

# Que necesita el sur-sureste para desarrollarse?

Erick Gonzalez Parada ID: 178145

Matemáticas discretas, Universidad de las Américas Puebla, Puebla, México

November 17, 2023

## Abstract

Análisis de Dijkstra.

*Keywords:* lista, adjacencia, vertices y aristas

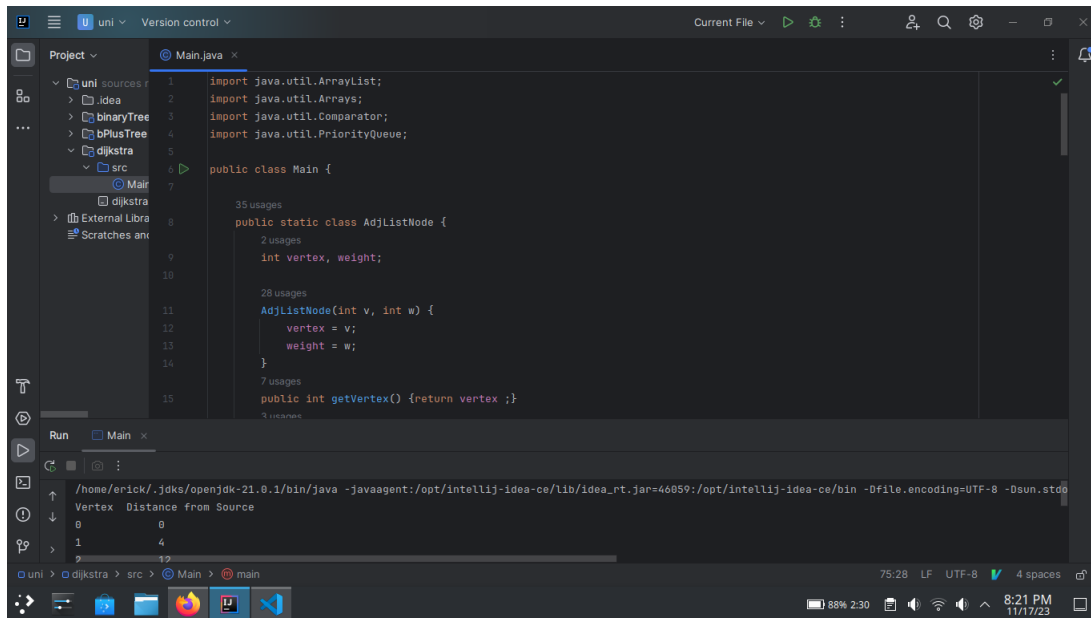
## 1 Introducción

El sur-sureste de México enfrenta desafíos significativos en términos de conexiones e infraestructura. La región se ve afectada por la falta de una red vial bien desarrollada, limitaciones en las conexiones ferroviarias, desigualdades en el desarrollo, carencias en la conectividad digital, necesidades en la infraestructura portuaria, desafíos topográficos, falta de inversiones y problemas socioeconómicos. Estos problemas, combinados, dificultan el desarrollo económico y social integral de la región. La superación de estos desafíos requerirá un enfoque integral que involucre inversiones significativas, planificación estratégica y coordinación entre diferentes niveles de gobierno y partes interesadas, mi objetivo es confirmar la desventaja de desconexión que tiene esta región de México

Además de los desafíos mencionados, es importante destacar que la región sur-sureste de México a menudo enfrenta dificultades relacionadas con la falta de acceso a servicios básicos, como educación y salud, lo que contribuye a la brecha socioeconómica. La planificación y ejecución de proyectos de infraestructura deben abordar estas necesidades fundamentales para lograr un desarrollo equitativo. Asimismo, la consideración de factores ambientales y sostenibles es esencial para mitigar impactos negativos en la ecología local.

## Dijkstra en Java

Nota: lo que se encuentra en el main de los fragmentos es una prueba de que el código funciona.

