Repaso de FAA - sacar coste teórico, teorema maestro

Si tenemos T(n/2) → n = 2^k, k = $\log_2(n)$

Tema 2 - Montículos y Hash

- Montículos: haces n/2, y empiezas a hundir desde ese elemente mirando si sus hijos son mayor(máximo), menor(mínimo) y cambiándolos. Después decrementamos en la posición de la tabla.

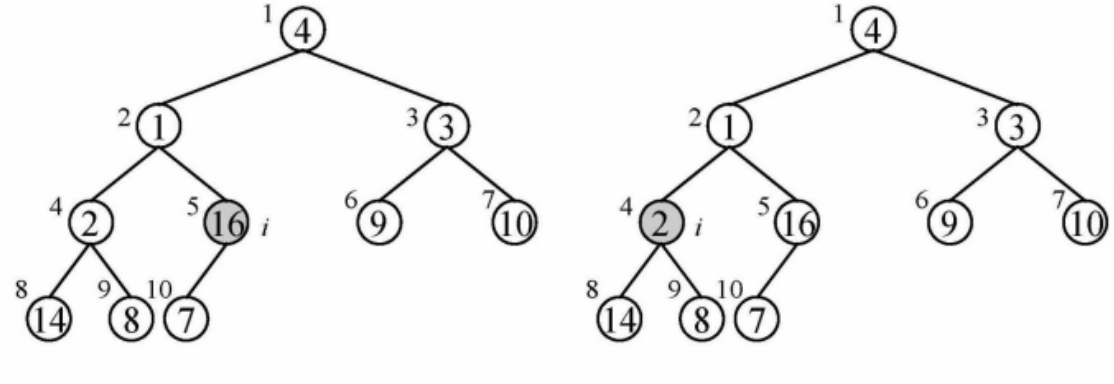
  Al terminar, ordenamos de menor a mayor, o de mayor a menor. Cogemos la ultima posición y lo cambiamos con el pivote, procedemos a hundir con el mayor(menor a mayor) o menor(mayor a menor)

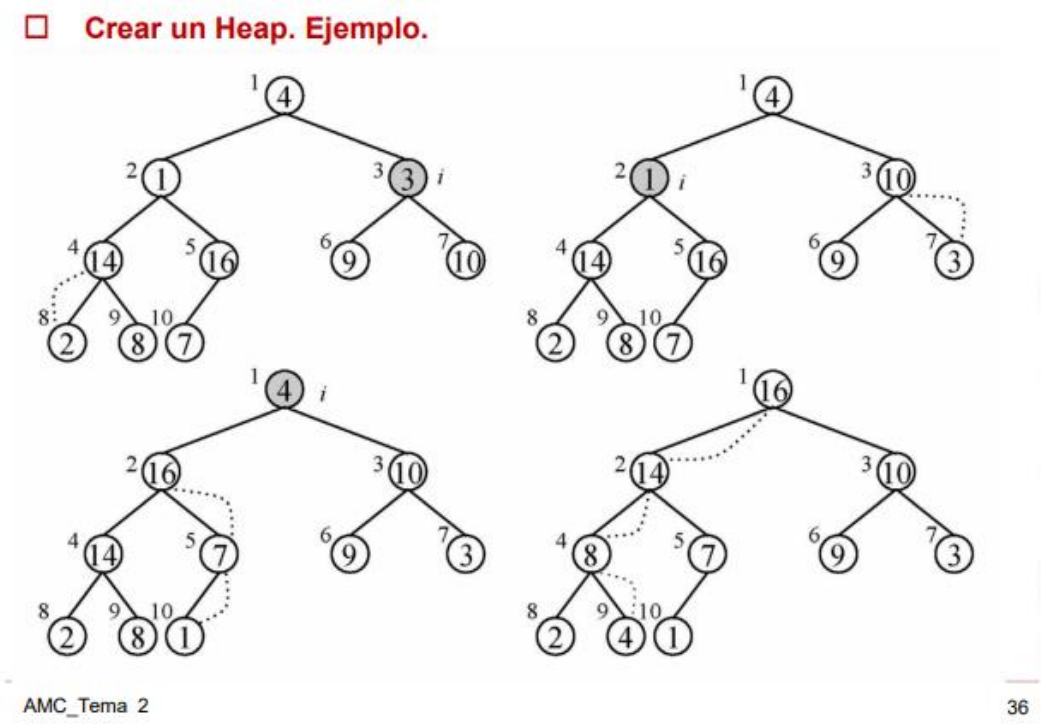
☐ **Crear un Heap. Ejemplo.**

```
procedimiento crear-montículo(T[1..n])
    para i ← [n/2] hasta 1 hacer
        hundir(T,i)
fprocedimiento
```

