Discrete Maths

For Internal Use Only

James W. Holman

April 12 2019

Contents

Chapter 1

Bases

1.1 Learning Outcomes

- Convert between different bases
- Be familiarised with binary and hexadecimal

1.2 A Brief Background on Bases

> "There are these two young fish swimming along and they happen to meet an older fish swimming the other way, who nods at them and says"Morning, boys. How's the water?" And the two young fish swim on for a bit, and then eventually one of them looks over at the other and goes "What the hell is water?"
> **- David Foster Wallace, This Is Water: Some Thoughts, Delivered on a Significant Occasion, about Living a Compassionate Life**

It's not often we need to think about bases, but in computer science it is fundamental. Bases, as the name suggests, are at the core of math we use. For example anything *Decimal* such as money, or anything measured under a metric system will reliably be in **Base 10**. Seconds and Minutes are, oddly enough, in **Base 60**.

There is nothing special about why these bases are chosen as our staples (Would we still like base 10 if we had 6 fingers on each had rather than 5?). Would we have base 60 if ancient Sumerians didn't have the tradition of counting to 12 on one hand using their knuckles, and found base 60 was the most compatible choice for working with fractions?

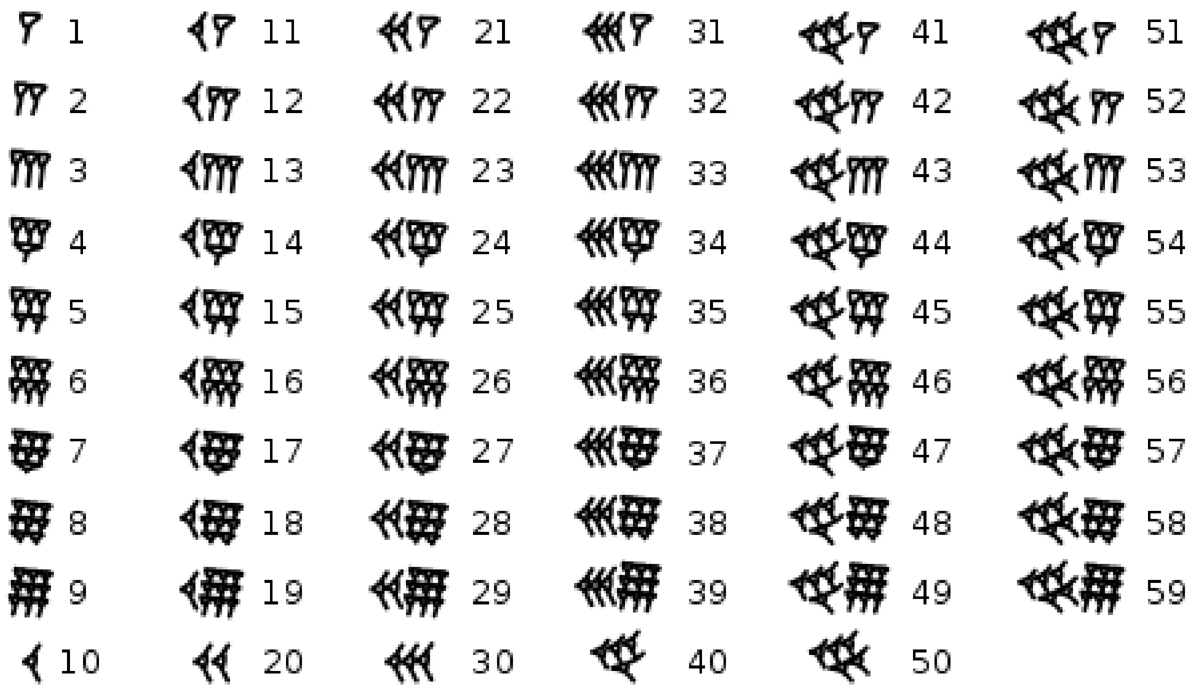

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59

Figure 1.1: Sumerian-inspired Babylonian Numeric System

Just like the seconds or minutes on a clock, **the base is the number you never reach**. You never meet a friend at 60 minutes past 3.

If you were to count to 5 in base 4 you would go:

Base 4	Base 10
0	0
1	1
2	2
3	3
10	4

That is to say, for 10_4 there is a single 4 (the base), and no ones. Base 10 doesn't have a numeral after 9, it just ticks over to 1 ten and 0 ones.

What in the world does he mean by that?

Don't worry aspiring mathematician, we'll get to that

Why are bases useful? Since the dawn of time humankind has developed increasingly clever ways to be lazy; a base allows us to be lazy by putting aside some of the numbers so that we don't need to think about them. A dozen-and-one-eggs saves you from having to think about 13 different eggs, you are just thinking about "a dozen eggs" and "one egg". Similarly now we think about 13 eggs; a group of ten eggs, and a group of three eggs.

1.3 Examples of Arbitrary Bases

Bases smaller than 10:

Alright enough self-indulgent intro for now. Here is a visual representation of how a base works.

Base	10			
BasePower	103	102	101	100

You may now also notice that when evaluating basepower we are displaying the result in base 10.

$$1 \times 64 + 8 \times 7 + 3 \times 1 = 123$$

Show/hide answer

Shortcut to appendix for PDF users

Extending this logic to converting between one arbitrary base to another, say from base 4 to base 7, is left as an exercise to the reader.

1.4 Binary

> "So the first distinction is between none and one (or 0 and 1). This is the yin (negative) and yang (positive) polarity of the Book of Changes, which Leibniz read in a Latin translation, and which gave him the idea that all numbers could be represented by the figures 0 and 1, so that for the series 1, 2, 3, 4, 5 we have 001, 010, 100, 101, 110, etc., which is now the arithmetic used by digital computers." **- Alan Watts**

Maybe people see everything as on-or-off due to the way our brains are wired, with neurons in on/off states to represent the experienced world. Maybe the world can boil down to a complicated expression of relative on/offs. Tall, short, fast, slow, divine, grotesque. In any case computing with 1's and 0's trace all the way back to their precursor of *yin* and *yang*, and can be used to describe anything from numbers, to sound, and images. If we can experience it, and model it, we can usually express that model in binary.

It is also it's on/off property which makes it perfectly suited for computers, which at a circuitry level, consist of transistors in either an on or off state.

Later, we look at binary logic, but for now let's call binary just base 2.

So with the knowledge of how we handled arbitrary base systems earlier, try converting 0001 0110 from base 2 into base 10.

Base	2				
BasePower	24	23	22	21	20
=	16	8	4	2	1
Numeral	1	0	1	1	0

16 + 4 + 2 = 22

*It's good practice to divide binary into groups of 4 bits at a time for ease of visibility (e.g. 1100 0011). A 'byte' in computing (yes, the same byte in **megabyte**) refers to 8 such bits of data*

Show/hide answer

Shortcut to appendix for PDF users

Mini-challenge: Now convert 7 into binary. How could you tell from looking at a string of bits whether a number in binary was odd or even?

1.5 Arbitrary bases larger than 10

Now it may seem trivial that 10 comes after 9, but what is 10 *really*? 10 isn't a numeral; it's 1 "ten" and zero "ones".

Well now he's just being confusing!
(Hang in there)

Say we wanted to represent 10 in base 12. How would we write it? Can we just write `10`?

12^1	12^0
1	0

Well that just doesn't work.

$$1 \times 12 + 0 \times 1 = 12$$

We've unintentionally written 12_{10} by thinking $10_{10} = 10_1 2$.
So what do we do when we run out of numerals after 9? Just use the alphabet.

`0, 1, 2,...,9, A, B, 10, 11`

Seeing as B is the 11th number, the 12th number becomes 10, which is to say there is 1×12^1 and 0×12^0.

There we go, we just counted to 13 in base 12.

Try counting to 15 in base 13.

1.6 Hexadecimal: How I Learned To Stop Worrying and Love Base 16

As mentioned previously (If spoiler tags are working, and you're not viewing this a book) a single byte is 8 bits, usually written out as two blocks of 4 bits.
What's the largest number that can be represented by 4 bits?

$$1111 = 8 + 4 + 2 + 1 \quad = \ ?$$

That's right! it's 15!

Hexidecimal (Street-lingo: hex) is commonly used to represent a byte, as rather than looking at a large batch of 1's and 0's you can look at a quarter as many 0-Fs. A few of you have seen hex used for representing color. For color hex is used as the intensity of Red, Green, and Blue; in the form `#RRGGBB`, e.g. : `#2a7a72` (Teal),

Side note: Numbers in hex (hexidecimal) are often shown in the form 0x___, such as 0x01.

1. What is $0x1E_{16}$ in decimal?
    - What is 127_{10} in decimal?
    - What is $6F_{15}$ in binary?
2. What is the largest value you can represent with a byte?
    - What does it look like in binary?
    - What does it look like in Hex?
    - How many different values can a byte represent?
3. What colour do you think (red, green, blue) = (247, 85, 49) would look like?
    - What would its value be as a hex code (Hint: `#__ __ __`)

1.6.0.1 Information Overload:

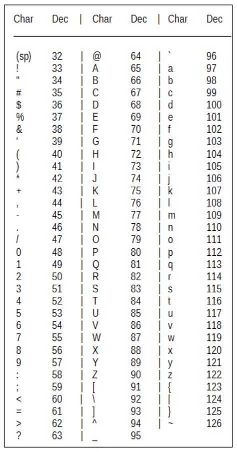
Funnily enough, every single thing stored on your computer is actually a pair of hex. There will be an address, like a street address, of where the information is stored. The number of the address is in hex, as in the information contained at that address.

```
0x0000: 14 00 20 00  0A 00 01 00  69 00 0F 00
0x000C: 4D 00 6F 00  72 00 6D 00  61 00 6C 00
.
.
.
0xFFF4: 0C 04 46 00  48 00 65 00  61 00 64 00
```

On the left are the addresses, on the right is the data stored at that address. The bits are converted back into a number, character, or other data type when they are accessed and used. How this occurs depends on the programming language.

1.7 Challenge:

You have intercepted the following message being sent. You realise that it is likely to be text and you are going to attempt to translate it back to ASCII (plain, readable text). First convert each hexadecimal number to decimal (base 10), then find which ASCII letter it maps to.



Char	Dec	Char	Dec	Char	Dec
(sp)	32	@	64	`	96
!	33	A	65	a	97
"	34	B	66	b	98
#	35	C	67	c	99
$	36	D	68	d	100
%	37	E	69	e	101
&	38	F	70	f	102
'	39	G	71	g	103
(	40	H	72	h	104
)	41	I	73	i	105
*	42	J	74	j	106
+	43	K	75	k	107
,	44	L	76	l	108
-	45	M	77	m	109
.	46	N	78	n	110
/	47	O	79	o	111
0	48	P	80	p	112
1	49	Q	81	q	113
2	50	R	82	r	114
3	51	S	83	s	115
4	52	T	84	t	116
5	53	U	85	u	117
6	54	V	86	v	118
7	55	W	87	w	119
8	56	X	88	x	120
9	57	Y	89	y	121
:	58	Z	90	z	122
;	59	[	91	{	123
<	60	\	92	|	124
=	61	]	93	}	125
>	62	^	94	~	126
?	63	_	95		

Figure 1.2: A list of ASCII characters and the decimal values they are mapped to

```
In hex:
0x0000: 49 27 6d 20  61 6c 72 65  61 64 79 20  54 72 61 63 65.
```

Hint: You may save time by writing a script in Ruby or JavaScript. This can be done by first making a function responsible for taking a single hex number and returning it as a decimal, then improving the function to convert the hex number straight to a character.

