ED2 - Proy 1 - Pablo Fallas, Leo Picado

Código

```
#include <fstream> // Usado para leer contenido de archivos.
#include <iostream> // Para manejar entrada y salida de texto.
#include <queue> // Usado la cola de prioridad de C++.
#include <stdio.h>  // Para leer/parsear la entrada de ints.
#include <vector> // Usado para la matriz de adyacencia.

#define COLUMNAS 4 // Cantidad de columnas que tendra la
matriz de adyacencia.
#define INF 1<<30 // Definimos un valor grande que represente
la distancia infinita inicial, basta conque sea superior al
maximo valor del distancia en alguna de las aristas
#define MAX 10005 // Maximo numero de puntos en el grafo.
#define MAXTIEMPO 1000.1000 // Maximo tiempo entre dos puntos.

using namespace std;

// Clase Punto
// Adyacente: punto con que conecta la ruta.
// Distancia: 'peso' entre puntos.
// Velocidad: en km/h, velocidad entre dos puntos.
// Tiempo: distancia/velocidad.
class Punto {
 public:
   Punto(int pAdyacente, int pDistancia, int pVelocidad, float
pTiempo) : adyacente(pAdyacente), distancia(pDistancia),
velocidad(pVelocidad), tiempo(pTiempo) {}
```

```
    Punto(int pAdyacente, int pDistancia) :
adyacente(pAdyacente), distancia(pDistancia), velocidad(0),
tiempo(0.0) {}
    Punto(): adyacente(0), distancia(0), velocidad(0),
tiempo(0.0) {}
    int adyacente;
    int distancia;
    int velocidad;
    float tiempo;
};

// La cola de prioridad de C++ obtiene el valor de mayor
valor, tenemos que
// sobrescribir la funcion de comparacion para que tome el
menor valor.
struct compararPuntos {
    bool operator() (const Punto &a, const Punto &b) {
        return a.tiempo > b.tiempo;
    }
};

// Variables globales
bool listaPuntosVisitados[MAX]; // Arreglo usado para puntos
visitados.
float listaTiempos[MAX]; // Arreglo de distancias acumuladas
desde el punto inicial.
int listaPuntosAnteriores[MAX]; // Arreglo usado para la
impresion de caminos cortos.
int totalPuntos; // Total de puntos en el grafo.
int totalRutas; // Total de rutas en el grafo (conexiones
entre puntos).
priority_queue<Punto, vector<Punto>, compararPuntos>
```

```
ColaRutas; // Usado para almacenar los datos durante Dijkstra.
vector<Punto> matrizAdyacencia[MAX]; // Matriz de adyacencia.

// Prototipos para las funciones:
void _init(); // Preparacion para Dijkstra.
void dijkstra(); // Implementacion del algoritmo.
void evaluarTiempo(); // Backtracking de Dijkstra.
void imprimirCamino(); // Imprime la ruta entre dos puntos.
void menu(); // Maneja la interaccion con el usuario.
void cargarPuntos(); // Lectura desde archivo.
void calcularRuta(); // Calcula distancia entre dos puntos.
void actualizarRuta(); // Sobrescribe valores leidos de
archivo.

// Inicializa los arreglos usados por Dijkstra con valores
iniciales.
void _init() {
   for (int i = 0 ; i <= totalPuntos; ++i) {
       listaPuntosAnteriores[i] = -1; // Inicializamos el
punto previo del punto i con -1.
       listaPuntosVisitados[i] = false; // Inicializamos todos
los puntos como no visitados.
       listaTiempos[i] = MAXTIEMPO; // Inicializamos todos los
tiempos con MAXTIEMPO.
   }
}

// Backtracking de Dijkstra: compara si el punto encontrado es
mas rapido que
// el punto anteriormente almacenado.
void evaluarTiempo(int actual, int adyacente, int distancia,
