Universidad de Buenos Aires
Facultad De Ingeniería
Año 2024 - 1er Cuatrimestre

CB100 - ALGORITMOS Y ESTRUCTURA DE DATOS

TRABAJO PRÁCTICO 2: INDEXANDO LA CIUDAD

MANUAL DEL USUARIO

ALUMNOS:

- USHIÑA PAILLACHO BRYAN JHOEL
- CANTIGO MARTIN RENZO
- ROSELLO EZEQUIEL
- PROVOST VALENTIN
- TORRE MAXIMILIANO

Índice

1. Objetivo del programa

El programa genera un índice sobre la información oficial de las paradas de colectivos de CABA obtenida desde la web oficial del gobierno de la ciudad, y permite al usuario realizar consultas en base a un barrio, línea de colectivo y coordenadas.

Las consultas disponibles para realizar son las siguientes:

- Cantidad de paradas por barrio
- Parada más cercana según coordenadas
- Paradas de una línea de colectivo
- Cantidad de paradas por línea de colectivo
- Paradas ordenadas por distancia según barrio, línea de colectivo y coordenadas de origen

2. Utilización del programa

2.1. Menú prinicipal

El menú principal del programa que permite al usuario elegir la opción deseada es el siguiente:

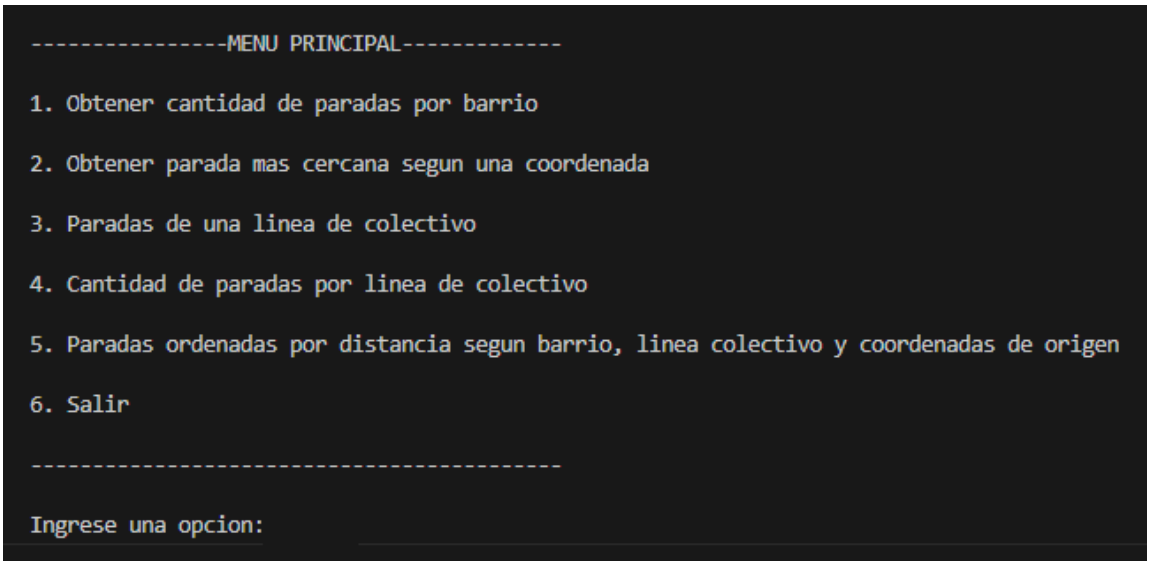

Figura 1: Menú principal.

Para seleccionar una opción se debe ingresar mediante el teclado el número correspondiente a la opción deseada y presionar Enter. En caso de ingresar una opción que no sea válida el programa le avisará al usuario y repetirá el menú principal para que el usuario ingrese una opción válida.

```
Ingrese una opcion: 7
Ingreso incorrecto, por favor ingrese una opcion valida

----------------MENU PRINCIPAL-------------

1. Obtener cantidad de paradas por barrio

2. Obtener parada mas cercana segun una coordenada

3. Paradas de una linea de colectivo

4. Cantidad de paradas por linea de colectivo

5. Paradas ordenadas por distancia segun barrio, linea colectivo y coordenadas de origen

6. Salir

----------------------------------------

Ingrese una opcion:
```

Figura 2: Selección de una opción incorrecta en el menú principal.