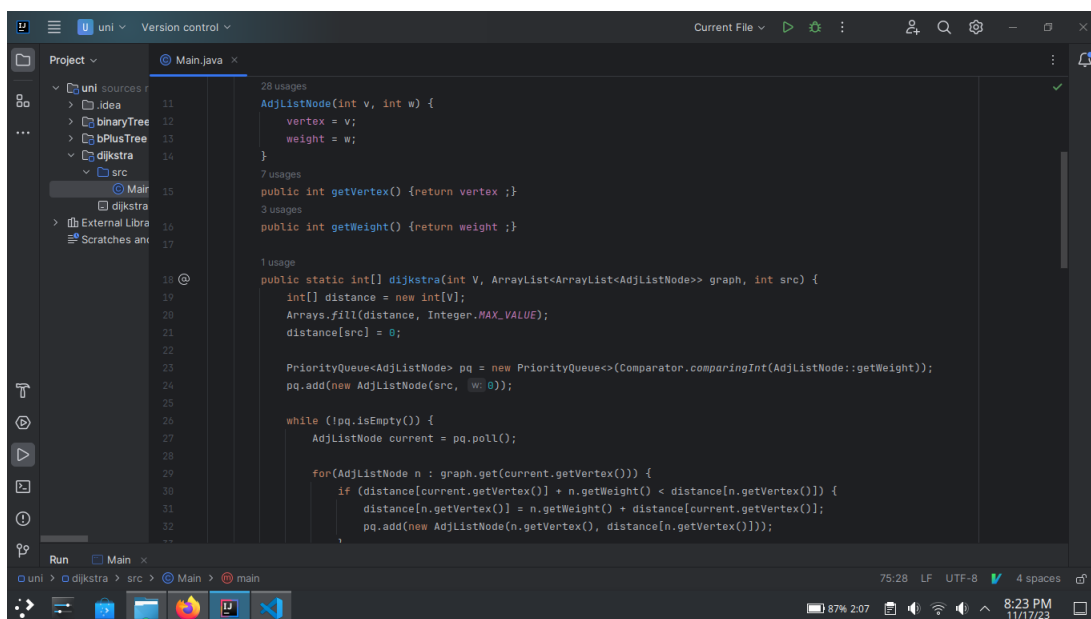


```

1  import java.util.ArrayList;
2  import java.util.Arrays;
3  import java.util.Comparator;
4  import java.util.PriorityQueue;
5
6  public class Main {
7
8      35 usages
9      public static class AdjListNode {
10
11          2 usages
12          int vertex, weight;
13
14          28 usages
15          AdjListNode(int v, int w) {
16              vertex = v;
17              weight = w;
18          }
19
20          7 usages
21          public int getVertex() {return vertex ;}
22
23          3 usages
24      }
25
26      1 usage
27      public static int[] dijkstra(int V, ArrayList<ArrayList<AdjListNode>> graph, int src) {
28          int[] distance = new int[V];
29          Arrays.fill(distance, Integer.MAX_VALUE);
30          distance[src] = 0;
31
32          PriorityQueue<AdjListNode> pq = new PriorityQueue<>(Comparator.comparingInt(AdjListNode::getWeight));
33          pq.add(new AdjListNode(src, 0));
34
35          while (!pq.isEmpty()) {
36              AdjListNode current = pq.poll();
37
38              for (AdjListNode n : graph.get(current.getVertex())) {
39                  if (distance[current.getVertex()] + n.getWeight() < distance[n.getVertex()]) {
40                      distance[n.getVertex()] = n.getWeight() + distance[current.getVertex()];
41                      pq.add(new AdjListNode(n.getVertex(), distance[n.getVertex()]));
42                  }
43              }
44          }
45          return distance;
46      }
47  }

