Lecture 19:
B-trees and Hash Tables

Friday, November 15, 2002

1

Outline

- B-trees (13.3)
- Hash-tables (13.4)

2

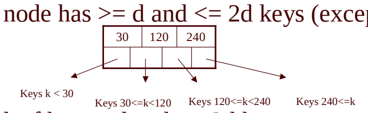
B+ Trees

- Search trees
- Idea in B Trees:
  - make 1 node = 1 block
- Idea in B+ Trees:
  - Make leaves into a linked list (range queries are easier)

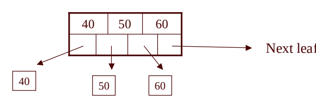
3

B+ Trees Basics

- Parameter d = the ***degree***
- Each node has >= d and <= 2d keys (except root)



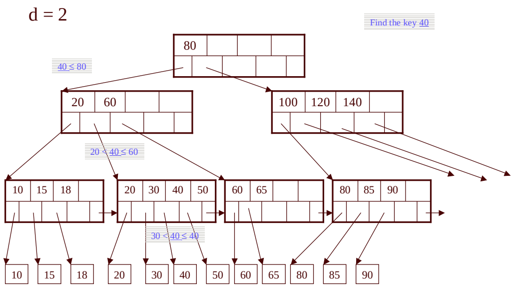
- Each leaf has >=d and <= 2d keys:



4

B+ Tree Example

d = 2



5

B+ Tree Design

- How large d ?
- Example:
  - Key size = 4 bytes
  - Pointer size = 8 bytes
  - Block size = 4096 byes
- 2d x 4 + (2d+1) x 8 <= 4096
- d = 170

6

Searching a B+ Tree

- Exact key values:
  - Start at the root
  - Proceed down, to the leaf

```
Select name
From people
Where age = 25
```

- Range queries:
  - As above
  - Then sequential traversal

```
Select name
From people
Where 20 <= age
  and  age <= 30
```

7

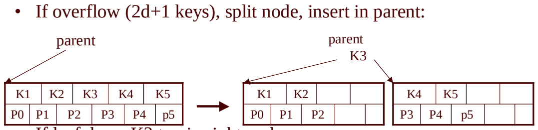
B+ Trees in Practice

- Typical order: 100. Typical fill-factor: 67%.
  - average fanout = 133
- Typical capacities:
  - Height 4: 133^4 = 312,900,700 records
  - Height 3: 133^3 = 2,352,637 records
- Can often hold top levels in buffer pool:
  - Level 1 = 1 page = 8 Kbytes
  - Level 2 = 133 pages = 1 Mbyte
  - Level 3 = 17,689 pages = 133 MBytes

Insertion in a B+ Tree

Insert (K, P)

- Find leaf where K belongs, insert
- If no overflow (2d keys or less), halt
- If overflow (2d+1 keys), split node, insert in parent:

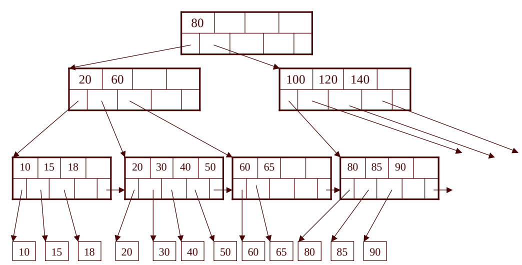


- If leaf, keep K3 too in right node
- When root splits, new root has 1 key only

9

Insertion in a B+ Tree

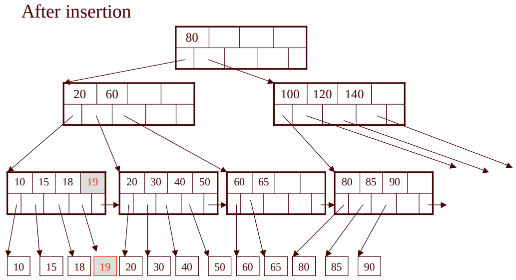
Insert K=19



10

Insertion in a B+ Tree

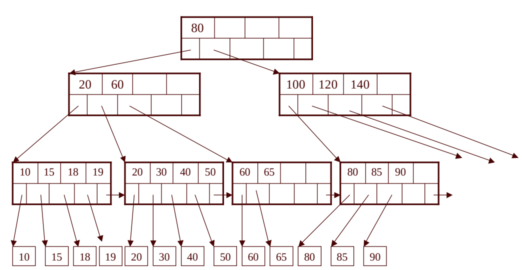
After insertion



11

Insertion in a B+ Tree

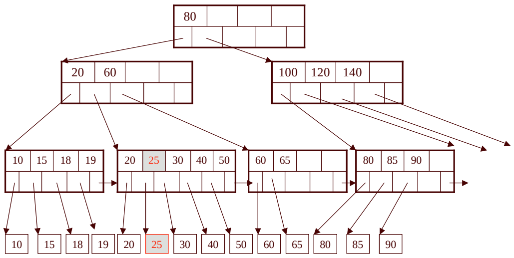
Now insert 25



12

Insertion in a B+ Tree

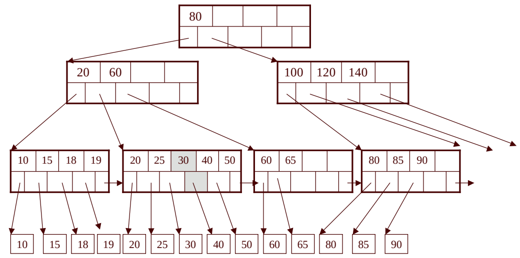
After insertion



13

Insertion in a B+ Tree

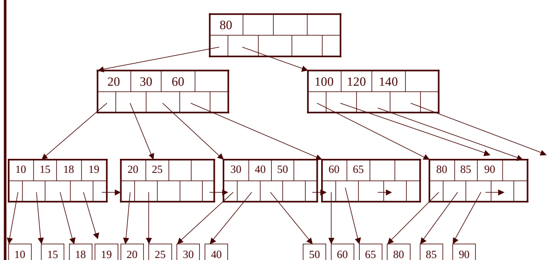
But now have to split !