2.2. Cantidad de paradas por barrio

Al seleccionar la opción 1 en el menú principal se le pedirá al usuario que ingrese el barrio e informará la cantidad de paradas para ese barrio:

```
Ingrese una opcion: 1
~~~     Has elejido la opcion 1         ~~~

    Introduce el barrio que quieras buscar :: Belgrano


                BELGRANO tiene 220 parada(s)
```

Figura 3: Opción 1 - Cantidad de paradas por barrio.

2.3. Parada más cercana según coordenadas

Al seleccionar la opción 2 en el menú principal se le pedirá al usuario que ingrese las coordenadas e informará la parada más cercana:

```
Ingrese una opcion: 2
Ingrese coordenada x: -58.1234
Ingrese coordenada y: -34.1234
La parada mas cercana es: 4233 OBLIGADO RAFAEL, AV.COSTANERA
```

Figura 4: Opción 1 - Parada más cercana según coordenadas.

2.4. Paradas de una línea de colectivo

Al seleccionar la opción 3 en el menú principal se le pedirá al usuario que ingrese la línea de colectivo e informará las paradas de esa línea:

```
Ingrese una opcion: 3
Ingrese una linea de colectivo: 152
Las paradas de la linea son las siguientes:

- Calle: PASEO COLON AV.
- Direccion: 1150 PASEO COLON AV.
- Coordenada X: -58.3682
- Coordenada Y: -34.6211
- Cantidad de lineas: 4
- Lineas: 64 129 152 126


- Calle: PASEO COLON AV.
- Direccion: 1121 PASEO COLON AV.
- Coordenada X: -58.3685
- Coordenada Y: -34.6209
- Cantidad de lineas: 2
- Lineas: 61 152
```

Figura 5: Opción 3 - Paradas de una línea de colectivo.

2.5. Cantidad de paradas por línea de colectivo

Al seleccionar la opción 4 en el menú principal se imprimirá una lista con la cantidad de paradas por línea de colectivo:

```
Ingrese una opcion: 4
Linea: 22 | Cantidad de paradas: 49
Linea: 53 | Cantidad de paradas: 136
Linea: 29 | Cantidad de paradas: 140
Linea: 24 | Cantidad de paradas: 109
Linea: 46 | Cantidad de paradas: 150
Linea: 39 | Cantidad de paradas: 115
Linea: 2 | Cantidad de paradas: 94
Linea: 126 | Cantidad de paradas: 117
Linea: 130 | Cantidad de paradas: 128
Linea: 159 | Cantidad de paradas: 26
Linea: 195 | Cantidad de paradas: 21
Linea: 129 | Cantidad de paradas: 87
Linea: 143 | Cantidad de paradas: 108
Linea: 4 | Cantidad de paradas: 134
Linea: 61 | Cantidad de paradas: 35
Linea: 62 | Cantidad de paradas: 40
Linea: 111 | Cantidad de paradas: 114
Linea: 103 | Cantidad de paradas: 92
Linea: 168 | Cantidad de paradas: 132
Linea: 33 | Cantidad de paradas: 88
Linea: 64 | Cantidad de paradas: 88
```

Figura 6: Opción 4 - Cantidad de paradas por línea de colectivo.

2.6. Paradas ordenadas por distancia según barrio, línea de colectivo y coordenadas de origen

Al seleccionar la opción 5 en el menú principal se le pedirá al usuario que ingrese un barrio, una línea de colectivo y las coordenadas de origen e informará las paradas de esa línea ordenadas por distancia a las coordenadas de origen:

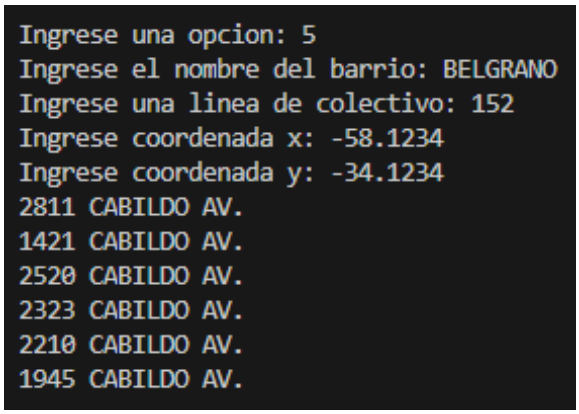

Figura 7: Opción 5 - Paradas ordenadas por distancia según barrio, línea de colectivo y coordenadas de origen.

2.7. Salir del programa

Para salir del programa se debe ingresar la opción 6 en el menú principal:

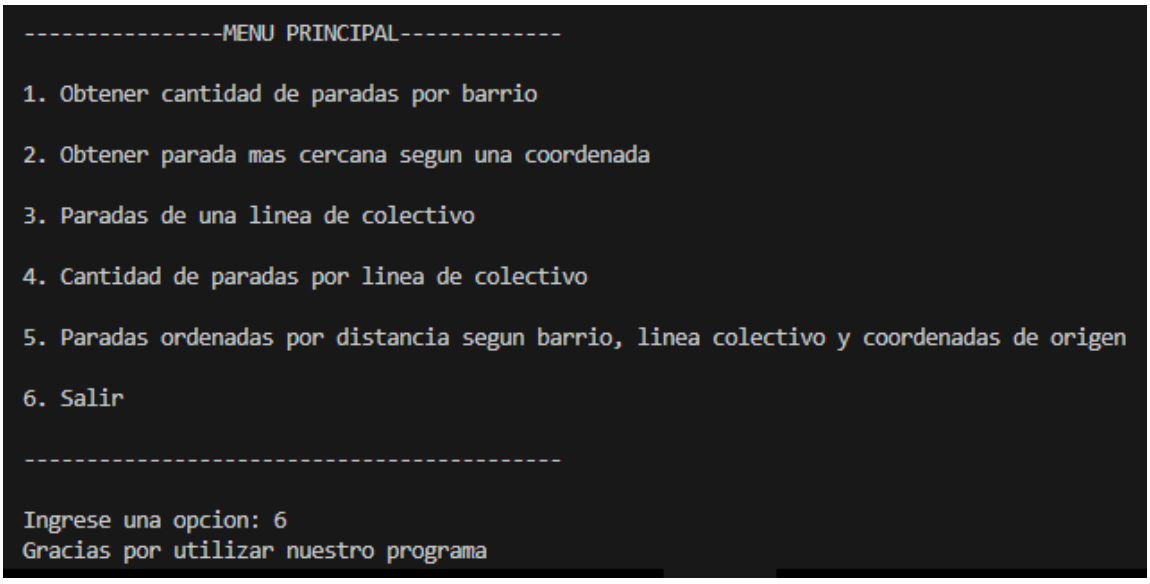

Figura 8: Opción 6 - Salir del programa.

Universidad de Buenos Aires
Facultad De Ingeniería
Año 2024 - 1er Cuatrimestre

CB100 - ALGORITMOS Y ESTRUCTURA DE DATOS

TRABAJO PRÁCTICO 2: INDEXANDO LA CIUDAD

