AGENTRA

Cấu trúc dữ liệu

Tin học · Lớp 11

Giáo viên: Phúc Hảo

Mục tiêu bài học

- Hiểu khái niệm **cây** và **cây nhị phân** trong cấu trúc dữ liệu.
- Nắm được **tính chất sắp xếp** của cây nhị phân tìm kiếm (BST).
- Biết cách **tìm kiếm** và **chèn** một khóa vào BST.
- Giải thích được vì sao BST giúp tra cứu dữ liệu **nhanh**.

Ý chính: Cây nhị phân tìm kiếm là cách tổ chức dữ liệu giúp tìm kiếm nhanh nhờ luôn giữ thứ tự trái $<$ gốc $<$ phải.

Tình huống thực tế

Hãy tưởng tượng em tra một từ trong **từ điển giấy** dày 2000 trang. Em có lật từng trang từ đầu không? Chắc chắn là không!

Em mở giữa cuốn, thấy chữ cái lớn hơn thì lật về sau, nhỏ hơn thì lật về trước. Mỗi lần lật, em **loại bỏ một nửa** số trang còn lại.

Ý chính: Máy tính cũng cần một cách tổ chức dữ liệu "thông minh" để tra cứu nhanh như vậy. Đó là lý do ta học **cây nhị phân tìm kiếm**.

Câu hỏi: Làm sao lưu dữ liệu để mỗi lần so sánh loại được nửa số phần tử?

1

Khái niệm cây và cây nhị phân

Cây là gì?

Định nghĩa

Cây (tree) là cấu trúc dữ liệu gồm các **nút** (node) nối với nhau, trong đó có một nút **gốc** (root) và mỗi nút có thể có các **nút con**.

Liên hệ thực tế

Giống như **cây gia phả** của một dòng họ: ông tổ là gốc, con cháu là các nhánh tỏa ra.

Nút không có con gọi là **lá** (leaf).

Ghi nhớ

Trong cây, mỗi nút (trừ gốc) có **đúng một** nút cha. Không có "vòng lặp" — đi từ gốc luôn tới đúng một đường xuống mỗi nút.

Định nghĩa

Cây nhị phân (binary tree) là cây mà mỗi nút có **tối đa 2 nút con**: một **con trái** và một **con phải**.

Mỗi nút gồm:

- **Khóa** (giá trị lưu trong nút).
- Con trỏ tới **cây con trái**.
- Con trỏ tới **cây con phải**.

Liên hệ thực tế

Giống trò chơi "20 câu hỏi": mỗi câu chỉ trả lời **Có**/**Không**, dẫn ta rẽ trái hoặc phải để đoán đúng đồ vật.

2

Cây nhị phân tìm kiếm (BST)

Định nghĩa cây nhị phân tìm kiếm

Định nghĩa

Cây nhị phân tìm kiếm (Binary Search Tree - BST) là cây nhị phân thỏa mãn: với mọi nút, **mọi khóa ở cây con trái** $<$ khóa của nút $<$ **mọi khóa ở cây con phải**.

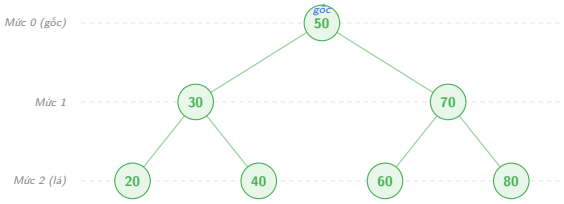
Ý chính: Nhờ tính chất "trái nhỏ - phải lớn", mỗi lần so sánh ta biết ngay phải đi về nhánh nào, loại bỏ hẳn một nửa cây.

Ghi nhớ

Quy tắc vàng của BST: **Trái** $<$ **Gốc** $<$ **Phải** — áp dụng cho *mọi* nút, không chỉ nút gốc.

Minh họa tính chất BST

Một cây nhị phân tìm kiếm hợp lệ



Kiểm tra tính chất BST tại gốc 50: nhánh trái $\{30, 20, 40\}$ đều < 50; nhánh phải $\{70, 60, 80\}$ đều > 50.
Tại nút 30: con trái $20 < 30 < 40$ con phải — đúng. Tại nút 70: $60 < 70 < 80$ — đúng.

