**HW3**

2014104272 이준영

**1. ADT\_llist.h / ADT\_tree.h**

**ADT\_llist.h 일부**

typedef struct node{

void\* data\_ptr;

struct node\* next;

}NODE;

**ADT\_tree.h 일부**

typedef struct nod**{ //node->nod로 수정**

int data;

struct nod\* left;

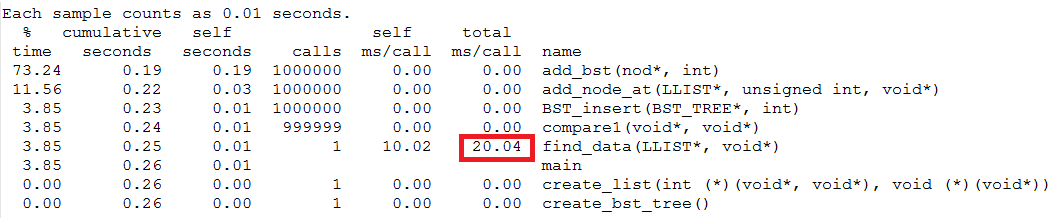
struct nod\* right;

}T\_NODE;

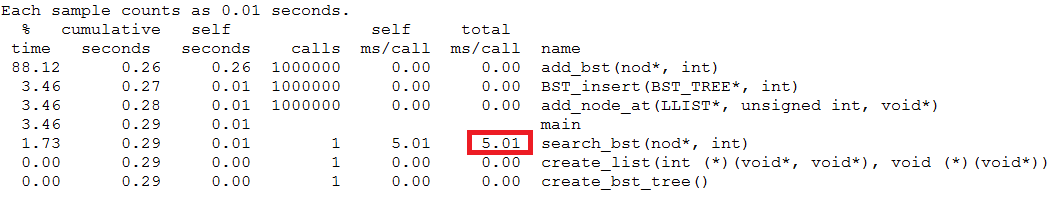
ADT\_llist.h와 ADT\_tree.h의 NODE와 T\_NODE 부분이 typedef struct node로 동일한 이름으로 정의되어 컴파일 오류가 떴습니다. 그래서 ADT\_tree.h의 typedef struct node 부분을 typedef struct nod로 수정하였습니다.

**2. 결과 (LLIST와 BST\_TREE 비교)**

**profile\_llist.txt (sample 1000000)**



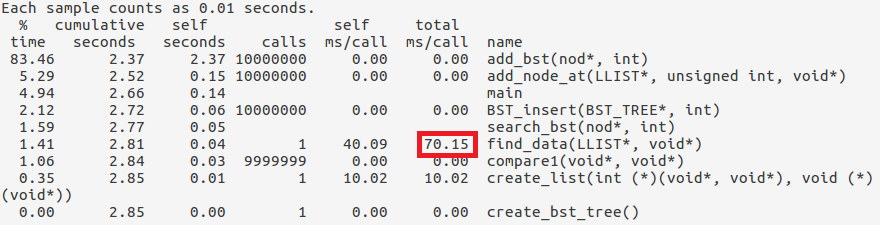
**profile\_tree.txt (sample 1000000)**



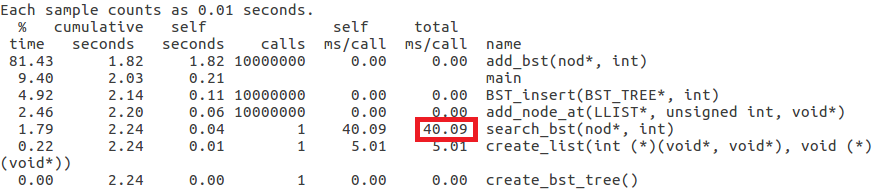
LLIST에서 data를 찾는데 걸리는 시간이 TREE에서 보다 오래 걸리는 것을 확인할 수 있습니다.

해당 결과가 컴파일 할 때 마다 많이 달라져서 반복해서 make run을 실행한 결과 가장 이상적인 결과가 나온 것을 첨부했습니다. (find\_data 나 search\_bst 부분의 total ms/call이 너무 작아 0으로 측정된 결과가 많이 나왔습니다.)

