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 1:
Max Tree

Component Tree v/s Max Tree

Supongamos que tenemos una imagen en escala de grises. Un umbral superior ($\geq$) da como resultado una imagen binaria, donde cada isla de resultado se denomina componente conectado (CC).

Cuanto mayor sea el valor de umbral, menor será el tamaño de los CC, es decir, existe una relación de inclusión entre los componentes conectados obtenidos en diferentes umbrales valores. El árbol de componentes representa una imagen a través de esta propiedad jerárquica de la descomposición del umbral. Para cada CC resultante de cada posible umbral de la imagen, hay un nodo de árbol de componentes y este nodo almacena todos los píxeles del componente conectado. Esto da como resultado una redundancia representación, ya que la mayoría de los píxeles pertenecerán a más de un nodo. Además, hay CC que no cambian de uno umbral al otro, por lo que es redundante utilizar más de un nodo para representarlos. El max-tree es un compacto representación para el árbol de componentes, sus nodos almacenan sólo la píxeles que son visibles en la imagen en el nivel de gris correspondiente, por lo tanto, CC que permanecieron iguales durante una secuencia de Los umbrales están representados en un solo nodo llamado compuesto.

Pensamos en términos del árbol de componentes cuando diseñar métodos y aplicaciones, pero procesamos utilizando el max-tree, ya que es una estructura más compacta y eficiente. Esa es la razón por la que muchos autores tratan los árboles componentes y max-trees como sinónimos.

0.1 Eficiencia

Es mucho más eficiente el max-tree, ya que no repite los pixeles en cada nivel, sino que solo los referencia una vez y a un nodo. En cambio en un component tree los asocia a cada vecindario con umbral menor al del pixel.

0.2 Ventajas de cada uno

- Max-Tree: Es mucho más eficiente ya que almacena y referencia todos los pixeles solo una vez. A la hora de crear y de iterar para filtros es más eficiente.
- Component-Tree: Al momento de ver las cosas de cada nivel, tienes al toque todos los pixeles de ese umbral o superior. Por lo que puedes afectar a mayor (o igual) cantidad de pixeles a la vez.

0.3 Desventajas de cada uno

- Max-Tree: A la hora de querer ver todos los pixeles que superen cierto umbral, deberás recorrer todos los nodos hijos del nodo asociado al umbral, lo cual sería un poco más de trabajo. Esto tambien aplica a la hora de querer saber el tamaño del vecindario asociado al nodo con cierto umbral (Para mi max-tree yo use un truco para evitar este posible problema).
- Component-Tree: Es ineficiente ya que hace referencia a los pixeles multiples veces, así que cuando el problema es mas grande y tiene muchos CC, estos son referencias una cantidad desproporciona de veces.

0.4 Conveniencia

- Max-Tree: A la hora de querer hacer el código más eficiente, rápido y liviano (en temas de memoria). Es la mejor opción casi siempre.
- Component-Tree: Es ineficiente ya que hace referencia a los pixeles multiples veces. pero en casos pequeños puede resultar muy similar a un Max-Tree. Tambien en casos donde se deban recorrer todos los pixeles que superan cierto umbral, es más fácil de acceder a todos ellos, ya que los tiene asociados directamente.

Complejidad de código

Usaremos las notaciones n para el total de pixeles y h para la altura del árbol.

0.5 Construcción del árbol

1) Instancio todos los pixeles que existen de la imagen proveniente. Complejidad $\mathcal{O}(n)$.

2) Recorro todos los pixeles y los asocia con $right$, $left$, $down$ y up según corresponda. Complejidad $\mathcal{O}(n)$.

3) Recorro ahora 2 veces todos los pixeles para obtener un posible y pixel inicial y luego otra para obtener el minimo umbral. Complejidad $\mathcal{O}(n)$.

4) Luego revisas todos los pixeles para ver cuales agregar al nodo padre. Complejidad $\mathcal{O}(n)$.

5) Despues de este nodo padre, recorro todos sus pixeles asociados y busco sí hay pixeles colindantes ($right$, $left$, $down$ y up) para cada uno de ellos, que no han sido asignados aún (uso un *boolean* en los atributos del struct pixel). Complejidad $\mathcal{O}(n)$.

6) En caso de no haber sido asignado aún creo un nodo apartir de este pixel. Agrego a todos los pixeles que superen al umbral del padre (podria decirse que momentaneamnete creo un "vecindario") (al asignarolos hago que agregado sea $true$). Asigno el umbral de este nodo, como el menor grisáceo de "vecindario". Acá le asigno su parentesco con padre y posibles hermanos. Complejidad $\mathcal{O}(n)$ y asignar el parentesco complejidad $\mathcal{O}(1)$.

7) Luego de hacer esto con todos los pixeles de este nodo padre, paso a limpiar los pixeles de este "vecindario", es decir, sacar de este arbol todos los que no cumplan con el umbral (hago que su *boolean* vuelva a ser $false$), transformándolo en un nodo propiamente tal. Complejidad $\mathcal{O}(n)$.

8) Luego hago que se llame recursivamente a la parte 5 con los nodos creados hijos del nodo padre. En el peor de los casos reviso h veces todos los pixeles del la imagen.

Como podemos ver hay hartas complejidades de valor n estando en esta escala, pero como recorremos recursivamente por el valor h obtenemos una complejidad final (en el peor de los casos claro) de $\mathcal{O}(n \times h)$, vale la pena mencionar que el exceso de n es simplificado por aproximación asintótica.
Entonces:

$$\mathcal{O}(n \times h)$$

0.6 Implementación de cada filtro

Para cada filtro, en el peor de los casos recorrido cada uno de los pixeles para analizar si lo debo cambiar o no (independiente de la cantidad de nodos o de la altura del arbol). Es por ello que tanto para el filtro de *área* como el *delta* obtengo la misma complejidad en el peor caso. Esta es:

$$\mathcal{O}(n)$$

Bibliografías

1. Building the component tree in quasi-linear time. Authors: Laurent Najman, Michel Couprie
2. A Comparison of Many Max-tree Computation Algorithms. Authors: Edwin Carlinet, Thierry Géraud