CS 91 USACO

Bronze Division

Unit 4: Basic Tree and Graphs

LECTURE 17: BASIC GRAPH THEORY (ADJACENCY MATRIX)

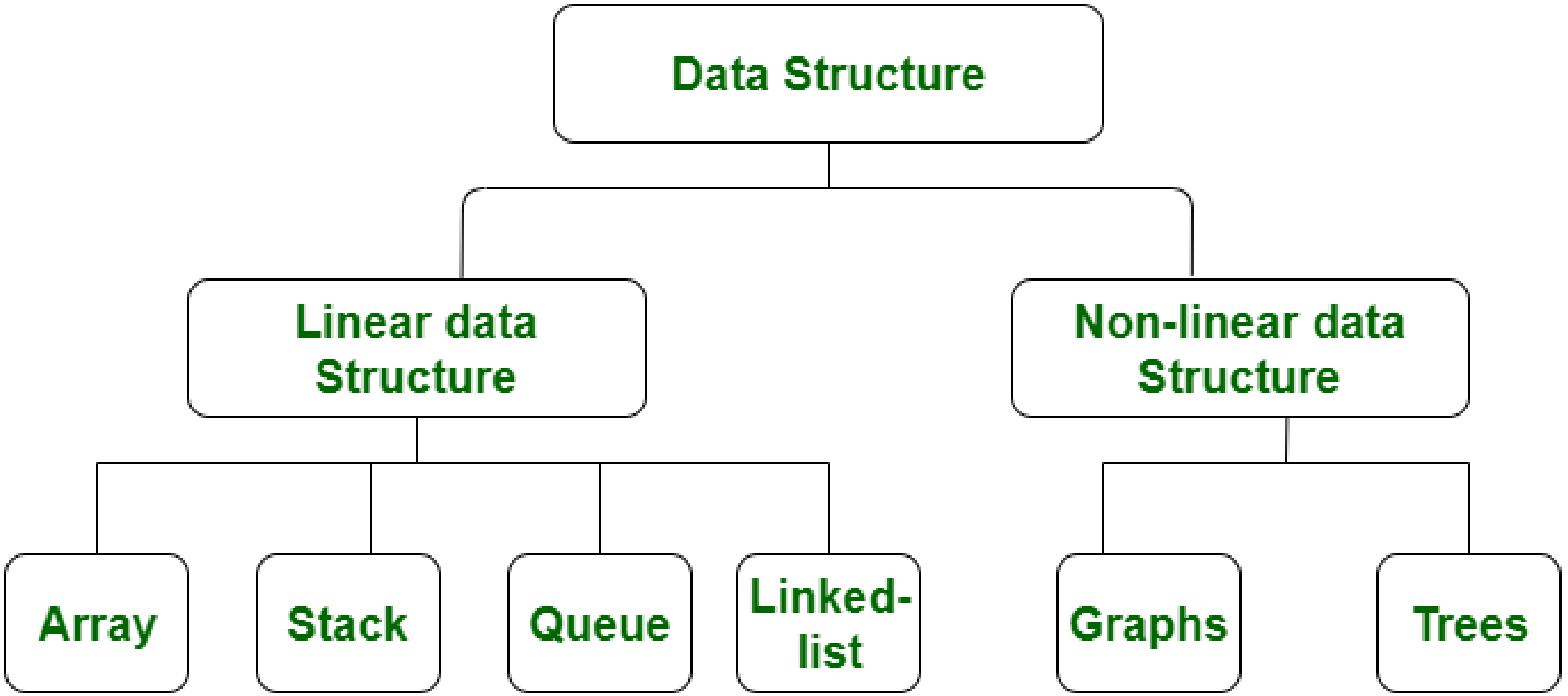
DR. ERIC CHOU IEEE SENIOR MEMBER

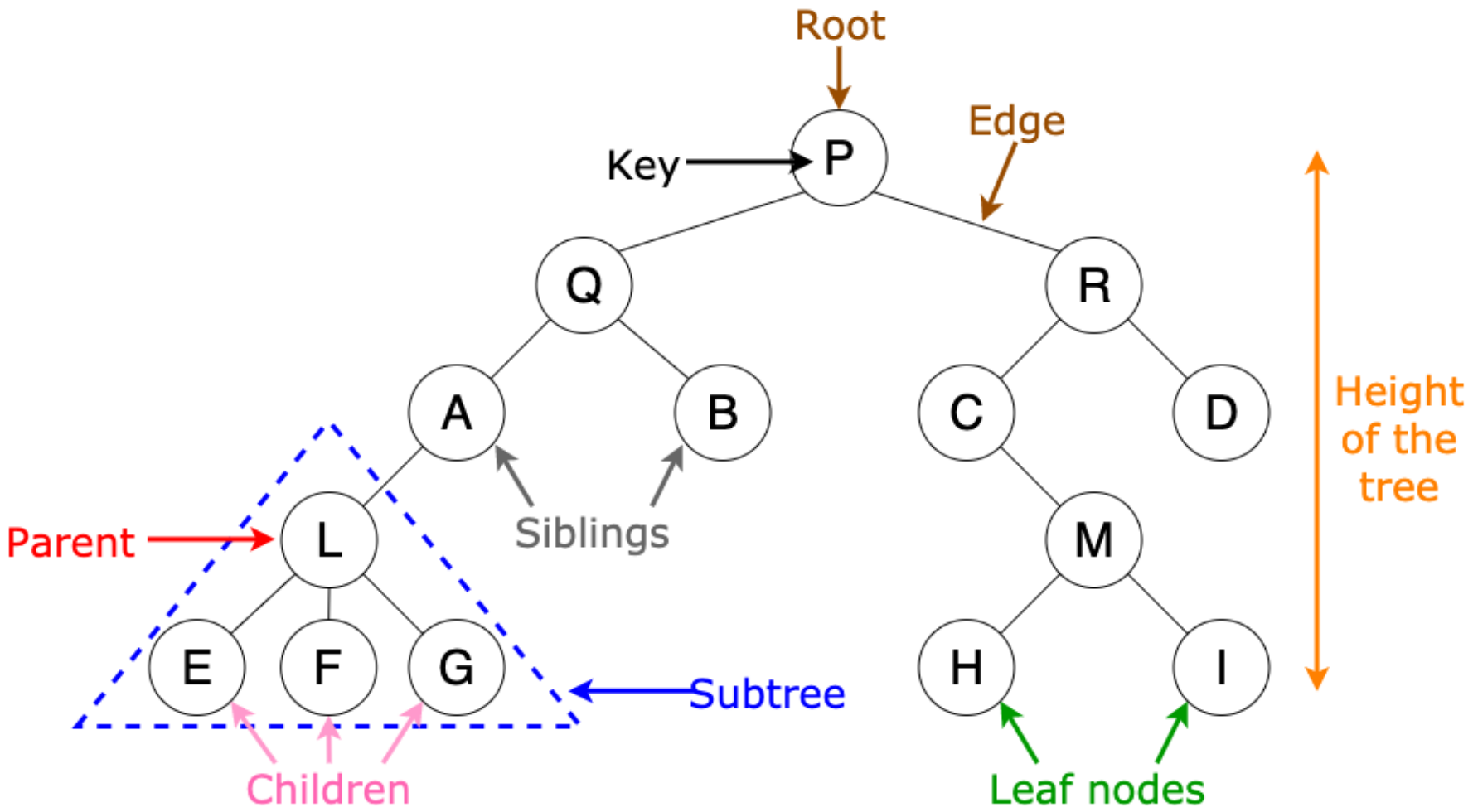
Objectives

