컴퓨터공학 All in One

C/C++ 문법, 자료구조 및 심화 프로젝트 (나동빈)
제 37강 - AVL 트리

학습 목표

AVL 트리

1) AVL 트리는 균형이 갖춰진 이진 트리(Binary Tree)를 의미합니다.
2) 완전 이진 트리는 검색에 있어서 $O(logN)$의 시간 복잡도를 유지할 수 있습니다.
3) AVL 트리는 간단한 구현 과정으로 특정 이진 트리가 완전 이진 트리에 가까운 형태를 유지하도록 해줍니다.

AVL 트리

AVL 트리

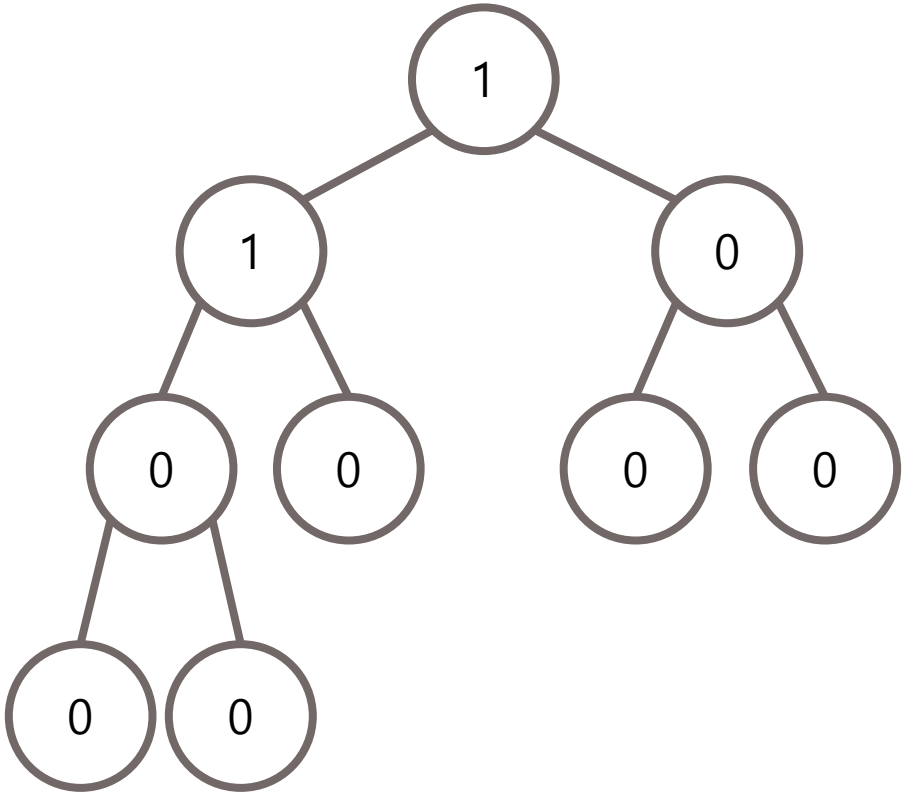
AVL 트리는 균형 인수(Balance Factor)라는 개념을 이용합니다.

균형 인수 = **| 왼쪽 자식 높이 - 오른쪽 자식 높이 |**

Fast campus

AVL 트리

AVL 트리



AVL 트리

AVL 트리

AVL 트리는 모든 노드에 대한 균형 인수가 +1, 0, -1인 트리를 의미합니다.

균형 인수가 위 세 가지 경우에 해당하지 않는 경우 '회전(Rotation)'을 통해 트리를 재구성해야 합니다.

AVL 트리

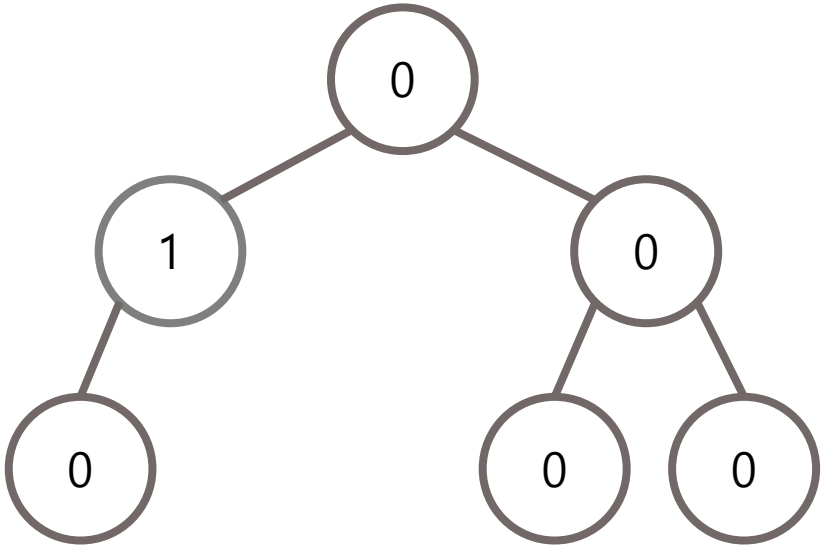
AVL 트리의 회전

AVL 트리는 총 4가지 형식에 의하여 균형이 깨질 수 있습니다. 균형 인수가 깨지는 노드를 X라고 했을 때 4가지 형식은 다음과 같습니다.

