Algos reductions

Bayard Walsh

March 2023

1 Introduction

problem 1
Problem Statement: From the description $G = (V, E, s, t), L$, please describe how to build a related directed graph $G' = (V', E', s', t')$ such that: Rabbit and Frog can save the world in $G = (V, E, s, t), L$ IF, AND ONLY IF, there is a directed path from s' to t' in G' . Prove the correctness property above for your reduction, by proving both directions in the "if and only if".

Reduction:
Starting with $G = (V, E, s, t), L$, we will performing the following steps to obtain $G' = (V', E', s', t')$.

First, we will introduce the subroutine, $p(u, v)$ which is defined as follows:
Given two nodes u and v, return a set of edges E_p, where E_p is the list of **all** possible outgoing edges **which are "reachable"** from u, given our other path is currently at v, with respect to the graph G. We are defining a "reachable" edge as $L(e) = NULL$ or $L(e) = v$. We will add every possible edge from G that is either "reachable" from u or is "reachable" from some vertex u' that we can reach starting at u along a path where every edge is "reachable" consider our other path is currently at v. Additionally, we will label edges from our $p(u, v)$ function with respect to the location of u and v. For example, say we can add edge (u, u') to our $p(u, v)$ edge set, because it is "reachable". Then we will label the edge as $((u, v), (u', v))$ and add that edge to E_p, because we can reach u and u' from one path and the other path stays at v. The important part is that we maintain that the edge is reachable with respect to our other path being at v, and this data is stored in every node. Note, we will only return a set of edges to be added to G', and we will add every potential (u, v) pair as a node to G', so therefore we know that any edge we return can be added to the graph.

With our $p(u, v)$ subroutine defined, we will now move to reduction implementation.

1. Create a matrix S such that if $G = (V, E)$, we have $S = |VxV|$. We will use this matrix to store our reachable edge sets from a given (u, v) node. Next,

compute all possible (u, v) pairs made from V. Note that a given (u, v) pair is **one** node, and will represent the current state of the two paths. If $G' = (V', E')$, add all these pairs as nodes to V'. Therefore we will have V^2 total nodes $\in V'$. Note that we also leave all nodes unconnected at this point.

2. For every possible (u, v) pair from V, compute $p(u, v)$, and store each given edge set in $S[u][v]$. There will be V^2 of these problems computed.

3. If $G' = (V', E')$, then $E' \cup S[s][s]$, or add the edge set of reachable edges from (s, s) to G'. Then create list of tuples T, and we have $T = T \cup [s, s]$, which will be a matrix where we will keep track of the reachable edges we have already added.

4. For every node $n = (u, v)$ in the connected component with (s, s) and $n \notin T$, we will append the reachable edges set $S[u][v]$ and $S[v][u]$ to G', and then $T = T \cup [v, u] \cup [u, v]$. We will also add an edge to connect (u, v) to (v, u), because while they are different nodes in our implementation, in relation to G they represent the same data. Therefore if any edge is connected to one of them from (s, s) (which is the only way to be added) it will now be connected to both

Some Notes on this step: This computation will cause G' to grow its amount of edges and therefore create more available nodes based on nodes added by previous edges and so on, which is the intention of the algorithm as we want paths from (s, s) to all (u, v) nodes that represent a reachable state for our two paths with respect to G. We also may have duplicate edges from an incoming edge set which is already in G', in which case we will only maintain one edge in G'. However note that first, our amount of nodes are set before this step, and we never add more nodes to G' after initiating the V^2 amount in the first step, and second our matrix T will bound the amount of edge checking computations to worst case V^2 iterations. Also note that we append the edge set from (v, u) and (u, v) in a given step, as if we achieve (v, u), we also have (u, v), because we can have either path explore more paths and or stay at the node, meaning both are reachable, even if they have different edge sets

5. $s' = (s, s)$ and $t' = (u, t)$ **AND** (t, u), where u is any node in V. Note here that if we have t for either u or v for any (u, v), we know that Rabbit and Frog can save the world, because that node represents a state where either Rabbit or Frog are at t

After we have completed these steps for $G = (V, E, s, t), L$, we have $G' = (V', E', s', t')$.

