Op&misingExpense Management: Leveraging Binary Search Trees for Efficient Categorisa&on

SESSION 2023-2024

BY: SAROJ KUMAR

INDEX

S. No.	Descrip&on	Page No.
1.	Acknowledgement	3
2.	Project Overview	4
4.	Source Code	5
5.	Output Screen	13
6.	TesBng	17
7.	References	20

ACKNOWLEDGEMENT

I would like to express my special thanks of graBtude to my Professor

“Dr. Suchetana Chakraborty”

who gave me this golden opportunity to work on this wonderful project. I would also like to thanks my group mates who also helped me in compleBng the project. I came to know about many things. I am thankful to them.

Also I would like to thank my Mentor TA

Ms. Akanksha Dwivedi

who helped me throughout the project and achieving this milestone.

PROJECT OVERVIEW

We usually face a problem while going with friends outside or on a trip, we didn’t have track of our expenses. There are earlier algorithms/ apps made for tracking expenses like PhonePe does offer spliRng bill opBon aSer payment but that is very complex to work with. We have made a beTer algorithm for storing our daily transacBons. We have implemented our code using BST in which all nodes will be having a category and a queue structure to store Date, Payee and Amount for a parBcular transacBon. Earlier use for storing transacBons was done using array based implementaBon which always give a Bme complexity. We have done categorisaBon of expenses whether it is their Bills payment or any other expense like Food or TransportaBon.

SOURCE CODE

➢ Declara&on of structures

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <4me.h>

// Structure to represent each expense
typedef struct ExpenseNode {     char
date[20];     char payee[50];     float
amount;
   struct ExpenseNode *next;
} ExpenseNode;

// Structure to represent a queue of expenses typedef
struct {
   ExpenseNode *front;
   ExpenseNode *rear;
} ExpenseQueue;

// Structure to represent each category node in the tree
typedef struct TreeNode {     char category[50];
ExpenseQueue queue;     struct TreeNode *leL;     struct
TreeNode *right; } TreeNode;
```

➢ Func&ons used in the code

- FuncBon to create a new node in a tree

```
struct TreeNode* createNode(char category[]) {
   TreeNode* newNode = (TreeNode*)malloc(sizeof(TreeNode));
strcpy(newNode->category, category);     newNode->leL =
NULL;     newNode->right = NULL;     newNode->queue.front =
NULL;     newNode->queue.rear = NULL;     return newNode;
}
```

- FuncBon to check if the queue is empty

```
int isQueueEmpty(ExpenseQueue *queue) {
return queue->front == NULL;
}
```

- FuncBon to add a new node in the queue

```
void enqueue(ExpenseQueue *queue, char date[], char payee[], float amount) {
ExpenseNode newNode = (ExpenseNode)malloc(sizeof(ExpenseNode));     if
(newNode == NULL) {
      return;
   }
   strcpy(newNode->date, date);
strcpy(newNode->payee, payee);
newNode->amount = amount;
newNode->next = NULL;     if
(isQueueEmpty(queue)) {
queue->front = newNode;
   } else {
      queue->rear->next = newNode;
   }
   queue->rear = newNode;
}
```

- FuncBon to display expenses based on the each category

```
void displayExpenses(TreeNode* root) {    if
(root != NULL) {       prinX("Category: %s\n",
root->category);       if
(!isQueueEmpty(&(root->queue))) {
        ExpenseNode *firstTransac4on = root->queue.front;          struct ExpenseNode * current =
firstTransac4on;          while(current != NULL){             prinX("Date: %s | Payee: %s | Amount: %.2f\n",
firstTransac4on->date, firstTransac4on->payee, firstTransac4on->amount);
            current = current->next;
        }
        prinX("\nPrin4ng Expenses Successful!\n");
    }

else {          prinX("No transac4ons
found.\n");
    }
    displayExpenses(root->leL);       displayExpenses(root-
>right);
  }
}
```

- FuncBon to display expenses of a single category based on user input