1) LL 형식: X의 왼쪽 자식의 왼쪽에 삽입하는 경우
2) LR 형식: X의 왼쪽 자식의 오른쪽에 삽입하는 경우
3) RR 형식: X의 오른쪽 자식의 오른쪽에 삽입하는 경우
4) RL 형식: X의 오른쪽 자식의 왼쪽에 삽입하는 경우

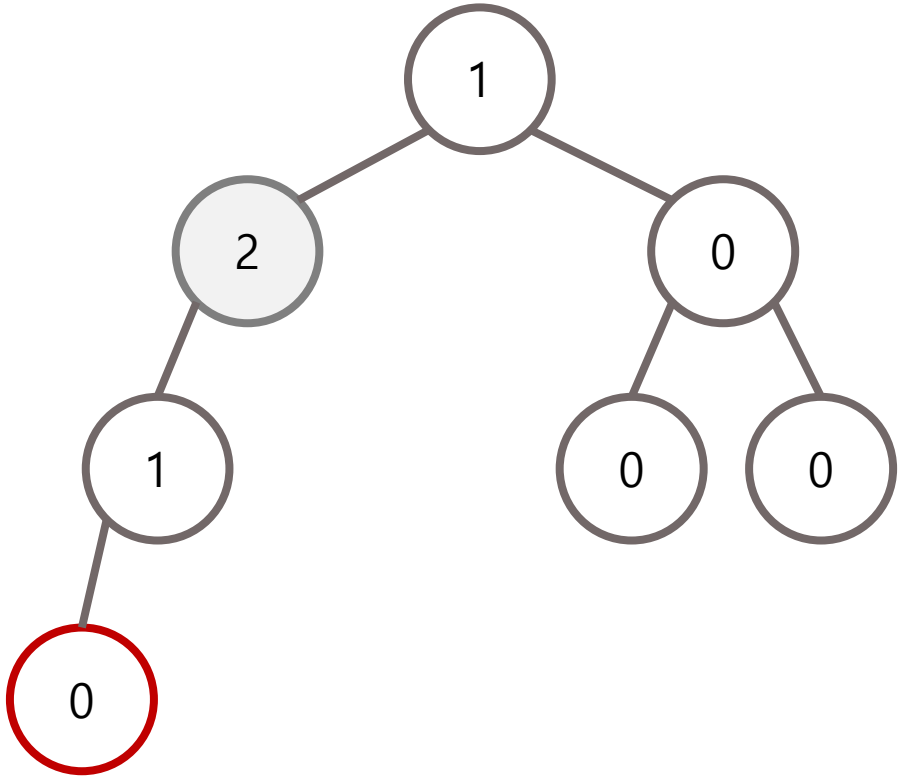
AVL 트리

AVL 트리의 LL 회전 조건



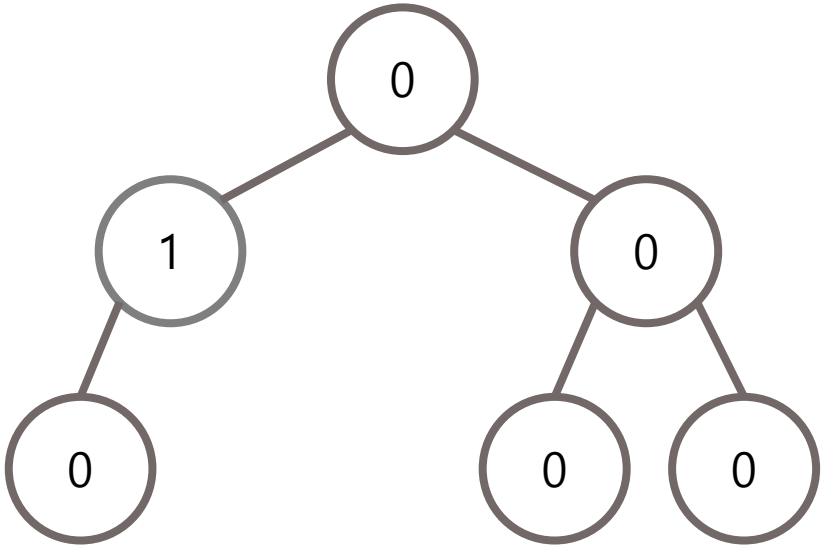
