B-TREE

# Kiến thức cơ bản:

* 1. Cấu trúc dữ liệu
     1. Khái niệm

Cấu trúc dữ liệu là một cách lưu dữ liệu trong máy tính để sử dụng dữ liệu một cách hiệu quả.​

Cấu trúc dữ liệu được xây dựng bằng cách kết hợp các kiểu dữ liệu cơ bản có sẵn trong ngôn ngữ lập trình. 1 cấu trúc dữ liệu còn bao gồm các thao tác xử lý như thêm, xóa, sửa trên cấu trúc dữ liệu đó.

* + 1. Phân loại

Một cách phân loại các cấu trúc dữ liệu là tuyến tính và không tuyến tính​

* Cấu trúc dữ liệu tuyến tính: các phần tử sẽ được lưu trữ liên tiếp, nối đuôi nhau.​
* Cấu trúc dữ liệu phi tuyến tính: tổ chức dữ liệu theo dạng cấp bậc và không theo trình tự nhất định nào cả, mỗi phần tử sẽ liên kết với 1 hoặc nhiều phần tử khác
  + 1. Mục đích

Cấu trúc dữ liệu được sử dụng với mục đích tạo thuận lợi cho việc:​

* Quản lí, cập nhật: các cấu trúc dữ liệu thường được định nghĩa các phương thức thêm, xóa, thay đổi dữ liệu tạo thuận lợi cho việc cập nhật dữ liệu.​
* Sắp xếp: một số cấu trúc dữ liệu có khả năng tự động cân bằng khi thêm dữ liệu mới vào, giảm thiểu chi phí khi cần sắp xếp dữ liệu​
* Tìm kiếm: một số cấu trúc dữ liệu cho phép giảm thiểu thời gian khi tìm kiếm
  1. B-Tree
     1. Khái niệm:

B-Tree là một cấu trúc dữ liệu không tuyến tính, dữ liệu được lưu trong các nút (node) liên kết với nhau và có thứ tự cấp bậc. Các node mang các giá trị khóa cùng với các con trỏ đến node con của nó. Node không có node con nào được gọi là node lá. B-Tree là một cây tìm kiếm tự cân bằng.

* + 1. Đặc điểm:

Khi sử dụng B-tree, ta định trước một số nguyên m gọi là bậc của cây (Max degree)

Một B-Tree cần thỏa mãn các tính chất:​

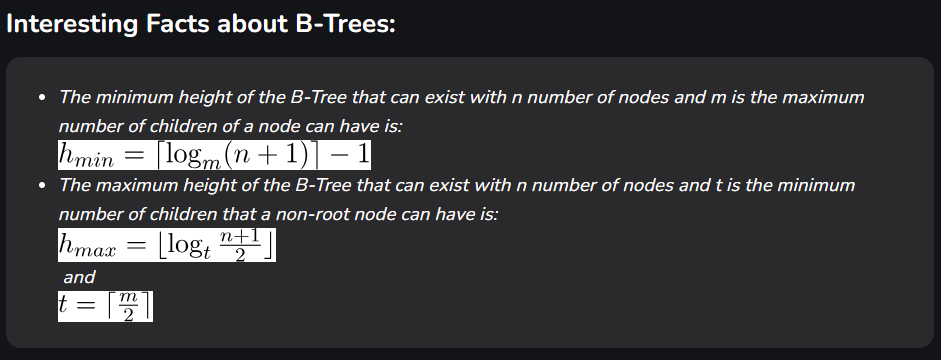
* Mỗi node có tối đa m node con và (m-1) khóa​
* Mỗi node có tối thiểu (m-1)/2 khóa (làm tròn xuống)​
* Số node con bằng số khóa cộng với 1 (trừ node lá)​
* Các khóa trong cây con thứ i đều nhỏ hơn khóa i, các khóa trong cây con thứ (i+1) đều lớn hơn khóa i​
* Các node lá cùng mức

Việc chọn giá trị m ảnh hưởng đến độ cao của cây. Khi m lớn, cây sẽ thấp hơn, có thể giúp việc tìm kiếm hiểu quả hơn, các node tốn nhiều bộ nhớ hơn.​

Thông thường 1 node sẽ chiếm 1 disk block trên ổ đĩa​

Nên chọn m phù hợp với disk block size, nghĩa là để dữ liệu tối đa trên 1 node gần bằng với kích thước 1 block