```

Figura 1: code fragment 1

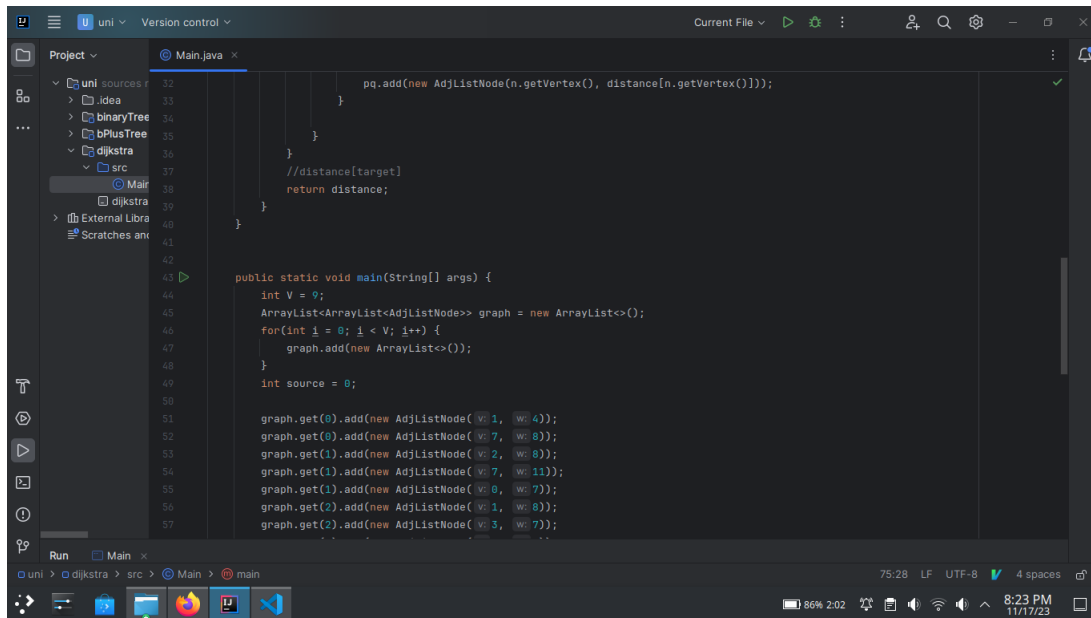


```

11  AdjListNode(int v, int w) {
12      vertex = v;
13      weight = w;
14  }
15
16  public int getVertex() {return vertex ;}
17
18  public int getWeight() {return weight ;}
19
20  1 usage
21  public static int[] dijkstra(int V, ArrayList<ArrayList<AdjListNode>> graph, int src) {
22      int[] distance = new int[V];
23      Arrays.fill(distance, Integer.MAX_VALUE);
24      distance[src] = 0;
25
26      PriorityQueue<AdjListNode> pq = new PriorityQueue<>(Comparator.comparingInt(AdjListNode::getWeight));
27      pq.add(new AdjListNode(src, 0));
28
29      while (!pq.isEmpty()) {
30          AdjListNode current = pq.poll();
31
32          for (AdjListNode n : graph.get(current.getVertex())) {
33              if (distance[current.getVertex()] + n.getWeight() < distance[n.getVertex()]) {
34                  distance[n.getVertex()] = n.getWeight() + distance[current.getVertex()];
35                  pq.add(new AdjListNode(n.getVertex(), distance[n.getVertex()]));
36              }
37          }
38      }
39      return distance;
40  }

```

Figura 2: code fragment 2

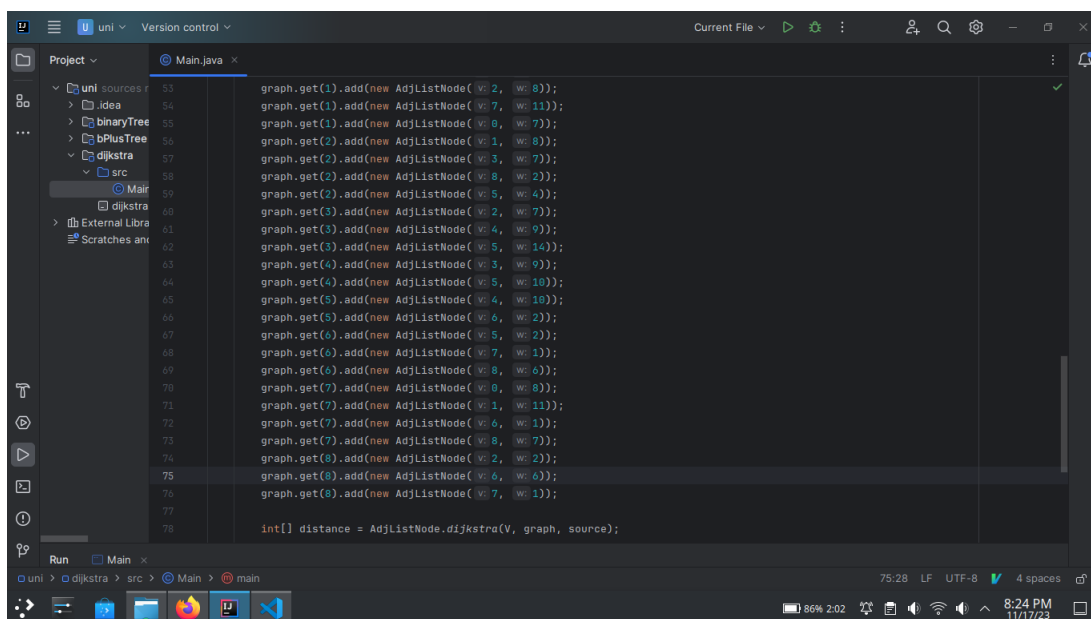


```

32      pq.add(new AdjListNode(n.getVertex(), distance[n.getVertex()]));
33    }
34  }
35
36  }
37
38  //distance[target]
39  return distance;
40
41  }
42
43  public static void main(String[] args) {
44    int V = 0;
45    ArrayList<ArrayList<AdjListNode>> graph = new ArrayList<>();
46    for(int i = 0; i < V; i++) {
47      graph.add(new ArrayList<>());
48    }
49    int source = 0;
50
51    graph.get(0).add(new AdjListNode(1, 4));
52    graph.get(0).add(new AdjListNode(7, 8));
53    graph.get(1).add(new AdjListNode(2, 8));
54    graph.get(1).add(new AdjListNode(7, 11));
55    graph.get(1).add(new AdjListNode(8, 7));
56    graph.get(2).add(new AdjListNode(1, 8));
57    graph.get(2).add(new AdjListNode(3, 7));
58    graph.get(2).add(new AdjListNode(8, 2));
59    graph.get(2).add(new AdjListNode(5, 4));
60    graph.get(3).add(new AdjListNode(2, 7));
61    graph.get(3).add(new AdjListNode(4, 9));
62    graph.get(3).add(new AdjListNode(5, 14));
63    graph.get(4).add(new AdjListNode(3, 9));
64    graph.get(4).add(new AdjListNode(5, 10));
65    graph.get(5).add(new AdjListNode(4, 10));
66    graph.get(5).add(new AdjListNode(6, 2));
67    graph.get(6).add(new AdjListNode(5, 2));
68    graph.get(6).add(new AdjListNode(7, 1));
69    graph.get(6).add(new AdjListNode(8, 6));
70    graph.get(7).add(new AdjListNode(8, 8));
71    graph.get(7).add(new AdjListNode(1, 11));
72    graph.get(7).add(new AdjListNode(6, 1));
73    graph.get(7).add(new AdjListNode(8, 7));
74    graph.get(8).add(new AdjListNode(2, 2));
75    graph.get(8).add(new AdjListNode(6, 6));
76    graph.get(8).add(new AdjListNode(7, 1));
77
78    int[] distance = AdjListNode.dijkstra(V, graph, source);

