**Apuntes Programación 3**

**Árboles Binarios:**

* ***Grado:*** de ni es el número de hijos del nodo ni*.*
* ***Altura:*** de ni es la longitud del camino más largo desde ni hasta una hoja. (Las hojas tienen altura cero) (La altura de un árbol es la altura del nodo raíz)
* ***Ancestro/Descendiente***: si existe un camino desde n1 a n2, se dice que n1 es ancestro de n2 y n2 es descendiente de n1.

**Árbol Binario lleno:** Dado un árbol binario T de altura **h**, diremos que T es lleno si cada nodo interno tiene grado 2 y todas las hojas están en el mismo nivel (h).

**Cantidad de nodos en un árbol binario lleno:** Sea T un árbol binario lleno de altura **h**, la cantidad de nodos N es: (2^(h+1)–1)

**Árbol Binario completo:** Dado un árbol binario T de altura **h**, diremos que T es completo si es lleno de altura h-1 y el nivel h se completa de izquierda a derecha.

**Cantidad de nodos en un árbol binario completo:** Sea T un árbol binario completo de altura **h**, la cantidad de nodos N varía entre: (2^h) y (2^(h+1)–1)

**Árboles Generales:**

* ***Grado:*** de ni es el número de hijos del nodo ni*.* (Grado del árbol es el grado del nodo con mayor grado)
* ***Altura:*** de ni es la longitud del camino más largo desde ni hasta una hoja. (Las hojas tienen altura **cero**) (La altura de un árbol es la altura del nodo raíz)
* ***Profundidad / Nivel***: de ni es la longitud del único camino desde la raíz hasta ni. (La raíz tiene profundidad o nivel **cero**)

**Árbol General lleno:** Dado un árbol T de grado **k** y altura **h**, diremos que T es lleno si cada nodo interno tiene grado k y todas las hojas están en el mismo nivel (h).

**Cantidad de nodos en un árbol binario lleno:** Sea T un árbol lleno de grado **k** y altura **h**, la cantidad de nodos N es: (k^(h+1) - 1) / (k-1)

**Árbol General completo:** Dado un árbol T de grado **k** y altura **h**, diremos que T es completo si es lleno de altura h-1 y el nivel h se completa de izquierda a derecha.

**Cantidad de nodos en un árbol completo:** Sea T un árbol completo de grado **k** y altura **h**, la cantidad de nodos N varía entre: (k^h + k-2) / (k-1) y (k^(h+1) - 1) / (k-1)

**Colas de Prioridades-Heap:**

Una **Cola de Prioridad** es una estructura de datos que permite al menos dos operaciones: Insert (Inserta un elemento en la estructura) y DeleteMin (Encuentra, recupera y elimina el elemento mínimo).

Una **Heap Binaria** es una implementación de colas de prioridad que no usa punteros y permite implementar ambas operaciones (Insert y DeleteMin) con O(log N) operaciones en el peor caso.

Cumple con dos propiedades:

**-Propiedad Estructural:** Una Heap es un árbol binario completo. Su altura es de O(log n).

**-Propiedad de Orden:**

* **Min Heap:** El elemento mínimo está almacenado en la raíz. El dato almacenado en cada nodo es menor o igual al de sus hijos.
* **Máx Heap:** Se usa la propiedad inversa.

**Algoritmo BuildHeap (Min Heap)**

1. Insertar los elementos desordenados en un árbol binario completo.
2. Filtrar hacia abajo cada uno de los elementos (Se filtran los nodos que tienen hijos, el resto son hojas):

- Se empieza filtrando el elemento de la posición (tamaño/2)

- Se elige el menor de los hijos y se lo compara con el padre (Filtrado).

**Algoritmo HeapSort (Ordenación de Vectores)**

**Orden Creciente:**

1. Construir una MáxHeap
2. Intercambiar el último elemento con el primero
3. Decrementar el tamaño de la Heap
4. Filtrar hacia abajo

**Orden Decreciente:**

1. Construir una MinHeap
2. Intercambiar el último elemento con el primero
3. Decrementar el tamaño de la Heap
4. Filtrar hacia abajo

**Análisis de algoritmos:**

A screenshot of a computer

Description automatically generated

***Notación Big-Oh (Definición):***

Decimos que

T(n) = O(f(n))

si existen constantes c > 0 y n0 tales que:

T(n) ≤ c f(n) para todo n ≥ n0

Se lee: T(n) es de orden de f(n)

f(n) representa una cota superior de T(n)

La tasa de crecimiento de T(n) es menor o igual que la de f(n)

***Reglas:***

Si T1(n)=O(f(n)) y T2(n)=O(g(n)), entonces:

* T1(n)+T2 (n)=max(O(f(n)),O(g(n))) 🡺 **(Regla de la Suma)**
* T1(n)\*T2 (n)=O(f(n)\*g(n)) 🡺 **(Regla del Producto)**
* T(n) es un polinomio de grado k ⇒ T(n) = O(n^k**) 🡺 (Regla para Polinomios)**
* T(n) = log^k (n) ⇒ O(n) para cualquier k n siempre crece más rápido que cualquier potencia de log(n) 🡺 **(Regla para logaritmos)**
* T(n) = cte ⇒ O(1) 🡺 **(Regla para Constantes)**
* T(n) = cte \* f(n) ⇒ T(n) = O(f(n)) 🡺 **(Regla para producto de función con cte)**