Lecture 3: Splay tree

- Cây tìm kiếm nhị phân tự hiệu chỉnh (splay tree) được các tác giả Daniel Dominic Sleator và Robert Endre Tarjan đề xuất năm 1985 [4]. Splay tree là cây tìm kiếm nhị phân, song mỗi phép toán trên cây đi kèm thao tác hiệu chỉnh lại cây (Self-Adjusting).

- Splay tree được xây dựng trên nguyên tắc các nút có tần suất truy cập lớn luôn là nút gốc hoặc nút gần nút gốc.

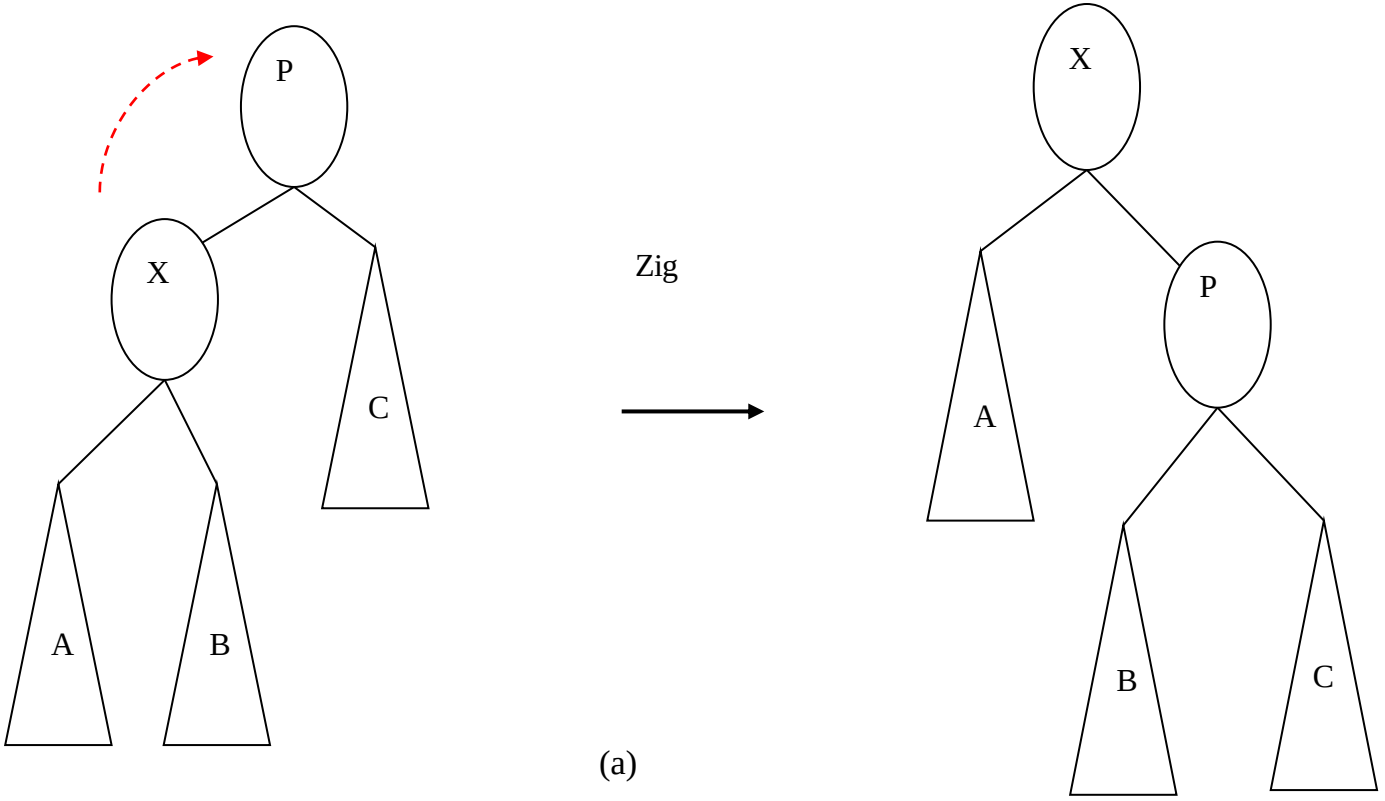
