Universidad de Guanajuato

Escuela de Matemáticas y Computación

Estructura de Datos y Algoritmos I

Enero - junio 2022

Práctica 5

OBJETIVOS:

- Familiarizar al estudiante con la estructura Tabla Hash y su implementación.
- Proporcionar al estudiante con un ejemplo de utilidad de una Tabla Hash.

Enunciado

Llevaremos a cabo la implementación de una tabla hash a través de dos estructuras con las que trabajamos en las prácticas pasadas. Declararemos la tabla hash como un arreglo dinámico el cual obtendrá su tamaño a través del constructor de la tabla hash. Luego, para solucionar las colisiones utilizaremos una lista enlazada.

La tabla hash será utilizada para almacenar el vocabulario de un determinado texto en donde cada palabra será la llave y la cantidad de veces que la palabra aparece será el valor, de manera que tenemos que:

key: <palabra>, **value:** <cuenta de palabra>.

Al hablar de vocabulario nos referimos a las palabras únicas que aparecen en un texto. Tomemos de ejemplo el siguiente texto.

“Este texto contiene pocas palabras como vocabulario. El vocabulario son las distintas palabras que aparecen en el texto.”

El vocabulario y su cuenta sería el siguiente: (Este, 1), (texto, 2), (contiene, 1), (pocas, 1), (palabras, 2), (como, 1), (vocabulario, 2), (El, 1), (son, 1), (las, 1), (distintas, 1), (que, 1), (aparecen, 1), (en, 1), (el, 1).

Esta tarea es indispensable en áreas como procesamiento del lenguaje natural (NLP). Notemos que en este caso el vocabulario es case sensitive, porque las palabras “El” y “el” son consideradas como palabras diferentes. Para la práctica considere también que el vocabulario es case sensitive, es decir las mayúsculas y minúsuclas son diferentes.

Utilice la siguiente firma.

```cpp
#include <iostream>
#include <string>
#include <list>
using namespace std;

class HashTable{
  public:
  HashTable(unsigned int n);                // constructor
  ~HashTable();                             // destructor
  void insert(string key, int value);       // insert or update
  pair<string, int>* find(string key);      // find key-value pair

  private:
  unsigned int N;                           // size of hash table
  unsigned int hash(string key);            // hash function
  list<pair<string, int>>* key_values;      // key-value array
};
```

Universidad de Guanajuato

Escuela de Matemáticas y Computación

Estructura de Datos y Algoritmos I

Enero - junio 2022

Notemos que estamos utilizando dos estructuras propias de la STL de c++ (Pair, List). **Pair** almacena pares de valores y son accesibles a través de sus atributos **pair.first** y **pair.second.** El tipo de valores que contendrá se indican a través de "<tipo1, tipo2>", es decir si tuvieramos pair <int, int> este par almacena en ambos atributos valores de tipo entero.

La otra estructura que estamos utilizando es la implementación de listas enlazadas de la STL a través de la librería **list**. De igual manera, el tipo de dato que la lista almacena se indica a través de <tipo>. Observemos que podemos declarar tipos muy complejos como el que tenemos en la imagen, una lista que almacena pares de datos de tipo cadena y entero, i.e. **list<pair<string, int>>.**

Notemos a través de la firma que el arreglo key_values almacenará listas enlazadas de la STL, que a su vez almacenan pares de <string, int>. Esto es porque los pares de key-value se almacenarán en las listas enlazadas. De manera que cuando ocurra una inserción, el par key-value en realidad se inserta en una lista enlazada.

Ejercicios:

1. Implemente el constructor de la tabla hash que reciba como parámetro el tamaño inicial del arreglo que almacenará las listas, que a su vez almacenan los pares de **key-value**.

2. Defina su propia función de hash para indexar a una posición de su arreglo de listas enlazadas. La función hash debe procesar cadenas como lo podemos ver en la firma de la clase. Puede auxiliarse de funciones de la stl como **std::hash<std::string>{}(str)** que se encargan de generar un hash para una cadena.

3. Implemente la función de buscar en la tabla hash. Notemos que esta función regresa un puntero del objeto pair. Esto es por el caso en el cual queramos actualizar el valor del par key-value, ya tenemos la referencia y podemos actualizar su valor sin tener que insertar nuevamente el valor en la tabla hash.

4. Implemente la función de insert de la tabla hash. Debido a que solucionamos las colisiones con una lista enlazada, esta función debe buscar inicialmente si la llave ya existe en la lista enlazada. En caso de que la llave ya exista, debemos cambiar su valor. En caso de que no exista, insertamos la nueva llave con su valor. Al momento de insertar en la lista enlazada utilice la función **list.push_back** de la STL. Esta función inserta el nodo deseado al final de la lista, lo cual se puede hacer en tiempo constante teniendo un puntero al final de la lista. Notemos entonces que la inserción siempre se realiza en tiempo constante. La búsqueda para verificar si el valor existe es la que es en el peor caso de orden lineal, pero con una buena función hash podemos llegar a un orden amortiguado constante, es decir se espera que en

promedio el tiempo de búsqueda sea constante debido a que la mayoría de listas enlazadas deberían contener pocos nodos o incluso solo un nodo. **NOTA:** Para esta función debe realizar la búsqueda de la llave para ver si existe, no debe escribir nuevamente el código de búsqueda en la inserción, utilice su función para buscar que ya implementó anteriormente.

5. Defina una función update (**de preferencia en el archivo main.cpp, la función ya se encuentra declarada en el archivo main.cpp adjunto, solo debe escribir el código**) que resuelva el problema del vocabulario. Es decir se debe insertar las palabras en la tabla hash. Si la palabra a insertar no existe, inserta la palabra con un valor de 1, es decir la palabra tiene una cuenta de 1 en el vocabulario. Si la palabra existe, entonces debe incrementar una unidad su valor, debido a que significa que se encontró otra coincidencia para la palabra y por lo tanto su cuenta debe aumentar en 1.

6. Libere la memoria resevarda para la tabla hash con el arreglo dinámico.

Comando para Compilar Solución

g++ `-std=c++11` main.cpp HashTable.cpp -o main

Entrada de Ejemplo y Salida

Se adjunta un archivo main.cpp y un archivo llamado “words.txt”. Este archivo contiene la entrada de prueba y la salida debe ser parecida a la de la imagen. Note que la función para realizar la lectura de las palabras del archivo ya se encuentra adjunta en el archivo main.cpp.