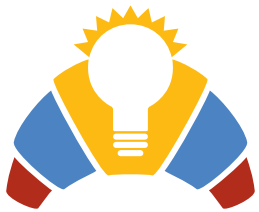
Centroid Decomposition y Virtual Trees

Avanzado

Eduardo Carranza Velez

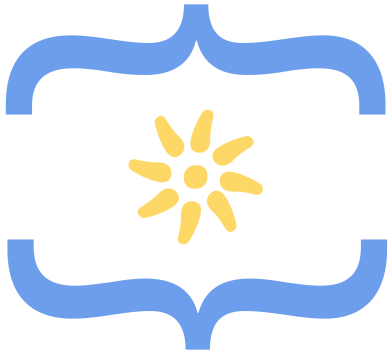
Universidad Nacional de Córdoba - Facultad de Matemática, Astronomía y Física

Training Camp 2026



Gracias Sponsors!

Organiza



Facultad de INFORMÁTICA

UNIVERSIDAD NACIONAL DE LA PLATA

Diamond Sponsor

Outline

Outline

Motivación

- El objetivo no es encasillar los problemas ("¿sale con DP? ¿sale con Centroid?"), sino **ampliar nuestra caja de herramientas**.

Motivación

- El objetivo no es encasillar los problemas ("¿sale con DP? ¿sale con Centroid?"), sino **ampliar nuestra caja de herramientas**.

- Hay problemas de estructuras de datos sobre árboles donde conocer estas técnicas facilita muchísimo el *approach* inicial. Tener más opciones disponibles nos permite elegir la herramienta con la que nos resulte más fácil de pensar el problema.

Motivación

- El objetivo no es encasillar los problemas ("¿sale con DP? ¿sale con Centroid?"), sino **ampliar nuestra caja de herramientas**.

- Hay problemas de estructuras de datos sobre árboles donde conocer estas técnicas facilita muchísimo el *approach* inicial. Tener más opciones disponibles nos permite elegir la herramienta con la que nos resulte más fácil de pensar el problema.

- **Problemas sobre los caminos del árbol:** Querríamos alguna forma de procesar información sobre **todos** los caminos posibles, sin tener que iterar explícitamente todos los pares de nodos que es $\mathcal{O}(N^2)$.

Motivación

- El objetivo no es encasillar los problemas ("¿sale con DP? ¿sale con Centroid?"), sino **ampliar nuestra caja de herramientas**.

- Hay problemas de estructuras de datos sobre árboles donde conocer estas técnicas facilita muchísimo el *approach* inicial. Tener más opciones disponibles nos permite elegir la herramienta con la que nos resulte más fácil de pensar el problema.

- **Problemas sobre los caminos del árbol:** Querríamos alguna forma de procesar información sobre **todos** los caminos posibles, sin tener que iterar explícitamente todos los pares de nodos que es $\mathcal{O}(N^2)$.

- **Problemas de subconjuntos de nodos:** Hay muchos problemas en los cuales hay que responder consultas sobre un subconjunto específico de nodos, y nosotros querríamos poder iterar solo el tamaño de ese subconjunto en lugar del árbol entero.

Objetivos de la clase

Al finalizar la clase, buscamos que sean capaces de:

Objetivos de la clase

Al finalizar la clase, buscamos que sean capaces de:

- Comprender la estructura del **Árbol de Centroides** y utilizar su profundidad $\mathcal{O}(\log N)$ para precalcular información de caminos.

Objetivos de la clase

Al finalizar la clase, buscamos que sean capaces de:

- Comprender la estructura del **Árbol de Centroides** y utilizar su profundidad $\mathcal{O}(\log N)$ para precalcular información de caminos.
- Construir **Virtual Trees** en tiempo $\mathcal{O}(k \log k)$ para resolver consultas sobre subconjuntos sin depender del N total.

Objetivos de la clase

Al finalizar la clase, buscamos que sean capaces de:

- Comprender la estructura del **Árbol de Centroides** y utilizar su profundidad $\mathcal{O}(\log N)$ para precalcular información de caminos.
- Construir **Virtual Trees** en tiempo $\mathcal{O}(k \log k)$ para resolver consultas sobre subconjuntos sin depender del N total.
- **Reconocer patrones:** Identificar rápidamente en un enunciado de competencia cuándo conviene usar cada una de estas estructuras.

