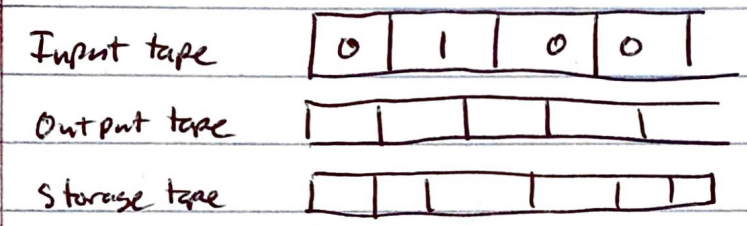
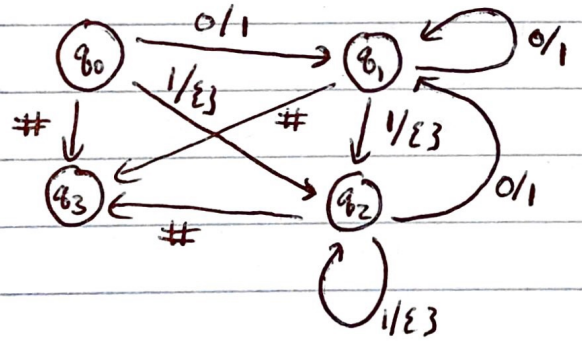
Complexity Theory

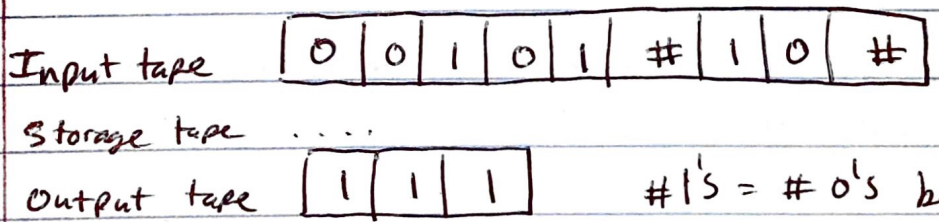
Defn: A Turing Machine includes the following

1. Q: a finite set of states $\quad Q = \{q_0, \ldots, q_m\}$ (q_0: initial, q_m: final)
2. Σ "alphabet" - a set of symbols
3. f "transition function" $\quad f: Q \times \Sigma \to Q \times \Sigma \cup \{e\}$ (e: empty symbol)
4. Tape : Input, Storage, & Output



Example: Suppose $\Sigma = \{0, 1, \#\}$





#1's = # 0's before first #

"Any reasonable model of computation is equivalent to a turing machine"

Time Complexity n = size of input

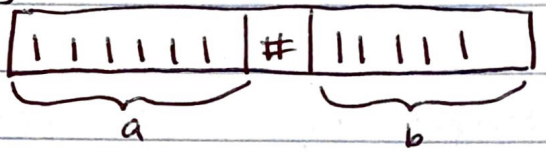
$T(n)$ = # of steps/transitions the turing machine makes

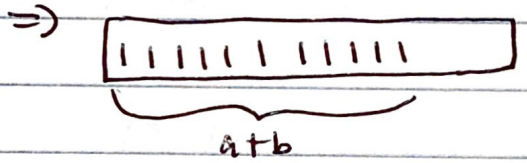
Space complexity

$S(n)$ = amount of storage required

Example: important to consider your input!!!

Adding: $a+b$





"unary encoding"

$T_1(n) = c_1 n$

length $a+b$.

versus:

a base 2 b base 2

a+b base 2

"binary encoding"

$T_2(n) = c_2 n$

only $\log(a) + \log(b)$ length

★ both linear time algorithms, but T_2 is exponentially better than T_1.

Time Complexity n = size of input

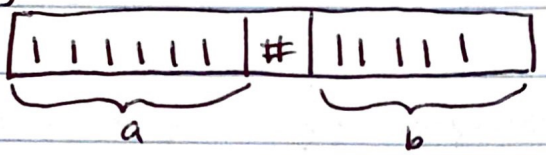
$T(n)$ = # of steps/transitions the turing machine makes

Space complexity

$S(n)$ = amount of storage required

Example: important to consider your input!!!

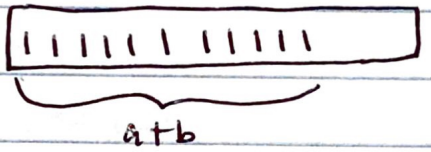
Adding: $a+b$



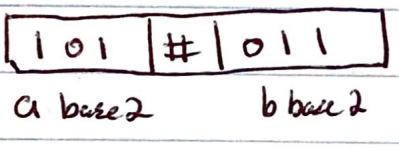
"unary encoding"

$T_1(n) = c_1 n$

length $a+b$.



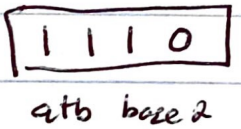
versus:



"binary encoding"

$T_2(n) = c_2 n$

only $\log(a) + \log(b)$ length



★ both linear time algorithms, but T_2 is exponentially better than T_1.

Defn: A problem is a set of strings on Σ

An instance is a particular string from the problem

Defn: The class of polynomial time problems P is the set of problems for which there exists a turing machine with polynomial time complexity.

Reduction: Problem B reduces to problem A if for all instances of B there exists a map

$R: B \to A$, $R(b) \in A$, s.t. R is polytime.

Defn: A and B are polynomially equivalent if A reduces to B and B reduces to A.

Example: Vertex Cover

- Optimization asks for the minimum vertex cover
- Feasibility asks for a vertex cover of size $\leq k$

Stable Set

- optimization asks for the maximum stable set
- feasibility asks for stable set of size $\geq n-k$

All of these are polynomially equivalent

Defn: A non-deterministic polynomial time (NP) turing machine has a map

$$f: Q \times \Sigma \to 2^Q \times \Sigma \cup \{e\}$$

it can carry out parallel computations.

Alternative Defn: A problem is NP if there exists a polynomial time algorithm to check a given certificate.

Clearly $P \subseteq NP$. Probably $P \neq NP$.

Defn: A problem is a set of strings on Σ

An instance is a particular string from the problem

Defn: The class of polynomial time problems P is the set of problems for which there exists a turing machine with polynomial time complexity.

Reduction: Problem B reduces to problem A if for all instances of B there exists a map

$R: B \to A$, $R(b) \in A$, s.t. R is polytime.

Defn: A and B are polynomially equivalent if A reduces to B and B reduces to A.

Example: Vertex Cover

- Optimization asks for the minimum vertex cover
- Feasibility asks for a vertex cover of size $\leq k$

Stable Set

- optimization asks for the maximum stable set
- feasibility asks for stable set of size $\geq n-k$

All of these are polynomially equivalent

Defn: A non-deterministic polynomial time (NP) turing machine has a map

$$f: Q \times \Sigma \to 2^Q \times \Sigma \cup \{e\}$$

it can carry out parallel computations.

Alternative Defn: A problem is NP if there exists a polynomial time algorithm to check a given certificate.

Clearly $P \subseteq NP$. Probably $P \neq NP$.