| CS/ECE 374, Fall 2018<br>Final: Problem 1 | Gradescope name: Xinrui Ying |
|---|---|

Short questions. No justification is required for your answers.

- Give an asymptotically tight bound for the following recurrence.

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lfloor n/3 \rfloor) + n \quad n \geq 3 \text{ and } T(n) = 1 \quad 1 \leq n \leq 2.$$

$n$

$\frac{n}{2}$ $\frac{n}{3}$ $\frac{5n}{6}$

$\Theta(n)$

$\boxed{\frac{5}{6}}$

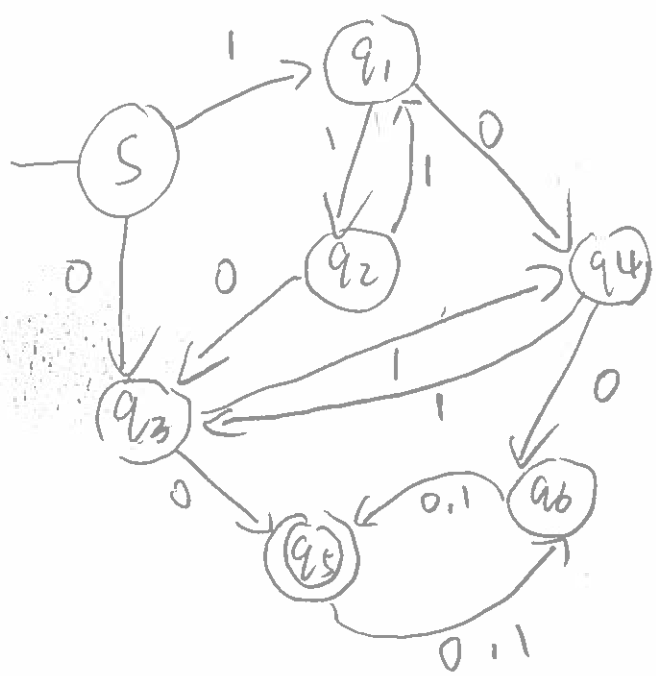
$\frac{1-(\frac{5}{6})^n}{1-\frac{5}{6}} =$

$\log_2 n \times n =$

$n \log_3 n =$ but decreasing $\boxed{n}$.

- Describe a DFA for the language below. Label the states and/or briefly explain their meaning.

$$\{w \in \{0,1\}^* \mid w \text{ has at least two 0's and has even length}\}$$




S: Start state.

$q_1$: No 0 and odd length.

$q_2$: No 0 and even length.

$q_3$: One 0 and odd length.

$q_4$: one 0 and even length

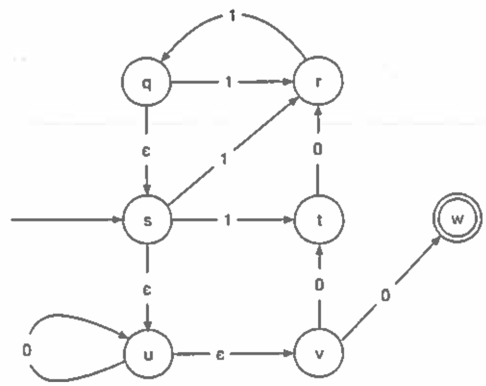
$q_5$: at least 2 0's and even length

$q_6$: at least 2 0's and odd length.

| CS/ECE 374, Fall 2018 | Gradescope name: |
|---|---|
| Final: Problem 2a | Xinrui Ying |

Recall that an NFA $N$ is specified as $(Q, \delta, \Sigma, s, F)$ where $Q$ is a finite set of states, $\Sigma$ is a finite alphabet, $s \in Q$ is the start state, $F \subseteq Q$ is the set of final (or accepting) states and $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q$ is the transition function. Recall that $\delta^*$ extends $\delta$ to strings: $\delta^*(q, w)$ is the set of states reachable in $N$ from state $q$ on input string $w$.

In the NFA shown in the figure below what is $\delta^*(q, 0)$?




It's the states reachable from q and q-ε reached by input 0, also after input, you can still reach state in ε-reached,

That is $\{u, t, w, v\}$

that is $\delta(u,0)=u$ then $\delta(u,\varepsilon)=v$.

$\delta^*(q,0) = \{u, t, w, v\}$.

