Algorithms and Data Structures 1

TIN060

Jan Hric

Lecture 1, v. 16.3.2015

Syllabus

Syllabus:

- An asymptotical notation
- (Binary trees,) AVL trees, Red-Black trees
- B-trees
- Hashing
- Graph alg: searching, topological sorting, strongly connected components
- Minimal spaning tree (d.s. Union-Find)
- Divide et Impera method
- Sorting: a lower bound on the complexity of sorting, average case of Quicksort, randomization of Quicksort, linear sorting alg.
- Algebraic alg. (LUP decomposition)

- Literature

  T.H. Cormen, Ch.E. Leiserson, R.L. Rivest, Introduction to Algorithms, MIT Press, 1991

- Organization
  - Lecture
  - Exercises

Comparing algorithms

- Measures:
  - A time complexity, in (elementary) steps
  - A space complexity, in words/cells
  - A communication complexity, in packets/bytes
  - (in practice: money ~ programmer/human time)
- How it is measured:
  - A worst case, an average case (wrt a probability distribution)
  - Usually an approximation: an upper bound
- Using functions depending on a *size* of input data
  - We abstract from particular data to a data size, |D|
  - We compare *functions*

Size of data

- Q: How to measure a size of (input) data?
- Formally: a number of bits of data
- Ex: Inputs are natural numbers $a_1, \ldots, a_n \in N$, a size D of input data is $|D| = \Sigma_{i=1}^{n} \lceil \log_2 a_i \rceil$
- A time complexity: a function f: N->N, such that f(|D|) gives a number of algorithm steps depending on data of the size |D|
- Intuitively: An asymptotical behaviour: an exact graph of a function *f* does not matter (ignoring additive and multiplicative constants), a class of *f* matters (linear, quadratic, exponential)

A step of an algorithm

- In theory: Based on an abstract machine: Random Access Machine (RAM), Turing m.
  - Informally: an algorithm step = an operation executable in a constant time (independent of data size)
- RAM, operations:
  - Arithmetical: +, -, *, mod, <<, && …
  - A comparision of two numbers
  - An assignment of basic data types (not for arrays)
    - Numbers have a fixed maximal size
- Ex: sorting of *n* numbers: $|D| = n$
- (Counter)Ex: a test for a zero vector

Why to measure a time complexity

- Why sometimes a faster machine doesn't help
- Time of f(n) for data of size n, 10^6 ops per second

			n				
f(n)	20	40	60	80	100	500	1000
n	20μs	40μs	60μs	80μs	100μs	500μs	1ms
n log n	86μs	0.2ms	0.35ms	0.5ms	0.7ms	4.5ms	10ms
n^2	0.4ms	1.6ms	3.6ms	6.4ms	10ms	0.25s	1s
n^3	8ms	64ms	0.22s	0.5s	1s	125s	17min
2^n	1s	11.7days	36ky				
n!	77ky						

Why to measure a time complexity 2

- A difference between polynomial and slower algs.
- How a speed-up of a computation enables to increase a size of a „workable“ data; the current size is *x*

		speed-up		
f(n)	original	10 times	100 times	1000 times
n	x	10x	100x	1000x
n log n	x	7.02x	53.56x	431.5x
n^2	x	3.16x	10x	31.62x
n^3	x	2.15x	4.64x	10x
2^n	x	x+3	x+6	x+9

Asymptotical complexity

- measures a behaviour of the algorithm on „big" data
  - Ignores a finite number of exceptions
- supresses additive and multiplicative constants
  - Abstracts from a processor, a language, (the Moore law)
- classifies algs to categories: linear, quadratic, logarithmic, exponential, constant ...
  - Compares functions

Asymptotical („Big“) O notation

- f(n) is *asymptotically less or equal* g(n),
  notation $f(n) \epsilon O(g(n))$, „big O“
  iff $\exists c > 0 \exists n_0 \forall n \geq n_0 : 0 \leq f(n) \leq c.g(n)$
- f(n) is *asymptotically greater or equal* g(n),
  notation $f(n) \in \Omega(g(n))$, „big Omega“
  iff $\exists c > 0 \exists n_0 \forall n \geq n_0 : 0 \leq c.g(n) \leq f(n)$
- f(n) is *asymptotically equal* g(n),
  notation $f(n) \in \Theta(g(n))$, „big Theta“
  iff $\exists c_1, c_2 > 0 \exists n_0 \forall n \geq n_0 : 0 \leq c_1.g(n) \leq f(n) \leq c_2.g(n)$

O-notation, def. II

- f(n) is *asymptotically strictly less* than g(n), notation $f(n) \in o(g(n))$, „small o“ iff $\forall c > 0 \exists n_0 \forall n \geq n_0 : 0 \leq f(n) \leq c.g(n)$
- f(n) is *asymptotically strictly greater* than g(n), notation $f(n) \in \omega(g(n))$, „small omega“ iff $\forall c > 0 \exists n_0 \forall n \geq n_0 : 0 \leq c.g(n) \leq f(n)$
- Examples of classes: O(1), log log n, log n, n, n log n, n^2, n^3, 2^n, 2^2n, n!, n^n, 2^(2^n),...
- Some functions are incomparable

Exercises

- Notation $f \underline{=} O(g)$ is sometimes used
- To prove: $\max(f,g) \in \Theta(f+g)$
- To prove: if $c,d>0$, $g(n)=c.f(n)+d$ then $g \in O(f)$
- Ex.: if $f \in O(h)$, $g \in O(h)$ then $(f+g) \in O(h)$
  - Application: A bound to a sequence of commands
- Compare n+100 to n^2 ; 2^10 n to n^2
  - Simple algorithms (with a low overhead) are sometimes better for small data