ESTRUCTURAS DE DATOS

2024

Parcial 2

Grafos-Tries-Arbol B

Profesores:

HECTOR REINAGA

FERNANDA DANIELA OYARZO

MIRTHA FABIANA MIRANDA

Alumno:

GONZALO ALEJANDRO ULLOA

UNPA

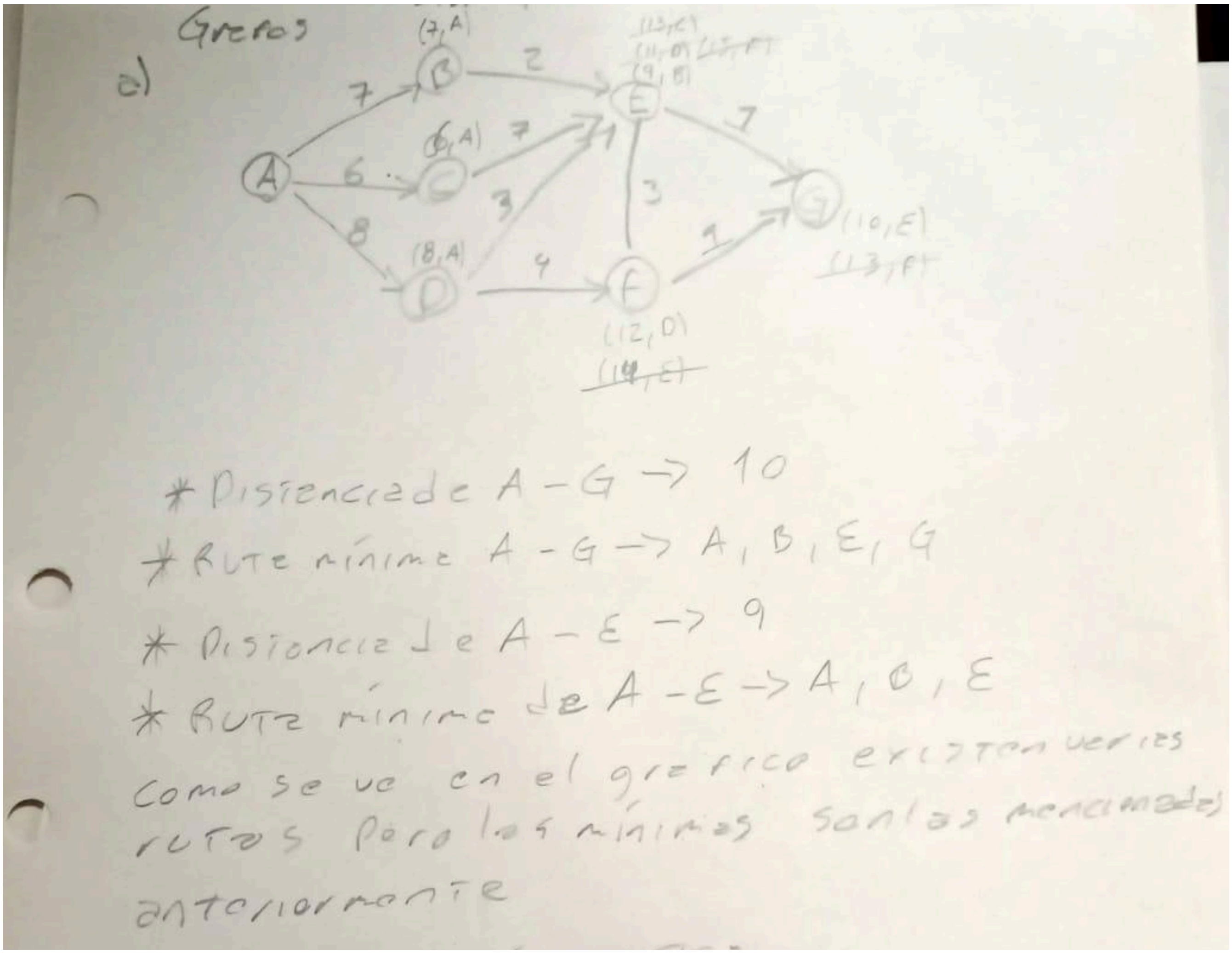
Universidad Nacional de la Patagonia Austral

Índice

Desarrollo

1. Grafos

Punto a



* Distancia de A - G -> 10

* Ruta mínima A - G -> A, B, E, G

* Distancia de A - E -> 9

* Ruta mínima de A - E -> A, B, E

Como se ve en el gráfico existen varias rutas para los mínimos son las mencionadas anteriormente

