Data Structure

Homework3: Binary search tree implementation

Author: B11002220 CHI-CHUN, LO

Date: November 19, 2023

國立臺灣科技大學
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Contents

1 Problem **2**
- 1.1 Definition . . . 2
- 1.2 InsertNode . . . 3
- 1.3 DeleteLeaf . . . 4
- 1.4 inorderTraversal . . . 4

2 Code **5**
- 2.1 useful.h . . . 5
- 2.2 main.c . . . 8

3 Result **11**

4 Discussion **13**

5 Conclusion **13**

1 Problem

Write a program to process binary search trees. including the following operations

1. Insert a sequence of characters into a binary search tree. one character at a time.
2. Delete all leaf nodes of the tree built in the previous operation.

After each operation. the program prints out all keys in increasing order for the inorder traversal.

1.1 Definition

簡單來說，這個題目要求我們透過二元搜尋樹完成插入（Insert）和刪除葉子節點（DeleteLeafs）這兩個操作。我們將逐步說明這兩個操作的含義和要求。首先，我們先定義一個二元搜尋樹的結構。二元搜尋樹的每個節點包含儲存自身資料的值，以及左右兩個子節點的指標。在每個操作之後，程序將按照中序遍歷的方式，打印出所有鍵值的遞增順序。

```
typedef struct node* treePointer;
typedef struct node
{
  int data;
  treePointer left;
  treePointer right;
} Node;

```

值得注意的是，二元搜尋樹有以下限制與要求：

- 所有左子樹的節點值必須小於父節點的值。
- 所有右子樹的節點值必須大於父節點的值。
- 每個節點的值必須是唯一的，不允許重複值。

針對第一個操作，對二元樹 (BST) 插入節點。我們可以定義一個函數 InsertNode，它的輸入是這個結構的 root，輸出是 root 節點，透過遞迴將資料插入到正確位置：

```
treePointer insertNode(treePointer root, char data)

```

針對第二個操作，將二元樹 (BST) 的葉子 (leaf) 節點都刪除。我們可以定義一個函數 DeleteLeafs，它的輸入是這個結構的 root，輸出是 root 節點，在遞迴的過程中若是發現葉子節點就進行刪除的動作：

```
treePointer deleteLeaf(treePointer root)

```

根據作業規定，我們將使用一個名為 Input.txt 的檔案作為輸入文件，Output.txt 作為輸出文件檔案。每當 Input.txt 的輸入一行文字時，就會輸出其 inorder 排序的結果與刪除子葉節點的結果。我們可以針對整個程式的 main.c 大致的將整個程式的流程設計好，如下:

```
int main()
{
  clear_file();
  // Set input file
  FILE* inputFile;
  if (fopen_s(&inputFile, "Input.txt", "r") != 0)
  {
    fprintf(stderr, "Failed to open Input.txt\n");
    return 1;
  }

  // Read each line
  char line[100];
  while (fgets(line, sizeof(line), inputFile) != NULL)
  {
    line[strcspn(line, "\n")] = '\0'; // delete  \n
    treePointer BST = NULL;
    // Read a char and send it into BST
    // first opeation
    for (int i = 0; line[i]; i++)
    {
      BST = insertNode(BST, line[i]);
    }
    inorderTraversal(BST);
    printf("\n");
    // second opeartion
    deleteLeaf(BST);
    inorderTraversal(BST);
    printf("\n");
  }
  fclose(inputFile);
  return 0;
}

```

1.2 InsertNode

對於 INSERTNODE() 函數，我們會根據數據大小往右或往左遞迴找到合適擺放新資料的位置。另外對於 INSERTNODE() 函數，我們會考慮到當插入的資料已經存在於 BST 中，則會直接跳過。

Algorithm 1: Insert a node into a binary search tree

```
Function insertNode(root, data):
    Input: root: treePointer, data: char
    if root is NULL then
        return createNode(data);
    end
    if root->data is less than data then
        root->right = insertNode(root->right, data);
    end
    if root->data is greater than data then
        root->left = insertNode(root->left, data);
    end
    return root;
```

