Phill-DS-1213, 1220

Tree 的名詞與概念

Node 節點 → data field, 資料欄位

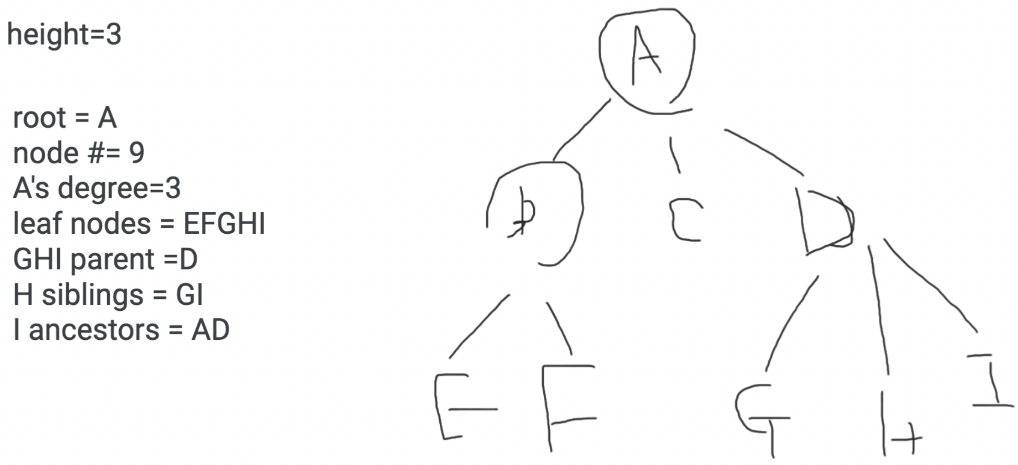
Link, edge, branch 分支 → 連接兩個節點的線

degree 分支度

- 某一個node : 多少個 subtrees → 跟多少個 nodes 相連 → $T_1, T_2, ..., T_N$ = N
- 一棵樹 degree → all nodes 最大 degree
- leaf node → degree =0
- non-leaf node → degree <> 0

hierarchical relationship

- child (children) → children of a node → roots of subtrees → X's children
- parent → children's parent → owns your subtree
- sibling → sibling nodes → parent node 相同
- ancestors → from root ~ X → path 包含的所有的 nodes → ancestors
- level : root level =1, children of root level=2, X level =N → X's children level =N+1
- level of tree → height, depth → all nodes , maximum level



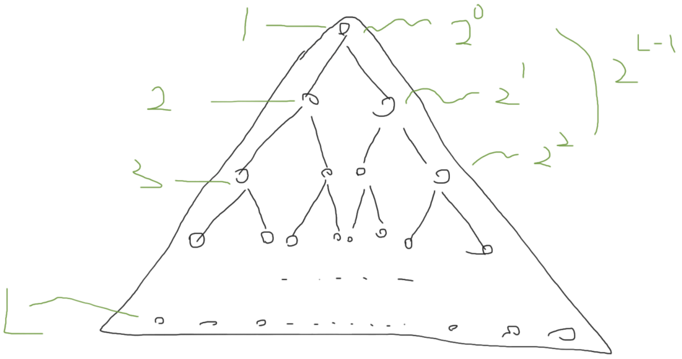
height=3

root = A
node #= 9
A's degree=3
leaf nodes = EFGHI
GHI parent =D
H siblings = GI
I ancestors = AD

x = y+z;

Nodes and edges 的計算

- 每一個 node 都一定有一個 edge → root 例外
- 任何一棵樹 → Node # N 跟 edge # B 有何關係 → N = B + 1
- 二元樹 binary tree
  - level-L 節點個數最多是多少? 2^{L-1}



  - depth K binary tree， nodes # = $2^K - 1$, K≥1

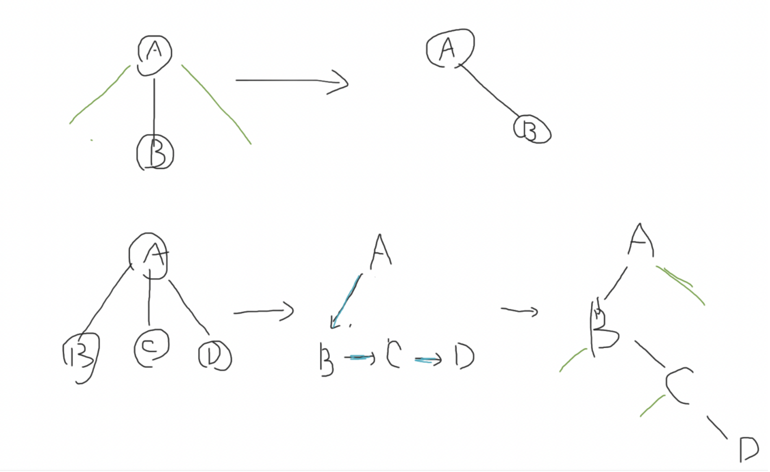
二元樹：skewed binary (傾斜樹), complete binary tree (perfect binary)