```

Figura 3: code fragment 3



```

33    graph.get(1).add(new AdjListNode(2, 8));
34    graph.get(1).add(new AdjListNode(7, 11));
35    graph.get(1).add(new AdjListNode(8, 7));
36    graph.get(2).add(new AdjListNode(1, 8));
37    graph.get(2).add(new AdjListNode(3, 7));
38    graph.get(2).add(new AdjListNode(8, 2));
39    graph.get(2).add(new AdjListNode(5, 4));
40    graph.get(3).add(new AdjListNode(2, 7));
41    graph.get(3).add(new AdjListNode(4, 9));
42    graph.get(3).add(new AdjListNode(5, 14));
43    graph.get(4).add(new AdjListNode(3, 9));
44    graph.get(4).add(new AdjListNode(5, 10));
45    graph.get(5).add(new AdjListNode(4, 10));
46    graph.get(5).add(new AdjListNode(6, 2));
47    graph.get(6).add(new AdjListNode(5, 2));
48    graph.get(6).add(new AdjListNode(7, 1));
49    graph.get(6).add(new AdjListNode(8, 6));
50    graph.get(7).add(new AdjListNode(8, 8));
51    graph.get(7).add(new AdjListNode(1, 11));
52    graph.get(7).add(new AdjListNode(6, 1));
53    graph.get(7).add(new AdjListNode(8, 7));
54    graph.get(8).add(new AdjListNode(2, 2));
55    graph.get(8).add(new AdjListNode(6, 6));
56    graph.get(8).add(new AdjListNode(7, 1));
57
58    int[] distance = AdjListNode.dijkstra(V, graph, source);