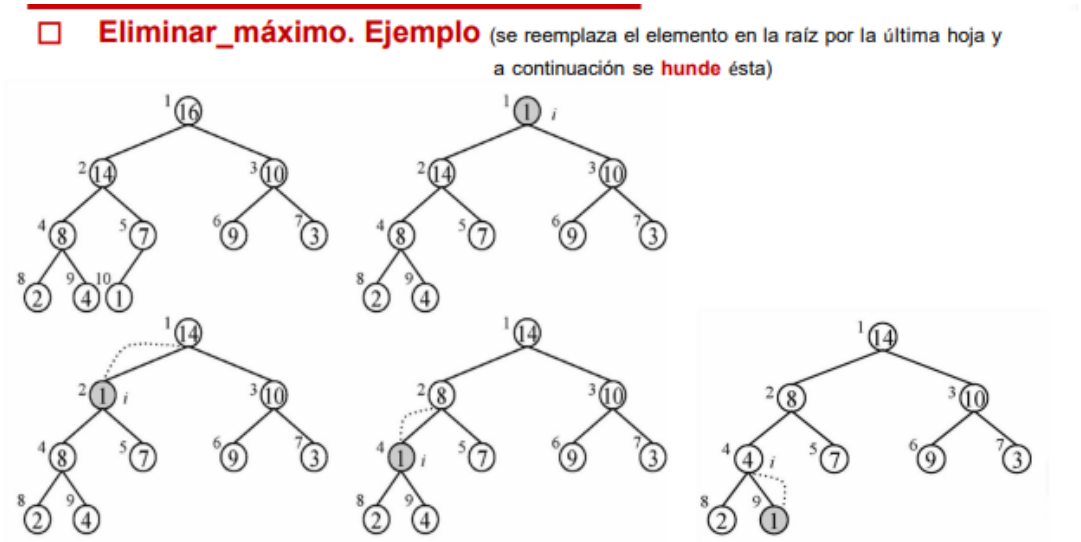
$$A = \{4, 1, 3, 2, 16, 9, 10, 14, 8, 7\}$$

Crear_heap: convierte un vector A en un Heap.

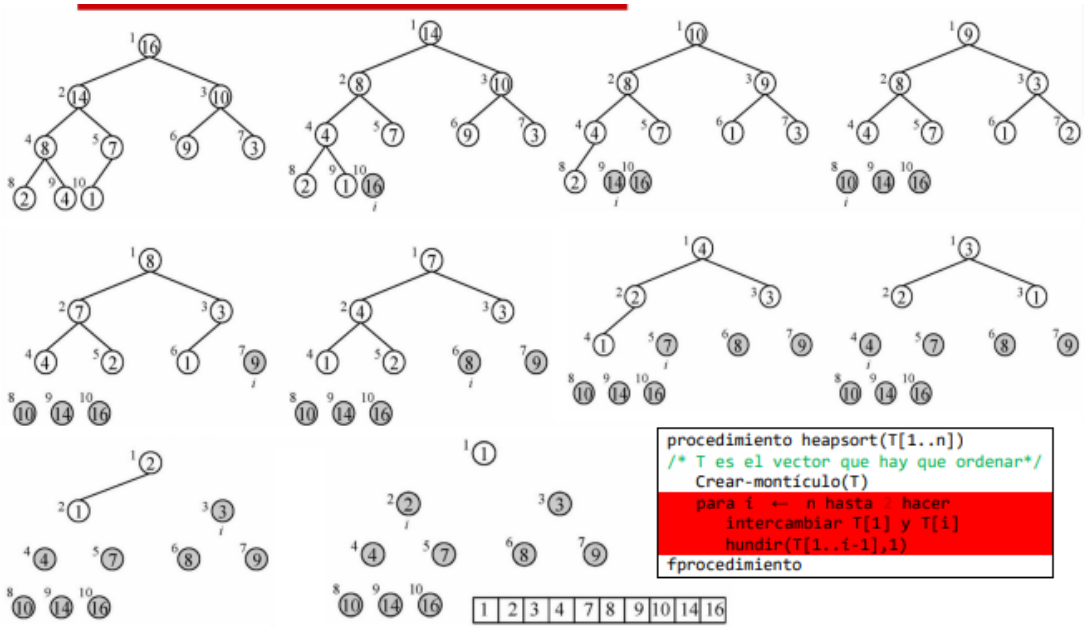




Aquí tenemos montículo de máximos, procedemos a ordenar el vector de menos a mayor.



Así hasta obtener el resultado.