Outline

Definición de centroide

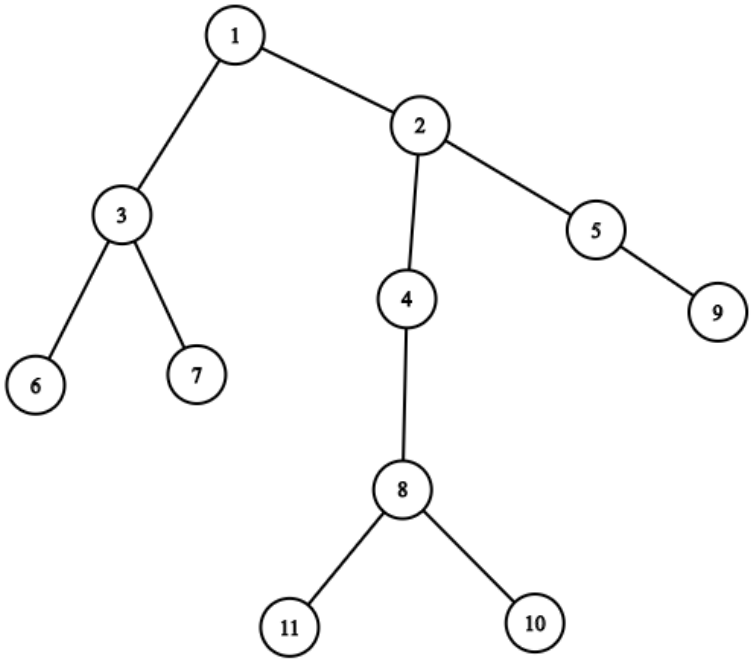
Dado un árbol T con N nodos, consideremos un nodo $c \in T$.

Definition (Centroide)

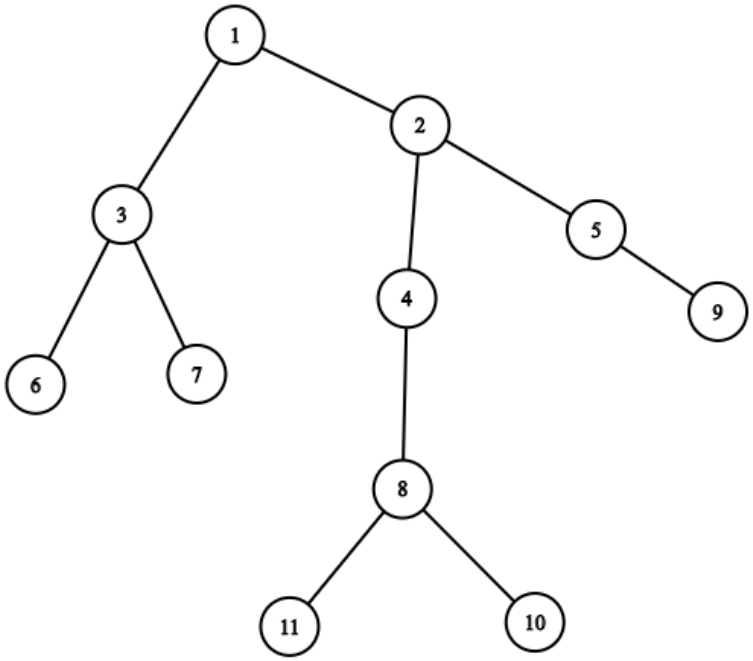
Al **eliminar** el nodo c junto con todas sus aristas incidentes, el árbol se descompone en varias componentes conexas (subárboles). Decimos que c es un **centroide** de T si **cada** una de esas componentes tiene tamaño a lo sumo $\lfloor N/2 \rfloor$.

Equivalentemente: para todo vecino u de c, la componente que queda del lado de u al cortar la arista (c, u) contiene a lo sumo $\lfloor N/2 \rfloor$ nodos.

Ejemplo visual



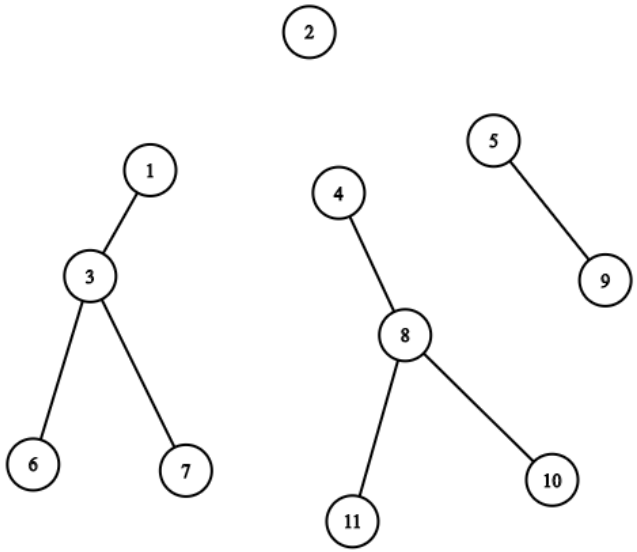
Ejemplo visual



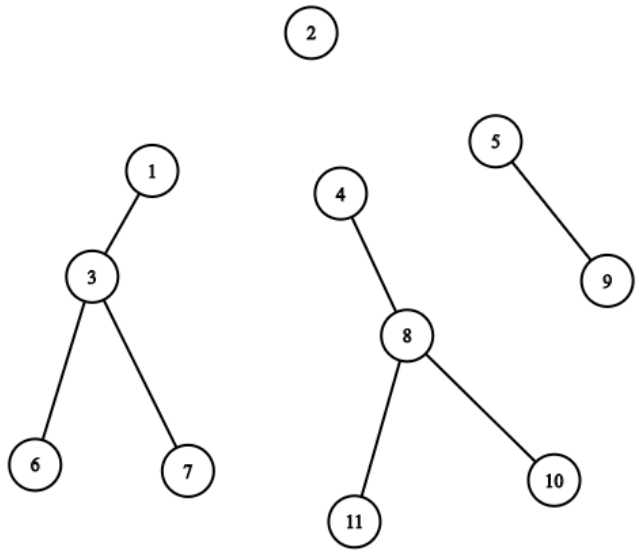
Centroide

El centroide es el nodo 2.

