Câu 1:

Dãy ban đầu: [60, 39, 52, 20, 90, 2, 15, 30]

Bước 1: Chọn pivot = 30 (chọn phần tử cuối của dãy)

Chia dãy:

- Nhỏ hơn pivot: [20, 2, 15]
- Bằng pivot: [30]
- Lớn hơn pivot: [60, 39, 52, 90]

Bước 2: Sắp xếp [20, 2, 15]

- Chọn pivot = 15
- Nhỏ hơn pivot: [2]
- Bằng pivot: [15]
- Lớn hơn pivot: [20]
- Ghép lại:
  - [2] + [5] + [20] = [2, 15, 20]

Bước 3: Sắp xếp [60, 39, 52, 90]

- Chọn pivot = 90
- Nhỏ hơn pivot: [60, 39, 52]
- Bằng pivot: [90]
- Lớn hơn pivot: []

Bước 4: Sắp xếp [60, 39, 52]

- Chọn pivot = 52
- Nhỏ hơn pivot: [39]
- Bằng pivot: [52]
- Lớn hơn pivot: [60]
- Ghép lại:
  - [39] + [52] + [60] = [39, 52, 60]

Ghép lại:

- [2, 15, 20] + [30] + [39, 52, 60] + [90] = [2, 15, 20, 30, 39, 52, 60, 90]

Dãy đã sắp xếp: [2, 15, 20, 30, 39, 52, 60, 90]

Câu 2:

File Appointment.h

```
#pragma once
#include <string>
using namespace std;

class Appointment {
public:
    string datetime; // yyyyMMddhhmm
    string patientName;
    string doctorName;
    string note;
    Appointment();
    Appointment(string dt, string pn, string dn, string nt);
};
```

File Appointment.cpp

```
#include "Appointment.h"
Appointment::Appointment() {}
Appointment::Appointment(string dt, string pn, string dn, string nt)
    : datetime(dt), patientName(pn), doctorName(dn), note(nt) {}
```

File AVLTree.h

```
#pragma once
#include "Appointment.h"
```

```
#include <vector>

class AVLNode {
public:
    Appointment data;
    AVLNode *left, *right;
    int height;
    AVLNode(const Appointment& appt);
};

class AVLTree {
private:
    AVLNode* root;
    AVLNode* insert(AVLNode* node, const Appointment& appt);
    AVLNode* deleteNode(AVLNode* root, string datetime);
    AVLNode* minValueNode(AVLNode* node);
    AVLNode* searchByDatetime(AVLNode* node, string datetime);
    void searchByName(AVLNode* node, const string& name,
std::vector<Appointment>& result);
    void inorder(AVLNode* node);
    int getHeight(AVLNode* node);
    int getBalance(AVLNode* node);
    AVLNode* rightRotate(AVLNode* y);
    AVLNode* leftRotate(AVLNode* x);
public:
```

```
    AVLTree();
    void addAppointment(const Appointment& appt);
    void removeAppointment(string datetime);
    void searchByDatetime(string datetime);
    void searchByName(string name);
    void printAll();
};
```