Problem continues on next page

| CS/ECE 374, Fall 2018 | Gradescope name: |
|---|---|
| **Final: Problem 2b** | Xinrui Ying |

Given an arbitrary NFA $N = (Q, \Sigma, \delta, s, F)$ and an arbitrary state $q \in Q$ and an arbitrary symbol $a \in \Sigma$, describe an efficient algorithm that computes $\delta^*(q, a)$ (in other words the set of all states that can be reached from $q$ on input $a$ which is now interpreted as a string). You may want to first think about how to compute $\delta^*(q, \epsilon)$ (the $\epsilon$-reach of state $q$). Express the running time of your algorithm in terms of $n$ the number of states of $N$ and $m = \sum_{p\in Q}\sum_{b\in\Sigma\cup\{\epsilon\}} |\delta(p, b)|$ which is the natural representation size of the NFA's transition function. Note that faster solutions can earn more points, but incorrect solutions will earn few points, if any.

By doing that, think NFA as a graph G. First, you need to traverse every node (state) in the graph to delete any self-loop on $\varepsilon$, otherwise it goes on and not stop.

Then create a graph G' by deleting all edges that is not $\varepsilon$ on G and do DFS on q to make Gscc: Gscc to find all vertices that can reach by the vertex in $\varepsilon$-reach and store them in a list S for that vertex. We the find vertices that are accessible by q in $\varepsilon$-reach in Gscc. It will be all node in same big vertex or the vertex can reach in Gscc from the vertex contains q. Then we perform $\delta(P, a)$ for $P \in L$ that L store in vertex in previous step in graph G. We will put all vertex in a List K and find the vertex in K in list S to find all vertex that can be reach by $\varepsilon$, put these vertex into the list K if it doesn't exist already.

The graph building, $\delta(p, a)$ $p \in L$ and Gscc will all be linear time. Finding the vertex $\varepsilon$-reach will take $O(n^2)$ in Gscc. So total time will be $O(n^2)$ in this case

| CS/ECE 374, Fall 2018 | Gradescope name: Xinrui Ying |
|---|---|
| **Final: Problem 3** | |

Let $G = (V, E)$ be an undirected graph with edge weights $w : E \to \mathbb{Z}_+$. Suppose the weight of each edge is either 1 or 25. Describe a linear time algorithm to find the MST of $G$.

First, decisn two lists one store edges with size 1 and one store edges with size 25. We build lists to store vertices that are connected and we perform Kruskal's Algorithms on the graph. Since we don't have to sort the edges, we just go through every edges in list of edges with weight 1 first. If edge connects two disconnected compoient, we add them, otherwise we don't. Then, we will do same algorithm on list with edges weight 25. Anytime we have all vertices connected, we can finish. Building list and store is linear time as we just check if it's 25 or 1. And we check edge once each time and we don't need to go back in any situation, so it's also linear time. Therefore, the algorithm is linear time.

| CS/ECE 374, Fall 2018 | Gradescope name: |
|---|---|
| Final: Problem 4 | Xinrui Ying |

Let $w \in \Sigma^*$ be a string. We say that $u_1, u_2, \ldots, u_h$ where each $u_i \in \Sigma^*$ is a valid split of $w$ iff $w = u_1 u_2 \ldots u_h$ (the concatenation of $u_1, u_2, \ldots, u_h$). Given a valid split $u_1, u_2, \ldots, u_h$ of $w$ we define its $\ell_\infty$ measure as $\max_{i=1}^{h} |u_i|$; in other words it is the length of the longest string in the split. For example, for the string ALGORITHM, the split AL · GOR · IT · HM has $\ell_\infty$ of 3, and the split ALGO · RITHM has $\ell_\infty$ of 5.

Given a language $L \subseteq \Sigma^*$ a string $w$ is in $L^*$ iff there is a valid split $u_1, u_2, \ldots, u_h$ of $w$ such that each $u_i \in L$; we call such a split an $L$-valid split of $w$. Assume you have access to a subroutine IsStringInL($x$) which outputs whether the input string $x$ is in $L$ or not. To evaluate the running time of your solution you can assume that each call to IsStringInL() takes constant time.

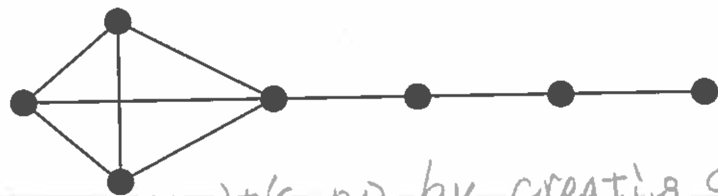
Describe an efficient algorithm that given a string $w$ and access to a language $L$ via IsStringInL($x$) outputs the minimum $\ell_\infty$ measure of a valid split if one exists. Your algorithm should output $\infty$ if there is no valid split. Note that a slower correct algorithm will earn more points than a faster but incorrect one.

