Trabajo Práctico: Cuckoo Hashing

Enunciado

Desarrollar en c++ una tabla de hashing que utilice el mecanismo de resolucion de colisiones basado en Cuckoo Hashing, junto con otras librerias que utilicen esta tabla de hashing.
La construcción de estas librerías se realizará en 4 entregas incrementales.

Motivación

[Se verán en clase]: link1

Descripción del Dominio para la primer entrega

El dominio de la primer entrega corresponde

- Formato CSV y parseo
- Funciones de hashing sobre strings, en particular las funciones de hashing universal

Sobre HASHING:
En esta primer entrega deberá implementar un TDA que devuelva siempre una función de la familia hashing Universal que sea distinta (ver detalles mas abajo).

En general, una funcion de hashing h(x) es una operacion aritmetica que para cualquier instancia devuelve un numero entero de K-bits:

- h(x) -> numero(K bits), donde x podria ser cualquier instancia

Hashing es un tema amplio que veremos en la cursada, el dominio que deberá investigar para esta entrega es el de **funciones de hashing para claves de tipo string** (ver bibliografia al final del enunciado) y en particular las funciones de Hashing Universal

Record (Record.h)
El tipo de dato record representa un registro CSV. Es decir que debera contener una colecion de valores de los campos (campo en ingles es field)
Ademas de conocer los valores de los campos y tener primitivas para cargalos y accederlos, el TDA record tendra la posibilidad de conocer cual es “la clave” del registro. Para nuestra implementacion la clave puede ser un solo valor del registro (no admitiremos claves compuestas)

El concepto de clave se desarrollara en la clase de Ordenamiento, para el dominio de la primer entrega solo basta con saber lo mencionado en el parrafo anterior

CsvParser(CsvParser.h)
Este modulo implementa una funcion creacional de un vector de punteros a instancias de Record. Es decir que dado el contenido de un archivo CSV, crea las intancias Record correspondientes. La implementacion de este modulo es un parser CSV por lo que en la implementacion deberan utilizar muchas funciones de procesamiento de cadena de caracteres

UniversalHashing (UniversalHashing.h)
Una funcion de Hashing Universal es la siguiente:

```
int Hash( const char* key, int maximumHashCode) {
	int hashCode;
	int a = 31415;
	int b = 27183;
	for (hashCode = 0; *key != '\0';++key, a = (a * b) % (maximumHashCode - 1)) {
		hashCode = (a * hashCode + *key) % maximumHashCode;
	}

	return hashCode;
}
```

Este algoritmo corresponde a la categoria de “randomized algotrithm” porque para cada caracter va armando el hashing con un numero pseudorandom (a).
Ese numero a se define con un valor inicial (31415) y con un parametro extra (b = 27183).
Esos numeros son primos y se espera que sean de tamano mucho mayor del parametro maximumHashCode), con esas restricciones es que se logra construir un algoritmo de universal HASHING.
El objetivo de este TDA es que siempre que se llame a la primitiva creacional UndavUniversalHashing::Create() se devuelva una version distinta de este algoritmo (por ejemplo podria poner los parametros a y b en su estructura UndavUniversalHashing y hacer que estos sean distinto siempre…)

Descripción del Dominio para la segunda entrega

El dominio de la segunda entrega corresponde a las colecciones de Listas y diccionarios, tambien a sorting

RecordList

Es una lista especializada en items de tipo record.
Hay dos primitivas especiales:

- Sort: Debe ordenar la lista utilizando cómo clave la clave del item UndavRecord::GetKey)
- Reverse: Consiste en invertir la lista. Si por ejemplo la lista son los items 56 -> 12 -> 22 ->-1 ->NULL entonces la lista deberia quedar: -1 -> 22 -> 12 ->56 -> NULL luego de invocar esta primitiva

HashTable
Es un diccionario donde las claves son cadena de caracteres y el valor asociado a cada clave es un record. La implementacion debe ser una tabla de hashing con el metodo de resolucion de colisiones linear probing. Se adjunta una demo de la implementacion pedida

