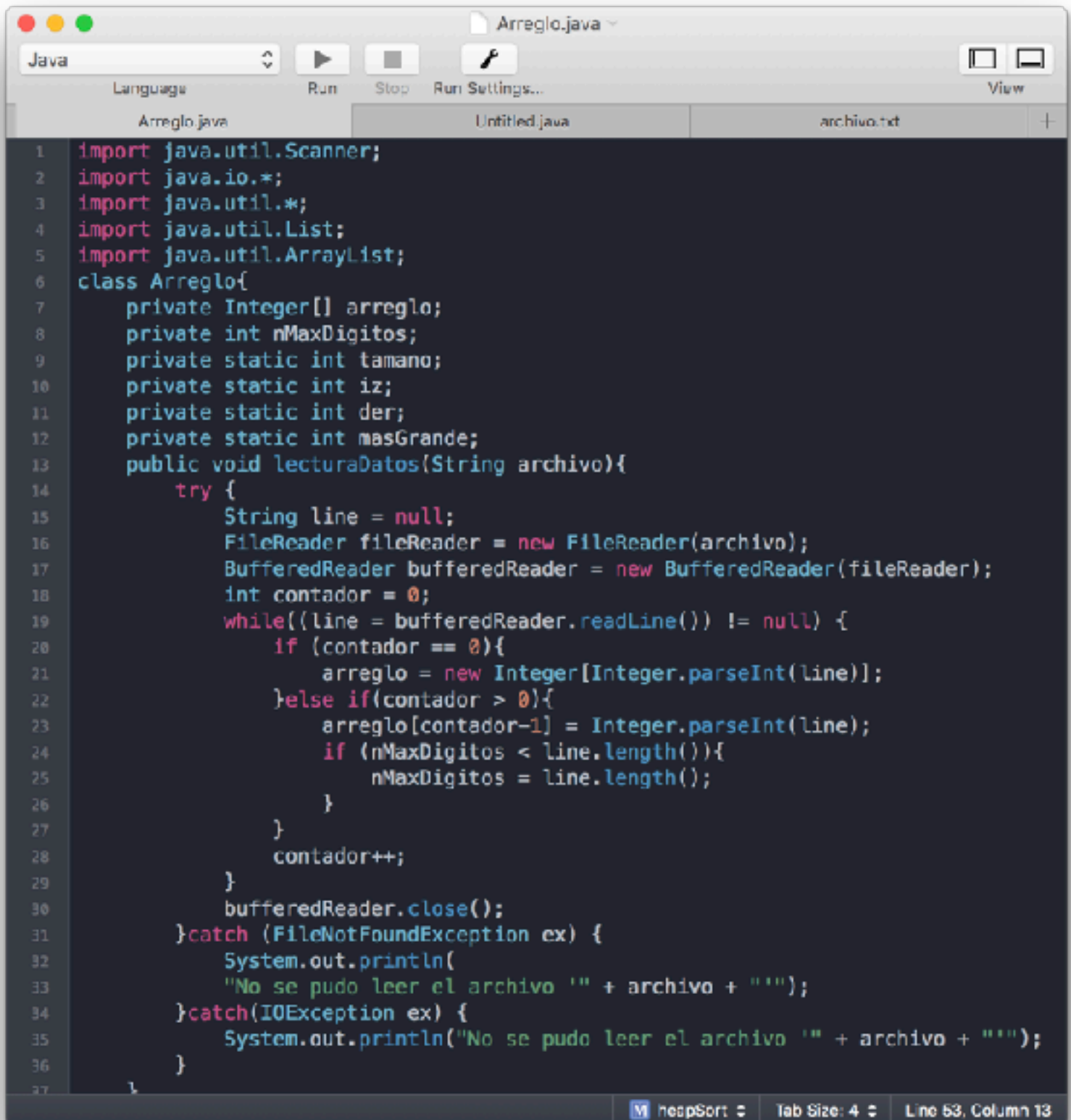


Documentación Proyecto 2

Emiliano Abascal Gurría

A01023234

