Data Structure2 Report

과목명 | 자료구조2

담당교수 | 홍 민 교수님

학과 | 컴퓨터소프트웨어공학과

학년 | 2학년

학번 | 20194059

이름 | 김태완

제출일 | 2020.10.08

목차

1. 이진 탐색 트리를 이용한 순환, 반복 삽입 연산의 소요 시간 비교와 여러가지 연산

1.1 문제 분석

SCH 순천향대학교

HW 2 (10월 8일까지 제출)

- 파일 data.txt(파일로 제공됨)에 아래와 같은 정수값들이 2만개 저장되어 있다.
  - 10000 13442 5302 1107 19005 …
- 순환과 반복으로 이루어진 이진 탐색 트리를 구현하고 모든 데이터를 삽입하는데 걸리는 시간을 비교 하시오.
- 현재 구성된 트리의 전체 노드 개수가 몇 개인지를 구하는 코드를 구현하여 출력 하시오.
- 현재 구성된 트리의 높이를 구하는 코드를 구현하여 출력 하시오.
- 현재 구성된 트리의 단말 노드가 몇 개인지를 구하는 코드를 구현하여 출력 하시오.

2 학기 자료구조 두번째 과제이다. 이번 프로그램은 수업시간에 배운 순환, 반복을 이용한 이진 탐색 트리의 삽입 연산을 주어진 2 만개의 데이터를 이용해 각각 알고리즘마다 얼마나 시간이 걸리는지 계산 하고, 두 알고리즘의 소요 시간을 비교 하는 것이 첫번째 주어진 미션이다. 삽입 연산을 마친 후에는 데이터가 삽입된 이진 탐색 트리에 대해 트리의 전체 노드 개수, 트리의 높이, 단말 노드의 개수를 구하는 알고리즘을 작성 하면 되는 프로그램이다. 이미 수업시간에 이 모든 알고리즘에 대해서 교수님께서 알려주셨기 때문에 코드를 작성 하는 것은 누구나 문제 없이 할 수 있지만 이것을 아무것도 모르고 그냥 따라 쓰는 것 보다는, 이 코드들이 어떻게 돌아가는지 하나하나의 로직을 파악하는 것이 가장 중요 한 것 같다.

