class 9

b-trees

prof. Stratos Idreos

HTTP://DASLAB.SEAS.HARVARD.EDU/CLASSES/CS165/
1 pass to sort each page (2N pages)

1 pass to merge into 2 sorted pages (2N pages)

1 pass to merge into 4 sorted pages (2N pages)

1 pass to merge into 8 sorted pages (2N pages)

$2N(\log_2(N)+1)$
take into account the whole memory hierarchy
measure cost in bytes moved across the memory hierarchy
build model step by step
(for model) focus on the weak link
data size: N pages
memory size: M pages

how much memory M do we need to sort N data in p passes only?

or

how much data can we sort in p passes if we have M memory?

\[ \log_{M-1}(N/M) + 1 \leq p \]
I spend a lot of time debugging
am I doing something wrong?
maybe, but probably not

1. learn to use gdb & valgrind
2. after spending X time debugging ask for help
3. enjoy it :)

test extensively every few lines of code
isolate problems - one change at a time
index knows order about the data

filtering data: point/range queries
A B C

initial state
columns in
insertion order

sorted A B C

propagate
order of A

sorted A B C
select max(D), min(E) from R where (A > 10 and A < 40) and (B > 20 and B < 60)

avoid scan of A
avoid TR on B
work on a restricted area across all columns
good for memory hierarchy

binary search for 10 & 40
for all B values between pos1 & 2: if B > 20 and B < 60
mark bit vector at pos i
for each marked position
max(D)
space overhead - update overhead - which ones to build?
**today+1**: more about indexing

**tree indexes in data systems**

what is interesting about trees from a design point of view
Goetz Graefe
Microsoft, HP Fellow, now Google
ACM Software System Award

“It could be said that the world’s information is at our fingertips because of B-trees”
primary/clustered (all columns)

secondary/unclustered indexes
subset of columns

you are doing both in the project
Btree on A, A is sorted, order is propagated to the rest of the columns

every table can/should have one (be a) clustered index

clustered index on A
(no need for mappings)
Btree on C, copy of C is sorted, we keep a copy of the positions that map on the clustered index.

Secondary index on any column(s) needs positions.
select C from R \textbf{where} A<x  
select A from R \textbf{where} B<z  
select D from R \textbf{where} A<x \textbf{and} B<z

which indexes to create
declarative interface
ask what you want

indexes/views/tuning knobs

but … db cracking, adaptive* ideas

db system
how to use a b-tree index (plans)?
(same discussion as we did for sorting!)
clustered index case plan vs secondary index plan

\[
\begin{align*}
&\text{select } \max(B) \\
&\text{from } R \\
&\text{where } A < 20
\end{align*}
\]

\[
\begin{align*}
&\text{select } \max(B) \\
&\text{from } R \\
&\text{where } C < 20
\end{align*}
\]
ok and how do we built, search, update a tree index efficiently?

structure = complexity = if statements, random access, instruction misses, etc.

= no free lunch

node size, data organization fanout ...
A sorted array is shown, with elements grouped into pages:

- 1, 2, 3...
- 12, 15, 17
- 20, ...
- ...

The page size is 64K, which holds 16K 4-byte integers. The number of elements, N, is spread across P pages.
info to navigate lower level value-pointer

sorted array

page size: 64K - holds 16K 4 byte ints
N elements, P pages
page size: 64K - holds 16K 4 byte ints
N elements, P pages

4+4 bytes for each page
(value+pointer)
64K/8 = index 8K pages

info to navigate
lower level
value-pointer

>=12
<12

12,20
35,...
50,...
12,15,17
20,...
...
Page size: 64K - holds 16K 4 byte ints
N elements, P pages

info to navigate lower level value-pointer

4+4 bytes for each page (value+pointer)
64K/8 = index 8K pages

can index 8K pages of the next level

sorted array
The diagram illustrates a data structure with a root node labeled "30,50". From the root, there are branches leading to internal nodes labeled "12,20", "35,...", and "50,...". These internal nodes are connected to leaves labeled "1,2,3,...", "12,15,17", "20,...", and "...". The structure is described as having a fanout with internal nodes and leaves.
The height of the tree is \( \log_{\text{fanout}} N \).

- **Root**: Contains 30, 50.
- **Internal Nodes**: Contains 12, 20, 35, 50, ...
- **Leaves**: Contains 1, 2, 3, ..., 12, 15, 17, 20, ..., ...

The tree is a balanced binary tree, where each node has a fanout of 3.
random accesses

height $\log_{\text{fanout}} N$

leaves

1, 2, 3, ...
12, 15, 17
20, ...

internal nodes

root

30, 50

fanout

35, ...
50, ...

12, 20

random accesses

height $\log_{\text{fanout}} N$
get 15
get 15-25

how do we search the leaves?
should we store leaves as independent nodes or as a single contiguous column?
should we store leaves as independent nodes or as a single contiguous column

diff in structure of clustered vs secondary index
1) compare tree search to binary search cost for clustered index

2) design cache conscious tree

assume: input 1 Billion integers array, page size 64Kb, system 64bit
thinking process:

1B ints 8 bytes each, 64 bit system: our pointers are 8 bytes - page size 64Kb

total data size /page size: 1B*8bytes/64Kb=122K pages

**binary search costs 16 pages** (every factor of 10 we add ~3.3 and assume result fits in one page)
in general: log2(pages)

internal b-tree node holds: 64Kb/(8+8)bytes=4K (assuming each node is one page big)
so fanout is 4K

search: starting from root we can search 4K pages, at next level we search 4K*4K=16M pages >122K
we have to search at most 1 node at each internal level + one or more pages at the leaves
so 2+1= 3 pages in total (assuming we find results in one leaf)
in general: logFanout(total pages)

if 100 billion data it would be log2 12M = ~23 pages for binary search
while for tree same as before =3 pages
search for key X:
search (Node, Key)
find where Key falls in the keys of Node
binary search local keys
then follow the respective pointer and repeat
internal node

pointer - key - pointer - key - pointer - ... - pointer

child 1  child 2  child 3  child n+1

n keys
n+1 pointers

internal node

key - key - key - key - key - key - key - ... - pointer

child 1  child 2  child 3  ...  child 2n+1

2n keys
1 pointer

single contiguous memory area
Modern B-Tree Techniques
by Goetz Graefe
Foundations and Trends in Databases, 2011
Sections: 1,2,3,5

textbook: Chapter 10 (b-trees)
textbook: Chapter 13 (qe)

Making B+trees Cache Conscious in Main Memory
Jun Rao and Ken Ross
ACM SIGMOD International Conference on Management of Data, 2000
b-trees

DATA SYSTEMS

prof. Stratos Idreos