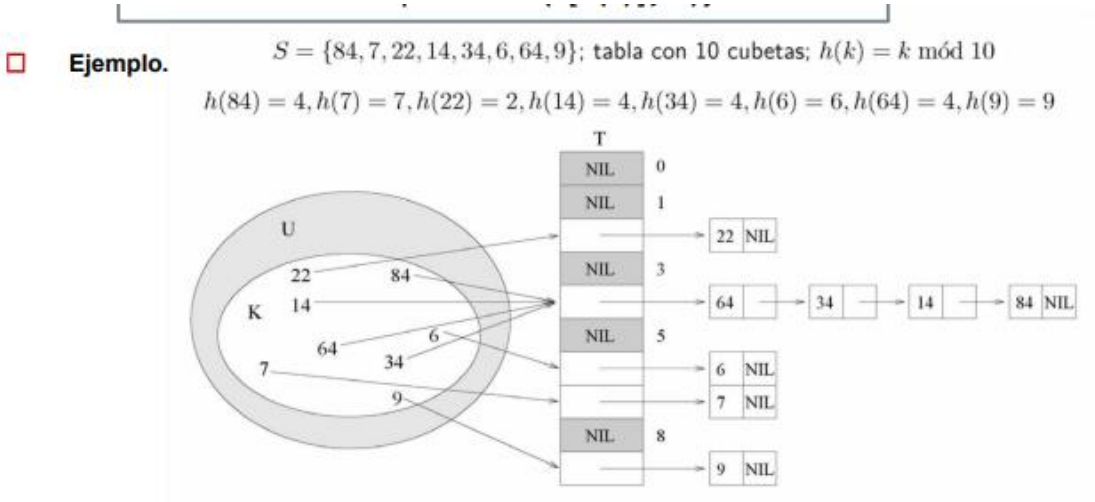


- Hash: elementos a almacenar/media elementos en una búsqueda = tamaño tabla

  Si Método de División -> primo cercano a tamaño tabla
  h(k) = k mod m

  Si Método de la Multiplicación -> potencia de 2 cercana a tam. tabla
  h(k) = L m ((kϕ) mód 1)⌋ 0 < ϕ < 1

  * Por encadenamiento.



  * Expresión cerrada (lineal o cuadrática)

Tabla hash inicializada a -1 todos sus elementos para indicar que está vacía

Clave	hash(clave)
1203	4
6754	35
32	33
8683	44
839	120
1363	44
1079	120

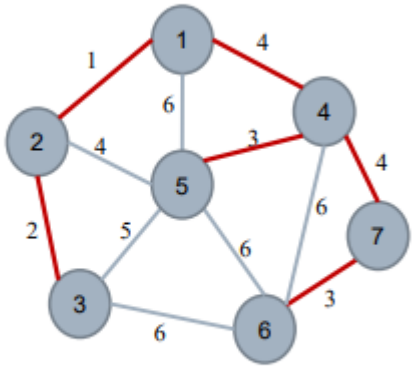
hash(clave) → clave **mod** 120 + 1

	Clave	Resto de campos
1	1079	
	-1	
4	1203	
	-1	
33	32	
	-1	
35	6754	
	-1	
	-1	
44	8683	
45	1363	
	-1	
	-1	
120	839	

Tema 3 - Algortimos Voraces

- Prim: empieza desde un nodo, busca las aristas de menor coste en los nodos abiertos, sin hacer un bucle.

❑ **Ejemplo:** (el mismo que para Kruskal):



L	1	2	3	4	5	6	7
1	∞	1	∞	4	6	∞	∞
2	1	∞	2	∞	4	∞	∞
3	∞	2	∞	∞	5	6	∞
4	4	∞	∞	∞	3	6	4
5	6	4	5	3	∞	6	∞
6	∞	∞	6	6	6	∞	3
7	∞	∞	∞	4	∞	3	∞

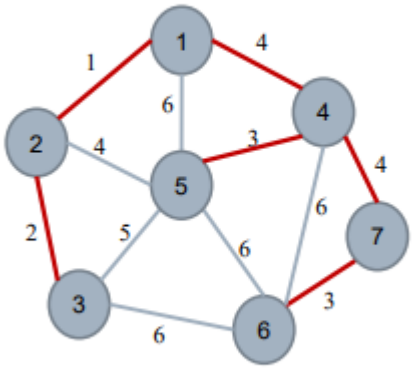
paso	selección	B
inicial	-	1
1	(1,2)	1,2
2	(2,3)	1,2,3
3	(1,4)	1,2,3,4
4	(4,5)	1,2,3,4,5
5	(4,7)	1,2,3,4,5,7
6	(7,6)	1,2,3,4,5,6,7=N

Solución T = {(1,2),(2,3),(1,4),(4,5),(4,7),(7,6)}

❑ Observación: No se producen rechazos.
❑ Teorema: Prim calcula el árbol expandido mínimo.
- Demostración por inducción sobre |T|.

- Kruskal: busca las aristas de menor coste de todo el grafo, sin hacer un bucle.

arista	(1,2)	(2,3)	(4,5)	(6,7)	(1,4)	(2,5)	(4,7)	(3,5) ...
peso	1	2	3	3	4	4	4	5

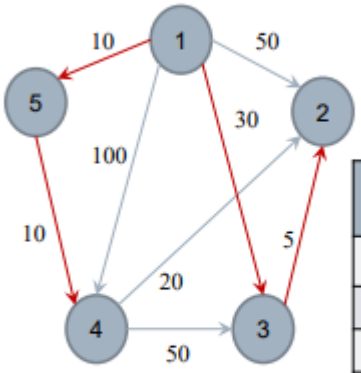


paso	selección	componentes conexas(conjuntos)						
inicial	-	1	2	3	4	5	6	7
1	(1,2)	1,2		3	4	5	6	7
2	(2,3)	1,2,3			4	5	6	7
3	(4,5)	1,2,3			4,5		6	7
4	(6,7)	1,2,3			4,5		6,7	
5	(1,4)	1,2,3,4,5					6,7	
6	(2,5) (ciclo-> rechazada)	1,2,3,4,5					6,7	
7	(4,7)	1,2,3,4,5,6,7						

Solución T = { (1,2), (2,3), (4,5), (6,7), (1,4) ,(4,7) }

- Dijkstra: empieza desde un nodo y va abriendo los nodos con menor coste para llegar y reevalúa los caminos a todos los nodos con ese nodo y sus aristas.