AVL 트리

AVL 트리의 LL 회전 조건



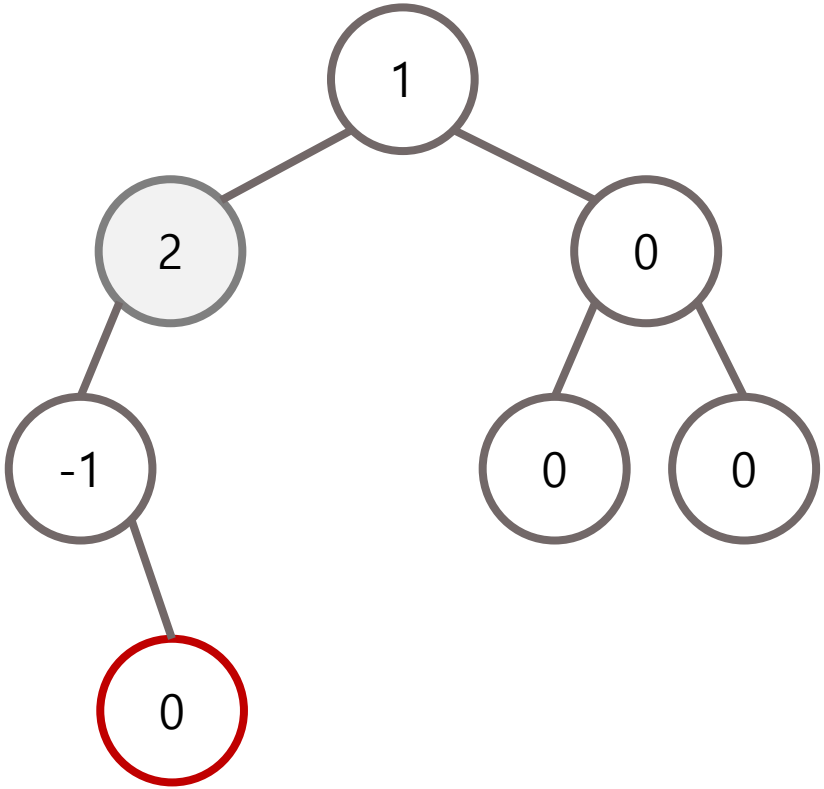
AVL 트리

AVL 트리의 LR 회전 조건



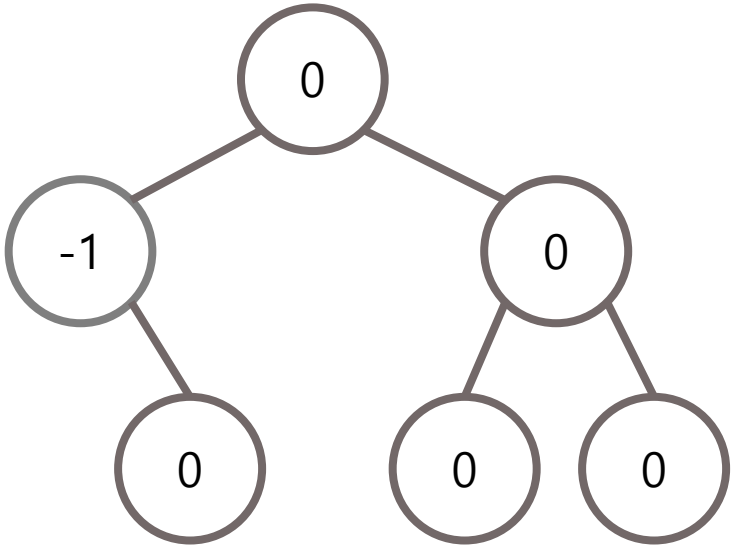
AVL 트리

AVL 트리의 LR 회전 조건



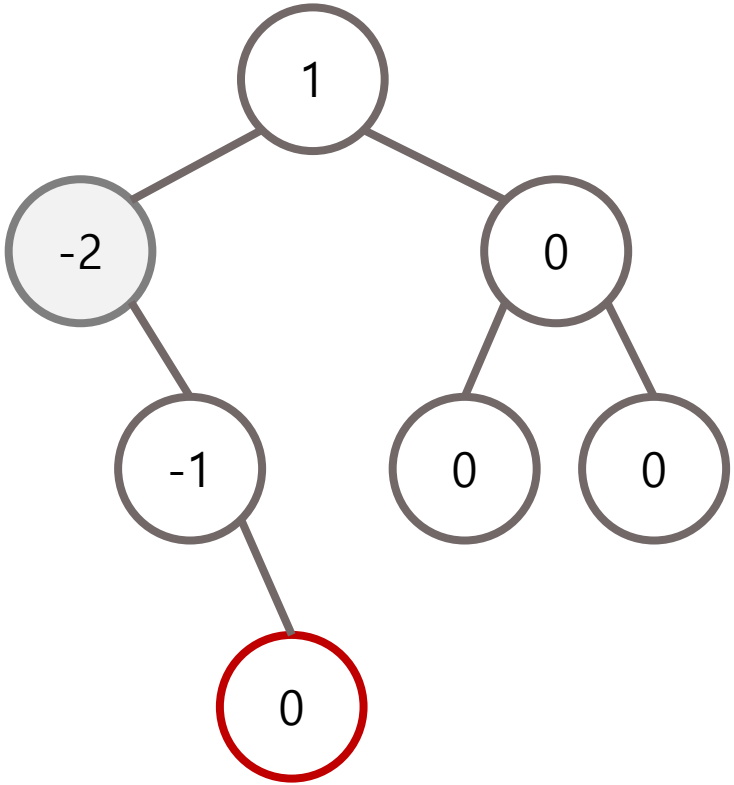
AVL 트리

AVL 트리의 RR 회전 조건



