Problem 1: The Class P (35 points)

1. (25) Show that the class P is closed under union, intersection, concatenation and complement.

Assume and are their TMs. For

We create a DTM that accepts input s and

* Run and
  + If and are both , so polynomial time, then is , which is also polynomial time.
* Accepts when either and accept.
* Rejects otherwise.

Then all languages of class P are closed under union since they all decide in polynomial time.

We create a DTM that accepts input s and

* Run and
  + If and are both , so polynomial time, then is , which is also polynomial time.
* Accepts when both and accept.
* Rejects otherwise.

Then all languages of class P are closed under intersection since they all decide in polynomial time.

We create a DTM that accepts input s and

* For each two substrings of s where
* Run and
  + If and are both , so polynomial time, then is , which is also polynomial time.
* Accepts when both and accept.
* Rejects otherwise.

Then all languages of class P are closed under concatenation since they all decide in polynomial time.

We create a DTM that accepts input s and

* Assume s’ is the reverse of s, or its complement.
* Run and
  + If and are both , so polynomial time, then is , which is still polynomial time.
* Accepts when both and accept.
* Rejects otherwise.

Then all languages of class P are closed under complement since they all decide in polynomial time.

1. (10) Show that *EQ*DFA is in the class P.

We create TM that accepts input s and TM

* Create that on any input, accepts.
* Create that on any input, run on , if it accepts, accept.
* Run and
  + is constant since it accepts on any input and is equivalent to
* Accepts when both and accept.
* Rejects otherwise.

Then *EQ*DFA is in class P since it decides in polynomial time

1. (10) *ALL*DFA = { <*D*> | *D* is a DFA with L(*D*) = Σ\* }. Show that *ALL*DFA is in the class P.

Mimic *ALL*DFA using TM where

* We receive input s
* We run the DFA using breadth first search.
  + If any accept states are found accept.
  + If any non-accept states are found or we’ve searched the entire tree, reject.

Moving through a graph’s nodes via breadth first search is in polynomial time. Then *ALL*DFA is in class P since it decides in polynomial time.

Problem 2: The Class NP (25 points)

1. (10) Show that the class NP is closed under union and concatenation.

Assume and are their TMs. For

We create a NDTM that accepts input s and

* For input s
* Run and
  + If and are both nondeterministic, they’ll randomly choose the next character and the total time they’ll take to run through the whole string is the number of times equal to string s.
  + If and are both , then is , which is still polynomial time.
* Accepts when both and accept.
* Rejects otherwise.

Then all languages of class NP are closed under union since they all decide in polynomial time.

Assume and are their TMs. For

We create a NDTM that accepts input s and

* For each two substrings of s where and = ||
* Run and
  + If and are both nondeterministic, they’ll randomly choose the next character and the total time they’ll take to run through their respective substrings is the number of times equal to half the length of the original string s.
  + If and are both , then is , which is still polynomial time.
* Accepts when both and accept.
* Rejects otherwise.

Then all languages of class NP are closed under concatenation since they all decide in polynomial time.

1. (15) We say that graphs *G* and *H* are **isomorphic** if the nodes of *G* may be reordered so that it is identical to *H*. Let *ISO* = { <*G*, *H*> | *G* and *H* are isomorphic graphs }.  Show that *ISO* is in the class NP.

Problem 3: NP-Completeness (25 points)

A subset *S* of the nodes of a graph *G* is a **dominating set** if every other node of *G* is adjacent to some node in *S*. Consider the language:

*DOMINATING-SET* = { <*G*, *k*> | *G*has a dominating set with *k* nodes }

Show that*DOMINATING-SET* is NP-complete by reduction from *VERTEX-COVER*.  You can find *VERTEX-COVER*, and the proof that *VERTEX-COVER*is NP-complete, in Chapter 7 of your textbook.

Problem 4: NP-not-so-Completeness (30 points)

You can find the *SUBSET-SUM* problem in our slides, and the proof of its NP-completeness in chapter 7.  Let *UNARY-SSUM* be *SUBSET-SUM* except with all numbers represented in unary.

1. (15) Why is *UNARY-SSUM* **not** NP-complete when *SUBSET-SUM*is?
2. (15) Show that *UNARY-SSUM* is in the class P.

on input :

* Reject if C is not a subset of X.
* Accept if is true
  + Assume
    - Since our numbers are unary
  + Create array Boolean array T of length and set true
  + For each value in X, where set to true if is true.
  + Since our numbers are in unary we are iterating over all the elements of our array T for each element of X.
* Reject otherwise

This algorithm ends up resulting in or the length of our set multiplied by our target number (so the length of our Boolean array)

Problem 5: A Contrasting Path (35 points)

A simple path in a graph is a path that contains no repeated vertices, and (therefore) no cycles. Let *G* be an undirected graph and consider the languages:

*SPATH*= { <*G*, *a*, *b*, *k*> | *G* has a simple path of length ≤ *k* from *a* to *b* }

*LPATH*= { <*G*, *a*, *b*, *k*> | *G* has a simple path of length ≥ *k* from *a* to *b* }

1. (10) Show that*SPATH*is in class P.

on input :

* Reject if a or b is not a node of G or if
* Assume an array T of length k used to track visited nodes.
  + Traverse every node in the tree starting at node a.
    - Push current node into the array.
    - If node is equal to node b then **accept**.
    - If the node is already in the array or the array is full, go back a node and try a different path
  + Otherwise, **reject**

This algorithm ends up resulting in or the number of nodes in our graph multiplied by our max path length

1. (25) Show that *LPATH*is NP-complete.

by the following

on input

Non-deterministically push an unchosen node into an array T of length k,

* if the node has already been traversed or T is full, go back and pick a different unchosen node
* if T forms a path to b, accept.
* Otherwise, if we’ve traversed all possible nodes, reject

on input :

Push each node into array T of length k in the graph starting after node a

* if the node has already been traversed or T is full, go back and pick a different unchosen node.
* if T forms a path to b, accept.
* Otherwise, if we’ve traversed all possible nodes, reject

So because

On input

* If is the number of nodes in G and a Hamilton path exists then
* Output and accept.
* Reject

Therefore

On input

* Run using
* Run or using
* If and accept then accept, otherwise reject.

If accepts graph then G has a Hamilton path of length k and will be accepted by or since by definition of a Hamilton path, it will contain a simple path within it. If or accept then the graph has a simple path that spans the entire graph since k will be the same length as the graph and thus also be a Hamilton path.