MANUAL DEL PROGRAMADOR

ALUMNOS:

- USHIÑA PAILLACHO BRYAN JHOEL
- CANTIGO MARTIN RENZO
- ROSELLO EZEQUIEL
- PROVOST VALENTIN
- TORRE MAXIMILIANO

3. Estructura del programa

La estructura del programa se basa en una lista de lista, la lista principal contiene los barrios de CABA el cual son un TDA, dentro de cada TDA barrio este contiene una lista donde se guardan las paradas que tiene cada barrio. al igual que barrio, cada parada es TDA donde se guardan los siguientes datos: dirrecion, coordenadas (X,Y), lineas de colectivo, calle.

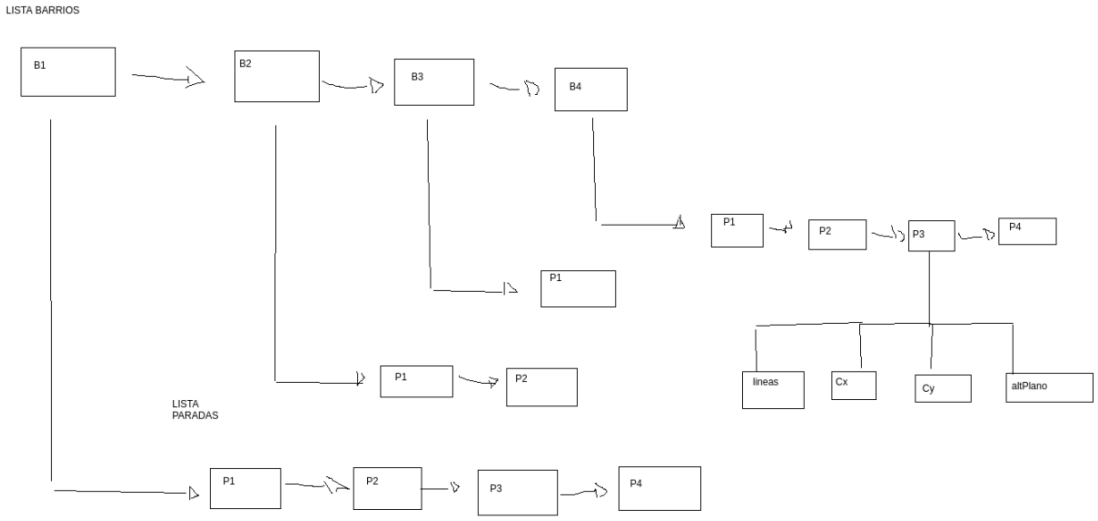

Figura 9: idea de la estructura

4. Archivos del programa y sus métodos

4.1. Inicializar datos.cpp

la funcionalidad de este cpp es obtener los datos del archivo csv para ser utilizados en nuestro programa. Para cumplir esto se utilizaron librerias como fstream, cstdlib y sstream. ademas utilizamos para la lectura y obtencion de datos la funcion getline.

obtener datos

ciertos datos del archivo csv vienen con comas o con comillas por lo que se dificulta la obtencion de los mismos. para solucionar esto utilizamos ciertas funciones de la libreria string para detectar y eliminar lo que no queremos

Algunas imagenes de como hacrlo:

Algunos datos como la calle, la altura del plano y la direccion contienen comas en el medio de los mismos, el problema esta en que el getline lee hasta cuando hay una coma, por lo que si hay una en el medio del dato guarda esa parte y continua leyendo. Esto hace que los datos queden sobrepasados y no se guarden en donde se deberia guardar. para solucionar esto hicimos lo siguiente, si se detecta una comilla que se elimine la coma si es que hay en el dato, haciendo esto se resuelve el problema.

