AVL트리 보고서

2021350007 기선주

Node 클래스

멤버변수

```
private:
  Node  *parent;
  Node  *left;
  Node  *right;
  int   data;
  int   balance;
  int   height;
```

`parent` : 부모노드를 가리키는 포인터

`left` : 왼쪽자식을 가리키는 포인터

`right` : 오른쪽자식을 가리키는 포인터

`data` : 노드가 가지는 데이터

`balance` : 트리구조 변경될 때마다 갱신되는 균형도

`height` : 말단노드로부터 해당노드의 높이

멤버함수

```
public:
  Node (Node *parent, int data) : parent(parent), left(NULL), right(NULL), data(data),
balance(0), height(1);
  Node (int data) : parent(NULL), left(NULL), right(NULL), data(data), balance(0), hei
ght(1);
  ~Node ();
  Node  *get_root();
  void  inorder_print(std::ofstream &out);
  void  preorder_print(std::ofstream &out);
  void  rotL();
  void  rotR();
  int renew_balance(bool &collapsed);
  void  recalibrate_height(void);
  void  increase_height(int childHeight);
  Node  *insert_node(int data);
  int get_data(void);
```

```
  int get_height(void);
  Node  *get_parent(void);
  void set_parent(Node *parent);
  Node  *get_left(void);
  void set_left(Node *left);
  Node  *get_right(void);
  void set_right(Node *right);
```

`Node (Node *, int)` 생성자

```
Node (Node *parent, int data) : parent(parent), left(NULL), right(NULL), data(data), b
alance(0), height(1)
{
  parent->increase_height(height);
}
```

인자로 받은 부모노드 포인터와 data 값을 이용해서 멤버변수를 초기화한다.

이후 부모 노드의 height를 증가시키는 함수 increase_height를 부른다.

`Node (int)` 생성자

```
Node (int data) : parent(NULL), left(NULL), right(NULL), data(data), balance(0), heigh
t(1) {}
```

부모가 없는 노드를 생성하기 위한 생성자이다.

`Node` 소멸자

```
~Node ()
{
  delete left;
  delete right;
}
```

Node 객체 소멸 시 left와 right까지 소멸시켜 주기 때문에, 부모 노드만 delete 해주면 전체 트리가 소멸될 수 있다.

`Node *get_root(void)`

```
Node  *get_root()
{
  if (parent != NULL)
```

```
    return (parent->get_root());
  else
    return (this);
}
```

parent까 NULL이 아닐 때 까지 부모의 get_root 함수를 호출하게 된다. 최종적으로 루트를 만나면 this 포인터를 반환하게 된다. 즉, 어떤 노드에서든 get_root 함수를 호출하면 루트노드의 포인터를 얻을 수 있다.

`void inorder_print(std::ofstream &out)`

```
void  inorder_print(std::ofstream &out)
{
  if (left != NULL)
    left->inorder_print(out);
  out << get_data() << " ";
  if (right != NULL)
    right->inorder_print(out);
}
```

중위순회를 하면서 노드의 값을 출력하는 함수이다. ofstream 객체를 이용해서 출력한다.

중위순회이므로 left→parent→right 순서로 출력하면 된다.

`void preorder_print(std::ofstream &out)`

```
void  preorder_print(std::ofstream &out)
{
  out << get_data() << " ";
  if (left != NULL)
    left->preorder_print(out);
  if (right != NULL)
    right->preorder_print(out);
}
```

전위순회를 하면서 노드의 값을 출력하는 함수이다. 마찬가지로 ofstream 객체를 이용한다.

parent→left→right 순으로 출력한다.

`void rotL()`

```
void  rotL()
{
  if (parent->get_right() == this)
    parent->set_right(get_left());
  else if (parent->get_left() == this)
    parent->set_left(get_left());
  left->set_parent(get_parent());
```

```
  set_parent(get_left());
  set_left(parent->get_right());
  left->set_parent(this);
  parent->set_right(this);
  recalibrate_height();
}
```

노드들을 시계방향으로 회전시키는 함수이다. 균형도가 깨지기 시작한 노드에서 규칙에 따라 호출되어야 한다.

1. 노드 자신이 부모 노드의 오른쪽에 있는지, 왼쪽에 있는지를 확인하여 자신의 왼쪽 노드를 부모의 자식에 연결시킨다.
2. 왼쪽 노드의 부모를 부모에 연결시킨다.
3. 자신의 부모를 자신의 왼쪽노드에 연결시킨다.
4. 자신의 왼쪽노드를 자신의 부모의 오른쪽 노드에 연결시킨다.
5. 자신의 왼쪽노드의 부모를 자신에 연결시킨다.
6. 부모의 오른쪽노드를 자신에 연결시킨다.
7. 마지막으로 `recalibrate_height` 함수를 호출하여 바뀐 노드의 높이를 재설정한다.

`void rotR()`

```
void  rotR()
{
  if (parent->get_right() == this)
    parent->set_right(get_right());
  else if (parent->get_left() == this)
    parent->set_left(get_right());
  right->set_parent(get_parent());
  set_parent(get_right());
  set_right(parent->get_left());
  right->set_parent(this);
  parent->set_left(this);
  recalibrate_height();
}
```

노드들을 반시계방향으로 회전시키는 함수이다. 균형도가 깨지기 시작한 노드에서 규칙에 따라 호출되어야 한다.

`rotL()`과 거의 모든 행위가 동일하지만, 연결하거나 연결당하는 객체의 right와 left를 전부 바꾸면 `rotR()`이 된다.

