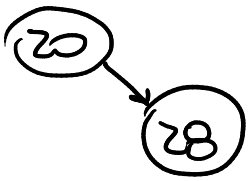
1) Dado a de tipo ABB, Indicar cómo queda en cada estado dando una breve explicación, sabiendo que comienza vacío y se hacen las siguientes operaciones:

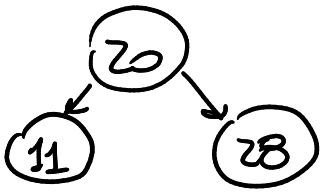
a. a.insertar(20)
b. a.insertar(28)
c. a.insertar(11)
d. a.insertar(18)
e. a.insertar(3)
f. a.insertar(22)
g. a.insertar(15)
h. a.eliminar(28)
i. a.insertar(16)
j. a.insertar(30)
k. a.eliminar(11)

a) ¿ABB Vacio? Si => Raiz = 20

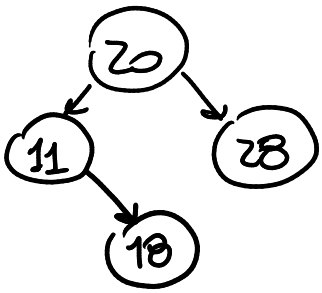
b) ¿ABB Vacio? No => ¿20 < 28? Si => Derecha



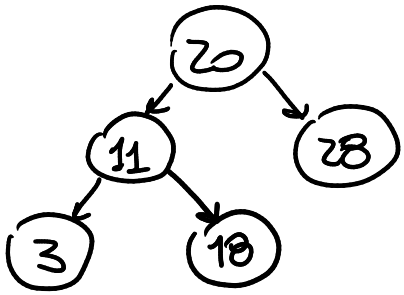
c) ¿ABB Vacio? No => ¿20 < 11? No => Izquierda



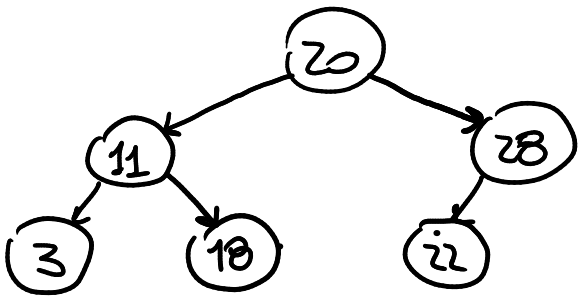
d) ¿ABB Vacio? No => ¿20 < 18? No => Izquierda
¿11 < 18? Si => Derecha



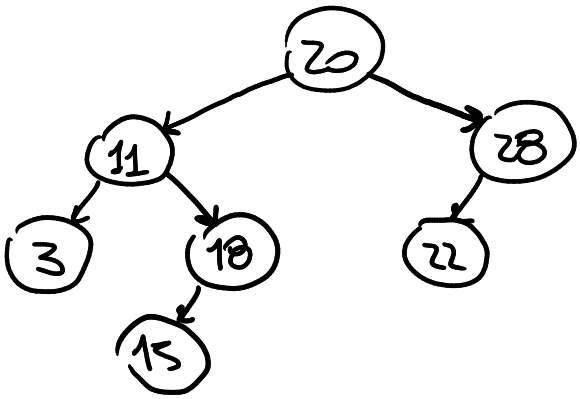
e) ¿ABB vacío? No => ¿20 < 3? No => Izquierda
¿11 < 3? No => Izquierda



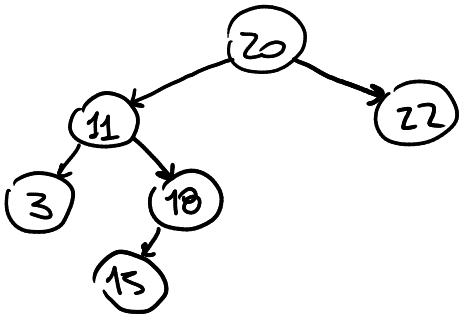
f) ¿ABB vacío? No => ¿20 < 22? Sí => Derecha
¿28 < 22? No => Izquierda



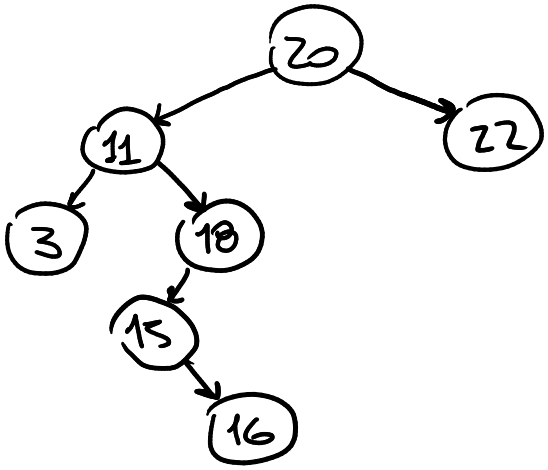
g) ¿ABB vacío? No => ¿20 < 15? No => Izquierda
¿11 < 15? Sí => Derecha
¿18 < 15? No => Izquierda



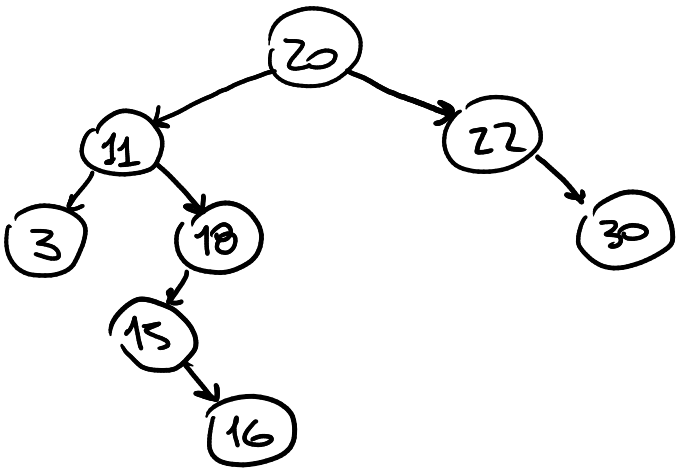
h) ¿ABB vacío? No => Busco el 28
¿28 tiene hijos a derecha? No =>



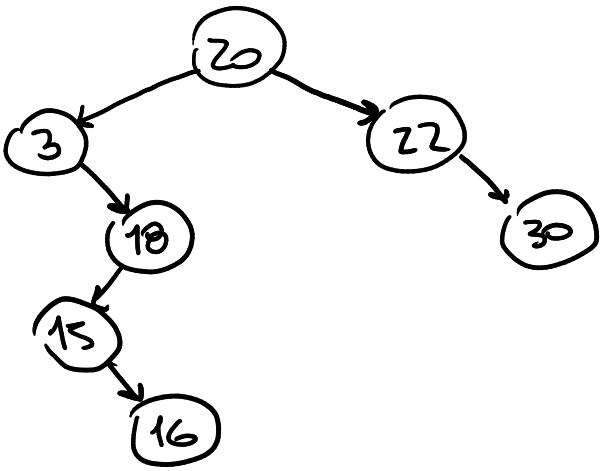
i) ¿ABB vacío? No => ¿20 < 16? No => Izquierda
¿11 < 16? Si => Derecha
¿18 < 16? No => Izquierda
¿15 < 16? Si => Derecha



j) ¿ABB vacío? No => ¿20 < 30? Si => Derecha
¿22 < 30? Si => Derecha



K) ¿ABB vacío? No => Busco el 11
¿11 tiene hijos a izquierda? Si =>



2) Árbol B.

a. Definir árbol B de orden m.

b. Sea un árbol B de orden m = 5. Indicar cómo queda en cada paso al insertarse las siguientes claves: 8, 14, 2, 15, 3, 1, 16, 6, 5, 27, 37, 18, 25, 7, 13, 20, 22, 23, 24. El árbol comienza vacío.