* + 1. Ưu, nhược điểm:
* Facts:



* Ưu điểm:
* Đảm bảo độ phức tạp luôn là O(logn) cho các tác vụ cơ bản.
* B Tree tự cân bằng.
* Hiệu quả cao khi sử dụng lên lượng dữ liệu cực lớn và các bộ nhớ điện tĩnh (non-volatile).
* Nhược điểm:
* Có thể tốn bộ nhớ do sử dụng nhiều con trỏ.
* Chậm hơn các cấu trúc tìm kiếm khác (Hash table) với dữ liệu nhỏ.
* Chưa thực hiện tốt range query.
  + 1. Các thao tác:
       1. Chèn B-Tree:

-Từ rễ, tìm lá thích hợp​

-Chèn khóa mới​

-Nếu thừa khóa, đưa khóa giữa lên node cha rồi tách làm 2. Có thể lan đến node trên

\*Chỉ chèn vào node lá

* + - 1. Duyệt cây:

Duyệt qua và in ra tất cả khóa theo thứ tự tăng dần​

-Nếu là lá thì in các khóa và kết thúc​

-Duyệt node con [0] rồi in khóa [0]​

...​

-Duyệt node con [m-1] rồi in khóa [m-1]​

-Duyệt node con [m]

* + - 1. Tìm kiếm:

Bắt đầu từ gốc. Nếu có nhiều hơn một khóa trong node hiện tại và khóa k cần tìm lớn hơn khóa đầu tiên, ta sẽ so sánh k với khóa tiếp theo. Nếu k nhỏ hơn khóa tiếp theo, ta sẽ tìm kiếm ở node con của khóa này. Nếu k lớn hơn tất cả khóa thì tìm ở node con ngoài cùng bên phải. Khi đến được một node lá mà không có k thì kết thúc tìm kiếm.

* + - 1. Xóa phần tử

Tìm kiếm phần tử đó​. Rồi xóa khóa tìm được và cân bằng cây nếu số khóa trong node nhỏ hơn tối thiểu

Nguyên tắc cân bằng:

Node lá:​

-Thiếu khóa: lấy khóa gần nhất của node cha ​

-Thừa khóa: đẩy khóa ở giữa lên node cha, tách làm 2​

Node nhánh, rễ:​

-Thiếu khóa: chuyền node con sang node bên cạnh, nếu không hợp lệ thì lấy khóa gần nhất ở node con, nếu không lấy được thì gộp 2 node con, nếu còn thiếu thì tiếp tục lấy ở node cha​

-Thiếu node con: truyền 1 khóa xuống node con​

-Thừa khóa: đẩy khóa ở giữa lên node cha, tách làm 2

# Kiến thức mở rộng:

* 1. Biến thể:
     1. B+Tree:
* Các node lá có toàn bộ các khóa trong cây
* Các node lá liên kết với nhau thành 1 danh sách liên kết
* Các khóa ở tầng ngay trên tầng lá là khóa nhỏ nhất của lá bên phải

Các thao tác của B+Tree có điểm khác so với B-Tree:

* + - * 1. Chèn: Khi node lá đầy, tách node lá thành 2 tại vị trí ⌊ m/2 ⌋ và thêm khóa thứ ⌊ m/2 ⌋ vào node cha. Nếu node cha đầy thì xử lí như B-Tree
        2. Xóa:

Khóa không có ở node cha:

Sau khi xóa, nếu cây cần cân bằng (số khóa nhỏ hơn tối thiểu) thì lấy một khóa ở node bên cạnh, nếu không lấy được thì gộp node hiện tại với node bên cạnh. Các tầng trên xử lí như B-Tree

Khóa cần xóa có ở node cha:

Xử lí tương tự nhưng khi xóa khóa ở lá thì xóa luôn ở node cha chứa khóa đó. Nếu node không ghép thì thêm tiếp theo khóa đã xóa vào node cha