1.8 References:

Source: https://www.youtube.com/watch?v=0sZ74NSiJkQ Discrete Maths - Bases

1.9 Answers

1.10 1.1

Base	8		
BasePower	82	81	80
=	64	8	1
Numeral	1	7	3

1.11 1.2

Base	2				
BasePower	24	23	22	21	20
=	16	8	4	2	1
Numeral	1	0	1	1	0

Leibniz had already invented a binary number system before he was introduced to the I Ching by Joachim Bouvet. However he was shocked and attributed the original discovery of binary representations to ancient Chinese philosophy

Chapter 2

Logic

2.1 Learning Outcomes

On successful completion of this Unit students will be able to,

- Describe logical statements in logic notation
- Construct simple truth tables

2.2 A Brief Background on Logic

Last lesson when we learned about er'rythang being represented with 1's and 0's. This only explains the format things are stored in, and not the methods by which stored, or calculated, in the first place. What magic lays behind everything from *IF* statements, to *multiplication*.

It all boils down to logic. In particular Boolean logic. *Boolean logic is the basis for short circuit logic in Javascript*

Say we make a statement such as "there is a cat". This statement would evaluate to `true` or `1` when there is a cat, otherwise it would be `0`.
Often we will represent this as a symbol, e.g. $let c = cat$ now $c = 1$ if there is a cat, and $c = 0$ if there isn't a cat.

These statements can be combined together:

"If I was a rich girl (na, na)
See, I'd have all the money in the world"
- Gwen Stefani, Rich Girl

This should be familiar to y'all. It's a very short example of an IF statement. **If** Gwen Stefani "was a rich girl" **then** she would have all the money in the world.

r := "Gwen Stefani is a rich girl"
m := "Gwen Stefani has all the money in the world"

We would then say that **r** *implies* **m**. If r, then m. Or written in logic notation:

$$r \Rightarrow m$$

Always choose a nice lower case letter to represent a bit of logic, and express which one you have chosen so that everyone knows what you mean when you use it in formulae

So **if** we said r = `True`, or $r = 1$ (Which are both essentially the same statement), **then** $m = 1$.
Does that mean we can assert "If Gwen Stefani has all the money in the world, she would be a rich girl".
Not necessarily; in fact that's why the arrow is there. A better example of this might be:

"If it is raining I will have my umbrella"
- Me, uninspired example.

That does not necessarily mean "If I have my umbrella then it is raining".

I like to visualise it this way.

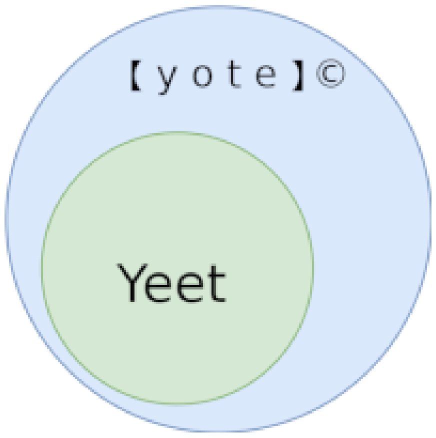


Figure 2.1: Logical Implies Diagram

You will notice that while yote© is in the blue circle, yeet is in both circles. This means we can assume something in the green circles is in the blue circle, but something in the blue circle isn't necessarily in the green circle.

Logic statements are not always straightforward, however.

"If you wanna be my lover, you gotta get with my friends"
- The Spice Girls, Wannabe

l := "Be my lover"
f := "Get with my friends"

If we were too haphazard here we might write $l \Rightarrow f$, following the same ordering as the previous examples. This would be *WRONG*. That means you would be saying "If you gotta get with my friends you wanna be my lover" which as we just addressed is not quite the same thing. Rather, it would be

$$f \Rightarrow l$$

At the core of it all we are still just trying to evaluate whether the statement is true though. Think about when the whole statement evaluates to true. It can be true if l and f are both true, or if l is true.

There are several logic operations which take input(s) and do interesting things with them. There are *AND*s, *OR*s, *XOR*s. When combined with with a *Negation* operator we even get an additional *NAND*, *NOR*, and the furtive *XNOR*. This is link between theoretical logic and physical circuitry, as these all can be represented as arrangements of transistors, and with enough of these logical operations you can perform any operation!

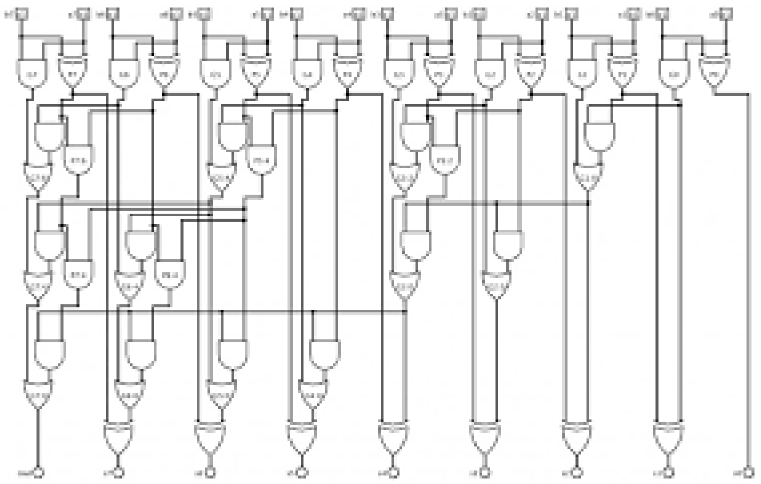


Figure 2.2: 8-Bit Ladner-Fischer Adder

Look! Lo and behold! It adds two numbers together bit by bit and all it uses are ANDs, NANDs, ORs, and XORs!

But we ain't here to do electrical engineering.

2.3 Common Logic Operations

2.3.1 ¬ Not

Takes an input and flips it. Consider the following.

$$a = 1 \iff \neg a = 0$$

Which is to say "*a* is *True*" is logically equivalent to "*not a* is *False*". Really all not does is flip whatever comes after it. If there are brackets, then not flips the evaluated expression inside of the brackets.

$$\neg(\neg a)$$

The external not is applied to the not expression inside the brackets. In this case the two nots would cancel each other out and the expression would be as good as saying a (*Two negatives make a positive*).

2.3.2 $\wedge$ AND

$$a \wedge b$$

$\wedge$ closely remembers an A. This can help you avoid confusion with symbols which are introduced later.

This statement evaluates to 1 ONLY when *both* a and b are true. I like to imagine there is a pipe with two valves in series, one after another. It is only when both valves are open that the *turmeric iced-latte* can pour through.

Used in a sentence:

> "He gon' skrrt and hit the dab like Wiz Khalifa"
> **- Mia Kalifa, Mia Khalifa on iLOVEFRIDAY ft. Mia Khalifa**

If he only either *skrrts* or *dabs* then this statement is false.

Or more simply:

> "You said"bell peppers and beef." There's no beef in here. So you wouldn't really call it "bell peppers **and** beef," now would you?"
>
> **- Spike Spiegel, aboard the Bebop.**

We can also show this in the form of a *truth table*.

bell peppers	beef	$bellpeppers \wedge beef$
0	0	0
0	1	0
1	0	0
1	1	1

It might not have been bell peppers *and* beef, but it could have been bell peppers *or* beef...

2.3.3 $\vee$ OR

$$a \vee b$$

For this expression to evaluate to *true*, all we need is either a or b to be *true*. Two parallel pipes with a shared source and sink. So long as one of them is open then the *Sriracha flavoured Dr. Pepper* will flow.

I would like an Nintendo Switch or a Playstation 4. Both would be great but I really would accept either.

2.3.4 $\oplus$ XOR

$$a \oplus b$$

Exclusive or (**xor**), is *false* when both a and b are *true*.

For example, my Grandfather infuriates my Grandmother. When she asks him "Tea or Coffee", he just responds "Yes".

The last or we dealt with was called an *inclusive* or, which means that when it is both a and b that are *true* it is as *true* as though only a or b were *true*.

Another example of exclusive or:

> "Hit or Miss"
> **- Mia Kalifa, Mia Khalifa by iLOVEFRIDAY ft. Mia Khalifa**

Hit can be *true* or **miss** can be *true*, but not both.

2.3.5 Bitstrings!

If you take two strings of bits, A = `1010`, B = `0110`, you can actually combine them together with a logical operator by performing it on each bit one at a time. i.e. for $C = A \wedge B$, $C_i = A_i \wedge B_i$.

Therefore:

$$1010 \wedge 0110 = 0010$$

2.4 Challenges

1. Convert the following into logic expressions:
   - If there is smoke then there's fire
   - If it is raining and I'm outside then I will have my umbrella
   - Either Trump is being blackmailed or he isn't, if he is then he is Putin's puppet
   - I will wear bunny ears on twitch if I get another subscriber or a donation
2. Fill out a truth table as done for AND:
   - To show the difference between **or** and **xor**
   - To evaluate $(\neg p \vee q) \wedge p$ for all values of p or q.
   - To evaluate $\neg(p \Rightarrow q)$ for all values of p or q.
   - To compare $p \oplus q$ with $(p \vee q) \wedge \neg(p \wedge q)$
3. Evaluate these:
   - $1101 \vee 1001$
   - $0101 \wedge 1001$
   - $\neg 0110$
   - $0110 \oplus 1100$

2.4.1 Beast Mode:

Hint: you may need to convert between bases, and treat the numbers as bitstrings

- $56 \vee 72$
- $24 \wedge 11$

2.4.2 Beast Mode+:

Hint: Feel free to find the hex code or decimal value for letters from an ascii table

- $33 \oplus 5$
- "F" $\vee$ "a"

2.4.3 Beast Mode ++:

Ever wondered how encryption works? XOR is at the foundation of AES, a common encryption algorithm

Cyphertext$_1$0 = 27 14 4 18 21 26 28 3 80

Key = "password1"

To find the encrypted word; you must xor each of the characters of Cyphertext with those of the key.
Hint: First letter is 'k'; and I recommend writing a quick script in ruby or JavaScript to do this.

2.5 References

Source: https://www.youtube.com/embed/e8LXrm0SUAk Discrete Math - Logic

Chapter 3

Sets

3.1 Learning Outcomes

On successful completion of this Unit students will be able to,

- Be able to write and describe set operations in set notation
- See databases in things. Anywhere and everywhere.

3.2 A Brief Background on Sets

You may have encountered sets and set operations while using databases. You most definitely have covered list. e.g.

```
my_list = ['fus','roh','dah']
```

A set differs from a list. The set describes what unique elements there are, where as a list is comprised of what each of the elements are.

```
my_other_list = [1,2,3,3,3,4,5]

list_to_set(my_other_list) = {1,2,3,4,5}
```

To put it plainly, there can be no repeated elements in a set (The curly fellas '{…}}' indicate this is a set rather than a list '[…]'). This also means that an element can actually exists in multiple sets at the same time.

If Dan likes Lo-Fi, and Jazz, and I like Lo-Fi and Funk- Then what do we both like? Expressed slightly differently:

$$D := A\ set\ of\ music\ genres\ Dan\ likes.$$
$$J := A\ set\ of\ music\ genres\ I\ likes.$$

To say what elements are in a set you would do so the following way:

$$D = \{Lo - Fi, Jazz\}$$

To say that an element is in a set you would write:

$$Lo - Fi \in D$$

Where the little $\in$ means that the thing on the left is *in* the thing on the right.

However you couldn't write

Try: - Defining set J - Saying what elements are in set J.

You can also list elements with a variable (Similar in some ways to a `for i in [1,2,3]` in code):

$$\{x|x \in D\} = \{Lo - Fi, Jazz\}$$

the $\{x|x\ is\ something\}$ part says that we are creating a set ($\{\}$) of all x where x is able to meet the rules on the right. If there were two variables $\{x, y|x > y\}$ then the set produced would be a set of pairs, where each pair had a larger number followed by a smaller number (e.g. $\{(2, 1), (5, 3), (33, -2)\}$).

