Introduction to Database Systems

CSE 444

Lecture #13
Feb 21 2001

Announcements

- Midterm grading completed
  - Problem 1,5: Yana
  - Problems 2,3,4 - Surajit
- HW#3 due today
- "Best 3" homeworks will be used for grades
  - It is in your benefit to turn in HW#4
- Reading list list for Last Wed 2/14 (Vol 2)
  - Section 3.1.3, 3.2, 3.3.1, 3.3.2, 3.4, 3.5, 4.1, 4.2

2

Announcements (2)

- Project Report Due next Wed
  - Project Web Page will be updated with guidelines
- Project Demo and Interview
  - March 6 (after class) and March 7
  - Yana will coordinate signing up
- Finals Overview – March 5
  - About 15 mins
  - Immediately after class
- Finals week office hours will be announced next week

3

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)

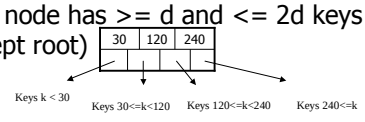
4

Indexing

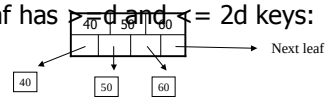
Reading: Section 4.3, 4.4, 5.4 (Vol 2)

B+ Trees Basics

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



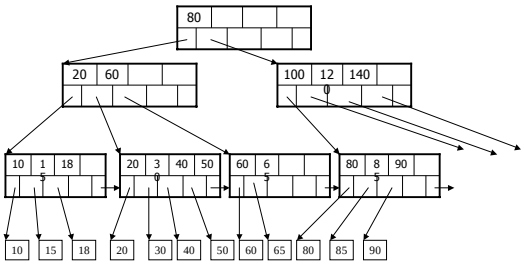
- Each leaf has >= d and <= 2d keys:



6

B+ Tree Example

d = 2



7

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

8

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
```

9

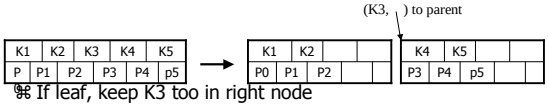
B+ Trees in Practice

- 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

10

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:

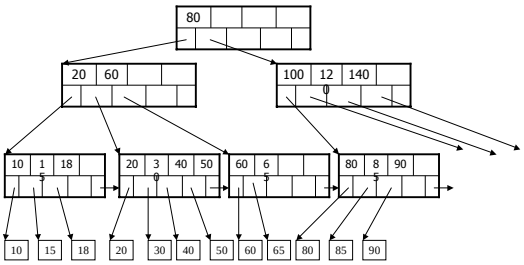


- If leaf, keep K3 too in right node
- When root splits, new root has 1 key only

11

Insertion in a B+ Tree

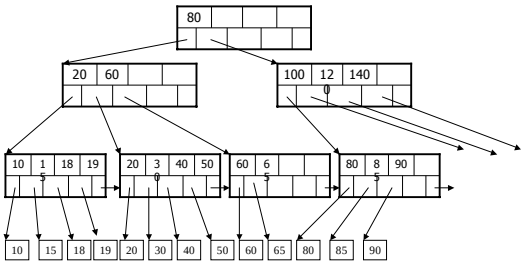
Insert K=19



12

Insertion in a B+ Tree

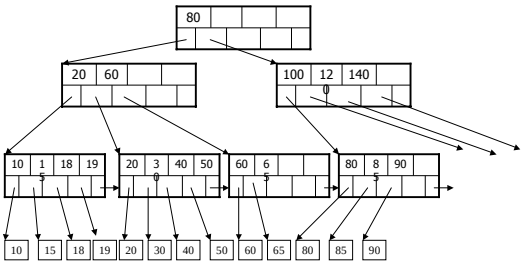
After insertion



13

Insertion in a B+ Tree

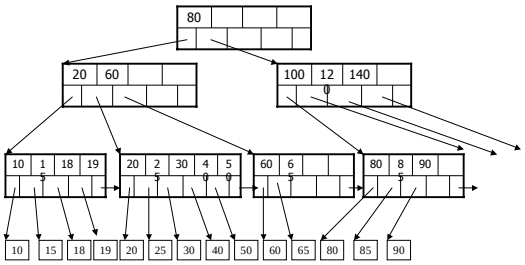
Now insert 25



14

Insertion in a B+ Tree

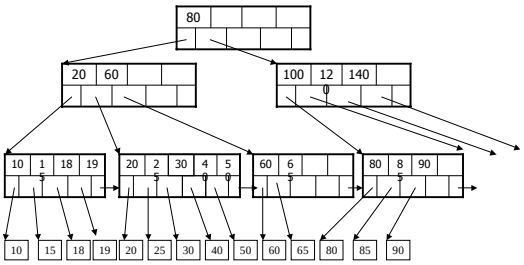
After insertion



15

Insertion in a B+ Tree

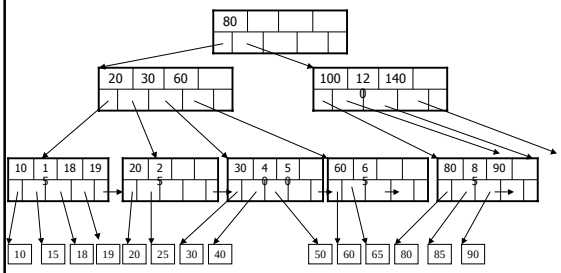
But now have to split !



16

Insertion in a B+ Tree

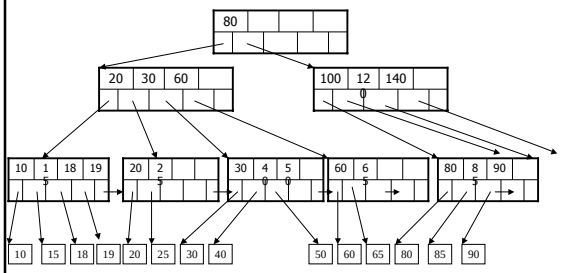
After the split



17

Deletion from a B+ Tree

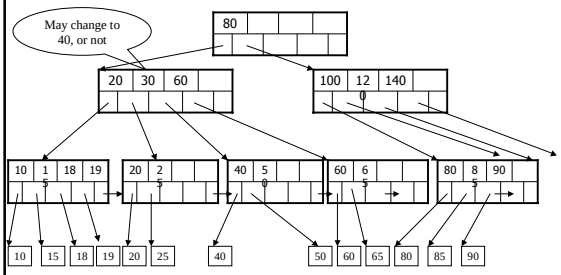
Delete 30



18

Deletion from a B+ Tree

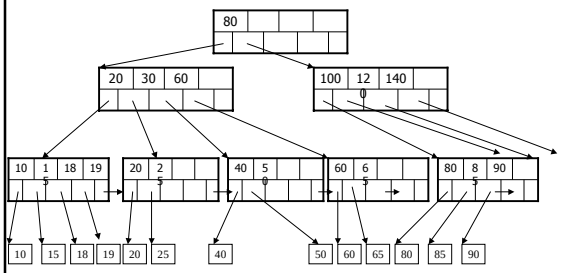
After deleting 30



19

Deletion from a B+ Tree

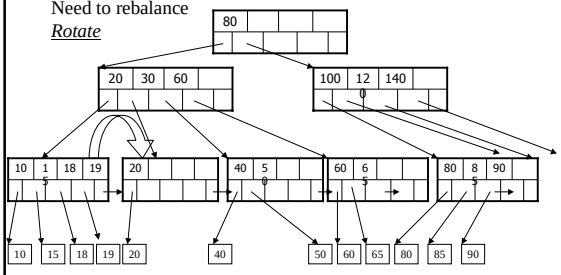
Now delete 25



20

Deletion from a B+ Tree

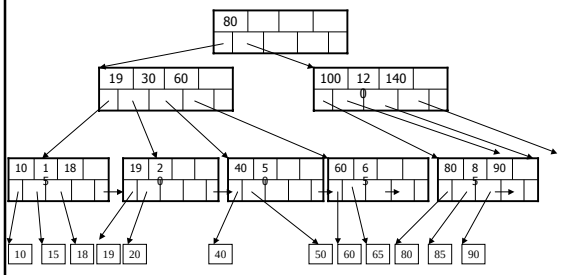
After deleting 25
Need to rebalance
Rotate



21

Deletion from a B+ Tree

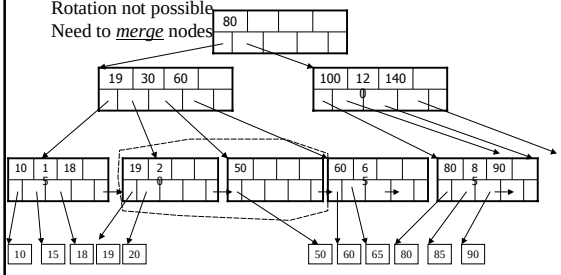
Now delete 40



22

Deletion from a B+ Tree

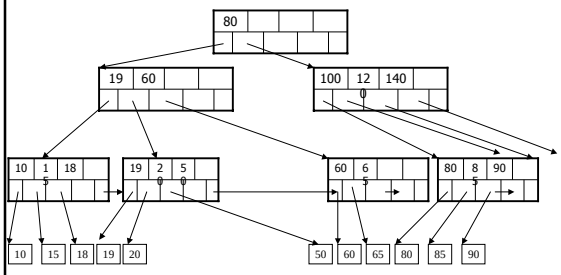
After deleting 40
Rotation not possible
Need to *merge* nodes



23

Deletion from a B+ Tree

Final tree



24

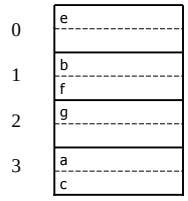
Hash Tables

- 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

25

Hash Table Example

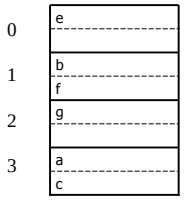
- Assume 1 bucket (block) stores 2 keys + pointers
- h(e)=0
- h(b)=h(f)=1
- h(g)=2
- h(a)=h(c)=3



26

Searching in a Hash Table

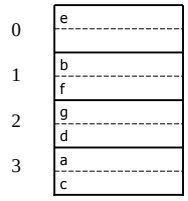
- Search for a:
- Compute h(a)=3
- Read bucket 3
- 1 disk access



27

Insertion in Hash Table

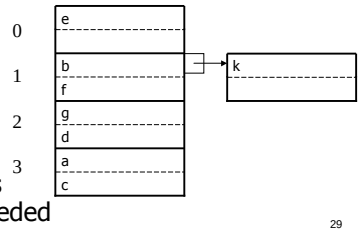
- Place in right bucket, if space
- E.g. h(d)=2



28

Insertion in Hash Table

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



