Matthew Martinez
Miguel Martinez
Joseph Reyes
Ernesto Valdez

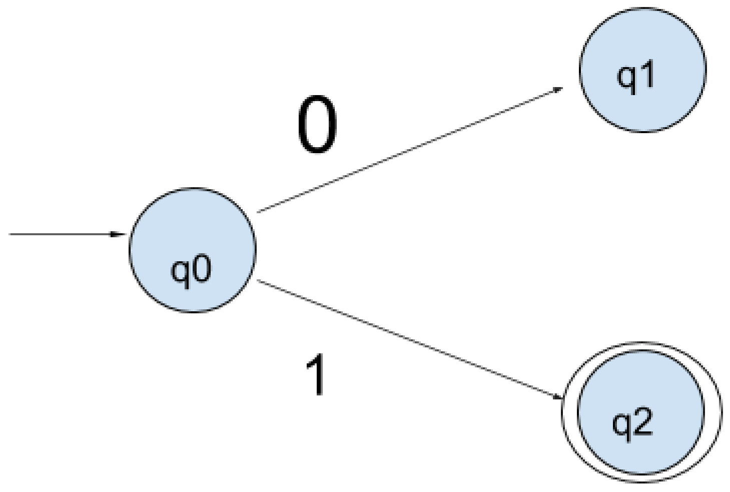
CSCI 4341: Games, Puzzles, & Computation
Class notes for 1/23/17 & 1/25/17
TEAM C

Intro: This weeks focus was on time complexity and turing machines as well as the games played in the lab. A few links are including for more in depth information on certain topics. Some information was taken from Michael Sipser's *Introduction to the Theory of Computation 3rd Edition*.

Turing Machine (T.M.): A computer with infinite memory. Can be deterministic and nondeterministic. Turing machines are important and useful because it's the theory behind what a computer is and what it can and can't do. Some models of a turing machine are

- Deterministic: A single path.
- Nondeterministic: All paths at the same time.

Ex. A Simple Deterministic Turing Machine. In a deterministic T.M. only one path can be taken at a time. In this example there are only two paths. If a 0 is fed to the T.M. change state from q_0, the start state, to state q_2 and reject. If a 1 is fed to the T.M. change state from q_0 to q_2 and **accept**, hence the **ring around q_2**.



M1.

A turing machine can be defined formally as a 5-tuple by writing M1 = (Q,Σ,δ,q1, F), where

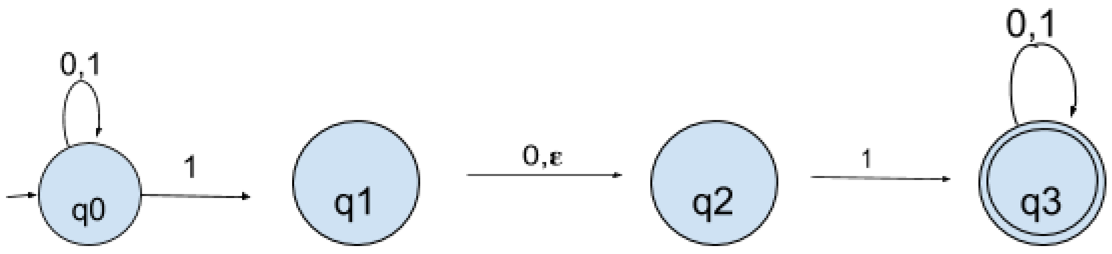
1. Q is the finite set called **states**.
2. Σ is a finite set called the **alphabet.**
3. δ: Q x $\Sigma \longrightarrow$ Q is the **transition function.**
4. $q_0 \in Q$ is the **start state**, and
5. F $\subseteq$ Q is the **set of accept states.**

In this case, M1 is described formally as:
1. Q = $\{q_0, q_1, q_2\}$
2. Σ = {0,1}
3. δ is describes as

	0	1
q_0	q_1	q_2

4. q_0 is the start state
5. F = $\{q_2\}$

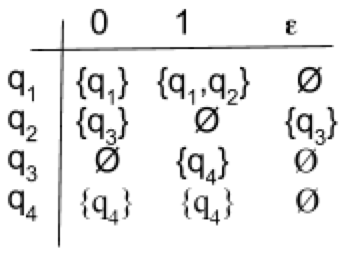
Ex. A Nondeterministic Turing Machine. The difference between a deterministic T.M. and a nondeterministic T.M. is the transition function. In a deterministic T.M., the transition function takes a state and an input symbol and produces the next state. In a nondeterministic T.M., the transition f**ε**nction takes a state and an input symbol ***or the empty string,*** , and produces *the set of possible next states*.