I will prove feasibility for our reduction, or show that given $G = (V, E, s, t), L$ we can create $G' = (V', E', s', t')$ in polynomial run time, based on analyzing each step made:

1. Initializes a matrix S of size VxV, then compute V^2 pairs and add to

$G' = (V', E')$. Both of these operations can be done in $O(V^2)$ run time.

2. We will consider computation for a single sub problem, then apply to every sub problem. Finding all reachable edges from a given node u if the other path is at v for G is worst case E run time, as we will check every edge for viability and can check every possible edge. We will do this work $O(V^2)$ times, so we have a bound of $O(E \cdot V^2)$ run time.

3. Here we add a single edge set to G', and create a list of tuples T. Initializing T and adding $[s, s]$ should be $O(1)$, and adding a single edge set to G' is worst case $O(E)$ run time.

4. Here, we can potentially add V^2 edges sets to G', and each edge set can potentially have E edges. Therefore, we have worst case $O(E \cdot V^2)$ run time. Note that once a node has had its edge set added once, we record that in T ($O(1)$ run time), which is a bound for our computation.

5. $O(1)$ for assigning constant nodes.

As each incremental step is polynomial run time, we know that we have an overall polynomial reduction, which proves feasibility.

Before the formal proof, I will prove the following:

Lemma 1: For G', every node (u, v) with a reachable path from (s, s) represents a reachable state where Rabbit is at u and Frog is at v on G, given both start at s
Base case, distance from (s, s) to (u, v) is 1
As both animals start at (s, s), and if there is some reachable path from (s, s) to (u, v) that is 1 long, then we know there must be an edge from (s, s) to (u, v). The only way for an edge to be added is if (u, v) is directly reachable from (s, s). Therefore, we know that Rabbit and Frog can reach this state by having one of them move to u on G, which we know is reachable because (u, v) is only added to the connected component with (s, s) if it is reachable with respect to G'. This proves our base case.

Now we will assume that any node (u', v') with a path k distance from (s, s) represents a reachable state for Rabbit and Frog on G.

Induction, distance from (s, s) to (u, v) is $k + 1$
We know that (s, s) can reach (u, v) in $k+1$ steps, and we know that any (u', v') with a path k distance from (s, s) represents a reachable state. Therefore, let (u', v') be the last node before (u, v) on the path from (s, s) to (u, v), which we know exists by assumption. Therefore we must show that from given Rabbit and Frog are at (u', v'), they can move to the state (u, v). However, as (u', v') has an edge to (u, v), we know the only way an edge could be added is if (u, v)

is a "reachable" state for Rabbit and Frog from (u', v'). Therefore, we know that (u, v) is a reachable state for Rabbit and Frog.

Lemma 2: If Rabbit and Frog both start at s and can do some number of steps where Rabbit reaches v and Frog is at u, with respect on G, then there will be some node (u, v) or (v, u), on G' connected to (s, s)
As we define (u, v) on G' as a state of Rabbit and Frog on G, we will do induction on the number of "steps" from the start to reach the state. A step is either Rabbit or Frog crossing an edge.
Base case, steps for Rabbit to reach u and Frog to reach v is 1
Then we know that either Frog or Rabbit move one step from s. This means that this edge, u' is "reachable" from s, given the other path stays at s. Therefore it will be added to G', so base case is correct.

Now we will assume that for any amount of steps k to reach a reachable state for Rabbit, u' and Frog, v' on G, there will be some node (u', v') on G' such that (u', v') is connected to (s, s).

Induction, steps for Rabbit to reach u and Frog to reach v is $k + 1$
By inductive hypothesis, we know that we will have a node after k steps. We also know that we can do some number of steps where Rabbit reaches v and Frog is at u. Therefore, have (u', v'), a node on G' connected to (s, s) by hypothesis, be a node that represents that state of Rabbit and Frog after k steps. As we know that we can have Rabbit reach u and Frog reach v by assumption, then there must be some valid step from the states u' and v' to u and v. Therefore, from the node (u', v') there must be some outgoing edge to (u, v), because we check every possible edge in our reduction, and we will check (u', v') as it is in G'. As (s, s) is connected to (u', v') by assumption, we know that there is some edge from (s, s) to (u, v). This proves our induction.

Now to prove correctness, I will show that $G = (V, E, s, t), L <=> G' = (V', E', s', t')$ for the given problem statement.