1.2 소스코드

```c
//자료구조2_HW2
//작성자 : 20194059 김태완
//작성일 : 2020.10.07
//프로그램명 : 이진트리를 이용한 여러가지 연산

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

typedef struct TreeNode
// TreeNode 구조체 정의
{
    int key; // key값
    struct TreeNode *left, *right; // 자식 노드
}TreeNode;

// 함수 원형 정의
void error(char* string);
TreeNode* new_node(int item);
TreeNode* insert_node_recursive(TreeNode *node, int data);
void insert_node_loop(TreeNode **root, int data);
int get_node_count(TreeNode *node);
int get_max(int n1, int n2);
int get_height(TreeNode *node);
int get_leaf_count(TreeNode *node);
void free_memory(TreeNode *root);
void display_format(char *string);
void display_time(double time);
void display_result(int count_node, int height, int count_leaf);

int main()
{
    FILE *fp = NULL; // 파일 포인터 fp
    int tmp; // 파일에서 받아온 정수를 임시로 저장해둘 변수 tmp
    TreeNode *node1 = NULL; // 첫번째 이진트리
    TreeNode *node2 = NULL; // 두번째 이진트리
    double time; // 소요 시간을 저장해둘 time
    clock_t start, finish; // 시작 시간과 끝나는 시간을 저장해주는 변수

    fp = fopen("data.txt", "r"); // 읽기 모드로 data.txt 파일 오픈

    if (fp == NULL) // 파일오픈 오류 처리
    {
        error("file open error");
    }

    display_format("     순환     "); // 순환으로 삽입 연산 진행
    start = clock(); // 시작 시간 측정
    while (!feof(fp)) // 파일의 끝까지
    {
        fscanf(fp, "%d", &tmp); // 정수 하나씩 읽어서
        node1 = insert_node_recursive(node1, tmp); // 삽입(순환) 연산 진행
    }
    finish = clock(); // 종료 시간 측정
    time = (double)(finish - start) / CLOCKS_PER_SEC; // 소요 시간 계산
    display_time(time); // 소요 시간 출력

    rewind(fp); // 파일포인터를 맨 앞으로 돌린다
    printf("₩n");
```

```
62      display_format("     반복     "); // 반복으로 삽입 연산 진행
63      start = clock(); // 시작 시간 측정
64      while (!feof(fp)) // 파일의 끝까지
65      {
66          fscanf(fp, "%d", &tmp); // 정수를 하나씩 읽어서
67          insert_node_loop(&node2, tmp); // 삽입(반복) 연산 진행
68      }
69      finish = clock(); // 종료 시간 측정
70      time = (double)(finish - start) / CLOCKS_PER_SEC; // 소요 시간 계산
71      display_time(time); // 소요 시간 출력
72
73      printf("Wn");
74
75      display_format("     연산     "); // 연산 결과
76      display_result(get_node_count(node1), get_height(node1), get_leaf_count(node1));
77      // 총 노드의 개수, 트리의 높이, 단말 노드의 개수 출력
78
86  void error(char* string)
87  // 에러를 출력하고 프로그램을 종료하는 함수
88  {
89      printf("%sWn", string); // 매개변수로 받아온 문자열 (오류문장) 출력
90      exit(1);
91  }
92
93  TreeNode* new_node(int data)
94  // 새로운 노드를 생성하고 자식 노드를 NULL로 설정하는 함수
95  {
96      TreeNode *temp = (TreeNode *)malloc(sizeof(TreeNode)); // 메모리 할당
97      if (temp == NULL) // 동적메모리 할당 오류 처리
98      {
99          error("memory allocate error");
100     }
101     temp->key = data; // key 에 받아온 item 저장
102     temp->left = NULL; // 왼쪽 자식 노드 NULL
103     temp->right = NULL; // 오른쪽 자식 노드 NULL
104
105     return temp;
106 }
107
108 TreeNode* insert_node_recursive(TreeNode *node, int data)
109 // 순환의 방식으로 이진 탐색 트리의 삽입 연산을 진행하는 함수
110 {
111     if (node == NULL) // 단말 노드일 경우
112     {
113         return new_node(data); // 새로운 노드를 생성
114     }
115
116     if (data < node->key) // 받아온 데이터가 부모노드의 key값보다 작을경우
117     {
118         node->left = insert_node_recursive(node->left, data);
119         // 왼쪽 노드로 이동
120     }
121     else if (data > node->key) // 받아온 데이터가 부모노드의 key값보다 클경우
122
123     {
124         node->right = insert_node_recursive(node->right, data);
125         // 오른쪽 노드로 이동
126     }
127
128     return node;
129 }
130
```

```
130
131 void insert_node_loop(TreeNode **root, int data)
132 // 반복의 방식으로 이진 탐색 트리의 삽입 연산을 진행하는 함수
133 {
134     TreeNode *p, *t;
135     TreeNode *n;
136
137     t = *root;
138     p = NULL;
139
140     while (t != NULL) // t 가 비어있지 않을 때 까지
141     {
142         if (data == t->key) // data와 key값이 같을 경우
143         {
144             return; // 함수 종려
145         }
146         p = t; // 이전 노드의 주소를 기억 하기 위해 포인터 변수 P 사용
147         if (data < t->key) // data 가 key 값보다 작을 경우
148         {
149             t = t->left; // 왼쪽 자식 노드 진입
150         }
151         else // data 가 key 값보다 클 경우
152         {
153             t = t->right; // 오른쪽 자식 노드 진입
154         }
155     }
156
157     n = new_node(data); // 새로운 노드 생성
158
159     if (p != NULL) // p가 비어있지 않을 경우
160     {
161         if (data < p->key) // data 가 key 값보다 작을 경우
162         {
163             p->left = n; // 왼쪽에 새로운 노드 연결
164         }
165         else if (data > p->key) // data 가 key 값보다 클 경우
166         {
167             p->right = n; // 오른쪽에 새로운 노드 연결
168         }
169     }
170     else // p 가 비어있는 경우
171     {
172         *root = n; // 새로운 노드를 루트로 지정
173     }
174 }
175
176 int get_node_count(TreeNode *node)
177 // 이진 탐색 트리의 노드 개수를 구하는 함수
178 {
179     int count = 0;
180     if (node != NULL) // 노드가 비어있지 않을 경우
181     {
182         count = 1 + get_node_count(node->left) + get_node_count(node->right);
183         // 각 자식 노드에 대해 순환 호출 한 후 1을 더해 전체 노드의 개수를 구한다.
184     }
185     return count;
186 }
187
188 int get_max(int n1, int n2)
189 // 두 값중 큰 수를 구하는 함수
190 {
191     if (n1 > n2) // n1이 더 큰 경우
192     {
193         return n1; // n1 반환
194     }
195     else // n2가 더 큰 경우
196     {
197         return n2; // n2 반환
198     }
199 }
200
```

```c
int get_height(TreeNode *node)
// 이진 탐색 트리의 높이를 구하는 함수
{
    int height = 0;
    if (node != NULL) // 노드가 비어있지 않은 경우
    {
        height = 1 + get_max(get_height(node->left), get_height(node->right));
        // 왼쪽 노드와 오른쪽 노드중 더 높이가 큰 쪽을 순환호출하고 1을 더해 트리의 전체 높이 계산.
    }
    return height;
}

int get_leaf_count(TreeNode *node)
// 이진 탐색 트리의 단말 노드 개수를 구하는 함수
{
    int count = 0;

    if (node != NULL)
    {
        if (node->left == NULL && node->right == NULL) // 단말 노드일 경우
        {
            return 1; // 1 반환
        }
        else // 비단말 노드의 경우
        {
            count = get_leaf_count(node->left) + get_leaf_count(node->right);
            // 왼쪽 오른쪽 자식 노드 각각에 대해 순환 호출 하며 반환값을 더해주어 단말 노드의 개수 계산.
        }
    }
    return count;
}

void free_memory(TreeNode *root)
// 후위순회 하며 사용한 메모리를 반납하는 함수
{
    if (root != NULL)
    {
        free_memory(root->left); // 왼쪽 자식 노드 접근
        free_memory(root->right); // 오른쪽 자식 노드 접근
        free(root); // 메모리 해제
    }
}

void display_format(char *string)
// 이후 나올 출력에 대한 형식을 출력 해주는 함수
{
    printf("┌──────────────────────────────────────────────────┐\n");
    printf("│        %4s      │\n", string); // 매개변수로 받아온 문자열 출력
    printf("└──────────────────────────────────────────────────┘\n");
}

void display_time(double time)
// 소요된 시간을 출력 해주는 함수
{
    printf("┌──────────────────────────────────────────────────┐\n");
    printf("│  소요시간 : %lf       │\n", time); // 매개변수로 받아온 소요 시간 출력
    printf("└──────────────────────────────────────────────────┘\n");
}

void display_result(int count_node, int height, int count_leaf)
// 이진 탐색 트리에 대한 추가 연산 결과를 출력 해주는 함수
{
    printf("┌──────────────────────────────────────────────────┐\n");
    printf("│  전체 노드의 개수 : %5d  │\n", count_node); // 전체 노드의 개수 출력
    printf("│  이진트리의 높이: %3d      │\n", height); // 트리의 높이 출력
    printf("│  단말 노드의 개수 : %5d  │\n", count_leaf); // 단말 노드의 개수 출력
    printf("└──────────────────────────────────────────────────┘\n");
}
```