File AVLTree.cpp

```
#include "AVLTree.h"
#include <iostream>
#include <vector>
using namespace std;

AVLNode::AVLNode(const Appointment& appt) : data(appt), left(nullptr),
right(nullptr), height(1) {}

AVLTree::AVLTree() : root(nullptr) {}

int AVLTree::getHeight(AVLNode* node) { return node ? node->height : 0; }
int AVLTree::getBalance(AVLNode* node) { return node ? getHeight(node->left) -
getHeight(node->right) : 0; }
AVLNode* AVLTree::rightRotate(AVLNode* y) {
    AVLNode* x = y->left;
    AVLNode* T2 = x->right;
    x->right = y;
```

```
    y->left = T2;
    y->height = max(getHeight(y->left), getHeight(y->right)) + 1;
    x->height = max(getHeight(x->left), getHeight(x->right)) + 1;
    return x;
}
AVLNode* AVLTree::leftRotate(AVLNode* x) {
    AVLNode* y = x->right;
    AVLNode* T2 = y->left;
    y->left = x;
    x->right = T2;
    x->height = max(getHeight(x->left), getHeight(x->right)) + 1;
    y->height = max(getHeight(y->left), getHeight(y->right)) + 1;
    return y;
}
AVLNode* AVLTree::insert(AVLNode* node, const Appointment& appt) {
    if (!node) return new AVLNode(appt);
    if (appt.datetime < node->data.datetime)
        node->left = insert(node->left, appt);
    else if (appt.datetime > node->data.datetime)
        node->right = insert(node->right, appt);
    else
        return node;
    node->height = 1 + max(getHeight(node->left), getHeight(node->right));
    int balance = getBalance(node);
```

```
    if (balance > 1 && appt.datetime < node->left->data.datetime)
        return rightRotate(node);
    if (balance < -1 && appt.datetime > node->right->data.datetime)
        return leftRotate(node);
    if (balance > 1 && appt.datetime > node->left->data.datetime) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }
    if (balance < -1 && appt.datetime < node->right->data.datetime) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }
    return node;
}
AVLNode* AVLTree::minValueNode(AVLNode* node) {
    AVLNode* current = node;
    while (current && current->left)
        current = current->left;
    return current;
}
AVLNode* AVLTree::deleteNode(AVLNode* root, string datetime) {
    if (!root) return root;
    if (datetime < root->data.datetime)
        root->left = deleteNode(root->left, datetime);
```

```
else if (datetime > root->data.datetime)
    root->right = deleteNode(root->right, datetime);
else {
    if (!root->left || !root->right) {
        AVLNode* temp = root->left ? root->left : root->right;
        if (!temp) {
            temp = root;
            root = nullptr;
        } else
            *root = *temp;
        delete temp;
    } else {
        AVLNode* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data.datetime);
    }
}
if (!root) return root;
root->height = 1 + max(getHeight(root->left), getHeight(root->right));
int balance = getBalance(root);
if (balance > 1 && getBalance(root->left) >= 0)
    return rightRotate(root);
if (balance > 1 && getBalance(root->left) < 0) {
    root->left = leftRotate(root->left);
```

```
        return rightRotate(root);
    }
    if (balance < -1 && getBalance(root->right) <= 0)
        return leftRotate(root);
    if (balance < -1 && getBalance(root->right) > 0) {
        root->right = rightRotate(root->right);
        return leftRotate(root);
    }
    return root;
}
AVLNode* AVLTree::searchByDatetime(AVLNode* node, string datetime) {
    if (!node) return nullptr;
    if (datetime == node->data.datetime) return node;
    if (datetime < node->data.datetime)
        return searchByDatetime(node->left, datetime);
    return searchByDatetime(node->right, datetime);
}
void AVLTree::searchByName(AVLNode* node, const string& name,
vector<Appointment>& result) {
    if (!node) return;
    searchByName(node->left, name, result);
    if (node->data.patientName == name)
        result.push_back(node->data);
    searchByName(node->right, name, result);
}
```

```
void AVLTree::inorder(AVLNode* node) {
    if (!node) return;
    inorder(node->left);
    cout << node->data.datetime << " | " << node->data.patientName << " | " << node->data.doctorName << " | " << node->data.note << endl;
    inorder(node->right);
}
void AVLTree::addAppointment(const Appointment& appt) {
    root = insert(root, appt);
}
void AVLTree::removeAppointment(string datetime) {
    root = deleteNode(root, datetime);
}
void AVLTree::searchByDatetime(string datetime) {
    AVLNode* res = searchByDatetime(root, datetime);
    if (res) {
        cout << "Tim thay lich hen: " << res->data.datetime << " | " << res->data.patientName << " | " << res->data.doctorName << " | " << res->data.note << endl;
    } else {
        cout << "Khong tim thay lich hen!" << endl;
    }
}
void AVLTree::searchByName(string name) {
    vector<Appointment> result;
```

```
    searchByName(root, name, result);
    if (result.empty()) cout << "Khong tim thay lich hen!" << endl;
    else {
        for (auto& appt : result) {
            cout << appt.datetime << " | " << appt.patientName << " | " << appt.doctorName << " | " << appt.note << endl;
        }
    }
}
void AVLTree::printAll() {
    cout << "Danh sach lich hen theo thu tu thoi gian:" << endl;
    inorder(root);
}
```

