TEC, Sede Cartago, Escuela de Computación, IC – 2001 Estructura de Datos

Proyecto 1 – Análisis de textos, ordenamientos y búsqueda.

Max Richard Lee Chung, 2019185076

21/7/2020, I Semestre 2020, Prof. Esteban Arias M.

Abstract: In this document, we're going to explain how does the program "proyecto.cpp" was made and comment all the purposes of each function with the idea of simulating a dictionary. Also, we're going to analyze how much time does an algorithm would take in order to run and compare it between the best-case with the worst-case if applies.

Tabla de contenido

Introducción

Dentro de este documento, se explicará detalladamente la creación de un programa que tiene como propósito analizar la cantidad de palabras, en donde se van a mostrar por medio de una lista ordenada alfabéticamente; cantidad de caracteres ingresados por un usuario y crear un gráfico que muestra la cantidad de letras de un texto.

Además, se van a definir algunos conceptos que se pusieron en práctica para trabajar de forma eficiente los datos.

Marco teórico

- Estructura de datos

Según Sena, M. (2019), se define una estructura de datos como una colección de valores, relaciones que existen entre los valores almacenados y operaciones que podemos hacer sobre ellos. Otra forma de definirlo es la manera en que los datos son organizados y cómo administrarlos, el cual contiene las descripciones de los formatos en que queremos almacenar, acceder y modificar los valores.

Las estructuras de datos pueden variar dependiendo de los problemas que se presentan, ya que existen diferentes formas de estructuras que pueden facilitar más el trabajo en comparación de otras. El tamaño de las estructuras puede variar dependiendo de las variables que queramos almacenar y usar. Además, se puede establecer un orden cualquiera a la hora de insertar los datos a la estructura definida. (Sena, M.,2019)

Uno de los usos de las estructuras son las estructuras enlazadas, las cuales utilizan punteros para indicar la dirección en donde se almacenan los valores. (Sena, M., 2019)

- Cola

Según EcuRed (s.f.), la cola es una estructura de datos, la cual se caracteriza por añadir un elemento al final de la “lista enlazada” y elimina o muestra los elementos al inicio de la “lista enlazada”. Se le llama “FIFO” por sus siglas en inglés, el cual significa “First in, first out” que describe la anterior definición.

Las operaciones básicas para manipular los datos de la lista enlazada cola son: encolar, el cual añade un nuevo elemento; desencolar, el cual elimina un elemento de la lista; crear, el cual inicia una lista vacía y frente, para verificar el primer elemento.

- Tabla de hash

Según Sama, S., una tabla de hash es un contenedor asociativo (el cual tiene la misma funcionalidad que un diccionario) que permite un almacenamiento y recuperación de elementos por medio de llaves o claves que identifiquen al elemento. Generalmente, las entradas de una tabla hash es el dato por guardar y su llave única.

Según EcuRed (s.f.), las operaciones básicas de las tablas hash son: inserción, eliminación y búsqueda de elementos.

- Archivos encabezados

Los archivos encabezados son archivos que usan la extensión ".h", los cuales sirven para declarar funciones para luego importarlos a un archivo C++. Dentro del archivo C++, es necesario incluirlo con el nombre del archivo en comillas para que el compilador reconozca, valide las funciones y usarlos. (Microsft Docs, 2019).

- Templates

Según Cplusplus (2020), los templates o modelos son funciones que operan con tipos de datos genéricos, el cual nos permiten hacer operaciones que se puedan adaptar a más de un tipo de dato o clase sin tener que repetir el tipo. Además, es posible crear modelos de clases con el fin de que los miembros tengan los mismos tipos de datos genéricos.

Parte A

Dentro de este enunciado, se explicará la funcionalidad del algoritmo para extraer los datos, con el fin de generar un "gráfico" para analizar con mayor facilidad los caracteres ingresados junto con su respectiva identificación de la tabla ASCII. Para simular el dicho "gráfico", se hace uso de un histograma textual, usando "X" para marcar cada aparición de dicho carácter. Además, se le otorga la posibilidad al usuario si se debe considerar las mayúsculas o minúsculas.

Los archivos encabezados a usar para realizar dicha funcionalidad son: cola.h y nodoColita.h. El archivo principal llama una función para los textos sensibles a las mayúsculas denominado mayusculas() y para los texto no sensibles a las mayúsculas, se denomina minusculas(). El archivo cola.h es una clase que simula una estructura de tipo abstracto de cola, el cual usa un archivo nodoColita.h para almacenar los datos. Tiene como funciones públicas: isEmpty(), el cual verifica si la cola se encuentra vacía o

no; queue(), el cual añade o inserta un nuevo nodo; dequeue(), el cual elimina un dato de la cola; y por último, revisar() y showStack(), se explicarán a continuación.

La función sensible, utiliza un “for” que va leyendo todos los caracteres para almacenarlos en una cola, la cual esta definida dentro del archivo cola.h, pero dentro del “for” revisa si el carácter ya existe para sumarle 1 al contador que tiene como atributo dicho carácter con la función “revisar()”. Además, se hace uso de una condicional para insertar el nuevo dato si no existe en la lista. Luego, se llama a la función “showStack”, el cual muestra el carácter y hace una iteración para imprimir la cantidad de “X” como apariciones.

```
void mayusculas(string texto){
        Cola<char> colita;
        for (int i=0; i<texto.length()-1; i++){
                bool resultado = colita.revisar(texto[i]);
                if (!resultado){
                        colita.queue(texto[i]);
                }
        }
        colita.showStack();
}
```

La función no sensible, minuscula(), tiene casi el mismo algoritmo mencionado anteriormente, sin embargo, antes de empezar la iteración para añadir los datos, se hace un “for” para convertir todos los caracteres en minúscula.

