Algoritmos y Programación II, cátedra Calvo.

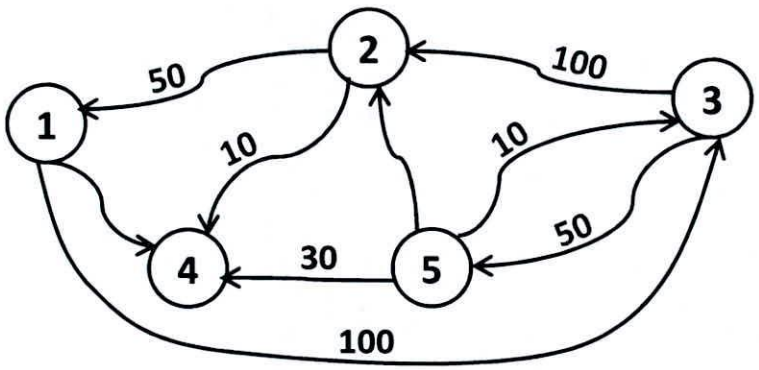
Final tomado en 30-6-15

1. Definir ABB, AVL, Trie, heap. Definir costos del alta en todos. Ubicar la secuencia en todos ellos (el ABB desbalanceado.

   1 12 13 23 30 232 123 223 22

2. Caracterizar estrategia Greedy. Cuándo se puede aplicar. Cuál de las partes de la estrategia influye en el coste temporal

3. Para el grafo, hallar los caminos de mínimo coste de 1 a los otros.
   Detallar algoritmos e indicar la estrategia.



4. Defina TDA Conjunto. Dar métodos, con pre y post condiciones. Proponer dos estructuras para su implementación. Analizar coste de alta, baja y búsqueda en esas dos estructuras

5. Qué es hashing, una buena función hash y para qué se usa. Definir hash perfecto. Explicar un método de solución de colisiones.

Para aprobar es necesario tener correctos y completos el 60% de cada ítem propuesto.
En la nota se ponderan, también, los resultados de los parciales y trabajos prácticos.
Duración del examen: 2 horas.

HOJA N°

FECHA 30-6-15

Calvo - Algo2

① Definir ABB, AVL, trie, heap. Definir costos del alta en todos. Crear la secuencia en todos ellos (el abb desbalanceado)

1 12 13 23 30 232 123 223 22

ABB: Estructura de datos que o bien es nulo o consta de un nodo y 2 subárboles hijos
Están ordenados (Árbol Binario de Búsqueda)
La raíz no tiene padre. Si un nodo tiene clave x entonces: los datos en el subárbol hijo izquierdo son menores que x
" " " " " derecho son mayores q' x
Costo de alta
si está balanceado $O(\log(n))$
si no está balanceado (degeneró en lista) $\rightarrow O(n)$

AVL: Es un árbol binario que se encuentra balanceado. La altura de sus subárboles izq y derecho no difiere en más que d (en general, d=1 o d=2)
Son árboles ordenados
Costo de alta: $O(\log(n))$

~~trie~~ AB: (Un árbol B de m vías que cumple con
① la raíz tiene, como mínimo, 2 subárboles, a menos que sea hoja → raíz es hoja o tiene, al menos, 2 hijos
② cada hoja contiene k-1 claves ($m/2 \leq k \leq m$)
③ todas las hojas están al mismo nivel
④ si un nodo no hoja tiene k claves → tiene k+1 hijos

m = # enlaces

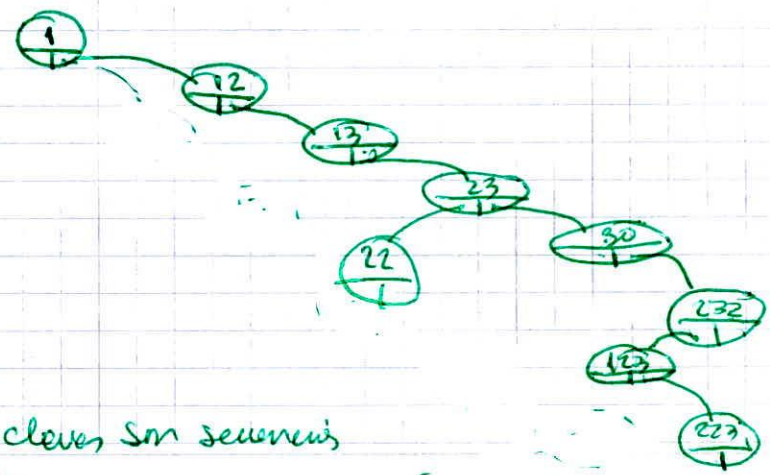
Pd Costo del alta: $O(\log(n))$ (?)