File main.cpp

```
#include "AVLTree.h"
#include <iostream>
using namespace std;

int main() {
    AVLTree system;
    // Thêm lịch hẹn mẫu
    system.addAppointment(Appointment("202505091000", "Nguyen Van A", "Nguyen Hoa Da", "Kham tong quat"));
    system.addAppointment(Appointment("202505091030", "Tran Quan Vu", "Le Van Sy", "Tu van dinh duong"));
```

```
    system.addAppointment(Appointment("202505091200", "Le Thi B", "Nguyen Hoa Da", "Kham mat"));
    // Hiển thị danh sách
    system.printAll();
    // Tìm kiếm theo tên
    cout << "\nTra cuu lich hen cua Tran Quan Vu:" << endl;
    system.searchByName("Tran Quan Vu");
    // Xóa lịch hẹn
    cout << "\nXoa lich hen 202505091030..." << endl;
    system.removeAppointment("202505091030");
    system.printAll();
    // Tìm kiếm theo ngày giờ
    cout << "\nTra cuu lich hen 202505091200:" << endl;
    system.searchByDatetime("202505091200");
    return 0;
}
```

Câu 3:

Dãy số được thêm vào cây theo thứ tự:
`65, 35, 51, 26, 90, -12, 18, 30, 37, 29, 63, 1`

A. Xây dựng cây BST

1. Thêm 65 → Gốc của cây là `65`.

```
65
```

2. Thêm 35 → `35 < 65`, Là con trái của `65`.

```
   65
  /
35
```

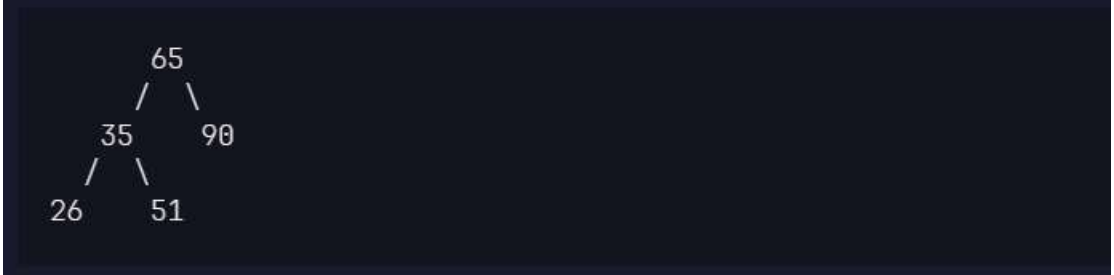
3. Thêm 51 → `51 < 65`, `51 > 35`, Là con phải của `35`.

```
   65
  /
35
  \
   51
```

