CS 91 USACO

Bronze Division

Unit 2: 1-D Data Structures

LECTURE 4: OVERVIEW OF DATA STRUCTURES

DR. ERIC CHOU

IEEE SENIOR MEMBER

Objectives

- What is problem solving?
- Programming Paradigm: linear programming, integer programming, 0/1 programming, quadratic programming, dynamic programming.
- What is Data Structure?

What is Problem Solving?

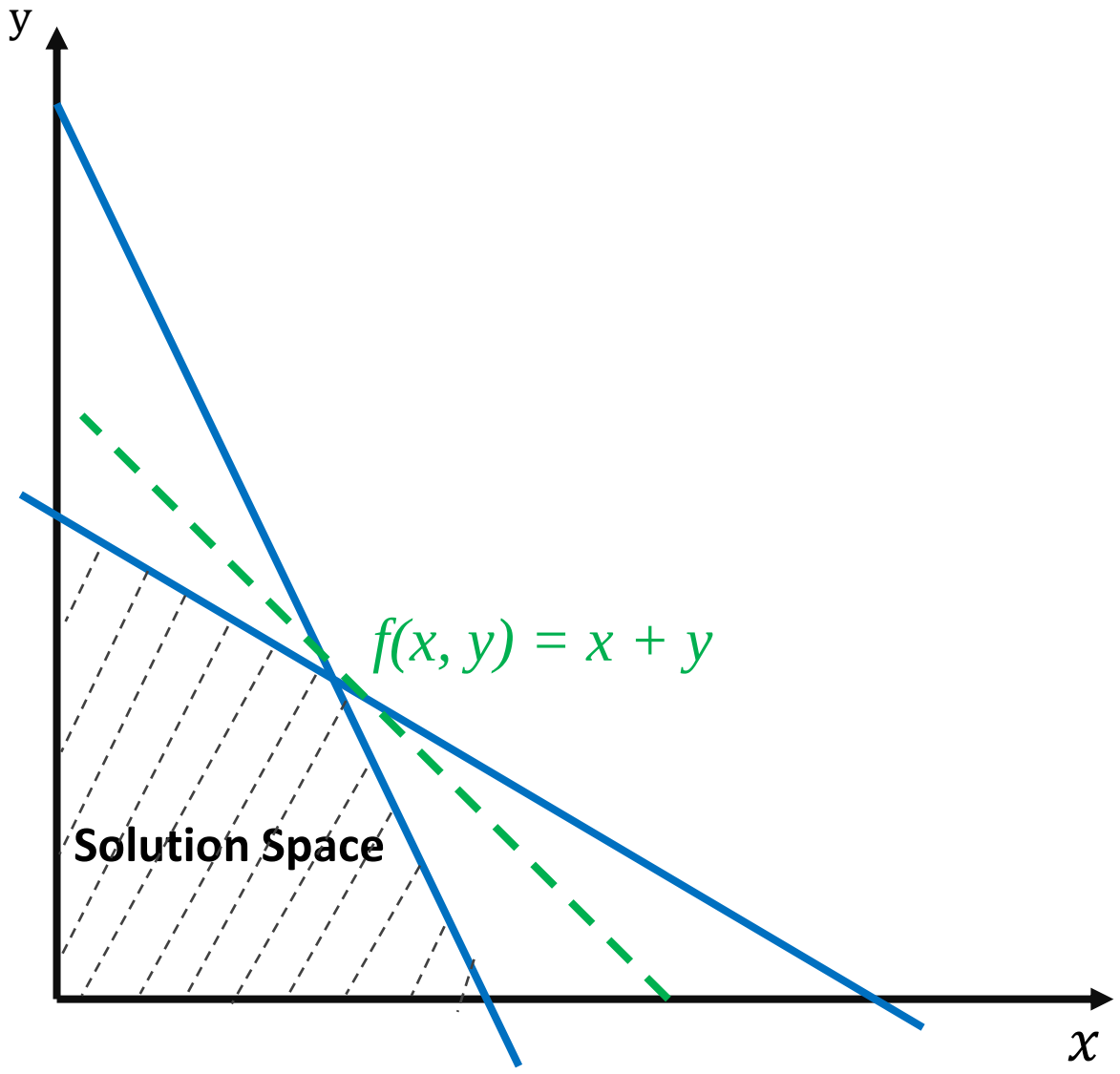
SECTION 1

Problem Solving

- Exact solution versus approximation solution.
- Algorithmic Solution.
- Optimization Solution. (Simulated solution)

Solution Space

- Discrete Space:
  - Integer Domain
  - 0-1 Domain
  - 1-D, 2-D, Tree, Graph
- Continuous Space:
  - Linear Equation (line)
  - Quadratic Equation (elliptic, circular, parabolic, hyperbolic)



Linear Programming

Goal Function:

$Max:\ f(x, y) = x + y$

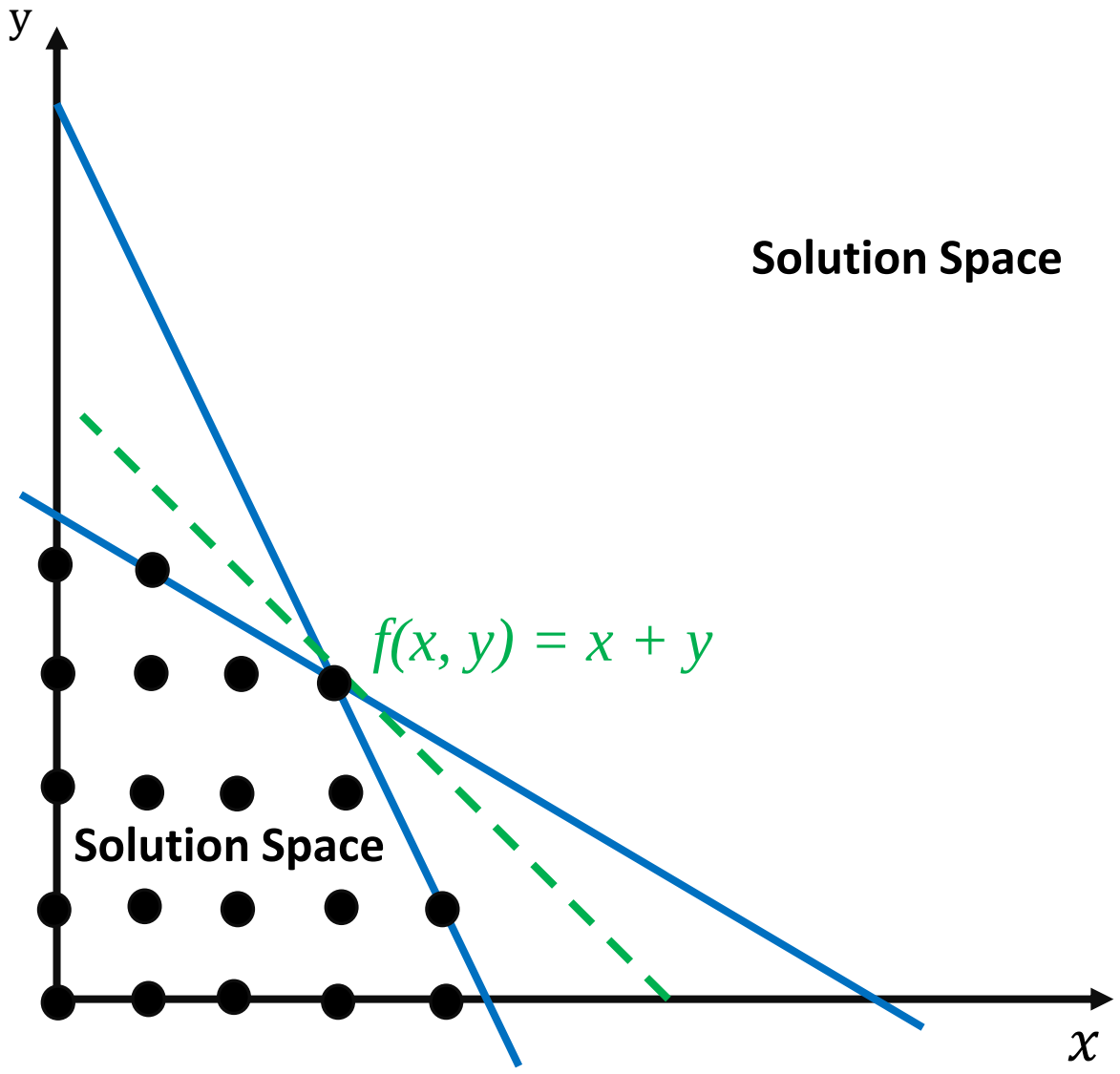
Constraints:

$x \geq 0$

$y \geq 0$

$2x + y \leq 9$

$x + 2y \leq 9$



Integer Programming

Goal Function:

$Max:\ f(x, y) = x + y$

Constraints:

$x \geq 0$

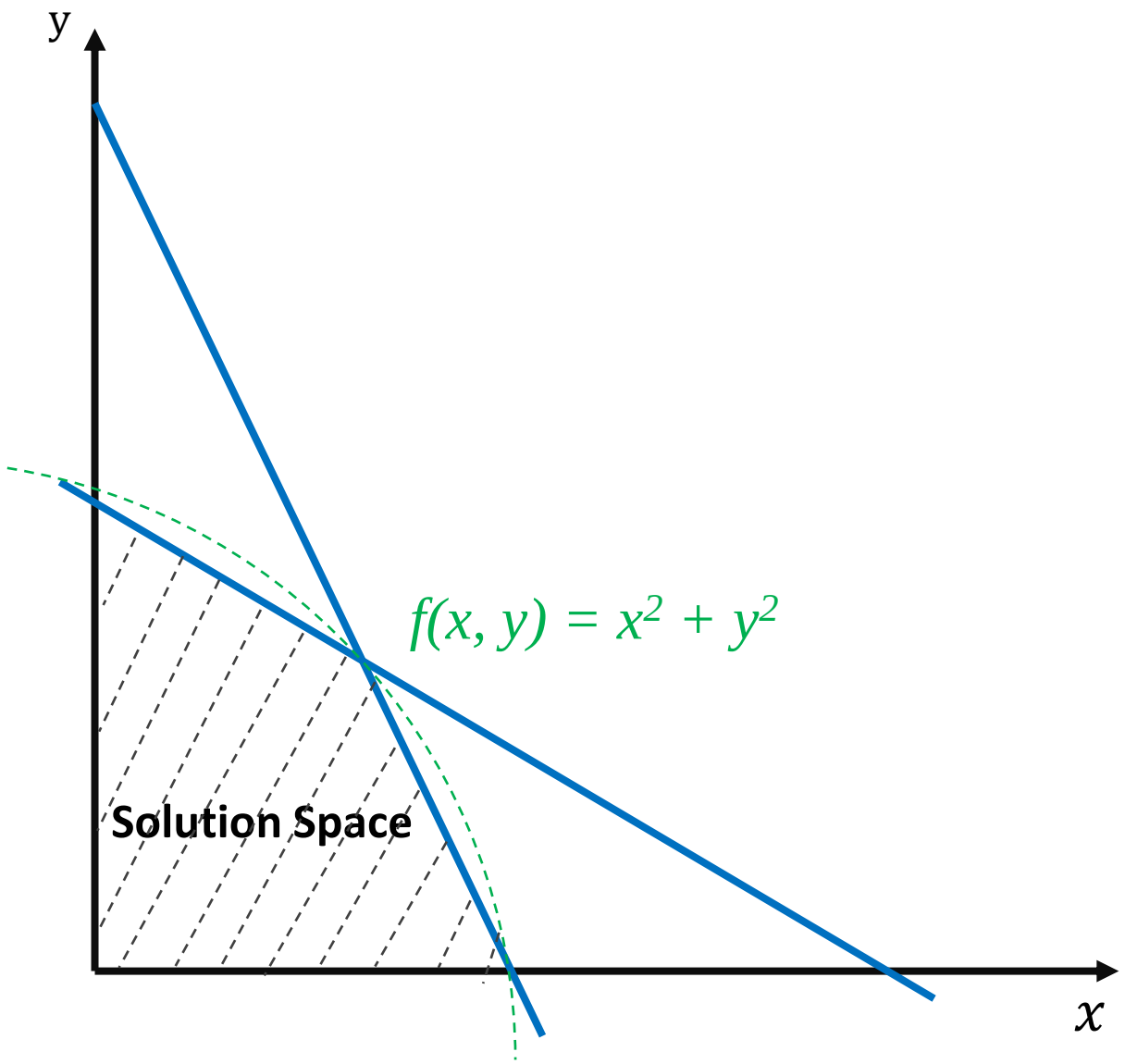
$y \geq 0$

$2x + y \leq 9$

$x + 2y \leq 9$

$x \in Z$

$y \in Z$



Quadratic Programming

Goal Function:

Max: $f(x, y) = x^2 + y^2$

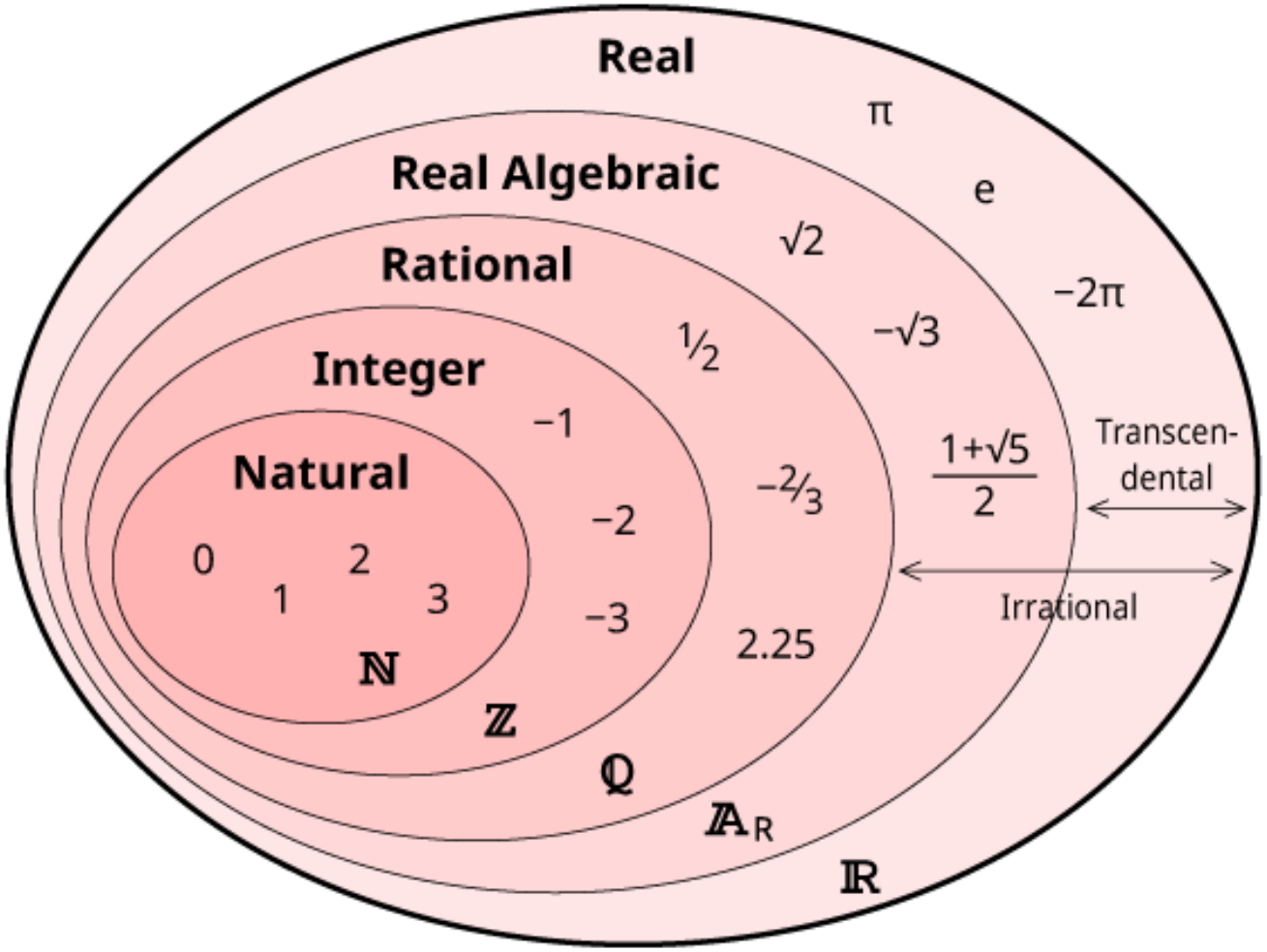
Constraints:

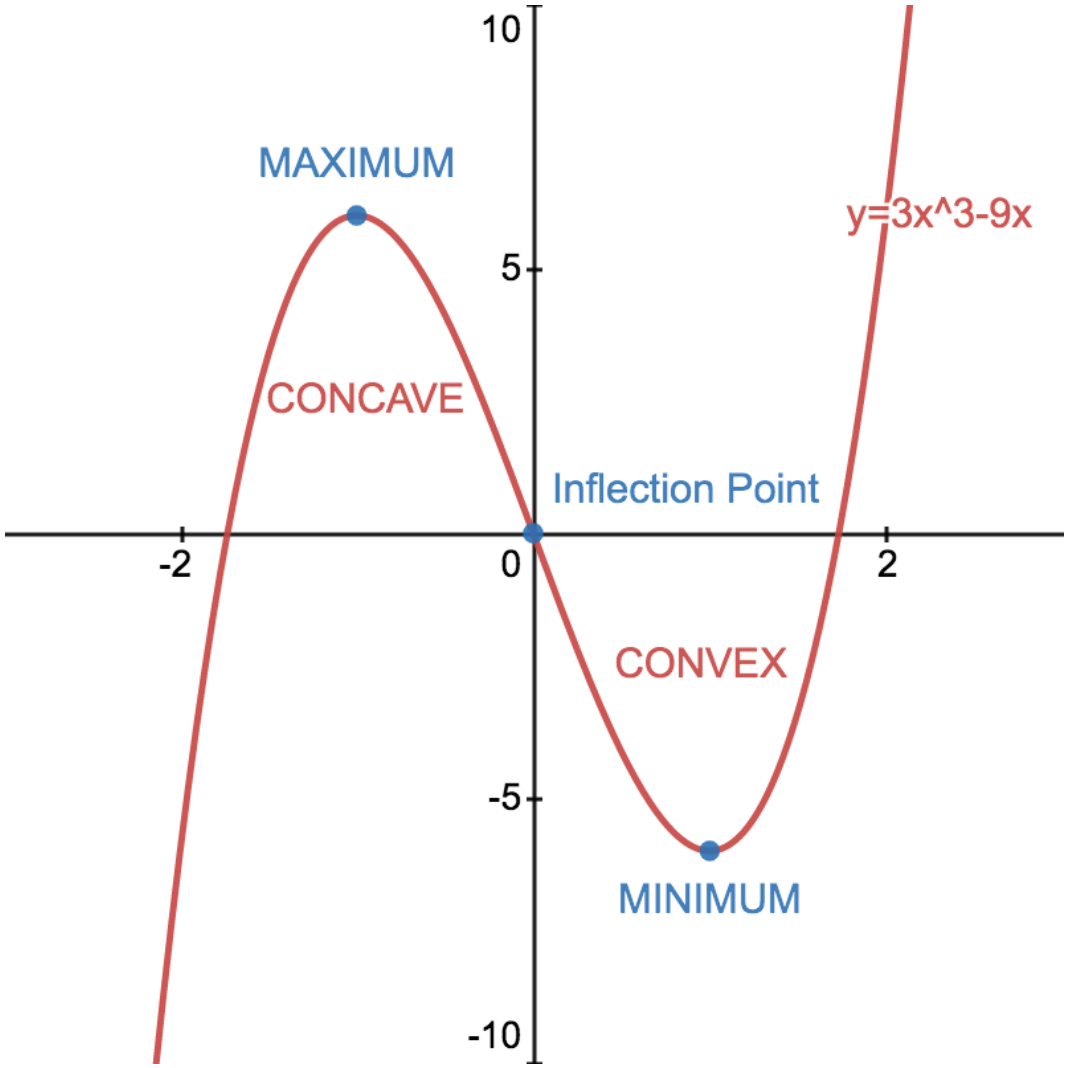
$x \geq 0$

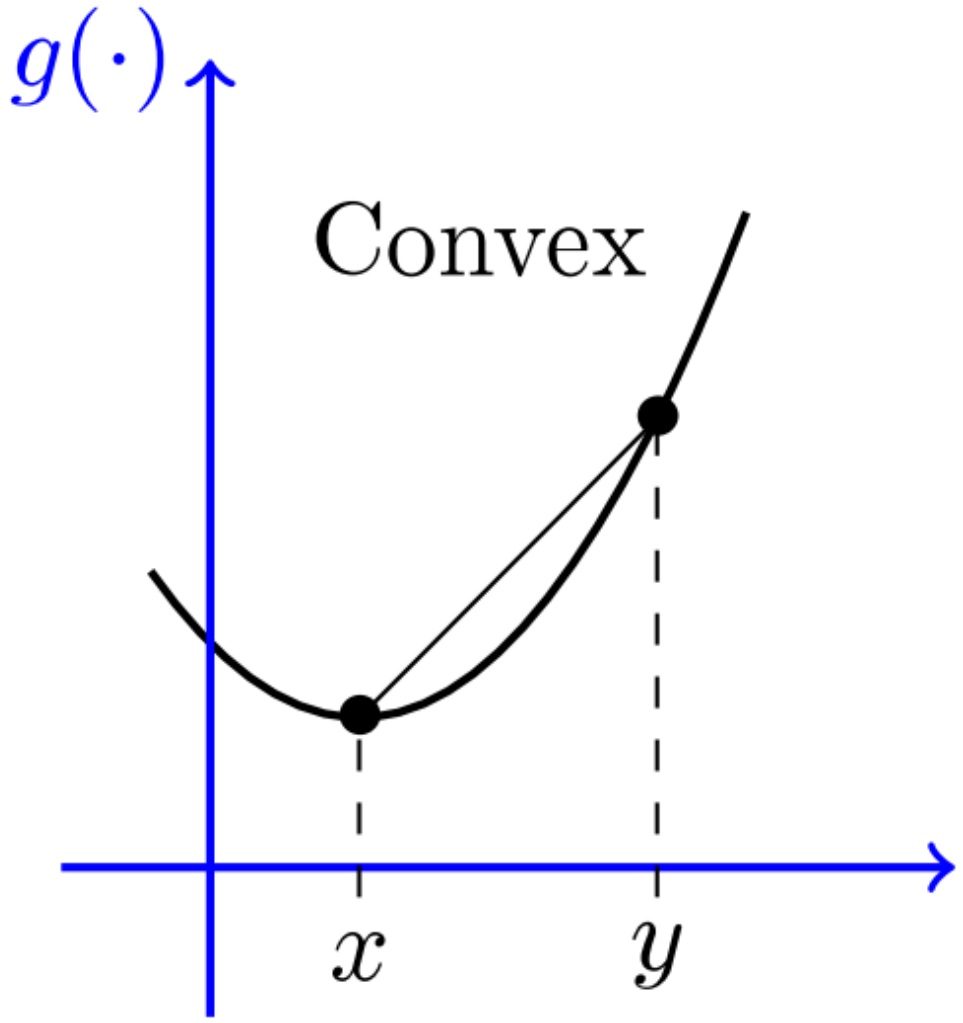
$y \geq 0$

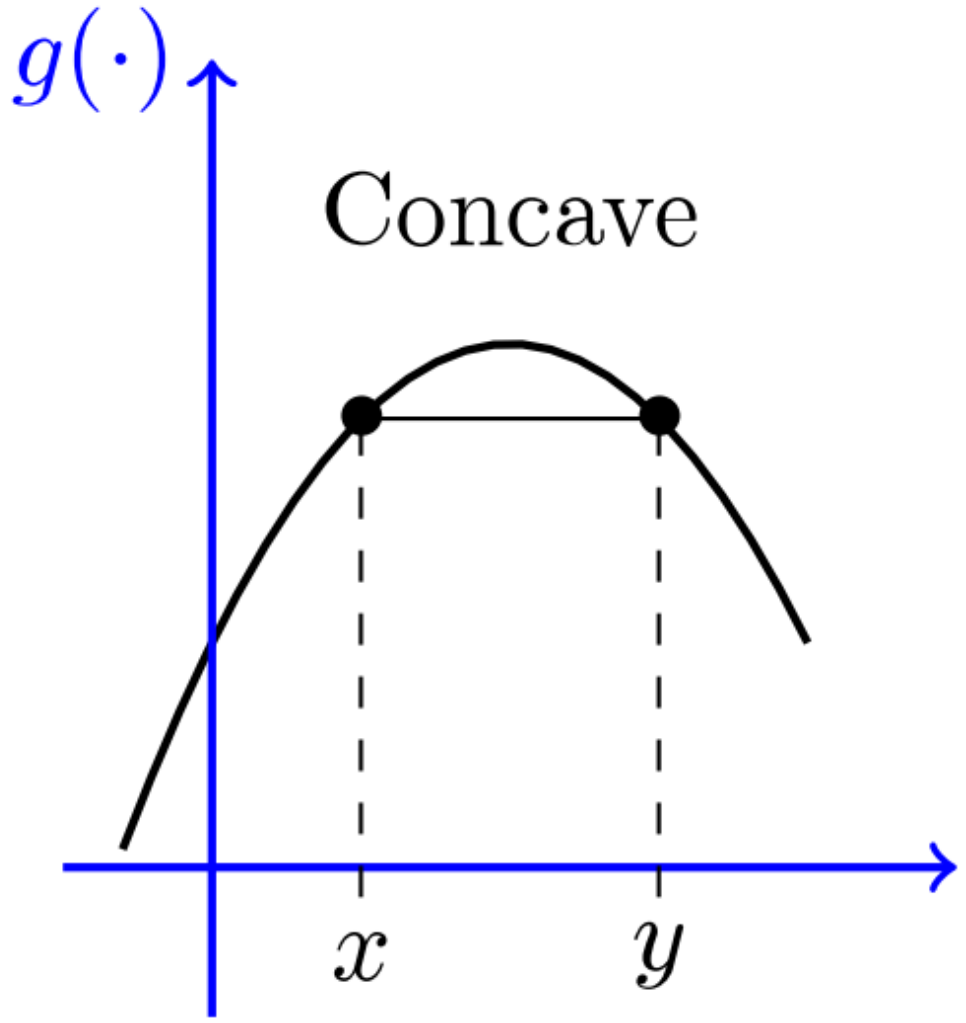
$2x + y \leq 9$

$x + 2y \leq 9$


Real
Real Algebraic
Rational
Integer
Natural
0
1
2
3
−1
−2
−3
1/2
−2/3
2.25
√2
−√3
π
e
−2π
(1+√5)/2
Transcen-
dental
Irrational
ℕ
ℤ
ℚ
𝔸R
ℝ


10
5
-5
-10
-2
0
2
MAXIMUM
CONCAVE
Inflection Point
CONVEX
MINIMUM
y=3x^3-9x


$g(\cdot)$
Convex
x
y


$g(\cdot)$
Concave
x
y

Problem Solving

Optimization

Machine Learning
Optimization with Historical Data

What is Data Structure?