```

Figura 4: code fragment 4

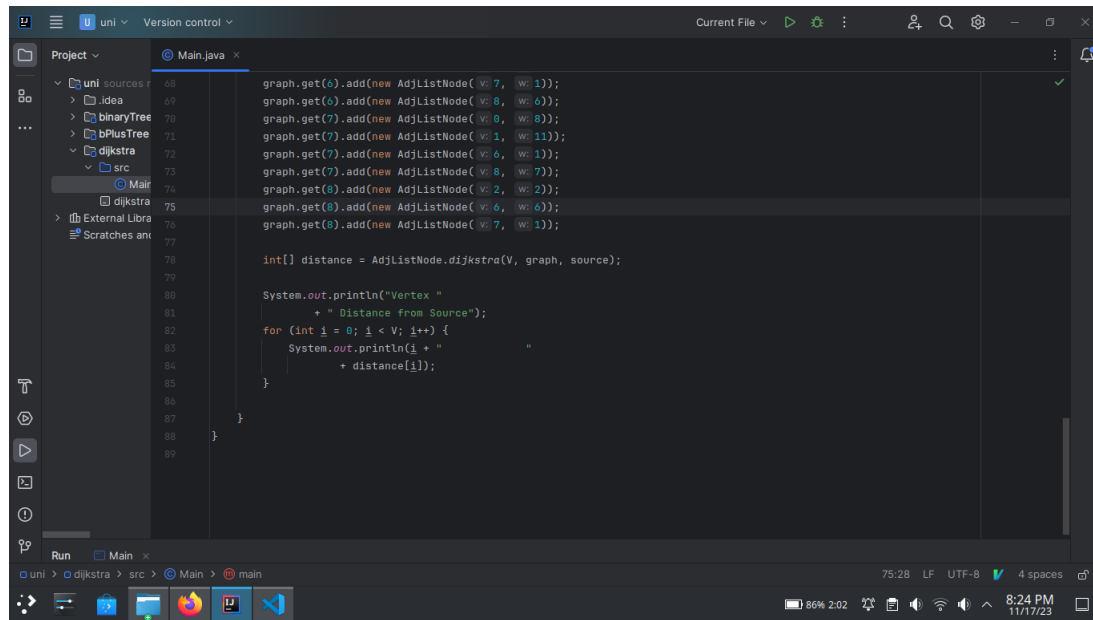


Figura 5: code fragment 5

## 2 Análisis de complejidad

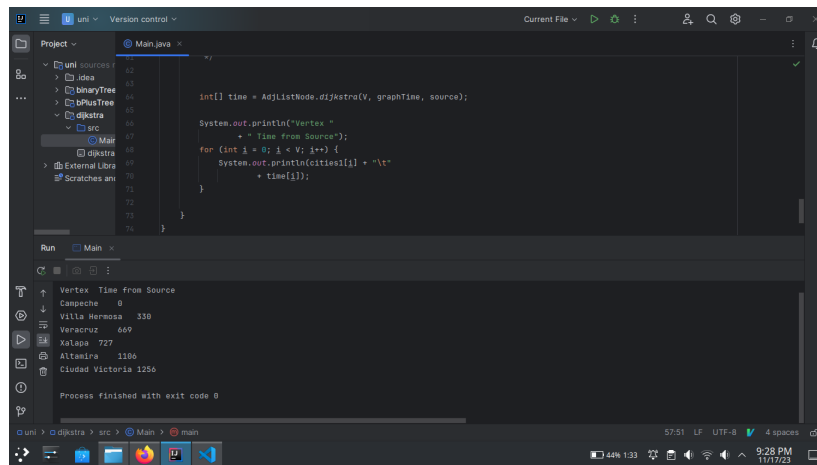
Por conteo de líneas obtenemos  $O(2 + V + 3 + E \log V + 2V - 1 + V + 1)$ , donde las constantes pues son las constantes el primer  $V$  (que sería como la  $n$  en otros algoritmos) es por el `Arrays.fill` luego el  $E \log V$  es por el `while` que analizado tiene ese comportamiento y lo restante es lo del último `if` statement.

La complejidad temporal del código parece  $O(V^2)$  ya que hay dos bucles `while` anidados. Si miramos más de cerca, podemos observar que las declaraciones en el bucle interno se ejecutan  $O(V + E)$  veces (similar a BFS). El bucle interno tiene una operación de `decreaseKey()` que requiere un tiempo  $O(\log V)$ . Entonces, la complejidad del tiempo general es  $O(E + V) * O(\log V)$ , que es  $O((E + V) * \log V) = O(E \log V)$ .

## 3 Análisis de caminos mínimos

El camino de campeche con ciudad victoria que es su capital más cercana con la frontera de USA tenemos el siguiente resultado

Tiempo



```

1  //
2
3  int[] time = AdjListNode.dijkstra(V, graphTime, source);
4
5  System.out.println("Vertex "
6      + " Time from Source");
7
8  for (int i = 0; i < V; i++) {
9      System.out.println(cities[i] + "\t"
10         + time[i]);
11  }
12  }