1.3 소스코드 분석

```
1  //자료구조2_HW2
2  //작성자 : 20194059 김태완
3  //작성일 : 2020.10.07
4  //프로그램명 : 이진트리를 이용한 여러가지 연산
5
```

1. 프로그램의 작성자, 작성일, 프로그램 이름을 포함한 주석을 작성한다.

```
6  #define _CRT_SECURE_NO_WARNINGS
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <time.h>
10
```

2. 프로그램 작성에 필요한 헤더파일들을 포함한다. 이때 시간 측정과 관련된 함수들을 사용해야 함을 time.h 도 포함 해준다.

```
11 typedef struct TreeNode
12 // TreeNode 구조체 정의
13 {
14     int key; // key값
15     struct TreeNode *left, *right; // 자식 노드
16 }TreeNode;
17
```

3. 이진탐색트리의 노드에 대한 구조체 TreeNode 를 정의한다.
key 값은 모두 정수로 이루어져 있으므로 int 형으로 선언 해주고 각각 자식 노드에 해당되는 left, right 도 함께 선언 해준다.

```
18 // 함수 원형 정의
19 void error(char* string);
20 TreeNode* new_node(int item);
21 TreeNode* insert_node_recursive(TreeNode *node, int data);
22 void insert_node_loop(TreeNode **root, int data);
23 int get_node_count(TreeNode *node);
24 int get_max(int n1, int n2);
25 int get_height(TreeNode *node);
26 int get_leaf_count(TreeNode *node);
27 void free_memory(TreeNode *root);
28 void display_format(char *string);
29 void display_time(double time);
30 void display_result(int count_node, int height, int count_leaf);
31
```

4. 밑에서 작성한 사용자 정의 함수들의 원형을 정의 해준다.

*** 원활한 코드 설명을 위해 직접 정의한 함수들에 대해 먼저 설명 하겠습니다.

```
86  void error(char* string)
87  // 에러를 출력하고 프로그램을 종료하는 함수
88  {
89      printf("%sWn", string); // 매개변수로 받아온 문자열 (오류문장) 출력
90      exit(1);
91  }
92
```

5. 매개변수로 받아온 문자열을 출력하고 프로그램을 종료시키는 error 함수를 정의한다.

```
93  TreeNode* new_node(int data)
94  // 새로운 노드를 생성하고 자식 노드를 NULL로 설정하는 함수
95  {
96      TreeNode *temp = (TreeNode *)malloc(sizeof(TreeNode)); // 메모리 할당
97      if (temp == NULL) // 동적메모리 할당 오류 처리
98      {
99          error("memory allocate error");
100     }
101     temp->key = data; // key 에 받아온 item 저장
102     temp->left = NULL; // 왼쪽 자식 노드 NULL
103     temp->right = NULL; // 오른쪽 자식 노드 NULL
104
105     return temp;
106 }
107
```

6. 새로운 노드를 생성하는 new_node 함수이다. 동정 메모리 할당을 통해 메모리를 새로 할당 받은 후 key 에는 매개변수로 불러온 data 를, left 와 right 에는 단말 노드로 만들기 위해 NULL 값을 넣어 준다.

```
108 TreeNode* insert_node_recursive(TreeNode *node, int data)
109 // 순환의 방식으로 이진 탐색 트리의 삽입 연산을 진행하는 함수
110 {
111     if (node == NULL) // 단말 노드일 경우
112     {
113         return new_node(data); // 새로운 노드를 생성
114     }
115
116     if (data < node->key) // 받아온 데이터가 부모노드의 key값보다 작을경우
117     {
118         node->left = insert_node_recursive(node->left, data);
119         // 왼쪽 노드로 이동
120     }
121     else if (data > node->key) // 받아온 데이터가 부모노드의 key값보다 클경우
122
123     {
124         node->right = insert_node_recursive(node->right, data);
125         // 오른쪽 노드로 이동
126     }
127
128     return node;
129 }
130
```