int velocidad, float tiempo) {
```

```
   if (listaTiempos[actual] + tiempo <
listaTiempos[adyacente]) { // Si el tiempo acumulado del
origen al punto actual mas el adyacente es menor al
almacenado.
       listaTiempos[adyacente] = listaTiempos[actual] +
tiempo;  // Reemplazamos el valor almacenado.
       listaPuntosAnteriores[adyacente] = actual; // Y tambien
el punto anterior.
       ColaRutas.push(Punto(adyacente, distancia, velocidad,
listaTiempos[adyacente])); // Finalmente agregamos el punto
adyacente a la cola de prioridad.
   }
}

// Funcion recursiva para la impresion del camino mas corto.
void imprimirCamino(int destino) {
   if (listaPuntosAnteriores[destino] != -1) { // Si todavia
hay un punto anterior...
       imprimirCamino(listaPuntosAnteriores[destino]);  //
llamamos la funcion con el valor actual.
   }
   cout << destino << "->"; // Si no hay puntos anteriores,
imprimimos el destino.
}

void dijkstra(int inicial) {
   int actual,
       adyacente,
       distancia,
       velocidad;
   float tiempo;
```

```
    _init(); //inicializamos nuestros arreglos
    ColaRutas.push(Punto(inicial, 0, 0, 0.0)); // Insertamos el
punto inicial en la Cola de Prioridad.
    listaTiempos[inicial] = 0.0; // Inicializamos la distancia
del inicial como 0

    while (!ColaRutas.empty()) { // Mientras existan elementos
en la cola de rutas.
        actual = ColaRutas.top().adyacente; // Obtenemos la
ruta con menor peso, en un comienzo será el inicial.
        ColaRutas.pop(); // Sacamos el elemento de la cola.
        if (listaPuntosVisitados[actual]) {
            continue; // Si el punto actual ya fue visitado
entonces seguimos con la proxima iteracion.
        }
        listaPuntosVisitados[actual] = true; // Marcamos como
visitado el punto actual.

        for (int i = 0; i < matrizAdyacencia[actual].size();
++i) { // Revisamos los adyacentes del punto actual.
            adyacente = matrizAdyacencia[actual][i].adyacente;
            distancia = matrizAdyacencia[actual][i].distancia;
            velocidad = matrizAdyacencia[actual][i].velocidad;
            tiempo = matrizAdyacencia[actual][i].tiempo;

            if(!listaPuntosVisitados[adyacente]) { // Si el
punto adyacente no ha sido visitado.
                evaluarTiempo(actual, adyacente, distancia,
velocidad, tiempo); // Hacemos el backtracking.
            }
        }
    }
```

```
}

// Cargando puntos desde un archivo.
void cargarPuntos() {
   float tiempo;
   ifstream archivo;
   int idArchivo,
       puntoActual,
       puntos,
       puntoOrigen,
       puntoDestino,
       distancia,
       velocidad,
       i,
       j;

   cout << "Quisiera leer el archivo 1 o el 2?" << endl;
   cin >> idArchivo;

   if (idArchivo == 1) {
       archivo.open("Grafo1.txt");
   }

   if (idArchivo == 2) {
       archivo.open("Grafo2.txt");
   }

   archivo >> totalPuntos;
   cout  << "- Total de Puntos: " << totalPuntos << endl;

   archivo >> totalRutas;
   cout << "- Total de Rutas: " << totalRutas << endl;
```

```
   int matriz[totalRutas][COLUMNAS]; // Matriz de adyacencia
temporal.

   for (i = 0; i < totalRutas; i++) {
       for(j = 0; j < COLUMNAS; j++) {
         archivo >> puntoActual;
         matriz[i][j] = puntoActual;
       }
   }

   archivo.close();