3. Algoritmo de Dijkstra. Ejemplo.



L	1	2	3	4	5
1	∞	50	30	100	10
2	∞	∞	∞	∞	∞
3	∞	5	∞	∞	∞
4	∞	20	50	∞	∞
5	∞	∞	∞	10	∞

paso	V	C	D (vector distancias)			
			2	3	4	5
inicial	-	{2,3,4,5}	50	30	100	**10**
1	5	{2,3,4}	50	30	**20**	10
2	4	{2,3}	40	**30**	20	10
3	3	{2}	**35**	30	20	10

Tabla: Evolución del conjunto C y de los caminos mínimos.

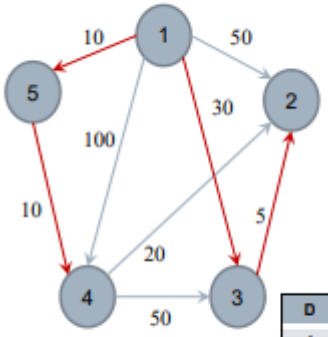
➢ **Observación**: 3 iteraciones = n – 2

```
D[w] :=min(D[w], D[v]+L[v,w]) /*si D[w]> D[v]+L[v,w] ⇒ D[w]:=D[v]+L[v,w] */
```

Solución D = {35,30,20,10} ≡ (distancia desde el nodo 1 a cada nodo del grafo)

- Floyd: Rellena tabla de coste de camino y antecesores. Va mirando cada fila y columna de cada nodo, va tomando los elementos de la fila y los suma a los de la columna, en caso de ser menor, lo sustituye.

4. Algoritmo de **Floyd**. Ejemplo.



1. Formar las **matrices iniciales *D* y *C***.
   - Inicialización de la matriz ***D***.
     - **D[i , j]**= valor que representa el **coste** de ir desde el nodo ***i*** al nodo ***j***
     - inicialmente en caso de no existir un arco entre ambos, el valor **D[i , j]**= ∞ caso de no existir un arco entre **el nodo *i* y el nodo *j***
   - Inicialización de la matriz ***C***.
     - **C[i , j]**= valor que representará el **nodo predecesor** a ***j*** en el camino mínimo desde ***i*** hasta ***j***.
     - Inicialmente se comienza con caminos de **longitud *1***, por lo que **C[i , j]**= **i** .

D	1	2	3	4	5
1	-	50	30	100	10
2	∞	-	∞	∞	∞
3	∞	5	-	∞	∞
4	∞	20	50	-	∞
5	∞	∞	∞	10	-

C	1	2	3	4	5
1	-	1	1	1	1
2	∞	-	∞	∞	∞
3	∞	3	-	∞	∞
4	∞	4	4	-	∞
5	∞	∞	∞	5	-

Tablas: Inicialización de las matrices de costes D y de los caminos mínimos C.

3. Tomamos ***k=1***:

D	1	2	3	4	5	C	1	2	3	4	5
1	-	50	30	100	10	1	-	1	1	1	1
2	∞	-	∞	∞	∞	2	∞	-	∞	∞	∞
3	∞	5	-	∞	∞	3	∞	3	-	∞	∞
4	∞	20	50	-	∞	4	∞	4	4	-	∞
5	∞	∞	∞	10	-	5	∞	∞	∞	5	-

Tablas: Evolución del conjunto D y de los caminos mínimos C

3.2. Tomamos *k=2*:

D	1	2	3	4	5	C	1	2	3	4	5
1	-	50	30	100	10	1	-	1	1	1	1
2	∞	-	∞	∞	∞	2	∞	-	∞	∞	∞
3	∞	5	-	∞	∞	3	∞	3	-	∞	∞
4	∞	20	50	-	∞	4	∞	4	4	-	∞
5	∞	∞	∞	10	-	5	∞	∞	∞	5	-

Tablas: Evolución del conjunto D y de los caminos mínimos C

3.3. Tomamos *k=3*:

D	1	2	3	4	5	C	1	2	3	4	5
1	-	35	30	100	10	1	-	3	1	1	1
2	∞	-	∞	∞	∞	2	∞	-	∞	∞	∞
3	∞	5	-	∞	∞	3	∞	3	-	∞	∞
4	∞	20	50	-	∞	4	∞	4	4	-	∞
5	∞	∞	∞	10	-	5	∞	∞	∞	5	-

Tablas: Evolución del conjunto D y de los caminos mínimos C

3.4. Tomamos *k=4*:

D	1	2	3	4	5	C	1	2	3	4	5
1	-	35	30	100	10	1	-	3	1	1	1
2	∞	-	∞	∞	∞	2	∞	-	∞	∞	∞
3	∞	5	-	∞	∞	3	∞	3	-	∞	∞
4	∞	20	50	-	∞	4	∞	4	4	-	∞
5	∞	30	60	10	-	5	∞	4	4	5	-

