Los siguientes algoritmos se realizaron desde que se empezaron a ver los temas por lo que se conto con el tiempo de investigar diferentes formas de realizarlos es por eso por lo que a continuación adjuntare lo que pude realizar y entender de lo aprendido en clase.

Primero\_el\_mejor\_c++

Este algoritmo lo tenia trabajado desde el primer parcial ya que también tiene un menú para búsqueda una profundidad y anchura, pero este fue una variante que hice aparte para ver un caso de un grafo ponderado ya que del pdf que encontré se encontraba el pseudo código que dejare en las citas bibliográficas.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DECLARACION DE VARIABLES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tenemos nuestra función PrimeroMejor que recibe como parámetro el vértice origen y vértice destino, se definen dos tipos de datos una que guardarán un par de valores el primero será el de string y el segundo el costo(VerticeCosto) y una segunda lista que guardara un par de vértices(VerticeVertice) , una vez esto de declaran dos listas una de Lista Costos de tipo VerticeCosto y la otra lista ListaOrdenada , esta slmacenara los costos pero de manera ordenada de también tipo VerticeCosto , además también se declara una pila de tipo VerticeVertice , aquí va ser como el origen/destino de vértice a vértice .

Ademas tenemos más variables como CostoActual que nos va a decir el costo total del camino recorrido, dos banderas que nos ayudaran y se explicara más delante, dos iteradores para recorrer las aristas, una arista auxiliar para recorrer el grafo y un vértice actual para saber dónde estamos y un destino actual.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamenteUna vez declarado esto se siguió el siguiente pseudo código:

Primero en la lista costos pusimos el vértice origen y en lo asociamos con un costo cero y para esto utilizamos push\_ back para que se insertara al último de la lista

(Esto por que estamos usando pila de tipo (LIFO)).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*HASTA QUE SE ENCUENTRE LA SOLUCION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Luego mientras la lista ordenada no este vacía nuestro vértice actual va a ser igual a nuestro primer vértice de nuestra lista ordenada (front) y first para el valor del vertice, recordemos que el segundo es el costo.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* VERTICE ACTUAL == DESTINO\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Y el costo actual va a ser igual va a ser igual al coso de ese primer vértice de la lista ordenada

(ahora front y second)

Una vez hecho esto eliminamos de la lista ordenada el primer vértice de la lista (pop).

Luego se hace la comparación si ese vértice actual es igual al vértice destino que se ingreso entonces se manda imprimir el costo actual y cambiamos el estado de nuestra bandera (band2 ) a 1 haciendo referencia a que si existe una ruta al vértice deseado, para desplegar esa ruta se siguió el siguiente pseudo código:

Captura de pantalla de un celular con texto

Descripción generada automáticamente

Como ya se encontró que el vértice actual es el destino, ahora nuestro destino actual es nuestro destino y mientras nuestra pila no este vacía se ira imprimiendo nuestro destino actual en el campo nombre y las fechas irán “<-“ya que se están usando pilas (LIFO).

Y mientras y pila no este vacía y la posición de al principio de la pila sea diferente a la de destino actual iremos desapilando los elementos de la pila.

Y si la pila es diferente de vacío entonces el destino actual va a ser la posición inicial de la pila en su primer campo que es el string.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* VERTICE ACTUAL DIFERENTE A DESTINO \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Diagrama

Descripción generada automáticamenteAquí se utiliza aux que es una arista auxiliar para ir viendo los demás vértices del grafo, entonces como el Vértice actual es diferente al de destino aux va a ser igual al vértice actual en su campo adyacente, para que se entienda mejor se tiene el siguiente dibujo:

Aquí si C seria nuestro vértice actual, tendría como vértice destino A y B (Líneas rojas), entonces hay que recorrer las aristas para ver cuales tiene como destino.

Anterior

Actual

Entonces mientras aux sea diferente de NULL (para recorrer todos los vértices destino que tiene el vértice actual), calculamos el costo actual que es el costo actual que se tiene más el peso.