`int renew_balance(bool &collapsed)`

```
int renew_balance(bool &collapsed)
{
  balance = 0;
  if (left != NULL)
    balance -= left->renew_balance(collapsed);
  if (right != NULL)
    balance += right->renew_balance(collapsed);
  if (collapsed)
    return (-1);
  if (balance <= -2)    // left side
  {
    collapsed = true;
    if (left->left->get_height() > left->right->get_height())
      rotL();
    else if (left->left->get_height() < left->right->get_height())
    {
      left->rotR();
      rotL();
    }
  }
  else if (balance >= 2)    // right side
  {
    collapsed = true;
    if (right->left->get_height() > right->right->get_height())
    {
      right->rotL();
      rotR();
    }
    else if (right->left->get_height() < right->right->get_height())
      rotR();
  }
  return (height);
}
```

균형도를 갱신하는 함수이다. collapsed 변수는 균형도가 깨졌는지를 확인하는 용도로 사용된다. `true`일 때 깨졌다고 판단한다.

`void recalibrate_height(void)`

```
void  recalibrate_height(void)
{
  if (this == NULL)
    return ;
  this->height = 1 + std::max(this->left->get_height(), this->right->get_height());
  parent->recalibrate_height();
}
```

높이를 재설정하는 함수이다. 해당 노드의 높이는 말단노드일때 1이고, 그렇지 않을 때는 왼쪽과 오른쪽 노드의 높이 중 최댓값에 1을 더한 값이다.

이 함수는 BN에서 실행되는데, `rotate`시킬 때 BN으로 부터 그 조상노드까지 높이가 변화할 수 있기 때문이다. 따라서 parent의 recalibrate_height 함수를 불러주는 것이다.

`void increase_height(int childHeight)`

```
void  increase_height(int childHeight)
{
  if (childHeight >= height)
  {
    height = childHeight + 1;
    if (parent != NULL)
      parent->increase_height(height);
  }
}
```

노드 삽입 시 호출되는 함수로, 현 노드의 height 값을 부모 노드의 함수에 전달하여 크기비교를 통해 높이를 갱신한다. 노드 삽입 시에만 사용할 수 있으며 recalibrate_height 보다 연산 횟수가 적다.

`Node *insert_node(int data)`

```
Node  *insert_node(int data)
{
  if (this->data > data)
  {
    if (left != NULL)
      return (left->insert_node(data));
    else
      return (left = new Node(this, data));
  }
  else
  {
    if (right != NULL)
      return (right->insert_node(data));
    else
      return (right = new Node(this, data));
  }
}
```

이진트리에서 노드를 삽입하는 것과 같은 방식으로 노드를 삽입한다. 삽입하려는 데이터가 현재 노드보다 크면 오른쪽을 검색하고 그렇지 않으면 왼쪽을 검색한다.

`settter & getter`

```
int get_data(void)
{
  if (this == NULL)
    return (-1);
```

```
  return (data);
}
int get_height(void)
{
  if (this == NULL)
    return (0);
  return (height);
}
Node  *get_parent(void)
{
  if (this == NULL)
    return (NULL);
  return (parent);
}
void set_parent(Node *parent)
{
  if (this == NULL)
    return ;
  this->parent = parent;
}
Node  *get_left(void)
{
  if (this == NULL)
    return (NULL);
  return (left);
}
void set_left(Node *left)
{
  if (this == NULL)
    return ;
  this->left = left;
}
Node  *get_right(void)
{
  if (this == NULL)
    return (NULL);
  return (right);
}
void set_right(Node *right)
{
  if (this == NULL)
    return ;
  this->right = right;
}
```

편의를 위해서 this가 NULL인지 체크하는 분기점을 만들었다. 혹시 모를 memory exception을 방지하기 위해서이다. 또한 this가 NULL인 상태를 의도하여 프로그래밍 하였다. `get_height` 함수는 this가 NULL일 때, 0을 반환함으로서 트리의 일관성에 관여하였다.

전역함수

`void insert_node(int data, Node *&AVLParent)`

```
void  insert_node(int data, Node *&AVLParent)
{
  if (AVLParent == NULL)
  {
    AVLParent = new Node(data);
    return ;
  }
  AVLParent->insert_node(data);
  bool  balanceCollapsed;
  do {
    balanceCollapsed = false;
    AVLParent->renew_balance(balanceCollapsed);
  } while (balanceCollapsed);
  AVLParent = AVLParent->get_root();
}
```

`AVLParent`가 NULL일 때, 부모를 전달하지 않는 생성자를 이용해서 루트노드를 만든다.

다음 삽입때는 루트노드의 멤버함수를 이용해서 노드를 삽입하고 균형도를 갱신하고 루트노드를 갱산한다.

`main 함수`

```
int main(void)
{
  Node      *AVLParent = NULL;
  std::ifstream in("AVL.in");
  std::ofstream out("AVL.out");
  char  op;
  int   content;
  while (!in.eof())
  {
    in >> op;
    if (in.eof())
      break ;
    in >> content;
    if (op == 'I')
    {
      insert_node(content, AVLParent);
      out << "I ";
      AVLParent->inorder_print(out);
      out << std::endl;
      out << "P ";
      AVLParent->preorder_print(out);
      out << std::endl << std::endl;
    }
  }
  in.close();
  out.close();
  delete AVLParent;
}
```

ifstream으로 `AVL.in` 함수를 연다. ifstream 객체에는 eof 함수가 있어서 파일의 끝을 만났는지를 확인할 수 있다. ofstream으로 `AVL.out` 함수를 연다. `print` 하는 함수에 객체를 넣어 보내면 잘 작동한다.