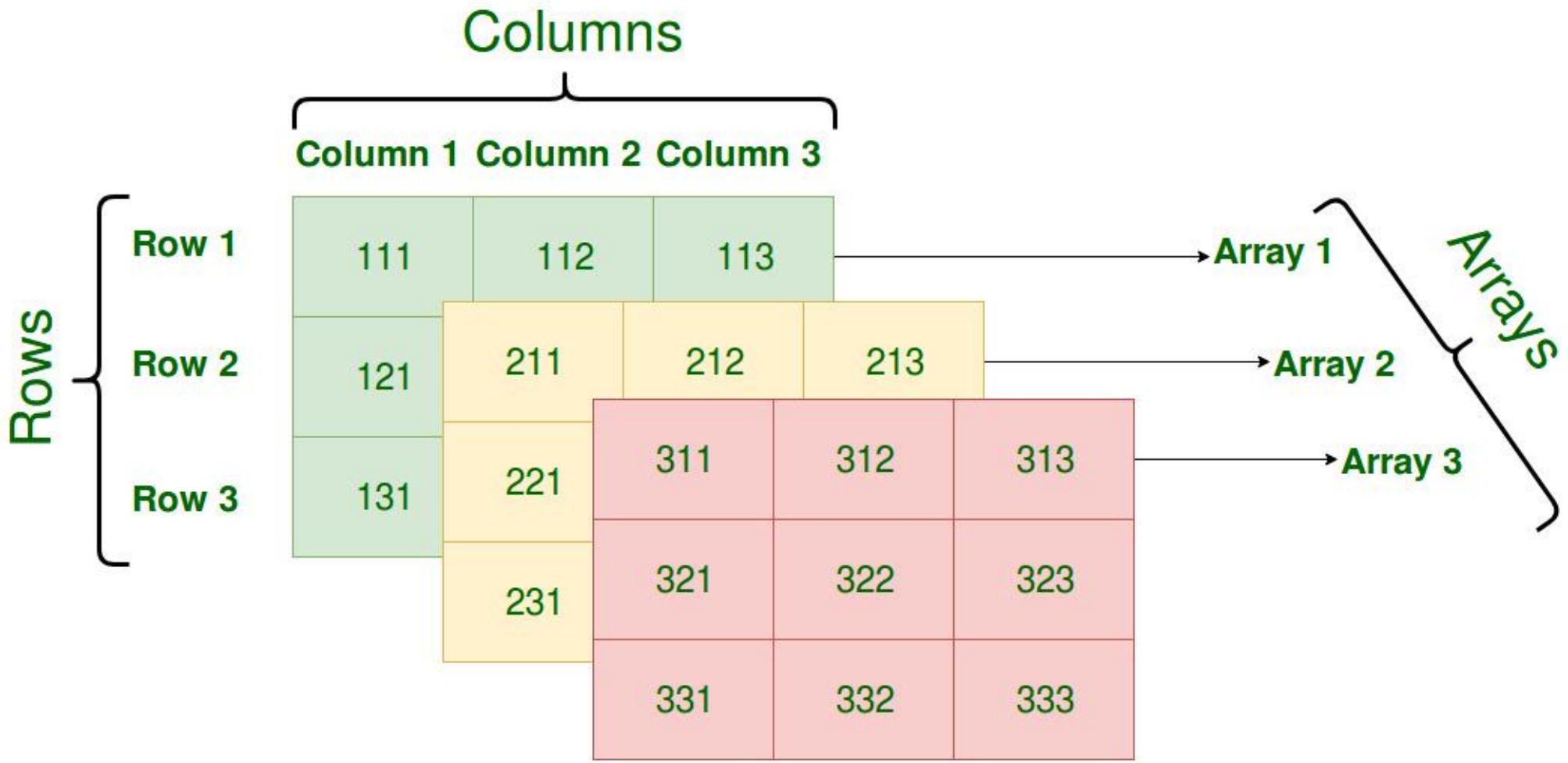
- Basic Graph Theory
- Adjacency Matrix and Depth First Search
- Practice Problem: milk3