```
void printCategoryExpenses(TreeNode* root, char category[]) {
if (root != NULL) {       if (strcmp(category, root->category) ==
0) {          prinX("Category: %s\n", root->category);
prinX("All Expenses:\n");
        ExpenseNode *current = root->queue.front;          while (current != NULL) {
prinX("Date: %s | Payee: %s | Amount: %.2f\n", current->date, current->payee, current-
>amount);
            current = current->next;
        }
        prinX("\nPrin4ng Category Successful!\n");
return;
    }
    printCategoryExpenses(root->leL, category);       printCategoryExpenses(root-
>right, category);
  }
}
```

- FuncBon to add expense based on the '*filename.txt'* file

```
void addExpense(TreeNode * root, char date[20], char payee[50], float amount, char category[50]){
if (root == NULL) {       root = createNode(category);
   } else {       if (strcmp(category, root->category) == 0) {
enqueue(&(root->queue), date, payee, amount);       } else if
(strcmp(category, root->category) < 0) {
addExpense(root->leL, date, payee, amount, category);
      } else {
         addExpense(root->right, date, payee, amount, category);
      }
   }
}
```

- FuncBon to break input file into lines and then into tokens

```
void expense_filename(TreeNode* root, char filename[], float *current_expenses) {

   FILE* file = fopen(filename, "r");

   char line[100]; // Assuming each line won't exceed 100 characters     char *token; //
Used to break the strings into single words separated by commas, spaces

   if (file == NULL) {
prinX("File doesn't exist.\n");
return;
   }

 while (fgets(line, sizeof(line), file) != NULL) {
char date[20], payee[50], category[20];
float amount;

      token = strtok(line, " ");       if (token != NULL) {          strcpy(date, token);          token =
strtok(NULL, " "); // Here we take NULL to con4nue in the same line, next line will be ini4ates by
fgets          if (token != NULL) {              strcpy(payee, token);              token = strtok(NULL, " ");
if (token != NULL) {                amount = atof(token);                token = strtok(NULL, " “);
```

```
            *current_expenses += amount;                if (token != NULL) {
strcpy(category, token);                prinX("Date: %s | Payee: %s | Amount: %.2f | Category:
%s\n", date, payee, amount, category);                addExpense(root, date, payee, amount,
category);
            }
          }
        }
      }
    }
    fclose(file);
}
```

- FuncBon to search transacBons based on opBon (payee, date, amount) and key

```
void searchTransac4ons(TreeNode* root, char key[], char op4on[]) {
if (root != NULL) {      searchTransac4ons(root->leL, key, op4on);
if (!isQueueEmpty(&(root->queue))) {
        ExpenseNode *current = root->queue.front;          while (current != NULL) {
if ((strcmp(op4on, "payee") == 0 && strcmp(current->payee, key) == 0) ||
            (strcmp(op4on, "date") == 0 && strcmp(current->date, key) == 0) ||
            (strcmp(op4on, "amount") == 0 && current->amount == atof(key))) { // atof converts strings to
float              prinX("Category: %s\n", root->category);              prinX("Date: %s | Payee: %s |
Amount: %.2f\n", current->date, current->payee, current>amount);
          }
          current = current->next;
        }
      }
      searchTransac4ons(root->right, key, op4on);
    }
    prinX("\nSearch Successful!");
}
```

➢ Main Func&on and Basic UI

```
int main() {
   TreeNode* root = NULL;
clock_t start, end;    double
4me_taken = 0.0f;

   // Defining some default categories to the tree
   // New categories id any given by user will be added based upon its lexicographic order
if (root == NULL) {       root = createNode("ALL");
      (root)->leL = createNode("Bills");
      (root)->right = createNode("Others");
      (root)->leL->leL = createNode("Housing");
      (root)->leL->right = createNode("Maintenance");
   }

   float current_income = 0;
float current_expenses = 0;