Un poco mas sobre el dominio de los diccionarios (se vera en clase en mas detalle):
Un diccionario (también conocido cómo Symbol Table) es una colección de pares clave-valor, es decir que permite almacenar items dónde cada item (valor) tiene asociada una clave. Las claves de un diccionario son únicas, es decir que no pueden haber claves duplicadas, y cada clave esta asociadas a un item esto significa que siempre que quiero agregar un item al diccionario tengo que dar la clave (ver la primitiva UndavHashTable::Add) y siempre que quiero obtener un item del diccionario tengo que dar la clave para buscarlo (ver la primitiva UndavHashTable::Get).
El nombre diccionario hace alución al clásico diccionario que se utilizaba en alguna época ya un poco remota… dónde la clave eran las palabras y el valor son el significado de cada palabra. Podemos decir que un diccionario es una generalización de un vector, puesto que en el caso del vector la clave seria el índice y el valor el item que se encuentra en el indice (vemos que para guardar un item en el vector necesito dar el indice y el valor, y para consultar un item necesito dar el indice).
Las implementaciones pueden ser

- Basada en listas desordenadas
- Basada en vector ordenado
- Basada en arboles (bst, RedBlack Tree, 2-3 tree)
- Basada en tablas de hashing (**es la que se pide**)

CsvParser(CsvParser.h)
Se modifica la interfaz. En lugar de devolver un vector de Record se devuelve la lista de record.

Descripción del Dominio para la Tercer entrega

La tercer entrega tiene cómo objetivo implementar la estrategia del metodo Cuckoo Hashing para resolver colisiones en su hash table. Se pedira implementar una variante (que tenga al menos dos funciones de hashing) y para esta 3er entrega se debera integrar el TDA grafo pero sin los algoritmos que chequeen los “pseudo arboles”

Cambios en HashTable

Luego de que estudie este metodo (puede utilizar el apunte de Argerich que esta en la bibliografia del enunciado) se pide lo siguiente

- Su estructura Hashing ahora no tendra una sola instancia de su TDA UniversalHashing, debera tener al menos dos
- Su estructura Hashing debera tener un grafo dirigido donde los vertices son numeros enteros que representan una posicion de su tabla de hashing (la cantidad de vertices es igual al espacio de direcciones) y las aristas representan la secuencia de posiciones en las que su dato (pajarito cuckoo) podria tener (Ver apunte).

TDA Grafo

El algoritmo de cuckoo hashing necesita un grafo dirigido para chequear de no entrar en un loop indefinido. Si bien no se pide implementar el chequeo de loop indefinido en esta entrega (eso sera en la entrega final) tendra que implementar el TDA grafo y su TDA Hashing debera ir modelando el grafo (realizando las conexiones).

Un poco sobre grafos:

Grafos es un tema que veremos en las siguientes clases. De hecho para la clase de grafos se pedira ver esta clase donde aqui esta el slide utilizado. Se recomienda ver esta clase para implementar este TDA

Matemáticamente un grafo **G** tiene dos conjuntos. El conjunto de vértices conocido como **V** (Vertexs) y el conjunto de aristas **E** (Edges) que son pares ordenados de elementos del conjunto V.

Por ejemplo. Supongamos un grafo de 6 vértices (numerados cada uno del 0 al 5) con las siguientes conexiones:

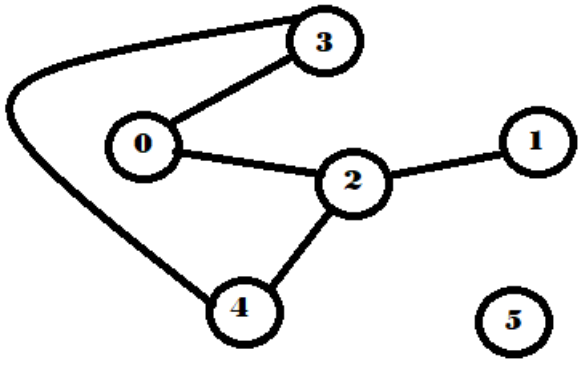
0-2 1-2 3-0 2-4 3-4

Define los siguientes conjuntos:

V:{0, 1 , 2 , 3 , 4, 5}

E:{(0,2) , (1,2) , (3,0) , (2,4), (3,4)}

Otra forma de visualizar el grafo anteriormente descrito es mediante el siguiente dibujo



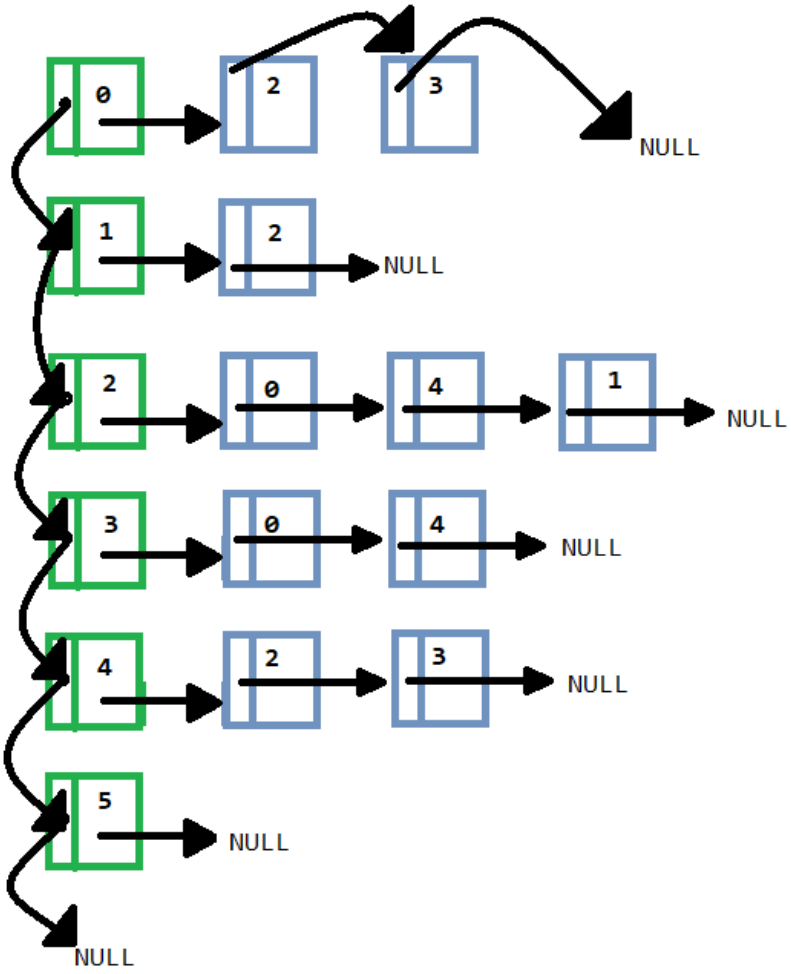
Definimos adyacencia en un grafo no dirigido cuando dos vértices están conectados por una arista. Es decir 3 y 0 son adyacentes (3 es adyacente a 0 y 0 es adyacente a 3, es una coneccion conmutativa) pero 3 y 2 no son adyacentes. Vemos que 5 no es adyacente a ningún vertice, esto se conoce como vértice aislado. Tambien se suele utilizar el término “vecino” en grafos no dirigidos como sinónimo de adyacencia (2 es vecino de uno).

A nosotros nos va a interesar implementar estos grafos en una estructura de datos. Las dos principales estructuras de datos son las conocidas como: matriz de adyacencia y lista de adyacencia.
La matriz de adyacencia es una matriz cuadrada de dimensión igual a la cantidad de vértices que tiene un valor distinto de cero cuando el vértice i es adyacente al vértice j.
Si representamos nuestro grafo en una matriz de adyacencia tendríamos la siguiente matriz.

	0	1	2	3	4	5
0	0	0	1	1	0	0
1	0	0	1	0	0	0
2	1	1	0	0	1	0
3	1	0	0	0	1	0
4	0	0	1	1	0	0
5	0	0	0	0	0	0

Y podemos identificar si el vértice i es adyacente al vértice j cuando es una valor distinto de cero. Ademas vemos que esta matriz siempre sera simétrica para un grafo no dirigido.

