**9.8. CÂY TÌM KIẾM CƠ SỐ (RADIX SEARCH TREE - RST)**

Trong cây tìm kiếm số học, cũng như cây nhị phân tìm kiếm, phép tìm kiếm tại mỗi bước phải so sánh giá trị khoá X với giá trị lưu trong một nút của cây. Trong trường hợp các khoá có cấu trúc lớn, việc so sánh này có thể mất nhiều thời gian.

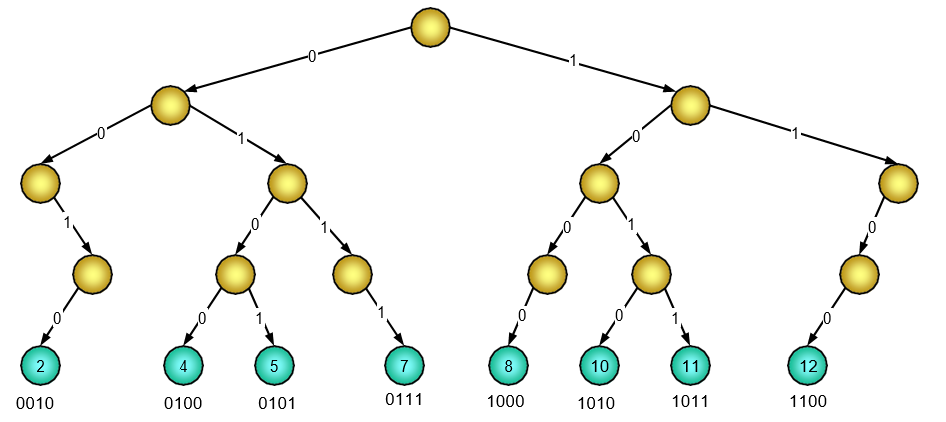
Cây tìm kiếm cơ số là một phương pháp khắc phục nhược điểm đó, nội dung của nó có thể tóm tắt như sau:

Trong cây tìm kiếm cơ số là một cây nhị phân, chỉ có nút lá chứa giá trị khoá, còn giá trị chứa trong các nút nhánh là vô nghĩa. Các nút lá của cây tìm kiếm cơ số đều nằm ở mức z + 1.

Đối với nút gốc của cây tìm kiếm cơ số, nó có tối đa hai nhánh con, mọi khoá chứa trong nút lá của nhánh con trái đều có bit cao nhất là 0, mọi khoá chứa trong nút lá của nhánh con phải đều có bit cao nhất là 1.

Đối với hai nhánh con của nút gốc, vấn đề tương tự với bit thứ z - 2, ví dụ với nhánh con trái của nút gốc, nó lại có tối đa hai nhánh con, mọi khoá chứa trong nút lá của nhánh con trái đều có bit thứ z - 2 là 0 (chúng bắt đầu bằng hai bit 00), mọi khoá chứa trong nút lá của nhánh con phải đều có bit thứ z - 2 là 1 (chúng bắt đầu bằng hai bit 01)…

Tổng quát với nút ở mức d, nó có tối đa hai nhánh con, mọi nút lá của nhánh con trái chứa khoá có bit z - d là 0, mọi nút lá của nhánh con phải chứa khoá có bit thứ z - d là 1.



**Cây tìm kiếm cơ số**

Khác với cây nhị phân tìm kiếm hay cây tìm kiếm số học. Cây tìm kiếm cơ số được khởi tạo gồm có một nút gốc, và **nút gốc tồn tại trong suốt quá trình sử dụng**: nó không bao giờ bị xoá đi cả.

Để tìm kiếm một giá trị X trong cây tìm kiếm cơ số, ban đầu ta đứng ở nút gốc và duyệt dãy bit của X từ trái qua phải (từ bit z - 1 đến bit 0), gặp bit bằng 0 thì rẽ sang nút con trái còn gặp bit bằng 1 thì rẽ sang nút con phải, cứ tiếp tục như vậy cho tới khi một trong hai tình huống sau xảy ra:

Hoặc đi tới một nút rỗng (do rẽ theo liên kết nil) quá trình tìm kiếm thất bại do X không có trong RST

Hoặc đã duyệt hết dãy bit của X và đang đứng ở một nút lá, quá trình tìm kiếm thành công vì chắc chắn nút lá đó chứa giá trị đúng bằng X.

