Section 1.1
section 2.1
section 2.2
section 2.5
Chapter 3 ← graphs

what we have done so far

Jul 7 180

A function $f(n)$ is a **diminishing upper bound** of a function $g(n)$ if

If for any constant $c > 0$, there is a natural number m such that

$$g(n) \le cf(n)$$

$$\text{for all } n > m$$

Formal definition

$$\forall c > 0 \ \ \exists m \in N \ \forall n > m (\ g(n) \le cf(n)\ )$$

It is denoted by $g(n) = o(f(n))$.

Examples:

$10x^2 + 100x + 1000 = o(x^3)$

$1000x^{100} = o(2^x)$

$10x^2 + 100x + 1000 \overset{\le\, cx^3}{=} o(x^3) \quad \forall c \quad c = 0.1$

$10x^2 + 100x + 1{,}000 \le 0.1x^3$

$100n^2 + 1{,}000n + 10{,}000 \le n^3 \quad n = 400$

$100n^2 + 1{,}000n + 10{,}000 \le 400n^2$

$200n^2 + (400)^2 n + (400)^3$

$100\, n^2 + 100\, n + 1000 \le c\, n^3$

$(100\, c^{-1})\, n^2 + (100\, c^{-1})\, n + 1000\, c^{-1} \le n^3$

$\sim + \sim + \sim < (400\, c^{-1})\, n^2$

A function $f(n)$ is a **growing lower bound** of a function $g(n)$ if

If for any constant $c > 0$, there is a natural number m such that

$$g(n) \geq cf(n)$$

for all $n > m$

Formal definition

$$\forall c > 0 \ \ \exists m \in N \forall n > m(\ g(n) \geq cf(n) \ )$$

It is denoted by $g(n) = \omega(f(n))$.

Examples:

$x^3 = \omega(10x^2 + 100x + 1000)$

$2^x = \omega(1000x^{100})$

$g = o(f) \quad \forall c \ \exists m \ \forall n > m \ (g(n) \leq c f(n))$

$g = \omega(f) \quad \forall c \ \exists k \ \forall n > k \ (g(n) \geq c f(n))$

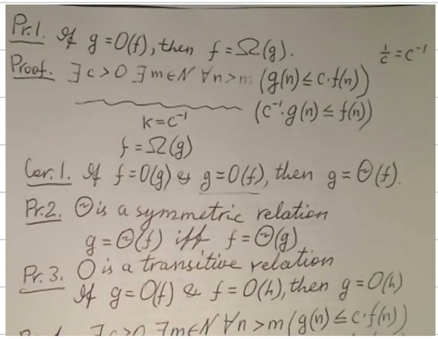
$h = \max(m, k)$

$c f(n) \leq g(n) \leq c f(n)$

$\Downarrow$ $\qquad c' \neq c$

$g(n) = c f(n)$

$g(n) = c' f(n)$



Pr. 1. If $g = O(f)$, then $f = \Omega(g)$. $\quad \frac{1}{c} = c^{-1}$

Proof. $\underbrace{\exists c > 0 \ \exists m \in N \ \forall n > m}_{k = c^{-1}} \ (g(n) \leq c \cdot f(n))$

$(c^{-1} \cdot g(n) \leq f(n))$

$f = \Omega(g)$

Cor. 1. If $f = O(g)$ & $g = O(f)$, then $g = \Theta(f)$.

Pr. 2. Θ is a symmetric relation

$g = \Theta(f)$ iff $f = \Theta(g)$

Pr. 3. O is a transitive relation

If $g = O(f)$ & $f = O(h)$, then $g = O(h)$

$\exists c > 0 \ \exists m \in N \ \forall n > m \ (g(n) \leq c \cdot f(n))$

Proposition 1° If $g = o(f)$, then $f = \omega(g)$

Proposition 3° If $g = o(f)$ & $f = o(h)$, then $g = o(h)$

Proposition 7° If $\lim\limits_{n \to \infty} \dfrac{g(n)}{f(n)} = 0$, then $g = o(f)$

Proposition 4° If $g = o(f)$, then $g = O(f)$

An important type of dynamic complexity measures is **Computational Complexity** of algorithms, which measures resources utilized by the algorithm.

Examples:

Time complexity $T_A(x)$

Space complexity $S_A(x)$

Worst-case complexity

$$T_A(n) = \max \{ T_A(x);\ l(x) = n \}$$

Average complexity

$$T_A(n) = \text{average} \{ T_A(x);\ l(x) = n \}$$

Best-case complexity

$$T_A(n) = \min \{ T_A(x);\ l(x) = n \}$$

All these measures are called **direct complexity measures** of algorithms.