- More over-flow blocks may be needed

29

Hash Table Performance

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

30

Extensible Hash Table

⌘Allows has table to grow, to avoid performance degradation

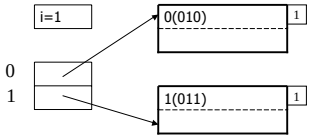
⌘Assume a hash function h that returns numbers in {0, …, $2^k - 1$}

⌘Start with n = 2^i << 2^k , only look at first I most significant bits

31

Extensible Hash Table

⌘E.g. i=1, n=2, k=4

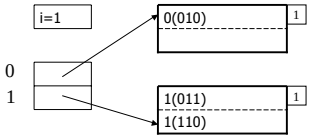


⌘Note: we only look at the first bit (0 or 1)

32

Insertion in Extensible Hash Table

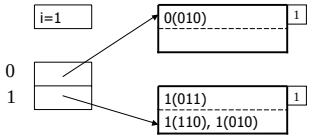
⌘Insert 1110



33

Insertion in Extensible Hash Table

⌘Now insert 1010



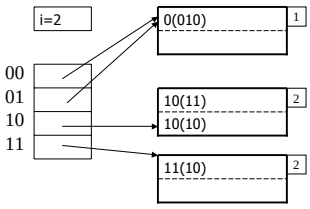
⌘Need to extend table, split blocks

⌘i becomes 2

34

Insertion in Extensible Hash Table

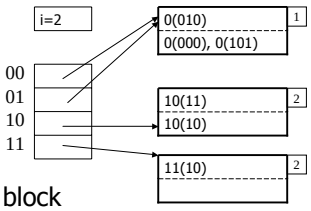
⌘Now insert 1110



35

Insertion in Extensible Hash Table

⌘Now insert 0000, then 0101

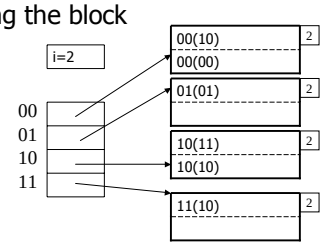


⌘Need to split block

36

Insertion in Extensible Hash Table

⌘After splitting the block



37

Performance Extensible Hash Table

⌘No overflow blocks: access always one read

⌘BUT:

- Extensions can be costly and siruptive
- After an extension table may no longer fit in memory

38

Linear Hash Table

⌘Idea: extend only one entry at a time

⌘Problem: n= no longer a power of 2

⌘Let i be such that $2^i <= n < 2^{i+1}$

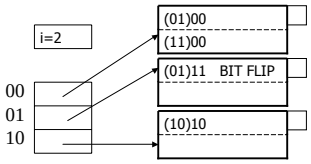
⌘After computing h(k), use last i bits:

- If last i bits represent a number > n, change msb from 1 to 0 (get a number <= n)

39

Linear Hash Table Example

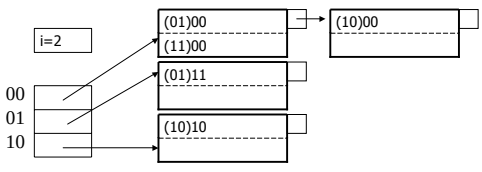
⌘N=3



40

Linear Hash Table Example

⌘Insert 1000: overflow blocks…



41

Linear Hash Tables

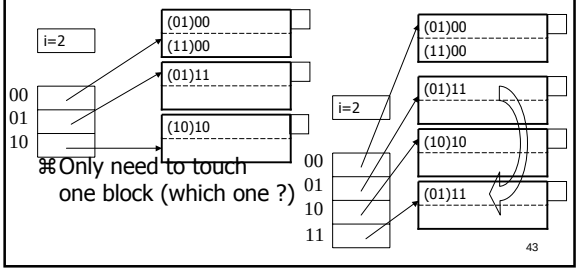
⌘Extension: independent on overflow blocks

⌘Extend n:=n+1 when average number of records per block exceeds (say) 80%

42

Linear Hash Table Extension

⌘From n=3 to n=4



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

43

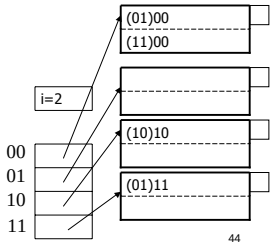
Linear Hash Table Extension

⌘From n=3 to n=4 finished

⌘Extension from n=4 to n=5 (new bit)

⌘Need to touch every single block (why ?)

⌘Book is wrong here…



44