LAB7

遞迴與樹狀資料結構

長榮大學

資訊工程學系

班級：資工 2B

姓名：郭智榮

學號：109B30612

日期：2022/04/25

版本校定紀錄：

版本	更新紀錄	發佈日期
0.0.0.0	初版完成	2022/04/25

一. 實驗需求：

(一). 題目說明

描述：

請撰寫一程式，透過輸入多對父子關係的資料，來判斷該家族最遠的血緣關係為多長，爸爸到兒子距離為 1，而爺爺到爸爸的距離也為 1，透過建構樹狀資料結構及遞迴的方式完成此程式。

輸入說明

第一行輸入一個正整數代表成員數量，成員數量有 0~n-1，因此若輸入 8 則代表有 0~7 共 8 位成員。後續共有 n-1 行，每行將以空格隔開兩整數，前者代表父親，後者代表兒子。

輸出說明：

透過輸入的資料來建構樹狀資料結構後，搜尋最遠的血緣距離並將該值輸出。

(二). 演算法

i. 虛擬碼

定義 using namespace 型態的 std；//若無此定義則 vector 宣告將報錯導入 STL 的 vector 函式庫；
宣告一個 vector 型態變數 data 並指定存放整數(int)型態的資料，大小為 10000；
定義一個整數型態變數 max_distance 存放每個節點計算的結果；

int main(){
宣告三個整數型態變數 number, father, kid 分別用於儲存測資的資料數，每次輸入的父親節點，每次輸入的兒子節點；

輸出題目提示訊息；
輸入一整數並存入 number 代表本次測資的資料數；

宣告一個布林(boolean)型態陣列 isKids 並全數存放 false，用於紀錄節點是否為小孩，其大小為 number；

```
    for i = 1 ~ i < number (for i++){
        輸出題目提示訊息；
        輸入兩整數以空格隔開，並分別存入 father 及 kid；

        將 data[father]的位置存入 kid，紀錄 kid 為 father 的小孩；
        將 true 存入 isKids[kid]，以記錄 kid 節點為有父親；
    }

    宣告一整數型態變數 root 並存入-1，此變數紀錄樹根節點的位置；

    for i = 0 ~ i < number (for i++){
        如果 isKids[i]為 false{
            將 root 設為 i；
            break；
        }
    }

    宣告一整數型態變數 relation_result，並呼叫 relation 函數及傳
    入參數 root，將獲得的函數回傳值存入 relation_result；

    將 relation_result 及 max_distance 做數值比較，數值較大者存入
    max_distance 中；

    輸出結束訊息並輸出 max_distance 已告知使用者最遠血緣距離；

    return 0;
}

宣告一個會回傳整數的函數 relation，並需傳入一個參數 person{
    宣告三個整數型態變數 Max_1, Max_2, result 並將三個變數都存入
    0，其用於存放該節點往下路徑中最長的路徑距離，該節點往下路徑
    中第二長的距離，下一個節點往下搜尋的最長距離；

    如果 data[person]內的元素數量為 0{
        return 0；
    }
```

```
或者 data[person] 內的元素數量為 1{
    return 呼叫函數 relation 並傳入參數 data[person][0]，並
    將回傳結果+1；
}
否則{
    for i = 0 ~ i < data[person] 內的元素數量 (for i++){
        呼叫 relation 函數並傳入 data[person][i]，將回傳值
        加 1 後存入 result 中；

        如果 i 為 0{
            將 result 存入 Max_1 中；
        }
        或者 i 為 1{
            如果 result 小於等於 Max_1{
                將 result 存入 Max_2；
            }
            否則{
                將 Max_1 存入 Max_2 中；
                將 result 存入 Max_1 中；
            }
        }
        否則{
            如果 result 大於等於 Max_1{
                將 Max_1 存入 Max_2 中；
                將 result 存入 Max_1 中；
            }
            或者 result 大於 Max_2{
                將 result 存入 Max_2 中；
            }
        }
    }
    將 max_distance 及(Max_1 + Max_2)做數值比較，數值較大者
    存入 max_distance 中；
    回傳 Max_1；
}
}
```

