Other programming languages

Adam Boult (www.bou.lt)

September 28, 2022

Contents

Preface

This is a live document, and is full of gaps, mistakes, typos etc.

Part I

Unix operating system

Chapter 1

Unix file system

Part II

Objects and C++

Chapter 2

Objects

2.1 Introduction

2.1.1 Keys and values

2.1.2 Classes

2.1.3 Integer caching

If we set $x = 2$ we can either create 2 in memory, or simply point x to 2, which is already in memory

That means if we do $x = 2$ $y = 2$ they have the same pointer.

Can also cache some other common data values, eg empty lists.

Makes sense if pointer is smaller in memory than value.

2.2 Representing objects

2.2.1 Representing a single object

2.2.2 Null in objects

2.2.3 Representing a class with a multiple array (ie 2d)

2.2.4 Representing a class with a single array (ie 1d)

2.3 Functions with objects

2.3.1 Creating new objects

2.3.2 Getting values by field

2.3.3 Adding fields

2.3.4 Changing values in fields

2.4 Hierarchies of objects

2.4.1 Inheritance

Chapter 3

Object-Oriented Programming

3.1 Introduction

3.1.1 Introduction

in objects, OOP. essentially, all variable types are objects. inc integers, floats, lists etc

Chapter 4

C++

Part III

Parallel programming

Chapter 5

Parallel processing

5.1 Parallel processing

5.1.1 MapReduce

Framework for parallising problems.

Say we have text and we want to do a word count. We can split the text into multiple subsections, do a word count on each and add these up.

MapReduce can do this.

1: Splitting

The main server splits the database into subsections.

For example, the text document could be divided into M chunks, to be split across different servers.

The master server can then pass data to be Mapped when they have capacity. Each server could run Map on a large number of documents.

Each document here is stored as a key value pair. The value here would be the text, and the key would identify the document.

$\{2 : "AppleAppleCarrot"\}$

2: Mapping

Rather than simply doing a word count on the subsections, we do a more complex, but more efficient approach.

We map the key value pair to a list of key value pairs. In our example we would map this to identifiers for each word.

The output here would be

$[\{"Apple" : 1\}, \{"Apple" : 1\}, \{"Carrot" : 1\}]$

The Map function will differ depending on the use case.

3: Shuffling

We now want to combine the key value pairs. There are R keys from the intermediate output of the Map program.

For each of the R keys, the master server assigns a server to reduce the key.

The master server then notifies the Reduce worker of the location of each relevant document output from the Map process. The Reduce server then copies the key value pair.

The Reduce worker now has an input of key value pairs. In our example one would have received:

$\{"Apple" : 1\}$

$\{"Apple" : 1\}$

It would also have received other instances from other servers.

The Reduce worker can then combine these to form the following:

$\{"Apple" : [1, 1]\}$

4: Reduce

Once all the data has been mapped, the data can be reduced.

In our example this involves creating:

$\{"Apple" : 2\}$

The Reduce function will differ depending on the use case.

5: Output

The outputs of each reduce can then be shared back to the main server.

We now have our word count.

Part IV

Databases

Chapter 6

Databases

6.1 Database operations

6.1.1 Joins

Have index common to two tables

Cross join

Keep all columns. can name [table name].[column name] to preserve any differences. including for index (could be missing for some)

Not really matched if original n length, and other m, then new is n*m length. so all combinations of matches are kept

Inner join

Drops those where no match. means data dropped in index missing

Any predicate, equi join, equality predicate

Outer join

Keep data if none matches. left outer join for one table, right outer join for other, or full outer join

6.1.2 Insert

6.1.3 Select

6.1.4 Update

6.1.5 Delete

6.1.6 Call

Chapter 7

Structured Query Language (SQL)

7.1 Working with databases

7.1.1 Working with databases

¿ATTACH DATABASE ”/home/adam/Downloads/chinook.db” AS chinook;

SQLite3 show databases ¿.database

7.2 Working with tables within a database

7.2.1 Working with tables within a database

Use a database

Show tables within a database

7.3 Working with a specific table

7.3.1 Working with a specific table

¿SELECT * FROM Table;

Doing operations

¿SELECT DISTINCT * FROM Table;

¿SELECT COUNT(DISTINCT *) FROM Table;

Filters:

¿SELECT * FROM Table WHERE Name=”Adam”;

More on filters:

+ LIKE for regex? + BETWEEN for ranges? + IN for vector + Can use multiple using AND, OR, NOT

Chapter 8

SQLite

Chapter 9

MongoDB and NoSQL

Part V

Networking

Part VI

2D graphics

Chapter 10

Fonts

Chapter 11

Markdown

Chapter 12

LaTeX

Chapter 13

HTML

Chapter 14

Image manipulation

14.1 Introduction

14.1.1 Anti-aliasing

14.1.2 Rotating images

14.1.3 Layers

14.1.4 Scaling images

14.1.5 Mode 7

14.1.6 Other

2d graphics. sprites. palattes

14.2 Picture Processing Unit (PPU)

14.2.1 Introduction

Dedicated hardware. Each clock output pixel.