Punto b

Clase Graph

```
package ParcialGrafosTrieB.Grafos;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Vector;
import java.util.Iterator;

public class Graph {
    private Vertex[] vertices;
    private int vertexPosition;
    private boolean[][] edges;
    private int[][] costs;
    private int vertexQuantity;
    private static final int INFINITO = Integer.MAX_VALUE;

    Graph(int quantity) {
        vertexQuantity = quantity;
        vertices = new Vertex[vertexQuantity];
        vertexPosition = 0;
        edges = new boolean[vertexQuantity][vertexQuantity];
        costs = new int[vertexQuantity][vertexQuantity];

        for (int i = 0; i < vertexQuantity; i++) {
            for (int j = 0; j < vertexQuantity; j++) {
                if (i == j) {
                    costs[i][j] = 0;
                } else {
                    costs[i][j] = INFINITO;
                }
            }
        }
    }

    public void insertVertex(Object element) {
        vertices[vertexPosition] = new Vertex();
        vertices[vertexPosition].setElement(element);
        vertexPosition++;
    }

    public void insertEdge(Object originElement, Object
finishElement, int cost) {
        int originPosition = getVertexOrder(originElement);
        int finishPosition = getVertexOrder(finishElement);
        edges[originPosition][finishPosition] = true;
        costs[originPosition][finishPosition] = cost;
    }

    private int getVertexOrder(Object element) {
        int position = 0, order = -1;
        boolean found = false;
        while (position < vertexQuantity & found == false) {
            if (vertices[position].getElement().equals(element)) {
                found = true;
                order = position;
            }
            position++;
        }
        return order;
    }
    public void depthFirstSearch(Object element){
        Vector visited = new Vector(vertexQuantity);
        depthFirst(getVertexOrder(element), visited);
    }

    private void depthFirst(int element, Vector visited){
        System.out.print(vertices[element].getElement() + " ");
        visited.addElement(new Integer(element));
        Enumeration adjs = adjacents(new Integer(element));
        while (adjs.hasMoreElements()) {
            Integer adjsOther = (Integer) adjs.nextElement();
            if (!visited.contains(adjsOther)) {
                depthFirst(adjsOther.intValue(), visited);
            }
        }
    }

    public void breadhFirstSearch(Object element) {
        breadhFirst(getVertexOrder(element));
    }
    private void breadhFirst(int element) {
        Vector<Integer> visited = new Vector(vertexQuantity);
        Queue<Integer> explore = new LinkedList<>();
        explore.add(new Integer(element));
        visited.addElement(new Integer(element));
        do {
            Integer vertexOther = (Integer) explore.poll();

System.out.print(vertices[vertexOther.intValue()].getElement() + "
");
            Enumeration adjs = adjacents(vertexOther);
            while (adjs.hasMoreElements()) {
                Integer adjsOther = (Integer) adjs.nextElement();
                if (!visited.contains(adjsOther)) {
                    explore.add(adjsOther);
                    visited.addElement(adjsOther);
                }
            }
        } while (!explore.isEmpty());
    }

    public ArrayList<Integer> dijkstraAlgorithm(Object vertex) {
        return dijkstra(getVertexOrder(vertex));
    }

    private ArrayList<Integer> dijkstra(int vertex) {
        int vs;
        ArrayList<Integer> distance = new
ArrayList<>(vertexQuantity);
        ArrayList<Integer> toVisit = new
ArrayList<>(vertexQuantity);

        for (vs = 0; vs < vertexQuantity; vs++) {
            if (vs == vertex) {
                distance.add(0);
            } else {
                distance.add(INFINITO);
            }
            toVisit.add(vs);
        }

        while (!toVisit.isEmpty()) {
            Integer u = minimum(distance, toVisit.iterator());
            toVisit.remove(u);
            int du = distance.get(u);

            if (du != INFINITO) {
                Enumeration<Integer> adjs = adjacents(u);
                while (adjs.hasMoreElements()) {
                    Integer w = adjs.nextElement();
                    if (toVisit.contains(w)) {
                        int cuw = costs[u][w];
                        if (du + cuw < distance.get(w)) {
                            distance.set(w, du + cuw);
                        }
                    }
                }
            }
        }
        return distance;
    }

    private Integer minimum(ArrayList<Integer> distance,
Iterator<Integer> toVisitI) {
        Integer vertexMinimum = toVisitI.next();
        int distanceMinimum = distance.get(vertexMinimum);

        while (toVisitI.hasNext()) {
            Integer vertex = toVisitI.next();
            int distanceValue = distance.get(vertex);
            if (distanceValue < distanceMinimum) {
                vertexMinimum = vertex;
                distanceMinimum = distanceValue;
            }
        }
        return vertexMinimum;
    }

    private Enumeration<Integer> adjacents(Integer element) {
        Vector<Integer> adjVertices = new Vector<>();
        for (int i = 0; i < vertexQuantity; i++) {
            if (edges[element][i]) {
                adjVertices.add(i);
            }
        }
        return adjVertices.elements();
    }

    // floyd con matriz P
    public int[][] floyd() {
        int[][] floydMat = new
int[vertexQuantity][vertexQuantity];
        int[][] P = new int[vertexQuantity][vertexQuantity];
        for (int i = 0; i < vertexQuantity; i++) {
            for (int j = 0; j < vertexQuantity; j++) {
                floydMat[i][j] = costs[i][j];
                if (i != j && costs[i][j] < INFINITO) {
                    P[i][j] = i;
                } else {
                    P[i][j] = -1;
                }
            }
        }
        for (int k = 0; k < vertexQuantity; k++){
            for (int i = 0; i < vertexQuantity; i++){
                for (int j = 0; j < vertexQuantity; j++){
                    if (floydMat[i][k] != INFINITO &&
floydMat[k][j] != INFINITO &&
                        floydMat[i][j] > floydMat[i][k] +
floydMat[k][j]) {

                        floydMat[i][j] = floydMat[i][k] +
floydMat[k][j];
                        P[i][j] = P[k][j];
                    }
                }
            }
        }

        // matriz de costos y predecesores
        System.out.println("Matriz de costos de Floyd:");
        System.out.println("-  A\tB\tC\tD\tE\tF\tG");
        mostrarMatriz(floydMat);
        System.out.println("Matriz de predecesores P:");
        System.out.println("-  A\tB\tC\tD\tE\tF\tG");
        mostrarMatriz(P);

        return floydMat;
    }

    private void mostrarMatriz(int[][] matrix) {
        for (int i = 0; i < matrix.length; i++) {
            System.out.print(vertices[i].getElement() + " ");
            for (int j = 0; j < matrix[i].length; j++) {
                System.out.print((matrix[i][j] == INFINITO ? "INF"
: matrix[i][j]) + "\t");
            }
            System.out.println();
        }
    }
}
```

Clase Edge

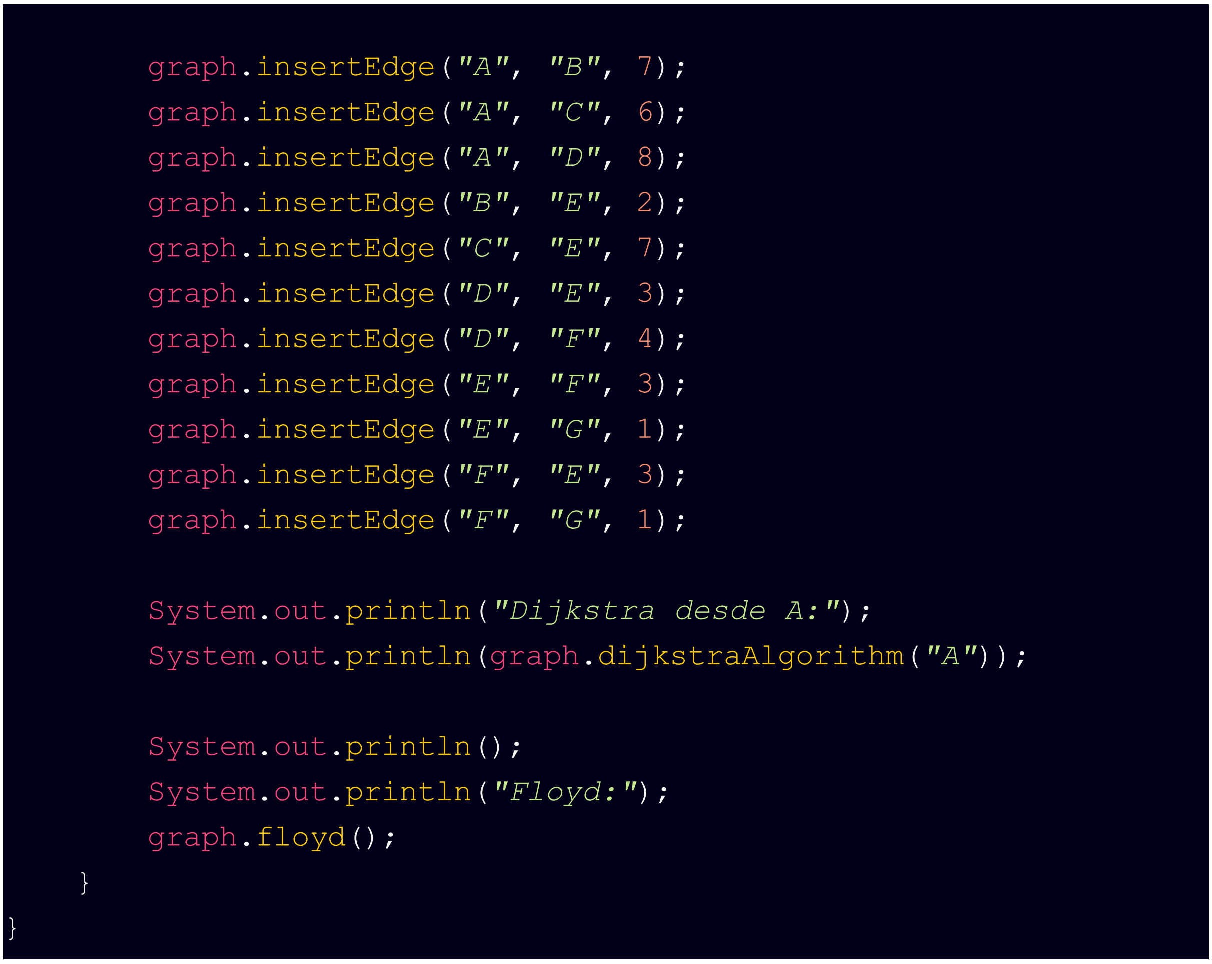
```
package ParcialGrafosTrieB.Grafos;
public class Edge {
    private int position;
    private Edge edge;
    private int coste;
    Edge() {
        position = 0;
        coste=0;
        edge = null;
    }
    Edge(int coste) {
        this.position = 0;
        this.coste = coste;
        this.edge = null;
    }
    public int getPosition() {
        return position;
    }
    public int getCoste() {
        return this.coste;
    }
    public void setCoste(int coste) {
        this.coste = coste;
    }
    public Edge getEdge() {
        return edge;
    }
    public void setPosition(int position) {
        this.position = position;
    }
    public void setEdge(Edge edge) {
        this.edge = edge;
    }
}
```

