Estructuras avanzadas sobre árboles

Centroid Decomposition y Heavy Light Decomposition

Lautaro Lasorsa

Universidad de Buenos Aires - FCEN, Universidad Nacional de la Matanza - DIIT

Training Camp 2024

Gracias Sponsors!

Organizador

Diamond

Gold

Outline

Outline

Motivación

La Descomposición en Centroides es una técnica que nos permite responder consultas sobre cáminos en árboles, que pueden presentarse en variedad de sabores

Pero la clave para poder aplicar está técnica es que conociendo pueda calcular el valor asociado a un cámino de u a v conociendo los valores asociados a los cáminos de u a w y de w a v para algún nodo intermedio w.

Casos de uso

- Multiples consultas de la forma "Dado un nodo u y un nodo v, calcular el valor asociado al camino de u a v". Por ejemplo, la distancia entre u y v.

- Una consulta global sobre el conjunto de los caminos entre pares de nodos que sea fácilmente agregable. Por ejemplo, la cantidad de pares de nodos con una distancia menor o igual a un k dado.

Centroides

Definition (Centroide)

Un nodo es un centroide de un árbol si al eliminarlo del mismo todos los árboles resultantes tienen a lo mucho la mitad de nodos del árbol original.

Theorem

Todo árbol tiene al menos un centroide y a lo sumo dos.

Recursión

- Antes de eliminar al centroide de un árbol, puedo procesar todos los cáminos que pasan por ese centroide.
- Al eliminarlo, se forman nuevos árboles que son subárboles del árbol original y a su vez tienen sus propios centroides.
- Puedo repetir el proceso recursivamente.
- Todos los cáminos de los nuevos árboles son cáminos del árbol original.
- Cada cámino del árbol original es considerado una y solo una vez.
- Notar que cada árbol tiene un tamaño menor o igual a la mitad del de su padre en la recursión.
- Por tanto, cada nodo es procesado a lo mucho $\log n$ veces.
- La complejidad total es $O(n \log n)$.

Ejemplo de uso

Enunciado

Dado un árbol T, calcular la cantidad de pares de nodos a una distancia menor o igual a k.

Solucion con descomposición en centroides

- Al procesar un centroide, puedo calcular la distancia de ese nodo a todos los demás nodos de su árbol.
- Al calcular la distancia d a un nodo u, puedo saber con una estructura cómo Fenwick/Segment Tree cuantos otros nodos del árbol hay a una distancia al centroide menor o igual a $k - d$ y pertenecen a otro subárbol.
- Esto permite resolver cada árbol T en $O(|T| * \log |T|)$.
- La complejidad total es $O(n \log^2 n)$.

Construir una estructura

Definition (Árbol de centroides)

Si dado un árbol T, al ejecutar la descomposición en centroides construimos un árbol C_T con los mismos nodos de T, donde en C_T el nodo u es hijo del nodo v si en las llamadas recursivas el subárbol que tiene por centroide a u es llamado por el subárbol que tiene por centroide a v.
C_T es el árbol de centroides de T.

Propiedades del árbol de centroides

- C_T tiene altura $O(\log n)$.
- Por tanto, la suma de las profundidades (cantidad de ancestros) de los nodos es $O(n \log n)$.
- El $lca(u, v)$ en C_T es el único centroide encargado de procesar el cámino entre u y v.
- Permite realizar queries y updates online de forma rápida.

Distancias

Recordar que si los ancestros de u en C_T son $u_1, u_2, \ldots, u_k$, el cámino en T (el que nos interesa) de u a u_2 no necesariamente pasa por u_1. Hay que guardar explicitamente el valor del cámino entre u y cada uno de sus ancestros (lo que calculamos en el algoritmo de Descomposición en Centroides).