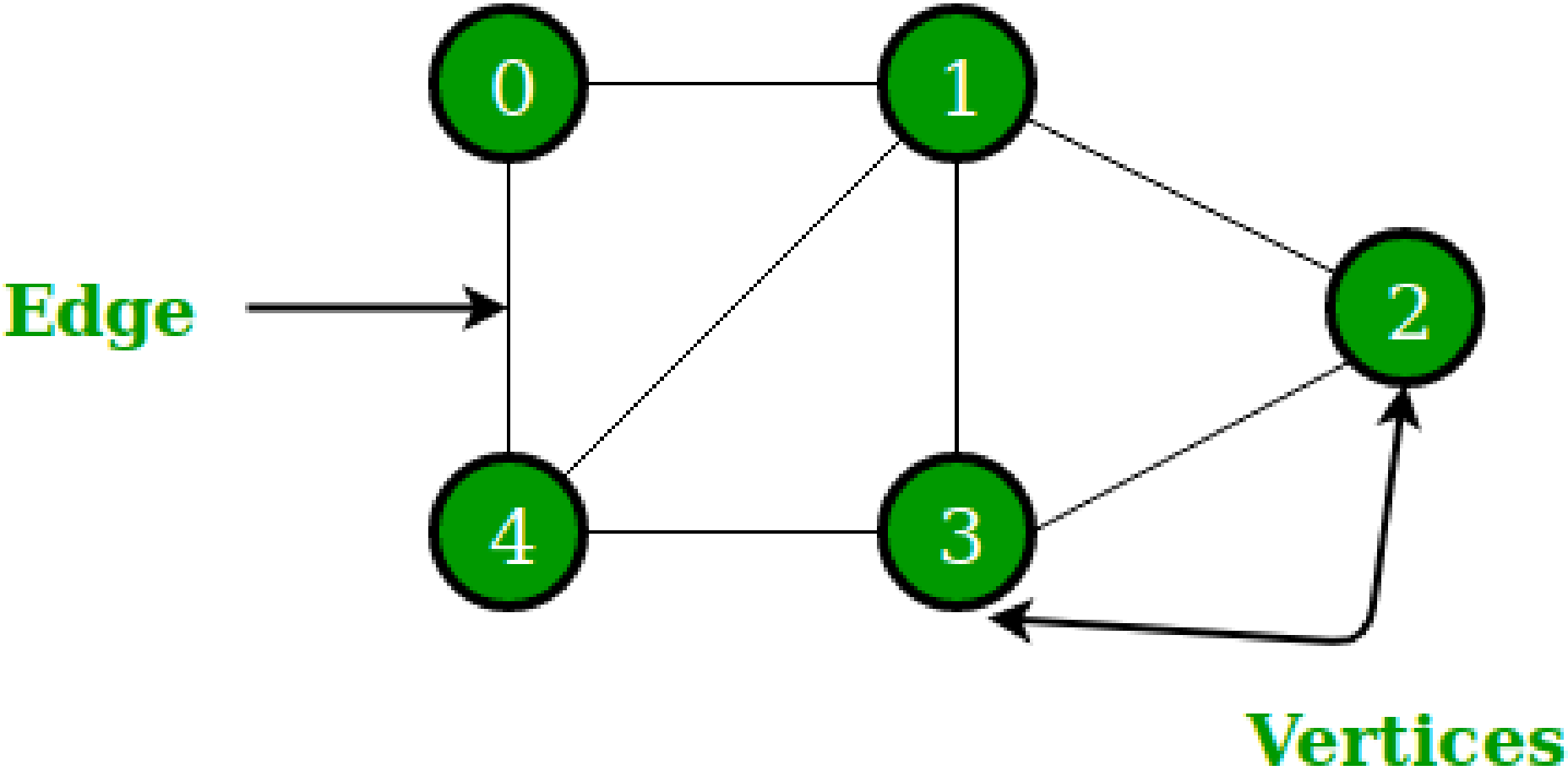
Nonlinear Data Structure

SECTION 1

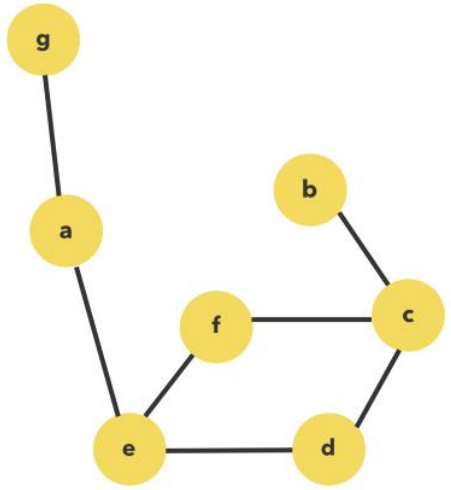

Data Structure
Linear data Structure
Non-linear data Structure
Array
Stack
Queue
Linked-list
Graphs
Trees


