# CSCI270 Homework 10

Jacob Ma

December 9, 2022

**Problem 1**

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.[a] Phrase this problem as a decision problem and prove that the decision problem is NP-complete.

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

**Solution**

DECISION: Given graph $G = (V, E)$ with $n$ nodes and $m$ edges, $k_M \geqslant 0$ male mountain lions, $k_F \geqslant 0$ female mountain lions, and *territoriality* $t_i \geqslant 0 \in \mathbb{Z}$ for each lion $i$, and an integer $C$, does there exist an assignment of lions to enclosures with total territoriality trouble at most $C$.

**Certificate**: One assignment of $k_F$ female lions and $k_M$ male lions to $n$ enclosures with $\{t_i\}_{i=1}^{k_F+k_M}$ territoriality.

**Certifier**: We iterate through all nodes $u$ in $V$ for the given assignment, and check for all its adjacent edges $e = (u, v)$ to see if the lion in $v$ is of the same sex. If there exists an adjacent edge $e = (u, v)$ where $u, v$ are of same sex, add $t_i$ to the total territoriality counter $T$. Also, while iterating through all $u \in V$, keep track of number of male and female lions by giving two counters $T_f$ and $T_m$, adding them by $1$ if the lion is male / female. After iterating all the nodes, if $T_f \neq k_f$ or $T_m$ or $T > C$, return "No"; else, return "Yes".

**Runtime**: Since we are only iterating all the node in $V$ and iterate all its adjacent edges in $E$, and $|V| + |E|$

is finite, the certifier will terminate in finite steps, and takes at most $\mathcal{O}(m+n)$ iterations where $m$ is $|E|$, $n$ is $|V|$. And also, all the operations done on each node and each edge is only a judging and number increment, it will take constant time. Thus, the total Certifier will run in $\mathcal{O}(m+n)$ time, which is polynomial.
Thus, the DECISION itself is a NP problem.

Now it left to be proven that DECISION is NP-hard, i.e. some NP-complete problems $\leqslant_p$ DECISION. In this case, we try to prove INDEPENDENT SET $\leqslant_p$ DECISION.

**Lemma 0.0.1**

INDEPENDENT SET $\leqslant_p$ DECISION

*Proof of Lemma.* This can be easily done by reducing INDEPENDENT SET to DECISION, and one of the reduction function could be the following. (The idea is trying to make all $t_i$ to $\infty$, and make $k$ to only male lions with $0$ female lions, to see if we can have assignment with total territoriality trouble at $0$)

**Algorithm 1:** INDEPENDENT SET Reduce to DECISION

```
  Data: Input graph $G = (V, E)$, integer $k$
1 Make $k_M = k$, $k_F = 0$, and $C = 0$;
2 Make all $\{t_i\}_{i=1}^{k_M+k_F} = \infty$ ;
3 return $G, k_M, k_F, C, \{t_i\}_i^{k_M+k_F}$
```

Since all male lions has $t_i = \infty$ and we only have male lions, any two adjacent lions will make the DECISION returns "No"; If there does not exist a valid assignment, it will also returns "No"; DECISION will only returns "Yes" when there exists a valid assignment with $k$ nodes in a set where no two nodes are adjacent. This algorithm also only iterates all $t_i$ with constant operations, thus is polynomial. □

Since DECISION is both NP and NP-hard, it is proved to be NP-complete, thus done.

**Problem 2**

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.

*Proof.* We first need to show THANKSGIVING HUGS is NP

**Lemma 0.0.2**

THANKSGIVING HUGS is NP.

*Proof of Lemma.* Let's consider the certificate and certifier of this problem.

**Certificate**: A sequence $S$ of hugging people.

**Certifier**: If $|S| < n$, returns "No" since you are not hugging everyone; Iterate through the sequence, for each $i$, if $f_i > s_{i+1}$, returns "No" since the sequence if not valid and some hugging is interrupted; When reaches to the end of the sequence, where you are hugging some people $i$, if $f_i > T$, where $T$ is the input desired time, returns "No"; returns "Yes" otherwise.

