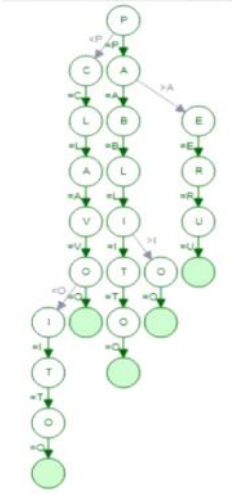
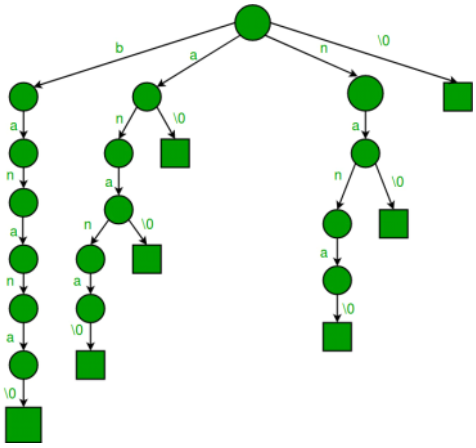
Tries

Monday, January 3, 2022 1:34 PM



TST

La diferencia es q no me pone los 27 hijos por letra del abecedario. Consume menos memoria pero si es mucha informacion se vuelve medio edioso y grande ya que depende del orden en el que se ingresan las palabras



TRIE

Se crean 27 nodos con 27 subhijos cada uno (representan las letras del abecedario). Se cargan todos comodatos nulos y a medida q se agregan palabraas se van cargando. Da igual el orden en el cual se carguen las palabras. Ideal para manejar mucha informacion y encontrar la info rapido

Hashing

Monday, January 3, 2022 3:09 PM

Las letras pasar a valer según la tabla ASCII. Almacena los elementos
egun su peso. Agarra el peso y los multiplica por 128

Funciones de hashing: Tamaño tabla

- Si la cantidad de claves a guardar es *n*, el tamaño de la tabla *m* debería ser tal que la proporción entre *n* y *m* sea aproximadamente 0.8.
- Esta proporción se llama *factor de carga* λ.
- Entonces:

$$\lambda = n/m$$

y pedimos:

$$\lambda <= 0,8$$

Funciones de hashing: Índices

- Si nuestras claves no son de tipo numéricas enteras debemos convertirlas a un formato de ese tipo.
- Si son letras, podríamos pasarlas a un *valor entero* con alguna convención, como tomar sus valores ASCII y sumarlos:

"ab" = 97 + 98 = 195

- Como los códigos ASCII se representan con 128 posiciones se aconseja aplicar esta base para dichos códigos:

ab = 97 x 128 + 98 x 128 = 12416 + 12544

0.8 es el valor ideal

DIVISION

Funciones de hashing: División

- Se divide la clave *k* por la cantidad de posiciones de la tabla t y se toma el resto (operadores mod , %).
- Entonces:

$$p = h(k) = k\%t$$

- El operador resto nos genera valores que van en el rango 0..*t* – 1.
- Se recomienda que el valor de *t*, que es el tamaño de la tabla, sea un *número primo*.

EJEMPLO →

Funciones de hashing: División

Ejemplo.

- La cantidad de claves a almacenar es 5000.
- Con un factor de carga de 0.8 deberíamos tener un tamaño de la tabla aproximado a:

$$t = 5000/0,8 = 6250$$

- El primo superior más cercano es **6257**.
- La función de hash a utilizar será:

$$p = k\%6257$$

Funciones de hashing: División

Ejemplo.

- Si necesitamos ingresar a la tabla la clave 113521 deberá ir en la posición:

$$p = 113521\%6257 = \mathbf{895}$$

- Y la clave 28413 debe ir a la posición:

$$p = 28413\%6257 = \mathbf{3385}$$

- Si luego debemos ingresar el dato con la clave 44694:

$$p = 44694\%6257 = \mathbf{895}$$

k= 195 en el ejemplo
T = tamaño de tabla.
% = modula. Tomar el resto de una division.
Resultado de k%t es la posicion

FOLDING

MULTIPLICACION

Funciones de hashing: Multiplicación

- Se multiplica a la clave por un valor *A* tal que $0 < A < 1$.
- Se toma la parte fraccionaria del resultado y se la multiplica por el tamaño de la tabla.
- El resultado final se redondea y da la posición final.

h(k) = t * ((k * A) mod 1)

Funciones de hashing: Folding

Ejemplo. El número de CUIT 23-31562313-7 podemos dividirlo en 4 partes tomando de a 3 dígitos: 233 - 156 - 231 – 37.

- Estos valores los sumamos: 233 + 156 + 231 + 37 = **657.**
- Si el tamaño de la tabla es menor al número obtenido se aplica la *función módulo*.
- En el caso de claves de tipo *string* se puede hacer algún corte obteniendo strings más pequeños para realizar un *xor* entre ellos.
- Logra mejor dispersión.

EJEMPLO →

Funciones de hashing: Folding

Ejemplo.

- Se tiene la cadena *s* = "*abcd*".
- La cortamos en dos cadenas: *s1* = "*ab*" y *s2* = "*cd*", tomamos los valores ASCII de cada una.
- El valor de a es 97, el de b, 98, etc.
- Tomamos los bits correspondientes y hacemos un *xor*.

posición / valores	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
ab	0	1	1	0	0	0	0	1	0	1	1	0	0	0	1	0
cd	0	1	1	0	0	0	1	1	0	1	1	0	0	1	0	0
xor	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0

- El resultado anterior nos da: 512 + 4 + 2 = **518.**

MID-SQUARE

Funciones de hashing: Mid-square

- Toma la clave, la eleva al cuadrado, y luego se queda con los dígitos centrales.

Ejemplo.

- La clave es 1536, entonces se hace 1536^2 = 2359296
- Nos quedamos con los dígitos centrales: 592.
- Siempre se puede aplicar si fuera necesario la función módulo.

EXTRACTION

Funciones de hashing: Extraction

- Se ignora una parte de la clave y se utiliza la parte restante.

Ejemplo.

- Si utilizamos el mismo ejemplo del número de CUIT **23-315623 13-7**
- Tomamos los 4 primeros dígitos: 2331, con los últimos 4: 3137, o una combinación tomando los dos primeros con los dos últimos: 2337, etc.
- No logra una buena distribución pero es muy veloz.

RADIX TRANSFORMATION

Función hashing: Radix Transformation

- Toma la clave y la cambia de base.

Ejemplo.

- Si la clave es 425, lo asume como que está expresado en base 16.
- La nueva clave será:

$$K' = 4 \cdot 16^2 + 2 \cdot 16^1 + 5 \cdot 16^0 =$$
$$= 4 \cdot 256 + 2 \cdot 16 + 5 = \mathbf{1061}$$

Función hashing: Universal hashing

- Elige para cada clave una función de hashing *aleatoria* de un conjunto de funciones a disposición.
- Reduce muchísimo las chances de que suceda el peor caso O(n), ya que es muy poco probable que todas las claves vayan a parar al mismo lugar.

Hashing: Colisiones

Monday, January 3, 2022 3:50 PM

Es un vector cerrado, no dinamico ni flexible.

