

Chương 4 - Btree

Chúng ta đã biết cây là một cách tiếp cận hoàn chỉnh để tổ chức dữ liệu trong bộ nhớ. Vậy cây có thể làm việc tốt với hệ thống tập tin hay không?

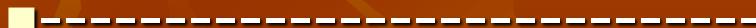
B-tree là cấu trúc dữ liệu phù hợp cho việc lưu trữ ngoài do R.Bayer và E.M.McCreight đưa ra năm 1972.

Để hiểu được B-Tree trước tiên chúng ta định nghĩa Cây nhiều nhánh tìm kiếm(Multiway Search Trees)

Cây nhiều nhánh tìm kiếm

Một cây nhiều nhánh bậc m là cây mà mỗi node có nhiều nhất m cây con. Gọi $count$ ($count \leq m$) là số cây con của một node thì số khoá của node này là $count - 1$ có cấu trúc mảng gồm $count - 1$ phần tử: $key[count - 1]$ được sắp xếp (tăng dần) và thỏa các điều kiện sau:

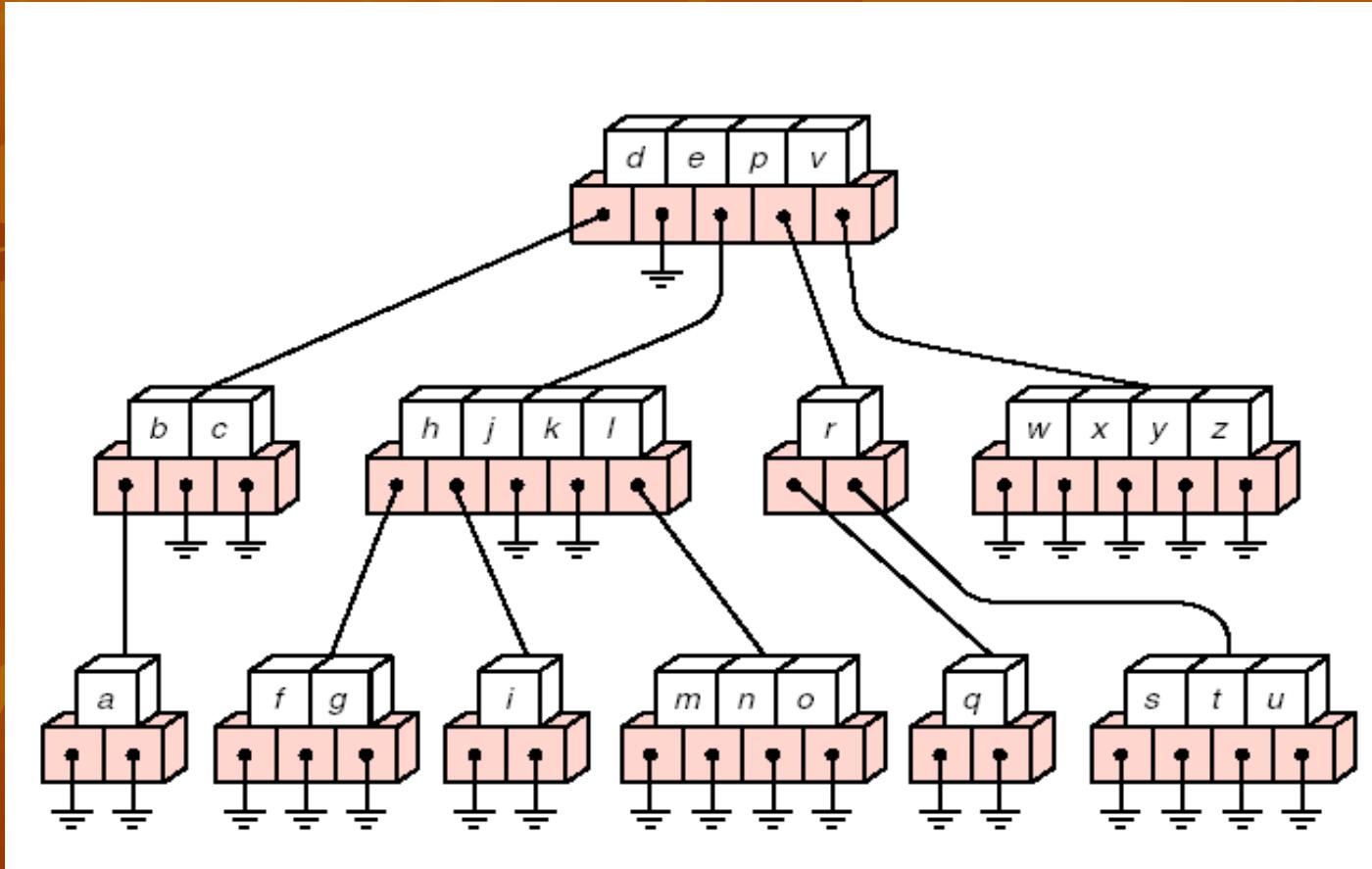
- Tất cả các node con của cây con có gốc tại node con thứ 0 thì có các giá trị khoá nhỏ hơn khoá $key[0]$.
- Tất cả các node con của cây con có gốc tại node con thứ 1 thì có các giá trị khoá lớn hơn khoá $key[0]$ và nhỏ hơn khoá $key[1]$.



Cây nhiều nhánh tìm kiếm

- Tất cả các node con của cây con có gốc tại node con thứ i thì có các giá trị khoá lớn hơn khoá $\text{key}[i-1]$ và nhỏ hơn khoá $\text{key}[i]$ ($0 \leq i \leq \text{count} - 1$).
- Tất cả các node con của cây con có gốc tại node con thứ count thì có các giá trị khoá lớn hơn khoá $\text{key}[\text{count} - 1]$.