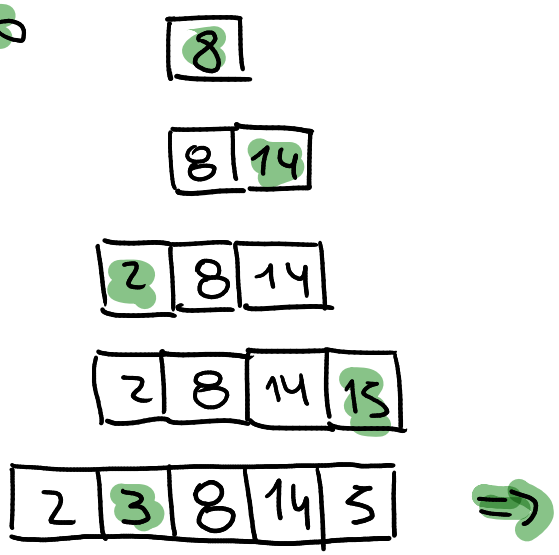
a) Un árbol B de órden m cumple las siguientes condiciones:

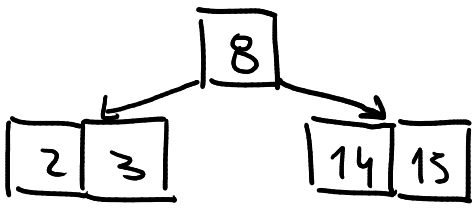
1) Todos los nodos (excepto la raiz) tienen por lo menos (M-1)/2 claves

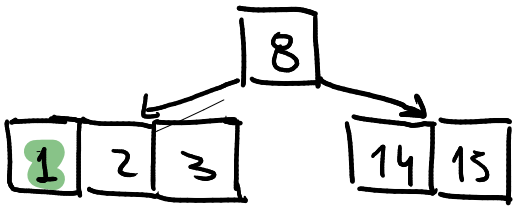
2) La raiz tiene entre 2 y M hijos o es hoja

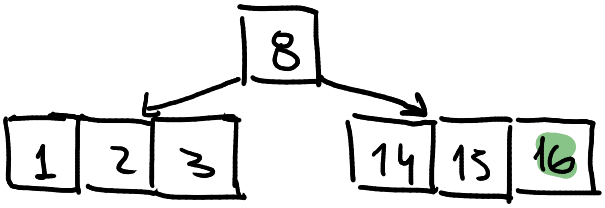
3) Todas las hojas están en el mismo nivel

