CSC3170 Assignment3

Correspondence: Yuyang Liang

November 2024

1 Introduction

- This is the third homework for CSC3170. It weighs 10% of your final grade.
- The deadline for this homework is December 7th 11:59 pm.
- This assignment includes two parts, a writing part, and a coding part, each part weighs 50%.
- Directly using AI-generated answers or other students answers will be considered as cheating.

2 Writing Part (50%)

2.1 Merge Algorithm (24%)

We have a database file with twelve million pages (N = 12,000,000 pages), and we want to sort it using external merge sort. Assume that the DBMS is not using double buffering or blocked I/O, and that it uses quicksort for in-memory sorting. Let B denote the number of buffers pages.

(a) Assume that the DBMS has 60 buffers. How many passes does the DBMS need to perform in order to sort the file? (4%)

$$1 + \lceil \log_{B-1} \left(\lceil \frac{N}{B} \rceil \right) \rceil$$
$$= 1 + 3 = 4$$

(b) Assume that the DBMS has 60 buffers. How many sorted runs are generated during the whole process? Note that the final sorted file does not count towards the sorted run count. (4%)

$$\lceil \frac{12000000}{60} \rceil + \lceil \frac{200000}{59} \rceil + \lceil \frac{3390}{59} \rceil = 203448$$

(c) Again, assuming that the DBMS has 60 buffers. How many pages (in general) does each sorted run have after the third pass (i.e., Pass #2 if you start counting from Pass #0)? (4%)

$$60 \times 59 \times 59 = 208860$$

(d) Suppose the DBMS has 120 buffers. What is the total I/O cost to sort the file? (4%)

$$2 \times N \times \#\text{Pass} = 96000000$$

(e) What is the smallest number of buffers B such that the DBMS can sort the target file using only three passes? (4%)

$$N \leq B(B-1)^2 \Longrightarrow B \geq 230$$

(f) Suppose the DBMS has 25 buffers. What is the largest database file (expressed in terms of the number of pages) that can be sorted with external merge sort using four passes? (4%)

$$N \leq B(B-1)^3 \Longrightarrow N \leq 345600$$

2.2 Join Algorithm (26%)

Consider relations $A(x, z)$, $B(x, y, w)$, and $C(y, v, u)$ to be joined on the common attribute x. Assume that there are no indexes available on the tables to speed up the join algorithms.

- The buffer has $F = 500$ pages.
- Table A spans $M = 3{,}000$ pages with 50 tuples per page.
- Table B spans $N = 800$ pages with 200 tuples per page.
- Table C spans $O = 2{,}000$ pages with 100 tuples per page.
- The join result of A and B spans $P = 600$ pages.

For the following questions, assume a simple cost model where pages are read and written one at a time. Also assume that one buffer block is needed for the evolving output block and one input block is needed for the current input block of the inner relation. You may ignore the cost of writing the final results.

(a) What is the I/O cost of a simple nested loop join with A as the outer relation and B as the inner relation? (4%)

$$3000 + 50 \times 800 \times 3000 = 120003000$$

(b) What is the I/O cost of a block nested loop join with C as the outer relation and B as the inner relation? (4%)

$$2000 + \lceil \frac{2000}{500 - 2} \rceil \times 800 = 6000$$

(c) For a sort-merge join with B as the outer relation and C as the inner relation: What is the cost of sorting the tuples in B on attribute y? And what is the cost of the merge phase in the worst-case scenario? (Assume external merge sort is applied and no clustered index is used.)(6%)

Sorting : 3200

Merging :

$N + \lceil \frac{N}{F - 2} \rceil \times O = 4800$ (buffer acceleration)

or $2000 * 800 = 1600000$,

or $800 + 2000 \times 800 = 1600800$ (considering the cost of reading outer table)

(d) Consider joining A,B and then joining the result with C using the sort-merge join algorithm. What is the cost of the final merge phase assuming there are no duplicates in the join attribute? (4%)

$$600 + 2000 = 2600$$

(e) Consider a hash join with B as the outer relation and C as the inner relation. Assume no recursive partitioning is needed, and the pages in each partition are exactly filled with tuples (i.e., no partially filled pages.) What are the costs of the partition phase and the probe phase? (8%)

Partition : $2 \times (800 + 2000) = 5600$

Probe : $800 + 2000 = 2800$

3 Coding Part (50%)

3.1 Task Description

Your task is to implement a B+ Tree data structure in your preferred programming language. The B+ Tree should support the following core functionalities:

(a) Insertion: Adds a new element into the B+ tree and maintains the tree property.

(b) Search: Searches for an element in the B+ tree.

(c) Deletion: Removes an element from the B+ tree if it exists.

(d) Display: display all elements stored in the B+ tree in ascending order.

We recommend using an object-oriented programming (OOP) approach. You may design and implement a dedicated class for tree nodes and use appropriate data structures to preserve the properties of a B+ Tree. Ensure that the insertion, deletion, and search operations have a time complexity of $O(\log n)$.

Here is an example of expected code usage in `Python` (you are allowed to use other programming languages like `C/C++/Java`). You are free to have your own design as long as it correctly realizes the B+ tree structure:

```python
# Create a B+ Tree with a specified degree (e.g., degree = 4)
tree = BPlusTree(degree=4)

# Insertion
# Input: a value to insert
# Output: A confirmation (e.g., True or a status message)
# indicating that the value has been successfully inserted
tree.insert(10)
tree.insert(20)
tree.insert(5)
tree.insert(15)

# Search
# Input: a value to search
# Output: True if the value exists, otherwise False.
result = tree.search(15) # True

# Deletion
# Input: a value to delete
# Output: A confirmation (e.g., True or a status message)
# indicating that the value has been successfully deleted,
# or an error message if the value was not found.
tree.delete(10)

# Display the values in the tree
# Input: None
# Output: all values in ascending order
tree.display() #Output: 5,15,20
```

Assume that all data is of type `int` by default. You only need to implement the basic functions of the B+ tree, no optimization techniques are required.

Hint: For this task, since we do not have the real data, the value stored in B+ tree is treated as the key itself.

3.2 Grading Criteria

- **Code Correctness (25%)**:
  - The implementation should correctly perform the required operations: insertion, search, deletion, and display.
  - The B+ Tree should maintain its structural properties after each operation.
- **Code Readability and Comments (5%)**:
  - Code should be well-structured and follow clean coding practices.
  - Proper and meaningful comments should be included to explain the logic and functionality of the code.
- **Report(20%)**: You should also submit a brief report (recommended within 3 pages, up to 6 pages) that includes, at a minimum, the following sections:
  - Introduction: summarize what you have done for this coding assignment.
  - Design & Implementation: detail the design choices, challenges faced, and solutions implemented.
  - Code Usage: explain how to run your code, and include examples of the output of functions.

  You will lose all points if you only submit the report without your source code. Please be concise and say something that matters.

4 Submission Requirements

You should submit three files via the Blackboard system:

- An answer sheet for the writing part.
- The source code of the coding part (as a single `zip` file if it consists of multiple files).
- A brief report of the coding part.