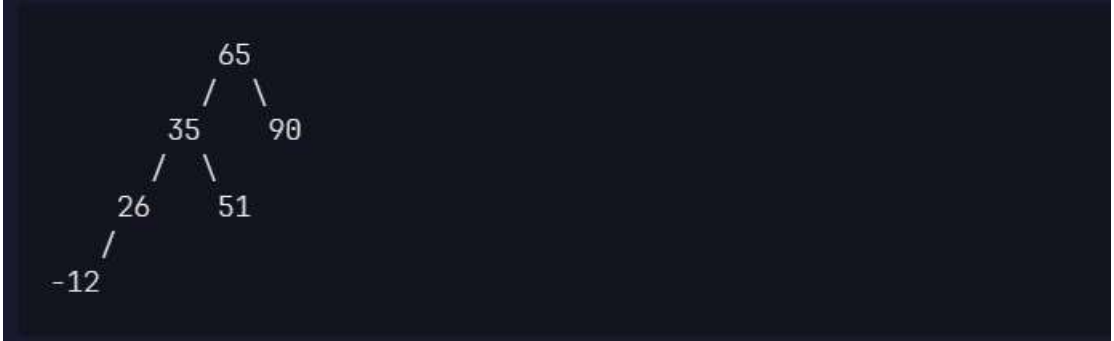
4. Thêm 26 → `26 < 65`, `26 < 35`, Là con trái của `35`.

```
      65
     /
   35
  /  \
26    51
```