{Hàm tìm kiếm trên cây tìm kiếm cơ số, nó trả về nút lá chứa khoá tìm kiếm X nếu tìm thấy, trả về nil nếu không tìm thấy. z là độ dài dãy bit biểu diễn một khoá}

function RSTSearch(X: TKey): PNode;

var

b: Integer; p: PNode;

begin

b := z; p := Root; {Bắt đầu với nút gốc, đối với RST thì gốc luôn có sẵn}

repeat

b := b - 1; {Xét bit b của X}

if <Bit b của X là 0> then p := p^.Left {Gặp 0 rẽ trái}

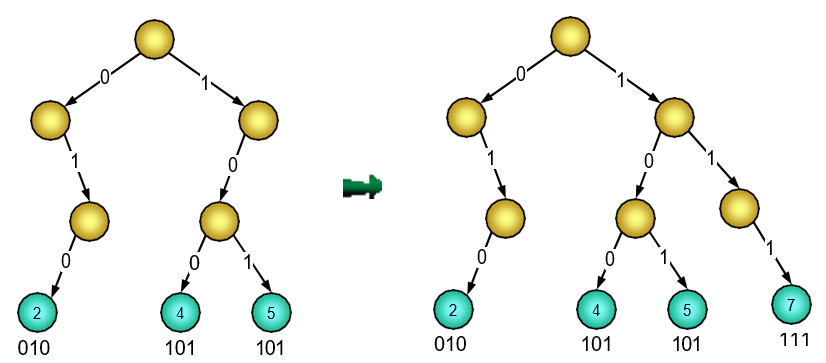
else p := p^.Right; {Gặp 1 rẽ phải}

until (p = nil) or (b = 0); RSTSearch := p;

end;

Thao tác chèn một giá trị X vào RST được thực hiện như sau: Đầu tiên, ta đứng ở gốc và duyệt dãy bit của X từ trái qua phải (từ bit z - 1 về bit 0), cứ gặp 0 thì rẽ trái, gặp 1 thì rẽ phải. Nếu quá trình rẽ theo một liên kết nil (đi tới nút rỗng) thì lập tức tạo ra một nút mới, và nối vào theo liên kết đó để có đường đi tiếp. Sau khi duyệt hết dãy bit của X, ta sẽ dừng lại ở một nút lá của RST, và công việc cuối cùng là đặt giá trị X vào nút lá đó.

Ví dụ:



**Với độ dài dãy bit z = 3, cây tìm kiếm cơ số gồm các khoá 2, 4, 5 và sau khi thêm giá trị 7**

{Thủ tục chèn khoá X vào cây tìm kiếm cơ số}

procedure RSTInsert(X: TKey);

var

b: Integer;

p, q: PNode;

begin

b := z; p := Root; {Bắt đầu từ nút gốc, đối với RST thì gốc luôn  nil}

repeat

b := b - 1; {Xét bit b của X}

q := p; {Khi p chạy xuống nút con thì q^ luôn giữ vai trò là nút cha của p^}

if <Bit b của X là 0> then p := p^.Left {Gặp 0 rẽ trái}

else p := p^.Right; {Gặp 1 rẽ phải}

if p = nil then {Không đi được thì đặt thêm nút để đi tiếp}

begin

New(p); {Tạo ra một nút mới và đem p trỏ tới nút đó}

p^.Left := nil; p^.Right := nil;

if <Bit b của X là 0> then q^.Left := p {Nối p^ vào bên trái q^}

else q^.Right := p; {Nối p^ vào bên phải q^}

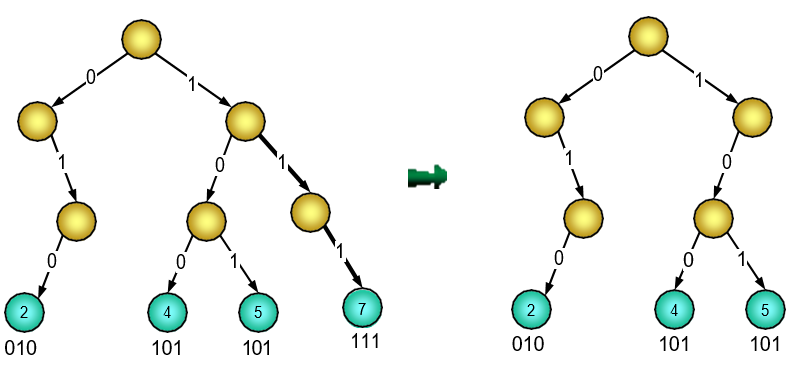
end;

until b = 0;

p^.Info := X; {p^ là nút lá để đặt X vào}

end;

Với cây tìm kiếm cơ số, việc xoá một giá trị khoá không phải chỉ là xoá riêng một nút lá mà còn phải xoá toàn bộ nhánh độc đạo đi tới nút đó để tránh lãng phí bộ nhớ.