Otra forma de representar el mismo grafo en una estructura de datos es utilizando una lista de adyacencia. Donde hay dos tipos de nodos: El nodo de los vértices y el nodo de las adyacencias. El nodo de los vértices que tiene como dato el vértice y el primer nodo de sus adyacencia. El nodo de las adyacencias tiene como dato el vertice al que es adyacente al nodo del vértice.
Gráficamente, una instancia de esa estructura de datos para nuestro grafo es la siguiente.



La implementacion recomendada para realizar (por los algoritmos que se necesitan) es la de Lista de Adyacnecia. Una variante util seria una en la cual la lista que esta en el grafico de verde en lugar de ser una lista sea un vector (el item del vector una estuctura que tiene un numero entero y la lista azul o directamente el item del vector una lista dado que el indice del vector sera igual al entero qeu representa el vertice)

Criterio de aprobación de las entregas

Para que una entrega esté aprobada debe cumplir con todos los requerimientos y restricciones descriptos en cada entrega, sin ninguna excepción y en la fecha de entrega de la misma.
Cada entrega es una instancia de evaluación y se puede recuperar. La re-entrega o recuperación, se realiza en la fecha de entrega de la próxima entrega (no se realizan entregas en fechas intermedias)

Cómo realizo las entregas incrementales?

Cada entrega debe constar del codigo fuente (SOLAMENTE los archivos .h y .cpp, ningun otro tipo de archivo!!!!) comprimidos en un archivo (.zip, .rar) y se debe entregar por slack de la siguiente forma: Crear un chat con todos los integrantes del grupo y el docente, adjuntar la entrega en ese chat.

Entrega 1

- **Requerimientos:**
  - Implementar todos los TDA's, según los requerimientos especificados en los archivos cabecera
  - Su implementación debe ser correcta para todos los casos de uso de la interfaz descripta y su diseño de código debe ser claro y simple.
  - Puede utilizar cualquier libreria estandard de c y c++.
- **Restricciones:**
  - No se pueden quitar ni modificar las primitivas de los archivos cabecera

- **Fecha límite de entrega**
  - Martes 30 de Mayo (23:59 horas)

Entrega 2

- **Requerimientos:**
  - Implementar todos los TDA's, según los requerimientos especificados en los archivos cabecera
  - Su implementación debe ser correcta para todos los casos de uso de la interfaz descripta y su diseño de código debe ser claro y simple.
  - Puede utilizar cualquier libreria estandard de c y c++.
- **Restricciones:**
  - Ninguna

- **Fecha límite de entrega**
  - Domingo 20 de Junio (23:59 horas)

Entrega 3

- **Requerimientos:**
  - Implementar todos los TDA's, según los requerimientos especificados en los archivos cabecera
  - Su implementación debe ser correcta para todos los casos de uso de la interfaz descripta y su diseño de código debe ser claro y simple.
  - Puede utilizar cualquier libreria estandard de c y c++.
- **Restricciones:**
  - Ninguna

- **Fecha límite de entrega**
  - 5 de Julio (23:59 horas)

Bibliografía

[Sedgewick I] - Algorithms in C Third Edition Parts 1-4 , Robert Sedgewick. Capitulo 14 (Hashing)
[ApunteArgerich] Apunte del curso de una materia de FIUBA con contenidos de DataScience, Luis Argerich. En el capitulo 6.6 encontrara una introduccion sobre Cuckoo Hashing que es suficiente para enternder el dominio del problema.
[Sedgewick II] - El segundo tomo de Sedgewick aborda solamente grafos. Algorithms in C Third Edition Part 5 , Robert Sedgewick: Capitulo 17 y 18
Clase de grafos: https://youtu.be/b7O3Uu4Wz1E slides: https://docs.google.com/presentation/d/1R7BS_GUD8wp3Uh9_pwzn941OTpZfpjxKeKWCuIezHHI/edit#slide=id.p
[PaperCuckooHashing] Paper donde los autores comparten el metodo de Cuckoo hashing con la comunidad cientifica