3.5. Tomamos *k=5*

D	1	2	3	4	5	C	1	2	3	4	5
1	-	35	30	20	10	1	-	3	1	5	1
2	∞	-	∞	∞	∞	2	∞	-	∞	∞	∞
3	∞	5	-	∞	∞	3	∞	3	-	∞	∞
4	∞	20	50	-	∞	4	∞	4	4	-	∞
5	∞	30	60	10	-	5	∞	4	4	5	-

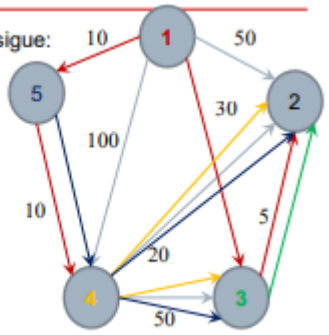
Tablas: Evolución del conjunto D y de los caminos mínimos C

4. k = n ⇒ **FIN** del proceso, las matrices quedan como sigue:

D	1	2	3	4	5
1	-	35	30	20	10
2	∞	-	∞	∞	∞
3	∞	5	-	∞	∞
4	∞	20	50	-	∞
5	∞	30	60	10	-

C	1	2	3	4	5
1	-	3	1	5	1
2	∞	-	∞	∞	∞
3	∞	3	-	∞	∞
4	∞	4	4	-	∞
5	∞	4	4	5	-

Tablas: Finales del conjunto D y de los caminos mínimos C



Programación dinámica

- Mochila: IMPORTANTE LA FÓRMULA!!!

Paso 3. **Ejemplo. n**= 3, **M**= 6, **pe**= (2, 3, 4), **be**= (1, 2, 5)

$(B[p, m]:= \max (B[p-1, m], B[p-1, m-pe_p]+ be_p))$

m

p \ B	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1
2	0	0	1	2	2	3	3
3	0	0	1	2	5	5	**6**

- Monedas: IMPORTANTE LA FÓRMULA!!!

- **Ejemplo. n**= 3, **C**= 8, **v**= (1, 4, 6)

$D[i,c] = \min(D[i-1, c], 1 + D[i, c-v_i])$ $+\infty$ Si col<0 ó fila≤0

D	0	1	2	3	4	5	6	7	8
0 c_0=0	0	+∞	+∞	+∞	+∞	+∞	+∞	+∞	+∞
1 c_1=1	0	1	2	3	4	5	6	7	8
2 c_2=4	0	1	2	3	1	2	3	4	2
3 c_1=6	0	1	2	3	1	2	1	2	**2**

- Subsecuencia: si es distinto coge el mayor de arriba o izquierda, si es igual coge la diagonal y suma 1.

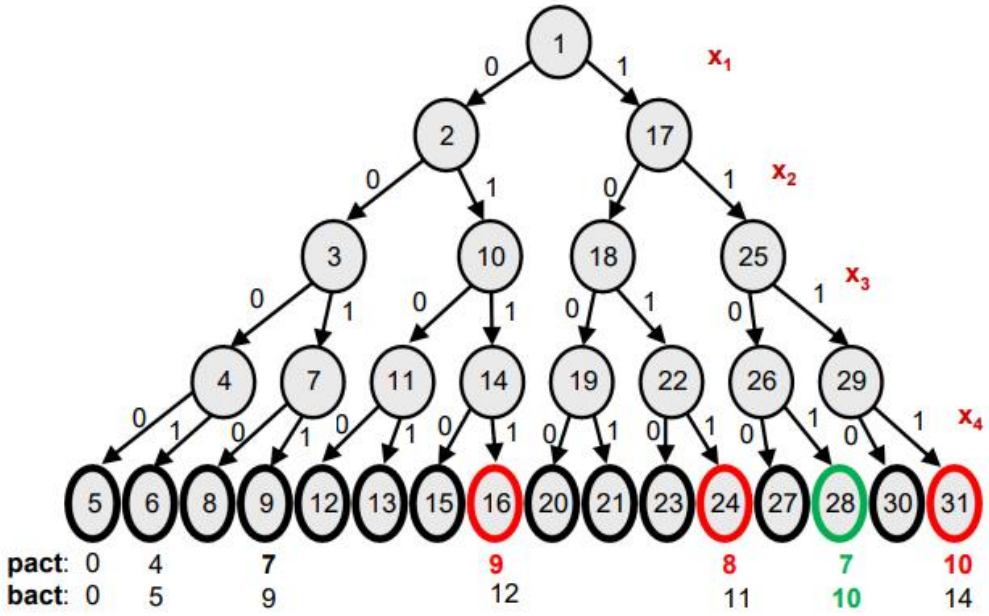
Ejemplo:

```
X = (A B C B),
Y = (B D C A B),

LCS = (B C B)

X = ( A B   C   B )
Y = (   B D C A B )
```

