طراحی و تحلیل الگوریتم‌ها

مبحث چهاردهم

مباحث پیشرفته در ساختمان داده‌ها

درخت‌های B

B-Trees

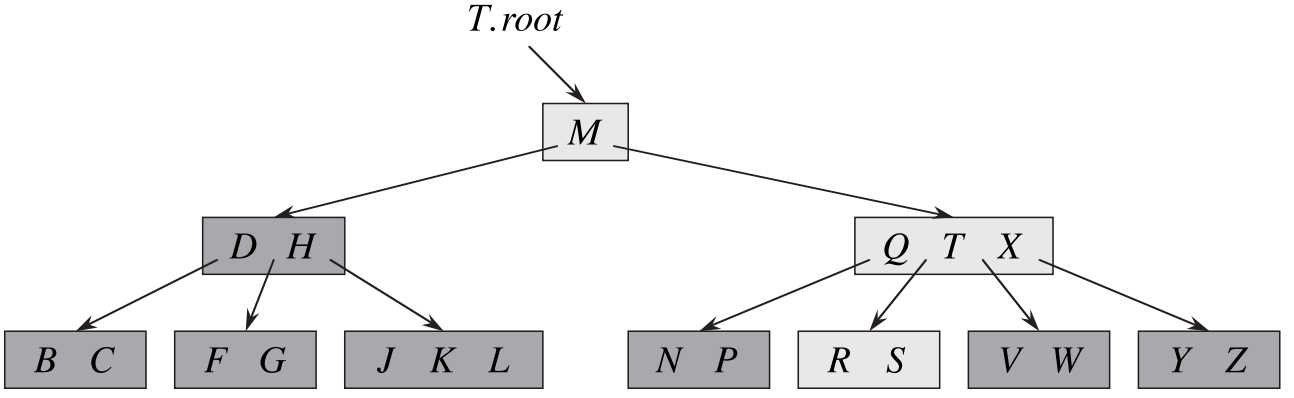
کاظم فولادی
دانشکده مهندسی برق و کامپیوتر
دانشگاه تهران

http://courses.fouladi.ir/algorithm

درخت B

B-TREE

در درخت‌های B، توازن درخت با ایجاد امکان متغیر بودن تعداد فرزندان یک گره به دست می‌آید.



از درخت‌های B، برای ایجاد مجموعه‌های پویا با عملیات در زمان $O(\lg n)$ استفاده می‌شود.

درخت B

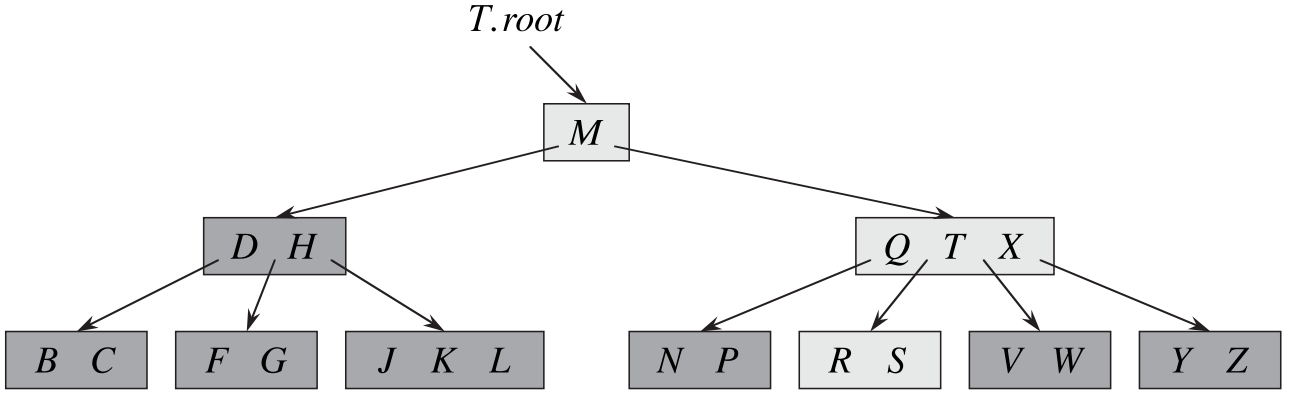
تعریف

B-TREE

درخت B
B-Tree

یک درخت B از حداقل درجه‌ی t (که t عدد صحیح بزرگتر یا مساوی ۲ است):

- در هر گره (بجز احتمالاً در ریشه) بین $t-1$ تا $2t-1$ کلید وجود دارد.
- همه‌ی مسیرها از ریشه تا یک برگ طول مساوی دارند.
- شرط ترتیب درخت جستجوی دودویی تعمیم‌یافته:
کلیدها در هر رأس مرتب‌شده هستند و زیردرخت‌ها بر اساس این کلیدها به هر رأس متصل می‌شوند.