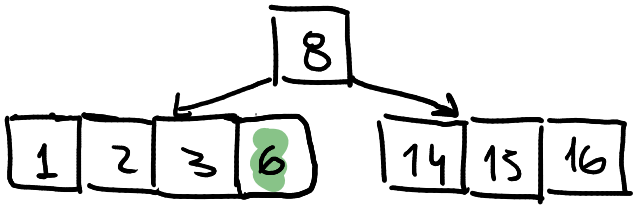
b) Nuevo

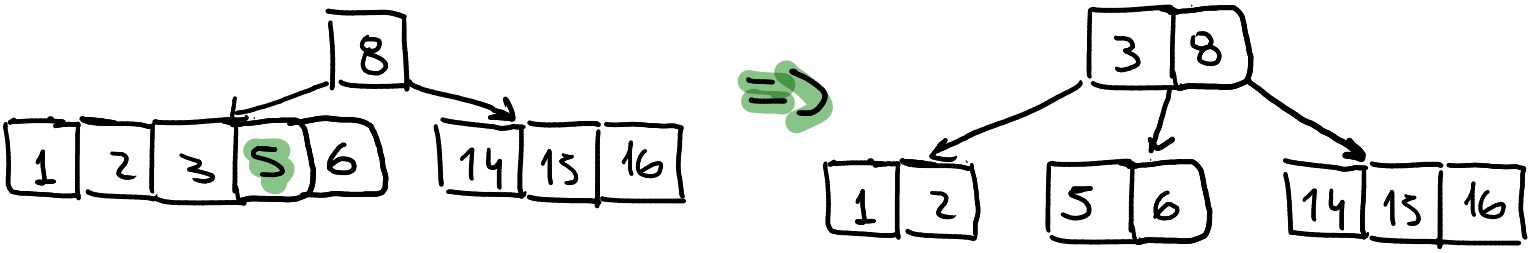


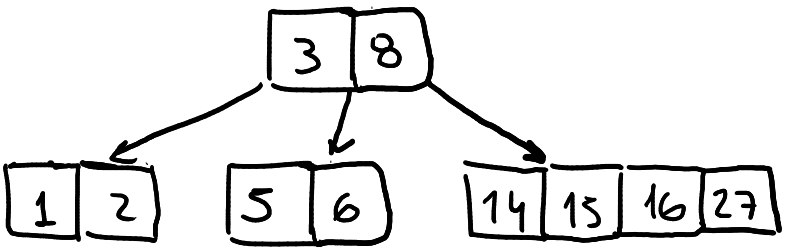


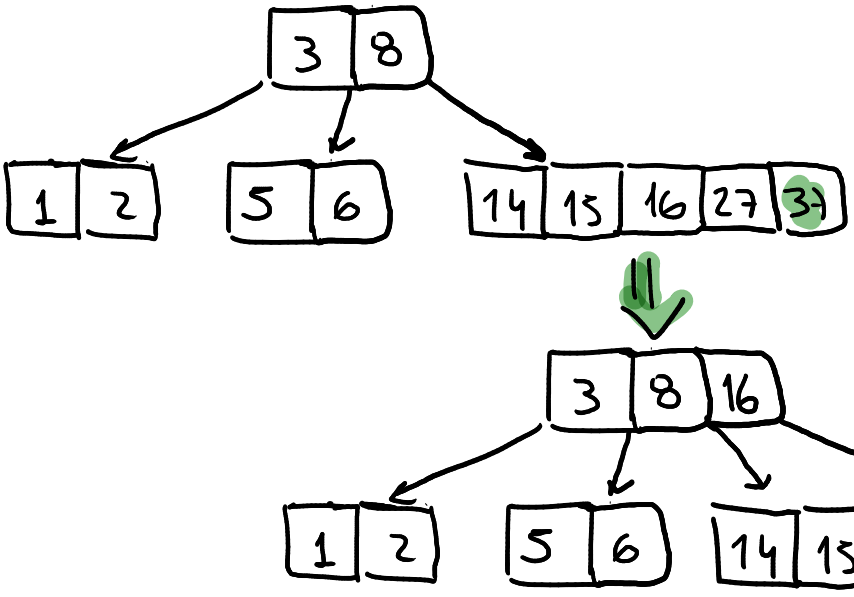


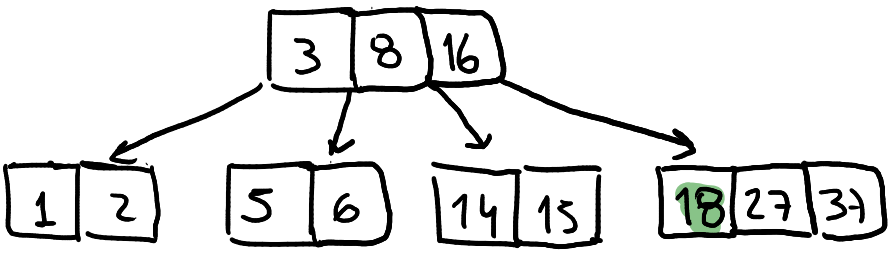


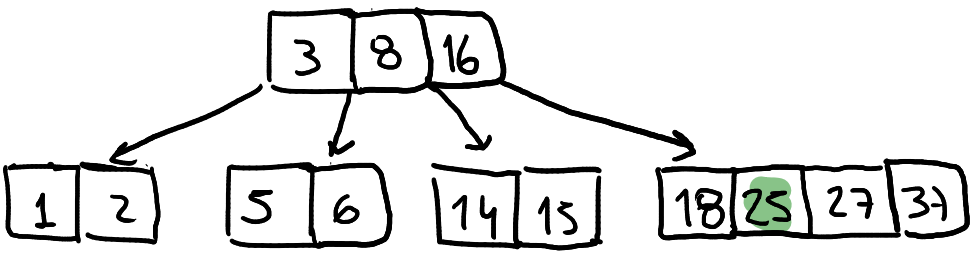

8
1 2 3 6
14 15 16

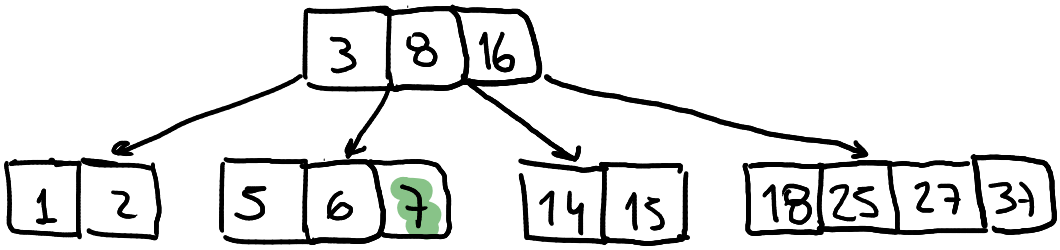

8
1 2 3 5 6
14 15 16
3 8
1 2
5 6
14 15 16

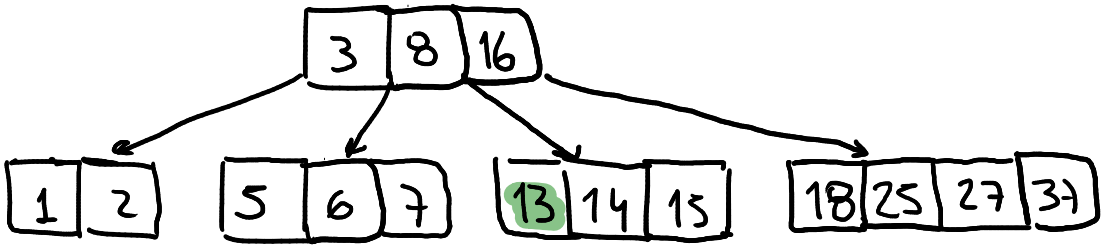

3 8
1 2
5 6
14 15 16 27

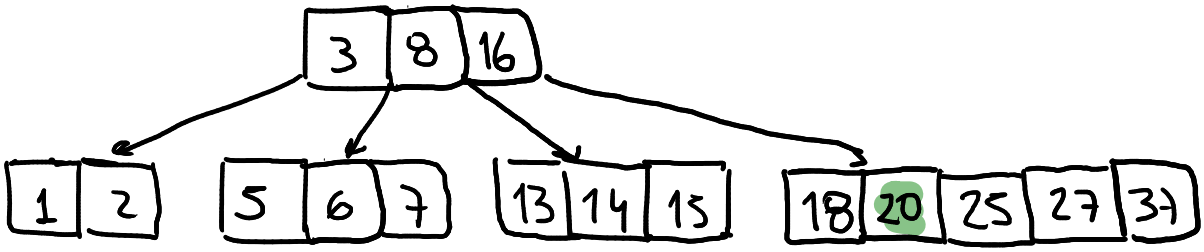

3 8
1 2
5 6
14 15 16 27 37
3 8 16
1 2
5 6
14 15
27 37

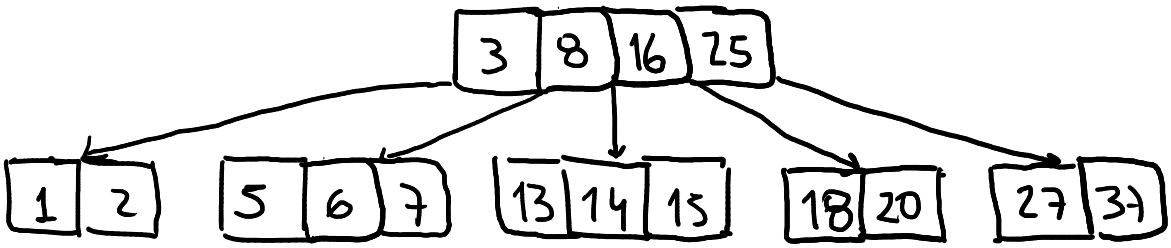

3 8 16
1 2
5 6
14 15
18 27 37

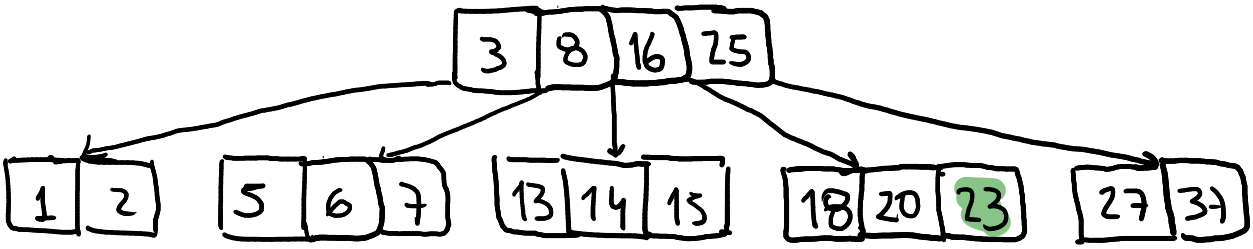

3 8 16
1 2
5 6
14 15
18 25 27 37

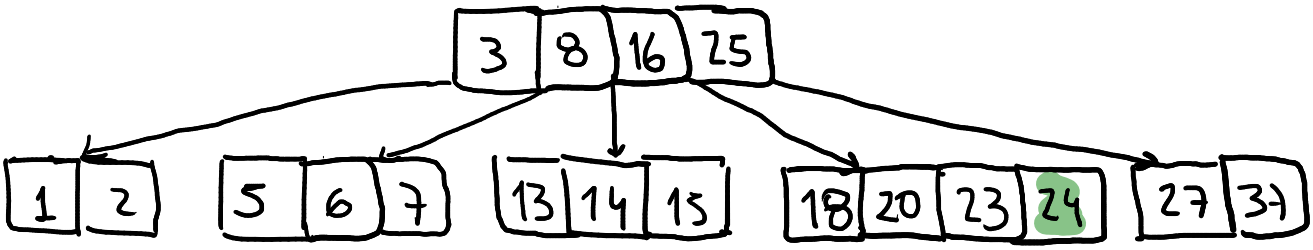

3 8 16
1 2
5 6 7
14 15
18 25 27 37


3 8 16
1 2
5 6 7
13 14 15
18 25 27 37

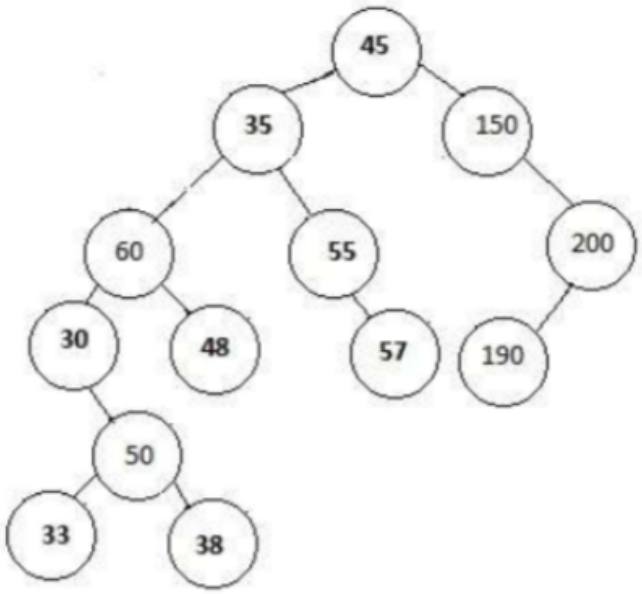

3 8 16
1 2
5 6 7
13 14 15
18 20 25 27 37


3 8 16 25
1 2
5 6 7
13 14 15
18 20
27 37


3 8 16 25
1 2
5 6 7
13 14 15
18 20 23
27 37

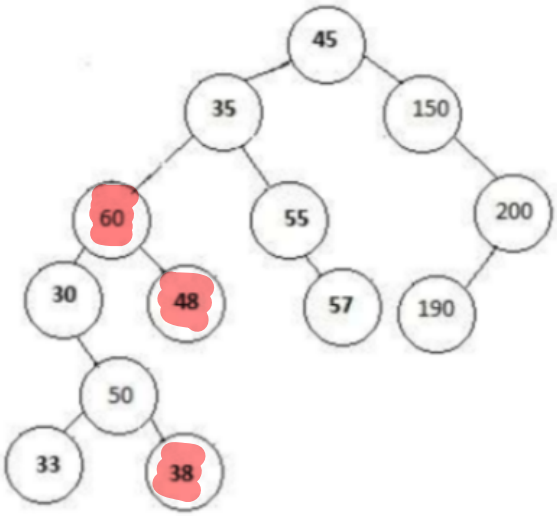

3 8 16 25
1 2
5 6 7
13 14 15
18 20 23 24
27 37

3) Dado el árbol:



a. Indicar si es un ABB. Justificar.
b. Indicar cómo queda la salida con un recorrido en pre orden.