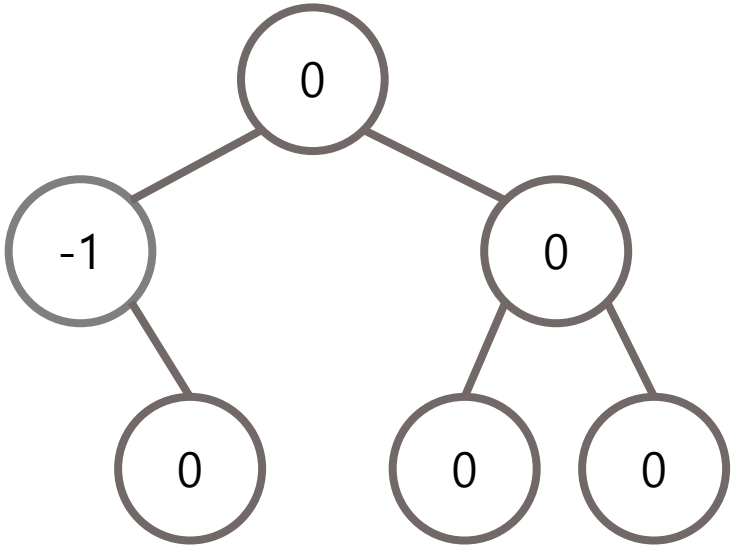
AVL 트리

AVL 트리의 RR 회전 조건



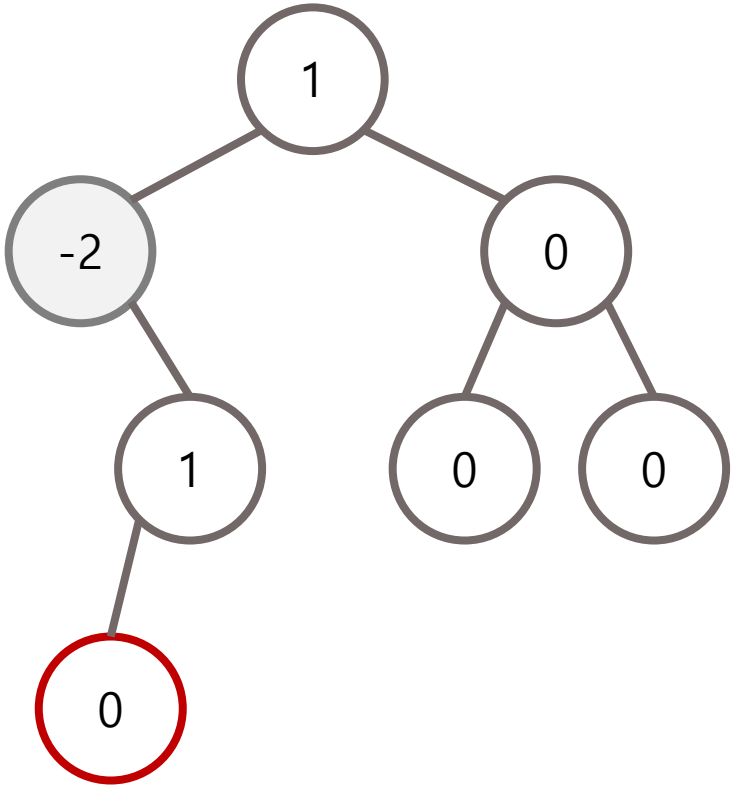
AVL 트리

AVL 트리의 RL 회전 조건



AVL 트리

AVL 트리의 RL 회전 조건



AVL 트리

AVL 트리의 높이

AVL 트리의 각 노드는 '균형 인수'를 계산하기 위한 목적으로 자신의 '높이(Height)' 값을 가집니다.