Root
Key
P
Edge
Q
R
A
B
C
D
Height of the tree
Parent
L
Siblings
M
E
F
G
Subtree
H
I
Children
Leaf nodes

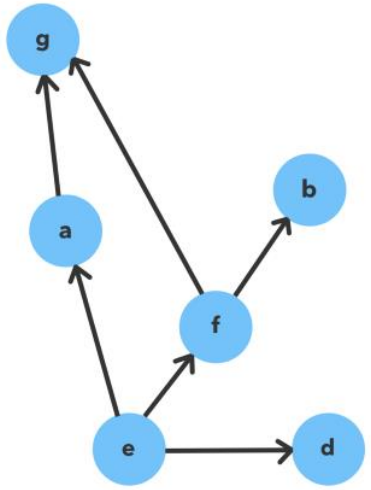

Columns
Column 1
Column 2
Column 3
Rows
Row 1
Row 2
Row 3
111
112
113
121
211
212
213
131
221
231
311
312
313
321
322
323
331
332
333
Array 1
Array 2
Array 3
Arrays


Edge
0
1
2
3
4
Vertices

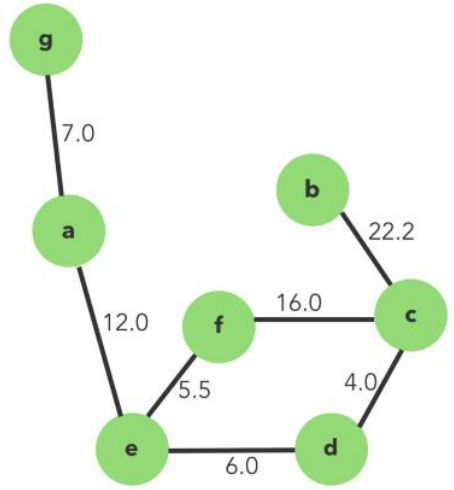
Undirected



Directed

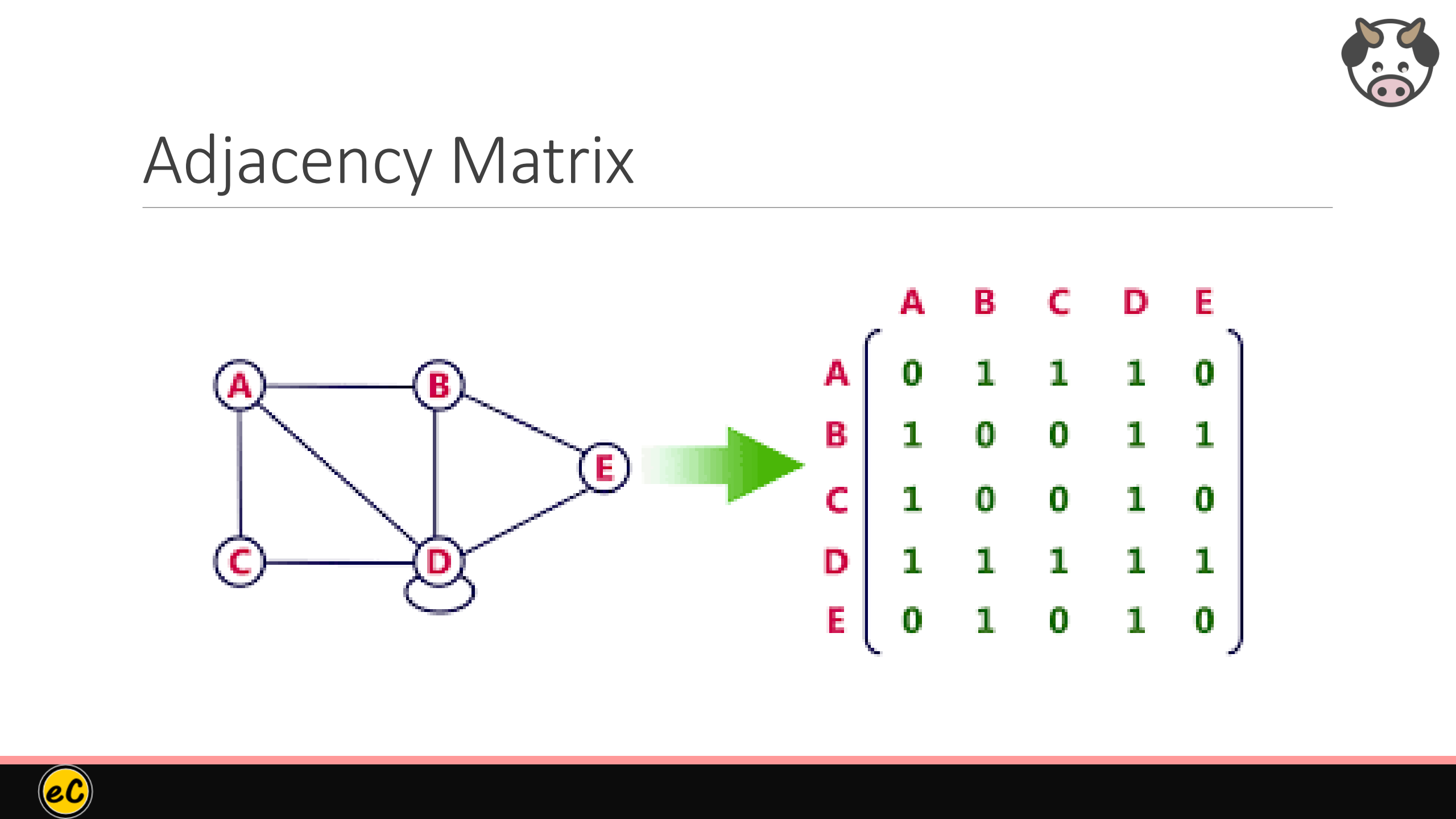


Weighted



Graph

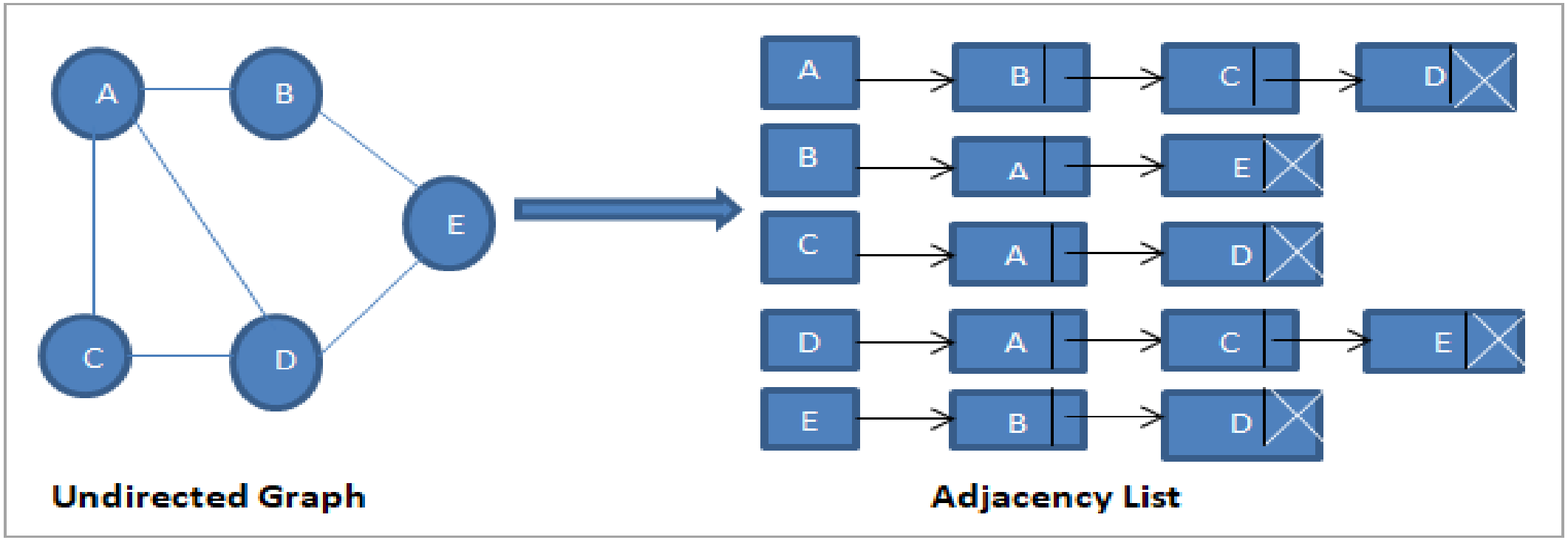
SECTION 2



Adjacency Matrix

	A	B	C	D	E
A	0	1	1	1	0
B	1	0	0	1	1
C	1	0	0	1	0
D	1	1	1	1	1
E	0	1	0	1	0

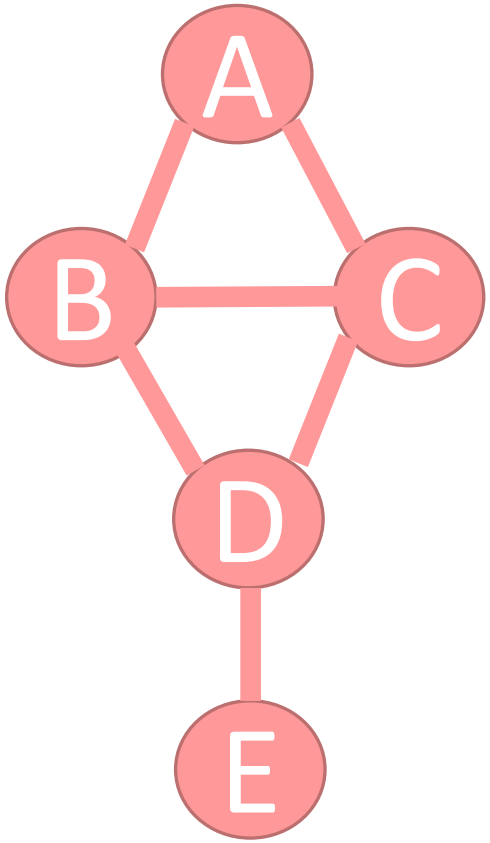
Adjacency List



Adjacency Matrix

SECTION 3

Graph of Study



AdjacencyMatrix

- Adjacency Matrix with visited array

```java
import java.util.*;