1.3 DeleteLeaf

這段程式碼的目的是刪除二元搜尋樹中的葉子節點，透過遞迴地遍歷樹的每個節點，並在遇到葉子節點時將其釋放。這樣可以逐步刪除所有的葉子節點，最終達到刪除目標的效果。

Algorithm 2: Delete leaf nodes in a binary search tree

```
Input: root: treePointer
Result: root after deleting leaf nodes
Function deleteLeaf(root):
    if root is not null then
        if both root->left and root->right are null then
            release current node;
            return null;
        end
        deleteLeaf(root->left);
        deleteLeaf(root->right);
    end
    return root;
```

1.4 inorderTraversal

這段程式碼的目的是對一個二元搜尋樹進行 inorder traversal。

Algorithm 3: Inorder traversal of a binary search tree

Input: root: treePointer
Result: root you gave before
Function `inorderTraversal(root)`**:**
  if root is not null **then**
    `inorderTraversal(root->left);`
    print root->data;
    `inorderTraversal(root->right);`
  end

2 Code

下面是 main.c 完整的程式碼，我有額外編寫一個常用的 useful.h 檔案，裡面有許多常用的函式，裡面有重新定義 printf 的程式，針對不同環境下的 C 編譯器設定。

2.1 useful.h

```c
#ifndef __useful__
#define __useful__

#include "stdio.h"
#include "stdlib.h"
#include "time.h"

/**
* @brief Define the filename for the output file.
*/
#define OUTPUT_FILE "Output.txt"

#ifdef __PRINTF_TO_FILE__BY_VISUAL_STUDIO
/**
* @brief Clear the content of the output file.
*/
#define clear_file() \
    do { \
      FILE *file; \
      if(fopen_s(&file, OUTPUT_FILE, "w") == 0) { \
        if (file != NULL) { \
          fprintf(file,""); \
          fclose(file); \
        } \
      } \
    } while (0)

/**
* @brief Print content to the output file.
*/
#define printf(...) \
    do { \
```

```
            FILE *file; \
            if(fopen_s(&file, OUTPUT_FILE, "a") == 0) { \
              fprintf(file,__VA_ARGS__); \
              fclose(file); \
            } \
          } while (0)

    #endif

    /**
    * @brief Standard C fopen
    */
    #ifdef __PRINTF_TO_FILE__
    /**
      * @brief Clear the content of the output file.
      */
    #define clear_file() \
        do { \
          FILE *file = fopen(OUTPUT_FILE, "w"); \
          if (file != NULL) { \
            fclose(file); \
          } \
        } while (0)

      /**
      * @brief Print content to the output file.
      */
    #define printf(...) \
        do { \
          FILE *file = fopen(OUTPUT_FILE, "a"); \
          if (file != NULL) { \
            fprintf(file, __VA_ARGS__); \
            fclose(file); \
          } \
        } while (0)

    #endif

    #ifndef __PRINTF_TO_FILE__BY_VISUAL_STUDIO &&
__PRINTF_TO_FILE__
      /**
      * @brief If __PRINTF_TO_FILE__ is not defined, do nothing.
      */
    #define clear_file() \
        do { \
        } while (0)
    #endif

    // Display the entire array
    #define PRINT_ARR(ARR, SIZE) \
      do \
      { \
        for (int i = 0; i < SIZE; i++) \
          printf("%d ", ARR[i]); \
        printf("\n"); \
      } while (0)
```

```
89          // Swap
90          #define SWAP(a, b, t) (t = a, a = b, b = t)
91
92          /**
93          * @brief Compare two numbers. Returns 1 if a > b, 0 if a = b,
    -1 if a < b.
94          * @param a
95          * @param b
96          */
97          #define COMPARE(a, b) (a > b ? 1 : (a < b ? -1 : 0))
98
99          // Allocate memory
100         #define MALLOC(arr, size) \
101           do \
102           { \
103             if (!(arr = malloc(size))) \
104             { \
105               fprintf(stderr, "Malloc error\n"); \
106               exit(EXIT_FAILURE); \
107             } \
108           } while (0)
109
110         #define REMALLOC(arr, size) \
111           do \
112           { \
113             if (!(arr = realloc(arr, size))) \
114             { \
115               fprintf(stderr, "Realloc error\n"); \
116               exit(EXIT_FAILURE); \
117             } \
118           } while (0)
119
120         // Display the execution time of a function (input: void)
121         #define TIME_FUNCTION(func)
                        \
122           do
                    \
123           {
                    \
124             clock_t start = clock();
                  \
125             func();
                   \
126             clock_t end = clock();
                  \
127             double cpu_time_used = ((double)(end - start)) /
    CLOCKS_PER_SEC;             \
128             printf("Function %s takes %f seconds to execute.\n", #func,
     cpu_time_used); \
129         } while (0)
130
131         #endif
132
133
```