AVL 트리

AVL 트리의 정의

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

int getMax(int a, int b) {
  if (a > b) return a;
  return b;
}

typedef struct {
  int data;
  int height; // 높이를 저장해야 시간 복잡도를 보장할 수 있음.
  struct Node* leftChild;
  struct Node* rightChild;
} Node;
```

AVL 트리

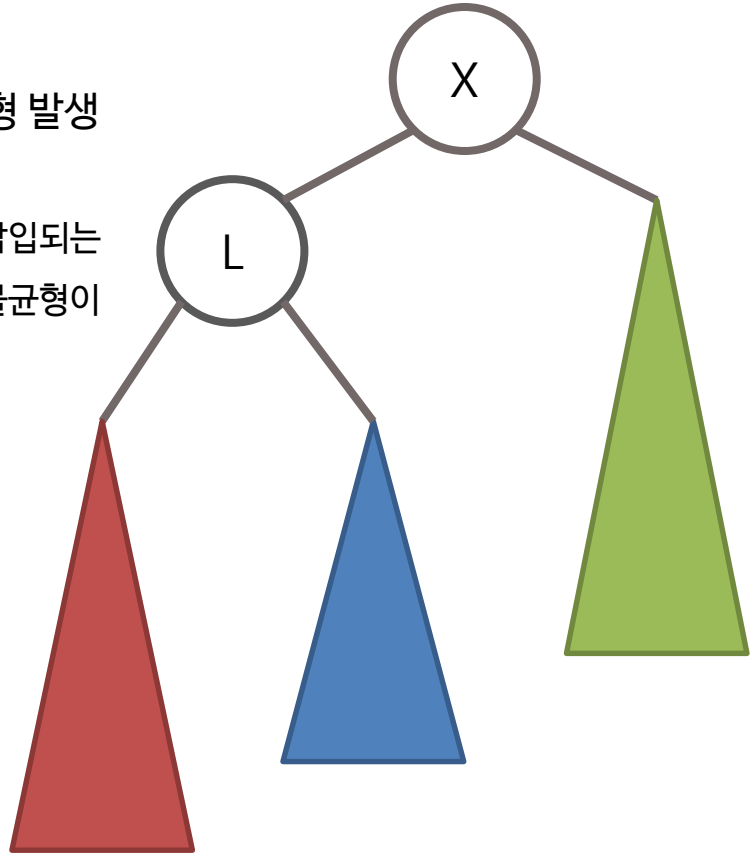
AVL 트리의 높이 계산 함수

```
int getHeight(Node* node) {
  if (node == NULL) return 0;
  return node->height;
}

// 모든 노드는 회전을 수행한 이후에 높이를 다시 계산
void setHeight(Node* node) {
  node->height = getMax(getHeight(node->leftChild), getHeight(node->rightChild)) + 1;
}

int getDifference(Node* node) {
  if (node == NULL) return 0;
  Node* leftChild = node->leftChild;
  Node* rightChild = node->rightChild;
  return getHeight(leftChild) - getHeight(rightChild);
}
```

AVL 트리

AVL 트리의 LL 회전 1) 불균형 발생

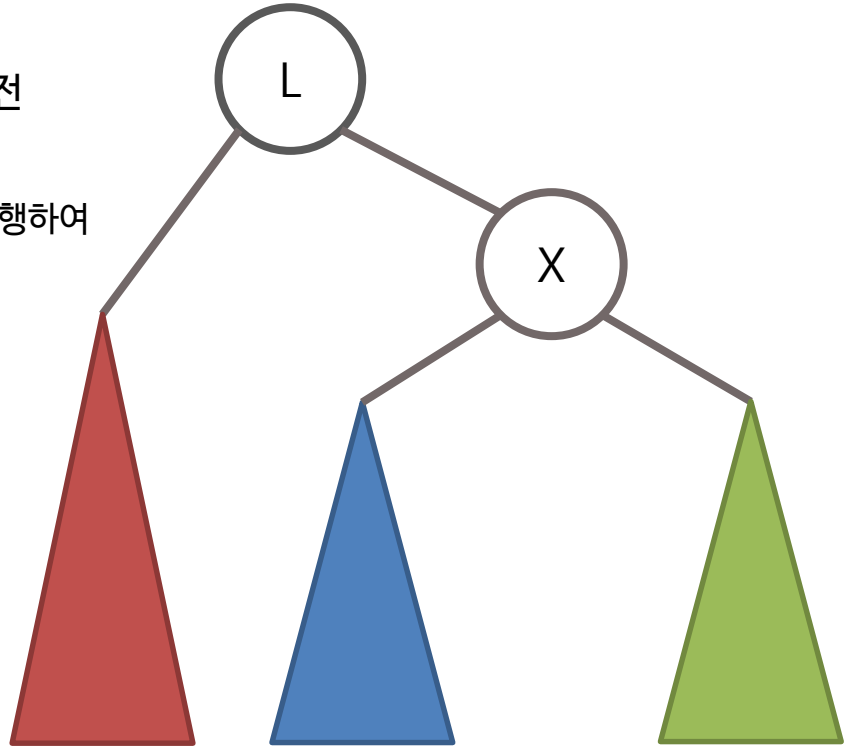
빨간색 영역에 새로운 노드가 삽입되는 경우 노드 X를 기준으로 했을 때 불균형이 발생했다는 것을 알 수 있습니다.



AVL 트리

AVL 트리의 LL 회전 2) LL 회전

노드 X를 기준으로 LL 회전을 수행하여 트리의 불균형을 해소합니다.



AVL 트리

AVL 트리의 LL 회전