Al remover el centroide



Al remover el centroide



¿Por qué 2 es centroide?

El árbol tiene $N = 11$ nodos. Al remover el nodo 2, las componentes resultantes tienen tamaños 4, 4 y 2, y todos son $\leq \lfloor 11/2 \rfloor = 5$.

Existencia del Centroide

- **Siempre existe:** Todo árbol con N nodos tiene *al menos* un centroide.
- **A lo sumo dos:** Un árbol puede tener a lo sumo dos centroides. Si existen dos, estos siempre están conectados por una arista.

Demostración en clase

Demostraremos la existencia del centroide construyendo un algoritmo que siempre lo encuentra.

Encontrar un centroide

Si probamos sacar un nodo arbitrario u y analizamos sus componentes conexas tenemos dos casos:

1. **Todas tienen tamaño $\leq \lfloor N/2 \rfloor$.**
   - Por definición, u **es el centroide**. Terminamos.
2. **Existe una única componente con tamaño $> \lfloor N/2 \rfloor$.**
   - El verdadero centroide debe estar en esa dirección.
   - Nos movemos hacia el vecino v de esa componente y repetimos.

Encontrar un centroide

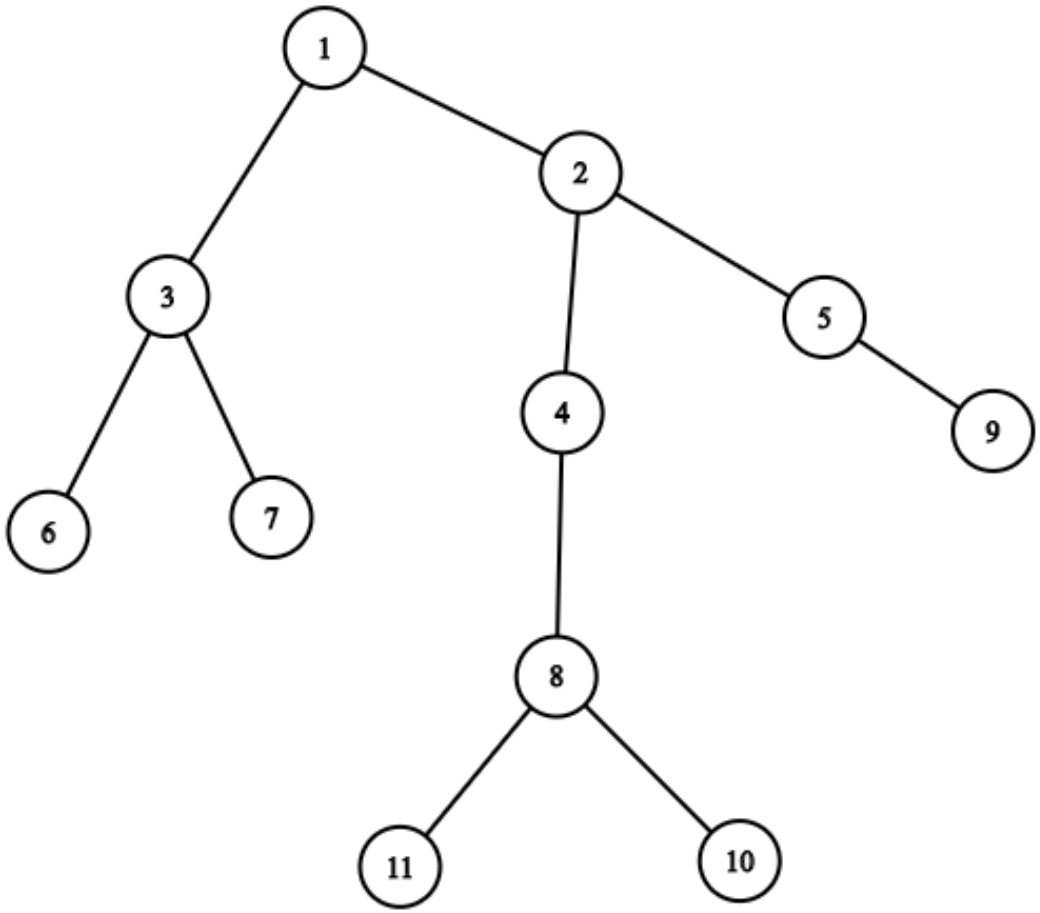
Si probamos sacar un nodo arbitrario u y analizamos sus componentes conexas tenemos dos casos:

1. **Todas tienen tamaño $\leq \lfloor N/2 \rfloor$.**
   - Por definición, u **es el centroide**. Terminamos.
2. **Existe una única componente con tamaño $> \lfloor N/2 \rfloor$.**
   - El verdadero centroide debe estar en esa dirección.
   - Nos movemos hacia el vecino v de esa componente y repetimos.