Splay tree

- Splay tree là một cây tìm kiếm nhị phân thỏa mãn điều kiện mọi nút trên cây, sau khi được truy cập, sẽ được hiệu chỉnh (splaying) về gốc của cây bằng một dãy các phép quay.

Cấu trúc dữ liệu cho cây splay tree

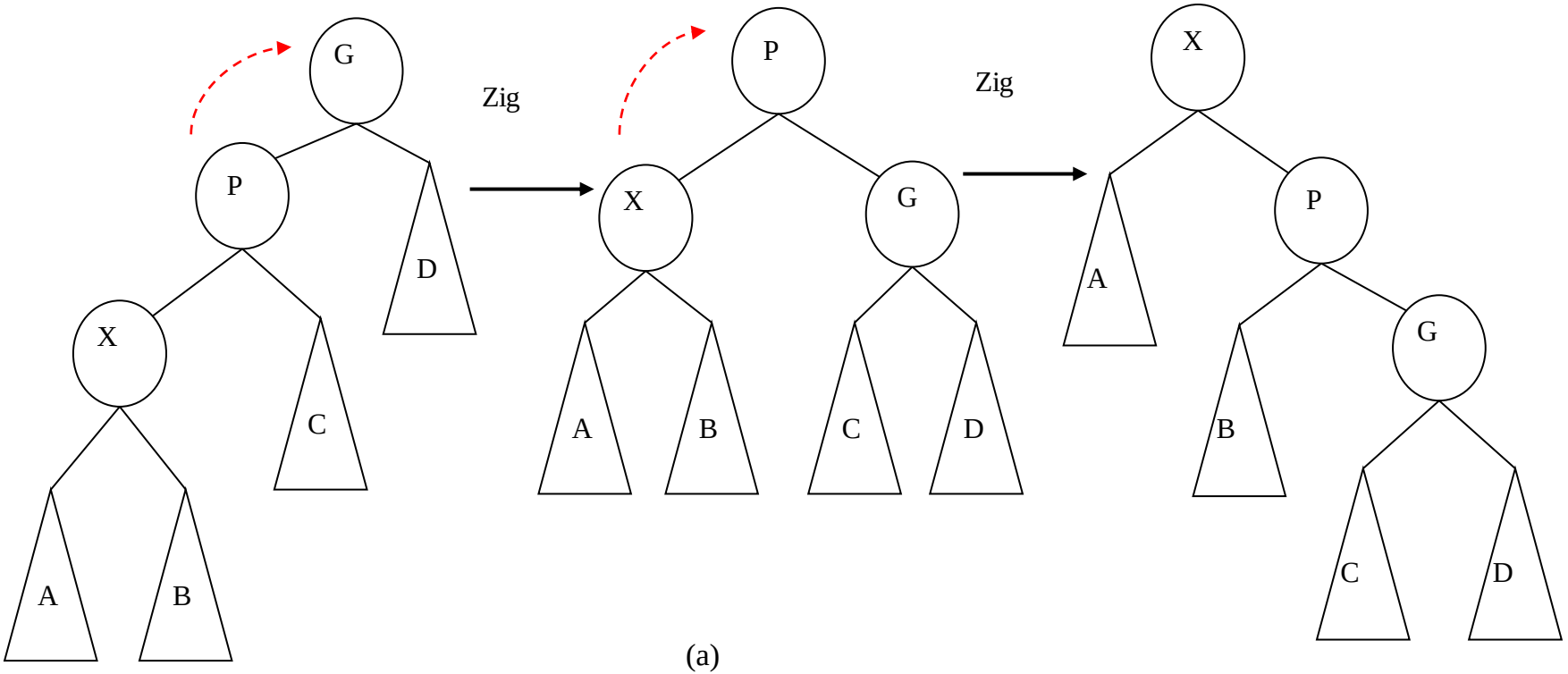
```
typedef int KEY;
struct tree_node {
    KEY key;
    tree_node *right;
    tree_node *left;
    tree_node(KEY k){
        key=k; left=right=NULL;
    }
};
```

