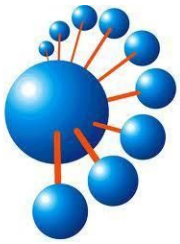


Universidad de Concepción

Departamento de Informática y Ciencias de la Computación

Proyecto II
Hashing

Nombre: Soledad Vásquez.
Roberto Avila
Asignatura: Estructura de Datos
Profesor: Diego Seco
Ayudantes: Paulo Olivares
Alexander Irribarra
Fecha: 08 de junio de 2018

Índice

Introducción

En computación a veces es necesario guardar datos que tiene una relación directa entre ellos , en donde un elemento tiene una conexión única , por ejemplo los alumnos en la universidad con su número de matrícula así la identificación de una persona matriculada en la universidad es directa siendo fácil reconocer luego su carrera , año de ingreso etc . Los mapas son estructuras útiles para almacenar ese tipo de información .

En este proyecto serán expuestas formas de implementar un mapa para luego ser analizar su rendimiento . son 3 implementaciones y luego dos de ellas tendrán variantes en cómo ordenan los datos.

La primera implementación es solo con vector simple en donde las inserciones serán ordenadas y la búsqueda de un par clave-valor es aplicada una búsqueda binaria .

Para las otras implementaciones se ocupará hash , las tablas hash son estructuras tipo vector que ayudan a asociar claves con valores o datos en donde la función hash proporciona una posición de inserción .Es la estructura preferida para la implementación de mapas.

El concepto clave no busca directamente el valor deseado si no a través de una función hash h(x) localizar la posición del valor buscado, en este proyecto se muestran 2 funciones hash , para manejar colisiones se ocuparan métodos como encadenamiento separado y Prueba Lineal que también serán comparados .

Descripción de las soluciones

Se implementará un Tipo Abstracto de Datos (ADT), llamado Map, proporciona las siguientes operaciones put (pair<string,int>), get(string), remove(string), isEmpty(), size().

put: Ingresa un par clave valor al mapa
get: Busca en el mapa si está el string que es ingresado como argumento de entrada y retorna el valor asociado a él.
remove: Elimina el par clave-valor del mapa asociado al string.

Se implementará tres estructuras de datos para Map: La primera basada en un vector con búsqueda binaria, la segunda basada en Hashing con mecanismo de resolución de colisiones de prueba lineal y encadenamiento separado.Para cada implementación de Hash hay dos variantes de la función hashing una que toma el código ascii del primer carácter del string ingresado operado con el módulo del tamaño de la estructura retornando un valor que se traduce en una dirección de insertado y la otra función de hash utilizada la acomulación polinomial esta basa en tomar cada código ascii de los caracteres del string para luego elevar a la potencia de la posición en que están .

Para el tamaño de la tabla de cada una de estas implementaciones se tomó la decisión de que usuario ingresa el número de pares clave-valor a insertar y cada implementación calcula el tamaño de tabla respecto a su factor de carga máximo , el factor de carga es una relación entre la cantidad de elementos insertado respecto al tamaño proporcionado la densidad que tienen los datos.

Vector con Búsqueda Binaria

ingresa pares clave-valor de forma ordenada en un vector , luego para realizar la operación get ocupa búsqueda binaria para localizar el elemento con su valor .

Prueba Lineal

En esta implementación consiste en ingresar pares en un arreglo circular , el valor del par indica la posición donde comienza la inserción , como las funciones de hash repiten valores para distintas claves existen colisiones en las inserciones ya que distintos valores deben ser insertados en la misma posición , esto se soluciona recorriendo secuencialmente a partir del punto de colisión hacia la derecha hasta encontrar una posición vacía o disponible para insertar par evitando así sobrescritura de pares clave-valor .Una posición disponible significa que

anteriormente hubo algo insertado pero se eliminó y para mantener la consistencia del mapa se llena ese hueco con disponible.

Para mantener la dispersión de los datos ocupa un factor de carga máximo de 0.5 esto es así ya que si la dispersión fuera mayor, el costo de recorrer el arreglo si hubiera una colisión puede ser muy alto y por tanto disminuye su eficiencia.