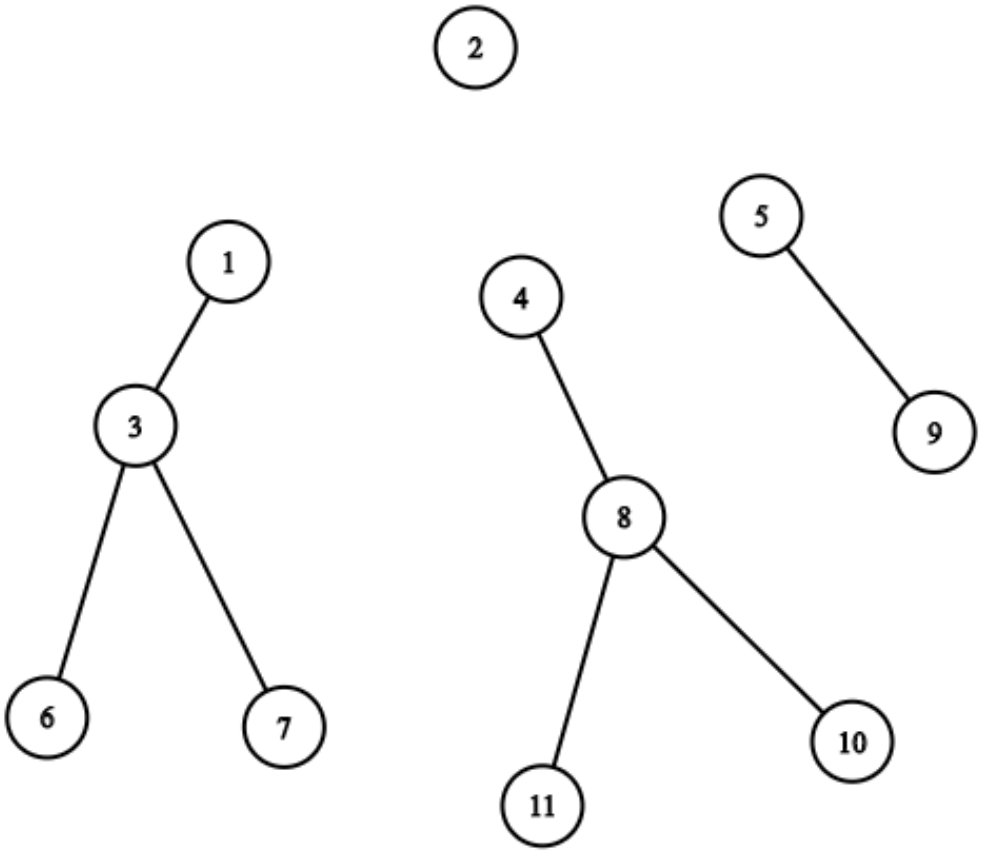
¿Por qué el algoritmo termina?
Al movernos a v, la componente que dejamos atrás pasa a tener $\leq \lfloor N/2 \rfloor$ nodos. Por lo tanto, **nunca volveremos** al nodo u. Al no poder volver sobre nuestros pasos en un árbol finito, obligatoriamente encontraremos el centroide.

Outline

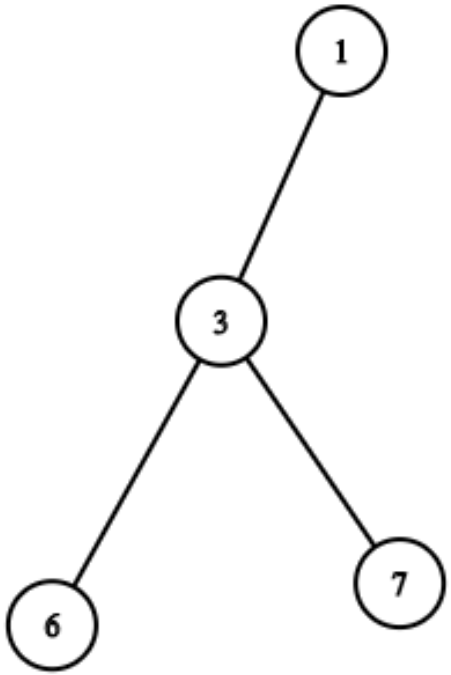
Recordamos: El árbol original



Recordamos: Al remover el centroide



Repetimos en los subárboles restantes



El Árbol de Centroides

Si vamos haciendo esta descomposición recursivamente, se nos forma una nueva estructura: el **Árbol de Centroides**.