Phương pháp hiệu chỉnh (Bottom-up) – Zig(Zag)



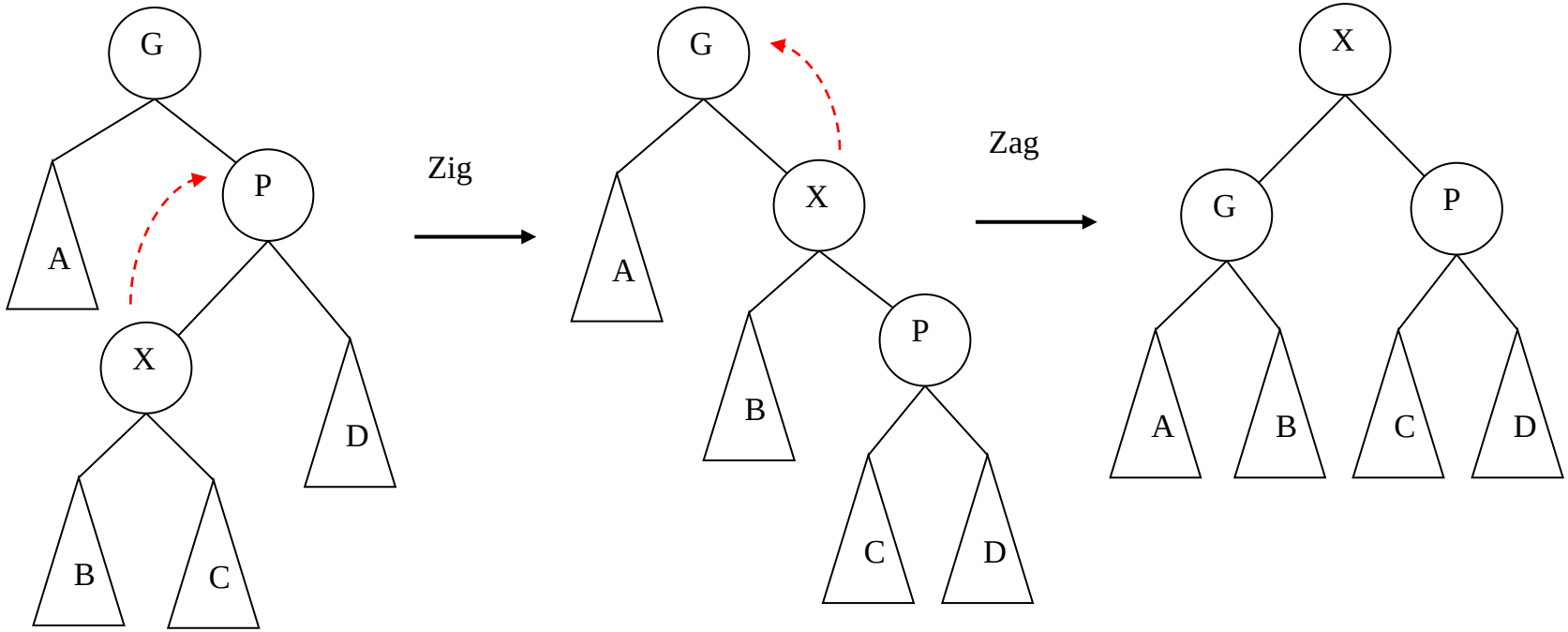
(a)