Encadenamiento separado

Esta implementación también consiste en ingresar pares clave-valor a una estructura .La técnica de encadenamiento, para manejar las colisiones en cada casilla del array cita una lista de los registros insertados que colisionan en la misma posición . La inserción consiste en encontrar la casilla correcta y agregar al final de la lista correspondiente. El borrado consiste en buscar y quitar de la lista.

Para mantener la dispersión de los datos su factor de carga va a ser menor a 0,75, ya que si aumenta el factor las colisiones aumentan y por consiguiente las operaciones comienzan a ser ineficientes.

Ventajas y desventajas de utilizar Prueba lineal o Encadenamiento separado

	Ventajas	Desventajas
Prueba Lineal	1)tiene el mejor rendimiento 2)se puede utilizar incluso si una entrada no se asigna a ella.	1)Este método puede haber un fuerte agrupamiento alrededor de ciertas claves, mientras que otras zonas del arreglo permanecerán vacías. 2)requiere un cuidado especial para evitar el agrupamiento y el factor de carga.
Encadenamiento Separado	1) Simple de implementar. 2) La tabla hash nunca se llena, siempre podemos agregar más elementos a la cadena. 3) Menos sensible a la función hash o factores de carga. 4) Se usa principalmente	1) El rendimiento de caché de encadenamiento no es bueno ya que las claves se almacenan usando la lista vinculada. 2) Desperdicio de espacio (algunas partes de la tabla hash nunca se usan) 3) Si la cadena se vuelve larga, el tiempo de búsqueda puede

	cuando no se sabe cuántas y con qué frecuencia se pueden insertar o eliminar las claves.	convertirse en O (n) en el peor de los casos. 4) Utiliza espacio extra para enlaces.

Detalle de implementación

Aquí se expone lo necesario para implementar el código a fin de poder realizar una lectura ágil y a su vez para facilitar la lectura.

Archivo de referencia: ADTmap.h

Declaración de la case ADTmap que se realiza solo las especificaciones en: public que permite el acceso a tal término desde dentro y fuera de la clase y por otro lado el private que permite que sean accesibles por los propios miembros de la clase.

Prueba Lineal:

Archivo de referencia: MapH_LP_B.cpp, MapH_LP_PA.cpp

La implementación se compone de un arreglo que contiene punteros a pares clave-valor , las clave es representada por un string y el valor por un entero, además la estructura contiene 2 variables globales de tipo entero , una que indica la cantidad de pares ingresados en arreglo (n) y la otra indica el tamaño del arreglo(m).

En las estructuras que implementan la prueba lineal todos los métodos a excepción de la hash están implementados del mismo modo. por consiguiente los archivos .cpp contienen los siguientes métodos:

Constructores: Los constructores calculan el tamaño del arreglo con un factor de carga ,además son inicializadas todas las posiciones del vector en NULL,así todas las posiciones son consideradas vacías .

Destructores: Se encargan de eliminar todo el contenido del array, esto se hace secuencialmente , haciendo delete a cada referencia contenida en la tabla.

size(): retorna la cantidad de pares insertados

isEmpty(): retorna verdadero si la estructura no contiene elementos

put(pair<string , int> v): en este método se realiza el ingreso del par a la estructura , primero se hace hash de la clave que indica la posición de inserción,

luego se ocupa get para saber si la clave ya está insertada si no es así , procede con la inserción , si la posición de inserción está ocupada se desplaza a la derecha en el arreglo hasta encontrar una posición con valor NULL o DISPONIBLE para ingresar el elemento , luego de encontrar una posición válida se realiza la inserción creando un par que es referenciado por la posición .

get(string clave): Como argumento de entrada recibe un string que es buscado en la estructura , primero hace hash del string en donde obtiene la posición de búsqueda , si no se entra la clave avanza a la derecha hasta que encuentra el valor y como el arreglo es tratado de forma circular la búsqueda se termina cuando llega hasta la posición inicial o hasta que se encuentra el string buscado en donde se retorna el valor asociado a la clave.

remove (string clave): es idéntico a lo que hace la función get , pero cuando encuentra la clave buscada , el par asociado a la posición en donde está la clave , pasa a ser el par DISPONIBLE con valor -1, luego es retornado el valor asociado a la clave. por último si el elemento no es encontrado es retornado -1.

Las funciones de hashing son implementadas como un método privado que retorna un entero que se traduce en una ubicación para comenzar la inserción .