These rules that come after the '|' can be chained together, in the following way:

$$\mathbb{N} = Natural\ Numbers\ (Integers\ above\ zero)$$

$$\{x|x \in N, x < 5\} = \{1, 2, 3, 4\}$$

In this case we are saying that for x to be included in the set, it must be a Natural number *and* less than 5

How would you, in this way, write an expression for the elements common to both D and J (The music Dan and I both like)?

3.3 Set Operations

It would become incredibly tedious to write out one of these full expressions every time you wanted to write an expression that said something to the effect of "What is in set A but also in set B?" or "What is in only in either set A or set B" or "What is in set A, but not set B" or the furtive "Which elements are in either set A or set B, but aren't in both?". *You may be noticing these resemble logic operations from the last topic.*
So what are the operations and their symbols?

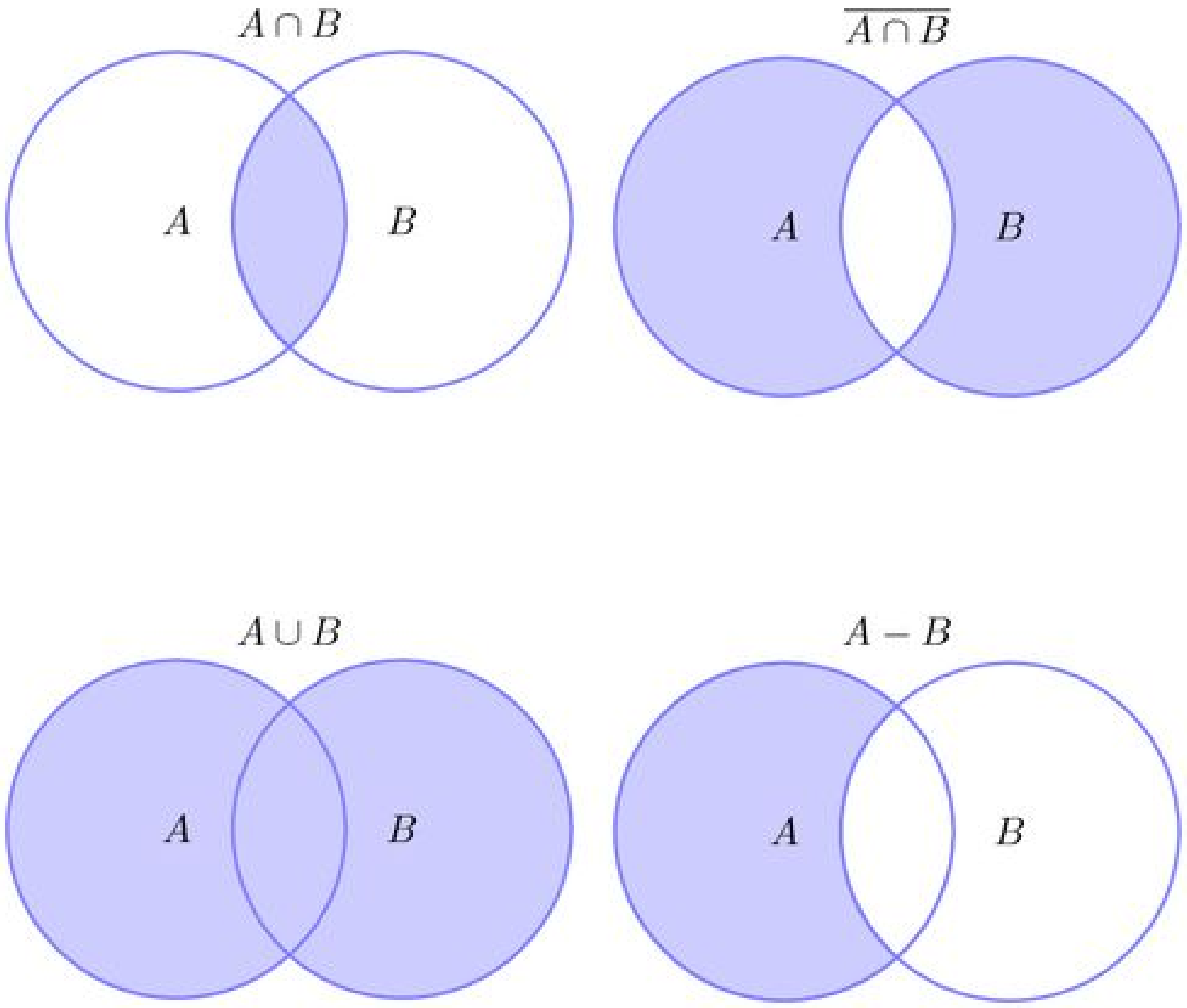


Figure 3.1: Set Venns

3.3.1 ∪ Union

You can remember the ∪ symbol as the **U** in **U**nion. A union performed on two sets will combine their elements into one set.

$$\begin{aligned} A &= \{1,2,3\} \\ B &= \{3,4,5\} \\ A \cup B &= \{1,2,3,4,5\} \end{aligned}$$

Notice, quite importantly, whilst 3 occurs in both A and B; there is only a single 3 in the combined set. We *never* see an element of a set repeated.

Equivalent in "Logic" topic: $\vee$

3.3.2 ∩ Intersection

The ∩ symbol for intersection resembles the **n** in i**n**tersection. An intersection of two sets A and B will be equal to a set of elements which can be found in both A and B.

$$\begin{aligned} A &= \{1,2,3\} \\ B &= \{3,4,5\} \\ A \cap B &= \{3\} \end{aligned}$$

Equivalent in "Logic" topic: $\wedge$

3.3.3 − Difference

Subtracting elements of one set from another. Symbol can also be / .

$$\begin{aligned} A &= \{1,2,3\} \\ B &= \{3,4,5\} \\ A - B &= \{1,2\} \end{aligned}$$

Equivalent in "Logic" topic: $a \wedge \neg b$ (Not this is less applicable to logic notation)

3.3.4 ⊕ Symmetric Difference

Can be seen as either subtracting the intersection (∩) from the union (∪) or alternatively; can be seen as subtracting A from B, then B from A.

$$\begin{aligned} A &= \{1,2,3\} \\ B &= \{3,4,5\} \\ A \oplus B &= \{1,2,4,5\} \end{aligned}$$

Equivalent in "Logic" topic: $\oplus$

3.4 A little extra

3.4.1 ⊂ Subset

What if all elements of set A were contained within set B? A set within a set?
We would then call A the subset of B, and B the superset of A.

$$A = \{1,2,3\}$$
$$B = \{1,2,3,4,5\}$$
$$\therefore A \subset B$$

Little triforce of dots is just "therefore"

3.5 $^{\complement}$ Complement

A complement of A is a set of every element not in A.

$$A = \{1,2,3\}$$
$$\{x | x \in \mathbb{N}, x < 10, x \in A^{\complement}\} = \{4,5,6,7,8,9\}$$

Which is also the same as.

$$A = \{1,2,3\}$$
$$\{x | x \in \mathbb{N}, x < 10, x \notin A\} = \{4,5,6,7,8,9\}$$

3.6 Closing thoughts

All these operations are incredibly important when it comes to how you handle information, especially in databases. While you learn different terms like "Inner Joins" and "Outer Joins", or "WHERE" as a search parameter in SQL; ultimately the concepts are underpinned by this set theory.

3.7 Challenges

1. Let Q = {a,b,c,d,e}, and R = {a,e,f,t}, find:
   - $A \cup B$
   - $A \cap B$
   - $(A - B) \cup (B - A)$
   - $A \oplus B$
   - Which of these is a subset of either Q or R? How would you express this in set notation?
2. Considering a string as an array of characters, let S = set("avacado"), T = set("mortgage")
   - What elements are in S?
   - What elements are in T?
   - When considering the *domain* of the alphabet (All the letters in the alphabet), what is the complement of both S and T?
   - How would you write this in set notation?

Beast mode:

- Write a function for each of the following, which takes two sets as arguments
  - The union
  - The intersection
  - The difference
  - The symmetric difference

(Hint: For the last one you can probably use calls to previous functions)

3.8 References

Source: https://www.youtube.com/embed/juadDxiHzuo Discrete Math - Sets

Chapter 4

Big O Notation

4.1 Learning Outcomes

On successful completion of this Unit students will be able to,

- Estimate the Big O of algorithms
- Be able to describe the different "Big O's" and give examples

4.2 A Brief Background on Big O notation

Whether they are only two lines of code or a complex black box abstracting the greatest works of a long-dead Russian Mathematician, we need a way to discuss and compare efficiency. Sometimes we are concerned with how long the algorithm might take, sometimes with the space it might take up. In either case; it's going to change with *the size of the input.* Writing down the name of 10,000 people is going to take longer than 5 people, and a lot more sheets of paper. We are looking at ways to reduce how much worse an algorithm performs with larger inputs; we are talking *efficiency* of an algorithm.

However, when we are concerned with time we don't actually talk about *time.* Instead we have the **number of operations** stand in. The reason for this is that the speed will differ from computer to computer, or depend on how many other applications you might have open. To work around this, we talk in terms of the *number of operations* that have to be completed in order for the algorithm to finish running. In most cases we are also considering the *worst case scenario.* If you made a password-guesser that tries every combination of letters randomly, the worst case scenario is that it guesses correctly on the last combination (hopefully before the sun burns out). Sometimes you want to consider the average case; if you were the person designing the security policy, you wouldn't want your security (and job security) relying on the hackers having *everything* go wrong.

We refer to the changing input-size as n. If there are multiple 'variables' which can change, such as the above food planner also being about to be used for many people. We would then refer to the changing number of people as m.

Let's give an example of a function.

```
function example(X)
    for element in X
        puts element
    return
```

Which part is the input with a variable size?

Yup. It's X. Good job.

Imagine that we give it a list to operate on:

```
L = [0,2,3,4,5,6]
```

There are 6 elements total in the list. How many *operations* would it take to make it through this list? 6.
If we were to graph how many operations it would take for other sized lists our graph would start to look a little like this.

Well that graph doesn't have anything going on that looks too special. The number of operations is linear, and this is represented as:

$$O(n)$$

Where n refers to the size of the input and $O(n)$ refers to *roughly* the number of operations.

There are also some rules about what we can and cant be bothered to talk about; the important thing is for the big O to give an impression of how efficient an algorithm will be.

```
function example2(X)
    len = length of X #O(1)
    for element in X #O(n)
        puts element
    i = 0
    while i < len/2 #O(?)
        puts element
        i++
    return
```

If we were to turn this into Big O notation, considering the second while loop will always be half the size of the first for loop, it might initially look like the following:

$$O(1) + O(n) + O(\frac{1}{2} \times n)$$

However there are two things wrong here.

1. We remove *coefficients*[1] of n. This means the above formula would now become.

$$O(1) + O(n) + O(n)$$

[1] A *coefficient* is the number next to the letter; for example in $3x$ the coefficient is 3.

