TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM

Báo cáo bài tập 07

- Phạm Quang Sang
- MSSV: 24120429

19/05/2025

I. Binary Tree

1. Initialize a NODE from a given value:

Cấp phát bộ nhớ bằng new, gán key với value truyền vào, gán nullptr cho thành viên left và right.

2-3-4: Sử dụng đệ quy ở 1 hàm cho truyền tham số là tham chiếu vector để tối ưu bộ nhớ

2. Pre-order traversal:

3. In-order traversal:

4. Post-order traversal:

*Nếu root null thì trả về vector trống ({}).

*Đệ quy ở "L" và "R", push "N" theo thứ tự LNR-NLR-LRN.

Các hàm duyệt sẽ được gọi liên tục cho đến lá bên trái nhất qua lá bên phải nhất, trong quá trình đó thì "N" sẽ được push theo thứ tự tương ứng.

5. Perform a level-order traversal:

Sử dụng queue để lưu các node cùng level từ trái sang phải, từ trên xuống dưới.

Mỗi vòng lặp sẽ xử lý một tầng (1level) của cây, đẩy giá trị vào vector, thêm nốt trái, phải vào hàng đợi ở mỗi node.

6. Calculate the number of NODEs in a binary tree:

Sử dụng đệ quy, ở mỗi node, count được cộng 1 giá trị, và gọi lại hàm ở con trái, phải cho đến khi hết node ở mỗi các nhánh con.

7. Calculate the sum of all NODE values in a binary tree:

Sử dụng đệ quy để đếm tổng số nút trong BST bằng cách cộng số nút trái, phải và chính nút hiện tại:

Nếu cây rỗng thì trả về 0.

Gọi đệ quy để đếm số nút ở cây con trái và cây con phải.

Cộng thêm 1 để tính luôn nút hiện tại.

8. Calculate the height of a NODE with a given value: (return -1 if not found):

Sử dụng hàm Height() để tính chiều cao của node hiện tại bằng cắch đệ quy.

Việc còn lại của hàm heightNode() là đệ quy đến node có value cần tính để dùng hàm Height() hoặc trả về -1 nếu không tồn tại value đó.

9. Calculate the level of a given NODE:

Tương tự như duyệt BFS ở mục I.5 nhưng không sử dụng vector để lưu các tầng mà chỉ dùng biến level để tính số tầng sau mỗi lần duyệt cây từ trên xuống, từ trái qua phải.

10. Count the number of leaf nodes in a binary tree:

Sử dụng đệ quy gọi con trái phải ở mỗi node, cho đến khi đến node là thì biến count được tăng 1 đơn vị.

II. Binary Tree - Binary Search Tree (BST)

1. Find and return a NODE with a specified value from a binary search tree

Tìm nút có giá trị bằng x trong BST bằng cách duyệt theo đặc tính phân nhánh của BST:

*Bắt đầu từ gốc pRoot, so sánh x với khóa (key) của nút hiện tại.

*Nếu bằng nhau, trả về nút đó.

*Nếu x lớn hơn, tiếp tục tìm bên phải (p_right); nếu nhỏ hơn, tìm bên trái (p_left).

*Nếu duyệt hết mà không thấy, trả về nullptr.

2. Add a NODE with a specified value to a binary search tree:

Chèn giá trị x vào BST bằng cách duyệt đến vị trí hợp lệ theo nguyên tắc phân nhánh của BST:

*Nếu cây rỗng, tạo nút mới làm gốc.

*Duyệt từ gốc, so sánh x với key của nút hiện tại:

* Nếu x lớn hơn, tiếp tục tìm bên phải (p_right); nếu nhỏ hơn, tìm bên trái (p_left).

*Khi gặp vị trí thích hợp (nhánh trái hoặc phải là nullptr), chèn nút mới vào.

*Nếu x đã tồn tại thì rời hàm.

3. Delete a NODE with a given value from a binary search tree:

Xóa giá trị x khỏi BST bằng cách tìm nút đó rồi xử lý theo số lượng con: 0–1 con thì nối trực tiếp, 2 con thì thay bằng nút kế thừa(thế mạng) trái nhất bên phải:

*Nếu cây rỗng thì thoát luôn.

*Duyệt cây để tìm nút có key == x, đồng thời giữ nút cha (prev) của nó.

