Lesson 3

Data Structures and Algorithms
DSA

In this lesson we will talk about:

- data structures
- algorithm design
- algorithm efficiency
- searching and sorting algorithms

Data Structures

What is a data structure?

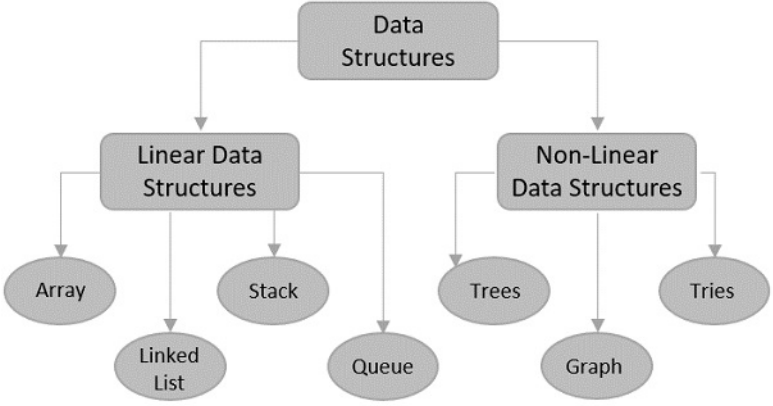
In computer science, a **data structure** is a data organization and storage format that is usually chosen for efficient access to data.[1][2][3] More precisely, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data,[4] i.e., it is an algebraic structure about data.

Data Structures

- How do we **organize** data
- For a very **efficient** access

It's a collection of data values, and the relationships among these values

Data Structures



Linear Data Structures: Arrays, Queues, Stacks

- Elements are arranged sequentially, one after the other
- The first element added will be the first one to be accessed or removed, and the last element added will be the last one to be accessed or removed
- Can have either fixed or dynamic sizes
- Offer very efficient data access

Non-linear Data Structures: Trees, Graphs

- Elements are arranged hierarchical
- We cant traverse all the elements in a single run only
- There are multiple levels which we must traverse
- It is more difficult to implement

Data structures

- Sets
- Arrays and Matrices
- Stacks
- Queues
- Linked lists
- Trees
- Graphs

Sets

Sets

A set is usually a **collection** of different things, fixed in size. Sets can also change size, usually when an algorithm will perform modifications against the set. We call these dynamic sets. These sets can change in size, grow or shrink, basically change over the time.

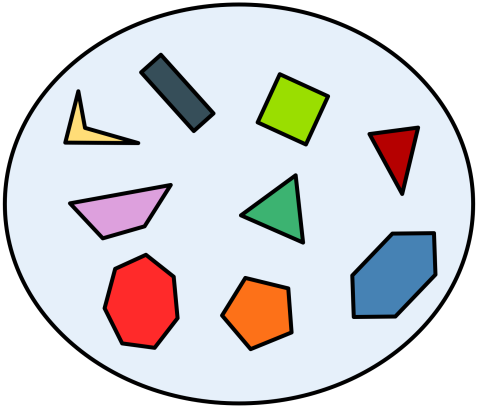
But what is a set?

Sets

The basic, fundamental data structure: $\{1,2,4,51,9\}$

- mathematical set
- unchanging, unique elements, no duplicates
- contains a fixed number of elements: finite set
- or it can contain an infinite number of elements

For example a set of polygons



Sets

A set is a **mathematical model** of a collection of different things. A set contains elements or members, which can be mathematical objects of any kind numbers, symbols, points in space, lines, other geometrical shapes, variables, or even other sets.

