Universidad de Concepción

Grafos II y Mates CIPC 2024

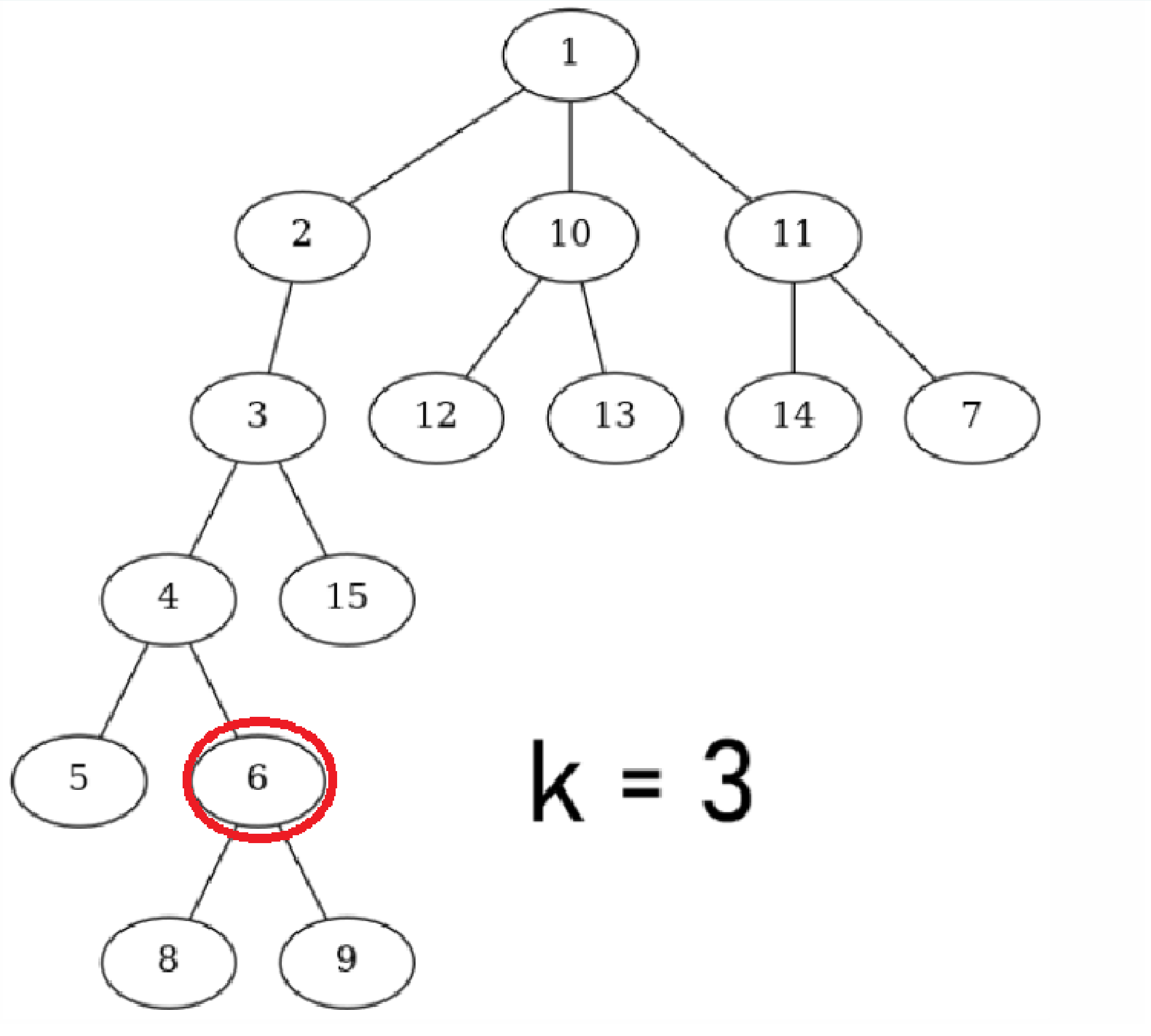
por Marcelo Lemus (PUC)

Arboles

Problema: Dado un arbol, queremos conseguir el k-esimo ancestro de un nodo "u" de forma eficiente. Queremos hacer muchas consultas de este tipo.

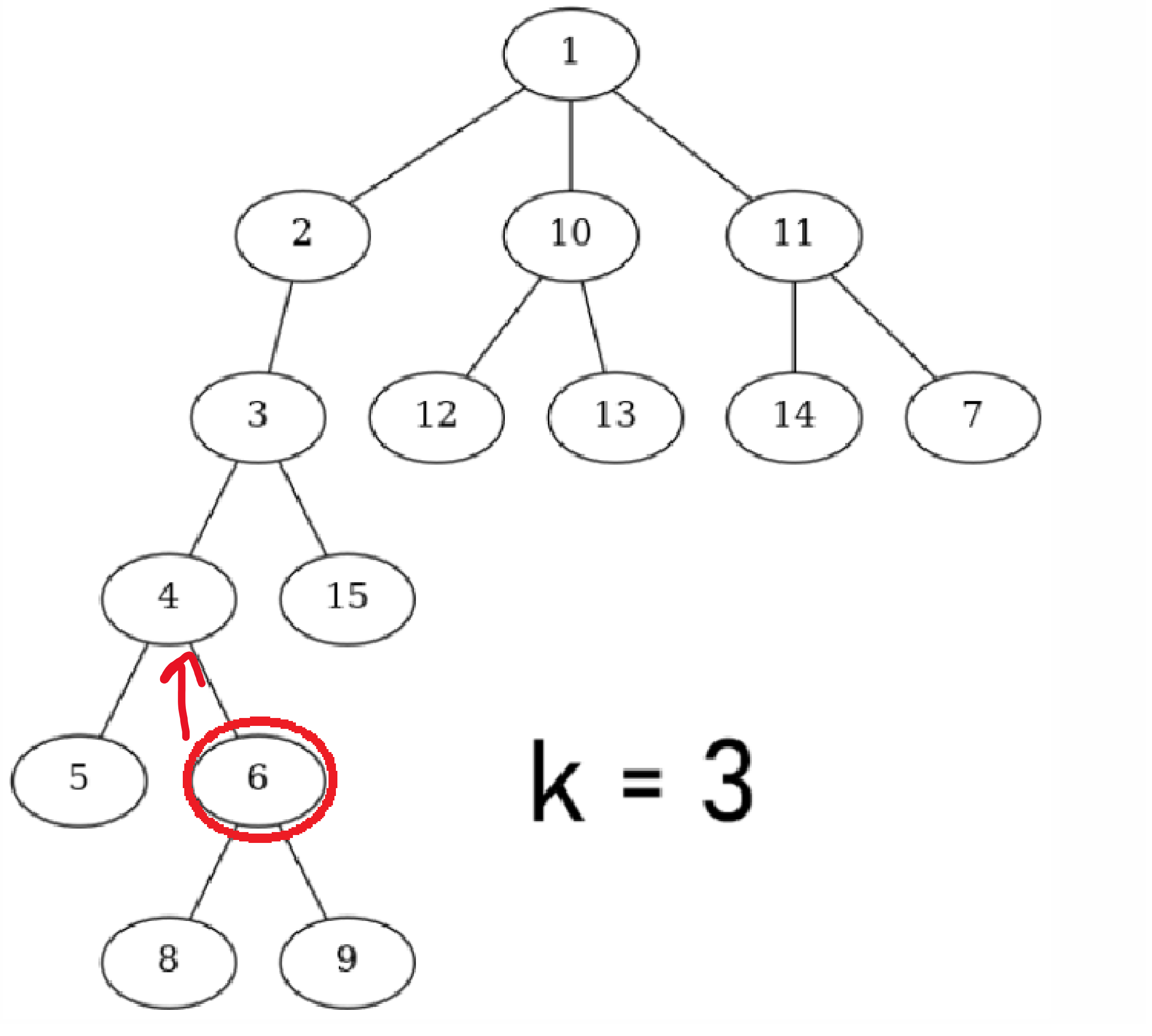
Arboles

Problema: Dado un árbol, queremos conseguir el k-esimo ancestro de un nodo "u" de forma eficiente. Queremos hacer muchas consultas de este tipo.