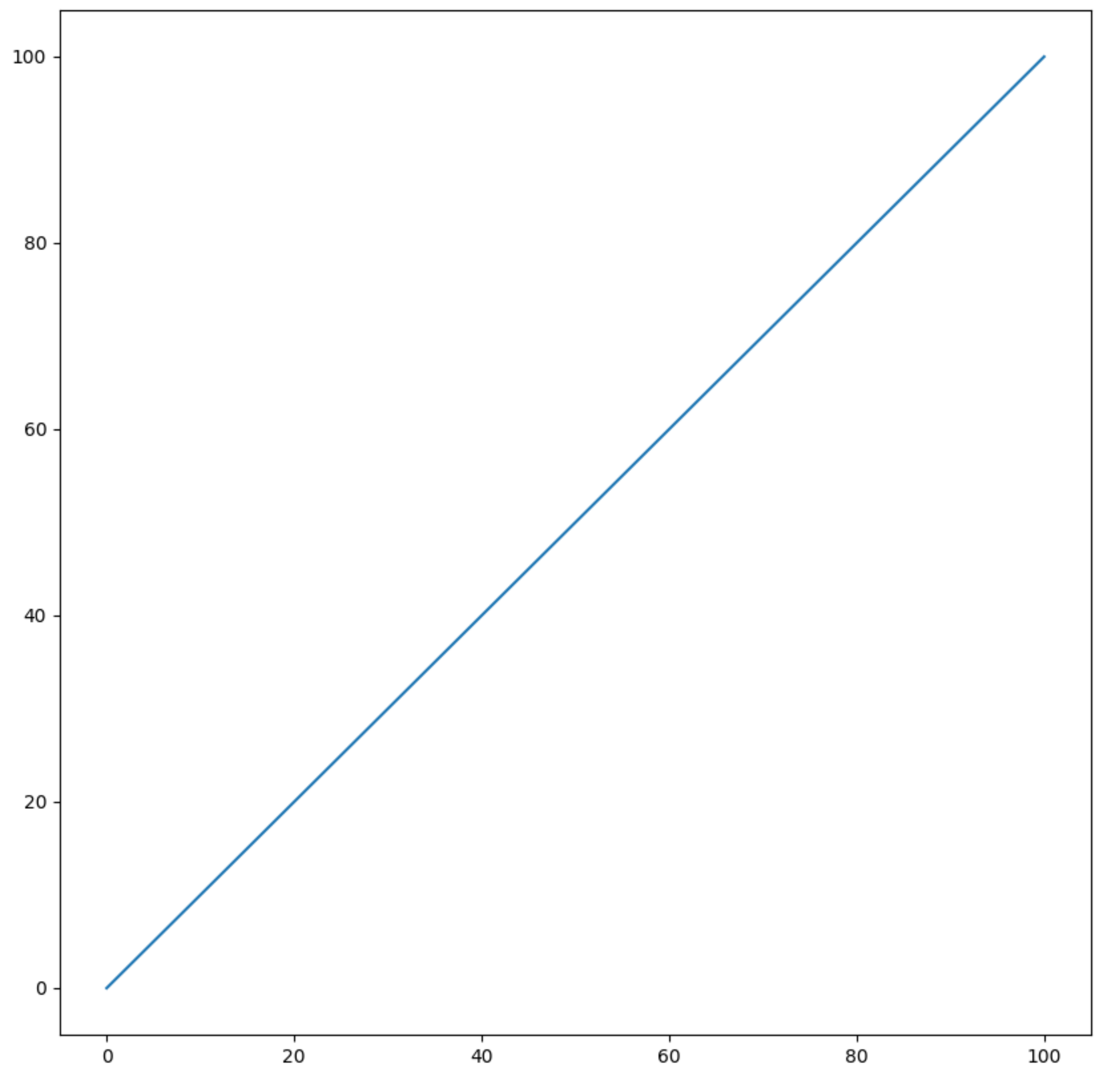


Figure 4.1: Linear Plot

In the practice the coefficient does influence the number of operations, especially for smaller sizes of n, or for very large coefficients. However, coefficients are still omitted in most cases, as we are not exactly profiling our code but rather providing an estimation of how much it changes with differently sized inputs.

2. We choose only to look at the least efficient part of the formula. We say that $O(n)$ *dominates* $O(1)$. We are now left with:

$$O(n)$$

Groovy.

It's kind of wicked we're able to reduce so many lines of code to a very short description of how much it's going to impact performance/efficiency. That is ultimately our goal, and knowing how to do this means being able to compare and communicate the efficiency of algorithms takign different approaches to solving a problem.

Another example!

```
function example3(X)
    len = length of X
    i = 0
    while i < len # O(n)
        for element in X # O(n)
            print element # O(1)
        i++
    return
```

This time the loops are nested. Meaning that for every step inside the while loop, the entire for loop is performed.

$$O(n \times n) = O(n^2)$$

4.3 Searching

Situation: We have two lists. 1. **List A**: Supplies we have on our space ship 2. **List B**: Supplies we will need to survive our next adventure

You are the ships token 'computer-person', and you need to develop some software to make this process easier.

First of all; how long would it take for an algorithm to work through either of these lists? (*it will be* $O(n)$)

How about if you wanted to see which items from A aren't already in B? Have a crack at writing out the code.

```
function we_still_need(A,B)
    .
    .
    .
    .
    .
    .
    .
    .
    .
    return
```

*Hint: If you are having trouble; this might be pretty similar to the **difference** code you had to write for the **Sets** topic (and if we were trying to find elements both in A and B it would be the intersection code you had written).*

How many comparisons would this take? For each element in A, you must compare it to each element in B. If you guessed $O(n^2)$, then you guessed right. We can make it more efficient, however.

Let's go back to finding a single element in a list. We said this was $O(n)$, but can this be done more efficiently?

For example if all the items are numbered we could perform a *binary search.* First we imagine a *tree* of the items. Where each *node* has:

- A value
- A left child
- A right child

Two children, hence *binary.*

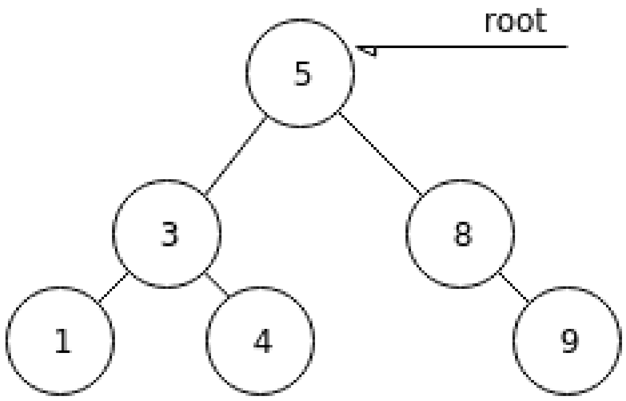


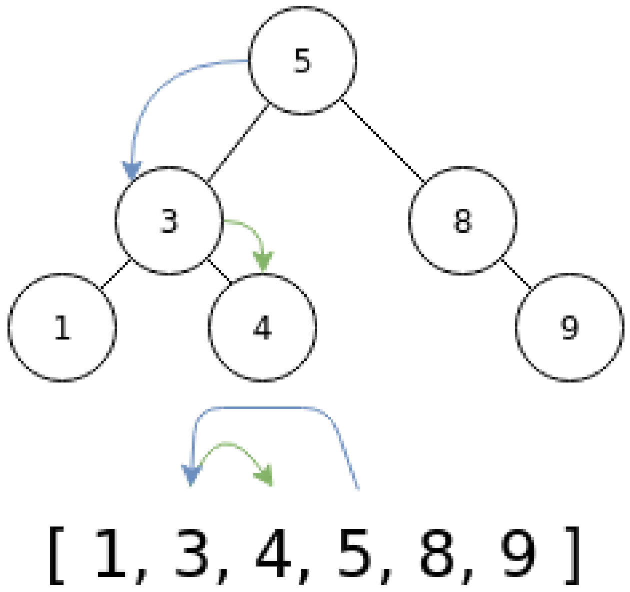
Figure 4.2: A binary tree

The list of sorted numbers which would generate this tree would resemble `1,2,4,5,8,9`. Notice the following three properties of the binary tree:

1. The root is roughly the center of the list
2. Each left child has a smaller value than its parent
3. Each right child has a larger value than its parent

Which means that if you are searching for a value, you compare it to node X; and if the value you are searching for is smaller than Node X's value you look at its left child. If your search value is larger you look next at its right child. Say we are looking for 4 in the above tree/list.
We:

- Compare 4 to 5, finding 4 is *smaller* so we look at 5's **left** child.
- Compare 4 to 3, finding 4 is *larger*, so we look at 4's **right** child.
- *Bingo*



4.3.0.0.0.1 With our 6 elements here how many steps has it taken us? 2! That's nothing! Just imagine the kind of performance gains you would get with thousands or hundreds of thousands of elements. Well... to be able to do that we need to find a way to mathematically represent it. It looks as though the worst case scenario is the number of layers (the depth) of the tree required to fin n nodes in.

How many nodes are there on each level? It appears to be $2\times$ the number on the previous level. Here there are 3 levels, so roughly $2^3 = 8$ nodes. Not so far off (Especially if 8 had a left child. If we were to abstract this it would be roughly:

$$2^{operations} \approx n$$

This wavy fella is just 'roughly equals'

or, alternatively:

$$operations \approx log_2(n)$$

We now arrive at the time complexity of $O(log_2(n))$, sometimes also written as $O\,log_2\,n$, which is surprisingly common in computing.

If we had **A** or **B** already sorted then we could find an element in one of them in $O(log_2(n))$ time. The above problem was seeing which elements we already have and which we still need? Could we perform a binary search on both? Unlikely, as we have to iterate through A entirely, then searching **B** to see if that element also exists there.

```
function example4(A,B)
   result = []
   for a in A # O(n)
        if binary_search(B,a) not True # Check if a in B using a binary search
            append a to result

    return
```

Think about what the time complexity might be for this code snippet. Think about how many times we might be performing the $O(log_2(n))$ operation.

In the end the time complexity is $O(log_2(n)) \times$ ____ = ____________

(*yes those a blanks for you to fill in*)

There's actually a cool tool for visualising this

4.4 Sorting

We're briefly going to touch on two sorting algorithms, as an example of when you have a choice of algorithms of different performances. Let's frame this problem in terms of playing chards.

4.4.1 Bubble Sort

- For each card, look at the next card.
- If they are in the wrong order then swap them.
- Move one card further into the deck.
- When you reach the end start from the beginning again.
- Stop when you do a full 'lap' of the deck without swapping the card.

This doesn't feel super efficient does it? Matter o' fact, it's unlikely you would even do this yourself if you were sorting by hand.
In Big O notation it is $O(n^2)$ as it takes at most n many laps of the deck to sort all cards.

4.4.2 Merge sort

- Split deck in half, and split again until you've split into 52x single card stacks.
- As you merge each of the stacks, first merging 2-card stacks then 4-card stacks then 8 etc, you sort these sub-stacks.
  - The magic of this is that each of the two stacks you are merging at any point is already sorted, so you only have to choose whether the next card in the merged stack comes from the one stack or the other (see image)

For example when you merge `[3,5,7]` with `[1,4,8,9]` you will end up with `[1,3,4,5,7,8]`.

This quite closely resembles a binary search, however at each individual layer there are at most n many comparisons. Meaning, that in total there are $O(nlog_2(n))$.

4.5 Resources

A cool page dedicated to the Big O of different algorithms, and using different data structures

4.6 Challenges

1. Go through each of the above time complexities and compile a list (There should be 5 in total), order this list of fastest-to-slowest (So that each Big O would dominate the one above it)

Beast Mode: - Find an example of each Big O time complexity from your code you've produced either within Discrete Math or elsewhere in coursework. It can be a single function or snippet. - Verify your answer by finding a big O graph.