a) No es un ABB porque no se cumple que los elementos del subárbol izq. sean menor que la raiz y los del subárbol der. mayores



60 > 35 => Deberia estar a la der. de 35
48 < 60 => Deberia estar a la izq. de 60
38 < 50 => Deberia estar a la izq. de 50

b) 45 - 35 - 60 - 30 - 50 - 33 - 38
48 - 55 - 57 - 150 - 200 - 190

4) Sea el array de bits

0	1	2	3	4	5	6	7
1	0	0	1	1	0	1	1

a. Indicar qué elementos se encuentran.
b. Qué pasos y operaciones se deben hacer para
   i. Dar de alta al valor 5.
   ii. Dar de baja al valor 3.
   iii. Consultar si está el valor 4.

a) El 0, 3, 4, 6, 7

b) i) Crear la máscara

0	0	0	0	0	1	0	0

Hacer un or con el array original

ii) Crear la máscara

1	1	1	0	1	1	1	1

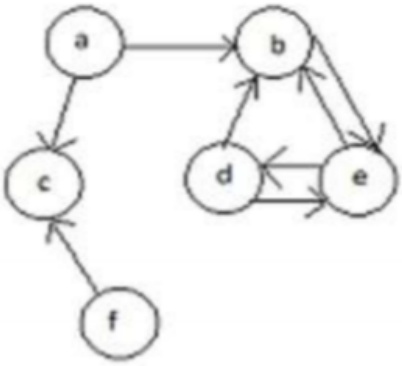
Hacer un and con el array original

iii) Crear la máscara

0	0	0	0	1	0	0	0

Hacer un and con el array original
Si el resultado es 0 no está y si es 1 sí

5) Dado el grafo dirigido:

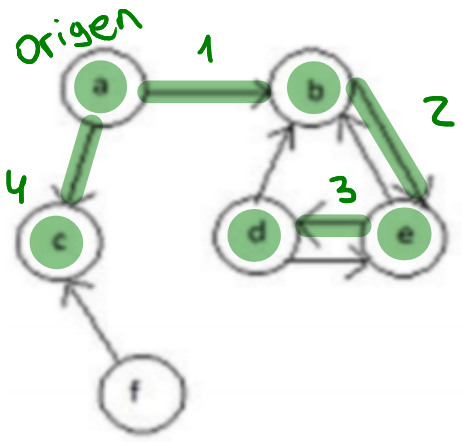


a. Indicar cómo es la matriz de adyacencia.
b. Indicar cómo queda el recorrido en profundidad primero. Imprimir los vértices y explicar el recorrido habiendo elegido la matriz de adyacencia o la lista de adyacencia.