SECTION 2

Data Structures

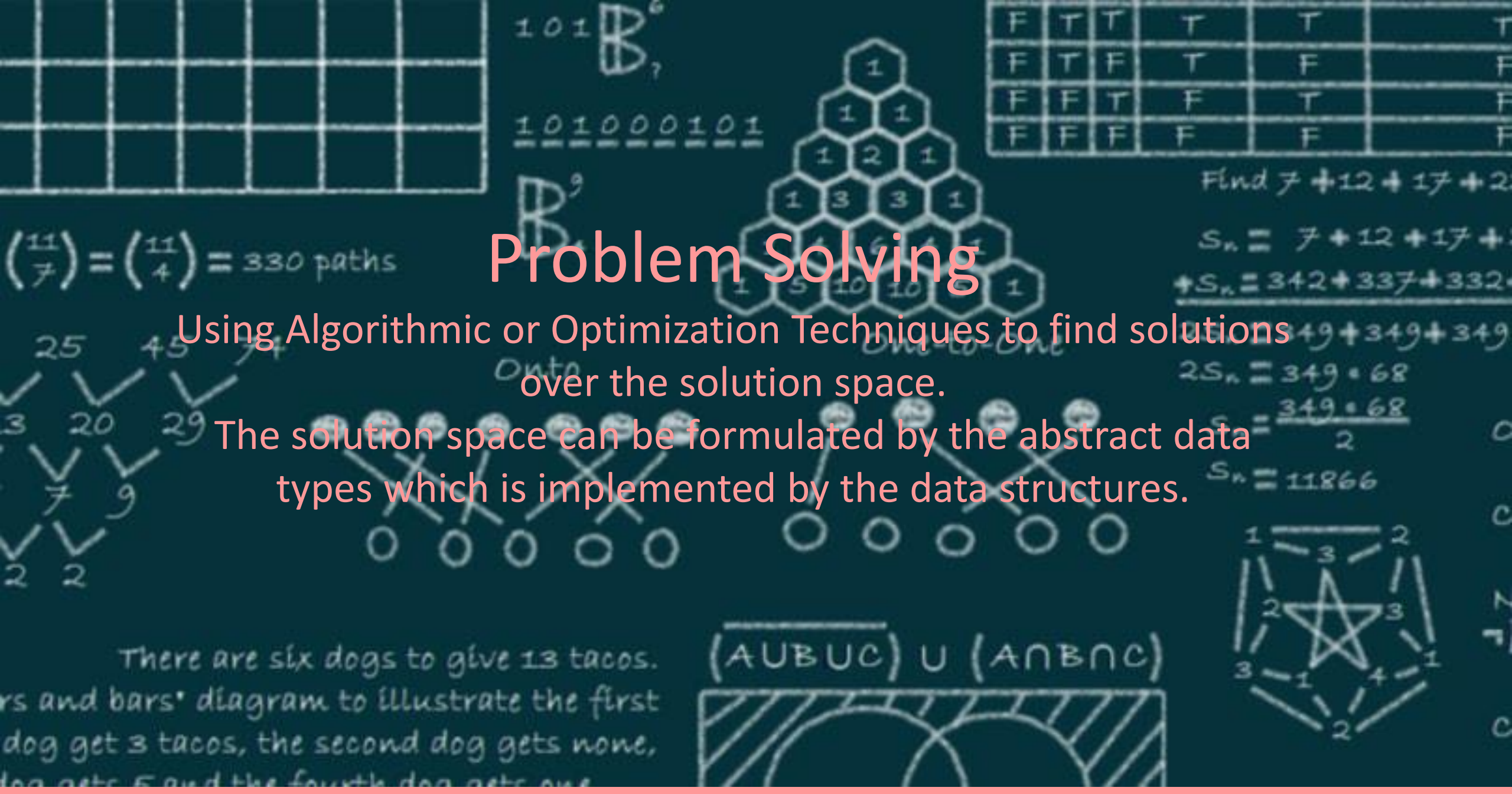
The Organization of Data

Array, Linked List, Linked Tree,
Linked Nodes

Abstract Datatypes

Abstract Concepts of Data Collections

List, Stack, Queue, Set, Map,
Priority Queue, Heap, Tree, Graph



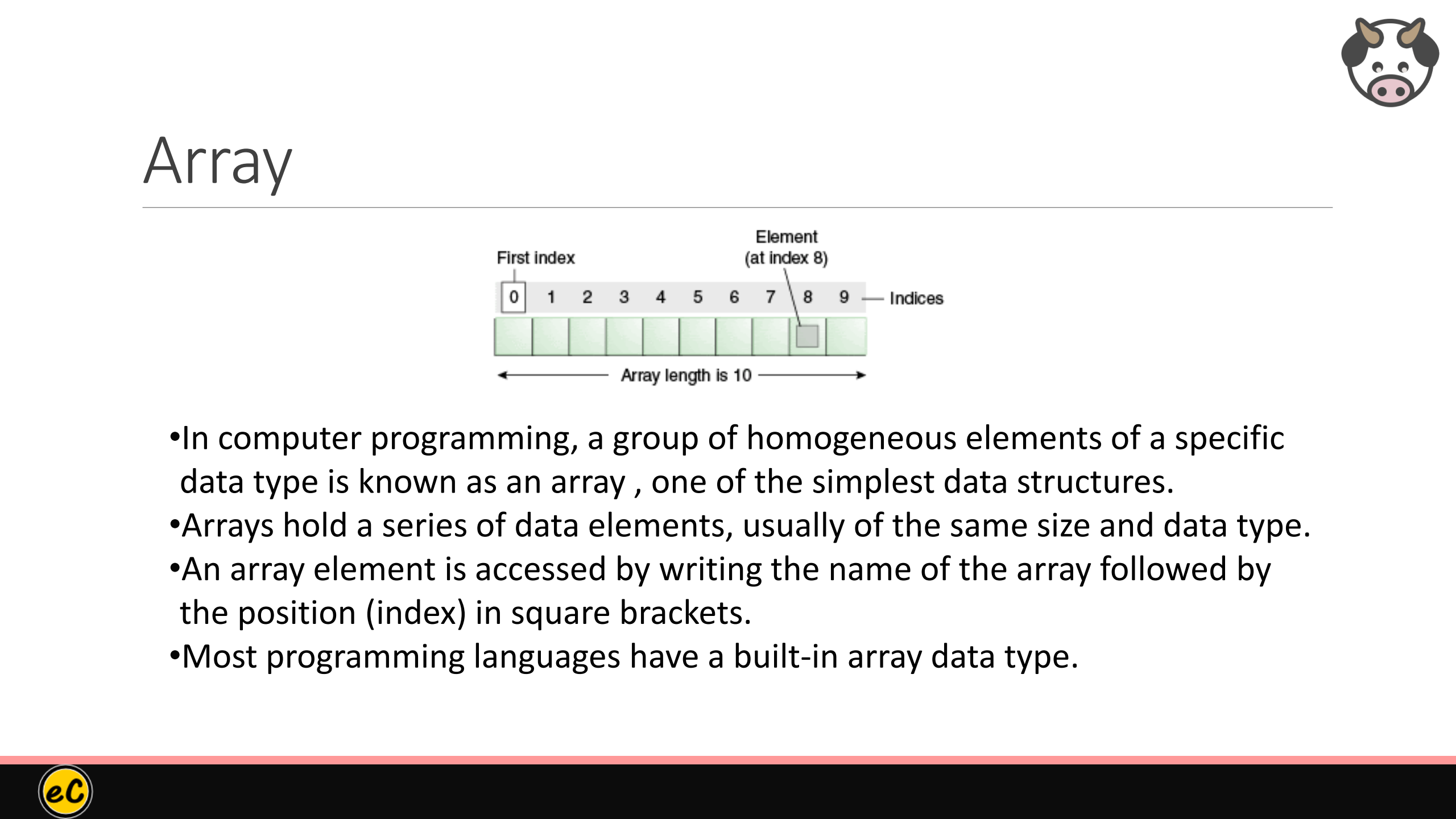
Problem Solving

Using Algorithmic or Optimization Techniques to find solutions over the solution space.

The solution space can be formulated by the abstract data types which is implemented by the data structures.

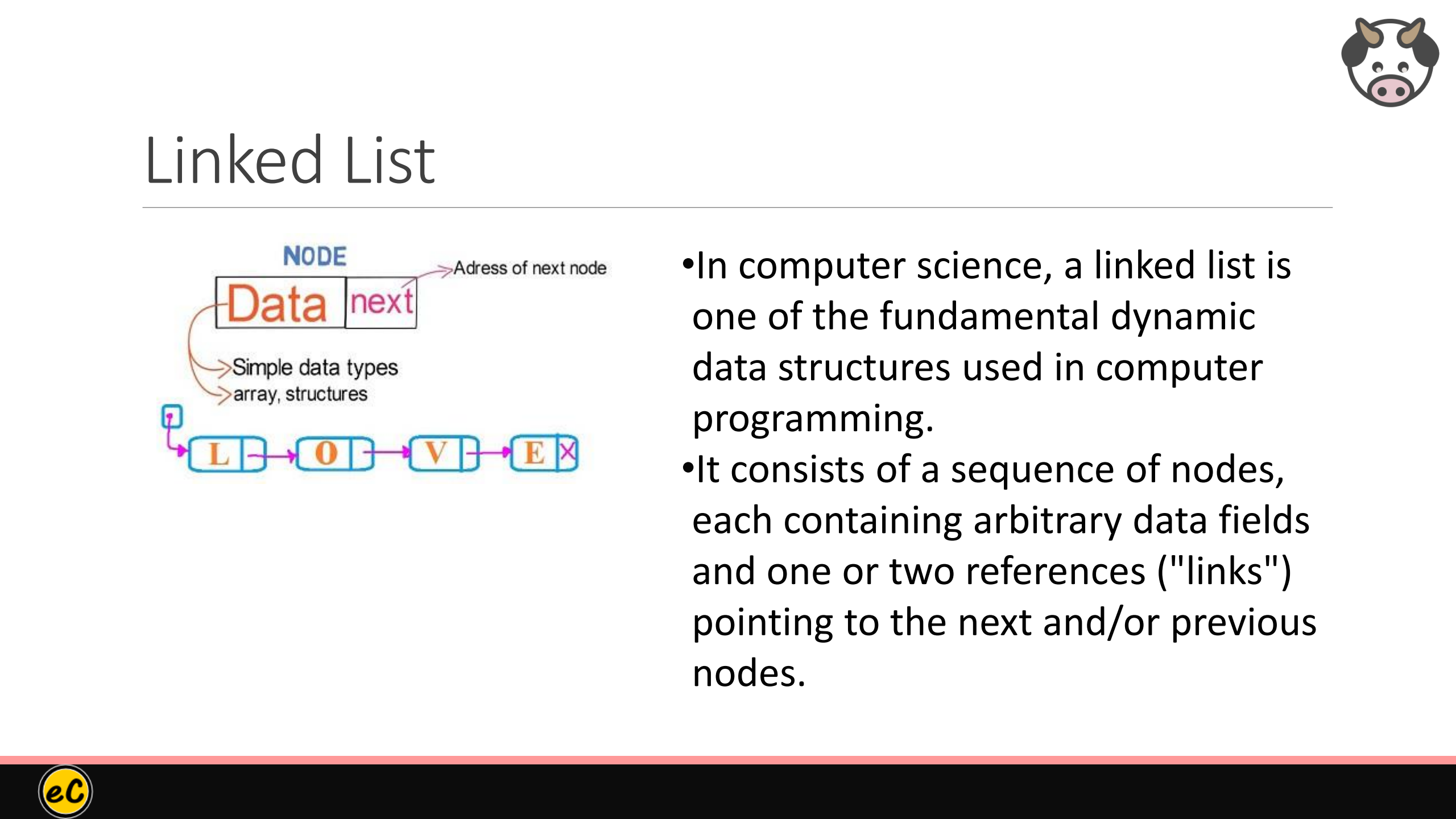
Data Structures by Java Language

- Array / Arraylist (Non-linked Structure)
- Linked Lists:
  - Single linked lists
  - Double linked lists
  - Circular linked lists
- Tree
  - Binary Tree
  - Heap Tree
  - AVL Tree
- Graph
  - Directed Graph
  - Un-directed Graph



Array

- In computer programming, a group of homogeneous elements of a specific data type is known as an array , one of the simplest data structures.
- Arrays hold a series of data elements, usually of the same size and data type.
- An array element is accessed by writing the name of the array followed by the position (index) in square brackets.
- Most programming languages have a built-in array data type.



Linked List

- In computer science, a linked list is one of the fundamental dynamic data structures used in computer programming.
- It consists of a sequence of nodes, each containing arbitrary data fields and one or two references ("links") pointing to the next and/or previous nodes.

Linked List

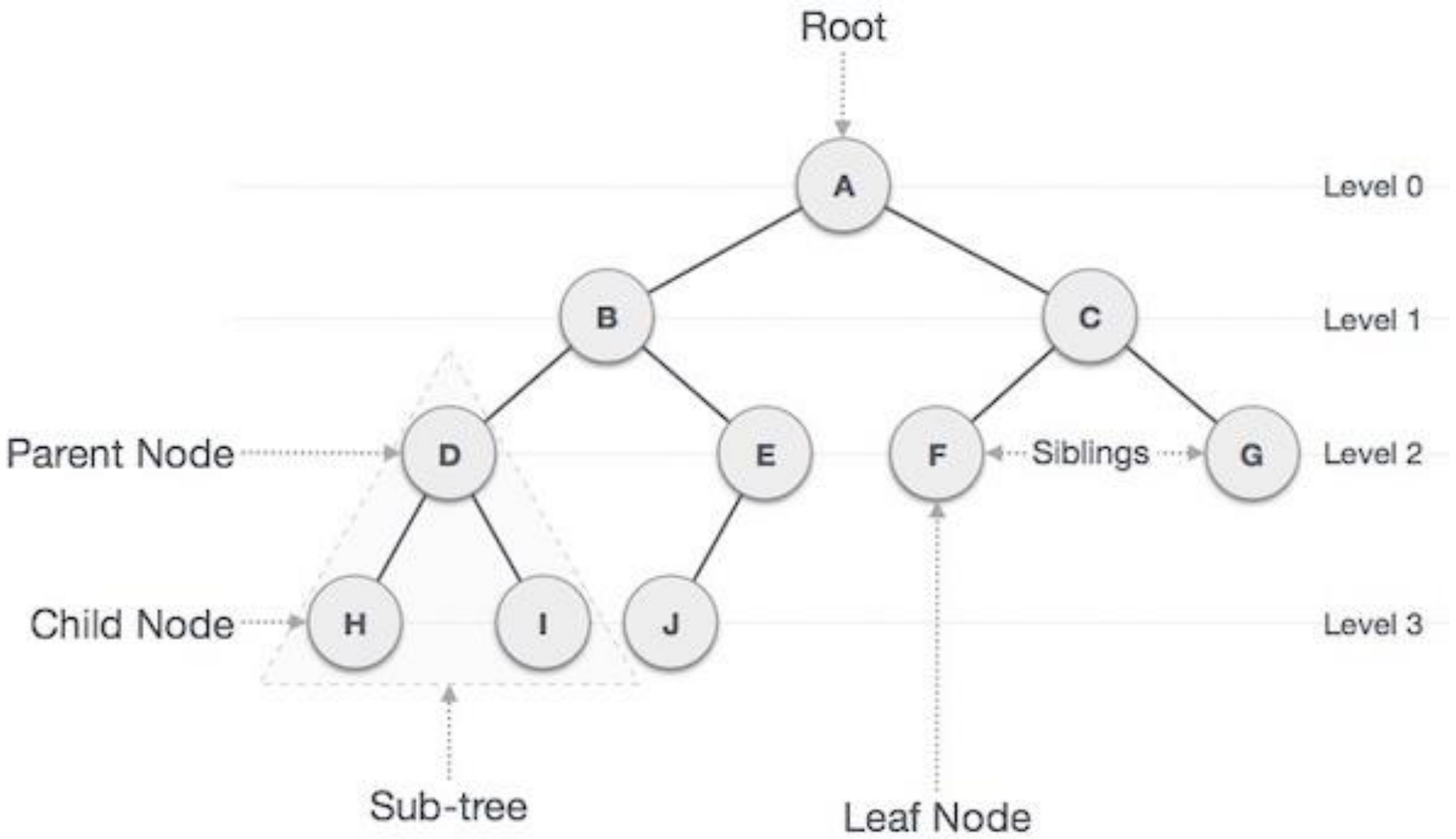
Some common examples of a linked list...