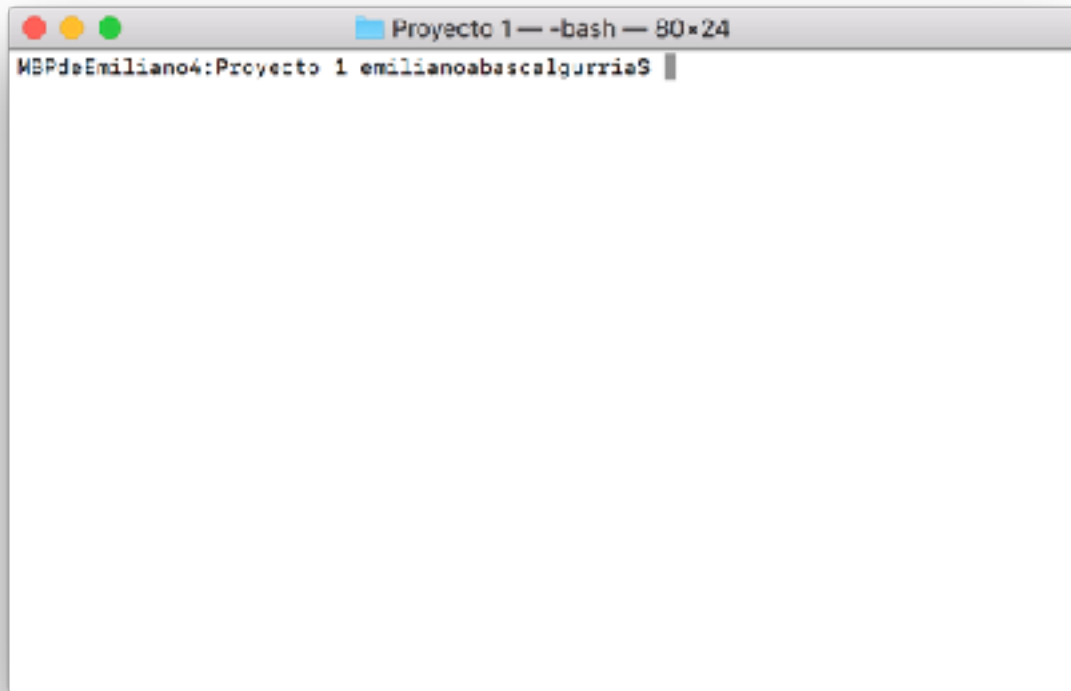
Instituto Tecnológico de Estudios Superiores de Monterrey
Campus Santa Fe