Ejemplo de uso: Enunciado

Enunciado

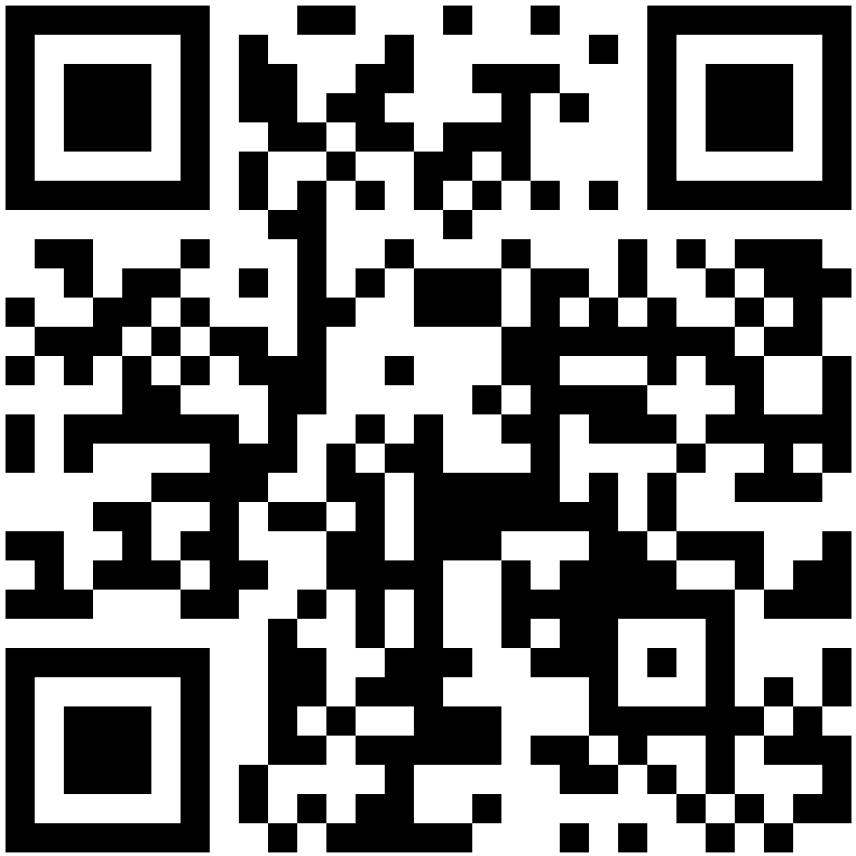
Dado un árbol T con n nodos negros y blancos, realizar q consultas de la forma: "Invertir el color del nodo u" Al inicio y después de cada update calcular la suma de las distancias entre los nodos negros.

Ejemplo de uso: Solución

- La respuesta para la situación inicial la calculo durante la construcción del árbol.
- En cada nodo guardo la suma de las distancias (en T) a los nodos negros de su subárbol (en C_T) y las distancias asociadas a los nodos negros de cada uno de sus hijos (en C_T).
- En cambos casos además de las distancias guardo la cantidad de nodos.
- Al pasar un nodo de negro a blanco, para él y todos sus ancestros primero reduzco la suma de distancias a nodos negros y cantidad de nodos (y la asociada al hijo por el que subimos) y luego resto la distnacia de este nodo al centroide multiplicada por la cantidad de nodos negros (que no son del mismo subárbol) y también la suma de las distancias a nodos negros (que no son del mismo subárbol)
- Al pasar un nodo de blanco a negro hago lo mismo que antes, solo que sumo en vez de restar, y primero calculo el cambio en la respuesta global y luego cambio la estructura.

Código

Implementación Árbol de Centroides



Outline

Motivación

Al igual que la descomposición en centroides, HLD nos permite responder consultas sobre caminos entre pares de nodos y aplicar actualizaciones a los valores de nodos o aristas.

Para poder aplicar esta técnica necesitamos que la operación que calcula el valor asociado a un cámino sea asociativa.

Es decir, $Op(u \dots v) = Op(Op(u \dots w), Op(w \dots v))$ para cualquier w en el cámino simple de u a v

Conceptos

La descomposición en (aristas) pesadas y livianas se aplica sobre árboles enraizados. Lo bueno es que podemos enraizar arbitraríamente cualquier árbol si nos es útil.

Definition (Arista pesada)

Una arista es pesada si el tamaño del subárbol que cuelga de ella es mayor a la mitad del tamaño del subárbol del padre.

Definition (Arista liviana)

Una arista es liviana si el tamaño del subárbol que cuelga de ella es la mitad o menos del tamaño del subárbol del padre.

Propiedades

- De cada nodo cuelga a lo mucho una arista pesada.
- En el cámino de la raíz a un nodo hay a lo mucho $\log n$ aristas livianas.
- Sea un heavy path un cámino formado unicamente por aristas pesadas, hay a lo mucho $\log n$ heavy paths disjuntos en el cámino de un nodo a la raíz.

Descomposición en Aristas Pesadas y Livianas

- Calculamos las aristas pesadas y livianas.
- Para cada heavy path construimos una estructura de datos que soporte consultas y actualizaciones eficientes.
- Si los valores están en las aristas, los asociamos al nodo que cuelga de ella.

Definition (Next u)

Sea un nodo u, llamo Next[u] al nodo menos profundo en el heavy path al que pertenece u.

- Si la arista de la que cuelga u es liviana, $Next[u] = u$.
- En cambio, si la arista de la que cuelga u es pesada, $Next[u] = Next[padre[u]]$.

Responder consultas de un nodo a un ancestro

- Empiezo en un nodo u y quiero llegar a un nodo v, ancestro de u.
- Si la arista de la que cuelga u es pesada, puede haber dos escenarios:
  - 1. $Next[v] = Next[u]$, en ese caso agrego a la respuesta $Op(u \dots v)$ que obtengo de la estructura de datos.
  - 2. $Next[v] \neq Next[u]$, en ese caso agrego a la respuesta $Op(u \dots Next[u])$ y sigo con $u = Next[u]$.
- Si en cambio u cuelga de una arista liviana, agrego $Op(u, padre[u])$ a la respuesta y sigo con $u = padre[u]$.
- Al final, si el valor está en los nodos, debo agregar el valor asociado a v a la respuesta.

Responder consultas entre cualquier par de nodos

Algorithm 1 Heavy Light algorithm query

Require: u, v nodos de la query, S estructura de datos a consultar.

```
res = NULL. Valor neutro de la respuesta
while u ≠ v do
    if prof[u] < prof[v] then
        swap(u, v) {Intercambia u y v si es necesario}
    end if
    if Next[u] = Next[v] then
        res = Op(res, S.query(u, v))
        break {Termina el bucle si alcanzamos el mismo heavy path}
    else
        res = Op(res, S.query(u, Next[u]))
        res = Op(res, S.query(Next[u]))
        u = padre[Next[u]]
    end if
end while
if VALOR EN NODOS then
    res = Op(res, S.query(v))
end if
return res
```

Preorder de un árbol

Definition (Preorder)

- El Preorder de un árbol consiste en recorrer el árbol con un DFS y cuando se llega a un nodo desde su padre (o a la raíz al principio) agregar el nodo a la lista.
- La posición del nodo u en la lista la llamamos $in[u]$
- La longitud de la lista al terminar de procesar el nodo u (es decir, el siguiente a la última posición de un nodo del subárbol de u) es $out[u]$
- Si y solo si v está en el subárbol de u, $in[u] < in[v] < out[u]$

Consultas en subárbol

Esto nos permite hacer rápidamente consultas sobre el subárbol de un nodo u si tenemos una estructura que soporte consultas en rango.

Example (Suma de los nodos en el subárbol)

Sea un árbol T de n nodos, quiero poder hacer q consultas de la forma:

- 1. Asignar un valor x a un nodo u (inicialmente todos valen 0)
- 2. Calcular la suma de los valores de los nodos en el subárbol de u (inclusive)

Con el preorder y un Fenwick/Segment Tree esto puede resolverse en $O(q \log n)$

- 1. $update(in[u], x)$
- 2. $query(in[u], out[u])$ (convención inclusive-exclusive)

Combinar ambas técnicas

La clave consiste en que, si cuando de un nodo cuelga una arista pesada procesamos esa arista primero, ocurre lo siguiente con el preorder del árbol:

- $[in[u], out[u])$ es el rango que contiene al subárbol con raíz en u.
- Si v es un ancestro de u en su mismo heavy path, $(in[v], in[u]]$ es el tramo del heavy path que va del nodo v al nodo u (incluyendo a u, excluyendo a v).
- De esta forma, podemos utilizar una unica estructura de datos para todos los heavy path y adicionalmente podemos responder consultas en subárboles.

Ejemplo de uso

Enunciado

Dado un árbol T con n nodos, donde inicialmente todos valen 0, responder q consultas de la forma:

- 1. Asignar un valor x a un nodo u.
- 2. Calcular el máximo de los valores en el subárbol de u (inclusive).
- 3. Calcular el máximo de los valores de los nodos en el cámino de u a v (inclusive).

Solución

Utilizo la descomposición antes descripta y un árbol de segmentos y obtengo las siguientes complejidades:

- 1. $O(\log n)$
- 2. $O(\log n)$
- 3. $O(\log^2 n)$

Operaciones no conmutativas

En el caso de operaciones no conmutativas la consulta sobre subárboles pierde interes pero sigue teniendo sentido la de caminos, y la implementación antes comentada es de todas formas muy cómoda y eficiente.

Ejemplo no conmutativo

Enunciado

Dado un árbol T con n nodos donde inicialmente cada nodo tiene asignado un valor 0. Aplicar el siguiente tipo de consultas:

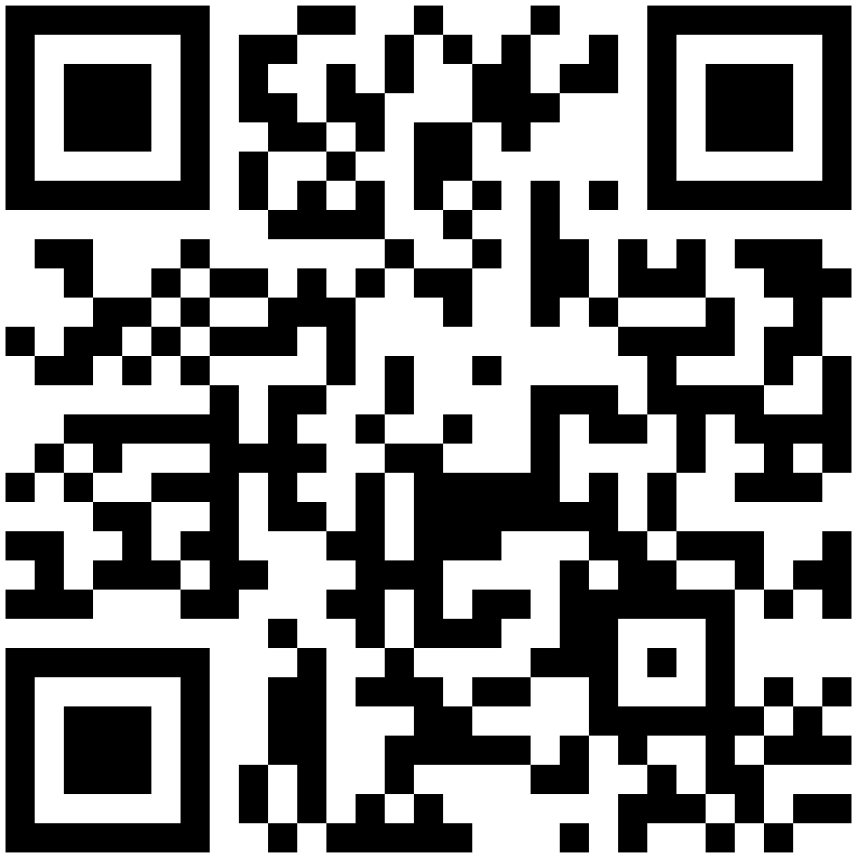
- 1. Asignar un valor x a un nodo u.
- 2. Dados dos pares de nodos (u_1, v_1) y (u_2, v_2), decidir si la secuencia de valores en los nodos en los caminos de u_1 a v_1 y de u_2 a v_2 son iguales.

Solución

Utilizando hashing y HLD podemos resolver el problema antes planteado con actualizaciones $O(\log n)$ y consultas $O(\log^2 n)$.
Notar que en este caso la implementación será más rebuscada porque nos importa si el cámino que estamos haciendo sube o baja en el árbol.
La estructura deberá permitir calcular los hashes en ambos sentidos.

Código

Implementación Heavy Light Decomposition



Gracias por ver

Medios de contacto:

- laulasorsa@unlam.edu.ar
- lautarolasorsa@gmail.com
- @lautaro-lasorsa en LinkedIn