HASH CERRADO

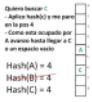
Colisiones: Hash cerrado

- Todos los elementos se guardan en la misma tabla (vector).
- Hash(clave) = posición en el vector.
- Si dos claves tienen la misma posición se tiene que resolver la colisión con los siguientes métodos: Open Addressing, Linear Probing, Quadratic Probing, Double Hashing

BORRADO

Colisiones: Borrado

- Esto trae otro problema, porque si yo ahora intento de buscar o borrar C.



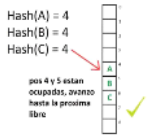
Colisiones: Open addresing

- Si se produce una colisión se busca una nueva posición tomando alguna nueva función que prueba en primer lugar con el valor 1, luego con el 2, etc.
- Si $h(k)$ está ocupada prueba con $h(k) + p(1)$, y si también lo está intenta con $h(k) + p(2)$, etc, donde p es una nueva función.
- Al finalizar siempre se aplica la función módulo.

Colisiones: Linear probing

- *Sondeo lineal* es la decisión más simple del direccionamiento abierto.
- Se define la función p como $p(i) = i$ por lo tanto, lo que hace es ir verificando las posiciones siguientes a la que la función de hash indica.
- Por ejemplo, si $h(k) = 132$ y esta posición no está libre, se intenta en la posición 133, luego en la 134, etc.
- Este procedimiento se repite hasta alcanzar el final de la tabla, en cuyo caso se comienza desde el principio hasta encontrar la primera posición libre.

EJEMPLO



Colisiones: Quadratic probing

- Una mejora es tomar la función p como cuadrática.
- En este caso $p(i) = i^2$. Por lo tanto, en el mismo ejemplo anterior, si $h(k) = 132$ se encuentra ocupada, intenta

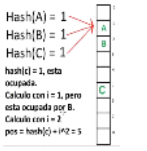
$$h(k) + p(1) = 132 + 1^2 = \mathbf{133}$$

- Si esta posición está también ocupada, intenta con

$$h(k) + p(2) = 132 + 2^2 = \mathbf{136}$$

- Los datos quedan de forma más dispersa y no se agrupan en ciertas zonas.

EJEMPLO



Colisiones: Double hashing

- Se usan 2 funciones de hash para obtener la posición.
- La *secuencia* hasta encontrar una posición vacía es la siguiente:

$$p = h(k) + i \cdot h_2(k)$$

con $i = 0, 1, 2...$

- Si se produce una colisión se suma una nueva función de dispersión, en el caso de seguir colisionando, se sumará el doble de dicha función, etc.
- Se debe finalizar tomándole módulo con el tamaño de la tabla.

EJEMPLO

Colisiones: Double hashing

Ejemplo.

- La función $h(k)$ toma el módulo de la clave con cierto valor t (tamaño de la tabla), digamos que $t = 1019$.
- La segunda función de hash también sea módulo pero con un valor distinto, por ejemplo, 1018.
- Si tenemos la clave 15924, $h(15924) = 639$.
- En caso de tener ocupada esa posición hacemos:

$$p = h(15924) + h_2(15924)$$
$$= 639 + 654 = \mathbf{1293}$$

Colisiones: Double hashing

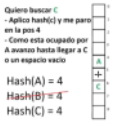
Ejemplo.

- A este valor volvemos a aplicarle módulo con 1019 y queda **274**.
- Si tuviéramos una nueva colisión, los cálculos serían:

$$p = h(15924) + 2 \cdot h_2(15924)$$
$$= 639 + 2 \cdot 654$$
$$= 639 + 1308 = \mathbf{1947}$$

Colisiones: Borrado

- Esto trae otro problema, porque si yo ahora intento de buscar o borrar C.



Es muy importante q al borrar dejemos una marca ya q sino, en el ejemplo, si quiero borrar C que deberia estar en la poosicion 4, voy a ella y bajo hasta encontrarla. Pero donde estaba B (5) ya no esta mas entonces el sistema piensa q no hay mas nodos, lo cual seria incorrecto. Entonces si dejamos la marca, el sistema va a seguir buscando xq sabe que no termina ahí.

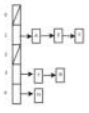
Colisiones: Borrado

- Dejar una marca indicando que el lugar esta libre, pero hubo un elemento.

HASH ABIERTO

Colisiones: Hash abierto (chaining)

- Cada posición de la tabla en realidad será un puntero a una lista enlazada con las claves.
- Por ejemplo, supongamos que las claves A, E y F devuelven el valor 1 luego de aplicarles la función de hash, las claves B y T, el valor 3 y H el valor 4.
- Entonces, si las claves ingresan en el siguiente orden: A, H, E, T, F y B, la tabla queda:

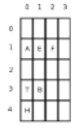


Desventaja es q ocupa mas memoria

BUCKET ADDRESING

Colisiones: Bucket addresing

- Varios elementos se almacenan en una misma posición de la tabla.
- Se utiliza una estructura de matriz.
- Cada posición es un bloque que contiene varios elementos por lo que se las llama "buckets".

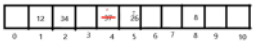


Ineficiente ya q al ser una matriz, voy a tener q usar mas memoria y no se van a utilizar todos sus lugares.

Ejercicio Resuelto Hashing

Monday, January 3, 2022 4:59 PM

Ejercicio 1: Si se usa módulo de 11 en una tabla de tamaño 11 y la solución de colisiones es direccionamiento abierto, indicar cómo funciona cada operación y cómo va quedando la tabla en cada paso: insertar(37), insertar(12), insertar(8), insertar(26), eliminar(37), insertar(34), buscar(26).



h(37)=37%11=4
h(12)=1
h(8)=8
h(26)=4
i=1, G(0,i)=(h(k)+i^2)%11
G(26,1)=(h(26) + 1^2) %11= (4+1)%11=5
h(37)=4
h(34)=34%11=1
i=1, G(34,1)=(h(34)+1^2)%11=2

Ejercicio 2 : Si se usa la función de hashing folding, tomando el ultimo digito de la clave y la resolución de colisiones es mediante direccionamiento abierto (sondeo cuadrático), indicar como funciona cada operación y como va quedando la tabla en cada paso:

insertar(47) h(47)=7

insertar(37) h(37)=7 -> i=1, G(37,1)=(7+1)%8=0

insertar(21) h(21)=1

insertar(24) h(24)=4

insertar(10) h(10)=0 ->i=1, G(10,1)=(0+1)%8=1
i=2,G(10,2)=(0+ 2^ 2)%8=4
i=3 G(10,3)=(0+ 3^2)%8=1
...

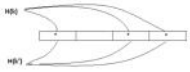
En este caso no funciona este algoritmo. Hay q aclararlo en un examen.

Colisiones: Linear/Quadratic probing

- Los algoritmos de sondeo tienen el mismo problema:
- Siendo:

k<>k'
H(k)=H(k')

- Los registros van a hacer el mismo recorrido para intentar resolver las colisiones.



Colisiones: Linear/Quadratic probing

- Para resolverlo se puede usar un método dependiente de la clave:

G(k,i)=(H(k) + di)%t con i:1..t

d=max(1,k /t)

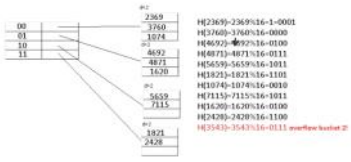
Colisiones: Bucket addressing (Extensible)

Ejercicio 3: (Hashing extensible). Suponiendo la siguiente función de hashing: h(k)=k mod 16, usando como tratamiento de colisiones el bucket adressing, con buckets de 3 registros, insertar los siguientes registros: 2369, 3760, 4642, 4871, 5657, 1821, 1074, 7115, 1620, 2428, 3543, 4750.

