دانشگاه کردستان

دانشکده مهندسی

گروه مهندسی کامپیوتر و فناوری اطلاعات

درس بازیابی اطلاعات

عنوان ارائه :

B-Tree

تهیه‌کنندگان :

ئاسو مفاخری و مسلم پرویزپور

نام استاد :

دکتر سلیمانی

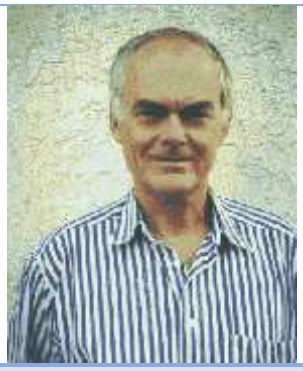
نیمسال دوم تحصیلی ۹۶-۹۷

فهرست

تعریف ۳
ویژگی‌ها ۳
کاربرد ۴
ارتفاع درخت ۵
عملیات در B-Tree ۵
عمل جستجو در B-Tree ۶
عمل درج در B-Tree ۷
عمل حذف در B-Tree ۱۱

تعریف

ساختمان داده B-Tree در سال ۱۹۷۲ توسط **"Bayer and McCreight"** با نام **"Height Balanced m-way Search Tree"** توسعه داده‌شد.

	
Edward Meyers McCreight	**Rudolf Bayer**

به طور کلی B-Tree یک درخت جستجوی متوازن چند سطحی است.

- ✓ B-Tree به معنای درخت متعادل (Balanced Tree) می‌باشد.
- ✓ در BST، AVL Tree، Red-Block Tree و... هر گره می‌تواند فقط یک مقدار و حداکثر دو فرزند داشته باشد اما در B-Tree چنین نیست.
- ✓ تعداد کلید ها و همچنین تعداد فرزندان هر گره به مرتبه B-Tree بستگی دارد. هر B-Tree یک مرتبه دارد.

ویژگی‌ها

B-Tree از مرتبه m ویژگی های زیر را دارد:

۱. ریشه درخت در بالاترین سطح قرار می‌گیرد.

۲. همه‌ی برگ ها (leaf nodes) باید در یک سطح قرار بگیرند.

۳. همه‌ی گره ها بجز ریشه باید حداقل [m/2]-1 و حداکثر m-1 کلید داشته باشند.

۴. همه‌ی گره ها بجز ریشه و برگ ها (intermediate level) باید حداقل m/2 فرزند داشته باشند.

۵. اگر ریشه درخت برگ نباشد (درخت بیشتر از یک سطح داشته باشد)؛ باید حداقل دو فرزند داشته باشد.

۶. همه‌ی گره ها بجز برگ ها که n-1 کلید دارند، باید n فرزند داشته باشند.

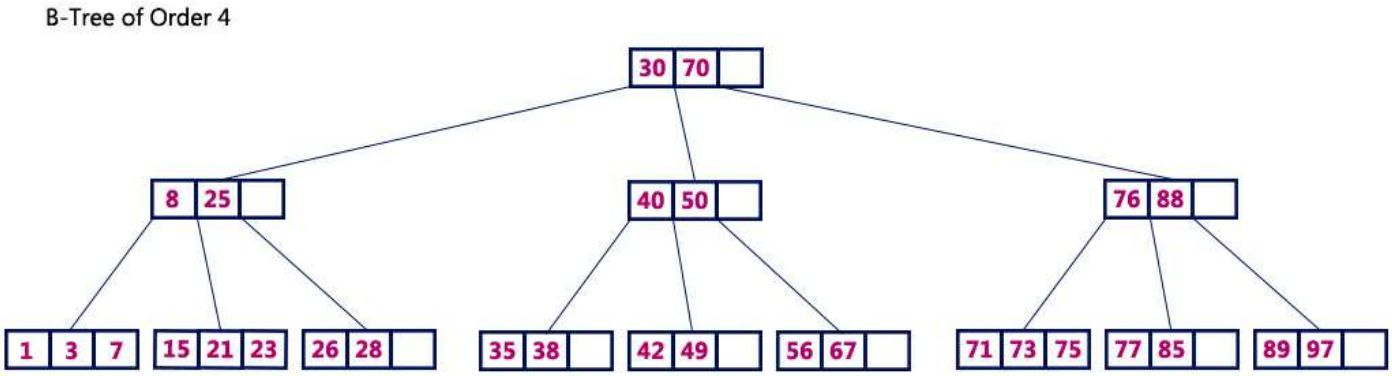
۷. کلید های یک گره باید به صورت صعودی مرتب شده باشند.