5. Thêm 90 → `90 > 65`**, Là con phải của** `65`**.**

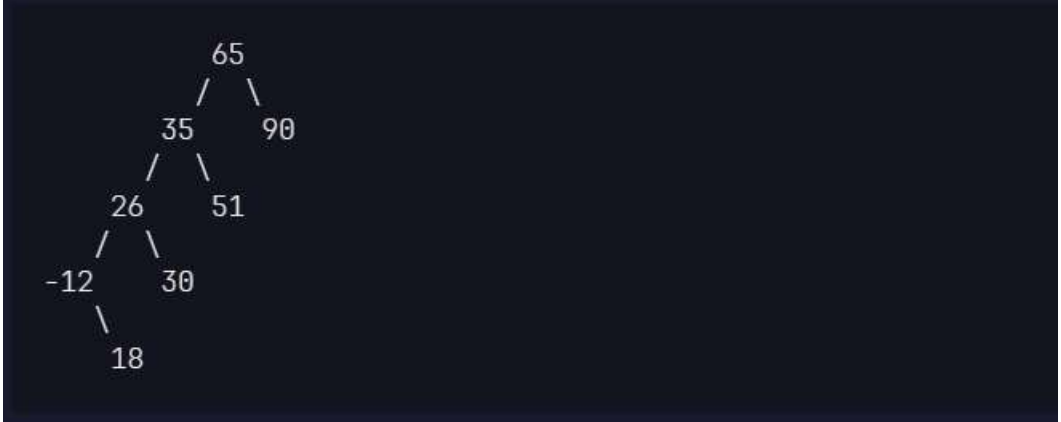


6. Thêm -12 → `-12 < 65`**,** `-12 < 35`**,** `-12 < 26`**, Là con trái của** `26`**.**

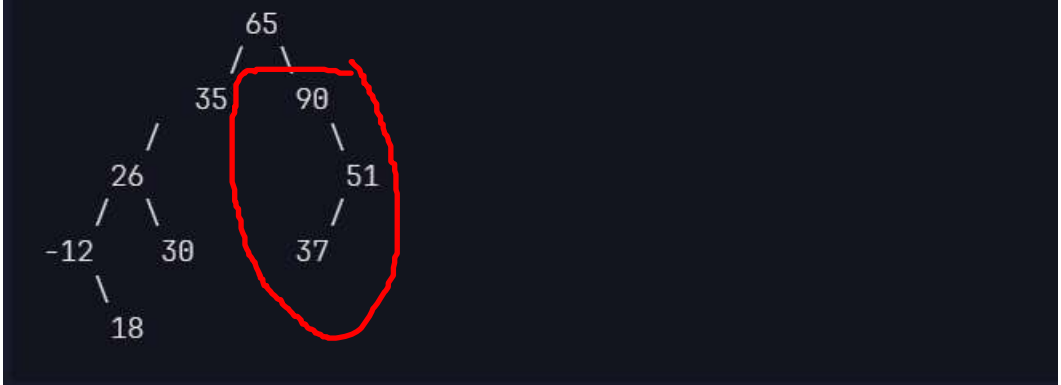


7. Thêm 18 → `18 < 65`**,** `18 < 35`**,** `18 < 26`**,** `18 > -12`**, Là con phải của** `-12`**.**

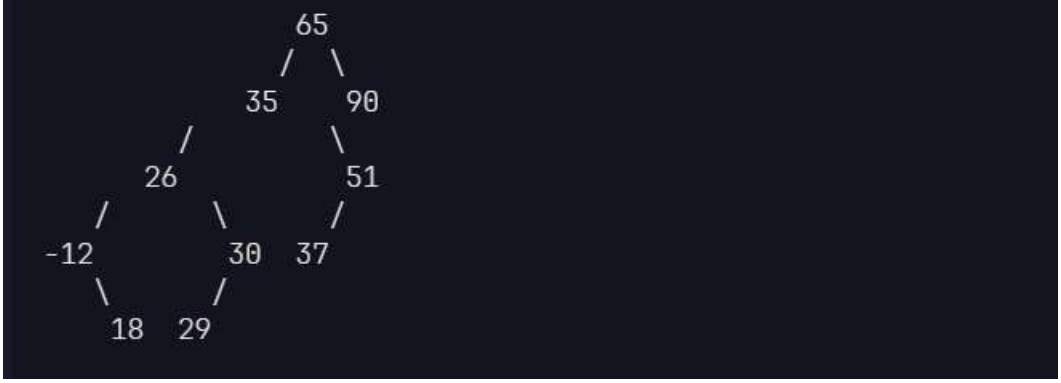
8. Thêm 30 → `30 < 65`, `30 < 35`, `30 > 26`, **Là con phải của** `26`.



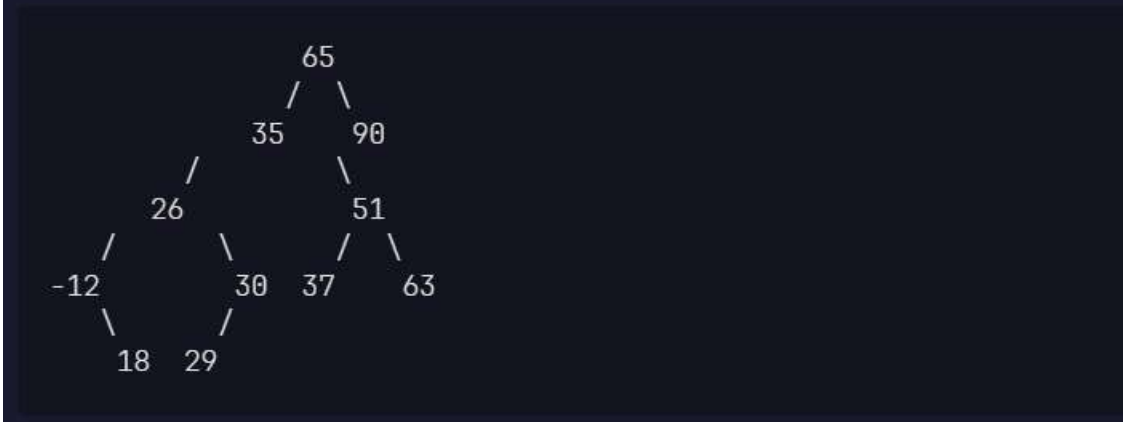
9. Thêm 37 → `37 < 65`, `37 > 35`, `37 < 51`, **Là con trái của** `51`.



