Algoritmos en árboles

Miguel Raggi

Algoritmos en grafos
Escuela Nacional de Estudios Superiores
UNAM

8 de abril de 2018

Índice:

Índice:

Árboles

Definición

Recordemos que un árbol es un grafo conexo sin ciclos, y que siempre tiene $|V| - 1$ *aristas.*

Árboles

Definición

Recordemos que un árbol es un grafo conexo sin ciclos, y que siempre tiene $|V| - 1$ *aristas.*

Problema

Para calentar: programar función `bool is_tree(const Graph& G)` *(hint: son dos líneas). Esto irá en un archivo que se llame* `TreeAlgorithms.hpp` *(y su correspondiente* `TreeAlgorithms.cpp`*).*

Escoger la raíz

- Ahora, si ya tienes un árbol, muchas veces conviene pensarlo como un árbol enraizado.

Escoger la raíz

- Ahora, si ya tienes un árbol, muchas veces conviene pensarlo como un árbol enraizado.
- Es decir, en donde cada nodo tiene un “padre” e hijos.

Escoger la raíz

- Ahora, si ya tienes un árbol, muchas veces conviene pensarlo como un árbol enraizado.
- Es decir, en donde cada nodo tiene un "padre" e hijos.
- Dado un árbol y un vértice, ¿cómo represento y encuentro el padre de cada nodo?

Escoger la raíz

- Ahora, si ya tienes un árbol, muchas veces conviene pensarlo como un árbol enraizado.
- Es decir, en donde cada nodo tiene un "padre" e hijos.
- Dado un árbol y un vértice, ¿cómo represento y encuentro el padre de cada nodo?
- Lo más sencillo: un arreglo, donde en la posición i está el padre de i.

Escoger la raíz

- Ahora, si ya tienes un árbol, muchas veces conviene pensarlo como un árbol enraizado.
- Es decir, en donde cada nodo tiene un "padre" e hijos.
- Dado un árbol y un vértice, ¿cómo represento y encuentro el padre de cada nodo?
- Lo más sencillo: un arreglo, donde en la posición i está el padre de i.
- Para encontrarlo, DFS es lo más apropiado.

Escoger la raíz

- Ahora, si ya tienes un árbol, muchas veces conviene pensarlo como un árbol enraizado.
- Es decir, en donde cada nodo tiene un "padre" e hijos.
- Dado un árbol y un vértice, ¿cómo represento y encuentro el padre de cada nodo?
- Lo más sencillo: un arreglo, donde en la posición i está el padre de i.
- Para encontrarlo, DFS es lo más apropiado.

Ejercicio

Programa la función

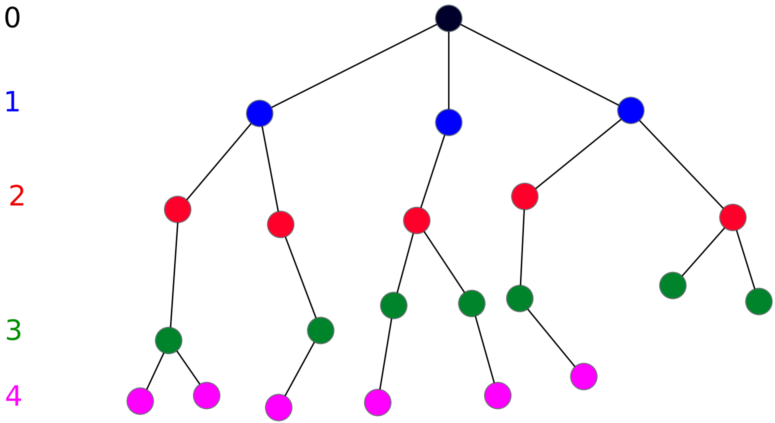
```
std::vector<Vertex> set_root(const Graph& G, Vertex v);
```

que te devuelva la lista de padres.

Nivel

Definición

El ***nivel*** *de un nodo es su distancia a la raíz. La* ***altura*** *del árbol es el nivel más bajo.*



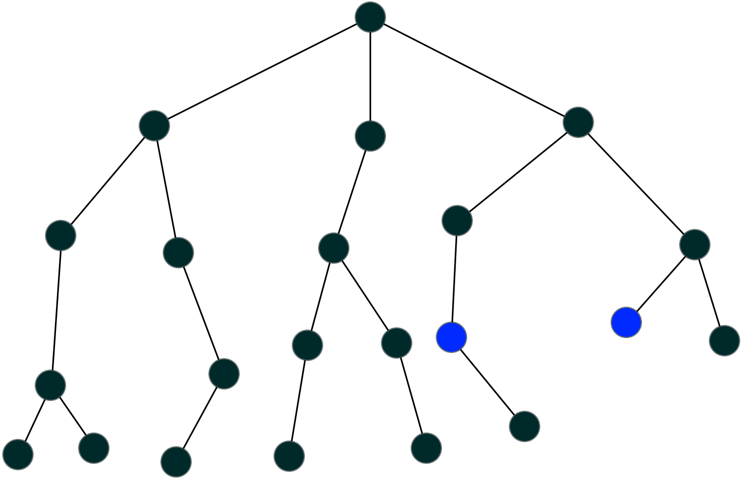
Índice:

Ancestro común más bajo

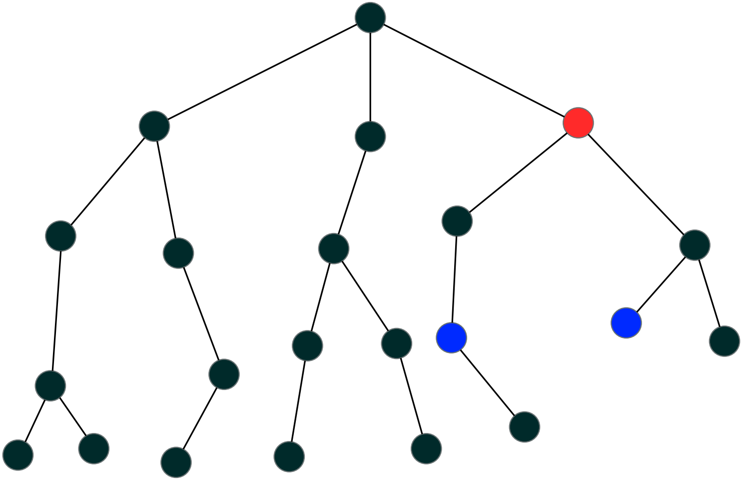
Definición

Dado un árbol G *enraizado y dos vértices* u *y* v*, su ancestro común más bajo o LCA es el ancestro común que no tiene descendientes que también sean ancestros comunes.*