Clase Vertex

```
package ParcialGrafosTrieB.Grafos;
public class Vertex {
    private Object element;
    private Edge edge;
    Vertex() {
        element = null;
        edge = null;
    }
    public Object getElement() {
        return element;
    }
    public Edge getEdge() {
        return edge;
    }
    public void setElement(Object element) {
        this.element = element;
    }
    public void setEdge(Edge edge) {
        this.edge = edge;
    }
}
```

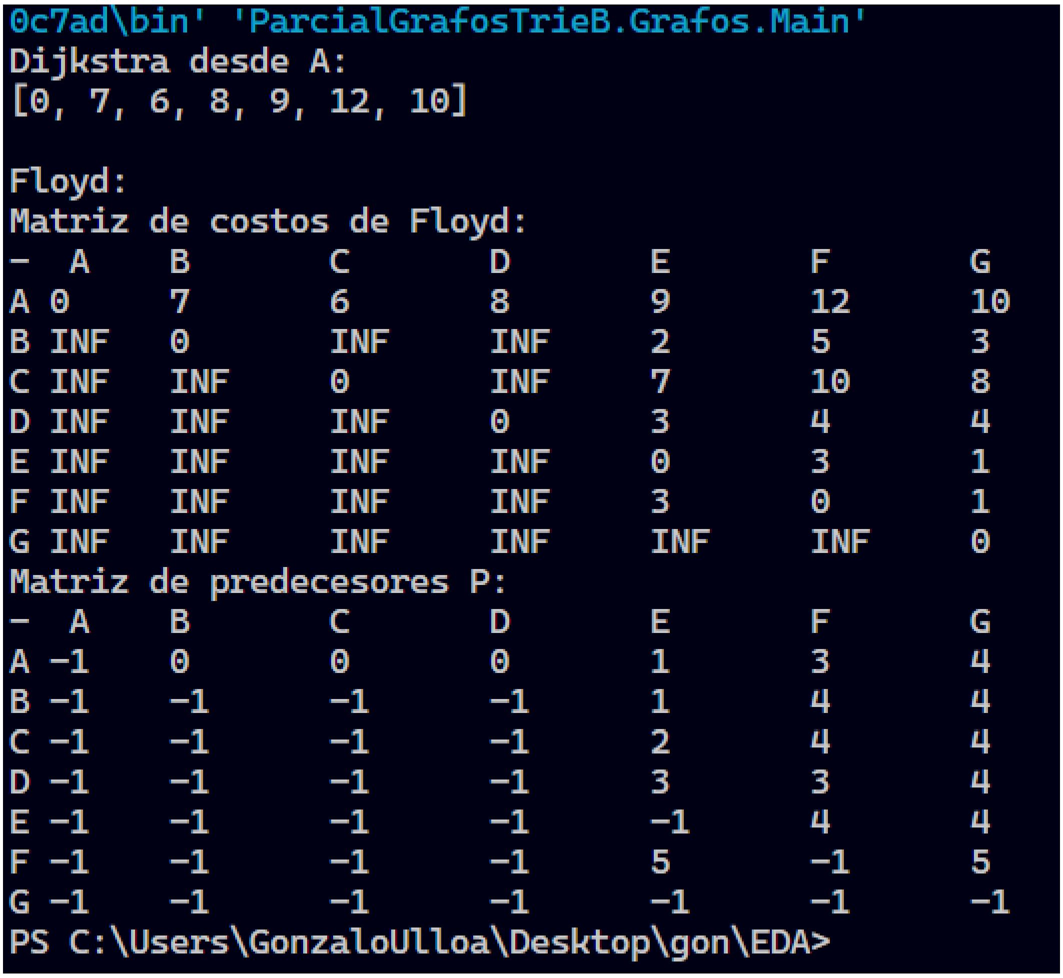
Clase Main

```
package ParcialGrafosTrieB.Grafos;

public class Main {
    public static void main(String[] args) {
        Graph graph = new Graph(7);

        graph.insertVertex("A");
        graph.insertVertex("B");
        graph.insertVertex("C");
        graph.insertVertex("D");
        graph.insertVertex("E");
        graph.insertVertex("F");
        graph.insertVertex("G");
```

