FACULTY OF ENGINEERING

CME 2202 DATA ORGANIZATION AND MANAGEMENT

ASSIGNMENT-2

Assignment: Implementing a B+ Tree with External Merge Sort

Student Names:

Saim Melih ÖZCAN

Bahadır Yavuz Yapar

Student IDs:

2021510093

2021510109

Submission Date : 15.06.2025

Instructor Name: D. Göksu Tüysüzoğlu

Table of Contents

1. Introduction

In this homework, we implement a B+ Tree structure to manage university data provided in CSV format. Each record consists of a department name, university name, and a score. Two approaches are used for loading the data:

- **Sequential Insertion:** Records are inserted into the B+ Tree one by one as they are read.
- **Bulk Loading:** Data is sorted externally using replacement selection and merged into a final sorted stream before being inserted efficiently into the tree.

We compare both methods in terms of time, node splits, memory usage, and tree height.

2. Conceptual Overview

2.1 B+ Tree

A B+ Tree is used as the indexing structure, with a maximum degree of 4. Each leaf node maintains a linked list of universities for a given department, sorted by score in descending order. Leaf nodes are linked to each other using a next pointer for efficient traversal.

2.2 External Sorting with Replacement Selection

To enable efficient bulk loading, we apply **replacement selection**, which creates longer sorted runs from the input data than a fixed-size buffer allows. This reduces the number of temporary files generated before the final k-way merge.

Replacement Selection Steps:

1. Load a buffer of records into a min-heap.
2. Extract the minimum (smallest department, highest score).
3. Replace with the next CSV record. If it violates heap order, freeze it for the next run.
4. Continue until all heap elements are frozen, then start a new run.
5. Perform a k-way merge of all runs to obtain a fully sorted list.

3. Implementation Architecture

3.1 Modules

- **ReplacementSelectionSortAndWriteRun:** Implements the replacement selection algorithm.
- **CreateSortedRuns:** Reads CSV and creates sorted temporary files.
- **MergeRuns:** Combines sorted runs into a unified stream.
- **BuildLeafLevel / BuildInternalLevels:** Constructs the B+ Tree from sorted records.
- **LoadCSVSequential:** Handles insertion of unsorted records one by one.
- **WriteSortedOutput:** Outputs final sorted structure to output.txt.

3.2 Program Flow

1. Prompt user to choose loading mode.
2. Start timer.
3. Perform loading (sequential or bulk).
4. Generate and print statistics: execution time, split count, memory use.
5. Ask user to search a department by rank if desired.

4. Key Code Sections

4.1 Replacement Selection

Records are stored in a heap structure. When outputting sorted runs, records that would violate ordering are deferred to the next run. This dynamic reorganization improves run length and reduces the number of merge phases.

```c
void replacementSelectionSortAndWriteRun(Record* buffer, int count, const c
    typedef struct HeapNode {
        Record rec;
        int active;  // 1: active in current run, 0: frozen
    } HeapNode;

    HeapNode* heap = malloc(sizeof(HeapNode) * count);
    for (int i = 0; i < count; i++) {
        heap[i].rec = buffer[i];
        heap[i].active = 1;
    }

    FILE* out = fopen(filename, "w");
    if (!out) { perror("run file write"); exit(1); }

    int heapSize = count;
    int runCompleted = 0;
    Record lastWritten = {"", "", -1};

    while (!runCompleted) {
        // Min-heapify
        for (int i = heapSize / 2 - 1; i >= 0; i--) {
            int root = i;
            while (2 * root + 1 < heapSize) {
                int child = 2 * root + 1;
                if (child + 1 < heapSize &&
                    (compareRecords(&heap[child + 1].rec, &heap[child].rec)
                    child++;

                if (compareRecords(&heap[root].rec, &heap[child].rec) <= 0)

                HeapNode tmp = heap[root];
```

4.2 Bulk Tree Construction

Leaves are created by grouping consecutive records. Internal nodes are built iteratively, grouping up to DEGREE children and promoting the first key of each child.