H(2369)=2369%16=1-0001
H(3760)=3760%16=0000
H(4692)=4692%16=0100
H(4871)=4871%16=0111
H(5659)=5659%16=1011
H(1821)=1821%16=1101
H(1074)=1074%16=0010
H(7115)=7115%16=1011
H(1620)=1620%16=0100
H(2428)=2428%16=1100

Pasa todos los numeros a binarios con su modulo.

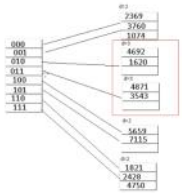
Colisiones: Bucket addressing (Extensible)



H(2369)=2369%16=1-0001
H(3760)=3760%16=0000
H(4692)=4692%16=0100
H(4871)=4871%16=0111
H(5659)=5659%16=1011
H(1821)=1821%16=1101
H(1074)=1074%16=0010
H(7115)=7115%16=1011
H(1620)=1620%16=0100
H(2428)=2428%16=1100
H(3543)=3543%16=0111 overflow bucket 2!

Divide en cuadrantes de 3 por sus dos primeros digitos binarios.
Hay overflow ya que en un bucket hay 4 correctas.

Colisiones: Bucket addressing (Extensib



Duplica los buckets iniciales agregando un digito mas.
Se puede almacenar todo correctamente.


le)

Estructura de datos

Tuesday, January 4, 2022 10:23 AM

Concepto de Pila:

- Una pila es una estructura de datos de entradas ordenadas tales que solo se pueden introducir y eliminar por un extremo, llamado cima.

El ultimo en entrar es el primero en salir

Colas: El primer dato en entrar es el primero en salir. Los nodos entran por atrás y salen por adelante.

Listas

Tuesday, January 4, 2022 11:31 AM

Listas Simplemente Enlazadas

- Cada nodo (elemento) contiene un único enlace que conecta ese nodo al nodo siguiente o nodo sucesor. La lista es eficiente en recorridos directos ((<adelante»).

Solo puntero siguiente

Listas Doblemente Enlazadas

- Cada nodo contiene dos enlaces, uno a su nodo predecesor y el otro a su nodo sucesor. La lista es eficiente tanto en recorrido directo («adelante») como en recorrido inverso («atrás»).

Puntero siguiente y puntero anterior

Lista Circular simplemente Enlazada

- Una lista enlazada simplemente en la que el último elemento (cola) se enlaza al primer elemento (cabeza) de tal modo que la lista puede ser recorrida de modo circular («en anillo»).

Solo puntero siguiente

Lista Circular Doblemente Enlazada:

- Una lista doblemente enlazada en la que el último elemento se enlaza al primer elemento y viceversa. Esta lista se puede recorrer de modo circular (en anillo) tanto en dirección directa («adelante») como inversa («atrás»).

Puntero siguiente y puntero anterior

Complejidad

Tuesday, January 4, 2022 12:45 PM

Eficiencia algorítmica

- Recursos
  - Complejidad temporal: tiempo necesario para ejecutar un algoritmo.
  - Complejidad espacial: cantidad de memoria necesaria para ejecutar un algoritmo.

Complejidad temporal

- Estudios
  - A posteriori o enfoque empírico: se mide el tiempo real de un algoritmo para distintos valores de entrada.
  - A priori o enfoque teórico: se determina una función que se relaciona con el tiempo de ejecución del algoritmo para cierto valor de entrada.

Contras:

- Depende de la compu, puede tardar más o menos. No sirve para comparar
- Depende la acción, puede q tengas q esperar mucho.

Algunos conceptos

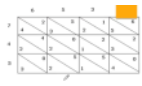
- Problema: necesidad inicial para la que se busca alcanzar una solución.
- Ejemplar: individuo de una especie o género.
- Algoritmo: serie de pasos precisos, ordenados y finitos cuyo objetivo es hallar la solución a un problema.

EJEMPLOS

- Problema: obtener el producto de dos números.
- Ejemplar: (15, 23)
- Algoritmo: mediante sumas, hindú, japonesa, ruso.

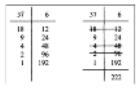
- Problema: ordenar un vector de n elementos.
- Ejemplar: (8, 15, 4, 23, 6) con n = 5
- Algoritmo: selección, burbujeo, inserción.

MÉTODO HINDU



MÉTODO JAPONES

MÉTODO RUSO



PRINCIPIO DE INVARIANZA

Principio de invarianza

Dado un algoritmo y dos implementaciones del mismo, las cuales podrían ser en la misma máquina o en dos distintas, llamemos $I1$ e $I2$ a estas implementaciones, que tardarán un tiempo $T1(n)$ y $T2(n)$ respectivamente, entonces:

existe una constante real $c > 0$, y $n_0 \in N$ tales que $\forall n \geq n_0$ se verifica que:

$$T1(n) \leq c.T2(n)$$

OPERACIONES ELEMENTALES

Conteo – Operaciones elementales

- Asignación de variables
- Operaciones aritméticas básicas (suma, resta, producto y división)
- Comparaciones lógicas
- Llamadas y retornos de funciones
- Acceso a una estructura indexada (vector)

CONTEO EJEMPLO 1

```
int a, b = 5;
a = b + 1;
a++;
```

RTA

```
int a, b = 5;        /
a = b + 1;           /
a++;                 /
```

Total : T(n) = 5 O.E.

Viendo la slide de Op.elem, sa
asignación de valores es 1, op
es 1 y etc. Entonces en cada li
cuantas de esas se cumplen y

EJEMPLO 2

```
int a, b = 2;
cin >> a;
if (a > 5)
  {
    a = a + 3;
    b = a + 1;
  }
else
  b = 1;
```

RTA

```
int a, b = 2;     // 1 O.E.
cin >> a;         // 1 O.E.
if (a > 5)        // 1 O.E.
  {
    a = a + 3; // 2 O.E
    b = a + 1; // 2 O.E
  }
else
  b = 1;          // 1 O.E.
```

Total: 4 OE o 7 OE

Depende si entra o no al if la car
operaciones elementales. Por es

Conteo – Casos a considerar

- Mejor caso: secuencia de un algoritmo (traza) en la cual se ejecuta la cantidad mínima de operaciones.
- Peor caso: traza del algoritmo en la cual se ejecuta la cantidad máxima de operaciones.
- Caso promedio: corresponde al promedio de todas las trazas ponderado según su probabilidad.

EJEMPLO

```
int busq
  int p
  while
    if
    el
  }
  if (
  retur
}
```

```
void ordenar (int v[ ], int n) {
    for (int i = 0; i < n - 1; i++)
        for (int j = i + 1; j < n; j++)
            if (v[j] < v[i])
                inter (v[i], v[j]);
}
```

```
void inter (int &a, int &b)
{
    int aux = a;
    a = b;
    b = aux;
}
```

Yo creo q es 17n + 1

1 o.e.

2 o.e.