Sets, examples

- $\{$white, blue, red, yellow$\}$
- The empty set $\{\}$
- Natural numbers: $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$
- Natural numbers except 0: $\mathbb{N}^* = \{1, 2, 3, \ldots\}$
- Integers: $\mathbb{Z} = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$
- Positive integers: $\mathbb{Z}_+ = \{0, 1, 2, 3, \ldots\}$

$\{\mathbf{1,2,3,4}\}$

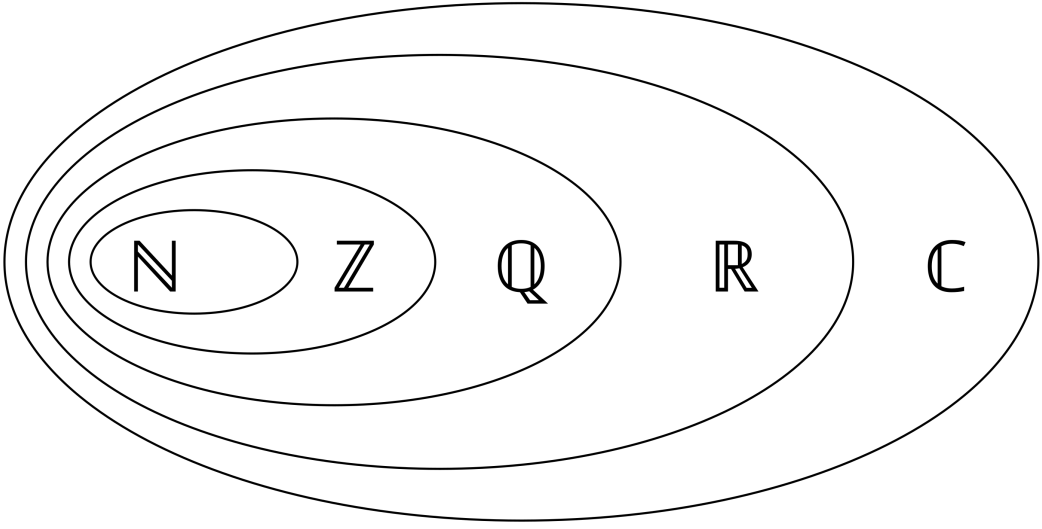
Defines a list of elements, using a simple enumeration notation (Roster notation) between curly brackets, separated by commas.

Basic Operations on Sets

- Insert - add a new element to a set
- Delete - remove an element from a set
- Test - if element X belongs to a set or not

A dynamic set which supports all these basic operations: **a dictionary**

Natural Numbers (Counting Numbers) ($\mathbb{N}$)	Numbers you use for counting: 1, 2, 3 ...	It's "natural" to count on your fingers: 1, 2, 3, ...
Whole Numbers	The natural numbers, plus 0: 0, 1, 2, 3 ...	The word "whole" has an "o" in it, so include 0.
Integers ($\mathbb{Z}$)	Whole numbers, their opposites (negatives), plus 0: ... −2, −1, 0, 1, 2 ...	Integers can be separated into negative, 0, and positive numbers.
Rationals ($\mathbb{Q}$)	Integers and all fractions, positive and negative, formed from integers. These include repeating fractions, such as $\frac{1}{3}$, or .33333.. or $.\overline{3}$.	The word "rational" is a derivation of "ratio", and rational numbers are numbers that can be written as a ratio of two integers. "Q" stands for quotient.
Irrationals	Numbers that cannot be expressed as a fraction, such as π, $\sqrt{2}$, e. (We'll learn about these later).	If something is "irrational", it's not easy to explain or understand.
Real Numbers ($\mathbb{R}$)	Rational numbers and Irrational Numbers. The real number system can be represented on a number line: 	If a number exists on a number line that you can see, it must be "real". Note that the "smallest" real number is negative (–) infinity ($-\infty$), and the largest real number is infinity (∞). We can never really get to these "numbers" ($-\infty$ and ∞), but we can indicate them as the "end" of the real numbers.
Complex Numbers ($\mathbb{C}$)	Real numbers, plus imaginary numbers (concept only, such as $\sqrt{-2}$).	"Imaginary" numbers are difficult to imagine, since they are so "complex".


ℕ
ℤ
ℚ
ℝ
ℂ

Advantages

Perform operations on a collection of elements in a very **efficient** and **organized** manner

Conclusions

Sets are basic, fundamental data structures, with:

- unique elements
- no duplicates
- unchanging
- fixed or infinite number of elements

Im confused. Does it mean a set is similar to a Python set? Or what is the difference?

Sets vs. Python Set

In computer science (CS), a set is an abstract data type that can store unique values, without any particular order. It is a computer implementation of the mathematical concept of a finite set.