**RST chứa các khoá 2, 4, 5, 7 và RST sau khi loại bỏ giá trị 7**

Ta lặp lại quá trình tìm kiếm giá trị khoá X, quá trình này sẽ đi từ gốc xuống lá, tại mỗi bước đi, mỗi khi gặp một nút ngã ba (nút có cả con trái và con phải - nút cấp hai), ta ghi nhận lại ngã ba đó và hướng rẽ. Kết thúc quá trình tìm kiếm ta giữ lại được ngã ba đi qua cuối cùng, từ nút đó tới nút lá chứa X là con đường độc đạo (không có chỗ rẽ), ta tiến hành dỡ bỏ tất cả các nút trên đoạn đường độc đạo khỏi cây tìm kiếm cơ số. Để không bị gặp lỗi khi cây suy biến (không có nút cấp 2) ta coi gốc cũng là nút ngã ba

{Thủ tục xoá khoá X khỏi cây tìm kiếm cơ số}

procedure RSTDelete(X: TKey);

var

b: Integer;

p, q, TurnNode, Child: PNode;

begin

{Trước hết, tìm kiếm giá trị X xem nó nằm ở nút nào}

b := z; p := Root;

repeat

b := b - 1;

q := p; {Mỗi lần p chuyển sang nút con, ta luôn đảm bảo cho q^ là nút cha của p^}

if <Bit b của X là 0> then p := p^.Left

else p := p^.Right;

if (b = z - 1) or (q^.Left  nil) and (q^.Right  nil) then {q^ là nút ngã ba}

begin

TurnNode := q; Child := p; {Ghi nhận lại q^ và hướng rẽ}

end;

until (p = nil) or (b = 0);

if p = nil then Exit; {X không tồn tại trong cây thì không xoá được}

{Trước hết, cắt nhánh độc đạo ra khỏi cây}

if TurnNode^.Left = Child then TurnNode^.Left := nil

else TurnNode^.Right := nil

p := Child; {Chuyển sang đoạn đường độc đạo, bắt đầu xoá}

repeat

q := p;

{Lưu ý rằng p^ chỉ có tối đa một nhánh con mà thôi, cho p trỏ sang nhánh con duy nhất nếu có}

if p^.Left  nil then p := p^.Left else p := p^.Right;

Dispose(q); {Giải phóng bộ nhớ cho nút q^}

until p = nil;

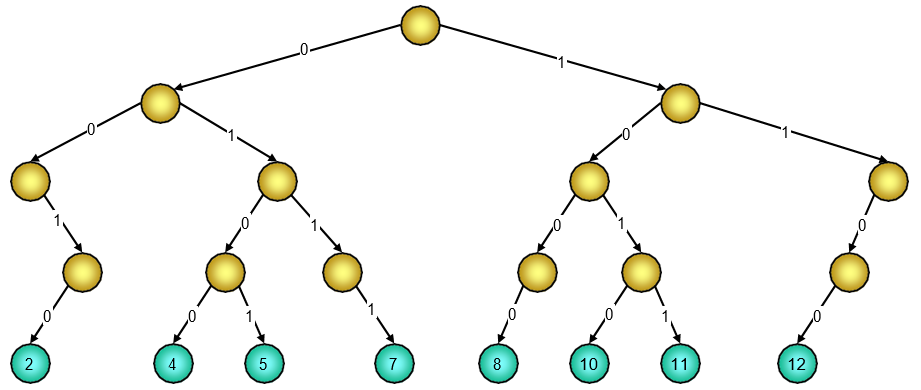
end;

Ta có một nhận xét là: Hình dáng của cây tìm kiếm cơ số không phụ thuộc vào thứ tự chèn các khoá vào mà chỉ phụ thuộc vào giá trị của các khoá chứa trong cây.