```

Run Main

Vertex Time from Source

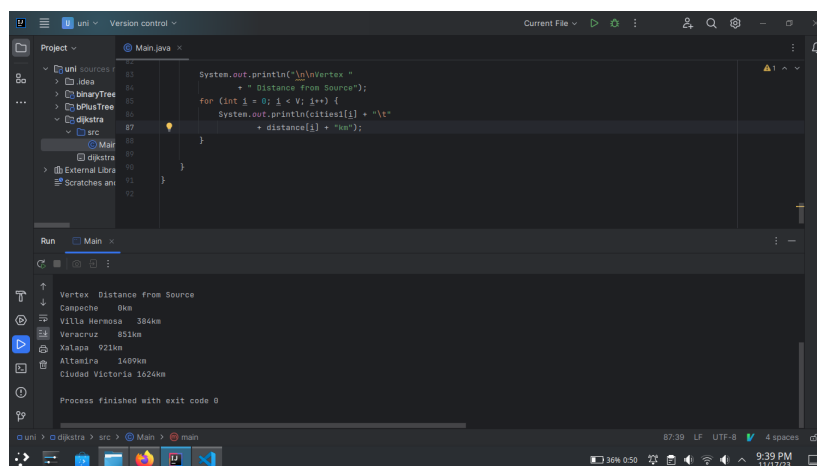
Vertex	Time from Source
Campeche	0
Villa Hermosa	330
Veracruz	669
Xalapa	727
Altamira	1196
Ciudad Victoria	1256

Process finished with exit code 0

Figura 6: time

el tiempo esta en minutos por lo que si ponemos una calculadora de minutos a hora, nos da un total de 21 hrs prácticamente.

Distancia



```

1  System.out.println("\n\nVertex "
2      + " Distance from Source");
3
4  for (int i = 0; i < V; i++) {
5      System.out.println(cities[i] + "\t"
6          + distance[i] + "km");
7  }
8  }

```

Run Main

Vertex Distance from Source

Vertex	Distance from Source
Campeche	0km
Villa Hermosa	384km
Veracruz	851km
Xalapa	921km
Altamira	1409km
Ciudad Victoria	1624km

Process finished with exit code 0

Figura 7: time

En la figura 7 ya nos da el resultado final: 1624 km  
Ahora con tuxtla gutierrez chiapas que termina siendo casi lo mismo

```

/home/erick/.jdk/openjdk-21.0.1/bin/java -javaagent:/opt/intellij-idea-ce/lib/idea_rt.jar=44445:/opt/intellij-idea-ce/bin -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8
Vertex Time (in minutes) from Source
Tuxtla 0mins
Villa Hermosa 234mins
Veracruz 577mins
Xalapa 61mins
Altamira 1018mins
Ciudad Victoria 1160mins

Vertex Distance from Source
Tuxtla 0km
Villa Hermosa 246km
Veracruz 713km
Xalapa 783km
Altamira 1271km
Ciudad Victoria 1486km

Process finished with exit code 0

```

Figura 8: time &amp; distance

Guerrero(chilpancingo)

```

/home/erick/.jdk/openjdk-21.0.1/bin/java -javaagent:/opt/intellij-idea-ce/lib/idea_rt.jar=34793:/opt/intellij-idea-ce/bin -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8
Vertex Time (in minutes) from Source
Chilpancingo 0mins
Toluca 244mins
Queretaro 412mins
San Luis Potosi 559mins
Ciudad Victoria 817mins

Vertex Distance from Source
Chilpancingo 0km
Toluca 324km
Queretaro 528km
San Luis Potosi 732km
Ciudad Victoria 1067km

Process finished with exit code 0

```

Figura 9: time &amp; distance

Oaxaca(Oaxaca de Juarez)

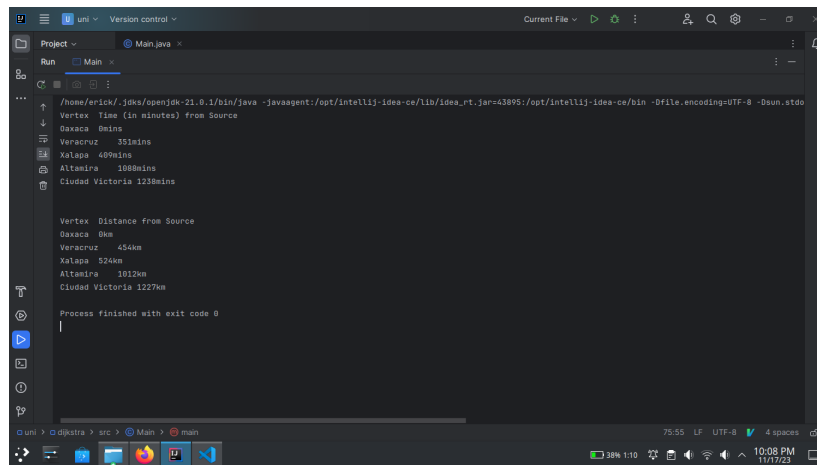


Figura 10: time &amp; distance

Quintana Roo(Chetumal)

