CÂY B-TREE

Giới thiệu

- 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.

Cây nhiều nhánh tìm kiếm

Một cây nhiều nhánh bậc m là cây mà mỗi node có nhiều nhất m cây con.

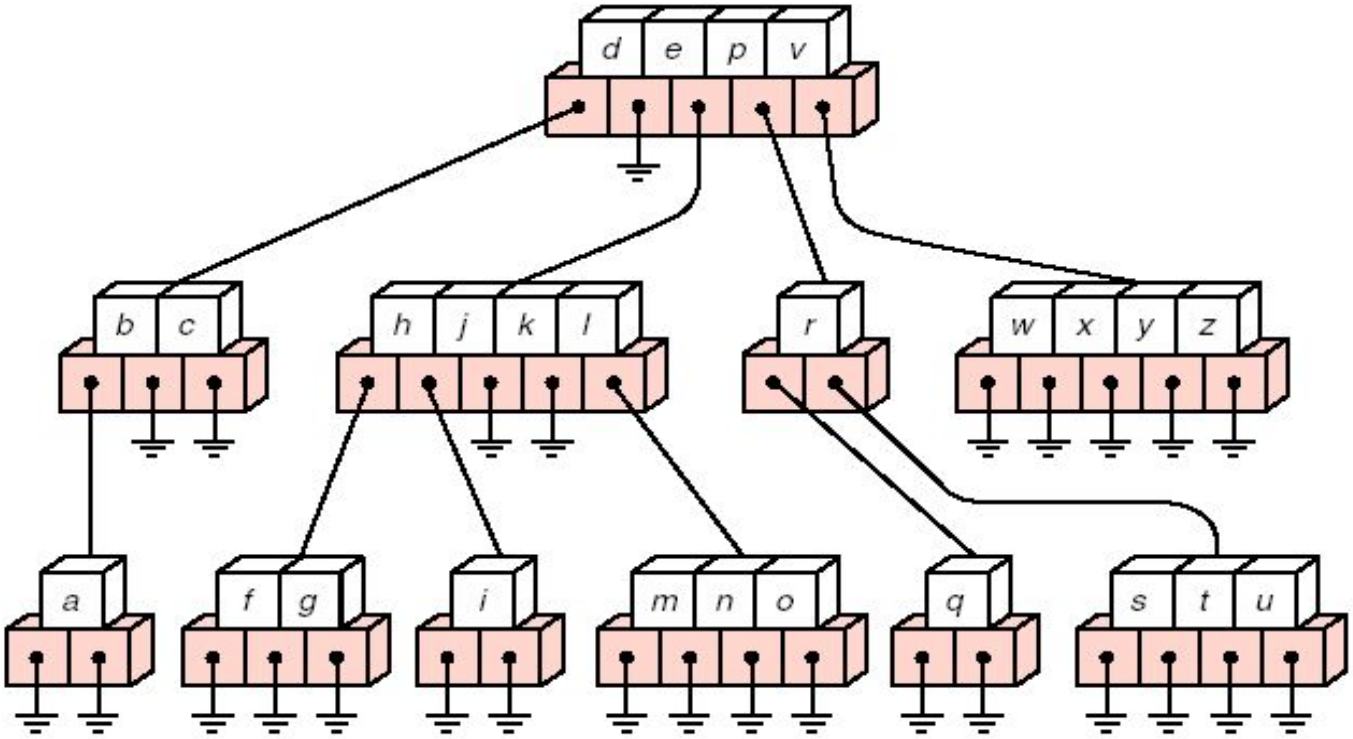
Gọi count (count <=m) là số cây con của một node, số khoá của node này là count -1 có cấu trúc mảng gồm count -1 phần tử key[count -1] được sắp xếp (tăng dần) và thỏa các điều kiện sau:

- 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 key[i-1] và nhỏ hơn khoá key[i] (0<=i<=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 key[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:

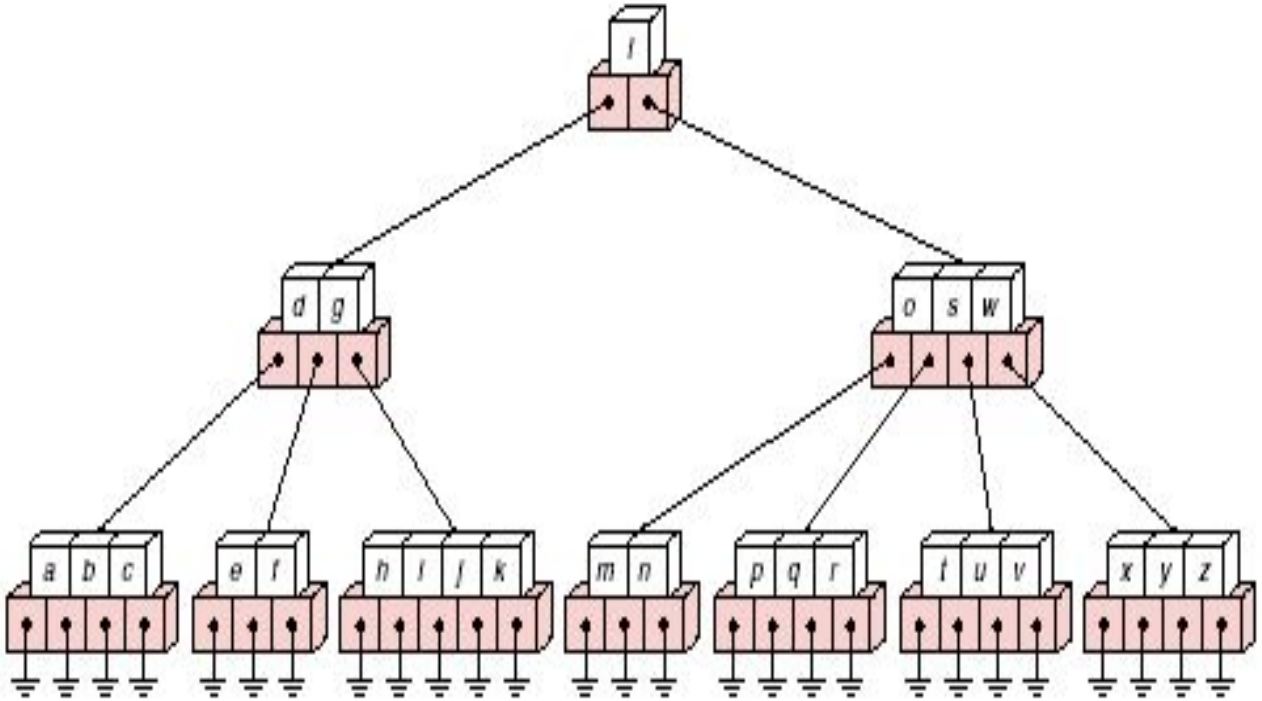
(i) Tất cả các node lá cùng mức.

(ii) Tất cả các node trung gian (trừ node gốc) có nhiều nhất m cây con và có ít nhất m/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



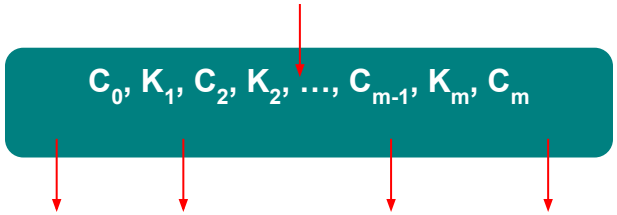
B-tree bậc 5 có 3 mức

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 tro node goc
```


Các Phép toán



Các trường hợp xảy ra khi tìm 1 node X. Nếu X không tìm thấy sẽ có 3 trường hợp sau xảy ra:

- $K_i < X < K_{i+1}$. Tiếp tục tìm kiếm trân 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

Phép toán tìm kiếm

Trả về vị trí nhỏ nhất của khóa trong nút hiện tại bắt đầu lớn hơn hay bằng k.

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

Các Phép toán

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 trả 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

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

Các Phép toán