- Hash tables use linked lists for collission resolution..
- Any "File Requester" dialog uses a linked list.
- Binary Trees , Stacks and Queues can be implemented with a doubly linked list.
- Relational Databases (e.g. Microsoft Access).

A linked list is a self-referential data type because it contains a link to another data of the same type. Linked lists permit insertion and removal of nodes at any point in the list in constant time, but do not allow random access.

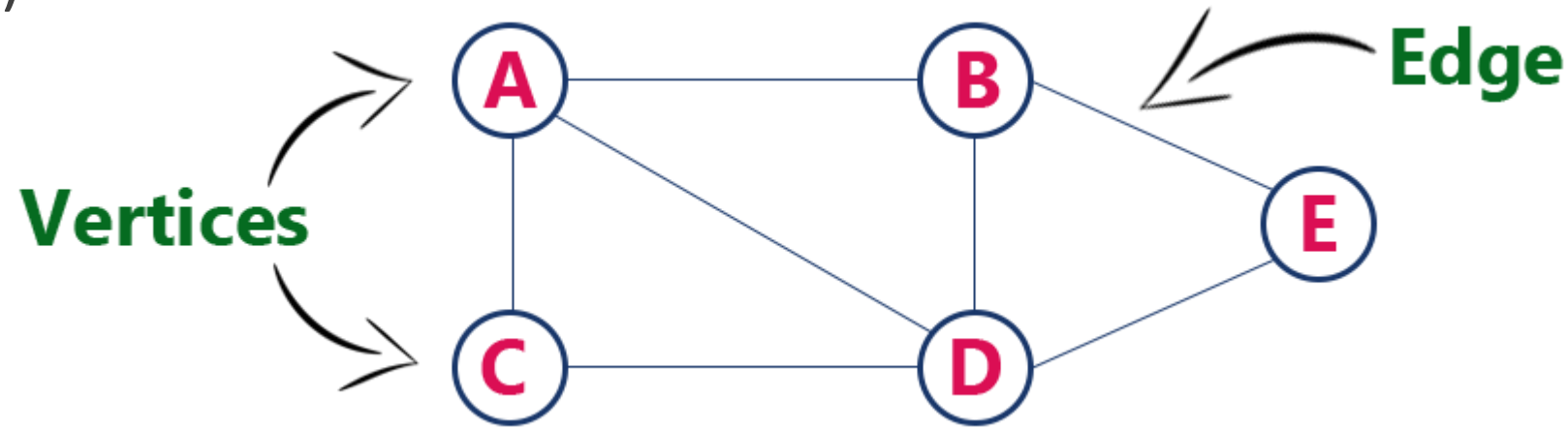
Tree

- A tree is hierarchical collection of nodes. One of the nodes, known as the root, is at the top of the hierarchy. Each node can have at most one link coming into it. The node where the link originates is called the parent node. The root node has no parent. The links leaving a node (any number of links are allowed) point to child nodes. Trees are recursive structures. Each child node is itself the root of a sub tree. At the bottom of the tree are leaf nodes, which have no children.


Root
A
B
C
D
E
F
G
H
I
J
Level 0
Level 1
Level 2
Level 3
Parent Node
Siblings
Child Node
Sub-tree
Leaf Node

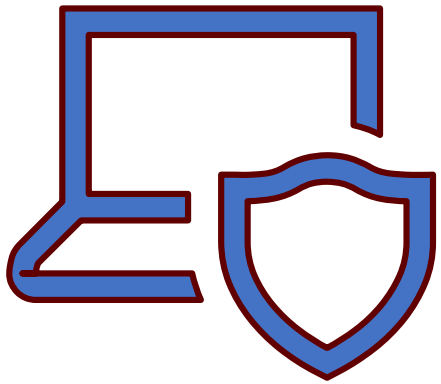
Graph

- Graph structures represents hierarchial relationship between individual data elements.
- Graphs are nothing but trees with certain restrictions removed.
- A graph consists of a set of nodes (or Vertices) and a set of arc (or edge).



Abstract Data Types

SECTION 3

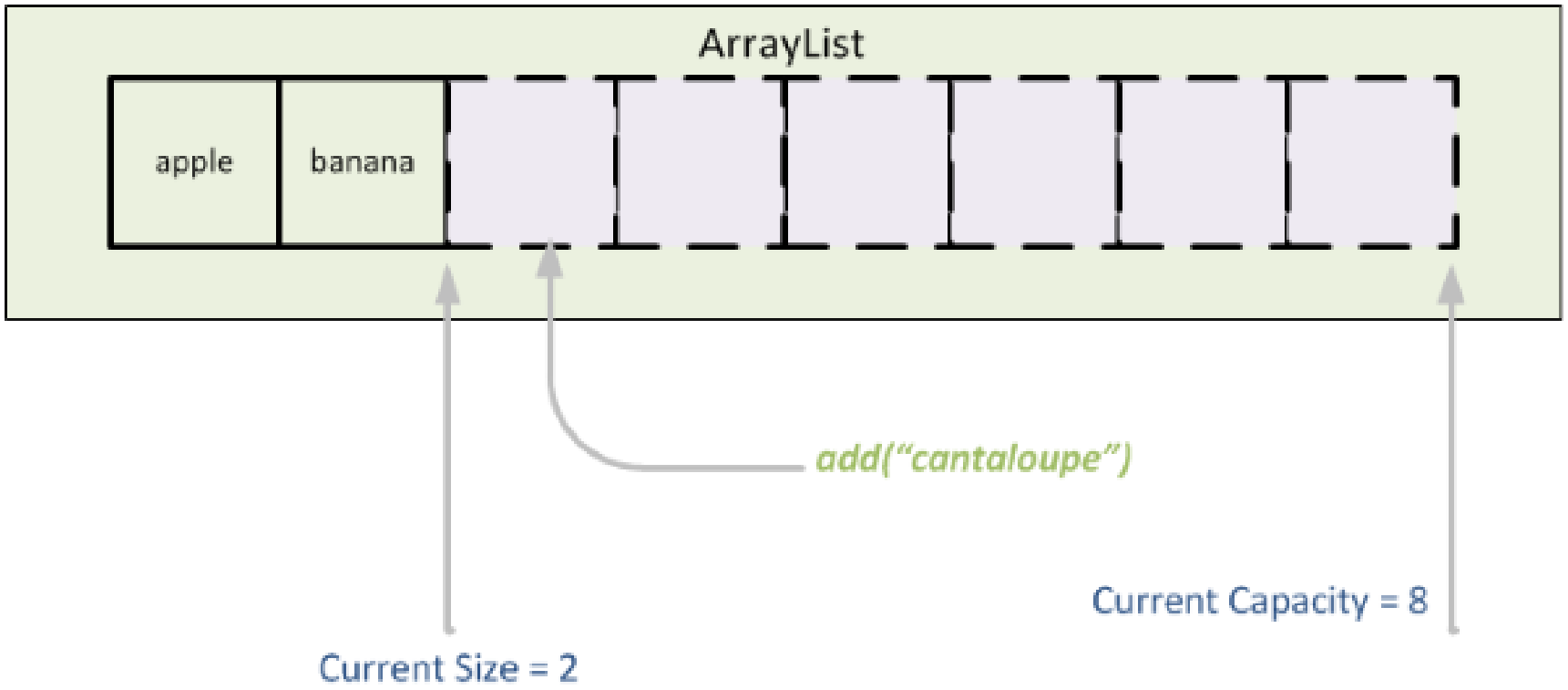


1-D Structures

SUB-SECTION 1

ArrayList

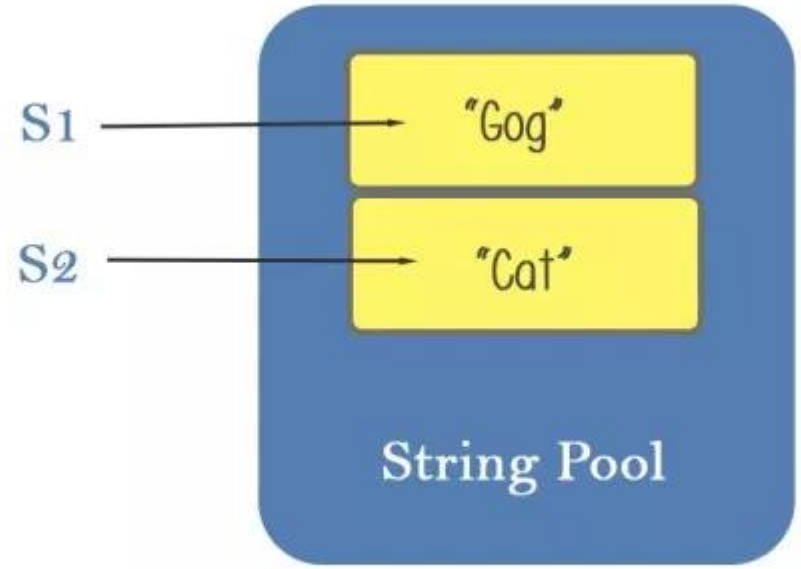
ArrayList class is a resizable-array implementation of the List interface. It implements all optional list operations and permits all elements.



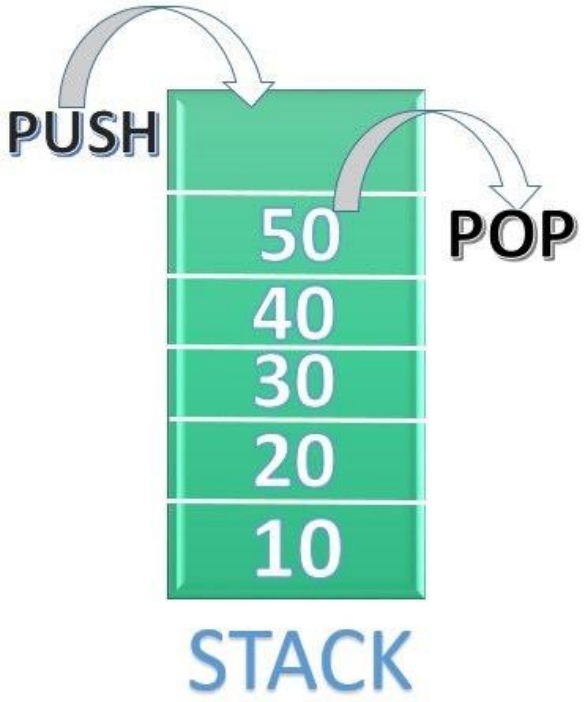
String

String class is used to create and manipulate strings.

- Array of Characters, Linked List of Characters



Stack



- A stack is a linear Structure in which item may be added or removed only at one end. There are certain frequent situations in computer science when one wants to restrict insertions and deletions so that they can take place only at the beginning or end of the list, not in the middle.
- A stack is a list of elements in which an elements may be inserted or deleted only at one end, called the Top. This means, in particular, the elements are removed from a stack in the reverse order of that which they are inserted in to the stack. The stack also called **"last-in first -out (LIFO) "** list.

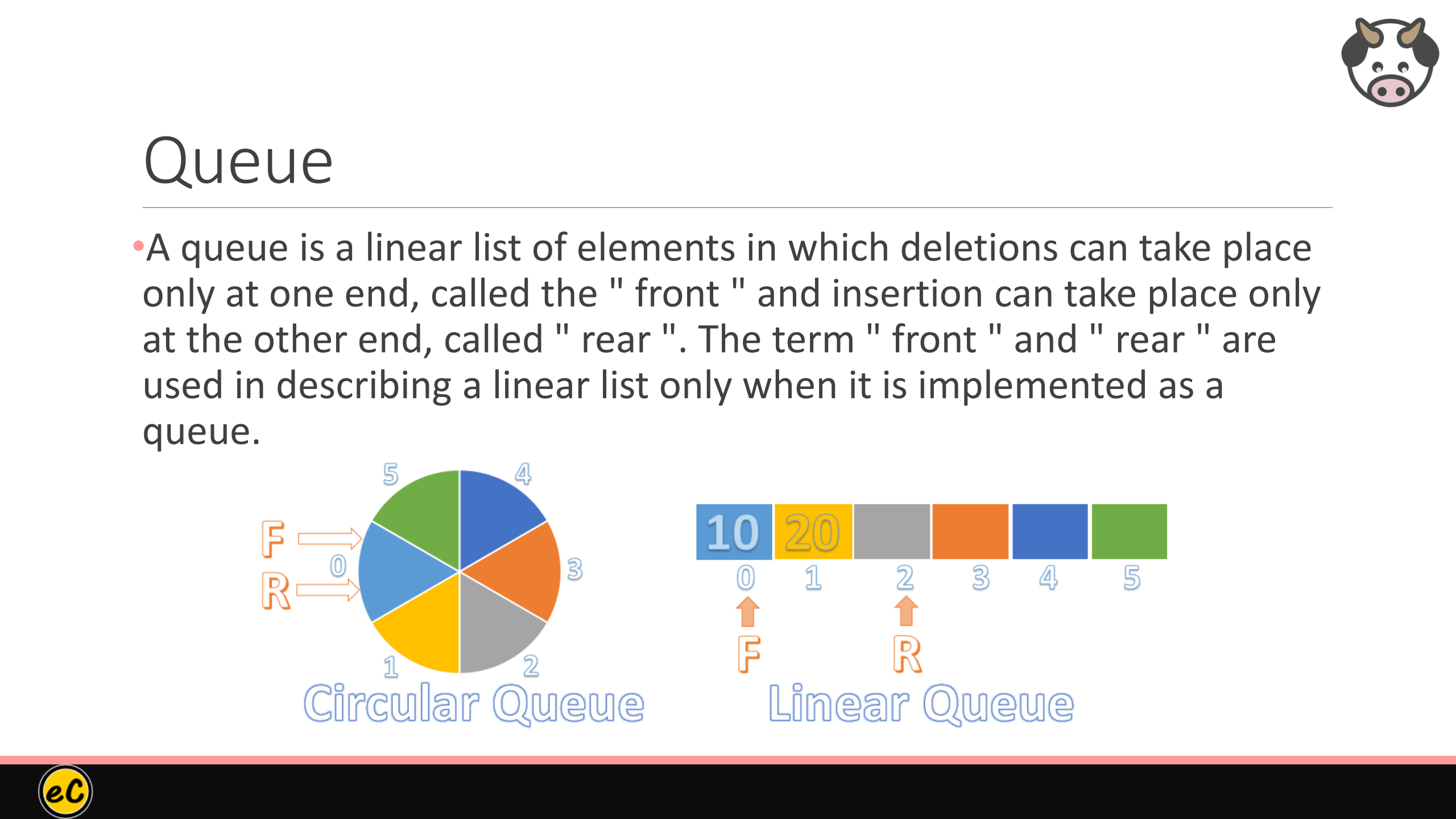
Primary operations defined on a stack

PUSH : Add an element at the top of the list.