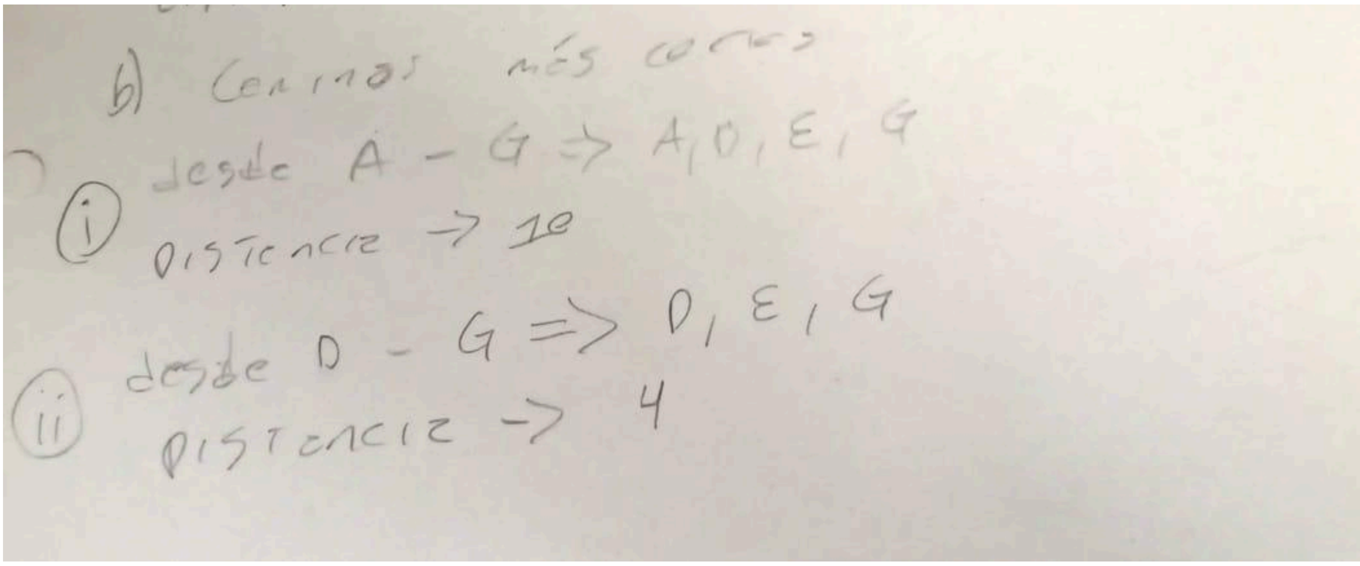


```
        graph.insertEdge("A", "B", 7);
        graph.insertEdge("A", "C", 6);
        graph.insertEdge("A", "D", 8);
        graph.insertEdge("B", "E", 2);
        graph.insertEdge("C", "E", 7);
        graph.insertEdge("D", "E", 3);
        graph.insertEdge("D", "F", 4);
        graph.insertEdge("E", "F", 3);
        graph.insertEdge("E", "G", 1);
        graph.insertEdge("F", "E", 3);
        graph.insertEdge("F", "G", 1);

        System.out.println("Dijkstra desde A:");
        System.out.println(graph.dijkstraAlgorithm("A"));

        System.out.println();
        System.out.println("Floyd:");
        graph.floyd();
    }
}
```

Resultado



```
0c7ad\bin' 'ParcialGrafosTrieB.Grafos.Main'
Dijkstra desde A:
[0, 7, 6, 8, 9, 12, 10]

Floyd:
Matriz de costos de Floyd:
-  A     B       C       D       E       F       G
A 0     7       6       8       9       12      10
B INF   0       INF     INF     2       5       3
C INF   INF     0       INF     7       10      8
D INF   INF     INF     0       3       4       4
E INF   INF     INF     INF     0       3       1
F INF   INF     INF     INF     3       0       1
G INF   INF     INF     INF     INF     INF     0
Matriz de predecesores P:
-  A     B       C       D       E       F       G
A -1    0       0       0       1       3       4
B -1    -1      -1      -1      1       4       4
C -1    -1      -1      -1      2       4       4
D -1    -1      -1      -1      3       3       4
E -1    -1      -1      -1      -1      4       4
F -1    -1      -1      -1      5       -1      5
G -1    -1      -1      -1      -1      -1      -1
PS C:\Users\GonzaloUlloa\Desktop\gon\EDA>
```