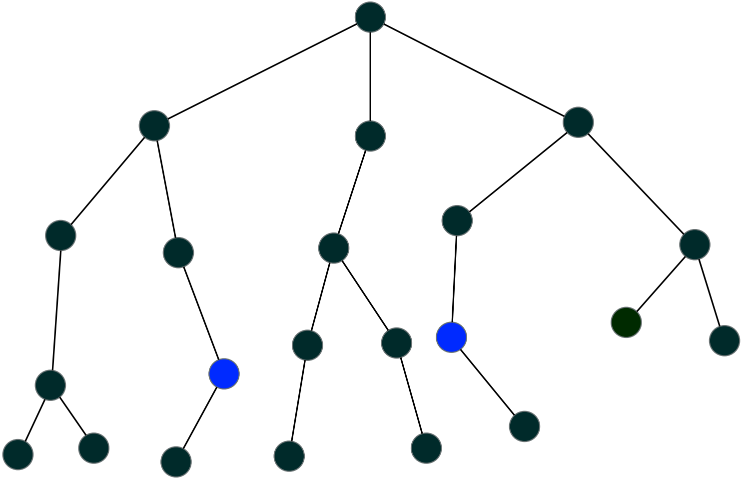
LCA



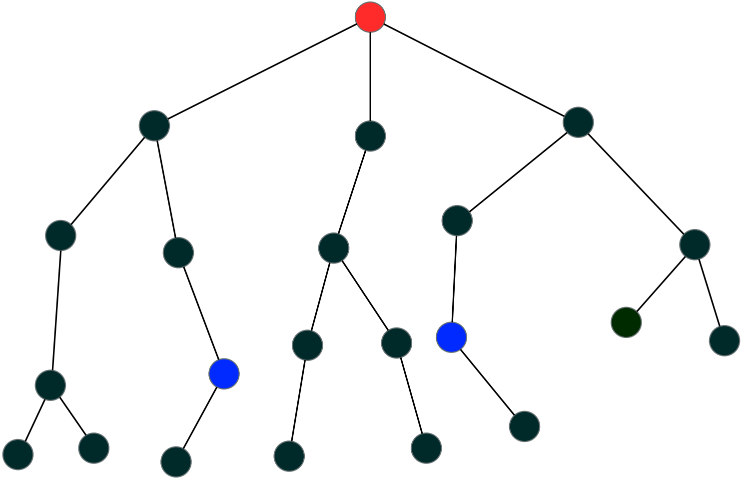
LCA



LCA



LCA



LCA: ¿cómo?

Problema

Dados dos nodos en el árbol, ¿cómo encuentro el ancestro común más bajo?

Método ingénuo

- Simplemente agarra u, $p(u)$, $p(p(u))$, etc. Y luego con v y ve si hay intersección.

Método ingénuo

- Simplemente agarra u, $p(u)$, $p(p(u))$, etc. Y luego con v y ve si hay intersección.
- Si sólo lo haremos una vez, no podemos hacerlo más eficiente que eso!

Método ingénuo

- Simplemente agarra u, $p(u)$, $p(p(u))$, etc. Y luego con v y ve si hay intersección.
- Si sólo lo haremos una vez, no podemos hacerlo más eficiente que eso!

Ejercicio

Programa la función

```
Vertex lowest_common_ancestor_naive(
          const std::vector<Vertex>& parents,
          Vertex u, Vertex v);
```

Muchas queries

- Pensemos que nos dan el árbol y luego nos van a preguntar varias veces el LCA de diferentes parejas de nodos.

Muchas queries

- Pensemos que nos dan el árbol y luego nos van a preguntar varias veces el LCA de diferentes parejas de nodos.
- Tal vez conviene trabajar un poco más para que luego podamos hacerlo más rápido.

Muchas queries

- Pensemos que nos dan el árbol y luego nos van a preguntar varias veces el LCA de diferentes parejas de nodos.
- Tal vez conviene trabajar un poco más para que luego podamos hacerlo más rápido.
- Vamos a ver tres algoritmos. Sea h la altura máxima (en el peor caso, $h = n$).

Muchas queries

- Pensemos que nos dan el árbol y luego nos van a preguntar varias veces el LCA de diferentes parejas de nodos.
- Tal vez conviene trabajar un poco más para que luego podamos hacerlo más rápido.
- Vamos a ver tres algoritmos. Sea h la altura máxima (en el peor caso, $h = n$).
    1. Descomposición raíz-cuadrática: cada query tarda $O(\sqrt{(}n))$ y preprocesar $O(n)$.

Muchas queries

- Pensemos que nos dan el árbol y luego nos van a preguntar varias veces el LCA de diferentes parejas de nodos.
- Tal vez conviene trabajar un poco más para que luego podamos hacerlo más rápido.
- Vamos a ver tres algoritmos. Sea h la altura máxima (en el peor caso, $h = n$).
    1. Descomposición raíz-cuadrática: cada query tarda $O(\sqrt{(}n))$ y preprocesar $O(n)$.
    2. 2^i ancestros: cada query tarda $O(\log(h))$ y preprocesar $O(n\log(n))$.