7. 순환 방식으로 이진 탐색 트리의 삽입 연산을 수행하는 함수이다. 노드가 비어있을 경우 새로운 노드를 만들어 반환 하고, 매개변수로 불러온 데이터가 key 값보다 작으면 왼쪽 자식 노드로 순환 호출, key 값보다 크면 오른쪽 자식 노드로 순환 호출하여 데이터를 삽입한다.

```
130
131 void insert_node_loop(TreeNode **root, int data)
132 // 반복의 방식으로 이진 탐색 트리의 삽입 연산을 진행하는 함수
133 {
134     TreeNode *p, *t;
135     TreeNode *n;
136
137     t = *root;
138     p = NULL;
139
140     while (t != NULL) // t 가 비어있지 않을 때 까지
141     {
142         if (data == t->key) // data와 key값이 같을 경우
143         {
144             return; // 함수 종려
145         }
146         p = t; // 이전 노드의 주소를 기억 하기 위해 포인터 변수 P 사용
147         if (data < t->key) // data 가 key 값보다 작을 경우
148         {
149             t = t->left; // 왼쪽 자식 노드 진입
150         }
151         else // data 가 key 값보다 클 경우
152         {
153             t = t->right; // 오른쪽 자식 노드 진입
154         }
155     }
156
157     n = new_node(data); // 새로운 노드 생성
158
159     if (p != NULL) // p가 비어있지 않을 경우
160     {
161         if (data < p->key) // data 가 key 값보다 작을 경우
162         {
163             p->left = n; // 왼쪽에 새로운 노드 연결
164         }
165         else if (data > p->key) // data 가 key 값보다 클 경우
166         {
167             p->right = n; // 오른쪽에 새로운 노드 연결
168         }
169     }
170     else // p 가 비어있는 경우
171     {
172         *root = n; // 새로운 노드를 루트로 지정
173     }
174 }
175
```

8. 반복을 이용하여 이진 탐색 트리의 삽입 연산을 수행하는 함수이다. 반복의 경우 먼저 매개변수로 불러온 데이터가 이진 탐색 트리에 존재 하는지 검사한다. 이때 데이터가 트리에 존재하는 경우 바로 함수를 종료한다.

9. 트리에 데이터가 존재하지 않는 경우 앞에서 탐색 알고리즘을 통해 데이터가 삽입 될 위치에 가 있으므로 새로운 노드를 연결 해준다. 이때 p 변수를 이용 하는 이유는 t 변수는 위에서 탐색 알고리즘을 수행 하면서 단말 노드의 자식 노드 즉 NULL 값을 갖고 있는 노드에 위치 해 있기 때문에 위에서 자식 노드에 접근 하기 전 주소 값을 p 라는 변수에 계속 저장 해주며 이 전 위치를 기억 하게 해준다. 따라서 새로운 노드의 위치는 p 의 자식 노드 중 하나 일 것이므로 데이터가 p 의 key 값보다 작다면 왼쪽 자식 노드에, 크다면 오른쪽 자식 노드에 삽입을 하게 되는 알고리즘 이다.

```
176 int get_node_count(TreeNode *node)
177 // 이진 탐색 트리의 노드 개수를 구하는 함수
178 {
179     int count = 0;
180     if (node != NULL) // 노드가 비어있지 않을 경우
181     {
182         count = 1 + get_node_count(node->left) + get_node_count(node->right);
183         // 각 자식 노드에 대해 순환 호출 한 후 1을 더해 전체 노드의 개수를 구한다.
184     }
185     return count;
186 }
187
```

10. 전체 노드의 개수를 반환하는 함수이다. 각 자식 노드에 대해 순환 호출을 하며 반환 값을 계속 더하고 맨 마지막으로 root 노드까지 1 을 더하여 그 값을 최종적으로 반환해 전체 노드의 개수를 구한다.

```
188 int get_max(int n1, int n2)
189 // 두 값중 큰 수를 구하는 함수
190 {
191     if (n1 > n2) // n1이 더 큰 경우
192     {
193         return n1; // n1 반환
194     }
195     else // n2가 더 큰 경우
196     {
197         return n2; // n2 반환
198     }
199 }
200
```

11. 트리의 높이를 구할 때 사용되는 함수로 두 정수 중 큰 값을 반환하는 함수이다.

```
201 int get_height(TreeNode *node)
202 // 이진 탐색 트리의 높이를 구하는 함수
203 {
204     int height = 0;
205     if (node != NULL) // 노드가 비어있지 않은 경우
206     {
207         height = 1 + get_max(get_height(node->left), get_height(node->right));
208         // 왼쪽 노드와 오른쪽 노드중 더 높이가 큰 쪽을 순환호출하고 1을 더해 트리의 전체 높이 계산.
209     }
210     return height;
211 }
212
```

12. 트리의 높이를 구하는 함수이다. 왼쪽과 오른쪽의 노드 중 더 높이가 높은 함수를 1 씩 더하며 순환 호출 하여 구하고 마지막으로 root 의 1 을 더해 트리의 최종 높이를 구한다.

```c
int get_leaf_count(TreeNode *node)
// 이진 탐색 트리의 단말 노드 개수를 구하는 함수
{
    int count = 0;

    if (node != NULL)
    {
        if (node->left == NULL && node->right == NULL) // 단말 노드일 경우
        {
            return 1; // 1 반환
        }
        else // 비단말 노드의 경우
        {
            count = get_leaf_count(node->left) + get_leaf_count(node->right);
            // 왼쪽 오른쪽 자식 노드 각각에 대해 순환 호출 하며 반환값을 더해주어 단말 노드의 개수 계산.
        }
    }
    return count;
}

