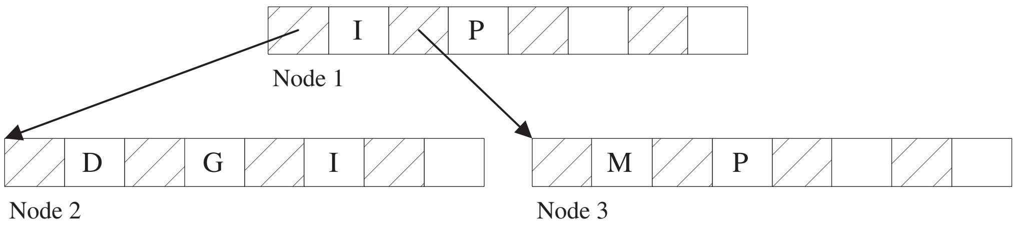
B-Trees

Last Time: Chapters 9.5, 9.6

Today: Chapters 9.8, 9.9, 9.10, 9.11, 9.12

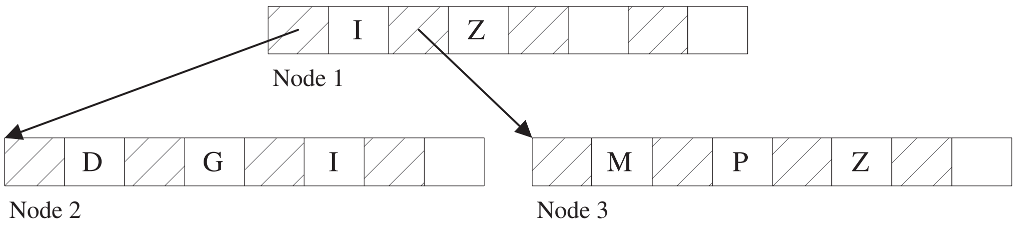
Note regarding insertions in B-trees

Special case of larger key:



Inserting Z

Z is larger than P but P is larger in Node 1, so the place for Z is Node 3.



B-Tree Properties

Properties of a B-tree of order m:

1. Every node has a maximum of m children.
2. Every node, except for the root and the leaves, has at least $\lceil m/2 \rceil$ children.
3. The root has at least two children (unless it is a leaf).
4. All the leafs appear on the same level.
5. The leaf level forms a complete index of the associated data file.

Worst-case search depth

The worst-case depth occurs when every node has the minimum number of children.

Level	**Minimum number of keys (children)**
1 (root)	2
2	$2 \cdot \lceil m/2 \rceil$
3	$2 \cdot \lceil m/2 \rceil \cdot \lceil m/2 \rceil = 2 \cdot \lceil m/2 \rceil^2$
4	$2 \cdot \lceil m/2 \rceil^3$
...	...
d	$2 \cdot \lceil m/2 \rceil^{d-1}$

If we have N keys in the leaves:

$$N \geq 2 \cdot \lceil m/2 \rceil^{d-1}$$

So, $d \leq 1 + \log_{m/2}(N/2)$

For $N = 1,000,000$ and order $m = 512$, we have

$$d \leq 1 + \log_{256} 500,000$$

$$d \leq 3.37$$

There is at most 3 levels in a B-tree of order 512 holding 1,000 000 keys.

Outline of Search and Insert algorithms

`Search (keytype key)`

1) Find leaf: find the leaf that could contain `key`, loading all the nodes in the path from root to leaf into an array in main memory.

2) Search for `key` in the leaf which was loaded in main memory.

`Insert (keytype key, int datarec_address)`

1) Find leaf (as above).

2) Handle special case of new largest key in tree: update largest key in all nodes that have been loaded into main memory and save them to disk.

3) Insertion, overflow detection and splitting on the update path:
   `currentnode` = leaf found in step 1)
   `recaddress` = `datarec_address`

   3.1) Insert the pair (`keys, recaddress`) into `currentnode`.

   3.2) If it caused overflow

   - Create `newnode`
   - Split contents between `newnode, currentnode`
   - Store `newnode, currentnode` in disk
   - If no parent node (root), go to step 4)
   - `currentnode` becomes parent node
   - `recaddress` = address in disk of `newnode`
   - `key` = largest key in new node
   - Go back to 3.1)

4) Creation of a new root if the current root was split.