4.3 Leaf Split (Sequential Mode)

If a leaf becomes full during sequential insertion, it is split and a new node is created. A new root is formed if the current root is split.

```
// splitting a leaf for B+ tree
void splitLeaf(BPTreeNode** rootRef, BPTreeNode* leaf, char* dept, char* un
    leafSplitCount++;
    BPTreeNode* newLeaf = createNode(1);
    char* tempKeys[DEGREE + 1];
    University* tempLists[DEGREE + 1];

    for (int i = 0; i < DEGREE; i++) {
        tempKeys[i] = leaf->keys[i];
        tempLists[i] = leaf->uniLists[i];
    }

    int i = DEGREE - 1;
    while (i >= 0 && strcmp(dept, tempKeys[i]) < 0) {
        tempKeys[i + 1] = tempKeys[i];
        tempLists[i + 1] = tempLists[i];
        i--;
    }
    tempKeys[i + 1] = strdup(dept);
    tempLists[i + 1] = insertUniversitySorted(NULL, uni, score);

    leaf->numKeys = 0;
    for (int j = 0; j < (DEGREE + 1) / 2; j++) {
        leaf->keys[j] = tempKeys[j];
        leaf->uniLists[j] = tempLists[j];
        leaf->numKeys++;
    }
    for (int j = (DEGREE + 1) / 2, k = 0; j < DEGREE + 1; j++, k++) {
        newLeaf->keys[k] = tempKeys[j];
        newLeaf->uniLists[k] = tempLists[j];
        newLeaf->numKeys++;
    }

    newLeaf->next = leaf->next;
    leaf->next = newLeaf;

    if (*rootRef == leaf) {
        BPTreeNode* newRoot = createNode(0);
        newRoot->keys[0] = strdup(newLeaf->keys[0]);
        newRoot->children[0] = leaf;
```

5. Experimental Results

5.1 Environment

- **CPU:** Intel Core i5 11th Gen
- **RAM:** 16 GB
- **Device:** MSI Katana GF66
- **Dataset:** ~7,000 records from yok_atlas.csv

5.2 Performance Comparison

```
root@LeDebianG:/home/bahog/Masaüstü/DOM_2# gcc -o bptree deneme.c
root@LeDebianG:/home/bahog/Masaüstü/DOM_2# ./bptree
Please choose a loading option:
1 - Sequential Insertion
2 - Bulk Loading (with external merge sort)
>> 1

Completed in 0.017 seconds.
Total splits: 118
Memory used: ~14400 bytes (14.06 KB)
Tree Height: 2

Would you like to search for a university? (y/n): y
Enter department name: Psikoloji
Enter university rank: 2
University at rank 2 in Psikoloji: BOGAZICI UNIVERSITESI (494.29)
```

```
root@LeDebianG:/home/bahog/Masaüstü/DOM_2# ./bptree
Please choose a loading option:
1 - Sequential Insertion
2 - Bulk Loading (with external merge sort)
>> 2

Completed in 0.050 seconds.
Total splits: 4
Memory used: ~17880 bytes (17.46 KB)
Tree Height: 5

Would you like to search for a university? (y/n): y
Enter department name: Tip
Enter university rank: 1
University at rank 1 in Tip: ISTANBUL MEDIPOL UNIVERSITESI (555.36)
```

5.3 Analysis

The bulk loading approach significantly reduces the number of splits due to pre-sorted input, leading to a flatter, more balanced tree. Replacement selection allows for longer runs than fixed-buffer sorting, improving I/O efficiency.

6. Discussion

- Replacement selection is effective for large datasets where memory is limited.
- Bulk load outperforms sequential insertion in execution time and structural stability.
- Node splits impact search performance and should be minimized.
- Further internal split logic can optimize tree balance even more.

7. Conclusion and Future Work

Conclusion:
Bulk loading with external sorting and replacement selection is superior for large-scale tree construction. It minimizes splits and memory overhead while maintaining query efficiency.