- skew binary tree → binary tree, left subtree or right subtree 是空集合
- full binary tree

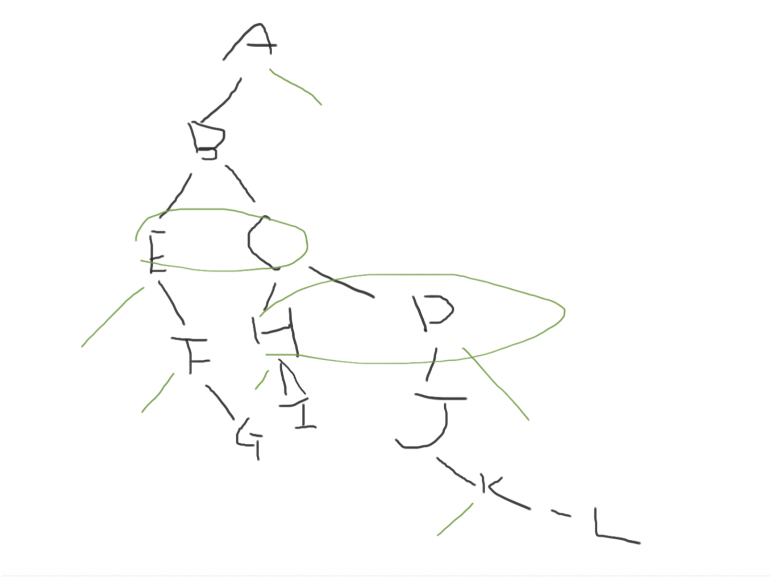
- ○ 節點最多的 binary tree
- ○ 高度 K → $2^K - 1$ nodes

