Sipser chapter 7 Time complexity

Definition 7.1 Let M be a DTM which halts on all inputs (a decider). The running time or time complexity of M is the function $f: \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that M takes on any input of length n.

Definition 7.2 Let $f, g: \mathbb{N} \to \mathbb{R}^+$ be functions. We say that $f(n) \in O(g(n))$ if there exist $c, n_0 \in \mathbb{Z}_+$ such that $\forall n \geq n_0: f(n) \leq c \cdot g(n)$

Definition 7.5 Let $f, g \in \mathbb{N} \to \mathbb{R}^+$. We say that $f(n) \in o(g(n))$ if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

That is, for every $c > 0$ there exist $n_0 = n_0(c)$ s.t. $\forall n \geq n_0 \quad f(n) < c\, g(n)$

Definition 7.7 Let $t: \mathbb{N} \to \mathbb{R}^+$ be a function. The time complexity class $\mathrm{TIME}(t(n))$ is the collection of all languages that are decidable in time $O(t(n))$ by a DTM

Important: We are only dealing with decision problems, that is

given $w \in \Sigma^*$ does $w \in L$?

(where L is a given language)

Remark

If a problem has both an optimization and a decision version, then the complexity of them two are closely related.

SpT_K: Given a connected edge-weighted graph $G=(V,E,w)$ and a natural number K

Does G have a spanning tree T s.t $w(T) \leq K$?

MST: Given a connected edge-weighted graph $G=(V,E,w)$

Find a minimum weight spanning tree T^* of G

$(w(T^*) \leq w(T) \ \forall$ spanning tree $T)$

Given $G=(V,E,w)$ and $K \in \mathbb{N}$ we can decide whether

$\langle G, K\rangle \in SpT_K$ by solving MST for $\langle G\rangle$

and compare $w(T^*)$ to K

$\langle G, K\rangle \in SpT_K \iff w(T^*) \leq K$

where T^* is a MST of G wrt w.

Conversely, if A is an algorithm for SpT_{K}, then we can solve MST for G as follows:

$K=1$ $\quad SpT_1$. ÷

$K=2$ $\quad SpT_2$ ÷

⋮

$K=2^{r-1}$ $\quad SpT_{2^{r-1}}$ ÷

$K=2^r$ $\quad SpT_{2^r}$ +

→ find best K^* via binary search on the interval $[2^{r-1}, 2^r]$

So making $O(\log K^*)$ calls to A we can solve the optimization version

This is a general result:

If A is an algorithm for solving the decision version of a problem L with parameter K, then we can solve the optimization version for L via $O(\log K^*)$ calls to A where K^* is the value of the optimal solution to the optimization version

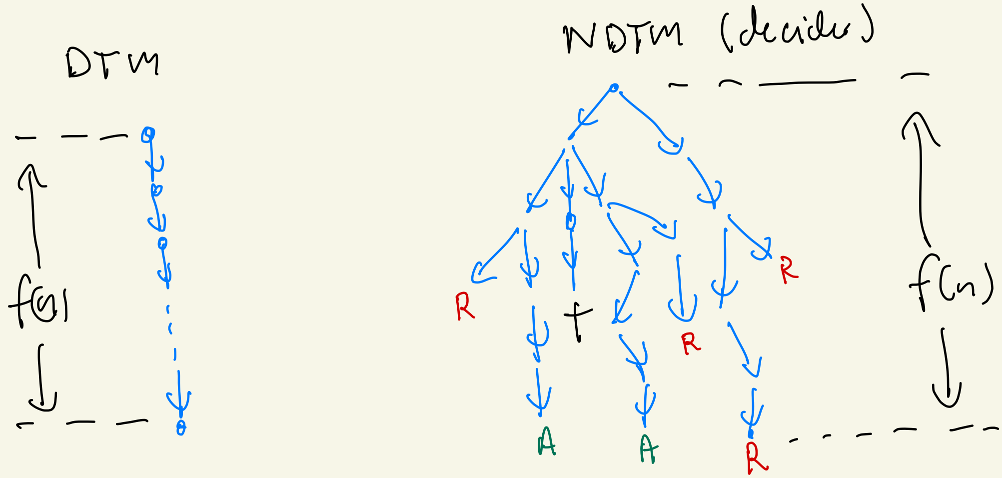
Recall from Turing machine theory:

Theorem 7.8

Let $t(n)$ be any function with $t(n) \geq n$.
Then every $t(n)$-time multitape TM M has an equivalent $O(t^2(n))$-time singletape TM

Definition 7.9

Let M be a NDTM which is a decider.
The running time of M is the function $f: \mathbb{N} \to \mathbb{N}$,
where $f(n)$ is the maximum number of steps that M uses on any branch of its computation on any input of length n



Recall from Chapter 3:

Theorem 7.11 Let $t(n)$ be a function with $t(n) \geq n$

Then every $t(n)$-time NDTM has an equivalent $2^{O(t(n))}$-time single tape TM

ALL resonable deterministic computational models are polynomially equivalent.