Encadenamiento Separado:

Archivo de referencia: MapH_SC_B.cpp

Archivo de referencia: MapH_SC_PA.cpp, MapH_SC_PA.cpp

La implementación está compuesta de un arreglo que contiene en cada posición una referencia a una lista enlazada de pares , la lista se arma con una secuencia de nodos que contiene el par y un apuntador al siguiente nodo. los métodos que contiene esta estructura son los siguientes :

Constructores : declara un arreglo de punteros de tamaño m , el tamaño es calculado por el factor de carga , todas las posiciones del array se inicializan en NULL .También es declarado una variable entera que registra cuántos pares han sido ingresados .

Destructor: Elimina de cada celda de arreglo las listas asociadas , las listas son eliminadas nodo a nodo y por último es removida el arreglo.

size(): retorna la cantidad de pares ingresados en la estructuras.

isEmpty(): retorna verdadero si la cantidad de pares ingresados es 0.

put(pair<string , int > v): se encarga de hacer efectiva la introducción del par a la estructura , primero hace hash de del string que indica donde debe posicionarse

pareja clave valor , si no hay una lista previamente en el lugar de inserción sólo se inserta el par creando un nodo el cual registra la clave y el valor. Sí hay una lista previamente busca si el valor ya está en la lista para asegurar que no se inserten valores repetidos , luego de revisar que no esté es insertado al final de lista.

get(string clave): se encarga de buscar si está insertada la clave que es el argumento de entrada , primero hace hash de clave y se sitúa en la posición indicada por el hash , por consiguiente se dispone a recorrer la lista insertada en esa ubicación , hace una comparación a por cada insertado y sí coincide que está retorna el valor afiliado con la clave, en cambio si la clave no es encontrada es retornado el valor -1

remove(string clave): realiza un procedimiento parecido a get y al momento de encontrar la clave elimina el par clave-valor de la lista asociada , también retorna el valor de la clave removida .

Las funciones de hashing son implementadas como un método privado en donde la función actual y retorna una valor.

Análisis Teórico

En el análisis teórico de peor caso podemos ver que al implementar con :

Prueba Lineal:

- Put O(n)
- get O(n)
- Remove O(n)

Encadenamiento separado:

- Put O(n)
- Get O(n)
- Remove O(n)

Vector Ordenado:

- Put O(n)
- Remove O(n)
- Get O(log n)

Utilizando con factor de carga y re-hashing a la mejor implementación

Suponiendo que los valores de hash son como números aleatorios y su factor de carga es <1 , se puede demostrar que el número esperado de pruebas para una inserción,borrado y buscar con direccionamiento abierto es 1/ (1 -a).

Análisis de espacio

- Encadenamiento separado utiliza espacio extra, porque crea listas por cada posición .
- Prueba lineal no usa espacio extra, todo se ingresa directo a la tabla .

Análisis Experimental

Gráficos de datos con sus respectivos tiempo para Put, get y remove :

Para el análisis las estructuras ocupan su mayor factor de carga permitido, por lo que para encadenamiento separado se utiliza un factor de carga de 0.75 y para prueba lineal se utiliza 0.5

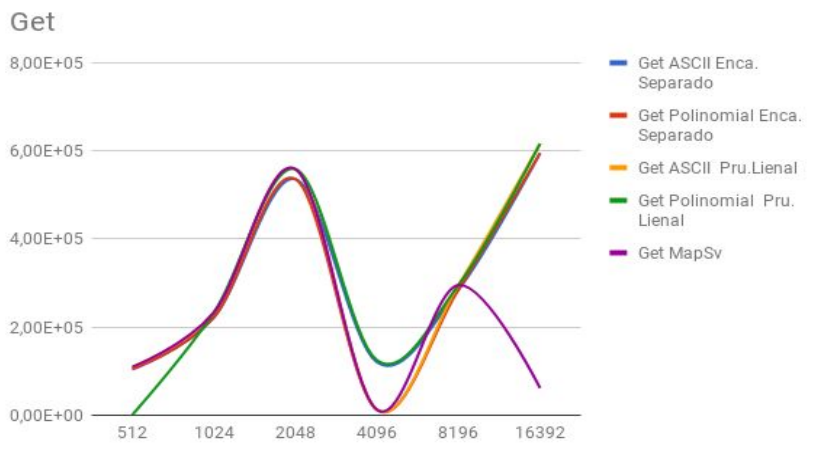


Imagen 1: Comparación de tiempo entre prueba lineal y Encadenamiento separado con hash ASCII y acumulacion polinomial.