各種 tree → binary tree

- 所有的 tree 都可轉成 unique binary tree
- 轉換原則

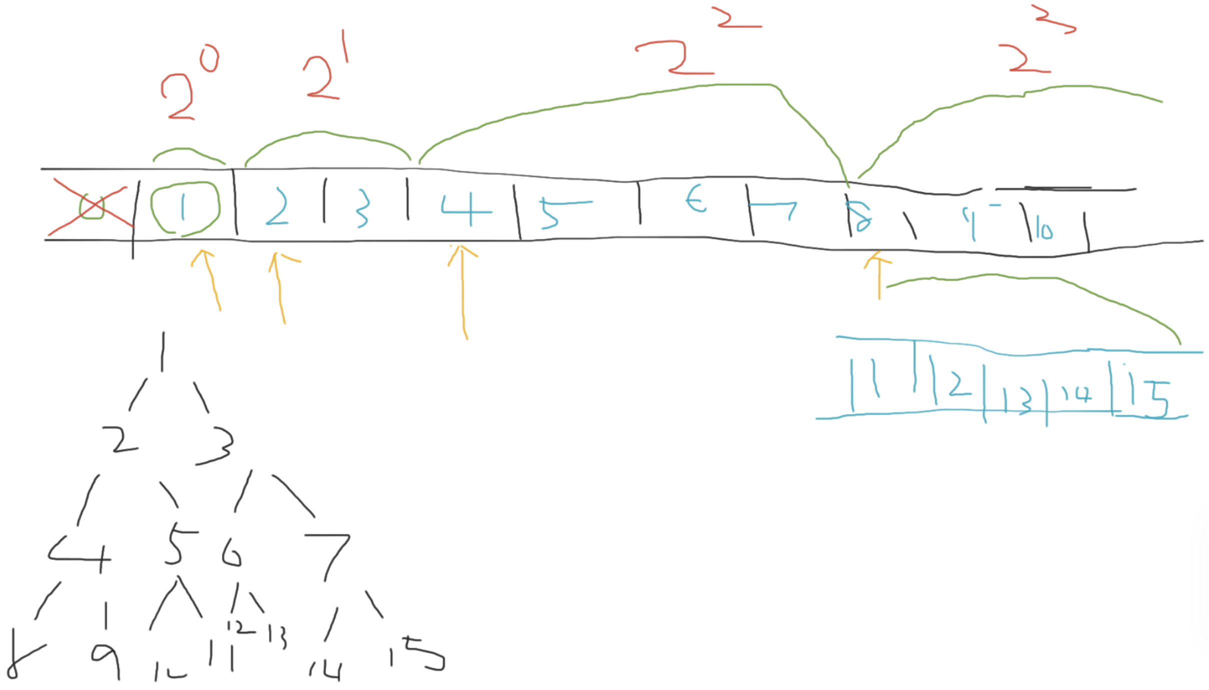


  - 鎖定 root
  - 連結 最 left 的 child node
  - 連結 sibling → siblings → end
  - 舊的 link 拿掉
  - 橫向的 link , 旋轉45度



怎麼實作二元樹 (陣列)

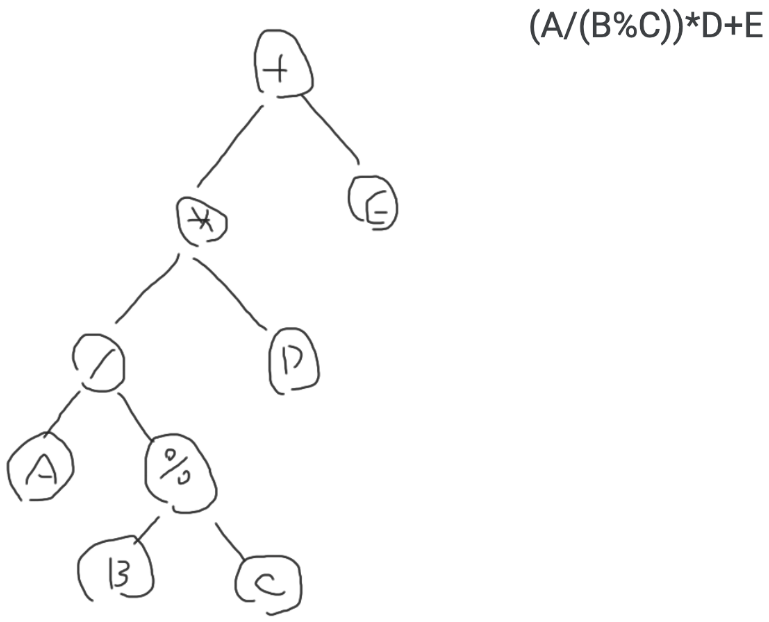
陣列的實作



- complete binary tree → efficient
- skewed binary tree → 浪費空間

Binary tree Traverse

- inorder traverse 中序 → 原則: 先找左子樹 自己 再找右子樹



- preorder 前序 → 原則: 先找自己 再找左子樹 再找右子樹
- postorder 後序 → 先找左子樹 再找右子樹 自己

實作

```
#include <iostream>
using namespace std;

struct TreeNode{
  int value;
  int leftChildIndex;
  int rightChildIndex;
```

```
};

class BinaryTree{
  private:
    TreeNode* nodes; // root ->index
    int capacity;
    int nextIndex; //即將要插入的 index

  public:
    BinaryTree(int capacity) : capacity(capacity), nextIndex(1)
      nodes = new TreeNode[capacity+1];
    }

    void insert(int value){
      nodes[nextIndex].value = value;
      nodes[nextIndex].leftChildIndex = -1; //初始值
      nodes[nextIndex].rightChildIndex = -1;

            if(nextIndex>1){ //root? 非root 就要處理向上的問題
              int parentIndex = nextIndex /2 ; //商數即parent

              if(nextIndex%2 ==0){
                nodes[parentIndex].leftChildIndex = nextIndex;
              }
              else{
                nodes[parentIndex].rightChildIndex = nextIndex;
              }
            }
            nextIndex++;
    }

    void preorderPrint(int index){
      if(index>0 && index < nextIndex){ //index 值不會錯 -> core
        cout << nodes[index].value << " ";
        preorderPrint(nodes[index].leftChildIndex);
        preorderPrint(nodes[index].rightChildIndex);
      }
    }
};

int main()
{
    BinaryTree tree(10); //給 capacity

    tree.insert(1);
    tree.insert(2);
    tree.insert(3);
    tree.insert(4);
    tree.insert(5);
    tree.insert(6);
    tree.insert(7);
    tree.insert(8);
    tree.insert(9);
    tree.insert(10);

    cout << "traverse:"<< endl;
    tree.preorderPrint(1); //從 root 開始 traverse
    cout << "end" << endl;

    return 0;
}
```