That is, any of them can simulate each of the others with only a polynomial increase in running time

We will focus on aspects of time-complexity that are unaffected by polynomial differences in running time

Our aim is to present fundamental properties of computation, rather than properties of Turing machines

Definition 7.12

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

i.e $\mathbf{P}$ is the class of languages which are decidable in polynomial time

Notes

1. **P** is invariant for all models of computation that are polynomially equivalent to the deterministic single tape TM

2. **P** ~ class of decision problems that are realistically solvable on a computer (for all instances)

Encodings of problems (denoted <...>)

We avoid unary encodings $10 \sim 1111111111$ as this is exponentially larger than any coding such as base k for any $k \geq 2$

(e.g. 1000 uses only 10 bits in base 2)

Coding graphs $\langle G \rangle$

1. list of vertices and edges + possible costs in binary

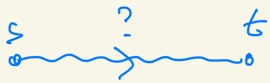
2. Adjacency matrix $n \times n$ matrix $n = |V(G)|$ with entry $(i,j) = \begin{cases} 1 & \text{if } ij \in E(G) \\ 0 & \text{otherwise} \end{cases}$

possibly with cost c_{ij} in binary instead of 1

Examples of problems in **P**

1. SpT_K from previous slide

2. $PATH = \{ \langle G, s, t \rangle \mid G$ is a digraph, $s, t \in V(G)$ and $\exists (s,t)$-path in $G \}$



$PATH \in$ **P**

3. $\text{MEMBERSHIP OF CFL} = \{ \langle G \rangle \langle w \rangle \mid G$ CFG and $w \in L(G) \}$

Given $w \in \Sigma^*$ and G a CFG in Chomsky normal form we know $S \stackrel{*}{\Rightarrow} w \iff$ there is a derivation with $2|w|-1$ steps

Method 1: try all derivations of length $2|w|-1$.
Not polynomial in $|w|$ as we have up to $|R(G)|$ possible rules in each step
(if we have at least 2 in each step, then it takes
$\Omega(2^{|w|-1})$ steps

$S \Rightarrow A_1 A_2 \Rightarrow A_1 A_2' A_3' \Rightarrow A_1 A_2 A_3'' A_4''$

Method 2: Dynamic programming

If $w = \varepsilon$ accept iff $S \to \varepsilon$ is in R

$|w| > 0$: $w = a_1 a_2 \cdots a_n$ $n = |w|$ $a_i \in \Sigma$

construct $n \times n$ matrix T where we will have $T_{ij} = \{A \mid A \Rightarrow a_i a_{i+1} \cdots a_j\}$ after the computation

A variable in G

initially $T_{ii} = \{A \mid A \to a_i \in R\}$

idea: If $A \to BC$ and $B \Rightarrow a_i \cdots a_j$

$C \Rightarrow a_{j+1} \cdots a_p$

then add A to T_{ip}

Solution:

For $i \leftarrow 1$ to n
 For $j \leftarrow i+1$ to n
  $T_{ij} \leftarrow \emptyset$

For $i \leftarrow 1$ to n
 $T_{ii} \leftarrow \{A \mid A \to a_i \in R\}$

For $r \leftarrow 1$ to $n-1$
 For $i \leftarrow 1$ to $n-r$
  For $k \leftarrow i$ to $i+r-1$
   For each rule $A \to BC$:
    if $B \in T_{i,k}$ and $C \in T_{k+1,i+r}$
    then $T_{i,i+r} \leftarrow T_{i,i+r} \cup \{A\}$

If $S \in T_{1,n}$ accept; else reject

$|w| = 5$

	1	2	3	4	5
1	X				
2		X		$\emptyset$	
3			X		
4				X	
5					X

$r = 1$

	1	2	3	4	5
1	X	X			
2		X	X		
3			X	X	
4				X	X
5					X

$r = 2$

	1	2	3	4	5
1	X	X	X		
2		X	X	X	
3			X	X	X
4				X	X
5					X

$r = 3$

	1	2	3	4	5
1	X	X	X	X	
2		X	X	X	X
3			X	X	X
4				X	X
5					X

$r = 4$

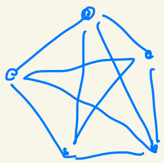
	1	2	3	4	5
1	X	X	X	X	X
2		X	X	X	X
3			X	X	X
4				X	X
5					X

More difficult problems:

- HAMPATH $= \{\langle G,s,t\rangle \mid G$ is a digraph, $s,t \in V(G)$ and G has an (s,t)-path P s.t. $V(P)=V(G)\}$

(very) difficult, but easy if we can guess:
Given an ordering $s=v_1, v_2, \dots, v_n=t$ it is easy to check whether v_iv_{i+1} is an arc for $i=1,2\dots n-1$

- CLIQUE $= \{\langle G,k\rangle \mid G$ is a graph that has a complete subgraph with k vertices$\}$



clique of size 5

Easy if we can guess vertices $v_{i_1}, v_{i_2}, \dots, v_{i_k}$
just check that $v_{i_p}v_{i_q} \in E(G)$ $\forall p\neq q$ $p,q\in[1,k]$

such a set is called a certificate for $\langle G,k\rangle \in$ CLIQUE
"
(proof)