```
pBNode search(int k, int &position, int &found)
{
    int i;
    pBNode current, q;
    q = NULL; current = Root;
    while (current !=NULL){
        i = nodesearch (current, k);
        if(i< current->count && k == current->key[i]) {
            found = TRUE;
            position = i; // vi trí tìm thấy khóa k
            return(current);
        }
        q = current; current = current ->Branch[i];
    }
    found = FALSE;      position = i;
    return q;
}
```

Duyệt cây

```
void Traverse(pBNode proot)
{
    if (proot == NULL)    return;
    else
    {
        for(i = 0; i < proot -> count; i++)
        {
            traverse (proot ->Branch[i]);
            printf ("%8d", proot -> key[i]);
        }
        traverse (proot -> Branch[proot -> count]);    }
}
```


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

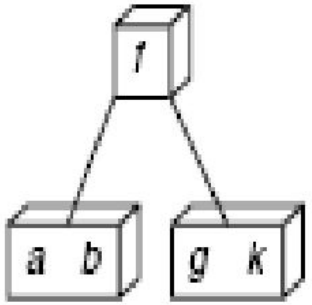
- Khi node được thêm vào bị đầy, node này sẽ được tách thành 2 node cùng mức, khoá median sẽ được đưa vào node mới
- Khi tách node, khoá median sẽ được dời lên node cha, quá trình này có thể lan truyền đến node gốc
- Trong trường hợp node gốc bị đầy, node gốc sẽ bị tách và dẫn đến việc tăng trưởng chiều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

1.

a, g, f, b:

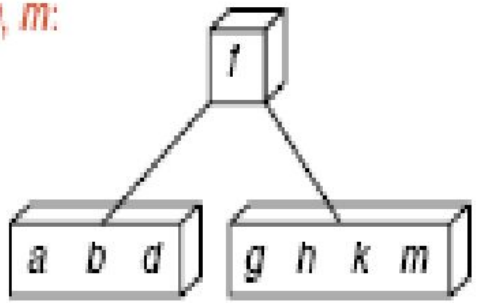
2.

k:



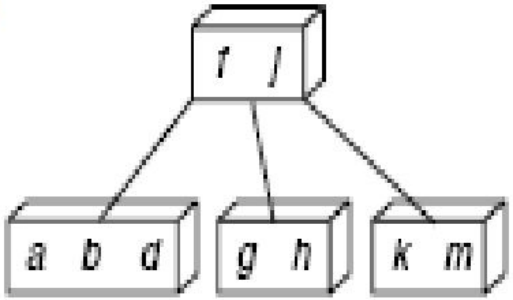
3.

d, h, m:



4.

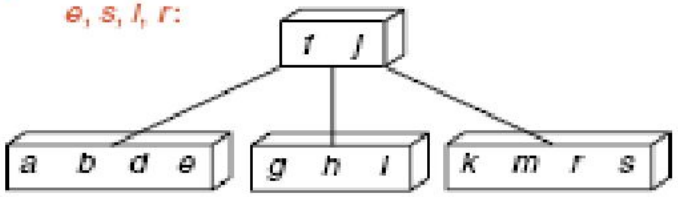
j:



Thêm node mới

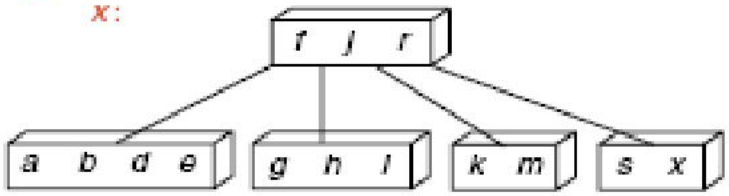
5.

e, s, l, r:



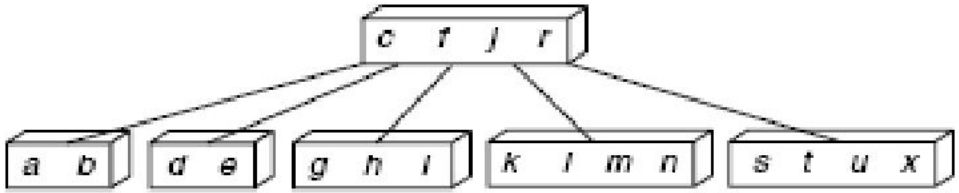
6.

x:



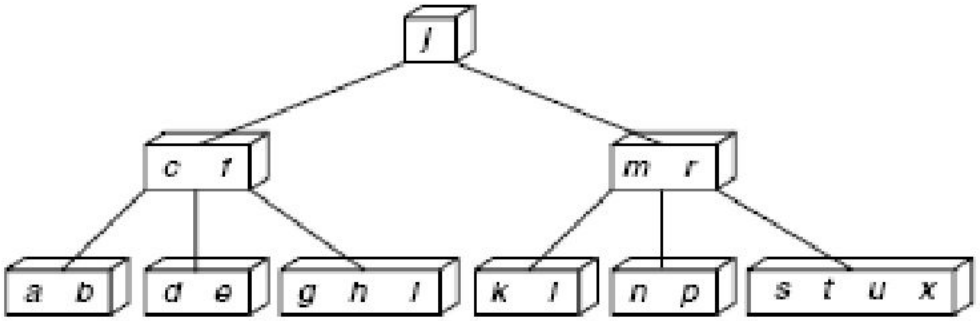
7.

c, l, n, t, u:



8.

p:



Thêm node mới

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

