TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

11회차 (2018-03-08)

강사 - Innova Lee(이상훈)
gcccompil3r@gmail.com

학생 - 정유경
ucong@naver.com

*. AVL 그림분석 좀 더 노력하겠습니다. 죄송합니다!
자료조사와 코드분석 제출합니다.

AVL 트리 (균형 이진 트리, (balanced binary tree)

1. 이진 탐색 트리의 단점을 해결한, AVL 트리

이진 트리에서 모든 노드의 왼쪽과 오른쪽 트리의 높이차를 1이하로 만든 트리이다.
T가 이진 트리이고 TL 과 TR 을 왼쪽과 오른쪽 부속 트리 라고 하면, T를 높이균형(height balanced) 트리라고 하며 다음을 만족한다
TL과 TR 이 높이균형(height balanced) 트리일때, |hL – hR| ≤ 1
(hL과 hR이 TL과 TR의 높이)

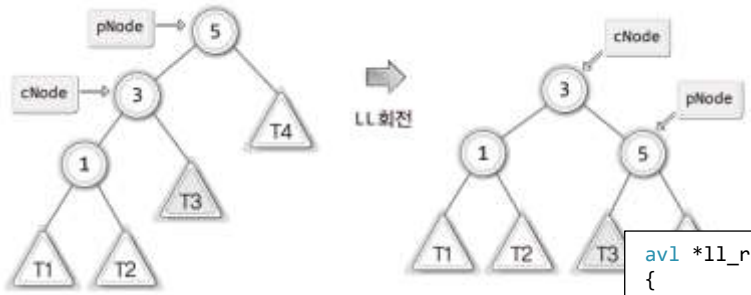
2. 균형인자(balance factor)

BF(T) = hL – hR (왼쪽 서브트리의 높이 – 오른쪽 서브트리의 높이)
AVL 트리의 모든 노드에 대하여 BF(T) = -1, 0, 혹은 1이다.

3. AVL트리 회전(리밸런싱)방법

비어있는 트리로 부터 혹은 이미 구성된 AVL 트리로 부터 노드를 삽입하거나 삭제할 경우,트리의 높이 균형이 깨지면서 BF 값이 +2 혹은 -2가 될 수 있다.
균형이 깨진 트리의 모양에 따라 다음과 같이 구분하여 균형을 다시 맞춘다. 균형을 맞추려면 다음과 같이 회전을 하여야 한다.
회전방법에는 LL, LR, RR, RL의 네 가지 방법이 있으며 LL과 RR은 한번만 회전이 필요한 단순회전이고 LR과 RL은 두 번의 회전이 필요한 이중회전이다.

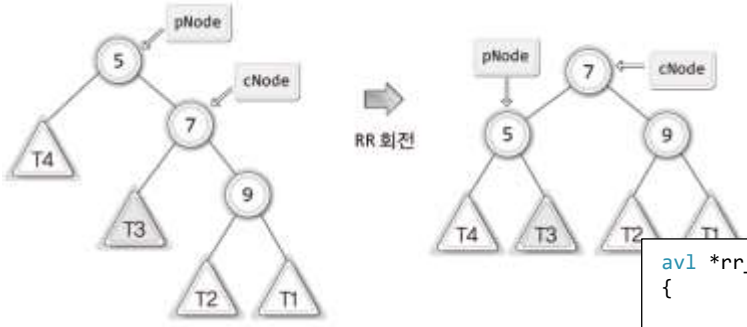
(1) LL상태를 균형잡는, LL회전



PN이 CN의 오른쪽 자식노드가 된다
CN의 오른쪽 서브트리를 PN의 왼쪽 서브트리로 옮긴다.

```
avl *ll_rot(avl *parent, avl *child)
{
        parent->left = child->right;
        child->right = parent;
        parent->lev = update_level(parent);
        child->lev = update_level(child);
        return child;
}
```

(2) RR상태를 균형잡는, RR회전




```
avl *rr_rot(avl *parent, avl *child)
{
        parent->right = child->left;
        child->left = parent;
        parent->lev = update_level(parent);
        child->lev = update_level(child);
        return child;
}
```

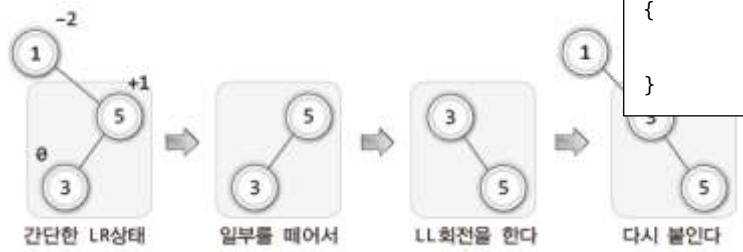
PN이 CN 의 왼쪽 자식노드가 된다
CN의 왼쪽 서브트리를 PN의 오른쪽 서브트리로 옮긴다.

(3) LR회전


```
avl *lr_rot(avl *parent, avl *child)
{
        child = rr_rot(child, child->right);
        return ll_rot(parent, child);
}
```

부분적 RR 회전에 이어서 LL회전을 한다.

(4) RL회전




```
avl *rl_rot(avl *parent, avl *child)
{
        child = ll_rot(child, child->left);
        return rr_rot(parent, child);
}
```

부분적 LL회전에 이어서 RR 회전을 진행한다.

응용 분야

이진 탐색 트리와 사용방식이나 구조가 같기 때문에 이진 탐색 트리를 사용할 수 있는 분야에선 모두 사용 가능하다. 트리의 주된 목적은 탐색이며 의사 결정, 파일 시스템(디렉터리 구조), 검색 엔진, DBMS(데이터베이스), 라우터 알고리즘, 계층적 데이터를 다루는 등 매우 다양한 곳에서 응용된다.

AVL TREE.c

```c
#include <math.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

typedef enum __rot{ // 열거형, 상수보다 의미파악이 쉽다
        RR,
        RL,
        LL,
        LR
} rot;

typedef struct __avl_tree{
        int lev;// 노드의 레벨
        int data;
        struct __avl_tree *left;
        struct __avl_tree *right;
} avl;

bool is_dup(int *arr, int cur_idx) // ?
{
        int i, tmp = arr[cur_idx];
        for (i = 0; i < cur_idx; i++)
                if (tmp == arr[i])
                        return true;
        return false;
}

// 1부터 100까지의 숫자를 이용 랜덤한 배열 arr[size]을 생성한다
void init_rand_arr(int *arr, int size)
{
        int i;
        for (i = 0; i < size; i++)
        {
        redo:
                arr[i] = rand() % 100 + 1;
                if (is_dup(arr, i))
                {
                        printf("%d dup! redo rand()\n", arr[i]);
                        goto redo;
                }
        }
}

void print_arr(int *arr, int size) //  배열 출력
{
        int i;
        for (i = 0; i < size; i++)
                printf("arr[%d] = %d\n", i, arr[i]);
}

avl *get_avl_node(void){ // avl노드 생성
        avl *tmp;
        tmp = (avl *)malloc(sizeof(avl));
        tmp->lev = 1; // 노드의 레벨은 1로 초기화한다.
        tmp->left = NULL;
        tmp->right = NULL;
        return tmp;
```

```
}

void print_tree(avl *root) // 트리출력
{
	if (root)
	{
		printf("data = %d, lev = %d, ", root->data, root->lev);
		if (root->left)
			printf("left = %d, ", root->left->data);
		else
			printf("left = NULL, ");
		if (root->right)
			printf("right = %d\n", root->right->data);
		else
			printf("right = NULL\n");

		print_tree(root->left);
		print_tree(root->right);
	}
}


int update_level(avl *root)
{ // 자식노드가 있으면 그 레벨을 가져오고 : 없으면 0
	int left = root->left ? root->left->lev : 0;
	int right = root->right ? root->right->lev : 0;
	if (left > right) // 왼쪽자식노드의 레벨이 크면
		return left + 1;
	return right + 1; // 오른쪽 자식노드의 레벨이 크면
}

int rotation_check(avl *root)
{
	int left = root->left ? root->left->lev : 0;
	int right = root->right ? root->right->lev : 0;
	return right - left;
// 오른쪽 노드와 왼쪽 노드의 차이를 리턴하여 로테이션 여부 결정
}
/*리밸런싱 방법 결정*/
int kinds_of_rot(avl *root, int data)
{
	printf("data = %d\n", data);
	// for RR and RL
	if (rotation_check(root) > 1) // R-L > 1 이면  R
	{// 루트의 오른쪽보다 작으면
		if (root->right->data > data)
			return RL; // RL상태
		return RR; // 오른쪽보다 크면, RR 상태
	}
	// for LL and LR
	else if (rotation_check(root) < -1) // R-L < -1이면 L
	{// 루트의 왼쪽보다 크면
		if (root->left->data < data)
			return LR; // LR상태
		return LL; // 왼쪽보다 작으면 LL상태
	}
}
```

```
/*PN이 CN 의 왼쪽 자식노드가 된다
CN의 왼쪽 서브트리를 PN의 오른쪽 서브트리로 옮긴다. */
avl *rr_rot(avl *parent, avl *child)
{
        parent->right = child->left;
        child->left = parent;
        parent->lev = update_level(parent);
        child->lev = update_level(child);
        return child;
}

/*PN이 CN의 오른쪽 자식노드가 된다
CN의 오른쪽 서브트리를 PN의 왼쪽 서브트리로 옮긴다.*/
avl *ll_rot(avl *parent, avl *child)
{
        parent->left = child->right;
        child->right = parent;
        parent->lev = update_level(parent);
        child->lev = update_level(child);
        return child;
}
/*부분적 LL회전에 이어서 RR 회전을 진행한다. */
avl *rl_rot(avl *parent, avl *child)
{
        child = ll_rot(child, child->left);
        return rr_rot(parent, child);
}
/*부분적 RR 회전에 이어서 LL회전을 한다.*/
avl *lr_rot(avl *parent, avl *child)
{
        child = rr_rot(child, child->right);
        return ll_rot(parent, child);
}

avl *rotation(avl *root, int ret)
{
        switch (ret) // rot에 따라 로테이션 함수 선택하여 진행
        {
        case RL:
                printf("RL Rotation\n");
                return rl_rot(root, root->right);
        case RR:
                printf("RR Rotation\n");
                return rr_rot(root, root->right);
        case LR:
                printf("LR Rotation\n");
                return lr_rot(root, root->left);
        case LL:
                printf("LL Rotation\n");
                return ll_rot(root, root->left);
        }
}

void avl_ins(avl **root, int data)
{
        if (!(*root))
        {
                (*root) = get_avl_node();
                (*root)->data = data;
                return;
        }
```

```
        if ((*root)->data > data)
                avl_ins(&(*root)->left, data);
        else if ((*root)->data < data)
                avl_ins(&(*root)->right, data);

        //update_level(root);
        (*root)->lev = update_level(*root);
        if (abs(rotation_check(*root)) > 1)
        {
                printf("Insert Rotation!\n");
                *root = rotation(*root, kinds_of_rot(*root, data));
        }
}

avl *chg_node(avl *root)
{
        avl *tmp = root;
        if (!root->right)
                root = root->left;
        else if (!root->left)
                root = root->right;
        free(tmp);
        return root;
}

avl *find_max(avl *root, int *data)
{
        if (root->right)
                root->right = find_max(root->right, data);
        else
        {
                *data = root->data;
                root = chg_node(root);
        }
        return root;
}


void avl_del(avl **root, int data)
{
        if (*root == NULL)
        {
                printf("There are no data that you find %d\n", data);
                return;
        }

        else if ((*root)->data > data)
                avl_del(&(*root)->left, data);

        else if ((*root)->data < data)
                avl_del(&(*root)->right, data);

        else if ((*root)->left && (*root)->right)
                (*root)->left = find_max((*root)->left, &(*root)->data);

        else
        {
                *root = chg_node(*root);
                return;
        }

        (*root)->lev = update_level(*root);

        if (abs(rotation_check(*root)) > 1)
```

```
        {
                printf("Delete Rotation!\n");
                *root = rotation(*root, kinds_of_rot(*root, data));
        }
}

int main(void)
{
        int i;
        avl *root = NULL;
        int arr[16] = { 0 };
        int size = sizeof(arr) / sizeof(int) - 1;

        srand(time(NULL));
        init_rand_arr(arr, size);
        print_arr(arr, size);

        for (i = 0; i < size; i++)
                avl_ins(&root, arr[i]);

        print_tree(root);
        printf("\nAfter Delete\n");
        avl_del(&root, arr[3]);
        avl_del(&root, arr[6]);
        avl_del(&root, arr[9]);

        print_tree(root);
        return 0;
}
```