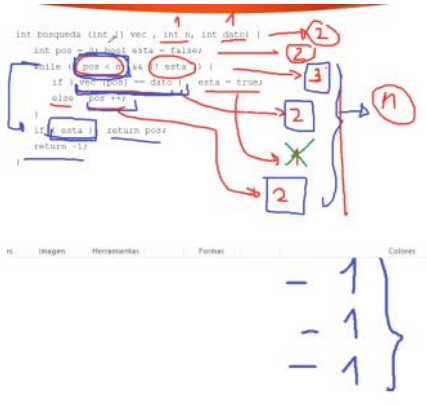
2 o.e.

bemos q la
eraciones mat
nea calculamos
sumamos

tidad de
o hay dos rtas.

```
ueda (int [] vec , int n, int dato) {
os = 0; bool esta = false;
 (( pos < n) && (! esta )) {
  ( vec [pos] == dato )   esta = true;
se   pos ++;

esta )   return pos;
n -1;
```



OPERACIONES ELEMENTALES TOTALES:

$2 + 2 + 7*N + 3 = 7n + 7$

$f(n) = 7n + 7$

1. La **primer linea** suma **2** ya que declara dos variables (si fueran punteros no cuentan, por eso el vector no aplica).

2. **Segunda linea** declara dos variables.
3. **Tercer linea** suma 3 ya que compara lo recuadrado y ademas los &&.
4. **Cuarta linea** suma dos ya que hay una comparacion y una busqueda en vector. En caso de ser correcta, sumaria uno mas.
5. Si es falsa, aplica la **quinta linea**, la cual suma y declara. Entonces 2.
6. La **sexta linea** suma 1 si es false y 2 si es true.
7. **Septima linea** suma uno.

La conclusion para las operaciones elementales totales considerando el peor caso, serian los 7 de la suma de las lineas antes del while y despues del mismo cumpliendo la condicion de la linea 6.
Luego, los otros 7 se multiplican ya que suma las 3 condiciones del while, las dos del if y las dos del else (ya q al ser el peor caso no se va a cumplir la condicion). Todo esto se multiplica por n ya que va a ser la longitud del vector, por ende la canidad de repeticiones del while.

Complejidad: Algrotimos recursivos

Friday, January 7, 2022 11:45 AM

- Métodos de expansión
- Métodos matemáticos
- Teorema maestro

- Si la ecuación de recurrencia es:

$$T(n) = \begin{cases} c.n^d & si\ 1 \leq n < b \\ a.T\left(\frac{n}{b}\right) + c.n^d & si\ n \geq b \end{cases}$$

- Entonces:

$$T(n) = \begin{cases} \theta(n^d) & si\ a < b^d \\ \theta(n^d.\log(n)) & si\ a = b^d \\ \theta(n^{log_b(a)}) & si\ a > b^d \end{cases}$$

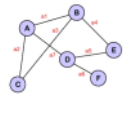
Teorema maestro. Ejemplo 2.

Grafos

Wednesday, January 5, 2022 10:27 AM

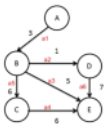
Representaciones - Matriz de Incidencia

	A	B	C	D	E	F
a1	1	1	0	0	0	0
a2	1	0	1	0	0	0
a3	0	1	1	0	0	0
a4	0	1	0	0	1	0
a5	0	0	0	1	1	0
a6	0	0	0	1	0	1
a7	1	0	0	1	0	0



Representaciones - Matriz de Incidencia (II)

	A	B	C	D	E
a1	-3	3	0	0	0
a2	0	-1	0	1	0
a3	0	-5	0	0	5
a4	0	0	-6	0	6
a5	0	-6	6	0	0
a6	0	0	0	-7	7



Representaciones - Matriz de Incidencia (III)

Cuánto cuesta:

- ¿En Espacio? O(V*E)
- ¿Agregar un vértice? O(V*E)
- ¿Agregar una arista? O(V*E)
- ¿Ver si dos vértices son adyacentes? O(E)
- ¿Obtener los adyacentes de un vértice? O(E)
- ¿Si está implementado como una lista de aristas?

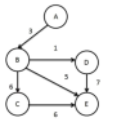
Esto seria con caminos con peso y direccion.

Representaciones - Matriz de Adyacencia

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

Representaciones - Matriz de Adyacencia (II)

	A	B	C	D	E
A	0	3	0	0	0
B	0	0	6	1	5
C	0	0	0	6	0
D	0	0	0	0	7
E	0	0	0	0	0



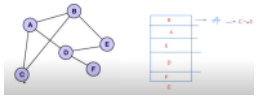
Representaciones - Matriz de Adyacencia (III)

Cuánto cuesta:

- ¿En Espacio? O(V^2)
- ¿Agregar un vértice? O(V^2)
- ¿Agregar una arista? O(1)
- ¿Ver si dos vértices son adyacentes? O(1)
- ¿Obtener los adyacentes de un vértice? O(V)

Esto seria con caminos con peso y direccion.

Representaciones - Listas de adyacencia



Representaciones - Listas de Adyacencia (II)

Cuánto cuesta:

- ¿En Espacio? O(V+E)
- ¿Agregar un vértice? O(1) u O(V)
- ¿Agregar una arista? O(V)
- ¿Ver si dos vértices son adyacentes? O(V)
- ¿Obtener los adyacentes de un vértice? O(V)

Representaciones - Matriz de Ady
Listas de adyacencia (II)

Cuánto cuesta:

- ¿En Espacio? O(V+
- ¿Agregar un vértice? O(1)
- ¿Agregar una arista? O(1)
- ¿Ver si dos vértices son adyacentes? O(1)
- ¿Obtener los adyacentes de un vértice? O(1) u

acencia ++

E)

O(V)

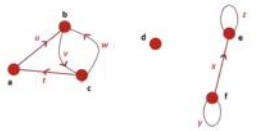
Grafos: Tipos

Wednesday, January 5, 2022 11:15 AM

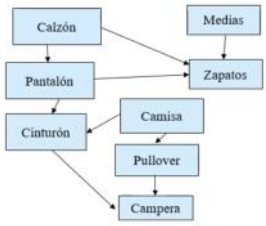
Grafos dirigidos (digrafo)

El conjunto de aristas lo indicamos como pares ordenados de vértices:

A = { (a, b), (c, a), (b, c), (c, b), (e, e), (f, e), (f, f) }



EJEMPLO

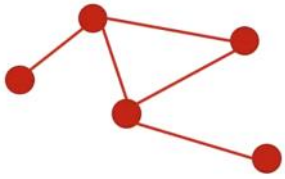


Reloj

Reglas para vestirse

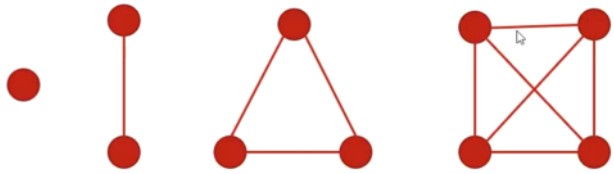
Grafo simple

Un grafo simple es el que no tiene bucles ni aristas paralelas.



Grafo completo

Un grafo completo de n vértices es un grafo simple con una arista que conecta a cada par de vértices.



Grafos: Caminos y conectividad

Wednesday, January 5, 2022 11:20 AM

Caminos, circuitos

Sea G un grafo y **u** y **v** vértices en G. Entonces:

- Un **camino** de u a v es una sucesión finita de vértices adyacentes y aristas.
  - Si el grafo es simple, alcanza con indicar solamente los vértices.
- Un **sendero** de u a v es un camino que no repite aristas.
- Una **trayectoria** de u a v es un sendero que no repite vértices.
- Un **camino cerrado** es un camino que comienza y termina en el mismo vértice.
- Un **circuito** es un camino cerrado que no repite aristas.
- Un **circuito simple** es un circuito que no repite vértices, a excepción del primero y el último.