   // Basic UI and choosing of which func4on to perform      prinX("
What do you want to do? Choose from the following:\n");    prinX("
1. Add Income\n");   prinX("   2. Add Expense\n");   prinX("   3. View
all category's transac4ons\n");      prinX("    4. View Category's all
transac4ons\n");    prinX("   5. Search all transac4ons\n");    prinX("
6. Money leL\n");    prinX("   7. Exit\n");

   char       choosen_op4on[50];
scanf("%s",     choosen_op4on);
start = clock();
```

- Calling of different funcBon based on user input (1 to 7)

```
while(strcmp(choosen_op4on, "7") != 0){

    if(strcmp(choosen_op4on,"1") == 0){
float to_add_income = 0;        prinX("Add the
amount you want to add:\n");        scanf("%f",
&to_add_income);        current_income +=
to_add_income;
    }
    else if(strcmp(choosen_op4on,"2") == 0){        char
filename[50];        prinX("Enter the expense filename:\n");
scanf("%s", filename);        expense_filename(root,
filename, &current_expenses);
    }
    else if(strcmp(choosen_op4on,"3") == 0){
displayExpenses(root);
    }
    else if(strcmp(choosen_op4on,"4") == 0){
char categoryToPrint[50];        prinX("\nEnter
category to print all expenses: ");        scanf("%s",
categoryToPrint);

      printCategoryExpenses(root, categoryToPrint);
    }
    else if(strcmp(choosen_op4on, "5") == 0){        char
key[50], op4on[10];        prinX("Enter search op4on
(payee/date/amount): \n");        scanf("%s", op4on);
prinX("Enter search key: \n");        scanf("%s", key);
searchTransac4ons(root, key, op4on);
    }
    else if(strcmp(choosen_op4on, "6") == 0){
prinX("%.2f", current_income - current_expenses);
    }
```

- Looping of the input to ask user what he wants to do next

```
      prinX("\n    What do you want to do? Choose from the following:\n”);
prinX("    1. Add Income\n");       prinX("    2. Add Expense\n");       prinX("
3. View all category's transac4ons\n");       prinX("    4. View Category's all
transac4on\n");       prinX("    5. Search all transac4ons\n");       prinX("
6. Money leL\n");      prinX("    7. Exit\n");      scanf("%s", choosen_op4on);


   }
   end = clock();
   4me_taken += ((double) (end-start)) / CLOCKS_PER_SEC;
prinX("\nTime Taken: %f", 4me_taken);    return 0; }
```

OUTPUT SCREEN

#Basic UI

```
(base) porassingh@Porass-MacBook-Air DSA_Project_Ideathon % gcc expenseManager.c
(base) porassingh@Porass-MacBook-Air DSA_Project_Ideathon % ./a.out
    What do you want to do? Choose from the following:
    1. Add Income
    2. Add Expense
    3. View all category's transactions
    4. View Category's all transactions
    5. Search all transactions
    6. Money left
    7. Exit
```

#1 Adding Income

```
1
Add the amount you want to add:
1000000

    What do you want to do? Choose from the following:
    1. Add Income
    2. Add Expense
    3. View all category's transactions
    4. View Category's all transaction
    5. Search all transactions
    6. Money left
    7. Exit
```

#2 Adding Expense

```
Time Taken: 0.000696%
(base) porassingh@Porass-MacBook-Air DSA_Project_Ideathon % ./a.out
    What do you want to do? Choose from the following:
    1. Add Income
    2. Add Expense
    3. View all category's transactions
    4. View Category's all transactions
    5. Search all transactions
    6. Money left
    7. Exit
2
Enter the expense filename:
expense_350.txt
```

#3 Prin&ng all transac&ons

```
    What do you want to do? Choose from the following:
    1. Add Income
    2. Add Expense
    3. View all category's transactions
    4. View Category's all transaction
    5. Search all transactions
    6. Money left
    7. Exit
3
Category: Bills