a)

	A	B	C	D	E	F
A	0	1	1	0	0	0
B	0	0	0	0	1	0
C	0	0	0	0	0	0
D	0	1	0	0	1	0
E	0	1	0	1	0	0
F	0	0	1	0	0	0

b)



El DFS con origen en A es:
A - B - E - D - C

Utilizando la matriz de ady. se recorre la Fila A hasta encontrar un 1
El primer 1 esta en (A, B) asi que imprimimos B, lo marcamos como procesado y vamos a la Fila B

Recorremos B hasta encontrar un 1, está en (B, E) así que imprimimos E lo marcamos como procesado y vamos a la Fila E
Recorremos E hasta encontrar un 1, está en (E, B) como B ya fue procesado buscamos el siguiente 1. Está en (E, D) así que imprimimos D, lo marcamos como procesado y vamos a la Fila D
Se repite el mismo procedimiento y en este caso los 1 están en (D, B) y (D, E) como ambos ya fueron procesados volvemos a la Fila anterior (E) a ver si quedó algo sin procesar. No quedó nada => Volvemos a la Fila anterior (B) a ver si quedó algo sin procesar. No quedó nada => Volvemos a la Fila anterior (A) a ver si quedó algo sin procesar. Quedó C => Lo imprimimos, lo marcamos como procesado y nos fijamos si quedó algo más por procesar: No => Listo

6) Sea una estructura de tipo heap de mínima
   a. Indicar cuánto es el costo de encontrar el mínimo elemento.
   b. Indicar cuánto es el costo de insertar un elemento.

a) O(1) porque está en la raiz, el 1er elemento del vector

b) $O(\log_2 N)$

7) Sea el vector v = [150, 31, 83, 325, 46, 186, 222]

a. Explicar cómo funciona el algoritmo de ordenamiento heap.
b. Ordenar el vector, indicando cómo queda en cada paso, aplicando dicho algoritmo.

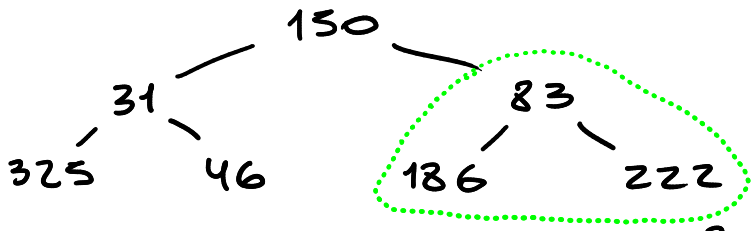
a) 1 - Armar el heap de máxima

2 - Extraer la raiz en cada iteración y restaurar el heap

b) v = [150, 31, 83, 325, 46, 186, 222]
0 1 2 3 4 5 6

n = 7

1 - Armo el heap

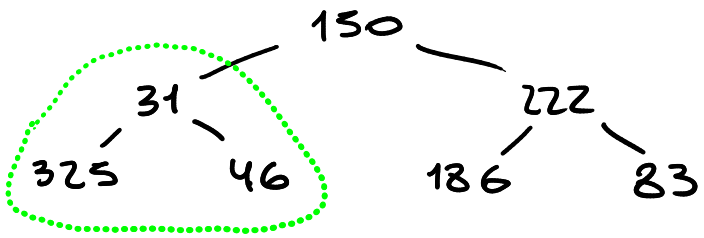


¿186 < 222?

Si => ¿83 < 222?

Si => los intercambio

=> v = [150, 31, 222, 325, 46, 186, 83]

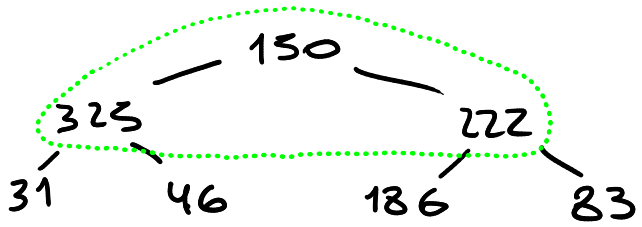


¿325 < 46?

No => ¿31 < 325?

Si => los intercambio

=> v = [150, 325, 222, 31, 46, 186, 83]

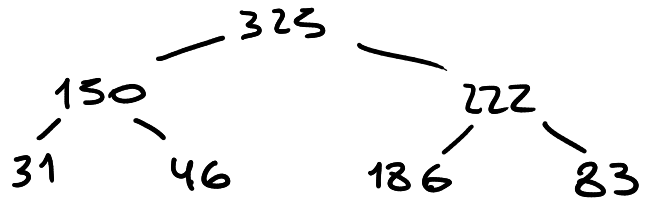


¿325 < 222 ?

No => ¿ 150 < 325?

Si => los intercambio

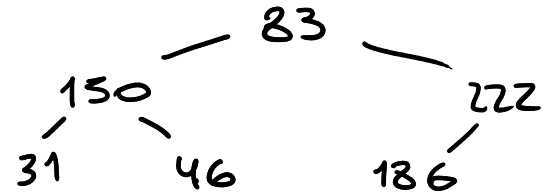
=> V = [325, 150, 222, 31, 46, 186, 83]



=> El heap quedó armado

2 - Ordeno el heap

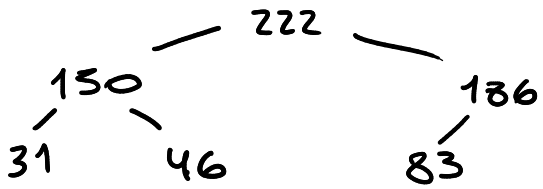
- Extraigo la raíz



V = [83, 150, 222, 31, 46, 186, 325]
0 1 2 3 4 5 6

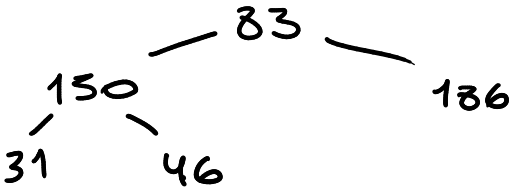
- Restauro el heap desde 0 hasta 5

Este proceso es igual al anterior, no voy a detallar cada paso pero finalmente queda



V = [222, 150, 186, 31, 46, 83, 325]
0 1 2 3 4 5 6

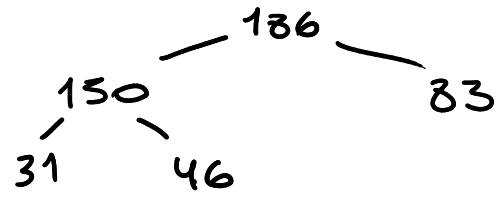
- Extraigo la raíz



V = [83, 150, 186, 31, 46, 222, 325]
0 1 2 3 4 5 6

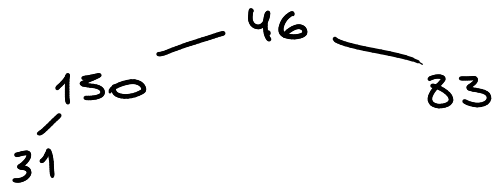
- Restauro el heap desde 0 hasta 4

Este proceso es igual al anterior, no voy a detallar cada paso pero finalmente queda



V = [186, 150, 83, 31, 46, 222, 325]
0 1 2 3 4 5 6

- Extraigo la raiz



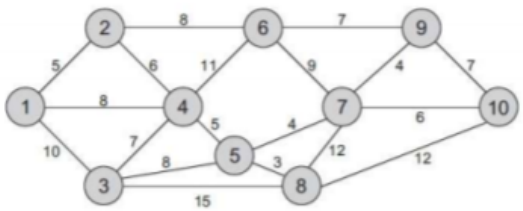
V = [46, 150, 83, 31, 186, 222, 325]
0 1 2 3 4 5 6

...

