CS Revision Guide by Jacob

This is done in the order specified in the subject syllabus.

Chapter 13: Data representation

13.1 User-defined data types

Purpose of user-defined data types:

- to match the programmer's exact requirements
- the language may not have a suitable data type
- (therefore) a new datatype is needed

Two types of datatypes: Composite and Non-composite

Non-Composite datatype

Definition: datatypes that do not reference to other datatypes.

Includes:

- built-in datatypes: `int`, `bool`, etc.
- pointers: e.g., `TYPE TIntegerPointer <- ^INTEGER` in pseudocode.
- enumerated data types; e.g., `TYPE TDirections = (North, East, South, West)`

The syntax for defining type aliases is inconsistent even just within my textbook (whether to use `<-` or `=`, or even `:`), so try to match the style/dialect used by the exam paper itself.

Pointer syntax:

```
// Type Aliasing
TYPE TIntegerPointer ← ^Integer
// Declaring a pointer
DECLARE MyIntegerPointer: TIntegerPointer
// Declaring some integers
DECLARE Number1, Number2: INTEGER
Number1 ← 100
// Getting the address of a variable, equivalent to & in C/C++
MyIntegerPointer ← @Number1
// Dereferencing a pointer (i.e., getting the value of the address stored)
Number2 ← MyIntegerPointer^ * 2 // Number2 is now 200.
```

Note that unlike C/C++, dereferencing pointers use a postfix operator.

Enumerated Data Type Syntax:

```
// Type Aliasing, notice the '=',
// this is the inconsistency I was talking about.
TYPE TDirections = (North, East, South, West)
// Declaring an enumerated variable
DECLARE ThisDirection, NextDirection: TDirections
// Assigning values
ThisDirection ← North
// Enumerated data types are implicitly ordinal.
NextDirection ← ThisDirection + 1 // NextDirection is now East.
// Shorthand for enumerated integer values
DECLARE TStudentID: 1..3000
```

Composite datatype

Definition: a datatype that has a definition with reference to at least one other type.

Includes:

- record data type. There may be built-in record data types, but typically the language will allow the user to define their own. Note that Python does not have a special keyword for record datatypes. Use classes with public attributes instead; or, if you are not taking an exam, you can learn to use a the `dataclass` metaclass.

  ```
  TYPE SomeRecord
      DECLARE Field1: INTEGER
      DECLARE Field2: STRING
  ENDTYPE
  ```

- class. They only exist in OOP languages. The language most likely would have many built-in classes, but the user can define their own.
- set. A set contains a collection of unordered and unique data values. Allowed operations include (but are not limited to):
  - checking if an element is a member of the set
  - adding a new element to the set
  - removing an element from the set
  - union, intersection, difference, symmetric difference
- array/list. Indexed collection of items. Arrays contain elements of the same type; lists contain elements of different types.

13.2 File Organization

3 types of file organization: Serial, Sequential, Random/Direction

Serial Files

Records are appended to the end of the file, so entries are automatically in chronological order. Useful when the data entry *should* be sorted in the time when the data was input, or for archiving purposes (data is unlikely to be used again).

The file can only be read in order, from start to finish.

Examples: Bank records, Electricity meter readings, etc.

Sequential Files

Records are stored in increasing order of a key field. (Each record insertion is like one pass in insertion sort.)

Best for long-term storage of data with a clear key field. E.g., Subscription services can keep track of their users by setting up a sequential file where the customer ID is the key field.

Normally the file can only be read in order, from start to finish.

Random Files

AKA direct access files.

Stores a record on some available location. The location is determined by using a hashing algorithm on the key of a record.

It works just like a hash table.

- Open Hashing: the records are stored outside the hash table, for example using a linked list (each element in the hash table points to the head of a linked list). This is hard to implement in files, so most files use a closed hashing system
- Closed hashing: Should a collision occur, a probing function is used to look for a suitable space. E.g., linear probing: find the next empty space.

Procedure:

1. The key is hashed to get an initial location
2. If the location is empty, put the record at that location. We are done.
3. Otherwise, look for the next location. Goto Step 2.
4. If every location is found to be full, insertion fails. (File is full.)

3 Types: when to use which

- Serial: well suited for batch processing or backing up data on media (e.g., magnetic tape)
- Sequential: when there is a key, and the data needs to remain in order of that key.
- Random/Direct-Access: when data is frequently read, written to, or deleted.

13.3 Floating-pointer numbers

The gist: express the number in scientific notation with base 2.

Number = $M \cdot 2^E$, where M is the mantissa and E the exponent.

E is represented as a conventional two's complement integer, i.e., the most significant bit is negative.

M is represented as a two's complement decimal, i.e., the most significant bit represent -1, and each bit after it represents 2^{-k}.

A floating point number is normalized if and only if the mantissa starts with 01 or 10.

More bits in mantissa $\longrightarrow$ more precision (significant digits)

More bits in exponent $\longrightarrow$ larger range of representable numbers

Chapter 14: Communication Technologies

14.1: Protocols

What is a (network) protocol: a set of agreed-upon rules that govern how data is transmitted over a network.

A protocol stack is a set of protocols with the following properties:

- Each layer only accept input from the next higher layer or the next lower layer.
- Action between adjacent layers is well defined and constitutes only the interaction between the layers.
- With the possible exception of the lowest layer, all other layers are made functioning by software.
- User interacts directly with the topmost layer
- Hardware interacts directly with the lowest layer (and any layer that interacts directly with hardware must be the lowest layer)

The TCP/IP Protocol Suite

Our book uses a 4-layer model: Application, Transport, Network, and (Data)Link. (in the order of top to bottom of the stack). Mnemonic: All Tea Is Lovely.

Application Layer

- HTTP: Hypertext Transfer Protocol
  The protocol underpinning the world wide web. For example, It is used when fetching an HTML document from a web server.
  When the user keys in the URL, HTTP add its own headers and assemble a request. It then passes the request to TCP.
- FTP: File Transfer Protocol
  For transferring files over a network. FTP also allows anonymous access and commands to be used on an FTP server.

- POP3: Post Office Protocol
  Pull protocol for receiving emails. The client periodically connects to the server, checks for and downloads new emails. The downloaded emails are then deleted from the server. This is arguably safer than IMAP as the emails are not stored on IMAP.
- IMAP: Internet Message Access Protocol
  Pull protocol for receiving emails. The client periodically connects to the server, checks for and downloads new emails. The downloaded emails are NOT deleted from the server.
- SMTP: Simple Mail Transfer Protocol
  Text-based protocol for **sending** emails. It's a push protocol, where emails are pushed onto the server by the client. Normal SMTP is text-based only and cannot handle binary files. (images, etc.)
- MIME: Multi-purpose Internet Mail Extension
  It is an extension to the standard email protocols. It allows binary attachments to be sent.
- BitTorrent:
  For peer-to-peer file sharing.
  Suppose Alice wants to share a file with a number of different peers.
  - First, Alice creates a (small) **torrent** file that contains metadata about the file.
  - The file is broken up into equal-sized chunks known as pieces.
  - Peers wishing to download the file first acquires the torrent file and connect to the appropriate tracker.
  - The **tracker** is a central server that contains data on which peers hold which pieces/files.
  - Once a peer has downloaded a piece/file, the piece/file is made available to other peers in the swarm. This peer becomes a **seed**. A **swarm** is a group of peers connected together.
  - A **leech** is a peer who uploads significantly less than it downloads, maybe by logging off once the transfer is complete.
  - A **lurker** is a peer that downloads many files, but does not make available any *new* content for the community.

Transport Layer: TCP/UDP

TCP sends data in segments. TCP ensures that every packet is delivered. If a packet is not acknowledged, it is re-sent.

UDP works differently: it forgoes reliability in exchange for speed. Packets are not acknowledged. Its advantage is that it is fast and lightweight.

13.2 Circuit switching and Packet switching

Circuit switching

A dedicated channel/circuit is set up between the sender and receiver.