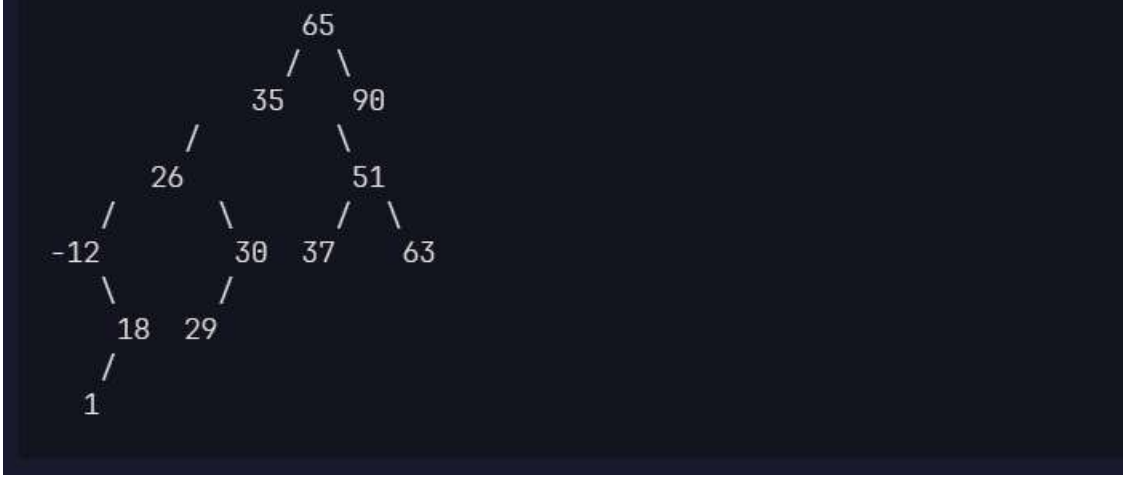
10. Thêm 29 → `29 < 65`, `29 < 35`, `29 > 26`, `29 < 30`, **Là con trái của** `30`.



11. Thêm 63 → `63 < 65`, `63 > 35`, `63 > 51`, **Là con phải của** `51`.



12. Thêm 1 → `1 < 65`, `1 < 35`, `1 < 26`, `1 > -12`, `1 < 18`, **Là con trái của** `18`.



B. Xây dựng cây AVL

1. Thêm 65 → **Gốc của cây là** `65`.

```
65
```

→ Cân bằng.

2. Thêm 35 → `35 < 65`, **Là con trái của** `65`.

```
   65
  /
35
```

→ Cân bằng.

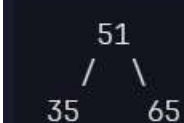
3. Thêm 51 → `51 < 65`, `51 > 35`, **Là con phải của** `35`.

```
   65
  /
35
  \
   51
```

→ Mất cân bằng tại 65 (Trái - Phải). → Xoay trái tại 35

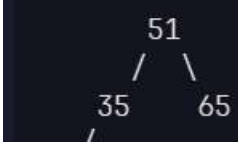
```
     65
    /
  51
 /
35
```

→ Xoay phải tại 65



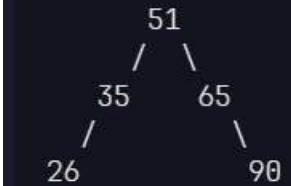
→ Cân bằng.

4. Thêm 26 → `26 < 51`, `26 < 35`, **Là con trái của** `35`.



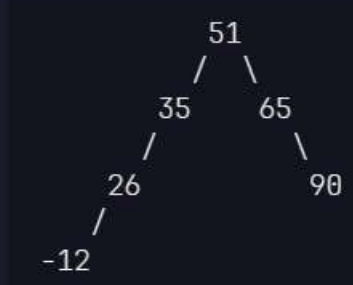
→ Cân bằng.

5. Thêm 90 → `90 > 51`, `90 > 65`, **Là con phải của** `65`.

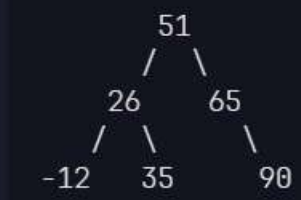


→ Cân bằng.