public class AdjacencyMatrxix
{                          // 0    1    2    3    4
    static String[] n = {"A", "B", "C", "D", "E"};  // vertex
    static boolean[][] m = {  // adjacency matrix, edges
        {false, true, true, false, false},
        {true, false, true, true, false},
        {true, true, false, true, false},
        {false, true, true, false, true},
        {false, false, false, true, false}
    };

    static boolean[] visited = new boolean[n.length];
```

AdjacencyMatrxix.java

```java
public static boolean dfs(int root, boolean[] v){ // depth first search
  if (v[root]) return false;
  System.out.println(n[root]);
  v[root] = true;
  boolean[] vv = new boolean[n.length];
  boolean all = true;
  for (int j=0; j<n.length; j++){
          all &= v[j];
          vv[j] = v[j];
     }
  if (all) return true;
  //System.out.println(Arrays.toString(vv));
  for (int i=0; i<n.length; i++){
      if (i!=root && !v[i] && m[root][i]){
       if (dfs(i, vv)) return true;
      }
    }
  return false;
}
```

AdjacencyMatrxix.java

```
    public static void main(String[] args){
        System.out.print("\f");
        dfs(0, visited);
    }
}
```

AdjacencyMatrixArrayList

- Adjacency Matrix with visited arraylist

```
import java.util.*;
public class AdjancecyMatrixArrayList
{
    static String[] n = {"A", "B", "C", "D", "E"};  // vertex
    static boolean[][] m = {  // adjacency matrix, edges
        {false, true, true, false, false},
        {true, false, true, true, false},
        {true, true, false, true, false},
        {false, true, true, false, true},
        {false, false, false, true, false}
    };
```