Prepared by Kazim Fouladi | Fall 2016 | 2nd Edition
کاظم فولادی

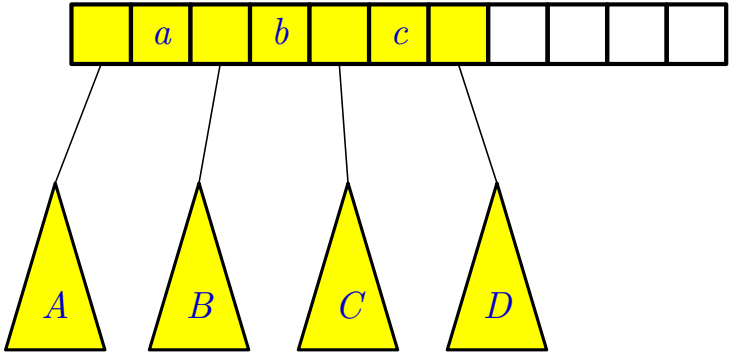
درخت B

مثال

B-TREE

$$t = 3$$

$$A \leq a \leq B \leq b \leq C \leq c \leq D$$



شرط ترتیب

درخت B

ویژگی‌ها (۱ از ۲)

B-TREE

A ***B-tree*** T is a rooted tree (whose root is $root[T]$) having the following properties:

1. Every node x has the following fields:
   a. $n[x]$, the number of keys currently stored in node x,
   b. the $n[x]$ keys themselves, stored in nondecreasing order, so that $key_1[x] \leq key_2[x] \leq \cdots \leq key_{n[x]}[x]$,
   c. $leaf[x]$, a boolean value that is TRUE if x is a leaf and FALSE if x is an internal node.
2. Each internal node x also contains $n[x]+1$ pointers $c_1[x], c_2[x], \ldots, c_{n[x]+1}[x]$ to its children. Leaf nodes have no children, so their c_i fields are undefined.

درخت B

ویژگی‌ها (۲ از ۲)

B-TREE

3. The keys $key_i[x]$ separate the ranges of keys stored in each subtree: if k_i is any key stored in the subtree with root $c_i[x]$, then

   $$k_1 \le key_1[x] \le k_2 \le key_2[x] \le \cdots \le key_{n[x]}[x] \le k_{n[x]+1} .$$

4. All leaves have the same depth, which is the tree's height h.
5. There are lower and upper bounds on the number of keys a node can contain. These bounds can be expressed in terms of a fixed integer $t \ge 2$ called the ***minimum degree*** of the B-tree:

   a. Every node other than the root must have at least $t - 1$ keys. Every internal node other than the root thus has at least t children. If the tree is nonempty, the root must have at least one key.

   b. Every node can contain at most $2t - 1$ keys. Therefore, an internal node can have at most $2t$ children. We say that a node is ***full*** if it contains exactly $2t - 1$ keys.

B درخت

ارتفاع درخت

B-TREE

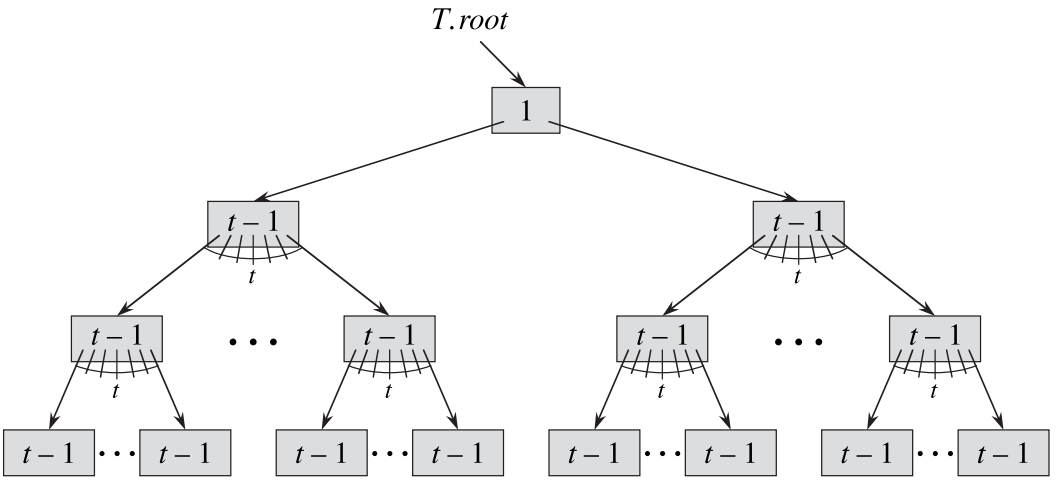
If $n \geq 1$, then for any n-key B-tree T of height h and minimum degree $t \geq 2$,