6. Thêm -12 → `-12 < 51`, `-12 < 35`, `-12 < 26`, **Là con trái của** `26`.

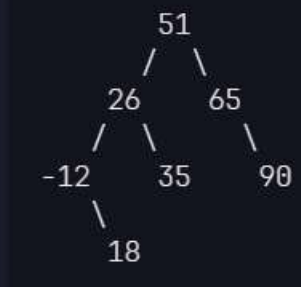


→ Mất cân bằng tại 35 (Trái-Trái). → Xoay phải tại 35.



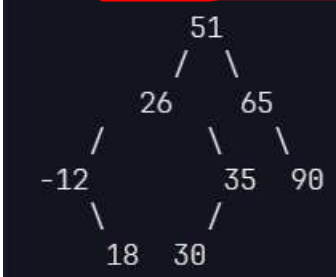
→ Cân bằng.

7. Thêm 18 → `18 < 51`, `18 < 26`, `18 > -12`, **Là con phải của** `-12`.



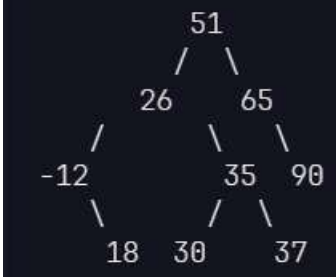
→ Cân bằng.

8. Thêm 30 → `30 < 51`, `30 > 26`, **Là con phải của** `26`.



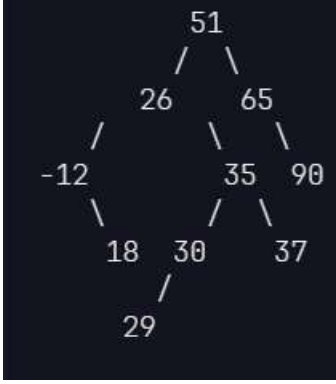
→ Cân bằng.

9. Thêm 37 → `37 < 51`, `37 > 26`, `37 > 35`, **Là con phải của** `35`

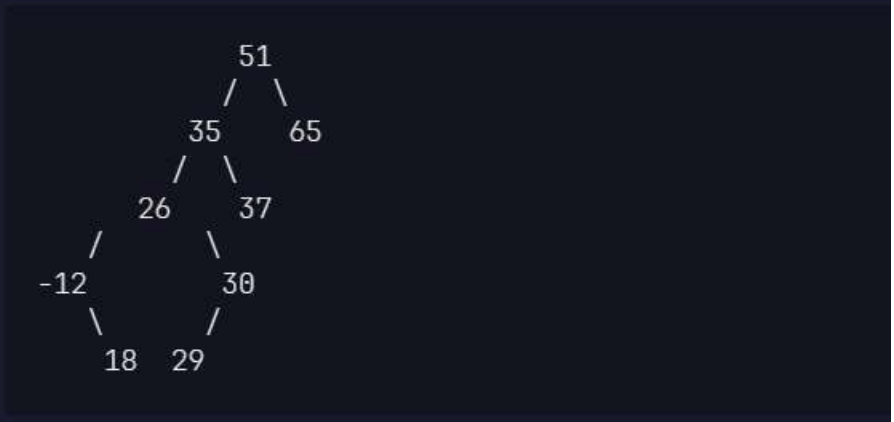


→ Cân bằng.

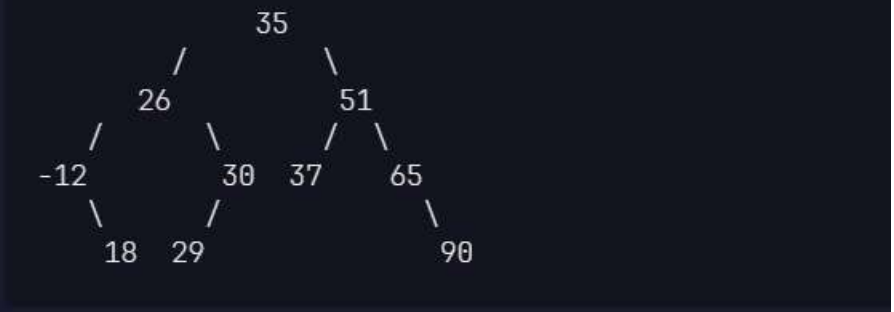
10. Thêm 29 → `29 < 51`, `29 > 26`, `29 < 35`, `29 > 30`, **Là con trái của** `30`.



