Introduction to Multilevel Indexing and B-Trees

Reference: Chapters 9.1-9.6

Problems with **simple indexes** that are kept in disk:

1. Seeking the index is still slow (binary searching):

   We don't want more than 3 or 4 seeks for a search.
   So, here $\log_2(N+1)$ is still slow:

N	$\log_2(N+1)$
15 keys	4
1,000	~10
100,000	~17
1,000,000	~20

2. Insertions and deletions should be as fast as searches:
   In simple indexes, insertion or deletion take O(n) disk accesses (since index should be kept sorted)

Indexing with Binary Search Trees

We could use **balanced** binary search trees:

- **AVL Trees**
  Worst-case search is $1.44 \log_2(N+2)$
  1,000,000 keys $\rightarrow$ 29 levels
  Still prohibitive...

- **Paged Binary Trees**
  Place subtrees of size K in a single page
  Worst-case search is $\log_{K+1}(N+1)$
  K=511, N=134,217,727
  Binary trees: 27 seeks
  Paged binary tree: 3 seeks
  This is good but there are lots of difficulties in **maintaining** (doing insertions and deletions in) a paged binary tree.

Multilevel Indexing

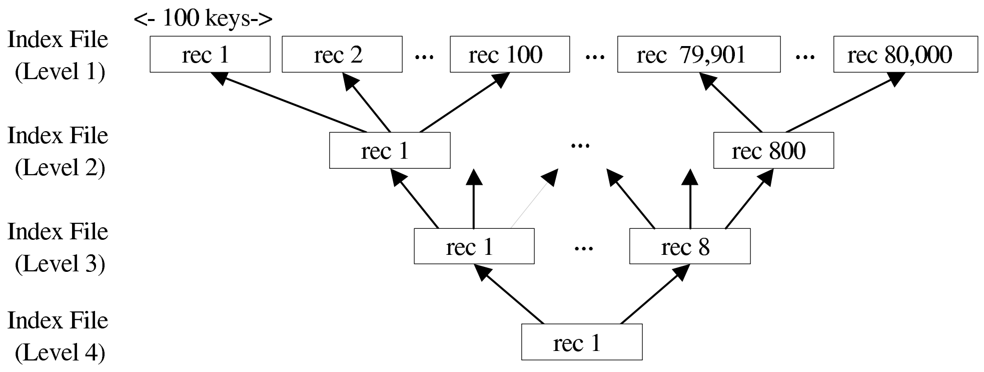
Consider our 8,000,000 example with keysize = 10 bytes.
Index file size = 80 MB
Each **record** in the **index** will contain 100 pairs (key, reference)
A **simple index** would contain: 80,000 records. Too expensive to search (still ∼ 16 seeks)

Multilevel Index

Build an index of an index file:

How:

- Build a simple index for the file sorting keys using the method for external sorting seen last class.
- Build an index for this index.
- Build another index for the previous index, and so on.

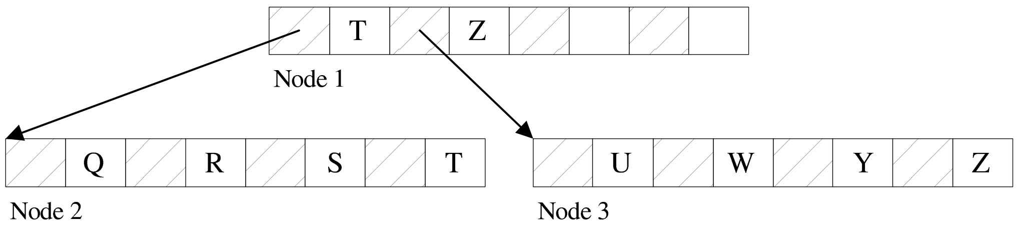


Note: That the index of an index stores the largest key in the record it is pointing to.

B-Trees - Working Bottom-Up

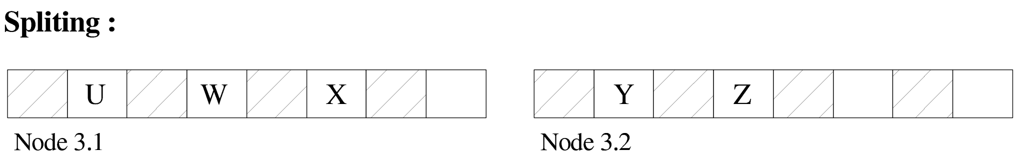
- Again an index record may contain 100 keys.
- An index record may be half full (each index record may have from 50 to 100 keys).
- When insertion in an index record causes it to overfull:
  - Split record in two
  - "Promote" the largest key in one of the records to the upper level

Example for order = 4 (instead of 100).