2.2 main.c

一開始的定義是為了調用 useful.h 重新定義好的 printf，可以讓 printf 到檔案中，而不是在螢幕上顯示。

```c
/**
* @file BST.c
* @author Leo (you@domain.com)
* @brief HW3 BST
* @version 0.1
* @date 2023-11-17
*
* @copyright Copyright (c) 2023
*
*/
#define __PRINTF_TO_FILE__BY_VISUAL_STUDIO

#include "useful.h"
typedef struct node *treePointer;
typedef struct node
{
  int data;
  treePointer left;
  treePointer right;
} Node;
treePointer createNode(char data)
{
  treePointer newNode;
  MALLOC(newNode, sizeof(*newNode));
  newNode->data = data;
  newNode->left = NULL;
  newNode->right = NULL;
  return newNode;
}
/**
* @brief Delete leaf nodes in a binary tree.
* @param root The root node of the binary tree.
* @return treePointer The modified root node of the binary tree.
*/
treePointer deleteLeaf(treePointer root)
{
  if (root)
  {
    if (root->left == NULL && root->right == NULL) // Check if it is a leaf node
    {
      free(root);  // Free the memory space of the leaf node
      return NULL; // Return a null pointer to indicate the deletion of the leaf node
    }
    // Recursively delete leaf nodes in the left subtree and right subtree
    root->left = deleteLeaf(root->left);
    root->right = deleteLeaf(root->right);
  }
  return root; // Return the modified root node of the binary tree
}

```

```
51  /**
52  * @brief Insert a node at the specified position.
53  *
54  * @param root The current node.
55  * @param data The character data to be inserted.
56  * @return treePointer
57  * @note: There are two possible situations that can occur.
58  *          1. Return the address of the new node if the root node
    is empty!
59  *          2. Return root to pass the previously  address to root->
    right or root->left.
60  */
61  treePointer insertNode(treePointer root, char data)
62  {
63    // Handle the case where the current node is empty
64    if (root == NULL)
65    {
66      return createNode(data);
67    }
68    // Search to the right
69    if (root->data < data)
70    {
71      root->right = insertNode(root->right, data);
72    }
73    // Search to the left
74    if (root->data > data)
75    {
76      root->left = insertNode(root->left, data);
77    }
78    // Make sure that right & left store the current root node to
    connect successfully
79    return root;
80  }
81
82  /**
83  * @brief Search for a node containing the specified data in a
    binary search tree.
84  * @param root The root node of the binary search tree.
85  * @param data The data to search for.
86  * @return treePointer a pointer to the node containing the search
    data, or NULL if not found.
87  */
88  treePointer search(treePointer root, char data)
89  {
90    if (!root || root->data == data) // if current node or data find
    then return current node
91      return root;
92    if (root->data < data) // search from right
93      return search(root->right, data);
94    else // search from left
95      return search(root->left, data);
96  }
97  /**
98  * @brief Perform an inorder traversal of a binary tree.
99  *
100 * @param root The root node of the binary tree.
101 * @return void
```

```
102     */
103     void inorderTraversal(treePointer root)
104     {
105       if (root != NULL)
106       {
107         inorderTraversal(root->left);
108         printf("%c", root->data);
109         inorderTraversal(root->right);
110       }
111     }
112
113     void printTree(treePointer root, int level)
114     {
115       if (root == NULL)
116       {
117         return;
118       }
119
120       printTree(root->right, level + 1);
121
122       for (int i = 0; i < level; i++)
123       {
124         printf("    ");
125       }
126       printf("%c\n", root->data);
127
128       printTree(root->left, level + 1);
129     }
130     int main()
131     {
132       clear_file();
133       // Set input file
134       FILE* inputFile;
135       if (fopen_s(&inputFile, "Input.txt", "r") != 0)
136       {
137         fprintf(stderr, "Failed to open Input.txt\n");
138         return 1;
139       }
140
141       // Read each line
142       char line[100];
143       while (fgets(line, sizeof(line), inputFile) != NULL)
144       {
145         line[strcspn(line, "\n")] = '\0'; // delete  \n
146         treePointer BST = NULL;
147         // Read a char and send it into BST
148         // first opeation
149         for (int i = 0; line[i]; i++)
150         {
151           BST = insertNode(BST, line[i]);
152         }
153         inorderTraversal(BST);
154         printf("\n");
155         // second opeartion
156         deleteLeaf(BST);
157         inorderTraversal(BST);
158         printf("\n");
```

```
    }
    fclose(inputFile);
    return 0;
}