۸. عمل درج کلید فقط در برگ صورت می‌گیرد.

۹. نحوه تکمیل شدن B-Tree از پایین به بالا و خواندن آن از بالا به پایین می باشد.

۱۰. پیچیدگی زمانی عملیات B-Tree به صورت O(Logn) می‌باشد.

مثال) B-Tree از مرتبه ۴ دارای حداکثر ۳ کلید در یک گره و حداکثر ۴ فرزند برای یک گره می‌باشد.



کاربرد

ساختمان داده B-Tree مکانیزمی برای حفظ توازن درخت دودویی در حافظه فراهم می‌کند.

- ✓ ساختمان داده‌ی اصلی برای indexing در بسیاری از سیستم های مدیریت پایگاه داده می‌باشد.
  - CouchDB
  - SQL Server
  - Oracle
  - SQLite
  - MySQL
  - Realm Database
  - NoSQL
  - PostgreSQL
- ✓ در سیستم فایل‌های HFS+، NTFS، AIX(JFS2)، EXT4 و... .

ارتفاع درخت

- ✓ در بدترین حالت (هر گره حداقل کلید ممکن را داشته باشد)

$$h \leq \left\lfloor log_{\lceil \frac{n}{2} \rceil}(\frac{k+1}{2}) \right\rfloor$$

- ✓ در بهترین حالت (هر گره حداکثر کلید ممکن را داشته باشد)

$$h = \lceil log_n(k+1) \rceil - 1$$

K	تعداد کلید ها در درخت
n	حداکثرتعداد فرزندان یک گره

عملیات در B-Tree

عملیات زیر روی B-Tree انجام می‌شود.

۱. جستجو

۲. درج

۳. حذف

شبیه ساز آنلاین عملیات B-Tree

عمل جستجو در B-Tree

در B-Tree فرایند جستجو از ریشه آغاز می‌شود و هر بار تصمیم‌گیری n حالته داریم که n تعداد همه‌ی فرزندان گره است.

به صورت زیر:

1. Read the search element from the user
2. Compare, the search element with first key value of root node in the tree.
3. If both are matching, then display "Given node found!!!" and terminate the function
4. If both are not matching, then check whether search element is smaller or larger than that key value.
5. If search element is smaller, then continue the search process in left subtree.
6. If search element is larger, then compare with next key value in the same node and repeat step 3, 4, 5 and 6 until we found exact match or comparison completed with last key value in a leaf node.
7. If we completed with last key value in a leaf node, then display "Element is not found" and terminate the function.

```
function B-TREE-SEARCH(x, k) returns (y, i) such that key_i[y] = k or NIL
    i ← 1
    while i ≤ n[x] and k > key_i[x]
        do i ← i + 1
    if i ≤ n[x] and k = key_i[x]
        then return (x, i)
    if leaf[x]
        then return NIL
        else DISK-READ(c_i[x])
                return B-TREE-SEARCH(c_i[x], k)
```

عمل درج در B-Tree

در B-Tree عنصر جدید فقط باید به گره برگ اضافه شود.

به صورت زیر:

1. Check whether tree is Empty.
2. If tree is Empty, then create a new node with new key value and insert into the tree as a root node.
3. If tree is Not Empty, then find a leaf node to which the new key value cab be added using Binary Search Tree logic.
4. If that leaf node has an empty position, then add the new key value to that leaf node by maintaining ascending order of key value within the node.
5. If that leaf node is already full, then split that leaf node by sending middle value to its parent node. Repeat the same until sending value is fixed into a node.
6. If the splitting is occurring to the root node, then the middle value becomes new root node for the tree and the height of the tree is increased by one.