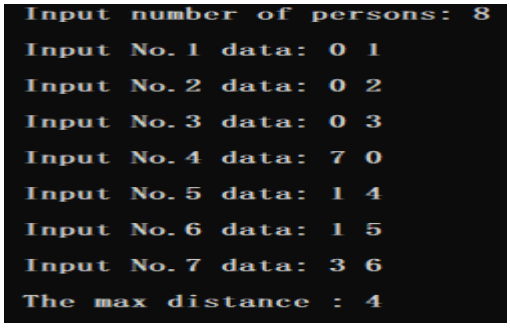
二. 完整程式碼：

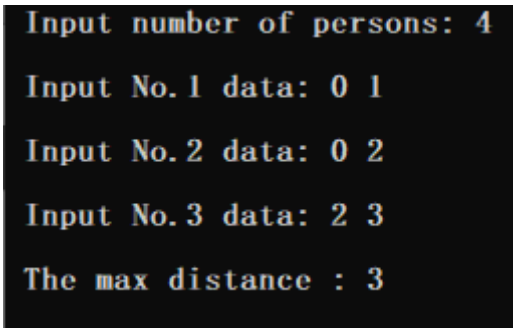
```
#include <iostream>
using namespace std;
#include<vector>
vector <int> data[10000];
int max_distance = 0;

int relation(int person) {
    int Max_1 = 0, Max_2 = 0, result = 0;
    //儲存節點往下最長的距離、第二長的距離及下一個子節點往下的最長距離。

    if(data[person].size() == 0){
    //該節點沒有往下的節點，意味著該節點沒有小孩。
        return 0;
        //沒有小孩則無法往下計算，故回傳 0。
    }
    else if(data[person].size() == 1){
    //該節點僅有一個小孩 。
        return (relation(data[person][0]) + 1);
        //回傳該節點小孩的回傳值+1。
    }
    else{
    //該節點有至少兩個小孩，因此需要迴圈搜尋所有小孩往下的距離。
        for(int i = 0 ; i < data[person].size() ; i++) {
        //迴圈需跑完該節點所有小孩，故執行到 0~size-1。
        result = relation(data[person][i]) + 1;
        //因為還有自己到小孩，故第 i 個小孩往下的最長距離結果需再加 1。
            if(i == 0){
            //若是第一個小孩，則無條件將 result 存入 Max_1。
                Max_1 = result;
            }
            else if(i == 1) {
            //第二個小孩需比較是否有比 Max_1 大，若無則存入 Max_2。
                if(result <= Max_1){
                //若 if 成立則代表第二個小孩往下距離比第一個小。
```

```
                    Max_2 = result;
                }
                else {
                //第二個小孩往下的距離比第一個大。
                    Max_2 = Max_1;
                    Max_1 = result;
                }
            }
            else {
            //第三個小孩(含)開始需看是否比Max_1及Max_2大才做對應調整。
                if(result >= Max_1) {
                //此小孩往下距離比 Max_1 大，故原 Max_1 需移到 Max_2。
                    Max_2 = Max_1;
                    Max_1 = result;
                }
                else if(result > Max_2){
                //此小孩往下距離比 Max_2 大，故只需修改 Max_2 即可。
                   Max_2 = result;
                }
            }
        }
        max_distance = max(max_distance, (Max_1 + Max_2));
        //將 max()內的兩個數值做比較，較大者存入 max_distance 中。
        return Max_1;
        //因最長距離已存入全域變數，故回傳 Max_1。
    }
}

int main(){
    int number, father, kid;
    //儲存資料數量、父節點及子節點。
    printf(" Input number of persons: ");
    scanf("%d", &number);
    //輸入資料數量。
    bool isKids[number] = {false};
    //儲存該節點是否為小孩的布林陣列。
```

```
    for(int i = 1 ; i < number ; i++){
    //輸入 number-1 筆資料。
        printf("\n Input No.%d data: ", i);
        scanf("%d %d", &father, &kid);
        //以空格隔開兩整數，前者為父節點，後者為子節點。
        data[father].push_back(kid);
        //紀錄父節點擁有的子節點及子節點編號。
        isKids[kid] = true;
        //將子節點在記錄是否為小孩的陣列中記為 true。
    }

    int root = -1;
    //儲存樹根節點編號。

    for(int i = 0 ; i < number ; i++){
    //檢查所有節點。
        if(isKids[i] == false){
        //代表節點 i 沒有父親，故為樹根節點。
            root = i;
            //將樹根節點編號改為 i。
            break;
            //已找到樹根，故跳出 for 迴圈。
        }
    }

    int relation_result = relation(root);
    //呼叫 function「relation」並傳入樹根編號。
    max_distance = max(relation_result, max_distance);
    //將 max()內的兩個數值做比較，較大者存入 max_distance 中。

    printf("\n The max distance : %d\n", max_distance);
    //輸出最遠血緣距離。

    return 0;
}
```

三. 輸入及輸出結果：





四. 心得與討論：

本次實作的樹狀資料結構是我第一次實作樹狀，但因為前面已有過 Linked List 的實作，所以在理解上也快了不少，整個程式碼大概看過一次就能稍微理解樹的產生方式；也因此在實作的過程中所花的時間也沒有到非常多，另外也知道 STL 這個函式庫的一些功能，在未來 C/C++ 的資料結構建構上也能更快速的產生。

遞迴的部分，因為平常就有在撰寫一些小專案的習慣，所以在這次遞迴的邏輯上並沒有太難，大概在腦中跑過一次測資的流程，就能大致理解整體運作的邏輯，因此在課堂結束後大概看過程式碼就能知道實際運作的流程並撰寫了。