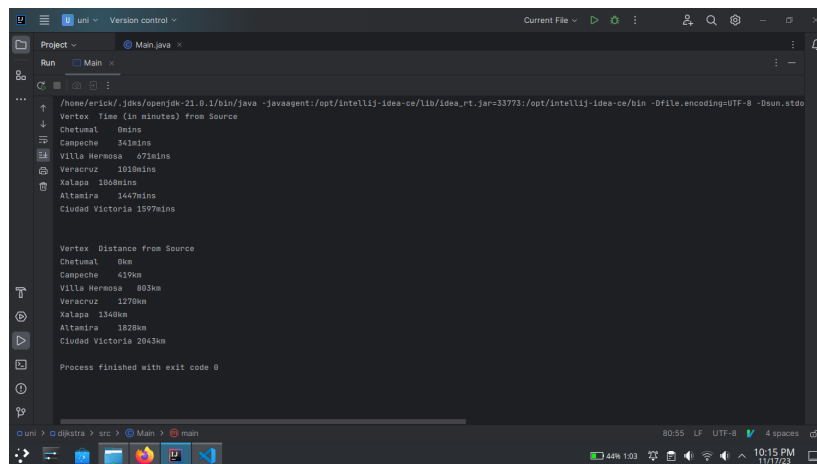


Figura 11: time &amp; distance

Tabasco(Villa hermosa)

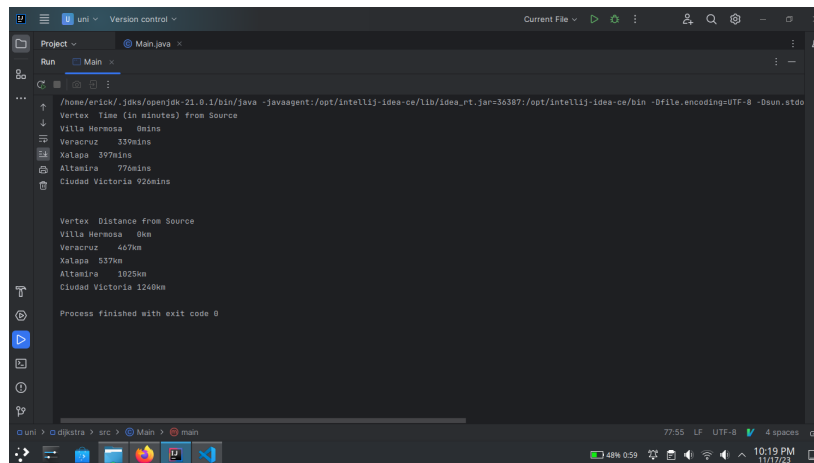


Figura 12: time &amp; distance

Yucatan(merida)

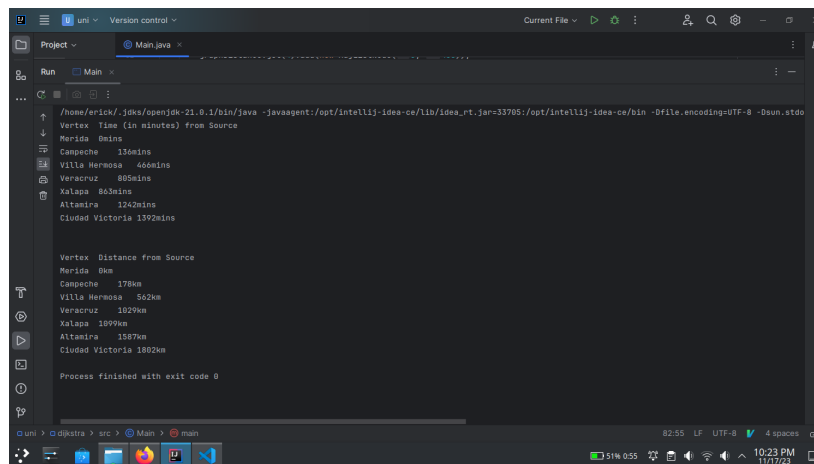


Figura 13: time &amp; distance

Ahora analizaremos los estados con mayor IDH  
CDMX



```

/home/erick/.jdk/openjdk-21.0.1/bin/java -javaagent:/opt/intellij-idea-ce/lib/idea_rt.jar=46487:/opt/intellij-idea-ce/bin -Dfile.encoding=UTF-8 -Dsun.stdout
Vertex Time (in minutes) from Source
CDMX 0mins
Queretaro 210mins
San Luis Potosi 397mins
Ciudad Victoria 565mins

Vertex Distance from Source
CDMX 0km
Queretaro 211km
San Luis Potosi 423km
Ciudad Victoria 758km

Process finished with exit code 0