14

Insertion in a B+ Tree

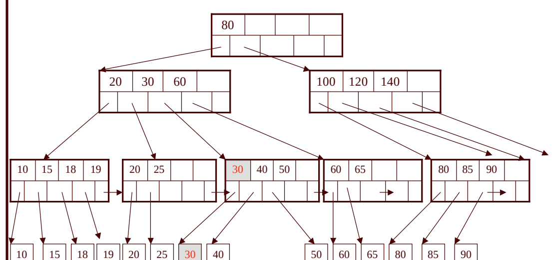
After the split



15

Deletion from a B+ Tree

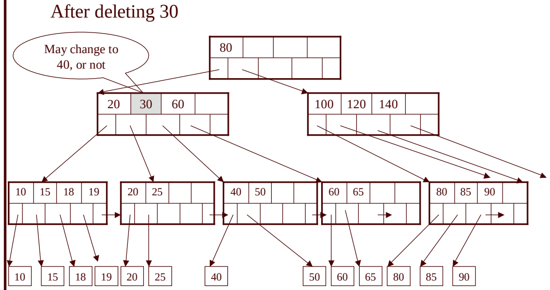
Delete 30



16

Deletion from a B+ Tree

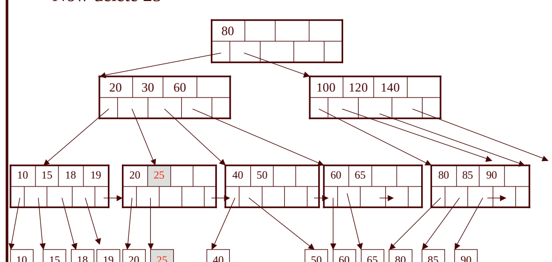
After deleting 30



17

Deletion from a B+ Tree

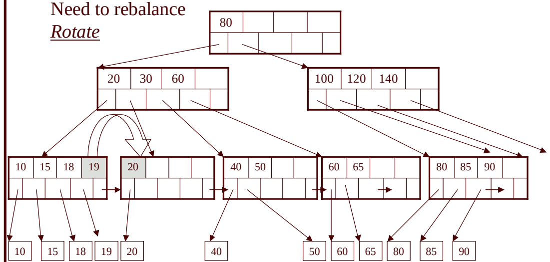
Now delete 25



18

Deletion from a B+ Tree

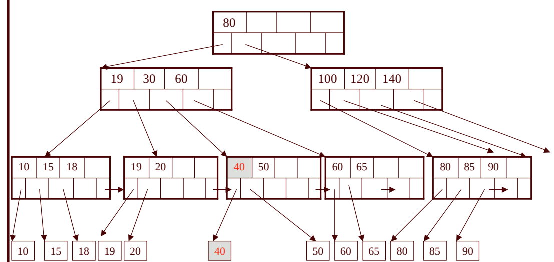
After deleting 25
Need to rebalance
Rotate



19

Deletion from a B+ Tree

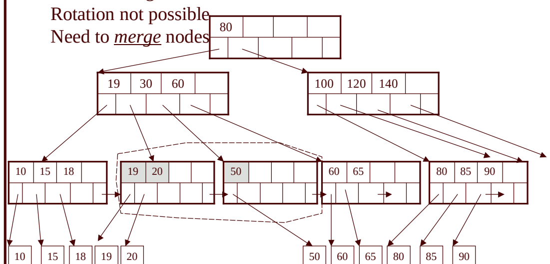
Now delete 40



20

Deletion from a B+ Tree

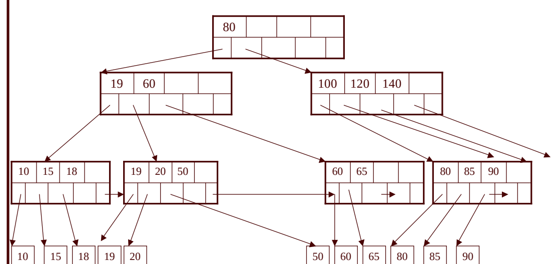
After deleting 40
Rotation not possible
Need to *merge* nodes



21

Deletion from a B+ Tree

Final tree



22

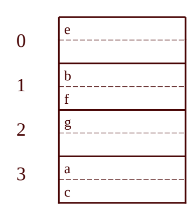
Hash Tables

- Secondary storage hash tables are much like main memory ones
- Recall basics:
  - There are n *buckets*
  - A hash function f(k) maps a key k to {0, 1, …, n-1}
  - Store in bucket f(k) a pointer to record with key k
- Secondary storage: bucket = block, use overflow blocks when needed

23

Hash Table Example

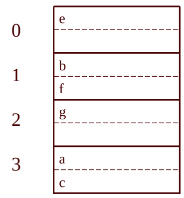
- Assume 1 bucket (block) stores 2 keys + pointers
- h(e)=0
- h(b)=h(f)=1
- h(g)=2
- h(a)=h(c)=3



24

Searching in a Hash Table

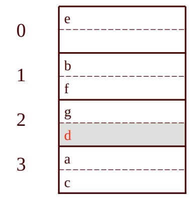
- Search for a:
- Compute h(a)=3
- Read bucket 3
- 1 disk access



25

Insertion in Hash Table

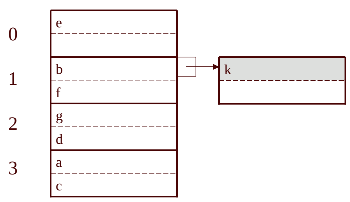
- Place in right bucket, if space
- E.g. h(d)=2



26

Insertion in Hash Table

- Create overflow block, if no space
- E.g. h(k)=1



- More over-flow blocks may be needed

27

Hash Table Performance

- Excellent, if no overflow blocks
- Degrades considerably when number of keys exceeds the number of buckets (I.e. many overflow blocks).