```
if(Root == NULL)
    Root = makeroot(k);
else
{
        s = search(k, &position, &timthay);
        if (s!=NULL)
            cout<<“Không thêm vào được”;
        else
            insert (s, k, position);
}
```

Thêm node mới

- 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

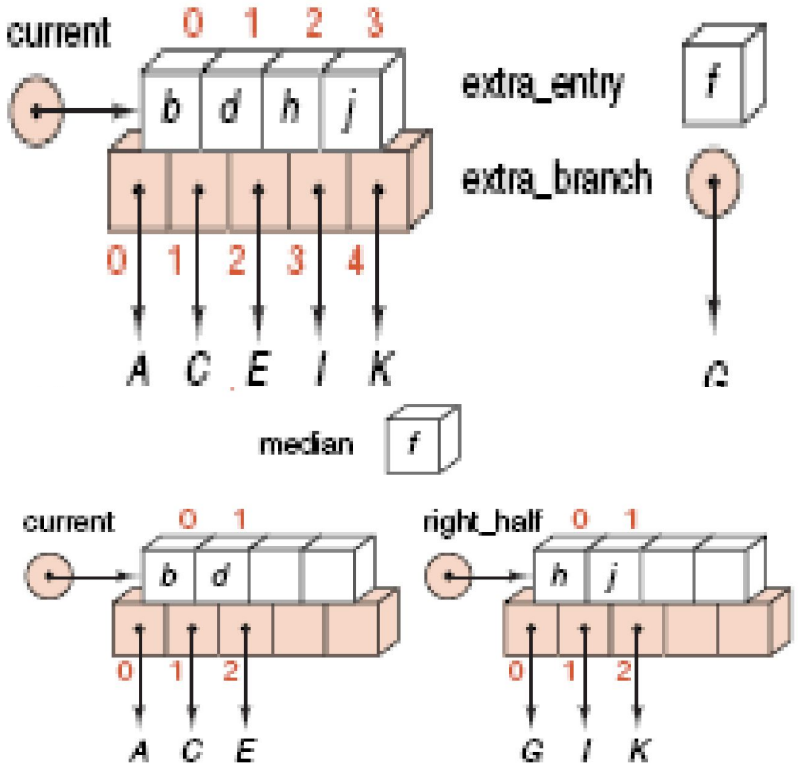
void insert (pBNode **s, int k, int position);**

Thêm node mới

```
void insert (pBNode s, int f, int position)
{
    pBNode current, right_half, P;
    pBNode extra_branch;

    int pos, extra_entry, median
    current = s;
    extra_entry = f;
    extra_branch = NULL;
    pos = position;
    p = father (current);
```

Position=2,order=5



Thêm node mới

```
while (p != NULL && current -> count == Order){
        split(current, extra_entry, extra_branch, position, right_half, median);
        current = p;
        extra_entry = median;
        extra_branch = right_half;
        pos = nodesearch (p, median);
        p = father (current);
}
if(current - > count+1 < Order){
        insnode (current, extra_entry, extra_branch, pos);
        return;
}
split (current, extra_entry, extra_branch, pos, right_half, median);
Root = makeroot (median);
Root -> Branch[0] = current;
Root -> Branch[1] = right_half;
}
```


Tách node

- Tách node đầy (current). 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 sẻ được chèn vào nút cha

Tách node

```
void split (pBNode current, int extra_entry, pBNode
      extra_branch, int position, pBNode &right_half, int
      &median)
{
      pBNode p;
      p = new pBNode;
      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;
            median = current -> key[Order/2];
            right_half = p ;
            return;
      }
```

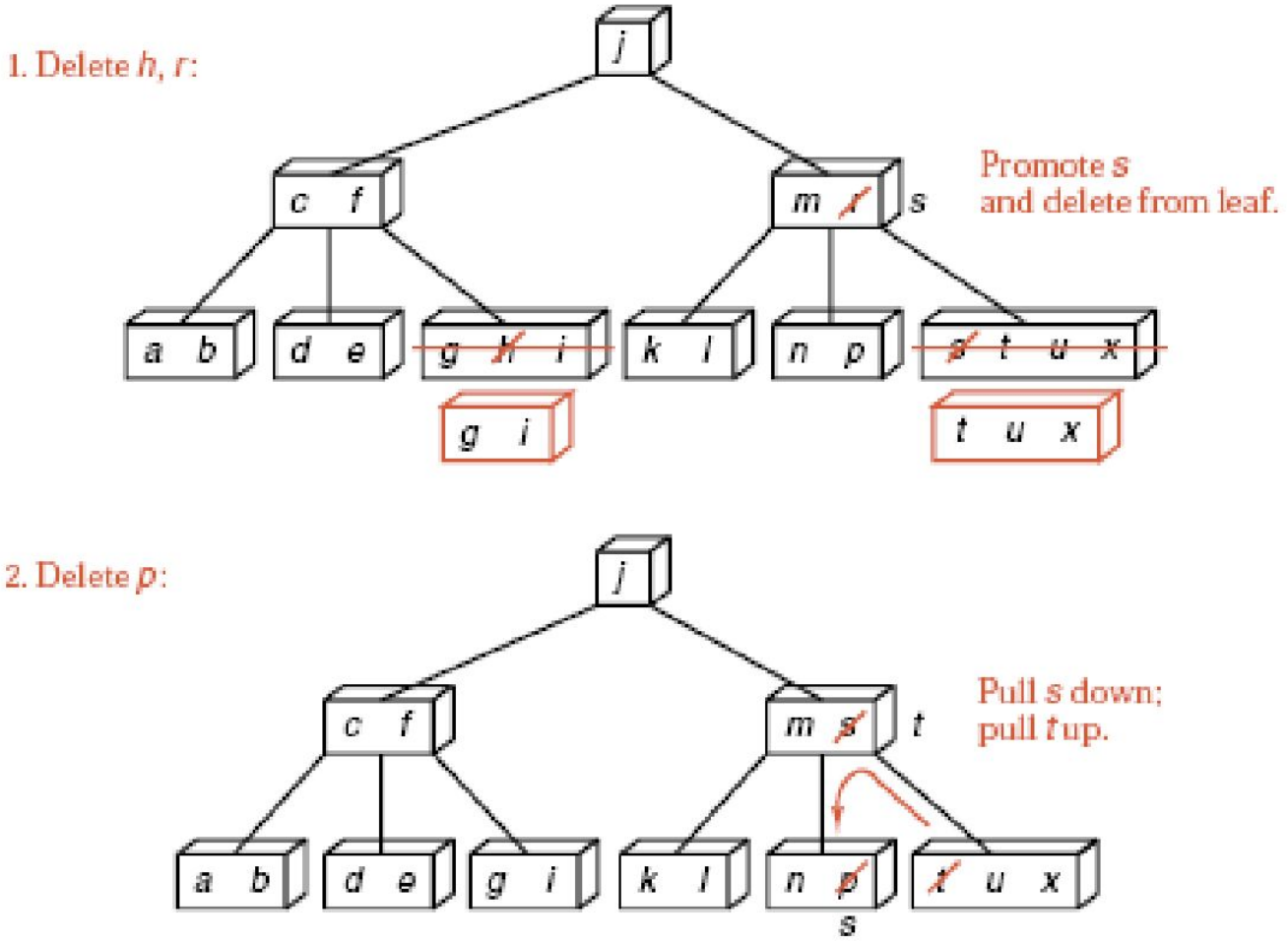

Tách node

```
        if (position == Order/2){
            copy(current, Order/2, Order-2, p);
            current->numtrees = Order/2+1;
            current -> Branch[0] = extra_branch;
            median = current -> key[Order/2];
            right_half = p;
            return;
        }
        if (position < Order/2){
            copy(current, Order/2, Order-2, p);
            current->numtrees = Order/2;
            median = current -> key[Order/2- 1];
            insnode(current, extra_entry, extra_branch,    position);
            right_half = p;
            return;
        }
}
```

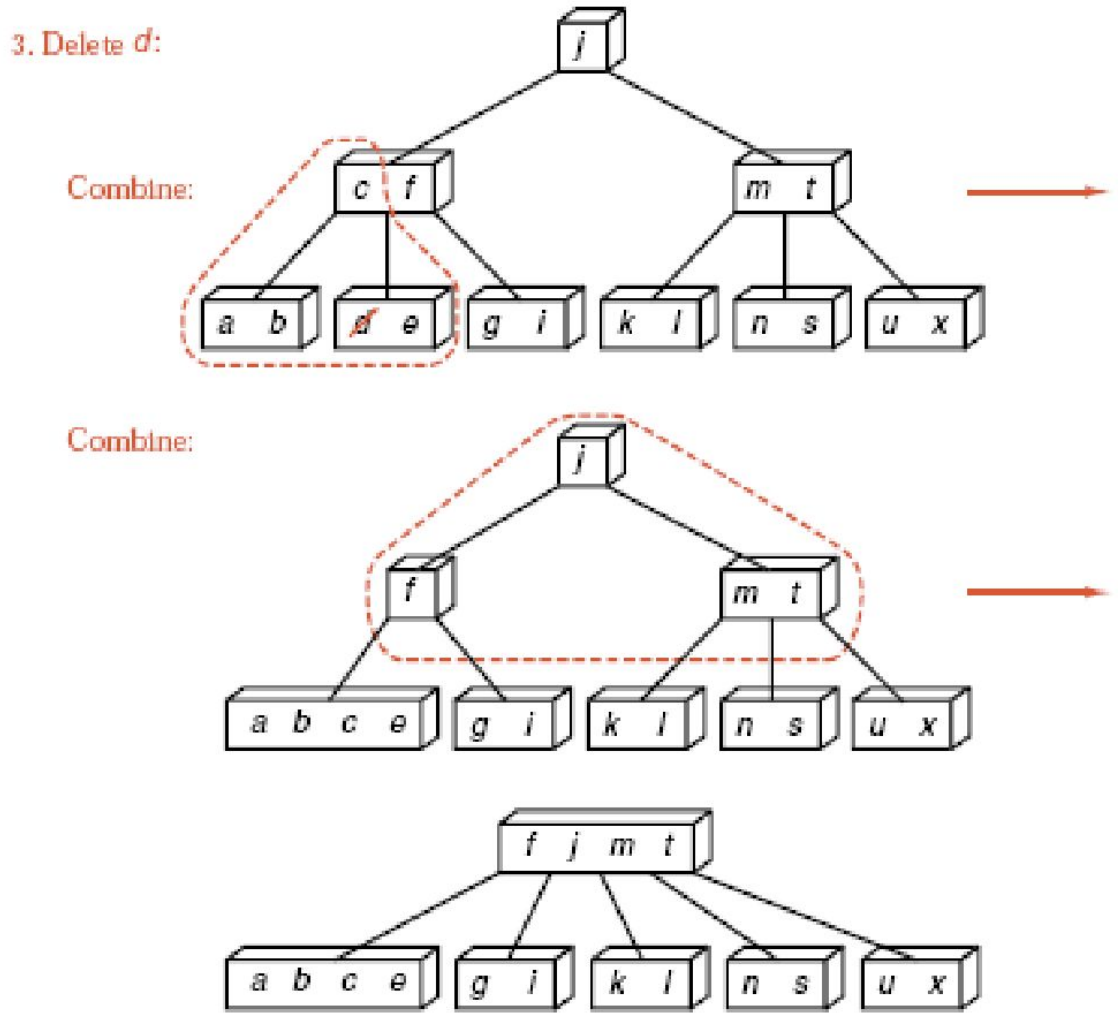

Thêm vào node lá

