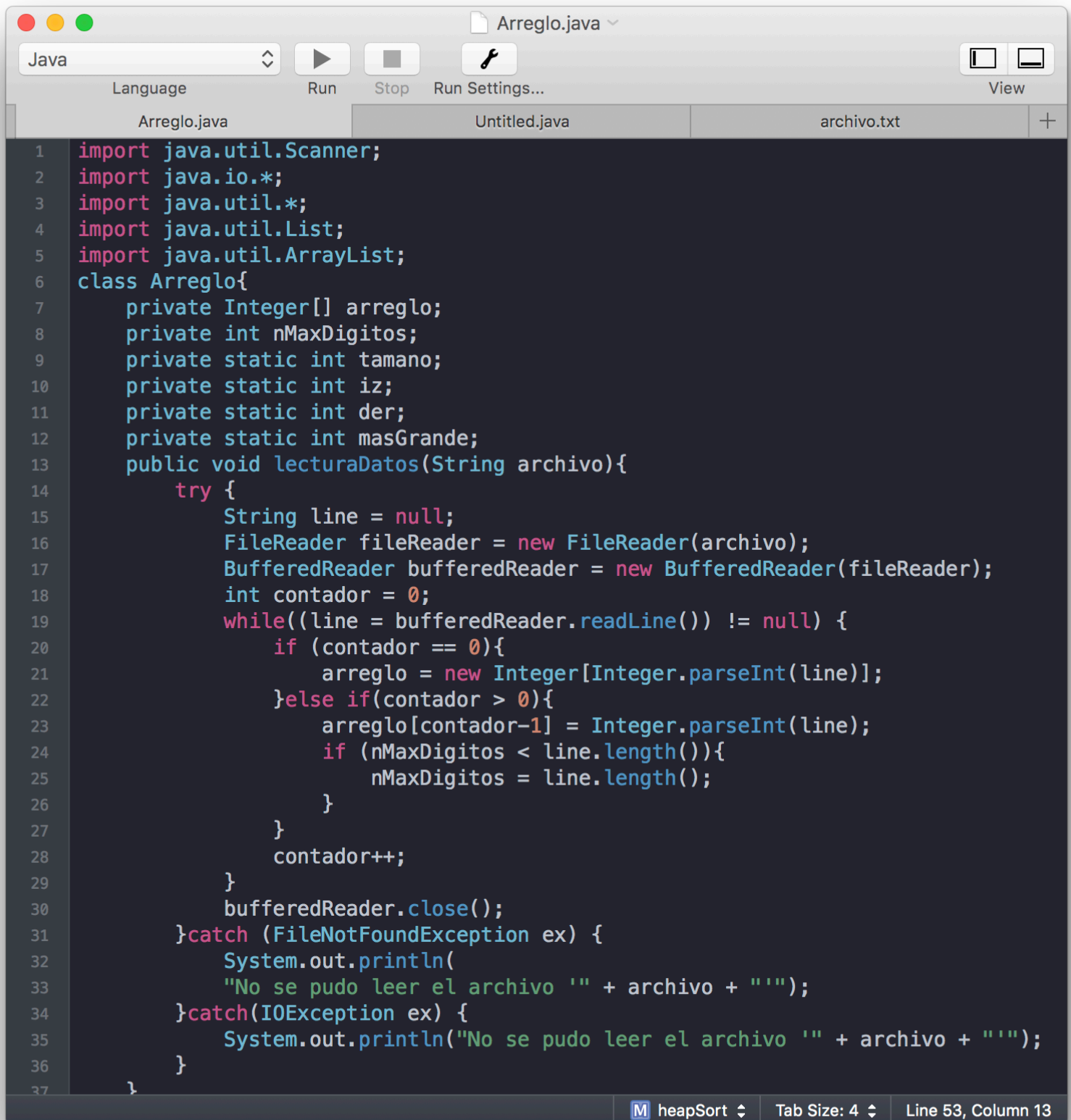


Documentación Proyecto 3

Emiliano Abascal Gurría

A01023234

Instituto Tecnológico de Estudios Superiores de Monterrey
Campus Santa Fe



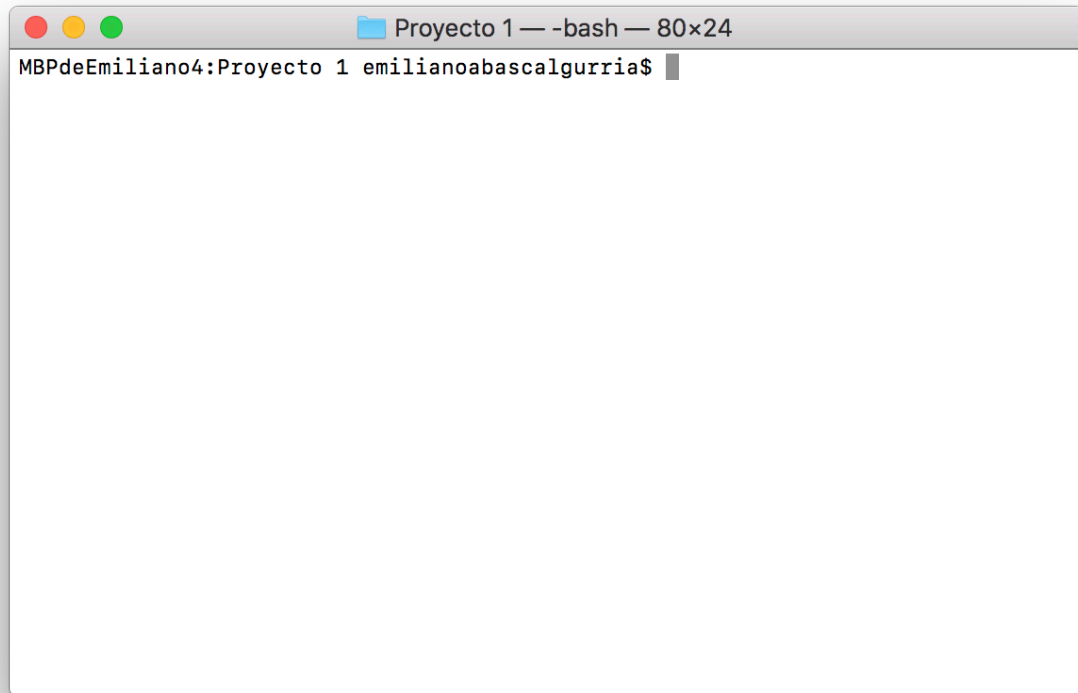
The screenshot shows an IDE window titled 'Arreglo.java'. The interface includes a top bar with a language dropdown set to 'Java', and buttons for 'Run', 'Stop', and 'Run Settings...'. Below this is a tab bar with 'Arreglo.java', 'Untitled.java', and 'archivo.txt'. The main editor area displays the following Java code:

```
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    }
38 }
```