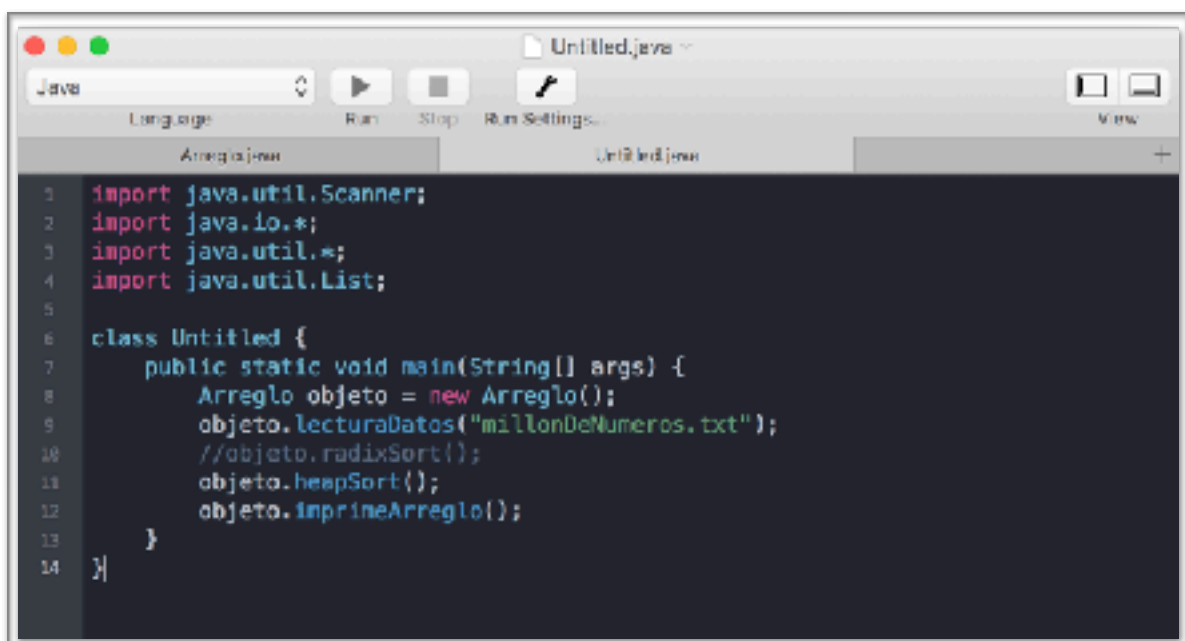
```
1 import java.util.Scanner;
2 import java.io.*;
3 import java.util.*;
4 import java.util.List;
5 import java.util.ArrayList;
6 class Arreglo{
7     private Integer[] arreglo;
8     private int nMaxDigitos;
9     private static int tamano;
10    private static int iz;
11    private static int der;
12    private static int masGrande;
13    public void lecturaDatos(String archivo){
14        try {
15            String line = null;
16            FileReader fileReader = new FileReader(archivo);
17            BufferedReader bufferedReader = new BufferedReader(fileReader);
18            int contador = 0;
19            while((line = bufferedReader.readLine()) != null) {
20                if (contador == 0){
21                    arreglo = new Integer[Integer.parseInt(line)];
22                }else if(contador > 0){
23                    arreglo[contador-1] = Integer.parseInt(line);
24                    if (nMaxDigitos < line.length()){
25                        nMaxDigitos = line.length();
26                    }
27                }
28                contador++;
29            }
30            bufferedReader.close();
31        }catch (FileNotFoundException ex) {
32            System.out.println(
33                "No se pudo leer el archivo '" + archivo + "'");
34        }catch(IOException ex) {
35            System.out.println("No se pudo leer el archivo '" + archivo + "'");
36        }
37    }
}
```

Manual de usuario:

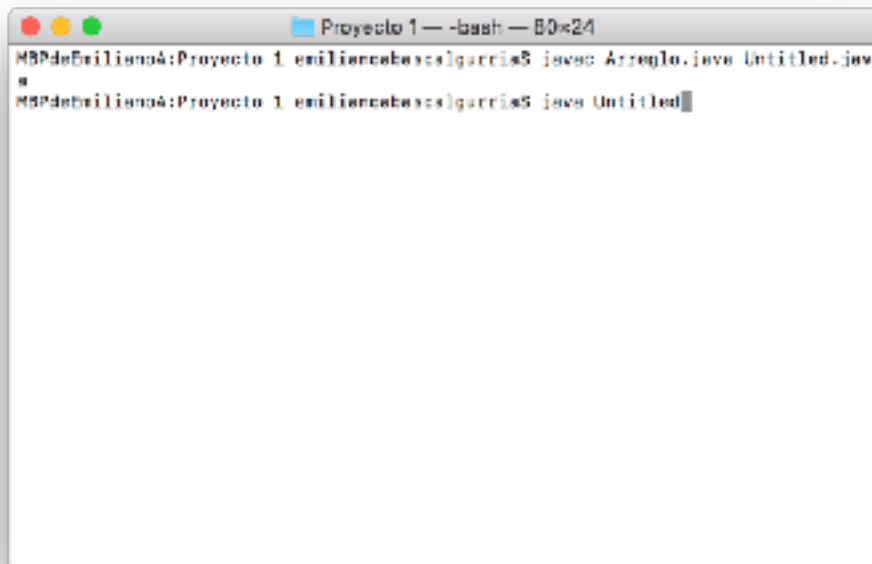
- Una vez obtenido el software ubicarlo en un directorio de preferencia, por ejemplo en el escritorio.
- Abrir la terminal o un compilador para correr programas hechos en Java.
- Escribir el siguiente comando: `cd "El directorio donde se encuentra el programa."`



