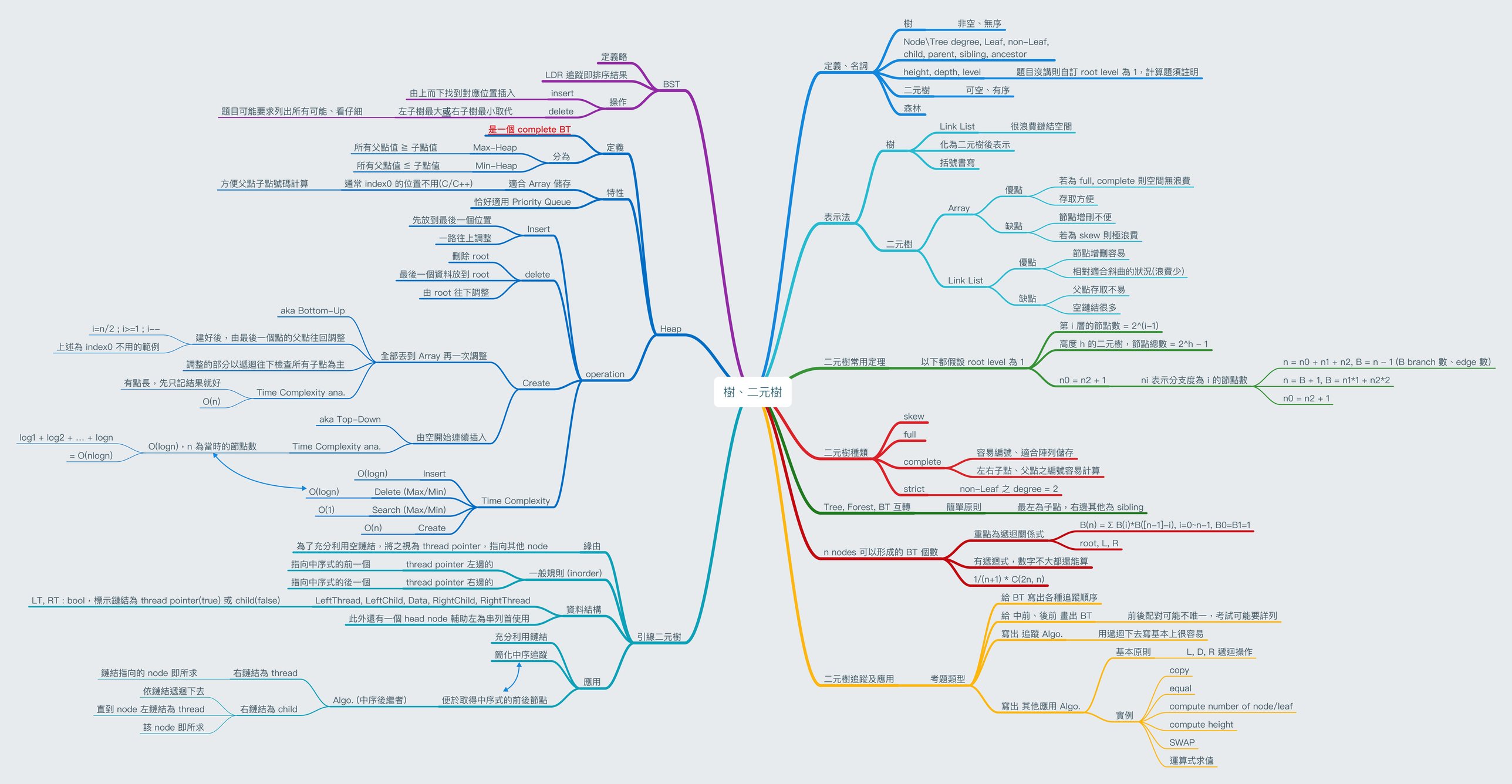
樹、二元樹
定義、名詞
樹 非空、無序
Node\Tree degree, Leaf, non-Leaf,
child, parent, sibling, ancestor
height, depth, level 題目沒講則自訂 root level 為 1，計算題須註明
二元樹 可空、有序
森林
表示法
樹
Link List 很浪費鏈結空間
化為二元樹後表示
括號書寫
二元樹
Array
優點
若為 full, complete 則空間無浪費
存取方便
缺點
節點增刪不便
若為 skew 則極浪費
Link List
優點
節點增刪容易
相對適合斜曲的狀況(浪費少)
缺點
父點存取不易
空鏈結很多
二元樹常用定理 以下都假設 root level 為 1
第 i 層的節點數 = 2^(i-1)
高度 h 的二元樹，節點總數 = 2^h - 1
n0 = n2 + 1 ni 表示分支度為 i 的節點數
n = n0 + n1 + n2, B = n - 1 (B branch 數、edge 數)
n = B + 1, B = n1*1 + n2*2
n0 = n2 + 1
二元樹種類
skew
full
complete
容易編號、適合陣列儲存
左右子點、父點之編號容易計算
strict non-Leaf 之 degree = 2
Tree, Forest, BT 互轉 簡單原則 最左為子點，右邊其他為 sibling
n nodes 可以形成的 BT 個數
重點為遞迴關係式
B(n) = Σ B(i)*B([n-1]-i), i=0~n-1, B0=B1=1
root, L, R
有遞迴式，數字不大都還能算
1/(n+1) * C(2n, n)
二元樹追蹤及應用
考題類型
給 BT 寫出各種追蹤順序
給 中前、後前 畫出 BT 前後配對可能不唯一，考試可能要詳列
寫出 追蹤 Algo. 用遞迴下去寫基本上很容易
寫出 其他應用 Algo.
基本原則 L, D, R 遞迴操作
實例
copy
equal
compute number of node/leaf
compute height
SWAP
運算式求值
引線二元樹
緣由