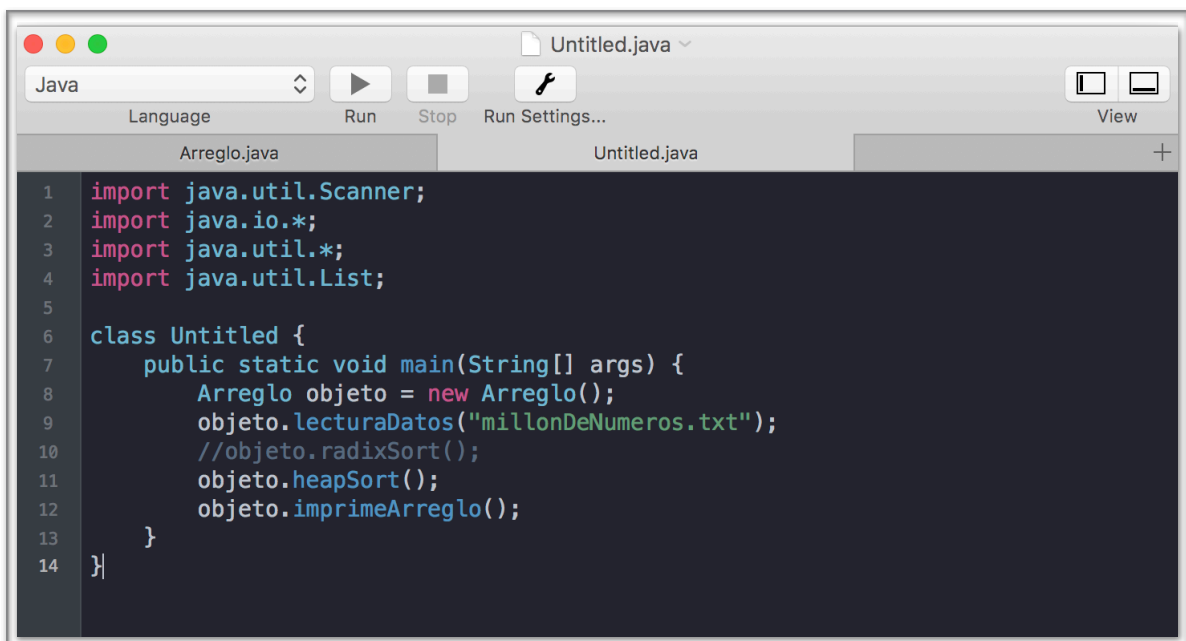
The status bar at the bottom indicates 'M heapSort', 'Tab Size: 4', and 'Line 53, Column 13'.

Manual de usuario:

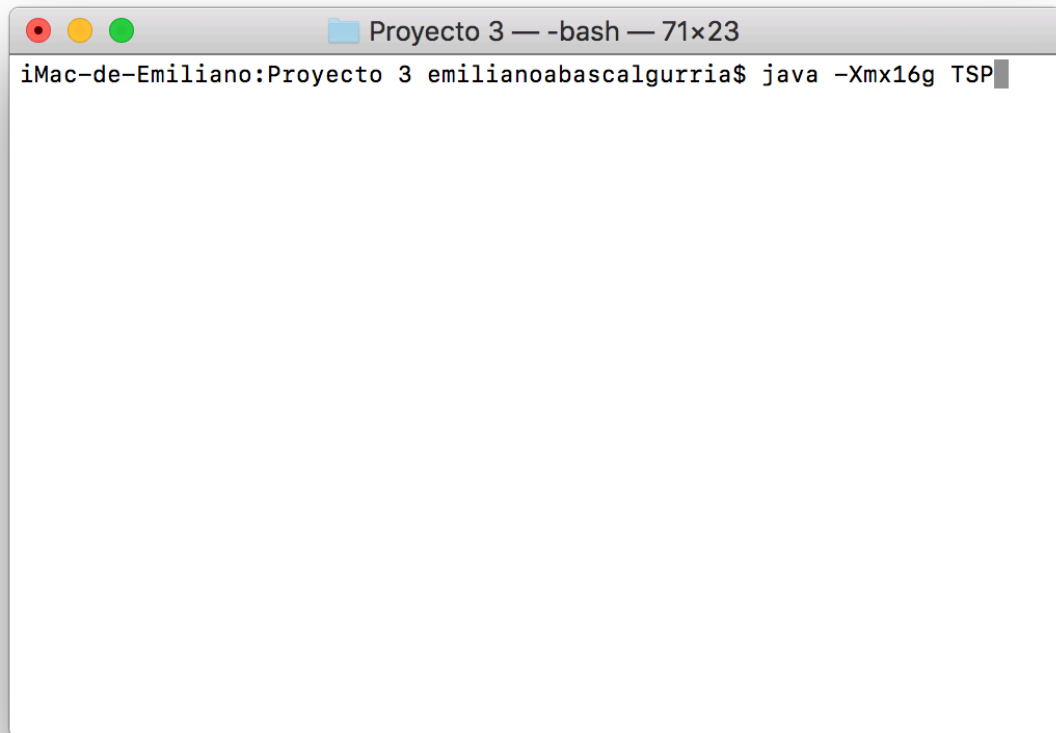
- Una vez obtenido el software ubicarlo en un directorio de preferencia, por ejemplo en el escritorio.
- Abrir la terminal o un compilador para correr programas hechos en Java.
- Escribir el siguiente comando: `cd "El directorio donde se encuentra el programa."`