28

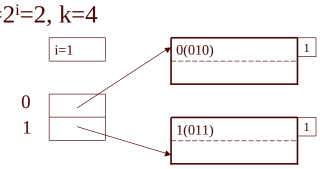
Extensible Hash Table

- Allows has table to grow, to avoid performance degradation
- Assume a hash function h that returns numbers in $\{0, \ldots, 2^k - 1\}$
- Start with $n = 2^i << 2^k$, only look at first i most significant bits

29

Extensible Hash Table

- E.g. $i=1$, $n=2^i=2$, $k=4$

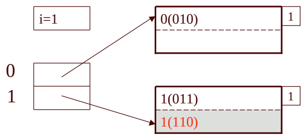


- Note: we only look at the first bit (0 or 1)

30

Insertion in Extensible Hash Table

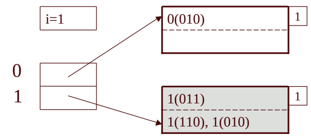
- Insert 1110



31

Insertion in Extensible Hash Table

- Now insert 1010

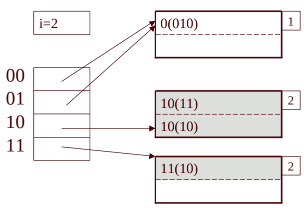


- Need to extend table, split blocks
- i becomes 2

32

Insertion in Extensible Hash Table

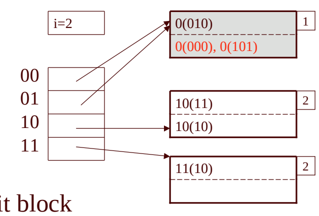
- Now insert 1110



33

Insertion in Extensible Hash Table

- Now insert 0000, then 0101

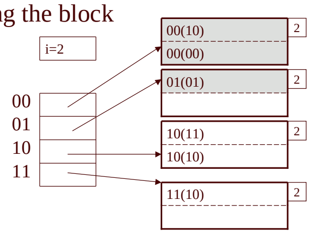


- Need to split block

34

Insertion in Extensible Hash Table

- After splitting the block



35

Performance Extensible Hash Table

- No overflow blocks: access always one read
- BUT:
  - Extensions can be costly and disruptive
  - After an extension table may no longer fit in memory

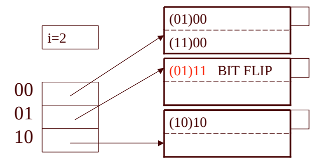
36

Linear Hash Table

- Idea: extend only one entry at a time
- Problem: n= no longer a power of 2
- Let i be such that $2^i <= n < 2^{i+1}$
- After computing h(k), use last i bits:
  - If last i bits represent a number > n, change msb from 1 to 0 (get a number <= n)

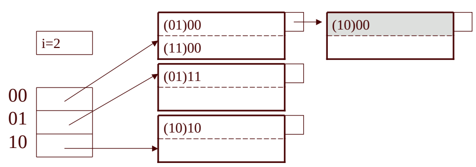
Linear Hash Table Example

- n=3



Linear Hash Table Example

- Insert 1000: overflow blocks…

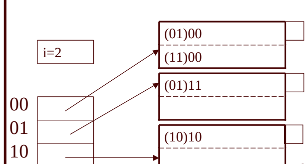


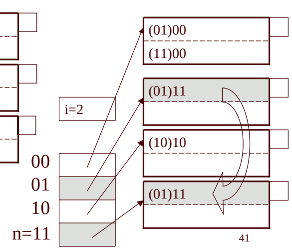
Linear Hash Tables

- Extension: independent on overflow blocks
- Extend n:=n+1 when average number of records per block exceeds (say) 80%

Linear Hash Table Extension

- From n=3 to n=4





- Only need to touch one block (which one ?)

Linear Hash Table Extension

- From n=3 to n=4 finished
- Extension from n=4 to n=5 (new bit)
- Need to touch every single block (why ?)