Muchas queries

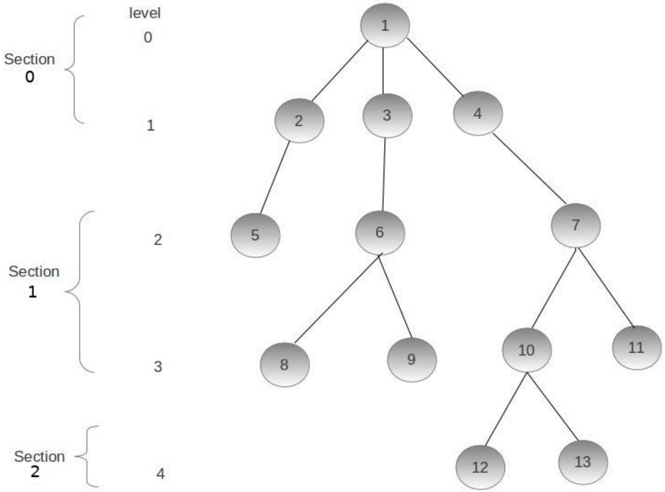
- Pensemos que nos dan el árbol y luego nos van a preguntar varias veces el LCA de diferentes parejas de nodos.
- Tal vez conviene trabajar un poco más para que luego podamos hacerlo más rápido.
- Vamos a ver tres algoritmos. Sea h la altura máxima (en el peor caso, $h = n$).
  1. Descomposición raíz-cuadrática: cada query tarda $O(\sqrt{(}n))$ y preprocesar $O(n)$.
  2. 2^i ancestros: cada query tarda $O(\log(h))$ y preprocesar $O(n \log(n))$.
  3. Range mininum queries: Cada query tarda $O(1)$ y preprocesar $O(n \log(n))$.

Descomposición raíz-cuadrática

Primero separa en $\sqrt{h}$ pedazos de tamaño $\sqrt{h}$.

Descomposición raíz-cuadrática

Primero separa en $\sqrt{h}$ pedazos de tamaño $\sqrt{h}$.



Descomposición raíz-cuadrática

- La sección de un nodo es simplemente

Descomposición raíz-cuadrática

- La sección de un nodo es simplemente $\lceil\sqrt{h}\rceil$-1, donde h es su altura.

Descomposición raíz-cuadrática

- La sección de un nodo es simplemente $\lceil\sqrt{h}\rceil$-1, donde h es su altura.
- Luego, a cada nodo encuéntrale su ancestro más bajo en la sección anterior.

Descomposición raíz-cuadrática

- La sección de un nodo es simplemente $\lceil\sqrt{h}\rceil$-1, donde h es su altura.
- Luego, a cada nodo encuéntrale su ancestro más bajo en la sección anterior.
- Por ejemplo, al 8 le encontraríamos el 3, y al 12 el 10.

Descomposición raíz-cuadrática

- La sección de un nodo es simplemente $\lceil\sqrt{h}\rceil$-1, donde h es su altura.
- Luego, a cada nodo encuéntrale su ancestro más bajo en la sección anterior.
- Por ejemplo, al 8 le encontraríamos el 3, y al 12 el 10.
- Ahora, ¿cómo le hacemos?

Casos

Hay dos casos. Sea A el arreglo anterior, o sea, $A[i]$ es el ancestro más bajo del nivel anterior.

Casos

Hay dos casos. Sea A el arreglo anterior, o sea, $A[i]$ es el ancestro más bajo del nivel anterior.

1. Si $A[u] \neq A[v]$.

Casos

Hay dos casos. Sea A el arreglo anterior, o sea, $A[i]$ es el ancestro más bajo del nivel anterior.

1. Si $A[u] \neq A[v]$.
2. Si $A[u] = A[v]$.

2^i-ancestros

- Ahora, imaginemos que para cada nodo tuviéramos algunos de sus ancestros.

2^i-ancestros

- Ahora, imaginemos que para cada nodo tuviéramos algunos de sus ancestros.
- Específicamente, supongamos que a cada nodo le encontramos:

2^i-ancestros

- Ahora, imaginemos que para cada nodo tuviéramos algunos de sus ancestros.
- Específicamente, supongamos que a cada nodo le encontramos:
    1. 0: Su padre (a distancia 1)

2^i-ancestros

- Ahora, imaginemos que para cada nodo tuviéramos algunos de sus ancestros.
- Específicamente, supongamos que a cada nodo le encontramos:
  1. 0: Su padre (a distancia 1)
  2. 1: Su abuelo (a distancia 2)

2^i-ancestros

- Ahora, imaginemos que para cada nodo tuviéramos algunos de sus ancestros.
- Específicamente, supongamos que a cada nodo le encontramos:
  1. 0: Su padre (a distancia 1)
  2. 1: Su abuelo (a distancia 2)
  3. 2: Su ancestro a distancia 4

2^i-ancestros

- Ahora, imaginemos que para cada nodo tuviéramos algunos de sus ancestros.
- Específicamente, supongamos que a cada nodo le encontramos:
  1. 0: Su padre (a distancia 1)
  2. 1: Su abuelo (a distancia 2)
  3. 2: Su ancestro a distancia 4
  4. 3: Su ancestro a distancia 8

2^i-ancestros

- Ahora, imaginemos que para cada nodo tuviéramos algunos de sus ancestros.
- Específicamente, supongamos que a cada nodo le encontramos:
  1. 0: Su padre (a distancia 1)
  2. 1: Su abuelo (a distancia 2)
  3. 2: Su ancestro a distancia 4
  4. 3: Su ancestro a distancia 8
  5. ...

2^i-ancestros

- Ahora, imaginemos que para cada nodo tuviéramos algunos de sus ancestros.
- Específicamente, supongamos que a cada nodo le encontramos:
  1. 0: Su padre (a distancia 1)
  2. 1: Su abuelo (a distancia 2)
  3. 2: Su ancestro a distancia 4
  4. 3: Su ancestro a distancia 8
  5. ...
- Entonces hay dos problemas:

2^i-ancestros

- Ahora, imaginemos que para cada nodo tuviéramos algunos de sus ancestros.
- Específicamente, supongamos que a cada nodo le encontramos:
  1. 0: Su padre (a distancia 1)
  2. 1: Su abuelo (a distancia 2)
  3. 2: Su ancestro a distancia 4
  4. 3: Su ancestro a distancia 8
  5. ...
- Entonces hay dos problemas:
  - ¿Cómo encontramos esta información?