- 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 TSP.java` da enter.
- Escribe en la terminal `java -Xmx16g` para darle la suficiente memoria al programa en caso de que se corra el archivo con 25 ciudades y el nombre de la clase, como se demuestra en el ejemplo a continuación.



```
Proyecto 3 — -bash — 71x23
iMac-de-Emiliano:Proyecto 3 emilianoabascalgurria$ java -Xmx16g TSP
```

Algoritmo

1. Inicio.
2. Nueva clase "TSP":
 1. Metodos:
 1. TSPDP(String archivo)
 2. TSP(entero nodoInicial, doble[][] distancia)
 3. List<Enteroeger> getrecorrido()
 4. doble getrecorridoCost()
 5. void solucionar()
 6. doble tsp(entero i, entero estadoInicial, Doble[][] memoria, Enteroeger[][] anterior)
 7. void bruteForce(String file)
 8. entero[] readFiles(String file)
 9. void evaluar()
 10. entero getCost(entero currentCity, entero input[], entero size)
 11. entero getMin(entero currentCity, entero input[], entero size)
 12. void constructorDeCamino()
 13. void prenteroResult()
 14. main(String[] args)
 2. Atributos

1. private final entero N
2. private final entero START_NODE
3. private final entero FINISHED_estadoInicial
4. public static entero pesoFB[][]
5. public static entero n
6. public static entero camino[]
7. public static entero cost
8. public static entero xValor[]
9. public static entero yValor[]
10. public static doble[][] aryNumbers
11. public static doble[][] costos
12. private doble[][] distancia
13. private doble costoMinimoDelRecorrido = Doble.POSITIVE_INFINITY
14. private List<Enteroeger> recorrido = nuevo ArrayList<>()
15. private boolean ranSolver = false
3. Funcionamiento
 1. TSPDP(String archivo)
 1. Función para leer un archivo e enteroroducirlo a un arreglo, agregamos la primera letra y se llena el grado y se imprimen los valores de la solución.
 2. TSP(entero nodoinicial, doble[][] distancia)
 1. Función para iniciar la solución del agente viajero, Cuando todos los nodos han sido visitados, entonces se regresara el estado final.
 3. List<Enteroeger> getrecorrido()
 1. Regresa el camino mas corto del agente viajero.
 2. Si no se ha corrido la función solve() entonces que se corra y se regrese la variable recorrido.
 4. doble getrecorridoCost()
 1. Regresa el costo del camino mas corto del agente viajero.
 5. void solucionar()
 1. Función solución
 2. Se establece el nodo inicial, dos arreglos, memoria y anterior los cuales guardaran los estados anteriores, así como el costo mínimo del recorrido, el cual su valor sera la llamada de la función tcp().
 3. Mientras sea verdad, entonces se agrega el estado actual al recorrido y se establece el siguiente estado que se metera al recorrido, pero si este es nulo entonces se sale del ciclo, sino el siguiente estado va a ser el estado inicial o el siguiente estado.
 4. Se establece el ultimo nodo como el inicial y se establece que se ha resuelto.
 6. doble tsp(entero i, entero estadoInicial, Doble[][] memoria, Enteroeger[][] anterior)
 1. Funcion principal para resolucion el TSP, recibe un indice, un estado inicial y los estados anteriores.
 2. Si ya se hizo el estado actual entonces regresa el costo del estado actual al inicial. Si el resultado guardado ya fue calculado, entonces regresa el calculado en el indice y el estado inicial.
 3. Desde 0 hasta el tamaño del arreglo, entonces si el siguiente estado ya fue visitado entonces se salta, sino el siguiente estado sera el estado inicial.
 4. Se le asigna el valor de la distancia actual mas la llamada recursiva de la función tcp.
 5. Si el costo nuevo es menor que el costo mínimo entonces el costo mínimo sera el nueo y el indice actual sera el siguiente.
 6. Se agrega el indice a los estados visitados y se regresa el costo mínimo hasta ahora.
 7. void bruteForce(String file)
 1. Función para empezar el TCP con fuerza bruta.
 2. Se lee el archivo y se asignan los valores oteidos a las variables que se utilizaran en la solución del problema.