   // Impriendo valores que van a ser ingresados en el grafo.
   cout << "Matriz de Adyacencia." << endl;
   cout << "-----------------------------------------------"
<< endl;
   cout << "| Origen    | Destino   | Distancia | Velocidad |"
<< endl;
   for (i = 0; i < totalRutas; i++) {
     for(j = 0; j < COLUMNAS; j++) {
       cout << "| ";
       cout.width(10);
       cout << matriz[i][j];

       if (j == COLUMNAS-1) {
         cout << "|" << endl;
       }
    }
   }
   cout <<
"-----------------------------------------------\n" << endl;
```

```
   // Ingresando valores de archivo en la Matriz de
Adyacencia.
   for (i = 0; i < totalRutas; i++) {
       puntoOrigen = matriz[i][0];
       puntoDestino = matriz[i][1];
       distancia = matriz[i][2];
       velocidad = matriz[i][3];
       tiempo = (float)(distancia) / (float)(velocidad);

matrizAdyacencia[puntoOrigen].push_back(Punto(puntoDestino,
distancia, velocidad, tiempo)); // Agregamos el punto a la
matriz.
   }
}

void calcularRuta() {
   int inicial,
       destino;

   float tiempo;

   cout << "~~ Impresion de camino mas corto ~~" << endl;
   cout << "Ingrese el punto inicial: " << endl;
   scanf("%d", &inicial);
   dijkstra(inicial);
   cout << "Ingrese punto final: " << endl;
   scanf("%d" , &destino);

   tiempo = (float)listaTiempos[destino];

   if (tiempo != 0) {
```

```
      cout << "Para el punto " << destino << ", el tiempo mas
corto es = " << tiempo << endl;
      cout << "Ruta: ";
      imprimirCamino(destino);
      cout << ".<\n" << endl;
    } else {
      cout << "No hay paso entre " << inicial << " y " <<
destino << ".\n" << endl;
    }
}

void actualizarRuta() {
    int nuevaVelocidad,
        ptOrigen,
        ptDestino,
        totalAdyacentes;

    cout << "Ingrese punto de origen: " << endl;
    scanf("%d", &ptOrigen);

    totalAdyacentes = matrizAdyacencia[ptOrigen].size();

    if (totalAdyacentes != 0) {
      cout << "Ingrese punto de destino: " << endl;
      scanf("%d", &ptDestino);

      cout << "Ingrese la nueva velocidad: " << endl;
      scanf("%d", &nuevaVelocidad);

      for (int i = 0; i < totalAdyacentes; ++i) { // Revisamos
todas las adyacentes desde el origen.
          if(matrizAdyacencia[ptOrigen][i].adyacente ==
```

```
ptDestino) {
                matrizAdyacencia[ptOrigen][i].velocidad =
nuevaVelocidad;
                matrizAdyacencia[ptOrigen][i].tiempo =
(float)matrizAdyacencia[ptOrigen][i].distancia /
(float)nuevaVelocidad;
            }
        }
    } else {
        cout << "No hay rutas que salgan de " << ptOrigen << "\n"
<<endl;
    }
}

void menu () {
 char c;

 do {
    cout << "~~ Optimizador de Rutas AutoAjustable ~~" << endl;
    cout << "Haga una seleccion:" << endl;
    cout << "1. Cargar puntos de ruta de archivo." << endl;
    cout << "2. Calcular ruta mas corta entre dos puntos." <<
endl;
    cout << "3. Actualizar datos de una ruta." << endl;
    cout << "q: Salir." << endl;

    cin >> c;

    switch(c) {
      case '1':
        cargarPuntos();
        break;
```

```
      case '2':
        calcularRuta();
        break;
      case '3':
        actualizarRuta();
        break;
      default:
        break;
    }
  } while(c != 'q' && c != EOF);
}

int main() {
    menu();
    return 1;
}
```