2^i-ancestros

- Ahora, imaginemos que para cada nodo tuviéramos algunos de sus ancestros.
- Específicamente, supongamos que a cada nodo le encontramos:
  1. 0: Su padre (a distancia 1)
  2. 1: Su abuelo (a distancia 2)
  3. 2: Su ancestro a distancia 4
  4. 3: Su ancestro a distancia 8
  5. ...
- Entonces hay dos problemas:
  - ¿Cómo encontramos esta información?
  - Teniendo esta info, ¿cómo procedemos?

Encontrar mis ancestros potencias de dos

Esa parte es sencilla:

- Representamos con un arreglo de arreglos A así: $A[v][i]$ será el 2^i-ésimo ancestro de v (si es que existe).

Encontrar mis ancestros potencias de dos

Esa parte es sencilla:

- Representamos con un arreglo de arreglos A así: $A[v][i]$ será el 2^i-ésimo ancestro de v (si es que existe).
- Mi 2^i-ésimo ancestro es simplemente el 2^{i-1}-ésimo ancestro de mi 2^{i-1}-ésimo ancestro.

Encontrar mis ancestros potencias de dos

Esa parte es sencilla:

- Representamos con un arreglo de arreglos A así: $A[v][i]$ será el 2^i-ésimo ancestro de v (si es que existe).
- Mi 2^i-ésimo ancestro es simplemente el 2^{i-1}-ésimo ancestro de mi 2^{i-1}-ésimo ancestro.
- Entonces vamos a encontrar los 2^0-ancestros de todos, luego los 2^1-ancestros, etc.

Encontrar mis ancestros potencias de dos

Esa parte es sencilla:

- Representamos con un arreglo de arreglos A así: $A[v][i]$ será el 2^i-ésimo ancestro de v (si es que existe).
- Mi 2^i-ésimo ancestro es simplemente el 2^{i-1}-ésimo ancestro de mi 2^{i-1}-ésimo ancestro.
- Entonces vamos a encontrar los 2^0-ancestros de todos, luego los 2^1-ancestros, etc.
- Ni siquiera necesito hacer DFS, porque si ya todos los nodos tienen su 2^{i-1}-ésimo ancestro, sacar el 2^i-ésimo ancestro es trivial:

$$A[v][i] = A[A[v][i-1]][i-1]$$

Notación

Definición

Sea L el arreglo de los niveles de cada nodo. Es decir, $L[v]$ es el nivel del nodo v.

Notación

Definición

Sea L el arreglo de los niveles de cada nodo. Es decir, $L[v]$ es el nivel del nodo v.

Definición

Llamemos $x = LCA(u, v)$, lo que queremos encontrar.

Notación

Definición

Sea L el arreglo de los niveles de cada nodo. Es decir, $L[v]$ es el nivel del nodo v.

Definición

Llamemos $x = LCA(u, v)$, lo que queremos encontrar.

Observación: Obviamente, $L[x] \leq \min\{L[u], L[v]\}$.

Notación

Definición

Sea L el arreglo de los niveles de cada nodo. Es decir, $L[v]$ es el nivel del nodo v.

Definición

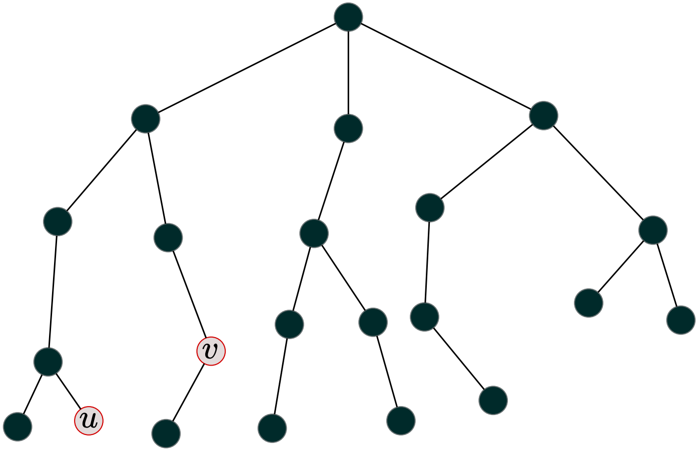
Llamemos $x = LCA(u, v)$, lo que queremos encontrar.

Observación: Obviamente, $L[x] \leq \text{mín}\{L[u], L[v]\}$.

SPG, supongamos que u está más abajo que v (o sea, su nivel es más grande).

Encontrar el LCA dados los ancestros

Esta parte no es tan sencilla.



Nivelar

1. Podemos "subir" u, con búsqueda binaria, hasta que esté en el mismo nivel que v. Entonces ya pensemos que u está al mismo nivel que v.

Nivelar

1. Podemos "subir" u, con búsqueda binaria, hasta que esté en el mismo nivel que v. Entonces ya pensemos que u está al mismo nivel que v.
2. Si $u = v$, pues ya, ese es su LCA.

Nivelar

1. Podemos “subir” u, con búsqueda binaria, hasta que esté en el mismo nivel que v. Entonces ya pensemos que u está al mismo nivel que v.
2. Si $u = v$, pues ya, ese es su LCA.
3. Si no, entonces fijémonos en su lista de ancestros potencia de dos:

$$A[u] = [p_1, p_2, ...]$$

$$A[v] = [q_1, q_2, ...]$$

Nivelar

1. Podemos "subir" u, con búsqueda binaria, hasta que esté en el mismo nivel que v. Entonces ya pensemos que u está al mismo nivel que v.
2. Si $u = v$, pues ya, ese es su LCA.
3. Si no, entonces fijémonos en su lista de ancestros potencia de dos:

$$A[u] = [p_1, p_2, ...]$$

$$A[v] = [q_1, q_2, ...]$$

4. Si en algún momento coinciden, ya van a coincidir para siempre.

Nivelar

1. Podemos "subir" u, con búsqueda binaria, hasta que esté en el mismo nivel que v. Entonces ya pensemos que u está al mismo nivel que v.
2. Si $u = v$, pues ya, ese es su LCA.
3. Si no, entonces fijémonos en su lista de ancestros potencia de dos:

$$A[u] = [p_1, p_2, ...]$$

$$A[v] = [q_1, q_2, ...]$$

4. Si en algún momento coinciden, ya van a coincidir para siempre.
5. Entonces con búsqueda binaria, podemos encontrar el último momento que no coinciden y subir u y v a ese punto.