Parte B

Dentro de este enunciado, se explicará la funcionalidad del algoritmo para crear un diccionario o “Token”, el cual va a mostrar todas las palabras únicas ingresadas de forma alfabética. Al usuario se le otorga la opción de mostrar todas las posiciones iniciales en donde se encontró dentro del texto o no. Además, se le da la posibilidad de escoger, al inicio del programa, si el diccionario debe ser sensible a las mayúsculas o no. Por último, la persona puede añadir nuevos símbolos que funcionan como “delimitadores”, los cuales sirven para separar palabras, al igual que los espacios en blanco, salto de página, un punto de oración y tabulaciones.

Los archivos encabezados utilizados para desarrollar dichas funcionalidades son: tablaHash.h, nodoHash.h, nodo.h y nodoNum.h. Dentro del archivo tablaHash.h, se simula la estructura básica de una tabla hash, el cual usa nodoHash.h para controlar o

almacenar los caracteres, con su respectiva lista de apariciones. Así mismo, el archivo nodoHash.h utiliza el archivo nodo.h, el cual almacena la palabra y crea la lista para todas las apariciones de cada una. Por último, el archivo nodo.h, utiliza el archivo nodoNum.h simular la lista de posiciones.

Dentro del archivo tablaHash.h, se encuentras las funciones públicas del constructor, insert(string,int), mostrarDatos(), mostrarDatosPos() y buscar(string).

La función “insertar”, inicia con una condicional “si la primera letra del string ingresado es un número”, entra a una iteración para buscar dónde debe de insertar los datos. Si la primera letra de la palabra no es un número, busca si es una letra mayúscula o minúscula, sin embargo, al buscar las letras, se divide la mitad todo el alfabeto para obtener un menor tiempo de ejecución.

La función “mostrarDatos” va leyendo cada nodo entre 0-9, a-z y A-Z para poder mostrar en pantalla los datos que el usuario ingresó. La función hace 3 iteraciones para cada conjunto mencionado anteriormente, si los punteros de la lista de posiciones se encuentran en vacío, ignora el elemento a mostrar, si no se encuentra en vacío, entra a los nodos para poder mostrar sólo las palabras ordenadas alfabéticamente. Sin embargo, la función “mostrarDatosPos” muestra todas las posiciones en la que se encontró.

Por último, la función “buscar”, tiene la misma estructura básica que la función “insertar”, sin embargo, el algoritmo interno va analizando si la palabra a buscar se encuentra dentro de la tabla hash. Si encuentra la palabra, retorna un string que contiene la cantidad de veces que el usuario ingresó la misma palabra y las posiciones de estas.

Dentro de los archivos de nodos, sólo contienen las estructuras básicas de una clase con el único fin de almacenar los datos deseados.

Dentro del archivo principal, las funciones de la tabla hash, la utiliza una función llama “tokenizador”, la cual recibe por parámetros una tabla hash, una lista de delimitadores, el texto y un booleano, el cual define si es sensible o no a mayúsculas. El algoritmo del tokenizador inicia con una iteración que va leyendo cada carácter del texto ingresado; si no se encuentra con algún delimitador predeterminado o un delimitador personalizado, el carácter se almacena en otro string para guardarlo dentro de la tabla hash con su

respetiva posición y si encuentra alguno, reinicia el string para crear otra palabra. Si no sensible a mayúsculas, la función convierte todo el texto a minúsculas.

Parte C

Dentro de este enunciado, se explicará las funciones que se usaron para crear las estadísticas básicas sobre la cantidad de caracteres, la cantidad de palabras ingresadas y un tiempo estimado de ejecución del programa.

Las cantidades se lograron adquirir y mostrar usando la función ya mencionada anteriormente, el “tokenizador”, ya que contiene el registro de la cantidad de caracteres al leerlos y también, contiene el registro de las palabras al insertarlas a la tabla de hash.

A la hora de pedir al usuario si desea que el programa sea sensible, el mejor caso de ejecución es de O(1), si ingresa el dato correctamente y el peor caso sería O(n), si el usuario no ingresa correctamente los datos. Además, el menú de selección tiene tiempo de ejecución de O(n) junto con los tiempos de cada una de las diferentes funcionalidades.

Al leer un archivo de texto, tiene un tiempo estimado de O(n) para el peor de los casos y el mejor de los casos, el tiempo estimado sería de O(1), por si el archivo sólo cuenta con una línea. Sin embargo, el tiempo estimado para ingresar el texto mediante la terminal, tiene un tiempo estimado de O(1).

La función del “tokenizador” tiene un tiempo estimado de O(n) por la longitud de un texto ingresado, sin embargo, a la hora de insertar, el mejor tiempo de ejecución es O(n) si no existe algún nodo mientras que el peor tiempo de ejecución es O(n^2) si ya existe un nodo con la misma letra inicial.

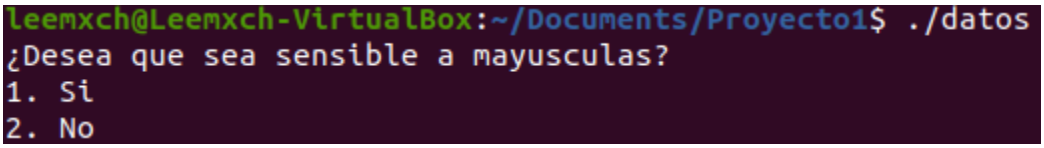
Las funciones de “mostrarDatos”, “mostrarDatosPos” y “buscar” tienen un tiempo estimado de 3O(n), ya que tienen que verificar todos los conjuntos de nodos.

Con respecto a las funciones de los delimitadores, ambas funciones, “delimitar” y “revisarDelimitar”, tienen un tiempo de O(n).