Pruebas

```
$: make proyecto && ./proyecto
g++     proyecto.cpp   -o proyecto
~~ Optimizador de Rutas AutoAjustable ~~
Haga una seleccion:
1. Cargar puntos de ruta de archivo.
2. Calcular ruta mas corta entre dos puntos.
3. Actualizar datos de una ruta.
q: Salir.
1
Quisiera leer el archivo 1 o el 2?
1
```

```
- Total de Puntos: 3
- Total de Rutas: 4
Matriz de Adyacencia.
-------------------------------------------------
| Origen    | Destino   | Distancia | Velocidad |
|          1|          2|        100|         20|
|          1|          3|         20|         10|
|          2|          3|         10|          5|
|          3|          2|         10|         10|
-------------------------------------------------

~~ Optimizador de Rutas AutoAjustable ~~
Haga una seleccion:
1. Cargar puntos de ruta de archivo.
2. Calcular ruta mas corta entre dos puntos.
3. Actualizar datos de una ruta.
q: Salir.
2
~~ Impresion de camino mas corto ~~
Ingrese el punto inicial:
1
Ingrese punto final:
2
Para el punto 2, el tiempo mas corto es = 3
Ruta: 1->3->2->.<

~~ Optimizador de Rutas AutoAjustable ~~
Haga una seleccion:
1. Cargar puntos de ruta de archivo.
```

```
2. Calcular ruta mas corta entre dos puntos.
3. Actualizar datos de una ruta.
q: Salir.
3
Ingrese punto de origen:
1
Ingrese punto de destino:
2
Ingrese la nueva velocidad:
40
~~ Optimizador de Rutas AutoAjustable ~~
Haga una seleccion:
1. Cargar puntos de ruta de archivo.
2. Calcular ruta mas corta entre dos puntos.
3. Actualizar datos de una ruta.
q: Salir.
2
~~ Impresion de camino mas corto ~~
Ingrese el punto inicial:
1
Ingrese punto final:
2
Para el punto 2, el tiempo mas corto es = 2.5
Ruta: 1->2->.<

~~ Optimizador de Rutas AutoAjustable ~~
Haga una seleccion:
1. Cargar puntos de ruta de archivo.
2. Calcular ruta mas corta entre dos puntos.
```

```
3. Actualizar datos de una ruta.
q: Salir.
q

$: make proyecto && ./proyecto
make: `proyecto' is up to date.
~~ Optimizador de Rutas AutoAjustable ~~
Haga una seleccion:
1. Cargar puntos de ruta de archivo.
2. Calcular ruta mas corta entre dos puntos.
3. Actualizar datos de una ruta.
q: Salir.
1
Quisiera leer el archivo 1 o el 2?
2
- Total de Puntos: 5
- Total de Rutas: 10
Matriz de Adyacencia.
-------------------------------------------------
| Origen    | Destino   | Distancia | Velocidad |
|          1|          2|          7|         20|
|          1|          4|          2|         20|
|          2|          4|          2|         20|
|          2|          3|          1|         20|
|          3|          5|          1|         20|
|          4|          2|          3|         20|
|          4|          3|          8|         20|
|          4|          5|          5|         10|
|          5|          3|          4|         20|
```

```
|          5|          1|          6|         20|
-----------------------------------------------

~~ Optimizador de Rutas AutoAjustable ~~
Haga una seleccion:
1. Cargar puntos de ruta de archivo.
2. Calcular ruta mas corta entre dos puntos.
3. Actualizar datos de una ruta.
q: Salir.
2
~~ Impresion de camino mas corto ~~
Ingrese el punto inicial:
1
Ingrese punto final:
5
Para el punto 5, el tiempo mas corto es = 0.35
Ruta: 1->4->2->3->5->.<

~~ Optimizador de Rutas AutoAjustable ~~
Haga una seleccion:
1. Cargar puntos de ruta de archivo.
2. Calcular ruta mas corta entre dos puntos.
3. Actualizar datos de una ruta.
q: Salir.
3
Ingrese punto de origen:
4
Ingrese punto de destino:
5
```

```
Ingrese la nueva velocidad:
100
~~ Optimizador de Rutas AutoAjustable ~~
Haga una seleccion:
1. Cargar puntos de ruta de archivo.
2. Calcular ruta mas corta entre dos puntos.
3. Actualizar datos de una ruta.
q: Salir.
2
~~ Impresion de camino mas corto ~~
Ingrese el punto inicial:
1
Ingrese punto final:
5
Para el punto 5, el tiempo mas corto es = 0.15
Ruta: 1->4->5->.<

~~ Optimizador de Rutas AutoAjustable ~~
Haga una seleccion:
1. Cargar puntos de ruta de archivo.
2. Calcular ruta mas corta entre dos puntos.
3. Actualizar datos de una ruta.
q: Salir.
2
~~ Impresion de camino mas corto ~~
Ingrese el punto inicial:
2
Ingrese punto final:
5
```

```
Para el punto 5, el tiempo mas corto es = 0.1
Ruta: 2->3->5->.<

~~ Optimizador de Rutas AutoAjustable ~~
Haga una seleccion:
1. Cargar puntos de ruta de archivo.
2. Calcular ruta mas corta entre dos puntos.
3. Actualizar datos de una ruta.
q: Salir.
3
Ingrese punto de origen:
3
Ingrese punto de destino:
5
Ingrese la nueva velocidad:
1
~~ Optimizador de Rutas AutoAjustable ~~
Haga una seleccion:
1. Cargar puntos de ruta de archivo.
2. Calcular ruta mas corta entre dos puntos.
3. Actualizar datos de una ruta.
q: Salir.
2
~~ Impresion de camino mas corto ~~
Ingrese el punto inicial:
2
Ingrese punto final:
5
Para el punto 5, el tiempo mas corto es = 0.15
```