**Runtime**: Since we are iterating through the set $S$, and $f_i > s_i$ for all people $i$, and the program directly terminates and returns "No" if $f_i < s_{i+1}$, the iteration will iterate $|S|$ times at most, which is $\mathcal{O}(|S|)$ iterations. In each iteration, we are only operating constant operations, thus the overall runt's is $\mathcal{O}(|S|)$, which is polynomial. (Also since valid certificate means $|S| \leqslant f(n)$ , where $f$ is a polynomial function, thus $\mathcal{O}(|S|)$ polynomial).

Thus, THANKSGIVING HUGS is NP. □

Now, it left to be proven that THANKSGIVING HUGS is NP-hard, i.e. some NP-complete problem could be reduced to THANKSGIVING HUGS , in this case, we make it SUBSET SUMS.

**Note**: Instead of using the definition on KT, we are using the decision of SUBSET SUMS during the discussion as follows:

SUBSET SUMS: : Given n natural numbers $w_1, w_2, \cdots, w_n$ (and a number $W$), is there a subset of $w_1, w_2, \cdots, w_n$ whose sum is exactly $C$?

**Lemma 0.0.3**

$$\text{SUBSET SUMS} \leqslant_p \text{THANKSGIVING HUGS}$$

**Algorithm 2:** SUBSET SUMS Reduce to THANKSGIVING HUGS

```
Data: Input weight set $S = \{w_i\}_{i=1}^{k}$, Projected Sum integer $C$
1  Let integer $W = 0$ ;
2  for all $w_i$ in $S$ do
3      $W += w_i$ ;
4  if $C > W$ then
5      return "No"
6  Initialize a set of guests $G = \{\}$ ;
7  for all $w_i$ in $S$ do
8      Construct a guest $g_i$ where $h_i = w_i$ and $[s_i, f_i] = [0, W+1]$;
9      Add $g_i$ to $G$;
10 Add another guest $g_u$ where $h_i = 1$ and $[s_i, f_i] = [C, C+1]$ to $G$;
11 return THANKSGIVING HUGS($G, W+1$)
```

*Proof of lemma.* Consider the deduction function above. The main idea is that we are making all $[s_i, f_i]$ extend to $[0, W+1]$, where $W$ is the sum of all weights, and make $s_i = w_i$ for all weights $w_i$ in $\{w_i\}_{i=1}^n$. Then, add a "helper guest" who is only amiable to hug during $[C, C+1]$ for 1 unit of time in to the guest set, than run the THANKSGIVING HUGS.

Firstly, the algorithm will directly returns "No" when $C > W$ since it is impossible to have the sum of subset bigger than the sum of the whole set itself.
We know that

$$W = \sum_{w_i \in S} w_i$$

Thus, $\{w_i\}_{i=1}^n$ can be arranged in any order as long as there is no time elapse between any two hugs, and THANKSGIVING HUGS will guaranteed to return "Yes" if we put these guests and $W$ as input.
Then, we add a "helper guest" who has $[s_i, f_i] = [C, C+1]$, and $h_i = 1$. After adding this guest to the guest set and input alongside with $W+1$. For the same reasoning, we know that there could not be any time elapse between any two hugs. Thus, in order to return "Yes" to this question, there are subsets of $S$ such that the sum of the subset is $K$ and $W - K$.
Thus, THANKSGIVING HUGS would only return "Yes" if and only if there is a subset of guests could be hugged in the first $C$ amount of time without any time elapsed in between.

Consider this algorithm, it is also only iterating the set $S$ twice, operating constant operations, thus, the runtime is $\mathcal{O}(|S| = n)$, which is polynomial. □

Thus, THANKSGIVING HUGS is NP-complete. □

## Problem 3

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.[a] 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.

[a] 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.

*Proof.* DECISION: Given an underacted graph $G = (V, E)$, a central station node $s \in V$, and a constant integer $C$, is there a set of edges $E'$ such that makes a new graph $G' = (V, E \cup E')$ where every node has a path at most 2 hops to $s$, where $|E'|$ is at most $C$?
CERTIFICATE: A set of edges $E'$.
CERTIFIER: If $|E'| > C$, directly returns "No"; Otherwise, iterate through all the nodes $v \in V$, run DFS on graph $G' = (V, E \cup E')$ from $v$ to $s$, getting a path $p$, if length$(p) > 2$, directly returns "No"; If we iterate all the nodes without returning "No", we return "Yes".