- Ya que se está en el directorio se requiere hacer una clase de Java en el cual se hace una instancia de Arreglo, como se muestra a continuación:



- Al tener un archivo java se regresa a la terminal y escribe el siguiente comando: `javac Grafo.java` da enter.
- Escribe en la terminal `./a.out` y el programa se correra, como se demuestra en al ejemplo a continuación.



```

Proyecto 1 — -bash — 80x24
MSPdeEmiliano4:Proyecto 1 emiliannobessalgarria$ javac Arreglo.java Untitled.jav
"
MSPdeEmiliano4:Proyecto 1 emiliannobessalgarria$ java Untitled

```

Algoritmo

1. Inicio.
2. Nueva clase "Grafo":
 1. Nueva clase estática "Edge":
 1. Metodos:
 1. entero compareTo(Edge compareEdge)
 2. Atributos:
 1. entero src, dest
 2. boolean visited
 3. Flotante weight
 2. Nueva clase estática "subConjunto":
 1. Atributos:
 1. entero padre, rango
3. Metodos:
 1. Grafo(entero v, entero e)
 2. Flotante kruskalDFS(String archivo)
 3. Flotante kruskalUF(String archivo)
 4. Flotante KruskalMST()
 5. find(subConjunto subConjuntoTemporal[], entero i)
 6. union(subConjunto subConjuntoTemporal[], entero x, entero y)
 7. flotante PrimMST(String archivo)
 8. entero computeMST()

9. void main()
4. Atributos:
 1. entero E,V,count
 2. Edge edge[]
 3. ArrayList<ArrayList<entero[]>> vertices
 4. boolean[] explorado
 5. Heap noExplorado
 6. flotante total
3. Funcionamiento:
 1. Grafo(entero v, entero e)
 1. Constructor de la clase Grafo, en la cual se le asigna el tamaño de los vertices y de los edges, asi como la enicializacion de la variable edge con un tamaño de el numero de Edges.
 2. class Edge():
 1. Declaracion de clase Edge para representar la union entre los nodos, entre sus atributos estan el nodo de origen, el nodo de destino, y el peso.
 3. class subConjunto:
 1. Clase para crear un nuevo sub-grafo y asignarle un rango y su nodo padre
 4. Flotante kruskalDFS(String archivo)
 1. Profesor, ya había acabado mi implementacion de kruskal con DFS, pero no funcionaba con todos los casos, lo volvi a empezar pero el tiempo se me acabo, le pido una disculpa pero aun asi, en el código esta hasta donde me quede, si quiere revisarlo se encuentra este mismo aviso en para ma de comentario, de nuevo una disculpa.
 5. Flotante kruskalUF(String archivo)
 1. Se lee el archivo y se asignan el numero de vertices y de aristas que hay en el grafo, se crea un nuevo grafo con estos tamaños y se asigna cada nodo a cada edge del grafo. Se cierra el archivo y se llama a la funcion KruskalMST.
 6. Flotante KruskalMST()
 1. Funcion para hacer el MST del grafo utilizando kruskal con union find, se guarda el tiempo en el que se empieza a correr la funcion y se asignan valores a los iteradores "e" e "i", Se le asigna un edge a la variable resultado por cada edge que haya en el grafo. Se ordena el arreglo edge en relacion a los pesos, de menor a mayor y se enstancia un subconjunto que va a tener el tamaño de la cantidad de los edges del grafo.
 2. Para cada vertice se le asigna un rango y un padre a cada subconjunto.
 3. Mientras no se hayan recorrido todos los vertices entonces se crea un edge siguiente, despues se llama a la funcion find dos veces con el subconjunto que creamos antes y el origen del siguiente nodo para la primera llamada y el destino para la segunda, estos valores se guardan en variables "x" y "y".
 4. Si x es diferente que y entonces el resultado en la posicion e + 1 sera el siguiente edge, y se llama a la funcion union con el subconjunto, "x" y "y".
 5. Se imprime cada uno de las conexiones resultantes asi como su peso total y el tiempo de ejecucion.
 7. entero find(subConjunto subConjuntoTemporal[], entero i):
 1. Funcion find la cual se llama desde kruskalMST
 2. Si el padre que se encuentra en el subconjunto es diferente que el valor de i entonces se llama recursivamente la funcion asi misma pero con el padre en el posicion de la i.
 3. Sino se regresa el padre que se encuentra en el subconjunto.
 8. void union(subConjunto subConjuntoTemporal[], entero x, entero y):
 1. Se llama a la funcion find con el subconjunto y el valor de x y despues con el valor de y, y se guardan sus valores.
 2. Si los valores anteriores son iguales, entonces no se hace nada, si el rango en el subconjunto en la posicion de xr es menor que el rango en el subconjunto en la