3. Se calculan los pesos que hay entre los nodos(Distancias Euclidianas) y se llama la función evaluar para empezar a solucionar el problema.
8. entero[] readFiles(String file)
 1. Función para leer el archivo.
9. void evaluar()
 1. Función para evaluar cada nodo en el grafo, para i que es igual a 1, va a asignar a la variable cost, el valor de la llamada de la función getCost, pasandole subconjuntos actuales y después se llamara el constructorDeCamino.
10. entero getCost(entero currentCity, entero input[], entero size)
 1. Función que se utilizara para conseguir el costo.
 2. Si el tamaño es igual a 0 entonces se regresara el peso en la ciudad actual y en el índice 0, después se establece el valor del costo máximo posible.
 3. Se inicializa un nuevo conjunto, después si el input en el espacio de i es diferente al input en j, entonces se establece un nuevo costo.
 4. Se llama recursivamente la función getCost.
11. entero getMin(entero currentCity, entero input[], entero size)
 1. Función que se utiliza para conseguir el valor mínimo.
 2. Si el tamaño es igual a 0 entonces se regresara el peso en la ciudad actual y en el índice 0, después se establece el valor del costo máximo posible.
 3. Se inicializa un nuevo conjunto, después si el input en el espacio de i es diferente al input en j, entonces se establece un nuevo costo.
 4. Se inicializa un nuevo conjunto, después si el input en el espacio de i es diferente al input en j, entonces se establece un nuevo costo.
 5. Se llama a la función getCost.
12. void constructorDeCamino()
 1. Función para construir el camino.
 2. Para i que es igual a 1, i tiene que ser menor que n, entonces se establece que el último set en i-1 sera i, el nuevo tamaño sera n - 1, el camino en su posición inicial sera la llamada de getMin().
 3. Se establece el siguiente set.
 4. Se llama la impresión del resultado.
13. void prenteroResult()
 1. Función para imprimir el camino y su costo.
14. main(String[] args)
 1. función main en el cual se llaman las funciones.

Descripción Técnica

Clase publica TSP

```

final entero N
final entero START_NODE
final entero FINISHED_estadoInicial
public static entero pesoFB[][]
public static entero n
public static entero camino[]
public static entero cost
public static entero xValor[]
public static entero yValor[]
public static doble[][] aryNumbers
public static doble[][] costos
doble[][] distancia

```

```

doble costoMinimoDelRecorrido = Doble.POSITIVE_INFINITY
List<Enteroeger> recorrido = nuevo ArrayList<>()
boolean correSolucion = false

```

```

void TSPDP(String archivo)throws FileNotFoundException

```

Funcion para leer un archivo e enteroroducirlo a un arreglo.

```

entero V, E
doble x1,y1,x2,y2,x,y
Scanner in = nuevo Scanner(nuevo File(archivo))
V = in.nextEntero()

```

agregamos la primera letra

```

aryNumbers = nuevo doble[V][2]
costos = nuevo doble[V][V]
para (entero i = 0 i < V i++)

```

Se llena el grafo

```

    para (entero j = 0 j < 2 j++)
        aryNumbers[i][j] = in.nextDoble()

para (entero i = 0 i < V i++)
    x1 = aryNumbers[i][0]
    y1 = aryNumbers[i][1]
    para (entero j = 0 j < V j++)
        x2 = aryNumbers[j][0]
        y2 = aryNumbers[j][1]
        x = x2-x1
        y = y2-y1
        costos[i][j]=Math.sqrt(Math.pow(x, 2) + Math.pow(y, 2))