**Runtime**: Other than the iteration through all the vertices, we have a constant operations in if statement and returning things. As for the iteration, it has $|V| = n$ iterations, and each iteration is dominated by the BFS search where other operations are constant time. The BFS search takes $\mathcal{O}(m + n)$ for each iteration, thus takes $\mathcal{O}(n(m + n))$ for the whole certifier, which is polynomial.

Thus, we now know DECISION is NP. Now it suffices to show DECISION is NP-hard, i.e. some NP-complete problem $\leqslant_p$ DECISION. We are using SET COVER as the NP-complete problem.

**Lemma 0.0.4**

SET COVER $\leqslant_p$ DECISION

**Algorithm 3:** SET COVER Reduce to DECISION

```
   Data: Input Set $\{S_i = \{v_1, \cdots, v_l\}\}_{i=1}^{p}$, Integer $k$
1  Initialize a graph $G = (V, E)$, dummy station set $D = \{\}$ ;
2  Let $C = k$;
3  for each set $S_i = \{v_1, \cdots, v_l\}$ do
4      Construct a dummy station $D_i$, add $D_i$ to $V$ and $D$;
5      for each node $v_j \in S_i$ do
6          Connect $v_j$ and $D_i$ with an edge $e = (v_j, D_i)$ ;
7          Add $e$ to $E$, add $v_j$ to $V$ ;
8  for each dummy station $D_i \in D$ do
9      for each dummy station $D_j \neq D_i \in D$ do
10         if $(D_i, D_j)$ is not connected then
11             Construct $e = (D_i, D_j)$, add $e$ to $E$ ;
12 return DECISION$(G, C)$
```

Consider the reduction above, the main idea is that we are constructing a dummy station $D_i$ for each set index $i$, connecting all the nodes(stations) inside the element; add a virtual "central station" point $s$; then, connecting all these dummy stations $\{D_i\}_{i=1}^{p=\#\text{ of sets}}$, ensuring connecting any one of the dummy stations will make all other dummy stations have a path no more than $2$ hops to $s$ (And we need to connect at least $1$ dummy station, which will be shown below); at last, pass into the DECISION with the newly constructed graph $G$ and $C = k$, where $k$ is the integer input in DECISION and wait for return.

It suffices to prove the following lemmas:

**Lemma 0.0.5**

DECISION would prefer to choose to connect dummy stations $D_i$ than real elements(stations) $v_i$ to "central station" $s$.

*Proof of Lemma.* Consider the new nodes that would be able to reach $s$ in two hops after connecting $D_i$ and $v_i$ :

$$\begin{cases} D_i : \{D_i\}_{i=1}^{p} \cup \{v_i\}^{\text{adjacent } v_i} \\ v_i : \left(D' = \{D_i\}^{\text{adjacent } D_i}\right) \cup v_i \end{cases}$$

Since $D' \subseteq \{D_i\}_{i=1}^{p}$ and $|v_i| = 1 \leqslant |\{v_i\}^{\text{adjacent } v_i}|$, we know the number of nodes covered within 2 hops from $s$ after connecting $D_i \geqslant$ connecting $v_i$.

And they are only equal when $D' = \{D_i\}_{i=1}^{p}$ and $v_i = S_i$, meaning $v_i$ is the only element in the universe, and it suffices to choose any dummy points or $v_i$ to make all $v \in V$ 2 hops from $s$, which is also consistent with our algorithm, and it does not matter if we choose $D_i$ or $v_i$ in this case.

Thus, we can say that the algorithm will choose $D_i$ to connect since $|V|$ is invariable, and number of new nodes covered after connecting $D_i$ is more. □

**Lemma 0.0.6**

DECISION will connect to at least 1 $D_i$ in $E'$ if there are at least 2 elements in the universe.