Chiều cao và hiệu quả của BST

Chiều cao của cây là số cạnh trên đường dài nhất từ gốc xuống lá.

Khi tìm kiếm, số phép so sánh tối đa **bằng chiều cao** của cây.

Ghi nhớ

Cây **cân đối** với n nút có chiều cao khoảng $\log_2 n$, nên tìm kiếm chỉ mất $O(\log_2 n)$ bước — rất nhanh.

Liên hệ thực tế

Với $n = 1000$ nút, cây cân đối cao khoảng 10. Nghĩa là chỉ cần **10 lần so sánh** là tìm ra! So với dò lần lượt 1000 lần.

Lỗi thường gặp: nếu chèn dữ liệu đã sắp xếp sẵn, cây bị "lệch" thành chuỗi dài $\rightarrow$ chậm như tìm tuần tự.

So sánh cách lưu trữ dữ liệu

Tiêu chí	Mảng thường	Mảng đã sắp	BST cân đối
Tìm kiếm	$O(n)$	$O(\log_2 n)$	$O(\log_2 n)$
Chèn phần tử	$O(1)$	$O(n)$	$O(\log_2 n)$
Xóa phần tử	$O(n)$	$O(n)$	$O(\log_2 n)$
Giữ thứ tự	Không	Có	Có

Ý chính: BST kết hợp ưu điểm của hai loại mảng — vừa tìm kiếm nhanh, vừa chèn/xóa hiệu quả mà không cần dịch chuyển hàng loạt phần tử.

3

Thao tác trên BST

Thuật toán tìm kiếm trên BST

Bắt đầu từ **gốc**, lặp lại:

- Nếu khóa cần tìm **bằng** nút hiện tại $\Rightarrow$ tìm thấy.
- Nếu khóa **nhỏ hơn** $\Rightarrow$ sang cây con **trái**.
- Nếu khóa **lớn hơn** $\Rightarrow$ sang cây con **phải**.

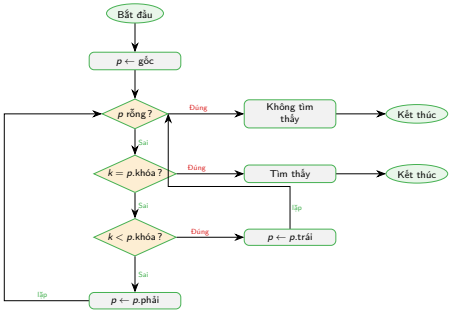
Nếu đi tới nút rỗng (null) $\Rightarrow$ **không có** khóa đó.

Ghi nhớ

Mỗi so sánh chỉ có 3 khả năng: bằng, nhỏ, lớn. Ta **không bao giờ quay lui**, luôn đi thẳng xuống dưới.

Lưu đồ thuật toán tìm kiếm

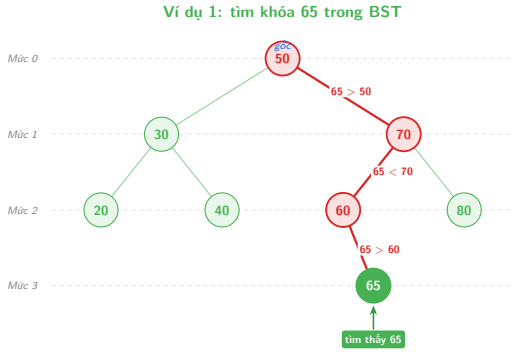
Lưu đồ: tìm khóa k trong BST



4

Ví dụ minh họa

Hình minh họa Ví dụ 1



Ví dụ 1: Tìm khóa 65 (lời giải)

Đề bài: Tìm khóa 65 trong cây BST ở hình trước, bắt đầu từ gốc 50.

Bước 1. So sánh 65 với gốc 50: vì $65 > 50$ nên khóa (nếu có) nằm ở nhánh **phải** $\Rightarrow$ đi sang nút 70.

$$65 > 50 \;\Rightarrow\; \text{sang phải}$$

Bước 2. So sánh 65 với 70: vì $65 < 70$ nên đi sang nhánh **trái** $\Rightarrow$ tới nút 60.

$$65 < 70 \;\Rightarrow\; \text{sang trái}$$

Bước 3. So sánh 65 với 60: vì $65 > 60$ nên đi sang nhánh **phải** $\Rightarrow$ tới nút 65.

$$65 > 60 \;\Rightarrow\; \text{sang phải}$$

Kết quả: Gặp đúng khóa 65 sau **4 phép so sánh** — bằng chiều dài đường đi từ gốc.