Date: 5/9/2018 | Payee: Zhong | Amount: 30.00
Date: 5/9/2018 | Payee: Zhong | Amount: 30.00
Date: 5/9/2018 | Payee: Zhong | Amount: 30.00
Date: 5/9/2018 | Payee: Zhong | Amount: 30.00
Date: 5/9/2018 | Payee: Zhong | Amount: 30.00
Date: 5/9/2018 | Payee: Zhong | Amount: 30.00
Date: 5/9/2018 | Payee: Zhong | Amount: 30.00
Date: 5/9/2018 | Payee: Zhong | Amount: 30.00

Printng Expenses Successful!
Category: Family

Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00

Printng Expenses Successful!
Category: Food

Date: 11/9/2018 | Payee: Bill | Amount: 650.00
Date: 11/9/2018 | Payee: Bill | Amount: 650.00
Date: 11/9/2018 | Payee: Bill | Amount: 650.00
Date: 11/9/2018 | Payee: Bill | Amount: 650.00
Date: 11/9/2018 | Payee: Bill | Amount: 650.00
Date: 11/9/2018 | Payee: Bill | Amount: 650.00
Date: 11/9/2018 | Payee: Bill | Amount: 650.00
Date: 11/9/2018 | Payee: Bill | Amount: 650.00
Date: 11/9/2018 | Payee: Bill | Amount: 650.00
Date: 11/9/2018 | Payee: Bill | Amount: 650.00
```

#4 Prin&ng a par&cular Category

```
Category: Family

Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
Date: 8/9/2018 | Payee: Steve | Amount: 40.00
```

#5 Searching a transac&on

```
    What do you want to do? Choose from the following:
    1. Add Income
    2. Add Expense
    3. View all category's transactions
    4. View Category's all transaction
    5. Search all transactions
    6. Money left
    7. Exit
5
Enter search option (payee/date/amount):
payee
Enter search key:
Larry

Date: 7/9/2018 | Payee: Larry | Amount: 1000.00

Search Successful!
```

#6 Check of the balance aher all expenses

```
      What do you want to do? Choose from the following:
      1. Add Income
      2. Add Expense
      3. View all category's transactions
      4. View Category's all transaction
      5. Search all transactions
      6. Money left
      7. Exit
 6
 853775.50
```

#7 Exi&ng the code

Displays 4me taken at the end

```
 7

 Time Taken: 0.001051%
(base) porassingh@Porass-MacBook-Air DSA_Project_Ideathon % ./a.out
    What do you want to do? Choose from the following:
    1. Add Income
    2. Add Expense
    3. View all category's transactions
    4. View Category's all transactions
    5. Search all transactions
    6. Money left
    7. Exit
 2
 Enter the expense filename:
 expense_728.txt

    What do you want to do? Choose from the following:
    1. Add Income
    2. Add Expense
    3. View all category's transactions
    4. View Category's last transaction
    5. Search all transactions
    6. Money left
    7. Exit
 7