Sigo restaurando y extrayendo la raiz y Finalmente queda

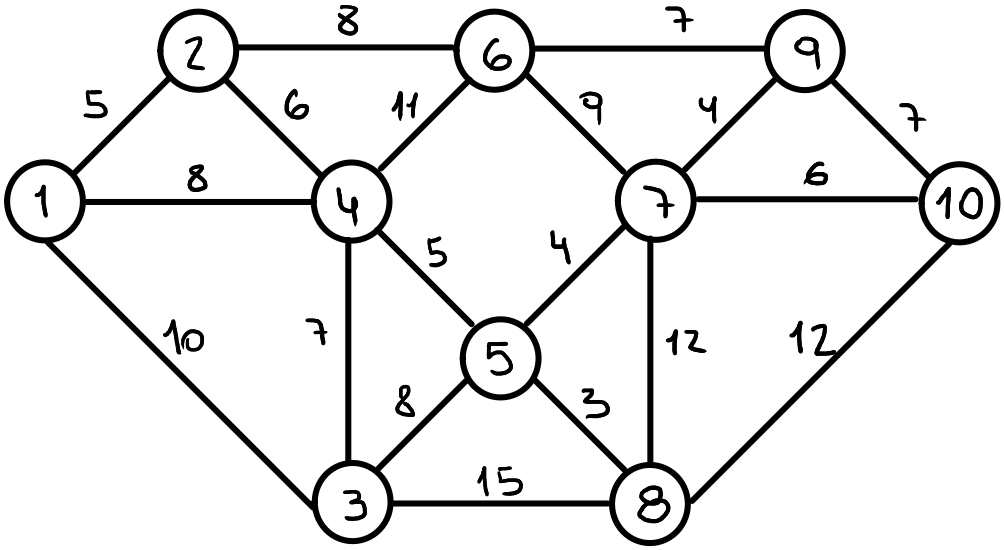
V = [31, 46, 83, 150, 186, 222, 325]
0 1 2 3 4 5 6

8) Explicar la diferencia entre el algoritmo de Prim y el de Kruskal para encontrar un árbol de expansión mínimo.
Aplicar ambos al siguiente grafo indicando cómo se construye paso por paso.



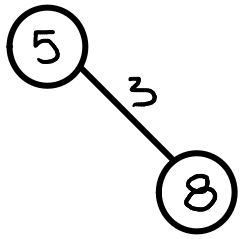
Kruskal construye el arbol de a una arista por vez teniendo en cuenta que siempre se elige la de menor peso que No crea un ciclo

Prim por otra parte construye el árbol agregando de a un vértice por vez teniendo en cuenta que siempre se elige el más cercano a un vértice que ya esté en el grafo

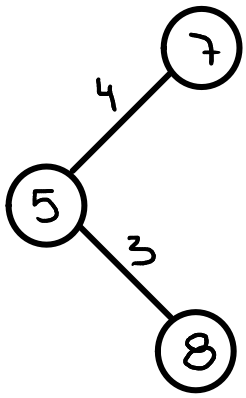


Kruskal:

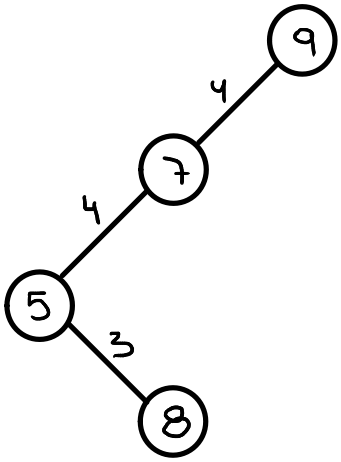
1 - Elijo la arista con menor costo (hay 1 sola)



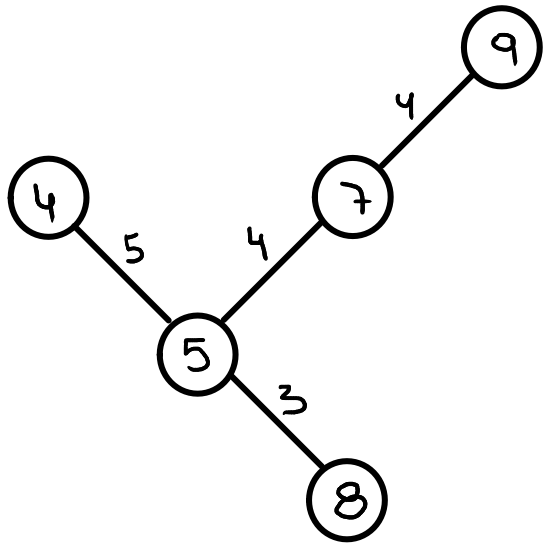
2 - Elijo la siguiente arista con menor costo que no cree un ciclo
(hay 2 que valen 4, elijo una random)



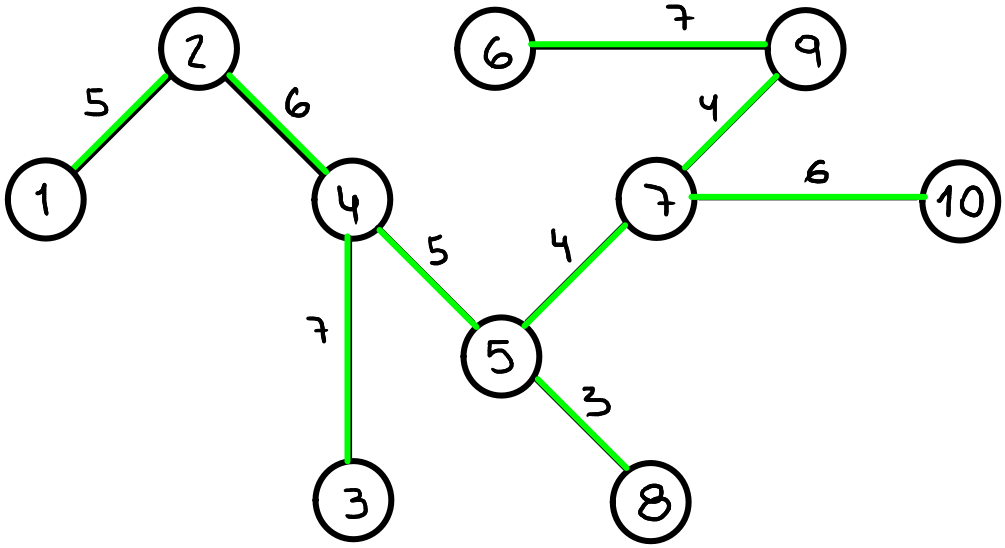
3 - Elijo la siguiente arista con menor costo que no cree un ciclo