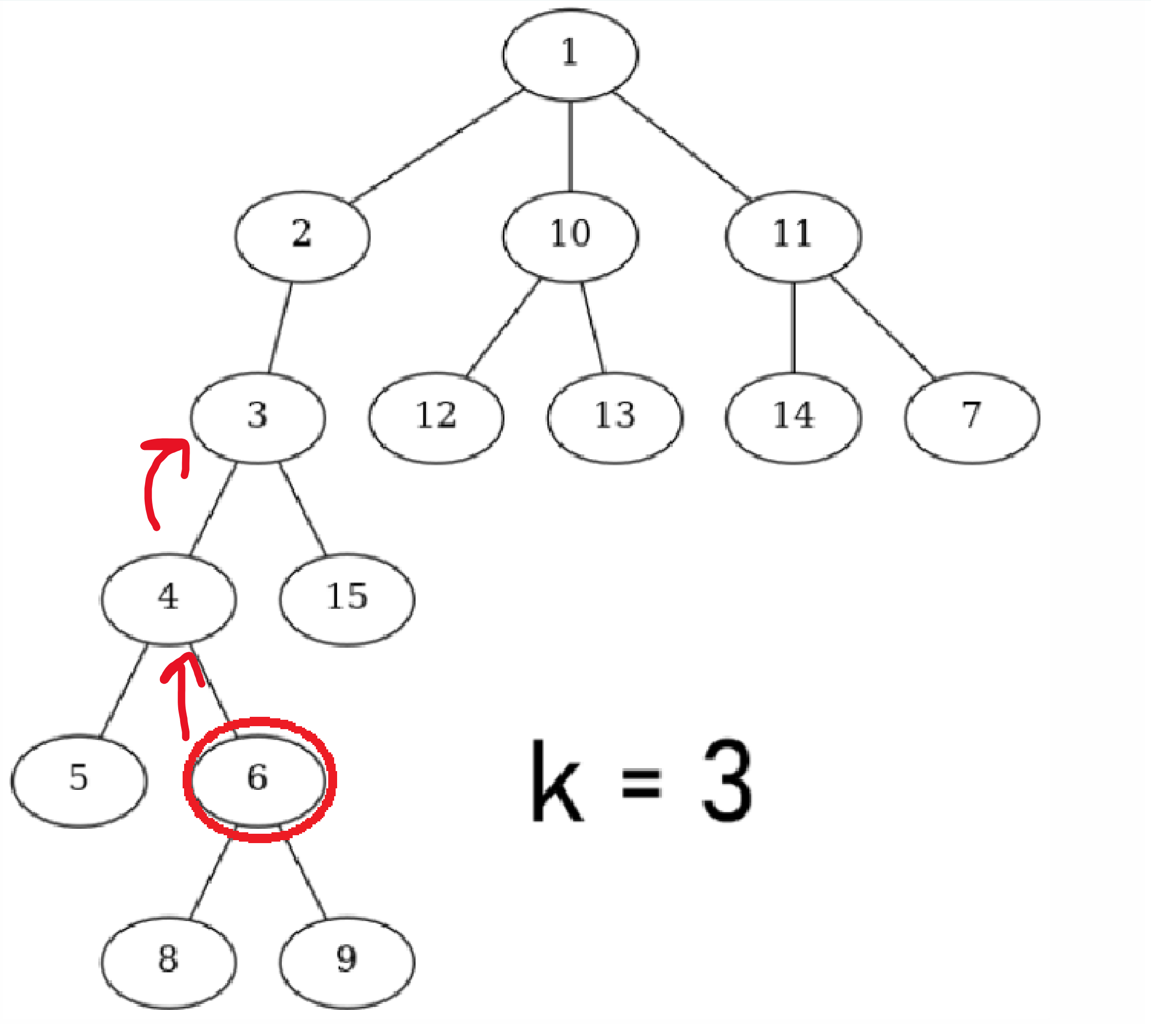
Arboles

Problema: Dado un árbol, queremos conseguir el k-esimo ancestro de un nodo "u" de forma eficiente. Queremos hacer muchas consultas de este tipo.



