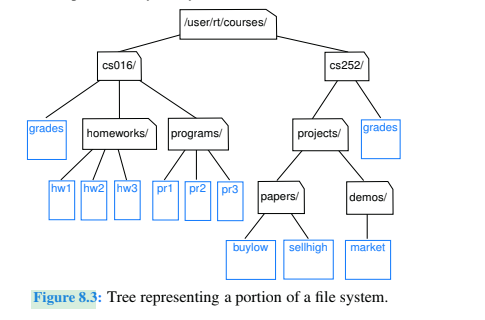
**Bài tập lý thuyết**



**R-8.1 The following questions refer to the tree of Figure 8.3.**

a. Which node is the root?

/user/rt/courses/

b. What are the internal nodes?

/user/rt/courses/

cs016/

homeworks/

programs/

cs252/

projects/

papers/

demos/

c. How many descendants does node cs016/ have?

9 descendants

d. How many ancestors does node cs016/ have?

1 ancestor: "/user/rt/courses/"

e. What are the siblings of node homeworks/?

"grades" và "programs/"

f. Which nodes are in the subtree rooted at node projects/?

papers/

buylow

sellhigh

market

demos/

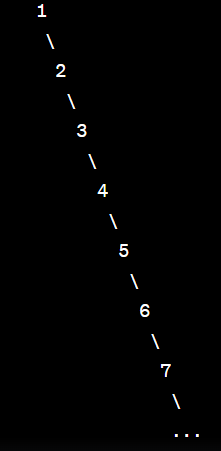
g. What is the depth of node papers/?

depth = 3

h. What is the height of the tree?

height = 4

**R-8.2 Show a tree achieving the worst-case running time for algorithm depth.**



Thời gian chạy trong trường hợp xấu nhất cho bất kỳ thuật toán nào hoạt động trên cây này sẽ là O(n), trong đó n là số nút trong cây.

**R-8.3 Give a justification of Proposition 8.3.**

The height of the root of a nonempty tree T, according to the recursive definition, equals the maximum depth among all leaves of tree T

Chiều cao của cây không rỗng T được định nghĩa đệ quy như sau:

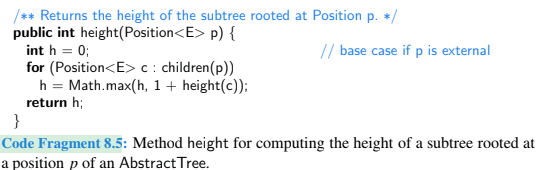
* Nếu T là cây có một nút thì chiều cao của nó bằng 0.
* Nếu T là một cây có nhiều nút, thì chiều cao của nó là chiều cao lớn nhất trong số tất cả các cây con của nó, cộng thêm 1.

Chiều cao của cây là thước đo độ sâu của cây từ gốc đến lá sâu nhất. Nó liên quan đến độ sâu của cây, là thước đo xem nút cách gốc bao xa. Độ sâu của một nút lá là độ dài của đường đi từ gốc đến lá đó.

Bây giờ, theo định nghĩa đệ quy về chiều cao, chiều cao của một cây khác rỗng T bằng chiều cao tối đa trong số tất cả các cây con của nó, cộng với 1. Điều này có nghĩa là để tìm chiều cao của T, ta cần tìm chiều cao của mỗi cây con trong số các cây con của nó. các cây con của nó và lấy giá trị lớn nhất của các độ cao đó. Chiều cao của cây con được định nghĩa giống như chiều cao của toàn bộ cây, nghĩa là độ sâu tối đa giữa tất cả các lá của cây con đó.

* chiều cao của một cây không trống T bằng với độ sâu tối đa trong số tất cả các lá của T. Điều này là do chiều cao của T được xác định bởi chiều cao của các cây con của nó và chiều cao của mỗi cây con được xác định bởi độ sâu tối đa giữa các tất cả các lá của nó.