{
0 if n > |W|
0 if IsStringInL(w[a,b], n) && Func(w[a, b+1], n+1) == 0 && Func(w[b+1, b+1], n+1) == 0
~~0 if !IsStringInL(w[a,b], n) && Func(w[a, b+1], n+1) == 0~~
Max(length(w[a,b]), Func(w[a,b+1], n+1), Func(W[b+1, b+1], n+1)) if IsStringInL(w[a,b], n) && (Func(w[a, b+1], n+1 > 0) or Func(w[b+1, b+1], n+1) > 0)
Func(w[a, b+1], n+1) if !IsStringInL(w[a,b], n)

Decide a function Func ( x[a,b] , n ) that take a string and the index of current last bit. If x[a,b] ∈ L, then we can either take x[a,b] as u; or don't take it and do Func on x[a,b+1] or x[b+1, b+1] to take the maximum. If x[a,b] ∉ L we can only keep doing Func on x[a, b+1]. If it's invaild, then we will return 0. If all Func situation it's 0, then No matter what current string length is, we return 0 as there will be no vaild split. We will take a dp algorithm with size 2d array with 2x|W|. First one will be value of Func on w[a,b] and second will be will on Func on w[b+1, b+1]. If we have 0 in the first cell we will output ∞, as no vaild spilt, else we output value in first cell. Each cell will depend on b-a and arr[b+1, b+1] and arr[a, b+1], the running time is O(n).

| CS/ECE 374, Fall 2018 | Gradescope name: |
| --- | --- |
| Final: Problem 5 | Xinrui Ying |

A *kite* of size $k$ is a complete graph (clique) on $k$ vertices plus a tail of $k$ vertices. See figure for a kite of size 4. The KITE problem is the following: given an undirected graph $G = (V, E)$ and an integer $k$ does $G$ contain a kite of size $k$ as a subgraph? Prove that KITE is NP-Complete.



First, we can prove it's np by creating cetificate and cetifier for it.

Cetificate: $S \subseteq G$

Cetifier: $S$ is a kite of size $k$.

We can reduce clique problem to kite. (to prove it's np-hard)

For a graph $G$, we add a tail of $k$ vertice like the example above to every vetices in graph $G$ to make a graph $G'$. We say $G$ has a clique subgraph of size $k$ if only if $G'$ has a KIte of size $k$ in the subgraph.

$\Longrightarrow$ Suppose $G$ has a clique subgraph of size $k$, then Take the corresponding subgraph in $G'$ and use any tail we add to any of the vertex in this subgraph, we have a kite of size $k$

$\Longleftarrow$ Suppose $G'$ has a subgraph of kite of size $k$, delete the tail of the subgraph we will have a clique set of size $k$. Take the vertex of the remaining and find the match in graph $G$. so graph $G$ has a clique set of size $k$. (Since a tail is a path and it will not contain a clique set of size $k$ ($k \geq 3$)

So we proved that kite is np-complete for np-hard and kite $\in$ NP

| CS/ECE 374, Fall 2018<br>**Final: Problem 6a** | Gradescope name: Xinrui Ying |
|---|---|

This problem (two parts combined) is worth 15 points.

Prove that the following language is undecidable.

$$L = \{\langle M\rangle \mid M \text{ accepts at least one string}\}.$$

We will create a DecideL be a decider for L.
We will use halt promblem to prove that it's not decidable.

DecideHalt (<M, w>):
Encode the following Turing Machine M':

```
M'(x):
   Run M on input w:
     If (M(w) == true):
        return true
   Else:
        false
```

If DecideL (<M'>):
  return true
Else
  return false.