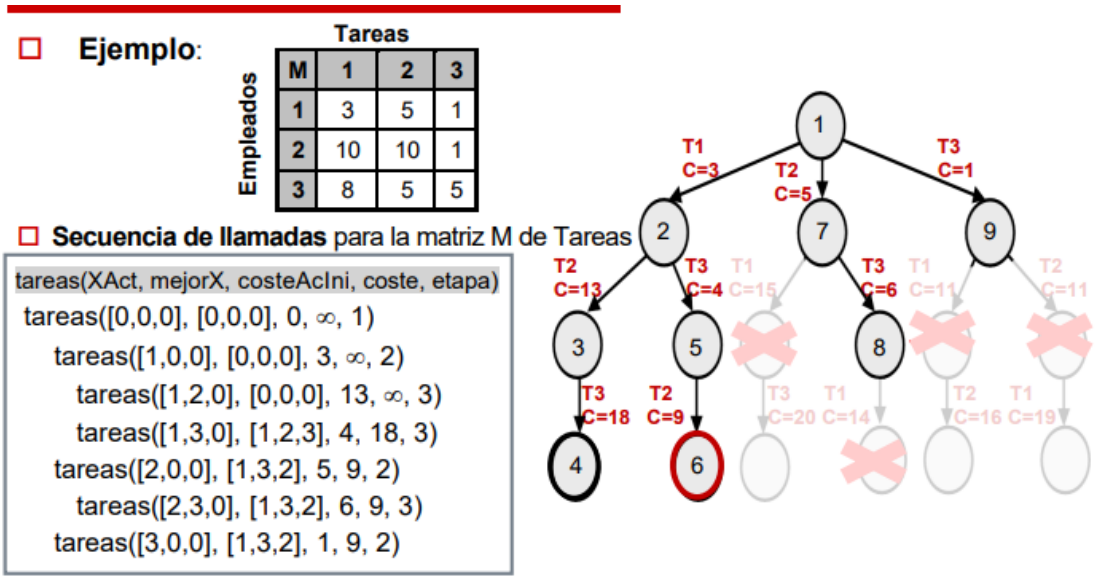
		Y_j	B	D	C	A	B
		0	1	2	3	4	5
X_i	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1
B	2	0	1	1	1	1	2
C	3	0	1	1	2	2	2
B	4	0	1	1	2	2	3

Backtracking

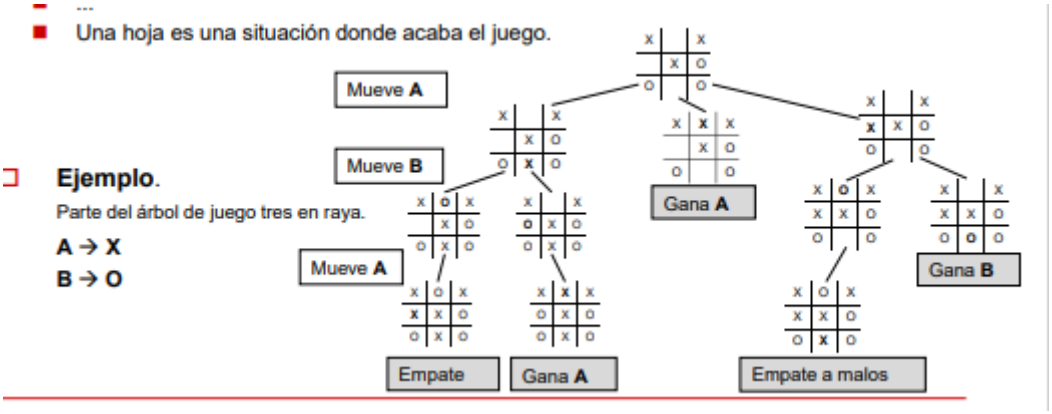
- Mochila: tomamos (1) o no (0) el objeto en orden del array (x1 = obj1, etc)
  - **Ejemplo:** n = 4; M = 7; b = (2, 3, 4, 5); p = (1, 2, 3, 4)



- Asignación de tareas: va comprobando cada tarea para cada empleado, si el coste es mayor poda.



