

**Análisis y diseño de Algoritmos**

Proyecto Nº3

Campus Santa Fe

Enrique Lira Martínez A01023351

Profesor: Dr. Víctor Manuel de la Cueva H

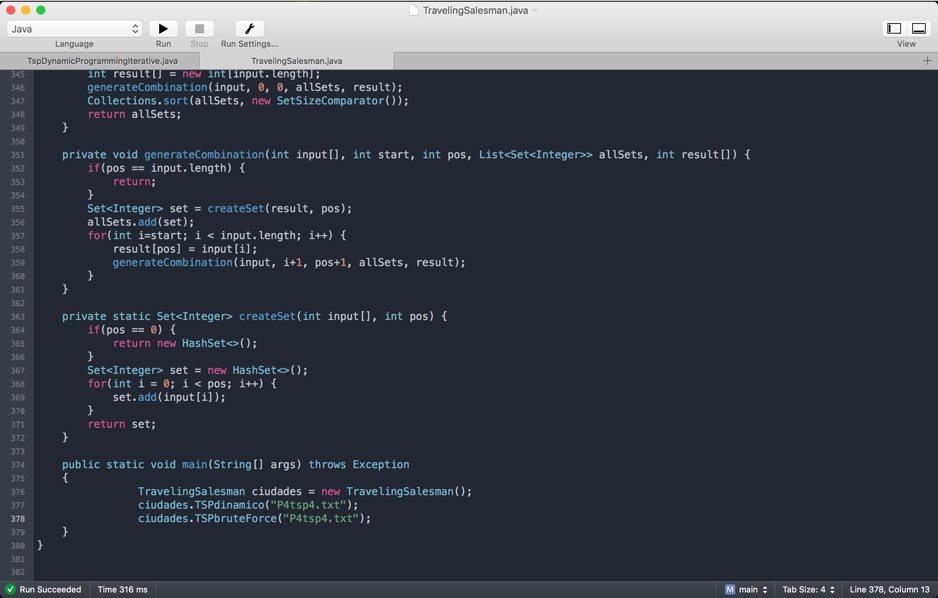
7 de mayo de 2018

**MANUAL DEL USUARIO**

**CONFIGURACIÓN**

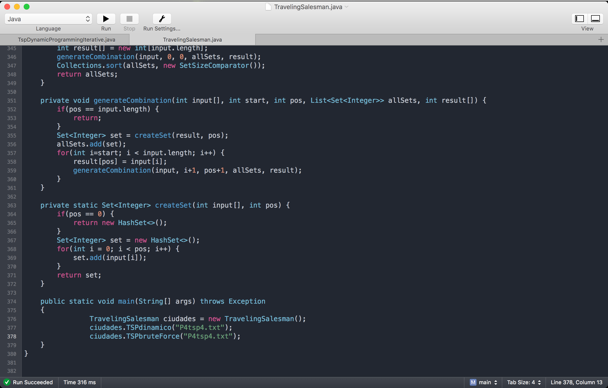
1. El usuario debe contar con un IDE o un editor de texto en caso de que el usuario tenga un IDE.

* Si el usuario tiene un IDE deberá de hacer los siguientes pasos, el usuario deberá solamente abrir el archivo en su IDE y correr el programa con el TravelingSalesman.java descrito a continuación.



* Si el usuario tiene un editor de texto deberá de hacer los siguientes pasos, el usuario deberá solamente su terminal y deberá de buscar el archivo (Ej. cd Desktop/carpeta……) el usuario deberá de saber dónde se ubica el archivo. Primero tendrá que ejecutar el comando javac <nombre del archivo> para crear el ejecutable y continuación tendrá correrlo con el siguiente formato java <nombre del archivo>

(IMPORTANTE para el caso de las 25 ciudacdes uste deberá de correr el código de la suiente manera ”java -Xms12g TravelingSalesman” y usted deberá de tener más o lo equivalente a 16 gb de RAM)



* Después de haber hecho el programa main usted deberá de correr el programa en la terminal con el comando java “nombre del archivo” y dará enter

**ESTRUCTURA GENERAL**

**Estructura del main propuesto**

1. Al crear su función main usted deberá poner un throws Exception ,esto se debe a que si no encuentra un archivo regresara un error ,continuación habrá un ejemplo.