Cây nhiều nhánh tìm kiếm



Cây nhiều nhánh tìm kiếm(Multiway Search Trees) bậc 5

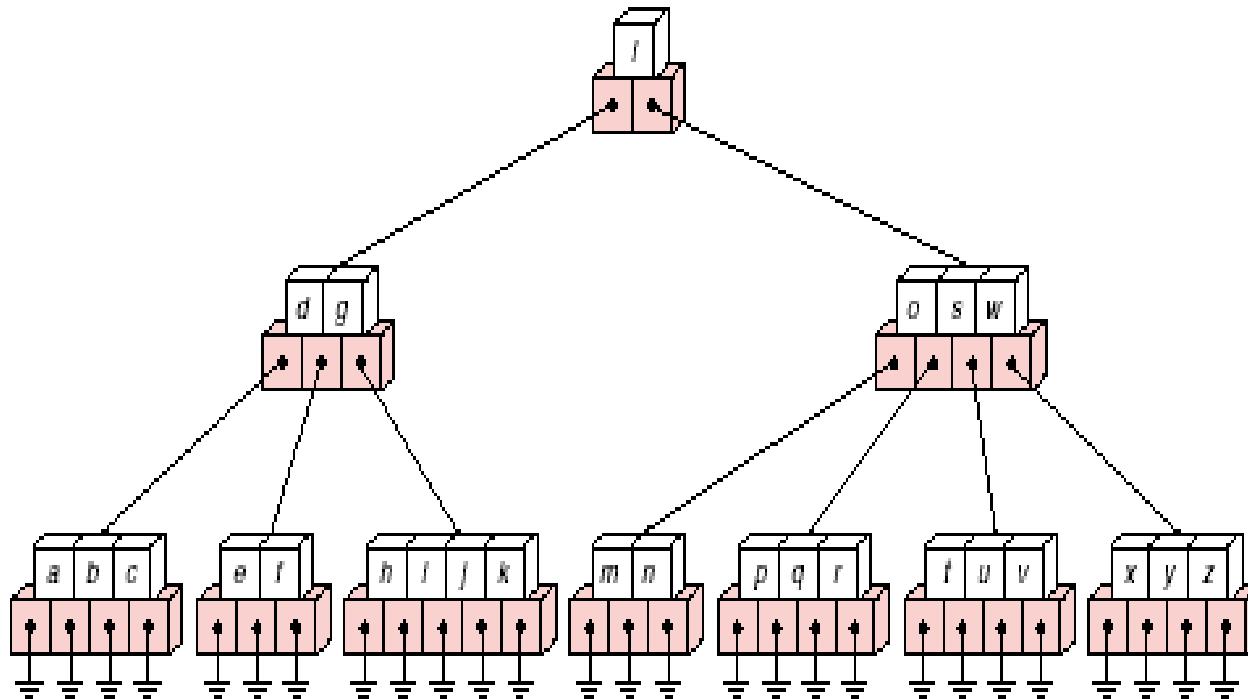
Định nghĩa B-tree

Định nghĩa

Một B-tree bậc m là cây nhiều nhánh tìm kiếm thỏa các điều kiện sau:

- (i) Tất cả các node lá là cùng mức.
- (ii) Tất cả các node trung gian (trừ node gốc) có nhiều nhất m cây con và có ít nhất $m/2$ ($m \text{ div } 2$) cây con (khác rỗng).
- (iii) Mỗi node hoặc là node lá hoặc có $k+1$ cây con (k là số khoá của node này)
- (iv) Node gốc có nhiều nhất m cây con hoặc có thể có 2 cây con (Node gốc có 1 khoá và không phải là node lá) hoặc không chứa cây con nào (node gốc có 1 khoá và cũng là node lá).

Định nghĩa B-tree



B-tree bậc 5 có 3 mức

Khai báo

Khai báo:

```
typedef struct
```

```
{
```

```
    int count; // số khoá của node hiện hành
```

```
    int Key[Order-1]; // mảng lưu trữ các khoá của node
```

```
    int *Branch[Order]; /* các con trỏ chỉ đến các  
cây con, Order-Bậc của cây*/
```

```
} BNode; //Kiểu dữ liệu của node
```

```
typedef struct BNode *pBNode // con trỏ node
```

```
pBNode Root // con trỏ node gốc
```

Các Phép toán

$C_0, K_1, C_2, K_2, \dots, C_{m-1}, K_m, C_m$

Xét trong hình trên, khóa cần tìm là X . Giả sử nội dung của node nằm trong bộ nhớ. Với m đủ lớn ta sử dụng phương pháp tìm kiếm nhị phân, nếu m nhỏ ta sử dụng phương pháp tìm kiếm tuần tự. Nếu X không tìm thấy sẽ có 3 trường hợp sau xảy ra:

- $K_i < X < K_{i+1}$. Tiếp tục tìm kiếm trên cây con C_i
- $K_m < X$. Tiếp tục tìm kiếm trên C_m
- $X < K_1$. Tiếp tục tìm kiếm trên C_0

Quá trình này tiếp tục cho đến khi node được tìm thấy. Nếu đã đi đến node lá mà vẫn không tìm thấy khoá, việc tìm kiếm là thất bại.

Các Phép toán (*tt*)

Phép toán nodesearch :

Trả về vị trí nhỏ nhất của khóa trong nút current bắt đầu lớn hơn hay bằng k. Trường hợp k lớn hơn tất cả các khóa trong nút current thì trả về vị trí current->count

```
int nodesearch (pBNode current, int k)
{
    int i;
    for(i=0;i<current->count && current-
        >key[i] < k; i++);
    return (i);
}
```

Các Phép toán (*tt*)

Tìm khóa k trên B-Tree. Con trỏ current xuất phát từ gốc và đi xuống các nhánh cây con phù hợp để tìm khóa k có trong một nút current hay không

Nếu có khóa k tại nút current trên cây:

- Biến found tra về giá trị TRUE
- Hàm search() trả về con trỏ chỉ nút current có chứa khóa k
- Biến position trả về vị trí của khóa k có trong nút current này

Các Phép toán (*tt*)