**R-8.4 What is the running time of a call to T.height(p) when called on a position p distinct from the root of tree T? (See Code Fragment 8.5.)**



Thời gian chạy của lệnh gọi chiều cao(p) khi được gọi trên vị trí p khác với gốc của cây T là O(m), trong đó m là số nút trong cây con có gốc tại vị trí p.

Gọi m là số nút trong cây con có gốc ở vị trí p. Sau đó, vòng lặp for trong phương thức height sẽ thực hiện m lần để thăm mỗi con của p. Lệnh gọi đệ quy tới height(c) được thực hiện cho mỗi nút con c của p và vì mỗi nút được truy cập một lần nên sẽ có tối đa m lệnh gọi đệ quy. Do đó, tổng thời gian chạy của height(p) là O(m) vì mỗi nút được truy cập một lần và công việc được thực hiện tại mỗi nút là O(1)

Nếu p là gốc của cây thì thời gian chạy của height(p) tỷ lệ thuận với chiều cao của cây, không nhất thiết phải bằng số nút trong cây. Trong trường hợp xấu nhất, cây là cây suy biến (nghĩa là danh sách được liên kết) và thời gian chạy là O(n), trong đó n là số nút trong cây.

**R-8.5 Describe an algorithm, relying only on the BinaryTree operations, that counts the number of leaves in a binary tree that are the left child of their respective parent.**

Để đếm số lá trong cây nhị phân là con trái của cha mẹ tương ứng, chúng ta có thể sử dụng thuật toán sau:

* Khởi tạo biến đếm thành 0.
* Duyệt cây nhị phân theo chiều sâu, bắt đầu từ nút gốc.
* Đối với mỗi nút được truy cập, hãy kiểm tra xem nó có phải là lá không và liệu nó có phải là con trái của nút cha hay không.
* Nếu nút là một lá và là con trái của nút cha, hãy tăng biến đếm.
* Duyệt đệ quy cây con bên trái của nút hiện tại, sau đó duyệt cây con bên phải.

public int countLeftLeaves(BinaryTree<E> tree) {

int count = 0;

if (tree.isEmpty()) {

return count;

}

count = countLeftLeaves(tree, tree.root());

return count;

}

private int countLeftLeaves(BinaryTree<E> tree, Position<E> position) {

int count = 0;

if (tree.isExternal(position)) {

return count;

}

if (tree.hasLeft(position)) {

Position<E> leftChild = tree.left(position);

if (tree.isExternal(leftChild) && position == tree.parent(leftChild)) {

count++;

} else {

count += countLeftLeaves(tree, leftChild);

}

}

if (tree.hasRight(position)) {

Position<E> rightChild = tree.right(position);

count += countLeftLeaves(tree, rightChild);

}

return count;

}

**R-8.6 Let T be an n-node binary tree that may be improper. Describe how to represent T by means of a proper binary tree T′ with O(n) nodes.**

Để biểu diễn một cây nhị phân n nút T, có thể không đúng, như một cây nhị phân T' thích hợp với các nút O(n), ta sử dụng thuật toán:

* Tạo một nút gốc mới cho T' và biến nó thành nút gốc của nút gốc ban đầu của T.
* Đối với mỗi nút trong T chỉ có một nút con, hãy tạo một nút mới làm nút anh chị em của nút con đó và biến nó thành nút cha của nút con.
* Gắn nhãn cho mỗi nút mới bằng một mã định danh phân biệt nó với tất cả các nút khác trong T và T'.
* Xóa tất cả các nút ban đầu được gắn nhãn là nút "anh chị em" trong bước 2.

Cây kết quả T' sẽ là một cây nhị phân thích hợp và tổng số nút trong T' sẽ là O(n). Cụ thể, số nút trong T' sẽ là 2n - 1, vì mỗi nút mới được tạo ở bước 2 sẽ thêm một nút vào cây và nút gốc mới thêm một nút nữa.

