AppData\Local\Temp\a5726dea-11ca-47cc-b420-7b0c76652a1f_OSAP_003_7_최종보고서(소스코드 포함).zip.a1f\src\erase.cc

```
/*
MIT License
This file is part of the INHA_OSAP_003_7 project.
Copyright (c) 2024 tbmyong

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, includingp without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.

작성자: 류남경
작성일(파일 생성일): 2024-12-05
작성일(파일 최종 수정일): 2024-12-20
*/

/*
<Erase 기능 구현>
Node* FindMin(Node* current_node)
: subtree에서 key값이 최소인 node를 찾는다

void ReplaceNode(Node* erased_node, Node* replace_node)
: 삭제할 노드와 대체할 노드의 자리를 바꾼다

Node* EraseNode(Node* current_node, int key)
: 노드를 삭제하고 subtree의 루트 노드를 반환한다
*/

#include <iostream>

#include "../base/avl.h"

void AVL::ReplaceNode(Node* erased_node, Node* replace_node) {
  // 삭제될 노드의 부모가 없는 경우 = root인 경우
  if (erased_node->get_parent() == nullptr) {
    set_root(replace_node);
  }
  // 삭제될 노드가 부모의 왼쪽 자식인 경우
  else if (erased_node == erased_node->get_parent()->get_left()) {
```

```cpp
      erased_node->get_parent()->set_left(replace_node);
    }
    // 삭제될 노드가 부모의 오른쪽 자식인 경우
    else if (erased_node == erased_node->get_parent()->get_right()) {
      erased_node->get_parent()->set_right(replace_node);
    }

    // 대체될 노드가 nullptr이 아닌 경우
    if (replace_node != nullptr) {
      replace_node->set_parent(erased_node->get_parent());
    }
  }

  Node* AVL::EraseNode(Node* current_node, int key) {
    // 노드가 없는 경우
    if (current_node == nullptr) {
      return nullptr;
    }

    // key가 작은 경우
    if (key < current_node->get_key()) {
      current_node->set_left(EraseNode(current_node->get_left(), key));
    }
    // key가 큰 경우
    else if (key > current_node->get_key()) {
      current_node->set_right(EraseNode(current_node->get_right(), key));
    }
    // key가 같은 경우 = 삭제할 노드를 찾은 경우
    else {
      // 왼쪽 자식이 없는 경우
      if (current_node->get_left() == nullptr) {
        Node* right_node = current_node->get_right();
        ReplaceNode(current_node, right_node);
        delete current_node;
        return right_node;
      }
      // 오른쪽 자식이 없는 경우
      else if (current_node->get_right() == nullptr) {
        Node* left_node = current_node->get_left();
        ReplaceNode(current_node, left_node);
        delete current_node;
        return left_node;
      }
      // 자식이 둘 다 있는 경우
      else {
        Node* succesor_node = FindMin(current_node->get_right());
        current_node->set_key(succesor_node->get_key());
        current_node->set_right(
            EraseNode(current_node->get_right(), succesor_node->get_key()));
      }
    }

    updater_.Update(current_node);
    balancer_.Balance(current_node);

    return current_node;
  }

  void AVL::Erase(int key) {
    Node* erased_node = FindNode(root_, key);

    // 삭제할 노드를 찾은 경우
    if (erased_node != nullptr) {
      std::cout << CalculateDepthHeightSum(root_, key) << "\n";
      Node* new_root = EraseNode(root_, key);
      set_root(new_root);
    }
    // 노드가 없는 경우
    else {
      std::cout << "0\n";
    }
  }

```