$$h \leq \log_t \frac{n+1}{2}$$

درخت B

شمارش تعداد گره‌ها

B-TREE



depth	number of nodes
0	1
1	2
2	$2t$
3	$2t^2$

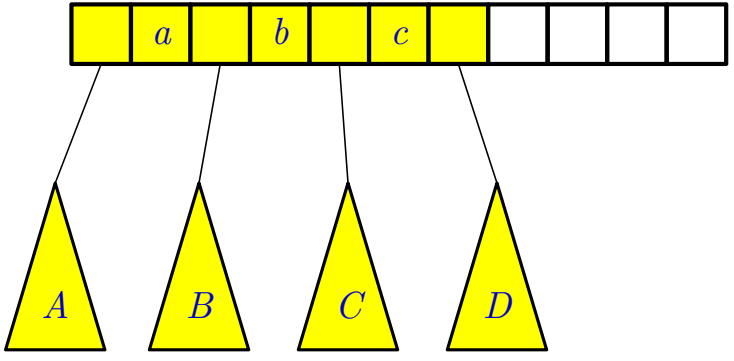
$$
\begin{aligned}
n &\geq 1 + (t-1)\sum_{i=1}^{h} 2t^{i-1} \\
&= 1 + 2(t-1)\left(\frac{t^h - 1}{t-1}\right) \qquad \Rightarrow \qquad t^h \leq (n+1)/2 \\
&= 2t^h - 1 \,.
\end{aligned}
$$

درخت B

جستجو

SEARCHING

جستجو در درخت B، شکل تعمیم‌یافته‌ی جستجو در BST است.



1. For a key $< a$, look in A
2. For a key $< b$, look in B
3. For a key $< c$, look in C
4. For a key $> c$, look in D

$$O(h) = O(\lg n)$$

درخت B

جستجو: شبه‌کد

SEARCHING

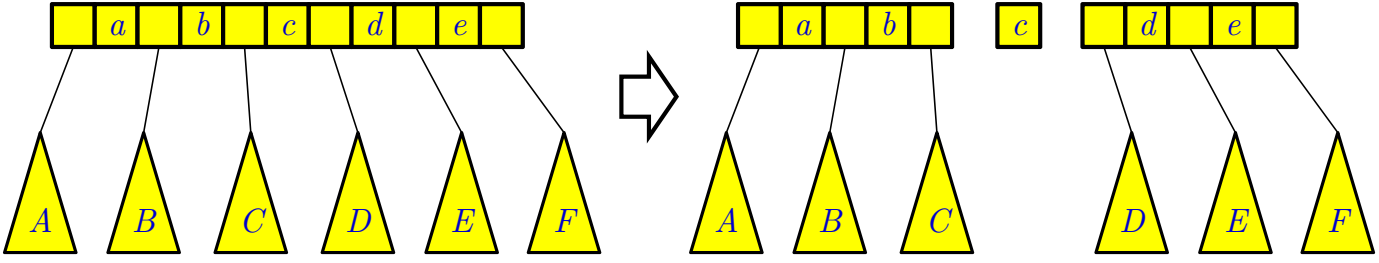
```
B-TREE-SEARCH(x, k)
1  i ← 1
2  while i ≤ n[x] and k > key_i[x]
3        do i ← i + 1
4  if i ≤ n[x] and k = key_i[x]
5     then return (x, i)
6  if leaf[x]
7     then return NIL
8     else  DISK-READ(c_i[x])
9           return B-TREE-SEARCH(c_i[x], k)
```

درخت B

تقسیم

SPLITTING NODES

یک گره پر (full) است، اگر حاوی $2t - 1$ کلید باشد.
یک گره‌ی پر می‌تواند به یک جفت گره‌ی کوچک‌تر (می‌نیمال) و یک «کلید میانه» شکسته شود.