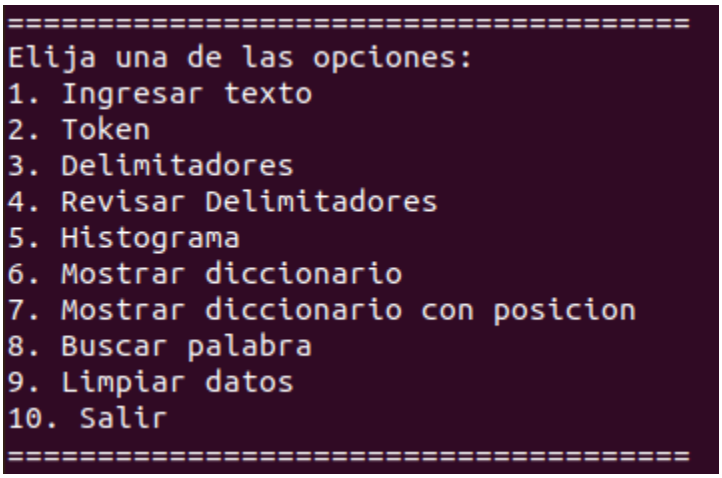
Por último, las mayúsculas y minúsculas tienen una diferencia de tiempo de O(n), ya que es necesario convertir todos los elementos a minúsculas si no se desea ser sensible a las mayúsculas, mientras que, al ser sensible, se puede obtener los datos sin necesidad de modificar los textos iniciales dados por el usuario.

Funcionamiento del programa

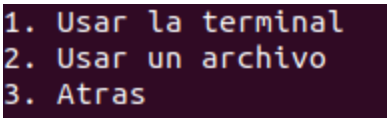
Al iniciar el programa, se le pide al usuario que indique si quiere que el programa completo sea sensible a las mayúsculas o no.



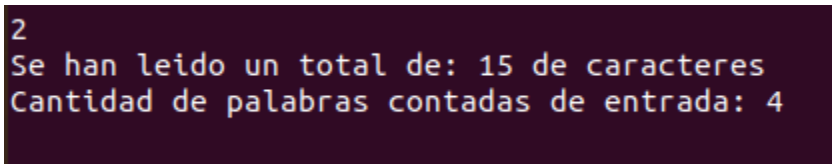
Luego de que el usuario seleccione alguna de las opciones válidas, se le desplegará un menú en el que se le presentan opciones para interactuar.



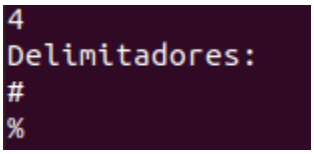
Al seleccionar la primera opción, el usuario puede optar por 2 opciones, ingresar un texto por medio de la terminal o usar un archivo de texto .txt, con el fin de poder interactuar con el resto del programa.



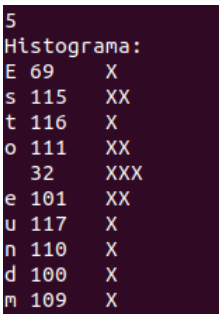
Luego de ingresar un texto por medio de la terminal o archivo de texto, la segunda opción, permite que el usuario pueda generar un diccionario a partir de las palabras ingresadas, en donde se le mostrará la cantidad de caracteres leídos y la cantidad de palabras.



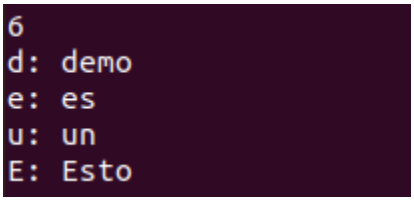
La opción 3, permite que el usuario añada nuevos símbolos simulando un separador de palabras y la opción 4, permite que el usuario pueda revisar los delimitadores personalizados. Por ejemplo, se le ingresan previamente los símbolos “#” y “%” con el fin de mostrar la opción 4.

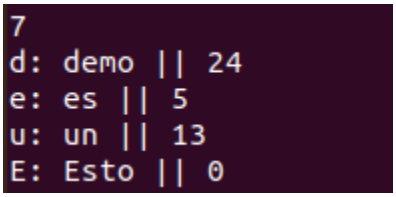


La opción 5, genera un histograma para verificar la cantidad de caracteres repetido ingresados previamente. Por ejemplo, se ingresó “Esto es un demo” junto con la opción de ser sensible a mayúsculas.

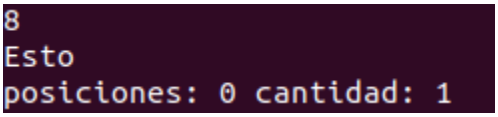


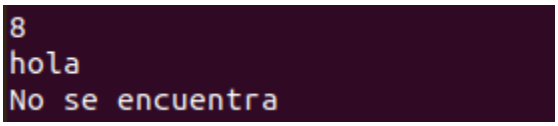
La opción 6 y 7 son casi similares, ya que la opción 6 muestra únicamente las palabras ingresadas, mientras que la opción 7, muestra las posiciones en donde se encontraron. Utilizando el mismo ejemplo de texto, tenemos:





La opción 8 permite al usuario buscar palabras dentro del diccionario, si la opción se encuentra dentro del sistema, devuelve la cantidad de veces y posiciones encontradas y si no se encuentra, devuelve que no existe.





Por último, la opción 9 permite al usuario limpiar completamente el diccionario para ingresar nuevos datos.

Conclusión

Las estructuras de tipo de datos abstractos tienen diversos usos que nos ayudan a almacenar información de manera eficiente dependiendo de las condiciones y los usos que se les otorguen. La cola y las tablas de hash fueron parte fundamental de este programa para poder manejar los datos de manera eficiente, ya que la misma tabla de hash tiene la misma funcionalidad de un diccionario.

Por último, se aprendió que dependiendo de los diferentes caracteres que pueden significar lo mismo, pero representado de diferente forma, puede alterar mucho el tiempo estimado de ejecución por tener que evaluar o crear un nuevo espacio para almacenar dicho carácter.