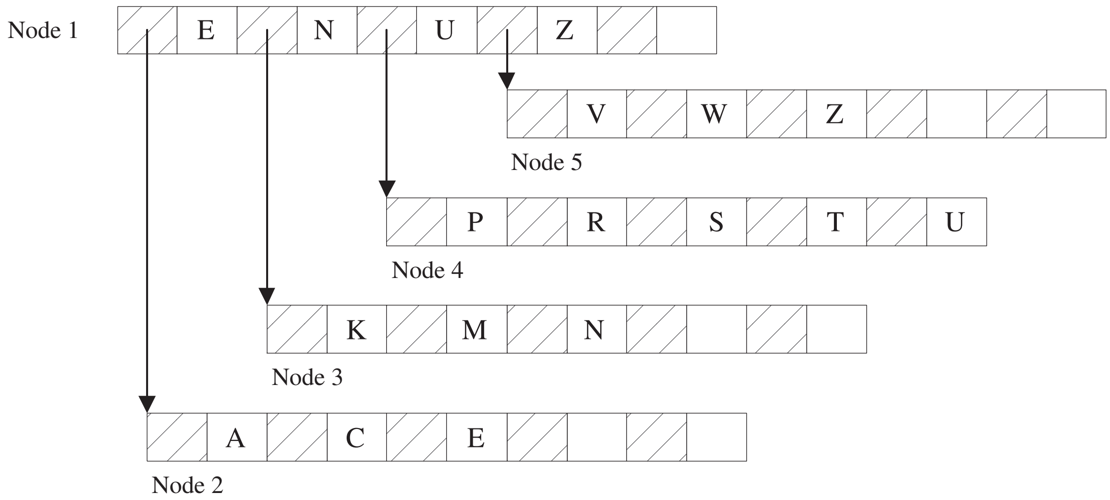
   Create root node pointing to `newnode` and `currentnode`. Save new root to disk.

Deletions in B-Trees

The rules for deleting a key K from a node n in a B-tree:

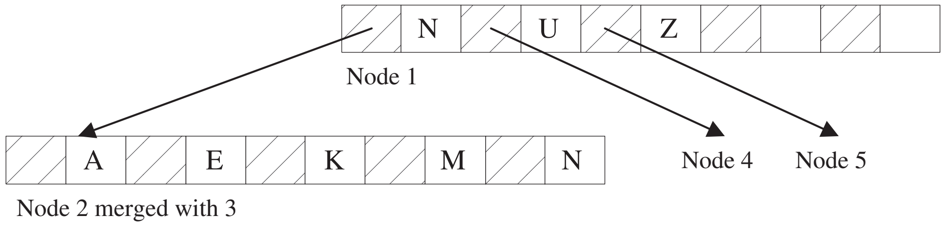
1. If n has more than the minimum number of keys and K is not the largest key in n, simply delete K from n.
2. If n has more than the minimum number of keys and K is the largest key in n, delete K from n and modify the higher level indexes to reflect the new largest key in n.
3. If n has exactly the minimum number of keys and one of the siblings has "few enough keys", **merge** n with its sibling and delete a key from the parent node.
4. If n has exactly the minimum number of keys and one of the siblings has extra keys, **redistribute** by moving some keys from a sibling to n, and modify higher levels to reflect the new largest keys in the affected nodes.

Consider the following example of a B-tree of **order 5** (minimum allowed in node is 3 keys)

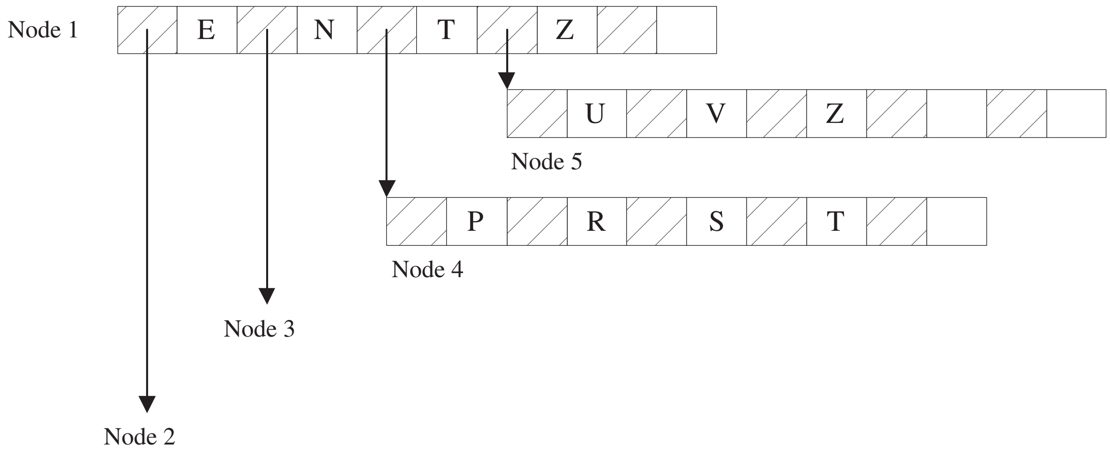


We consider the following 5 alternative modifications on the previous tree:

- Deleting "T" falls into case 1)
- Deleting "U" falls into case 2)
- Deleting "C" falls into case 3)



- Deleting "W" falls into case 4)



- Deleting "M" allows for two possibilities: case 3) or 4)
  - Merge Node 3 with Node 2; or
  - Redistribute keys between Node 3 and Node 4

Note that "sibling" here refers only to nodes that have the same parent and are **next to each other**.