Pros:

- The circuit is dedicated to this single transmission $\implies$ Security
- The entirety of the bandwidth is available $\implies$ Higher speeds

- Packets (Frames) of data arrive in the same order as they were sent $\Longrightarrow$ No need to perform extra sorting
- Packets cannot get lost because they all follow the same and only route. $\Longrightarrow$ No need for re-transmission.
- Works well for real-time applications

Cons:

- Not flexible: It has to use a single, dedicated line.
- The line is wasted when it is idle: nobody else can use it. / The circuit is there no matter if it is used
- If there is a failure on the line, there is no alternative routing.
- Dedicated channels require greater bandwidth
- Time to establish such a line can be long.

Packet Switching

A message is broken up into small packets. Each packet is sent independently to each other from the sender to the receiver. Each packet follows its own path.

The path is determined by the router the packet goes to. For each hop, the router determines a shortest path for the packet and sends it to the next router.

The router uses routing tables to determine which path a packet should take. A routing table contains metrics (a cost assigned to each route so that the most efficient route can be found)

To avoid packets being sent infinitely and clogging up the system, a **hop number** is added to the **header** of the packet. Each time a packet goes through a router, the hop number decreases by 1. When the hop number=0, the packet is deleted.

A header contains the following values:

- Source IP
- Destination IP
- Hop number
- Length of packet
- Sequence number (needed for reassembling the packets)

Pros:

- No need for a dedicated line
- Failures and faults can be overcome by simply re-routing packets
- Packet switching charges the user only for the duration of connectivity (circuit switching charges according to distance and duration)
- High data transmission is possible
- Packet switching always transmits digital data

Cons:

- More complex protocols
- Packets can get lost and have to be re-sent
- Not good for real-time data streams because receiver has to re-assemble the packets
- Bandwidth is shared with other packets

- Extra delay caused by packets arriving not in order and having to be re-assembled
- High RAM usage when handling lots of data

Chapter 15: Hardware and Virtual Machines

15.1 Processors, Parallel Processing and VMs

RISC and CISC processors

RISC: Reduced Instruction Set Computer

CISC: Complex Instruction Set Computer

Feature	CISC	RISC
No. instructions	More	Fewer
Instruction formats	More	Fewer
Addressing modes	More	Fewer
Number of clock-cycles an instruction can use	Single/Multi	Single
Length of instructions	Variable	Fixed
Execution time for 1 instruction	Longer	Shorter
Ease of decoding instructions	More complex	Simpler
Ease of pipelining	Harder	Easier
Number of registers	Fewer	More
Design emphasis is on ?	Hardware	Software
Processor complexity (Amount of transistors)	Complex (More)	Simple (Fewer)
Control Unit	Microprogrammed	Hard-wired

CISC processors have more built-in instruction formats. Code for CISC processors are shorter, because each instruction can do more advanced things. However, the processor may take more time to execute each instruction. (Vice versa for RISC)

Pipelining

Typically, an instruction is executed in several stages. Each stage uses a different part of the processor.

Without pipelining, to execute instructions A and B, the processor is used like this: (Assuming 5-stage execution)

```
Clock Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |10 |
Stage 1     | A |   |   |   |   | B |   |   |   |   |
Stage 2     |   | A |   |   |   |   | B |   |   |   |
Stage 3     |   |   | A |   |   |   |   | B |   |   |
Stage 4     |   |   |   | A |   |   |   |   | B |   |
Stage 5     |   |   |   |   | A |   |   |   |   | B |
```

With pipelining, the processor can be used much more efficiently.

```
Clock Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |10 |
Stage 1     | A | B | C |   |   |   |   |   |   |   |
Stage 2     |   | A | B | C |   |   |   |   |   |   |
Stage 3     |   |   | A | B | C |   |   |   |   |   |
Stage 4     |   |   |   | A | B | C |   |   |   |   |
Stage 5     |   |   |   |   | A | B | C |   |   |   |
```

How pipelining speeds up processing: by reducing idle time of the different stages in the processor.

Interrupt handling

In a pipelined system, interrupts are more complex. The two most simple options are listed here:

1. Erase all pipeline contents for the latest 4 instructions to have entered the pipeline. Then the normal interrupt handling routine can work on the remaining instruction. This means those erased instructions have to be carried out again.
2. Create an individual unit for each pipelined instruction with its own program counter register(s). This does not discard any data that has already been calculated.

Parallel Processing

The 4 basic architectures

1. SISD: single instruction single data
   A single processor does a executes a single instruction on a single data source.
   This does not support parallel processing.
2. SIMD: single instruction multiple data
   Multiple processors each execute the same instruction, but on different data sources.
   They are useful for graphics processing, as each pixel is one data source.
3. MISD: multiple instruction single data
   Multiple processors execute different instructions on the same data source.
   This is not common. Typically MIMD is used instead.
4. MIMD: multiple instruction multiple data
   There are multiple processors, each of which can execute different instructions and on different data sources.
   Most modern multi-core computer systems use this architecture.

Massively parallel computers

Multiple computers linked together by a powerful (low-latency) backing **communications infrastructure**. Each computer carries out part of the computation and communicates with each other using the infrastructure.

Virtual Machines

Virtual Machine: an emulation of an existing computer system using software. A computer OS running within another computer's OS.

Emulation: the use of application/device to imitate the behavior of another program/device. E.g., Running an OS on a computer which is not normally compatible.

Host OS: the OS controlling the physical hardware.

Guest OS: the OS being run on a virtual machine.

Hypervisor/Virtual Machine Software: Virtual machine software that creates and runs VMs.

Advantages of a VM:

- Security: Anything running inside the guest OS cannot affect the host OS. The host computer is protected by the Hypervisor. (Use: to test possibly malicious software)
- Cheap/Compatibility: Software not compatible with the new hardware/system can be emulated by running a compatible OS in a VM. $\implies$ No need to buy extra hardware (Use: to test new systems)

Disadvantages of a VM:

- Performance loss: the performance in a VM is often significantly lower than in the original system.
- They can be complex to manage and maintain on a large scale.

15.2 Boolean Algebra and Logic Circuits

This section assumes you are comfortable with symbols of common basic logic gates.

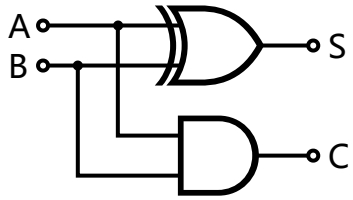
TRUE = 1; FALSE = 0

Laws:

Name	OR version	AND version
Commutative	$A+B=B+A$	$A.B=B.A$
Associative	$(A+B)+C=A+(B+C)$	$(A.B).C=A.(B.C)$
Distributive	$A+B.C=(A+B).(A+C)$	$A.(B+C)=A.B+A.C$
Idempotent	$A+A=A$	$A.A=A$
Identity	$A+0=A$	$A.1=A$
Null	$A+1=1$	$A.0=0$
Inverse	$A+\overline{A}=1$	$A.\overline{A}=0$
De Morgan	$\overline{A+B}=\overline{A}.\overline{B}$	$\overline{A.B}=\overline{A}+\overline{B}$

Half adder & Full adder

Half Adder

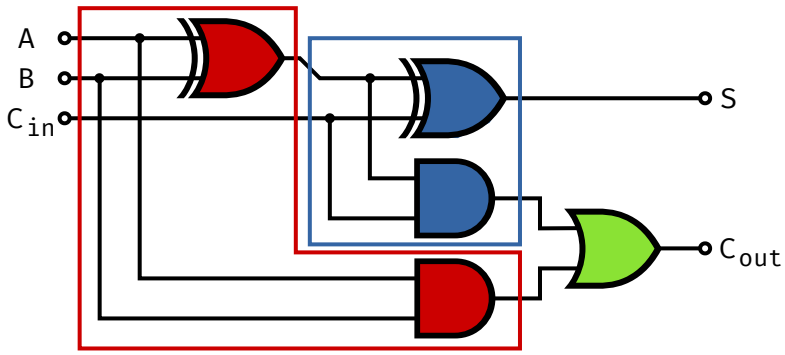


S = A XOR B; C = A AND B

Truth table:

INPUT		OUTPUT	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Full Adder

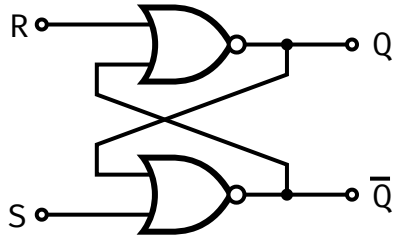


C_{out} and S form a binary number whose value is equal to the sum of A, B and C_{in}.

INPUT			OUTPUT	
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Flip-flop circuits

SR Flip-flop



R stands for reset; S stands for Set.

When R and S are both 0, the circuit retains its state. (Does not change)
When R is 1 and S is 0, Q is *reset* to 0. (NOT Q is 1)
When R is 0 and S is 1, Q is *set* to 1. (NOT Q is 0)

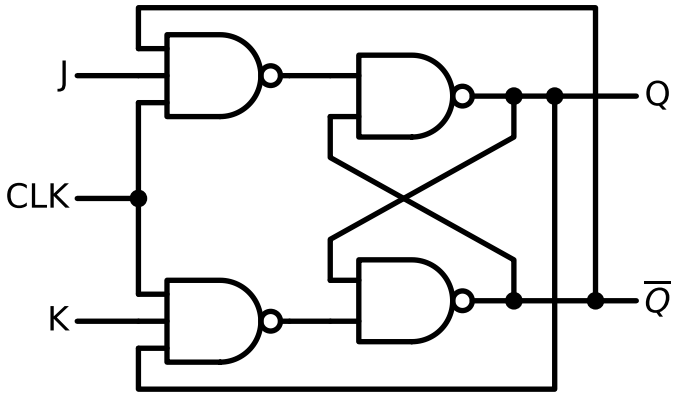
The state in which R and S are both 1 is considered illegal. This is because both Q and NOT Q are 0.

(When R and S are set back to 0 simultaneously, the circuit enters a race condition, and the state of Q and NOT Q is not definite.)

JK Flip-flop

This is invented to overcome the problems of a SR flip-flop.

- Invalid S,R conditions have to be avoided
- If inputs do not arrive at the same time, the flip-flop can become unstable.



For each time CLK goes from 0 to 1, Q and NOT Q are adjusted according to J and K.

If J and K are both 0, outputs do not change.
IF J is 1 and K is 0, Q is set to 1, (NOT Q is 0)
IF J is 0 and K is 1, Q is set to 0, (NOT Q is 1)
IF J and K are both 1, Q and NOT Q is toggled. (i.e., 0 becomes 1, 1 becomes 0)

The flip-flop changes state ONLY at the moment when CLK signal goes from 0 to 1.

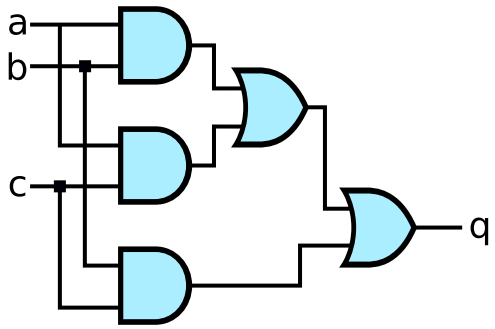
Logic circuits and Boolean Algebra

This section assumes you are familiar with how to simplify a Boolean expression.

There are two ways to find the Boolean expression of a logic circuit:

1. Start from the input side and write down the logic expression for each logic gate.
2. Draw the truth table of the diagram and apply the sum-of-products method.

Below is an example of such an exercise:



Using method 1:

- The left three AND gates are respectively: (A AND B), (A AND C), (B AND C)
- The OR gate in the middle column is therefore: (A AND B) OR (A AND C)
- Q is henceforth (A AND B) OR (A AND C) OR (B AND C)
  i.e, Q=A.B+A.C+B.C

Using method 2: Q=A.B+A.C+B.C+A.B.C

Inputs			Output
A	**B**	**C**	**Q**
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Karnaugh Maps (K-maps)

K-maps are a way of producing a simplified Boolean expression from a truth table.

K-maps that can be written out on a sheet of paper can only handle 4 variables.

Below is a checklist for creating a K-map and obtaining a Boolean expression:

1. Construct a table (4×2 if 3 variables, 4×4 if 4 variables), where each side represent all combinations of 1 or 2 variables. Adjacent combinations shall only differ by one variable. Below are the classic arrangements (you can't be wrong with these).

		AB			
		00	01	11	10
C	0				
	1				

		AB			
		00	01	11	10
CD	00				
	01				
	11				
	10				

2. Fill in the 0s and 1s according to the truth table.
   Below is the filled-in K-map of the truth table in the previous section

		AB			
		00	01	11	10
C	0	0	0	1	0
	1	0	1	1	1

3. Find all rectangles that satisfy these properties:
   - All values in the rectangle are 1s.
   - The rectangle can only have side lengths 1, 2, 4, 8, etc. (A 1×1 rectangle is allowed)
   - The rectangles can overlap.
   - The rectangle can wrap around the edges.
   - Each rectangle should be as large as it can be.
4. Start from the largest rectangle: find the terms all of its elements have in common, add to the answer the product of those common terms.

   The answer for the K-map above is: A.C+B.C+A.C

PLEASE READ TEXTBOOK PAGE 360-362 CAREFULLY IF YOU HAVE ANY DOUBTS. IF YOU HAVE ANY QUESTIONS AT ALL, GO ASK SOMEBODY FOR HELP.

Chapter 16: System Software

OS: operating system.

16.1 Purposes of an OS

Startup procedure of a computer

1. Power is switched on
2. The BIOS (normally stored on ROM) starts a **bootstrap** program.
   Bootstrap: A small program that is used to load other programs to start up a computer
3. The bootstrap loads parts of the OS from the hard drive into RAM and initiates the start-up procedures.
   - On a phone, the flash drive is split into two parts: The part where the OS resides, and the part where apps and their data are stored. The

bootstrap loads only the OS part.

- On a computer, there is no strict boundary between the OS and other programs. (They are all stored on the same disk partition) Therefore the bootstrap loads only the portion of the OS responsible for the startup. The OS then takes over the start-up procedure.

How an OS maximize the use of resources

Resource management is normally split into 3 areas: CPU, Memory, and Input/output system management.

High-level scheduling decides which programs are loaded from **disk** (I/O devices) into RAM.
Medium-level scheduling decides which programs are to be *temporarily* swapped out of **RAM**.
Low-level scheduling decides which programs get to use the **CPU** (i.e. to be executed). Also known as CPU scheduling or process scheduling.

I/O is much slower than the CPU, so the OS needs to ensure that the CPU is not idle when I/O is taking place.

The syllabus places the emphasis on low-level scheduling, and does not cover the strategies used by the other two levels.

Kernel: The central part of the OS responsible for communication between hardware, software and memory. It is responsible for process management, device management, memory management, interrupt handling and input/output file communications.

How the OS hides complexity from users:

- Uses GUI rather than CLI
- Device drivers (provides simple interfacing with hardware)
- Simple process for reading data from and writing data to memory and storage devices
- Can carry out background utilities (e.g., virus scanning). The user doesn't need to carry these out themselves.

Process Management

Multitasking

Definition: function allowing a computer to process more than one task/process at a time.

Process: a program **that has started** to be executed.

The kernel uses a scheduling algorithm to manage which processes get to use the CPU. (Low-level scheduling)

Two types of scheduling algorithms:
(TL;DR: Preemptive scheduling interrupt processes; non-preemptive ones don't.)

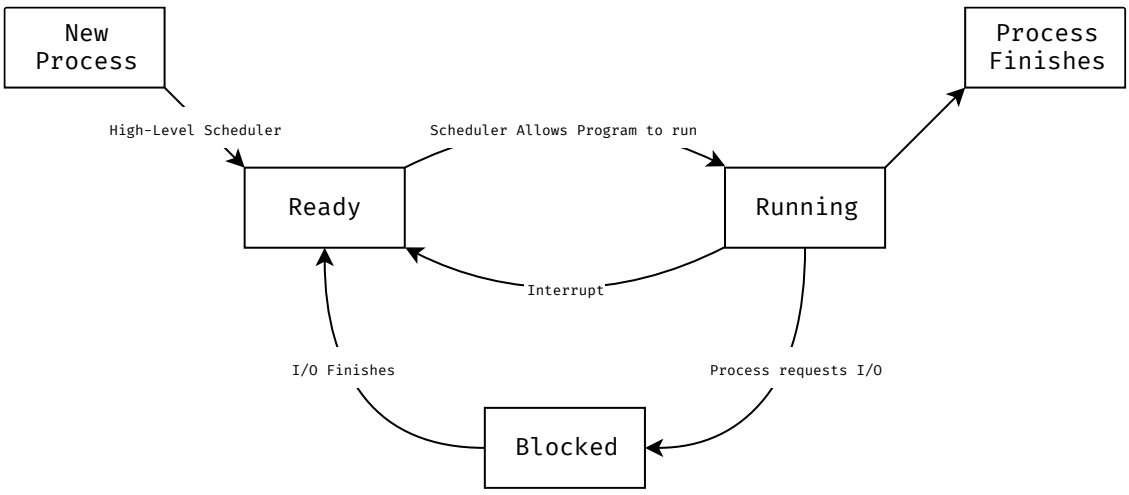
Preemptive	Non-preemptive
Resources are allocated for a limited time	Resources are held by the process until it completes its burst time or switches to the waiting state
The process can be interrupted	Processes are not interrupted. They either finish or switch to the waiting state
Starvation is caused by frequent arrivals of high-priority processes	Starvation is caused by processes with long burst times
More flexible	More rigid

Process scheduling is controlled using **Process Control Blocks** (PCBs), which include:

- Process ID: a unique ID for the process
- State: One of Running, Ready, or Blocked (Waiting)
- Register states: the values each register holds as the process is being executed
- Memory information: amount of memory allocated to the process
- I/O information: list of I/O devices associated with the process
- Priority: How important it is that the process be executed as soon as possible.

At any time, a process can be in 1 of three states: Ready, Running or Blocked (Waiting).

- Ready state: the process is ready to be executed, that is, it has already finished required I/O.
- Running state: the process is using CPU and being executed.
- Waiting state / Blocked state: the process is waiting for I/O and is not ready to be executed.



Process scheduling Algorithms

1. First come first served (FCFS)

   Non-preemptive.
   The process added to the queue first finishes first.

   This is the simplest scheduling algorithm. If 4 processes P1, P2, P3 and P4 are added to the ready queue in this order, the CPU would execute P1 until it finishes, then P2 until it finishes, and so on.

2. Shortest job first (SJF)

   Non-preemptive.
   When the CPU has finished the current process, the shortest process is executed.

   Note: Other processes are not checked until the current process is finished.

3. Shortest remaining time first scheduling (SRTF)

   Preemptive.
   Each time a process is placed in the ready queue, the process with the shortest remaining burst time is executed.

   Note: The process with the shortest remaining burst time is not necessarily the one that's being executed. This means the process being executed may be interrupted.

4. Round robin

   Preemptive.
   A fixed time slice (usually around 5ms) is given to each process. This time slice is known as a **quantum**.

   Each time a process executes for at most a quantum. If it does not finish or switch to the blocked state before the quantum ends, it is interrupted and placed at the end of the ready queue.

Interrupt handling and OS Kernels

The CPU checks for interrupt signals. The system calls interrupt handling if any of the following types of interrupt signals are sent:

- Device interrupt (e.g., printer out of paper, device not ready, etc.)
- Exceptions (e.g., division by zero, segmentation fault, etc.)
- Traps / Software Interrupt (e.g, software requesting resources from disk)

When such a signal is received, the kernel consults the **interrupt dispatch table (IDT)**.

Interrupt dispatch table: a table that implements an interrupt vector table, which links types of interrupts with their corresponding handlers.

Interrupts are prioritized using interrupt priority levels. Only interrupts with priorities higher than the current task would cause the current task to be interrupted.

The procedure for handling an interrupt is as follows:

1. An interrupt is received (Other interrupts are disabled)
2. The state of the current process is saved on the kernel stack
3. The source of the interrupt is identified. The priority of the interrupt is checked.

4. The system jumps to the interrupt service routine using the IDT to either rectify the error and/or alert the user that an error has occurred.
5. Once completed, the state of the interrupted process is restored from state stored on the kernel stack. The process is then free to continue.
6. Interrupts are restored (re-enabled).

Memory Management

We obviously can't allocate *all* of the available memory to one process. Therefore we need to split the memory up into chunks. There are two ways to do this: paging and segmentation.

Paging

The memory is split up into fixed-size partitions (blocks).

The physical memory blocks are called **frames**; logical memory blocks are called **pages**.
(physical memory blocks refer to the actual location on the RAM; logical memory blocks refer to the addresses that a program uses in its logic/calculations)

A **page table** stores information such as page number, flag status, page frame address and time of entry.
Important flag statuses include: if a page has been accessed; if the page has been modified (the 'dirty' flag); if the page is present in memory.

To access a memory location, the user (program) supplies a single (logical) address.

The page table maps this address to the physical address so the data can be retrieved.

Segmentation

The memory is split up into variable-size partitions/blocks called segments.

Each segment has a name and size.

To access a memory location, the user (program) supplies two values: one to identify which segment the data is in, and the other one to specify the offset of the requested address, relative to the start of the segment.

A **segment map table** is used to map the supplied address to the physical memory. The segment map table stores information of each segment: the segment number, the size of the segment, and the physical address of the starting location of this segment.

Paging	Segmentation
Fixed-size blocks	Variable-size blocks
Blocks may be unfull Internal Fragmentation	Less risk of internal fragmentation More risk of external fragmentation
A single value needed for access	Two values are needed
Page table maps logical to physical addresses Contains base address of each page frame	Segment map table maps logical to physical addresses Contains segment number + segment size
Paging is essentially invisible to the programmer The system handles everything	Segmentation is very visible to the programmer Programmer needs to manually manage segments
Procedures and associated data cannot be separated	Procedures can be separated from their data
Static linking (Procedure and data are together) Dynamic loading (Page may be loaded from disk during execution)	Dynamic linking (Data is loaded during execution) Dynamic loading (Segment may be loaded from disk during execution)

Virtual memory

Definition: A type of paging that gives the illusion of unlimited memory being available.

Essentially, RAM = physical memory; RAM + swap space = virtual memory.

Unloading a page = moving a page from RAM to the swap space (HDD/SSD)
Loading a page = copying a page from swap to RAM

Swap space: space on the HDD/SSD used in virtual memory.

In-demand paging: A form of data swapping where pages loaded from disk into RAM only at the time when they are required.

The RAM may not be large enough to store all the pages, so some pages are stored in the swap space.

Pros:

- Programs can be larger than the RAM and can still be run
- More efficient multi-programming with less I/O loading and swapping programs into and out of memory (Imagine without paging, the programmer would have to divide the program into smaller pieces and load each one manually, which may be inefficient.)
- No need to waste memory with data not being used (the data can be unloaded from RAM and put back into disk)
- eliminates external fragmentation / reduces internal fragmentation
- cheap: no need to buy more expensive RAM

Cons: **Disk thrashing**: when pages are being excessively swapped in and out of RAM. This leads to a high rate of hard disk read/write head movements and causes the system to slow down.

Thrash point: the point at which the system is so busy loading and unloading pages that execution halts.

Procedure for reading from memory with virtual memory enabled:

1. The program requests for data at a virtual address
2. The system translates the address to a physical address.
3. If this physical address is not in main memory, the OS loads it from the hard drive (swap space)
4. The data is read from RAM using the physical address and returns the data to the program.

Page replacement algorithms

Pages may not be on RAM. Page replacement algorithms determine which pages are and which are not on RAM.