AdjancecyMatrixArrayList.java

```java
public static void dfs(int root){
    ArrayList<Integer> visited = new ArrayList<Integer>();
    dfs(root, visited);
    System.out.println();
}
public static void dfs(int root, ArrayList<Integer> v){
  if (v.contains(root)) return;
  System.out.print(n[root]+" ");
  v.add(root);

  if (v.size()==n.length) return; // all visited

  for (int i=0; i<n.length; i++){
      if (i!=root && !v.contains(i) && m[root][i]){
        dfs(i, v);
        if (v.size()==n.length) return; // all visited
        v.remove(new Integer(i));
      }
    }
}
```

AdjancecyMatrixArrayList.java

```
    public static void main(String[] args){
        System.out.print("\f");
        dfs(0);
        System.out.println();
        dfs(2);
        System.out.println();
        dfs(4);
        System.out.println();
    }
}
```

AdjancecyMatrixArrayList.java

Practice: Mother's Milk (milk3)

SECTION 4

Problem Statement

- Farmer John has three milking buckets of capacity A, B, and C liters. Each of the numbers A, B, and C is an integer from 1 through 20, inclusive. Initially, buckets A and B are empty while bucket C is full of milk. Sometimes, FJ pours milk from one bucket to another until the second bucket is filled or the first bucket is empty. Once begun, a pour must be completed, of course. Being thrifty, no milk may be tossed out.
- Write a program to help FJ determine what amounts of milk he can leave in bucket C when he begins with three buckets as above, pours milk among the buckets for a while, and then notes that bucket A is empty.

INPUT FORMAT (milk3.in):

A single line with the three integers A, B, and C.

SAMPLE INPUT:

```
8 9 10
```

OUTPUT FORMAT (milk3.out):

- A single line with a sorted list of all the possible amounts of milk that can be in bucket C when bucket A is empty.

SAMPLE OUTPUT:

```
1 2 8 9 10
```

Other Input/Output:

SAMPLE INPUT:

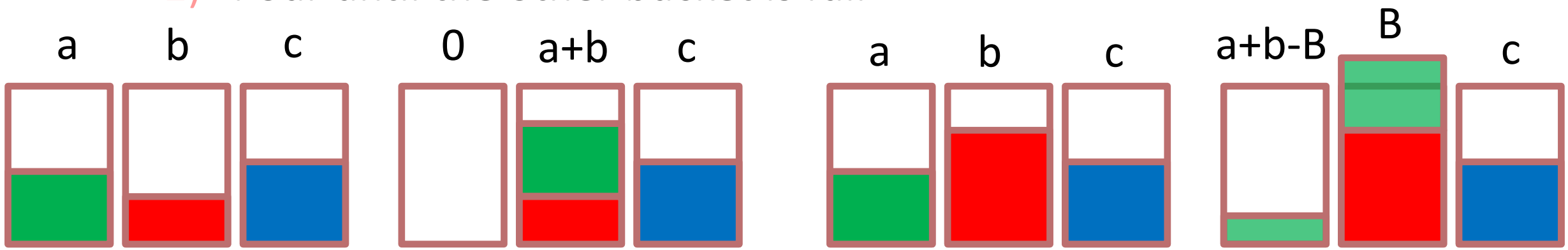
```
2 5 10
```

SAMPLE OUTPUT:

```
5 6 7 8 9 10
```

Nature of Problem

- Recursive Call
- Keep visited list and solutions in the arraylist.
- There are two possible pouring conditions:
  1) Pour all in one bucket to the other
  2) Pour until the other bucket is full



Nature of Problem

```
static void run(a combination of (a, b, c)){
     if (the combination of (a, b, c) has already been tried) return;
     visited_list.add(the combination of (a, b, c));
     if (a==0) completion_list.add(c);
     if (a!=0){
          if (case 1) run(pour a->b case 1);  else run(pour a->b case 2);
          if (case 1) run(pour a->c case 1);  else run(pour a->c case 2);
     }
    if (b!=0){
          if (case 1) run(pour b->a case 1);  else run(pour b->a case 2);
          if (case 1) run(pour b->c case 1);  else run(pour b->c case 2);
     }
    if (c!=0){
          if (case 1) run(pour c->a case 1);  else run(pour c->a case 2);
          if (case 1) run(pour c->b case 1);  else run(pour c->b case 2);
     }
}
```