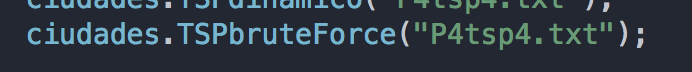


1. Primero debemos definir una clase de la siguiente manera



1. El usuario tendrá que definirá el nombre del archivo lo podrá poner en una variable como se muestra a continuación o directamente en la función TSPdinamico o TSPbruteForce.



****

**ALGORITMO**

Se crearon siguientes la funciones:

* TSPbruteForce ()

- getCost()

-getMin()

-crearCamino()

* TSPdinamico ()

- TSPdinamico()

- SetSizeComparator()

- minCost()

- generateCombination()

-createSet()

cada función tiene un funcionamiento diferente que a continuación se describirá su algoritmo

* TSPbruteForce

Paso 1: Inicio

Paso 2: se inicializa T (el árbol de expansión final) se define como el conjunto vacío

Paso 3: Para cada vértice v de G, haga el conjunto vacío de v

Paso 4: Clasificamos los nodos de G en orden ascendente

Paso 5: Para cada nodo (u, v).

       Si u y v pertenecen a conjuntos diferentes

          Agregue (u, v) a T;

Paso 6: Devuelve T

Paso 7: Fin

* getCost()

Paso 1: Inicio

Paso 2: se inicializa T (el árbol de expansión final) se define como el conjunto vacío

Paso 3: Para cada vértice v de G, haga el conjunto vacío de v

Paso 4: Clasificamos los nodos de G en orden ascendente

Paso 5: Para cada nodo (u, v).

       Si u y v pertenecen a conjuntos diferentes

          Agregue (u, v) a T;

Paso 6: Devuelve T

Paso 7: Fin

* getMin()

Paso 1: Inicio

Paso 2: se inicializa T (el árbol de expansión final) se define como el conjunto vacío

Paso 3: Para cada vértice v de G, haga el conjunto vacío de v

Paso 4: Clasificamos los nodos de G en orden ascendente

Paso 5: Para cada nodo (u, v).

       Si u y v pertenecen a conjuntos diferentes

          Agregue (u, v) a T;

Paso 6: Devuelve T

Paso 7: Fin

* crearCamino()

Paso 1: Inicio

Paso 2: se inicializa T (el árbol de expansión final) se define como el conjunto vacío

Paso 3: Para cada vértice v de G, haga el conjunto vacío de v

Paso 4: Clasificamos los nodos de G en orden ascendente

Paso 5: Para cada nodo (u, v).

       Si u y v pertenecen a conjuntos diferentes

          Agregue (u, v) a T;

Paso 6: Devuelve T

Paso 7: Fin

* Kruskal union-Find()

Paso 1: Inicio

Paso 2: se inicializa T (el árbol de expansión final) se define como el conjunto vacío

Paso 3: Para cada vértice v de G, haga el conjunto vacío de v

Paso 4: Clasificamos los nodos de G en orden ascendente

Paso 5: Para cada nodo (u, v).

       Si u y v pertenecen a conjuntos diferentes

          Agregue (u, v) a T;

Paso 6: Devuelve T

Paso 7: Fin

* Kruskal DFS ()

Paso 1: Inicio

Paso 2: se inicializa T (el árbol de expansión final) se define como el conjunto vacío

Paso 3: Para cada vértice v de G, haga el conjunto vacío de v

Paso 4: Clasificamos los nodos de G en orden ascendente

Paso 5: Para cada nodo (u, v).

Si analizamos el heap y aparece el valor entonces se elimina sino seguimos iterando

Paso 6: Devuelve T

Paso 7: Fin

**Descripción Técnica**

**Función Prim**

**Argumentos:** la función tiene como parámetro un string el cual será el nombre del archivo

**Variables:**

* file = archivo del texto
* Edges
* Heap

**Lógica:**

La función recibe el nombre del archivo y este lo leerá línea por línea, para poder ingresarlo a un nodo para eso se creo la clase edge la cual podremos hacer uso y lo que realiza esta función es la creación del nodo con su inicio ,fin y peso ,acontinuacion pasara al la funcion MST. El algoritmo de Prim es mucho más eficiente ya que se realizon con un heap al estar organizado todo ya no es necesario iterar sobre ella como originalmente lo haría. Ya que no es necesario iterar la lista, se debe crear una variable que vaya checando los diferentes elementos de la lista hasta llegar al siguiente nodo que hará conexión (i).