```
Node* rotateLL(Node* node) {
  Node *leftChild = node->leftChild;
  node->leftChild = leftChild->rightChild;
  leftChild->rightChild = node;
  setHeight(node); // 회전 이후에 높이를 다시 계산
  return leftChild;
}
```

AVL 트리

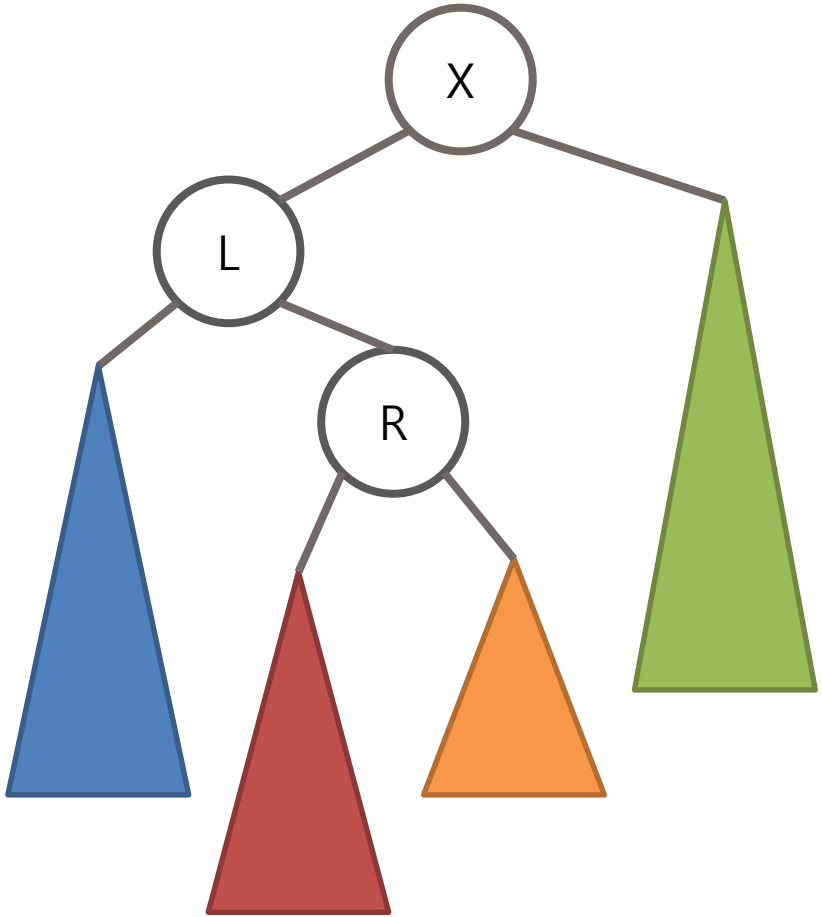
AVL 트리의 RR 회전 (LL 회전을 반대로 수행합니다.)

