TI DSP, MCU 및 Xilinx Zynq FPGA

프로그래밍 실무 교재

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 최대성

c3d4s19@naver.com

2018.03.09. – 012일차 수업

---

* Red-Black 트리 조건

1. 노드의 색상은 Red or Black
2. 루트노드의 색상은 Black
3. 모든 NIL노드는 Black
4. Red노드의 자식 노드 색상은 Black이고 Black노드의 자식 노드 색상은 아무거나 가능
5. 어떤 노드로부터 NIL노드까지 도달하는 모든 경로에는 NIL노드를 제외하면 모두 같은 수의 Black노드가 존재

---

* Red-Black 트리 데이터 삽입 함수(재귀 이용)

헤더파일 추가 및 구조체 정의

```
#include <stdio.h>
#include <stdlib.h>

typedef int Data;
typedef enum _color {
	red = 0,
	black = 1
}Color;

typedef struct _tree {
	Data data;
	Color color;
	struct _tree* Right;
	struct _tree* Left;
}Tree;

typedef struct _rootTree {
	Tree** root;
	int dataNum;
}RootTree;
```

함수 선언

```
Tree* getNewNIL();
int isNIL(Tree* root);
void printTree(Tree** Tree);
void RBT_insertDataChangeColor(Tree** root, Data
data);
void RBT_insertData(Tree** root, Data data);
void RBT_chgRootColor(Tree** root);
Tree* RRrot(Tree** tree);
Tree* LLrot(Tree** tree);
Tree* RLrot(Tree** tree);
Tree* LRrot(Tree** tree);
Tree* RotTree(Tree** root);
```

새로운 NIL노드 얻는 함수

```
Tree* getNewNIL() {
	Tree *NIL = (Tree *)malloc(sizeof(Tree));
	NIL->color = black;
	NIL->data = 0;
	NIL->Right = NULL;
	NIL->Left = NULL;
	return NIL;

}
```

노드가 NIL노드인지 확인하여 반환하는 함수

```
int isNIL(Tree* root) {
	if (root == NULL)
		return 0;
	if (root->Left == NULL && root->Right ==
NULL)
		return 1; //True
	else
		return 0; //False
}
```

Red-Black 트리 print 함수

```
void printTree(Tree** Tree) {
	//노드가 NIL이 아니면
	if (!isNIL(*Tree)) {
		printf("%d ₩n", (*Tree)->data);
		if (!isNIL((*Tree)->Left))
		{
			printTree(&((*Tree)->Left));
		}
		if (!isNIL((*Tree)->Right))
		{
			printTree(&((*Tree)->Right));
		}
	}
}
```

Red-Black 최종 삽입 함수

```
void RBT_insertDataChangeColor(Tree** root, Data
data) {
	//루트 노드가 NULL인 경우
	if (*root == NULL) {
		*root = getNewNIL();
	}
	//root가 NIL인 경우
	if (isNIL(*root)) {
		(*root)->Right = getNewNIL();
		(*root)->Left = getNewNIL();
		(*root)->color = black;
		(*root)->data = data;
	}
	//root가 NIL이 아닌 경우
	else {
		RBT_insertData(root, data);
		RBT_chgRootColor(root);
	}
}
```

Red-Black삽입함수(루트노드 색상변경 문제로 불완전함)

```
void RBT_insertData(Tree** root, Data data) {

	//루트노드가 NIL인 경우 -> 재귀 탈출
	if (isNIL(*root)) {
		(*root)->data = data;
		(*root)->color = red;
		(*root)->Right = getNewNIL();
		(*root)->Left = getNewNIL();
	}

	//새로운 data값이 노드 data 값보다 작으면
	else if (data <= (*root)->data) {
		RBT_insertData(&((*root)->Left),
data);
	}
	//새로운 data값이 노드 data 값보다 크면
	else {
		RBT_insertData(&((*root)->Right),
data);
	}

	//재귀함수 뒤로 돌아가면서 실행
	*root = RotTree(root);
}
```

