Министерство образования Республики Беларусь

Учреждения образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №4

По дисциплине МОИС за 1 семестр

«Проверка бинарных деревьев на изоморфность»

Выполнил

Студент группы ИИ-21

Кирилович А.А.

Проверил

Козинский А.А.

Брест 2023

Цель: изучить обходы бинарных деревьев, алгоритмы проверки на изморфность.

Задание:

1. Написать программу проверки бинарных деревьев на изоморфность, используя обходы бинарных деревьев. Варианты заданий указаны в таблице 1. Дерево задано списком ребер. Корнем дерева является вершина ***a.***
2. Добавлением и удалением вершин в деревьях привести деревья к изоморфному виду.
3. Определить высоту каждого дерева и указать является ли оно сбалансированным.
4. Изобразить деревья.

Вариант:

|  |  |  |
| --- | --- | --- |
| № | Дерево | Дерево |
| 5. | (a,b),(a,c),(b,d),(c,e),(e,f),(e,g),(d,h) | (a,b),(a,c),(b,d),(b,e),(c,f),(c,g),(d,h) |

Выполнение:

task.cpp:

#include "../graph\_LIB.hh"

void test(std::string file\_path1, std::string file\_path2)

{

    convert c;

    std::vector<std::string> nodes1, nodes2;

    nodes1 = c.reading\_file\_symb(file\_path1);

    nodes2 = c.reading\_file\_symb(file\_path2);

    tree\_symb tree1, tree2;

    for(int i = 0; i < nodes1.size(); i += 2)

    {

        tree1.add\_node(nodes1[i], nodes1[i + 1]);

    }

    for(int i = 0; i < nodes2.size(); i += 2)

    {

        tree2.add\_node(nodes2[i], nodes2[i + 1]);

    }

    tree1.print\_tree();

    std::cout << std::endl << std::endl;

    tree2.print\_tree();

    std::cout << std::endl << std::endl;

    std::cout << "T1 height : " << tree1.height(tree1.get\_root()) << std::endl;

    std::cout << "T2 height : " << tree2.height(tree2.get\_root()) << std::endl;

    std::cout << std::endl << std::endl;

    std::cout << "T1 is balanced? : " << tree1.isBalanced(tree1.get\_root()) << std::endl;

    std::cout << "T2 is balanced? : " << tree2.isBalanced(tree2.get\_root()) << std::endl;

    alg a;

    bool check = a.is\_isomorphic(tree1.get\_root(), tree2.get\_root());

    std::cout << "Isomorphic? : " << check;

    std::cout << std::endl << std::endl;

    a.try2make\_isomorphic(tree1, tree2);

    tree1.print\_tree();

    std::cout << std::endl << std::endl;

    tree2.print\_tree();

    std::cout << std::endl << std::endl;

    check = a.is\_isomorphic(tree1.get\_root(), tree2.get\_root());

    std::cout << "Isomorphic? : " << check;

    std::cout << std::endl << std::endl;

}

int main()

{

    int v = 5;

    test("T1/T1v" + std::to\_string(v) + ".txt", "T2/T2v" + std::to\_string(v) + ".txt");

}

Функция height:

    int height(Node\* root) {

        if (root == nullptr) {

            return 0;

        }

        return 1 + std::max(height(root->left), height(root->right));

    }

Фукнция isBalanced:

    bool isBalanced(Node\* root) {

        if (root == NULL) {

            return true;

        }

        int leftHeight = height(root->left);

        int rightHeight = height(root->right);

        if (abs(leftHeight - rightHeight) <= 1 && isBalanced(root->left) && isBalanced(root->right)) {

            return true;

        }

        return false;

    }

Функция is\_isomorphic:

bool is\_isomorphic(Node\* node1, Node\* node2)

    {

        if (node1 == nullptr && node2 == nullptr)

        {

            return true;

        }

        if (node1 == nullptr || node2 == nullptr)

        {

            return false;

        }

        if (node1->value != node2->value)

        {

            return false;

        }

        bool is\_left\_isomorphic = is\_isomorphic(node1->left, node2->left) && is\_isomorphic(node1->right, node2->right);

        bool is\_right\_isomorphic = is\_isomorphic(node1->left, node2->right) && is\_isomorphic(node1->right, node2->left);

        return is\_left\_isomorphic || is\_right\_isomorphic;

    }

Функция try2make\_isomorphic:

void try2make\_isomorphic(tree\_symb &T1, tree\_symb &T2)

    {

        VEC\_S nodes1, nodes2;

        T1.pre\_order\_traversal(T1.get\_root(), nodes1);

        T2.pre\_order\_traversal(T2.get\_root(), nodes2);

        VEC\_S not\_in\_T1, not\_in\_T2;

        find\_missing\_elements(nodes1, nodes2, not\_in\_T1, not\_in\_T2);

        add\_and\_delete(T1, T2, not\_in\_T1, not\_in\_T2);

        std::string s;

        nodes1.clear();

        nodes2.clear();

        T1.pre\_order\_traversal(T1.get\_root(), nodes1);

        T2.pre\_order\_traversal(T2.get\_root(), nodes2);

        stack<std::string> to\_delete;

        find\_swap\_to\_increase\_similarity(nodes1, nodes2, to\_delete);

        while(!is\_isomorphic(T1.get\_root(), T2.get\_root()))

        {

            s = to\_delete.front();

            to\_delete.pop();

            T1.remove\_node(s);

            T2.remove\_node(s);

            not\_in\_T1.clear();

            not\_in\_T2.clear();

            nodes1.clear();

            nodes2.clear();

            T1.pre\_order\_traversal(T1.get\_root(), nodes1);

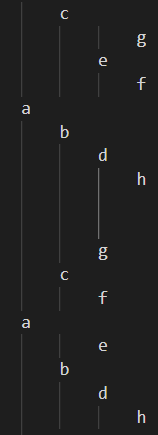
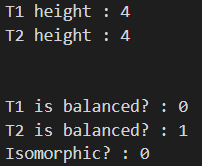
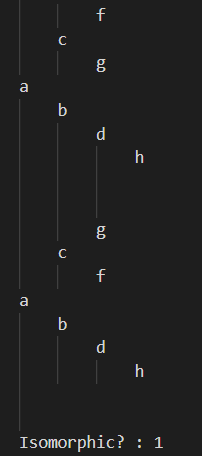
            T2.pre\_order\_traversal(T2.get\_root(), nodes2);

            find\_missing\_elements(nodes1, nodes2, not\_in\_T1, not\_in\_T2);

            add\_and\_delete(T1, T2, not\_in\_T1, not\_in\_T2);

        }

    }

Вывод: изучил обходы бинарных деревьев, алгоритмы проверки на изморфность.