b) Caminos más cortos

(i) desde A - G => A, D, E, G

Distancia -> 10

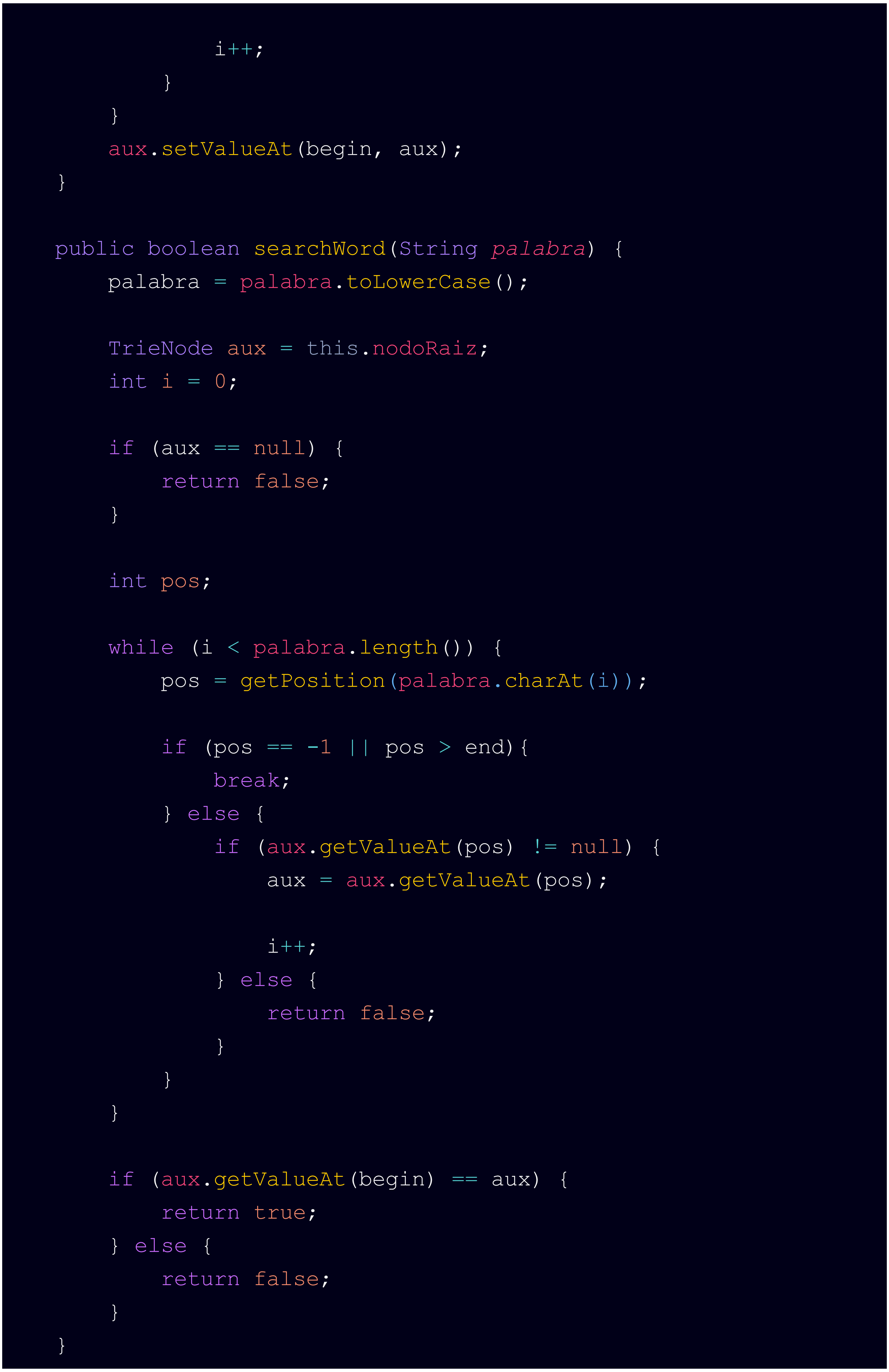
(ii) desde D - G => D, E, G

Distancia -> 4

2. Tries

Clase Trie

```java
package ParcialGrafosTrieB;

//Matriz
class Trie {
    private TrieNode nodoRaiz;
    private int cantidad;
    private int begin = 0;
    private int end = 26;
    private int nodos=0;
    public Trie(){
        this.nodoRaiz = null;
    }
    public void insertWord(String palabra) {
        palabra = palabra.toLowerCase();

        TrieNode aux;
        int i = 0;
        int pos;

        if (nodoRaiz == null){
            nodoRaiz = new TrieNode(end);
        }

        aux = nodoRaiz;

        while (i < palabra.length()){

            pos = getPosition(palabra.charAt(i));

            if (pos == -1 || pos > end) {
                break;
            } else {
                if (aux.getValueAt(pos) == null){
                    aux.setValueAt(pos, new TrieNode(end));
                    nodos++;
                }

                aux = aux.getValueAt(pos);
```



```
                i++;
            }
        }
        aux.setValueAt(begin, aux);
    }

    public boolean searchWord(String palabra) {
        palabra = palabra.toLowerCase();

        TrieNode aux = this.nodoRaiz;
        int i = 0;

        if (aux == null) {
            return false;
        }

        int pos;

        while (i < palabra.length()) {
            pos = getPosition(palabra.charAt(i));

            if (pos == -1 || pos > end){
                break;
            } else {
                if (aux.getValueAt(pos) != null) {
                    aux = aux.getValueAt(pos);

                    i++;
                } else {
                    return false;
                }
            }
        }

        if (aux.getValueAt(begin) == aux) {
            return true;
        } else {
            return false;
        }
    }


    public void mostrarPalabras(){
        this.mostrarPalabras(this.nodoRaiz, "");
    }

    private void mostrarPalabras(TrieNode nodo, String imp){
        for (int i = 0; i <= end; i++) {
            if (nodo.getValueAt(i)!=null){
                if (i==0) {
                    System.out.println(imp);
                } else {
                    this.mostrarPalabras(nodo.getValueAt(i),
imp+(recuperarCaracter(i)));
                }
            }
        }
    }

    public int contarPalabras(){
        this.cantidad = 0;
        contarPalabras(nodoRaiz);

        return this.cantidad;
    }

    private void contarPalabras(TrieNode nodo){
        for (int i = 0; i <= end; i++){

            if (nodo.getValueAt(i) != null){

                if (i == 0){
                    cantidad++;
                } else {
                    contarPalabras(nodo.getValueAt(i));
                }
            }
        }
    }
    public int contarPrefijos() {
        TrieNode aux=nodoRaiz;
        if (aux == null) {
            return 0;
        }
        return contarPrefijos(aux) - 1; // - 1 al descontar la 
raiz
    }

    private int contarPrefijos(TrieNode nodo) {
        if (nodo == null) {
            return 0;
        }
        int prefijos = 0;
        int hijos = 0;
        for (int i = 1; i <= end; i++){
            if (nodo.getValueAt(i) != null) {
                hijos++;
                prefijos += contarPrefijos(nodo.getValueAt(i));
            }
        }
        if (hijos > 1) {
            prefijos++;
        }
        return prefijos;
    }

    public void buscarPrefijos(String pref){
        this.buscarPrefijos(nodoRaiz, "", pref);
    }

    private void buscarPrefijos(TrieNode nodo, String imp , String 
pref){
        boolean ok = false;

        for (int i = 0; i <= end; i++) {
            if (nodo.getValueAt(i) != null) {
                if (i == 0) {
                    if (!ok && (imp.length() > pref.length()) && 
(imp.substring(0, pref.length()).equals(pref))) {
                        ok=true;
                        System.out.println(imp);
                    }
                } else {
                    this.buscarPrefijos(nodo.getValueAt(i),
imp+(recuperarCaracter(i)),pref);
                }
            }
        }
    }

    public static int getPosition(char c){
        switch(c) {
            case '@': return 0;
            case 'a': return 1;
            case 'b': return 2;
            case 'c': return 3;
            case 'd': return 4;
            case 'e': return 5;
            case 'f': return 6;
            case 'g': return 7;
            case 'h': return 8;
            case 'i': return 9;
            case 'j': return 10;
            case 'k': return 11;
            case 'l': return 12;
            case 'm': return 13;
            case 'n': return 14;
            case 'o': return 15;
            case 'p': return 16;
            case 'q': return 17;
            case 'r': return 18;
            case 's': return 19;
            case 't': return 20;
            case 'u': return 21;
            case 'v': return 22;
            case 'w': return 23;
            case 'x': return 24;
            case 'y': return 25;
            case 'z': return 26;
        }

        return -1;
    }

    public static char recuperarCaracter(int val){
        char c='a';

        for (int i = 1; i < val; i++) {
            c++;
        }

        return c;
    }

    public int contarNodos() {
        return nodos;
    }


    public void contarAccesos(String palabra){
        int num=contarAccesosP(palabra);
        if (num==-1){
            System.out.println("La palabra "+palabra+" no se 
encuentra en el Trie");
        }else{
            System.out.println("La cantidad de accesos para "+
palabra+" es de: "+num);
        }
    }

    private int contarAccesosP(String palabra) {
        palabra = palabra.toLowerCase();
        TrieNode aux = nodoRaiz;
        int accesos = 0;

        if (aux == null) {
            return -1;
        }

        for (int i = 0; i < palabra.length(); i++) {
            int pos = getPosition(palabra.charAt(i));

            if (pos == -1 || pos > end) {
                return -1;
            }

            if (aux.getValueAt(pos) == null) {
                return -1; // la palabra no existe en el Trie
            }

            aux = aux.getValueAt(pos);
            accesos++;
        }

        return accesos;
    }

}
```