Page fault: when a requested page is not in RAM and needs to be loaded from the hard drive.

When the RAM is full, the OS needs to decide which page to swap out in order to load the requested page.

Algorithms to determine which page(s) to replace:

1. First in first out (FIFO):

   The OS tracks all pages with a queue structure. The oldest page is at the front and is the one to be removed when a new page is loaded.

   This algorithm does not consider how often a page is used: a page may be swapped out simply because it arrived earlier.

2. Optimal page replacement:

   This is a theoretical best-case algorithm. It looks forward in time to see which frame it can replace.

   This is impossible to make, we can't predict accurately which pages can be unloaded.

3. Least recently used page replacement:

   The page that has not been used for the longest time is replaced.

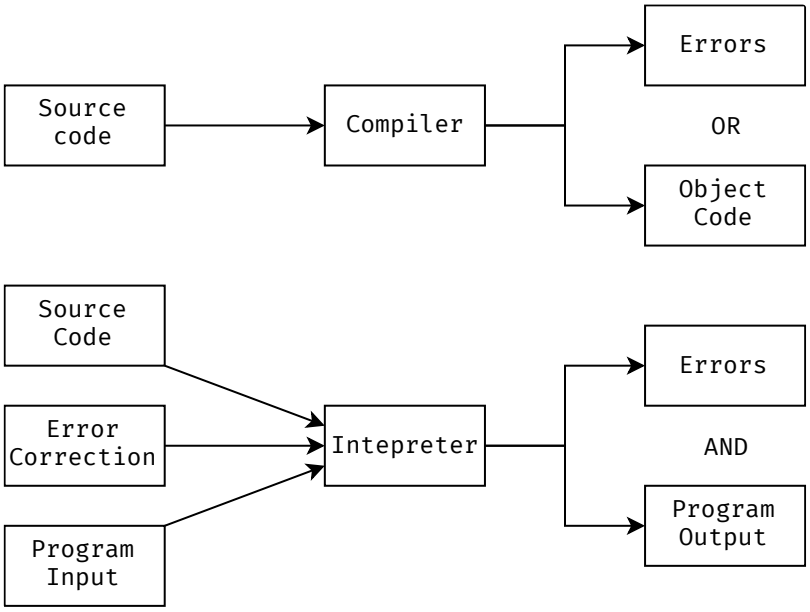
16.2 Translation software

Assemblers, Interpreters and compilers

An assembler translates assembly code into machine code that the CPU can understand.

Interpreters execute high-level languages without producing a full translated version. They do this by reading the program line-by-line and executing each line on the go.

Compilers translate high-level languages into a translated version called object code.



Stages in the compilation of a program

The stages are: **lexical analysis**, **syntax analysis**, **code generation** and **optimization**.

Lexical analysis

The program is converted into tokens using a keyword table and a symbol table.

A keyword table stores a keyword/symbol and a corresponding token in each entry. This does not change for a particular language. An example is shown below.

Symbol/Keyword	Token
←	01
&	02
-	03
...	
INTEGER	31
INPUT	32
OUTPUT	33
...	...

A symbol table stores each identifier/constant and their corresponding token, type (constant or variable) and datatype. The symbol table is different for each program. An example of a symbol table is shown below.
Note that the value column (shown in red) is NOT the value the symbol holds. It is the value of the token that represents the symbol.

	Token		
Symbol	**Value**	**Type**	**Datatype**
myInteger	81	Variable	Integer
myString	82	Variable	String
"This is a literal string"	83	Constant	String

Each symbol and keyword is substituted with its value. The output of the lexical analysis stage is a sequence of token values.

For instance, the program `myString <- myString & "This is a literal string"` is converted by lexical analysis into `82 01 82 02 83`.

Syntax analysis

The sequence of tokens is then analyzed for grammatical / syntax errors.

The rules can be set out in Backus-Naur Form (**BNF**) notation. (See next section)

Code generation

The code generation stage produces an object program. The previous 2 stages must pass without error to reach this stage.

The object program is no longer readable by humans. It is either machine code that can be read by the CPU directly, or it may be in an intermediate format that is converted to machine code when it is loaded.

Optimization

Optimization is the process of making the object code more efficient whether in terms of time, storage space, memory or CPU usage (or a combination of those factors).

An example is shown below:

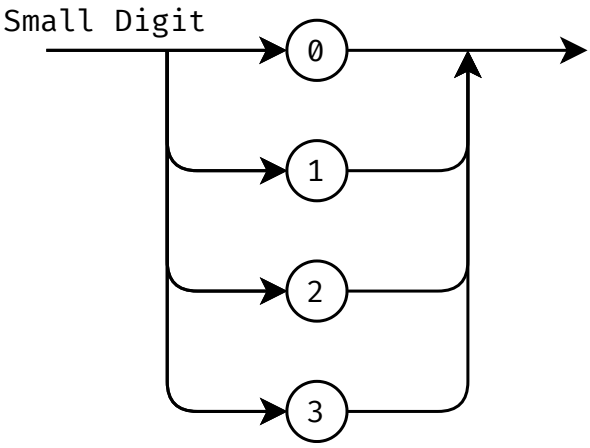
```
Original Code: w=x+y; v=x+y+z
Object Code:                Optimized Object Code:
LDD x                       LDD x
ADD y                       ADD y
STO w                       STO w
LDD x                       ADD z
ADD y                       STO v
ADD z
STO v
```

This example is included because the exam may ask you to perform simple optimizations like this.

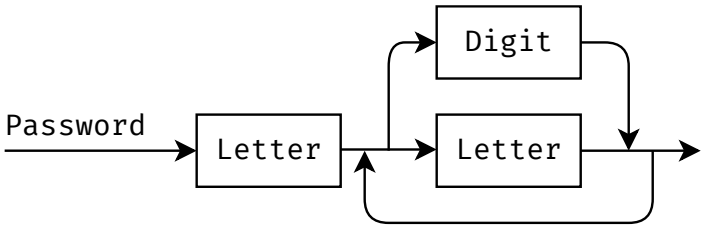
Syntax diagrams and Backus-Naur form

This section of the Hodder Education book is well written. If you want detailed information, go there. Otherwise, here are some syntax diagrams and their BNF just as a refresher.

```
<small_digit> ::= "0" | "1" | "2" | "3"
<letter> ::= "A" | "B" | "C" // diagram for letter is omitted
```




```
<password> ::= <password><letter> | <password><digit> | <letter>
```



Reverse Polish Notation (RPN)

Definition: a way of representing an arithmetic expression without the use of brackets or special punctuation.

Also known as: postfix expressions.

Normally, when we write math, we write infix expressions: the operator in between the two operands, like `A + B`

In RPN: `A + B` is written as `A B +`

More complex examples of turning infix expression into RPN is below (RPN is enclosed in "" for clarity:

```
  A - (B+C)*D
= A - "B C +" * D
= A - "B C + D *"
= "A B C + D * -"
  A * B + C * (D*E + F)
= A * B + C * ("D E *" + F)
= A * B + C * "D E * F +"
= A * B + "C D E * F + *"
= "A B *" + "C D E * F + *"
= "A B * C D E * F + * +"
```

The procedure for convert infix to postfix is as follows:

1. Follow the normal arithmetic precedence: in the above expression, what do you calculate first?
2. Convert this into postfix (`A + B` becomes `A B +`), mark them so you know that part is postfix. Now this part can be treated as one single item.
3. Repeat until everything is in postfix

The procedure for calculating a postfix expression is as follows:

1. Make an empty stack
2. Scan each item from left to right:
    1. If the item is a number, push it into the stack
    2. If the item is an operator, take the top 2 numbers and apply the operator on the 2 numbers. Push the result back into the stack
3. After the scan is finished, the stack should only contain 1 item. This item is your answer.