posicion de yr, entonces el padre en la posicion xr sera yr. Sino si el rango en el subconjunto en la posicion de xr es mayor que el rango en el subconjunto en la posicion de yr entonces el padre en la posicion yr sera xr. Sino el padre en el subconjunto en la posicion yr sera xr y el rango en el subconjunto en la posicion xr sera el rango + 1.

9. flotante PrimMST(String archivo):

1. Se lee el archivo y se asignan el numero de vertices y de aristas que hay en el grafo y se asignan los valores para poder llevar a cabo el algoritmo.
2. Se cierra el archivo y se llama la funcion computeMST, la cual su resultado se guardara.
3. Se imprime el tiempo de ejecucion, las conexiones resultantes y el costo total.
4. Se regresa el costo total.

10. entero computeMST():

1. Se inicializa la variable explorado con el numero de vertices que tiene el grafo y el primer valor se asigna como explorado. Se hace un nuevo heap para los no explorados.
2. Para cada i hasta que i sea menor que el numero de vertices, el costo minimo sera de 1000000000
3. Para cada edge que hay en los vertices en la posicion i, si el primer edge es igual a 1 y el edge en la posicion siguiente es menor que el costo minimo, entonces el costo minimo es el edge en la posicion 1.
4. Se agrega a no explorado i + 1 y el valor del costo minimo.
5. Para cada i que empieza en 1, hasta el numero de vertices en el grafo, entonces se explora cada edge que esta en los vertices, si el costo es menor que el no explorado con el costo minimo en el vertice inicial, entonces el no explorado sera el vertice inicial con el costo que se comparo.
6. Se imprime cada conexion, se marca el nodo actual como explorado, y se acumulan los valores del peso.
7. Se regresa el valor de la suma total del costo.

Descripción Técnica

Clase Grafo:

entero E,V,count

Declaracion del numero de vertices y de edges, asimismo como un contador, un arreglo de edges, una lista de listas de vertices y un heap.

Edge edge[]

ArrayList<ArrayList<entero[]>> vertices

boolean[] explorado

Heap noExplorado

flotante total

Grafo(entero v, entero e)

Constructor de la clase Grafo, en la cual se le asigna el tamaño de los vertices y de los edges, asi como la inicializacion de la variable edge con un tamaño de el numero de Edges.

V = v

E = e

```

edge = nuevo Edge[E]
para (entero i=0 i<e ++i)
    edge[i] = nuevo Edge()

```

Flotante kruskalUF(String archivo)throws FileNotFoundException:
 Scanner en = nuevo Scanner(nuevo File(archivo))

Se lee el archivo y se asignan el numero de vertices y de aristas que hay en el grafo, se crea un nuevo grafo con estos tamaños y se asigna cada nodo a cada edge del grafo.

```

V = en.nextEntero()
E = en.nextEntero()
Grafo g = nuevo Grafo(V, E)
para (entero i = 0 i < E i++)
    g.edge[i].src = en.nextEntero()
    g.edge[i].dest = en.nextEntero()
    g.edge[i].weight = en.nextFlotante()

in.close()

```

Se cierra el archivo y se llama a la funcion KruskalMST

```

regresa g.KruskalMST()

```

Flotante KruskalMST():