* Ưu điểm so với B-Tree:
* Độ bạt (fan-out) lớn hơn, phù hợp với range query trong khi vẫn giữ đc O(logn) với các thao tác cơ bản nhờ các node trung gian.
* Chiều cao của cây thấp hơn B-Tree
* Nhược:
* Cài đặt phức tạp hơn.
  + 1. B\*Tree: Có cấu trúc tương tự như B-Tree nhưng với cấp độ là m thì số khóa tối thiểu (2m-1)/3 khóa thay vì (m-1)/2 khóa. Mỗi node chứa nhiều khóa hơn, chiều cao của cây thấp hơn so với B-Tree cùng cấp. Khi chèn, nếu node lá và node cạnh đều có số khóa tối đa thì chuyền khóa ở giữa lên khóa cha, rồi tách 2 node đó thành 3 node đều mang số khóa tối thiểu. Hàm xóa phức tạp so với B-Tree.
    2. B\*+Tree: Kết hợp các đặc điểm của B+Tree và B\*Tree
  1. Ứng dụng:
     1. Indexing

Indexing là sử dụng một cấu trúc dữ liệu để định vị và truy cập nhanh nhất vào dữ liệu trong các bảng database

Với mỗi khóa, index trỏ tới địa chỉ dữ liệu trong một bảng, giống như Mục lục của một cuốn sách (Gồm tên đề mục và số trang), nó giúp truy vấn trở nên nhanh chóng

B-Tree được sử dụng cho Indexing bởi vì:

Tìm kiếm hiệu quả: Thời gian tìm kiếm trong một B-tree tương đương với độ sâu của cây. Mà B-Tree được thiết kế nhằm tối thiểu độ sâu

Chèn và xóa hiệu quả: Khi chèn, xóa, B-tree chỉ cần thay đổi một số block dữ liệu trên ổ đĩa thay vì di chuyển toàn bộ dữ liệu

Thích hợp cho ổ đĩa: Có thể linh hoạt thay đổi cấp độ của cây khi tạo B-Tree cho phù hợp với ổ đĩa

So sánh với hash index:

-Chậm hơn vì phải qua nhiều node​

+ Có thể sử dụng cho toán từ tìm kiếm 1 khoảng giá trị như >, < ​

+ Có thể tối ưu hóa toán tử ORDER BY trong khi Hash index không thể tìm kiếm được phần từ tiếp theo trong Order.​

+ Một phần của khóa cũng có thể được sử dụng để tìm kiếm​

+ Khóa có thể trùng nhau

# Ví dụ các hệ thống sử dụng:

Nhiều hệ thống sử dụng B-Tree trong quản lý file như Apple File System (APFS), New Technology File System (NTFS).

Các cơ sở dữ liệu quan hệ (RDB) Oracle và MySQL sử dụng B-Tree (và các biến thể) trong đánh index bởi tối ưu trong hiệu năng so với Binary Tree thông thường.

# Quiz:

* Câu 1: Cho B Tree bậc m, số khóa và số cây con tối đa của 1 node trung gian là?

Đáp án: m-1 và m

* Câu 2: Cho B Tree bậc m, số khóa tối thiểu của 1 node không phải gốc là?

Đáp án: (m-1)/2 làm tròn xuống

* Câu 3: 1 disk block có kích cỡ 4kB thì nên chọn m khoảng bao nhiêu để tối ưu, biết thông tin trong 1 khóa chiếm 40 byte?

Đáp án: 100

* Câu 4: Độ phức tạp của các tác vụ trong B Tree là?

Đáp án: O(logn)

* Câu 5: Độ phức tạp không gian của B Tree là?

Đáp án: O(n)

* Câu 6: Khi chèn 1 khóa vào cây, có thể chèn trực tiếp vào node trung gian?

Đáp án: Sai

* Câu 7: Khi thực hiện xóa 1 khóa, nếu số khóa còn lại không đủ tối thiểu và những node kề không thừa khóa thì thao tác nào được thực hiện?

Đáp án: Hợp với node kề (catenate)

* Câu 8: Trong B+Tree, các node trung gian có chứa con trỏ trỏ đến địa chỉ dữ liệu

Đáp án: Sai

* Câu 9: B+Tree có những ưu điểm nào so với B Tree?

Đáp án: Độ bạt cao hơn, cây thấp hơn

* Câu 10: B Tree được ứng dụng ở những hệ thống nào?

Đáp án: Hệ thống lưu trữ file, Hệ quản trị CSDL

* Câu 11: B+Tree không cho phép khóa trùng nhau.

Đáp án: Sai

# Bài về nhà:

[Link BTVN](https://docs.google.com/forms/d/e/1FAIpQLSfMlOrEBJ41ggUMmMBbYIZe7mAi--300SjB06ZBtHKWolxR_w/viewform)