POP : Remove an element from the top of the list.

Example :**Practical daily life :** A pile of heavy books kept in a vertical box,dishes kept one on top of another.

In computer world : In processing of subroutine calls and returns , there is an explicit use of stack of return addresses.



Queue

- A queue is a linear list of elements in which deletions can take place only at one end, called the " front " and insertion can take place only at the other end, called " rear ". The term " front " and " rear " are used in describing a linear list only when it is implemented as a queue.

Circular Queue

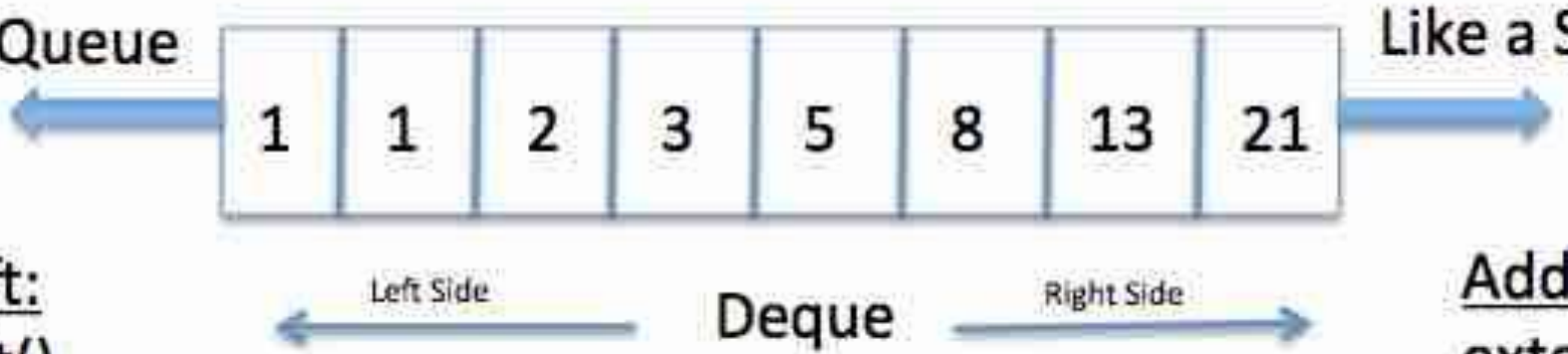
Linear Queue

Queue

- Queues are also called " first-in first-out " (FIFO) list. Since the first element in a queue will be the first element out of the queue. In other words, the order in which elements enter in a queue is the order in which they leave.
- The real life example: the people waiting in a line at Railway ticket Counter form a queue, where the first person in a line is the first person to be waited on. An important example of a queue in computer science occurs in time sharing system, in which programs with the same priority form a queue while waiting to be executed.

Deque

- **deque** is a container which can hold a collection of python objects.
- A deque is a **double-ended queue** on which elements can be added or removed from either side - that is on left end or right end, head or tail.
- A **deque** is like both a **stack** and **queue**.
- On a deque, adding an element or removing an element on either side of a deque instance takes constant time O(1).

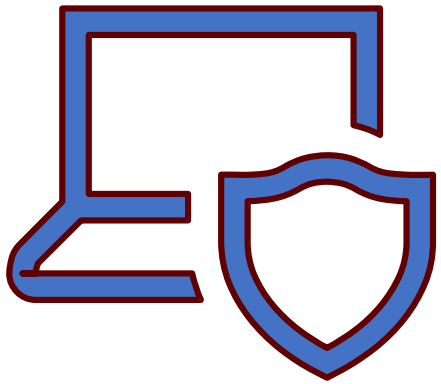


Add to left:
extendleft()
appendleft()

Remove from left:
popleft()

Add to right:
extend ()
append ()

Remove from right:
pop ()

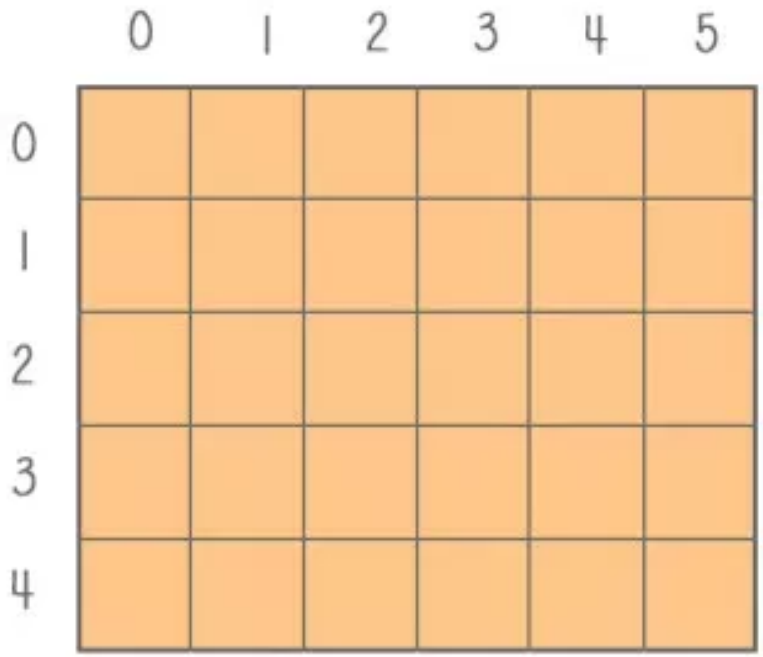


2-D Structures

SUB-SECTION 2

Matrix

A matrix is a double-dimensioned array. It makes use of two indexes rows and columns to store data.



Sparse Matrix

Array Implementation

rows = 5
columns = 5
size = 12
capacity = 16

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0.2 & 0 & 0 & 0 & 0.6 & 0 \\ 0 & 1.1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2.2 & 0 & 0 & 2.5 & 2.6 & 0 \\ 3.0 & 0 & 0 & 0 & 0 & 0 & 0 & 3.7 \\ 0 & 0 & 0 & 0 & 4.4 & 0 & 0 & 0 \\ 0 & 0 & 5.2 & 0 & 0 & 5.5 & 5.6 & 0 \\ 0 & 6.1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 7.2 & 7.3 & 7.4 & 0 & 0 & 7.7 \end{pmatrix}$$

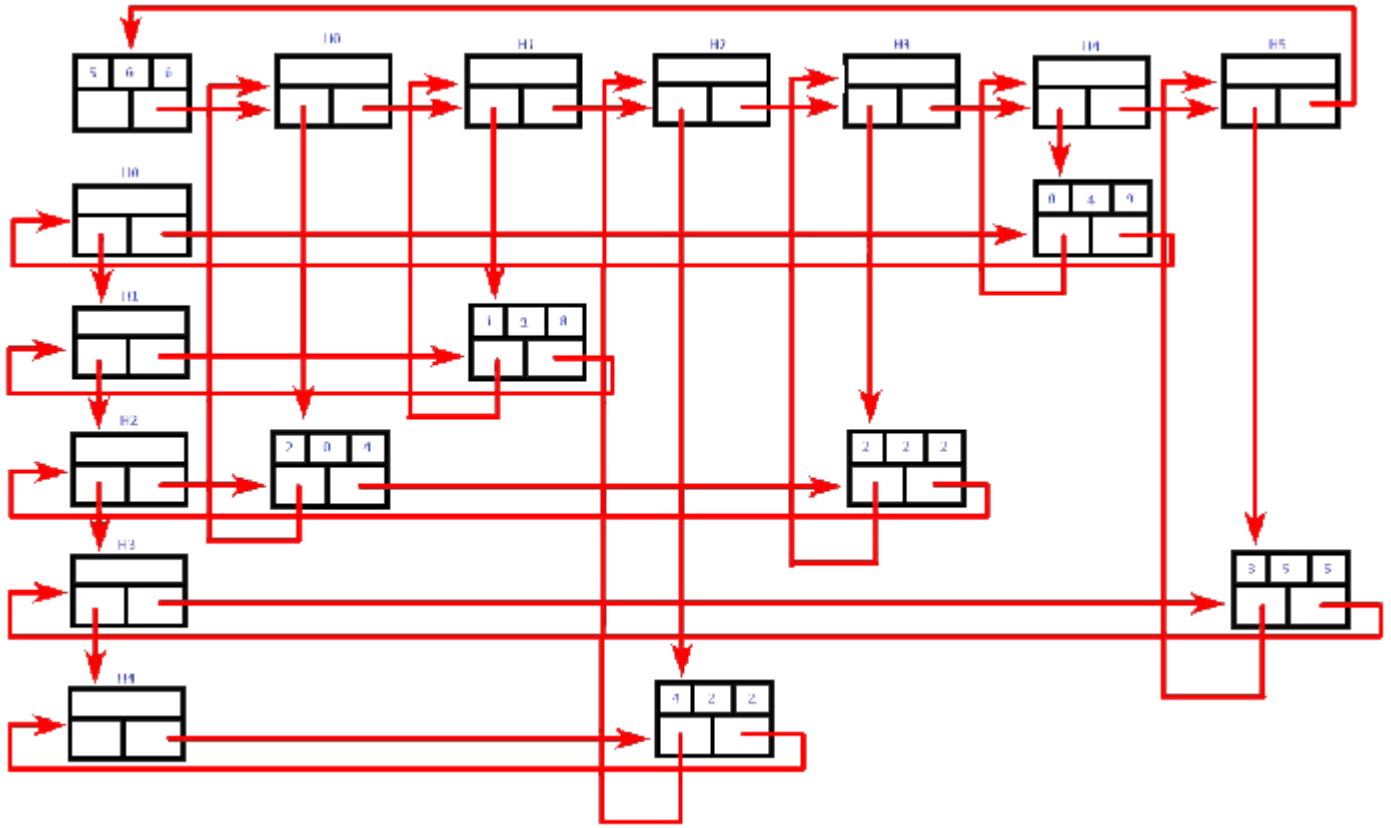
diagonal	0	1.1	2.2	0	4.4	5.5	0	7.7

row_index	0	2	2	4	6	6	8	9	12

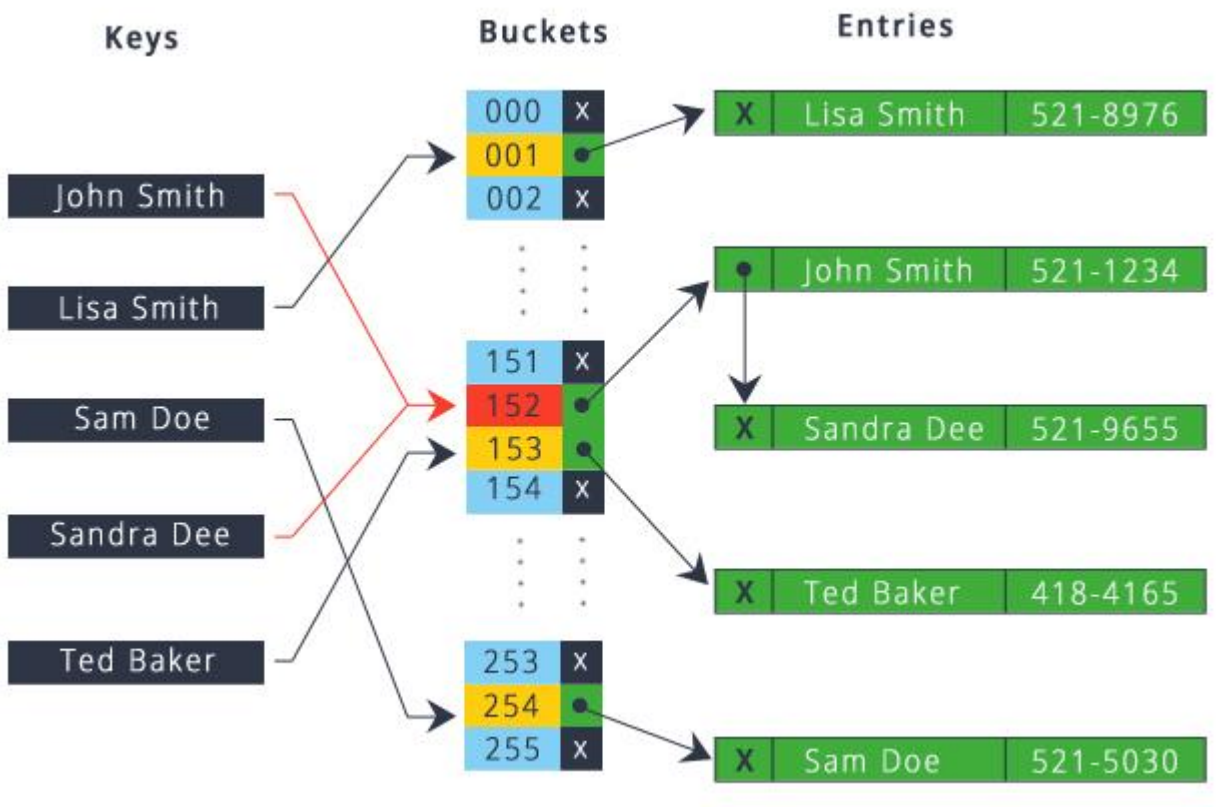
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
column_index	2	6	5	6	0	7	2	6	1	2	3	4				
off_diagonal	0.2	0.6	2.5	2.6	3.0	3.7	5.2	5.6	6.1	7.2	7.3	7.4				

