Data Structures (Estructuras de Datos) (ED):

Types of D.S.:

- Linear (Lists, stacks, queues)
- Network (Graphs)
- Hierarchial (Trees)
- Dictionaries (HashTables)

Graphs: $G = \{V, E, W\}$

- Directed: 1 → 2
- Undirected: 1 — 2

V → Vertex/Node: (V_n)

E → Edge: (V_n, V_m)

W → Weight: (X)

Degree of a Node = In D + Out D.

Algorithms: For graphs

- Dijkstra: Given a node A, min cost from A to any other node. $O(n^2)$

  - V_D: Min Cost Vector
  - V_p: Min Cost Path Vector
  - S: Already Passed Nodes

  $S = \{A\}$

  V_D = Edges from A weights

  $V_P = (-)$ For all

  look in V_D for → lowest weight that is not on S

  $S = \{A, X\}$

  Check in V_D if an edge from $X \to y$ + $A \to X$ is lower than $A \to y$. If so, update $A \to y$

  Check every edge of X

  repeat until $S = V$

- Floyd-Warshall: Is Dijkstra For every starting node $O(n^3)$

  - A: Min Cost Matrix
  - P: Min cost Path Matrix

From \ To	0	1	2	...
0	0	x	y	
1	z	0	...	
2	w	..	0	
:				

From X to X → weight 0

A = Weights, ∞ otherwise

P = (-) For all

start with 0 → continue 1, 2 ...

Nodes	0	1	2	...
0	0	1	5	
1	2	0	∞ → 7	
2	1	3	0	
:				

You start 0→1→1 : if < then 1 + 0 = 1 ≯ 1 Change

1→0→2 : 2 + 5 = 7 < ∞ → Change (1→2)

2→0→1 : 1 + 1 = 2 < 3 → Change (2→1) and update P

...

Finished Floyd:

- Eccentricity of a node = max of the column
- Center of the graph = node with min eccentricity

For 2 is 7
For 1 is 3
For 0 is 2 lowest

∅

- DFPrint (Depth-First): From node A checks reachability to any node

  - N: Nodes Vector
  - E: Edges Matrix

  If a node is reachable T in N

  From A take and Edge. to X → Mark X as T → Repeat with X

  take another Edge ...

  until you don't have more

  A in N is always T

- Prim: Minimum spanning tree (without cycles)

  - T: Edges of minimum spanning tree
  - U: Used nodes
  - E: Edges matrix
  - V: All nodes

  T: Empty

  U: {X} ← random node any

  while V ≠ U → Check lowest edge

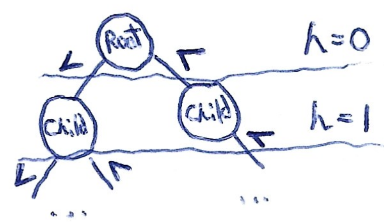
  U → (V-U)

  {U, X}

  T += {U, X}

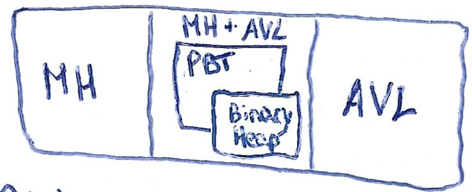
  U += X

Trees: Graphs without cycles.



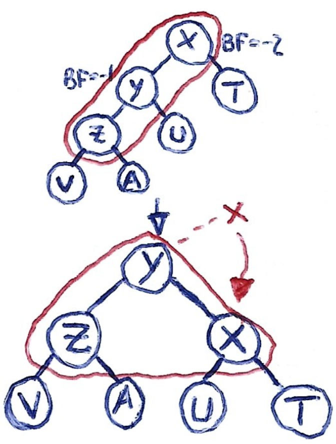
For removing a node:

- If it has no children: Remove it directly
- If 1 childrem: Replace by children
- If 2 c.: Replace by maximum children of left child branch

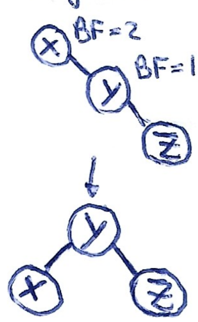


Rotations in AVL:

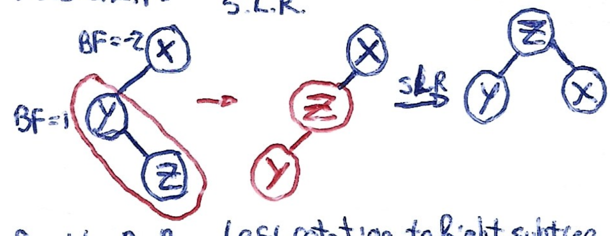
- S.L.R:



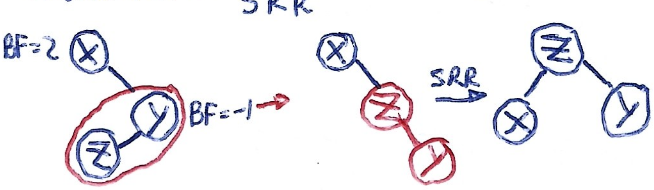
- S.Right.R



- Double.L.R: Right R. to left subtree
  S.L.R.