Phương pháp hiệu chỉnh (Bottom-up) – Zig - Zig(Zag-Zag)



(a)

Phương pháp hiệu chỉnh (Bottom-up) – Zig - Zag(Zag-Zig)



(b)

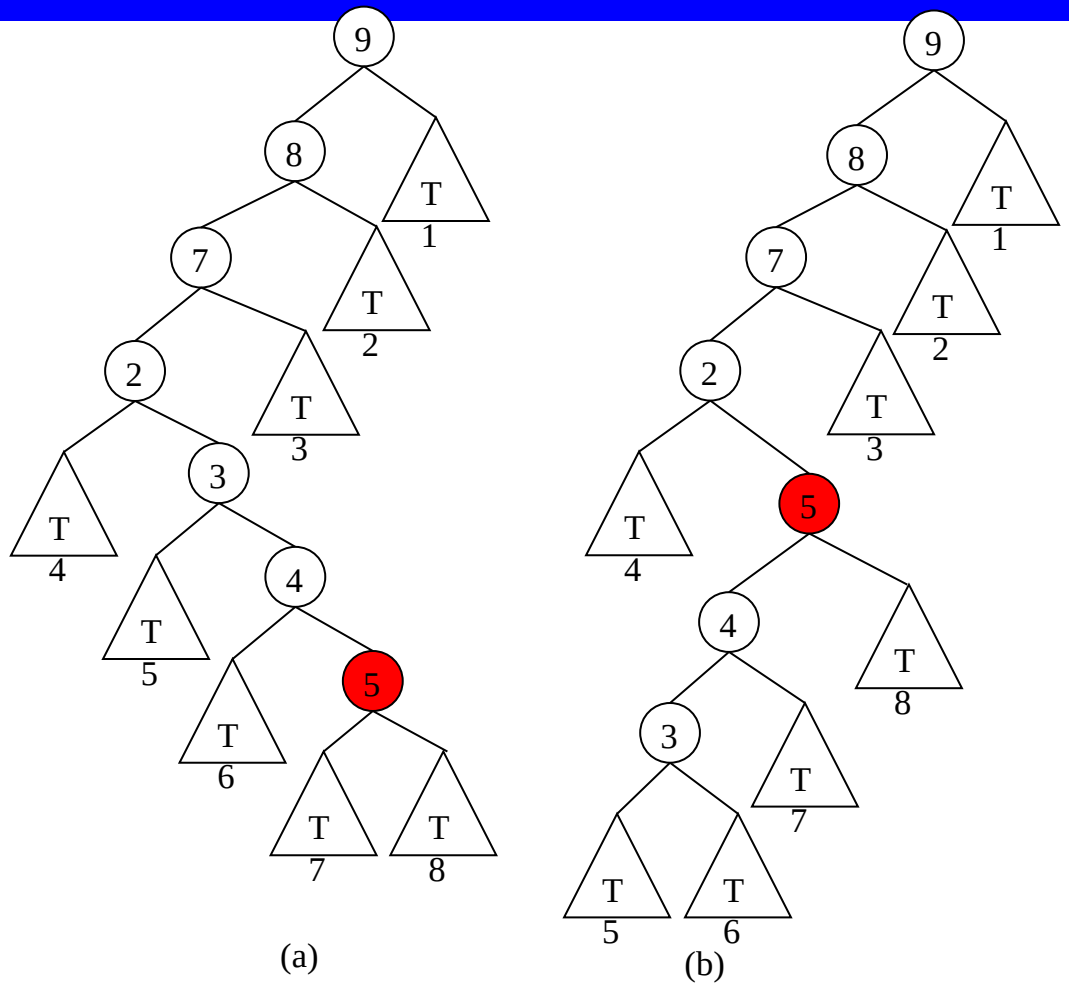
Hình 3-3. ***(a)*** *Quay trái nút X quanh nút P, sau đó quay phải nút X quanh nút G.* ***(b)*** *Quay phải nút X quanh nút P, sau đó quay tría nút X quanh nút G.*

Tìm kiếm

- Thực hiện tìm kiếm khóa k trên splay tree như trên cây tìm kiếm nhị phân.