M2. ***ε** is the empty string.*

A nondeterministic turing machine can be defined formally as M2 = M1 = (Q,Σ,δ,q1, F), where:
1. Q is the finite set of **states.**
2. Σ is a finite **alphabet.**
3. δ: Q x $\Sigma_\varepsilon \longrightarrow$ P(Q) is the **transition function.**
4. $q_0 \in Q$ is the **start state**, and
5. F $\subseteq$ Q is the **set of accept states.**

M2 can be described formally as:
1. $Q = \{q_0, q_1, q_2, q_3\}$
2. $\Sigma = \{0,1\}$
3. δ is given as



	0	1	ε
q_1	$\{q_1\}$	$\{q_1,q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

4. Q_1 is the start state
5. $F = \{q_4\}$

Complexity: Time complexity of an algorithm is the number of steps taken to decide based on the input. Complexity can be based on space as well as time.

Asymptotic Analysis: Highest term of a polynomial term that describes the running time based on the size of the input.

Asymptotic analysis is important because knowing the complexity of algorithms allows you to answer questions such as:
- How long will a program run on an input?
- How much space will it take?
- Is the problem solvable?

Orders of Growth:

O(*1*)	Constant
O(*log n*)	Logarithmic
O(*n*)	Linear
O(*n Log n*)	"n log n"
O(n^2)	Quadratic
O(n^3)	Cubic
$n^{O(1)}$	Polynomial
$2^{O(n)}$	Exponential

More information on asymptotic analysis:
http://www.cs.cornell.edu/courses/cs312/2004fa/lectures/lecture16.htm

Notation:

- $\mathbb{N}$ is the set of natural numbers. ex) 0,1,2,3,4... or 1,2,3,4,...
- $\mathbb{R}^+$ is the set of real positive numbers. ex) 0.5, 2, e, 3, π, 4, 5,...
- ∃ means "there exists".
- ∀ means "for all".
- ϵ means "an element of".
- s.t. is the abbreviation for "such that".

Def'n: Let f and g be functions f,g: $\mathbb{N} \to \mathbb{R}^+$. We say that $f(n) \in O(g(n))$ if ∃ c, $n_o >$ s.t. $0 \leq f(n) \leq c * g(n)$ ∀ $n \geq n_o$.

Ex.1) $f(n) = 5n^3 + 2n^2 + 22n + 6$ $\qquad f(n) = O(n^3)$, $g(n) = n^3$

Let c = 6 and $n_o = 10$

$f(n) \leq 6n^3$ ∀ $n \geq 10$

$5{,}428 \leq 6{,}000$

Ex.2) $f(n) = 5n^3 + 2n^2 + 22n + 6 \leq 5n^3 + 2n^3 + 22n^3 + 6n^3$

$f(n) = 35n^3$

$f(n) = O(n^3)$ where c = 35 and $n_o = 1$

Little-o Notation:

Def'n: Let f and g be functions $f, g \in \mathbb{N} \to \mathbb{R}^+$. We say that $f(n) \in o(g(n)) < \lim_{n \to \infty} f(n) / g(n) = 0$.

Ex.) $f(n) = 3n^3$

$g(n) = n^4$

$f(n) / g(n) = 3n^3 / n^4 = 1 / n \Rightarrow 0$

Def'n: Let t:$\mathbb{N} \to \mathbb{R}^+$ be a function.