```

Se utiliza para imprimir los valores de la solución.

```

entero nodoInicial = 0
TSP solucion = nuevo TSP(nodoInicial, costos)
System.out.prenteroln("Usando Programacion Dinamica:")
long startTime = System.currentTimeMillis()
System.out.prenteroln("Camino: " + solucion.getrecorrido())

System.out.prenteroln("Costo Del Camino: " + solucion.getrecorridoCost())
long endTime = System.currentTimeMillis()
long totalTime = endTime - startTime
System.out.prenteroln("Milisegundos: " + totalTime)

```

```
public TSP(entero nodoInicial, doble[][] distancia)
```

Función para iniciar la solución del agente viajero, Cuando todos los nodos han sido visitados, entonces se regresara el estado final.

```
    esta.distancia = distancia  
    N = distancia.length  
    START_NODE = nodoInicial  
    FINISHED_estadoInicial = (1 << N) - 1
```

Regresa el camino mas corto del agente viajero

```
public List<Enteroeger> getrecorrido()  
    si (!correSolucion)  
        solucionar()
```

Si no se ha corrido la funicon solucionar() entonces que se corra y se regrese la variable recorrido.

```
    regresa recorrido
```

Regresa el costo del camino mas corto del agente viajero

```
public doble getrecorridoCost()  
    si (!correSolucion) solucionar()  
    regresa costoMinimoDelRecorrido
```

Funcion solucion

```
public void solucionar()  
    entero estadoInicial = 1 << START_NODE
```

Se establece el nodo inicial, dos arreglos, memoria y anterior los cuales guardaran los estados anteriores, asi como el costo minimo del recorrido, el cual su valor sera la llamada de la funcion tcp().

```
    Doble[][] memoria = nuevo Doble[N][1 << N]  
    Entero[][] anterior = nuevo Enteroeger[N][1 << N]  
    costoMinimoDelRecorrido = tsp(START_NODE, estadoInicial, memoria,  
anterior)  
    entero index = START_NODE  
    mientras (verdadero)
```

Mientras sea verdad, entonces se agrega el estado actual al recorrido y se establece el siguiente estado que se meta al recorrido, pero si este es nulo entonces se sale del ciclo, sino el siguiente estado va a ser el estado inicial o el siguiente estado.

```
recorrido.agregar(index)
Entero siguienteIndice = anterior[index][estadoInicial]
si (siguienteIndice == null) break
entero siguienteEstado = estadoInicial | (1 << siguienteIndice)
estadoInicial = siguienteEstado
index = siguienteIndice
```

```
recorrido.agregar(START_NODE)
```

Se establece el ultimo nodo como el inicial y se establece que se ha resuelto.

```
correSolucion = verdadero
```

Funcion principal para resolucion el TSP, recibe un indice, un estado inicial y los estados anteriores.

```
doble tsp(entero i, entero estadoInicial, Doble[][] memoria, Enteroeger[][] anterior)
si (estadoInicial == FINISHED_estadoInicial)
```

Si ya se hizo el estado actual entonces regresa el costo del estado actual al inicial. Si el resultado guardado ya fue calculado, entonces regresa el calculado en el indice y el estado inicial.

```
regresa distancia[i][START_NODE]
```

```
si (memoria[i][estadoInicial] != null)
regresa memoria[i][estadoInicial]
```

```
doble minCost = Doble.POSITIVE_INFINITY
entero index = -1
para (entero next = 0 next < N next++)
```

Desde 0 hasta el tamaño del arreglo, entonces si el siguiente estado ya fue visitado entonces se salta, sino el siguiente estado sera el estado inicial.

```
si ((estadoInicial & (1 << next)) != 0) continue
entero siguienteEstado = estadoInicial | (1 << next)
doble nuevoCost = distancia[i][next] + tsp(next, siguienteEstado, memoria,
anterior)
```

Se le asigna el valor de la distancia actual mas la llamada recursiva de la funcion tcp si el costo nuevo es menor que el costo minimo entonces el costo minimo sera el nuevo y el indice actual sera el siguiente.

```
si (nuevoCost < minCost)
```



```
minCost = nuevoCost
index = next
```

```
anterior[i][estadoInicial] = index
```

se agrega el indice a los estados visitados y se regresa el costo minimo hasta ahora.
regresa memoria[i][estadoInicial] = minCost

```
public static void bruteParace(String file)
```