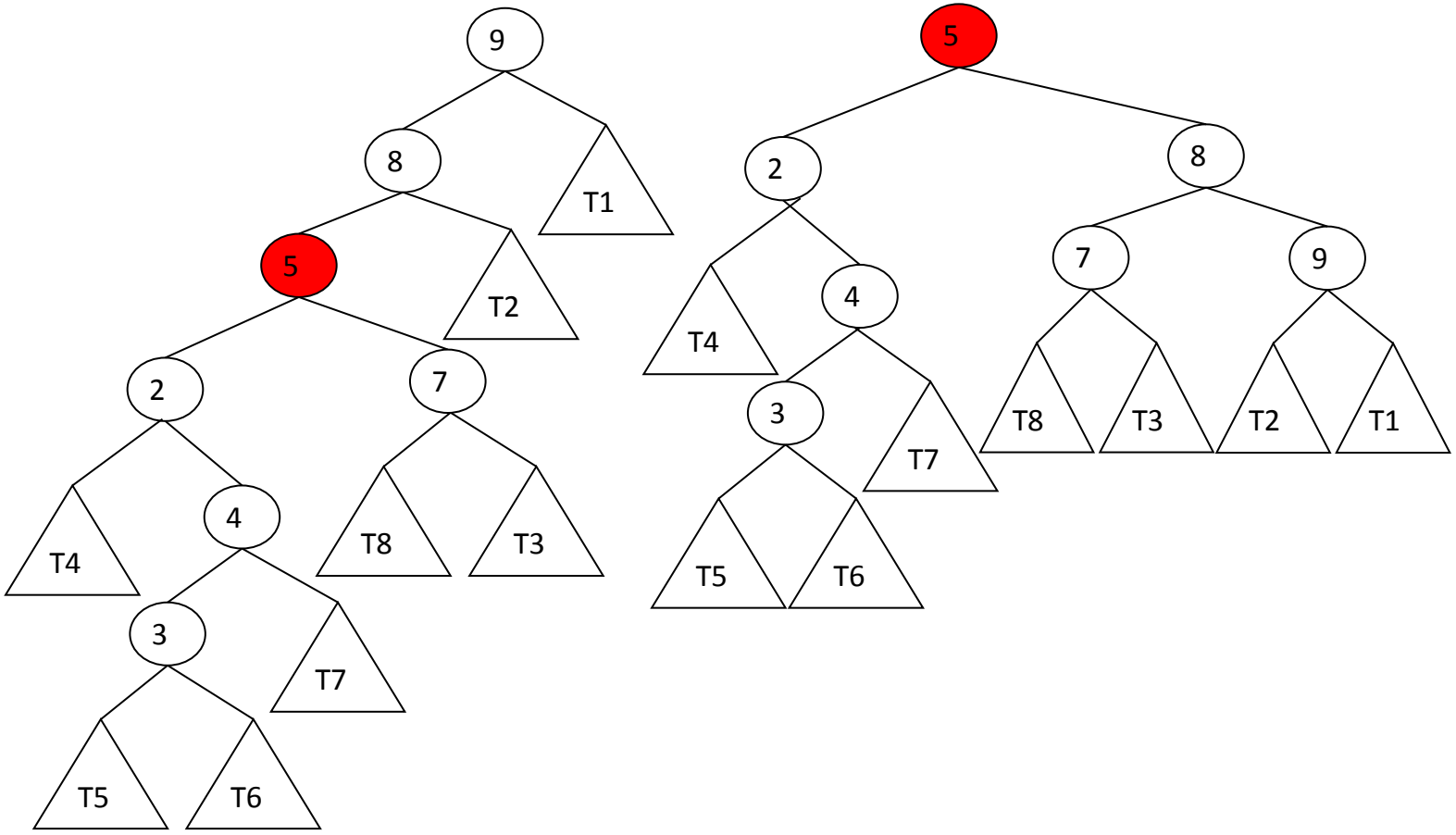
- Nếu tìm kiếm thành công thì quay nút tìm thấy về gốc, ngược lại thì quay nút cuối cùng trên đường tìm kiếm về gốc.