Referencias

Cplusplus (2020). Templates. Recuperado de http://www.cplusplus.com/doc/oldtutorial/templates/

EcuRed, (s.f.). Cola (Estructura de datos). Concepto de cola. Recuperado de https://www.ecured.cu/Cola_(Estructura_de_datos)

EcuRed, (s.f.). Tabla hash. Concepto. Recuperado de https://www.ecured.cu/Tabla_hash

Moshiri N., Izhikevich L. (2016). Data Structures. Stepik. Recuperado de https://stepik.org/course/579/syllabus

Microsoft Docs, (2019). Microsoft, Header files(C++). Recuperado de https://docs.microsoft.com/en-us/cpp/cpp/header-files-cpp?view=vs-2019

Sena, M. (2019). Medium, Estructuras de Datos, Primera parte – Array, linked lists, stacks, queues. Recuperado de https://medium.com/techwomenc/estructuras-de-datos-a29062de5483

Sama, S. (s.f.). DSTool, DataStructurres Tool, Qué son las tablas hash. Concepto. Recuperado de http://www.hci.uniovi.es/Products/DSTool/hash/hash-queSon.html

Anexo

proyecto.cpp

```
1  #include <iostream>
2  #include <fstream>
3  #include <stdlib.h>
4  #include <string>
5  #include <list>
6  #include <iterator>
7  #include <bits/stdc++.h>
8  #include "tablaHash.h"
9  #include "cola.h"
10 using namespace std;
11 /*
12 Funcion para leer un archivo de texto
13 */
14 string leer(){
15       fstream archivo("texto.txt");
16       string linea;
17       string texto;
18       if (archivo.is_open()){
19             while(getline(archivo,linea)){
20                   texto += linea + "\n";
21             }
22             archivo.close();
23       }
24       return texto;
25 }
26
27 /*
28 Funcion sensible a mayusculas
29 */
```

```
void tokenizadorSens(tablaHash<char>* tabla, list<char> lista, string texto,
bool sensible){
      list<char>::iterator revisar;
      string guardar = "";
      int contador = 0;
      int contadorPalabras = 0;
      int cantidadCaracter = 0;
      for (int i=0; i<texto.length();i++){
            if (texto[i] == ' ' or texto[i] == '\n' or texto[i] == '\t' or
texto[i] == '.'){
                  if (!sensible){
      transform(guardar.begin(),guardar.end(),guardar.begin(),::tolower);
                  }
                  tabla->insert(guardar,contador);
                  cantidadCaracter++;
                  contadorPalabras++;
                  contador += i+1;
                  guardar = "";
            }
            else{
                  for (revisar = lista.begin(); revisar != lista.end();
revisar++){
                        if (*revisar == texto[i]){
                              tabla->insert(guardar,contador);
                              cantidadCaracter++;
                              contadorPalabras++;
                              i++;
                              contador += i+1;
                              guardar = "";
                              break;
                        }

                  }
                  guardar += texto[i];
```

```
                    cantidadCaracter++;
            }
      }
      cout << "Se han leido un total de: " << cantidadCaracter-1 << " de caracteres" << endl;
      cout << "Cantidad de palabras contadas de entrada: " << contadorPalabras << endl << endl;
}
/*
Funcion para mostrar los datos de entrada ordenados
*/
void tokenizadorMostrar(tablaHash<char>* tabla){
      tabla->mostrarDatos();
}
/*
Funcion para mostrar los datos de entrada ordenados y sus posiciones
*/
void tokenizadorPosMostrar(tablaHash<char>* tabla){
      tabla->mostrarDatosPos();
}
/*
Funcion para meter delimitadores a la lista
*/
list<char> delimitar(list<char> lista){
      char texto[10];
      cin >> texto;
      cin.ignore();
      lista.push_back(*texto);
      return lista;
}
/*
Funcion para reivsar delimitadores en la lista
*/
```

```
97  void revisarDelimitar(list<char> lista){
98        list<char>::iterator revisar;
99        cout << "Delimitadores:" << endl;
100       for (revisar = lista.begin(); revisar != lista.end(); revisar++){
101             cout << *revisar << endl;
102       }
103 }
104 /*
105 Funcion para ver un histograma sensible a mayusculas
106 */
107 void mayusculas(string texto){
108       Cola<char> colita;
109       for (int i=0; i<texto.length()-1; i++){
110             bool resultado = colita.revisar(texto[i]);
111             if (!resultado){
112                   colita.queue(texto[i]);
113             }
114       }
115       colita.showStack();
116 }
117 /*
118 Funcion para ver un histograma no sensible a mayusculas
119 */
120 void minusculas(string texto){
121       Cola<char> colita;
122       for (int j=0; j<texto.length()-1; j++){
123             if (texto[j]<96 && texto[j]>64){
124                   texto[j] = texto[j] + 32;
125             }
126       }
127       for (int i=0; i<texto.length(); i++){
128             bool resultado = colita.revisar(texto[i]);
```

```
129               if (!resultado){
130                     colita.queue(texto[i]);
131               }
132         }
133         colita.showStack();
134   }
135
136   /*
137         Main
138   */
139   int main(){
140         tablaHash<char>* tablita = new tablaHash<char>();
141         list<char> delimitadores;
142         int opcion;
143         bool sensible = false;
144         string texto;
145         string encontrar;
146         while (true){
147               cout << "¿Desea que sea sensible a mayusculas?" << endl;
148               cout << "1. Si" << endl;
149               cout << "2. No" << endl;
150               cin >> opcion;
151               cin.ignore();
152               if (opcion == 1 || opcion == 2){
153                     if (opcion == 1){
154                           sensible = true;
155                           break;
156                     }
157                     break;
158               }
159         }
160         do{
```

```
161             cout << "====================================" << endl;
162             cout << "Elija una de las opciones: " << endl;
163             cout << "1. Ingresar texto" << endl;
164             cout << "2. Token" << endl;
165             cout << "3. Delimitadores" << endl;
166             cout << "4. Revisar Delimitadores" << endl;
167             cout << "5. Histograma" << endl;
168             cout << "6. Mostrar diccionario" << endl;
169             cout << "7. Mostrar diccionario con posicion" << endl;
170             cout << "8. Buscar palabra" << endl;
171             cout << "9. Limpiar datos" << endl;
172             cout << "10. Salir" << endl;
173             cout << "====================================" << endl;
174             cin >> opcion;
175             cin.ignore();
176             switch(opcion){
177                   case 1:
178                         do{
179                               cout << "1. Usar la terminal" << endl;
180                               cout << "2. Usar un archivo" << endl;
181                               cout << "3. Atras" << endl;
182                               cin >> opcion;
183                               cin.ignore();
184                               switch(opcion){
185                                     case 1:
186                                     cout << "Ingrese el texto: ";
187                                     getline(cin,texto);
188                                     texto += "\n";
189                                     opcion = 3;
190                                     break;
191                                     case 2:
192                                     texto = leer();
```

```
193                                     opcion = 3;
194                                     break;
195                                     default:
196                                     break;
197                               }
198                         }while (opcion!=3);
199                         break;
200                   case 2:
201                         tokenizadorSens(tablita,     delimitadores,     texto,
202   sensible);
203                         break;
204                   case 3:
205                         delimitadores = delimitar (delimitadores);
206                         break;
207                   case 4:
208                         revisarDelimitar (delimitadores);
209                         break;
210                   case 5:
211                         if (sensible){
212                               mayusculas(texto);
213                         }
214                         else{
215                               minusculas(texto);
216                         }
217                         break;
218                   case 6:
219                         tokenizadorMostrar(tablita);
220                         break;
221                   case 7:
222                         tokenizadorPosMostrar(tablita);
223                         break;
224                   case 8:
```

```
                        getline(cin, encontrar);
                        if (sensible){
                              encontrar=tablita->buscar(encontrar);
                              cout << encontrar << endl;
                        }
                        else{

      transform(encontrar.begin(),encontrar.end(),encontrar.begin(),::tolower
);
                              encontrar=tablita->buscar(encontrar);
                              cout << encontrar << endl;
                        }
                        break;
                  case 9:
                        delete(tablita);
                        tablita = new tablaHash<char>();
                        cout << "Se limpio la memoria con exito" << endl;
                        break;
                  case 10:
                        cout << "Saliendo..." << endl;
                        break;
                  default:
                        break;
                  }
      }while (opcion != 10);
      return 0;
}
```