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

```
B-TREE-SPLIT-CHILD(x, i, y)
    z ← ALLOCATE-NODE()
    leaf[z] ← leaf[y]
    n[z] ← t − 1
    for j ← 1 to t − 1
        do key_j[z] ← key_{j+t}[y]
    if not leaf[y]
        then for j ← 1 to t
            do c_j[z] ← c_{j+t}[y]
    n[y] ← t − 1

    for j ← n[x] + 1 downto i + 1
        do c_{j+1}[x] ← c_j[x]
    c_{i+1}[x] ← z
    for j ← n[x] downto i
        do key_{j+1}[x] ← key_j[x]
    key_i[x] ← key_t[y]
    n[x] ← n[x] + 1
    DISK-WRITE(y)
    DISK-WRITE(z)
    DISK-WRITE(x)
```

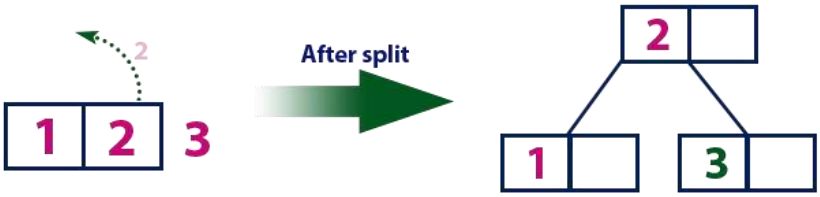
```
B-TREE-INSERT-NONFULL(x, k)
  i ← n[x]
  if leaf[x]
    then while i ≥ 1 and k < key_i[x]
           do key_{i+1}[x] ← key_i[x]
              i ← i − 1
         key_{i+1}[x] ← k
         n[x] ← n[x] + 1
         DISK-WRITE(x)
    else while i ≥ 1 and k < key_i[x]
           do i ← i − 1
         i ← i + 1
         DISK-READ(c_i[x])
         if n[c_i[x]] = 2t − 1
           then B-TREE-SPLIT-CHILD(x, i, c_i[x])
                if k > key_i[x]
                  then i ← i + 1
         B-TREE-INSERT-NONFULL(c_i[x], k)
```

مثال) ساخت B-Tree از مرتبه 3 با درج مقادیر ۱ تا ۱۰ .

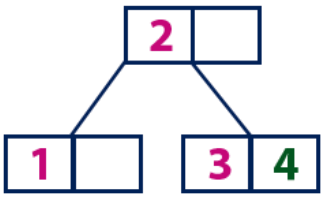
Insert (1):

Insert (2):

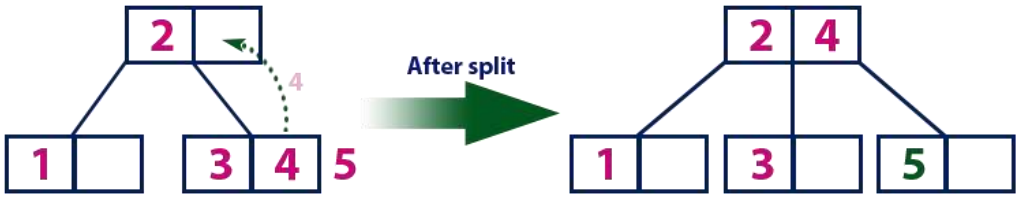
Insert (3):



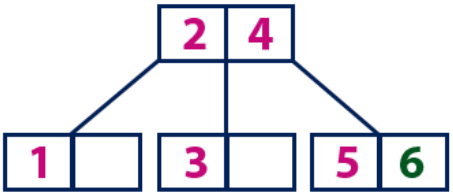
Insert (4):