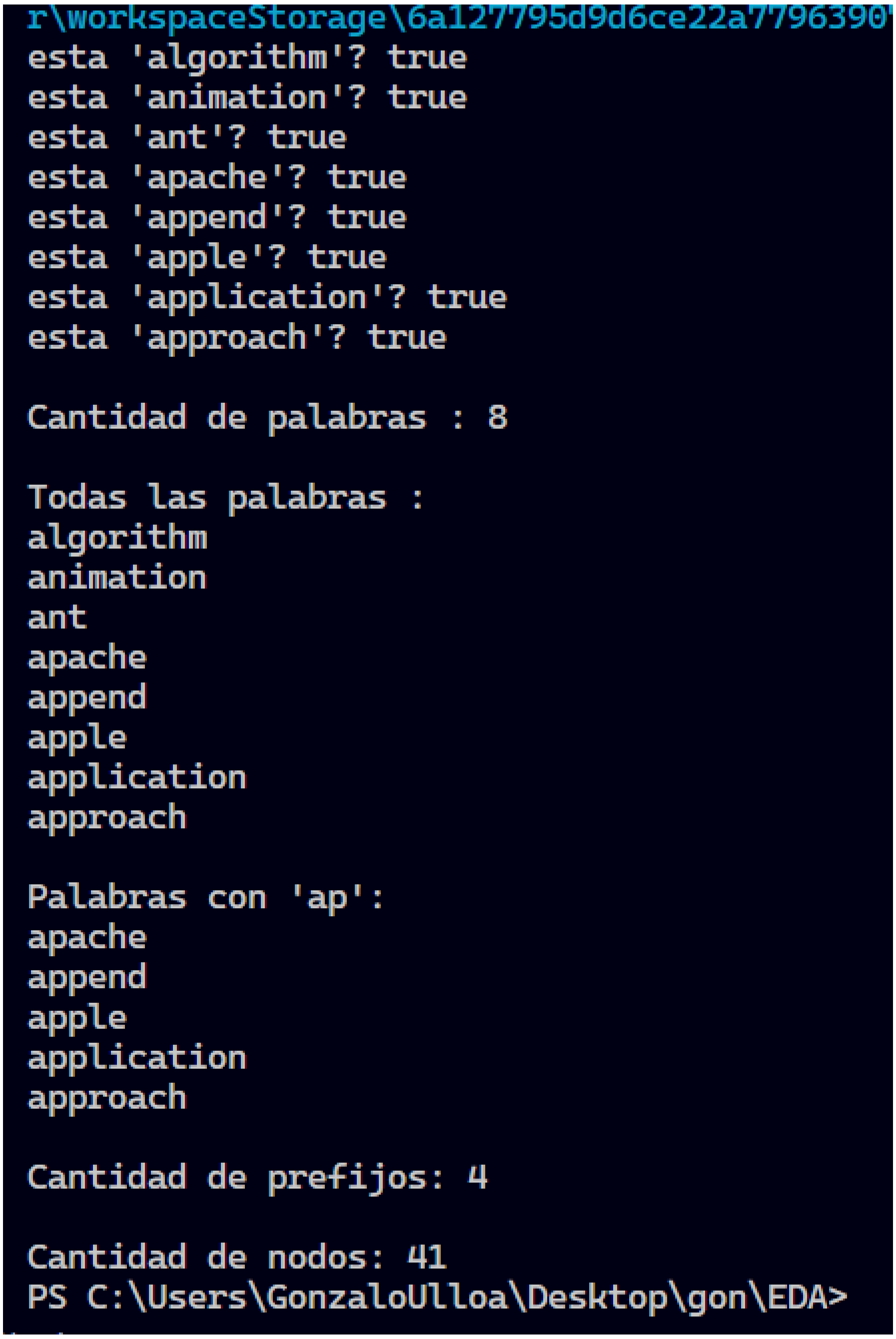
Clase TrieNode

```
package ParcialGrafosTrieB;

class TrieNode {
    TrieNode[] chars;
    TrieNode(int end) {
        chars = new TrieNode[end + 1];
    }
    TrieNode getValueAt(int pos) {
        if(pos>=chars.length || pos<0){
            return null;
        }
        return chars[pos];
    }
    void setValueAt(int pos, TrieNode newNode) {
        if(pos<chars.length && pos>=0){
            chars[pos] = newNode;
        } else{
            System.err.println("error: el valor no pudo ser
establecido");
        }
    }
}
```

Clase Main

```java
package ParcialGrafosTrieB;
public class Main {
    public static void main(String[] args) {
        String[] words = {"algorithm","animation","ant","apache","append","apple","application","approach"};
        //--------MATRIZ--------
        Trie trie = new Trie();
        // insertar palabras
        for (String word : words) {
            trie.insertWord(word);
        }
        // buscar palabras
        for (String word : words) {
            System.out.println("esta '" + word + "'? " + trie.searchWord(word));
        }
        // cantidad de palabras
        System.out.println("\nCantidad de palabras : " + trie.contarPalabras());
        // mostrar todas las palabras
        System.out.println("\nTodas las palabras : ");
        trie.mostrarPalabras();
        // mostrar palabras con un prefijo
        String pref = "ap";
        System.out.println("\nPalabras con '" + pref + "':");
        trie.buscarPrefijos(pref);
        // cantidad de prefijos
        System.out.println("\nCantidad de prefijos: "+ trie.contarPrefijos());
        //Cantidad de nodos
        System.out.println("\nCantidad de nodos: " + trie.contarNodos());
    }
}
```