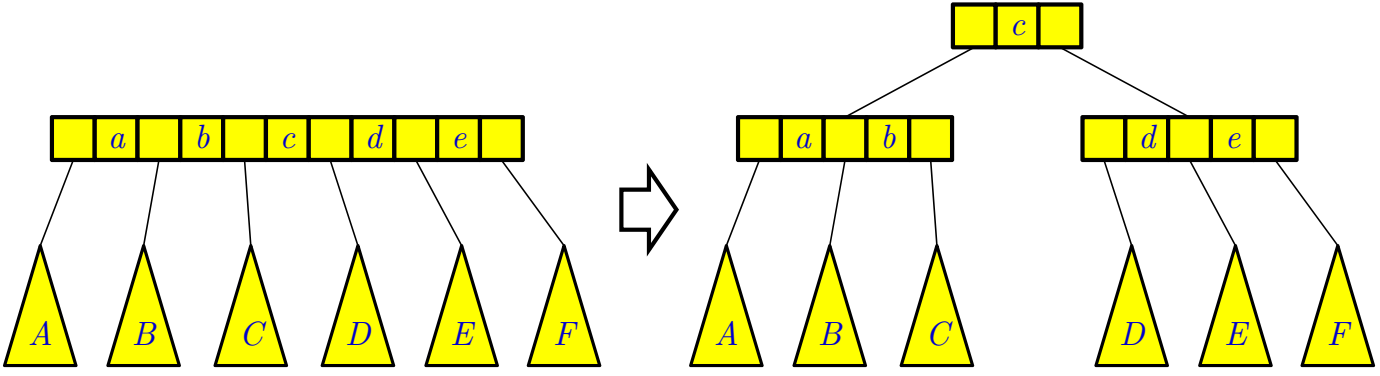
درخت B

تقسیم

SPLITTING NODES

یک گره پر (full) است، اگر حاوی $2t - 1$ کلید باشد.
یک گره‌ی پر می‌تواند به یک جفت گره‌ی کوچک‌تر (می‌نیمال) و یک «کلید میانه» شکسته شود.

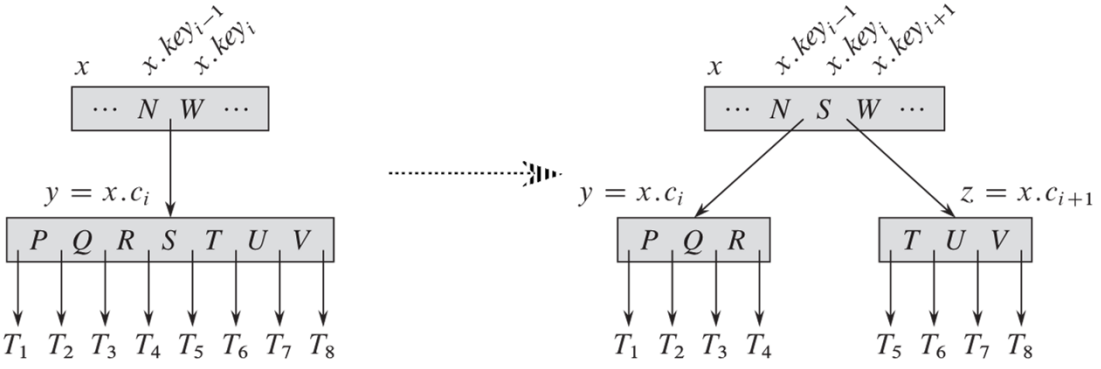
اگر گره‌ی پر، ریشه باشد: می‌توانیم با اضافه کردن ارتفاع، یک درخت B جدید بسازیم که ریشه‌ی آن پر نیست.