Ex.1)
```
for(int i = 0; i < n; i++)
{
      add += arr[i];
}
```

time complexity = O(n)

Ex.2)
```
for(int i = 0; i < n; i++)
{
      for(int j = 0; j < n; j++)
      {
            add += arr[i];
      }
}
```

time complexity = $O(n^2)$

<u>Polynomial Time</u>: All reasonable deterministic models of computation can be found in a polynomial factor and are polynomially equivalent. This allows us to develop a theory and look at the complexity of problems not specific to a single model of computation.

<u>Complexity Classes</u>:

- P: Decision problems that can be solved on a DTM in polynomial time.
- NP: Decision problems that can be solved on a NTM in polynomial time.

<u>Time:</u> Define the time complexity class TIME(t(n)), to be the set of all languages decidable by a o(t(n)) time TM.

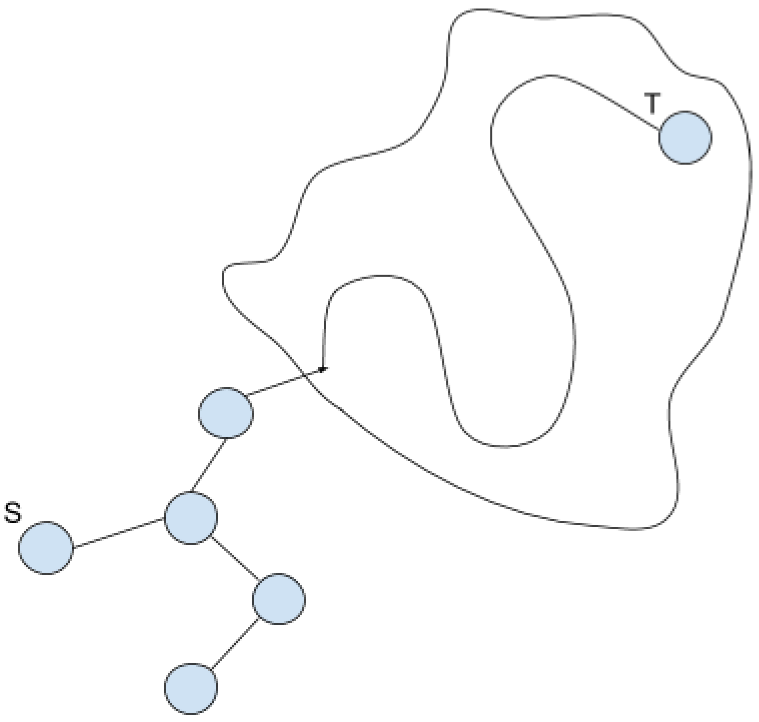
Decidable: a solution (yes/no) can always be found in a finite amount of time

<u>The class P</u>:

Def'n: P is the class of languages decidable in polynomial time on a deterministic single-tape TM.

$$P = \bigcup_k \mathrm{TIME}(n^k)$$

Ex.) PATH = {<G, s, t> : G is a directed graph with a path from s to t}



PATH ε TIME(n^2)
PATH ε TIME(nlogn) ε TIME(n^2) ε …

P ε O(n) <= O(n^2) <= …
P ε O ε (n^2)

<u>The class NP</u>:

<u>NTIME:</u> NTIME(t(n)) = {L: L is a language decided by a O(t(n)) time nondeterministic TM.}

NP = U_k NTIME(n^k)

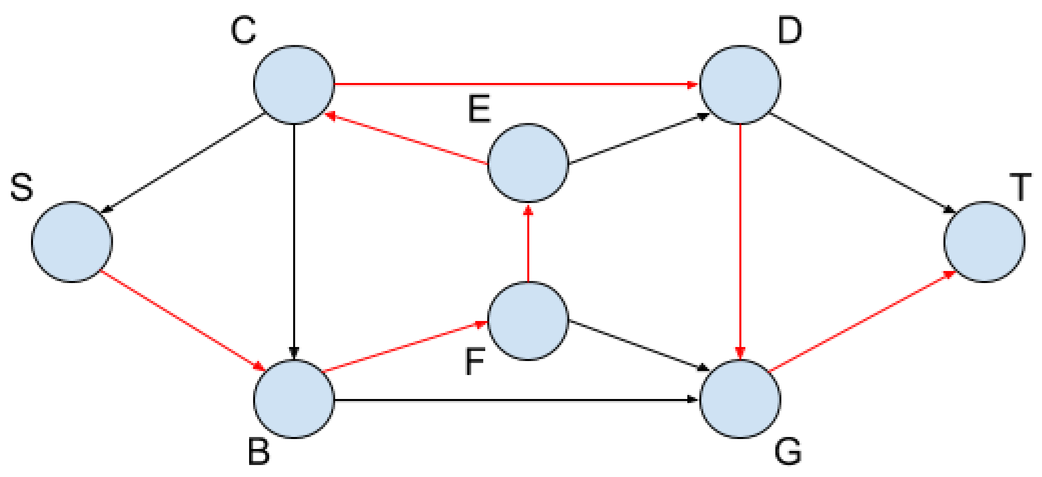
*NP is the class of languages that have polynomial time verifiers.