Tìm kiếm



Hình 3-4. ***(a)*** *Cây ban đầu.* ***(b)*** *Cây sau khi thực hiện phép quay trái-trái: quay trái nút 4 quanh nút 3, sau đó lại tiếp tục quay trái nút 5 quanh nút 4.*

Tìm kiếm



Hình 3-4 (tiếp). ***(c)*** *Cây sau khi thực hiện phép quay trái-phải: quay trái nút 5 quanh nút 2, sau đó lại tiếp tục quay phải nút 5 quanh nút 7.* ***(d)*** *Cây sau khi thực hiện phép quay phải-phải: quay phải nút 8 quanh nút 9, sau đó lại tiếp tục quay phải nút 5 quanh nút 8.*

Phép chèn

- Thực hiện chèn một khóa k vào splay tree như trên cây tìm kiếm nhị phân.

- Nếu việc thêm mới thành công thì tiến hành quay nút vừa mới thêm vào về gốc, ngược lại thì quay nút có giá trị trùng với khóa cần thêm về gốc.

Phép xóa

- Thực hiện xóa một khóa k từ splay tree như trên cây tìm kiếm nhị phân. Nếu giá trị khóa cần xóa không có ở trên cây thì quay nút cuối cùng trên đường tìm kiếm về gốc, ngược lại thì quay nút cần xóa về gốc và thực hiện các bước sau:

Bước 1: Tách cây ban đầu thành 3 cây: cây trái L, cây phải R và cây giữa chỉ còn lại một nút gốc chính là nút cần xóa (ký hiệu lại delete_node).

Bước 2: Nếu L rỗng thì R chính là cây mới

Ngược lại:

Quay nút phải cùng của cây con trái về gốc.

Nối cây con phải R vào con phải cùng của L.

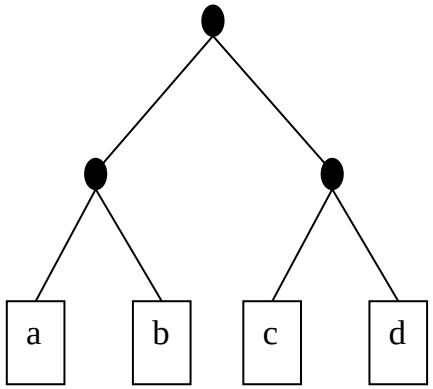
Bước 3: Xóa nút delete_node.