Funcion para hacer el MST del grafo utilizando kruskal con union find.
 long startTime = System.currentTimeMillis()

se guarda el tiempo en el que se empieza a correr la funcion y se asignan valores a los iteradores "e" e "i".

```

Edge result[] = nuevo Edge[E]
entero e = 0
entero i = 0
para (i = 0 i < E ++i)

```

Se le asigna un edge a la variable resultado por cada edge que haya en el grafo.
 result[i] = nuevo Edge()

```

Arrays.sort(edge)

```

Se ordena el arreglo edge en relacion a los pesos, de menor a mayor y se enstancia un subconjunto que va a tener el tamaño de la cantidad de los edges del grafo.

```

subConjunto subConjuntoTemporal[] = nuevo subConjunto[E]
para (i = 0 i < E ++i)
    subConjuntoTemporal[i] = nuevo subConjunto()

```

```

para (entero v = 0 v < V ++v)

```

Para cada vertice se le asigna un rango y un padre a cada subconjunto.

```

    subConjuntoTemporal[v].rango = 0
    subConjuntoTemporal[v].padre = v

```

```

i = 0
mientras (e < V - 1)

```

Mientras no se hayan recorrido todos los vertices entonces se crea un edge siguiente, despues se llama a la funcion find dos veces con el subconjunto que creamos antes y el origen del siguiente nodo para la primera llamada y el destino para la segunda, estos valores se guardan en variables "x" y "y".

```

    Edge next_edge = nuevo Edge()
    next_edge = edge[i++]
    entero x = find(subConjuntoTemporal, next_edge.src)
    entero y = find(subConjuntoTemporal, next_edge.dest)

```

```

    if (x != y)

```

Si x es diferente que y entonces el resultado en la posicion e + 1 sera el siguiente edge, y se llama a la funcion union con el subconjunto, "x" y "y".

```

        result[e++] = next_edge
        union(subConjuntoTemporal, x, y)

```

```

Imprime("El MST Resultante obtenido con Kruskal es el siguiente:")

```

Se imprime cada uno de las conexiones resultantes asi como su peso total y el tiempo de ejecucion.

```

    flotante total = 0
    para (i = 0 i < e ++i)
        total = total + result[i].weight
        Imprime("(" + result[i].src + ", " +
            result[i].dest + ", " + result[i].weight + ")")

```

```

    long endTime = System.currentTimeMillis()
    long totalTime = endTime - startTime
    Imprime("Tiempo de ejecucion: " + totalTime + " milisegundos")
    Imprime("Costo Total: " + total)
    regresa total

```

```

entero find(subConjunto subConjuntoTemporal[], entero i):

```

Funcion find la cual se llama desde kruskalMST
if (subConjuntoTemporal[i].padre != i)

Si el padre que se encuentra en el subconjunto es diferente que el valor de i entonces se llama recursivamente la funcion asi misma pero con el padre en el posicion de la i.

subConjuntoTemporal[i].padre = find(subConjuntoTemporal,
subConjuntoTemporal[i].padre)

Sino se regresa el padre que se encuentra en el subconjunto.
regresa subConjuntoTemporal[i].padre

void union(subConjunto subConjuntoTemporal[], entero x, entero y):

Funcion union que se llama desde kruskalMST.
entero xr = find(subConjuntoTemporal, x)

Se llama a la funcion find con el subconjunto y el valor de x y despues con el valor de y, y se guardan sus valores.

entero yr = find(subConjuntoTemporal, y)
if (xr == yr)

Si los valores anteriores son iguales, entonces no se hace nada, si el rango en el subconjunto en la posicion de xr es menor que el rango en el subconjunto en la posicion de yr, entonces el padre en la posicion xr sera yr. Sino si el rango en el subconjunto en la posicion de xr es mayor que el rango en el subconjunto en la posicion de yr entonces el padre en la posicion yr sera xr. Sino el padre en el subconjunto en la posicion yr sera xr y el rango en el subconjunto en la posicion xr sera el rango + 1.