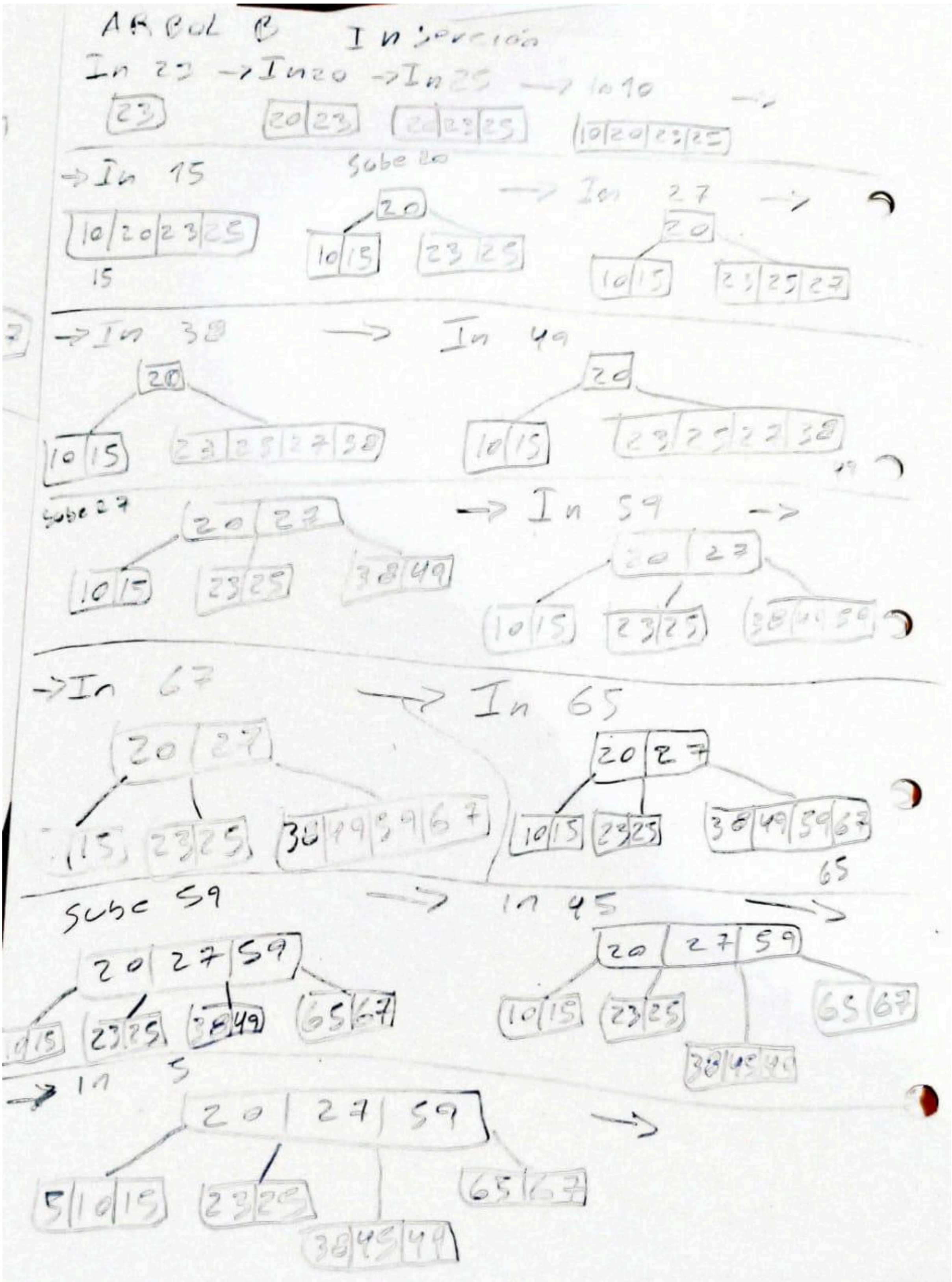
Resultado de consola

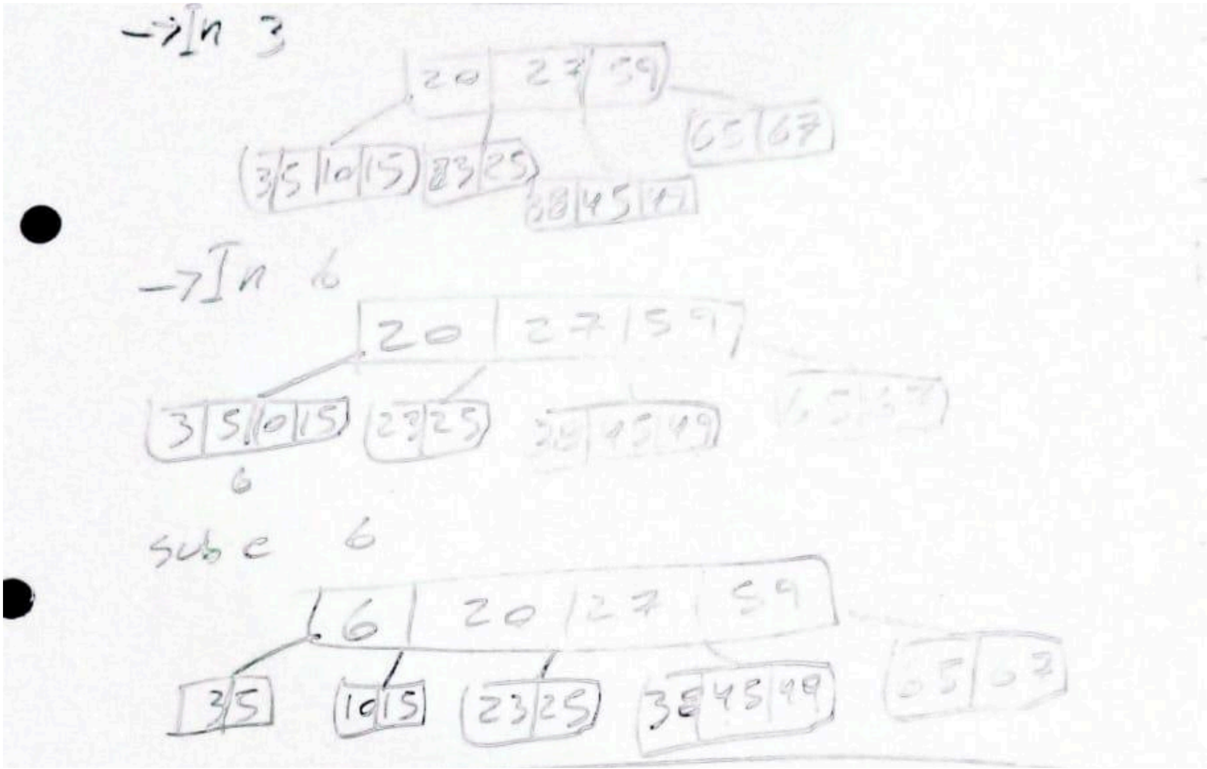


```
r\workspaceStorage\6a127795d9d6ce22a7796390
esta 'algorithm'? true
esta 'animation'? true
esta 'ant'? true
esta 'apache'? true
esta 'append'? true
esta 'apple'? true
esta 'application'? true
esta 'approach'? true

Cantidad de palabras : 8

Todas las palabras :
algorithm
animation
ant
apache
append
apple
application
approach

Palabras con 'ap':
apache
append
apple
application
approach

Cantidad de prefijos: 4

Cantidad de nodos: 41
PS C:\Users\GonzaloUlloa\Desktop\gon\EDA>
```