درخت B

تقسیم (گره‌ی معمولی)

SPLITTING NODES

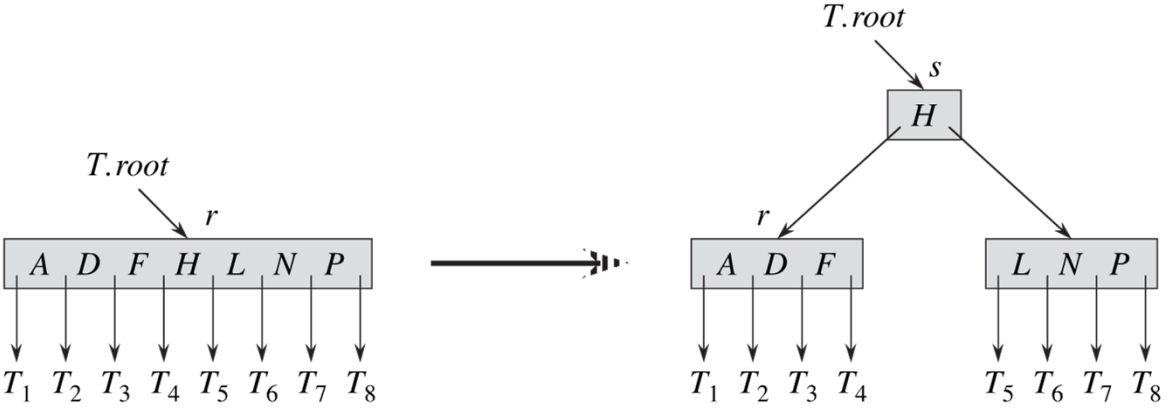


Splitting a node with $t = 4$. Node $y = x.c_i$ splits into two nodes, y and z, and the median key S of y moves up into y's parent.

درخت B

تقسیم (گره‌ی ریشه)

SPLITTING NODES



Splitting the root with $t = 4$. Root node r splits in two, and a new root node s is created. The new root contains the median key of r and has the two halves of r as children. The B-tree grows in height by one when the root is split.

درخت B

تقسیم: شبه‌کد

SPLITTING NODES

```
B-TREE-SPLIT-CHILD(x, i, y)
 1  z ← ALLOCATE-NODE()
 2  leaf[z] ← leaf[y]
 3  n[z] ← t − 1
 4  for j ← 1 to t − 1
 5        do key_j[z] ← key_{j+t}[y]
 6  if not leaf[y]
 7     then for j ← 1 to t
 8               do c_j[z] ← c_{j+t}[y]
 9  n[y] ← t − 1
10  for j ← n[x] + 1 downto i + 1
11        do c_{j+1}[x] ← c_j[x]
12  c_{i+1}[x] ← z
13  for j ← n[x] downto i
14        do key_{j+1}[x] ← key_j[x]
15  key_i[x] ← key_t[y]
16  n[x] ← n[x] + 1
17  DISK-WRITE(y)
18  DISK-WRITE(z)
19  DISK-WRITE(x)
```

درخت B

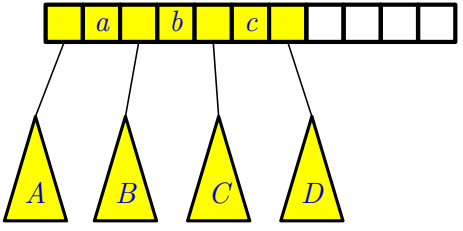
درج

INSERTING

اگر ریشه پر باشد، ابتدا آن را تقسیم می‌کنیم. حال درج را برای درخت‌های B با ریشه‌ی غیرپر تعریف می‌کنیم:

فرض می‌کنیم مقدار x باید بین b و c درج شود:

حالت ۱) گره، برگ است: x به‌سادگی در آرایه‌ی کلیدها درج می‌شود [نیاز به شیفت دادن: زمان $O(t)$ ثابت اگرچه بزرگ]



درخت B

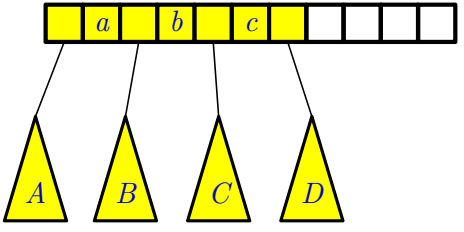
درج

INSERTING

اگر ریشه پر باشد، ابتدا آن را تقسیم می‌کنیم. حال درج را برای درخت‌های B با ریشه‌ی غیرپر تعریف می‌کنیم:

فرض می‌کنیم مقدار x باید بین b و c درج شود:

حالت ۲) C یک درخت B با ریشه‌ی غیر پر است: به‌صورت بازگشتی x را در C درج می‌کنیم.



