Xilinx Zynq FPGA, TI DSP, MCU
기반의 프로그래밍 및 회로 설계
전문가 과정

<자료구조>

2018.03.13-14일차

강사 – 이상훈
gcccompil3r@gmail.com

학생 – 안상재
sangjae2015@naver.com

*** 재귀함수를 사용하지 않는 AVL 트리 구현**

- **알고리즘 순서**

1) avl 트리의 맨 밑바닥까지 가기 위해, 삽입하려는 데이터에 따라 왼쪽 또는 오른쪽 노드로 이동한다. 이 때 거쳐가는 노드들은 모두 스택에 차례로 집어넣는다.

2) 맨 밑바닥까지 도달하면 노드를 생성하고 삽입하려는 데이터를 저장한다.

3) 노드를 생성한 지점부터 시작해서 스택에 저장되어 있는 노드들을 차례로 빼내면서 회전을 할 지 말지 결정한다.

- 핵심 개념

재귀함수와 스택의 개념이 비슷한 것을 이용했다. 재귀함수로 들어가는 것은 스택의 push에 해당하고, 재귀함수에서 반환되는 것은 스택의 pop에 해당한다. 재귀함수를 사용하면 직관적이고 단순하게 프로그래밍을 할 수 있지만, 재귀함수를 호출할 때마다 재귀함수만큼의 크기가 메모리에 계속 할당이 되어서, 만약 자칫 재귀함수의 호출이 무한반복이 된다면 스택 오버플로우가 발생할 수 있다. 또한 실행시간이 지연되어서 전체 프로그램의 성능이 저하되는 이유가 된다.

반면, 재귀함수 대신 스택을 사용하면 프로그램 성능이 좋아지지만, 코드가 더 복잡해지고 어려워진다. 그러므로 생산성이 낮아지는 결과를 초래한다.

- **소스 코드**

- 삽입

```
void avl_ins(avl **root, int data)
{
	int cnt = 0;			// 현재 스택에 저장되어 있는 노드의 개수 표현.
	avl **tmp = root;		// main문의 노드와 포맷을 똑같이 맞춰주기 위해 이중포인터로 선언함.
	stack *top = NULL;		// 처음 스택에 들어갈 노드의 주소를 0으로 함.
	while(*tmp)
	{
		printf("Save Stack: %d, data = %d₩n", ++cnt, data); // 스택에 노드가 쌓일 때마다 현재 몇 개의
                                                                    노드가 들어있는지 표시함.
		push(&top, tmp);		// 노드를 거쳐갈 때마다 스택에 해당 노드를 집어넣음.
		if((*tmp)->data > data)
			tmp = &(*tmp)->left;
		else if((*tmp)->data < data)
			tmp = &(*tmp)->right;
	}

	*tmp = get_avl_node();	// 데이터를 삽입하려는 지점에서 노드를 생성함.
	(*tmp)->data = data;	// 데이터를 해당노드에 저장함.

	while(stack_is_not_empty(top))	// 스택이 빈 상태가 될 때까지 반복함.
	{
```

```
                printf("Extract Stack: %d, data = %d₩n", --cnt, data);  // 스택에서 노드가 빠져나갈 때마다 현재
                                                                        몇 개의 노드가 들어있는지 표시함.
                avl **t = (avl **)pop(&top);          // 스택에서 노드를 빼내옴.
                (*t)->lev = update_level(*t);         // 해당 노드의 level을 갱신함.
                if(abs(rotation_check(*t)) > 1)       // 해당노드의 위치에서 회전을 해야 할 지 검사함.
                {
                        printf("Insert Rotation₩n");
                        *t = rotation(*t, kinds_of_rot(*t, data));      // 해당 지점에 알맞은 회전을 선택하고, 회
전을 수행함.
                }
        }
}
```

- 스택 구조체

```
typedef struct __stack
{
        void *data;       // void *data는 void의 주소를 저장할 수 있는 변수이다. avl 구조체의 주소를 저장하기
                          위함이다. 또한 void * 형으로 변수를 선언하면 어떠한 자료형의 주소도 저장할 수 있
                          다.
        struct __stack *link;
} stack;
```

- 스택 push

```
void push(stack **top, void *data)  // void *data는 avl 트리의 노드를 저장하기 위한 포인터 변수(avl *data로 해도
                                    무관)
{
        if(data == NULL)
                return;
        stack *tmp = *top;
        *top = get_stack_node();
        (*top)->data = malloc(sizeof(void *));        // 새로 push된 노드의 data 멤버에 메모리 공간을 할당함.
        (*top)->data = data;  // 새로 push된 노드의 data 멤버에 push하려는 avl 트리의 노드의 주소를 저장함.
        (*top)->link = tmp;         // 새로 push된 노드와 그 전 노드를 연결함.
}
```

- 스택 pop

```
void *pop(stack **top)
{
        stack *tmp = *top;
        void *data = NULL;
        if(*top == NULL)            // 스택에 어떤 노드도 없으면 그냥 반환한다.
        {
                printf("stack is empty!₩n");
                return NULL;
```

```
    }
    data = (*top)->data;        // 스택의 pop하려는 노드의 멤버 중 avl 트리 노드의 주소를 void형 포인터
                                                    변수 data에 저장함.
    *top = (*top)->link;        // *top의 위치를 조정함.
    free(tmp);                  // pop하려는 노드의 메모리 공간을 해제함.
    return data;                // pop하려는 avl 트리 노드를 반환함.
}
```

- **알고리즘 그림**