- Double R.R.: Left rotation to Right subtree
  SRR

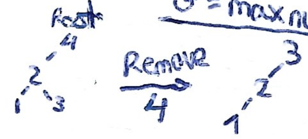


Traversing:

- pre order: Root - Left - Right
- in order: Left - Root - Right
- post order: Left - Right - Root

Types of trees:

- BST (Binary Search): $G=2$



G = max number of branches

- PBT (Perfect Balanced): $\left|\#_{izq\ nodes} - \#_{der\ nodes}\right| \leq 1$
  For every node in the tree
- AVL (Weakly Balanced): $\left|h_{izq} - h_{der}\right| \leq 1$
  BF (Balance Factor)
- MH (Minimum Height): No tiene que haber huecos en alturas anteriores para empezar una nueva
- Fibonacci Trees: Worst AVL trees $O(1'44 \log_2 n)$
- B-n Trees: n = max number of elements per page
  - Root Page elements: $1 \leq m \leq 2n$
  - Other Page elements: $n \leq m \leq 2n$

  h=1
  h+1

  m = elements in a page
  m+1 = links

  $$\log_{2n+1} N < h < \log_{n+1} N$$

  N = # nodes in the B-n tree

B-n min capacity:

level = h	n/h	min m/node	min links/node	Total elements/h
1	1	1	1+1=2	1
2	2	n	n+1	2n
3	2(n+1)	n	n+1	2n·(n+1)
4	2(n+1)^2	⋮	⋮	2n·(n+1)^2
				(...)^3

links of previous h × n/h of previous h ; +1

$n/h \cdot \min m / h$; + → N_{min}

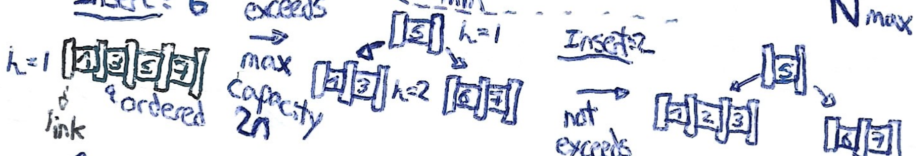
B-n max capacity:

h	n/h	max m/...	max links/...	Total el. /h
1	1	2n	2n+1	2n
2	2n+1	2n	2n+1	2n·(2n+1)
3	(2n+1)^2	2n	2n+1	2n·(2n+1)^2
4	(2n+1)^3	⋮	⋮	2n (...)^3

+1

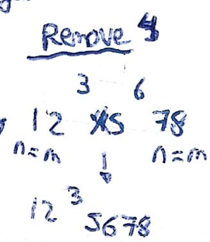
$n/h \cdot \max m / h$; + → N_{max}

Insert: 6



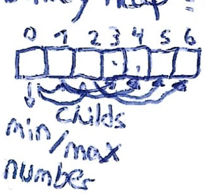
Remove:

- Leaf: $n < m$ → Remove directly
  $n = m$ → Look right brother ($n < m$?)
  → Look left brother ($n < m$?)
  → Else merge right
  Changes the mother.
  Removes the mother



- Not Leaf: Substitute by succesor if ($n < m$) of the succesor page
  Substitute by predecesor if ($n < m$) of ... pred page.
  Else substitute by succesor and repeat on the child page.

- Binary Heap: Filled from left to right (can be stored in an array)



Left child = $2p + 1$
Right child = $2p + 2$
Parent Node = $\left\lfloor \frac{child - 1}{2} \right\rfloor$ (integer division)

Estructuras de Datos (ED):

Search Paths (Average lenght):

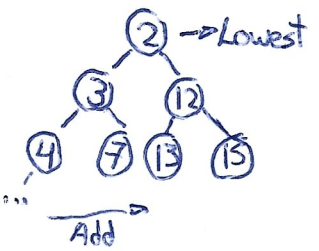
- Found Node (Internal Path): $IP = \frac{\sum_{i=1}^{h} i \cdot n_i}{N} = \frac{\sum \text{level} \cdot \text{nodes in level}}{\text{number of nodes}}$ (N = # nodes)
- Not Found Node (External Path): $EP = \sum_{i=2}^{h+1} \frac{i \cdot sn_i}{sn}$ (sn = # sin nodos)

$sn_i + n_i = G$

- sn_i: sin nodo
- G: max number of branches per Father node

Minimum Heaps: Lowest in array [Ø]

A parent is lower than it's children



Pop: Quita el menor, en su lugar inserta el último elemento (15 en el heap anterior) y reordena intercambiando por el menor de los hijos

- HashSet <T> and HashMap <K,V>
  - HashSet → contains
  - HashMap → get

Hashing: HashCode()

F(...) [-∞, +∞]

↓ |Abs|

[0, +∞)

↓ % B → Prime

[0, B-1]

Load Factor: $LF = \frac{n}{B}$

- Separate Chaining (OpenHashTables): In each cell, we have a dynamic data structure
- Open Addresing (ClosedHashTables): Stores in surroundings if collision
  - ✗ Linear: In the next, and next ... $[x + attempts] \% B$
  - ✗ Quadratic: $[x + att^2] \% B$ → LF < 0'5 and B prime for working ok
  - ✗ Double Hashing: $[x + att \cdot (R - x \% R)] \% B$ ($R - x \% R = \theta(x)$)
    - R is previous prime to B

☐ Dynamic Resize:
- [Add] B → Next prime from 2B+1 (5 → 11), (7 → 17)
  - Recalculate R
  - Recalculate F(x) for every non deleted element
- [Remove] Previous prime B

IF LF is low:
- Separate Chaining LF ≤ 0'33
- Open Addresing LF ≤ 0'16