**R-8.7 What are the minimum and maximum number of internal and external nodes in an improper binary tree with n nodes?**

Minimum internal nodes: 0

Maximum internal nodes: n-1

Maximum external nodes: If n is even: n/2

If n is odd: (n-1)/2

Minimum external nodes: 1

**R-8.8 Answer the following questions so as to justify Proposition 8.7.**

Proposition 8.7: Let T be a nonempty binary tree, and let n, nE, nI, and h denote the number of nodes, number of external nodes, number of internal nodes, and height of T, respectively. Then T has the following properties:

1. h+1 ≤ n ≤ 2^(h+1) −1

2. 1 ≤ nE ≤ 2^h

3. h ≤ nI ≤ 2^h −1

4. log(n+1) − 1 ≤ h ≤ n−1

Also, if T is proper, then T has the following properties:

1. 2h+1 ≤ n ≤ 2^(h+1) −1

2. h+1 ≤ nE ≤ 2^h

3. h ≤ nI ≤ 2^h −1

4. log(n+1)−1 ≤ h ≤ (n−1)/2

a. What is the minimum number of external nodes for a proper binary tree with height h? Justify your answer.

Số nút bên ngoài tối thiểu cho một cây nhị phân thích hợp có chiều cao h là 1, xảy ra khi cây chỉ bao gồm một nút bên ngoài. Điều này là do một cây nhị phân phù hợp phải có ít nhất hai nút bên ngoài ở cùng một cấp độ để tạo một cấp độ mới với hai con trên mỗi nút, có nghĩa là chiều cao tối thiểu cho một cây nhị phân phù hợp là 0 (một nút bên ngoài duy nhất).

b. What is the maximum number of external nodes for a proper binary tree with height h? Justify your answer.

Số nút bên ngoài tối đa cho một cây nhị phân thích hợp có chiều cao h là 2^h, xảy ra khi cây là một cây nhị phân hoàn chỉnh. Trong một cây nhị phân hoàn chỉnh, tất cả các cấp ngoại trừ có thể là cấp cuối cùng được điền đầy đủ và cấp cuối cùng được điền từ trái sang phải. Điều này có nghĩa là cấp cuối cùng có 2^h nút bên ngoài và tất cả các cấp trước đó có tổng cộng 2^h - 1 nút bên ngoài. Do đó, số nút bên ngoài tối đa là 2^h.

c. Let T be a proper binary tree with height h and n nodes. Show that log(n+1)−1 ≤ h ≤ (n−1)/2.

Đầu tiên, ta chứng minh rằng log(n+1) − 1 ≤ h. Vì n ≥ 2h+1, nên có thể lấy logarit của cả hai vế để được log(n) ≥ log(2h+1). Mà log(a^b) = b\*log(a) (tích chất của logarit), ta có thể viết lại giá trị này dưới dạng log(n) ≥ h+1. Trừ 1 cho cả hai vế cho ta log(n)-1 ≥ h, tương đương với log(n+1)-1 ≥ h.

Tiếp, chứng minh rằng h ≤ (n-1)/2. Vì n ≤ 2^(h+1)-1, nên ta có thể cộng 1 vào cả hai vế để được n+1 ≤ 2^(h+1). Lấy logarit của cả hai vế ta được log(n+1) ≤ h+1. Trừ 1 cho cả hai vế ta được log(n+1)-1 ≤ h. Nhân cả hai vế với 2 ta được 2\*log(n+1)-2 ≤ 2h, đơn giản thành (n-1)/2 ≤ h.

Từ 2 điều trên, suy ra điều phải chứng minh

d. For which values of n and h can the above lower and upper bounds on h be attained with equality?

Giới hạn dưới của h là log(n+1)-1, có thể đạt được equality khi cây nhị phân là một cây nhị phân đầy đủ. Trong một cây nhị phân hoàn chỉnh, mọi cấp độ ngoại trừ có thể là cấp độ cuối cùng được lấp đầy hoàn toàn và tất cả các nút ở cấp độ cuối cùng càng xa càng tốt. Trong trường hợp này, số nút n trong cây bằng 2^(h+1)-1, trong đó h là chiều cao của cây.

Giới hạn trên của h, là (n-1)/2, có thể đạt được equality khi cây nhị phân là cây nhị phân lệch. Trong cây nhị phân lệch, mỗi nút chỉ có một nút con, trái hoặc phải. Trong trường hợp này, số nút n trong cây bằng h+1, trong đó h là chiều cao của cây.