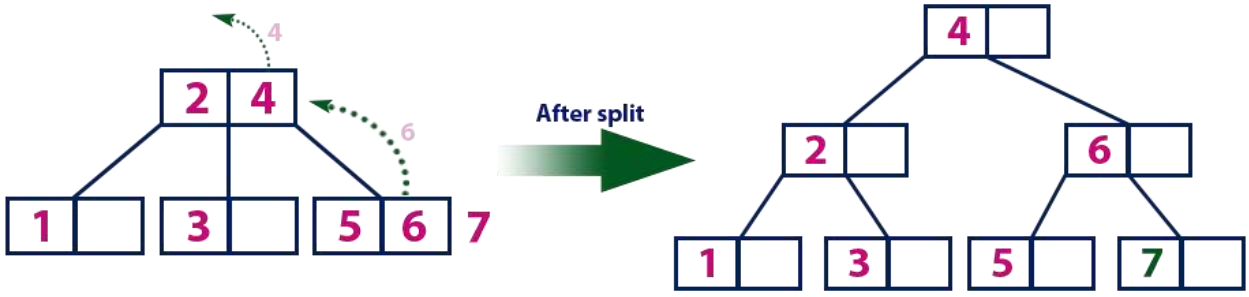
Insert (5):



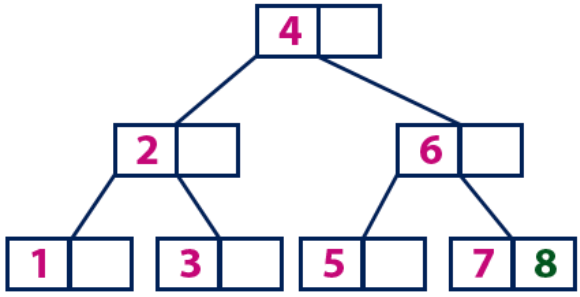
Insert (6):



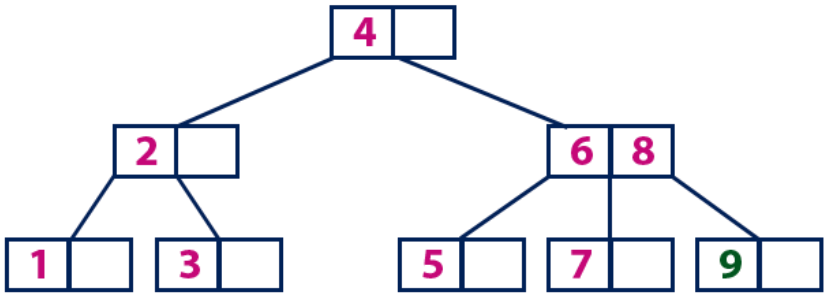
Insert (7):



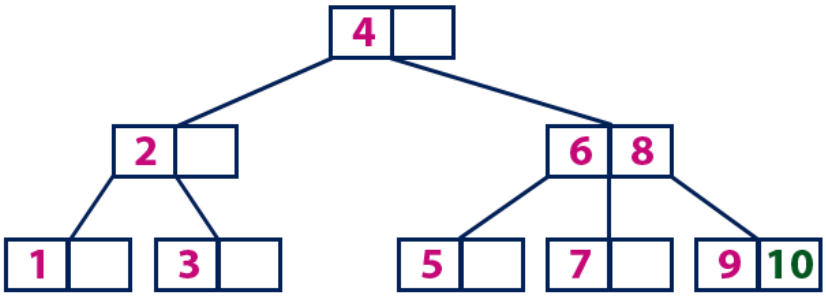
Insert (8):



Insert (9):



Insert (10):



عمل حذف در B-Tree

به صورت ریز:

1. x is a leaf and contains the key (it will have at least m keys). This is trivial - just delete the key.
2. x is an internal node and contains the key. There are three subcases:

   2.1. Predecessor child node has at least m keys

   2.2. Successor child node has at least m keys

   2.3. Neither predecessor nor successor child has m keys
3. x is an internal node, but doesn't contain the key. Find the child subtree of x that contains the key if it exists (call the child c). There are three subcases:

   3.1. c has at least m keys. Simply recourse to c.

   3.2. c has m−1 keys and one of its siblings has m keys.

   3.3. c and both siblings have m−1 keys.