- La raíz es el centroide del árbol original.
- Los hijos de un centroide en este nuevo árbol son los centroides de los subárboles resultantes.

El Árbol de Centroides

Si vamos haciendo esta descomposición recursivamente, se nos forma una nueva estructura: el **Árbol de Centroides**.

- La raíz es el centroide del árbol original.
- Los hijos de un centroide en este nuevo árbol son los centroides de los subárboles resultantes.

Profundidad Logarítmica

Como en cada paso el tamaño de la componente se reduce a la mitad o menos (tamaño $\leq \lfloor N/2 \rfloor$), la altura máxima de este árbol es $\mathcal{O}(\log N)$.

En otras palabras: Todo nodo tiene **a lo sumo** $\log(N)$ **ancestros** en el Árbol de Centroides.

El Árbol de Centroides

Si vamos haciendo esta descomposición recursivamente, se nos forma una nueva estructura: el **Árbol de Centroides**.

- La raíz es el centroide del árbol original.
- Los hijos de un centroide en este nuevo árbol son los centroides de los subárboles resultantes.

Profundidad Logarítmica

Como en cada paso el tamaño de la componente se reduce a la mitad o menos (tamaño $\leq \lfloor N/2 \rfloor$), la altura máxima de este árbol es $\mathcal{O}(\log N)$.

En otras palabras: Todo nodo tiene **a lo sumo** $\log(N)$ **ancestros** en el Árbol de Centroides.

Esta propiedad hace que la complejidad del algoritmo de Centroid Decomposition sea $\mathcal{O}(N \log N)$.

Propiedad Fundamental

Dado cualquier par de nodos u y v en el árbol original, existe **exactamente un** centroide c que cumple dos condiciones:

1. c se encuentra en el camino original entre u y v.
2. Ambos nodos, u y v, pertenecen al subárbol de c (en el momento de la descomposición donde elegimos a c).

Propiedad Fundamental

Dado cualquier par de nodos u y v en el árbol original, existe **exactamente un** centroide c que cumple dos condiciones:

1. c se encuentra en el camino original entre u y v.
2. Ambos nodos, u y v, pertenecen al subárbol de c (en el momento de la descomposición donde elegimos a c).

¿Qué significa esto en la práctica?

Cualquier camino entre dos nodos puede dividirse en **dos tramos**: desde u hasta c, y desde c hasta v.
Y ambos caminos se pueden procesar al momento de sacar c.

Propiedad Fundamental

Dado cualquier par de nodos u y v en el árbol original, existe **exactamente un** centroide c que cumple dos condiciones:

1. c se encuentra en el camino original entre u y v.
2. Ambos nodos, u y v, pertenecen al subárbol de c (en el momento de la descomposición donde elegimos a c).

¿Qué significa esto en la práctica?

Cualquier camino entre dos nodos puede dividirse en **dos tramos**: desde u hasta c, y desde c hasta v.
Y ambos caminos se pueden procesar al momento de sacar c.

Es decir, estos tipos de caminos (de un nodo a un ancestro centroid) son fáciles de procesar y hay pocos ($\mathcal{O}(N \log N)$), y con dos de esos caminos podemos armar cualquier camino en el árbol.

Implementación — Centroid Decomposition

Construcción en $\mathcal{O}(N \log N)$

```
// tag[y]>=tag[x] for every y in x's connected subgraph (unrooted subtree)
vector<int> g[MAXN];
int tag[MAXN]; // time of discovery of centroid
int fat[MAXN]; // father in centroid decomposition
int szt[MAXN]; // size of subtree
int calcsz(int x, int f){
    szt[x] = 1;
    for(auto y : g[x]) if(y != f && tag[y] < 0) szt[x] += calcsz(y, x);
    return szt[x];
}
int ccnt = 0;
void cdfs(int x, int f, int sz = -1){ // O(nlogn)
    if(sz < 0) sz = calcsz(x, -1);
    for(auto y : g[x]) if(tag[y] < 0 && szt[y] * 2 >= sz){
        szt[x] = 0; cdfs(y, f, sz); return;
    }
    tag[x] = ccnt++; fat[x] = f;
    for(auto y : g[x]) if(tag[y] < 0) cdfs(y, x);
}
void centroid(){ mset(tag, -1); ccnt = 0; cdfs(0, -1); }
```

Problema de ejemplo

Xenia and Tree

Dado un árbol de N nodos, inicialmente todos los nodos son azules excepto el nodo 1, que es rojo. Se deben procesar consultas de dos tipos de forma online:

1. **Actualización:** Pintar un nodo dado u de rojo.
2. **Consulta:** Dado un nodo v, encontrar la mínima distancia desde v hasta algún nodo rojo.

Calcular distancias a los ancestros centroid

En el problema anterior necesitamos obtener distancias frecuentemente. Podríamos usar la forma genérica de calcularlas en un árbol con LCA (Binary Lifting o RMQ), pero con Centroid Decomposition podemos hacerlo de forma mucho más directa.

Calcular distancias a los ancestros centroid

En el problema anterior necesitamos obtener distancias frecuentemente. Podríamos usar la forma genérica de calcularlas en un árbol con LCA (Binary Lifting o RMQ), pero con Centroid Decomposition podemos hacerlo de forma mucho más directa.

Optimizando el cálculo de distancias

Dado que **solo** nos interesan las distancias desde un nodo hacia sus $\mathcal{O}(\log N)$ ancestros centroid, podemos **precalcularlas**.

Calcular distancias a los ancestros centroid

En el problema anterior necesitamos obtener distancias frecuentemente. Podríamos usar la forma genérica de calcularlas en un árbol con LCA (Binary Lifting o RMQ), pero con Centroid Decomposition podemos hacerlo de forma mucho más directa.

Optimizando el cálculo de distancias

Dado que **solo** nos interesan las distancias desde un nodo hacia sus $\mathcal{O}(\log N)$ ancestros centroid, podemos **precalcularlas**.

- Durante la construcción, cada vez que encontramos un centroide c, hacemos un DFS/BFS simple desde c para calcular la distancia a todos los nodos de su componente actual.
- Guardamos estas distancias en un arreglo, por ejemplo `dist[nivel][u]`.
- Nos ahorramos la implementación y el $\log N$ adicional del LCA !!

Otro problema de ejemplo

Fixed-Length Paths I

Dado un árbol de N nodos y un entero K.

El objetivo es contar la cantidad total de caminos distintos en el árbol que tienen una longitud de exactamente K aristas.

Cuidado con los caminos inválidos

Al contar caminos que pasan por el centroide c, un error muy común es contar **caminos inválidos** que "suben" hasta c y "bajan" por la misma rama, es decir, caminos que inician y terminan en la misma componente.

Cuidado con los caminos inválidos

Al contar caminos que pasan por el centroide c, un error muy común es contar **caminos inválidos** que "suben" hasta c y "bajan" por la misma rama, es decir, caminos que inician y terminan en la misma componente.

La trampa de la misma componente

- En *Xenia and Tree* no nos preocupamos por esto, porque buscábamos la **mínima** distancia, y los caminos que suben y bajan por la misma rama siempre son más largos (subóptimos).
- En problemas de **contar** (como *Fixed-Length Paths*), estos caminos nos van a arruinar la respuesta.

Cuidado con los caminos inválidos

Al contar caminos que pasan por el centroide c, un error muy común es contar **caminos inválidos** que "suben" hasta c y "bajan" por la misma rama, es decir, caminos que inician y terminan en la misma componente.

La trampa de la misma componente

- En *Xenia and Tree* no nos preocupamos por esto, porque buscábamos la **mínima** distancia, y los caminos que suben y bajan por la misma rama siempre son más largos (subóptimos).
- En problemas de **contar** (como *Fixed-Length Paths*), estos caminos nos van a arruinar la respuesta.

Solución típica:
Procesar los subárboles de c de a uno por vez, actualizando la respuesta con la estructura de datos histórica, o usar inclusión-exclusión (sumar para la componente total de c y restar para cada subárbol individual).

Outline

Problema Motivador

Leaf Color

Dado un árbol de N nodos, donde cada nodo tiene asignado un color C_i.

Debemos contar cuántos subconjuntos de nodos se pueden elegir de tal forma que el subgrafo inducido por ellos sea un árbol, y todas las hojas de ese subárbol tengan el mismo color.

Ignoremos los colores

Ignoremos los colores

- Si ignoramos los colores, el problema es contar la cantidad de subárboles.

Ignoremos los colores

- Si ignoramos los colores, el problema es contar la cantidad de subárboles.
- Y sale con la siguiente DP:

Ignoremos los colores

- Si ignoramos los colores, el problema es contar la cantidad de subárboles.
- Y sale con la siguiente DP:
(pizarrón)

Ahora sí tengamos en cuenta los colores

Ahora sí tengamos en cuenta los colores

- Podríamos iterar el color que van a tener las hojas.

Ahora sí tengamos en cuenta los colores

- Podríamos iterar el color que van a tener las hojas.
- Se puede adaptar la misma DP para tener en cuenta que las hojas tengan el color iterado.

Ahora sí tengamos en cuenta los colores

- Podríamos iterar el color que van a tener las hojas.
- Se puede adaptar la misma DP para tener en cuenta que las hojas tengan el color iterado.
- Pero si iteramos todo el árbol para todos los colores nos queda cuadrático.

Ahora sí tengamos en cuenta los colores

- Podríamos iterar el color que van a tener las hojas.
- Se puede adaptar la misma DP para tener en cuenta que las hojas tengan el color iterado.
- Pero si iteramos todo el árbol para todos los colores nos queda cuadrático.
- Así que querríamos de alguna forma, por cada color, iterar los nodos de ese color y algunos más para preservar la "forma" del árbol.

Ahora sí tengamos en cuenta los colores

- Podríamos iterar el color que van a tener las hojas.
- Se puede adaptar la misma DP para tener en cuenta que las hojas tengan el color iterado.
- Pero si iteramos todo el árbol para todos los colores nos queda cuadrático.
- Así que querríamos de alguna forma, por cada color, iterar los nodos de ese color y algunos más para preservar la "forma" del árbol.
- Ese nuevo árbol se puede calcular y va tener $\mathcal{O}(k)$ nodos, donde k es la cantidad de nodos de ese color.

Ahora sí tengamos en cuenta los colores

- Podríamos iterar el color que van a tener las hojas.
- Se puede adaptar la misma DP para tener en cuenta que las hojas tengan el color iterado.
- Pero si iteramos todo el árbol para todos los colores nos queda cuadrático.
- Así que querríamos de alguna forma, por cada color, iterar los nodos de ese color y algunos más para preservar la "forma" del árbol.
- Ese nuevo árbol se puede calcular y va tener $\mathcal{O}(k)$ nodos, donde k es la cantidad de nodos de ese color.
- Ese nuevo árbol es el **virtual tree** del conjunto de nodos que estamos viendo (en este caso los nodos de un color).

Formalmente: Virtual Tree

Dado un árbol original y un subconjunto de nodos destacados S (con $|S| = k$):

Formalmente: Virtual Tree

Dado un árbol original y un subconjunto de nodos destacados S (con $|S| = k$):

- **Vértices:** El Virtual Tree contiene a todos los nodos de S, **más** el LCA de cualquier par de nodos en S.

Formalmente: Virtual Tree

Dado un árbol original y un subconjunto de nodos destacados S (con $|S| = k$):

- **Vértices:** El Virtual Tree contiene a todos los nodos de S, **más** el LCA de cualquier par de nodos en S.
- Matemáticamente: $V = S \cup \{\text{LCA}(u, v) \mid u, v \in S\}$.

Formalmente: Virtual Tree

Dado un árbol original y un subconjunto de nodos destacados S (con $|S| = k$):

- **Vértices:** El Virtual Tree contiene a todos los nodos de S, **más** el LCA de cualquier par de nodos en S.
- Matemáticamente: $V = S \cup \{\text{LCA}(u, v) \mid u, v \in S\}$.
- **Aristas:** Preservan la estructura original. Se conecta u con v si u es ancestro de v en el árbol original y no hay ningún otro nodo de V en el camino original entre ellos.

Formalmente: Virtual Tree

Dado un árbol original y un subconjunto de nodos destacados S (con $|S| = k$):

- **Vértices:** El Virtual Tree contiene a todos los nodos de S, **más** el LCA de cualquier par de nodos en S.
- Matemáticamente: $V = S \cup \{\text{LCA}(u, v) \mid u, v \in S\}$.
- **Aristas:** Preservan la estructura original. Se conecta u con v si u es ancestro de v en el árbol original y no hay ningún otro nodo de V en el camino original entre ellos.

Propiedad Mágica del Tamaño

Aunque existan $\mathcal{O}(k^2)$ pares posibles en S, la cantidad de LCA **únicos** nuevos que se generan y se agregan es, como máximo, $k - 1$.

Por lo tanto, el tamaño final del Virtual Tree es siempre $\leq 2k - 1 \implies \mathcal{O}(k)$.

¿Cómo construir el Virtual Tree?

Para armarlo en $\mathcal{O}(k \log k)$ y evitar el $\mathcal{O}(k^2)$ de buscar todos los pares, hacemos lo siguiente:

¿Cómo construir el Virtual Tree?

Para armarlo en $\mathcal{O}(k \log k)$ y evitar el $\mathcal{O}(k^2)$ de buscar todos los pares, hacemos lo siguiente:

1. **Ordenar por DFS:** Ordenamos los nodos del conjunto S según su tiempo de descubrimiento de un DFS previo sobre el árbol original.

¿Cómo construir el Virtual Tree?

Para armarlo en $\mathcal{O}(k \log k)$ y evitar el $\mathcal{O}(k^2)$ de buscar todos los pares, hacemos lo siguiente:

1. **Ordenar por DFS:** Ordenamos los nodos del conjunto S según su tiempo de descubrimiento de un DFS previo sobre el árbol original.
2. **LCAs adyacentes:** La magia está en que solo necesitamos los LCAs de los elementos **adyacentes** en ese arreglo ordenado.
   - Agregamos $\text{LCA}(S_i, S_{i+1})$ para todo i.

¿Cómo construir el Virtual Tree?

Para armarlo en $\mathcal{O}(k \log k)$ y evitar el $\mathcal{O}(k^2)$ de buscar todos los pares, hacemos lo siguiente:

1. **Ordenar por DFS:** Ordenamos los nodos del conjunto S según su tiempo de descubrimiento de un DFS previo sobre el árbol original.
2. **LCAs adyacentes:** La magia está en que solo necesitamos los LCAs de los elementos **adyacentes** en ese arreglo ordenado.
   - Agregamos $\text{LCA}(S_i, S_{i+1})$ para todo i.
3. **Limpiar el conjunto:** Unimos el S original con los LCAs calculados. Ordenamos este nuevo conjunto de vértices V nuevamente por orden DFS y eliminamos duplicados.