An example of this is given here:

```
"A B * C D E * F + * +"
Item: A    Stack: A
Item: B    Stack: A, B
Item: *    Stack: [A*B]
Item: C    Stack: [A*B], C
Item: D    Stack: [A*B], C, D
Item: E    Stack: [A*B], C, D, E
Item: *    Stack: [A*B], C, [D*E]
Item: F    Stack: [A*B], C, [D*E], F
Item: +    Stack: [A*B], C, [D*E + F]
Item: *    Stack: [A*B], C * (D*E + F)
Item: +    Stack: (A*B) + C * (D*E + F)

Result: (A*B) + C * (D*E + F)
```

This looks horrifying simply because I used letters instead of numbers.

Chapter 17: Security

17.1 Encryption, Encryption Protocols and Digital Certificates

Encryption basics

Purposes of encryption

1. **Confidentiality**: Only the intended recipient can understand the message.
2. **Authenticity**: It is possible to identify who sent the message and verify the source is legitimate.
3. **Integrity**: Data should reach data without changes. (If the data is modified, it should be noticeable)
4. **Non-repudiation**: neither party can deny that they were part of the data transmission.

Eavesdropper: a malicious entity trying to spy on a conversation.

The modern network infrastructure cannot prevent eavesdroppers from getting the transmitted data, so encryption is used to make sure that eavesdroppers cannot understand the data.

Plaintext and ciphertext

Plaintext: the original unencrypted data to be sent / being sent

Ciphertext: The resulting encrypted data from the plaintext going through the encryption algorithm.

```
ciphertext = encrypt(plaintext, key_encrypt)
plaintext = decrypt(ciphertext, key_decrypt)
```

The keys are in pairs.

I.e., If a message is encrypted with a certain encryption key, it can only be decrypted with the corresponding decryption key.

Symmetric and Asymmetric encryption

If the keys to encrypt and to decrypt the message are the same, the encryption is **symmetric**. This key is called the **secret key**.

If the keys to encrypt and to decrypt the message are not the same, the encryption is **asymmetric**.

In a symmetrically encrypted conversation, both parties have to have the same key.

They cannot transmit the key in plaintext. In most cases asymmetric encryption is used for transmitting the key.

In an asymmetrically encrypted conversation, both parties have a **private key** and corresponding **public key**. The public key is public and can be sent over the internet as plaintext (without encryption).

The public key is used for encrypting a message.
The private key is used for decrypting a message.

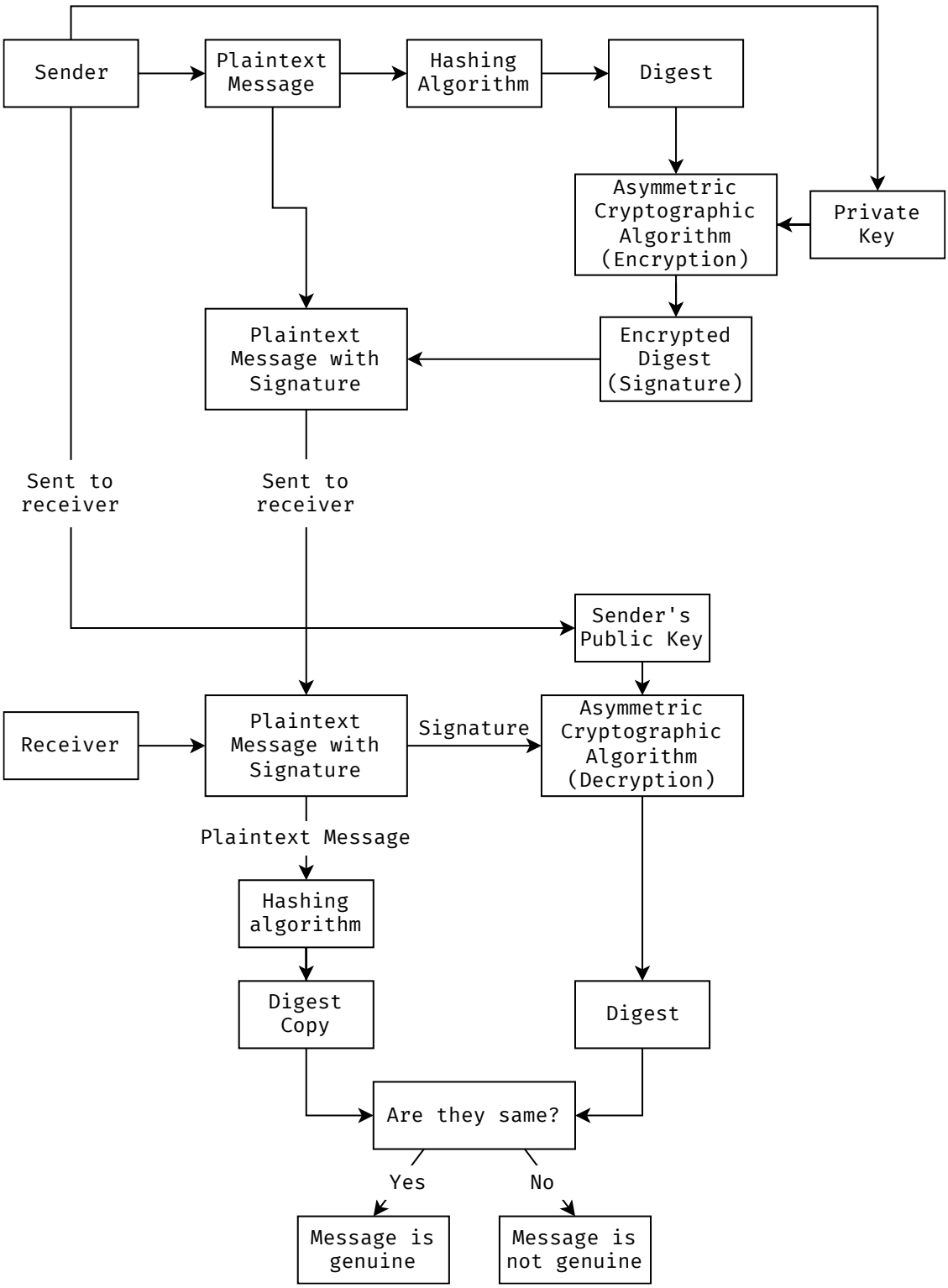
Therefore, if Alice wants to send Bob a message,

1. Bob first shares his public key.
2. Alice can then encrypt the message with Bob's public key and send the ciphertext.
3. Bob can then decrypt the message using his private key.

There is one more problem: how can Alice be sure that it was really Bob who shared his key? A third party can still forge public keys. To solve this we need to introduce digital signatures and certificates.

Digital signatures and digital certificates

A **digital signature** is an electronic way of validating the authenticity of digital documents and proving that the document was sent by a known user. The process is listed below



Note that a digital signature alone does not solve the problem: a third party can still forge the public key.

This is why digital certificates are needed. A **digital certificate** is an electronic document used to prove the identity of a website or individual. In contains the public key and information identifying the website owner or individual. It is issued by a CA (Certificate authority).

A **certificate authority** is a commercial organization used to generate a digital certificate requested by website owners or individuals.

Put simply, the website owner / individual submits an application to the certificate authority. The application must contain the public key and other information identifying the applicant. The CA then generates a digital certificate. The certificate contains:

- Name of the certificate issuer
- Name of entity the certificate is issued to
- The public key of the entity
- Start and expiry dates of the certificate
- The digital signature signed by the CA

Each computer has pre-installed a list of CAs and their public keys, so any certificate can be easily verified.

As the certificate contains the public key, we can know if a third party forged the key or not.

SSL and TLS

SSL: secure sockets layer

TLS: transport layer security

The purpose of SSL and TLS is to ensure that communication between two parties is confidential and authenticated.

TLS is the upgraded version of SSL.

- It separates the record protocol from the handshaking process.
- It supports session caching, which boost performance.
  Session caching is when TLS remembers previous sessions and tries to resume the session instead of opening another one.