```

3 Result

在這邊我們會針對不同的例子去測試程式的穩定性、正確性。

- **example1:** 討論輸入順序數字

  Input:

  ```
  123

  ```

  Output:

  ```
  123
  12

  ```

  解釋: 第一列是插入後以 Inorder 的順序輸出結果，第二列是刪除葉子節點的輸出結果。

- **example2:** 討論輸入的節點有兩個葉子節點

  Input:

  ```
  213

  ```

  Output:

  ```
  123
  2

  ```

  解釋: 第一列是插入後以 Inorder 的順序輸出結果，第二列是刪除葉子節點的輸出結果。確實葉子節點 1 與 3 都被刪除了。

- **example3:** 重複數字輸入造成是否會輸入到二元搜索樹內?

  Input:

  ```
  223311

  ```

  Output:

  ```
  123
  2

  ```

  第一列是插入後以 Inorder 的順序輸出結果，第二列是刪除葉子節點的輸出結果。可以看出重複的數字會被拋棄掉!

- **example4:** 測試只有輸入一個數字時，刪除葉子節點是否會有問題發生?

  如果依照之前設計的程式葉子節點會被刪除，但實際上我們不能把 root 節點刪除，程式需要把這個狀態排除，所以下面是我針對 root 節點被刪除的問題做的修正。

  Imporve program:

```
/**
* @brief Delete leaf nodes in a binary tree.
* @param root The root node of the binary tree.
* @param level it save the current root's level .
* @return treePointer The modified root node of the binary tree.
*/
treePointer deleteLeaf(treePointer root,int level)
{
  if (root)
  {
    if (root->left == NULL && root->right == NULL && level!= 0) // Check if it is a leaf node
    {
      free(root);  // Free the memory space of the leaf node
      return NULL; // Return a null pointer to indicate the deletion of the leaf node
    }
    // Recursively delete leaf nodes in the left subtree and right subtree
    root->left = deleteLeaf(root->left,level+1);
    root->right = deleteLeaf(root->right, level + 1);
  }
  return root; // Return the modified root node of the binary tree
}


```

  Input:

```
    2

```

  Output:

```
    2
    2

```

  第一列是插入後以 Inorder 的順序輸出結果，第二列是刪除葉子節點的輸出結果。這個例子本身是 root 節點同時也是葉子節點，可以有兩種結果發生，一是把 root 節點刪除，二是保留 root 節點。這邊理應把 root 節點保留避免發生錯誤。這邊是做成不刪除。

4 Discussion

在本節中，我們對二元搜尋樹（Binary Search Tree，BST）的整體輸入和輸出進行了簡要說明。只要當前的葉子節點不是 BST 樹的根節點，就可以進行刪除葉子節點的操作。在插入過程中，重複的節點會被拋棄，確保 BST 中不會存儲重複的數字。

接下來，我們討論了程式的效率，其中 n 表示當前節點的數量。

- 插入操作的時間複雜度為 $O(\log n)$
- 刪除操作的時間複雜度為 $O(n)$。

為了進一步提高效率，我們可以檢測並跳過重複數字的插入。在原始設計中，相同的數字不會被插入，但仍會執行對數時間的運算。我們可以修改程式碼，直接跳過插入操作。

```
// 新增accessed_before偵測是否有使用過這個文字!
char accessed_before[256] = {0};
for (int i = 0; line[i]; i++) {
if (accessed_before[line[i]] == 0) {
accessed_before[line[i]]++ ;
BST = insertNode(BST, line[i]);
} else {
continue ;
}
}

```

5 Conclusion

這次的作業，我花的比較多的時間在重定義的設計上，以及程式碼的優化上。因為之前的作業中常常會用到輸入、輸出的功能，格式和內容都比較固定，所以對於這次的作業，我花了很多時間在研究如何設計出符合作業需求的輸入、輸出格式。

在程式碼的優化方面，我發現了程式碼中重複的節點檢查，以及重複的數字檢查。這些檢查可以被合併，以提高程式效率。這些都是非常有用的技能，在日後的程式開發中會派上用場。