Mathematical vs. Python Sets

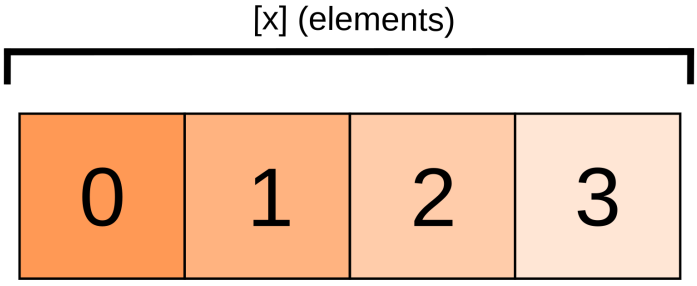
- Mathematical finite set: $\{1,2,3,4\}$
- Python set: $S = \{1,2,3,4\}$

Arrays

Arrays

In computer science, an **array** is a data structure consisting of a collection of elements, each identified by an **index** or a **key**. The simplest type of such data structure is a linear array, the one-dimensional array.

Typical "1 Dimensional" array



Element indexes are typically defined in the format *[x]*
[x] being the number of elements
For example: this array could be defined as *array[4]*

Arrays

Arrays are among the oldest and most important data structures, and are used by almost every program and programming language. They are also used to implement many other data structures, such as lists.

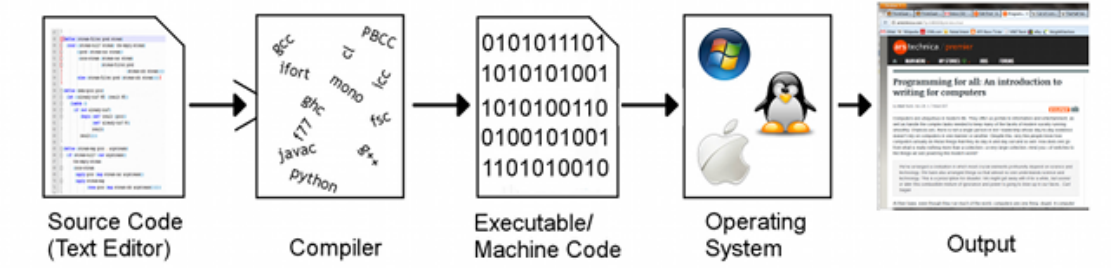
Arrays

Arrays are useful because the element indices can be computed at run time. Among other things, this feature allows a single iterative statement to process arbitrarily many elements of an array. For that reason, the elements of an array data structure are required to have the same size and should use the same data representation.

Run-time?

Runtime, run time, or execution time is the final phase of a computer program's life cycle, in which the code is being executed on the computer's central processing unit (CPU) as machine code. In other words, "runtime" is the running phase of a program.

Remember this? From source code to executable



Basic Array Operations

- traversal of an array
- access element X in an array
- searching element X in an array
- sorting an array

Example 1: Traversing the array A

```
1: procedure GETARRAY(A)                ▷ Returns the max value in A
2:     L ← length(A)
3:     for i=0 to L-1 do
4:         print A[i]
5:     end for
6: end procedure
```

Traversing the array

```
A = [1, 2, 3, 4, 5, 6, 7,8,9]

# Traversing the array
for element in A:
    print(element, end=" ")
```

Traversing the array, version 2

```
A = [1, 2, 3, 4, 5, 6, 7,8,9]

# Traversing the array
for i in range(len(A)):
    print(A[i], end=" ")
```

Example 2: Find the max value in the array A

```
1:  procedure MaxArray(A)                    ▷ Returns the max value in A
2:      N ← length(A)
3:      MAX ← A[0]
4:      for from i=1 to N-1 do
5:          if A[i] > MAX then
6:              MAX = A[i]                   ▷ The MAX is A[i]
7:          end if
8:      end for
9:      return MAX
10: end procedure
```

Example 3: Search element X in array A

```
1:  procedure SEARCHARRAY(A)   ▷ Returns the max value in A
2:      X ← MyElement
3:      N ← length(A)
4:      for from i=0 to N-1 do
5:          if X = A[i] then
6:              return i                    ▷ The index for my match
7:          end if
8:      end for
9:      return -1                           ▷ otherwise return -1
10: end procedure
```

Search element in array

```
A = [1, 2, 3, 4, 5, 6, 7,8,9]

def find_element(A, n, key):
    for i in range(n):
        if A[i] == key:
            return i
    return -1
```

There are numerous applications of arrays

- Storing data in databases. Storing a list of customer names.
- Traffic Management.Traffic management systems use arrays to track vehicles and their flow. By analyzing data stored in arrays, traffic control centers can implement efficient signal timings and manage congestion effectively.