Prepared by Kazim Fouladi | Fall 2016 | 2nd Edition
کاظم فولادی

درخت B

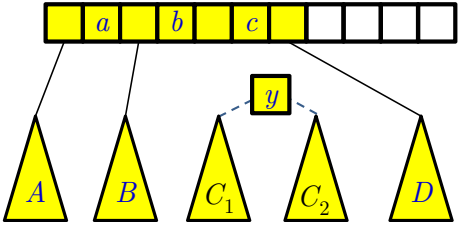
درج

INSERTING

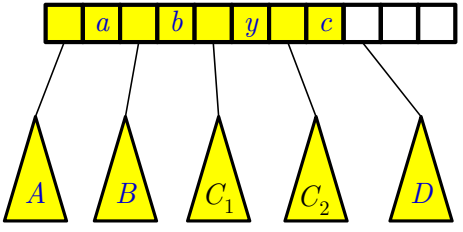
اگر ریشه پر باشد، ابتدا آن را تقسیم می‌کنیم. حال درج را برای درخت‌های B با ریشه‌ی غیرپر تعریف می‌کنیم:

فرض می‌کنیم مقدار x باید بین b و c درج شود:

حالت ۳) C یک درخت B با ریشه‌ی پر است: C را تقسیم می‌کنیم (با میانه‌ی y):



و کلید میانه‌ی y را در آرایه‌ی کلیدهای گره درج می‌کنیم.



و به‌صورت بازگشتی x را در C_1 $(x \le y)$ یا C_2 $(x > y)$ درج می‌کنیم.

$$O(h) = O(\lg n)$$

درخت B

درج : شبه‌کد

INSERTING

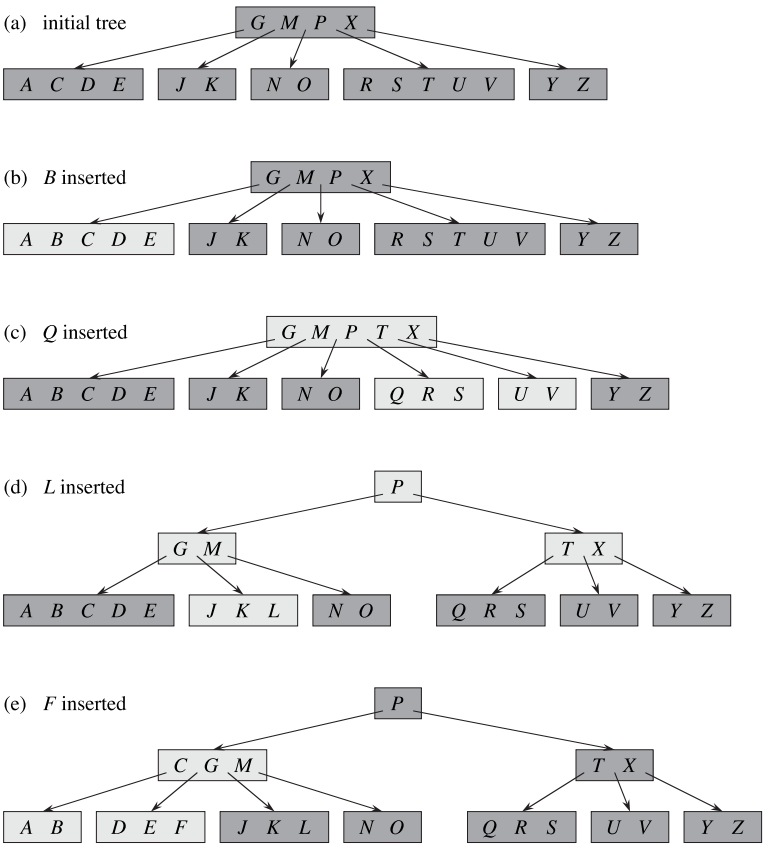
```
B-TREE-INSERT(T, k)
 1  r ← root[T]
 2  if n[r] = 2t − 1
 3     then s ← ALLOCATE-NODE()
 4          root[T] ← s
 5          leaf[s] ← FALSE
 6          n[s] ← 0
 7          c1[s] ← r
 8          B-TREE-SPLIT-CHILD(s, 1, r)
 9          B-TREE-INSERT-NONFULL(s, k)
10     else B-TREE-INSERT-NONFULL(r, k)
```

```
B-TREE-INSERT-NONFULL(x, k)
 1  i ← n[x]
 2  if leaf[x]
 3     then while i ≥ 1 and k < key_i[x]
 4               do key_{i+1}[x] ← key_i[x]
 5                  i ← i − 1
 6          key_{i+1}[x] ← k
 7          n[x] ← n[x] + 1
 8          DISK-WRITE(x)
 9     else while i ≥ 1 and k < key_i[x]
10               do i ← i − 1
11          i ← i + 1
12          DISK-READ(c_i[x])
13          if n[c_i[x]] = 2t − 1
14            then B-TREE-SPLIT-CHILD(x, i, c_i[x])
15                 if k > key_i[x]
16                   then i ← i + 1
17          B-TREE-INSERT-NONFULL(c_i[x], k)
```