*Proof of Lemma.* As we discussed in the previous lemma, the algorithm will tend to connect $D_i$, and the central station is not connect to any station before entering the algorithm. Thus, the first edge will connect $s$ and some $D_i$. Also, this will also suffices to show that DECISION will be able to cover all $D_i$ within 2 hops range as long as we are choosing any random set. □

**Lemma 0.0.7**

Any $v_i$ will only be able to be in 2 hops of distance from $s$ only if some $D_i$ is connected to $s$, and $v_i \in S_i$.

*Proof of Lemma.* We have proved that $v_i$ will not be directly connected to $s$, thus there will be at least 2 hops from $s$, which should at least contain $s$ and some $D_i$ where $v_i \in S_i$.

Assume to the contrary, $v_i$ is connected to $s$ not through some $D_i$ where $v_i \in S_i$. Since all $v_i \in \{v_i\}_{i=1}^{p}$ is not connected to $s$, and $v_i$ is not directly connect to any $D_j$ where $v_i \notin D_j$: $v_i$ must first connect to some $v_j$, and $v_j$ connect to some $D_j$ where $v_j \in S_j$, and $D_j$ connect to $s$. However, this is more than 2 hops, thus exists contradiction. □

Thus, it also suffices to say that connecting a dummy node with $s$ means making all and only all the nodes in the set dummy nodes represented within 2 hops away from $s$.

Also, it is obvious to observe the following:

**Lemma 0.0.8**

If we connect $s$ to some $D_i$, all elements $v_i \in S_i$ will be in 2 hops away from $s$.

*Proof of Lemma.* There exists a path $v_i \to D_i \to s$, 2 hops, done. □

Thus, with all these lemmas, it suffices to say that DECISION will return "Yes" if and only if we can connect $|E'|$ different $D_i$ to $s$, to cover all the $v_i$ within 2 hops from $s$, and the $|E'| \leqslant k$; returns "No" otherwise.

At last

**Lemma 0.0.9**

The algorithm terminates, and run in $\mathcal{O}\left(l^2\right)$ time, which is polynomial.

*Proof of lemma.* There is no while loop in the reduction algorithm, thus terminates.
Line 1 and 2 are constant runtime. And from 3 to 7 are actually iterating all $v_i$ in the universe, which takes $n$ iterations ($n$ is total number of elements). From 8 to 11, since each set has a dummy station $D_i$, we are connecting all the $D_i$ to each other, which takes $l\left(l-1\right)$ iterations, and constant operation in each iteration.
Thus, the total run time is $\mathcal{O}\left(l\left(l-1\right)\cdot 1\right) = \mathcal{O}\left(l^2\right)$, which dominates other runtime.
Since the overall runtime is $\mathcal{O}\left(l^2\right)$, the reduction is polynomial. □

Thus, it now suffices to show DECISION is NP-hard. And since it is also NP, it is NP-complete. □

**Problem 4**

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.* Call this problem MST DECISION. We prove by reduce from HALT: given a program $P$, decide whether or not $P\left(P\right)$ halts.
Our reduction takes an input $P$ to HALT and maps it to $Q_P$ . We define $Q_P$ as the program that takes in $G$ and $c_e$, then runs $P\left(P\right)$, then returns KRUSCAL'S MINIMUM SPANNING ALGORITHM using $G$ and $c_e$.

**Algorithm 4:** HALT Reduce to MST DECISION

```
Data: Function $Q_P$, $G$, $c_e$
1 Run $P(P)$ ;
2 return KRUECAL'S$(G, c_e)$
```

If $P\left(P\right)$ halts, then $Q_P$ halts and returns minimum spanning tree on input $G, c_e$, and thus $Q_P$ is in MST DECISION.
If $P\left(P\right)$ does not halt, then that means $Q_P$ does not halt on any input $G, c_e$ since $Q_P\left(G, c_e\right)$ calls $P\left(P\right)$. Thus, $Q_P$ does not terminate for any $G, c_e$, meaning that $Q_P \notin$MST DECISION.
Lastly, the reduction constructs the program $Q_P$ by simply adding the source code for $P$, calling $P\left(P\right)$, then returning KRUSCAL'S which will terminate, and is thus computable.
Thus, our reduction is a valid many-to-one reduction. □