5, 8.6, 8.12, 8.13, 8.15, 8.16, 8.19, 8.26, 8.27, 8.28, 8.31, 8.37

When you do reductions, you are welcome to use any of the problems that were listed at the end of the 11:00am lecture on November 22 (see notes) — they will also be listed at the beginning of the 2:00pm lecture on November 29. If you want to use any other problem, you have to prove that problem NP-complete yourself first. Unless we give a hint otherwise, hardness proofs can be accomplished reasonably easily using the problems explicitly discussed in class: 3SAT, INDEPENDENT SET, VERTEX COVER, SET COVER.

Problem 1. [10]

Imagine that you are planning the allocation of animals to enclosures in a mountain lion rehabilitation program. The layout of the rehabilitation center is given to you as an undirected graph $G = (V, E)$ with n nodes and m edges, in which the nodes are the enclosures, and the edges capture proximity, i.e., an edge between enclosures u and v means that if you put mountain lions in u and v, they might interact (see below). There are $k_M \geq 0$ male mountain lions, and $k_F \geq 0$ female mountain lions. For each mountain lion i, you are given their sex and their *territoriality* $t_i \geq 0$ (integer). Each enclosure can only contain one animal, because animals of the same sex might fight, and animals of opposite sex might mate, leaving you with even more mountain lions to take care of. If a male mountain lion i is put in an enclosure adjacent to (one or more) other male mountain lions, due to his territoriality, he will create territoriality trouble t_i; similarly, a female mountain lion i in an enclosure adjacent to (one or more) other female mountain lions will create territoriality trouble t_i. You can put male and female mountain lions in enclosures adjacent to each other without any resulting territoriality problems.

Your goal is to assign the mountain lions to enclosures so as to minimize the total territoriality trouble.[1] Phrase this problem as a decision problem and prove that the decision problem is NP-complete.

Definition 1. *Define* TERRITORIALITY *as the following decision problem:*

Given a undirected graph $G = (V, E)$, k_M lions with territorialities t_i and k_F lionesses with territorialities t_i, decide whether there exists an assignment of the nodes such that the total territoriality is less than some K.

Proposition 1. TERRITORIALITY *is in NP.*

Proof. Given an instance of TERRITORIALITY and an assignment of lions and lionesses, we can certify the assignment with the following: For each lion i, iterate through the edges incident the its node. If any adjacent nodes are lions, then add t_i to the total territoriality. Then for each lioness i, iterate through the edges incident the its node. If any adjacent nodes are lionesses, then add t_i to the total territoriality. If the total territoriality in the end is at most K, return "yes," otherwise return "no." This iterates through each node once and each edge twice, so the running time is $O(V + 2E)$, which is polynomial. Thus we have an efficient certifier, and TERRITORIALITY is in NP. □

Proposition 2. TERRITORIALITY *is NP-Hard.*

Proof. We perform a reduction from INDEPENDENT SET to TERRITORIALITY. Given an instance of INDEPENDENT SET with graph $G = (V, E)$ and an integer k, create the following TERRITORIALITY instance on the same graph G: Have k lions and $n - k$ lionesses. The territoriality of each lion i is $t_i = 1$ and the territoriality of each lioness i is $t_i = 0$. Set $K = 0$, i.e. the total territoriality must be exactly 0.

We claim that this is a Karp reduction from INDEPENDENT SET to TERRITORIALITY. Clearly, this construction function run in polynomial time. We next show that our reduction is correct.

Suppose we had a "yes" instance of INDEPENDENT SET. Then there is some $S \subseteq V$ of size k such that no two nodes in S share an edge. Then for the constructed TERRITORIALITY, assign the k lions to S. Because no two nodes in S share an edge, the total territoriality of the lions is 0. Because the territorialities of the lionesses are 0, it doesn't matter if they share an edge. Thus the total territoriality of this assignment is 0; thus this is a "yes" instance of TERRITORIALITY.

We procedd by the contrapositive. Suppose we had a "yes" instance of TERRITORIALITY. Then there is some assignment of the k lions that don't share edges between each other, because otherwise our territoriality would at least be 1. This chosen set S of k nodes is then a independent set of size k in G. This we have a "yes" instance of INDEPENDENT SET.