Función para empezar el TCP con fuerza bruta.

```
Scanner s = nuevo Scanner (System.in)
```

Se lee el archivo y se asignan los valores obtenidos a las variables que se utilizaran en la solución del problema.

```
entero[] Array = readFiles(file)
n = Array[0]
pesoFB = nuevo entero[n][n]
xValor = nuevo entero[n]
yValor = nuevo entero[n]
camino = nuevo entero[n-1]
entero k = 0
para (entero i=1 i<Array.length i = i+2)
    xValor[k] = Array[i]
    k++
```

```
entero d = 0
para (entero i=2 i<Array.length i = i+2)
    yValor[d] = Array[i]
    d++
```

```
para (entero i = 0 i < n i++)
```

Se calculan los pesos que hay entre los nodos(Distancias Eucledianas) y se llama la función evaluar para empezar a solucionar el problema.

```
para (entero j = 0 j < n j++)
    doble x = Math.pow((xValor[j] - xValor[i]),2) entero x_ = (entero)
    doble y = Math.pow((yValor[j] - yValor[i]),2)
    entero y_ = (entero) y
    doble r = Math.sqrt( x_ + y_ )
    entero w = (entero) r
    pesoFB[i][j] = w
```

```
evaluar()
```

Función para leer el archivo.

```

public static entero[] readFiles(String file)
    try
        File f = nuevo File(file)
        Scanner s = nuevo Scanner(f)
        entero ctr = 0
        mientras (s.hasNextEntero())
            ctr++
            s.nextEntero()

        entero[] Array = nuevo entero [ctr]
        Scanner s1 = nuevo Scanner(f)
        para (entero i = 0 i < Array.length i++)
            Array[i] = s1.nextEntero()
        retorna Array

    catch(Exception e)
        retorna null

```

```

public static void evaluar()

```

Función para evaluar cada nodo en el grafo, para i que es igual a 1, va a asignar a la variable cost, el valor de la llamada de la función getCost, pasandole subconjuntos actuales y después se llamara el constructorDeCamino.

```

    entero trySet[] = nuevo entero [n-1]
    para(entero i = 1 i < n i++)
        trySet[i-1] = i
        cost = getCost(0, trySet, n-1)
        constructorDeCamino()

```

```

public static entero getCost(entero currentCity, entero input[], entero size)

```

Funcion que se utilizara para conseguir el costo.

```

    si (size == 0)

```

Si el tamaño es igual a 0 entonces se regresara el peso en la ciudad actual y en el indice 0, despues se establece el valor del costo maximo posible.

```

        retorna pesoFB[currentCity][0]
        entero min=999999
        entero minIndex = 0
        entero setToCost[] = nuevo entero[n-1]
        para (entero i = 0 i < size i++)
            entero k = 0

```

Se inicializa un nuevo conjunto, despues si el input en el espacio de i es dsierente al input en j, entonces se establece un nuevo costo.

```
para (entero j = 0 j < size j++)  
    si(input[i] != input[j])  
        setToCost[k++] = input[j]
```

```
entero tmp = getCost(input[i], setToCost, size-1)
```

Se llama recursivamente la funcion **getCost**.

```
si((pesoFB[currentCity][input[i]]+tmp)<min)  
    min = pesoFB[currentCity][input[i]]+tmp  
    minIndex = input[i]
```

```
regresa min
```

```
public static entero getMin(entero currentCity, entero input[], entero size)
```

Funcion que se utiliza para conseguir el valor minimo.

```
si(size == 0)  
    regresa pesoFB[currentCity][0]
```

Si el tamaño es igual a 0 entonces se regresara el peso en la ciudad actual y en el indice 0, despues se establece el valor del costo máximo posible.

```
entero min = 999999  
entero minIndex = 0  
entero setToCost[] = nuevo entero [n-1]  
para (entero i = 0 i < size i++)
```

Se inicializa un nuevo conjunto, después si el input en el espacio de i es dsierente al input en j, entonces se establece un nuevo costo.

```
entero k = 0
```