Vì vậy, đối với một cây nhị phân hoàn chỉnh có chiều cao h, cận dưới của h có thể đạt được equality khi n = 2^(h+1)-1. Đối với cây nhị phân lệch có chiều cao h, giới hạn trên của h có thể đạt được bằng khi n = h+1.

**R-8.9 Give a proof by induction of Proposition 8.8.**

Proposition 8.8: In a nonempty proper binary tree T, with nE external nodes and nI internal nodes, we have nE = nI +1.

Xét cây nhị phân thích hợp khác rỗng T chỉ có một nút. Trong trường hợp này, cây có nE = 1 nút bên ngoài và nI = 0 nút bên trong. Vì vậy, nE = nI + 1 đúng cho trường hợp cơ bản.

Ta chứng minh bằng quy nạp: Giả sử rằng mệnh đề đúng cho tất cả các cây nhị phân đúng khác rỗng có k nút trở xuống, trong đó k ≥ 1. Chúng ta cần chứng minh rằng mệnh đề cũng đúng cho cây nhị phân đúng khác rỗng có k+1 nút.

Xét một cây nhị phân thích hợp khác rỗng T với k+1 nút. Gọi T\_L và T\_R lần lượt là các cây con trái và phải của gốc T. Đặt nE\_L, nI\_L, nE\_R và nI\_R lần lượt là số nút bên ngoài và bên trong trong T\_L và T\_R.

Vì cả T\_L và T\_R đều có ít hơn k+1 nút nên theo giả thuyết quy nạp, ta có:

nE\_L = nI\_L + 1

nE\_R = nI\_R + 1

Xét tổng 2 phương trình:

nE\_L + nE\_R = nI\_L + nI\_R + 2

Vì gốc của T là một nút trong nên ta có:

nE = nE\_L + nE\_R

nI = nI\_L + nI\_R + 1

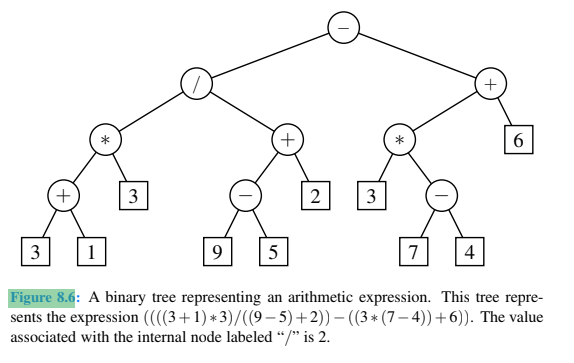
Thay thế các giá trị này vào phương trình trên:

nE = nI + 1

Do đó, mệnh đề đúng cho một cây nhị phân thích hợp khác rỗng với k+1 nút.

Bằng quy nạp, ta đã chỉ ra rằng đối với bất kỳ cây nhị phân thích hợp khác rỗng T, với nE nút ngoài và nI nút trong, nE = nI + 1.

**R-8.10 Find the value of the arithmetic expression associated with each subtree of the binary tree of Figure 8.6.**



3+1 = 4

(3+1) \*3 = 12

(3+1) \*3 / ((9-5) + 2)) = 2

9-5 = 4

(9-5) + 2 = 6

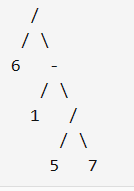
7-4 = 3

(7-4) \* 3 = 9

(3 ∗ (7−4)) + 6 = 15

((((3+1) ∗ 3) / ((9−5) + 2)) − ((3 ∗ (7−4)) + 6)) = -13

**R-8.11 Draw an arithmetic expression tree that has four external nodes, storing the numbers 1, 5, 6, and 7 (with each number stored in a distinct external node, but not necessarily in this order), and has three internal nodes, each storing an operator from the set {+,−,∗, /}, so that the value of the root is 21. The operators may return and act on fractions, and an operator may be used more than once.**



(6 / (1 - (5/7))) = 21