cola.h

```
#include <iostream>
#include <string>
#include "nodoColita.h"
using namespace std;

```

```
template <class T> class Cola{
private:
	NodoColita<T> *inicio;
	NodoColita<T> *ultimo;
public:
	Cola();
	bool isEmpty();
	void queue(T);
	T dequeue();
	bool revisar(char);
	void showStack();
};
template <class T> Cola<T>::Cola(){
	inicio = NULL;
	ultimo = NULL;
}
template <class T> bool Cola<T>::isEmpty(){
	if (inicio==NULL && ultimo==NULL){
		return true;
	}
	return false;
}
template <class T> void Cola<T>::queue(T patata){
	if (isEmpty()){
		NodoColita<T> * newNodoColita = new NodoColita<T>(patata);
		inicio = newNodoColita;
		ultimo = newNodoColita;
	}
	else{
		NodoColita<T> *newNodoColita = new NodoColita<T>(patata);
		ultimo->siguiente=newNodoColita;
		newNodoColita->anterior=ultimo;
```

```
38              ultimo = newNodoColita;
39        }
40  }
41  template <class T> T Cola<T>::dequeue(){
42        if(!isEmpty()){
43              if(inicio == ultimo){
44                    T temp = inicio->dato;
45                    free(inicio);
46                    inicio = NULL;
47                    ultimo = NULL;
48                    return temp;
49              }
50              else{
51                    T temp = inicio->dato;
52                    NodoColita<T>* recuerdo = inicio->siguiente;
53                    recuerdo->anterior = NULL;
54                    free(inicio);
55                    inicio = recuerdo;
56                    return temp;
57              }
58        }
59        return 0;
60  }
61  template <class T> bool Cola<T>::revisar(char texto){
62        if (!isEmpty()){
63              NodoColita<T>* temp = inicio;
64              while(temp->siguiente != NULL){
65                    if (temp->dato==texto){
66                          temp->cantidad += 1;
67                          return true;
68                    }
69                    temp = temp->siguiente;
```

```
70              }
71              if (temp->dato==texto){
72                    temp->cantidad += 1;
73                    return true;
74              }
75        }
76        return false;
77  }
78  template <class T> void Cola<T>::showStack(){
79        cout << "Histograma:" << endl;
80        if (!isEmpty()){
81              NodoColita<T>* temp = inicio;
82              while(temp->siguiente != NULL){
83                    cout << temp->dato << " " << (int)temp->dato <<  "\t";
84                    for (int i=0; i<temp->cantidad;i++){
85                          cout << "X";
86                    }
87                    cout << endl;
88                    temp = temp->siguiente;
89              }
90              cout << temp->dato << " " << (int)temp->dato <<  "\t";
91              for (int i=0; i<temp->cantidad;i++){
92                    cout << "X";
93              }
94              cout << endl;
95        }
96        else{
97              cout << "No hay datos en la pila" << endl;
98        }
99  }
```

nodoColita.h

```
1  #include <iostream>
2  using namespace std;
3  template <class T> class NodoColita{
4  public:
5        T dato;
6        int cantidad;
7        NodoColita* siguiente;
8        NodoColita* anterior;
9        NodoColita(T);
10       int getCantidad();
11       void setCantidad(int);
12 };
13 template <class T> NodoColita<T>::NodoColita(T datos){
14       dato = datos;
15       cantidad = 1;
16       siguiente = NULL;
17       anterior = NULL;
18 }
```