We have proven that TERRITORIALITY is NP-hard and in NP, thus it is NP-complete. □

[1] Bonus problem: Practice saying "territoriality trouble" ten times in a row fast!

Problem 2. [10]

Remember the THANKSGIVING HUGS problem? Well, now you will show it NP-complete. You have n guests. For each guest i, you are given three integers h_i, s_i, f_i. This means that guest i will be available to hug during the time interval $[s_i, f_i]$. You want to hug them for $h_i \geq 0$ time. You can only hug one person at a time, and hugs cannot be interrupted/resumed.

Show that deciding whether you can hug everyone for the desired amount of time is NP-complete.

Hint 1: SUBSET SUM

Hint 2: Have one guest whom you want to hug for one unit of time who is only available for exactly one unit of time at a specific time.

Proposition 3. *Thanksgiving Hugs is in NP.*

Proof. Suppose we had an input and certificate for THANKSGIVING HUGS with n guests and 3 integers h_i, s_i, f_i for each guest i. The certificate is a set of n times t_i indicating the time to start each hug h_i. Our certifier first sorts the times in ascending order. Then we iterate and check the following conditions for each time t_i:

1. $[t_i, t_i + h_i] \subseteq [s_i, f_i]$. This checks that each hugs conforms to the constraints of our problem.
2. $t_{i-1} + h_{i-1} \leq t_i$ and $t_i + h_i \leq t_{i+1}$. This checks that all of the hugs are disjoint and do not overlap with each other.

If any of these checks fail, we immediately terminate and answer "no." Otherwise, if we go through all the times, we anwer "yes." This is clearly polynomial as our certifier sorts and makes a linear scan, which both take polynomial time in the size of the input. □

Proposition 4. *Thanksgiving Hugs is NP-hard.*

Proof. We will perform a reduction from SUBSET SUM to THANKSGIVING HUGS. Suppose we have an instance of SUBSET SUM, that is n integers $x_1, \ldots, x_n$ and a target integer T. If $\sum_{i=1}^{n} x_i < T$, then output is the impossible $(s_1 = 0, f_1 = 0, h_1 = 1)$. Otherwise, output the following THANKSGIVING HUGS instance: Create n triples where triple i has $h_i = x_i$ and $s_i = 0, f_i = 1 + \sum_{n=1}^{n} x_i$. We also create one extra triple such that $h_0 = 1$, $s_0 = T$, and $f_0 = T + 1$.

We claim that this construction is a Karp reduction from SUBSET SUM to THANKSGIVING HUGS. Clearly the function creating the instances run in polynomial time. Now we will prove the correctness of our reduction.

Suppose we have a "yes" instance of SUBSET SUM, then there exists a set of integers that add up to T. That implies that we have a subset of our $n + 1$ hugs that add up to T. We could start hugging people in this subset one by one with no gaps in between hug times until we reach time T. Then we hug the additional person we've added from T to $T + 1$. At this point, we've hugged for a total of $T + 1$ time, and we have $1 + \sum_{i=1}^{n} x_i - 1 - W$ time left. Our final deadline is $1 + \sum_{i=1}^{n} x_i$, so we have exactly enough time left to hug everyone else. Thus we have a "yes" instance of THANKSGIVING HUGS.

Now the other side. Suppose we have a "no" instance of SUBSET SUM, i.e. we have n integers x_i with no subset that sums to T. If it is the case that $\sum_{i=1}^{n} x_i < T$, then automatically we have a "no" instance $(s_1 = 0, f_1 = 0, h_1 = 1)$ for THANKSGIVING HUGS.

Otherwise, suppose for the sake of contradiction that the THANKSGIVING HUGS instance we construct is a "yes." Then we have $1 + \sum_{i=1}^{n} x_i$ total hugging time and deadlines between 0 and $1 + \sum_{i+1}^{n} x_i$, so we cannot have any gaps between our hugs. We also know that we must have a hug between T and $T + 1$. Thus we be continuously hugging some k people between 0 to T. Thus the hugging times of this subset of k people sum to T! But this is a contradiction since $h_i = x_i$, and we assumed that no subset of the x_i sum to T. Thus we must also have a "no" instance of THANKSGIVING HUGS.

