B+ Trees

Today : Chapters 10.1 - 10.5

Motivation

Some applications require two views of a file :

Indexed view :	**Sequential view :**
Records are indexed by a key	Records can be sequentially accessed in order by key
Direct, indexed access	Sequential access (physically contiguous records)
Interactive, random access	Batch processing (Ex: co-sequential processing)

Example of applications

- Student record system in a university :

  - Indexed view : access to individual records

  - Sequential view : batch processing when posting grades or when fees are paid

- Credit card system :

  - Indexed view : interactive check of accounts

  - Sequential view : batch processing of payment slips

We will look at the following two aspects of the problem :

1. Maintaining a **sequence set** : keeping records in sequential order
2. Adding an **index set** to the sequence set

Maintaining a Sequence Set

Sorting and re-organizing after insertions and deletions is out of question. We organize the sequence set in teh following way:

- Records are grouped in **blocks**
- Blocks should be **at least half full**.
- **Link fields** are used to point to the preceding block and the following block (similarly to doubly linked lists)
- Changes (insertion/deletion) are localized into blocks by performing :

  - **Block Splitting** when **insertion** causes overflow

  - **Block Merging** or **Redistribution** when **deletion** causes underflow

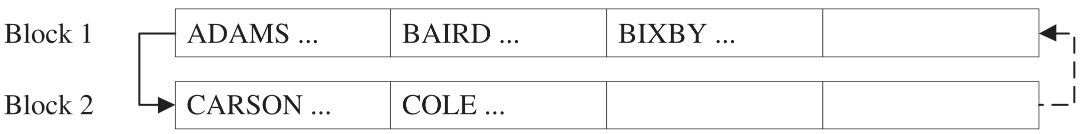
Example:

Block size = 4
key : **Last Name**

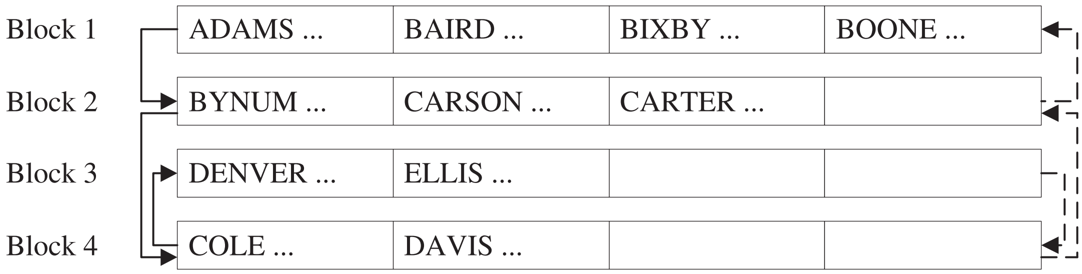
- **Insertion with overflow:**

Block 1	ADAMS ...	BIXBY ...	CARSON ...	COLE ...

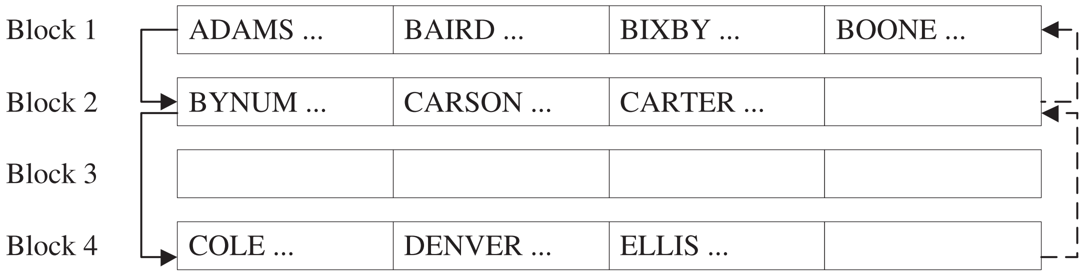
Insert "BAIRD ..."