Nivelar

1. Podemos “subir” u, con búsqueda binaria, hasta que esté en el mismo nivel que v. Entonces ya pensemos que u está al mismo nivel que v.
2. Si $u = v$, pues ya, ese es su LCA.
3. Si no, entonces fijémonos en su lista de ancestros potencia de dos:

$$A[u] = [p_1, p_2, ...]$$

$$A[v] = [q_1, q_2, ...]$$

4. Si en algún momento coinciden, ya van a coincidir para siempre.
5. Entonces con búsqueda binaria, podemos encontrar el último momento que no coinciden y subir u y v a ese punto.
6. Es decir, sea r tal que $p_r \neq q_r$ pero $p_{r+1} = q_{r+1}$ (o r es el último).

Nivelar

1. Podemos "subir" u, con búsqueda binaria, hasta que esté en el mismo nivel que v. Entonces ya pensemos que u está al mismo nivel que v.
2. Si $u = v$, pues ya, ese es su LCA.
3. Si no, entonces fijémonos en su lista de ancestros potencia de dos:

$$A[u] = [p_1, p_2, ...]$$

$$A[v] = [q_1, q_2, ...]$$

4. Si en algún momento coinciden, ya van a coincidir para siempre.
5. Entonces con búsqueda binaria, podemos encontrar el último momento que no coinciden y subir u y v a ese punto.
6. Es decir, sea r tal que $p_r \neq q_r$ pero $p_{r+1} = q_{r+1}$ (o r es el último).
7. Asignamos $u := p_r$ y $v := q_r$ y vamos al paso 2

Nivelar

1. Podemos "subir" u, con búsqueda binaria, hasta que esté en el mismo nivel que v. Entonces ya pensemos que u está al mismo nivel que v.
2. Si $u = v$, pues ya, ese es su LCA.
3. Si no, entonces fijémonos en su lista de ancestros potencia de dos:

$$A[u] = [p_1, p_2, ...]$$

$$A[v] = [q_1, q_2, ...]$$

4. Si en algún momento coinciden, ya van a coincidir para siempre.
5. Entonces con búsqueda binaria, podemos encontrar el último momento que no coinciden y subir u y v a ese punto.
6. Es decir, sea r tal que $p_r \neq q_r$ pero $p_{r+1} = q_{r+1}$ (o r es el último).
7. Asignamos $u := p_r$ y $v := q_r$ y vamos al paso 2
8. Si $u! = v$ pero $p_1 = q_1$, entonces p_1 es el lca.

A programar!!

A mi me tomó como casi 3 horas que funcionara esto :S. Pero confío en uds!

Necesitamos hacer lo siguiente:

- `std::vector<int> nivel(const Graph& G, Vertex root);`
- `class LCAWithPowersOfTwo` o algo así, que guardará los ancestros y el nivel.
- Dentro de `LCAWithPowersOfTwo`:
- El constructor se encarga de llenar `A` y `L`.
- Pública: `Vertex FindLCA(Vertex u, Vertex v)`
- Privada: `Vertex AncestorAtLevel(Vertex u, int lvl)`

Range minimum queries

Primero vamos a estudiar un problema que es aparentemente completamente diferente.

Range minimum queries

Primero vamos a estudiar un problema que es aparentemente completamente diferente.

Problema

Sea A *un arreglo (fijo) de tamaño* n*. Dados dos números,* $0 \leq L \leq R < n$*, queremos encontrar el índice con el menor elemento de* A *en el intervalo* $[L, R)$*.*

Range minimum queries

Primero vamos a estudiar un problema que es aparentemente completamente diferente.

Problema

Sea A un arreglo (fijo) de tamaño n. Dados dos números, $0 \leq L \leq R < n$, queremos encontrar el índice con el menor elemento de A en el intervalo $[L, R)$.

- Claro, si nada más lo haremos una vez, no hay otra cosa que hacer que explorar todos los números $A[i]$ para $i \in [L, R)$

Range minimum queries

Primero vamos a estudiar un problema que es aparentemente completamente diferente.

Problema

Sea A un arreglo (fijo) de tamaño n. Dados dos números, $0 \leq L \leq R < n$, queremos encontrar el índice con el menor elemento de A en el intervalo $[L, R)$.

- Claro, si nada más lo haremos una vez, no hay otra cosa que hacer que explorar todos los números $A[i]$ para $i \in [L, R)$
- Pero estamos pensando que nos harán muchísimos queries.

Solución tonta

- Lo primero que podríamos hacer es simplemente guardar, para cada pareja posible $[L, R]$ la respuesta.

Solución tonta

- Lo primero que podríamos hacer es simplemente guardar, para cada pareja posible $[L, R]$ la respuesta.
- Así podríamos contestar en tiempo lineal.

Solución tonta

- Lo primero que podríamos hacer es simplemente guardar, para cada pareja posible $[L, R]$ la respuesta.
- Así podríamos contestar en tiempo lineal.
- PERO está super chafa eso!

Solución tonta

- Lo primero que podríamos hacer es simplemente guardar, para cada pareja posible $[L, R]$ la respuesta.
- Así podríamos contestar en tiempo lineal.
- PERO está super chafa eso!
- El preprocesamiento es cuadrático y la catnidad de memoria es cuadrática.

Solución buena

- En vez de eso, vamos a da una solución que usa $O(n \log(n))$ memoria y preprocesamiento, y las queries las responde en $O(1)$.