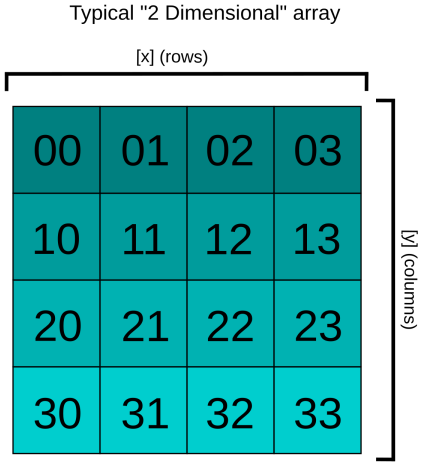
- Financial Analysis. It keeps track of various financial instruments, including stocks, bonds, and mutual funds. By organizing data in arrays, companies can perform analyses and make predictions easier
- Machine Learning. Machine learning algorithms often accept arrays as input, helping to train models and make predictions.

Matrices

What is a matrix?

In mathematics, a matrix (pl.: matrices) is a rectangular array or table of numbers, symbols, or expressions, with elements or entries arranged in rows and columns, which is used to represent a mathematical object or property of such an object.

Matrices



Element indexes are typically defined in the format*[x][y]*
[x] being the number of rows
[y] being the number of columns

Matrices

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ \vdots \\ m \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & \cdots & n \end{array} \\ \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \end{array}$$

Basic Matrix Operations

- access X element in a matrix
- traversal of a matrix
- searching a matrix
- sorting a matrix

Accessing the elements of a matrix

```
A = [[1, 2, 3], [4, 5, 6], [7,8,9]]

# Accessing certain elements in a matrix
print("1st element of 1st row:", A[0][0])
print("2nd element of the 2nd row:", A[1][2])
print("2nd element of 3rd row:", A[2][1])
```

Traversing the matrix

```
A = [[1, 2, 3], [4, 5, 6], [7,8,9]]

# Traversing the matrix
for row in A:
    # Traversing the matrix
    for x in row:
        print(x, end=" ")
    print()
```

There are numerous applications of matrices

- Encryption: Matrices encrypt data into unreadable formats and decode it for secure communication.
- Computer Graphics: transformations like scaling, rotation, and translation of objects in 2D and 3D graphics
- Machine Learning: fundamental data structures for neural networks

- Economics and Business: optimize business operations like supply chains and financial forecasting
- Navigation Systems: GPS systems use matrices to calculate positions, distances, and directions in 2D and 3D space.
- Weather Prediction: Matrices solve systems of differential equations to model and predict climate and weather patterns

Stacks

What is a stack?

The stack is an analogy to a set of physical items stacked one atop another, such as a stack of plates.

Lesson 3

Stacks

Stacks

Stacks are collections of elements, which support two main operations:

- **PUSH**: which adds an element to the collection
- **POP**: which removes the most recent elements from the collection