Ví dụ 2: Chèn khóa mới vào BST

Đề bài: Chèn khóa 45 vào cây BST có gốc 50 (các nút: $50, 30, 70, 20, 40, 60, 80$).

Bước 1. So sánh 45 với gốc 50: vì $45 < 50$ nên đi sang **trái** tới nút 30.

$$45 < 50 \;\Rightarrow\; \text{sang trái}$$

Bước 2. So sánh 45 với 30: vì $45 > 30$ nên đi sang **phải** tới nút 40.

$$45 > 30 \;\Rightarrow\; \text{sang phải}$$

Bước 3. So sánh 45 với 40: vì $45 > 40$ nên đi sang **phải**, nhưng con phải của 40 đang **rỗng**.

$$45 > 40,\ \text{con phải rỗng} \;\Rightarrow\; \text{đặt } 45 \text{ vào đây}$$

Ghi nhớ

Khi chèn, ta đi xuống như tìm kiếm cho tới khi gặp **chỗ rỗng**, rồi đặt khóa mới vào đó. Cách này luôn giữ đúng tính chất BST.

Tổng kết phần kiến thức

Ý chính: Cây nhị phân tìm kiếm tổ chức dữ liệu theo quy tắc **Trái** $<$ **Gốc** $<$ **Phải**, giúp tìm kiếm, chèn và xóa đều nhanh $O(\log_2 n)$ khi cây cân đối.

- **Cây** gồm các nút nối nhau, có gốc và lá.
- **Cây nhị phân**: mỗi nút tối đa 2 con (trái, phải).
- **BST**: mọi khóa trái $<$ nút $<$ mọi khóa phải.
- **Tìm kiếm/chèn**: đi từ gốc, so sánh để rẽ trái hoặc phải.

Lưu ý

Cây bị "lệch" (do chèn dữ liệu đã sắp xếp) sẽ mất ưu thế tốc độ — cần các kỹ thuật cân bằng cây (học ở bậc cao hơn).

Luyện tập nhanh - Câu 1

Luyện tập nhanh · Nhận biết

Cây nhị phân tìm kiếm (BST) có tính chất nào sau đây tại MỖI nút?

A. Con trái lớn hơn nút, con phải nhỏ hơn nút.

B. Con trái nhỏ hơn nút, con phải lớn hơn nút.

C. Mọi nút con đều lớn hơn nút cha.

D. Mọi nút con đều bằng nút cha.

Đáp án - Câu 1

Đáp án đúng: **B**. Mọi khoá ở cây con trái $<$ khoá nút, mọi khoá ở cây con phải $>$ khoá nút.

Bẫy: A đảo ngược thứ tự (lỗi rất phổ biến); C, D nhầm với cấu trúc heap/cây khác — BST so sánh theo TRÁI–PHẢI, không phải cha–con chung chung.

Luyện tập nhanh - Câu 2

Luyện tập nhanh · Nhận biết

Trong BST, muốn tìm giá trị NHỎ NHẤT ta đi theo hướng nào từ gốc?

A. Luôn rẽ phải đến khi hết con phải.

B. Luôn rẽ trái đến khi hết con trái.

C. Đi xuống nút giữa mỗi tầng.

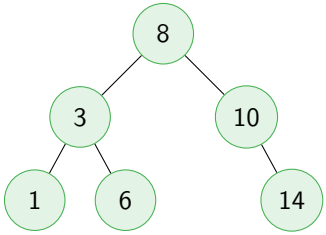
D. Duyệt toàn bộ cây rồi so sánh.

Đáp án - Câu 2

Đáp án đúng: **B**. Vì con trái luôn nhỏ hơn, giá trị nhỏ nhất nằm ở nút tận cùng bên trái.

Bẫy: A cho giá trị LỚN nhất (nhầm chiều); D đúng kết quả nhưng chậm — không tận dụng tính chất BST.

Luyện tập nhanh - Câu 3



Luyện tập nhanh · Nhận biết

Nút gốc của cây bên là nút nào?

