Program Optimization – algorithmic complexity

simon.marechal@synacktiv.com

2023/2024

Why is it important?

Good data structures, architecture, is the most crucial factor with regards to performance

Multiple queries

20-queries (bar) | Data table | Latency | Framework overhead

Responses per second at 20 queries per request, Azure D3v2 instances (240 tests)

Rnk	Framework	Performance (higher is better)		Errors	Cls	Lng	Plt	FE	Aos	DB	Dos	Orm	IA
1	vertx-postgres	4,751	100.0%	0	Plt	Jav	ver	Non	Lin	Pg	Lin	Raw	Rea
2	vertx-web-postgres	4,233	89.1%	0	Mcr	Jav	ver	Non	Lin	Pg	Lin	Raw	Rea
3	hapi	3,609	76.0%	0	Mcr	JS	njs	Non	Lin	Mo	Lin	Ful	Rea
4	http4k	2,230	46.9%	0	Mcr	Kot	sun	Non	Lin	Pg	Lin	Raw	Rea
5	vibe.d-ldc	2,156	45.4%	0	Plt	D	Non	Non	Lin	Mo	Lin	Mcr	Rea
6	http4k-apache	2,144	45.1%	0	Mcr	Kot	apa	Non	Lin	Pg	Lin	Raw	Rea
7	act-eclipselink-pgsql	2,122	44.7%	0	Ful	Jav	Utw	Non	Lin	Pg	Lin	Ful	Rea
8	vibe.d	2,118	44.6%	0	Mcr	D	Non	Non	Lin	Mo	Lin	Mcr	Rea
9	light-4j	2,089	44.0%	0	Plt	Jav	lig	Non	Lin	Pg	Lin	Raw	Rea
10	act-hibernate-pgsql	2,084	43.9%	0	Ful	Jav	Utw	Non	Lin	Pg	Lin	Ful	Rea
11	undertow-postgresql	2,078	43.7%	0	Plt	Jav	Utw	Non	Lin	Pg	Lin	Raw	Rea
12	chi	2,061	43.4%	0	Mcr	Go	Non	Non	Lin	My	Lin	Raw	Rea
13	chi-gojay	2,052	43.2%	0	Mcr	Go	Non	Non	Lin	My	Lin	Raw	Rea
14	chi-sjson-prefork	2,019	42.5%	0	Mcr	Go	Non	Non	Lin	My	Lin	Raw	Rea
15	act-hibernate-mysql	2,011	42.3%	0	Ful	Jav	Utw	Non	Lin	My	Lin	Ful	Rea
16	echo	2,006	42.2%	0	Mcr	Go	Non	Non	Lin	Pg	Lin	Raw	Rea

Figure 1: TechEmpowered benchmark, round 17

Big-O notation

Examples

Which complexity corresponds to which task?

- lookup in unsorted array
- lookup in a tree
- sort
- traveling salesman problem
- get n^{th} element of a vector

- $O(n.log(n))$
- $O(1)$
- $O(c^n)$
- $O(n)$
- $O(log(n))$

Definition

- describes asymptotic cost of some operation at the limit (usually, infinity)
- very commonly used to describe operation costs
- ignores constant factors

What should you know ?

- it is **very important** to know the expected cost of the important operations for several data structures,
  - cf. accidentally quadratic
- memory and computing efficiency will dominate big-O concerns for small enough n : do not underestimate compact vectors!
- there are trade-offs between space and time efficiency,
- $O(1)$ operations are usually lies, because of memory hierarchies.

Amortized cost

Amortized cost : when the cost of an operation is the mean cost over multiple runs

Amortized cost – example

```haskell
data DQ a = DQ [a] [a]

push :: a -> DQ a -> DQ a
push a (DQ front back) = DQ front (a:back)

pop :: DQ a -> Maybe (DQ a, a)
pop dq = case dq of
  DQ []        []   -> Nothing
  DQ (a:front) back -> Just (DQ front back, a)
  DQ []        back -> pop (DQ (reverse back) [])
```

push and pop are *amortized* $O(1)$, can you see why?

When does it fail?

- amortized costs on persisted structures;
- mean cost $\neq$ worst-case cost, beware of malicious and/or degenerate input
  - example: hash tables

Example, C++ data structures (source: Baptiste Wicht)

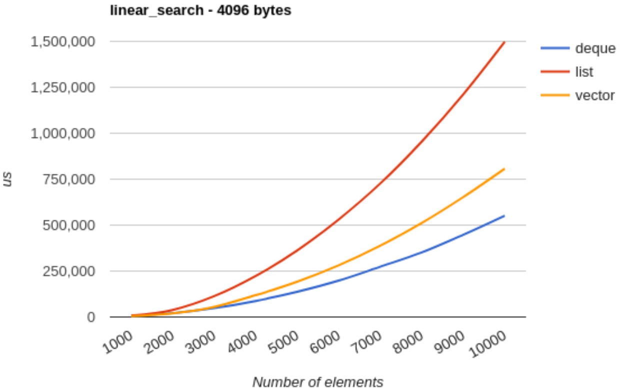

Figure 2: This is advertised as $O(n)$

Approximate solutions

Why?

Some problems are too costly to tackle directly, but an approximate solution might be just good enough!

Example:

- counting problem: HyperLogLog
- set membership: bloom, cuckoo filters
- compressing a video: all major formats

Important classes of approximation algorithm

- greedy algorithms
- local optimization
- black box optimizers, such as GA, sampling

Greedy algorithms sometimes acceptable

For the traveling salesman problem:

- brute force solution $O(n!)$
- dynamic programming $O(n^2 2^n)$
- greedy algorithm $O(n)$, 25% longer than optimal on average

QUESTIONS?

Thank you for your attention

SYNACKTIV
DIGITAL SECURITY