*Nếu không tìm thấy (curr == nullptr), kết thúc.

*Xử lý ba trường hợp khi tìm thấy:

Nút cần xóa có 0 hoặc 1 con:

Gán con của curr (nếu có) cho cha của curr (prev).

Nếu curr là gốc, cập nhật lại pRoot.

Xóa curr.

Nút cần xóa có 2 con:

Tìm nút kế thừa (node trái nhất bên phải – successor).

Gán successor->key vào curr->key.

Xóa successor bằng cách nối cha của successor với successor->p_right.

4. Initialize a binary search tree from a given array:

Tạo cây BST từ mảng bằng cách chèn tuần tự từng phần tử vào cây sử dụng hàm Insert() được đề cập ở mục II.2.

5. Delete a binary search tree entirely:

Dựa trên ý tưởng duyệt cây ở mục I, sử dụng kiểu duyệt LRN để xử lý các con trước cha để không bị truy cập không hợp lệ.

6. Calculate the height of a binary search tree:

Tính chiều cao BST bằng cách đệ quy lấy chiều cao lớn nhất của hai nhánh con cộng thêm 1:

Nếu cây rỗng (nullptr) thì trả về -1.

Gọi đệ quy tính chiều cao của cây con bên trái và bên phải.

Chiều cao của cây là chiều cao lớn hơn giữa hai cây con cộng thêm 1.

7. Count the number of NODEs in a binary search tree with key values smaller than a given value:

Sử dụng hàm countNode() ở mục I.7 để tính: số nốt có giá trị nhỏ hơn = tổng số nút – số nút có giá trị lớn hơn + số nút con bên trái(nếu có nốt trùng giá trị được đưa vào):

*Nếu x nhỏ hơn khóa của nút hiện tại, thì mọi nút lớn hơn hoặc bằng đều nằm bên phải → chỉ cần đếm ở cây con trái.

*Ngược lại, tìm nút gốc con nhỏ nhất (root subtree) mà có giá trị ≥ x (tạm gọi là root_subtree_greater) trong cây.

*Khi đã tìm được root_subtree_greater, lấy tổng số nút trong cây trừ đi số nút của nhánh đó, vì tất cả nút ≥ x đều nằm trong đó.

*Nếu x bằng đúng khóa của root_subtree_greater, cộng thêm số nút bên trái vì tất cả đều nhỏ hơn x.

8. Count the number of NODEs in a given binary search tree whose key values are greater than a specified value:

Sử dụng hàm countLess(), Search() và countNode() ở các mục trên ta được: *countGreater() = countNode() – countLess() – Search()!=null(trừ đi trường hợp dấu '=').*

9. Determine if a binary tree is a binary search tree:

Cây nhị phân là cây nhị phân tìm kiếm khi: dãy số được lưu qua quá trình duyệt LNR ở mục I là 1 dãy tăng tuyệt đối, từ đó ta lưu các giá trị của cây vào vector và kiểm tra tính tăng tuyệt đối của nó là có thể xác định được.

10. Check if a binary tree is a full binary search tree (BST):

Kiểm tra cây có phải full BST bằng cách đảm bảo mỗi nút hoặc có 0 con hoặc có đủ 2 con:

*Cây rỗng được xem là đầy đủ → trả về true.

*Nếu là lá (không có con trái lẫn phải) → cũng là đầy đủ → trả về true.

*Nếu có cả hai con → đệ quy kiểm tra tiếp cả hai nhánh.

*Nếu chỉ có 1 con (trái hoặc phải) → không phải cây đầy đủ → trả về false.

III. AVL Tree

1. Initialize a NODE from a given value:

NODE* được cấp phát bộ nhớ, gán key với value được truyền vào, gán nullptr cho thành viên con trái phải, height được gán 0 (độ cao 1 nút luôn là 0).

2. Insert a NODE with a given value into a given AVL tree (Note that the given value may already exist):

Trước tiên chèn node mới vào cây như BST thông thường nhưng theo đệ quy để dễ dàng cập nhật chiều cao của các node trên nó. Sau khi thêm vào cây xong, do sử

dụng đệ quy nên chúng ta có thể quay lại các node trước nó, cập nhật chiều cao cho mỗi node *(1+ max(chiều cao con trái, con phải)).* Sau khi cập nhật chiều cao, hàm getBalance() trả về 1 hệ số cân bằng được tính như sau: balance = (h con trái – h con phải), theo lý thuyết độ chênh lệch độ cao của 2 con cùng cha là 0->1, nếu balance < -1 => bên phải cao hơn cho phép và ngược lại.

