Chapter 7: Time Complexity

Time complexity

Let *M* be a deterministic Turing machine that halts on all inputs. The running time or *time complexity* of *M* is the function f: $N \rightarrow N$, where f(n) is the maximum number of steps that *M* uses on any input of length n. If f(n) is the running time of *M,* we say that

- *M* runs in time f(n)
- *M* is an f(n) time Turing machine

Big-O notation

- Let f and g be functions f, g: $N \rightarrow R^+$. $f(n)=O(g(n))$ if positive integers c and n_0 exist such that for every integer $n \geq n_0$

$$f(n) \leq cg(n)$$

g(n) is an asymptotic upper bound for f(n).

Small-o notation

- Let f and g be functions f, g: N → R^+.

  $f(n) = o(g(n))$ if for every positive integer c, n_0 exists such that for every integer $n \geq n_0$

  $$f(n) < cg(n)$$

Time complexity class

$\text{TIME}(t(n)) = \{L \mid \exists$ TM M s.t.

1) $L(M) = L$, and

2) $\forall$ input w, M halts on w in $O(t(n))$ steps (language L is decided by an $O(t(n))$ time Turing machine)$\}$

The class **P**

P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine. In other words,

$$P = \bigcup_k TIME(n^k)$$

Example 1: A language known to be in **P**

CONN ={<G>| G is a connected graph}

Example 2: A language known to be in **P**

PATH ={<G, s, t>| G is a graph, and s, t are two vertices. There is a path from s to t.}

PATH is in P

A polynomial time algorithm *M* for PATH operates as follows:

M = "On input <G, s, t>, where *G* is a directed graph with nodes *s* and *t*:

1. Place a mark on node *s*.
2. Repeat the following until no additional nodes are marked:
3. Scan all the edges of G. If an edge (a, b) is found going from a marked node *a* to an unmarked node *b*, mark node b.
4. If *t* is marked, *accept*. Otherwise, *reject*."

Example 3: A language known to be in **P**

Every context-free language is in P

Every CFL is in P

Let G be a CFG in CNF generating the CFL L. Assume that S is the start variable. A polynomial time algorithm *D* works as follows:

D = “ On input $w = w_1 \ldots w_n$:

1. If w = ε and S → ε is a rule, accept.
2. For i = 1 to n:
3.   For each variable A:
4.   Test whether A → b is a rule, where $b=w_i$.
5.   If so, place A in table (i, i).
6. For l = 2 to n:
7.   For i = 1 to n-l+1:
8.   Let j = i+l-1,
9.   For k = i to j-1:
10.   For each rule A →BC:
11.   If table(i, k) contains B and table(k+1, j) contains C, put A in table(i, j).
12. If *S* is in table(1, n), *accept*. Otherwise, *reject*. “

Verifier

A **verifier** for a language A is an algorithm V, where

A = {w | V accepts <w, c> for some string c}.

The time of a verifier is measured in terms of the length of w: |w|.

A **polynomial time verifier** runs in a polynomial time in the length of w. A language A is **polynomially verifiable** if it has a polynomial time verifier.

The class NP, definition

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

HAMPATH = {<G, s, t>| G is a directed graph with a Hamiltonian path from s to t} is in NP

Proof: the following is a NTM that decides HAMPATH problem in nondeterministic polynomial time.

HAMPATH={<G, s, t>| G is a directed graph with a Hamiltonian path from s to t} is in NP

N="On input <G, s, t>, where G is a directed graph with nodes s and t:

1. Write a list of m numbers, $p_1, \ldots, p_m$, where m is the number of nodes in G. Each number in the list is nondeterministically selected to be between 1 and m.
2. Check for repetitions in the list. If any are found, reject.
3. Check whether $s= p_1$ and $t= p_m$. If either fail, reject.
4. For each i between 1 and m-1, check whether (p_i, p_{i+1}) is an edge of G. If any are not, reject. Otherwise, all tests have been passed, so accept."

Theorem

Language L is in NP iff it is decided by some nondeterministic polynomial Time Turing machine.

Proof

($\longrightarrow$) Let A ∈ NP and A is decided by a polynomial time NTM *N*. Let *V* be the polynomial time verifier for A that exists by the definition of NP. Assume that *V* is a TM that runs in time n^k and construct *N* as follows:

N= "On input *w* of length n:

1. Nondeterministically select string *c* of length at most n^k
2. Run V on input (w, c).
3. If V accepts, accept; otherwise, reject.

Proof

(⟵) Assume that A is decided by a polynomial time NTM N and constructs a polynomial time verifier V as follows:

N= “On input <*w*, *c*>, where w and c are strings:

1. Simulate N on input w, treating each symbol of c as a description of the nondeterministic choice to make at each step.
2. If this branch of N’s computation accepts, accept; otherwise, reject.”