	¿Arista repetida?	¿Vértice repetido?	¿Inicia y finaliza en el mismo punto?	¿Debe contener al menos una arista?
Camino	Permitido	Permitido	Permitido	No
Sendero	No	Permitido	Permitido	No
Trayectoria	No	No	No	No
Camino cerrado	Permitido	Permitido	Sí	No
Circuito	No	Permitido	Sí	Sí
Circuito simple	No	Solo primero y último	Sí	Sí

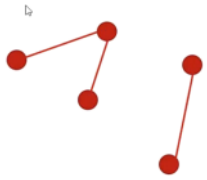
CONECTIVIDAD

Sea G un grafo y u y v dos vértices de G. Decimos que u y v son conexos sí y solo sí existe un camino de u a v.

El grafo G es conexo si y solo si, dados dos vértices cualquiera, hay un camino que los conecta.

EJEMPLO

Grafo conexo



Grafo desconexo

Si un grafo G es conexo y contiene un circuito, se puede quitar una arista del circuito que G seguirá siendo conexo.

Si un grafo G es conexo y está libre de circuitos, es un **árbol**.

Si un grafo G está libre de circuitos pero no es conexo, se llama **bosque** (varios árboles).

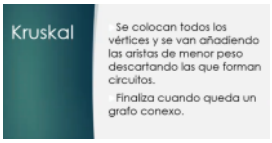
Dijkstra

Wednesday, January 5, 2022 12:02 PM

- Es un algoritmo para la determinación del *camino mínimo*, dado un **vértice origen**, hacia el resto de los vértices en un grafo que tiene pesos en cada arista.
- Consiste en ir *explorando todos los caminos más cortos* que parten del vértice origen y que llevan a todos los demás vértices.
- Cuando se obtiene el camino más corto desde el vértice origen hasta el resto de los vértices que componen el grafo, el algoritmo se detiene.

Kruskal y Prim

Friday, January 7, 2022 10:05 AM

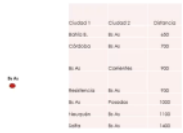


Ordeno las listas por menor costo y empiezo a agregar cada vertice e ir uniendo. Los que estan en rojo no estan incluidos ya que ya hay otra forma de llegar a ese destino y seria gastar mas plata por algo q ya esta hecho. Recordemos que la idea era conectar todo con el mejor costo posible.

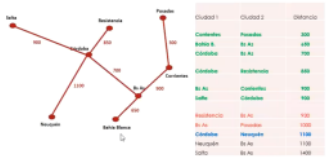
No se pueden formar **CIRCUITOS**.

A diferencia de dikxtra o floyd no me busca el camino minimo.

PRIM



Se arma una lista de aristas que inciden en dicho vértice, ordenadas por peso. Luego se añade la arista de menor peso de la lista junto con el vértice adyacente.

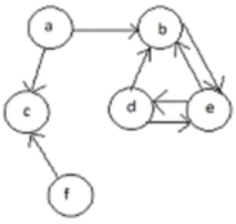


Arranco formando mi arbol a travez de BsAs.
Y de ahi empiezo a extenderbasado en el costo minimo. Cuando llego al vertice nuevo, tengo q fijarme si de ese vertice sale otro mas. En caso de q ocurra, lo agrego y asi sucesivamente. Si no hay mas vertices para agregar, continuo agregando de BsAs. Cada nuevo vertice se debe agregar a la lista de manera ordenada.

No se pueden formar **CIRCUITOS**.

Ejercicios Grafos

Friday, January 7, 2022 10:28 AM



Dado el grafo de la figura, indicar cómo es la salida con
1. Un recorrido en profundidad.
2. Un recorrido en anchura.

Indicar cómo implementaría cada uno.

Dado el grafo de la figura, indicar cómo queda el árbol de expansión mínimo según:
1. Kruskal
2. Prim

Indicar en cada uno, paso a paso cómo va quedando el grafo.

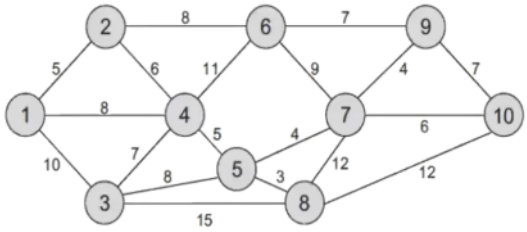
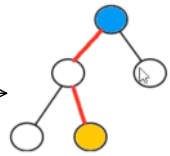


ABB: Balanceo de Arboles

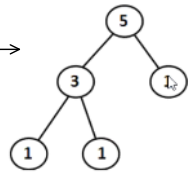
Friday, January 7, 2022 10:33 AM

Longitud del Camino → Cantidad de aristas que hay que recorrer para ir desde el nodo raiz hasta un nodo cualquiera →

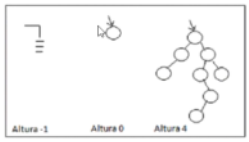


Longitud del camino: 2

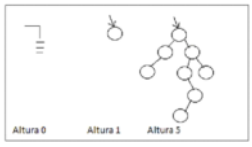
Peso del camino → La cantidad de nodos que conecta ese nodo (contandose a si mismo) →



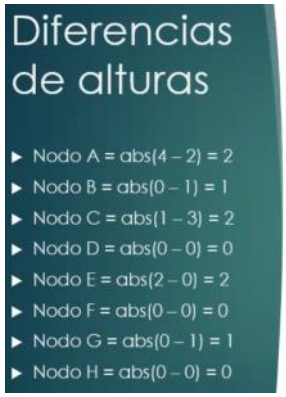
Altura →

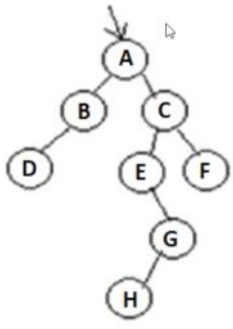


Cormen: longitud del camino más largo.

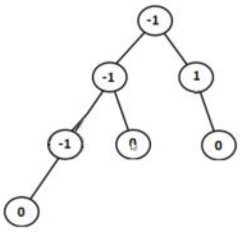


Drozdek: cantidad de niveles.



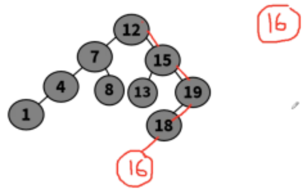


AVL

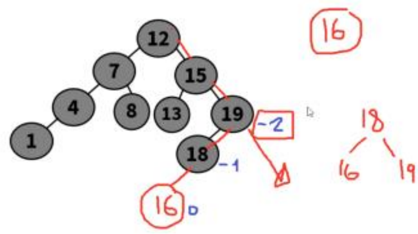


Me paro en un nodo y cuento el camino mas largo para la derecha y para la izquierda,
Ahi hago camino derecha - camino izq y tengo el fb.