regresa
else if(subConjuntoTemporal[xr].rango < subConjuntoTemporal[yr].rango)
subConjuntoTemporal[xr].padre = yr
else if(subConjuntoTemporal[xr].rango > subConjuntoTemporal[yr].rango)
subConjuntoTemporal[yr].padre = xr
else
subConjuntoTemporal[yr].padre = xr
subConjuntoTemporal[xr].rango++

flotante PrimMST(String archivo)throws FileNotFoundException:

La implementacion de PrimMST

Grafo g = nuevo Grafo()

Se lee el archivo y se asignan el numero de vertices y de aristas que hay en el grafo y se asignan los valores para poder llevar a cabo el algoritmo.

long startTime = System.currentTimeMillis()
Scanner en = nuevo Scanner(nuevo File(archivo))
V = en.nextEntero()
E = en.nextEntero()


```

vertices = nuevo ArrayList<ArrayList<entero[]>>()
para (entero i = 0 i < V i++)
    vertices.add( nuevo ArrayList<entero[]>())

para (entero i = 0 i < E i++)
    entero u = en.nextEntero()
    entero v = en.nextEntero()
    entero cost = en.nextEntero()
    vertices.get(u - 1).add( nuevo entero[] v, cost )
    vertices.get(v - 1).add( nuevo entero[] u, cost )

in.close()

```

Se cierra el archivo y se llama la funcion computeMST, la cual su resultado se guardara.

```

flotante res = g.computeMST()
long endTime = System.currentTimeMillis()

```

Se imprime el tiempo de ejecucion, las conexiones resultantes y el costo total.

```

long totalTime = endTime - startTime
Imprime("El tiempo de ejecucion fue de: " + totalTime + " milisegundos")
Imprime("El costo total es de: "+res)
regresa res

```

Se regresa el costo total.

```

entero computeMST():

```

La implementacion de computeMST.

```

explorado = nuevo boolean[V]

```

Se enicializa la variable explorado con el numero de vertices que tiene el grafo y el primer valor se asigna como explorado. Se hace un nuevo heap para los no explorados.

```

explorado[0] = true
noExplorado = nuevo Heap()
entero sumCost = 0
para (entero i = 1 i < V i++)

```

Para cada i hasta que i sea menor que el numero de vertices, el costo minimo sera de 1000000000.

```

entero minCost = 1000000000
para (entero[] edge : vertices.get(i))

```

Para cada edge que hay en los vertices en la posicion i, si el primer edge es igual a 1 y el edge en la posicion siguiente es menor que el costo minimo, entonces el costo minimo es el edge en la posicion 1.

```

    if (edge[0] == 1 && edge[1] < minCost)
        minCost = edge[1]

```

```

    noExplorado.add( nuevo entero[] i + 1, minCost )

```

Se agrega a no explorado $i + 1$ y el valor del costo minimo.

```
para (entero i = 1 i < V i++)
```

Para cada i que empieza en 1, hasta el numero de vertices en el grafo, entonces se explora cada edge que esta en los vertices, si el costo es menor que el no explorado con el costo minimo en el vertice inicial, entonces el no explorado sera el vertice inicial con el costo que se comparo.

```
    entero v = 0
    entero[] u = noExplorado.poll()
    para (entero[] edge : vertices.get(u[0] - 1))
        v = edge[0]
        if (explorado[v - 1] == false)
            entero cost = edge[1]
            if (cost < noExplorado.costoMinimo(v))
                noExplorado.update( nuevo entero[] v, cost )
```

```
    Imprime("(" + u[0] + ", " + v + ", " + u[1] + ")")
```

Se imprime cada conexion, se marca el nodo actual como explorado, y se acumulan los valores del peso.

```
    explorado[u[0] - 1] = true
    sumCost = sumCost + u[1]
```

```
    regresa sumCost
```

Se regresa el valor de la suma total del costo.

```
void main (String[] args) throws Exception:
```

```
kruskalUF("P2Edges.txt")
kruskalDFS("P2Edges.txt")
PrimMST("P2Edges.txt")
```

Referencias

- Apoyo y asesoría por Enrique Lira Martinez y Daniela Flores Javier
- <https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/>