- **Deletion with merging:**



Delete "DAVIS ..." (Merging)

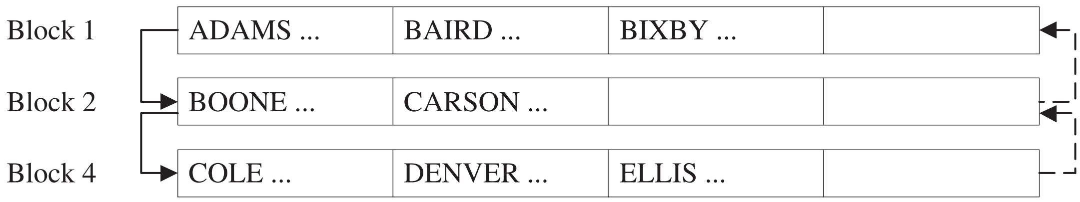


Block 3 is available for re-use

Delete ‘BYNUM’:
Just remove it from Block 2

Then, delete ‘CARTER’ :
We can either merge Block 2 and 4 or redistribute records among Blocks 1 and 2.

- **Deletion with redistribution:**



When previous and next blocks are full then redistribution is the only option.

Advantages and disadvantages of the scheme described

Advantages:
- No need to re-organize the whole file after insertions/deletions.

Disadvantages:
- File takes more space than unblocked files (since blocks may be half full).
- The order of the records is not necessarily **physically** sequential (we only guarantee physical sequentiality within a block).

Choosing Block Size

Consider :
- Main memory constraints (must hold at least 2 blocks)
- Avoid seeking within a block (Ex: in sector formatted disks choose block size equal to cluster size).

Adding an Index Set to the Sequential Set

Index Containing Separators Instead of Keys

Choose the **Shortest Separator** (a prefix)

```
Block     Range of Keys  Separator
-----     -------------  ---------
  1       ADAMS - BERNE
                         BO
  2       BOLEN - CAGE
                         CAM
  3       CAMP  - DUTTON
                         E
  4       EMBRY - EVANS
                         F
  5       FABER - FOLK
                         FOLKS
  6       FOLKS - GADDIS
```

To find a separator for `key1`= "CAGE" and `key2` ="CAMP" find the smallest prefix of `key2` that is not a prefix of `key1`, which is "CAM" in this example.

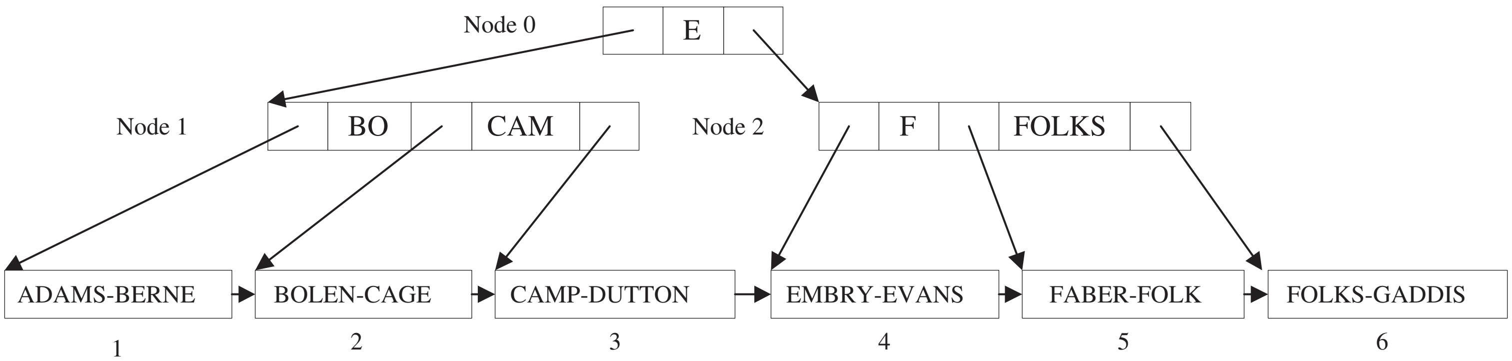
The Simple Prefix B+ Tree

The **simple prefix B+ tree** consists of :
- the **sequence set** (as seen previously).
- the **index set**: similar to a B-tree index, but storing the shortest separators (prefixes) for the sequence set.

Note : If a node contains N separators, it will contain N+1 children. The fact that we are dealing with separators slightly modifies the operations in the B-tree index.

Example :

Order of the index set is 3 (i.e. maximum of 2 separators and 3 children).
Note: The order is usually much larger, but we made it small for this example.



Search in a simple prefix B+ tree: Search for "EMBRY":

- Retrieve Node 0 (root).
- "EMBRY" > "E", so go right, and retrieve Node2.
- Since "EMBRY" < "F" go left, and retrieve block number 4.
- Look for the record with key "EMBRY" in block number 4.