Stacks

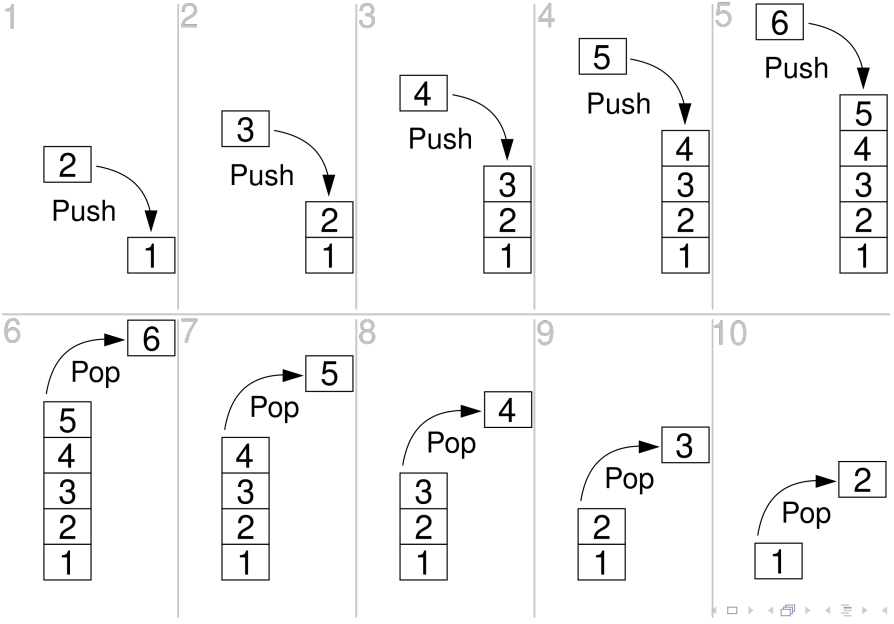
There might be, another operation called **PEEK**, which without modifying the stack, return the value of the last element added.

Stacks

- **PUSH**: which adds an element to the collection
- **POP**: which removes the most recent elements from the collection
- **PEEK**: returns the value of the last element added

Stacks

The stack supports few operations. And it operates in a certain, predefined order. The order in which an element added to or removed from a stack is described as last in, first out, referred to by the acronym **LIFO**


1
2
Push
1
2
3
Push
2
1
3
4
Push
3
2
1
4
5
Push
4
3
2
1
5
6
Push
5
4
3
2
1
6
6
Pop
5
4
3
2
1
7
5
Pop
4
3
2
1
8
4
Pop
3
2
1
9
3
Pop
2
1
10
2
Pop
1

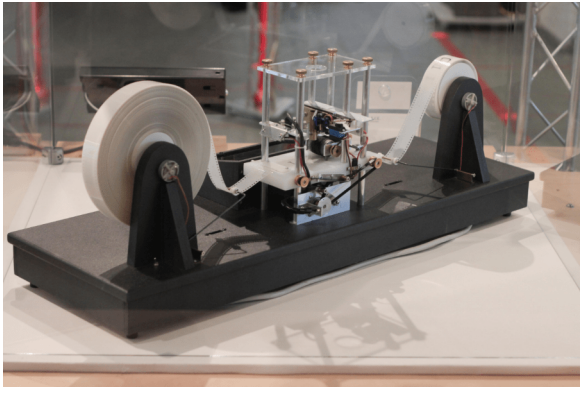
Stacks

"Stacks entered the computer science literature in 1946, when Alan Turing used the terms "bury" and "unbury" as a means of calling and returning from subroutines."

Who was Alan Turing?