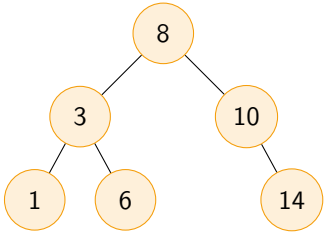
A. 1
B. 3
C. 8
D. 14

Đáp án - Câu 3

Đáp án đúng: **C**. Nút gốc là nút trên cùng, không có cha — chính là nút 8.

Bẫy: A là giá trị nhỏ nhất, D là giá trị lớn nhất; nhiều bạn nhầm "gốc" với "nhỏ nhất/lớn nhất". Gốc phụ thuộc thứ tự CHÈN, không phải giá trị.

Luyện tập nhanh - Câu 4



Luyện tập nhanh · Thông hiểu

Tìm giá trị 6: đường đi các nút được so sánh lần lượt là?

A. $8 \to 10 \to 6$

B. $8 \to 3 \to 6$

C. $8 \to 3 \to 1 \to 6$

D. $8 \to 6$

Đáp án - Câu 4

Đáp án đúng: **B**. $6 < 8$ rẽ trái tới 3; $6 > 3$ rẽ phải tới 6 — tìm thấy.

Bẫy: A rẽ nhầm phải ở gốc ($6 < 8$ phải rẽ trái); C đi thừa xuống nút 1 ($6 > 3$ nên không rẽ trái); D bỏ qua bước so sánh trung gian.

Luyện tập nhanh - Câu 5

Luyện tập nhanh · Thông hiểu

Chèn lần lượt các khoá $5, 3, 8, 2$ vào BST rỗng. Nút 2 sẽ là con của nút nào?

A. Con trái của nút 3

B. Con phải của nút 3

C. Con trái của nút 5

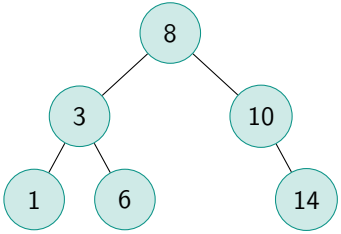
D. Con trái của nút 8

Đáp án - Câu 5

Đáp án đúng: **A**. $2 < 5$ rẽ trái tới 3; $2 < 3$ rẽ trái, chỗ trống $\Rightarrow$ con trái của 3.

Bẫy: B nhầm chiều ($2 < 3$ phải sang trái); C, D không theo đúng đường đi so sánh từ gốc.

Luyện tập nhanh - Câu 6



Luyện tập nhanh · Thông hiểu

Duyệt theo thứ tự giữa (trái–gốc–phải) cho dãy nào?

A. 8,3,1,6,10,14

B. 1,3,6,8,10,14

C. 1,6,3,14,10,8

D. 14,10,8,6,3,1

Đáp án - Câu 6

Đáp án đúng: **B**. Duyệt giữa (in-order) của BST luôn cho dãy TĂNG DẦN.

Bẫy: A là duyệt trước (gốc trước); C là duyệt sau; D là dãy giảm dần. Nhớ: in-order = sắp xếp tăng.

Luyện tập nhanh - Câu 7

Luyện tập nhanh · Vận dụng

Một BST CÂN BẰNG chứa 1000 nút. Số phép so sánh tối đa để tìm một khoá khoảng bao nhiêu?

A. Khoảng 1000

B. Khoảng 500

C. Khoảng 10

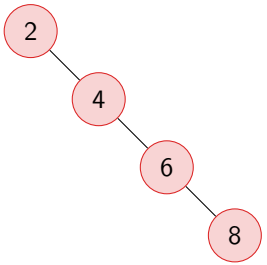
D. Khoảng 3

Đáp án - Câu 7

Đáp án đúng: **C**. Chiều cao cây cân bằng $\approx \log_2 1000 \approx 10$, nên tối đa khoảng 10 phép so sánh.

Bẫy: A là tìm kiếm TUYẾN TÍNH (không dùng tính chất BST); B nhầm chia đôi số nút $n/2$; D nhầm dùng $\log_{10}$ thay vì $\log_2$.

Luyện tập nhanh - Câu 8



Luyện tập nhanh · Vận dụng

Chèn dãy đã sắp xếp $2, 4, 6, 8$ tạo cây bên. Vì sao tìm kiếm ở đây kém hiệu quả?