درخت B

درج : مثال

INSERTING



درخت B

حذف

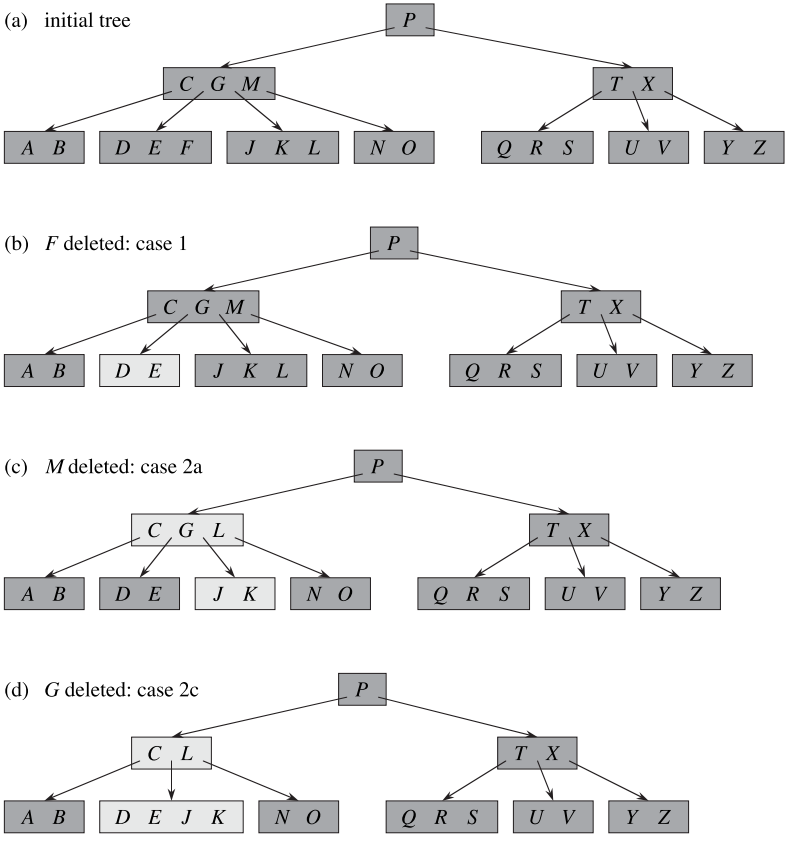
DELETING

1. If the key k is in node x and x is a leaf, delete the key k from x.
2. If the key k is in node x and x is an internal node, do the following.
   a. If the child y that precedes k in node x has at least t keys, then find the predecessor k' of k in the subtree rooted at y. Recursively delete k', and replace k by k' in x. (Finding k' and deleting it can be performed in a single downward pass.)
   b. Symmetrically, if the child z that follows k in node x has at least t keys, then find the successor k' of k in the subtree rooted at z. Recursively delete k', and replace k by k' in x. (Finding k' and deleting it can be performed in a single downward pass.)
   c. Otherwise, if both y and z have only $t - 1$ keys, merge k and all of z into y, so that x loses both k and the pointer to z, and y now contains $2t - 1$ keys. Then, free z and recursively delete k from y.
3. If the key k is not present in internal node x, determine the root $c_i[x]$ of the appropriate subtree that must contain k, if k is in the tree at all. If $c_i[x]$ has only $t - 1$ keys, execute step 3a or 3b as necessary to guarantee that we descend to a node containing at least t keys. Then, finish by recursing on the appropriate child of x.