En la operación get no se aprecia una gran diferencia entre los métodos a excepción de MapSV que se demora más que todos por tanto es el menos eficiente , pero para tomar una decisión los datos no permiten tomar una decisión concreta .

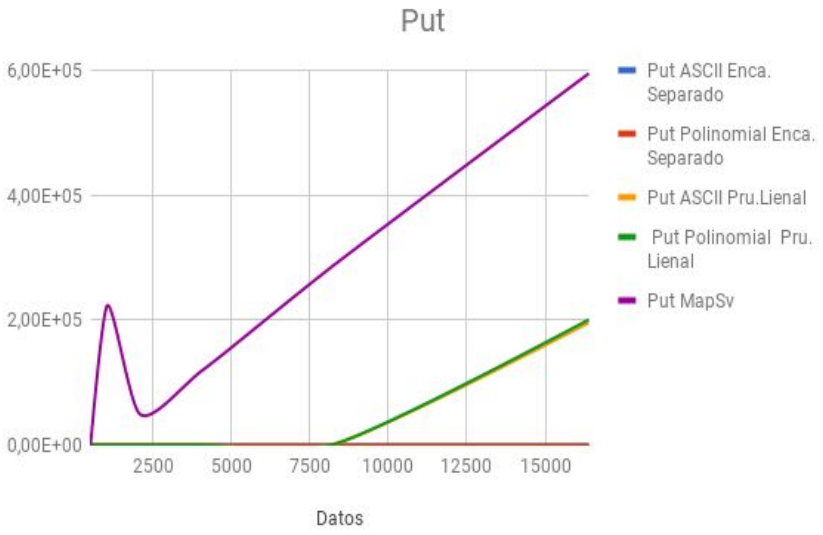


Imagen 2: Comparación de tiempo entre prueba lineal y Encadenamiento separado con hash ASCII y acumulacion polinomial.

se puede apreciar que el encadenamiento separado se demora menos en promedio en realizar el put al aumentar los datos que se insertan en la estructura, su comportamiento casi es constante por lo que la implementación es la más eficiente

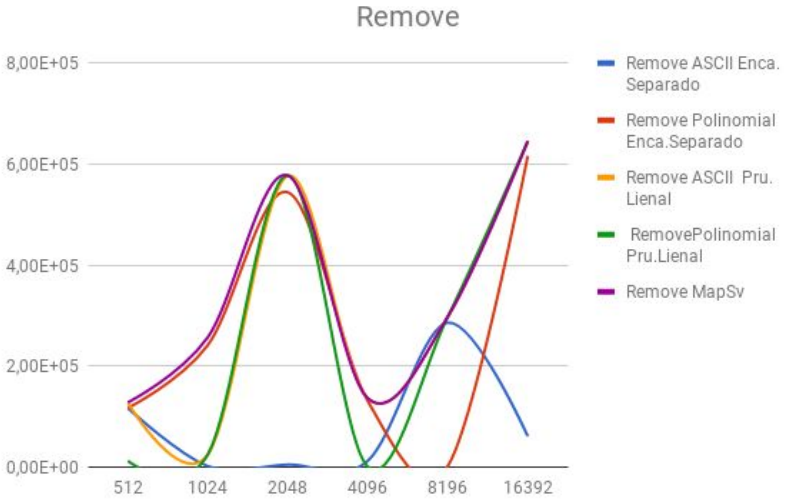


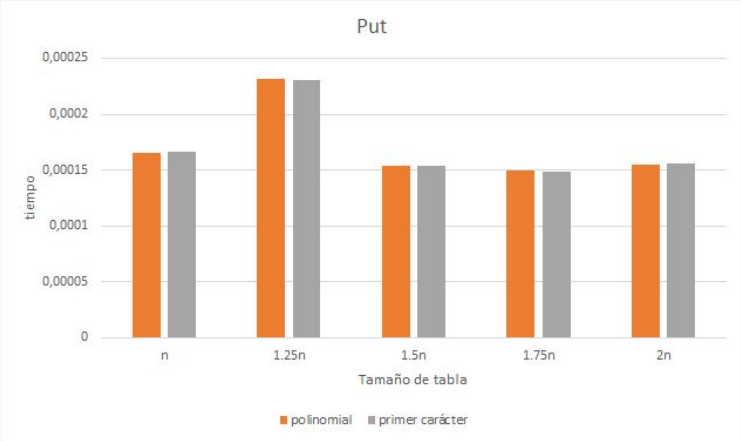
Imagen 3: Comparación de tiempo entre prueba lineal , Encadenamiento separado con hash ASCII y acumulacion polinomial.

