Tarea 9: Análisis de algoritmos

Dalia Camacho, Gabriela Vargas, Elizabeth Monroy

Noviembre 2018

1 Estructuras van Emde Boas, vEB

La estructura de datos de van Emde Boas representa una idea básica del algoritmo divide y vencerás. Dada una estructura de datos de u elementos, $S = 0, 1, \ldots, u-1$ en la que es posible aplicar diversas operaciones tales como:

- insertar(x),$x \in S$
- borrar(x), $x \in S$
- mínimo(x) y máximo(x), regresa el mínimo y máximo de S
- sucesor(x) regresa el elemento más chico en S mayor que x
- predecesor(x) regresa el elemento más grande en S menor que x

La estructura de datos de **van Emde Boas** puede realizar estas operaciones eficientemente. Dado un vector V de tamaño u tal que $V[x] = 1$ si y solo si $x \in S$. Insertar o borrar un elemento solo requiere de ubicar el correspondiente bit, en el vector; sin embargo encontrar un sucesor o un predecesor implica recorrer el vector para encontrar el siguiente 1-bit.

- Insertar/Borrar es de orden $O(1)$
- Sucesor/Predecesor es de orden $O(u)$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	1

Figure 1: Muestra un Bit vector de tamaño $u = 16$, y un conjunto $1, 9, 10, 15$

Divide el universo en *clusters*, es decir, dividimos el rango $0, 1, \ldots, u-1$ en $/rootu\ clusters$. Si $x = i\sqrt{u} + j$, entonces $V[x] = V.Cluster[i][j]$

- $low(x) = xmod\sqrt{u} = j$

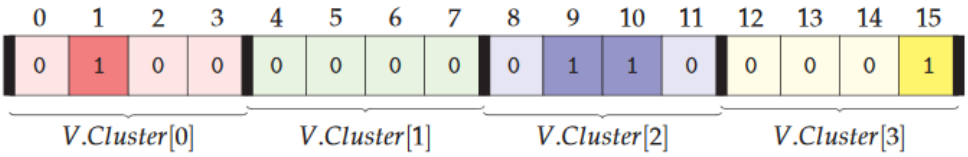

Figure 2: Se obtienen cuatro *clusters* de tamaño 4

- $high(x) = x/\sqrt{u} = i$
- $index(i, j) = i\sqrt{u} + j$

Insertar

Dado un Bit vector V.Cluster $[high(x)][low(x)] = 1$ es de orden $O(1)$ Marcar el cluster $high(x)$ como no vacío es de orden $O(1)$

Sucesor

1. Buscar el cluster en el que se encuentra el mayor número mayor que x high(x)
2. Sino, encuentra el cluster no vacio i
3. Encuentra la mínima entrada j en el cluster
4. Regresa index(i,j)

Estos pasos son de orden $O(\sqrt{u})$ y el total de los pasos también es $O(\sqrt{u})$ Para acelerar el algoritmo dado que las operaciones llaman a un vector de tamaño$\sqrt{u}$. Se puede realizar un proceso recursivo. V.cluster[i] es una estructura de tamaño $\sqrt{u}$ van Emde Boas $\vee\ 0 < \sqrt{u}$ V.summary[i] es de tamaño $\sqrt{u}$ en una estructura van Embe Boas V.summary[i] indica si V.cluster[i] es no vacía

Para realizar la operación de inserción el mínimo se almacena en V.min. Para verificar si la estructura está vacía, si se encuentra el mínimo.

```
insert(V, x = <c, i>):
    if x > V.max:
        V.max = x
    if V is empty:
        V.min = x;
        return;
    if x < V.min: swap(x, V.min)
    if V.cluster[c].min == null:
        insert(V.summary, c)
    insert(V.cluster[c], i)
```

Cuando hacemos una llamada recursiva, en un conjunto de recurrencias $T(u) = T(\sqrt{u}) + O(1)$,lo que implica que $T(u) = O(\log \log u) = O(\log w)$.

Para la operación de inserción se hacen 2 llamadas recursivas en el peor de los casos cuando el cluster está vacío. Por lo que tenemos la siguiente expresión: $T(u) = T(\sqrt{u}) + O(1)$. Entonces, $T(u) = O(\log \log u)$ Mientras que en términos de estructuras vEB, tenemos q $S(u) = (\sqrt{u} + 1)S(\sqrt{u}) + O(1)$, lo que implica que $S(u) = \Theta(u)$.

2 Algoritmo de Huffman y Entropía

En teoría de la información, el concepto de **entropía** se entiende como la cantidad de información contenida en una variable. En el contexto del algoritmo de compresión de Huffman, la entropía nos indicaría la cantidad de información promedio que contienen los símbolos usados. Los símbolos con menor probabilidad son los que aportan mayor información; por ejemplo, en un sistema de símbolos que consiste en las palabras de un cuerpo de correo, los conectores *que*, *el*, *a* aportarían poca información sobre el contenido del mismo, mientras que palabras menos frecuentes como *cita*, *acuerdo* o *reunión* nos darían una mejor idea. Cuando todos los símbolos son igualmente probables (con distribución de probabilidad plana), todos aportan información relevante y la entropía es máxima.

El algoritmo de Huffman se considera como un *codificador óptimo*, ya que utiliza el mínimo número de bits para codificar un mensaje. Un codificador óptimo usará códigos cortos para codificar mensajes frecuentes y dejará los códigos de mayor longitud para aquellos mensajes que sean menos frecuentes. De esta forma se optimiza el rendimiento del canal o zona de almacenamiento y el sistema es eficiente en términos del número de bits para representar el mensaje.

En este caso, la entropía denotaría el mínimo número de bits por símbolo necesarios para representar una cadena. De esta forma, se puede cuantificar la cantidad de información que existe en una fuente de datos (la cadena a codificar). Su definición matemática es la siguiente:

$$H = \sum_{a_i \in A} P(a_i) log_2 \frac{1}{P(a_i)} \tag{1}$$

Considerando como ejemplo la cadena de símbolos $S = \{aabaacc\}$ y el alfabeto $A = \{a, b, c\}$, la probabilidad de cada uno de los símbolos vendrá dada por las siguientes expresiones: $P(a) = 4/7, P(b) = 1/7 y P(c) = 2/7$.

Siguiendo la definición anteriormente mencionada, la entropía de este sistema de símbolos sería la siguiente:

$$\begin{aligned} H &= P(a) log_2 \frac{1}{P(a)} + P(b) log_2 \frac{1}{P(b)} + P(c) log_2 \frac{1}{P(c)} \\ H &= \frac{4}{7} log_2 \frac{7}{4} + \frac{1}{7} log_2 7 + \frac{2}{7} log_2 \frac{7}{2} = 1.38 \end{aligned} \tag{2}$$

Siendo 1.38 la cantidad mínima de bits necesarios para cada símbolo del sistema.