tablaHash.h

```
1  #include <iostream>
2  #include <string>
3  #include "nodoHash.h"
4  using namespace std;
5
6  template <class T> class tablaHash{
7  private:
8        nodoHash<T>* inicioNum;
9        nodoHash<T>* ultimoNum;
10       nodoHash<T>* inicioLetra;
11       nodoHash<T>* ultimoLetra;
12       nodoHash<T>* inicioLetraMayus;
```

```
13          nodoHash<T>* ultimoLetraMayus;
14    public:
15          tablaHash();
16          bool insert(string,int);
17          void mostrarDatos();
18          void mostrarDatosPos();
19          string buscar(string);
20    };
21    template <class T> tablaHash<T>::tablaHash(){
22          inicioNum = new nodoHash<T>('0');
23          ultimoNum = inicioNum;
24          for (int i=0; i<10;i++){
25                nodoHash<T>* newNodo = new nodoHash<T>('1'+i);
26                ultimoNum->siguiente=newNodo;
27                newNodo->anterior=ultimoNum;
28                ultimoNum = newNodo;
29          }
30          inicioLetra = new nodoHash<T>('a');
31          ultimoLetra = inicioLetra;
32          for (int i=0; i<25;i++){
33                nodoHash<T>* newNodo = new nodoHash<T>('b'+i);
34                ultimoLetra->siguiente=newNodo;
35                newNodo->anterior=ultimoLetra;
36                ultimoLetra = newNodo;
37          }
38          inicioLetraMayus = new nodoHash<T>('A');
39          ultimoLetraMayus = inicioLetraMayus;
40          for (int i=0; i<25;i++){
41                nodoHash<T>* newNodo = new nodoHash<T>('B'+i);
42                ultimoLetraMayus->siguiente=newNodo;
43                newNodo->anterior=ultimoLetraMayus;
44                ultimoLetraMayus = newNodo;
```

```
45          }
46    }
47    template <class T> bool tablaHash<T>::insert(string patata, int pos){
48          if (patata[0] >= '0' && patata[0] <= '9'){
49                nodoHash<T>* auxiliar = inicioNum;
50                for(int i=0; i<patata[0]-'0';i++){
51                      auxiliar = auxiliar->siguiente;
52                }
53                Nodo<string>* auxiliarNodo = auxiliar->lista;
54                while(auxiliarNodo != NULL){
55                      if (auxiliarNodo->dato == patata){
56                            auxiliarNodo->insertarPos(pos);
57                            return true;
58                      }
59                      auxiliarNodo = auxiliarNodo->siguiente;
60                }
61                Nodo<string>* newNodo = new Nodo<string>(patata,pos);
62                if (auxiliar->lista == NULL){
63                      auxiliar->lista = newNodo;
64                      auxiliar->ultimo = newNodo;
65                }
66                else{
67                      Nodo<string>* auxiliar2 = auxiliar->ultimo;
68                      auxiliar2->siguiente = newNodo;
69                      auxiliar->ultimo = newNodo;
70                }
71                return true;
72          }
73          else{
74                if (patata[0] >= 'a' && patata[0] <= 'z'){
75                      if (patata[0] >= 'a' && patata[0] <= 'n'){
76                            nodoHash<T>* auxiliar = inicioLetra;
```

```
77                      for(int i=0; i<patata[0]-'a';i++){
78                              auxiliar = auxiliar->siguiente;
79                      }
80                      Nodo<string>* auxiliarNodo = auxiliar->lista;
81                      while(auxiliarNodo != NULL){
82                              if (auxiliarNodo->dato == patata){
83                                      auxiliarNodo->insertarPos(pos);
84                                      return true;
85                              }
86                              auxiliarNodo = auxiliarNodo->siguiente;
87                      }
88                      Nodo<string>* newNodo = new Nodo<string>(patata,pos);
89                      if (auxiliar->lista == NULL){
90                              auxiliar->lista = newNodo;
91                              auxiliar->ultimo = newNodo;
92                      }
93                      else{
94                              Nodo<string>* auxiliar2 = auxiliar->ultimo;
95                              auxiliar2->siguiente = newNodo;
96                              auxiliar->ultimo = newNodo;
97                      }
98                      return true;
99              }
100             else{
101                     nodoHash<T>* auxiliar = ultimoLetra;
102                     for(int i=0; i<'z'-patata[0];i++){
103                             auxiliar = auxiliar->anterior;
104                     }
105                     Nodo<string>* auxiliarNodo = auxiliar->lista;
106                     while(auxiliarNodo != NULL){
107                             if (auxiliarNodo->dato == patata){
108                                     auxiliarNodo->insertarPos(pos);
```

```
109                                     return true;
110                               }
111                               auxiliarNodo = auxiliarNodo->siguiente;
112                         }
113                         Nodo<string>* newNodo = new Nodo<string>(patata,pos);
114                         if (auxiliar->lista == NULL){
115                               auxiliar->lista = newNodo;
116                               auxiliar->ultimo = newNodo;
117                         }
118                         else{
119                               Nodo<string>* auxiliar2 = auxiliar->ultimo;
120                               auxiliar2->siguiente = newNodo;
121                               auxiliar->ultimo = newNodo;
122                         }
123                         return true;
124                   }
125             }
126             else{
127                   if (patata[0] >= 'A' && patata[0] <= 'N'){
128                         nodoHash<T>* auxiliar = inicioLetraMayus;
129                         for(int i=0; i<patata[0]-'A';i++){
130                               auxiliar = auxiliar->siguiente;
131                         }
132                         Nodo<string>* auxiliarNodo = auxiliar->lista;
133                         while(auxiliarNodo != NULL){
134                               if (auxiliarNodo->dato == patata){
135                                     auxiliarNodo->insertarPos(pos);
136                                     return true;
137                               }
138                               auxiliarNodo = auxiliarNodo->siguiente;
139                         }
140                         Nodo<string>* newNodo = new Nodo<string>(patata,pos);
```

```
141                     if (auxiliar->lista == NULL){
142                           auxiliar->lista = newNodo;
143                           auxiliar->ultimo = newNodo;
144                     }
145                     else{
146                           Nodo<string>* auxiliar2 = auxiliar->ultimo;
147                           auxiliar2->siguiente = newNodo;
148                           auxiliar->ultimo = newNodo;
149                     }
150                     return true;
151               }
152               else{
153                     nodoHash<T>* auxiliar = ultimoLetraMayus;
154                     for(int i=0; i<'Z'-patata[0];i++){
155                            auxiliar = auxiliar->anterior;
156                     }
157                     Nodo<string>* auxiliarNodo = auxiliar->lista;
158                     while(auxiliarNodo != NULL){
159                           if (auxiliarNodo->dato == patata){
160                                 auxiliarNodo->insertarPos(pos);
161                                 return true;
162                           }
163                           auxiliarNodo = auxiliarNodo->siguiente;
164                     }
165                     Nodo<string>* newNodo = new Nodo<string>(patata,pos);
166                     if (auxiliar->lista == NULL){
167                           auxiliar->lista = newNodo;
168                           auxiliar->ultimo = newNodo;
169                     }
170                     else{
171                           Nodo<string>* auxiliar2 = auxiliar->ultimo;
172                           auxiliar2->siguiente = newNodo;
```

```
                                    auxiliar->ultimo = newNodo;
                              }
                              return true;
                        }
                  }
            }
            return false;
}
template <class T> void tablaHash<T>::mostrarDatos(){
            nodoHash<T>* auxiliar = inicioNum;
            for(int i=0; i<10;i++){
                  if(auxiliar->lista != NULL){
                        Nodo<string>* auxiliar2 = auxiliar->lista;
                        cout << auxiliar->letra << ": ";
                        while (auxiliar2 != NULL){
                              cout << auxiliar2->dato;
                              if(auxiliar2->siguiente != NULL){
                                    cout << ", ";
                              }
                              auxiliar2 = auxiliar2->siguiente;
                        }
                        cout << "\n";
                  }
                  auxiliar = auxiliar->siguiente;
            }
            nodoHash<T>* auxiliar3 = inicioLetra;
            for(int i=0; i<26;i++){
                  if (auxiliar3->lista != NULL){
                        Nodo<string>* auxiliar4 = auxiliar3->lista;
                        cout << auxiliar3->letra << ": ";
                        while (auxiliar4 != NULL){
                              cout << auxiliar4->dato;
```

```
                        if(auxiliar4->siguiente != NULL){
                              cout << ", ";
                        }
                        auxiliar4 = auxiliar4->siguiente;
                  }
                  cout << "\n";
            }
            auxiliar3 = auxiliar3->siguiente;
      }
      nodoHash<T>* auxiliar5 = inicioLetraMayus;
      for(int i=0; i<26;i++){
            if (auxiliar5->lista != NULL){
                  Nodo<string>* auxiliar6 = auxiliar5->lista;
                  cout << auxiliar5->letra << ": ";
                  while (auxiliar6 != NULL){
                        cout << auxiliar6->dato;
                        if(auxiliar6->siguiente != NULL){
                              cout << ", ";
                        }
                        auxiliar6 = auxiliar6->siguiente;
                  }
                  cout << "\n";
            }
            auxiliar5 = auxiliar5->siguiente;
      }
}