為了充分利用空鏈結，將之視為 thread pointer，指向其他 node
一般規則 (inorder)
指向中序式的前一個 thread pointer 左邊的
指向中序式的後一個 thread pointer 右邊的
資料結構
LT, RT : bool，標示鏈結為 thread pointer(true) 或 child(false)
LeftThread, LeftChild, Data, RightChild, RightThread
此外還有一個 head node 輔助左為串列首使用
應用
充分利用鏈結
簡化中序追蹤
便於取得中序式的前後節點
Algo. (中序後繼者)
右鏈結為 thread
鏈結指向的 node 即所求
右鏈結為 child
依鏈結遞迴下去
直到 node 左鏈結為 thread
該 node 即所求
Heap
BST
定義略
LDR 追蹤即排序結果
操作
insert
由上而下找到對應位置插入
delete
左子樹最大或右子樹最小取代
題目可能要求列出所有可能、看仔細
定義
是一個 complete BT
分為
Max-Heap
所有父點值 ≧ 子點值
Min-Heap
所有父點值 ≦ 子點值
特性
適合 Array 儲存
通常 index0 的位置不用(C/C++)
方便父點子點號碼計算
恰好適用 Priority Queue
operation
Insert
先放到最後一個位置
一路往上調整
delete
刪除 root
最後一個資料放到 root
由 root 往下調整
Create
全部丟到 Array 再一次調整
aka Bottom-Up
建好後，由最後一個點的父點往回調整
i=n/2 ; i>=1 ; i--
上述為 index0 不用的範例
調整的部分以遞迴往下檢查所有子點為主
Time Complexity ana.
有點長，先只記結果就好
O(n)
由空開始連續插入
aka Top-Down
Time Complexity ana.
O(logn)，n 為當時的節點數
log1 + log2 + ... + logn
= O(nlogn)
Time Complexity
O(logn) Insert
O(logn) Delete (Max/Min)
O(1) Search (Max/Min)
O(n) Create