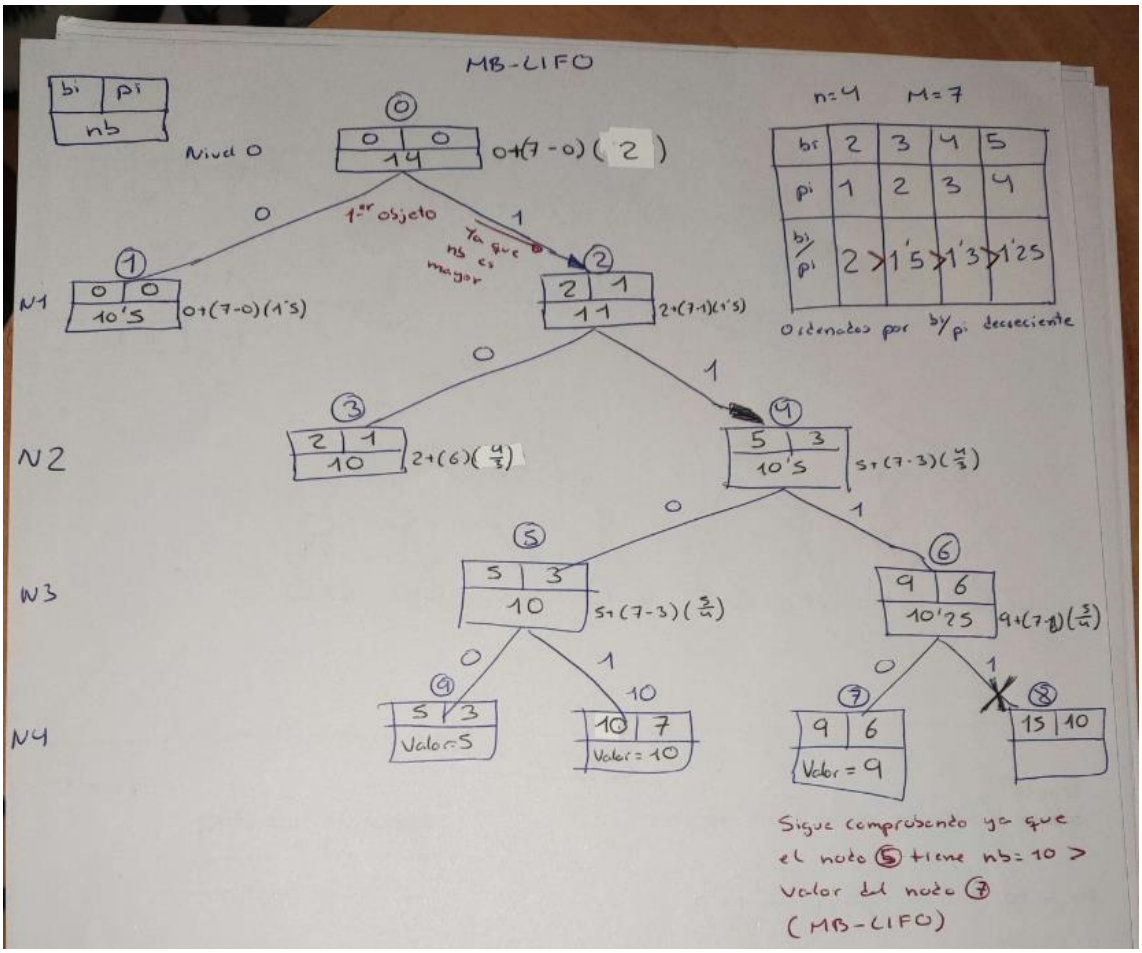
- Resolución de juegos: los nodos son distintas opciones de poner una ficha o lo que sea.



Ramifica y Poda

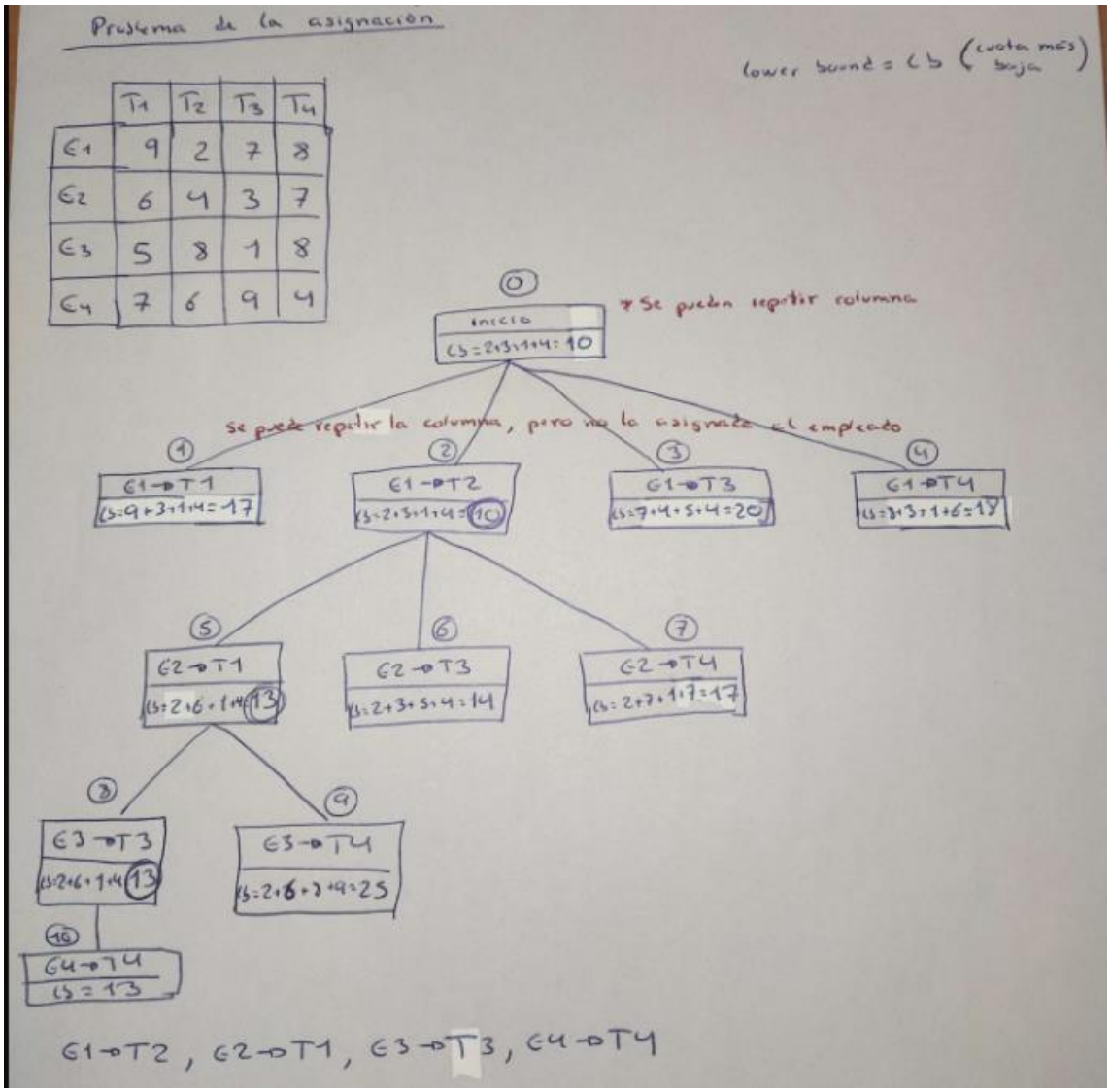
UB -> beneficio acumulado + (Peso Mochila – peso acumulado) * (bi/pi [del siguiente objeto]) UB=beneficio estimado