Red-Black 트리의 루트노드 색상 변경 함수

```
void RBT_chgRootColor(Tree** root) {
	(*root)->color = black;
}
```

RR, LL, RL, LR 회전 함수

```
Tree* RRrot(Tree** tree) {
	Tree* tmpTree = (*tree)->Right;
	(*tree)->Right = tmpTree->Left;
	tmpTree->Left = *tree;
	return tmpTree;
}
Tree* LLrot(Tree** tree) {
	Tree* tmpTree = (*tree)->Left;
	(*tree)->Left = tmpTree->Right;
	tmpTree->Right = (*tree);
	return tmpTree;
}
Tree* RLrot(Tree** tree) {
	Tree* tmpTree = (*tree)->Right;
	tmpTree = LLrot(&tmpTree);
	(*tree)->Right = tmpTree;
	tmpTree = RRrot(tree);
	return tmpTree;
}
Tree* LRrot(Tree** tree) {
	Tree* tmpTree = (*tree)->Left;
	tmpTree = RRrot(&tmpTree);
	(*tree)->Left = tmpTree;
	tmpTree = LLrot(tree);
	return tmpTree;
}
```

선택 노드 기준으로 자식노드와 손자노드 색상 비교 후 경우에 따라 회전 및 색상 변경하는 함수

```
Tree* RotTree(Tree** root) {

	Tree* tmp = *root;
	Tree* tmp2;

	//root가 NULL인 경우
	if (*root == NULL) {
		return NULL;
	}
	//root가 NIL인 경우
	if (isNIL(*root)) {
		return *root;
	}
	//노드 색상이 Red인 경우
	if ((*root)->color == red) {
		return *root;
	}

	//양쪽 자식 노드 색상이 Red인 경우 ->
루트노드 색상을 Red, 양쪽 자식 노드 색상을 Black으로
바꾸고 함수 break;
	if ((*root)->Right->color == red && (*root)-
>Left->color == red) {
		(*root)->color = red;
		(*root)->Right->color = black;
		(*root)->Left->color = black;
		return *root;
	}
```

```
        // RR RL  필요한 경우
        if (tmp->Right->color == red) {
                tmp2 = tmp->Right;
                //RR 회전 + 색상 변경
                if (tmp2->Right->color == red) {
                        *root = RRrot(root);
                        (*root)->color = black;
                        (*root)->Left->color = red;
                }
                //RL 회전
                else if (tmp2->Left->color == red) {
                        *root = RLrot(root);
                        (*root)->color = black;
                        (*root)->Left->color = red;
                }
                //둘다 B-R-R 해당 안되면 그냥 나감
(B-R-B인 경우)
        }
        // LR LL  필요한 경우
        else if (tmp->Left->color == red) {
                tmp2 = tmp->Left;
                //LL 회전
                if (tmp2->Left->color == red) {
                        *root = LLrot(root);
                        (*root)->color = black;
                        (*root)->Right->color = red;
                }
                //LR 회전
                else if (tmp2->Right->color == red) {
                        *root = LRrot(root);
                        (*root)->color = black;
                        (*root)->Right->color = red;
                }
                //둘다 B-R-R 해당 안되면 그냥 나감
(B-R-B인 경우)
        }
        //B-B-?인 경우 그냥 나감
        return *root;
}
```

테스트용 Main 함수

```
int main() {
        Tree* tree = getNewNIL();

        printf("₩n---------------₩n");
        RBT_insertDataChangeColor(&tree, 6);
        RBT_insertDataChangeColor(&tree, 7);
        RBT_insertDataChangeColor(&tree, 8);
        RBT_insertDataChangeColor(&tree, 9);
        RBT_insertDataChangeColor(&tree, 10);

        printTree(&tree);

        printf("₩n--------------₩n");
        RBT_insertDataChangeColor(&tree, 1);
        RBT_insertDataChangeColor(&tree, 2);
        RBT_insertDataChangeColor(&tree, 3);
        RBT_insertDataChangeColor(&tree, 4);
        RBT_insertDataChangeColor(&tree, 5);

        printTree(&tree);
        return 0;
}
```