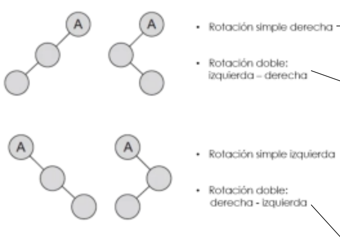
REBALANCEO

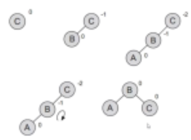


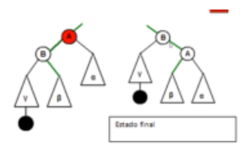
Agregue el 16, entonces tengo que calcluar devuelta los factores de balanceo (FB). Solo cambiano los que agregue, es decir los que tocan la linea roja

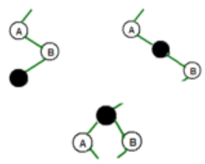


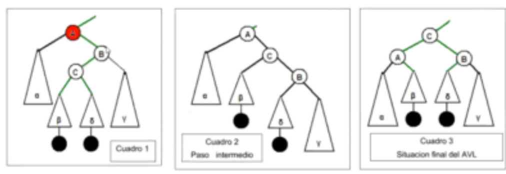
Quedo desbalanzeado (-2), entonces movemos los nodos para que queden bien devuelta. Solucion en rojo.









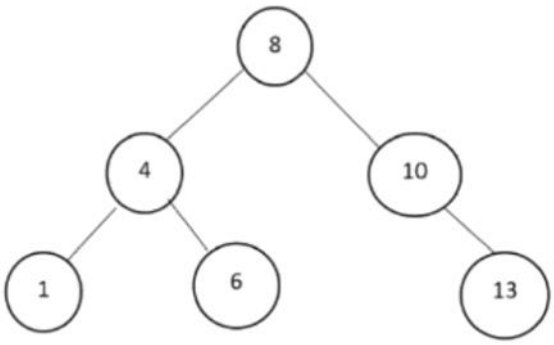


Caso extremo, muy complejo ya q tienen hijos. No nos lo van a tomar

Rotación	FB (p)	FB (q)	FB (r)
SD	- 2	0 o - 1	0
SI	2	0 o 1	0
D-ID	- 2	1	0
D-DI	2	- 1	0

Ejercicio AVL

Friday, January 7, 2022 11:27 AM



Insertar los próximos números en el árbol binario que se encuentra graficado:

5 – 22-17-2-12

2) dado un vector de números, crear un árbol avl insertando uno por uno empezando por el primero hasta el último:

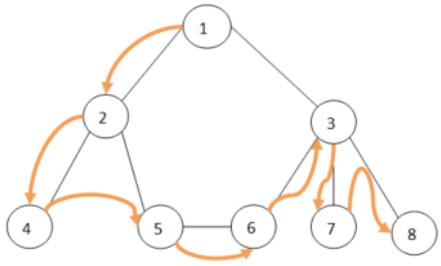
7-4-8-10-12-2-1-50-18

Graficar el árbol al eliminar el elemento 10

Busqueda en profundidad: BFS

Friday, January 7, 2022 12:25 PM

Una búsqueda en profundidad (DFS) es un algoritmo de búsqueda para lo cual recorre los nodos de un grafo. Su funcionamiento consiste en ir expandiendo cada uno de los nodos que va localizando, de forma recurrente (desde el nodo padre hacia el nodo hijo). Cuando ya no quedan más nodos que visitar en dicho camino, regresa al nodo predecesor, de modo que repite el mismo proceso con cada uno de los vecinos del nodo. Cabe resaltar que si se encuentra el nodo antes de recorrer todos los nodos, concluye la búsqueda



Aplicaciones →

- Encontrar nodos conectados en un grafo
- Ordenamiento topológico en un grafo acíclico dirigido
- Encontrar puentes en un grafo de nodos
- Resolver puzzles con una sola solución, como los laberintos
- Encontrar nodos fuertemente conectados

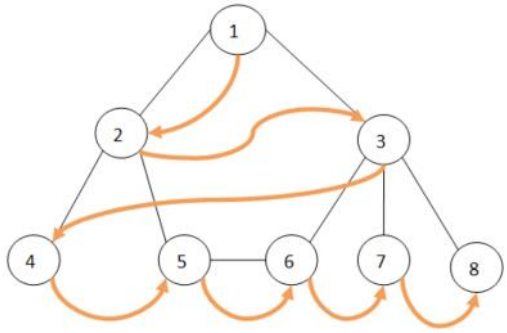
Árbol binario

- **Preorden**: (**raiz**, izquierdo, derecho). Para recorrer un árbol binario no vacío en preorden, hay que realizar las siguientes operaciones recursivamente en cada nodo, comenzando con el nodo de raíz:
  1. **Visite la raiz**
  2. Atraviese el sub-árbol izquierdo
  3. Atraviese el sub-árbol derecho
- **Inorden**: (izquierdo, **raiz**, derecho). Para recorrer un árbol binario no vacío en inorden (simétrico), hay que realizar las siguientes operaciones recursivamente en cada nodo:
  1. Atraviese el sub-árbol izquierdo
  2. **Visite la raiz**
  3. Atraviese el sub-árbol derecho
- **Postorden**: (izquierdo, derecho, **raiz**). Para recorrer un árbol binario no vacío en postorden, hay que realizar las siguientes operaciones recursivamente en cada nodo:
  1. Atraviese el sub-árbol izquierdo
  2. Atraviese el sub-árbol derecho
  3. **Visite la raiz**

Busqueda en anchura: BFS

Friday, January 7, 2022 12:31 PM

Una búsqueda en anchura (BFS) es un algoritmo de búsqueda para lo cual recorre los nodos de un grafo, comenzando en la raíz (eligiendo algún nodo como elemento raíz en el caso de un grafo), para luego explorar todos los vecinos de este nodo. A continuación, para cada uno de los vecinos se exploran sus respectivos vecinos adyacentes, y así hasta que se recorra todo el grafo. Cabe resaltar que si se encuentra el nodo antes de recorrer todos los nodos, concluye la búsqueda.



La búsqueda por anchura se usa para aquellos algoritmos en donde resulta crítico elegir el mejor camino posible en cada momento del recorrido.

APLICACIONES ⟶

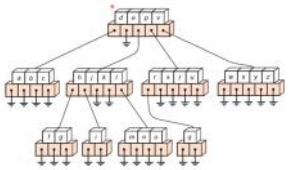
- Encontrar el camino más corto entre 2 nodos, medido por el número de nodos conectados
- Probar si un grafo de nodos es bipartito (si se puede dividir en 2 conjuntos)
- Encontrar el árbol de expansión mínima en un grafo no ponderado
- Hacer un Web Crawler
- Sistemas de navegación GPS, para encontrar localizaciones vecinas

Arboles Multivias, B y B+

Friday, January 7, 2022 1:02 PM

Arboles Multivia →

- Problema: cuando los datos se guardan en disco, se busca minimizar los accesos al mismo.
- Los ABB evolucionan a árboles de búsqueda de múltiples (m) vías:
  - Un nodo tiene a lo sumo m hijos y m – 1 claves.
  - Las claves en cada nodo están ordenadas de menor a mayor en espacios contiguos.
  - Cada clave tiene un puntero izquierdo a un nodo con claves menores (puede ser null) y uno derecho a un nodo con claves mayores (puede ser null).

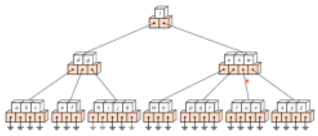


Tiene 5 vias y 4 claves.