SSL and TLS work using a similar handshake procedure:

1. The client (user) requests to visit a website securely,
   (DNS things happen, omitted from this procedure)
2. The web server sends back its SSL digital certificate (which contains the public key). This certificate is signed by a certificate authority.
3. The client's browser then checks the following:
   - If the certificate is genuine, by first seeing if the CA is in the list of trusted CAs, and then verifying that the digital signature of the CA is genuine (see diagram in section above)
   - If the certificate is still valid, using the start and end dates on the certificate.
   - If the domain name listed on the certificate matches the website domain exactly.
4. If the browser trusts the digital certificate, it generates a session key. The key is encrypted using the web server's public key (which is part of the certificate) and is sent over to the web server.
5. The web server decrypts the session key with its private key. It sends back an acknowledgement that is encrypted using this session key.
6. The browser and web server have now established a secure encrypted connection.

When to use SSL/TLS?

- All online financial transactions. E.g., online banking
- Online shopping / E-Commerce
- Sending messages to a restricted / select list of users
- Emails
- VPNs
- Instant messengers
- etc.

(Basically, anything that does not need to be publicly available.)

Quantum Cryptography

The purpose of quantum cryptography: To provide an alternative method to share a session key. It is also extremely secure -- even an attempt at eavesdropping will destroy the message. (because physics)

The RSA algorithm will no longer work once quantum computers can factor very large numbers. The RSA algorithm is the basis for all asymmetric encryption, so if it breaks, asymmetric encryption will no longer work.

We can still set up a symmetrically encrypted session if a common secret key is shared by the two parties. Quantum cryptography can provide such a way to set up a common secret.

Advantage: Extremely secure, eavesdropping is almost completely impossible.

Disadvantages:

- requires dedicated line and special hardware (expensive for now)
- limited range
- possible for polarization to change during transmission (due to various conditions)

How quantum cryptography works

(This is not *explicitly* required by the syllabus, but both books included this.)

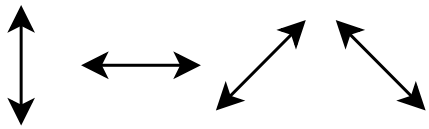
(Physics people please help me with this)

This section assumes the use of photons rather than other particles that exhibit quantum properties (e.g., electrons)

The procedure is as follows:

1. The sender uses a light source to generate the photons
   (From Jacob: what else would you generate photons with?)
2. For each bit, the sender randomly chooses one of the two possible polarization basis (directions in which to polarize the photon):
   - orthogonal (Vertical & Horizontal)
   - diagonal (45 degrees in both directions)

   Then according to the chosen basis, the photons are polarized below:



1 0 1 0

E.g., if the chosen basis is the orthogonal basis, and the bit to be sent is 0, the photon would be polarized horizontally; if the basis is diagonal, and the bit is 1, the photon would be polarized like ↗. (I don't have a diagonal double arrow)

3. The polarized photons travel along a fiber optic cable to the destination
4. At the destination, there are two beam splitters and two photon detectors.

   A vertical/horizontal splitter +: they can tell the first two polarizations apart, but the other two polarizations give random results on this filter.

   A diagonal splitter ×: They can tell the last two polarizations apart, but the other two polarizations give random results on this filter.
5. The receiver randomly chooses one of the two splitters for each bit. The result is recorded for each bit.
6. The sender sends over the whether each bit is polarized orthogonally or diagonally; the receiver sends over the chosen splitter for each bit. This is done using traditional methods (not quantum) and does not need to be encrypted.
7. The bits where the sender's choice and the receiver's choice are the same are known to be the same, the rest are discarded.

Alice's random bits	1	0	0	1	0	1	1	0
Alice's random sending basis	+	+	×	+	×	×	×	+
Photon polarization Alice sends	↑	→	↘	↑	↘	↗	↗	→
Bob's random measuring basis	+	×	×	×	+	×	+	+
Photon polarization Bob measures	↑	↗	↘	↗	→	↗	→	→
PUBLIC DISCUSSION OF BASIS								
Shared secret key	1		0			1		0

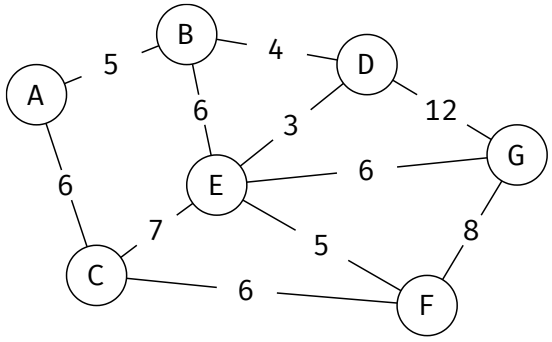
8. The shared secret key can then be used to create a new symmetric encryption session key.

Chapter 18: Artificial Intelligence

Dijkstra's algorithm

A graph is a collection of nodes (vertices) and edges connecting the nodes. Each edge can have a weight.

Below is an example of a graph. It may be helpful to imagine nodes as cities and edges as roads connecting the cities (the edge's weight would be the length of that road).



To find the shortest route from some node (source node), we have Dijkstra's Algorithm.

The algorithm is described below in text, and in pseudocode:

1. For each node, initialize its `distance` to be $+\infty$.
   This value represents the length of the shortest path from the source. These values are to be calculated.
   As no paths are known, the nodes are assumed to be "unreachable" from the source and therefore has `distance` infinity.
2. Create an empty set `visited`.
   (Or you can create a set `unvisited` that contains all nodes initially and is always the opposite of `visited`.)
3. Set the `distance` of the source node to be `0`.
4. While `visited` does not cover all the nodes, do:
   (i.e., repeat until all nodes are in `visited`)
   1. Pick the node `u` with the least `distance` value that is not in `visited`.
   2. Insert this node into `visited`.
   3. For each neighbor `v` of `u` that is not in `visited`:
      Let `new_distance` be `distance` of `u` + length of edge connecting `u` and `v`.
      If `new_distance` is less than the `distance` of `v`:
      1. update the `distance` of `v` to `new_distance`.
      2. mark the `previous` of `v` to `u`
5. `distance` now stores the shortest distance from the source node to every node.

   `previous` now stores the previous node on the path to some node. To reconstruct the shortest path, simply repeatedly follow `previous` until the source is reached.

```
function dijkstra(graph, source):

    visited ← new empty set

    foreach vertex v in graph.vertices:
        distance[v] ← INFINITY
```

```
        previous[v] ← UNDEFINED
    distance[source] ← 0

    while size(visited) ≠ number of vertices:
        u ← vertex in Q with minimum distance[u]
        add u to visited

        foreach neighbor v of u still in Q:
            alt ← distance[u] + graph.edges(u, v)
            if alt < distance[v]:
                distance[v] ← alt
                previous[v] ← u

    return distance, previous

function get_path_to_target(target):
    path ← empty sequence
    v ← target
    if prev[v] is defined or u == source:
        while v is defined:
            insert v at the beginning of S
            v ← prev[v]
    return S
```

A* algorithm

Dijkstra's algorithm in essence, does the following:

- pick the closest node to the source, and explore the neighbors

The closest node is picked because Dijkstra's algorithm expands its exploration evenly in all directions. This makes it slow because it is blind towards the destination: even if a route takes you farther from the destination, it still considers this route equally as it does with other routes.

The A* algorithm attempts to optimize Dijkstra's algorithm by adding a **heuristic** factor.

A **heuristic** employs a practical solution to a problem; when applied to algorithms this

The heuristic in the A* algorithm estimates the distance from the current node to the destination.

The only modification is: When picking a node in Dijkstra's algorithm, instead of picking the node with the minimum `distance`, add the heuristic to each node, then pick the node with the minimum total. (Procedure 4.1, or Line 10)

```
function a_star(graph, source, destination):
    visited ← new empty set

    foreach vertex v in graph.vertices:
        distance[v] ← INFINITY
        previous[v] ← UNDEFINED
    distance[source] ← 0

    while size(visited) ≠ number of vertices:
```

```
        u ← vertex in Q with minimum (distance[u] + heuristic(u,
destination))
        add u to visited

        foreach neighbor v of u still in Q:
            alt ← distance[u] + graph.edges(u, v)
            if alt < distance[v]:
                distance[v] ← alt
                previous[v] ← u

    return distance, previous

function get_path_to_target(target):
    path ← empty sequence
    v ← target
    if prev[v] is defined or u == source:
        while v is defined:
            insert v at the beginning of S
            v ← prev[v]
    return S
```

This makes nodes farther from source less likely to be updated, as they have a larger heuristic.