```

Figura 14: time &amp; distance

Nuevo Leon(Monterrey), este ya es una frontera  
 Baja California(Mexicali), igual, ya es una frontera  
 Aguascalientes

```

/home/erick/.jdk/openjdk-21.0.1/bin/java -javaagent:/opt/intellij-idea-ce/lib/idea_rt.jar=45911:/opt/intellij-idea-ce/bin -Dfile.encoding=UTF-8 -Dsun.stdout
Vertex Time (in minutes) from Source
Aguascalientes 0mins
Zacatecas 92mins
San Luis Potosi 237mins
Ciudad Victoria 495mins

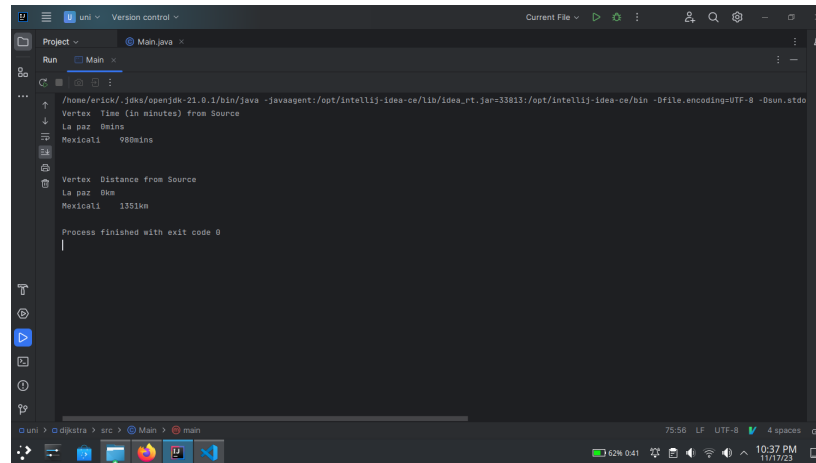
Vertex Distance from Source
Aguascalientes 0km
Zacatecas 116km
San Luis Potosi 312km
Ciudad Victoria 447km

Process finished with exit code 0

```

Figura 15: time &amp; distance

Baja California Sur (La Paz)



```

Project: uni
Main.java
Run
Main
/home/eric/.idea/openjdk-21.0.1/bin/java -javaagent:/opt/intellij-idea-ce/lib/idea_rt.jar=33813:/opt/intellij-idea-ce/bin -Dfile.encoding=UTF-8 -Dsun.stdout
Vertex Time (in minutes) from source
La paz 0mins
Mexicali 980mins
Vertex Distance from Source
La paz 0km
Mexicali 1351km
Process finished with exit code 0

```

Figura 16: time &amp; distance

## análisis

Tomando en cuenta que el promedio de velocidad que también lo calculan los softwares de google maps son muy precisos y como se tomo el tiempo de ahí podemos simplemente concluir que a nuestro sureste del país le queda muy inaccesible la frontera con USA y es simple comparación.

## 4 Conclusiones

Nota: abajo de las conclusiones se encuentran los puntos extras

Yo considero que esto afecta levemente debido a que no estamos considerando el comercio del mar en esta problemática sin embargo plantear una solución a la falta de conexiones no es nada fácil, simplemente añadir que hacer carreteras lo mas estrechas posibles ayuda mucho y podría reducir tiempo y gasolina para cualquier vehículo.

## 5 puntos extras



Figura 17: tren maya

En mi opinion el comercio mejora insignificantlymente debido a dos razones muy simples

- El tren se esta construyendo para turistas: Estoy seguro de que el tren si va a tener "vagones" para transportar muchos productos, etc no obstante su primordial rol es mover al turismo entre estos estados cuyo poder viene del turismo esto si influye muy positivamente al mercado sobre todo a la economía, pero cuando se trata de transporte no vemos mucha diferencia, el tren va a poder ser mas rápido que un coche, pero este es muy limitado y no sabemos aun el producto final de este gran proyecto.
- Las rutas no cambian mucho pese a que con una via de tren yo la puedo poner logrando muchas intersecciones, estas siguen estando situadas en el sureste del país sin llegar tan lejos donde solo en ese caso ya podríamos notar un poco mas de mejora ya que los estados con mas conexiones se encuentran todavía mas al centro del país.