Sparse Matrix

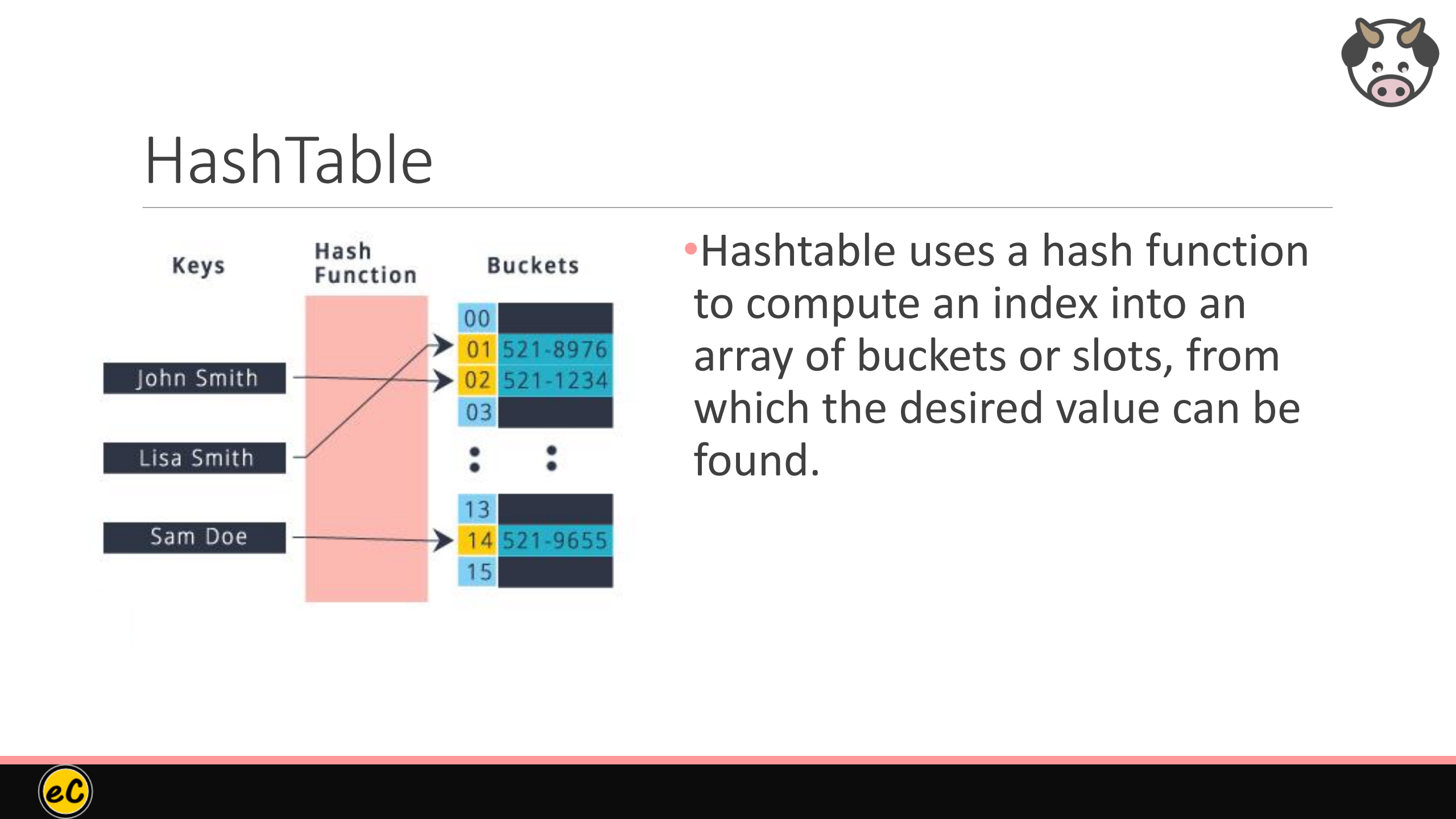
Implementation by Linked Structure



HashMap

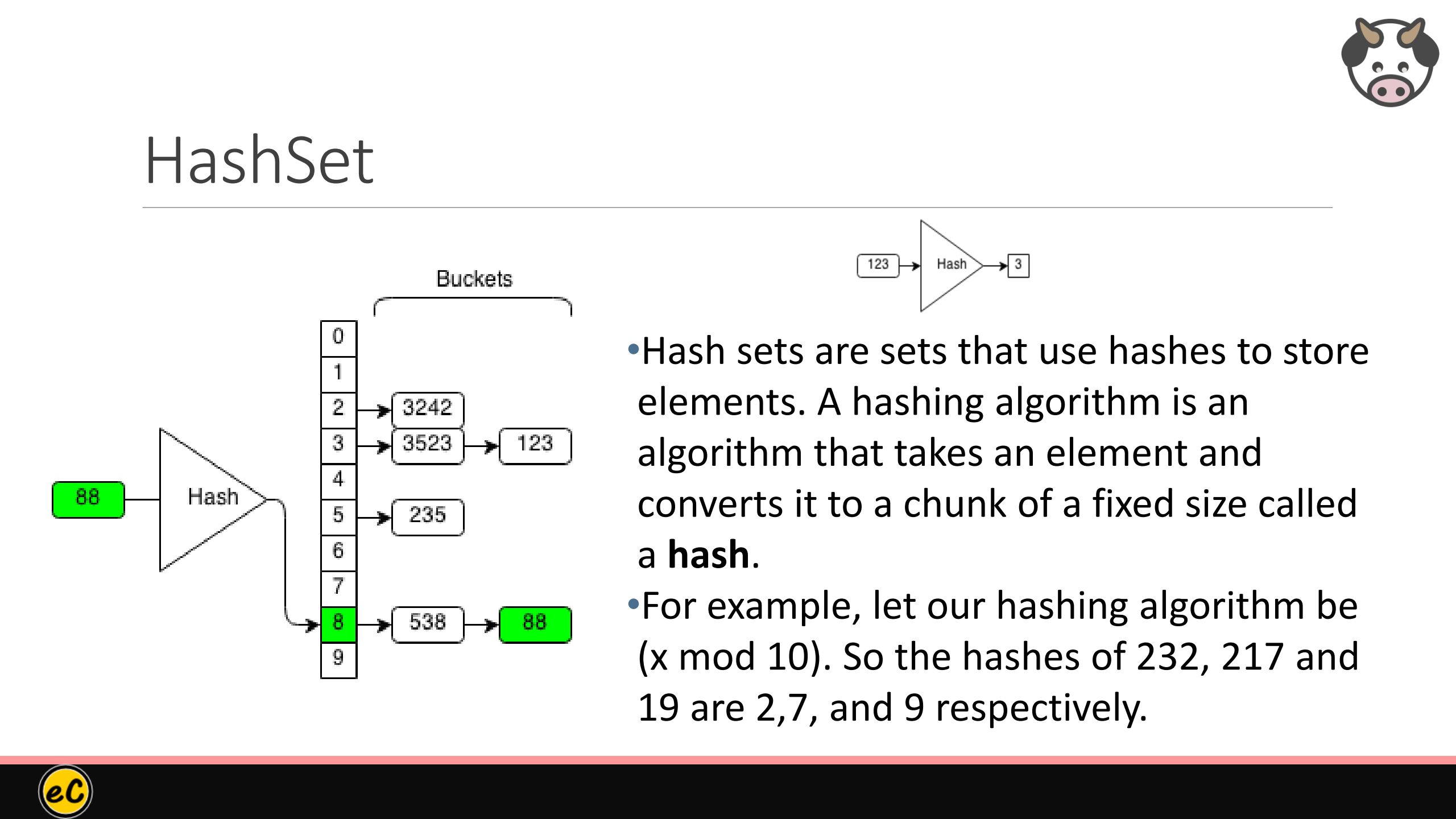


- HashMap calls hashCode method on the key object and applies returned hash value to its own static hash function to find a bucket location where keys and values are stored in form of a nested class called Entry (Map.Entry)



HashTable

- Hashtable uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found.

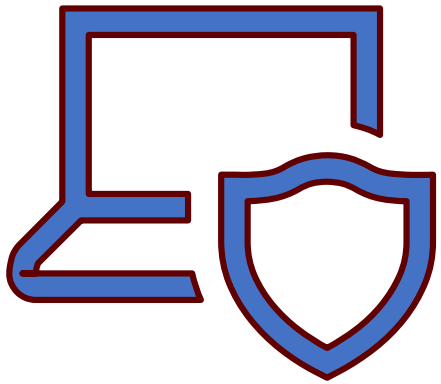


HashSet

- Hash sets are sets that use hashes to store elements. A hashing algorithm is an algorithm that takes an element and converts it to a chunk of a fixed size called a **hash**.
- For example, let our hashing algorithm be (x mod 10). So the hashes of 232, 217 and 19 are 2,7, and 9 respectively.

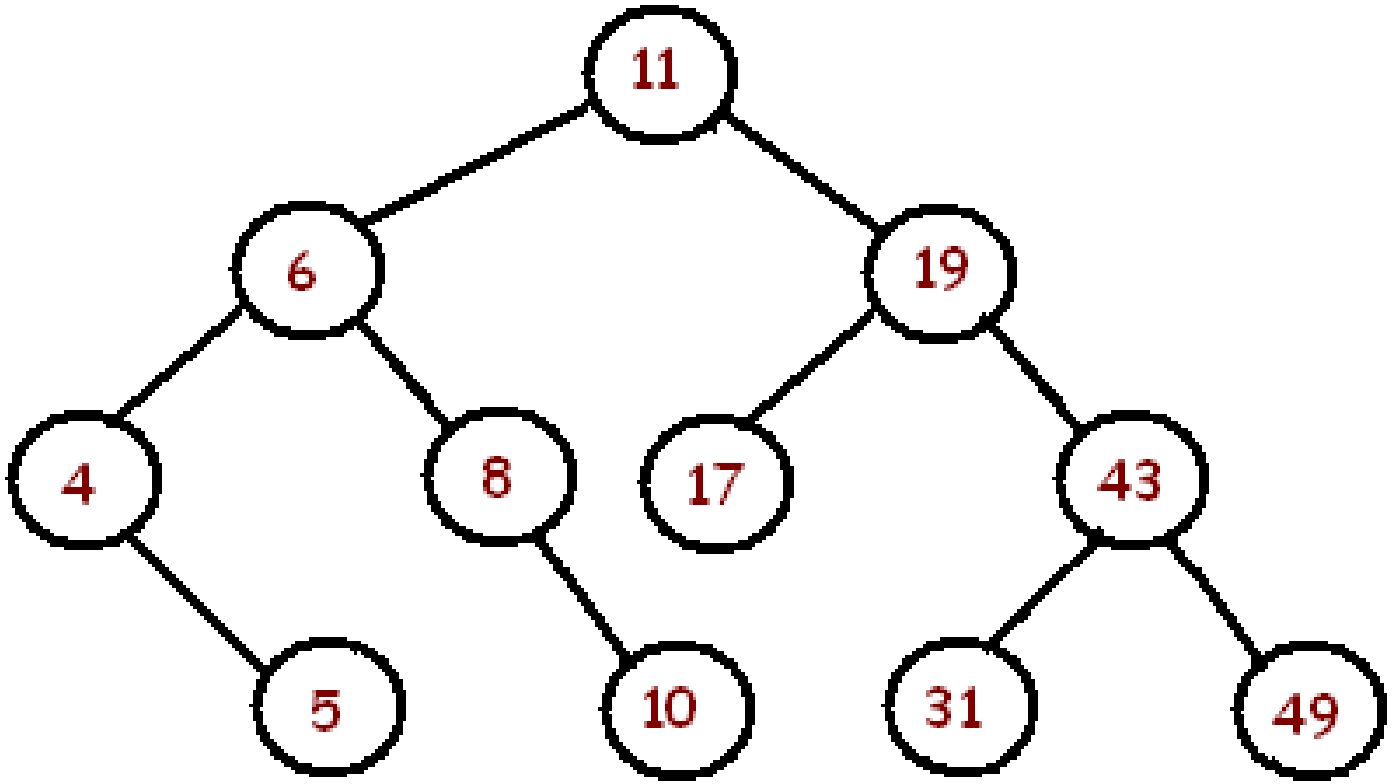
HashSet

- For every element in a hash set, the hash is computed and elements with the same hash are grouped together. This group of similar hashes is called a **bucket** and they are usually stored as **linked lists**.
- If we want to check if an element already exists within the set, we first compute the hash of the element and then search through the bucket associated with the hash to see if the element is contained.

