Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación
ICC2133 – Estructuras de Datos y Algoritmos – Sección 2
Profesores Yadrán Eterovic y Cristobal Gazali
Ayudante Jefe Trinidad Vargas
Primer Semestre 2021

Alumno: José Antonio Castro

Informe Tarea 2:
Hash de Walter White

0.1 ¿Como usar un diccionario para responder esto de forma eficiente?

Para responder esto de forma eficiente, se usa un diccionario para agrupar las subimagenes de la imagen original. Esto se hace para que, por ejemplo, al hacer hacer una consulta (como la de encontrar insectos), no se deba iterar sobre todas las subimagenes existentes, sino que simplemente solo por las que esten en el valor (casilla) del diccionario según cierta llave. Para calcular como distribuir las subimages en el diccionario se utiliza una funcón de hash.

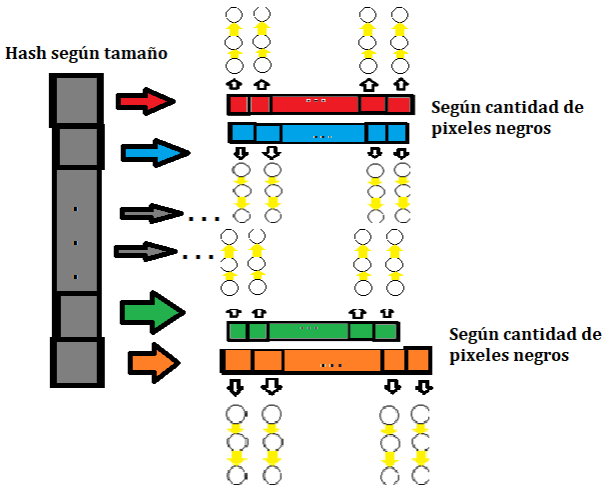
La idea de la función de hash es que es algo determinista, recreable y relativo a la subimagen. Tambien es útil que sea incremental para no repetir todos los cálculos. Con una buena función de hash se pueden distribuir los datos de forma uniforme y sin colisiones (es decir, para más de una clasificación se le asigna las misma casilla del diccionario).

Ahora bien por que usar diccionarios es hacerlo de forma eficiente. Como ya dijimos no iteramos sobre todos las subimagenes sino que solo sobre las que caben en cierta clasificación (iteramos sobre una lista ligada de estas subimagenes), ¿pero como identificamos rápidamente a que clasificación pertenece? Al momento de buscar en el diccionario la complejidad de llegar al valor si tenemos llave es de ni más ni menos que $\mathcal{O}(1)$ y de ahí es el porque es tan optimo usar dicionarios distribuidos según funciones de hash.

0.2 Función de hash uniforme para una imagen (n × n) incremental

Para resolver esto mi implementación será crear 2 fuciones de hash. La primera clasificará según tamaño de subimagen y la segunda según cantidad de pixeles negros. Tendré un arreglo de arreglos, donde la distribución del más grande tendrá en su valores a los arreglos que clasifican según número de pixeles negros. Este valor corresponderá a una subimagen que tiene a su vez una lista ligada de subimagenes pertenecientes a esta clasificación. De esta forma con solo 2 llaves se puede llegar a las subimagenes posibles en donde estaría el insecto de Heisenberg (debido a que con el tamaño y cantidad de negros es algo muy simple de calcular y con ello reduzco mi espacio de búsqueda enormemente).

A continución hay una imagen de como sería:



Ahora bien, ¿como hacer a esta función de doble hash incremental? Para ello primero veamos cual es el calculo que se puede hacer incremental. El cálculo que hacemos es la cantidad de pixeles negros por subimagen y no es tan inteligente calcular esto siempre todo el tiempo para cada caso. Es por ello que lo que haremos nosotros es que solo se calcule la información nueva y la obsolta para eliminarla de nuestros datos.

Ej: Tengo la subimagen de tamaño 3×3 y que su origen (esquina superior izquierda) esta en el pixel 0. Para calcular la siguiente subimagen, es decir, la 3×3 que su origen se ubique en el pixel 1. No haremos todo el calculo otra vez, sino que solo quitaremos la primera columna y agregaremos la nueva. De esta forma nos saltamos calculos de más.

Es por ello que nuestra función es incremental a la hora de moverse a los lados cuando se calculan las de cierto tamaño. Reduce la complejidad de a la hora de hashear de forma notoria.

0.3 Demuestres que tu función de hash es uniforme

La uniformidad de mi función de hash depende de la cantidad de pixeles que tengan la imagen original. Para una imagen muy chica mi distribucuón raramente será uniforme, pero mientras más grande sea y más distribuida este más uniforme será. Si se apróxima al infinito de una forma asintótica, la tendencia a la uniformidad ira creciendo.
Ahora bien ese hash en uniforme en ciertas condiciones y no siempre. La distribución de datos debe ser más o menos regular, sino (como cualquier funcion de hash en verdad) si son por ejemplo todos los pixeles negros o todos blancos, habrán valores más concentradas que otros.

0.4 Elección de parámetros

Factor de carga

Esto habla de que tan llena esta mi tabla, en mi caso nunca se llenará, ya que en caso de haber más de una subimagen que tiene misma cantidad de pixeles y de pixeles negros se anexa como lista ligada al detrás de la ultima subimagen. Aparte de esto nunca se llenarán todas las casillas ya que matemáticamente es imposible que de esta forma hayan subimagenes de todos los tamaños y con toda la cantidad de píxeles negros diferentes. Para un caso de encadenamiento como este, el factor de carga es un número entero cercano a 1.

Resize

Para mi función de hash no es necesario, ya que mi factor de carga no es $>> 1$ y también porque ya hago una función fija desde el principio para que se adapta según el tamaño de la imagen completa y no necesita arreglos de más post construcción.

Direccionamiento

Uso direccionamiento cerrado, es decir, encadenamiento.

Probing

La forma que tiene mi algoritmo para lidiar con colisiones es encademamiento.