Uses of graphs

The syllabus requires us to be able to describe "the purposes of graphs". I do not understand this at all -- it's like letting us describe "the purpose of numbers" in math or "the purpose of a photon"...

So here is a past paper question whose answers I copy-pasted from the mark scheme. I think some of them are wrong though... (my thoughts are given in red)

Explain the use of graphs to aid Artificial Intelligence (AI):

- Artificial Neural Networks can be represented using graphs
  So what? It isn't useful to represent neural networks as graphs. Vectors, matrices, and tensors give more useful properties. Graphs and neural networks are not at all closely associated.
- Graphs provide structures for relationships // graphs provide relationships between nodes
  Fair, this is a "useful" property of a graph. (It isn't that useful... Any data structure imply relationships, and graphs are sorta defined by nodes (and other stuff))
- AI problems can be defined/solved as finding a path in a graph
  Quite a sweeping statement. Only a subset of problems can be reduced to path-finding. Also, seeing this sentence and the first point at the same time may give the impression that ML algorithms find a shortest path in a neural network. This is not correct.
- Graphs may be analyzed/ingested by a range of algorithms
  - ... e.g. A* / Dijkstra's algorithm
  - ... used in machine learning.

  Fair. (Still feels weird to me though...)

- Example of method e.g. Back propagation of errors / regression methods
  As I said, Graphs and neural networks are not at all closely associated. Moreover, Dijkstra's algo / A* don't use back propogation or regression. They, in fact, are in the opposing branch of AI.

AI

AI: intelligent machines that are capable of doing tasks which normally require human-level intelligence.

Machine Learning: the system learns without being programmed to learn.
(I don't quite understand, but whatever, it's in the Hodder Education book.)

Deep Learning: machines think in a way similar to the human brain. They handle huge amounts of data using artificial neural networks.

Deep Learning is a subset of Machine Learning, which is a subset of AI.

Examples of AI:

- Recommendation algorithms
- Smart home devices
- etc.

Narrow AI: has superior performance to a human in one particular task

General AI: similar performance to a human in any intellectual task

Strong AI: superior performance to a human in many tasks

Machine Learning

Algorithms are "trained" and learn from their past experience and examples.

The system make predictions or take decisions based on previous scenarios.

They can manage and analyze huge amounts of data quickly.

Labelled data and unlabelled data

Labelled data: data where the target answer is known and the object that the data describes is sufficiently recognized.
Basically, it's data that can give machine direct feedback on what the answer should be.

Unlabelled data: data where the objects are not defined well enough and need to be manually recognized.

Example: if we are to make a program that tries to predict the price of a house based on the area of the house,

- labelled data would be a known data point that contains BOTH the area of the house AND the price.
- unlabelled data would be a data point with ONLY ONE of the two values.

Types of machine learning

Supervised; Unsupervised; Reinforcement; Semi-supervised

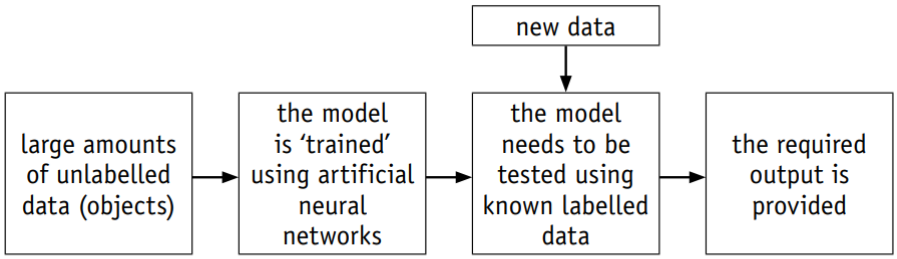
- Supervised learning
  - Predict future outcomes based on past data
  - Uses regression and classification analysis

    - During training, the model is fed labelled data. I.e., the model gets input and the **correct** output and learns the correlation between them.
    - The model can then be tested: the model is run with more labelled data. However, the model this time sees only the input
      and tries to make predictions of what the output should be.
      If the predicted output and the correct output differ too much, the model needs further refinement.
    - The model can then be used for predictions: it receives unlabelled data and predicts the outcome.
    - Examples: email spam filtering; the house price example from above; etc.
- Unsupervised learning
    - The system identify hidden patterns from the input data. They are not trained using the "correct" answer. (Such answers may not exist)
    - They can classify unlabelled data.
    - Examples:
        - If the house price example is modified so that the algorithm needs to detect what types of houses there are, unsupervised learning can be used.
        - In product marketing, customers can be classified based on their purchasing habits. Unsupervised learning can be used to detect which customers belong in what groups.
- Reinforcement learning
    - This system is not trained. It learns on the basis of "reward and punishment". It tries to take a decision, and then receives a reward or punishment based on the outcome of that decision.
      (If the consequence is desirable, it gets rewarded; If the consequence is bad, it gets punished. The magnitude of the reward/punishment often depends on how good/bad the consequence is.)
      Repeating this process enables the system to know which actions are better.
    - Examples
        - Search engines: the system suggests some results. If the user clicks on the top ones it receives a reward; otherwise it receives a punishment.
        - Bots in games: they play the game and receive a reward if they win, and a punishment if they lose.
- Semi-supervised learning
    - The book says: Semi-supervised learning makes use of labelled and unlabelled data to train algorithms that can interactively query the source data and produce a desired output.
    - It uses a lot of unlabelled data and a little bit of labelled data. This is because labelled data is more expensive to get than unlabelled data.
    - This typically yields better results than if only unlabelled data is used.
    - This also does classification. (Like unsupervised learning)
    - Example:
        - Classifying web pages according to their topic: sport, science, finance, etc. The system looks at large amounts of unlabelled web pages (unknown topic) and a few labelled web pages (known topic).

> The system can see which pages are similar to each other and know they are about the same topic. Using the labelled pages, they can have a better judgement.

Deep learning

The book directly contradicts with what I have learnt, so here's the book.

> This summarises how deep learning works:
>
> 
>
>
> ▲ **Figure 18.25**
>
> Large amounts of unlabelled data (data which is undefined and needs to be recognised) is input into the model. One of the methods of object recognition, using pixel densities, was described above. Using artificial neural networks, each of the objects is identified by the system. Labelled data (data which has already been defined and is, therefore, recognised) is then entered into the model to make sure it gives the correct responses. If the output is not sufficiently accurate, the model is refined until it gives satisfactory results (known as **back propagation** – see Section 18.2.6). The refinement process may take several adjustments until it provides reliable and consistent outputs.

The book also directly contradicts itself. Despite explicitly saying that deep learning is a subset of machine learning, it still states that they have opposing properties:

machine learning	deep learning
enables machines to make decisions on their own based on past data	enables machines to make decisions using an artificial neural network
needs only a small amount of data to carry out the training	the system needs large amounts of data during the training stages
most of the features in the data used need to be identified in advance and then manually coded into the system	deep learning machine learns the features of the data from the data itself and it does not need to be identified in advance
a modular approach is taken to solve a given problem/task; each module is then combined to produce the final model	the problem is solved from beginning to end as a single entity
testing of the system takes a long time to carry out	testing of the system takes much less time to carry out
there are clear rules which explain why each stage in the model was made	since the system makes decisions based on its own logic, the reasoning behind those decisions may be very difficult to understand (they are often referred to as a **black box**)

Regression

An example of regression we are all familiar with is finding the line of best fit.

Regression is estimating relationships between a dependent variable and one or more independent variables.

Therefore it is useful in understanding how the value of the dependent variable changes when independent variables are changed.

(The book:) In ML, it is usually used to predict the outcome of an event based on any relationship between variables obtained from input data and the hidden parameters.