Heap: tipo particular de A.B. que tiene: → completo o con espacios parciales ordenados
① el valor de cada nodo es mayor (o menor, según el caso) que sus hijos
② el árbol está balanceado y las hojas del último nivel están en el extremo izq.

costo alta: $O(\log(n))$ (peores)

Secuencia

ABB

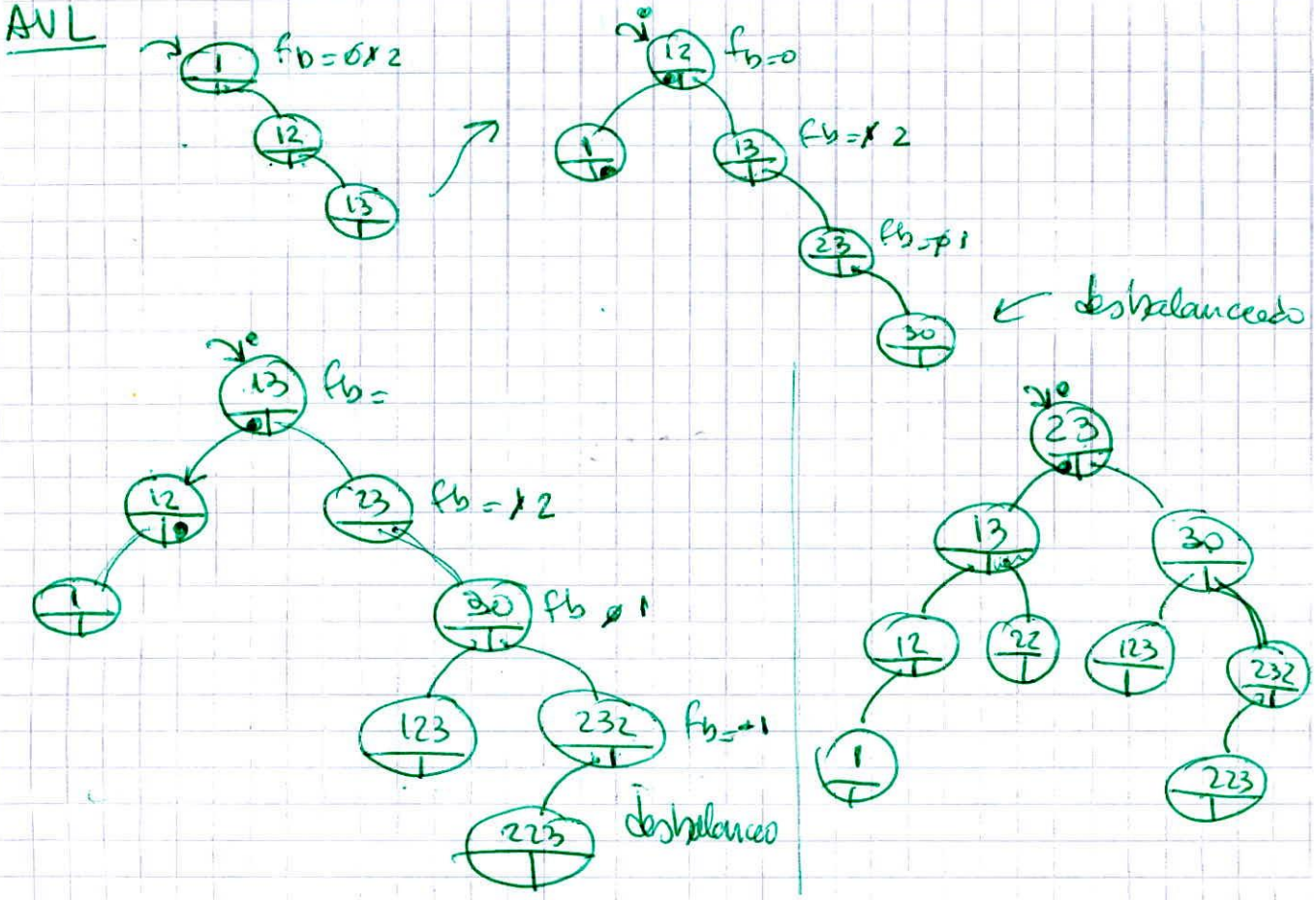


TRIE: Árbol digital. Las claves son secuencias de ~~números~~ símbolos

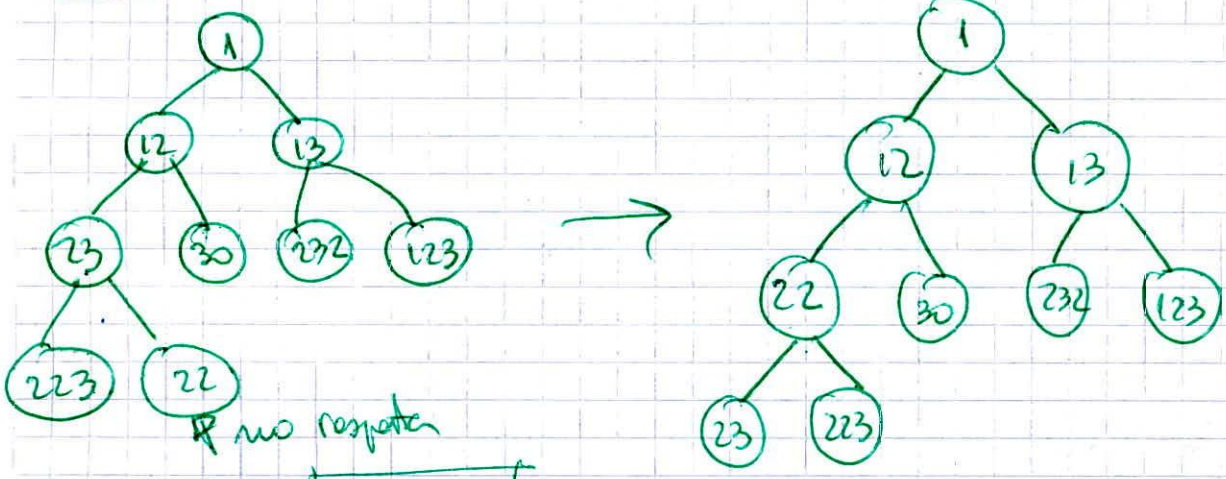
NOTA

1 12 13 23 30 232 123 223 22

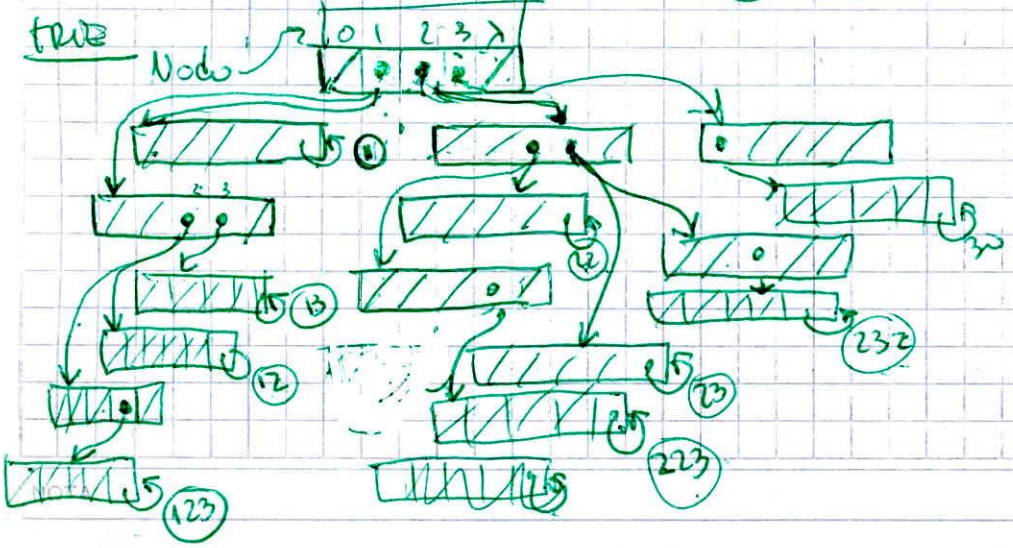
AVL



Heap - de mínimo



true



Paradigma de programación que busco dividir un problema en varias partes y encontrar la sol óptima utilizando los ~~óptimos~~ óptimos locales con la esperanza de hallar un óptimo global

② Caracterizar estrategia Greedy. Cuándo se puede aplicar. Cuál de las partes de la estrategia influye en el coste temporal

En la estrategia Greedy la idea es seleccionar el máximo (o mínimo) elemento ~~de~~ Se suele obtener con árboles heap máximo (o mínimo) ya que en dicha raíz se dotione ese valor