```

13. 트리의 단말 노드 개수를 구하는 함수이다. 비단말 노드의 경우 왼쪽과 오른쪽 노드들을 순환 호출 하고, 단말 노드이 경우엔 1 을 반환해 그 값들을 더하면서 순환 호출해 단말 노드의 개수를 구한다.

```c
void free_memory(TreeNode *root)
// 후위순회 하며 사용한 메모리를 반납하는 함수
{
    if (root != NULL)
    {
        free_memory(root->left); // 왼쪽 자식 노드 접근
        free_memory(root->right); // 오른쪽 자식 노드 접근
        free(root); // 메모리 해제
    }
}

```

14. 동적으로 메모리를 할당 하였으므로 프로그램 종료 직전 메모리를 반납 해주는 함수이다. 이때 전위 순회, 중위 순회가 아닌 후위 순회 방식을 사용 한 이유는 전위 중위 순회의 경우 오른쪽 노드를 거치기 전에 부모 노드를 거치기 때문에 부모 노드의 메모리가 오른쪽 자식 노드보다 먼저 해제 되면 오른쪽 자식 노드는 연결되어있는 부모 노드가 없어 메모리 해제를 할 수 없기 때문에 자식 노드를 모두 거치고 부모 노드로 가는 후위 순회 방식을 사용 하였다.

```c
void display_format(char *string)
// 이후 나올 출력에 대한 형식을 출력 해주는 함수
{
    printf("┌──────────────────────────────────────────────────┐\n");
    printf("│        %4s      │\n", string); // 매개변수로 받아온 문자열 출력
    printf("└──────────────────────────────────────────────────┘\n");
}

```

15. 이후 나올 결과들에 대한 설명을 출력 해주는 함수이다. 매개변수로 받아온 문자열을 주어진 형식으로 출력 해준다.

```c
void display_time(double time)
// 소요된 시간을 출력 해주는 함수
{
    printf("┌──────────────────────────────────────────┐\n");
    printf("| 소요시간 : %lf      |\n", time); // 매개변수로 받아온 소요 시간 출력
    printf("└──────────────────────────────────────────┘\n");
}
```

16. 알고리즘 수행 소요 시간을 지정 형식으로 출력해주는 함수로 매개변수로 시간 값을 받아 오고 그것을 출력 해준다.

```c
void display_result(int count_node, int height, int count_leaf)
// 이진 탐색 트리에 대한 추가 연산 결과를 출력 해주는 함수
{
    printf("┌──────────────────────────────────────────┐\n");
    printf("| 전체 노드의 개수 : %5d |\n", count_node); // 전체 노드의 개수 출력
    printf("| 이진트리의 높이: %3d     |\n", height); // 트리의 높이 출력
    printf("| 단말 노드의 개수 : %5d |\n", count_leaf); // 단말 노드의 개수 출력
    printf("└──────────────────────────────────────────┘\n");
}
```

17. 이진 탐색 트리의 추가 연산 즉 전체 노드의 개수, 트리의 높이, 단말 노드의 개수를 출력 해주는 함수이다.

```c
int main()
{
    FILE *fp = NULL; // 파일 포인터 fp
    int tmp; // 파일에서 받아온 정수를 임시로 저장해둘 변수 tmp
    TreeNode *node1 = NULL; // 첫번째 이진트리
    TreeNode *node2 = NULL; // 두번째 이진트리
    double time; // 소요 시간을 저장해둘 time
    clock_t start, finish; // 시작 시간과 끝나는 시간을 저장해주는 변수

```

18. 메인 함수를 시작 하고 프로그램에 필요한 변수들을 선언 해준다.
파일 포인터 fp, 파일에서 받아온 정수를 임시로 저장해두는 정수형 변수 tmp, 순환 방식으로 삽입된 데이터가 저장될 이진 탐색 트리 node1, 반복 방식으로 삽입된 데이터가 저장될 이진 탐색 트리 node2, 시간을 측정 하기 위한 clock_t 타입의 start 와 finish, 그리고 소요 시간을 저장할 실수형 time 변수이다.

```c
    fp = fopen("data.txt", "r"); // 읽기 모드로 data.txt 파일 오픈

    if (fp == NULL) // 파일오픈 오류 처리
    {
        error("file open error");
    }

```

19. data.txt 파일을 일기 모드로 열고 파일을 열 수 없을 시 오류를 처리해준다.

```
48      display_format("     순환     "); // 순환으로 삽입 연산 진행
49      start = clock(); // 시작 시간 측정
50      while (!feof(fp)) // 파일의 끝까지
51      {
52          fscanf(fp, "%d", &tmp); // 정수 하나씩 읽어서
53          node1 = insert_node_recursive(node1, tmp); // 삽입(순환) 연산 진행
54      }
55      finish = clock(); // 종료 시간 측정
56      time = (double)(finish - start) / CLOCKS_PER_SEC; // 소요 시간 계산
57      display_time(time); // 소요 시간 출력
58
```

20. 순환방식으로 삽입 연산을 진행한다.

21. 데이터 삽입 전 시간을 측정 하고, 파일 포인터에서 받아온 정수를 하나씩 이진 탐색 트리에 삽입 한다.