We have proven that THANKSGIVING HUGS is NP-hard and in NP, thus it is NP-complete. □

Problem 3. [10]

A city government is trying to improve their public transportation system. Their idea is to make their Central Station the main hub, allowing you to get to/from Central Station in at most two hops from anywhere in the city.[2] This is modeled as follows: the current transportation system is given as an undirected graph $G = (V, E)$. The Central Station is a given node $s \in V$. They want to add a smallest possible set of additional edges E' such that in the new graph $G' = (V, E \cup E')$, every node has a path of length at most 2 hops to s.

Phrase this problem as a decision problem and prove that it is NP-complete.

Definition 2. *We define the* STATION *decision problem as follows:*

Given an undirected graph $G = (V, E)$, and a start node s, decide whether there exists a set E' of k edges you can add to E such that every node in the graph $G' = (V, E \cup E')$ can be reached within 2 edges of s. Denote a node "reachable" if it can be reached within 2 edges of s.

Proposition 5. STATION *is in NP.*

Proof. Suppose we had an input and certificate for STATION with graph G. The certificate are k edges E' that we add to G. Our certifier simply does BFS with a counter that counts that distance away from the start node s. If at any time the counter for any node increases above 2, we return "no." Otherwise, if our BFS completes, return "yes." This is clearly a correct certifier, and running BFS takes $O(m)$, which is polynomial in out input. □

Proposition 6. STATION *is NP-complete.*

Proof. We perform a reduction from SET COVER to STATION. Given an instance of SET COVER, that is, a universe $U = \{x_1, \ldots, x_n\}$ and m subsets $S_j \subseteq U$, and a target number of subsets k, we construct the following instance of STATION:

1. Create a start node s.
2. Create m subset nodes $s_1, \ldots, s_m$, and make the subset nodes fully connected to each other, i.e. for every $1 \leq j_1, j_2 \leq m$, add the edge (s_{j_1}, s_{j_2}).
3. Create n element nodes $x_1, \ldots, x_n$.
4. For each subset S_j, and for every $x_i \in S_j$, add an edge (x_i, s_j).

Call this graph G.

We claim that this construction is a Karp reduction from SET COVER to STATION. Indeed, the construction itself runs in polynomial time, since we we just linearly iterating through the subsets. Now we will provide the correctness of our reduction.

Suppose we have a "yes" instance of SET COVER. Then there is a family of subsets I with size k such that $\bigcup_{S_j \in I} S_j = U$. Then in G, simply add the k edges (s, s_j) for each $S_j \in I$. Since $\bigcup_{S_j \in I} S_j = U$, every x_i is in some $S_j \in I$. Thus every node x_i is connected to that same S_j. We just added an edge $s \to s_j$, so now we have a path $s \to s_j \to x_i$. This holds for all x_i, so all of the element nodes are reachable. Furthermore, since all the s_j nodes are connected to each other, we can literally take the path $s \to s_j \to s_k$ for any $S_j \in I$ and $1 \leq k \leq m$. So all the subset nodes are reachable as well. Thus all the nodes of G are reachable, and G is a "yes" instance of STATION.

Now for the other side, we process by contrapositive. Suppose the constructed graph G is a "yes" instance of STATION. Then there exists a set of k edges such that adding those k edges makes everything in G reachable. Ideally, we could use the same idea from above and say that all the edges connect to a family of $s_{j_1}, \ldots, s_{j_k}$ nodes, which creates $I = \{S_{j_1}, \ldots, S_{j_k}\}$ that solves the SET COVER instance. However, there are potentially "bad" edges that don't connect from s to s_j directly. So first, we show that these "bad" edges can be removed. There are 3 types of "bad" edges: 1. edges between the element nodes x_i; 2. edges going from s to x_i; and 3. edges going between s_j and x_i. (Note we can't have edges between the subset nodes s_j since that layer is already fully connected.) We proceed by casework:

[2] It's not clear if that's a very good idea — such centralization would mean that if they ever need to temporarily close the Central Station, it will be a disaster. But hey — they didn't ask your opinion about what to do; they only wanted to rely on your algorithmic expertise.

1. Suppose one of our k edges connects (x_i, x_j). Since all nodes in G are reachable, let's analyze paths to x_i and x_j. Now, if both x_i and x_j are reachable via other nodes, i.e. either x_i and x_j directly connect to s, or there is some other node $v \neq x_i, x_j$ such that $s \to v \to x_i, x_j$. Then we can simply remove the edge (x_i, x_j). This doesn't affect the fact that x_i and x_j are reachable, and removing (x_i, x_j) doesn't affect any other nodes, so all nodes in G are still reachable.