There are also **dual complexity measures.** They measure complexity of the results of algorithms as well as of the problems solved by algorithms.

The most popular dual complexity measure is called *algorithmic complexity* or *Kolmogorov complexity*.

Informally, it is defined as the length of the shortest program, which is necessary compute the given result.

Types of problems:

1. Undecidable/unsolvable
2. Solvable/decidable
3. Tractable

A problem is **solvable** if there is an algorithm that can solve it.

A problem is **tractable** if there is an algorithm that has admissible complexity and can solve it. Usually it's mostly time tractability.

A problem is **tractable** if it is actually possible to find solutions to such problems in a reasonable amount of time.

$T_A(n) = O(p(n))$

Problems with the deterministic polynomial time complexity form the class **P**.

Problems with the nondeterministic polynomial time complexity form the class **NP**.

$P = NP$?

given set A of n #s find if $m \in A$

$T_A(n) = n$ linear complexity

$T_A(n) = a\,n$

Proposition 8 $g = O(f)$ iff $g = O(af)$ for $\forall$ # a

proof suppose $g = O(f)$. $\exists c \exists m \forall n > m \, (g(n) \le c f(n))$

$\exists d = ac \exists m \forall n > m \, (g(n) \le c f(n) \le a c f(n))$

1) $a > 1$

2) $0 < a < 1$ $d = c a^{-1}$ $g = O(af)$

$g(n) \le c f(n) = d(a f(n))$

b) suppose $g = O(af)$ $g = O(af)$

Sub linear Complexity

$T_A(n) = O(n)$

A is an ordered list of n #s $n = 2^k$

$m \in A$?

A $\underbrace{\quad}_{A_\ell}$ $\underbrace{\quad}_{A_u}$

P_{middle}

$P_{middle} = mid(A)$

Compare m with P_{middle}

1. $m = P_{middle} \to$ end $m \in A$
2. $m < P_{middle} \to$ compare m with $mid(A_\ell)$
3. $m > mid(A) \to$ compare m with $mid(A_u)$

k iterations $|A_{(k)}| = \left(\frac{1}{2}\right)^k n = 1$

$\log_2 n = k$ $2^{k-1} < n \le 2^k$

$k = \lceil \log_2 n \rceil$

$T(n) = O(\log_2 n)$

$\log_2 n = o(n)$

$T_A(n) = O(n^2)$ $T_A(n) = O(n^{100})$

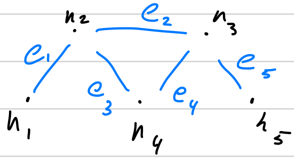
$T_A(n) = 2^n$

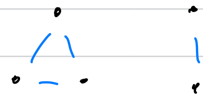
$O(2^n)$ exponential complexity

Missed stuff here 1:24 ←

Algorithims on Graphs

disconnected graph



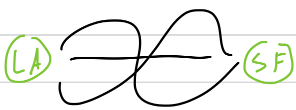


$g = (V, E)$

↑ Vertices / nodes ↑ edges

$g = (v, E, V)$ named set relation

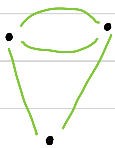
$e \in E$ $e = (u, v)$
$u, v \in V$



	n_1	n_2	n_3
n_1	0	1	0
n_2	1	0	1
n_3	0	1	0

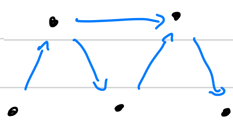
multigraph

$m = (V, E)$



directed graph

$g = (V, R)$



Def 1 path in g from u to v

$P = (v_1, v_2, \ldots, v_k)$ s.t. $v_1 = u$ & $v_k = v$ &

$(v_i, v_{i+1}) \in E$

Def 2 g is a connected if

$\forall u, v \in V \; \exists$ path p that connects u & v

Def 3 A connected component C of g

A subgraph H of $g = (V, E)$

$H = (V_1, E_1)$ if $V_1 \subseteq V$ & $E_1 \subseteq E$

C is a subgraph of g st

$\forall\, u, v \in V(C)$ can be connected by a path

C is a max connected sub graph of g

Def 4 A cycle in g is a path

$P = (v_1, v_2, \dots, v_k)$ st $v_1 = v_k$

Def 5 a tree is a connected graph without cycles