22. 삽입이 끝나면 시간 측정을 종료하고 두 시간의 차를 이용해 소요 시간을 계산 하고 이를 출력한다.

```
59      rewind(fp); // 파일포인터를 맨 앞으로 돌린다
60      printf("Wn");
61
```

23. 똑같은 파일에서 똑같은 데이터를 다시 받아와야 하기 때문에 파일 포인터를 맨 앞으로 돌려 준다.

```
62      display_format("     반복     "); // 반복으로 삽입 연산 진행
63      start = clock(); // 시작 시간 측정
64      while (!feof(fp)) // 파일의 끝까지
65      {
66          fscanf(fp, "%d", &tmp); // 정수를 하나씩 읽어서
67          insert_node_loop(&node2, tmp); // 삽입(반복) 연산 진행
68      }
69      finish = clock(); // 종료 시간 측정
70      time = (double)(finish - start) / CLOCKS_PER_SEC; // 소요 시간 계산
71      display_time(time); // 소요 시간 출력
72
73      printf("Wn");
74
```

24. 반복방식으로 삽입 연산을 진행한다.

25. 데이터 삽입 전 시간을 측정하고, 파일에서 읽어온 정수를 하나씩 이진 탐색 트리에 삽입 한다.

26. 삽입이 모두 끝나면 시간 측정을 종료하고 소요 시간을 계산한 후 출력 한다.

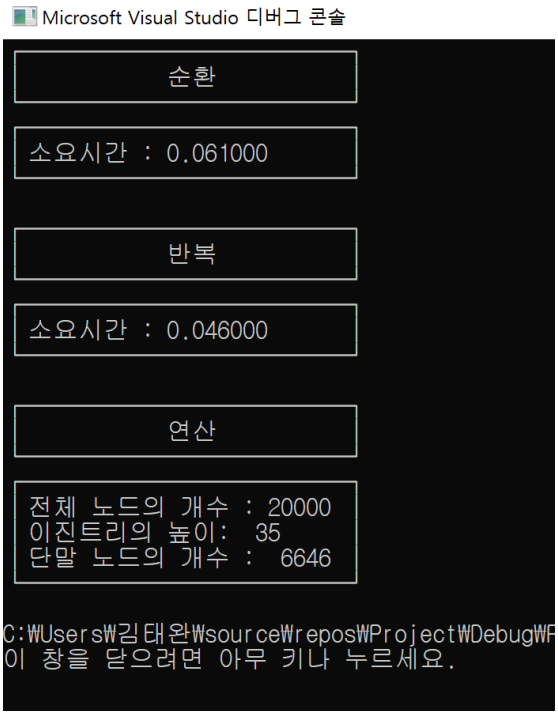
```
75      display_format("     연산     "); // 연산 결과
76      display_result(get_node_count(node1), get_height(node1), get_leaf_count(node1));
77      // 총 노드의 개수, 트리의 높이, 단말 노드의 개수 출력
78
```

27. 이진 탐색 트리의 추가 연산(총 노드의 개수, 트리의 높이, 단말 노드의 개수) 들을 진행 하고 이것의 결과 를 출력 한다.

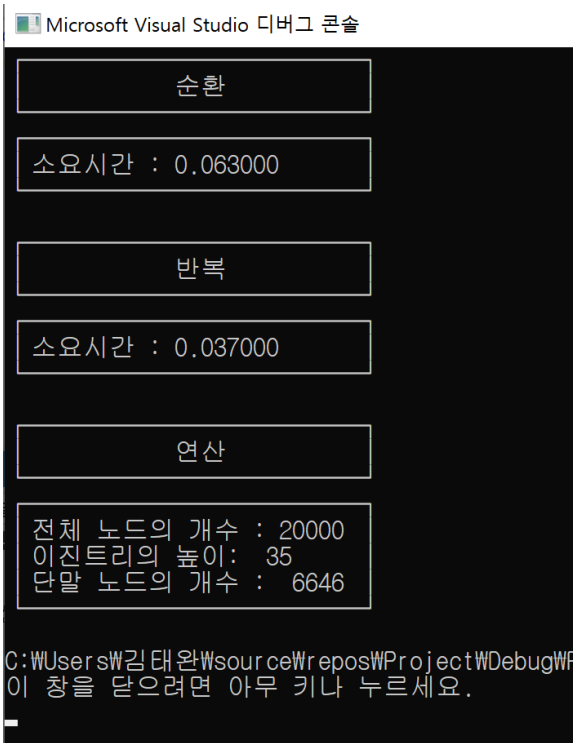
```
79      fclose(fp); // 파일 닫기
80      free_memory(node1); // node1 메모리 반납
81      free_memory(node2); // node2 메모리 반납
82
83      return 0; //프로그램 종료
84  }
85
```

28. 파일을 닫고 메모리들을 반납하고 프로그램을 종료한다.