- Mochila:



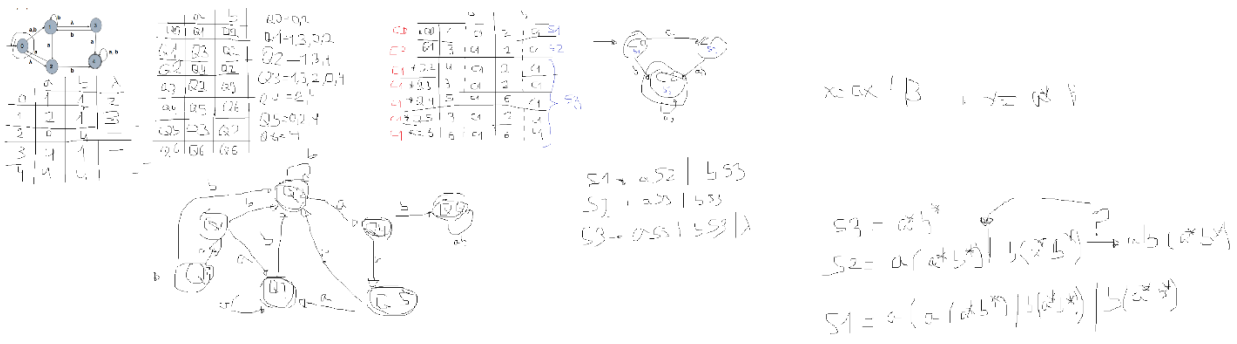
- Asignación de tareas:

LB = tomamos los valores de menor coste para cada trabajador, cuando asignemos ya un trabajador a alguna tarea, no podremos tomar el coste de esa tarea para otro trabajador. LB=cuota más baja



Automatás

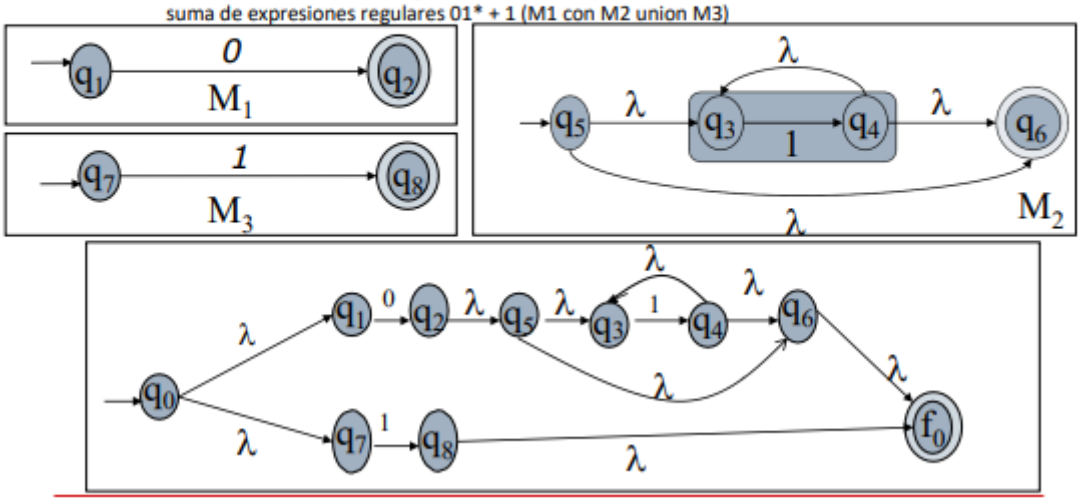
- Pasar de AFND a AFD y reducir este último.
- Expresión regular.



- De expresión regular a AFND-lambda

Ejemplo: AF construido para la expresión regular 01* + 1:

- M_1 representa el autómata para la expresión regular 0
- M_2 representa el autómata para la expresión regular 1*
- M_3 representa el autómata para la expresión regular 1
- En el Autómata final se integran los autómatas para la concatenación (0 con 1*, M1 con M2) y la suma de expresiones regulares 01* + 1 (M1 con M2 union M3)



(Ejercicio ER)