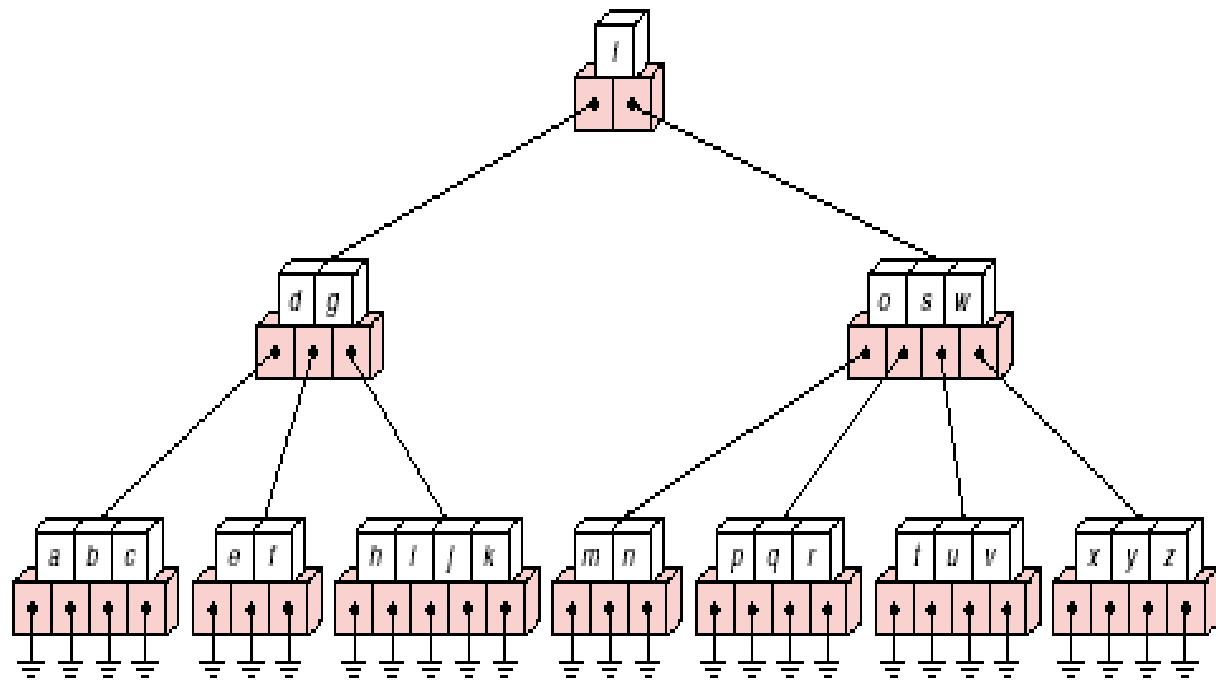
Nếu không có khóa k trên cây:

- Lúc này current=NULL và q(nút cha của current) chỉ nút lá có thể thêm khóa k vào nút này được.
- Biến found trả về giá trị FALSE
- Hàm search() trả về con trỏ q là nút lá có thêm nút k vào
- Biến position trả về vị trí có thể chèn khóa k vào nút lá q này

```

C
pBNode search(int k,
{
    int i;
    pBNode current;
    q = NULL;
    current = Root;
    while (current != NULL)
    {
        i = nodesearch(current, k);
        if(i< current->

```



thay

```

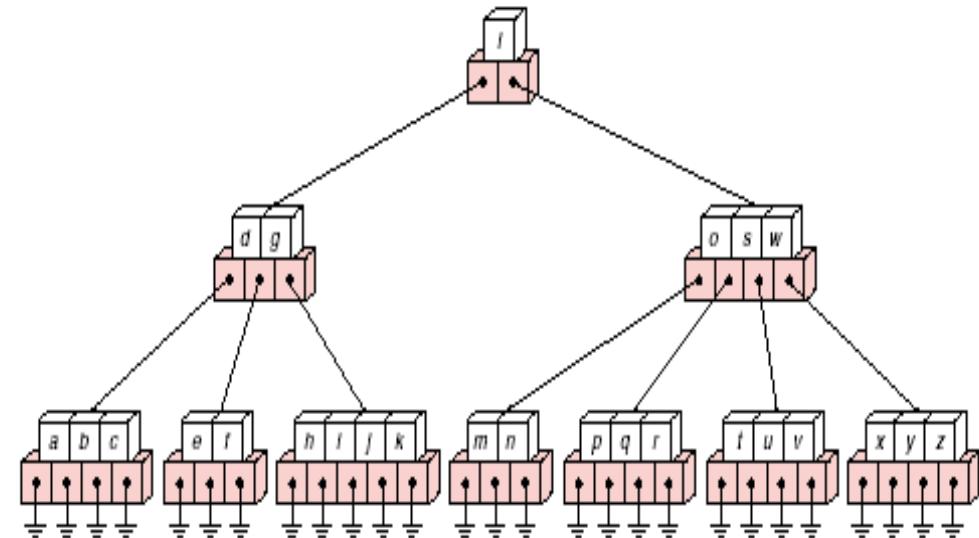
    {
        found = TRUE;
        position = i; // vi trí tìm thay khoa k
        return(current); // node có chứa khoa k
    }
}

```

Các Phép toán (*tt*)

```
q = current;  
current = current ->Branch[i];  
}  
/*Khi thoat khoi vong lap tren la khong tim thay, luc nay  
current=NULL, q la node la co the them khoa k vao  
node nay, position la vi tri  
found = FALSE;  
position = i;  
return (q); //tra ve  
}//end search()
```

Truong Hai



Duyệt cây

```
void traverse(pBNode proot)
{
    if(proot == NULL)      return; //dieu kien dung
    else                  // de qui
        {/* vong lap duyet nhanh cay con Branch[i] va in khoa key[i] cua
           node proor*/}
    for(i = 0; i < proot -> count; i++)
    {
        traverse (proot ->Branch[i]);
        printf ("%8d", proot -> key[i]);
    }
    traverse (proot -> Branch[proot -> count]); //duyet nhanh cay
                                                //con cuoi cung cua node proot
}
}
```

Thêm node mới

Quá trình thêm một khoá mới(newkey) vào B-tree có thể được mô tả như sau:

- Tìm node newkey nếu có trên cây thì kết thúc công việc này tại node lá (không thêm vào nữa)
- Thêm newkey vào node lá, nếu chưa đầy thì thực hiện thêm vào và kết thúc

Node đầy là node có số khoá = (bậc của cây)-1

Thêm node mới(tt)

- Khi node được thêm vào bị đầy, node này sẽ được tách thành 2 node cùng mức, khoá median sẽ được đưa vào node mới
- Khi tách node, khoá median sẽ được dời lên node cha, quá trình này có thể lan truyền đến node gốc
- Trong trường hợp node gốc bị đầy, node gốc sẽ bị tách và dẫn đến việc tăng trưởng chiều cao của cây.

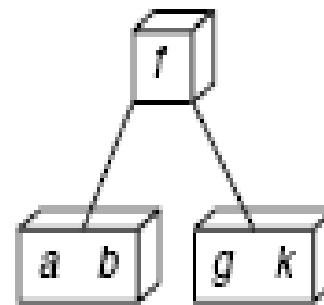
1.

a, g, f, b:



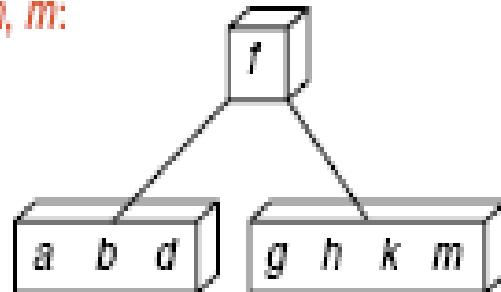
2.

k:



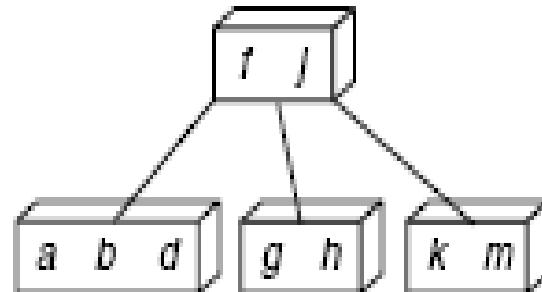
3.

d, h, m:



4.