****

**Función Kruskal**

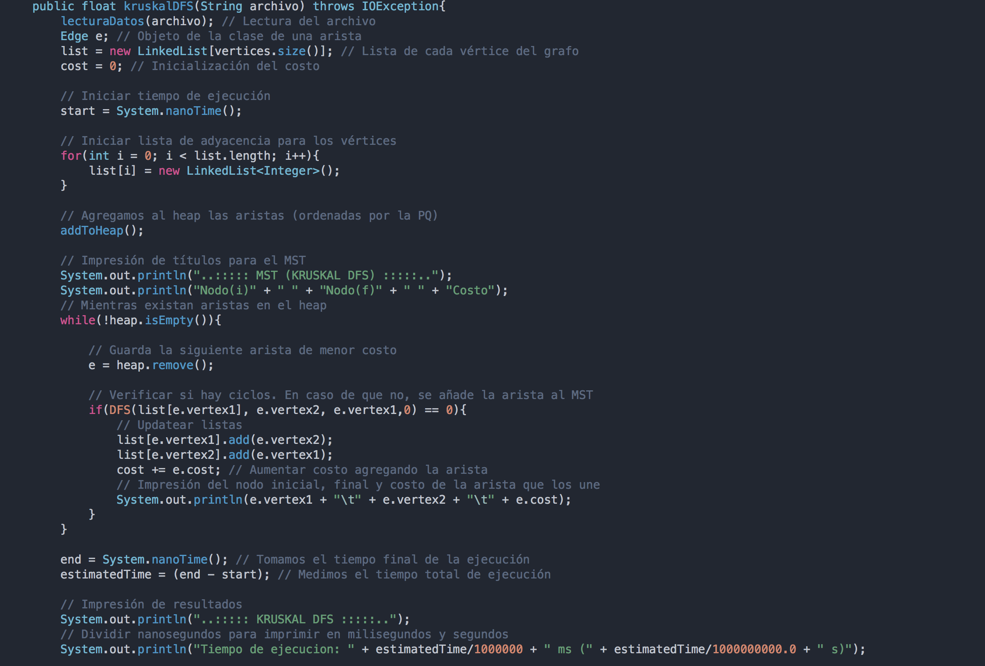
**Argumentos** la función tiene como parámetro un string el cual será el nombre del archivo

**Variables:**

* file = archivo del texto
* Edges
* Heap

**Lógica:**

La función recibe el nombre del archivo y este lo leerá línea por línea, para poder ingresarlo a un nodo para eso se creo la clase edge la cual podremos hacer uso y lo que realiza esta función es la creación del nodo con su inicio ,fin y peso ,acontinuacion pasara al la función MST .Se empezara a itera para saber cuales nodos no realizan ciclos esto se sabe gracias a la función dfs la cual analiza si el heap en el que se guardan los nodos se repiten si se repiten entonces se eliminan y regresa al estado anterior.

****

**Función Kruskal Union-Find**

**Argumentos:** la función tiene como parámetro un string el cual será el nombre del archivo

**Variables:**

* file = archivo del texto
* Edges
* Subsets
* Padre y rango

**Lógica:**

La función recibe el nombre del archivo y este lo leerá línea por línea, para poder ingresarlo a un nodo para eso se creo la clase edge la cual podremos hacer uso y lo que realiza esta función es la creación del nodo con su inicio ,fin y peso ,acontinuacion pasara al la función MST, el algoritmo no genera ciclos a comparación de la implementación directa de Kruskal que utiliza, por ejemplo, depth-fisrt search. Antes de comenzar con el algoritmo como tal se necesita crear los sets del union find asignando cada uno con su mismo padre y rango . Después, se tendrá que relizar un sort para la lista con algún algoritmo de ordenamiento .Cuando se tiene esto, entramos a un ciclo que iterará hasta que se cree el MST. Habrá un contador i, que recorrerá la lista hasta obtener el siguiente nodo que hará conexión. Para ello se guardará en dos variables el destino y origen, que será evaluado en la estructura de union find con un findset. Si el valor de estas variables es diferente significa que no hay ciclos por lo que se sumará el peso, se agrega dicho nodo a la lista de conectados y se borra de la lista original.

****

Resultados

Prim con heap :



Kruskal DFS:

Kruskal unión-find:

