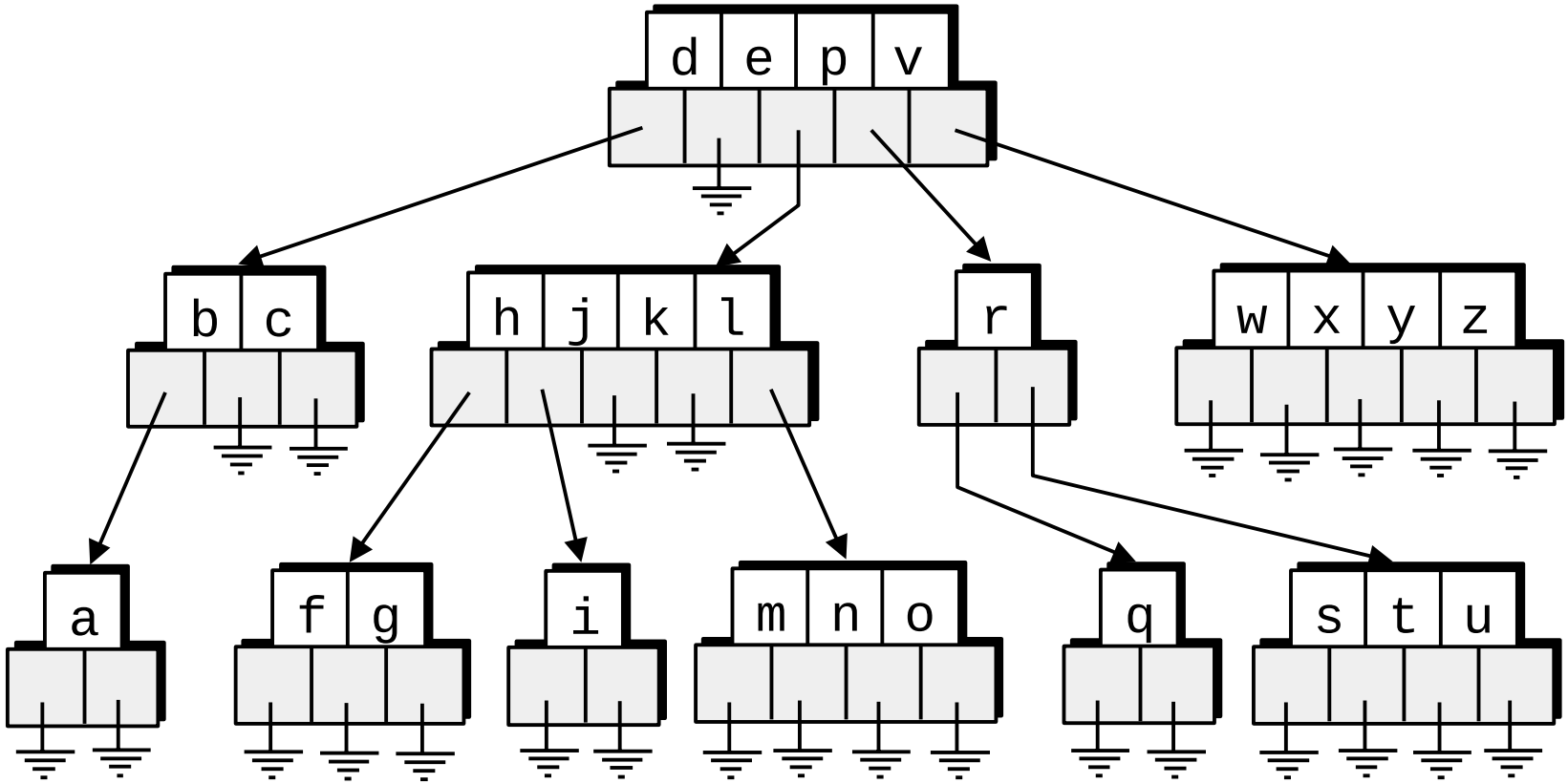
B-TREES

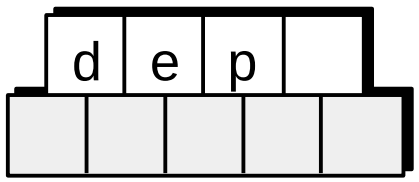
Multiway Search Trees

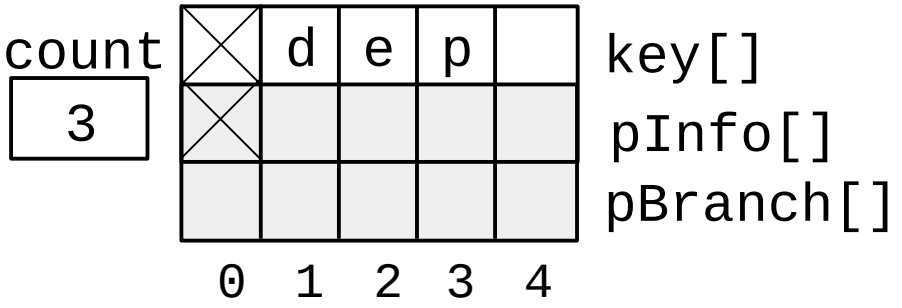


A balanced 5-way search tree

Multiway Search Trees

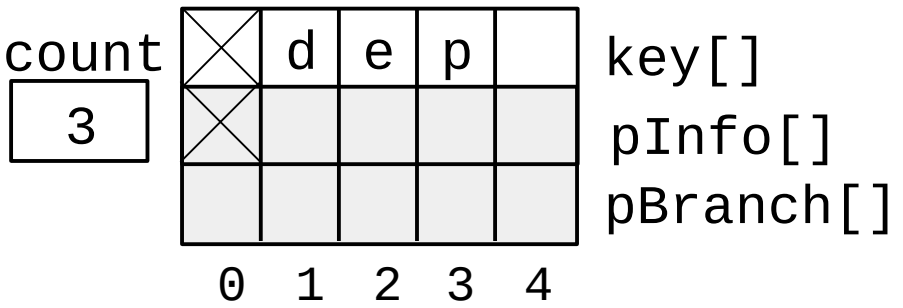
```
#define NUMKEY_MAX    4  /* maximum number of keys */

typedef int    KeyType;

typedef struct NodeTag {
        int                count;
        KeyType            key[ NUMKEY_MAX + 1 ];
        InfoType           *pInfo[ NUMKEY_MAX + 1 ];
        struct NodeTag     *pBranch[ NUMKEY_MAX + 1 ];
} NodeType;
```



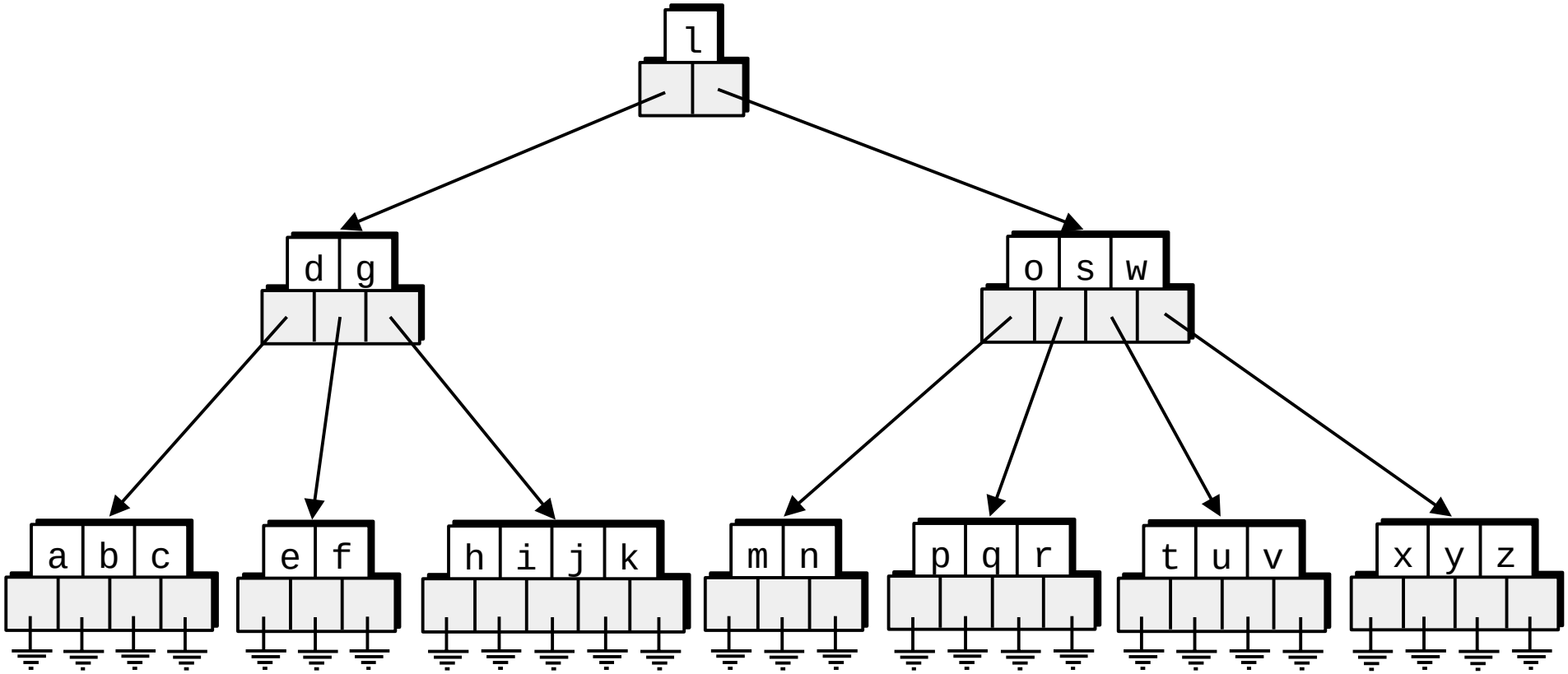


Multiway Search Trees

```
Nodetype *MSearch(KeyType Target, NodeType *pRoot, int *pPos )
{
  if ( pRoot == NULL ) return( NULL );
  if ( LT(Target, pRoot-key[1]) )
    return( MSearch( Target, pRoot->pBranch[0], pPos ) );
  else {
    *pPos = 1;
    while( *pPos <= pRoot->count && GT(Target,pRoot->key[*pPos])
      (*pPos)++;
    if ( EQ( Target, pRoot->key[ *pPos ] ) )
      return( pRoot );
    else
      return( MSearch( Target, pRoot->pBranch[*pPos], pPos );
  }
}
```



B-Trees

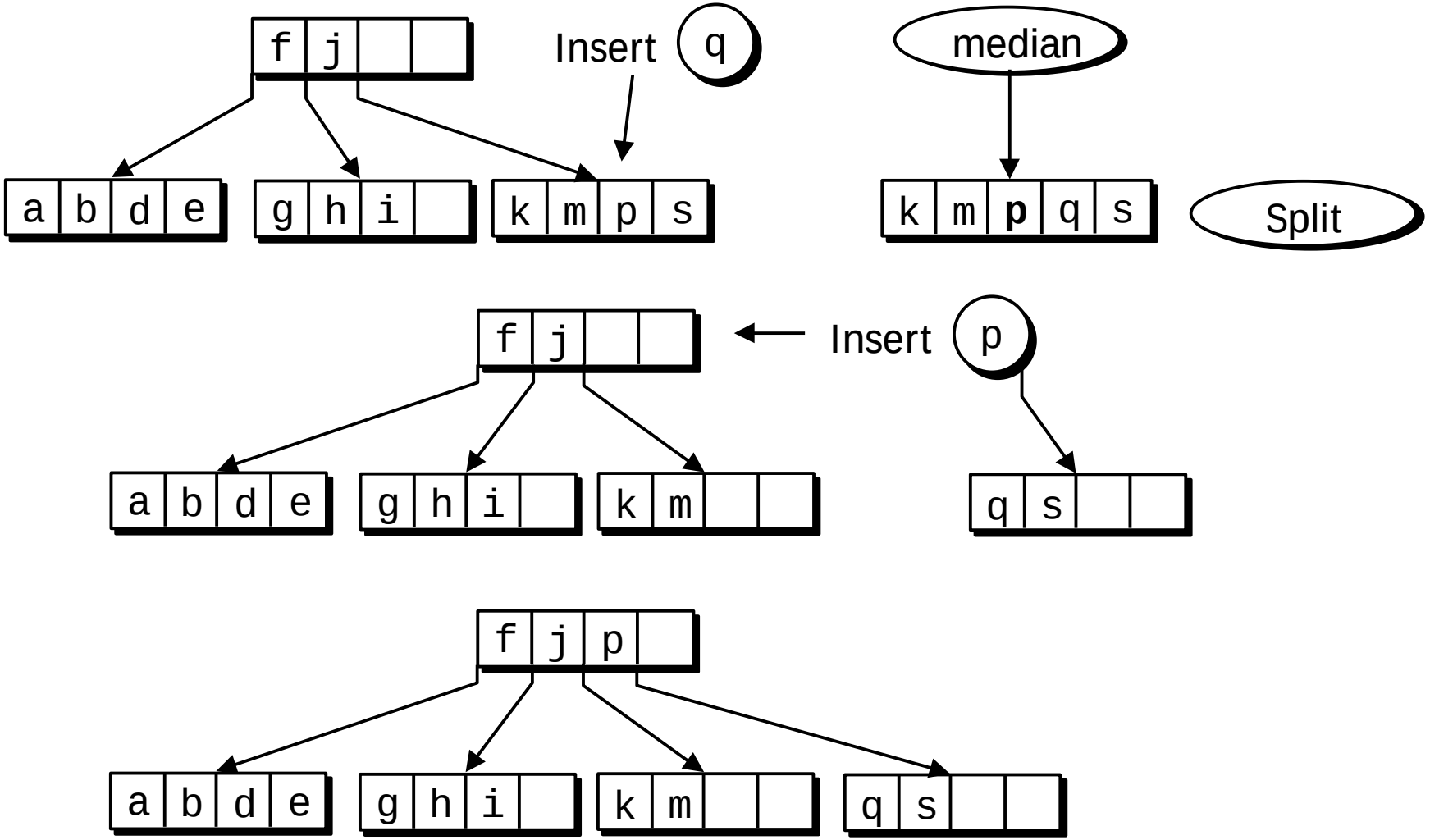


A B-tree of order 5

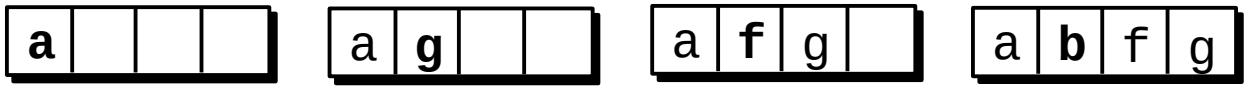
B-Trees

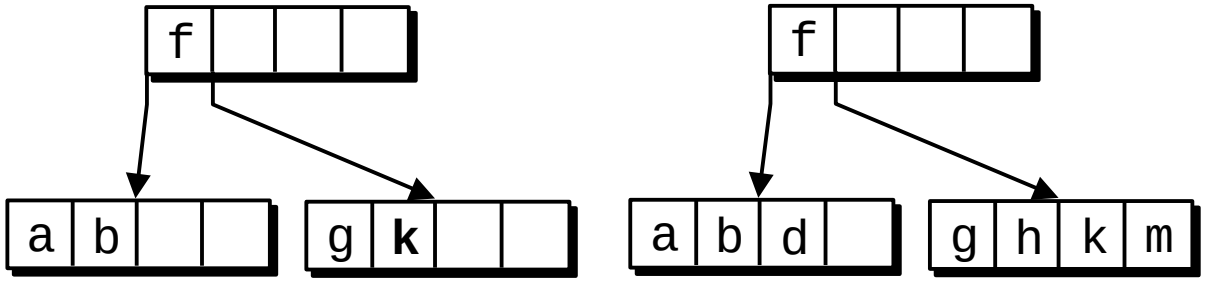
- B-tree of order m คือ m-way search tree ที่
- แต่ละจุดเก็บคีย์ได้อย่างมาก m-1 คีย์
- จุดที่ไม่ใช่ราก จะต้องเก็บคีย์อย่างน้อย $\lceil m/2 \rceil - 1$ คีย์
- รากจะต้องเก็บคีย์อย่างน้อย 1 (ยกเว้นว่าต้นไม้ไม่มีข้อมูล)
- ใบทุกใบของต้นไม้อยู่ในระดับเดียวกัน

Insertion into a B-Tree

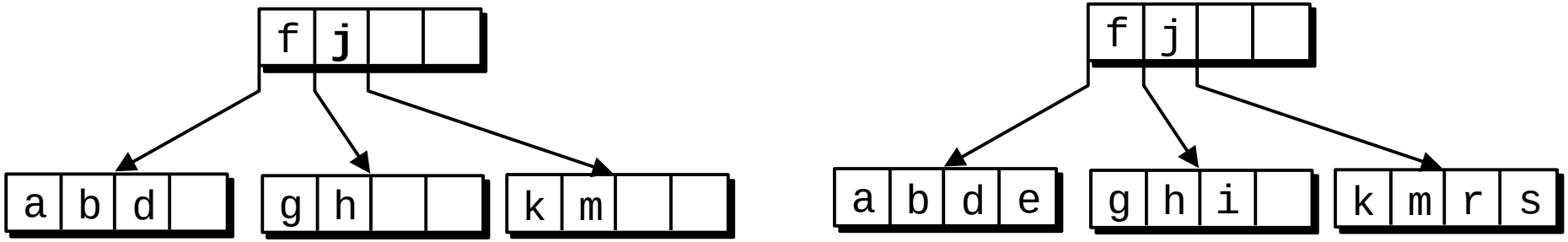


Growth of a B-Tree





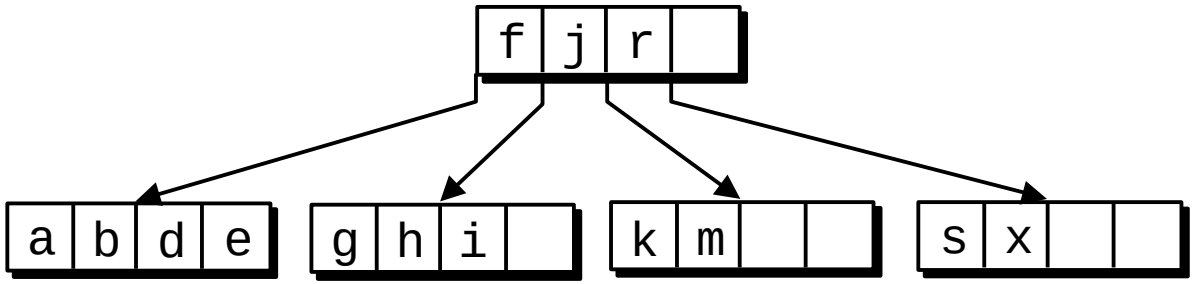
Insert: d,h,m



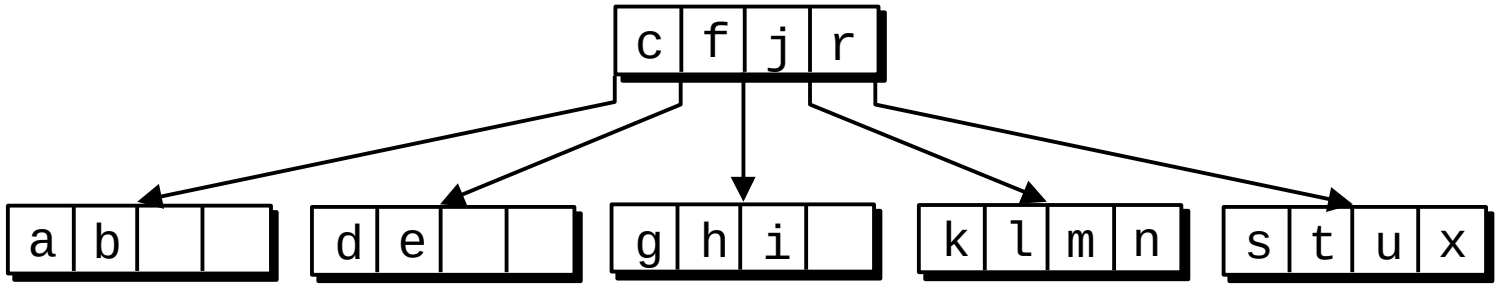
Insert: j

Insert: e,s,i,r

Growth of a B-Trees

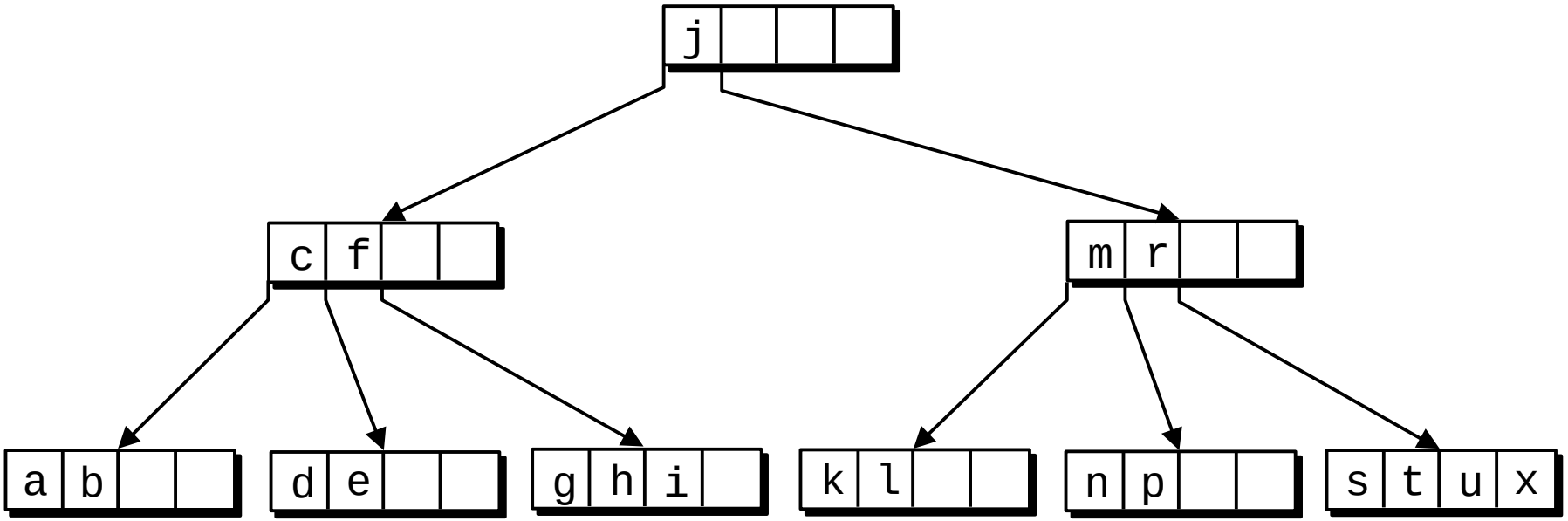


Insert: x



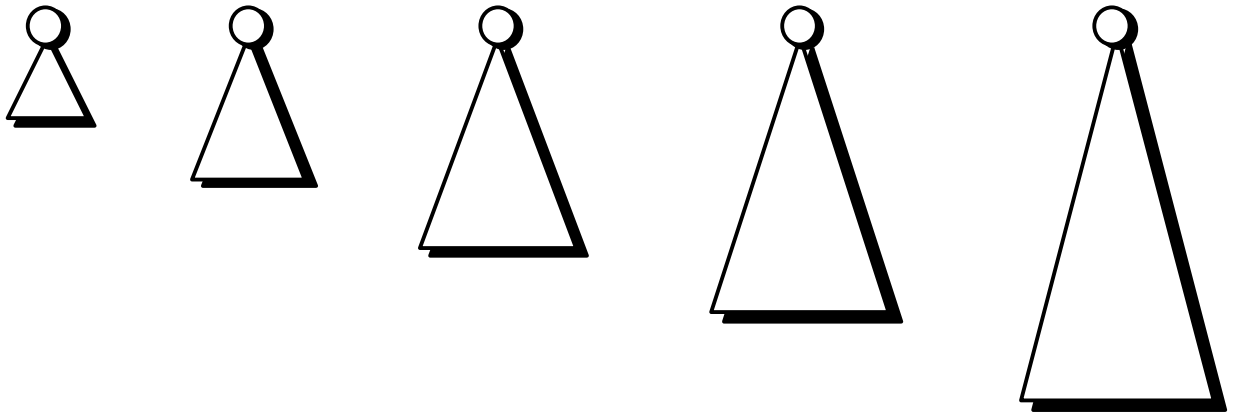
Insert: c, l, n, t, u

Growth of a B-Trees

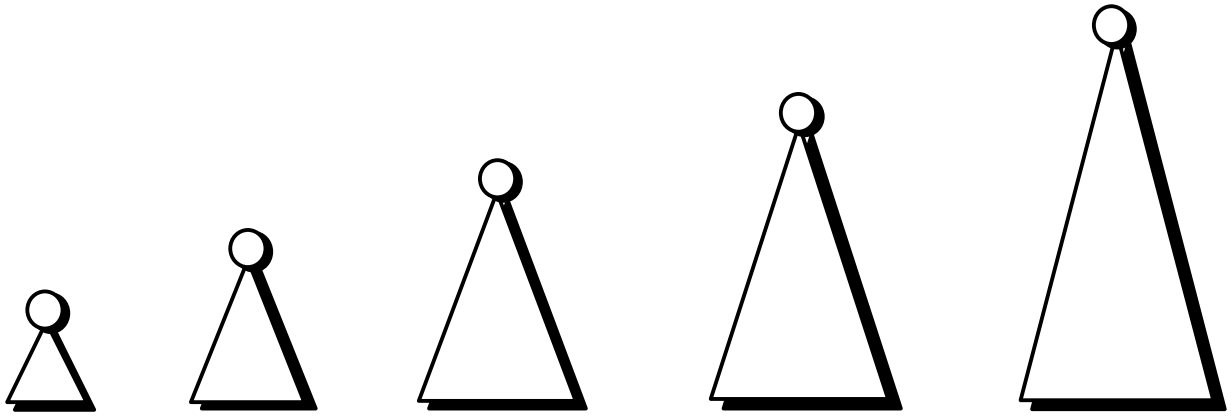


Insert: p

Binary Search-Trees vs B-Trees



Seach Trees



B-Trees

Deletion from a B-Tree

สมมติว่าต้องการลบข้อมูลที่คีย์มีค่า X

ถ้าคีย์ X นี้อยู่ใน node ภายใน B-tree

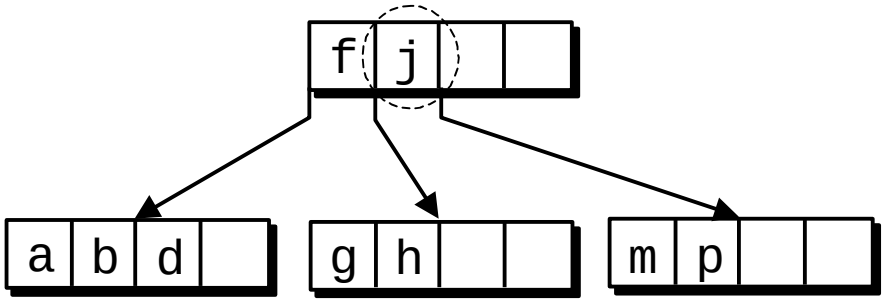
1. ให้หาคีย์ที่มีค่ามากที่สุดที่น้อยกว่า X หรือหาคีย์ที่มีค่าน้อยที่สุดที่มากกว่า X 2. ถ

ให้คีย์ดังกล่าวนี้คือ y ให้ย้ายคีย์ y นี้ไปแทนที่ ณ ตำแหน่งที่ X อยู่

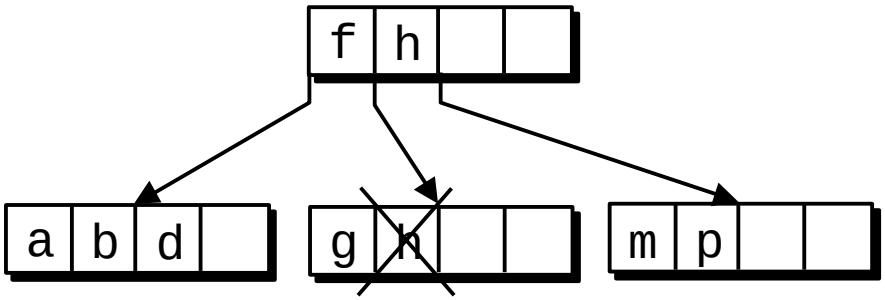
ซึ่งเหมือนกับกรณีที่ลบคีย์ใน binary search tree หรือ AVL-tree

ให้สังเกตุว่า คีย์ y ที่หาได้นี้ จะต้องอยู่ใน node ที่เป็นใบเสมอ จึงเป็นการเปลี่ยนปั□หาการ

ใน node ใดๆ เป็นการลบคีย์ y ที่อยู่ในใบ

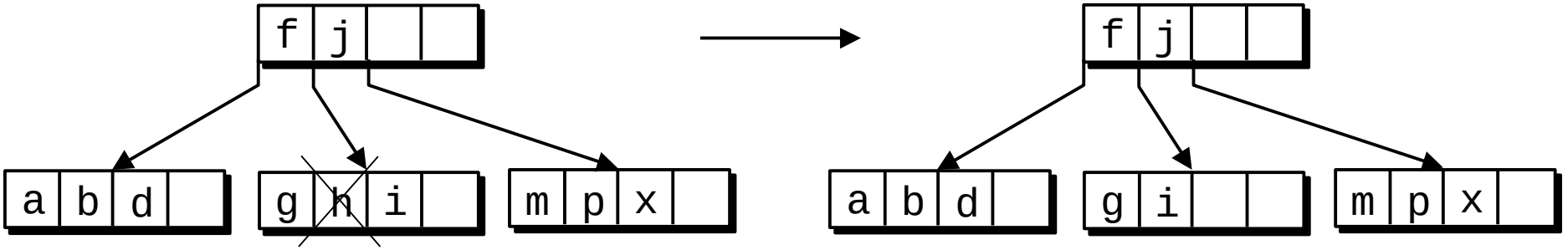


Delete : j

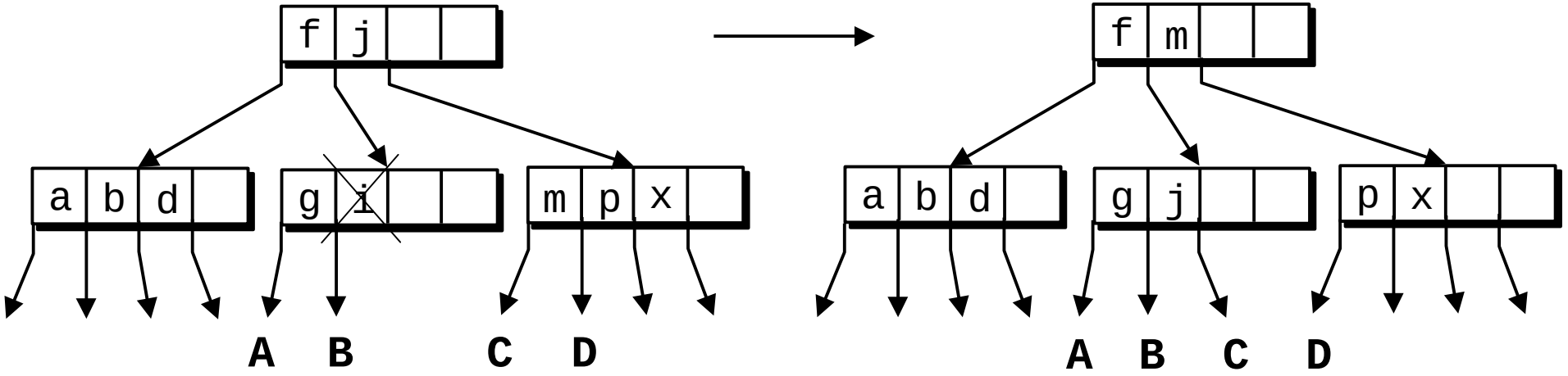


Delete : h

Deletion from a B-Tree

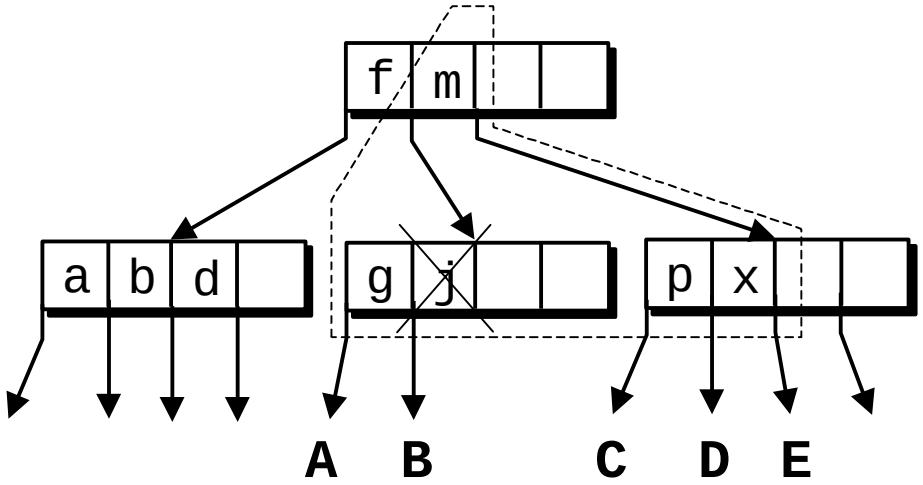


Delete : h

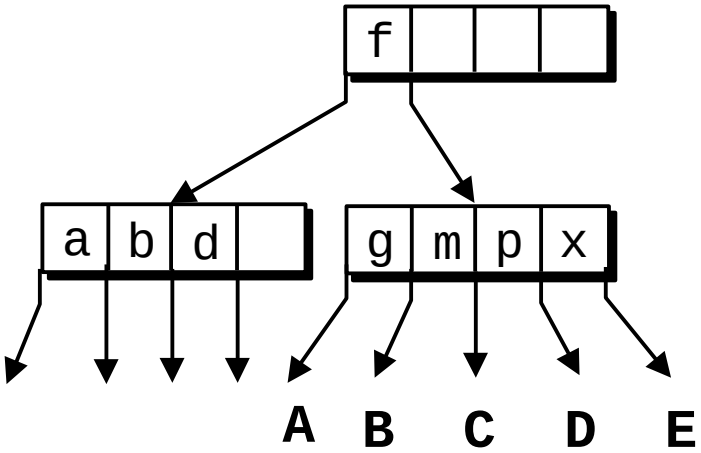


Delete i : ดึง j ลง, ผลัก m ขึ้น

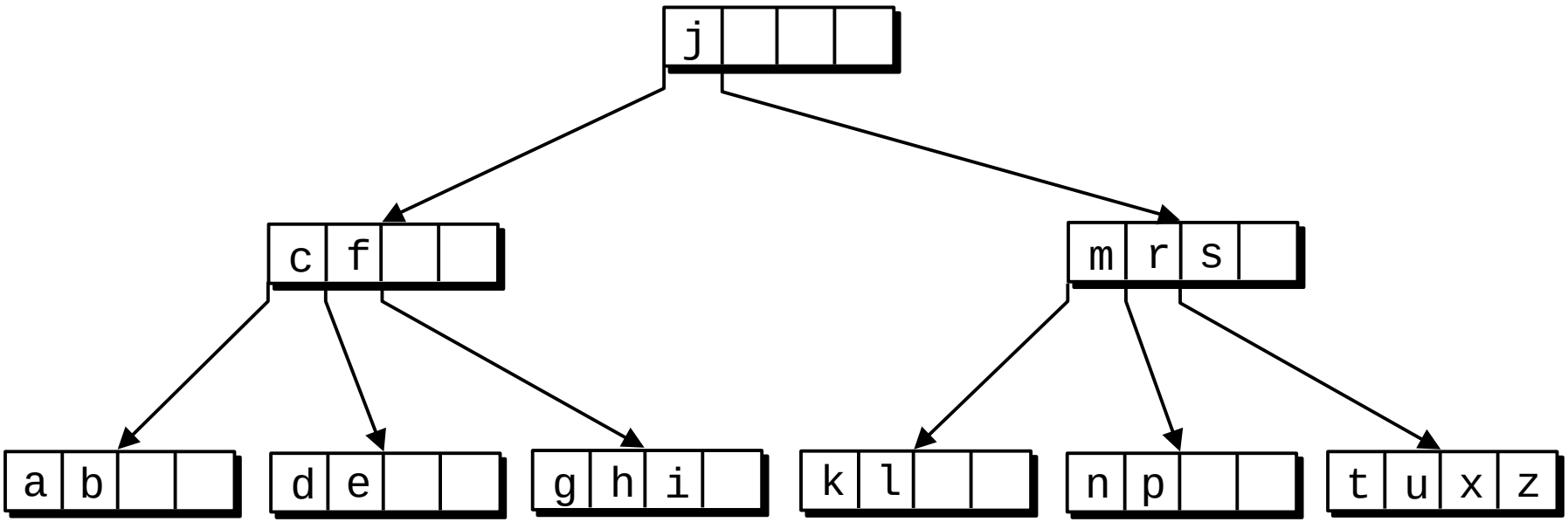
Deletion from a B-Tree



Delete `j` : Combine `g,m,p,x`



Deletion : Example



Delete : h, r, p, d

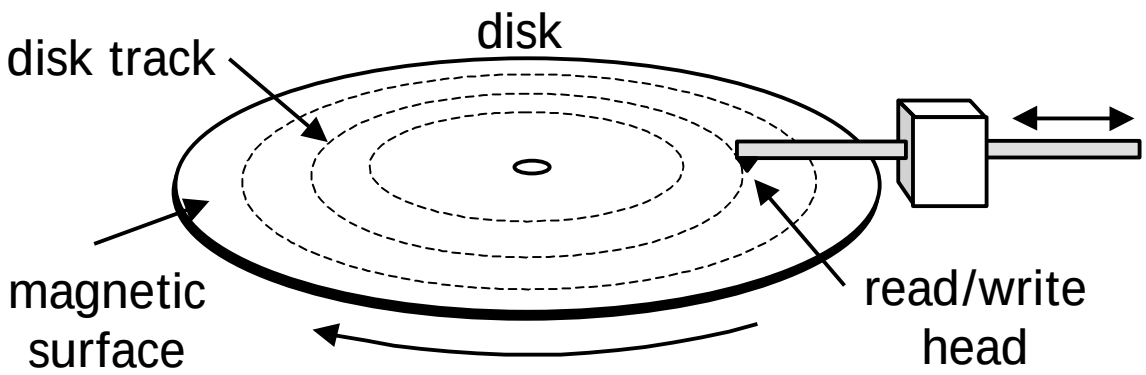
Secondary Storages

Cost measure

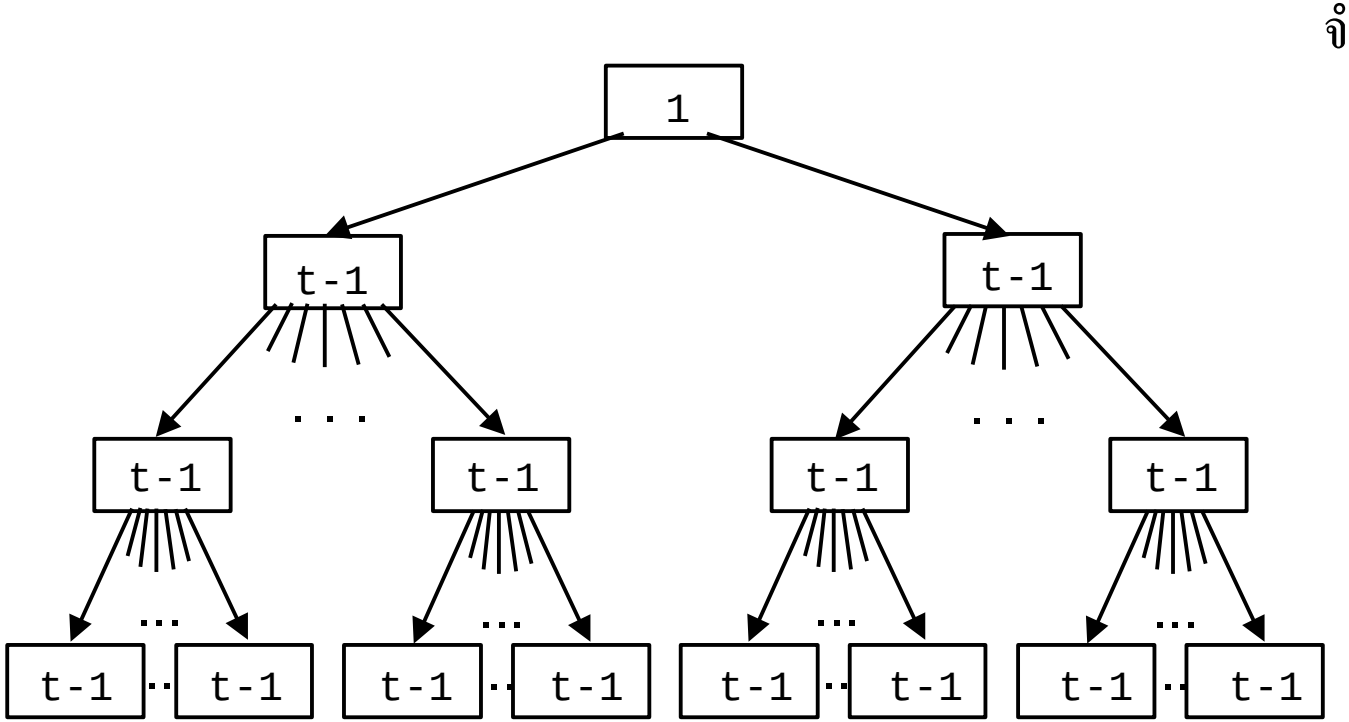
1. เวลาในการเคลื่อนหัวอ่าน/เขียนข้อมูล ไปยัง track ที่ต้องการ (t_s)
2. เวลาในการรอให้ block ที่เก็บข้อมูลบน track หมุนมาที่ตรงหัวอ่าน (t_s)
3. เวลาในการอ่าน/เขียน (ถ่ายเท) ข้อมูล 1 ไบต์ (t_s)

(b = จำนวนไบต์ต่อหนึ่ง block)