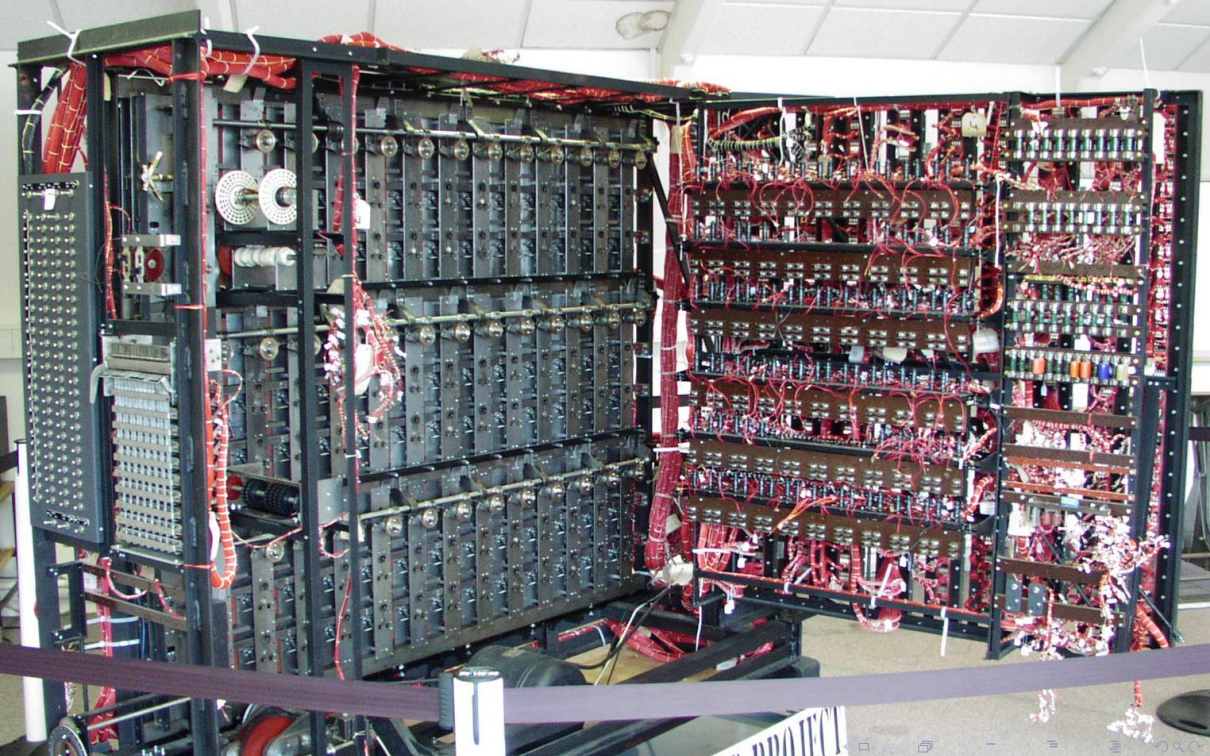
- The father of Computer Science
- English mathematician, computer scientist, logician, cryptanalyst
- Established the very first formalisation of the concepts of algorithm and computation with the Turing machine (a model of a general-purpose computer)

Remember this? The Turing Machine



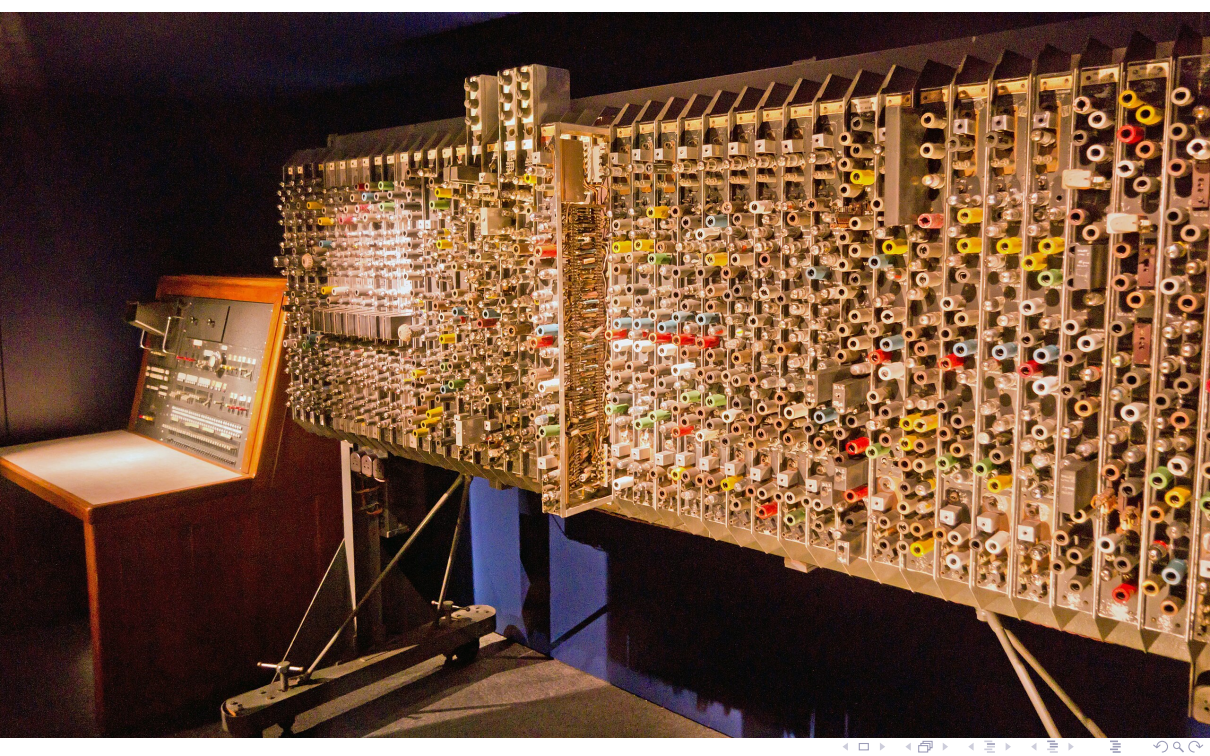
- Infinite tape, divided into cells with symbols
- A head can read/write symbols on the tape
- A register that stores the state of the machine