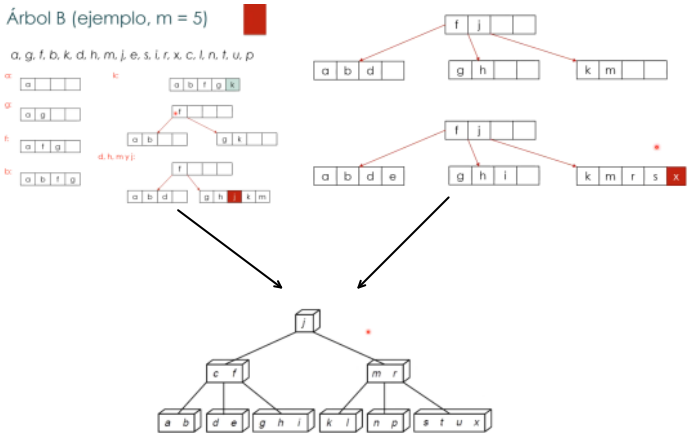
Arboles B →

Un árbol B de orden m es un árbol de búsqueda de m-vías en el que

1. La raíz, o bien es hoja, o bien tiene al menos dos hijos.
2. Todas las hojas están al mismo nivel.
3. Cada nodo, a excepción quizá del nodo raíz, tiene k – 1 claves, donde m/2 ≤ k ≤ m.
4. Cada nodo interno, a excepción quizá del nodo raíz, tiene k hijos, donde m/2 ≤ k ≤ m.

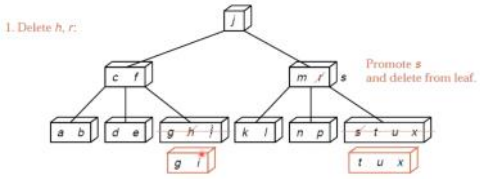


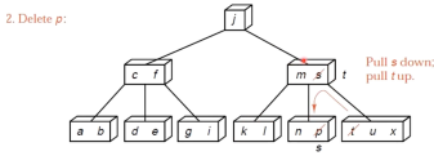
Tiene 5 vias, 4 claves. Lo podes sacar viendo el que mas tiene o multiplicando el minimo por dos y le sumas 1.

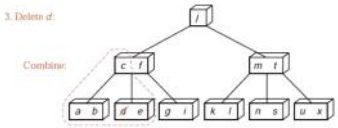


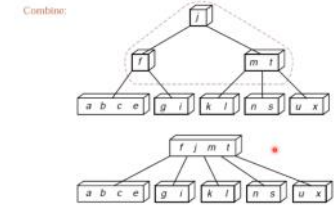
En este tipo de arboles, yo intento agregar un valor, pero si ya esta lleno (en este caso 4 claves), agrego la letra como si fuera la 5 y saco la letra del medio y la llevo para arriba como raiz. En caso de que el de arriba tmbn este lleno, hago el mismo proceso hasta termminar.

ELIMINACION



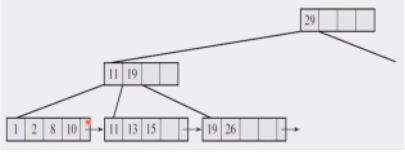




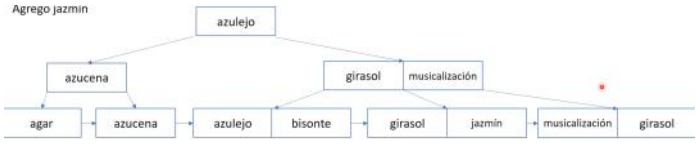


B+ →

- Toda la información está almacenada en los nodos Hoja.
- Todas las hojas están en el mismo nivel.
- Los nodos forman una lista simplemente enlazada.



La informacion esta siempre en los nodos hojas. Cuando el numero es igual, se agrega a la derecha pero siempre es el primero.



El orden de ingreso es azucena, girasol, agar,azulejo, xilofon, bisonete, musicalizacion, jazmin (el girasol de abajo a la izq es xilofon)

Ejercicios árboles multivías, B y B+

Mauro tiene una lista de palabras favoritas y le gustaría ver como varia agregar esas palabras en cada uno de los árboles vistos en la clase del día de algoritmos 2. Cuando finalmente termino de armar los árboles se encontró que no tiene forma de verificar si sus resultados son correctos por lo que les pidió a los alumnos de algoritmos 2 que realicen sus ejercicios para ayudarlo a comprobar si realizo bien los mismos.

1. Agregar las siguientes palabras a un árbol multivías con 3 vías: azucena, girasol, agar, azulejo, xilofón, bisonte, musicalización, jazmín.
2. Agregar las mismas palabras, pero en un árbol B. Luego eliminar la palabra azulejo. ¿Qué ocurriría si inserto las palabras al revés, es decir, empezando por jazmín y terminando con azucena?
3. Agregar las mismas palabras, pero en un árbol B+.

En cada uno de los ejercicios se debe mostrar paso por paso como se llega al árbol final.

Heap

Sunday, January 9, 2022 9:56 AM

Min-Heap

- The root node has the **minimum** value.
- The value of each node is equal to or greater than the value of its parent node.
- A complete binary tree.

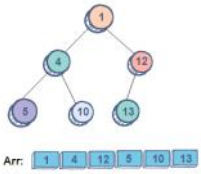
Max-Heap

- The root node has the **maximum** value.
- The value of each node is equal to or less than the value of its parent node.
- A complete binary tree.

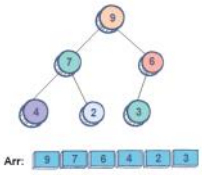
→ Por lo general estan representados como arrays.

COMLEJIDAD →

- **Get Max or Min Element**
  - The Time Complexity of this operation is O(1).
- **Remove Max or Min Element**
  - The time complexity of this operation is O(Log n) because we need to maintain the max/mix at their root node, which takes Log n operations.
- **Insert an Element**
  - Time Complexity of this operation is O(Log n) because we insert the value at the end of the tree and traverse up to remove violated property of min/max heap.



Arr: 1 4 12 5 10 13



Arr: 9 7 6 4 2 3

No importa el izq derecha como en los abb. Solo tiene q cumplir que sea mas chico o mayor.
NO EXSISTEN LOS **ARBOLES** EN **HEAP**, SON SOLO PARA VERLOS. **SIEMPRE** SON **VECTORES**

Árboles heap o montículos

Un heap debe cumplir:

- Restricciones de forma: AB completo o semicompleto (el último nivel se completa desde el lado izquierdo)
- Restricciones de orden.

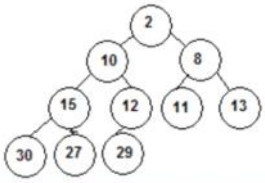

AB completo


AB semicompleto


No cumple la forma


AB completo

Estructura	Mínimo	Eliminar	Alta
Lista des.	O(N)	O(N)	O(1)
Lista ordenada	O(1)	O(1)	O(N)
Vector de colas	O(k)	O(k)	O(1)
ABB	O(N)	O(N)	O(N)
Heap	O(1)	O(log(N))	O(log(N))



2	10	8	15	12	11	13	30	27	29
0	1	2	3	4	5	6	7	8	9

p: nodo padre	hi: hijo izq.	hd: hijo der.
k	2k + 1	2k + 2
(m – 1) / 2	m	m

En el vector, si hago 2k +1 obtengo el hijo izq, y con 2k + 2 el derecho.
A medida que bajan los niveles estan mas lejos los hijos, por eso es a formula.
K = posicion en el vector. La division se redondea para abajo

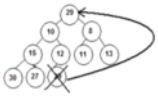
EXTRACCION DEL MINIMO

Extracción del mínimo (baja)

Se intercambia el mínimo que está en la primera posición del vector (raíz del árbol) por la última hoja (última posición del vector).