Y aquí tenemos band =0 que significa que el vertice no se encuentra en la lista de costos

Después aquí utilizamos el primer iterador i para recorre la lista de costos y ver si aux en su campo adyacente es igual al elemento que estamos recorriendo en la lista (first -> string )

Y aquí band =1 por que si se encuentra en la lista de costos

Y si el costo actual es menor al costo del elemento de la lista en la que estamos (second)

Actualizamos el valor en la lista con el costo actual y luego en la lista con los costos ordenados también lo actualizamos donde se encuentre el mismo valor en el primer campo (string) y de igual manera con sort y la subfunción comparación ordenamos los valores para que este ordenada y apilamos estos vértices origen/destino y volvemos a calcular su costo actual con

CostoActual= CostoActual- aux->peso ( se explicara mas adelante por que ).

Pero si después de acabar de recorrer la lista nuestra band sigue siendo cero quiere decir que no esta en la lista de costos, haciendo lo siguiente:

Si band es igual a cero, insertamos el vértice en la lista costos con push\_back , asociando el nuevo costo y lo mismo para lista ordenada solo que aquí como es ordenada tenemos que llamar a nuestra subfunción comparación que va a ordenar los valores de menor a mayor (sort) y apilamos en pila el vértice actual y el vértice destino.

Y volvemos a calcular el costo actual de esta forma CostoActual= CostoActual- aux->peso ya que en varias iteraciones se estaría sumando de más el costo, aquí una imagen para que se entienda:

Diagrama

Descripción generada automáticamente

Si fuera un el nodo origen TIJ y el nodo destino TAM entonces llegando a BJX tiene 2 opciones y que deberá tomar es el de 400 pero ya habrá pasado y sumado por el de 900, entonces para quitar ese y únicamente sume el de menor costo hacemos ese cálculo otra vez.

Una vez terminado eso se avanza al siguiente vértice destino del vértice actual

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Si la lista ordenada se vacío sin encontrar el destino, no existe una ruta entre esos dos vértices, es decir band2 es igual a 0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FIN DEL ALGORITMO\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Básicamente este algoritmo tiene dos listas principales una para guardar un par de datos (vértice y costo) esta para guardar los costos y otra del mismo tipo, pero esta guardara los costos de manera ordenada y una pila para guardar dos vértices (origen y destino) cuando se encuentre la de menor costo.

Como se va a trabar con pilas entonces en ambas listas insertamos el primer vértice (origen) con push\_ back (al final de la lista) con un costo de 0 ya que en dado costo que el vértice origen sea igual al de destino, es costo seria este.

Después mientras la lista ordenada no este vacía lo que se va a estar haciendo es primero obtener el vértice actual que va a ser igual al primer elemento de la lista ordenada en su campo first (string) y el costo actual va a ser igual al costo del primer elemento de la lista ordenada en su campo second y una vez obtenido esto desapilamos el frente de la lista.

Si el vértice actual es igual al del destino se despliega el camino que se recorrió , ero si no entonces tenemos que recorrer los vértices adyacentes al actual, entonces nos posicionamos en el primero y mientras no sea nulo vamos a ir calculando el costo actual que va a ser el costo actual que ya teníamos mas el adyacente en su peso, después vamos a ir recorriendo la lista de costos para saber si el vértice donde estamos esta ahí o no, si no esta va a terminar de recorrer la lista y band va a seguir siendo cero entonces tendremos que agregar ese vértice a la lista de costos y a la lista de costos ordenada , luego con sort ordenar la lista de costos otra vez y apilar el vértice origen y destino en pila y calcular el costo actual , y si esta en la lista de costos va a ser igual a uno y si el costo actual es menor al costo del que estamos recorriendo (second), actualizamos el costo, y también lo actualizamos en la lista ordenda por o que usamos otro iterador y si el elemento que estamos recorriendo es igual a auxiliar en su campo adyacente , actualizamos el costo y apilamos la pareja vértice actual, vértice destino y como actualizamos el costo en lista ordenada , volvemos a llamar a sort para acomode los costos y actualizamos otra vez costo actual.

Y si no se encontró una ruta, se imprime un mensaje diciendo lo mencionado.

Ejemplo de ejecución:

Grafo precargado:

Una captura de pantalla de una computadora

Descripción generada automáticamenteUna captura de pantalla de una computadora

Descripción generada automáticamente

Una captura de pantalla de una computadora

Descripción generada automáticamentePrimero el mejor:

Una captura de pantalla de una computadora

Descripción generada automáticamente

Una captura de pantalla de una computadora

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

500

250

450

700

800

300

400

500

650

650

350

400

450

450

900

1200

**MEX**

**CUN**

**GDL**

**MZT**

**BJX**

**TAM**

**MID**

**SAN**

**MTY**

**TIJ**