Khi có được hệ số cân bằng, nếu cây mất cân bằng tức |balance|>1, ta chia được 2 trường hợp là 'hỏng' bên trái hoặc bên phải, ở mỗi trường hợp lại chia ra 2 trường hợp con: hỏng trái-trái, trái-phải, phải-phải, phải-trái. Với x là giá trị vừa được chèn vào:

Balance > 1:

Nếu x > giá trị con trái=> lệch trái-phải

Nếu x < giá trị con trái=> lệch trái-trái

Balance <-1:

Nếu x > giá trị con phải=> lệch phải-phải

Nếu x < giá trị con phải=> lệch phải-trái

Thực hiện xoay theo các hàm tương ứng, với p-p t-t thì chỉ cần xoay chiều ngược hướng lệch, còn p-t t-p thì xoay ngược chiều từ ngoài vào trong(từ con tới cha tương ứng).

3. Delete a NODE with a given value from a given AVL tree (Note that the value may not exist):

Tương tự như xoá ở mục II, tuy nhiên ta sẽ đệ quy đến node có value cần xoá để có thể cập nhật chiều cao cho các node phía trên, sau khi xoá xong ta sẽ kiểm tra tính mất cân bằng như ở Insert() mục III.

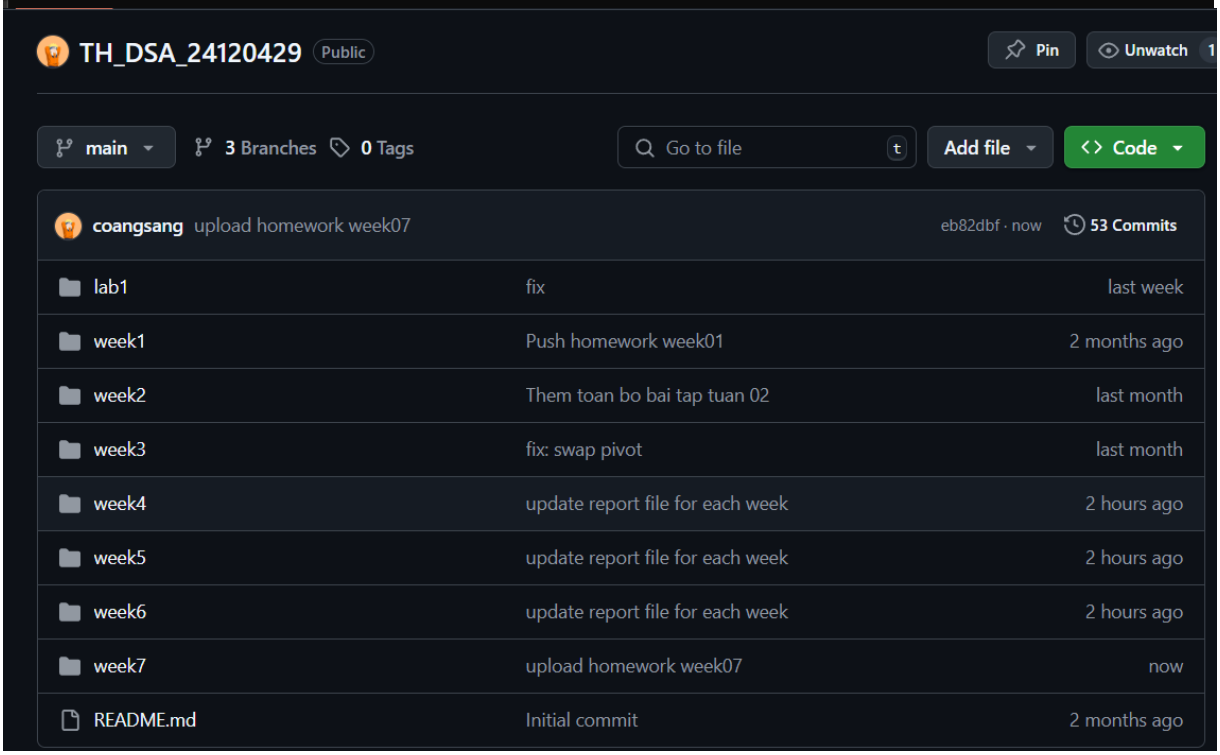
4. Determine if a binary tree is an AVL tree:

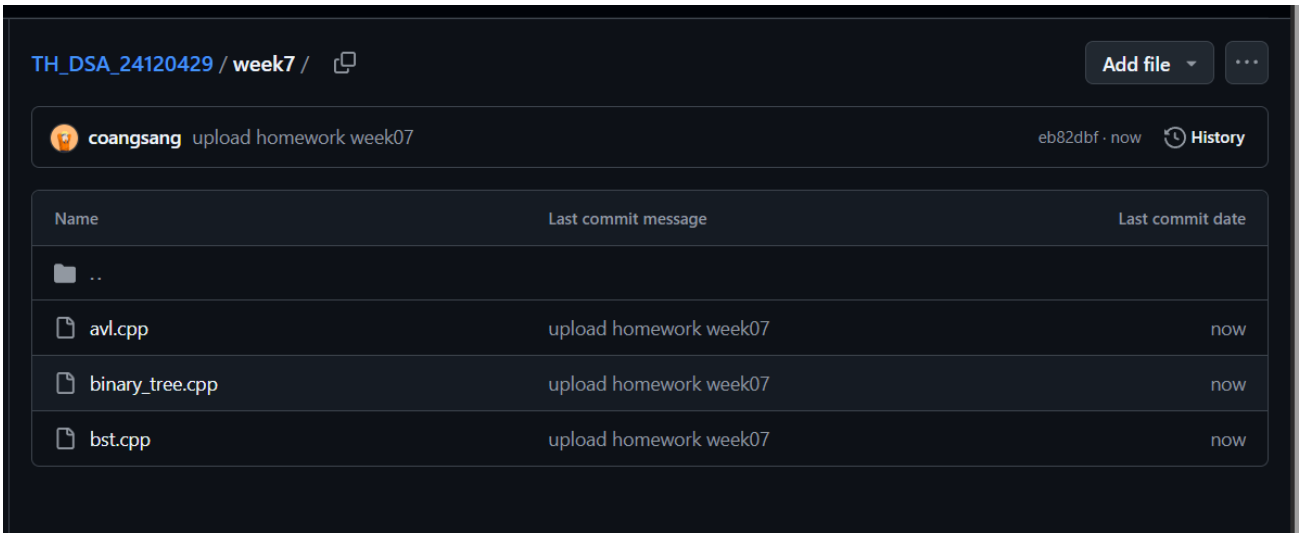
AVL là 1 cây nhị phân với mỗi node có 0 hoặc 2 con. Trước tiên xem thử nó có phải là cây nhị phân bằng các hàm ở mục II, sau đó:

Dùng đệ quy tới các con để xem mỗi node có đúng 0 hoặc 2 con hay không.

IV. GIT- GITHUB

```
PS D:\0HOCTAP\TH_DSA_24120429> git add week7/bst.cpp
PS D:\0HOCTAP\TH_DSA_24120429> git add week7/avl.cpp
PS D:\0HOCTAP\TH_DSA_24120429> git add week7/binary-tree.cpp
fatal: pathspec 'week7/binary-tree.cpp' did not match any files
PS D:\0HOCTAP\TH_DSA_24120429> ^C
PS D:\0HOCTAP\TH_DSA_24120429> git add week7/binary_tree.cpp
PS D:\0HOCTAP\TH_DSA_24120429> git commit -m "upload homework week07"
[main eb82dbf] upload homework week07
 3 files changed, 671 insertions(+)
 create mode 100644 week7/avl.cpp
 create mode 100644 week7/binary_tree.cpp
 create mode 100644 week7/bst.cpp
PS D:\0HOCTAP\TH_DSA_24120429> git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 20 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 4.05 KiB | 2.02 MiB/s, done.
Total 6 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com/coangsang/TH_DSA_24120429
   4efa4ed..eb82dbf  main -> main
PS D:\0HOCTAP\TH_DSA_24120429>
```




TH_DSA_24120429 / week7 /
Add file
coangsang upload homework week07
eb82dbf · now
History
Name
Last commit message
Last commit date
..
avl.cpp
upload homework week07
now
binary_tree.cpp
upload homework week07
now
bst.cpp
upload homework week07
now