```
getline(ss, direccion, ',');
if (direccion.size() > 0 && direccion[0] == '"') {
    direccion.erase(0, 1);
    if (direccion[direccion.size() - 1] == '"') {
        direccion.erase(direccion.size() - 1);
    } else {
        string resto;
        getline(ss, resto, '"');
        direccion += "," + resto;
        getline(ss, resto, ',');
    }
}
```

para obtener las lineas hay que eliminar las comillas que hacen que se dificulte su lectura para ser pasada de string a numero.

```
getline(ss, l1, ',');
if (l1.size() > 0 && l1[0] == '"') {
    l1.erase(0, 1);
    if (l1[l1.size() - 1] == '"') {
        l1.erase(l1.size() - 1);
    } else {
        string resto;
        getline(ss, resto, '"');
        l1 += "," + resto;
        getline(ss, resto, ',');
    }
}
```

los datos numericos los leamos como string para luego pasarlos a numero. a continuacion se ve como pasarlo de string a numero, en este caso lo datos que necesitan esto son las coordenadas X e Y y las lineas de colectivo.

```
double coordX = atof(coordenadaX.c_str());  //std::string *
double coordY = atof(coordenadaY.c_str());

int lineasAux[6] = {0};
unsigned int cantidadDeLineas = 0;

if (!l1.empty()) lineasAux[cantidadDeLineas++] = atoi(l1.c_str());  //string *
if (!l2.empty()) lineasAux[cantidadDeLineas++] = atoi(l2.c_str());
if (!l3.empty()) lineasAux[cantidadDeLineas++] = atoi(l3.c_str());
if (!l4.empty()) lineasAux[cantidadDeLineas++] = atoi(l4.c_str());
if (!l5.empty()) lineasAux[cantidadDeLineas++] = atoi(l5.c_str());
if (!l6.empty()) lineasAux[cantidadDeLineas++] = atoi(l6.c_str());
```

4.2. barrio.cpp

Métodos

Barrio()

Pre: El nombre del barrio no puede estar vacío.

Post: Crea un barrio con el nombre proporcionado.

Barrio()

Pre: -

Post: Libera la memoria utilizada por el barrio (incluyendo la lista de paradas).

std::string getNombre

Pre: -

Post: Devuelve el nombre del barrio

getParadas

Pre: -

Post: Devuelve un puntero a la lista de paradas del barrio.

4.3. parada.cpp

Métodos

Parada()

Pre: La cadena de la calle y la dirección no deben estar vacías.

Post: Crea una parada con los datos proporcionados.

Parada()

Pre: -

Post: Libera la memoria utilizada por la parada.

std::string getCalle

Pre: -

Post: Devuelve la calle de la parada.

std::string getDireccion

Pre: -

Post: Devuelve la dirección de la parada.

double getCoordenadaX

Pre: -

Post: Devuelve la coordenada X de la parada.

double getCoordenadaY

Pre: -

Post: Devuelve la coordenada Y de la parada.

unsigned int getCantidadDeLineas

Pre: -

Post: Devuelve la cantidad de líneas de la parada.

int *getLineas

Pre: -

Post: Devuelve un puntero a las líneas de la parada.

4.4. menu.cpp

Métodos

void imprimirOpciones

Pre: -

Post: imprime las opciones disponibles.

bool validarOpcionRango

Pre: la opcion ingresada tiene que estar en el rango de opciones

Post: devuelve true si la opcion ingresada es valida, sino false.

bool validarOpcion

Pre: la opcion ingresada tiene que ser un string de un solo caracter

Post: devuelve true si la opcion ingresada es valida, sino false.

4.5. linea.cpp

Métodos

Linea()

Pre: -

Post: crea una linea con el numero de la linea y la cantidad de paradas en 0.

virtual Linea()

Pre: -

Post: -

int getNumeroLinea

Pre: -

Post: devuelve el numero de la linea.

int getCantidadParadas

Pre: -

Post: devuelve la cantidad de paradas de la linea.

void sumarCantidad

Pre: -

Post: suma 1 a la cantidad de paradas de la linea.

void agregarNumeroLinea

Pre: -

Post: agrega el numero de la linea.

4.6. punto$_1$.*cpp*

Métodos

bool verificarIngresoValido

Pre: -

Post: Devuelve true si el texto ingresado es válido para búsqueda, false en caso contrario.

void transformarMayusculas

Pre: -

Post: Transforma todos los caracteres de la cadena a mayúsculas.

Barrio* buscarBarrio

Pre: -

Post: Devuelve el barrio buscado en la lista de barrios.

Barrio* mostrarPorBarrio

Pre: -

Post: Devuelve el barrio buscado.

4.7. punto$_2$.*cpp*

Métodos

std::string obtenerParadaMasCercana

Pre: Las coordenadas ingresadas deben estar dentro del rango solicitado y ser números.

Post: Devuelve la parada más cercana a las coordenadas ingresadas.

4.8. punto$_3$.*cpp*

Métodos

void obtenerParadasDeLinea

Pre: La línea ingresada debe ser válida y pertenecer a las líneas de colectivo de la ciudad.

Post: Imprime un listado de las paradas de la línea de colectivo ingresada.

void imprimirParadasDeLinea

Pre: -

Post: Imprime un listado de las paradas de la línea de colectivo ingresada.

4.9. punto$_4$.*cpp*

Métodos

void cantidadDeParadasPorLineaDeColectivo

Pre: Recibe una lista de barrios no vacía.

Post: Imprime por pantalla el número de la línea de colectivo seguido por la cantidad de paradas que realiza.

void imprimirParadasPorLinea

Pre: La lista línea no puede estar vacía.

Post: Imprime por pantalla el número de la línea y la cantidad de paradas que realiza.

bool estaEnListaLineas

Pre: La línea tiene que estar en la lista de líneas.

Post: Devuelve true si la línea está en la lista, false en caso contrario.

void recorrerParada

Pre: -

Post: Recorre la parada y agrega las líneas de colectivo a la lista de líneas.

void crearListaLineas

Pre: -

Post: Crea e inicializa una lista de líneas de colectivo.

4.10. punto5.cpp

Métodos

void paradasLineaBarrioOrdenadas

Pre: El nombre del barrio debe ser válido y existir en la lista de barrios, las coordenadas del usuario deben ser válidas, la línea de colectivo debe ser válida.

Post: Imprime por pantalla las paradas de la línea de colectivo ingresada, ordenadas por cercanía al usuario.

4.11. main.cpp

El main contiene la interaccion con el usuario, es donde se recopilan todos los archivos para ser utilizados en nuestro programa. Contiene una estructura de swtich case con el fin de que solo funcione el programa con estos casos que interpretarian las opciones del menu

Cada case cumple con una de las opciones dadas del menu. El programa no termina hasta que lo diga el usuario, para salir del mismo tiene que seleccionar la ultima opcion (punto 6) el cual terminara el ciclo while y finalizara el programa.

5. Liberación de memoria

Para la liberacion de memoria del programa en cada new hecho hay un delete, al ser una estructura de listas de listas no basta con eliminar la lista principal sino que hay que eliminar lo que contiene.

liberacion de memoria en los archivos

main.cpp

En el main es donde se crea la lista principal donde se va a guardar toda la informacion del archivo csv. al finalizar el programa recorremos la lista principal y eliminamos cada .ºbjeto/TDA", una vez que se elimino todo, procedemos a eliminar la lista.