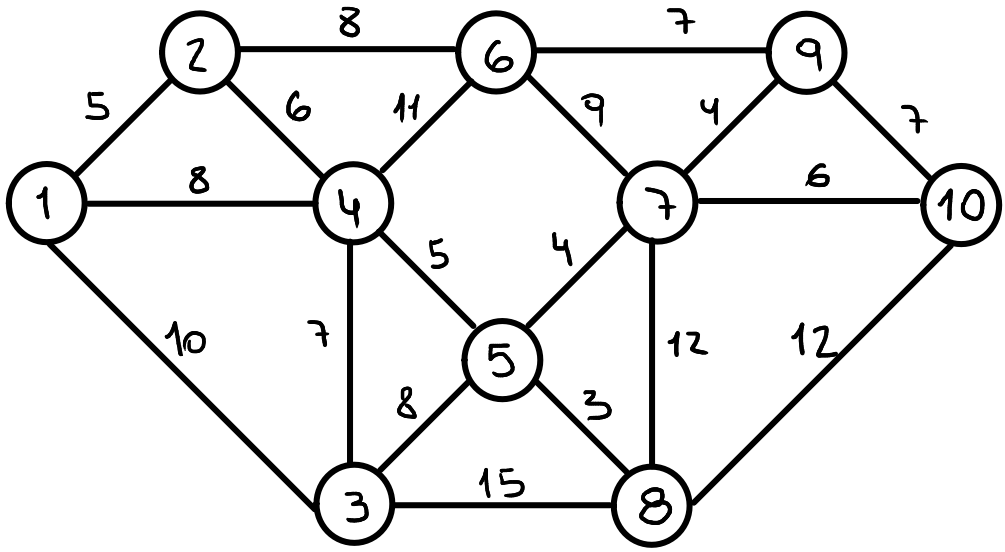


4 - Elijo la siguiente arista con menor costo que no cree un ciclo
(hay 2 que valen 5, elijo una random)



5 - Sigo repitiendo lo mismo hasta obtener

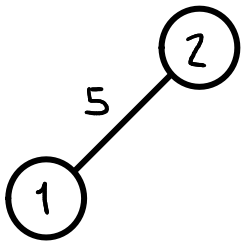




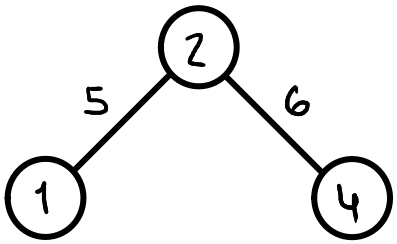
Prim:

1 - Elijo un vértice random → (1)

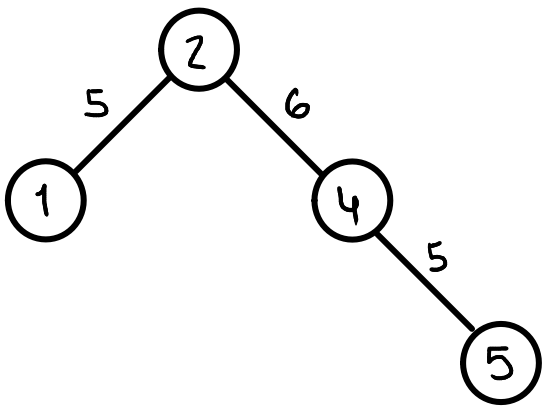
2 - Elijo la arista de menor costo que conecte al vértice anterior con uno nuevo



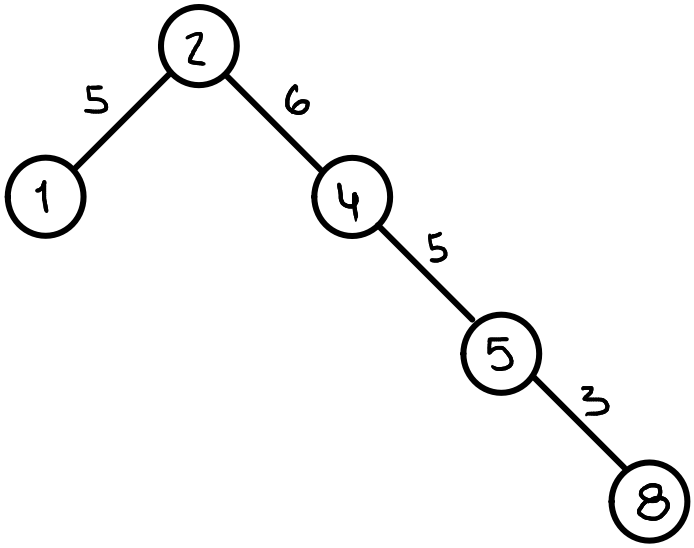
3 - Elijo la arista de menor costo que conecte a alguno de los vértices anteriores con uno nuevo



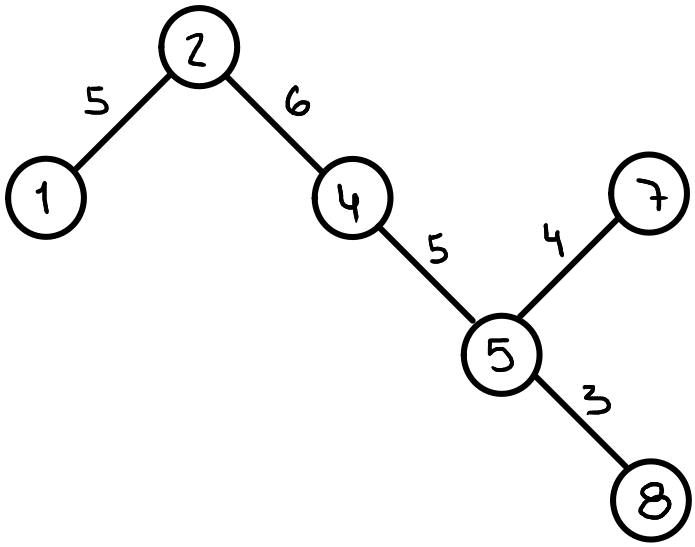
4 - Elijo la arista de menor costo que conecte a alguno de los vértices anteriores con uno nuevo