Time classes: **NTIME**(f)

NTIME(t(n)) = {L| L is a language decided by an O(t(n)) time nondeterministic Turing machine}

The class **NP**

$$NP = \bigcup_{k} NTIME(n^k)$$

Languages in NP

CLIQUE, SUBSET-SUM, SAT, 3SAT, FACTOR, ISO, VERTEX-COVER

SAT = {ϕ: ϕ is a boolean formula that has a satisfying assignment}

Polynomial verifier: guess assignment to variables, plug into ϕ, then verify

FACTOR= {(k, N): N has a non-trivial factor less than k}

Polynomial verifier:

- guess $1<d<k$
- divide d by N
- determine whether the remainder is 0

Polynomial time computable function

- A function *f* is a polynomial time computable function if some polynomial time Turing machine *M* exists that halts with just *f(w)* on its tape, when started on any input *w*.

Polynomial-time Reduction

A function **f** : $\sum^* \rightarrow \sum^*$ is a polynomial-time reduction from language L_1 to language L_2, $\mathbf{L_1 \leq_p L_2}$, iff

- **f** is a polynomial time computable function.
- for all w,

$$w \in L_1 \leftrightarrow \mathbf{f}(w) \in L_2$$

Theorem: If $A \leq_p B$ and $B \in P$, then $A \in P$

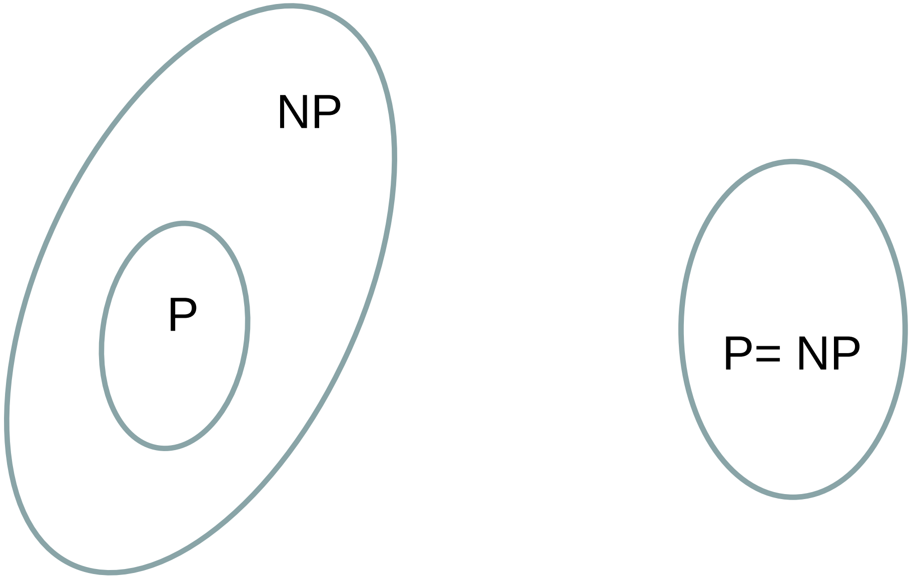
Proof:

Let M be the polynomial time algorithm deciding B and f be the polynomial time reduction from A to B. We describe a polynomial time algorithm N that decides A as follows.

N = "On input w:

1. Compute f(w).
2. Run M on input f(w) and output whatever M outputs."

P vs. NP



NP-hard

A language L is **NP-hard**

iff

$\forall A \in NP, \; A \leq_p L$

NP-complete

A language L is **NP-complete**

iff

1) $L \in NP$

2) L is NP-hard

The Cook-Levin Theorem:
SAT is NP-complete

$\text{SAT} \in \text{P}$ iff $\text{P} = \text{NP}$

3-SAT

3-SAT = $\{\phi(x_1, x_2 \ldots x_n) \mid \phi \in \text{SAT},$

$\phi = C_1$ **AND** C_2 **AND** $\ldots\ C_n$

where $C_i = \alpha_{i1}$ **OR** α_{i2} **OR** α_{i3},

and $\forall i,j\ \exists k$: $\alpha_{ij} = x_k$ or $\alpha_{ij} = \underline{x_k}\ \}$

Conjunctive Normal Form

3-SAT is NP-complete

Proof:

1) **3-SAT $\in$ NP** (by guessing the assignment to variables and verifying that $\phi(x_1, x_2 \ldots x_n) = 1$)

2) $\forall$**L, L $\in$ NP L $\leq_p$ 3-SAT**

Vertex Cover

Given graph G = (V,E), a subset V_1 of vertices is a vertex cover if each edge in E is adjacent to at least one vertex in V_1

Vertex-Cover = {(G,k)| G has a vertex cover of size at most k}

Directed Hamiltonian Path

Directed_HP = {(G,s,t)| G is a directed graph with a path that starts at vertex s, ends at vertex t, and visits every vertex of G exactly once}

CLIQUE

CLIQUE = {{(G, k)| $\exists$ S, S $\subseteq$ $V_{G,}$ s.t. |S| = k, and S is fully connected inside G}

SUBSET_SUM

SUBSET_SUM = {(A,k)| A is a set (list) of integers, s.t. $\exists B: B \subseteq A$ where $\Sigma_{w \in B} w = k$}

Reference

- www.cs.unm.edu/~gemmell/500.html by Dr. Peter Gemmell.