Harshit Hiremath

1BM18CS036

ADS Lab 7

11/11/2020

B-Tree insertion :-

Algorithm:-

```
1) Initialize ~~a~~ (x ← root)
2) while x is non-leaf do
   → Find child 'y' of x going to be traversed next.
   → If y is not full, update ~~[illegible]~~ pointer in x = y
   → If y is full, split it & update pointer in x to
     point to one of the child.
        ⇒ while splitting, if k is smaller than ~~a~~ mid
          key of y, set pointer in x as first part of
          y.
        ⇒ else, second part.
3) Loop in 2) stops when x is leaf, insert k in it.
```

Utility functions :

```
void  BTreeNode :: traverse(){
    int i;
    for (i = 0; i<n; i++) {
        if (!leaf) c[i]->traverse();
        cout << ~~[illegible]~~ keys[i] << " ";
    }
    if (leaf == false) c[i] -> traverse();
}

void BTreeNode :: splitChild (int i, BTreeNode *y) {
    BTreeNode *z = new BTreeNode (y->t, y->leaf);
    for( int a =0 ;a<t-1; a++)  z->keys[a] = y->keys[a+t];

    if (y->leaf == false){
        for(int a=0; a<t; a++) z->c[a]=y->c[a+t];
    }

    y->n = t-1;

    for ( int a = n; a>= i+1; a--) ~~c~~c[a+1] = c[a];

    c[i+1] = z

    for( int  a = n-1; a>=i; a--)  keys[a+1]= keys[a];

    keys[i] = y->keys[t-1];

    n+=1;
}
void  BTreeNode::insertNonfull(int k){
      int i = n-1;
      if (leaf ==true){
          while( i >=0 && keys[i] > k){
              keys[i+1]= keys[i];
              i--;
          }
          keys[i+1]=k;
          n+=1;
      } else {
          while( i >=0 && keys[i] > k)  i--;
          if ( c[i+1] ->n == 2*t -1){
              splitChild( i+1, c[i+1]);
              if( keys[i+1] < k)  i++;
          }
          c[i+1] -> insertNonfull(k);
      }
}
```

[illegible]

Insert Function :

```
void  BTree:: insert(int k) {
    if (root == NULL) {
        root = new BTreeNode(t, true);
        root -> keys[0] = k;
        root -> n = 1;
    } else {
        if (root -> n = 2*t - 1) {
            BTreeNode *s = new BTreeNode(t, false);

            s -> C[0] = root
            s -> splitChild (0, root);
            int i = 0;
            if (s -> keys[0] < k)  i++;
            s -> C[i] -> insertNonFull(k);

            root = s;
        } else   root -> insertNonFull(k);

    }
}
```