⟹ Suppose M halts on input w,
then M' accepts all strings
M' accepts at least one string.
Then DecideL accepts the encoding <M'>
Then DecideHalt correctly accepts the encoding <M, w>

⟸ Suppose M don't halt on input w
the M' diverge on all strings
M' accepts no string
then DecideL rejects the encoding <M'>
DecideHalt correctly rejects the encoding <M, w>

Problem continues on next page

In both case, decideHalt is correct. But that's impossible since decideHalt is undecidable. So L is undecidable

| CS/ECE 374, Fall 2018<br>Final: Problem 6b | Gradescope name: Xinrui Ying |
|---|---|

Prove that $L$ is recursively enumerable. What this means is that you should describe a TM/program $M'$ (at a high-level) that takes as its input the description/code of an arbitrary TM/program $M$ and halts and says YES if $M$ accepts at least one string; if $M$ does not accept any string your program $M'$ is allowed to run forever but if it halts it should correcly say NO. *Hint:* Use dovetailing.

M'(x):
run M on input w.
If M accept w:
~~halt~~
say no
Else
~~loop forever.~~
say yes

| CS/ECE 374, Fall 2018 | Gradescope name: |
| --- | --- |
| **Final: Problem 7** | Xinrui Ying |

Let $S = \{x_1, x_2, \ldots, x_n\}$ be a set of $n$ points on the real line $\mathbb{R}$. For simplicity assume that the points are in sorted order, that is $x_1 < x_2 < \ldots < x_n$. We say that an interval $[a, b] \subseteq \mathbb{R}$ *covers* a point $x_i$ if $x_i \in [a, b]$. A unit interval is an interval of the form $[a, a+1]$.

Describe an efficient algorithm to find the *smallest* number of unit intervals that cover every point in $S$. In other words, every point in $S$ must belong to some unit interval chosen by your algorithm, and the set of unit intervals chosen must be smallest possible. Note that you get to decide where to place the unit intervals. See figure below for an example of set of points and a feasible solution; although this particular solution does not have any overlapping intervals, they are allowed. Remember that fast but incorrect algorithms will earn few, if any points.

Greedy algorithm: 1. look at the current Point, find the intervals that ends last but Still cover the point, take it.
2. find the largest point that covered by the interval chosen before, and then set current point to the next point of this largest point, end if there is none.
3. do the above steps recursively.

Since we check the chosen interval two-times, first by the step 1 and second by step 2. So the running time will be 2 times the number of chosen intervals. That in worst case is $O(2m) = O(m)$ Where $m$ is the number of intervals. (Linear Time)

## List of some useful NP-Complete Problems/Languages

- **SAT** $\{\varphi \mid \varphi$ is a boolean formula in CNF form and is satisfiable$\}$
- **3SAT** $\{\varphi \mid \varphi$ is a boolean formula in CNF form with exactly 3 literals per clause and is satisfiable$\}$
- **Circuit-SAT** $\{C \mid C$ is a boolean circuit such that there is a setting of values to the inputs to $C$ that make it evaluate to TRUE$\}$
- **Independent Set** $\{\langle G, k\rangle \mid$ Graph $G = (V, E)$ has a subset of vertices $V' \subseteq V$ of size at least $k$ such that no two vertices in $V'$ are connected by an edge$\}$
- **Vertex Cover** $\{\langle G, k\rangle \mid$ Graph $G = (V, E)$ has a subset of vertices $V' \subseteq V$ of size at most $k$ such that every edge in $E$ has at least one of its endpoints in $V'\}$
- **Clique** $\{\langle G, k\rangle \mid$ Graph $G$ has a complete subgraph of size at least $k\}$
- **3Color** $\{\langle G\rangle \mid$ The vertices of graph $G$ can be colored with 3 colors so that no two adjacent vertices share a color$\}$
- **Coloring** $\{\langle G, k\rangle \mid$ The vertices of graph $G$ can be colored with $k$ colors so that no two adjacent vertices share a color$\}$
- **Hamiltonian Cycle** $\{\langle G\rangle \mid$ Directed graph $G$ contains a directed cycle visiting each vertex exactly once$\}$
- **Undir Hamiltonian Cycle** $\{\langle G\rangle \mid$ Undirected graph $G$ contains a cycle visiting each vertex exactly once$\}$
- **Hamiltonian Path** $\{\langle G\rangle \mid$ Directed graph $G$ contains a directed path visiting each vertex exactly once$\}$
- **Undir Hamiltonian Path** $\{\langle G\rangle \mid$ Undirected graph $G$ contains a path visiting each vertex exactly once$\}$

This page for extra work.

This page for extra work.

This page for extra work.

This page for extra work.

This page for extra work.

MaxtillNow = max ( Maxtillnow, length of (X[a,b]), Func( X[a,b+1], n+1) Func[ X[b+1,b+1], n+1) )

( , ( )