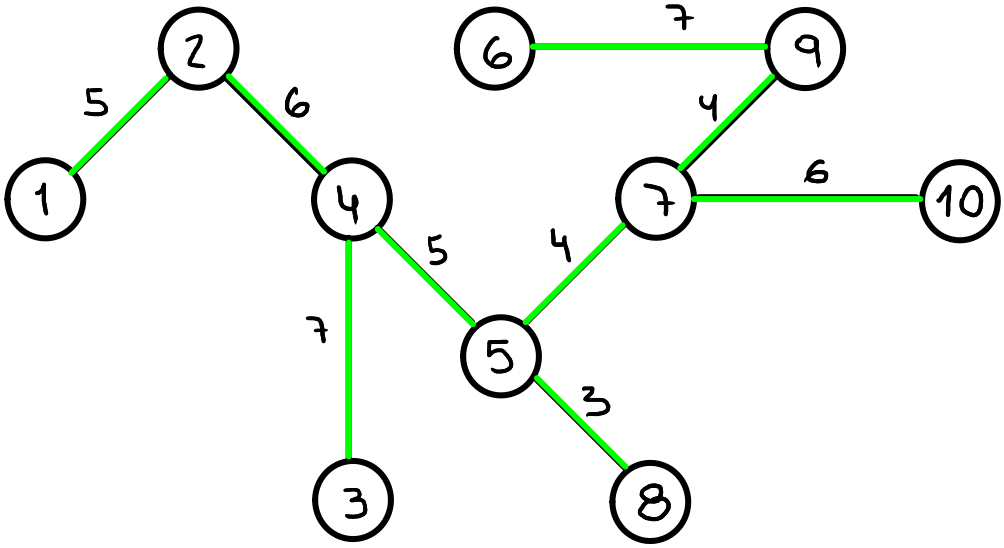
5 - Elijo la arista de menor costo que conecte a alguno de los vértices anteriores con uno nuevo



6 - Elijo la arista de menor costo que conecte a alguno de los vértices anteriores con uno nuevo



7 - Sigo repitiendo lo mismo hasta obtener



9) Nombre y explique brevemente las funciones de hashing que conoce.

9)
- División → Se divide la clave por el tamaño de la tabla y se toma el módulo
- Folding → Se multiplica la clave por un valor entre 0 y 1, se toma la parte Fraccionaria y se la multiplica por el tamaño de la tabla
- Mid - Square → Se eleva la clave al cuadrado y se toman los digitos centrales
- Extraction → Se extrae una parte de la clave
- Radix Transformation → Se cambia la base de la clave

10) ¿Qué es una colisión? ¿Qué formas de resolverla conoce? Nombrar y explicar.

Una colisión se produce cuando dos claves devuelven un mismo valor luego de aplicar una función de hash.

Open Addresing: Las claves están en una sola tabla

Linear probing → Si p está ocupado se prueba con $p+1, p+2, \cdots, p+n$ hasta encontrar una posición vacia

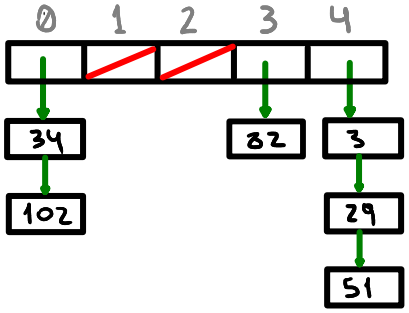
El problema es que quedan muchas claves agrupadas

Quadratic probing → Si p está ocupado se prueba con $p+1^2, p+2^2, \cdots, p+n^2$ hasta encontrar una posición vacia

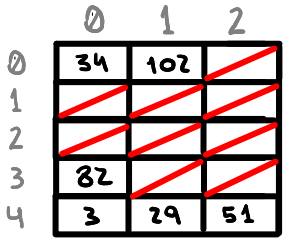
Double hashing → Se utilizan dos funciones de hashing $p = h_1(K) + i \cdot h_2(K)$ con $i = 1, 2, \cdots, n$

Chaining: Las claves están en distintas tablas.

Lo que hay en cada pos. de la tabla es una lista enlazada con las claves

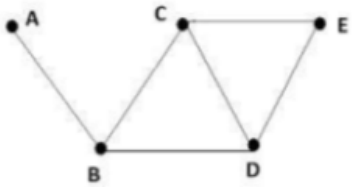


Bucket Addresing: Las claves se guardan en una misma tabla pero cada posición tiene muchas posiciones



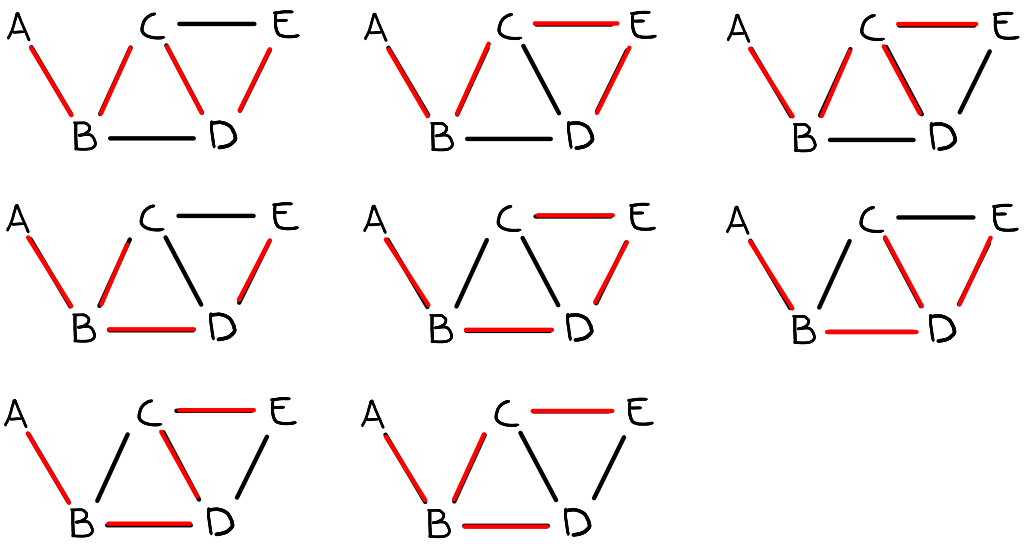
11) Definir árbol de expansión mínimo de un grafo.

Dado el siguiente grafo



Encontrar todos los árboles de expansión mínimo (las aristas tienen peso 1).

Un árbol de expansión minimo es aquel que dado un grafo conexo, tiene todos los vértices conectado por el menor peso



12) Programar en C++ una función recursiva que calcule la suma de los nodos internos (no hojas) de un árbol binario de enteros.

```
void ABB:: sumaNodosNoHojas() {
    std::cout << "\n\tLa suma de todos los elementos del arbol que no son hojas es: ";
    std::cout << sumaNodosNoHojas(raiz, 0);
}

int ABB:: sumaNodosNoHojas(Nodo* nodoActual, int sumaActual) {
    int total = sumaActual;
    if (nodoActual && !nodoActual->esHoja()) {
            total = sumaNodosNoHojas(nodoActual->obtenerIzquierda(), total);
            total = sumaNodosNoHojas(nodoActual->obtenerDerecha(), total);
            total += nodoActual->getKey();
    }
    return total;
}
```