14.3 Audio Processing Unit (APU)

14.3.1 Introduction

Dedicated hardware.

Part VII

3D graphics

Chapter 15

2D ray casting and texture mapping

Chapter 16

Binary Space Partitioning

Chapter 17

3D Projection

Chapter 18

Texture mapping

Chapter 19

3D modelling

19.1 Introduction

19.1.1 Rasterisation

19.1.2 Shading

19.2 Ray tracing

19.3 Subsurface scattering

19.4 Ambient occlusion

Chapter 20

Texture filtering and anti-aliasing

Part VIII

Virtual Machines and interpreters

Chapter 21

Java

Chapter 22

Emulation

22.1 Low-level emulation

22.1.1 Emulating memory

big array of bytes hex numbers of some length

22.1.2 Emulating processors

emulation. cpu has operations. need to emulate them. convert to native. table with lookup for clock cycle requirements, mode etc.

22.2 High-level emulation

22.3 Containers

22.4 Virtualisation

Chapter 23

JavaScript

Chapter 24

Lua

Part IX

Python

Chapter 25

Python

Chapter 26

NumPy

Chapter 27

SciPy

27.1 Statistics

27.1.1 Introduction

SciPy includes statistics functionality. This is out of scope of here.

Chapter 28

Matplotlib

Chapter 29

pandas

Chapter 30

seaborn

Part X

R

Chapter 31

R

Chapter 32

tidyr

Chapter 33

ggplot2

Chapter 34

dplyr

Chapter 35

purrr

Chapter 36

tibble

Chapter 37

RMarkdown

Chapter 38

Shiny (R)

Chapter 39

data.table (R)

Part XI

Other languages

Chapter 40

STATA

Part XII

Improving knowledge

Chapter 41

Knowledge bases

41.1 Storing knowledge

41.1.1 Resource Description Framework (RDF)

Introduction

How can we store information like "Joe Blogs was born in the city London"?

Information is described as an RDF triple:

- Subject
- Predicate
- Object

Examples

"Joe Blogs was born in the city London" can be written as:

(Joe Blogs, BornCity, London)

Confidence

We can associate a confidence with each triple.

41.1.2 Knowledge bases as graphs

Introduction

We can consider each fact to be a mini graph.

For "Joe Blogs was born in the city London" we have:

Joe Blogs $\rightarrow^{wasborninthecity}$ London

41.2 Using knowledge

41.2.1 Inferring facts

Introduction

Now we add another fact:

London $\rightarrow^{isacityinthecountry}$ UK

We can use these to define a new predicate: "was born in the country" and generate the fact:

Joe Blogs $\rightarrow^{wasborninthecountry}$ UK

41.2.2 Relational learning

Introduction

Consider two facts:

Alice $\rightarrow^{IsA}$ Doctor

Bob $\rightarrow^{HasMother}$ Alice

We can consider another fact:

Bob $\rightarrow^{IsA}$ Doctor

Confidence of inferred facts

How confident should we be of this?

In practice the graph between Bob and Doctor will have many paths (qualifications)

41.2.3 Prior and posterior confidence

Introduction

If we have a new fact, and a prior, we can create a posterior condfidence on the fact.

41.3 Collecting knowledge

Chapter 42

Expert systems

42.1 Logic-based agents

42.1.1 Forward and backward chaining

Introduction

Forward and backward chaining

Has a knowledge base (domain specific content)

Has an inference mechanism (domain independent algorithms)

If we want to find out whether the knowledge base implies a statement, we can check using semantics or sytax.

Inference is similar to search. We start with the knowledge base. We can apply the inference rule to all statements which match the top line of modus ponens (or another rule).

The actions are inference rules. We add new sentences to the b

Agents can also interact with the world in order to add to the knowledge base

Convert each item in knowledge base to CNF for resolution rule.