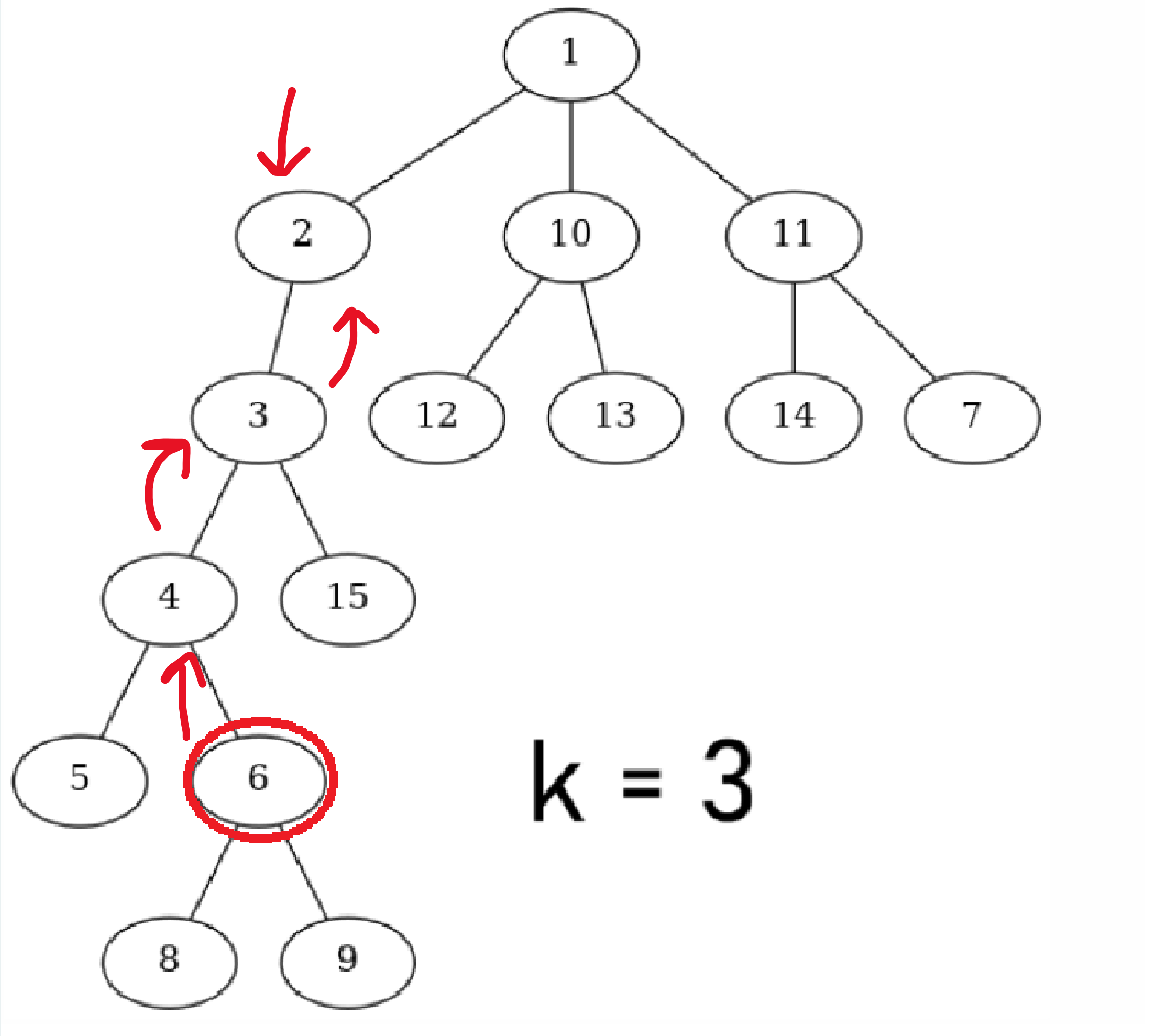
Arboles

Problema: Dado un árbol, queremos conseguir el k-esimo ancestro de un nodo "u" de forma eficiente. Queremos hacer muchas consultas de este tipo.



Arboles

Problema: Dado un árbol, queremos conseguir el k-esimo ancestro de un nodo "u" de forma eficiente. Queremos hacer muchas consultas de este tipo.



Arboles

Iterar k veces, cada vez, subo un nodo hacia arriba

```
for(int i = 0; i<k; ++i){
    u = parent[u];
}
```

Arboles

Iterar k veces, cada vez, subo un nodo hacia arriba

```
for(int i = 0; i<k; ++i){
    u = parent[u];
}
```

O(k) -> O(N) con muchas queries -> O(Q*N)

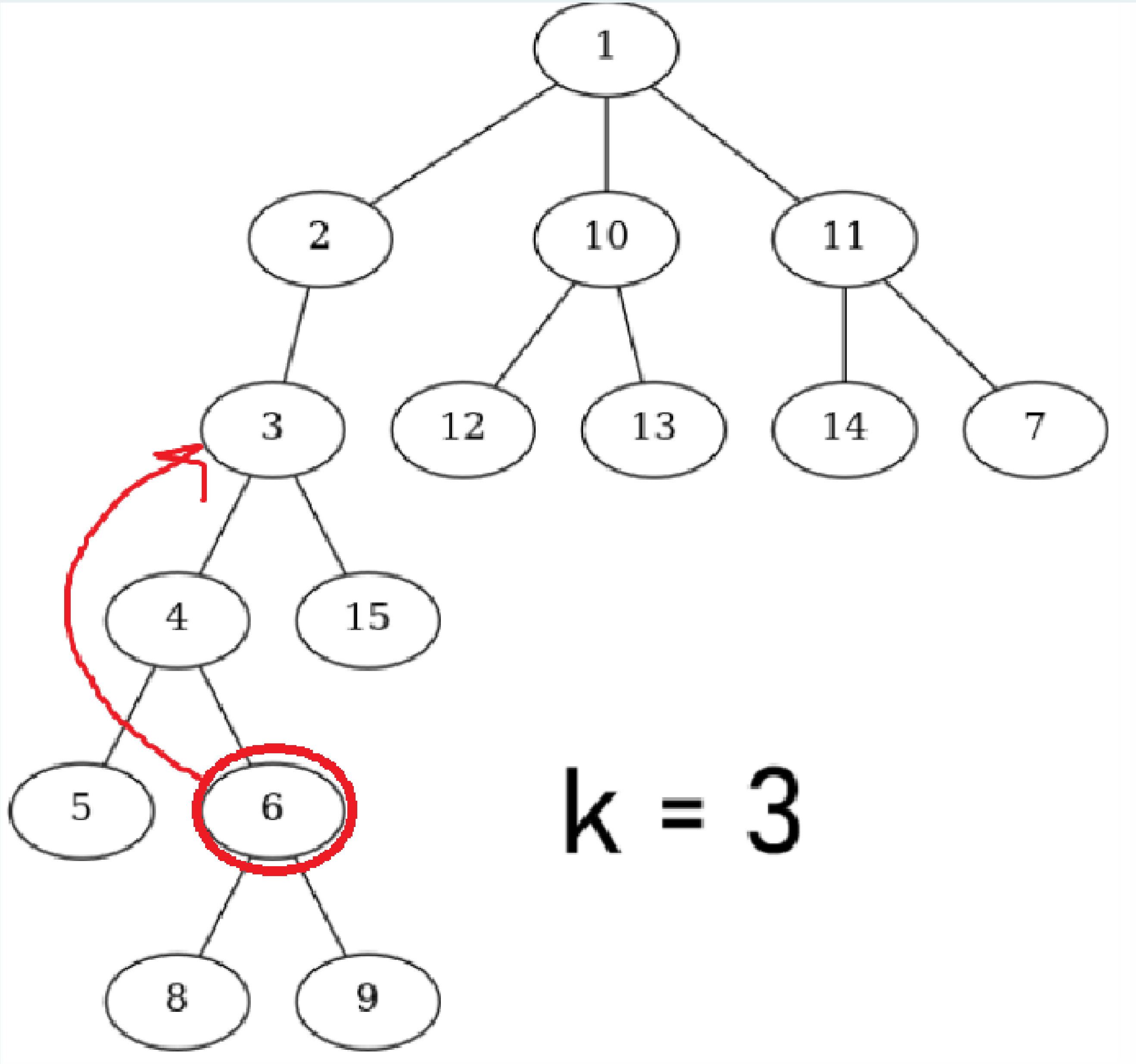
Como hacerlo más rapido

Podemos tomar la representacion binaria de k, sabiendo que tiene a lo mucho log_2 bits y precalcular los "saltos" en potencias de 2