→ Mất cân bằng tại 51 (Trái - Phải). → Xoay trái tại 26

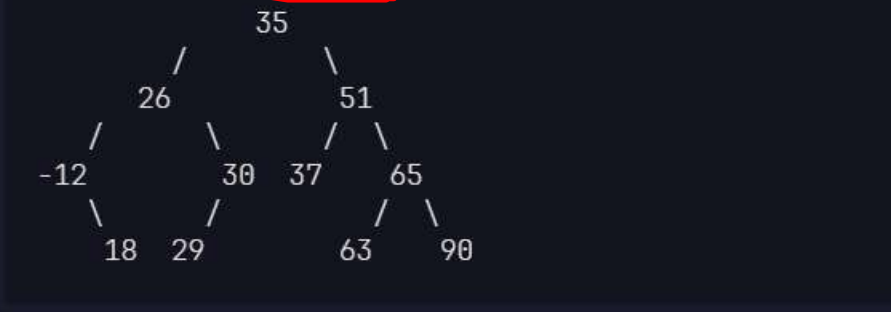


→ Xoay phải tại 51



→ Cân bằng.

11. Thêm 63 → `63 >35`, `63 > 51`, `63 > 65`, **là con trái của** `65`.

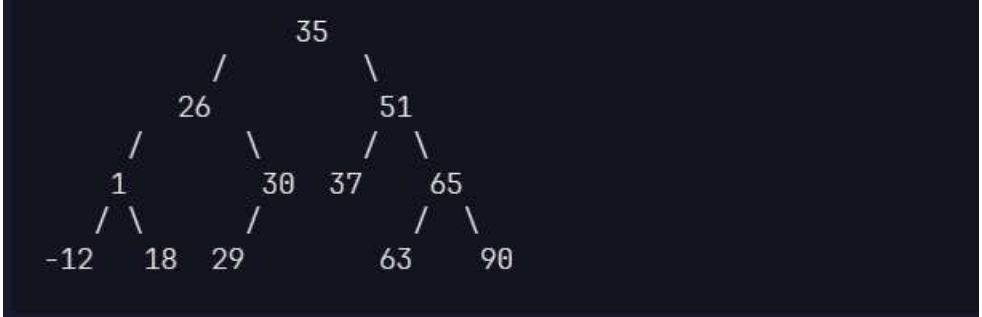


→ Cân bằng.

12. Thêm 1 → `1 < 35`, `1 < 26`, `1 > -12`, `1 < 18`, Là con trái của `18`.

→ Mất cân bằng tại -12 (Phải - Trái). → Xoay phải tại 18

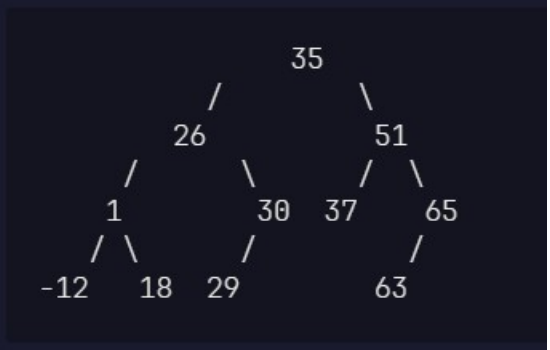
→ Xoay trái tại -12



→ Cân bằng.

C. Xóa 90

- 90 là nút lá, chỉ cần xóa trực tiếp.



→ Cân bằng.