If $G = (V, E, s, t), L$, then $G' = (V', E', s', t')$
Here, we know there must be some valid state where Rabbit is at t or Frog is at t, because they save the world. Therefore, we will apply **Lemma 2**, with regard to the state where one is at t. Therefore, this proves that there must be some connected path (u, t) or (t, u) from (s, s), proving there is a path on G' given we have the save the world problem.

If $G' = (V', E', s', t')$, then $G = (V, E, s, t), L$
Here, we know there must be some reachable path from (s, s) to either (u, t) or (t, u) in G', as given. Therefore, we will apply **Lemma 1**, with regard to this (u, t) or (t, u). Therefore, this proves that there must be reachable state where Rabbit as at t or Frog is at t. Regardless, one of the paths has reached t, so this proves our save the world problem, given we have solved G'.

Now we have $G = (V, E, s, t), L <=> G' = (V', E', s', t')$ which completes the reduction relationship stated in the problem statement.

problem 2

Problem Statement: Please give a reduction from the INDEPENDENT SET problem to HAPPY HERMITS. That is, describe a transformation that, given a pair (G, k), outputs a pair (H, r) such that (G contains an independent set of k vertices, i.e., a set of k vertices no two of which are adjacent in G) **IF, AND ONLY IF**, (H contains r vertices each at distance at least 3 from each other)

Reduction:
Starting with (G, k), we will performing the following steps to obtain (H, r).

1. Add every vertex from $G = (E, V)$ to H, or $H = (E', V')$, $V' \cup V$

2. For $G = (E, V)$, and for every edge $e \in E$, where $e = (u, v)$: create vertex, v_n, create two new edges $e_1 = (u, v_n)$ and $e_2 = (v_n, v)$, then $H(E', V') = E' \cup e_1 \cup e_2$ and $V' \cup v_n$.

3. $r = k$

After we have completed these steps for (G, k), we have (H, r).

I will prove feasibility for our reduction, or show that given (G, k) we can create (H, r) in polynomial run time, based on analyzing each step made:

1. Copies all vertices to another graph, which can be done in $O(V)$ runtime.

2. We will consider computation for a single edge, then apply to every edge. Creating a new vertex, two new edges, and then adding the vertex to V' and the two edges to E' are all $O(1)$ operations. We repeat this for E edges, so $O(E)$ runtime.

3. $O(1)$, assignment of a variable

As all our steps our polynomial and done in sequence, our reduction is a feasible operation in polynomial run time.

Before continuing with the proof we will prove the following lemma:

Lemma 1: Assume u, v are both in G and H. If two vertices u, v are not adjacent in G, then we know the minimum possible distance from u, v in H is 4.
In our reduction, for every edge $e = (u', v')$ in G we created a new node v_n in between (u', v') and added $e_1 = (u', v_n)$ and $e_2 = (v_n, v')$, in place of e. Therefore, for every $e = (u', v')$, in G, we replaced it with two edges e_1 and e_2, on the path of (u', v') for H. Also note that **we only add edges through building new nodes, and none of the original edges from G are in H.** Therefore, for any (u', v') path in G, we know that we have replaced every edge

on the path with two other edges; so we have essentially doubled the length of the (u', v') path from G to H. As we know (u, v) are both in G and H by assumption, if they are not adjacent in G, then the minimum path must be 2 in G and therefore 4 in H, by our doubling property, completing our proof.

Lemma 2: Assume u, v are both in G and H. If two vertices u, v have a minimum possible of 3 in H, then we know they are not adjacent in G
In our reduction from G to H, we add a number of nodes, such that each node added to H replaces an edge in G with itself and two more edges, which maintains connection but doubles distance between any (u', v') path from G to H, which was proved above. Therefore, the only difference between G and H is the doubling of edges with the addition of these middle vertices. As we know u, v are both in G and H, they cannot be removed, and as we know the u, v have a minimum possible of 3 in H, then at least two nodes, say x and y along the path which must be removed for u, v to be adjacent in G. However, we only add nodes in H as replacements of edges, and therefore to preexisting nodes in G (as each node replaces an edge between two nodes in G), and if we want to replaced both x and y they cannot be connected, so therefore they can't be on the path and we have a contradiction. Therefore, if two vertices have a minimum distance of 3 in H, then they can not be adjacent in G.