Nén dữ liệu

- Sử dụng splay tree nén xâu ký tự sau: aabcd, biết bảng chữ cái Σ ={a,b,c,d}

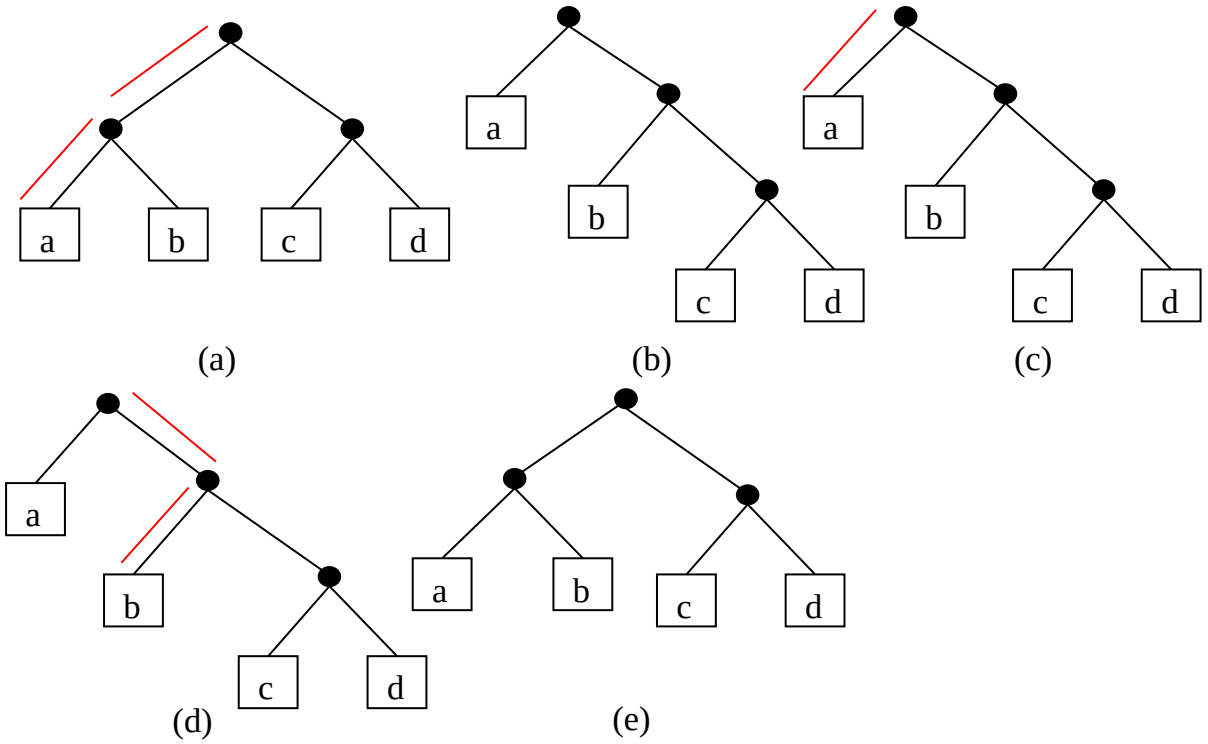
- Bước 1: Xây dựng cây nhị phân mà các chữ cái ở lá của nó.
- Bước 2: Đọc lần lượt từng kí tự và tìm kiếm trên cây.

Nén dữ liệu – Bước 1



Nén dữ liệu – Bước 1

aabcd



Hình 3-. **(a)** *Đọc kí tự a, tìm kiếm trên cây và xây dựng mã, được mã của a là 00, quay nút cha của nút a về gốc được cây như hình (b).* **(c)** *Đọc tiếp được kí tự a, tìm kiếm trên cây và xây dựng mã, được mã của a là 0, cây không thay đổi do nút cha của nút a đang ở gốc.* **(d)** *Đọc tiếp được kí tự b, tìm kiếm trên cây và xây dựng mã, được mã của b là 10, quay nút cha của nút b về gốc được cây như hình (e).*

Giải nén

- Như vậy dãy bít nhận được sau khi nén xâu aabcd là 00010101.
- ***Giải nén***
- Thực hiện ngược với quá trình nén. Giả sử ta có một bảng chữ cái Σ và một dãy bít mã. Để giải nén, ta thực hiện hai bước sau:
- Bước 1: Xây dựng một cây nhị phân đầy đủ hoặc gần đầy đủ chứa tất các chữ cái của Σ ở các nút lá của nó.
- Bước 2: Đọc từng bít trong dãy bít và đi liên tiếp trên các nhánh tương ứng ở trên cây với bít 0 đi xuống bên trái, bít 1 đi xuống bên phải, đến khi gặp nút lá thì ghi chữ cái tương ứng ra văn bản giải nén, quay nút cha của nút lá đó về gốc và tiếp tục đọc cho đến khi hết dãy bít thì kết thúc.