¿Cómo construir el Virtual Tree?

Para armarlo en $\mathcal{O}(k \log k)$ y evitar el $\mathcal{O}(k^2)$ de buscar todos los pares, hacemos lo siguiente:

1. **Ordenar por DFS:** Ordenamos los nodos del conjunto S según su tiempo de descubrimiento de un DFS previo sobre el árbol original.
2. **LCAs adyacentes:** La magia está en que solo necesitamos los LCAs de los elementos **adyacentes** en ese arreglo ordenado.
   - Agregamos $\text{LCA}(S_i, S_{i+1})$ para todo i.
3. **Limpiar el conjunto:** Unimos el S original con los LCAs calculados. Ordenamos este nuevo conjunto de vértices V nuevamente por orden DFS y eliminamos duplicados.
4. **Conectar aristas:** Iteramos los nodos de V manteniendo un **stack** de ancestros para armar las ramas:
   - Si el tope de la pila no es ancestro del nodo actual u, hacemos pop hasta que lo sea.
   - El tope resultante es el padre de u en el Virtual Tree.
   - Agregamos la arista y hacemos `push(u)`.

Virtual tree en tiempo lineal

Implementación Lineal (y más corta)

```
// assumes sorted v (in dfs order)
// virtual (directed) tree is t
void agr(ll x, ll y){if(y!=-1)t[x].pb(y);}
ll virtu(vector<ll> v){ // O(|v|) * O(lca)
    stack<ll>s; s.push(v[0]); ll ult=-1,p;
    auto vacia=[&](bool fg){
        while(SZ(s)&&(fg||lca(s.top(),p)!=s.top())){
            agr(s.top(),ult);
            ult=s.top(); s.pop();
        }
    };
    vector<ll> vi; // virtual nodes and possibly normal
    fore(i,1,SZ(v)){
        ll x=v[i]; p=lca(s.top(),x); vi.pb(p); vacia(0);
        if(s.empty()||p!=s.top())s.push(p);
        agr(p,ult); ult=-1; if(p!=x)s.push(x);
    }
    vacia(1); ll rt=ult; // root of t
    // do stuff, then reset t with both v and vi
}
```

Outline

Para pensar

Desafío: Alarmas en cascada

Dado un árbol de N nodos con pesos en las aristas. En cada nodo i hay instalada una alarma que tiene un rango de alcance d_i.

Si una alarma en el nodo u suena, su sonido viaja por el árbol y **activa automáticamente** a cualquier otra alarma ubicada en un nodo v siempre que se cumpla:

$$\text{dist}(u, v) \leq d_u$$

Si una alarma percibe el sonido de otra, esta también se activa de forma automática, propagando su propio sonido hasta su rango d_v.

Objetivo:
Encontrar la **mínima cantidad** de alarmas que se deben activar manualmente para que **todas** las alarmas del árbol terminen activándose.

Para pensar

Desafío: Alarmas en cascada

Dado un árbol de N nodos con pesos en las aristas. En cada nodo i hay instalada una alarma que tiene un rango de alcance d_i.

Si una alarma en el nodo u suena, su sonido viaja por el árbol y **activa automáticamente** a cualquier otra alarma ubicada en un nodo v siempre que se cumpla:

$$\text{dist}(u, v) \leq d_u$$

Si una alarma percibe el sonido de otra, esta también se activa de forma automática, propagando su propio sonido hasta su rango d_v.

Objetivo:
Encontrar la **mínima cantidad** de alarmas que se deben activar manualmente para que **todas** las alarmas del árbol terminen activándose.

Hint: Pensar el problema en términos de componentes fuertemente conexas.

Problemas Propuestos - Centroid

Para practicar los temas de hoy:

- Ciel the Commander - Codeforces *(Easy)*
- Creating Offices - CSES *(Medium)*
- Journey of the Robber - Regional Latinoamericano *(Medium)*
- Hormigas - Selectivo OIA *(Hard)*

Problemas Propuestos - Virtual Trees

Para practicar los temas de hoy:

- Listing Tedious Paths - Subregional Brasil *(Easy)*
- Treehouse Telegram - Meta Hacker Cup *(Medium)*
- Electrical Efficiency - Codeforces Gym *(Medium/Hard)*
- Kuantum - TAP *(Hard)*

Problemas Propuestos - Ambos

Para practicar los temas de hoy:

- Sum of Tree Distance - AtCoder *(Easy/Medium - Sale con ambas técnicas)*

Outline

Consultas

Pueden consultarme durante estas semanas, o al telegram, o me pueden enviar un mail a:

- *eduardocarranzavelez@gmail.com*

Gracias por la atención.