Source: https://www.youtube.com/embed/rcsyf33vVmA Discrete Maths - Big O

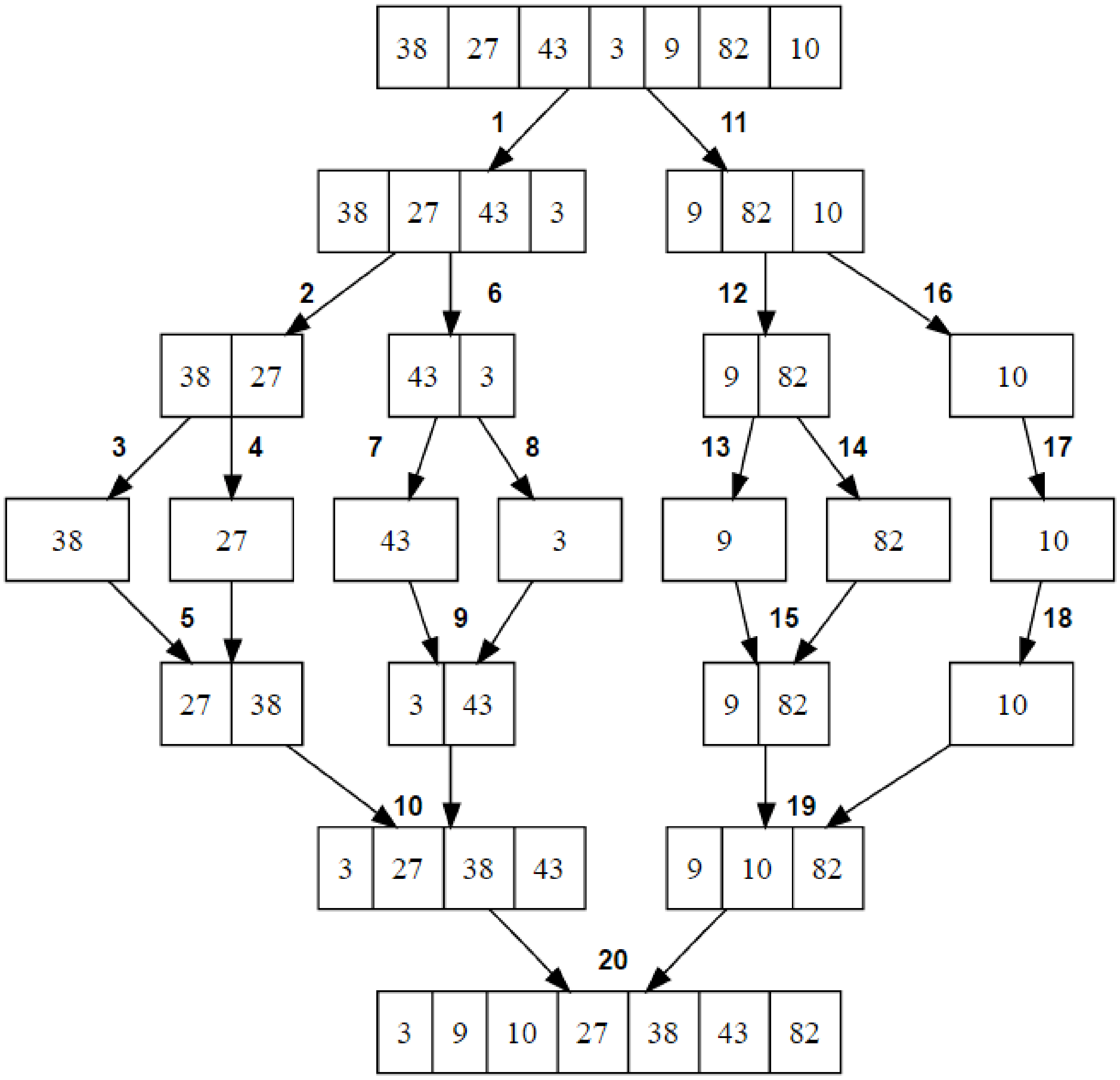


Figure 4.3: Merge sort visualised

Chapter 5

Matrices

5.1 Learning Outcomes

On successful completion of this Unit students will be able to,

- Communicate about matrices, being able to refer to points on a matrix
- Be able to perform some basic matrix operations
- Store and retrieve data from matrices in Ruby

5.2 Vectors

Think back to lists in programming.

```
L = [1,2,3,4]
```

These can also be thought of as a *Vector.* A vector just contains some number of values, existing in a dimension.

A column vector:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

A row vector:

$$Y = [y_1\ y_2\ \dots\ y_m]$$

It is common convention to have a capital letter represent a matrix, and the lower case indicate individual elements within that vector. The subscript character (1,2,m) is the index of an element within the vector.

Vectors are used for heaps of things. Besides mathematical purposes like Velocity Vectors and Force Vectors, they are used for storing information like colour values, colour vectors, and 3D points vectors for video games and movies.

A colour vector for example is:

$$colour = \begin{bmatrix} red\ intensity \\ green\ intensity \\ blue\ intensity \end{bmatrix}$$

Operations must be performed on these matrices to put them to use.

- Scalar multiplication
- Vector (Addition | Subtraction)
- Vector Product

5.2.1 Scalar Multiplication

Perhaps the simplest operation. Scalar multiplication is where the entire vector is multiplied by a single number. E.g.

$$\begin{aligned} X &= 4 \cdot [1\ 2\ 3\ 4] \\ &= [4\ 8\ 12\ 16] \end{aligned}$$

the dot just means multiplication

This works for either column or row vectors.

5.2.2 Vector (Addition / Subtraction)

This only works when the length and height of the two vectors are equal. We would refer to this as being the "Same shape".

$$\begin{aligned} A &= [4\ 3\ 8\ 2] \\ B &= [2\ 1\ 5\ 1] \\ C &= A - B \\ &= [2\ 2\ 3\ 1] \end{aligned}$$

You can see here we are subtracting a 1×4 vector from a 1×4 vector. Really we just subtract each individual element at a matching position. This could be expressed the following way:

$$c_i = a_i - b_i$$

This is saying that an element in vector C at position i (c_i) is an element from vector A at position i (a_i) minus an element in vector B at index i (b_i). i is between 0 and 3, as there are 4 elements in each of these vectors, such that $a_0 = 4$, or $c_3 = 2$.

Vector addition is trivial and left as an exercise for the reader.

5.2.3 Transpose

The transpose of vector V is represented by V^t. The transpose flips the vector across an imaginary diagonal line from top left to bottom right, such that:

$$V = [1\ 2\ 3\ 4]$$

$$\therefore V^t = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

another way to think about this is that we are rotating it counterclockwise by 90° *, then flipping it vertically*

$$\therefore (V^t)^t = [1\ 2\ 3\ 4]$$

5.2.4 Vector Product

Vector product is a little more odd. In this case one vector has to be a column, and one has to be a row.

We're about to walk through the formula. Don't let it overwhelm you, as we are going to take a look at each piece and by the end hopefully the whole thing will make a bit of sense. You don't need to get it right away, and you will get it eventually.

$$c_{i,j} = \sum_{k=0}^{n} a_{i,k} \times b_{k,j}$$

Lets pick this apart:

$$n$$

Here we are using n to represent the height of the column vector / the length of the row vector.

$$\sum_{k=0}^{n}$$

This part is a sum. The E like shape is the part telling us we are summing. It's saying that it will add together whatever comes after it, iterating through each index between 0 and n. e.g.

$$\begin{aligned} \sum_{k=0}^{4} j^2 &= 0^2 + 1^2 + 2^2 + 3^2 + 4^2 \\ &= 30 \end{aligned}$$

The other component is the $c_{i,j}$ format again. We are saying that the number in the ith row, and the jth column of **C** (aka $c_{i,j}$) is calculated from an operation on the ith row of **A** and the jth column of **B**. This operation involves summing the product of **0** through to **n** of each kth position of **A**'s row and **B**'s column.

If you understand the formula, you will understand the process. If you understand the process you will understand the formula. It will be much easier to grasp both once you bring yourself to understand one.

Now that you are beginning to understand that formula.

$$\begin{aligned}
A &= [4\ 3\ 8\ 2] \\
B &= \begin{bmatrix} 2 \\ 3 \\ 1 \\ 0 \end{bmatrix} \\
C &= A \cdot B \\
c_{0,0} &= \sum_{k=0}^{3} a_{i,k} \times b_{k,j} \\
&= (a_{0,0} \times b_{0,0}) + (a_{0,1} \times b_{1,0}) + (a_{0,2} \times b_{2,0}) + (a_{0,3} \times b_{3,0}) \\
&= 4 \times 2 + 3 \times 3 + 8 \times 1 + 2 \times 0 \\
&= 25 \\
c_{0,1} &= ? \\
c_{1,0} &= ?
\end{aligned}$$

Does $c_{0,1}$ exist? Or $c_{1,0}$? They can't. For $c_{0,1}$ there would have to be another column of B, and for the $c_{1,0}$ there would have to be another row of A. This means that the overall size of the answer should only be a 1×1 vector.

$$C = [25]$$

You can actually work out the shape of the final vector. If you multiply $1 \times N$ vector by a $N \times 1$ vector you end up with a 1×1 vector (Which is what just happened then).

If you were to do a $N \times 1$ by a $1 \times N$ you would end up with an $N \times N$ **matrix**.

5.3 Matrices

A matrix appears like a table or a spreadsheet. You can also think of it as a vector of vectors

$$M = \begin{bmatrix} 2 & 3 \\ 3 & 4 \end{bmatrix}$$

Matrices are absolutely everywhere. You remember the color vector from earlier? A picture is just a matrix of color vectors. They are also used to store the points of a 3D shape for computer graphics, and are the core of Machine Learning and Neural Networks.

As with vectors we have a few operations. The operations are almost identical (After all, a vector is just a matrix where there is either only one row or one column).

5.3.1 Matrix Scalar Multiplication

Multiplying individual elements by the the same scalar value (in this case 3).

$$\begin{aligned}
M &= 3 \times \begin{bmatrix} 2 & 3 \\ 3 & 4 \end{bmatrix} \\
&= \begin{bmatrix} 6 & 9 \\ 9 & 12 \end{bmatrix}
\end{aligned}$$

5.3.2 Matrix (Addition|Subtraction).

Once again, $c_{i,j} = a_{i,j}(+|-)b_{i,j}$, so that:

$$\begin{aligned} C &= \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix} - \begin{bmatrix} 2 & 8 \\ 4 & 7 \end{bmatrix} \\ &= \begin{bmatrix} 2 & -5 \\ -2 & -6 \end{bmatrix} \end{aligned}$$

5.3.3 Transpose

Transpose affects matrices as it does for vectors (see previous if you are diving in here), however with additional numbers it can be a little more confusing.
Start by imagining a line which begins at the top left element and then moves down and to the right at a 45 degree angle. The table then flips on this angle.

You could also visualise it as rotating 90° counterclockwise then mirroring it vertically.