template <class T> void tablaHash<T>::mostrarDatosPos(){
      nodoHash<T>* auxiliar = inicioNum;
      for(int i=0; i<10;i++){
            if (auxiliar->lista!=NULL){
                  Nodo<string>* auxiliar2 = auxiliar->lista;
                  cout << auxiliar->letra << ": ";
```

```
237                     while (auxiliar2 != NULL){
238                           cout << auxiliar2->dato << " || ";
239                           nodoNum<int>* auxiliar5 = auxiliar2->pos;
240                           while(auxiliar5 != NULL){
241                                 cout << auxiliar5->pos;
242                                 if(auxiliar5->siguiente != NULL){
243                                       cout << ", ";
244                                 }
245                                 auxiliar5 = auxiliar5->siguiente;
246                           }
247                           auxiliar2 = auxiliar2->siguiente;
248                     }
249                     cout << "\n";
250               }
251               auxiliar = auxiliar->siguiente;
252         }
253         nodoHash<T>* auxiliar3 = inicioLetra;
254         for(int i=0; i<26;i++){
255               if (auxiliar3->lista != NULL){
256                     Nodo<string>* auxiliar4 = auxiliar3->lista;
257                     cout << auxiliar3->letra << ": ";
258                     while (auxiliar4 != NULL){
259                           cout << auxiliar4->dato << " || ";
260                           nodoNum<int>* auxiliar6 = auxiliar4->pos;
261                           while(auxiliar6 != NULL){
262                                 cout << auxiliar6->pos;
263                                 if(auxiliar6->siguiente != NULL){
264                                       cout << ", ";
265                                 }
266                                 auxiliar6 = auxiliar6->siguiente;
267                           }
268                           auxiliar4 = auxiliar4->siguiente;
```

```
269                     }
270                     cout << "\n";
271               }
272               auxiliar3 = auxiliar3->siguiente;
273         }
274         nodoHash<T>* auxiliar7 = inicioLetraMayus;
275         for(int i=0; i<26;i++){
276               if (auxiliar7->lista != NULL){
277                     Nodo<string>* auxiliar8 = auxiliar7->lista;
278                     cout << auxiliar7->letra << ": ";
279                     while (auxiliar8 != NULL){
280                           cout << auxiliar8->dato << " || ";
281                           nodoNum<int>* auxiliar9 = auxiliar8->pos;
282                           while(auxiliar9 != NULL){
283                                 cout << auxiliar9->pos;
284                                 if(auxiliar9->siguiente != NULL){
285                                       cout << ", ";
286                                 }
287                                 auxiliar9 = auxiliar9->siguiente;
288                           }
289                           auxiliar8 = auxiliar8->siguiente;
290                     }
291                     cout << "\n";
292               }
293               auxiliar7 = auxiliar7->siguiente;
294         }
295   }
296   template <class T> string tablaHash<T>::buscar(string patata){
297         string texto = "";
298         if (patata[0] >= '0' && patata[0] <= '9'){
299               nodoHash<T>* auxiliar = inicioNum;
300               for(int i=0; i<patata[0]-'0';i++){
```

```
301                     auxiliar = auxiliar->siguiente;
302               }
303               Nodo<string>* auxiliarNodo = auxiliar->lista;
304               while(auxiliarNodo != NULL){
305                     if (auxiliarNodo->dato == patata){
306                           texto += "posiciones: ";
307                           nodoNum<int>* recorrido = auxiliarNodo->pos;
308                           int count=0;
309                           while(recorrido!=NULL){
310                                 texto += to_string(recorrido->pos);
311                                 count++;
312                                 if (recorrido->siguiente != NULL){
313                                       texto += ", ";
314                                 }
315                                 recorrido = recorrido->siguiente;
316                           }
317                           texto += " cantidad: " + to_string(count);
318                           return texto;
319                     }
320                     auxiliarNodo = auxiliarNodo->siguiente;
321               }
322         }
323         else{
324               if (patata[0] >= 'a' && patata[0] <= 'z'){
325                     if (patata[0] >= 'a' && patata[0] <= 'n'){
326                           nodoHash<T>* auxiliar = inicioLetra;
327                           for(int i=0; i<patata[0]-'a';i++){
328                                 auxiliar = auxiliar->siguiente;
329                           }
330                           Nodo<string>* auxiliarNodo = auxiliar->lista;
331                           while(auxiliarNodo != NULL){
332                                 if (auxiliarNodo->dato == patata){
```

```
                                    texto += "posiciones: ";
                                    nodoNum<int>*  recorrido  =  auxiliarNodo->pos;
                                    int count=0;
                                    while(recorrido!=NULL){
                                          texto += to_string(recorrido->pos);
                                          count++;
                                          if (recorrido->siguiente != NULL){
                                                texto += ", ";
                                          }
                                          recorrido = recorrido->siguiente;
                                    }
                                    texto += " cantidad: " + to_string(count);
                                    return texto;
                              }
                              auxiliarNodo = auxiliarNodo->siguiente;
                        }
                  }
                  else{
                        nodoHash<T>* auxiliar = ultimoLetra;
                        for(int i=0; i< 'z' - patata[0];i++){
                              auxiliar = auxiliar->anterior;
                        }
                        Nodo<string>* auxiliarNodo = auxiliar->lista;
                        while(auxiliarNodo != NULL){
                              if (auxiliarNodo->dato == patata){
                                    texto += "posiciones: ";
                                    nodoNum<int>*  recorrido  =  auxiliarNodo->pos;
                                    int count=0;
                                    while(recorrido!=NULL){
                                          texto += to_string(recorrido->pos);
                                          count++;
```

```
366                                         if (recorrido->siguiente != NULL){
367                                               texto += ", ";
368                                         }
369                                         recorrido = recorrido->siguiente;
370                                   }
371                                   texto += " cantidad: " + to_string(count);
372                                   return texto;
373                             }
374                             auxiliarNodo = auxiliarNodo->siguiente;
375                       }
376                 }
377           }
378           else{
379                 if (patata[0] >= 'A' && patata[0] <= 'N'){
380                       nodoHash<T>* auxiliar = inicioLetraMayus;
381                       for(int i=0; i<patata[0]-'A';i++){
382                             auxiliar = auxiliar->siguiente;
383                       }
384                       Nodo<string>* auxiliarNodo = auxiliar->lista;
385                       while(auxiliarNodo != NULL){
386                             if (auxiliarNodo->dato == patata){
387                                   texto += "posiciones: ";
388                                   nodoNum<int>*  recorrido  =  auxiliarNodo-
389 >pos;
390                                   int count=0;
391                                   while(recorrido!=NULL){
392                                         texto += to_string(recorrido->pos);
393                                         count++;
394                                         if (recorrido->siguiente != NULL){
395                                               texto += ", ";
396                                         }
397                                         recorrido = recorrido->siguiente;
```

```
                                    }
                                    texto += " cantidad: " + to_string(count);
                                    return texto;
                              }
                              auxiliarNodo = auxiliarNodo->siguiente;
                        }
                  }
                  else{
                        nodoHash<T>* auxiliar = ultimoLetraMayus;
                        for(int i=0; i<'Z'-patata[0];i++){
                              auxiliar = auxiliar->anterior;
                        }
                        Nodo<string>* auxiliarNodo = auxiliar->lista;
                        while(auxiliarNodo != NULL){
                              if (auxiliarNodo->dato == patata){
                                    texto += "posiciones: ";
                                    nodoNum<int>*  recorrido  =  auxiliarNodo-
>pos;
                                    int count=0;
                                    while(recorrido!=NULL){
                                          texto += to_string(recorrido->pos);
                                          count++;
                                          if (recorrido->siguiente != NULL){
                                                texto += ", ";
                                          }
                                          recorrido = recorrido->siguiente;
                                    }
                                    texto += " cantidad: " + to_string(count);
                                    return texto;
                              }
                              auxiliarNodo = auxiliarNodo->siguiente;
                        }
```

```
                    }
                }

        }
        return "No se encuentra";
}
```