 Time Taken: 0.001049%
```

Test Phase

Time Complexity (theoreBcal)

- For the '***addExpense***' funcBon : In the worst-case scenario, if the tree is unbalanced and resembles a linked list, the Bme complexity for inserBon would be ***O(n)***, where ***'n'*** is the number of nodes in the tree. However, in the average case, assuming the tree is balanced, the Bme complexity for inserBon is ***O(log n)***.

- For '***displayExpenses***' and '***printCategoryExpenses***' funcBon : As each node is visited only once, the Bme complexity of these operaBons is ***O(n)***, where ***'n'*** is the number of nodes in the tree.

- For ***'searchTransac:ons'*** funcBon : In the worst-case scenario, if the tree is unbalanced and resembles a linked list, and if every category has a large number of expenses, the Bme complexity for searching could be ***O(n*m)***, where ***'n'*** is the number of categories and ***'m'*** is the average number of expenses per category. However, in the average case, assuming the tree is balanced and the number of expenses per category is reasonable, the Bme complexity for searching would be ***O(log n + m).***

Running the code on different datasets

Dataset Size	Time (Average)	T1	T2	T3	T4	T5	T6
92	0.000582666666666	0.000676	0.000453	0.000531	0.000358	0.000786	0.000692
191	0.000649166666666	0.000526	0.000761	0.000572	0.000560	0.000724	0.000752
350	0.000709833333333	0.000696	0.000722	0.000700	0.000742	0.000878	0.000521
480	0.000893666666666	0.000906	0.000839	0.001002	0.000646	0.001055	0.000914
728	0.001031166666666	0.001051	0.001049	0.00106	0.000847	0.000988	0.001192
1367	0.001285666666666	0.001086	0.001246	0.001352	0.001502	0.001213	0.001315

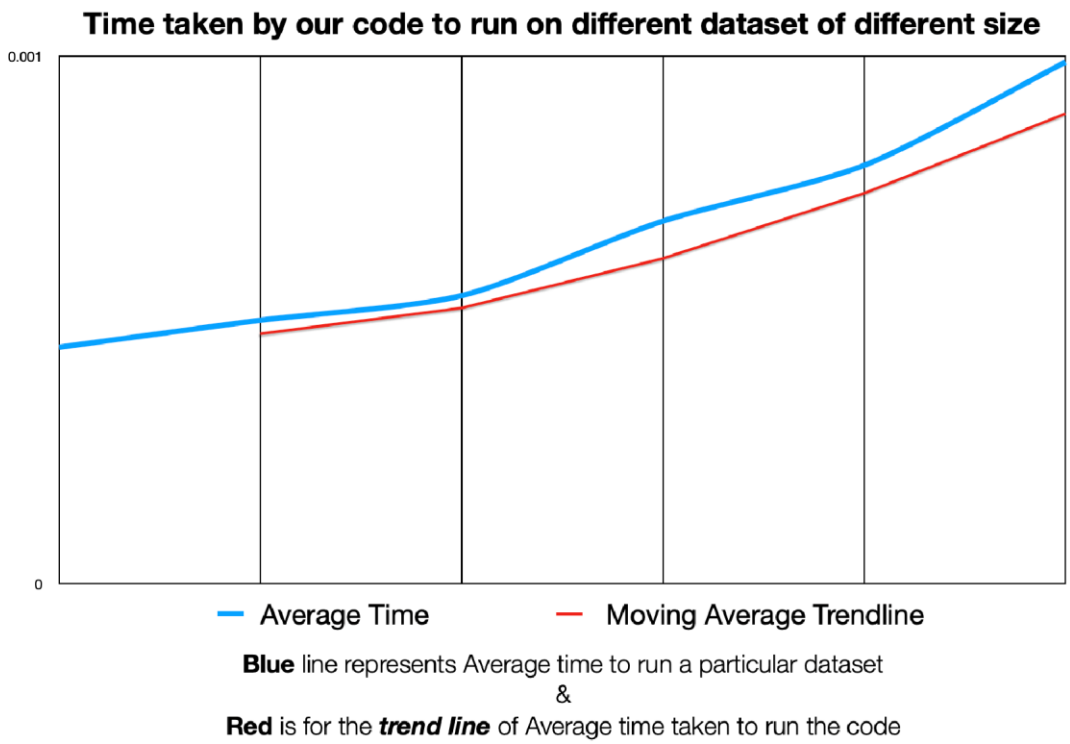
Time taken by our code to run on different dataset of different size*

* Size represents the number of lines each file contains and each line contains - date, payee, amount, category

➢ Different datasets used to test &mings

hTps://drive.google.com/drive/folders/1h_XQghcZm8StUnvq2318B6SBS4lXGGi

➢Result



We get approximately a exponenBal or logarithmic graph which shows that our Bme complexity for **'addingExpense'** funcBon beT er than earlier algorithms which use array based implementaBons.

REFERENCES

- GeeksForGeeks

- w3schools