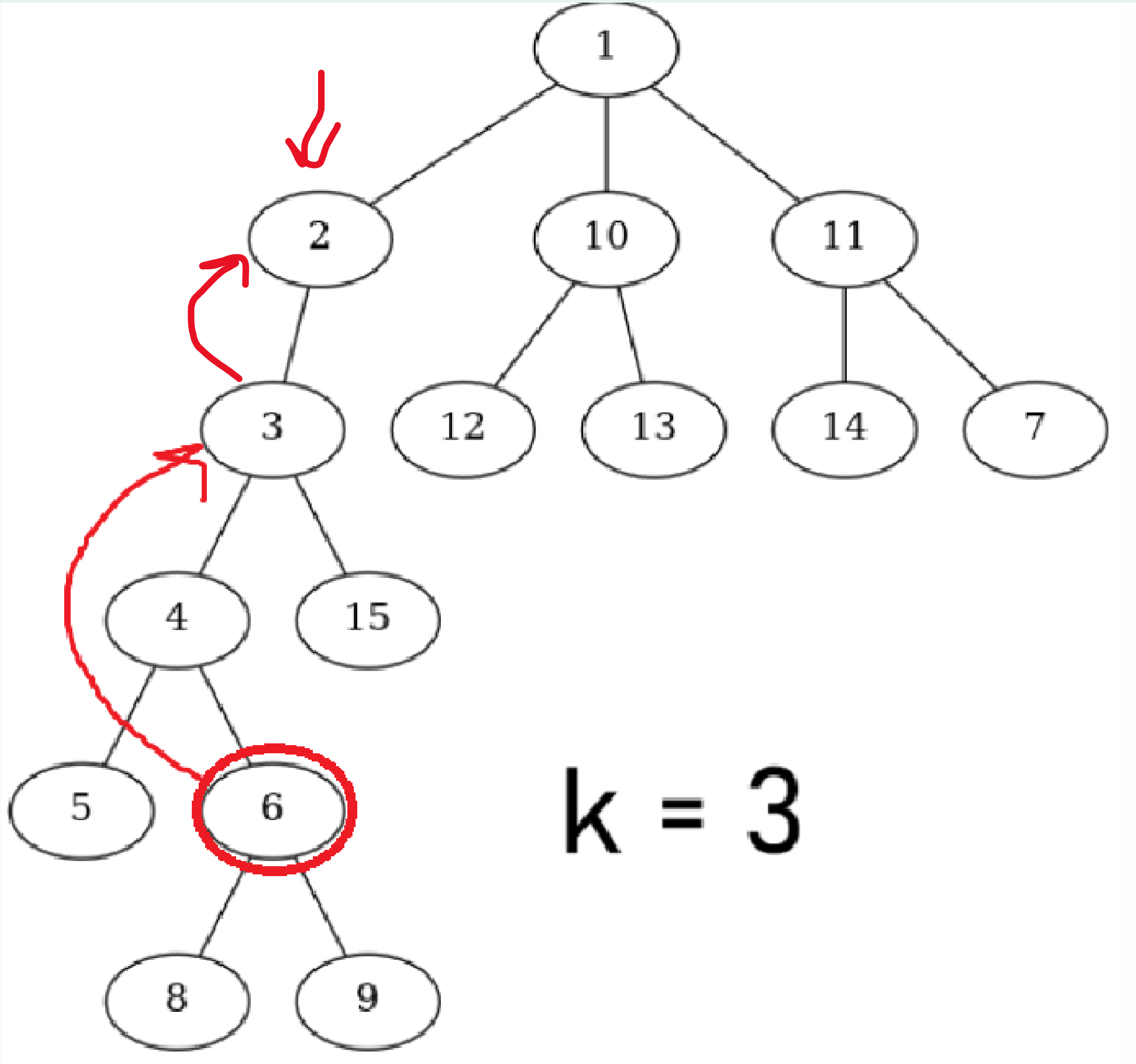
k = 19 = 10011 = 16+2+1

Si precalculamos estos saltos en potencias de 2 por cada nodo, podemos resolver cada consulta en tiempo O(log_2(k))

Binary Lifting



Binary Lifting



Binary Lifting

```cpp
const int LOG = 25, mxN = 2e5+5;
int parent[mxN], depth[mxN], up[LOG][mxN];
// up[x][v] = salto a 2^x ancestros desde el nodo v
void preprocess(int u = 0, int p = 0){
    parent[u] = up[0][u] = p;
    for(int x = 1; x<LOG; ++x){
        up[x][u] = up[x-1][up[x-1][u]];
    }
    for(int v : adj[u]){
        if(v == p) continue;

        depth[v] = depth[u] + 1;
        preprocess(v, u);
    }
}
```

```cpp
int k_ancestro(int u, int k){
    for(int bit = 0; bit<LOG; ++bit){
        if(k & (1<<bit)){
            u = up[bit][u];
        }
    }
    return u;
}
```

LCA

Problema: Dado un árbol, queremos obtener el ancestro común mas bajo de dos nodos (LCA).

El LCA de dos nodos es un nodo tal que es el primer ancestro de ambos de estos nodos, para calcularlo debemos seguir los siguientes pasos:

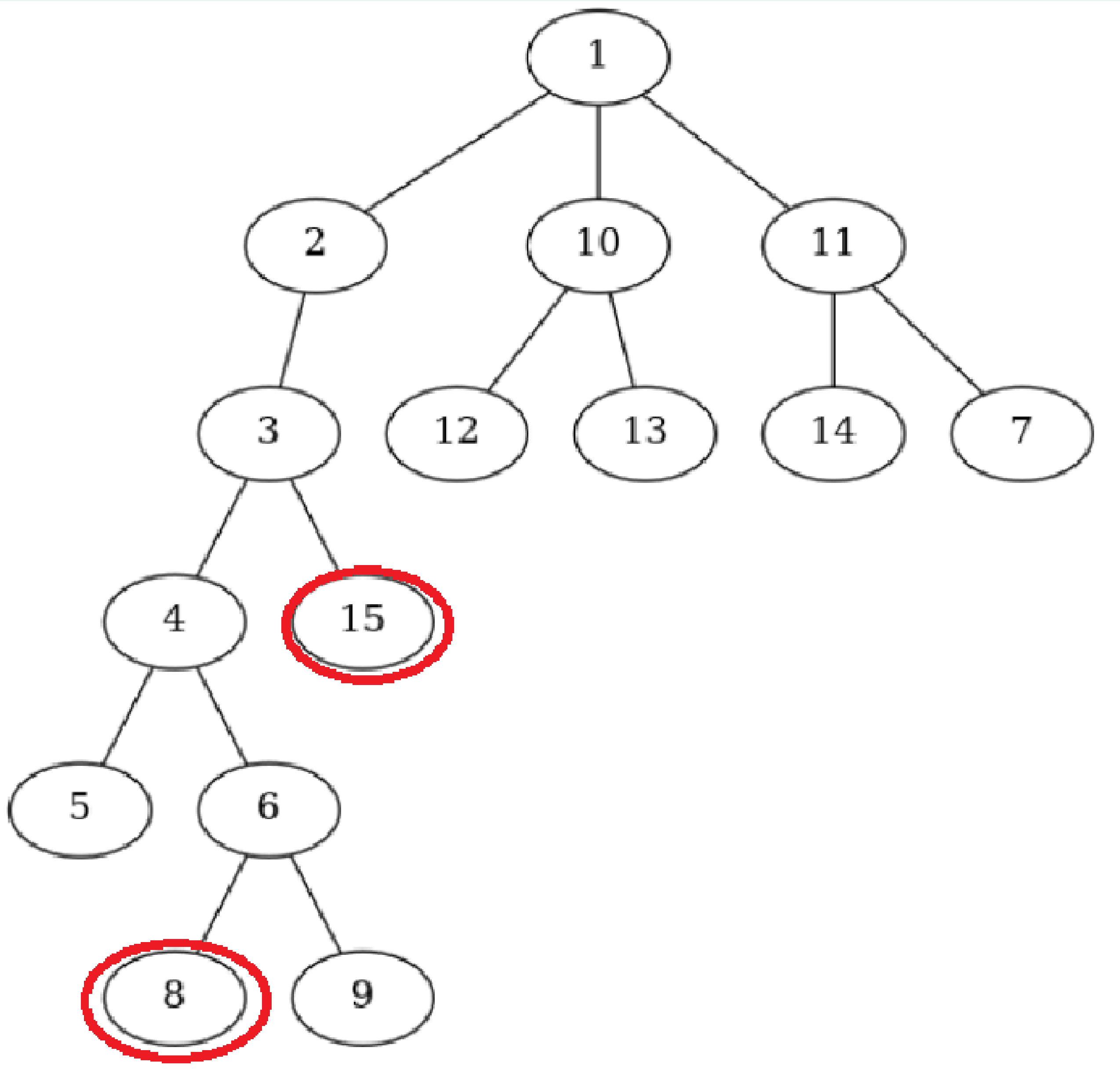
1. igualar ambos nodos en altura (usando binary lifting).

