B-Trees I

- definition
- example
- types
- search
- insertion

B-Tree introduction

- a search structure, i.e., yet another possible representation for the Table ADT.
- example of a balanced search tree: O(log n) search time *worst* case.
- other balanced search trees: AVL trees, red-black trees.
- Well suited for *large* amounts of data (on disk), such as for a database application.
  (B+ Trees is a variation commonly used)
- Most DBMS use them to represent data.

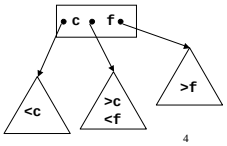
Balanced search structures: main idea

- search tree with extra properties that limit its height (so search will be log n)
- means insert and delete are more complicated, because the resulting trees have to satisfy the extra properties
- but, to keep it fast, insert and delete also need to be log n.

B-Tree: definition

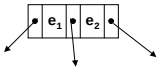
A B-Tree of order m is an m-way tree such that:

1. non-leaf nodes store up to (m-1) entries that partition the data in the subtrees in a way similar to a BST (see diagram below)
   - the number of entries is one less than the number of children
   - the entries are sorted from smallest to largest
2. non-leaf nodes (except root): have between $\lceil m/2 \rceil$ and m children.
3. all nodes (except root): have between $\lceil m/2 \rceil$-1 and m-1 entries
4. root (special case):
   - either it's a leaf with at least one entry
   - or it has between 2 and m children.
5. all leaves are at the same depth



Example: 5-way B-Tree

- a.k.a. B-Tree of order 5
- M = 5
- Main and Savitch call this a B-Tree with MAXIMUM = 4 (i.e, max entries)



nodes (except root)

3-5 children (unless leaf)

2-4 entries (i.e., <key,value>)

root

2-5 children (unless it's a leaf)

1-4 keys

5-way B-Tree (cont)

Example type definition for a node:

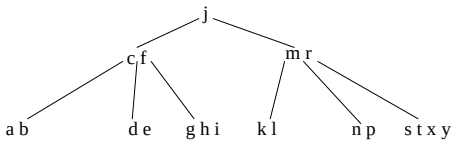
```
const int MAXCHILD = 5;

class Bnode {
 private:
  friend class Table;
  int nKeys;
  EntryType entry[MAXCHILD-1];
  Bnode *branch[MAXCHILD];
};

typedef Bnode *BTreeType;
```

B-Tree: searching

- keys in interior nodes partition subtrees just like BST nodes do
- Example B-Tree: (keys are letters)



B-Tree: insertion

1. search terminates in failure at a leaf.
2. add a new value to leaf in sorted order.
3. if node has more than M-1 keys then split the node.
   How to split:
   1. middle value gets pushed up into the parent
   2. first half of values in a new left node.
   3. second half of values in right node.
4. If the parent is now bigger than M, split it recursively.

Note$_1$: if this happens at the root, the root splits in two and get a new root with only 1 key and 2 children. This is the only time we increase the number of levels in the tree.

Note$_2$: node splits produce two half full nodes; thus we won't have to split again for a while.

Insertion: example

- Start with an empty 5-way B-tree and insert the following keys in the order shown (should end up with tree from slide 8):

```
a  g  f  b  k  d  h  m
j  e  s  i  r  x  c  l
n  t  y  p
```


B-Tree variations

- 2-3 tree
  - 2-3 tree is a name for the special case of a 3-way B-Tree (all nodes have between 2 and 3 children)
  - Main and Savitch examples where MAX = 2 is a 2-3 tree.
- only store data at the leaves
  - another B-Tree formulation is one where interior nodes store keys, but all actual entries are in leaf nodes (more about this later)