Solución buena

- En vez de eso, vamos a da una solución que usa $O(n \log(n))$ memoria y preprocesamiento, y las queries las responde en $O(1)$.
- A cada índice x le guardaremos un arreglo en donde guardaremos los menores índices en:

Solución buena

- En vez de eso, vamos a da una solución que usa $O(n \log(n))$ memoria y preprocesamiento, y las queries las responde en $O(1)$.
- A cada índice x le guardaremos un arreglo en donde guardaremos los menores índices en:
  - $[x, x+1)$,

Solución buena

- En vez de eso, vamos a da una solución que usa $O(n \log(n))$ memoria y preprocesamiento, y las queries las responde en $O(1)$.
- A cada índice x le guardaremos un arreglo en donde guardaremos los menores índices en:
  - $[x, x+1)$,
  - $[x, x+2)$,

Solución buena

- En vez de eso, vamos a da una solución que usa $O(n \log(n))$ memoria y preprocesamiento, y las queries las responde en $O(1)$.
- A cada índice x le guardaremos un arreglo en donde guardaremos los menores índices en:
  - $[x, x+1)$,
  - $[x, x+2)$,
  - $[x, x+4)$,

Solución buena

- En vez de eso, vamos a da una solución que usa $O(n \log(n))$ memoria y preprocesamiento, y las queries las responde en $O(1)$.
- A cada índice x le guardaremos un arreglo en donde guardaremos los menores índices en:
  - $[x, x+1)$,
  - $[x, x+2)$,
  - $[x, x+4)$,
  - $[x, x+8)$,
  - ...
- Así, ¿cuánto guardamos?

Solución buena

- En vez de eso, vamos a da una solución que usa $O(n\log(n))$ memoria y preprocesamiento, y las queries las responde en $O(1)$.
- A cada índice x le guardaremos un arreglo en donde guardaremos los menores índices en:
  - $[x, x+1)$,
  - $[x, x+2)$,
  - $[x, x+4)$,
  - $[x, x+8)$,
  - ...
- Así, ¿cuánto guardamos? Pues $\log(n)$ por cada vértice.

Solución buena

- En vez de eso, vamos a da una solución que usa $O(n\log(n))$ memoria y preprocesamiento, y las queries las responde en $O(1)$.
- A cada índice x le guardaremos un arreglo en donde guardaremos los menores índices en:
  - $[x, x+1)$,
  - $[x, x+2)$,
  - $[x, x+4)$,
  - $[x, x+8)$,
  - ...
- Así, ¿cuánto guardamos? Pues $\log(n)$ por cada vértice.
- ¿Cómo conseguimos esto?

Solución buena

- En vez de eso, vamos a da una solución que usa $O(n \log(n))$ memoria y preprocesamiento, y las queries las responde en $O(1)$.
- A cada índice x le guardaremos un arreglo en donde guardaremos los menores índices en:
  - $[x, x+1)$,
  - $[x, x+2)$,
  - $[x, x+4)$,
  - $[x, x+8)$,
  - ...
- Así, ¿cuánto guardamos? Pues $\log(n)$ por cada vértice.
- ¿Cómo conseguimos esto?
- De la mismita manera que en LCA con potencias de dos:

$$[x, x+2^k) = [x, x+2^{k-1}) \cup [x+2^{k-1}, x+2^k)$$

- Pero ahora... cómo respondemos una query?

- Pero ahora... cómo respondemos una query?
- Muy fácil:

$$[L, R) = [L, L + 2^i) \cup [R - 2^i, R),$$

- Pero ahora... cómo respondemos una query?
- Muy fácil:

$$[L, R) = [L, L + 2^i) \cup [R - 2^i, R),$$

donde i es el valor más grande tal que $L + 2^i \leq R$.

- Pero ahora... cómo respondemos una query?
- Muy fácil:
$$[L, R) = [L, L + 2^i) \cup [R - 2^i, R),$$
donde i es el valor más grande tal que $L + 2^i \leq R$.
- O sea, $i = \log_2(R - L)$.

- Pero ahora... cómo respondemos una query?
- Muy fácil:
$$[L, R) = [L, L + 2^i) \cup [R - 2^i, R),$$
donde i es el valor más grande tal que $L + 2^i \leq R$.
- O sea, $i = \log_2(R - L)$.
- Ya! A programar!

RMQ aplicado a LCA

¿Cómo aplicamos RMQ a LCA?

RMQ aplicado a LCA

¿Cómo aplicamos RMQ a LCA?

- Si se fijan, hay exactamente un camino por cada pareja de vértices en un árbol.

RMQ aplicado a LCA

¿Cómo aplicamos RMQ a LCA?

- Si se fijan, hay exactamente un camino por cada pareja de vértices en un árbol.
- Lo que queremos es encontrar el vértice más alto en este caminito.

RMQ aplicado a LCA

¿Cómo aplicamos RMQ a LCA?

- Si se fijan, hay exactamente un camino por cada pareja de vértices en un árbol.
- Lo que queremos es encontrar el vértice más alto en este caminito.
- Es que en realidad RMQ es un range minimum query (en el orden dado por la altura) de un arreglo.

RMQ aplicado a LCA

¿Cómo aplicamos RMQ a LCA?

- Si se fijan, hay exactamente un camino por cada pareja de vértices en un árbol.
- Lo que queremos es encontrar el vértice más alto en este caminito.
- Es que en realidad RMQ es un range minimum query (en el orden dado por la altura) de un arreglo.
- ¿Qué arreglo?

RMQ aplicado a LCA

¿Cómo aplicamos RMQ a LCA?