E.g.

$$\begin{aligned} M &= \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^t \\ &= \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} \end{aligned}$$

5.3.4 Matrix Multiplication

Again we are bringing back this formula.

$$c_{i,j} = \sum_{k=0}^{n} a_{i,k} \times b_{k,j}$$

Time for a more visual representation :

These x's are multiplication

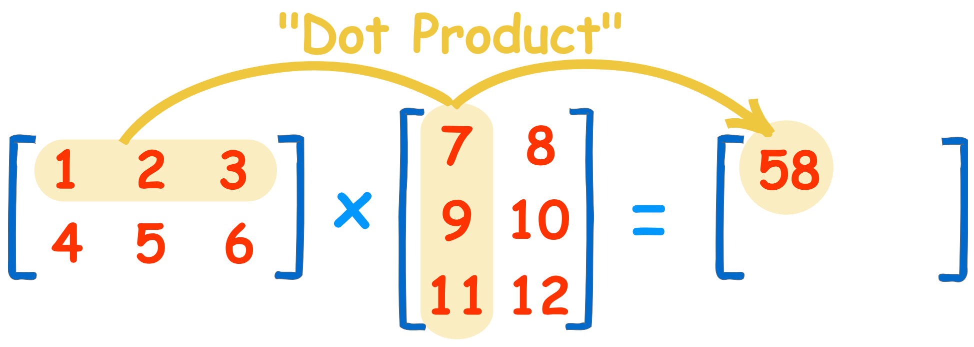


Figure 5.1: Matrix Dot a

Imagine that the first matrix is called **A** and the second is **B**, and the third is **C**. We are finding $c_{0,0}$ in this image. Here's what the calculation would look like:

$$\begin{aligned} c_{0,0} &= \sum_{k=0}^{n} a_{0,k} \times b_{k,0} \\ &= (a_{0,0} \times b_{0,0}) + (a_{0,1} \times b_{1,0}) + (a_{0,2} \times b_{2,0}) \\ &= (1 \times 7) + (2 \times 9) + (3 \times 11) \\ &= 58 \end{aligned}$$

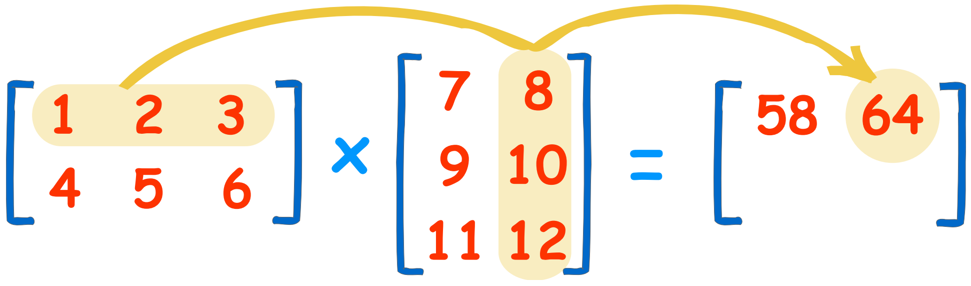


Figure 5.2: Matrix Dot b

Another example, for contrast.

$$\begin{aligned} c_{0,1} &= \sum_{k=0}^{n} a_{0,k} \times b_{k,1} \\ &= (a_{0,0} \times b_{0,1}) + (a_{0,1} \times b_{1,1}) + (a_{0,2} \times b_{2,1}) \\ &= (1 \times 8) + (2 \times 10) + (3 \times 12) \\ &= 64 \end{aligned}$$

You will also notice the dimensions of the final answer. If A has dimensions $N \times M$ and B has dimensions $M \times N$, then the final matrix will have dimensions $N \times N$. In this case 2×3 and 3×2 becomes a matrix of 2×2.

Try working out the last two values.

5.4 Challenges

$$A = \begin{bmatrix} 2 & 3 \\ 4 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 6 & 4 \\ 2 & -1 \end{bmatrix}$$

For the above, work out the following:

1. $A - B$
2. $B + A$
3. $A \times B$
4. $B \times A^t$

$$D = \begin{bmatrix} -1 & 6 & 2 \\ 4 & 1 & 1 \end{bmatrix}$$

$$E = \begin{bmatrix} 2 & 1 \\ -1 & -2 \\ 0 & 4 \end{bmatrix}$$

For the above, find:

5. $D + E^t$
6. $E - D^t$
7. $E \times D$
8. $D \times E$

Beast Mode:

1. Find the dot product of B and A for the vectors from the Vector Product section above.
2. Write ruby/JavaScript functions to take two matrices and do each of the following:
   - Add
   - Subtract
   - Multiply

Chapter 6

Reference

Matrix Drawings

Source: https://www.youtube.com/embed/slV587 Discrete Math - Matrix

---

Chapter 7

Graphs

7.1 Learning Outcomes

On successful completion of this Unit students will be able to,

- Frame problems as graph problems
- Describe graph structures in terms of matrices or edge lists

7.2 Intro

Graphs are a way of abstracting the structure or layout of something. It uses the concept of *nodes* and *edges* which involves specific nodes connected by edges create webs of connections. This is how map programs and algorithms are able to abstract maps, it is how 3D points of computer graphics are connected to each other, and it's how Google's search algorithm PageRank was able to rank pages.

The *node* is an individual point, like a city, a street corner, or a website. The *edge* is the road which connects the cities, the street which connects the street corner, or the hyperlink which points at the website. We can represent this as:

$$e(v, w)$$

This states that v and w are nodes and e is an edge connecting v to w. Another way of representing this is:

$$(v, w) \in E$$

Remember from sets that ∈ means "in"

This says that among all the edges there are (the set E), there are two nodes which are connected (v, w)

E in this case would contain a pair to represent every connecting edge which exists in a graph.

The simplest example of these is one we covered in the Big O Notation topic- Trees.

7.3 Trees

A tree is a graph which contains no *cycles* (aka, there are no closed loops), meaning that every *child* has a parent all the way up to the *root*. In fact, the file system of a computer is a tree. In Unix based systems like Linux or Mac, it is a single one-rooted tree.

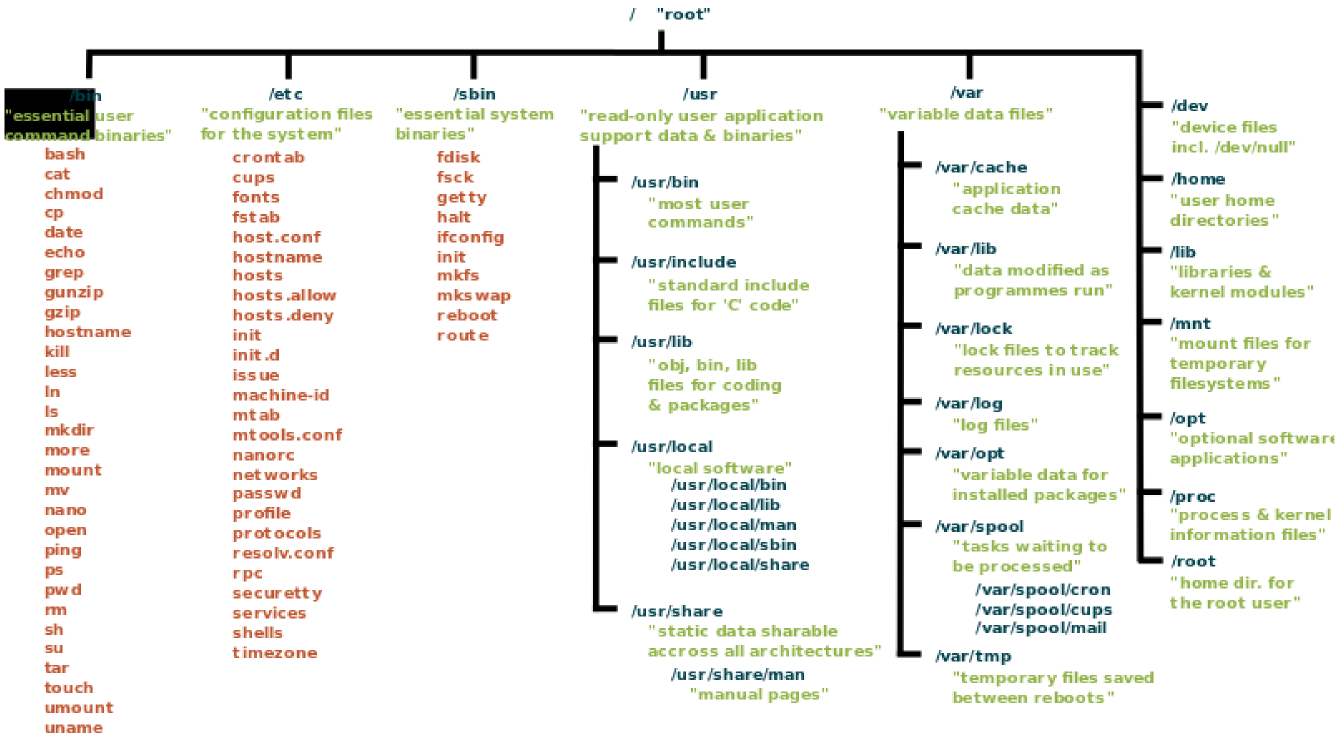


Figure 7.1: Unix tree

A family tree is a simple example of a tree, if you are tracing a individual trait; ignoring marriages and funny business.

The simplest of trees is a binary tree. Like all trees it has a root, which is the node from which the rest of the tree originates. Every node, including the root, can have up to two children, and every node, besides the root, has a single parent.

If the tree had more than two children then it wouldn't be a *binary tree*. Also, if a node had an edge with any node besides a direct parent or child, then it *wouldn't be a tree*.

The binary tree's other special property is that the left child of a node will always be smaller than the node, and the right child will always be larger. This enables highly efficient searches. Are there any problems with the above tree according to this?

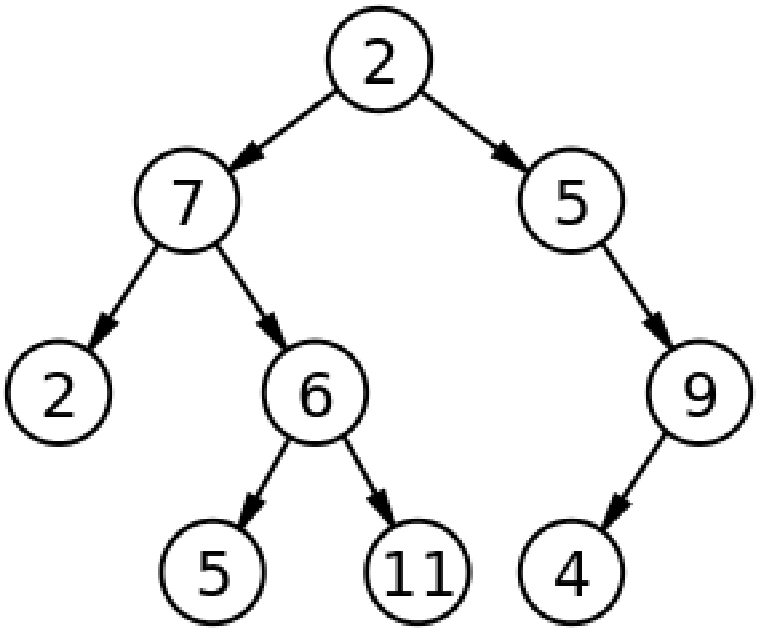


Figure 7.2: tree

7.4 Graphs

A graph doesn't follow this parent/child relationship, though it is still comprised of nodes with edges. So how on earth will we represent one?

Notice here that the edges are represented by arrows. This would be the road-example equivalent of a one-way road. If there are lines with arrows on them we call the entire graph a *Directed* graph, or a digraph. You can get from A to C, but you are unable to get from C to A. If there had been lines without arrow-heads then it would be an *undirected*. Any edge could be traversed in other direction.

Which is to say, for an udirected graph:

$$(v, w) \in E \Rightarrow (w, v) \in E$$

Where the right arrow is showing that the left hand side *implies* the right hand side (logic section). In other words: if there is an edge from v to w then there is a line from w to v.

For a directed graph:

$$(v, w) \in E \nRightarrow (w, v) \in E$$

An edge from v to w does not mean there has to be a line from w to v . There could be? but we cannot infer that from $(v, w) \in E$

But we face a challenge: We cant use the same parent-child relationship as we do for trees. How do we represent a graph? There are a few ways.