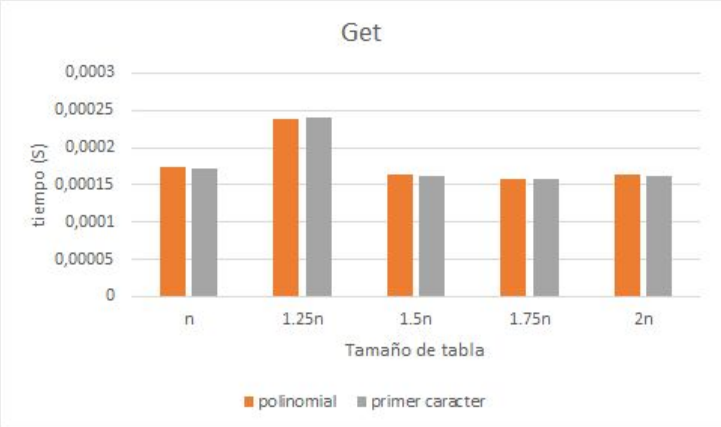
se puede apreciar que el remove en encadenamiento separado toma menos tiempo al aumentar los datos por lo que es más eficiente que las demás

En resumen en el único método en donde se puede apreciar una diferencia de tiempo notable es en encadenamiento separado, en conclusión la estructura es más eficiente al estar ocupando el máximo factor de carga comparado con las demás implementaciones .

Análisis de tamaño de tabla

Para el análisis se escogió la implementación más eficiente en este caso fue encadenamiento separado , se ha experimentado con ambas funciones de hash.





Al observar se puede apreciar que cuando el tamaño es 1.25 n veces más grande que la cantidad de datos el tiempo es mayor y la tabla maneja las operaciones más lento , esto es porque se acerca al factor de carga máxima (0,75) , en cambio cuando el tamaño de tabla se amplía la cantidad el tiempo disminuye porque los datos están más dispersos y se acercan al factor de carga mínimo)0.5)

Conclusión

Una tabla hash tiene como principal ventaja que el acceso a los datos suele ser muy rápido por lo que es útil para ser utilizada para implementar mapas eficientes.

Si tenemos una función hash en una matriz que puede contener pares clave-valor, entonces necesitamos una función que pueda transformar cualquier clave dada en un índice en esa matriz. Como requisito para que sea una buena función hash las claves iguales deben producir el mismo valor hash,ser eficiente para calcular y distribuir uniformemente las llaves.

Al igual que con el encadenamiento separado, el rendimiento de los métodos de prueba lineal depende de la relación (Factor de Carga) $\alpha = N / M$, pero lo interpretamos de manera diferente. Para encadenamiento separado, α es el número promedio de elementos por lista y generalmente es mayor que 1. Para la prueba lineal, α es el porcentaje de posiciones de tabla que están ocupadas; que debe ser inferior a 1.

En conclusión es mejor el encadenamiento separado, por que la inserción consiste en encontrar la casilla correcta y agregar al final de la lista correspondiente. El borrado consiste en buscar quitando de la lista asociada a la posición .

El crecimiento de la tabla puede ser pospuesto durante mucho más tiempo dado que el rendimiento disminuye mucho más lentamente incluso cuando todas las casillas ya están ocupadas por otro lado es sencillo de implementar.Pero prueba lineal ofrece un mejor rendimiento en memoria(caché).

Finalmente se puede experimentar con los factores de carga para obtener una buena eficiencia y al estudiarlos anticipar cual es mejor para nuestras necesidades.

Bibliografía

Software Projects- Sunil Savanur, Design and Development,(2018),Polynomial Hash Function, recuperado el 2013, de

https://sunilsavanur.wordpress.com/2012/08/14/polynomial-hash-function-for-dictionary-words/comment-page-1/#comment-54