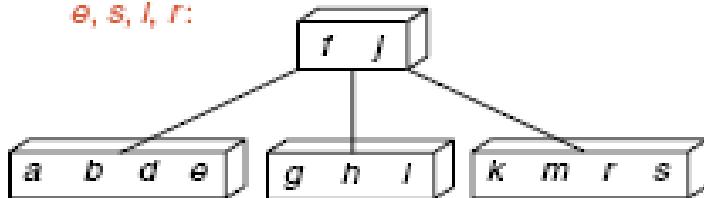
f:



Thêm node mới(tt)

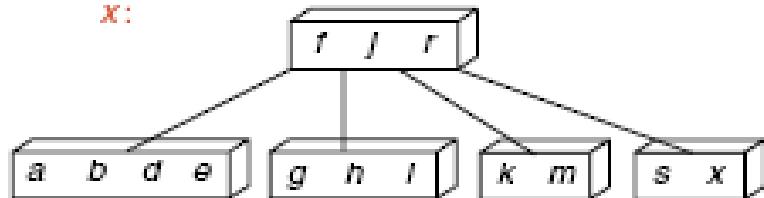
5.

$e, s, l, r:$



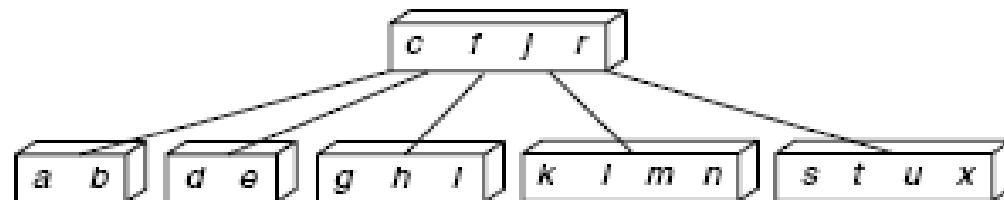
6.

$x:$



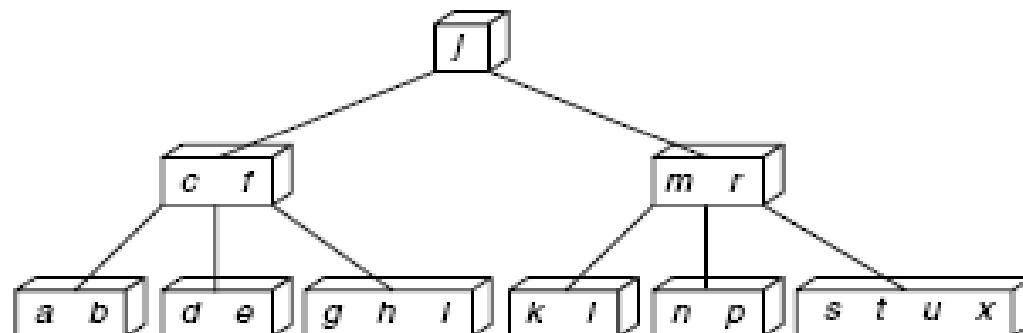
7.

$c, l, n, t, u:$



8.

$p:$



Thêm node mới(*tt*)

Khi thêm một khóa vào B-Tree chúng ta có thể viết như sau:

```
// truong hop B-Tree bi rong khi tao node goc  
if(Root == NULL)  
Root = makeroot(k);  
else  
{  
    s = search(k, &position, &timthay);  
    if (timthay) cout<<“Không thêm vào được”;  
    else insert (s, k, position);  
}
```

Thêm node mới(*tt*)

Thêm khóa k vào vị trí position của nút lá s (s và position do phép toán search() trả về)

Nếu nút lá s chưa đầy: gọi phép toán insnode để chèn khóa k vào nút s

Nếu nút lá s đã đầy: tách nút lá này thành 2 nút nửa trái và nửa phải

void insert (pBNode s, int k, int position)

Thêm node mới(*tt*)

```
void insert (pBNode s, int f, int position)
```

```
{pBNode current, right_half, P, extra_branch;
```

```
    int pos, extra_entry, median;
```

```
    //khoi nong cac tri truoc khi va  
node day current
```

```
    current = s;
```

```
    extra_entry = f;
```

```
    extra_branch = NULL; // vi cu
```

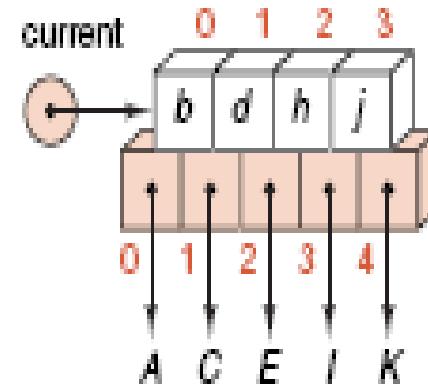
```
    // extra_branch la NULL
```

```
    pos = position;
```

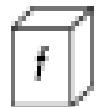
```
    p = father (current);
```

```
    // Vong lap tach cac node day c
```