If we wanted to represent the above graph, we might use either:

A matrix: The row is the *from* and the column is the *two.* Aka, if you pick the B on the left, and the D on the top you have chosen *a line from B to D.*

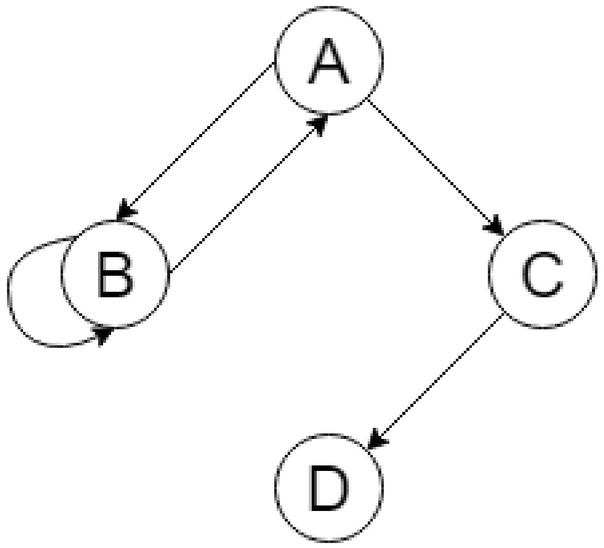


Figure 7.3: Digraph

$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & 0 & 1 & 1 & 0 \\ B & 1 & 1 & 0 & 0 \\ C & 0 & 0 & 0 & 1 \\ D & 0 & 0 & 0 & 0 \end{array}$$

Where a 1 represents an edge as existing, and 0 represents no edge.

An edge list Literally a list of edges. Presented in the form, as above, of (v, w) ; where v is the *from* and w is the *to* (i.e. from v to w).

$$E = [(A, B), (B, A), (B, B), (A, C), (C, D)]$$

If the entire graph was undirected then you could use either just (A, B) or (B, A) to represent an undirected edge or $(A, B), (B, A)$. However as there are some edges which aren't undirected we need to be specific that undirected edges are merely two single, directed edges. In other words we must use the latter form.

We may also need to have information about the edges store in a graph. This may be things such as the distance of a road, the delay in an internet connection, or the number of clicks on a link. Both the matrix and edge list representations have their own way of allowing for these *edge weights*.

This is a *weighted directed graph* or *weighted digraph* for short.

From this we can make statements like:

$$w(B, A) = 3$$

Where w is short for *weight*.

When we add in weights, the matrix now appears as follows.

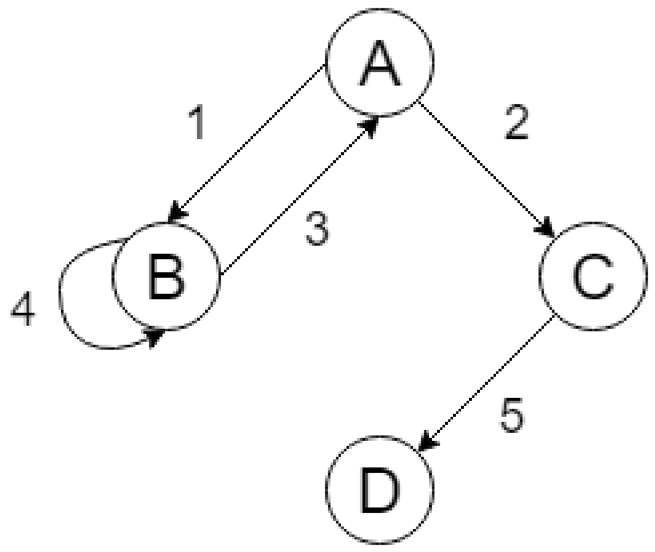


Figure 7.4: Weighted Digraph

$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & 0 & 1 & 2 & 0 \\ B & 3 & 4 & 0 & 0 \\ C & 0 & 0 & 0 & 5 \\ D & 0 & 0 & 0 & 0 \end{array}$$

The edge list has a bit more freedom. Rather than pairs we now have three-element tuples which contain the weight of the edge. Personally I prefer to put the weight between the two nodes as follows:

$$E = [(A, 1, B), (B, 3, A), (A, 2, C), (C, 5, D)]$$

7.5 Shortest Path

If there is a path one can take between one point and another then there will be a shortest path (If there are multiple shortest paths any one will do). If you were to find the shortest way to get to the station from a house, you would pick the roads which would sum to the least distance, and shortest path is much the same. Sometimes it will be finding the route with the least delay to send a message over the internet, sometimes it's something like the directions search in Google Maps.

7.6 Challenges

1. Try creating a node class in ruby, with a left child and right child property. Create a tree class which has functions to add new nodes to the tree. It doesn't need to be sorted, but try making it add a new node to either a left or right child based on whether it is larger or smaller than the parent. (This is a binary tree)
2. Draw out the graph for

$$(S, 5, A), (S, 2, B), (A, 4, C), (B, 8, A), (C, 6, D), (A, 2, D), (C, 3, F), (D, 1, F)$$

3. Find the shortest path from S to F in the graph drawn for the previous question.

4. Draw out the graph for

$$G = \begin{bmatrix} 0 & 4 & 0 & 0 & 0 & 0 & 0 & 8 & 0 \\ 4 & 0 & 8 & 0 & 0 & 0 & 0 & 11 & 0 \\ 0 & 8 & 0 & 7 & 0 & 4 & 0 & 0 & 2 \\ 0 & 0 & 7 & 0 & 9 & 14 & 0 & 0 & 0 \\ 0 & 0 & 0 & 9 & 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 4 & 14 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 11 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

5. Now find the shortest path from G_0 to G_8.
6. When do you think it will use less computing space to use a Matrix? How about an edge list?
7. Consider Fig 7.5. Convert it to a matrix.

7.7 Beast mode++

1. Add the ability of the tree to delete individual nodes in a tree
2. Make the tree able to sort itself (You might need to check out tree rotation, and bubble up/bubble down)
3. Create a class to add nodes to a graph, and a function which says the weight of an edge between two given nodes.

7.8 Beast Mode MEGA

- Make a shortest path function

7.9 Reference

Source: https://www.youtube.com/embed/o3_AAYVGfjQ Discrete Math - Graphs

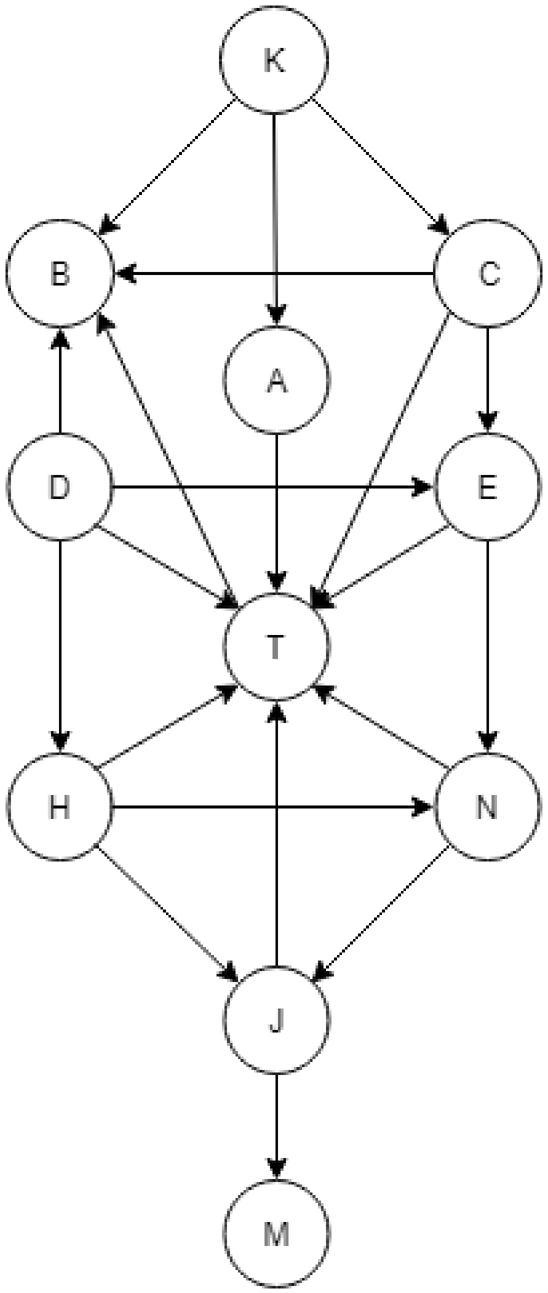


Figure 7.5: Kab

Chapter 8

Functions and Relations

8.1 Learning Outcomes

On successful completion of this Unit students will be able to,

- Identify functions which can or can't have an inverse
- Be able to give examples of reflexive, symmetric, and transitive relations
- Draw up relation matrices

8.2 Functions

When we think of a function we often think of a black box that takes some kind of input and spits out an output.

Or we might think of a mathematical function which takes x and returns a y, also called the function of x (aka $f(x)).

Sometimes in programming we are the one that gets to see inside this black box by being the person who constructs it:

```
def powerTwo(a)
    return a^2
end
```

You could also represent this function slightly differently, as a set of pairs between x and $f(x)$ (meaning, the function applied to x).

