Search for key \( R \)

Work = \( \Theta(h) = \Theta(\log n) \)

\( x \) ← a pointer to some object

- **DISK-READ\((x)\)**
  - Operations that access and/or modify the fields of \( x \)
  - Omitted if no fields of \( x \) were changed.
  - Other operations that access but do not modify fields of \( x \)

Each Disk-Read or Disk-Write

= one Basic unit of work \( O(1) \)

\( t - 1 \leq \# \text{Keys} \leq 2t - 1 \)
\( t \leq \# \text{Children} \leq 2t \)

\#Keys = 2t - 1 ⇒ node is full
\( t = 2 ⇒ 2 \cdot 3 \cdot 4 \) B-Tree

Each node is approx. one page of HD memory

**Thm:** Let \( T \) be a B-tree with \( n > 2 \) keys and of minimum degree \( t \geq 2 \). Then the height \( h \) of the B-tree is bounded above by

\[
h \leq \log \left( \frac{n + 1}{2} \right)
\]
\( \Theta(h) = \Theta(\log, n) \)

Work = \( \Theta(h) = \Theta(\log, n) \)

To INSERT, we will need procedures for splitting full nodes.
**B-Trees**

B-TREE-INSERT(T, k)
1. \( r \leftarrow \text{root}(T) \)
2. \( \text{if } n(r) = 2t - 1 \)
   3. \( s \leftarrow \text{ALLOCATE-NODE}() \)
   4. \( \text{root}(T) \leftarrow s \)
   5. \( \text{leaf}(s) \leftarrow \text{FALSE} \)
   6. \( n[s] \leftarrow 0 \)
   7. \( c_0[s] \leftarrow r \)
   8. \( \text{B-TREE-SPLIT-CHILD}(s, 1, r) \)
   9. \( \text{B-TREE-INSERT-NONFULL}(s, k) \)
else \( \text{B-TREE-INSERT-NONFULL}(r, k) \)

**B-Tree Split**

Inserting Keys in a B-Tree

We must make sure that the number of keys in a non-root node is always at least \( t \)
Deleting Keys in a B-Tree

(a) Initial tree

(b) Key deleted: case 1

(c) Key deleted: case 2

(d) Key deleted: case 2a

(e) Key deleted: case 2b

(f) Key deleted: case 2c

Deleting Keys in a B-Tree (Cont.)

(g) D deleted; case 3b

(h) Tree shrinks in height

(i) A, C, J, X