During World War II, Turing worked for the Government Code and Cypher School at Bletchley Park, Britain's codebreaking centre that produced Ultra intelligence. He led Hut 8, the section responsible for German naval cryptanalysis. Turing devised techniques for speeding the breaking of German ciphers, including improvements to the pre-war Polish bomba method, an electromechanical machine that could find settings for the Enigma machine.



Alan Turing - the inventor of ACE

Automatic Computing Engine (ACE) was a British early electronic serial stored-program computer



Want to know more? Read about Turing. See the **Imitation Game** movie.


THE
IMITATION GAME

Stacks

How can we implement a stack?

Stacks

A stack can be easily implemented using an array. The first element, usually at the zero offset, is the bottom, resulting in array[0] being the first element pushed onto the stack and the last element popped off.

Stacks

The program must keep track of the size (length) of the stack, using a variable top that records the number of items pushed so far, therefore pointing to the place in the array where the next element is to be inserted (assuming a zero-based index convention).

Stacks

The program must keep track of the size (length) of the stack, using a variable top that records the number of items pushed so far, therefore pointing to the place in the array where the next element is to be inserted (assuming a zero-based index convention).

Stack using arrays

```
# Create a stack. It initializes size of stack as 0
def createStack():
    stack = []
    return stack
```

```
# Stack is empty when stack size is 0
def isEmpty(stack):
    return len(stack) == 0

# Add an item to stack. It increases size by 1
def push(stack, item):
    stack.append(item)
    print(item + " pushed to stack ")
```

```
# Remove an item from stack. It decreases size by 1
def pop(stack):
    if (isEmpty(stack)):
        # return minus infinite
        return str(-maxsize -1)
    return stack.pop()

# Return the top from stack without removing it
def peek(stack):
    if (isEmpty(stack)):
        # return minus infinite
        return str(-maxsize -1)
    return stack[len(stack) - 1]
```

There are numerous applications of stacks

- Undo mechanism in text editors
- Fwd and back buttons on web browsers
- Memory management in computer programming (static memory allocation. It can be used to keep track of functions calls)
- Implementing recursion

But what is recursion?

Recursion

- It is a programming technique, a way to program and develop a program where a function calls itself many times
- You take a big problem, into smaller problems, trying to apply a solution to solve the small problems
- You define a problem in terms of itself

Recursion

Learning to think recursively is learning to look for big things that are made from smaller things - exactly the same type as the big thing.

Recursion

You are standing in a long queue of people. You must answer, how many people are behind you in the line?

Note: One person can see only the person standing directly in front and behind. One cannot look back and count. Each person is allowed to ask questions from the person standing in front or behind.

Recursion

You look behind and see if there is a person there. If not, then you can return the answer "0". If there is a person, repeat this step and wait for a response from the person standing behind. Once a person receives a response, they add 1 and respond to the person that asked them or the person standing in front of them.

Recursion example

```
1: procedure personCount(currPerson)
2:     if noOneBehind(currPerson) == TRUE then
3:         return 0
4:     else personBehind == currPerson.checkBehind
5:         return 1 + personCount()
6:     end if
7: end procedure
```

Queues

Linked lists