Lemma 3: If we have a set S_H of hermit nodes for H, we will always have a set S_G, where $|S_G| = |S_H|$ of independent nodes for G
We want to consider if some node in S_H is not in G. Therefore we will prove this lemma to show that we can always create an independent set S_G for G, such that $|S_G| = |S_H|$. We will now do case analysis.
Let u, v be two hermit nodes in S_H, so that v is closet node to u. Therefore consider the following cases:

Neither u, v is deleted: As neither are deleted, here we can simply apply **Lemma** 2, because we meet the assumption. Therefore, we can keep these nodes from S_G, so we have the same sized set.

v is deleted: Therefore, v must be some node added to replace some edge e in G, and therefore must be in between two other nodes in H. Therefore, there must be some other node v' which is adjacent to v in H, and is in G, because v was created to replace some edge in G. Furthermore, let us pick the v' that is farther from u. We know that v was connected to two edges in H, because that's how we added nodes from G to H, and therefore there on the shortest u, v path there must be some node on the path and some node with is one edge farther than v. Because v was at least 3 edges away from u, and v' is one more edge away from v, u, v' are both in G and H and the minimum possible distance is greater than 2, so we apply Lemma 2, and know they are not adjacent. Additionally, if v' is adjacent to v in H, it cannot be in the set of S_H. This proves that even if we lose an edge in u, v, we can always pick a v'

to make up for that lost edge, meaning we can have an independent set of the same size.

u, v **are both deleted:** Therefore, both u and v must be nodes added in between two other nodes in the creation of H from G. Therefore, there must be two other nodes u' and v' which are both adjacent to u and v and currently not in S_H as they would violate the hermit set status. Therefore, we will pick u', such that u' is closer to v than u and v' such that v' is farther from u than v and add them to our set. We know the two edges are not adjacent because the minimum distance between any two edges u, v is 3, and as u' is adjacent and closer to v, it is 2 from v, and as v' is adjacent and farther from u, we know it is 3 from u' along the same path. Therefore we have distance 3 for (u', v'), and as u', v' are in both G and H (we deleted their neighbors so they must be in both), we can apply **Lemma** 2 and add them both. Therefore we add both these vertices to S_H, meaning we haven't lost any vertices from the two we have deleted

Therefore, we can consider every case between two vertices, we can always add back the amount of vertices deleted. This proves our proof, implying that we can always generate an S_G such that $|S_G| = |S_H|$, based on the particular way that we add nodes from G to H.

Now to prove correctness, I will show that (G, k) <=> (H, r) for the given problem statement.

if (G, k)**, then** (H, r)
Therefore on the graph G we have an independent set of vertices S_G such that $|S_G| \geq k$, and we want to show we have a hermit set S_H for the graph H, such that $|S_H| \geq r$. Note that in our reduction, we only create new nodes, and don't delete any nodes from G to H. Therefore every node in G is in H. Now consider the graph H. As proved in Lemma 1 above (note that assumption is met by confines of the problem), for any u, v pair that aren't adjacent in G, the minimum distance of the u, v pair is 4 in H, meeting the conifenes of the hermit set. As every node in G is in H, and S_G is a set of k vertices where no two vertices are adjacent in G, then S_G must also be a set of vertices where the minimum distance is 4 in H. Therefore, we have $S_H = S_G$ is a valid set for H, and as we have $r = k$ from our reduction, we have $|S_H| = |S_G| \geq k = r$, which implies $|S_H| \geq r$, finishing our proof.

if (H, r)**, then** (G, k)
Therefore on the graph H we have a hermit set of vertices S such that $|S_H| \geq r$, and we want to show we have a independent set S_G for the graph G, such that $|S_G| \geq k$. Here we are faced with a considerable issue; not every node in H is in G, and therefore we could have a node in S_H which is not in the set we want to make S_G. This issue is the inspiration for **Lemma** 3. As proved in our Lemma 3 above, if we have $|S_H|$ for H, we have a set for G, S_G, such that

$|S_H| = |S_G|$. Therefore, we can create a set such that $|S_H| = |S_G| \geq k$, and as we have $k = r$ from our reduction, we have $|S_G| = |S_H| \geq r = k$, which implies $|S_G| \geq k$, finishing our proof.

Now we have (G, k) <=> (H, r) which completes the reduction relationship stated in the problem statement.