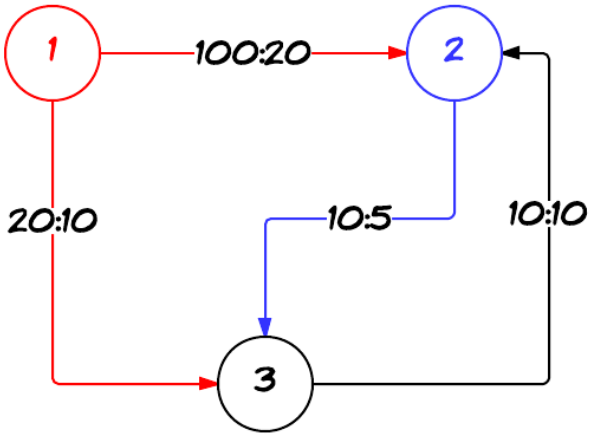
```
Ruta: 2->4->5->.<

~~ Optimizador de Rutas AutoAjustable ~~
Haga una seleccion:
1. Cargar puntos de ruta de archivo.
2. Calcular ruta mas corta entre dos puntos.
3. Actualizar datos de una ruta.
q: Salir.
q
```

Grafos utilizados

GRAFO1.TXT

GRAFO *



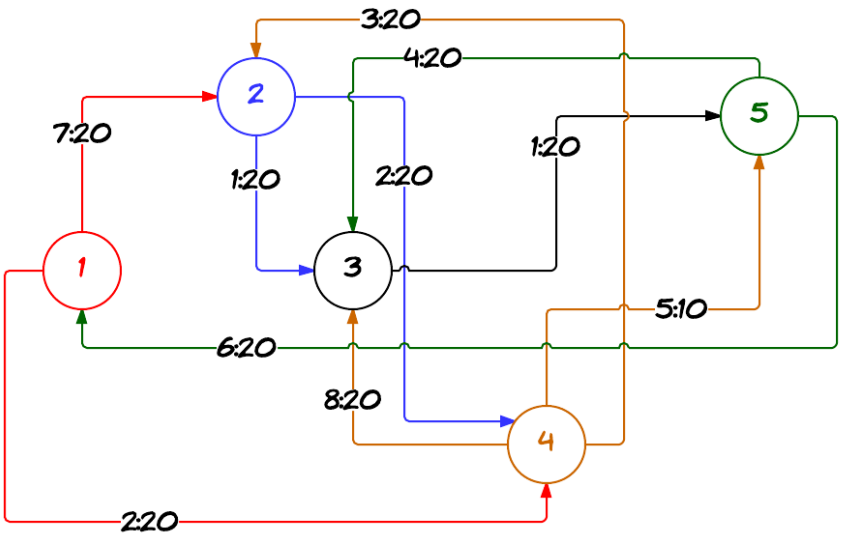
CONTENIDO DEL ARCHIVO

```
3
4
1       2       100     20
1       3       20      10
2       3       10      5
3       2       10      10
```

* NOTACION DISTANCIA:VELOCIDAD

GRAFO2.TXT

GRAFO *



CONTENIDO DEL ARCHIVO

```
5
10
1       2       7       20
1       4       2       20
2       4       2       20
2       3       1       20
3       5       1       20
4       2       3       20
4       3       8       20
4       5       5       10
5       3       4       20
5       1       6       20
```

* NOTACION DISTANCIA:VELOCIDAD