   Otherwise, without loss of generality suppose x_j is only reachable via $s \to x_i \to x_j$. Note that we can assume x_i connects to s, or else both x_i would be at least 2 away from s, making x_j at least 3 away from s, thus contradicting the reachablility of x_j. Then again we can simply remove (x_i, x_j) and add (s, x_j). Again this doesn't affect any other nodes, and we maintain that x_i and x_j are both still reachable.

   Thus in this case, we can always remove the (x_i, x_j) edge while maintaining a valid "yes" instance.

2. Suppose one of our k edges connects (s, x_i). Since all of the S_j in total form a cover of U, there is at least one j such that $x_i \in S_j$. Then we might as well remove (s, x_i) and add (s, s_j). This doesn't destroy any paths to other element nodes x_j, like $s \to x_i \to x_j$, since in the above case we proved that we can remove all (x_i, x_j) edges. Thus in this case, we can always remove the (s, x_i) edge while maintaining a valid "yes" instance.

3. Finally, suppose one of our k edges connects (s_j, x_i). Again, there is at least one other j' such that $x_i \in S_{j'}$. Then we might as well remove (s_j, x_i) and add $(s, s_{j'})$. This doesn't destroy any paths from like $s \to x_i \to s_j$, since from the above we showed that we can remove all edges between $s \to x_i$. Furthermore, we maintain that x_i is reachable via $s \to s_{j'} \to x_i$. Thus in this case too, we maintain a valid "yes" instance.

So, we have shown that we can remove all "bad" edges and replace them with connections bewteen (s, s_j). Then take these k connections and collect the subsets S_j that are connected. Since all the element nodes of G are reachable via some $s \to s_j \to x_i$, we have that S_j forms a cover of U. Thus we also have a "yes" instance of SET COVER.

Thus we have a proper reduction from SET COVER to STATION, so STATION is NP-hard, and thus also NP-complete. □

Problem 4. [10]

Prove that the following problem is undecidable: Given a C++ program P which takes as input a graph G with edge costs c_e, does P correctly compute a Minimum Spanning Tree of its input (for each input graph)?

Note: This problem will not make sense until Tuesday at the earliest, possibly Thursday.

Proof. Denote the problem by COMPUTEMST. We will give reduction from HALT to COMPUTEMST. For any program P instance of HALT, define a program Q which takes as input a simple graph G as follows: Q will first run $P(P)$. Then it will return the MST of G by running Kruskal's Algorithm. Our reduction is computable in polynomial time as the code for P is given and Kruskal's Algorithm is computable.

Now let's analyze the behavior of Q. Suppose P is a "yes" instance of HALT. Then $P(P)$ terminates, and thus Q terminates – therefore Q is a "yes" instance of COMPUTEMST. For the other side, suppose P is a "no" instance of HALT. Then $P(P)$ does not terminate, and thus Q does not terminate – therefore Q is a "no" instance of COMPUTEMST.

So we have constructed a reduction from HALT to COMPUTEMST. Thus COMPUTEMST is undecidable. □

Problem 0. [0]

Chocolate Problem (1 chocolate bar): Imagine that you own a toll road, and want to set prices. The toll road has m ramps to get on/off, numbered $1, 2, \ldots, m$ from start to end of the road (with 1 being the start and m the endpoint). This divides your toll road into segments $T_j = [j, j+1)$. You are given a list of n drivers. For each driver i, you are told the starting and ending ramp of their trip $s_i < t_i$, as well as a budget b_i. To drive from s_i to t_i, the driver has to drive along all the intermediate segments $T_{s_i}, T_{s_i+1}, \ldots, T_{t_i-1}$.

You get to choose a price $p_j \geq 0$ for each segment T_j. If the sum of prices on the trip is affordable to the driver (at most b_i), then they will drive on your toll road and pay you the total for the segments. If the sum exceeds b_i, driver

i will take surface streets for the entire trip, and not pay you anything. Your goal is to choose prices for the segments that maximize your revenue from the drivers under this model.

Phrase this problem as a decision problem, and prove that it is NP-complete.