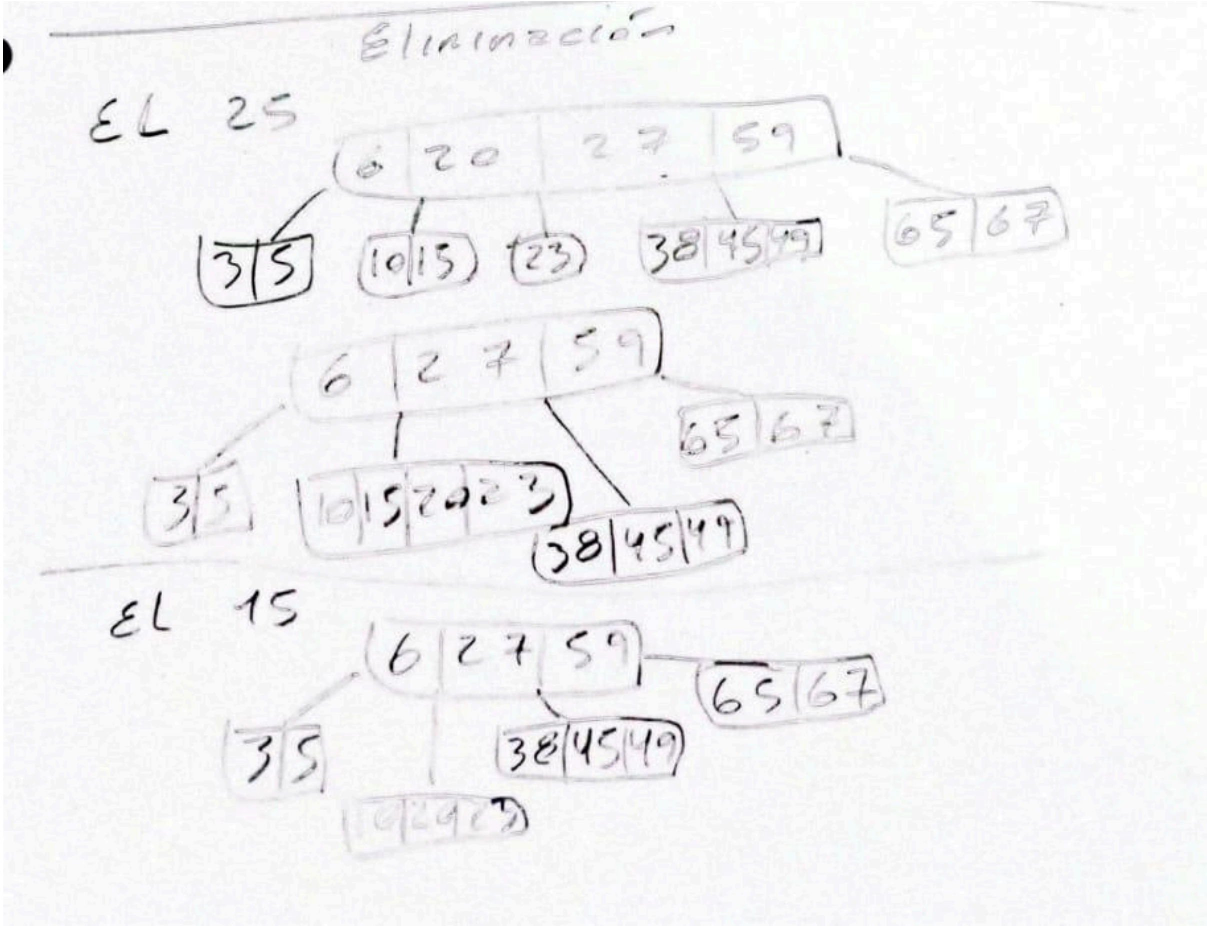
3. Árbol B

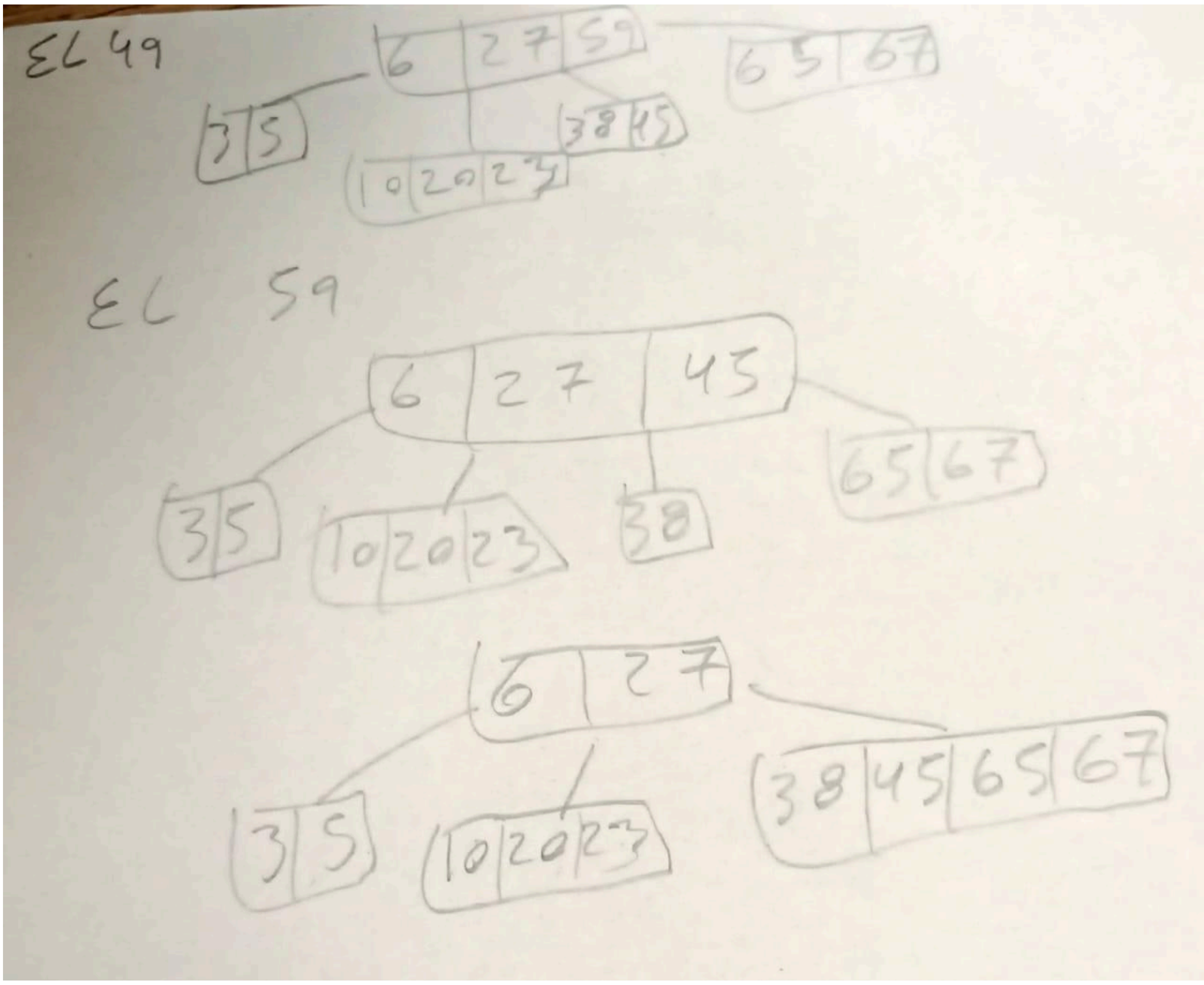
Inserción





Eliminación




EL 49
6 | 27 | 59
65 | 67
3 | 5
38 | 45
10 | 20 | 23
EL 59
6 | 27 | 45
3 | 5
10 | 20 | 23
38
65 | 67
6 | 27
3 | 5
10 | 20 | 23
38 | 45 | 65 | 67