Position=2,order=5



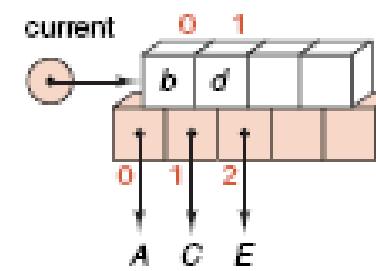
extra_entry



extra_branch



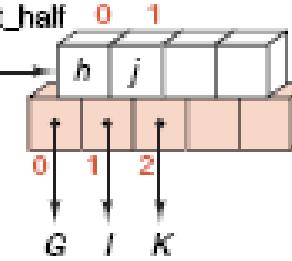
G



median



right_half



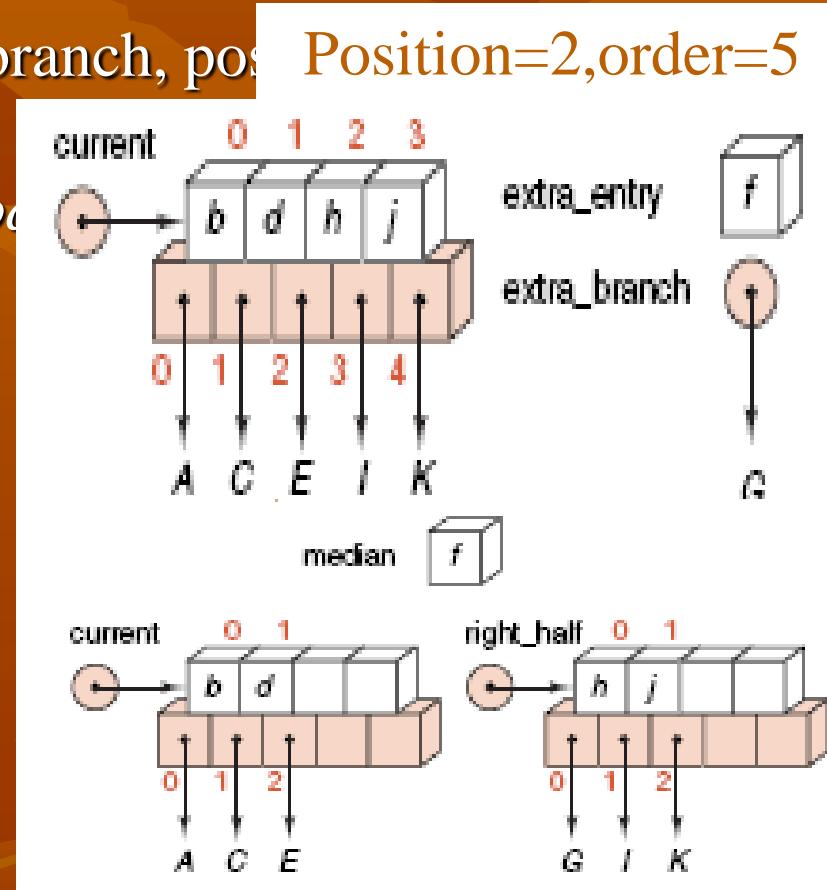
G

Thêm node mới(*tt*)

```
while (p != NULL && current -> count == Order)
```

```
{
```

```
    split(current, extra_entry, extra_branch, pos, Position=2,order=5  
          right_half, median);  
    // Gan lai cac tri sau lan tach node  
    current = p;  
    extra_entry = median;  
    extra_branch = right_half;  
    pos = nodesearch (p, median);  
    p = father (current);
```



Thêm node mới(*tt*)

```
// Trường hợp node current chưa day và current không phải là node gốc  
if(current -> count+1 < Order)  
{  
    // Chèn extra_entry và extra_branch tại vị trí position của node current  
    insnode (current, extra_entry, extra_branch, pos);  
    return;  
}  
// Trường hợp node current là node gốc bị day, tách node gốc này và tạo node  
// gốc mới  
split (current, extra_entry, extra_branch, pos, right_half, median);  
Root = makeroot (median);           // tạo node gốc mới  
// Gắn lại hai nhánh cay con của node gốc mới là current và right_half  
Root -> Branch[0] = current;  
Root -> Branch[1] = right_half;  
}
```

Tách node

Tách node đầy current, phép toán này được gọi bởi phép toán Insert

current là nút đầy bị tách, sau khi tách xong nút current chỉ còn lại một nửa số khóa bên trái

extra_entry, extra_branch và position là khóa mới, nhánh cây con và vị trí chèn vào nút current

Nút right_half là nút nửa phải có được sau lần tách, nút right_half chiếm một nửa số khóa bên phải

Median là khóa ngay chính giữa sẽ được chèn vào nút cha

Tách node

```
void split (pBNode current, int extra_entry, pBNode extra_branch)
pBNode &right_half, int &median)
```

```
{
```

```
    pBNode p;
```

```
    p = new pBNode; //cap phat node nua phai
```

*/*truong hop chen extra_entry va extra_branch vao node
if(position > Order/2)*

```
{
```

```
    copy(current, Order/2+1, Order - 2, p);
```

insnode (P , extra_entry, extra_branch, position- Order/2 -1)

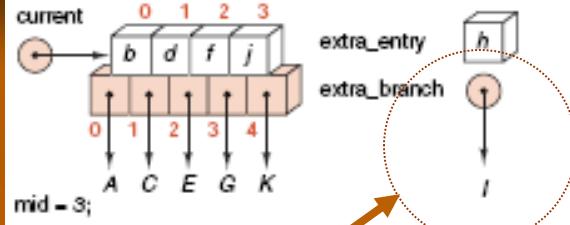
*current->numtrees = Order/2+1; /*so nhanh cay con con lai cua node*

```
    median = current -> key[Order/2];
```

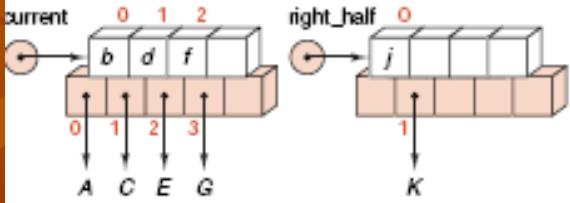
```
    right_half = p ;
```

```
    return;
```

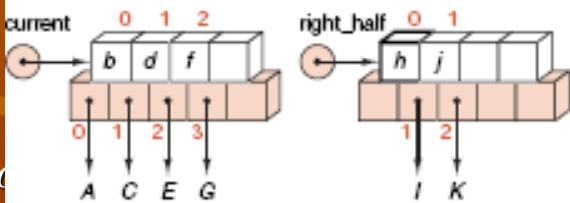
```
}
```



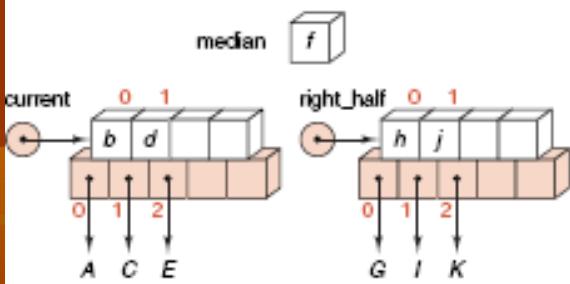
Position=3, Order=5



Position=3, Order=5



Position=3, Order=5



Tách node

```
// truong hop extra_entry la median
if(position == Order/2)
{
    copy(current, Order/2, Order-2, p);
    current->numtrees = Order/2+1; /*so nhanh cay con con
lai cua node nua trai*/
    /*Dieu chinh lai node con dau tien cua node nua
phai*/
    current -> Branch[0] = extra_branch;
    median = current -> key[Order/2];
    right_half = p;
    return;
}
```