```
void insnode (pBNode current, int extra_entry,
    pBNode extra_branch, int position)
{
    int i;
    for(i = current->count; i >= position+1; i--){
        current -> Branch[i+1] = current -> Branch[i];
        current -> key[i] = current -> key[i - 1];
    }
    current -> key[position] = extra_entry;
    current -> Branch[position + 1] = extra_branch;
    current -> count +=1;
}
```


Loại bỏ


1. Delete h, r:
j
c f
m r
s
Promote s
and delete from leaf.
a b
d e
g h i
k l
n p
s t u x
g i
t u x
2. Delete p:
j
c f
m s
t
Pull s down;
pull t up.
a b
d e
g i
k l
n p
t u x
s

Loại bỏ


3. Delete d:
Combine:
j
c f
m t
a b
d e
g i
k l
n s
u x
Combine:
j
f
m t
a b c e
g i
k l
n s
u x
f j m t
a b c e
g i
k l
n s
u x

Câu hỏi và Bài tập

1. Nêu định nghĩa và các tính chất của cây B-Tree.
2. Cài đặt tất cả các thao trên cây B-Tree.
3. Cho B-tree bậc 5 gồm các khóa sau (chèn vào theo thứ tự): 3, 7, 9, 23, 45, 1, 5, 14, 25, 24, 13, 11, 8, 19, 4, 31, 35, 56

   Thực hiện các yêu cầu sau:

   - Thêm các khóa: 2, 6,12
   - Xóa khóa: 4, 5, 7, 3, 14
4. Khởi tạo B Tree bậc 7 với các thao tác Insert: 34, 12, 55, 21, 6, 84, 5, 33, 15, 74, 54, 28, 10, 19

   Thực hiện các chuỗi thao tác sau:

   - Insert(11), Delete(15), Delete(6), Insert(98), Delete(34), Delete(5)