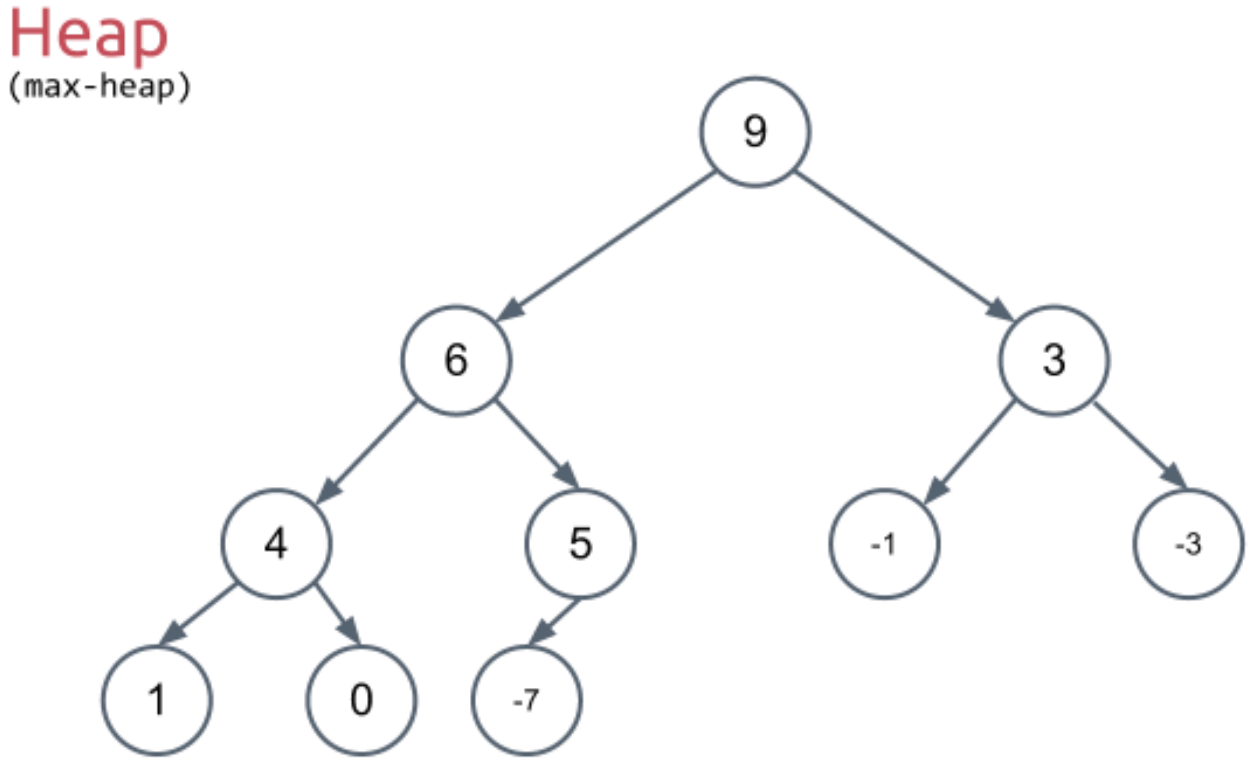


Tree Structures

SUB-SECTION 3



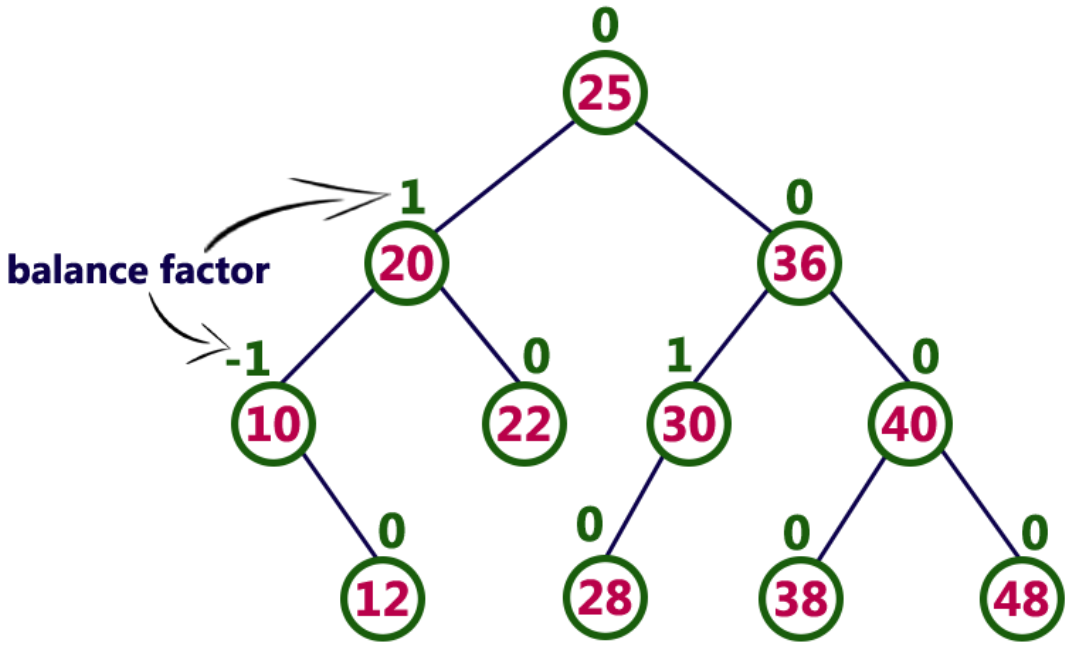
Binary tree



Heap Tree (Max Heap – Priority Queue)

PriorityQueue

- **PriorityQueue** class is the implementation of Queue, with each element having a priority associated with it. The elements of the priority queue are ordered according to their natural ordering, or by a Comparator provided at queue construction time.
- Implementation: Array, Linked List, and HashTree

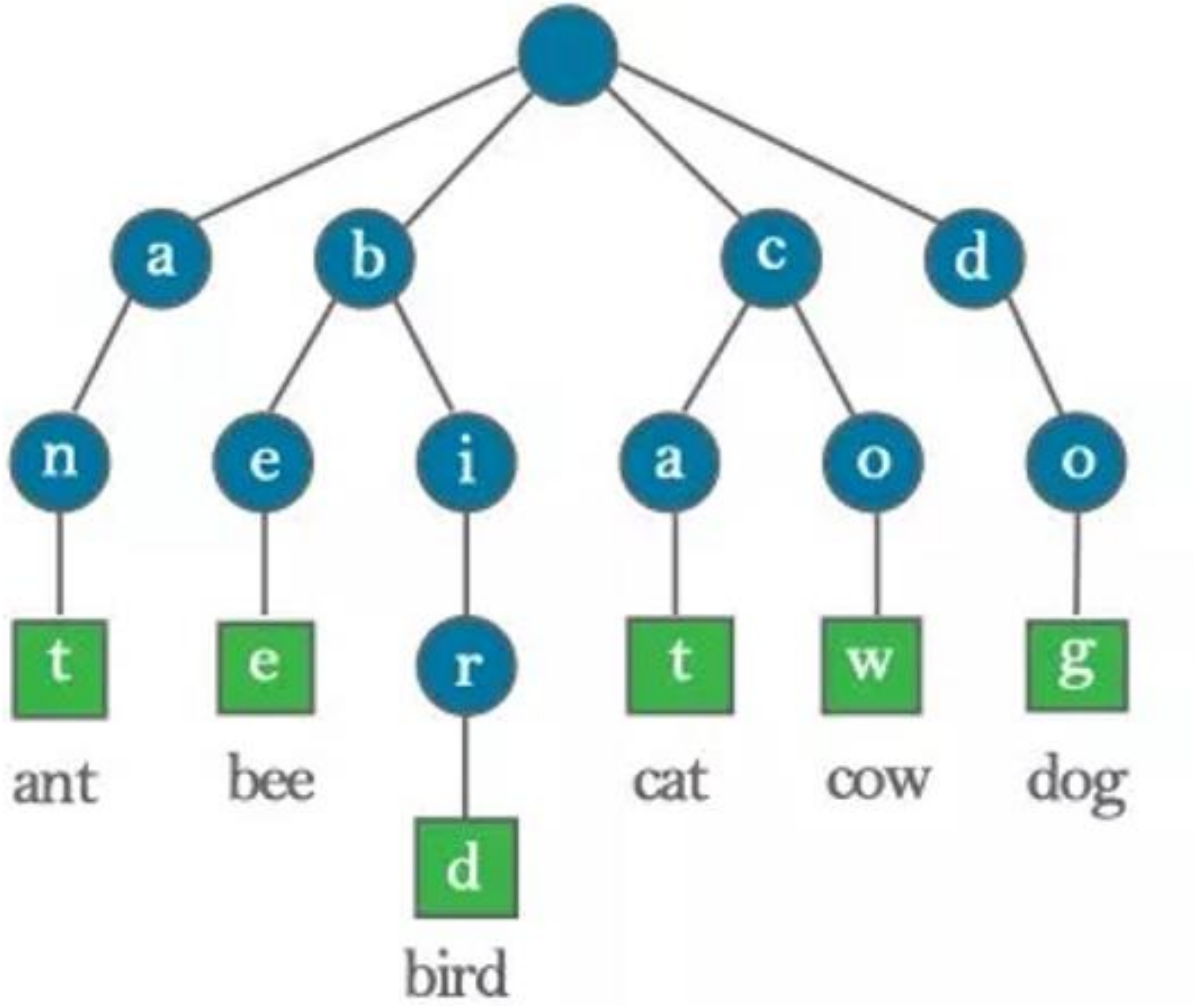


AVL Tree

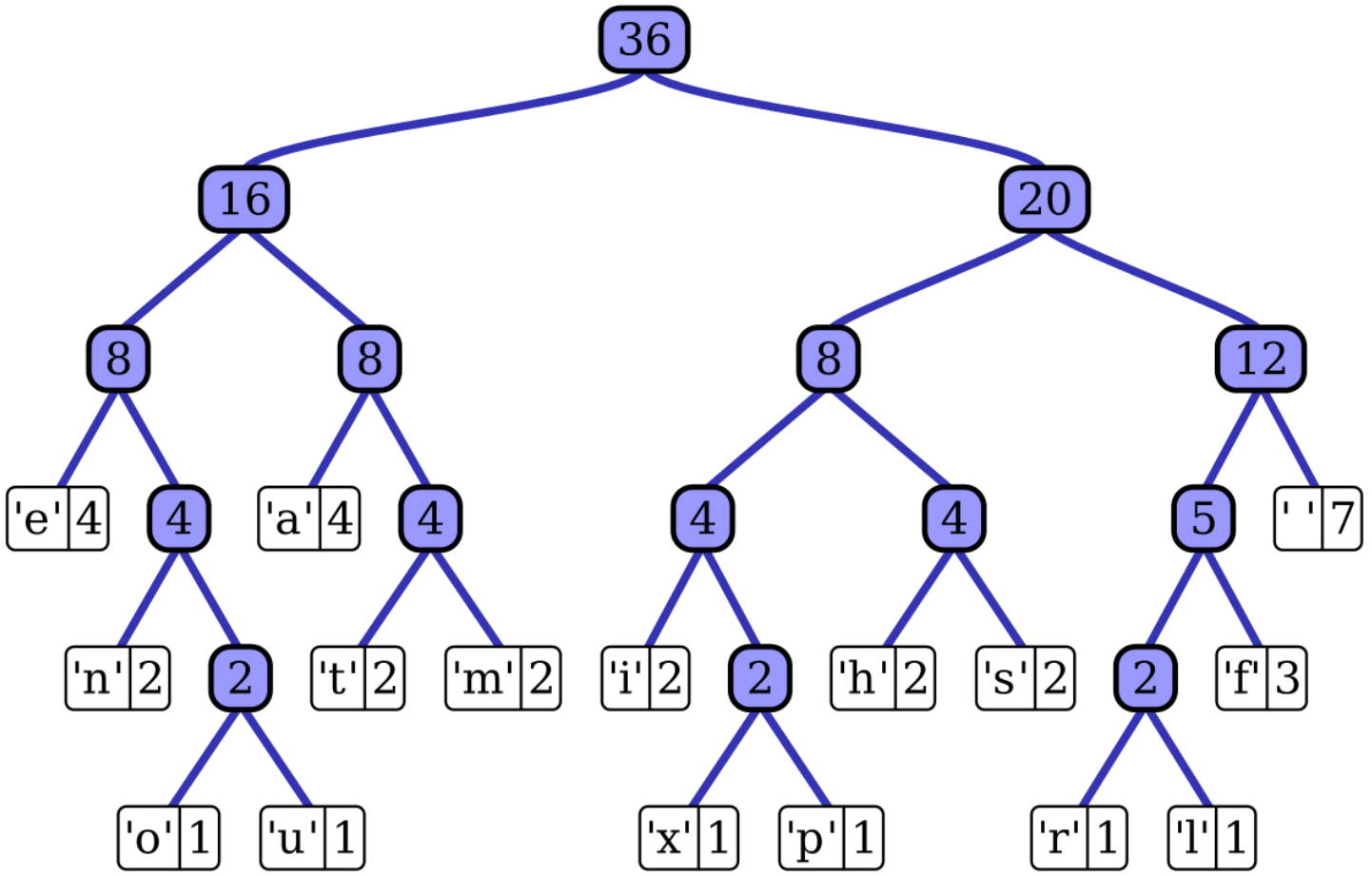
Trie

Huffman Tree (Code Book)

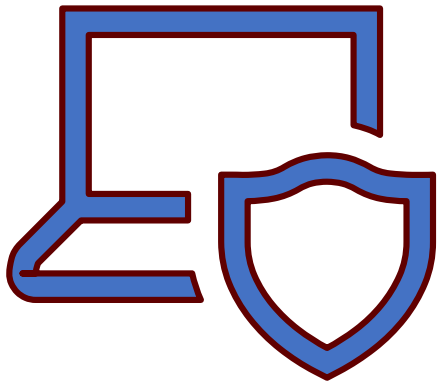
- A Trie is a tree. In a trie, every node (except the root node) stores one character or a digit.
- By traversing the trie down from the root node to a particular node **n**, a common prefix of characters or digits can be formed which is shared by other branches of the trie as well.