Remark on CLIQUE:

If we just have G and k then (essentially) no better method is known than trying all $\binom{n}{k} \in O(n^k)$ possible subsets of size k

Remark on HAMPATH:

Here we could try all $(n-2)!$ permutations of $V(G) - \{s,t\}$

Definition 7.18

A verifier for a language L is an algorithm A_L such that $L = \{w \mid \exists$ string c s.t. A_L accepts input $\langle w,c \rangle\}$

The running time of A_L is measured in terms of $n = |w|$

A_L is a polynomial verifier if it has running time $O(n^k)$

The string $c = c(w)$ is a certificate for $w \in L$

Note that $|c(w)| \le$ running time of A_L

Definition 7.19 **NP** = { L | L has a polynomial verifier }

NDTM for HAMPATH:

1. Guess n numbers $i_1, i_2, \ldots, i_n$ with $i_j \in [1, n]$
2. Check for repetitions ($i_p = i_q$, $p \neq q$)
   If a repetition is found reject
3. Check whether $s = v_{i_1}$ and $t = v_{i_n}$
   if not reject
4. Check whether $v_{i_p} v_{i_{p+1}}$ is an arc for
   $p = 1, 2, \ldots n-1$
   If yes accept
   Else reject

Theorem 7.20

$L \in NP \iff L$ is decided by a NDTM

Proof: let $L \in \mathbf{NP}$ and let A_L be a verifier for L s.t A_L runs in time at most dn^k for some constant d on input of length n.

NDTM: on input w, $n = |w|$

1. Select non-deterministically a string c s.t $|c| \le dn^k$
2. Run A_L on $\langle w, c\rangle$
3. Accept if A_L accepts else reject

Conversely suppose L is decided by a NDTM M

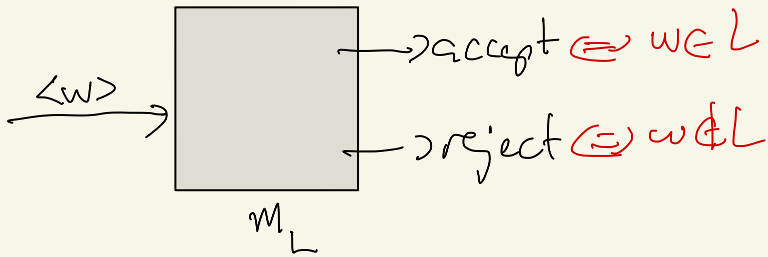
construct A_M: on input $\langle w, c\rangle$

1. Simulate M on $\langle w\rangle$ using $\langle c\rangle$ to guide which branch to take (as in proof of Thm 3.16)
2. If this branch of M's computation results in M accepting w, then A_M accepts $\langle w, c\rangle$ otherwise A_M rejects $\langle w, c\rangle$

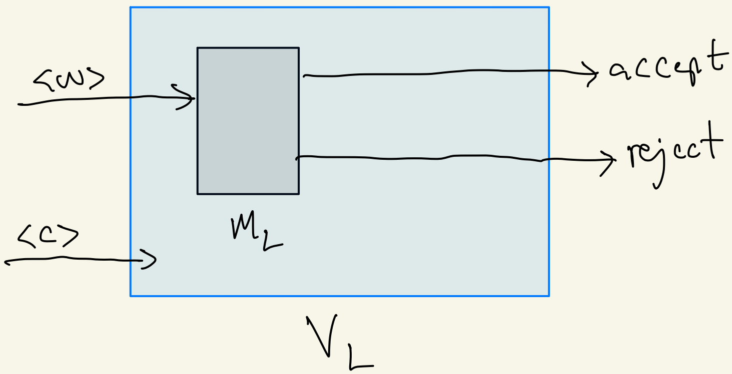
Now $\exists c$ s.t. A_M accepts $\langle w, c\rangle \iff M$ accepts $\langle w\rangle$

$P \subseteq NP$

- let $L \in P$ and let M_L be a polynomial decider for M_L



- Build V_L as follows



V_L is a verifier for L since it will accept $\langle w, c \rangle$ for some $\langle c \rangle$ if and only if $w \in L$ (in which case V_L accepts $\langle w, c \rangle$ for all $\langle c \rangle$)

$$\mathrm{NTIME}(t(n)) = \{L \mid L \text{ is decided by an } O(t(n))\text{-time NDTM}\}$$

Corollary 7.22 $\mathbf{NP} = \bigcup_k \mathrm{NTIME}(n^k)$

Summary

$\mathbf{P} = \{L \mid L$ can be decided fast$\}$

$\mathbf{NP} = \{L \mid L$ can be verified fast$\}$

open: $\mathbf{P} = \mathbf{NP}$?

know $\mathbf{NP} \subseteq \mathrm{EXPTIME} = \bigcup_k \mathrm{TIME}(2^{n^k})$