```
Node* rotateRR(Node* node) {
  Node *rightChild = node->rightChild;
  node->rightChild = rightChild->leftChild;
  rightChild->leftChild = node;
  setHeight(node); // 회전 이후에 높이를 다시 계산
  return rightChild;
}
```

AVL 트리

AVL 트리의 LR 회전 1) 불균형 발생

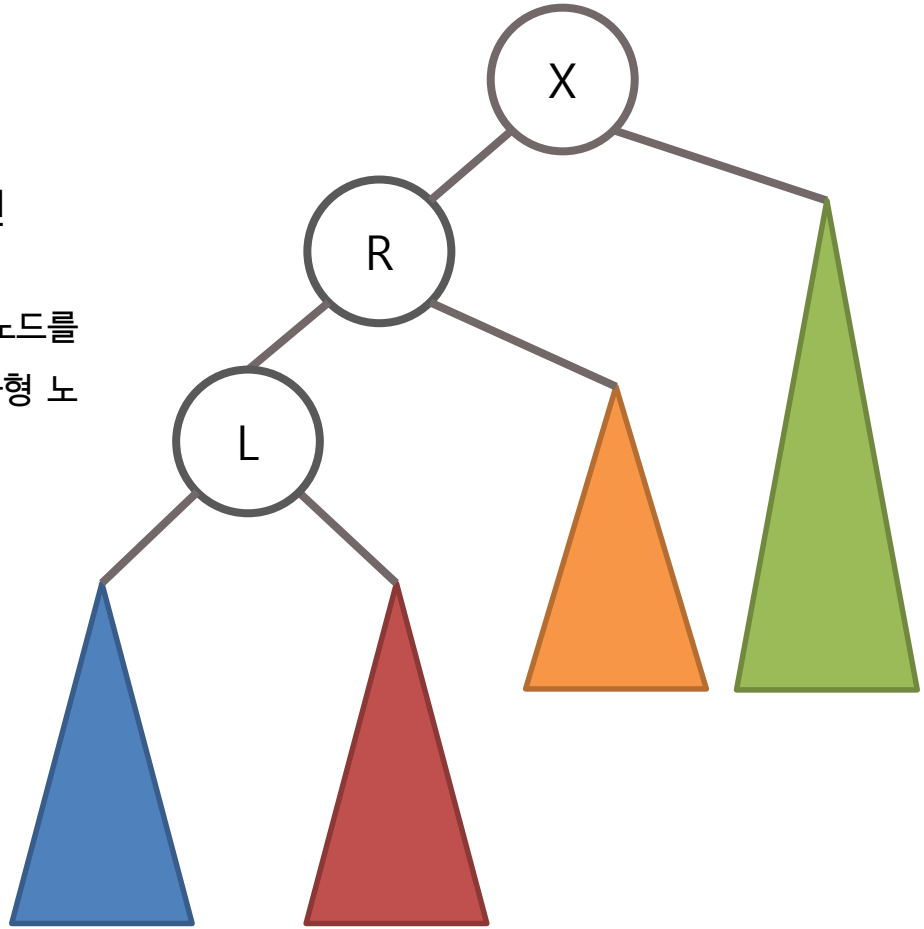
빨간색 영역에 새로운 노드가 삽입되는 경우 노드 X를 기준으로 했을 때 불균형이 발생했다는 것을 알 수 있습니다.



AVL 트리

AVL 트리의 LR 회전 2) RR 회전

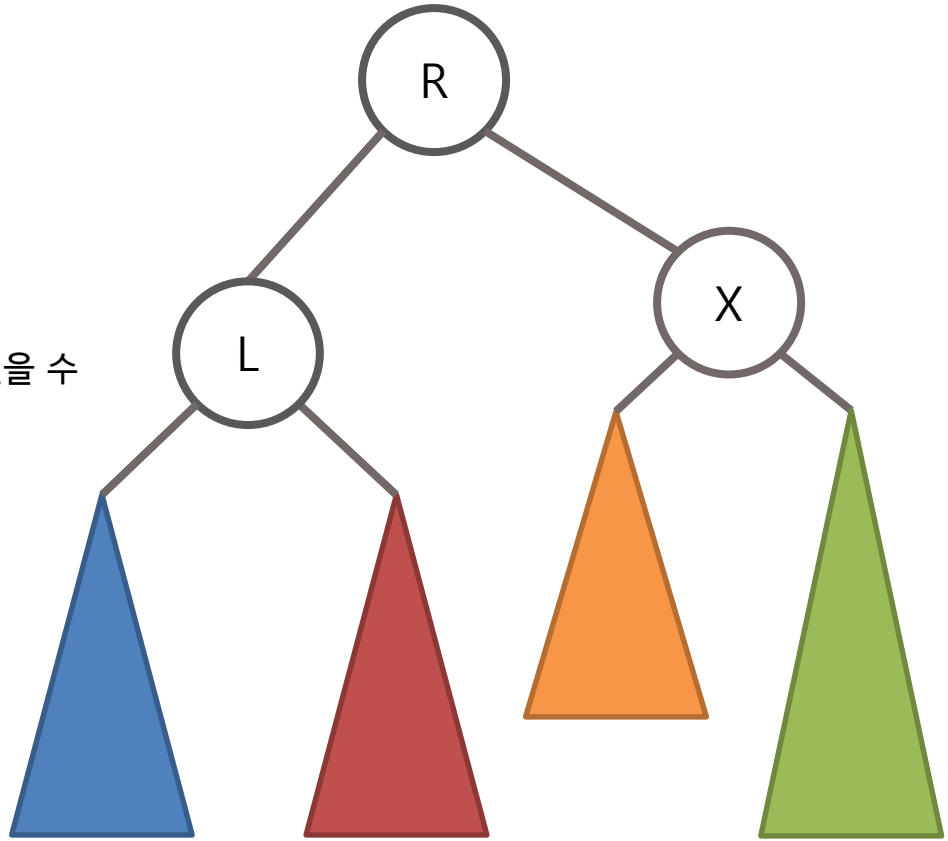
가장 먼저 자신의 왼쪽 노드인 L 노드를 기준으로 RR 회전을 수행하여, 불균형 노드를 왼쪽으로 몰아 넣습니다.



AVL 트리

AVL 트리의 LR 회전 3) LL 회전

이후에 노드 X를 기준으로 LL 회전을 수행하면 불균형이 해소됩니다.



AVL 트리

AVL 트리의 LR 회전

```
Node *rotateLR(Node* node) {
  Node *leftChild = node->leftChild;
  node->leftChild = rotateRR(leftChild);
  return rotateLL(node);
}
```

AVL 트리

AVL 트리의 RL 회전 (LR 회전을 반대로 수행합니다.)