2. si ambos nodos quedan en la misma altura, entonces uno de los nodos era LCA del otro, por lo que retornamos el nodo que estaba más arriba.

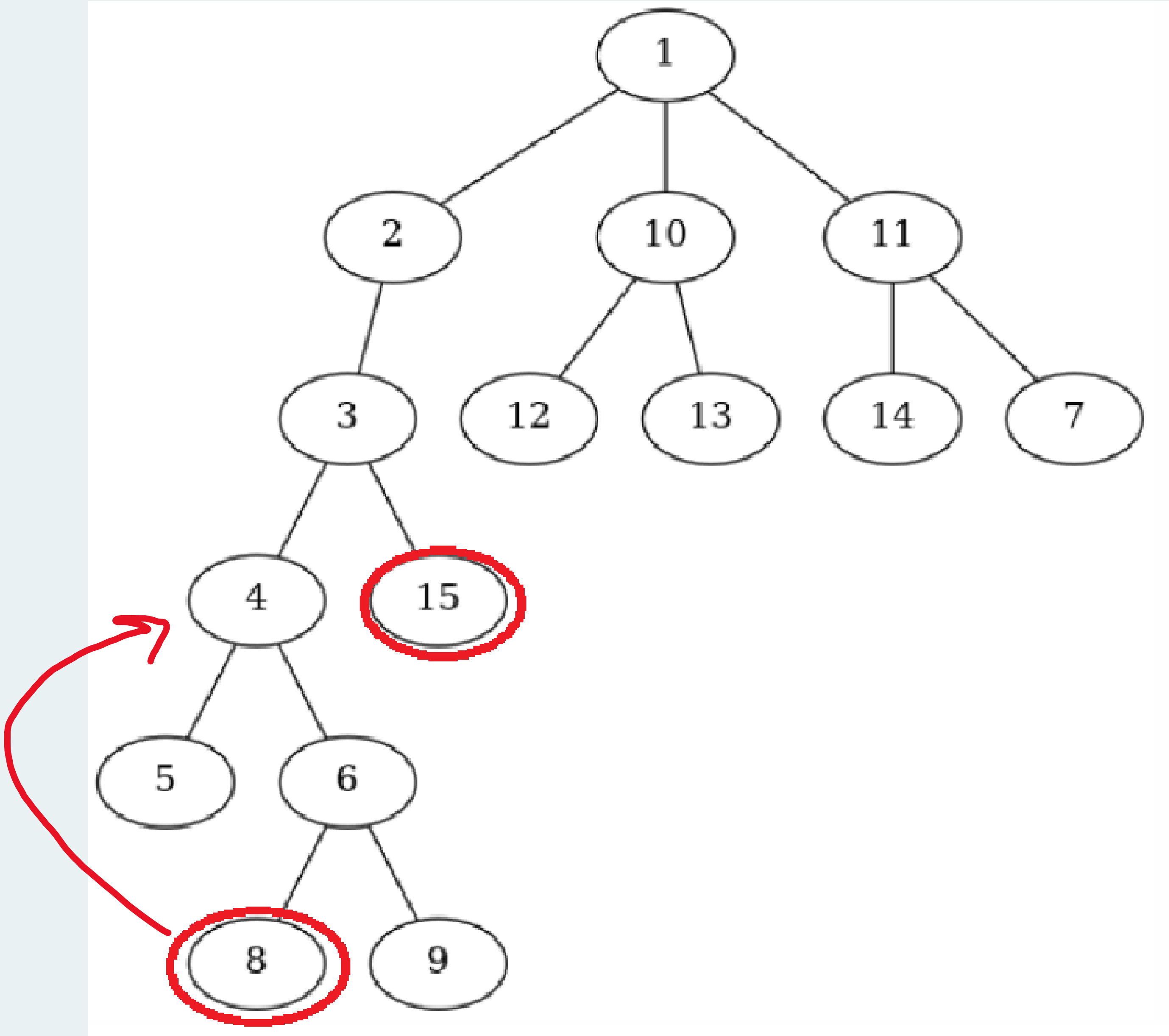
3. en caso contrario, iteramos desde la potencia de 2 más grande a la más pequeña, si el 2^k ancestro de ambos nodos es el mismo, entonces estamos revisando más arriba del LCA, en caso contrario, aún no llegamos al LCA, actualizamos ambos nodos a su 2^k ancestro y repetimos este paso