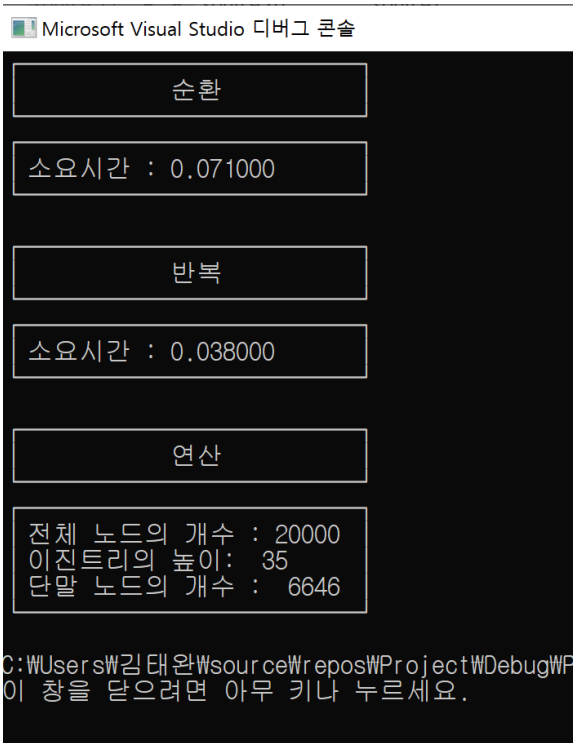
1.4 실행창



〈 출력 화면 1 〉

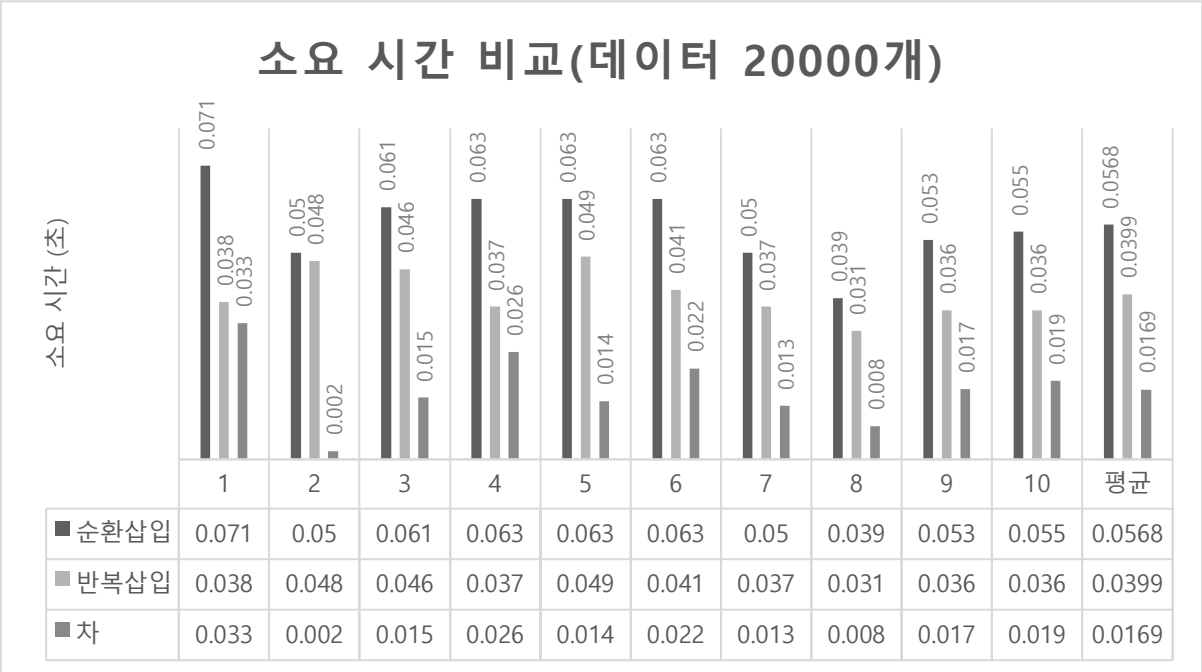

Microsoft Visual Studio 디버그 콘솔
순환
소요시간 : 0.063000
반복
소요시간 : 0.037000
연산
전체 노드의 개수 : 20000
이진트리의 높이: 35
단말 노드의 개수 : 6646
C:₩Users₩김태완₩source₩repos₩Project₩Debug₩P
이 창을 닫으려면 아무 키나 누르세요.