```
Node *rotateRL(Node* node) {
  Node *rightChild = node->rightChild;
  node->rightChild = rotateLL(rightChild);
  return rotateRR(node);
}
```

AVL 트리

AVL 트리의 균형 잡기

AVL 트리의 균형 잡기는 각 노드가 '삽입 될 때'마다 수행되어야 합니다. '삽입' 과정에 소요되는 시간 복잡도는 $O(logN)$ 입니다. 따라서 각 트리의 균형 잡기는 삽입 시에 거치는 모든 노드에 대하여 수행된다는 점에서 $O(1)$의 시간 복잡도를 만족해야 합니다.

AVL 트리

AVL 트리의 균형 잡기

```
Node* balance(Node* node) {
  int difference = getDifference(node);
  if (difference >= 2) {
    if (getDifference(node->leftChild) >= 1) {
      node = rotateLL(node);
    }
    else {
      node = rotateLR(node);
    }
  }
  else if (difference <= -2) {
    if (getDifference(node->rightChild) <= -1) {
      node = rotateRR(node);
    }
    else {
      node = rotateRL(node);
    }
  }
  setHeight(node); // 회전 이후에 높이를 다시 계산
  return node;
}
```

AVL 트리

AVL 트리의 삽입

AVL 트리의 삽입 과정은 일반적인 이진 탐색 트리와 흡사합니다. 다만 삽입 시에 거치는 모든 노드에 대하여 균형 잡기를 수행해주어야 한다는 특징이 있습니다.

AVL 트리

AVL 트리의 삽입

```
Node *insertNode(Node* node, int data) {
  if (!node) {
    node = (Node*)malloc(sizeof(Node));
    node->data = data;
    node->height = 1;
    node->leftChild = node->rightChild = NULL;
  }
  else if (data < node->data) {
    node->leftChild = insertNode(node->leftChild, data);
    node = balance(node);
  }
  else if (data > node->data) {
    node->rightChild = insertNode(node->rightChild, data);
    node = balance(node);
  }
  else {
    printf("데이터 중복이 발생했습니다.\n");
  }
  return node;
}
```

AVL 트리

AVL 트리의 출력 함수

AVL 트리는 삽입되는 과정을 면밀히 살펴보는 것이 중요하므로, 트리 구조를 적절히 보여 줄 수 있는 방식으로 출력해야 합니다. 일반적으로 트리의 너비가 높이보다 크다는 점에서 세로 방향으로 출력하는 함수를 구현할 수 있습니다.

AVL 트리

AVL 트리의 출력 함수

```
Node* root = NULL;

void display(Node* node, int level) {
  if (node != NULL) {
    display(node->rightChild, level + 1);
    printf("\n");
    if (node == root) {
      printf("Root: ");
    }
    for (int i = 0; i < level && node != root; i++) {
      printf("    ");
    }
    printf("%d(%d)", node->data, getHeight(node));
    display(node->leftChild, level + 1);
  }
}
```

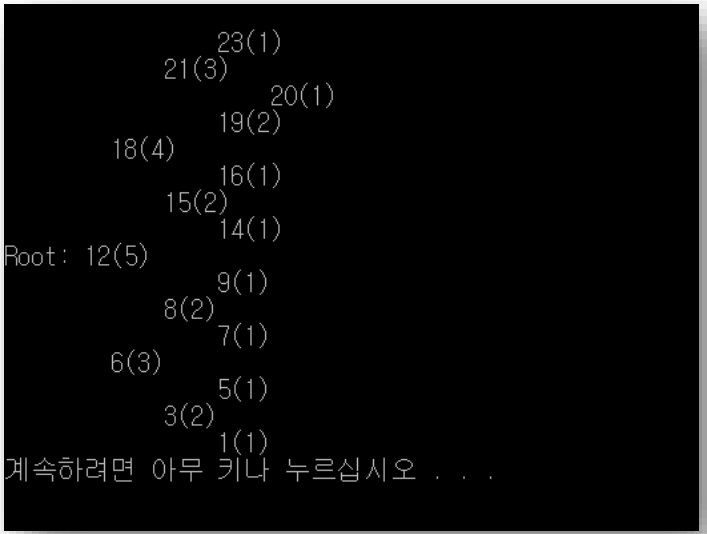
AVL 트리

AVL 트리 이용해보기

```
int main(void) {
  root = insertNode(root, 7);
  root = insertNode(root, 6);
  root = insertNode(root, 5);
  root = insertNode(root, 3);
  root = insertNode(root, 1);
  root = insertNode(root, 9);
  root = insertNode(root, 8);
  root = insertNode(root, 12);
  root = insertNode(root, 16);
  root = insertNode(root, 18);
  root = insertNode(root, 23);
  root = insertNode(root, 21);
  root = insertNode(root, 14);
  root = insertNode(root, 15);
  root = insertNode(root, 19);
  root = insertNode(root, 20);
  display(root, 1); printf("\n");
  system("pause");
}
```

AVL 트리

AVL 트리 구현체 실행 결과



배운 내용 정리하기

AVL 트리

1) 편향 이진 트리(Skewed Binary Tree)의 경우 탐색에 있어 $O(N)$의 시간 복잡도를 가집니다.
2) 따라서 AVL 트리를 이용하여 탐색에 있어서 $O(logN)$의 시간 복잡도를 유지할 수 있습니다.