درخت B

حذف: مثال (۱ از ۲)

DELETING

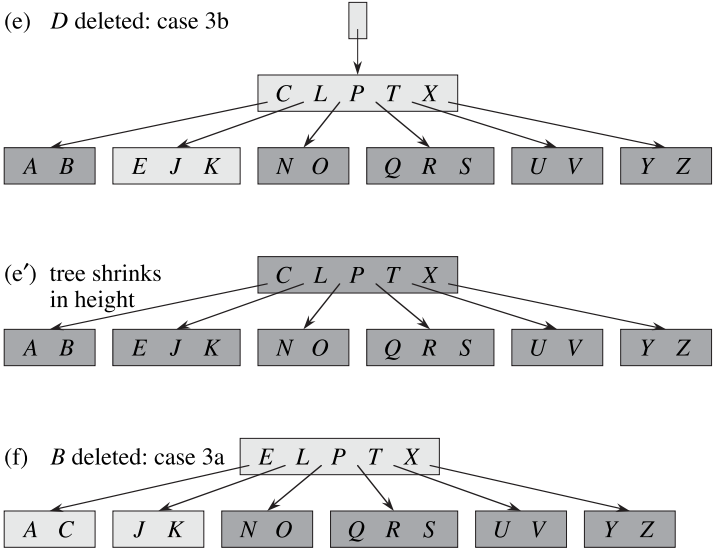


Prepared by Kazim Fouladi | Fall 2016 | 2nd Edition
کاظم فولادی

درخت B

حذف: مثال (۲ از ۲)

DELETING

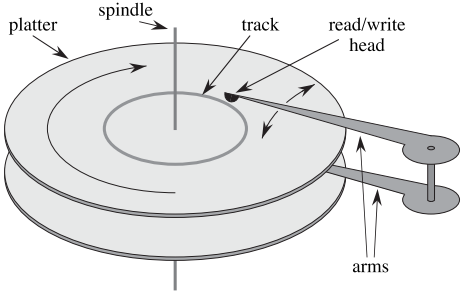


درخت B

کاربرد در حافظه‌ی ثانویه

B-TREE

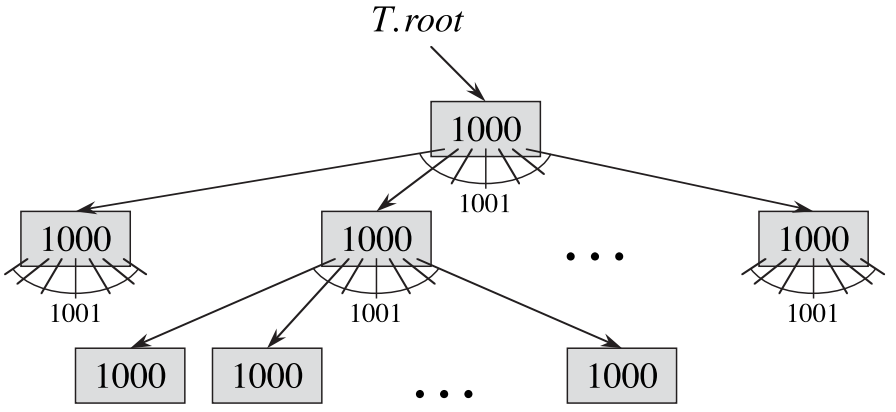
درخت‌های B با مقادیر بزرگ t برای پیاده‌سازی پایگاه‌های داده استفاده می‌شوند.
وقتی گره‌ها بر روی دیسک ذخیره می‌شوند،
می‌خواهیم تعداد گره‌هایی که در هنگام جستجو بررسی می‌شوند را به حداقل برسانیم؛
زیرا هر گره‌ی بیشتر می‌تواند هزینه‌ی گران و اضافی دسترسی به دیسک را داشته باشد.
به همین دلیل، می‌خواهیم ارتفاع به‌کندی رشد کند و این برای مقادیر بزرگ t حاصل می‌شود.



درخت B

کاربرد در حافظه‌ی ثانویه: مثال

B-TREE



1 node,
1000 keys

1001 nodes,
1,001,000 keys

1,002,001 nodes,
1,002,001,000 keys

A B-tree of height 2 containing over one billion keys. Each internal node and leaf contains 1000 keys. There are 1001 nodes at depth 1 and over one million leaves at depth 2. Shown inside each node x is $n[x]$, the number of keys in x.