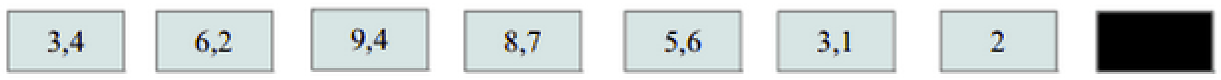
CS186 Discussion Section 6

External Merge Sort

Exercise 1: Complete the tree below for a general merge sort assuming B=4



Consider the following 3 scenarios, using the generalized merge sort algorithm:

1. A file with 10k pages and 3 buffer pages in memory
2. A file with 20k pages and 5 buffer pages in memory
3. A file with 2 million pages and 17 buffer pages in memory

For each scenario, answer the following exercises. Don't worry if you don't have a calculator, just write out as much as you can compute by hand.

Exercise 2: How many runs will the algorithm produce in the first pass (pass 0)?

(1)

(2)

(3)

Exercise 3: How many passes will it take to sort the whole file completely?

(1)

(2)

(3)

Exercise 4: What is the total I/O cost of sorting the file (in number of I/Os)?
(1)

(2)

(3)

Exercise 5: How many buffer pages (i.e., B = ?) do you need to sort the file completely in two passes?
(1)

(2)

(3)

External Hashing

a) Consider a simple, 2-pass disk-based hashing strategy as described in lecture, where in the first pass the file is partitioned and in the second pass each of the partitions is hashed (i.e., not hybrid hashing!).

Ignoring any space overhead for building hash tables and assuming a perfect hash function, what is the minimum number of memory pages (same size as file blocks) required to hash the Students relation in 2 passes? (The Students relation has 5000 pages.)

b) In part (a) above, how many I/Os will be required? Assume that the relation is originally on disk that the result of the hashing operation must also be written to disk.

c) Now, suppose that the hash function used for part (a) does not work very well. Describe briefly why 2 passes may not be sufficient and how a complete algorithm would solve this problem.

Loop Joins

- Nested Loop Join (NLJ) (A ⋈ B) (TA - # tuples in A, PA - # pages of A, TB - # tuples in B, PB - # pages of B)
    - IOs: TA * PB + PA (scan B for every TUPLE of A)
- Page/Block NLJ (blocks are usually multiple pages, BA - # blocks/pages of A)
    - IOs: BA * PB + PA (scan B for every BLOCK/PAGE of A)

- Index NLJ (CB - cost of using index for lookup, usually on the order of 5 or fewer)
    - IOs: TA * CB + PA (lookup index on B for every TUPLE of A)
- Sort-Merge Join
    - IOs: 3*(PA + PB) (sort A and sort B, then merge. refinements for fewer passes possible)
- Hash Join
    - IOs: 3*(PA + PB) (hash A and has B, then match corresponding partitions together)
- Index Only Scan
    - only applicable if all relevant attributes are in the index key. just read the index entries, not the data.
    - index entries probably smaller than actual records, so more fit on a page, so fewer IOs!

Questions about Joins

Car(license, owner_ssn, year, company, model)
Accident(license, accident_date, damage_amount, zipcode)
Owner(ssn, license, name, gender, street, city, zipcode)

NTuples(Car) = 1000 ; NPages(Car) = 100
NTuples(Accident) = 500 ; NPages(Accident) = 20
NTuples(Owner) = 800 ; NPages(Owner) = 50

B+Tree index on Car(owner_ssn, license, year):

- 2 levels (including the leaf level)
- 20 leaf pages
- alternative 2 (rid to actual data records)

1) What is the IO cost of "Owner ⋈ Car" (Car is inner table) using NLJ?

2) What is the IO cost of "Owner ⋈ Car" (Car is inner table) using BNLJ (block size is 5 pages)?

3) Use the index on Car, what is the IO cost of "Owner ⋈ Car" (Car is inner table) using INLJ?

4) For the query: "SELECT AVG(year) FROM Car;" will an index only scan work? If so, what would be the IO cost