4. retorna el padre de alguno de los dos nodos

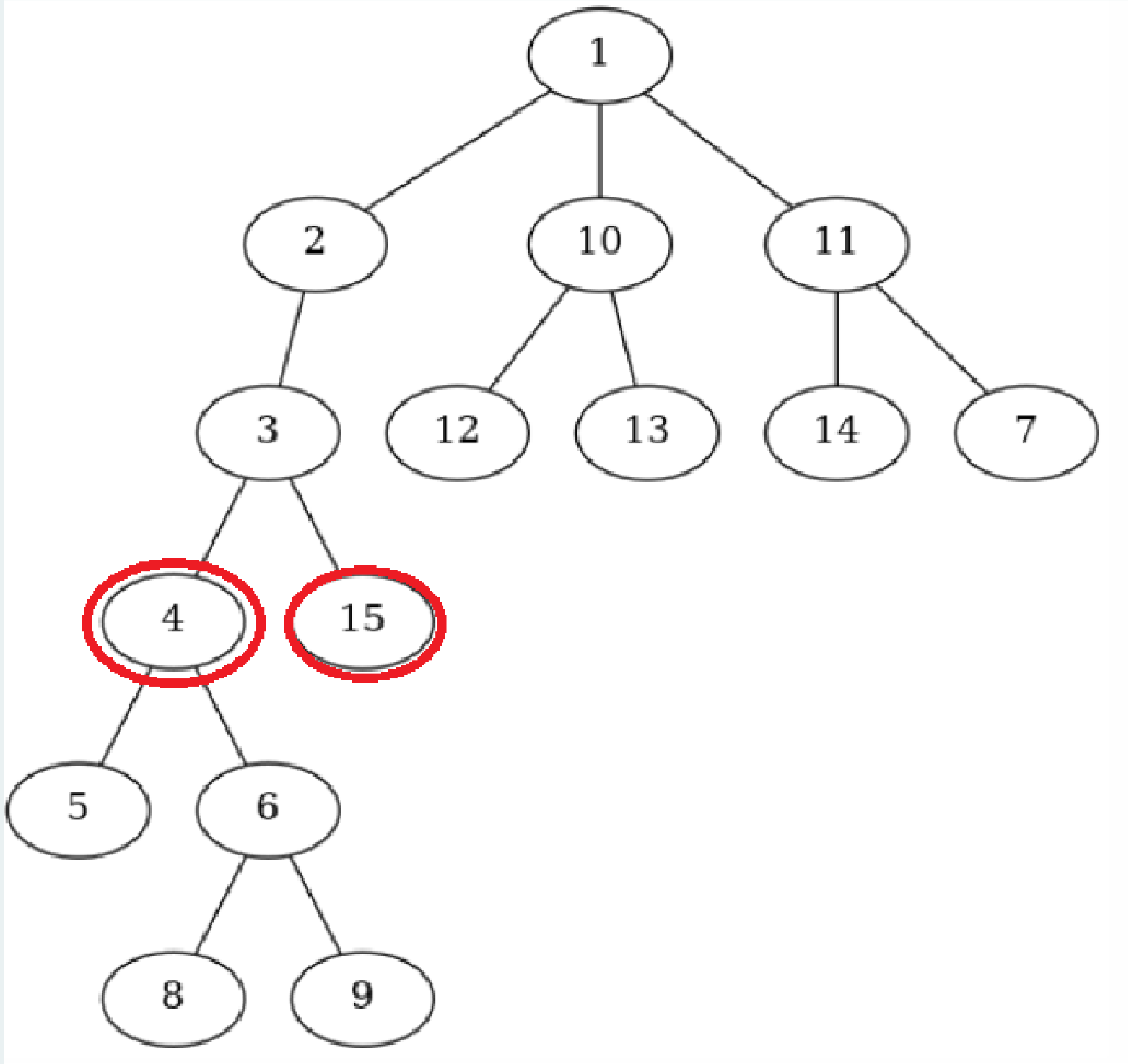
LCA



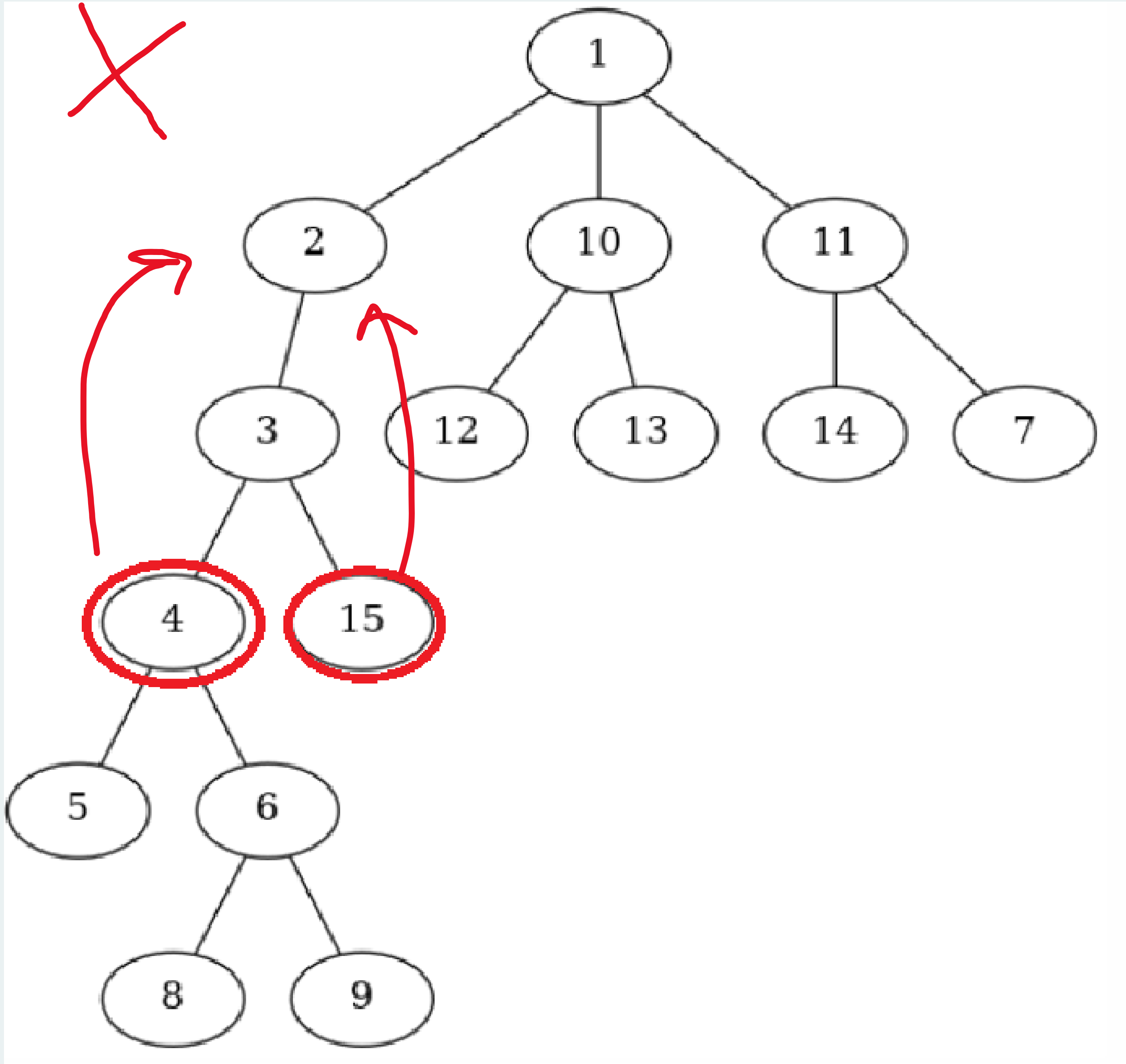
LCA



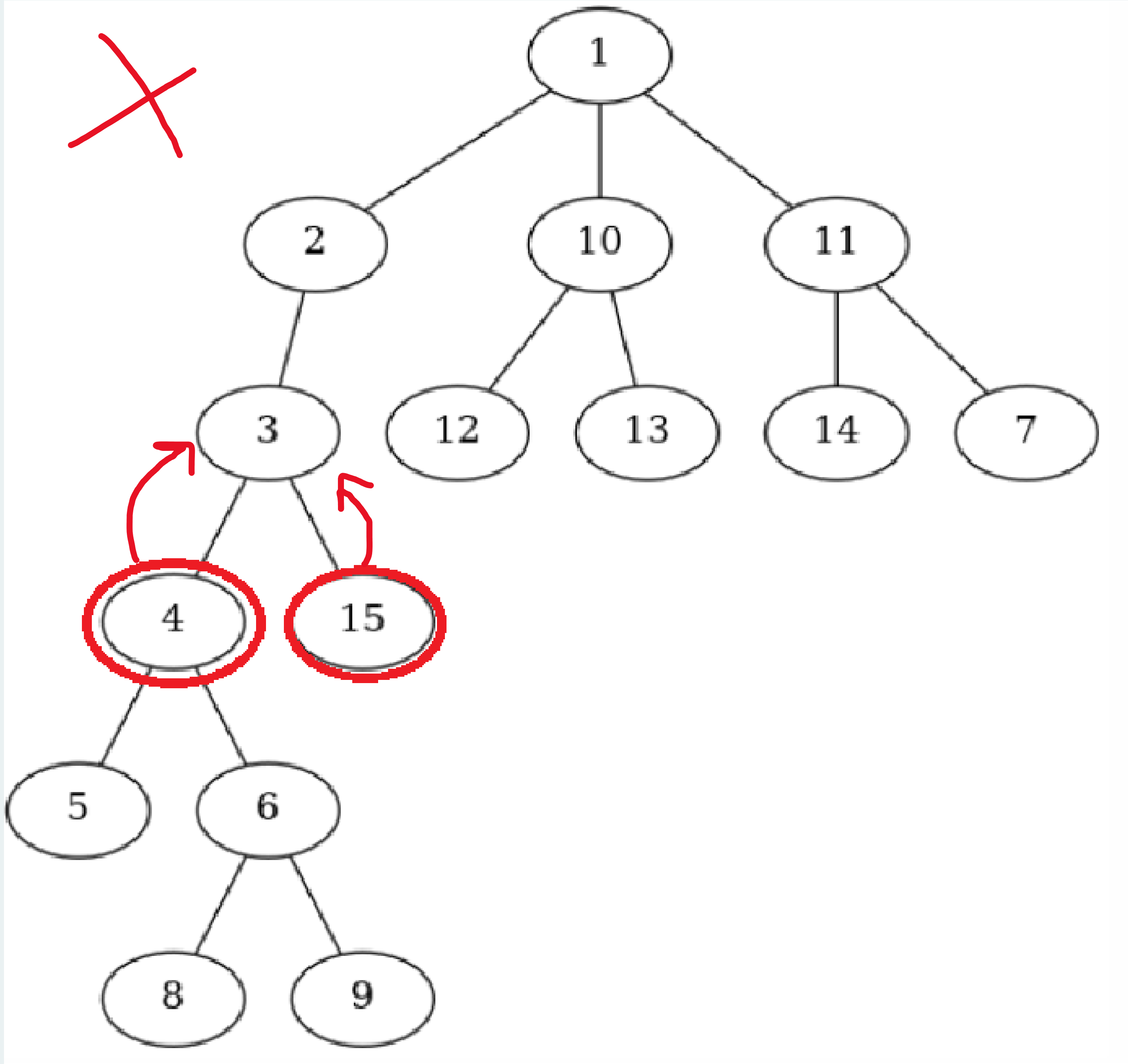
LCA



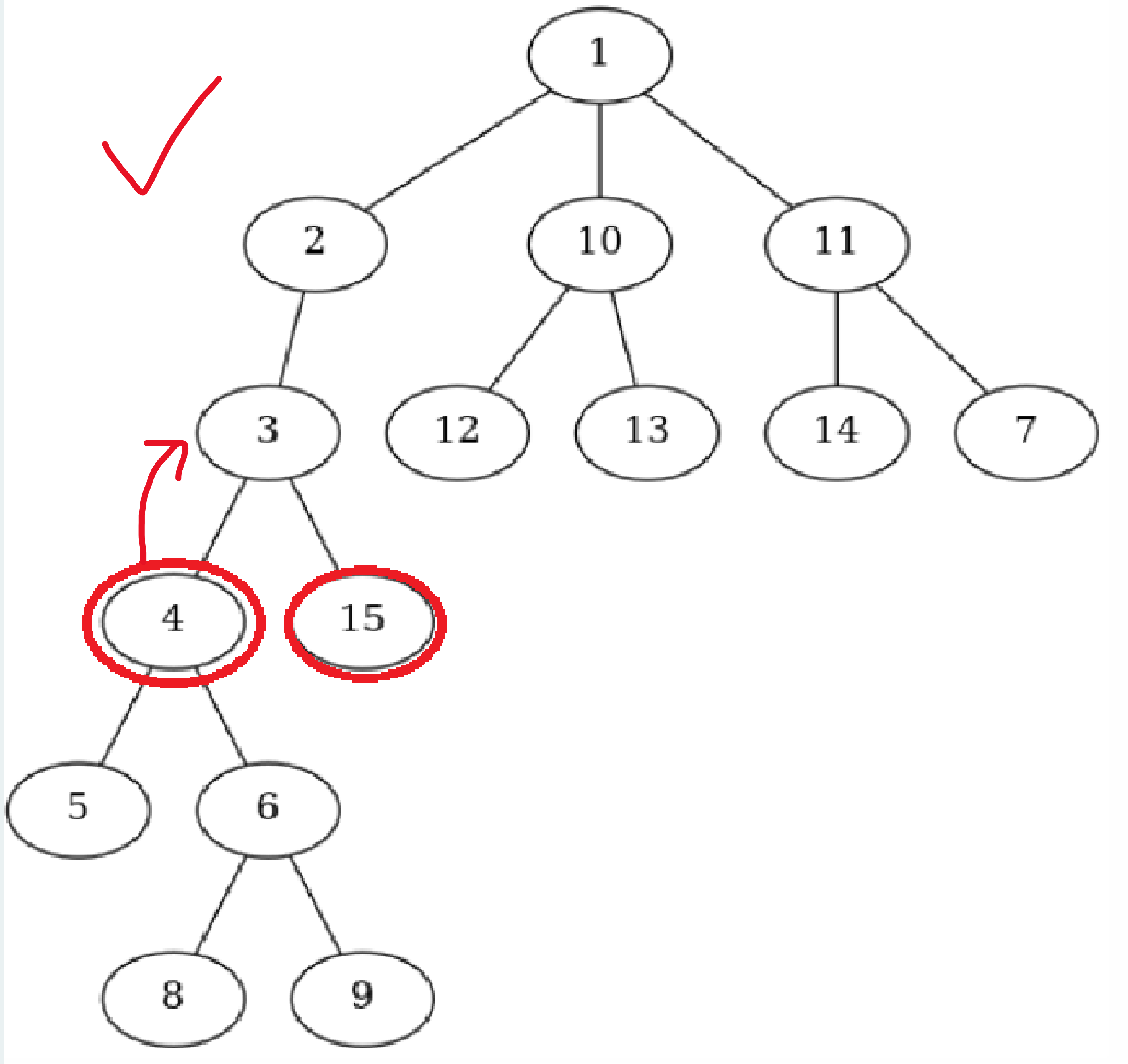
LCA



LCA



LCA



LCA

```
ll lca(int a, int b){
    if(depth[a]<depth[b]) swap(a, b);
    ll diff = depth[a]-depth[b];
    for(ll i = 0; i<20; ++i){
        if(diff & (1<<i)){
            a = up[i][a];
        }
    }
    for(ll i = 19; i>=0; --i){
        if(up[i][a] != up[i][b]){
            a = up[i][a];
            b = up[i][b];
        }
    }
    if(a == b) return a;

    return up[0][a];
}
```

Disjoint Set Union (DSU)

- Cuando en un problema tenemos distintos conjuntos y tenemos la operación de "juntar conjuntos" e identificar a que conjunto pertenecen, es cuando queremos usar DSU.
- Ejemplo común: Minimun Spanning Tree, dado un grafo con aristas con peso, queremos elegir un subconjunto de aristas tal que exista un camino entre cada par de nodos y la suma de las aristas escogidas sea mínima.
- Estrategia Greedy: Cada nodo es un conjunto independiente de los demás, queremos iterar sobre las aristas con menor coste, si la arista actual conecta dos nodos cuyos conjuntos son distintos, entonces combinamos los conjuntos y consideramos esta arista como parte de la respuesta.

Problema: MST for each Edge.

Nos dan un Grafo con N <= 2*10^5 nodos y M <= 2*10^5 aristas, por cada una de las aristas, queremos determinar el valor del Mínimum Spanning Tree usando obligadamente esta arista.

