

# B-trees

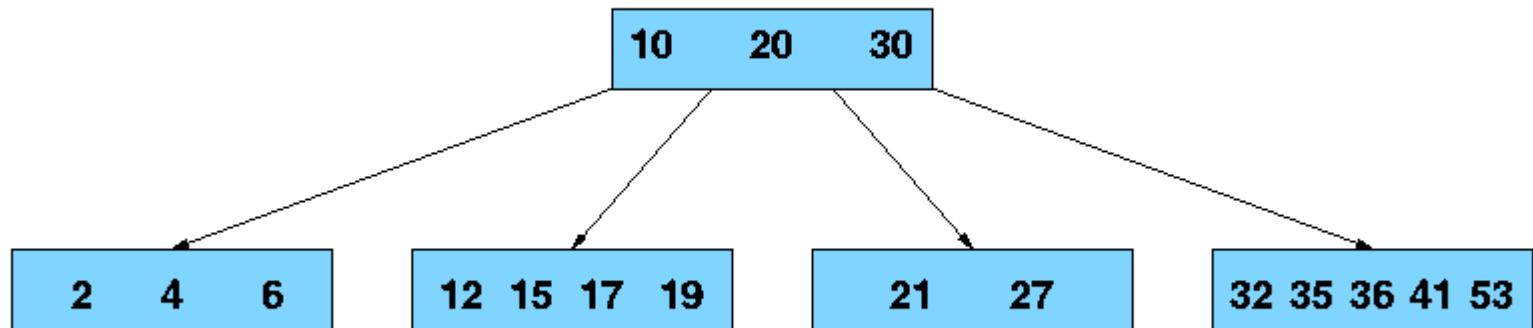
[anhtt-fit@mail.hut.edu.vn](mailto:anhtt-fit@mail.hut.edu.vn)

# B-Tree

- Generalizes 2-3-4 trees by allowing up to  $M$  links per node.
- Main application: file systems.
  - Reading a page into memory from disk is expensive.
  - Accessing info on a page in memory is free.
  - Goal: minimize # page accesses.
  - Node size  $M$  = page size.
- Space-time tradeoff.
  - $M$  large ! only a few levels in tree.
  - $M$  small ! less wasted space.
  - Number of page accesses is  $\log_M N$  per op.
  - Typical  $M = 1000$ ,  $N < 1$  trillion.

# Search

**B-Tree: Minimization Factor  $t = 3$ , Minimum Degree = 2, Maximum Degree = 5**



**Search(21)**

# Insert



# B-Tree in the wild

- Red-black trees: widely used as system symbol tables
  - Java: `java.util.TreeMap`, `java.util.TreeSet`.
  - C++ STL: `map`, `multimap`, `multiset`.
  - Linux kernel: `linux/rbtree.h`.
- B-Trees: widely used for file systems and databases
  - Windows: HPFS.
  - Mac: HFS, HFS+.
  - Linux: ReiserFS, XFS, Ext3FS, JFS.
  - Databases: ORACLE, DB2, INGRES, SQL, PostgreSQL
- All nodes in B-Tree are assumed to be stored in secondary storage (disk) rather than primary storage (memory),
- There basic operations for accessing a page: *Disk-Read()*, *Disk-Write()*, *Allocate-Node()*

# B-Tree Library

- Software and documentation is accessed at <http://www.hydrus.org.uk/doc/bt/html/index.htm>  
|

# API

- Creating a B Tree File

BTA\* btcrt(char\* *fid*, int *nkeys*, int *shared*);

- Opening a B Tree File

BTA\* btopn(char\* *fid*, int *mode*, int *shared*);

- Closing a B Tree File

int btcls(BTA\* *btact*);

# API (cont.)

- Inserting a key and data

```
int btins(BTA* btact, char* key, char* data, int dsize);
```

- Updating data for an existing key

```
int btupd(BTA* btact, char* key, char* data, int dsize);
```

- Locating data for an existing key

```
int btisel(BTA* btact, char* key, char* data, int dsize, int*  
rsiz);
```

- Deleting a key and associated data

```
int btidel(BTA* btact, char* key);
```

- Locating data for the next key in sequence

```
int btiseln(BTA* btact, char* key, char* data, int dsize, int*  
rsiz);
```



# Building and installing the BT Library

- Unpack the tar file into a convenient directory.

```
$cd <bt library>
```

```
$make clean
```

```
$make
```

- Make built an UNIX static library **libbt.a**, a BT test harness **bt**, and a utility, **kcp**, which performs intelligent copies of BT index files.

# Quiz 1

- Install and compile BT Library in your machine
- Run BT test harness to verify if successful installed
- See documentation at  
<http://www.hydrus.org.uk/doc/bt/html/ch05.htm>

## Quiz 2

- Use the BT library to write a phone book program that manipulates data on the secondary disk.

# Another library for B-Tree

- Download at  
[http://www.mycplus.com/utilitiesdetail.asp?  
iPro=10](http://www.mycplus.com/utilitiesdetail.asp?iPro=10)
- This library allows specifying different comparison functions for keys.

# Mini project 1

- Make a program to manage a computer dictionary
  - Add/Search/Delete a word (using B-Tree)
  - Auto complete search. Ex. When we enter “comput” and <tab>, the word “computer” should be auto completed (like in Bash Shell)
  - Suggestion search => Use soundex library
- Please test the performance of your program with a dictionary of millions words (the words can be randomly created)
  - Test for the two basic operations: search and insert
- Project in group of 3-4 persons