$$\{(1,1),(2,4),(3,9)\}$$

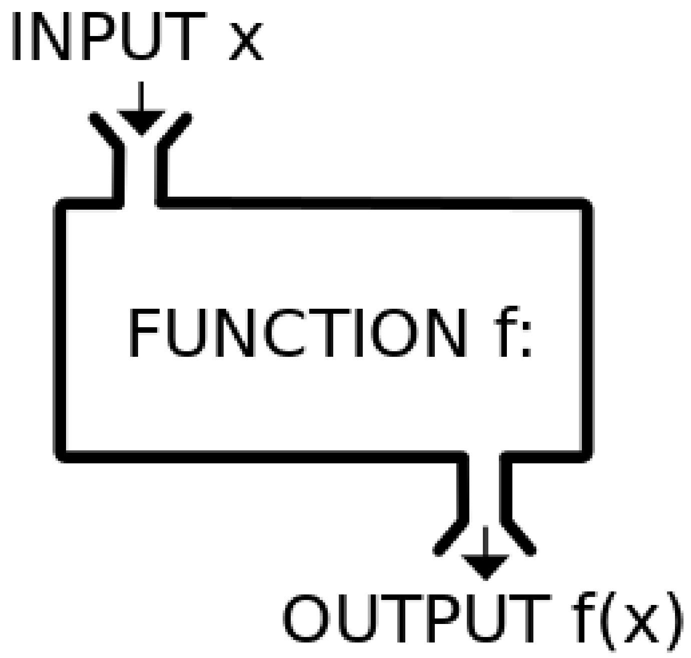


Figure 8.1: A function as a black box

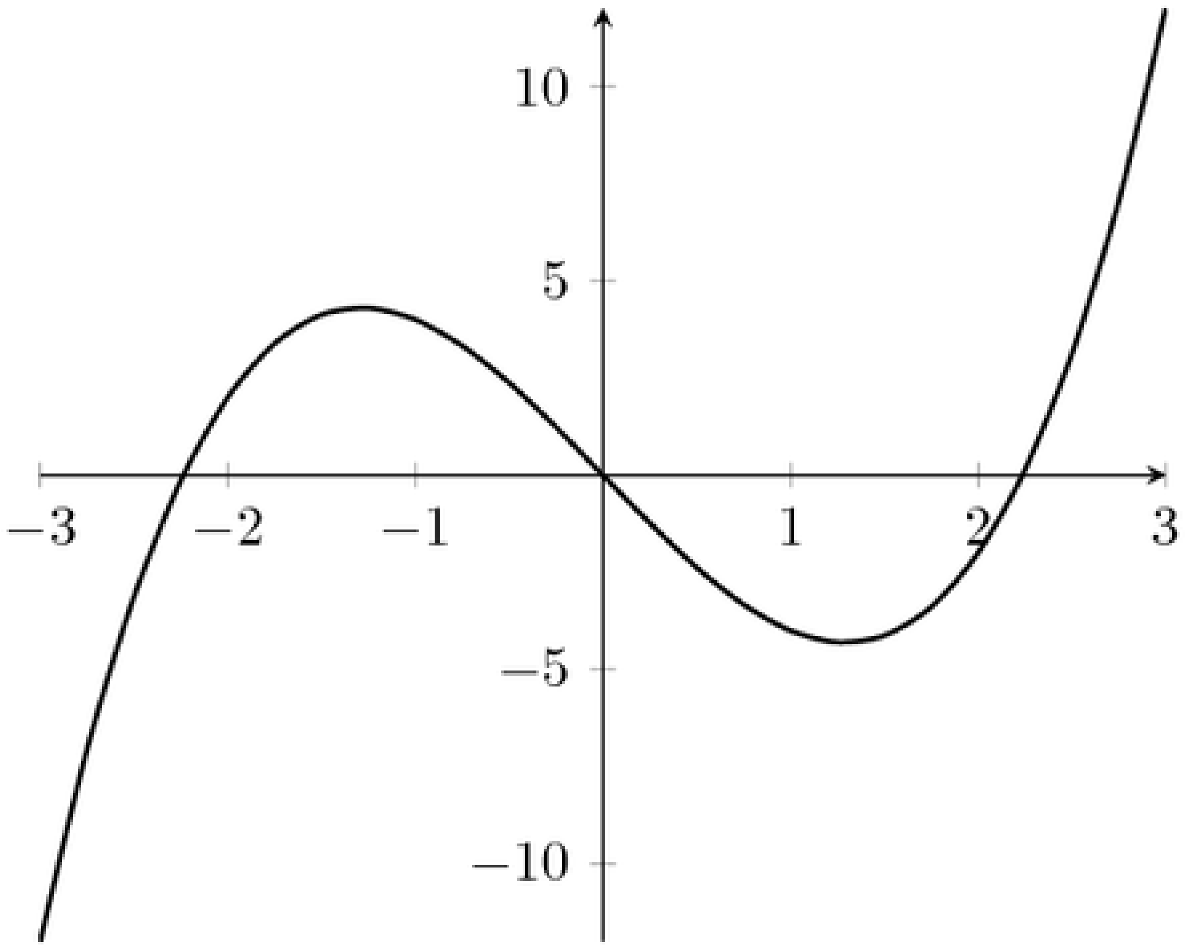


Figure 8.2: A simple graphed function

These are only the first three pairs, and writing out every possible pair for this function *could get very tedious*. In fact writing a function in the form $f(x) = x^2$ is essentially a shorthand for describing a *relationship* between any input and output.
The element on the left of a pair, or the input, is a member of the *domain*, the one on the right, the output, is a member of the *range*.

For the above set these would be:

$$Domain = \{1, 2, 3\}$$
$$Range = \{1, 4, 9\}$$

The pairing between an element of the domain, and an element of the range is referred to as mapping; as in: *2 maps to 4* for the above example. A *relation* can be thought of as a set that contains every pair which maps from an element in the domain to an element in the range.

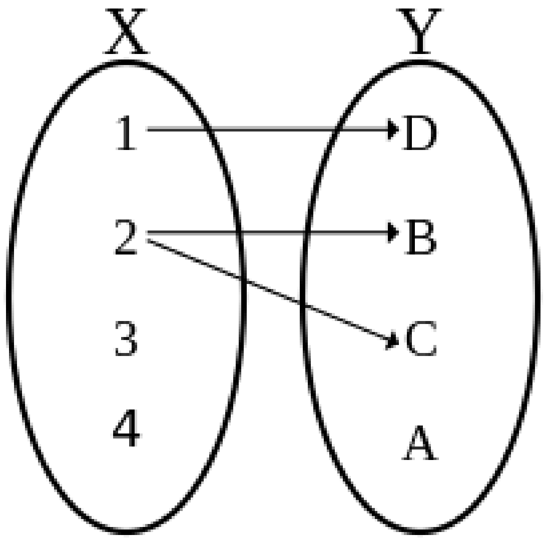


Figure 8.3: A Mapping Diagram

For the above example the *relation*, represented as R, would be:

$$\{(1, D), (2, B), (2, C)\}$$

Can you think of a way you would write a function which *maps* from each of the elements in the *domain* to each of those in the *range*? Might be a bit difficult- 2 seems to map to two different values!
This is where functions and relations are different. For a function, every element in the range is mapped to from a unique element in the domain. This is to say, that an element on the left of this diagram can ONLY map to ONE element on the right.

Consider the following:

$$\{(1, 0), (2, 1), (3, 0), (4, 1), ...\}$$

Here the *relation* is “Is it even?”. If the element from the domain (x) is even, then the element in the range (y) is “1”, otherwise it is “0”. Is this a function or just a relation?

$$\{(0, 1), (0, 2), (0, 3), ..., (1, 2), (1, 3), ..., (2, 3), ...\}$$

Here the *relation* is *x is smaller than y.* Is this a function or just a relation?

The first was a function, which mapped multiple elements on the left to either 0 or 1. The second was a function which mapped each x to multiple y.

8.3 Relations

We have seen how else we can write functions to save time ($f(x) = blah$), but how else might we write relations (without writing the entire set)?

Well for the "less than" example we might write it the previous way.

$$(x, y) \in R \;\; IFF \; x < y$$

Meaning: The pair is in the *Relation* (remember, we are treating it like a set), *if and only if* x is less than y. Another way of writing this might be:

$$x \; R \; y \mid x < y$$

Meaning: x *relates to* y, *given* (|) that x < y. Really IFF and the given symbol are interchangeable, and you an either say the pair is in the set OR that x relates to y.

If you are having any trouble grasping what a relation is mathematically, consider what it is in the real world.

E.g. "They are my parent"

Parent is a relation, it pairs a person who is the x, you, with another person who is the y, aka one of your parents. How about "Taller than". One person is taller than another, that's a pair. In which case any person will occur multiple times in both the domain and the range (Besides the tallest and shortest person).

For mathematical examples there is another cool way to visualise a relation, especially in order to notice patterns.

$$f(t, v) : t + v = 4$$

$$\begin{array}{cc} & t \\ & \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ v \;\; \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} & \begin{bmatrix} \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & \bullet \\ \circ & \circ & \bullet & \circ \\ \circ & \bullet & \circ & \circ \end{bmatrix} \end{array}$$

The values for t and v are not terribly important, but usually its worth picking enough of them that you can spot patterns.

At the top you notice we are declaring a relation. It could also be written as $(t, v) \in R \mid t + v = 4$. Immediately after that is a matrix with t for the horizontal axis and v on the vertical axis. The filled in dots represent that the corresponding t and v are related. For example $3 \; R \; 1$ because in the above formula $t + v = 4$ is true (i.e. $1 + 3 = 4$). The empty circles represent where t doesn't relate to v ($1 + 1 \neq 4$).

There are a few special properties some relations have that you should be able to spot.

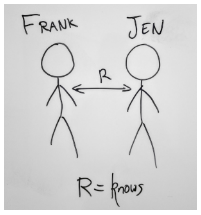


Figure 8.4: Frank knows Jen, Jen knows Frank

8.3.1 Reflexivity

If a pair (t, v) is in R *whenever* $t = v$, then it is reflexive. You could also word this as $t\ R\ v\ IFF\ t = v$.

Putting this into a real-world scenario:

Frank and Jen *might* know each other. However because person A knows person B, it doesn't guarantee person B knows person A; for example Alex knows Kim Jon Un, but that doesn't mean that Kim Jon Un knows Alex.

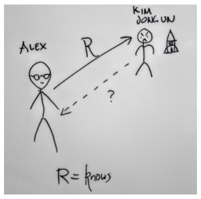


Figure 8.5: Alex knows Kim Jon Un, Kim Jon Un doesn't know Alex

However there is one case when person A will always know person B and person B will always know person A; this is when person A and person B are the same person. This would be the statement "Bianca knows herself". Avoiding the potential can of worms of *can one really know themself*, this statement is universally true.

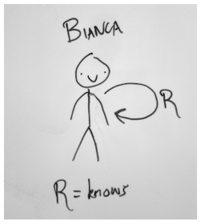


Figure 8.6: Bianca knows herself

A mathematical example of reflexivity would be a statement like $f(t, v) : t + v = 2t$. The relational matrix for this would appear as follows:

$$\begin{array}{cc} & t \\ & \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ v & \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{bmatrix} \bullet & \circ & \circ & \circ \\ \circ & \bullet & \circ & \circ \\ \circ & \circ & \bullet & \circ \\ \circ & \circ & \circ & \bullet \end{bmatrix} \end{array}$$

You can see here that any location where $t = v$ is filled in; meaning that $t\ R\ v$ for those values of t and v. Visually this can be seen as a line which goes along diagonally from top left to bottom right.

8.3.2 Symmetry

Symmetry is stating that if t relates to v then v relates to t. This is expressed as:

$$(t, v) \in R \implies (v, t) \in R$$

Or alternatively.

$$t \, R \, v \implies v \, R \, t$$

You might say something like "You are standing across from me, therefore I am standing across from you", or "They are my cousin, therefore I am their cousin".

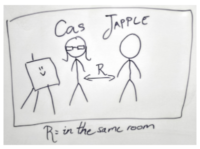


Figure 8.7: Cas is in the same room as Japple, Japple is in the same room as Cas

In this drawing "across from" or "in the same room as" both work. A relation that fits this description is: $f(t, v) : t + v|2$ (AKA $t + v$ is divisible by 2).

$$\begin{array}{cc} & t \\ & \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ v & \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{bmatrix} \bullet & \circ & \bullet & \circ \\ \circ & \bullet & \circ & \bullet \\ \bullet & \circ & \bullet & \circ \\ \circ & \bullet & \circ & \bullet \end{bmatrix} \end{array}$$

If you were to take the diagonal line from earlier and attempt to fold the matrix over it (i.e. Butterfly pattern style), you would find the matrix is symmetric along that diagonal axis. *Is it also reflexive?*

8.3.3 Transitivity

Transitivity is stating that if x relates to y, and y relates to z then x relates to z.

$$x \, R \, y, y \, R \, z \implies x \, R \, z$$

Writing this in the other form of relation notation is left as an exercise to the reader. "Bob is smellier than Fred, Fred is smellier than Susie. Therefore Bob is smellier than Susie" or "The roof is above me, I am above the floor. Therefore the roof is above the floor".

And on the math side of things: $f(x, y) : x < y$

$$\begin{array}{cc} & x \\ & \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ y & \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{bmatrix} \circ & \circ & \circ & \circ \\ \bullet & \circ & \circ & \circ \\ \bullet & \bullet & \circ & \circ \\ \bullet & \bullet & \bullet & \circ \end{bmatrix} \end{array}$$

Is this symmetric or reflexive?

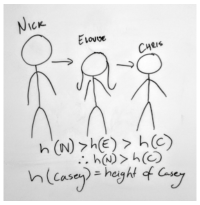


Figure 8.8: Nick is taller than Elouise, who is taller than Chris

8.4 Challenge

1. Write out a relation matrix for $t \leq v$ and state which of the properties it meets.
2. Is "Brother" a reflexive relation? Explain.
3. State a real world example for reflexivity, symmetry, and transitivity which doesn't appear above.
4. Write out two different ways of expressing each reflexivity, symmetry, and transitivity mathematically.
5. State a mathematical example for reflexivity, symmetry, and transitivity which doesn't appear above.

8.5 Reference

Source: https://www.youtube.com/embed/FWg2qZuK3EE Discrete Math - Functions