Matematicas

Criba de Erathostenes

En ciertos problemas nos interesa encontrar los numeros primos hasta cierto valor, generalmente se usa como subrutina para calcular algo, pero si queremos almacenar un vector que nos indique cuales son los numeros primero, entonces la manera más eficaz (y más usada) es con criba de erathostenes. O(n log log n).

Criba de Erathostenes

En ciertos problemas nos interesa encontrar los numeros primos hasta cierto valor, generalmente se usa como subrutina para calcular algo, pero si queremos almacenar un vector que nos indique cuales son los numeros primero, entonces la manera más eficaz (y más usada) es con criba de erathostenes. O(n log log n).

Algoritmo: mantenemos un vector que nos indica si cierto numero a sido marcado, iteramos desde el numero 2, si no a sido marcada, entonces es un numero primo, iteramos sobre todos sus multiplos y los marcamos.

Criba de Erathostenes

En ciertos problemas nos interesa encontrar los numeros primos hasta cierto valor, generalmente se usa como subrutina para calcular algo, pero si queremos almacenar un vector que nos indique cuales son los numeros primero, entonces la manera más eficaz (y más usada) es con criba de erathostenes. O(n log log n).

Algoritmo: mantenemos un vector que nos indica si cierto numero a sido marcado, iteramos desde el numero 2, si no a sido marcada, entonces es primero, iteramos sobre todos sus multiplos y los marcamos.

```
int n;
vector<bool> is_prime(n+1, true);
is_prime[0] = is_prime[1] = false;
for (int i = 2; i <= n; i++) {
    if (is_prime[i] && (long long)i * i <= n) {
        for (int j = i * i; j <= n; j += i)
            is_prime[j] = false;
    }
}
```

Problema

Hay una joyeria que posee n joyas, la i-esima joya tiene un precio de i+1 (2, 3, 4, …, n+1). Quieres colorear las joyas de tal forma que ninguna joya sea divisor primo de otra joya del mismo color.

Imprima la minima cantidad de colores para pintar todas las joyas e imprima como colorear las joyas.

n <= 10^5

Ejemplo:

Input : 3

Output : 2

1 1 2

Factorizacion Prima

Generalmente los limites donde queremos la factorizacion prima de un numero es de <= 10^12, ya que podemos encontrar la factorizacion prima en O(sqrt(N)).

Factorizacion Prima

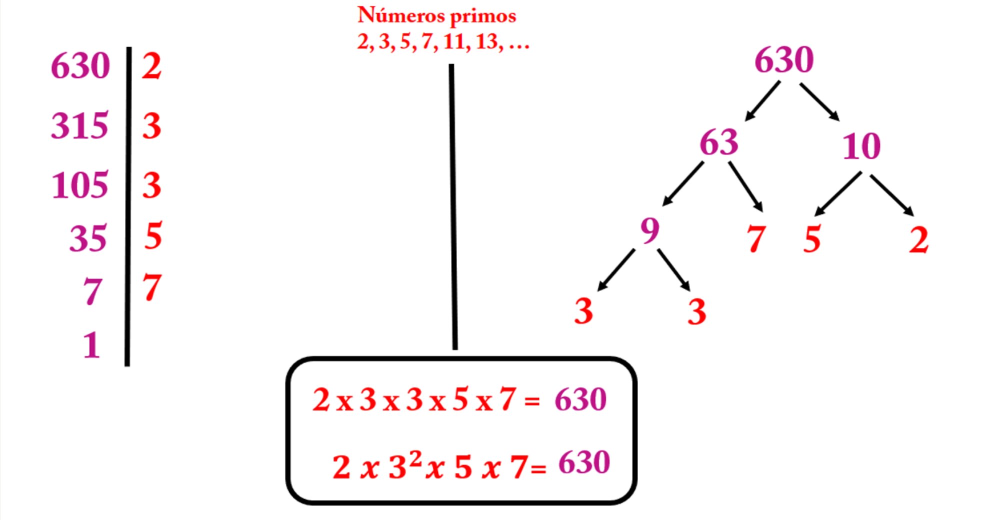
Generalmente los limites donde queremos la factorizacion prima de un numero es de <= 10^12, ya que podemos encontrar la factorizacion prima en O(sqrt(N)).

Una factorizacion prima de un numero es la multiplicación de numeros primos tal que su producto sea el numero en cuestion.

Factorizacion Prima

Generalmente los limites donde queremos la factorizacion prima de un numero es de <= 10^12, ya que podemos encontrar la factorizacion prima en O(sqrt(N)).

Una factorizacion prima de un numero es la multiplicación de numeros primos tal que su producto sea el numero en cuestion.



Factorizacion Prima

```
ll n2 = n;
vector<ll> factorizacion;
for(ll i = 2; i*i<=n; ++i){
    while(n2%i == 0){
        factorizacion.push_back(i);
        n2 /= i;
    }
}
// hay exactamente un primo en la factorizacion
// mayor a raiz de n, lo obtenemos como caso extra
if(n2 > 1){
    factorizacion.push_back(n2);
}
```

Problema

Nos entregan un numero N, Alice y Bob juegan un juego con todos los divisores de N, empezando por Alice y turnandose, cada jugador toma uno de los divisores y lo elimina de la lista, Alice ganará el juego si al final el maximo comun divisor (GCD) de sus numeros elegidos es mayor a 1, Bob ganará el juego si este no es el caso.

Determina si Alice ganará el juego considerando que ambos jugadores juegan optimamente.

Ej: 10 - Yes

Alice gana porque los divisores de 10 son: 1, 2, 5, 10

Probabilidades

Este campo es muy amplio y hay que tener mucha experiencia para ver y modelar este tipo de problemas, asi como tambien tener algunas precauciones y sutilezas al momento de programar.

Probabilidades

Este campo es muy amplio y hay que tener mucha experiencia para ver y modelar este tipo de problemas, asi como tambien tener algunas precauciones y sutilezas al momento de programar.

Si el problema da "wrong answer" y estamos seguros de nuestro codigo, podriamos tener cuidado con la precision, quizas cambiar de double a long double (añadiendo mas precision) puede ser una solución.

Probabilidades

Este campo es muy amplio y hay que tener mucha experiencia para ver y modelar este tipo de problemas, asi como tambien tener algunas precauciones y sutilezas al momento de programar.

Si el problema da "wrong answer" y estamos seguros de nuestro codigo, podriamos tener cuidado con la precision, quizas cambiar de double a long double (añadiendo mas precision) puede ser una solución.

En caso de tener un problema de conteo (ej: cuantas formas de elegir x cosa tenemos), es probable que nos pidan la respuesta en modulo M, con M siendo un numero primo relativamente. En este caso, NUNCA hay que hacer modulo normal cuando tenemos un numero dividiendo a otro.

Probabilidades

Solucion a esto ultimo: expresar el calculo que queramos hacer como una multiplicacion del inverso del divisor. Esto funciona ya que la multiplicacion es compatible con el modulo, no asi con la division.

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!} = n!\,(k!)^{-1}\,((n-k)!)^{-1}$$

El inverso multiplicativo de X (modulo M) se calcula como X^(M-2), como M suele ser del orden 10^9, necesitamos utilizar exponenciación binaria para calcular este valor.

https://cp-algorithms.com/algebra/binary-exp.html#effective-computation-of-large-exponents-modulo-a-number