Trie

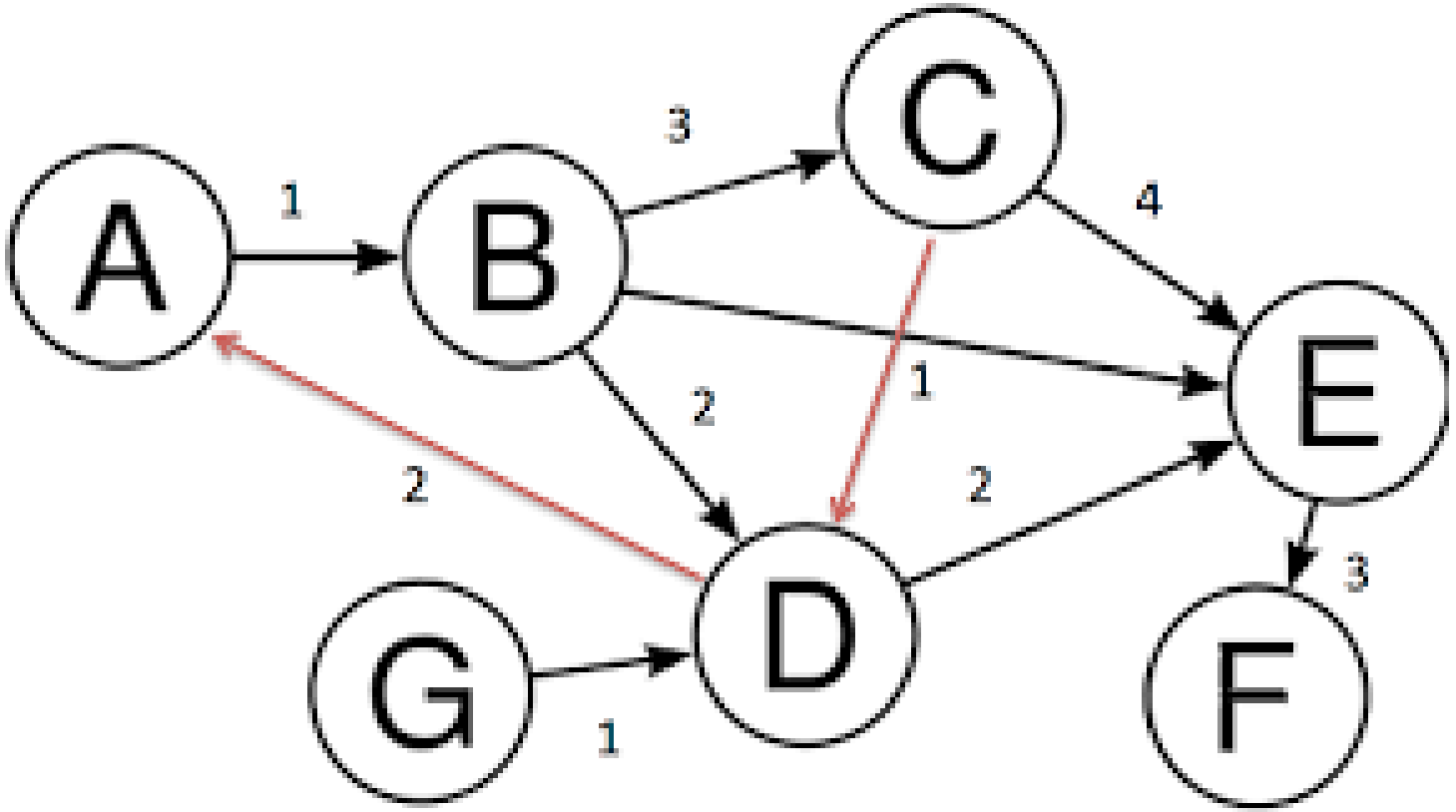


Huffman Tree

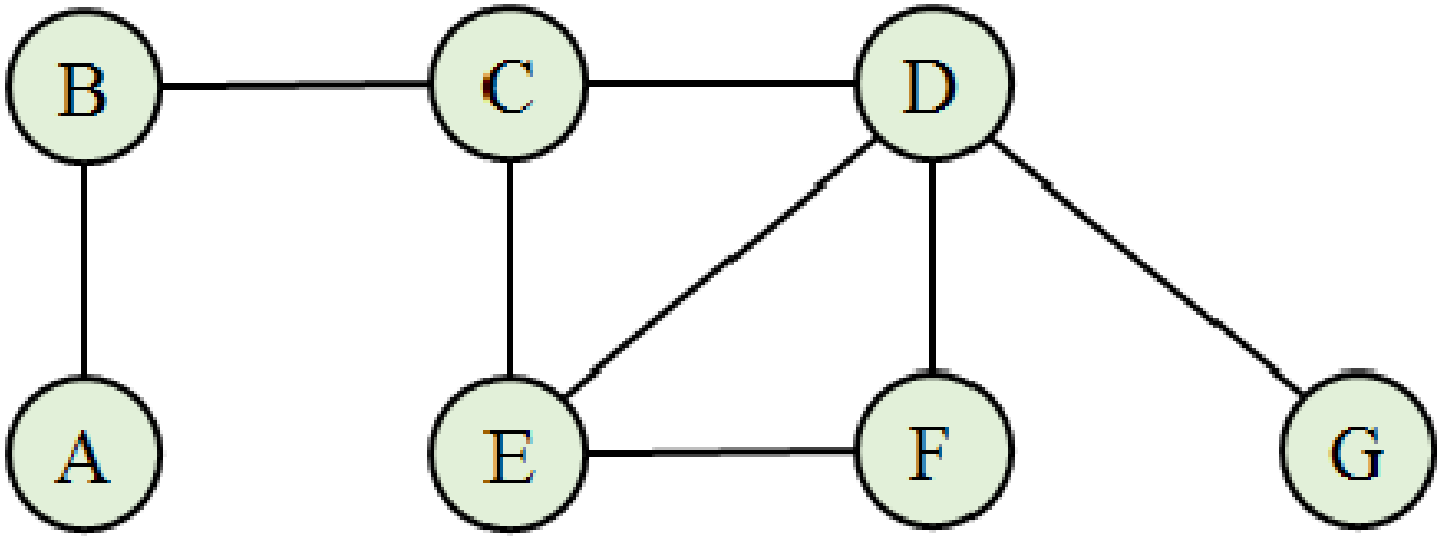


Graph Structures

SUB-SECTION 3



Directed Graph



Un-directed Graph

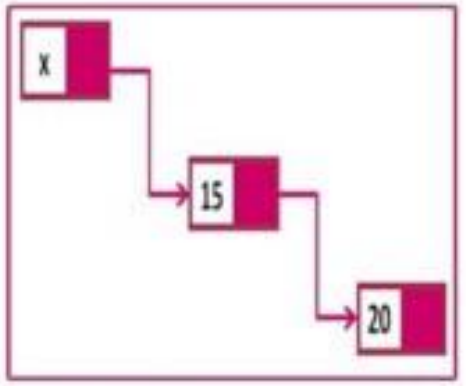
Design of a Linked Structure

SECTION 4

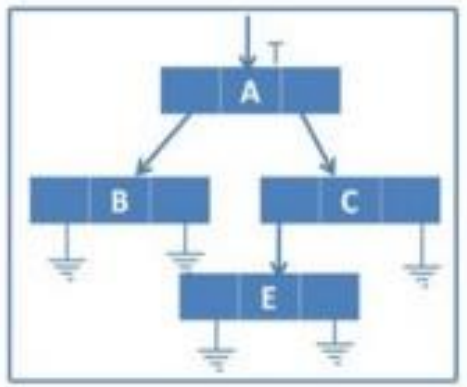
The Design of a Linked Structure

- Design goals of the linked structure (proper data structure for problem solving).
- Design Building Blocks (Nodes)
- Design of Construction and Destruction Operations of the Data Structure.
- Traversal of the Data Structure
- Evaluation of the Data Structure

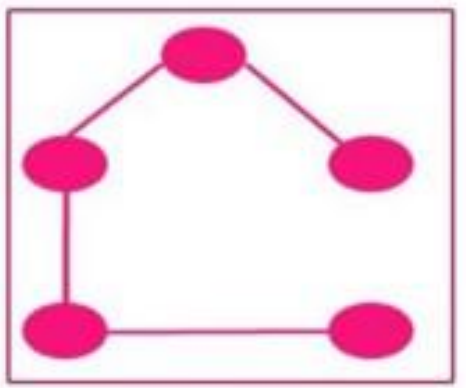
Sorting



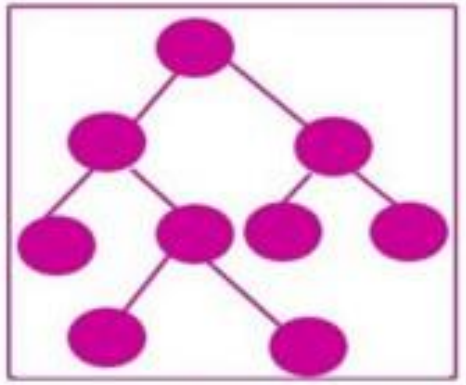
Link list



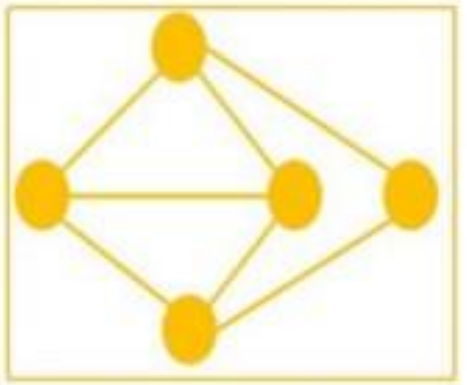
list



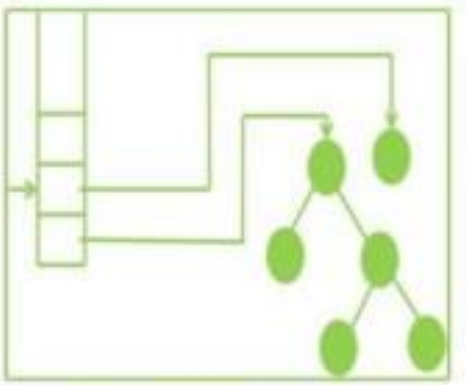
spanning tree



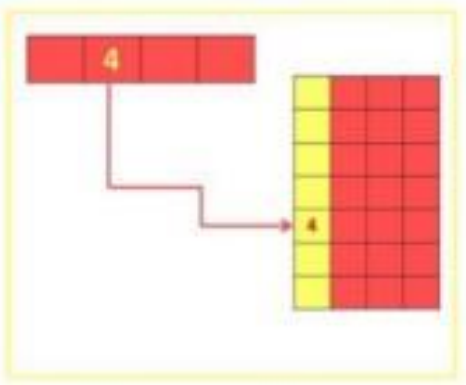
Tree



Graph



Stack



Hashing

Data Structure	Time Complexity			
	Average			
	Access	Search	Insertion	Deletion
Array	O(1)	O(n)	O(n)	O(n)
Stack	O(n)	O(n)	O(1)	O(1)
Queue	O(n)	O(n)	O(1)	O(1)
Singly-Linked List	O(n)	O(n)	O(1)	O(1)
Doubly-Linked List	O(n)	O(n)	O(1)	O(1)
Skip List	O(log(n))	O(log(n))	O(log(n))	O(log(n))
Hash Table	N/A	O(1)	O(1)	O(1)
Binary Search Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))
Cartesian Tree	N/A	O(log(n))	O(log(n))	O(log(n))
B-Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))
Red-Black Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))
Splay Tree	N/A	O(log(n))	O(log(n))	O(log(n))
AVL Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))
KD Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))

Goal of This Course

- Cover the basic linked structures in Java language to support the target data structures.
- Provide baseline design for all linked structures. (They may need to be customized or generalized for different project purposes).
- Each chapter will cover all the topics of a basic linked structure.

Practice: Broken Necklace

SECTION 5

Problem Statement

- You have a necklace of N red, white, or blue beads (3<=N<=350) some of which are red, others blue, and others white, arranged at random. Here are two examples for n=29:

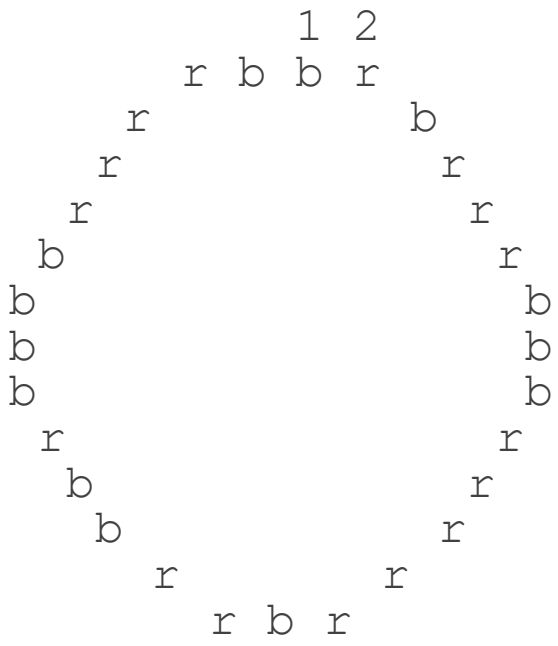


Figure A

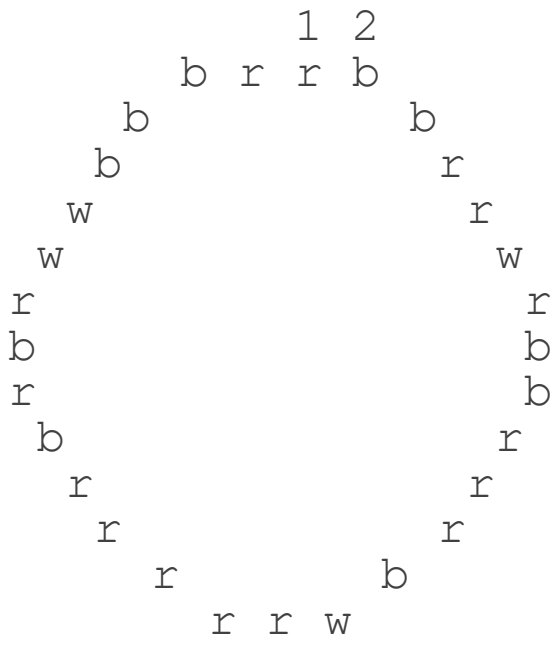


Figure B

r red bead, b blue bead, w white bead

Problem Statement

- The beads considered first and second in the text that follows have been marked in the picture.
- The configuration in Figure A may be represented as a string of b's and r's, where b represents a blue bead and r represents a red one, as follows: brbrrrbbbrrrrrbrrbbrbbbbrrrrb .

Problem Statement

- Suppose you are to break the necklace at some point, lay it out straight, and then collect beads of the same color from one end until you reach a bead of a different color, and do the same for the other end (which might not be of the same color as the beads collected before this).
- Determine the point where the necklace should be broken so that the most number of beads can be collected. No bead can be collected more than once.

INPUT FORMAT (file beads.in):

- Line 1: N, the number of beads
- Line 2: a string of N characters, each of which is r, b, or w

SAMPLE INPUT:

```
29
wwwbbrwrbrbrrbrbrwrwwrbwrwrrb
```

OUTPUT FORMAT (beads.out):

- A single line containing the maximum of number of beads that can be collected from the supplied necklace.

SAMPLE OUTPUT:

```
11
```

OUTPUT EXPLANATION

Consider two copies of the beads (kind of like being able to runaround the ends). The string of 11 is marked.

Two necklace copies joined here

```
                                  v
wwwbbrwrbrbrrbrbrbrwrwwrbwrwrrb|wwwbbrwrbrbrrbrbrbrwrwwrbwrwrrb
                        ******|*****
                            rrrrrb|bbbbb  <-- assignments
                       5xr .....#|#####  6xb
```

5+6 = 11 total

Nature of the Problem

- Duplicate the necklace to mimic wrapping around the beads
- The maximum of beads collected should be <= N
- Maximum number of beads collected will be in a run of
- w w w r w r r w r w w r w b w b w b b b w w w w

It does not matter whether this white bead is colored to blue or red

Main Idea

- Leading white or trailing white can be colored this. After coloring the leading and trailing whites, both ends of the beads should be solid color (either red or blue)
- Then, find a way to calculate the maximum run of a red-span joined with a blue span (whites belong to one end)

Run Intervals

- Each color of the beads may have a run interval. In this special session, we are going to demonstrate how to create run intervals.

Flatten the Circular Operation to A String

Original String :

wwwbb**r**wrbrbrrbrbrwrwwrbwrwrrbwwwbbrwrbrbrrbrbrwrwwrbwrwrrb

Run Intervals:

[[0, 3, w], [3, 5, b], [5, 6, r], [6, 7, w], [7, 8, r], [8, 9, b], [9, 10, r], [10, 11, b]
, [11, 13, r], [13, 14, b], [14, 15, r], [15, 16, b], [16, 17, r], [17, 18, w], [18, 19, r], [19, 21, w]
, [21, 22, r], [22, 23, b], [23, 24, w], [24, 25, r], [25, 26, w], [26, 28, r], [28, 29, b]
, [29, 32, w], [32, 34, b], [34, 35, r], [35, 36, w], [36, 37, r], [37, 38, b], [38, 39, r], [39, 40, b]
, [40, 42, r], [42, 43, b], [43, 44, r], [44, 45, b], [45, 46, r], [46, 47, w], [47, 48, r], [48, 50, w]
, [50, 51, r], [51, 52, b], [52, 53, w], [53, 54, r], [54, 55, w], [55, 57, r], [55, 58, b]
]

Finding the Maximum Cut

Original String :

w w w r w r r w r w w r w b w b w b b b w w w w