<u>Polynomial Time Verifier</u>:

Def'n: A verifier for a language A is an algorithm V where A = {w: V accepts <w, c> for some string c}. c is the certificate.

Hamiltonian Path:

HP = {<*G*, *s*, *t*> | *G* is a directed graph with a Hamiltonian path from *s* to *t*}



Path = <*s*, *b*, *f*, *e*, *c*, *d*, *g*, *t*>

1. Does this visit all the nodes? Yes // $O(n)$
2. Does this form a valid path? Yes // $O(n^2)$
3. Does the path start at *s* and end at *t*? Yes // $O(1)$

$P \subseteq NP$

Verifying answer is equivalent to U_k NTIME(n^k).

Clique ={ < <G,K> , C> > : G is an undirected graph with a **K-clique** }

Thm. - Clique ε NP
Verifier ⇒ < <G,K> ,C > - C is a list of vertices

1) Is C ε V(G) ?, all vertices in C are in G
2) All connected to each other? $O(|C|^2)$
3) If both 1 ,2 true

-Solve Problem NonDeterministically

1) Select K vertices from G
2) Do they form a Clique?

<u>Subset-sum</u> ={ <S,T> : S = {x_1 ….. X_k} and for some {y_1 ….. y_L} <= S such that Σy_i= T

S = {1, -3, 4, 8, -5, 7}
T=10
⇒ yes {8,7,-5}

Thm. - Subset-sum ε NP
Verifier → < <S, T>, C > - lists of number is subset

1) Check that C <= S
2) Check that ΣC_i= T
3) If 1 and 2 , yes ; no

<u>CoNP</u>

- Satisfiability
  - Boolean Variables X=0/1
  - Boolean Operators ⇒ AND (^), OR(⋁), NOT(¬)
  - Boolean Formula θ = (x ⋁ z) ^ (y ⋁ ¬z) ^ (¬z ⋁ ¬y)

  Satisfiable assignments such that θ ⇒ 1

Thm. - SAT = { < θ > | θ is a satisfiable boolean formula} is NP complete.

<u>P and NP relation:</u>

<u>P versus NP question:</u> asks whether every problem whose solution can be quickly verified by a computer can also be quickly solved by a computer. An answer to the P = NP question would determine whether problems that can be verified in polynomial time, like the subset-sum problem, can also be solved in polynomial time. Aside from being an important problem in computational theory, a proof either way would have profound implications for mathematics, cryptography, algorithm research, artificial intelligence, game theory, multimedia processing, philosophy, economics and many other fields.

Papers claiming to settle P versus NP:
https://www.win.tue.nl/~gwoegi/P-versus-NP.htm

Lab 1

In this week's lab, we played Hex, Dots and Boxes, and 3D tic-tac-toe. The goal was to figure out whether first or second player had a winning advantage. IN Hex, first player has an initial winning advantage, but it can be easily trumped if second player can make a good block, hypothetically creating a fair game. In Dots and Boxes, the first player also has an initial advantage, however, given the right moves and player experience, the second player can definitely turn it around. In all variations of 3D tic-tac-toe with equal numbers of levels and columns, the first player has an advantage throughout the entire game. He/She can always either win or tie, never lose, that is, if they know the strategy to do so.