- Si se fijan, hay exactamente un camino por cada pareja de vértices en un árbol.
- Lo que queremos es encontrar el vértice más alto en este caminito.
- Es que en realidad RMQ es un range minimum query (en el orden dado por la altura) de un arreglo.
- ¿Qué arreglo?
- Pues uno en que dados (u, v) vértices, pueda localizar un subarreglo que me de el caminito de vértices

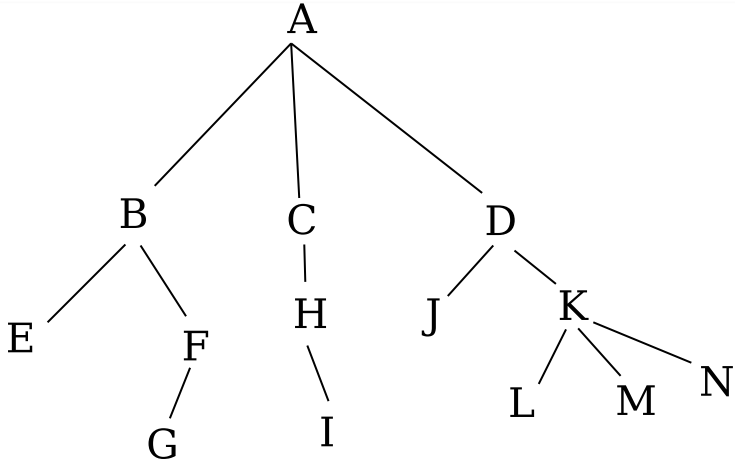
RMQ aplicado a LCA

¿Cómo aplicamos RMQ a LCA?

- Si se fijan, hay exactamente un camino por cada pareja de vértices en un árbol.
- Lo que queremos es encontrar el vértice más alto en este caminito.
- Es que en realidad RMQ es un range minimum query (en el orden dado por la altura) de un arreglo.
- ¿Qué arreglo?
- Pues uno en que dados (u, v) vértices, pueda localizar un subarreglo que me de el caminito de vértices (y quizás otros vértices más bajos)

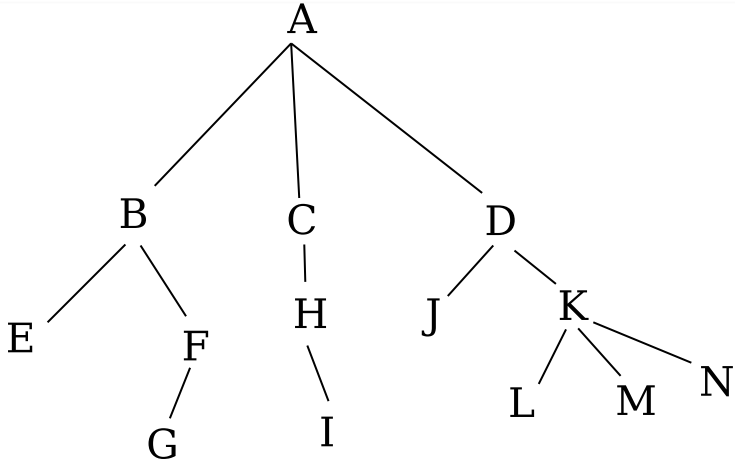
Tours de Euler

Creo que se ve con un ejemplo:



Tours de Euler

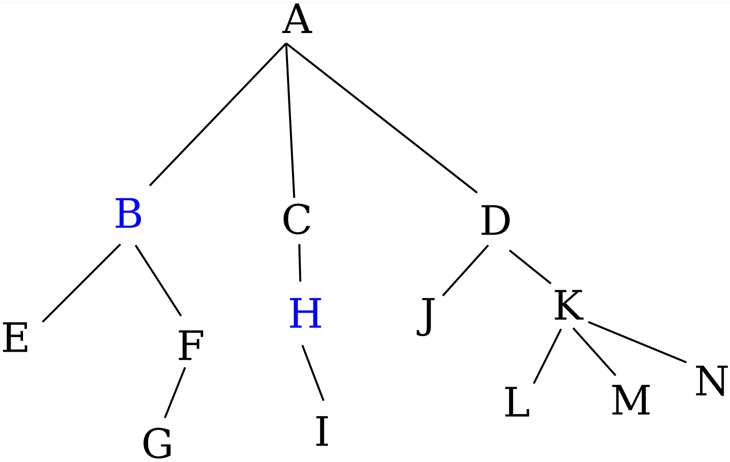
Creo que se ve con un ejemplo:



ABEBFGFBACHIHCADJDKLKMKNKDA

Tours de Euler

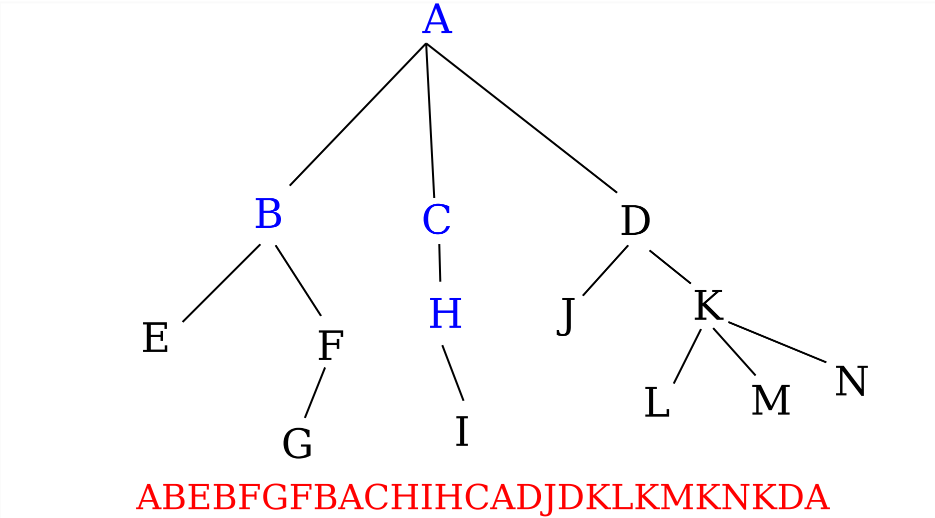
Creo que se ve con un ejemplo:



ABEBFGFBACHIHCADJDKLKMKNKDA

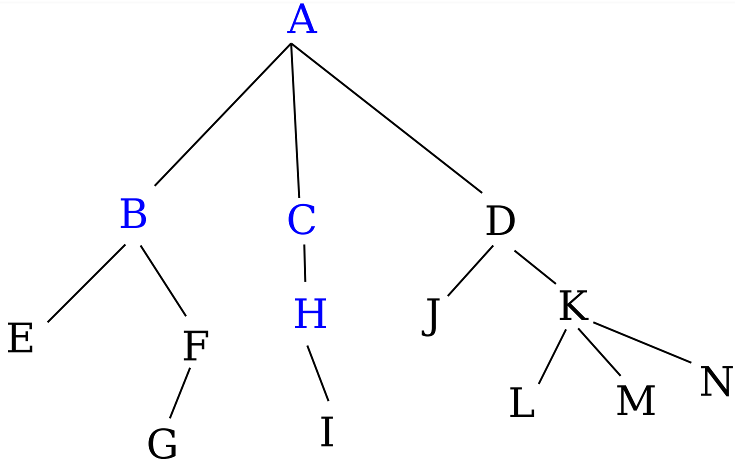
Tours de Euler

Creo que se ve con un ejemplo:



Tours de Euler

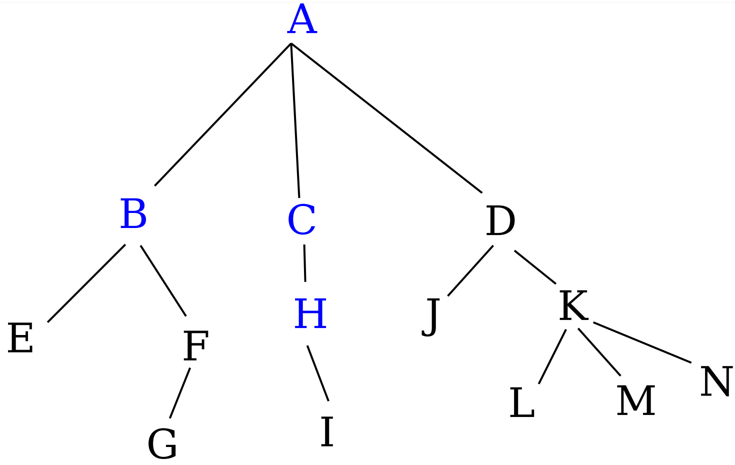
Creo que se ve con un ejemplo:



ABEBFGFBACHIHCADJDKLKMKNKDA

Tours de Euler

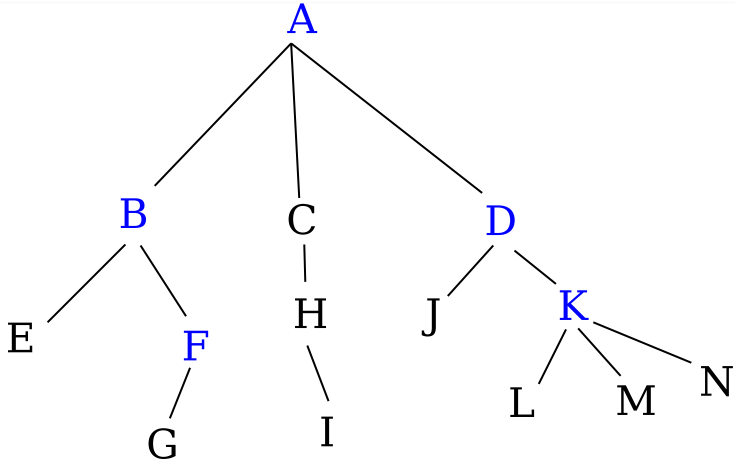
Creo que se ve con un ejemplo:



ABEBFGFBACHIHCADJDKLKMKNKDA

Tours de Euler

Creo que se ve con un ejemplo:



ABEBFGFBACHIHCADJDKLKMKNKDA

Tours de Euler

- Digamos que nos preguntamos por (u, v).

Tours de Euler

- Digamos que nos preguntamos por (u, v).
- Fijémonos en la primera vez que aparece u y en la última que aparece v.

Tours de Euler

- Digamos que nos preguntamos por (u, v).
- Fijémonos en la primera vez que aparece u y en la última que aparece v.
- Si simplemente la primera de u está después de la última de v, ese es el query.

Tours de Euler

- Digamos que nos preguntamos por (u, v).
- Fijémonos en la primera vez que aparece u y en la última que aparece v.
- Si simplemente la primera de u está después de la última de v, ese es el query.
- Si la primera de v está después de la última de u, pues ya, también.

Tours de Euler

- Digamos que nos preguntamos por (u, v).
- Fijémonos en la primera vez que aparece u y en la última que aparece v.
- Si simplemente la primera de u está después de la última de v, ese es el query.
- Si la primera de v está después de la última de u, pues ya, también.
- Si no, es que uno es ancestro del otro, y entonces regresamos el que tenga menor nivel.

Tamaño y consideraciones

- Entonces a cada vértice le guardamos el índice de su primer y última aparición en el tour de euler.

Tamaño y consideraciones

- Entonces a cada vértice le guardamos el índice de su primer y última aparición en el tour de euler.
- Luego hacemos un range minimum query con la primera de uno y la última del otro, (o simplemente regresamos uno de ellos).

Tamaño y consideraciones

- Entonces a cada vértice le guardamos el índice de su primer y última aparición en el tour de euler.
- Luego hacemos un range minimum query con la primera de uno y la última del otro, (o simplemente regresamos uno de ellos).
- P: ¿Qué tamaño tiene el arreglo total?

Tamaño y consideraciones

- Entonces a cada vértice le guardamos el índice de su primer y última aparición en el tour de euler.
- Luego hacemos un range minimum query con la primera de uno y la última del otro, (o simplemente regresamos uno de ellos).
- P: ¿Qué tamaño tiene el arreglo total?
- R: Pues cada arista la pasamos dos veces: una de ida y otra de vuelta.

Tamaño y consideraciones

- Entonces a cada vértice le guardamos el índice de su primer y última aparición en el tour de euler.
- Luego hacemos un range minimum query con la primera de uno y la última del otro, (o simplemente regresamos uno de ellos).
- P: ¿Qué tamaño tiene el arreglo total?
- R: Pues cada arista la pasamos dos veces: una de ida y otra de vuelta.
- Así que tenemos $2n - 2 + 1$ vértices en el arreglo. Bien!