**profile\_llist.txt (sample 10000000)**



**profile\_tree.txt (sample 10000000)**



Sample값을 10000000으로 바꾸어 컴파일 한 경우 LLIST에서 찾는 시간이 BST\_TREE보다 훨씬 많이 걸리는 것을 쉽게 확인할 수 있습니다. 이 경우에는 여러 번 컴파일을 하더라도 항상 LLIST의 경우가 훨씬 더 오랜 시간이 걸리는 것을 확인했습니다.

**<결과에 대한 추가설명>**

LLIST는 front에서부터 쭉 찾고자 하는 data값을 비교하기 때문에 뒷부분에 있는 경우 앞의 모든 노드에 대하여 비교해서 찾아 시간이 오래 걸립니다. 즉, LLIST의 경우 BST\_TREE에서 height가 최대인 경우와 같이 data들이 저장되어 있습니다.

하지만, BST\_TREE의 경우 root에서 left와 right를 비교하여 내려가기 때문에 height가 낮을수록 시간이 data를 찾는데 시간이 단축됩니다. 이 예제의 경우 거의 LLIST의 height보다 훨씬 낮기 때문에 data를 찾는데 비교하는 횟수가 줄어들어 LLIST의 경우 보다 빠르게 data를 찾을 수 있습니다.

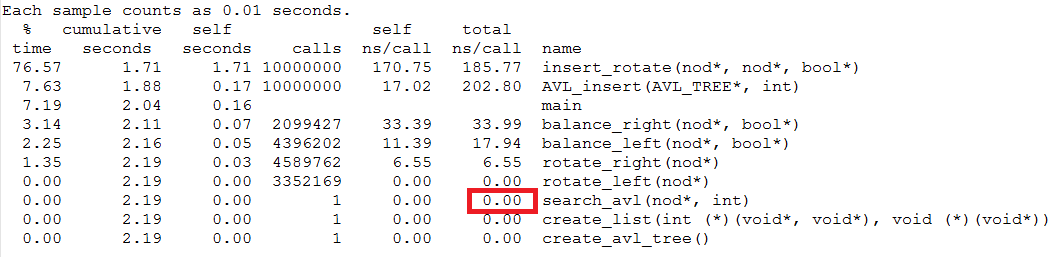
만약, BST\_TREE에서 data가 left와 right에 골고루 분포되어 있지 않고 한쪽으로 일부 치우쳐져 있다면 height가 항상 최소인 AVL\_TREE를 이용하면 시간을 더 단축할 수 있음을 예상할 수 있습니다.

결론: 특정 data를 찾는데 소모되는 시간: LLIST >= BST\_TREE >= AVL\_TREE 방식

**3. 추가 ( +AVL\_TREE)**

AVL\_TREE 코드를 짜서 위와 같은 조건에서 AVL\_TREE에 정보를 삽입하고 특정 data를 찾는데 걸리는 시간을 확인해보았습니다.

**profile\_avl\_tree.txt (sample 10000000)**



위의 경우와 비교하여 data를 찾는데 소모되는 시간이 거의 없음(0ns/call)을 볼 수 있었고 2번에서 작성한 결론을 (data를 찾는데 소모되는 시간: LLIST >= BST\_TREE >= AVL\_TREE) 확인할 수 있었습니다.

**4. 느낀점**

수업시간에 배운 LLIST와 BST\_TREE, AVL\_TREE의 코드를 구현하고 실행한 결과 LLIST의 경우보다 TREE가 효율적인 것을 확인할 수 있었습니다. 자료구조 과제를 통하여 수업시간에 배운 내용을 코드로 짜면서 복습도 하고 계속 생각을 하기 때문에 공부에 많은 도움이 되었습니다.

특히, AVL\_TREE의 경우 balance\_right와 AVL\_delete 부분을 수업시간에 생략했기 때문에 앞서 배운 내용들과 책을 이용해 공부하고 생각해서 코드를 구성하는 과정에 실력이 향상된 느낌이 들었습니다.