〈 출력 화면 2 〉

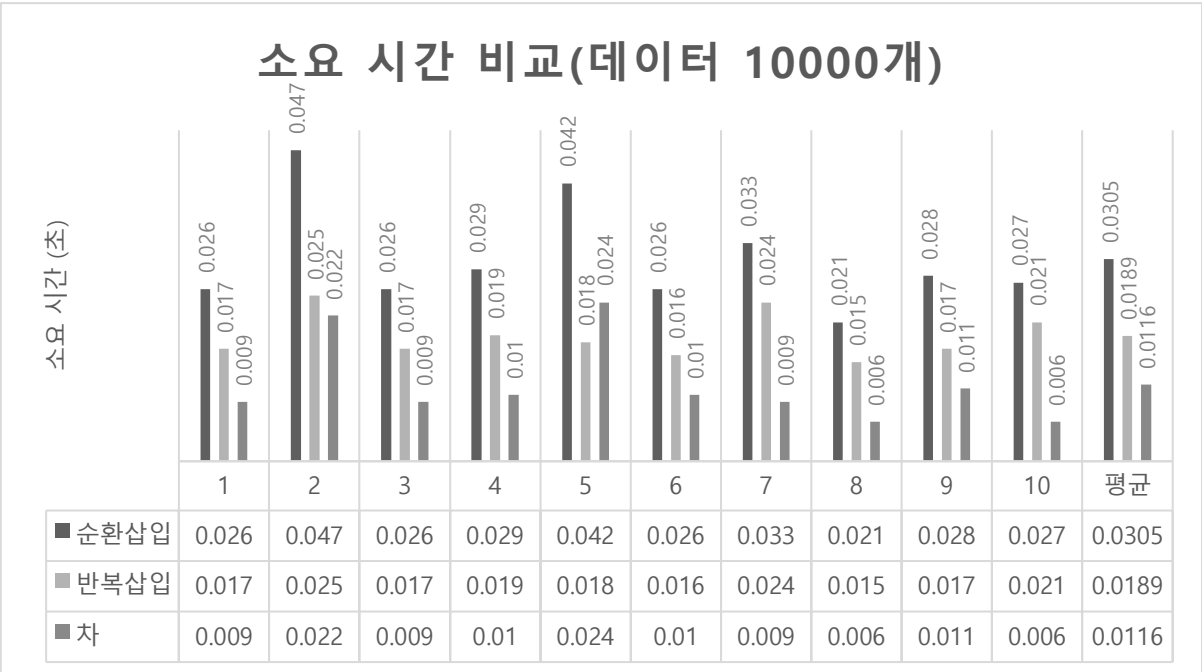

Microsoft Visual Studio 디버그 콘솔
순환
소요시간 : 0.071000
반복
소요시간 : 0.038000
연산
전체 노드의 개수 : 20000
이진트리의 높이: 35
단말 노드의 개수 : 6646
C:₩Users₩김태완₩source₩repos₩Project₩Debug₩P
이 창을 닫으려면 아무 키나 누르세요.

〈 출력 화면 3 〉

1.5 소요 시간 비교



	1	2	3	4	5	6	7	8	9	10	평균
순환삽입	0.071	0.05	0.061	0.063	0.063	0.063	0.05	0.039	0.053	0.055	0.0568
반복삽입	0.038	0.048	0.046	0.037	0.049	0.041	0.037	0.031	0.036	0.036	0.0399
차	0.033	0.002	0.015	0.026	0.014	0.022	0.013	0.008	0.017	0.019	0.0169



	1	2	3	4	5	6	7	8	9	10	평균
순환삽입	0.026	0.047	0.026	0.029	0.042	0.026	0.033	0.021	0.028	0.027	0.0305
반복삽입	0.017	0.025	0.017	0.019	0.018	0.016	0.024	0.015	0.017	0.021	0.0189
차	0.009	0.022	0.009	0.01	0.024	0.01	0.009	0.006	0.011	0.006	0.0116

데이터 2000 개와 10000 개 두개의 경우 모두 반복을 이용한 삽입 연산이 순환을 이용한 삽입 연산보다 약 0.01 초 정도 빠르다는 것을 확인 할 수 있다.

1.6 느낀점

이번 프로그램은 수업시간에 교수님 설명을 들으며 한번씩 보고 배운 코드들이라 따로 작성 하는데 큰 어려움은 없었다. 또한 함수를 호출하는 과정에서 조금 시간이 걸려 순환 호출 하여 삽입 연산을 하는 연산이 반복문을 이용하여 연산하는 방법보다 조금 더 느리다는 것도 1 학기 2 번째 과제에서 내가 추가로 해보며 이미 알고 있던 사실 이었고, 교수님께서도 수업시간 중 반복문을 이용한 연산이 더 빠르다는 것을 여러 번 강조 하셨기 때문에 이 부분에 대해서 이론적으로는 잘 알고 있었지만 실제로 데이터의 개수가 2 만개일 경우와 만개 일 경우 각각 10 번 이상의 실행을 해보면서 두 알고리즘의 실행 시간 차가 많을 경우엔 0.03 초 적게는 0.08 초까지 약간의 차이가 있긴 하지만 항상 반복문을 이용한 삽입 연산이 순환 호출을 이용한 삽입 연산보다 빠른 것을 직접 눈으로 확인 했다. 또한 눈으로만 보았을 때는 잘 이해가 가지 않던 이진 탐색 트리의 전체 노드 개수를 구하는 알고리즘, 트리의 높이를 구하는 알고리즘, 단말 노드의 개수를 구하는 알고리즘 모두 직접 해보고 나니 완전히 이해가 되었다. 선배들께서 장난 반 진담 반으로 '백문이 불여일타' 라며 아무리 코드를 눈으로 보고 이해하려고 해봐도 직접 작성 해보고 디버깅을 돌려보는 것이 훨씬 빠르게 이해 되고 너한테도 도움이 된다는 말을 다시 한번 깨닫게 되는 과제 였다.

2. 느낀점

지금까지 자료구조 1 에서 배웠던 내용들은 수업시간에 한번 집중해서 들으면 완전 이해가 되는 느낌을 받아 복습을 조금 소홀이 했던 경우가 종종 있었다. 하지만 2 학기가 되고 자료구조 2 수업을 들으며 트리 라는 개념을 배우고 이진 탐색 트리에 대해 배우기 시작하면서부터는 수업시간에 아무리 집중을 하고 들어도 이게 무슨 이야기 인지 한번에 이해가 안되는 부분이 많이 있었다. 하지만 다음 수업을 진도를 따라가려면 이 전에 배웠던 내용들을 다 알아야 다음 수업도 그나마 열심히 따라갈 수 있을 것 같아 다음 수업 전에 이전에 배웠던 내용들을 혼자 열심히 복습 하였던 것이 이번 과제를 수월하게 끝낼 수 있게 도와주었던 것 같다. 앞으로 배울 내용들은 지금까지 배웠던 내용들 보다 훨씬 더 어려워 질 것 이기 때문에 앞으로도 예습, 복습 꼭 철저히 해야할 것 같다. 또한 지금까지는 직접 코드를 쳐 보진 않고 그냥 눈으로만 보고 말았는데 앞으로는 수업시간에 나왔던 예시들 모두 한번씩 쳐 보고 실행 시켜 볼 것이다.