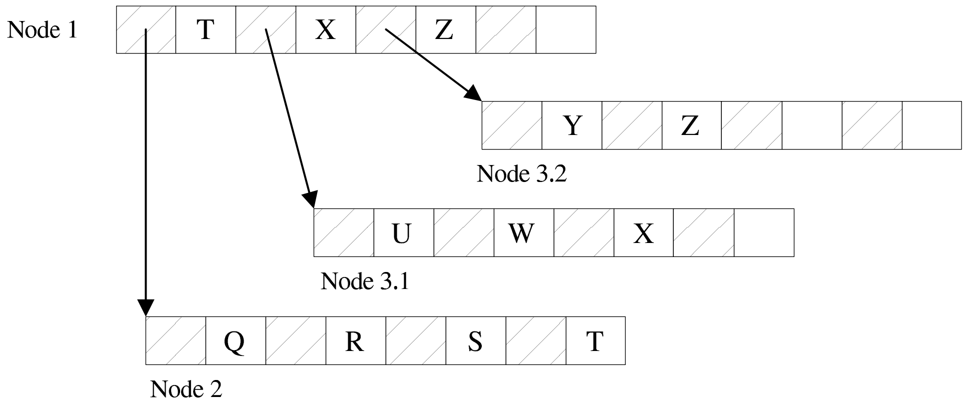


Inserting X

X is between T and Z: insertion in node 3 splits it and generates a promotion of node X.



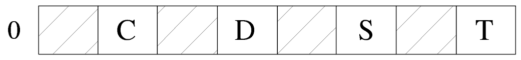
Promoting largest of Node 3.1.



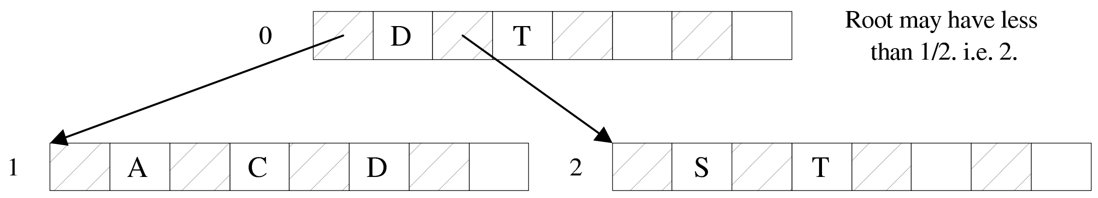
Important: If Node 1 was full, this would generate a new split-promotion of Node 1. This could be propagated up to the root.

An example showing insertions:

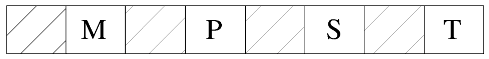
Inserting keys: order = 4
C, S, D, T, A, M, P, I, B, W, N, G, U, R, K, E, H, O, L, J, Y, Q, Z, F, X, V



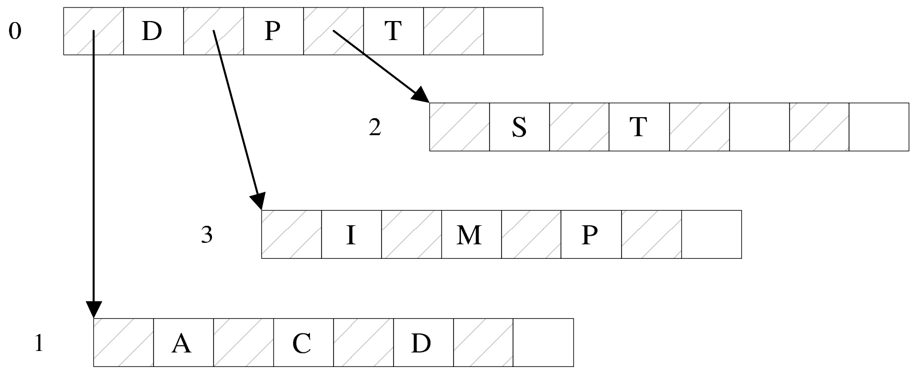
Inserting A: Split and promotion



Inserting M,P only changes Node 2 to :



But inserting "I" Splits and promotes "P"



There is room for one more node at level 2. After that the root may get split!

Complete the above example as an exercise.

Important: At each level, no more than 2 nodes are affected. We got search and updates with cost equal to the height of the tree !!!