nodoHash.h

```
#include <iostream>
#include <string>
#include "nodo.h"
using namespace std;
template <class T> class nodoHash{
public:
        char letra;
        Nodo<string>* lista;
        Nodo<string>* ultimo;
        nodoHash* siguiente;
        nodoHash* anterior;
        nodoHash(char);
};
template <class T> nodoHash<T>::nodoHash(char caracter){
        letra = caracter;
        lista = NULL;
        ultimo = NULL;
        siguiente = NULL;
        anterior = NULL;
}
```

nodo.h

```
#include <iostream>
#include <string>
#include "nodoNum.h"
using namespace std;
```

```
5  template <class T> class Nodo{
6  public:
7        T dato;
8        nodoNum<int>* pos;
9        Nodo<T>* siguiente;
10       nodoNum<int>* ultimo;
11       Nodo(T,int);
12       void insertarPos(int);
13 };
14 template <class T> Nodo<T>::Nodo(T datos,int posicion){
15       dato = datos;
16       pos = new nodoNum<int>(posicion);
17       ultimo = pos;
18       siguiente = NULL;
19 }
20 template <class T> void Nodo<T>::insertarPos(int pos){
21       nodoNum<int> *auxiliar = new nodoNum<int>(pos);
22       ultimo->siguiente = auxiliar;
23       ultimo = auxiliar;
24 }
```

nodoNum.h

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  template <class T> class nodoNum{
6  public:
7        int pos;
8        nodoNum<T>* siguiente;
9        nodoNum<T>(int);
10 };
11
```

```
12  template <class T> nodoNum<T>::nodoNum(int posision){
13        pos = posision;
14        siguiente = NULL;
15  }
```