Tách node

```
/* Truong hop chen extra_entry va extra_branch vao node nua  
trai*/  
if(position < Order/2)  
{  
    copy(current, Order/2, Order-2, p);  
    current->numtrees = Order/2; /*so nhanh cay con con lai cua  
node nua trai*/  
    median = current -> key[Order/2- 1];  
    insnode(current, extra_entry, extra_branch, position);  
    right_half = p;  
    return;  
}  
}
```

Thêm vào node lá

```
void insnode (pBNode current, int extra_entry,  
pBNode extra_branch, int position)
```

```
{
```

```
    int i;
```

```
/*doi cac nhanh cay con va cac khoa tu vi tri  
position tro ve sau xuong mot vi tri*/
```

```
for(i = current->count; i >= position+1; i--)
```

```
{
```

```
    current -> Branch[i+1] = current ->  
Branch[i];
```

```
    current -> key[i] = current -> key[i - 1];
```

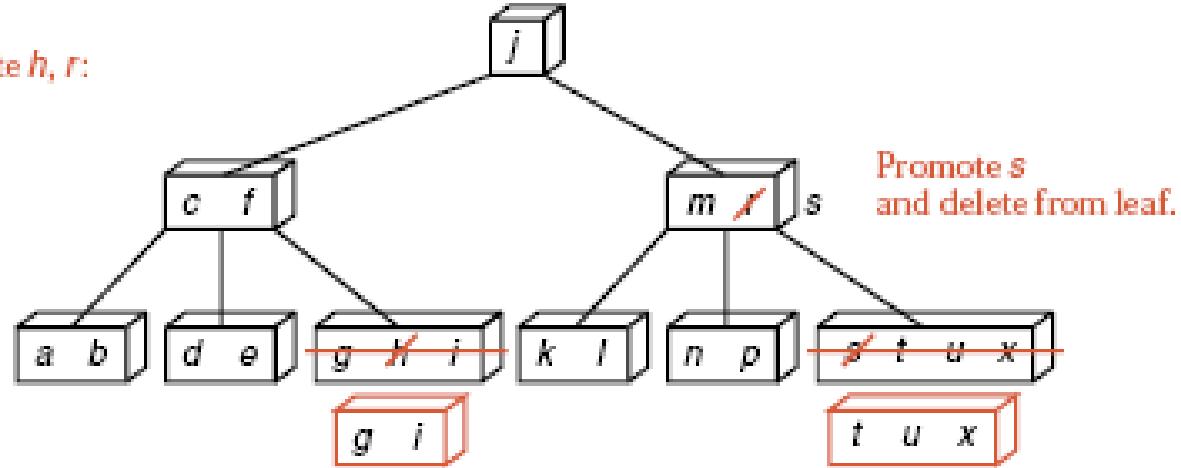
```
}
```

Thêm vào node lá (tt)

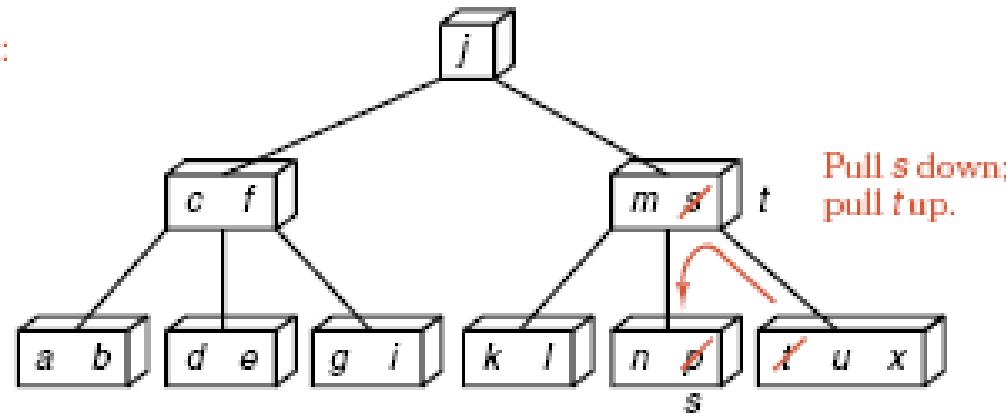
```
// gan khoa extra_entry vao vi tri position  
current -> key[position] = extra_entry;  
  
// Gan nhanh extra_branch la nhanh cay con ben  
phai cua khoa extra_entry  
current -> Branch[position + 1] = extra_branch;  
  
// tang so khoa cua node current len 1  
current -> count +=1;  
}
```

Loại bỏ (tt)