Se tiene en cuenta:

- conjunto de candidatos (elementos seleccionables)
- ~~Sol~~ Solución parcial (candidatos seleccionados)
- Función de selección (determina el mejor candidato del conj de cand. selecc.)
- Función de factibilidad
- Criterio que define lo que es una solución (indica si la sol. parcial ~~es~~ resuelve el problema)
- Función objetivo (valor de la sol. alcanzada)

```
Greedy (conj. candidatos C) : solución
S = ∅
mientras (S no sea solución y C ≠ ∅) {
        X = selección C
        C = C - {X}
        si (S ∪ {X} es factible)
                S = S ∪ {x}
        }
if S es solución
    return S.
else  no se encontró solución
```

Función de selección influye en el coste temporal

NOTA

③ Para el grafo, hallar los caminos de mínimo costo de 1 a los otros
Detallar algoritmo e analizar la estrategia

Dijkstra : // Grafo con N vértices numerados a partir de 1
// Matriz de pesos y vectores visitados
// D es vector costos actuales estrategia Greedy

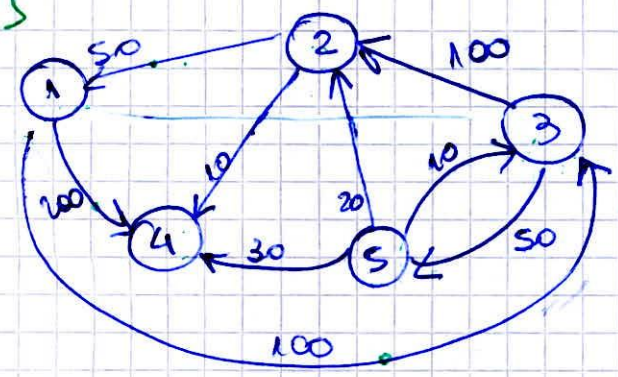
Ciclo de N-1 iteraciones

{ elegir un vértice V no visitado y de costo mínimo y marcarlo en vector de visitados.

Para cada vértice u adyacente a v:

$$D[u] = \min(D[u], D[v] + M[v][u])$$

}



M

(v) \ (u)	1	2	3	4	5
1	0	∞	100	200	∞
2	50	0	∞	10	∞
3	∞	100	0	∞	50
4	∞	∞	∞	0	∞
5	∞	20	10	∞	0

D

2	3	4	5	
∞	100	200	∞	→ v = 3
200	100	200	150	→ v = 5
170	100	180	150	→ v = 2
170	100	180	150	→ v = 4

v=2	v=3	v=4	v=5
2	3	4	5

u=1 u=5 u=3 u=3
u=4 u=2 u=2 u=4

Los caminos mínimos desde el vértice 1 son:

hasta	2	3	4	5
#	170	100	180	150

NOTA

HOJA N°

FECHA

(4) Definir TDA conjunto. Dar métodos, con pre y post condiciones. Proponer dos estructuras para su implementación. Analizar costo del alta, baja y búsqueda en esas dos estructuras.

Un conjunto es una colección de elementos miembros del conjunto, que no se repiten

Métodos del TDA conjunto:

- agregar elemento
- remover elemento
- ¿está vacío?
- ¿está el elemento?
- unión de conjuntos
- inters. de conj.
- diferencia de conjuntos

Estructuras de implementación: → Vectores de bits.
→ listas
→ árboles
→ tablas de hash

listas: alta O(n) tiene que buscar para ver si está
baja O(n)
búsqueda O(n)

árboles: alta: O(log(n)) si está balanceado, O(n) si no lo está y degeneró en lista
baja: O log(n)
búsqueda:

NOTA

(5) Qué es hashing, una buena función hash y para qué se usa. Definir hash perfecto. Explicar un método de solución de colisiones

Hashing: es una alternativa a los algoritmos de búsqueda basados en comparación entre claves. Se basan en una transformación de la clave.

Una buena función hash es aquella en la que ~~se~~ se maximiza el grado de unicidad. No desperdicia espacios de memoria y ~~casi~~ no genera (o casi no genera) colisiones.

Se utiliza para optimizar la búsqueda.

Hash perfecto: es aquella función en la que, al aplicarla a una clave, nunca genera colisiones

Colisión: se da cuando $h(x_1) = h(x_2)$ donde x_1 y x_2 son claves y h es la función hash

Método de solución de colisiones: ~~x~~ direcc. directo → rehasheo (aplicar otra función cuando se genera una colisión → doble hashing)

NOTA