```
    }
    for (int i = 0; i < barrios->getTamanio(); ++i)
    {
    delete barrios->obtener(i);
    }
    delete barrios;

    return 0;
```

barrio.cpp

Al ser los barrios la lista principal que va a contener la lista de paradas, para liberar la memoria de paradas se opto por recorrer cada parada e ir elimnando cada TDA de parada contenida en esa lista, luego cuando la lista parada quede completamente vacia, la elimina.

```
Barrio::~Barrio()
{
  for (int i = 0; i < this->paradas->getTamanio(); ++i) {
    delete this->paradas->obtener(i);
  }
  delete this->paradas;
};
```

parada.cpp

eliminamos de parada las lineas de colectivo, al ser un array que puede llevar a ocupar mucho espacio se opta por tener el array en el heap, luego para eliminar este array del heap optamos por hacer lo siguiente.

```cpp
Parada::~Parada()
{
  delete [] this->lineas;
}
```

punto$_3$*.cpp*

En este punto se creo una lista resultado"para guardar la informacion solicitada, luego al finalizar su funcion se elimino la lista.

```cpp
delete resultado;
```

punto$_4$*.cpp*

En este punto se creo una lista "lineas"donde se guarda una linea de colectivo que pasa a ser un tda en este caso. luego para eliminar la lista la recorremos y eliminamos cada TDA linea de la misma. al finalizar el recorrido, cuando este vacia, eliminamos la lista lineas.

```cpp
if(!lineas->estaVacia()){

    lineas->iniciarCursor();
    while(lineas->avanzarCursor()){

        if(lineas->obtenerCursor() != NULL){

            delete lineas->obtenerCursor();
        }
    }

    delete lineas;
```

punto$_5$*.cpp*

En este punto creamos una lista nueva para guardar la informacion que necesitamos, al terminar el programa eliminamos la lista contenedora de la informacion.