1. Delete h, r :

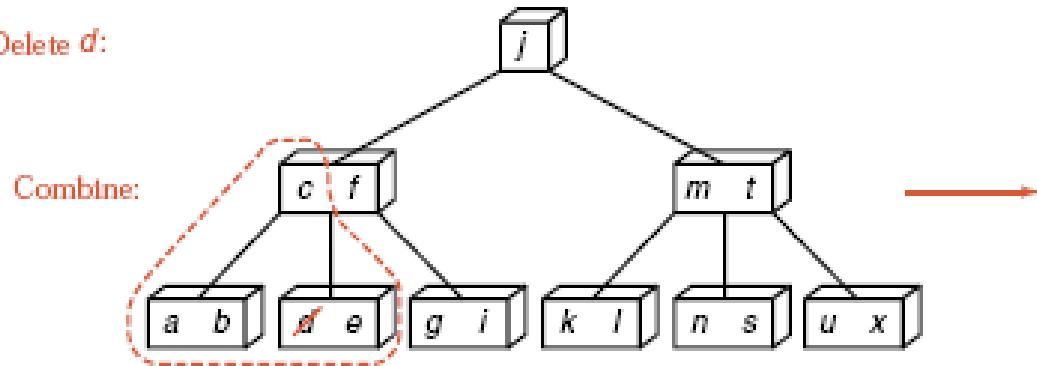


2. Delete p :

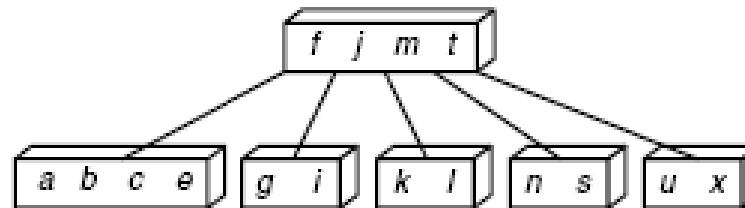
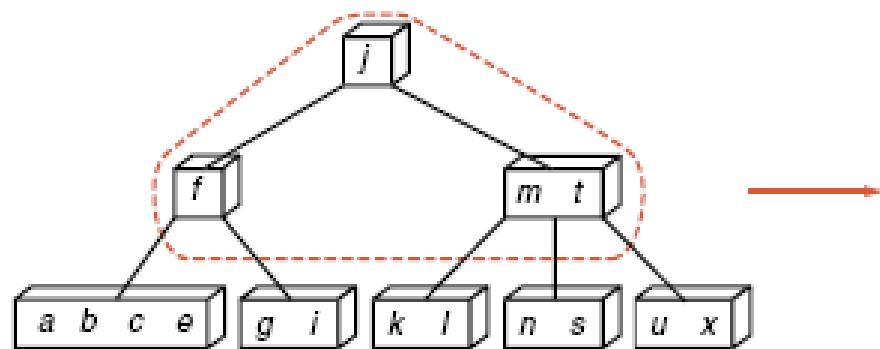


Loại bỏ (*tt*)

3. Delete *d*:



Combine:



Loại bỏ (tt)

pBNode remove(const int &key)

```
/*+++++Kết quả: Nếu tìm thấy khóa key sẽ xoá khóa này và trả về địa chỉ node chứa
key, nếu không tìm thấy trả về NULL
Sử dụng hàm đệ quy recursive_remove()
+++++*/
{
    PBNode result = recursive_remove(root, key);
    if (root != NULL && root->count == 0)
    { // root is now empty.
        pDNode old_root = root;
        root = root->branch[0];
        delete old_root;
    }
    return result;
}
```

Loại bỏ (tt)

pBNode recursive_remove(pBNode current, const int &key)

/*+++++ccurrent hoặc rỗng hoặc là con trỏ đến cây con.

Hàm này Thực hiện xoá key nếu tìm thấy khoá key, Ngược lại trả về NULL

Sử dụng các hàm **nodeSearch()**, **copy_in_predecessor()**,
recursive_remove (recursively), **remove_data()**và
restore().

```
+++++*/  
{  
int position;  
PBNode result;  
if (current == NULL) return NULL;  
else
```

Loại bỏ (*tt*)

```
{  
if (search(current, key, position))  
{  
    // Tìm thấy key trong node current.  
    if (current->branch[position] != NULL)  
    { // không phải là node lá  
        copy_in_predecessor(current, position);  
        result=recursive_remove(current->branch[position],current->data[position]);  
    }  
    else remove_data(current, position);//Xoá tại node lá  
    }  
    else result = recursive_remove(current->branch[position], key);  
    if (current->branch[position] != NULL)  
    if (current->branch[position]->count <(order - 1)/2)  
        restore(current, position);  
    }  
    return result;  
}
```

Loại bỏ (tt)

void remove_data(pBNode current,int position)

```
/*+++++  
Xóa tại vị trí position trên node lá current.  
*/  
{  
    for (int i = position; i < current->count - 1; i++)  
        current->data[i] = current->data[i + 1];  
    current->count --;  
}
```

Loại bỏ (tt)

```
void copy_in_predecessor(pBNode current, int position)
/*+++++
+++++
Tại vị trí position trên node current(current không phải là lá ) sẽ
được thay bởi khoá key cực phải của cây con bên trái của current
+++++
+++++*/
{
    pBNode leaf = current->branch[position];
    while (leaf->branch[leaf->count ] != NULL)
        leaf = leaf->branch[leaf->count ];
    current->data[position]=leaf->data[leaf->count-1];
}
```

Loại bỏ (tt)

```
void restore(pBNode current,int position)
```

```
/*+++++*/*
```

Sử dụng các hàm: **move_left**, **move_right**, **combine**.

```
*/
```

```
{
```

```
if(position==current->count)
```

//Trường hợp nhánh cực phải

```
if(current->branch[position-1]->count>(order- )/ 2)
```

```
    move_right(current, position - 1);
```

```
else
```

```
    combine(current, position);
```

```
else if (position == 0)
```

Loại bỏ (*tt*)

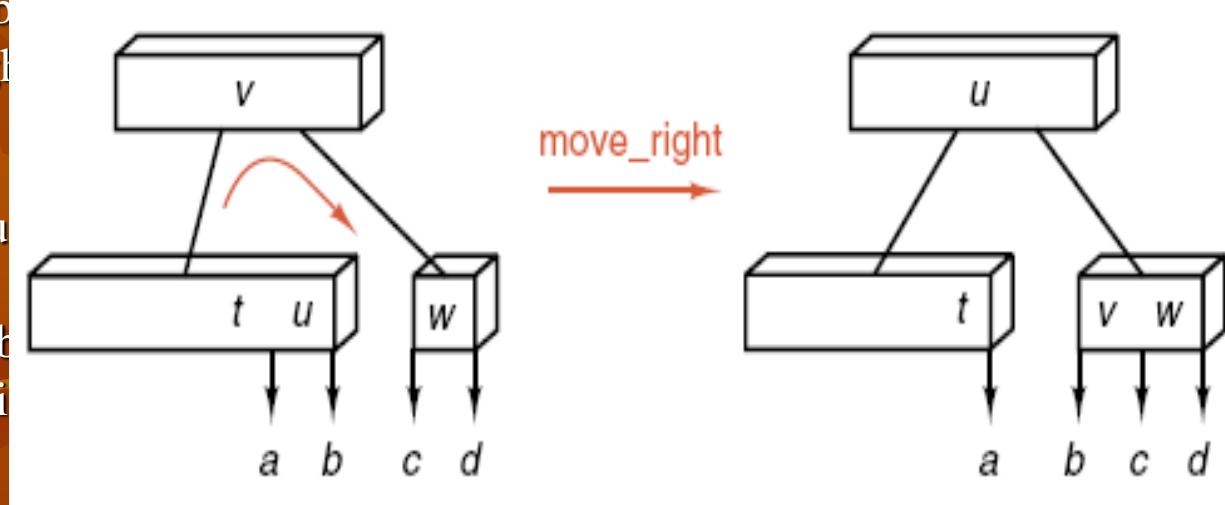
```
// Trường hợp nhánh cực trái
if (current->branch[1]->count < (order - 1) / 2)
    move_left(current, 1);
else
    combine(current, 1);
else //các trường hợp khác: nhánh giữa
if(current->branch[position-1]->count>(order-1)/ 2)
    move_right(current, position - 1);
else if(current->branch[position+1]->count>(order- 1) / 2)
    move_left(current, position + 1);
else
    combine(current, position);
}
```