$(\quad t_s \quad)$



The Height of a B-Tree



$$\left(\boxed{t \quad s} \right)$$

Searching a B-Tree : Analysis

สมมติว่าเวลาในการอ่าน 1 node ของ B-tree ที่มี order m จาก disk คือ

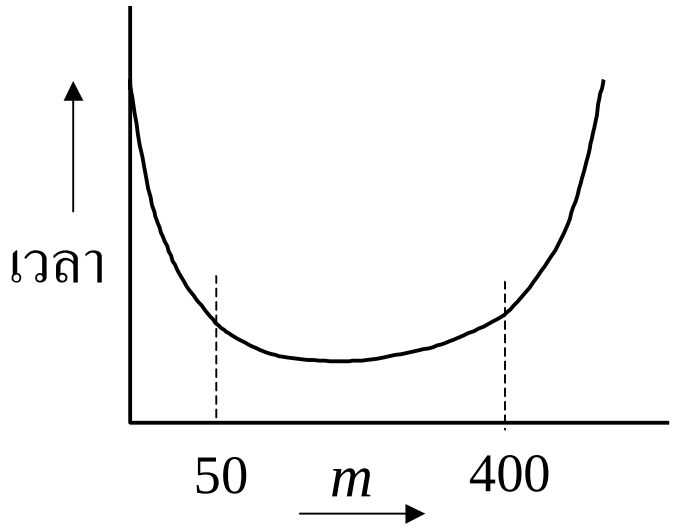
$$a + bm$$

โดยใช้ binary search ในการหาคีย์ที่ต้องการใน node ที่อ่านได้ ซึ่งใช้เวลา

$$a + bm$$

และจำนวน nodes ที่ต้องอ่าน (worst case) ใน B-tree (ซึ่งมี n nodes) เพื่อหาคีย์ที่ต้องการ จะ
เท่ากับความสูงของต้นไม้ h ดังนั้นเวลาในการหาคีย์คือ

$$a + br$$



Insertion into a B-Tree : Analysis

สมมติให้ B-tree order m สูง h มี p nodes และเก็บข้อมูล n ข้อมูล

การแทรกข้อมูลประกอบด้วย

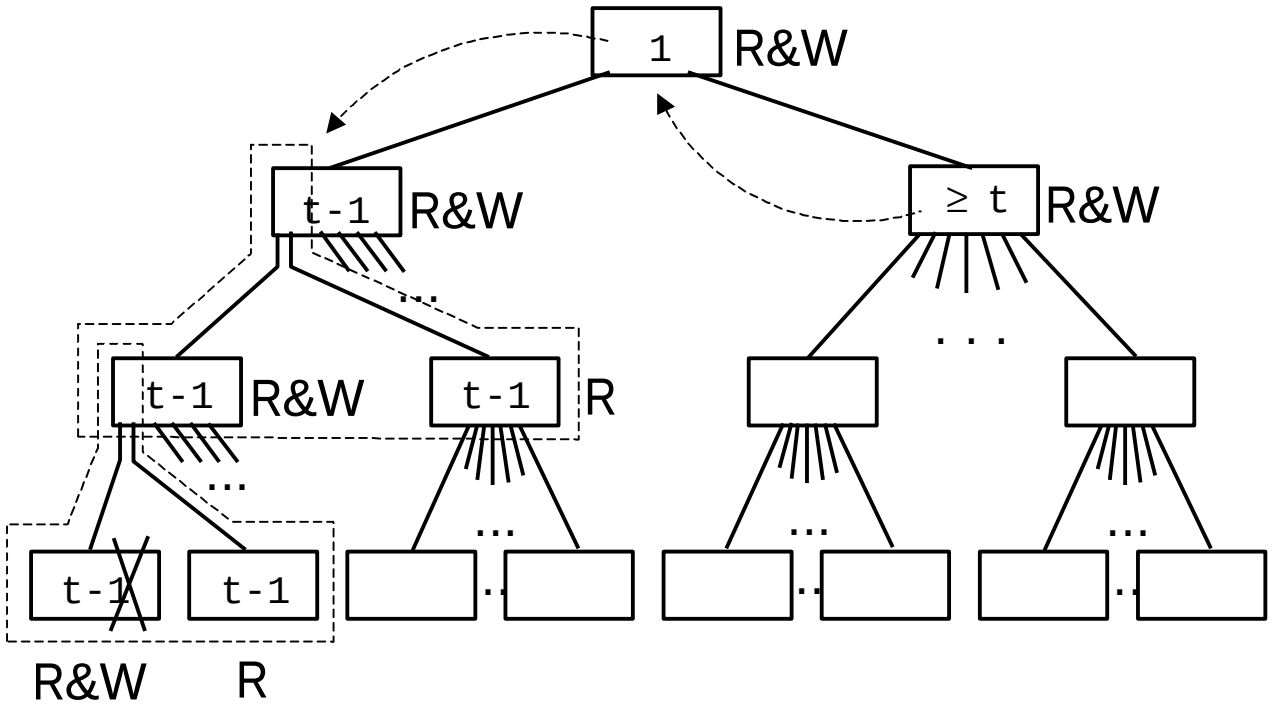
1. การหาใบที่ต้องการ ซึ่งต้องอ่านข้อมูล h+1 ครั้ง
2. อาจเกิดการ split node สมมติว่าเกิดขึ้น k ครั้ง โดยที่ $k \leq p$-2
   โดยการ split แต่ละครั้ง ต้องเขียนข้อมูล 2 ครั้ง
3. ปิดท้ายด้วยการแทรกข้อมูลลงใน node ที่ไม่เต็ม และเขียนข้อมูลอีก 1 ครั้ง

$$a + bm$$

ดังนั้นจำนวนการอ่าน/เขียน ในการแทรกข้อมูล

$$a+br$$

Deletion from a B-Tree : Analysis



Worst case : กรณีที่มีการ combine จากใบขึ้นไปเรื่อยๆ จนถึงระดับที่ 2

แต่เมื่อถึงระดับที่ 1 เกิดการย้ายคีย์ขึ้นลงใน node ข้างเคียง กับคีย์ที่ราก

(ให้ต้นไม้มีความสูง h)

จำนวนการอ่าน/เขียนทั้งหมดที่มากสุด = $2(h+1)+h+1 = 3(h+1)$