A. Cây suy biến thành danh sách, chiều cao lớn.

B. Cây quá cân bằng nên chậm.

C. Vì thiếu nút gốc.

D. Vì các khoá đều là số chẵn.

Đáp án - Câu 8

Đáp án đúng: **A**. Chèn dãy đã sắp xếp làm cây "lệch" thành chuỗi, chiều cao $\approx n$, tìm kiếm trở về $O(n)$.

Bẫy: B ngược thực tế (cân bằng thì NHANH); C sai vì nút 2 vẫn là gốc; D vô lí — giá trị chẵn/lẻ không ảnh hưởng cấu trúc.

Nội dung luyện tập ở lớp

1. Chèn lần lượt dãy $50, 30, 70, 20, 40, 60, 80$ vào BST rỗng và vẽ cây kết quả.
2. Với cây vừa vẽ, ghi lại đường đi (các nút được so sánh) khi tìm khoá 60 và khoá 45 (không có).
3. Viết dãy kết quả khi duyệt cây theo thứ tự giữa và nhận xét.

Gợi ý

Luôn xuất phát từ gốc: nhỏ hơn thì rẽ trái, lớn hơn thì rẽ phải. Khoá không tồn tại $\Rightarrow$ dừng khi gặp chỗ trống. Duyệt giữa của BST luôn cho dãy tăng dần — dùng để kiểm tra cây vẽ đúng.

Bài tập về nhà

1. Chèn dãy $15, 9, 20, 4, 12, 17, 25$ vào BST và vẽ cây.
2. Tìm giá trị lớn nhất, nhỏ nhất trên cây câu 1; nêu đường đi.
3. Cho BST 15 nút cân bằng, ước lượng số phép so sánh tối đa khi tìm kiếm.
4. Giải thích vì sao chèn dãy đã sắp xếp lại tạo cây suy biến; đề xuất cách khắc phục.
5. Viết (bằng lời hoặc mã giả) thuật toán tìm kiếm một khoá trong BST.

Đọc trước: khái niệm *cây cân bằng* (AVL) và ý tưởng giữ chiều cao thấp.

Bảng tóm tắt - Thao tác trên BST

Thao tác	Cách làm	Độ phức tạp
Tìm kiếm	So sánh, rẽ trái/phải	$O(h)$
Chèn	Tìm chỗ trống rồi gắn nút	$O(h)$
Nhỏ nhất	Rẽ trái đến cùng	$O(h)$
Lớn nhất	Rẽ phải đến cùng	$O(h)$
Duyệt giữa	Cho dãy tăng dần	$O(n)$

h là chiều cao cây. Cây cân bằng: $h \approx \log_2 n$; cây suy biến: $h \approx n$.

Bảng tóm tắt - So sánh cấu trúc lưu trữ

Tiêu chí	Mảng chưa SX	Mảng đã SX	BST cân bằng
Tìm kiếm	$O(n)$	$O(\log n)$	$O(\log n)$
Chèn	$O(1)$	$O(n)$	$O(\log n)$
Duyệt tăng	cần sắp xếp	sẵn có	in-order

BST kết hợp ưu điểm: tìm kiếm nhanh MÀ vẫn chèn/xoá linh hoạt — miễn là giữ cây cân bằng.

Tổng kết

- **Tính chất cốt lõi:** với mỗi nút, cây con trái $<$ nút $<$ cây con phải — nền tảng cho mọi thao tác.
- **Tìm kiếm/chèn:** xuất phát từ gốc, nhỏ rẽ trái, lớn rẽ phải; chi phí tỉ lệ với chiều cao h.
- **Duyệt giữa (in-order)** luôn cho dãy tăng dần — cách kiểm tra cây và lấy dữ liệu đã sắp xếp.
- **Hiệu quả phụ thuộc dáng cây:** cân bằng $\Rightarrow O(\log n)$; suy biến (chèn dãy đã sắp) $\Rightarrow O(n)$.

Liên hệ: ý tưởng "chia đôi để tìm" của BST cũng chính là cách từ điển, cơ sở dữ liệu và công cụ tìm kiếm tổ chức dữ liệu để tra cứu cực nhanh.

Cảm ơn các em đã lắng nghe!

Bài giảng biên soạn với EduFlow · agentra.io.vn