Estructura de tabla hash (Hashtable)

Gustavo Gutiérrez-Sabogal

Introduction

A continuación encontrará una serie de ejercicios que tienen como propósito evaluar el uso y la comprensión de la estructura de datos Hashtable. Durante las clases hemos considerado todos los aspectos de esta estructura de datos, incluyendo:

- El problema que resuelven: almacenan eficientemente asociaciones del tipo llave valor. Es decir, elementos (llaves) que están asociados a valures.
- Su funcionamiento base interno: utilizan una función hash que tiene ciertas propiedades y bajo esas propiedades la estructura de datos garantiza su eficiencia. Adicionalmente esto trae como consecuencia la existencia de colisiones.
- Su implementación. En particular hemos hablado sobre las diferentes maneras de manejar las colisiones cuando ocurren. La implementación que realizamos en clase utilizaba listas enlazadas para almacenar las colisiones.

Ahora es el momento de evaluar el funcionamiento de la estructura de datos y contrastarlo contra otras estructuras que solucionan problemas similares. La idea es que las siguientes preguntas y ejercicios lo lleven a reflexionar en los diferentes aspectos de la estructura de datos. La solución a las preguntas y los ejercicios será evaluada de manera individual mediante una sustentación oral. En dicha sustentación se revisarán y evaluarán sus respuestas.

Listas de colisión

Primero es importante establecer una base de comparación. Para esto comenzaremos con el uso de listas de colisión para resolver las colisiones (*separate chaining*). Como lo vimos en clase, en lugar de almacenar directamente sobre el vector o arreglo los elementos, cada posición de este vector tendrá una lista enlazada.

Construya una tabla hash con un vector de $M = 5$ posiciones.Suponga la siguiente función hash $h(k) = k \bmod 5$ luego inserte los elementos de la siguiente secuencia: $\langle 10, 7, 12 \rangle$. Al final de la inserción la tabla quedaría de la siguiente forma.

Índice	Lista de colisión
0	$\langle 10 \rangle$
1	$\langle \rangle$
2	$\langle 7, 12 \rangle$
3	$\langle \rangle$
4	$\langle \rangle$

Ejercicio

Suponga ahora $M = 6$ y la función hash $h(k) = k \bmod 6$. Inserte la siguiente secuencia de elementos: $\langle 24, 14, 8, 32, 3 \rangle$. Proponga una representación tabular como en el ejemplo anterior.

- Observe la posición 2 del vector. Cuántas operaciones se requieren para encontrar el número 32?
- Cuantas operaciones se requieren para encontrar el número 100 que no se encuentra en la tabla?

Si en un caso dado, todas las llaves dan como resultado un mismo valor b, en qué estructura de datos se degrada la tabla hash?

Función hash determinística

Ahora consideraremos por qué la función hash asociada a una tabla hash debe ser determinística. Considere la siguiente función hash que solo funciona con llaves de tipo entero.

```
unsigned int hash(int key) {
  int random_salt = time(NULL);
  return (key * random_salt) % M;
}
```

Pregunta

Pruebe la función anterior al interior de la clase `Hashtable`. Desde el programa principal inserte y consulte el elemento insertado de la siguiente forma:

```cpp
#include <thread>
#include <chrono>
#include <string>
#include "hashtable.hh"

using namespace std;

int main() {
  Hashtable test;
  test.insert(10, "Hola");
  test.insert(12, "Mundo");
  this_thread::sleep_for(chrono::seconds(1));
  string s = test.get(10);
  cout << s << endl;
  this_thread::sleep_for(chrono::seconds(1));
  string t = test.get(12);
  cout << t << endl;
  return 0;
}
```

- Qué hace la función hash propuesta?
- Qué conclusiones puede sacar del ejercicio anterior?

Direccionamiento abierto

Sondeo Lineal (Linear Probing) es una técnica de resolución de colisiones utilizada en el Direccionamiento Abierto (Open Addressing). Cuando una función hash asigna una clave a un índice que ya está ocupado, el algoritmo busca el siguiente espacio disponible revisando los índices secuencialmente (índice + 1, índice + 2, etc.) hasta encontrar una celda vacía. Esta técnica trata la tabla hash como un arreglo circular, volviendo al inicio si llega al final de la tabla.

La Analogía del "Estacionamiento": Imagina que tienes un lugar de estacionamiento asignado. Si llegas y encuentras que tu lugar está ocupado por otro auto, no te vas a casa; simplemente tomas el siguiente lugar disponible a tu derecha.

Para implementar esta técnica deberá cambiar la implementación de la estructura de datos. Esta vez, en el arreglo o vector interno habrán elementos del tipo `pair<K, V>`.

```cpp
template <typename K, typename V>
class Hashtable {
private:
```

```
  unsigned int m;
  vector<pair<K, V>> storage;

  /* ... */
};
```

Esta técnica puede usar funciones hash ya existentes. Supongamos que existe una función que calcula $h(k)$. El índice donde se almacenaría el elemento con llave k será $i = h(k)$ si esa posición ya está ocupada entonces se tratará de insertar en $(i + 1) \bmod m$, si esa ya está ocupada entonces se intentará en $(i + 2) \bmod m$ y así sucesivamente.

> **Ejercicio**
>
> Realice la implementación de la clase `HashtableLP` que utiliza la técnica de *linear probing* para resolver colisiones. Compare su nueva implementación con la que usa separate chaining. Describa sus conclusiones.