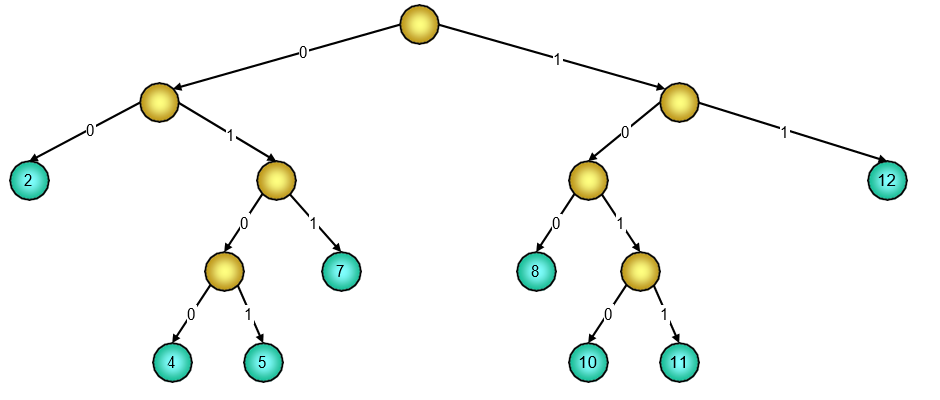
Đối với cây tìm kiếm cơ số, độ phức tạp tính toán cho các thao tác tìm kiếm, chèn, xoá trong trường hợp xấu nhất cũng như trung bình đều là O(z). Do không phải so sánh giá trị khoá dọc đường đi, nó nhanh hơn cây tìm kiếm số học nếu như gặp các khoá cấu trúc lớn. Tốc độ như vậy có thể nói là tốt, nhưng vấn đề bộ nhớ khiến ta phải xem xét: Giá trị chứa trong các nút nhánh của cây tìm kiếm cơ số là vô nghĩa dẫn tới sự lãng phí bộ nhớ.

Một giải pháp cho vấn đề này là: Duy trì hai dạng nút trên cây tìm kiếm cơ số: Dạng nút nhánh chỉ chứa các liên kết trái, phải và dạng nút lá chỉ chứa giá trị khoá. Cài đặt cây này trên một số ngôn ngữ định kiểu quá mạnh đôi khi rất khó.

Giải pháp thứ hai là đặc tả một cây tương tự như RST, nhưng sửa đổi một chút: nếu có nút lá chứa giá trị X được nối với cây bằng một nhánh độc đạo thì cắt bỏ nhánh độc đạo đó, và thay vào chỗ nhánh này chỉ một nút chứa giá trị X. Như vậy các giá trị khoá vẫn chỉ chứa trong các nút lá nhưng các nút lá giờ đây không chỉ nằm trên mức z + 1 mà còn nằm trên những mức khác nữa. Phương pháp này không những tiết kiệm bộ nhớ hơn mà còn làm cho quá trình tìm kiếm nhanh hơn. Giá phải trả cho phương pháp này là thao tác chèn, xoá khá phức tạp. Tên của cấu trúc dữ liệu này là Trie (Trie chứ không phải Tree) tìm kiếm cơ số.



a)



b)

**Cây tìm kiếm cơ số a) và Trie tìm kiếm cơ số b)**

Tương tự như phương pháp sắp xếp bằng cơ số, phép tìm kiếm bằng cơ số không nhất thiết phải chọn hệ cơ số 2. Ta có thể chọn hệ cơ số lớn hơn để có tốc độ nhanh hơn (kèm theo sự tốn kém bộ nhớ), chỉ lưu ý là cây tìm kiếm số học cũng như cây tìm kiếm cơ số trong trường hợp này không còn là cây nhị phân mà là cây R\_phân với R là hệ cơ số được chọn.

Trong các phương pháp tìm kiếm bằng cơ số, thực ra còn một phương pháp tinh tuý và thông minh nhất, nó có cấu trúc gần giống như cây nhưng không có nút dư thừa, và quá trình duyệt bit của khoá tìm kiếm không phải từ trái qua phải mà theo thứ tự của các bit kiểm soát lưu tại mỗi nút đi qua. Phương pháp đó có tên gọi là Practical Algorithm To Retrieve Information Coded In Alphanumeric (PATRICIA) do Morrison đề xuất. Tuy nhiên, việc cài đặt phương pháp này khá phức tạp (đặc biệt là thao tác xoá giá trị khoá), ta có thể tham khảo nội dung của nó trong các tài liệu khác.