

TAREA 2: Análisis de Algoritmos

Profes. Cecilia Hernández

Algoritmos aleatorizados

La fecha de entrega es el lunes 27 de mayo a las 23:59 horas.

1. En esta tarea se implementará y analizará un algoritmo aleatorizado de tipo las vegas para almacenar estáticamente un conjunto de elementos usando *Hashing perfecto*.
2. Para la implementación considere los siguientes puntos.
 - a) Asumir un conjunto S de n elementos a almacenar.
 - b) Usar la siguiente familia de funciones hash universales. Donde p es un número primo, m el tamaño de la tabla y a y b son valores aleatorios en el rango $[0..p-1]$ y m es el número de buckets en la tabla.
$$h(k) = ((a \times k + b) \bmod p) \bmod m$$
 - c) Conjunto S se almacena en tablas de dos niveles usando hashing perfecto, el cual usa un algoritmo aleatorizado del tipo las vegas, según como se explica a continuación:
 - 1) Tabla de primer nivel: Para determinar función hash, h , de primer nivel. Elegir $m = n$, $p > |S|$ o bien $p > \max(S)$. Elegir a y b aleatoriamente en un rango $[0..p-1]$.
 - 2) Test h de la siguiente manera:
 - Aplicar h a cada clave en S , y para cada bucket i en la tabla mantener un contador c_i que cuente el número de elementos que se asignan a bucket i .
 - Si se cumple $\sum_{i=0}^{m-1} c_i^2 < 4n$, entonces la función hash h es buena, si no, tomar otro a y b aleatorio y repetir hasta que sea buena (condición las vegas).
 - Una vez encontrada la función hash h , se construyen los conjuntos S_i , que son los subconjuntos de S que se asocian a cada bucket i en la tabla de primer nivel aplicando

h . Dado que $c_i = |S_i|$, cada tabla de segundo nivel se define de tamaño $m_i = c_i^2$. Teniendo m_i para cada tabla de segundo nivel se procede a encontrar las funciones hash de segundo nivel (h_i), cada una asociada a cada bucket i de la tabla de primer nivel.

d) Tablas de segundo nivel.

- Cada bucket i en la tabla de primer nivel tiene un m_i , ahora para calcular el h_i asociado a bucket i considerar $p_i = p$ y entonces sólo se necesita determinar a_i y b_i .
- Elegir a_i y b_i aleatoriamente hasta encontrar valores de a_i y b_i de manera que el número de elementos en S_i no colisionen en $tabla_i$ y se almacenan los elementos de S_i .
- Una vez encontrados los valores de a_i y b_i para h_i , se almacena la tripleta a_i, b_i, m_i asociado al bucket i en la tabla de primer nivel. La tripleta a_i, b_i, m_i determina la función hash h_i asociada al bucket i .

e) Una vez construida la tabla hash con hash perfecto se pueden buscar los elementos de la siguiente manera:

- Dado un x a buscar, se aplica $h(x)$ y se obtiene el bucket y en tabla de primer nivel. El bucket y tiene almacenado la tripleta (m_y, a_y, b_y) asociada a h_y , luego se aplica $h_y(x)$ y se encuentra el elemento que se busca en la tabla de segundo nivel correspondiente.
- Si se busca un elemento que no pertenece a S puede ser producto de una colisión así que es necesario verificar el valor almacenado.

3. Para la implementación, puede elegir entre C, C++, Java y Python.

4. Contenidos mínimos del informe:

- a) Breve descripción de las estructuras de datos implementadas.
- b) Pseudocódigo de todas las operaciones implementadas.
- c) Análisis de número esperado de colisiones para hashing perfecto. Explique por qué se necesita hacer $m_i = c_i^2$.
- d) Análisis de complejidad de tiempo esperado para todas las operaciones.

- e) Evaluación experimental de las operaciones *construir* la tabla y *búsqueda* de elementos. Esta evaluación debe incluir gráficas donde en el eje X se representa el tamaño de las tablas y en el eje Y el tiempo que toma la operación de búsquedas aleatorias. Además, se debe mostrar un gráfico donde se muestre el número promedio de veces que se necesita ejecutar la búsqueda de los coeficientes a y b para la determinar la función hash h .
- f) Discusión acerca de los resultados obtenidos.

La entrega consiste en un informe de no más de 5 páginas y el código fuente documentado. Una vez entregado el informe y revisado, nos reuniremos para conversar de la tarea en una hora acordada.

Observaciones

- El capítulo 11 del Cormen tiene una sección con Perfect Hashing y un ejemplo.
- El apunte de algoritmos de Gonzalo Navarro disponible en Infoda contiene el algoritmo descrito acá. Tener cuidado que la variable b_i en el apunte esta sobrecargada porque se usa en dos contextos distintos uno para el coeficiente de la función hash y otro para el tamaño de los elementos de S_i (aquí se usa c_i para el tamaño de S_i)