Se inicializa un nuevo conjunto, después si el input en el espacio de i es dsierente al input en j, entonces se establece un nuevo costo.

```
para (entero j = 0 j < size j++)  
    si(input[i] != input[j])  
        setToCost[k++] = input[j]
```

```
entero tmp = getCost(input[i], setToCost, size-1)
```

Se llama a la funcion **getCost**.

```
si((pesoFB[currentCity][input[i]]+tmp) < min)  
    min = pesoFB[currentCity][input[i]]+tmp  
    minIndex = input[i]
```

```
regresa minIndex
```

```
public static void constructorDeCamino()
```

Función para construir el camino.

```
entero ultimoSet[] = nuevo entero[n - 1]
entero siguienteSet[] = nuevo entero[n - 2]
para(entero i = 1 i < n i++)
```

Para i que es igual a 1, i tiene que ser menor que n, entonces se establece que el ultimo set en i-1 sera i, el nuevo tamaño sera n - 1, el camino en su posición inicial sera la llamada de getMin().

```
ultimoSet[i - 1] = i
entero size = n - 1
camino[0] = getMin(0,ultimoSet,size)
```

```
para(entero i = 1 i < n-1 i++)
```

Se establece el siguiente set.

```
entero k = 0
para(entero j = 0 j < size j++)
    si(camino[i - 1] != ultimoSet[j])
        siguienteSet[k++] = ultimoSet[j]
```

```
--size
camino[i] = getMin(camino[i-1], siguienteSet, size)
para(entero j = 0 j < size j++)
    ultimoSet[j] = siguienteSet[j]
```

```
prenteroResult()
```

Se llama la impresión del resultado.

```
public static void prenteroResult()
```

Función para imprimir el camino y su costo.

```
System.out.prentereln("Usando Fuerza Bruta:")
System.out.prentero("Camino: [0, ")
para(entero i = 0 i < n-1 i++)
    System.out.prentero((camino[i] )+" , ")
```

```
System.out.prentereln("0]")
System.out.prentereln("Costo del Camino: " + cost)
```

```
public static void main(String[] args) throws FileNotFoundException
```

función main en el cual se llaman las funciones.

```
long startTime = System.currentTimeMillis()
bruteForce("mediumMap.txt")
long endTime = System.currentTimeMillis()
long totalTime = endTime - startTime
System.out.println("Milisegundos: " + totalTime)
System.out.println()

TSPDP("mediumMap.txt")
```

Resultados

Con el archivo de 25 ciudades solo se puede correr con programación dinamica, ya que por fuerza bruta nunca acabo ya que su orden es muy grande.

Con estas especificaciones:

procesador a 4.0 GHz

memoria 16gb

Tiempo: 3.53 minutos

```
iMac-de-Emiliano:Proyecto 3 emilianoabascalgurria$ java -Xmx16g TSP
Usando Programacion Dinamica:
[Camino: [0, 4, 7, 3, 2, 6, 8, 12, 13, 15, 23, 24, 19, 16, 20, 22, 21, 17, 18, 14, 11, 10, 9, 5, 1, 0]
Costo Del Camino: 26442.73030895475
[Milisegundos: 211974
```

Con el archivo de 10 ciudades:

Tiempo con fuerza bruta: 85 milisegundos

Tiempo con Programación dinámica: 2 milisegundos

```
Usando Fuerza Bruta:
Camino: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
Costo del Camino: 10
Milisegundos: 85

Usando Programacion Dinamica:
Camino: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
Costo Del Camino: 10.0
Milisegundos: 2
```

Con el archivo de 4 ciudades

Tiempo con fuerza bruta: 35 milisegundos

Tiempo con Programación dinámica: 2 milisegundos

```
Usando Fuerza Bruta:  
Camino: [0, 1, 2, 3, 0]  
Costo del Camino: 12  
Milisegundos: 35  
  
Usando Programacion Dinamica:  
Camino: [0, 1, 2, 3, 0]  
Costo Del Camino: 12.0  
Milisegundos: 2
```

Referencias

- Apoyo y asesoría por Enrique Lira Martinez y Daniela Flores Javier