Se reduce el tamaño lógico del vector (heap).

El heap rompe las condiciones en la raíz, por lo que se restaura el mismo hacia abajo.



Se reemplaza 2 por 29

Se elimina el nodo hoja con 29.


El menor de los hijos es 8, que es menor que 29, por lo cual se intercambia con él


Ahora el problema se analiza para el nodo con 29 y sus hijos, 11 y 13.

29 se intercambia con 11


Árbol heap restaurado

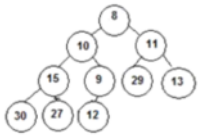
Borramos el 2. Entonces se pasa el ultimo adelante y lo vamos bajando intercambiando hasta encontrar un nuevo minimo. Se actualiza el tope del array.

INSERTAR NUMERO

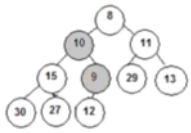
Agregar nuevo elemento (alta)

Se agrega el valor 9 al heap, como última hoja

El nuevo elemento se ubica como 'última hoja'. Luego se restaura el montículo analizando ternas hacia arriba hasta ubicar el nuevo elemento en su posición definitiva.

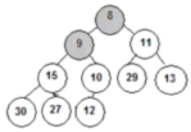


Como 9 es menor que 12, se intercambia con él. Ahora el análisis se debe realizar entre 9, y su padre, 10.



Como 9 es menor que 10, se intercambia con él

Ahora se analiza la situación de 9, y su padre 8.



Como 9 supera a 8 el proceso termina

Array de bits

Sunday, January 9, 2022 10:40 AM

- Se utilizan para indicar si un elemento, determinado por un subíndice, está o no en un conjunto de datos.
- Los datos tienen que ser de tipo numérico. De lo contrario, debe implementarse una función que los convierta a números.

En este array de un solo byte, se encuentran los valores que tienen un 1. Es decir, el 0, 2, 3 y 5.

D	H	B
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100
5	5	101
6	6	110
7	7	111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

D = Decimal
H = Hexadecimal
B = Binario

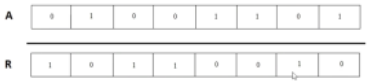
AND: &

Solo lo agrega al 1 si ambos lo tienen

OR: |

En el **or** si agrega si hay un 1. En caso de que hayan dos tmbn lo agrega

NOT: ~

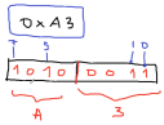


Es la inversa

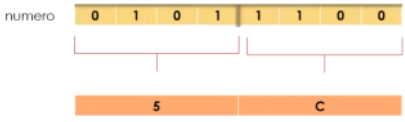
CORRIMIENTO

R = A << 4

Corre de lugar un bit la cantidad de lugares puestos.



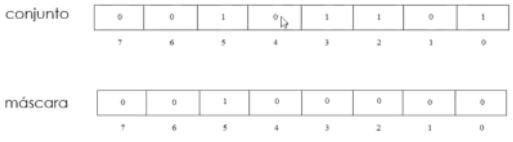
Con la direccion podes interpretar los bits completos y los que no.



Se multiplica por 16 ya que lo pasamos al hexadecimal.2c

numero = 5 * 16 + 12 = 80 + 12 = 92

CONSULTAR UN ELEMENTO



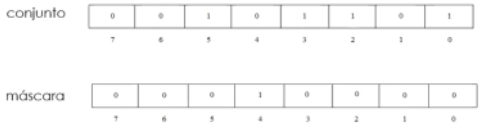

```cpp
#include <iostream>

using namespace std;

int main()
{
    unsigned char A = 0x2d;
    //unsigned char B = 0x29;
    unsigned char mascara = 0x01;
    mascara = A << 5;
    unsigned char resultado = A & mascara;
    if (resultado)
        cout << "El elemento esta en el conjunto" << endl;
    else
        cout << "El elemento NO esta en el conjunto" << endl;
    return 0;
}
```

Lo que se hace es crear una mascara con un unoy moverlo a la posicion que se quiere verificar si hay algo o no. Entonces al hacer el and, si hay un 1 ahi es por que en el conjunto tambien lo habias. Sino queda en 0 y no lo encuentra.

ALTA

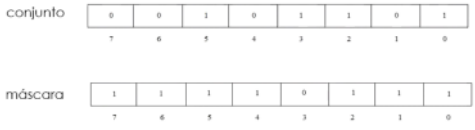


El elemento 4 se quiere agregar al conjunto. ¿Qué máscara hay que usar? ¿Qué operación entre bits hay que hacer?

Respuesta: hay que hacer un or entre ambos conjuntos.

Se hace un and para verificar que no haya nada en ese lugar, y

BAJA



El elemento 3 se quiere dar de baja del conjunto. ¿Qué máscara hay que usar? ¿Qué operación entre bits hay que hacer?

Respuesta: hay que hacer un not en la máscara y luego un and entre ambos conjuntos.

que hacer?

Respuesta: hay que hacer un or entre ambos conjuntos.

Se hace un and para verificar que no haya nada en ese lugar, y luego el or guardandolo.

```
unsigned char A = 0x2d;
//unsigned char A = 0x2d;
unsigned char mascara = 0x01;
mascara = A << 4;
unsigned char resultado = A & mascara;
/*
if (resultado)
    cout << "El elemento esta en el conjunto" << endl;
else
    cout << "El elemento NO esta en el conjunto" << endl;
    */
cout << (int)A << endl;

A = A | mascara;
cout << (int)A << endl;

return 0;
```

máscara hay que usar? ¿Qué operación entre bits hay que hacer?

Respuesta: hay que hacer un not en la máscara y luego un and entre ambos conjuntos.

COMPLEJIDAD

El costo de cualquier operación es constante, ya que el acceso a un vector así lo es, por lo que tanto la consulta, como la baja y el alta son O(1).

Ejercicio array de bits

Sunday, January 9, 2022 11:04 AM

```
unsigned char x = 0x6, y = 0x10, z = 0xb1;
unsigned char u, v, w;

u = ~z;
v = x & y;
w = x | z;

cout << x << y << z << endl;
cout << u << v << w << endl;
```

X =	0	0	0	0	0	1	1	0	176
Y=	0	0	0	1	0	0	0	0	16
Z=	1	0	1	1	0	0	0	1	177
U=	0	1	0	0	1	1	1	0	78
V=	0	0	0	0	0	0	0	0	0
W=	1	0	1	1	0	1	1	1	123

Array de int

Sunday, January 9, 2022 11:22 AM

Array de int

unsigned int vec[MAX];

¿Qué cálculo hay que hacer para consultar por el valor 85?

Se deben hacer dos cálculos:

1. Uno para calcular en qué posición del vector está dicho valor.
2. Luego, otro para calcular el deslizamiento.

Cálculos para consultar por un valor k.

1. Uno para calcular en qué posición del vector está dicho valor: índice = k / (sizeof(int) * 8)
2. Luego, otro para calcular el deslizamiento:
   d = k % (sizeof(int) * 8)

K = valor a consultar