Loại bỏ (*tt*)

```
void move_left(pBNode current,int position)
/*+++++*+/+*/{pBNode left_branch = current->branch[position - 1];
pBNode right_branch = current->branch[position];
left_branch->data[left_branch->count] = current->data[position - 1];
left_branch->branch[++left_branch->count] = right_branch->branch[0];
current->data[position - 1] = right_branch->data[0];    right_branch->count--;
for (int i = 0; i < right_branch->count ; i++)
{
right_branch->data[i]=right_branch->data[i + 1];
right_branch->branch[i]=right_branch->branch[i+1];
}
right_branch->branch[right_branch->count] =
right_branch->branch[right_branch->count + 1];
}
```

Loại bỏ (*tt*)

```
void move_right(pBNode current,int position)
{
    pBNode right_branch = current->branch[position + 1];
    pBNode left_branch = current->branch[position];
    right_branch->branch[right_b
        right_branch->branch[rig
for (int i = right_branch->cou
{
    right_branch->data[i]=right_b
        right_branch->branch[i]=ri
}
    right_branch->count++;
    right_branch->data[0] = current->data[position];  right_branch-
>branch[0]=left_branch->branch[left_branch->count--];
current->data[position]= left_branch->data[left_branch-
```

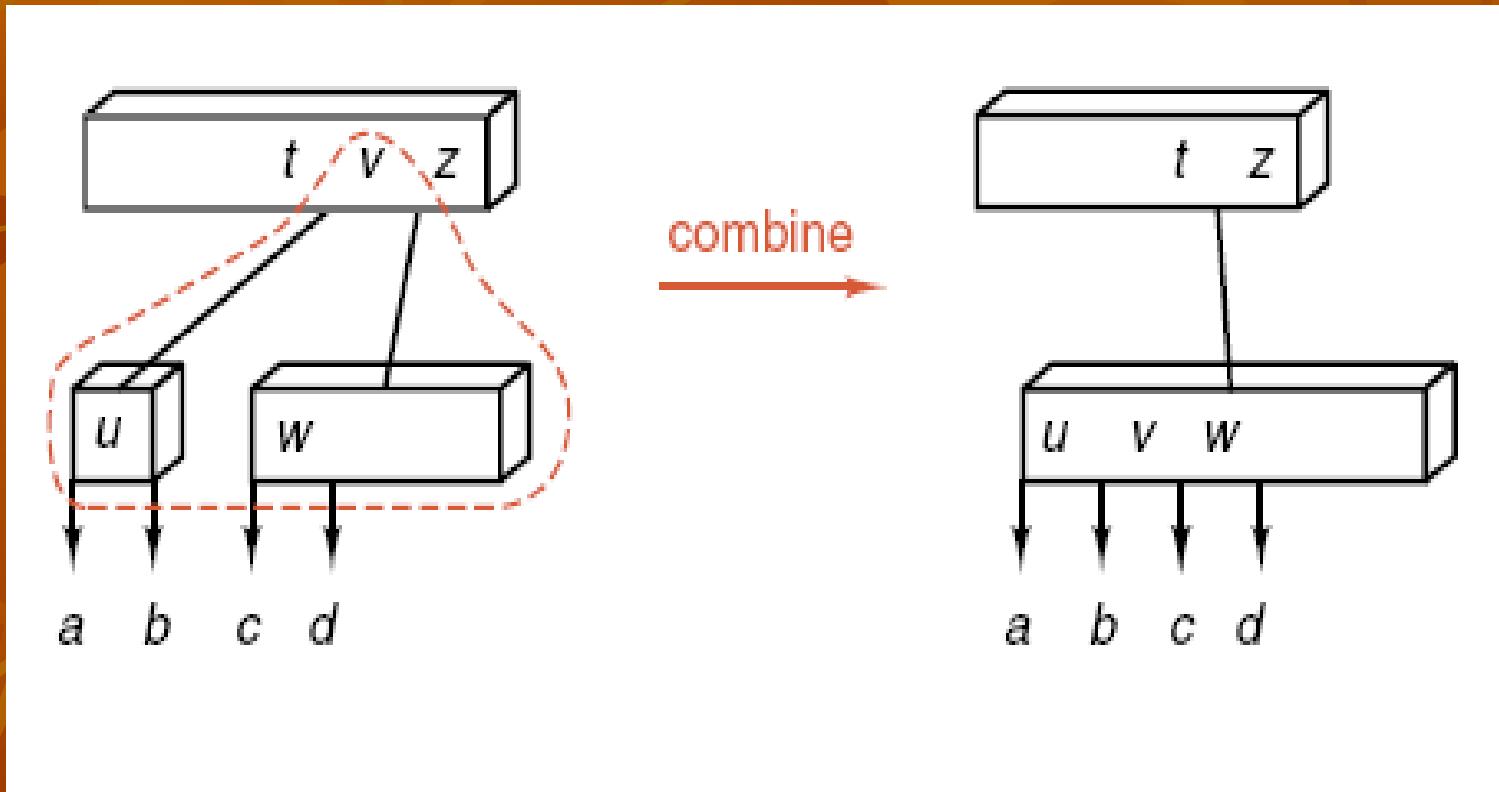


Loại bỏ- (*tt*)

void combine(pBNode current,int position)

```
{  
    int i;  
    pBNode left_branch = current->branch[position - 1];  
    pBNode right_branch = current->branch[position];  
    left_branch->data[left_branch->count] = current-  
    >data[position - 1];  
    left_branch->branch[++left_branch->count] =  
    right_branch->branch[0];
```

Loại bỏ (*tt*)



Loại bỏ (*tt*)

```
for (i = 0; i < right_branch->count ; i++)
{
    left_branch->data[left_branch->count]= right_branch->data[i];
    left_branch->branch[++left_branch->count ] =
right_branch->branch[i + 1];
}
current->count --;
for (i = position - 1; i < current->count ; i++)
{
    current->data[i] = current->data[i + 1];
    current->branch[i + 1] = current->branch[i + 2];
}
delete right_branch;
}
```