**ПМ-91** Моргунов Семён

**ПМ-91** Павлов Сергей

**ПМ-91** Попков Роман

**ПМ-93** Назаров Даниил

**Условие задачи**

Определить, является ли граф деревом, для неориентированного графа. Граф в памяти представлять в виде списка смежных вершин для каждой вершины.

**Анализ задачи**

*Входные данные:* количество вершин n и рёбер m графа, набор рёбер в виде пары вершин.

*Выходные данные:* булева функция, показывающая, является ли граф деревом.

*Решение:* первоочередная проверка на дерево: условие, верно ли, что количество рёбер на единицу меньше количества вершин. Для обоих обходов дополнительно применяется булев вектор длиной N, определяющий, проверялась ли вершина в функции обхода.

* *Поиск в ширину*: используется дополнительно очередь. Выполняется цикл: с начала очереди берётся вершина, и в конец очереди добавляются все смежные с ней непроверенные вершины, при этом вершина из начала исключается из очереди; до какого-то момента вершины будут добавляться в очередь, затем они будут только удаляться из очереди. Цикл повторяется, пока очередь не пуста. Если булев вектор содержит полностью истинные значения, то граф является деревом.
* Поиск в глубину: если вершина не проверена, то рекурсивно вызывается функция обхода для её смежных вершин. Если все вершины будут проверены, то функция завершается. Если булев вектор содержит полностью истинные значения, то граф является деревом.

Особенность обходов заключается в том, что если будут находиться циклы в графе, то найдётся как минимум один элемент вектора, являющийся ложным (в смысле не прошла проверка одной вершины). Аналогичная ситуация возникает, если в графе найдётся вершина, связанная сама с собой (то есть будет образовывать петлю, что по сути эквивалентно наличию цикла в графе).

**Код программы проверки на дерево с обходом в ширину**

#include <iostream>

#include <fstream>

#include <vector>

#include <queue>

using namespace std;

vector<bool> marks;

void bfs(int n, const vector<vector<int>> G)

{

queue <int> q;

q.push(0);

marks[0] = true;

while (!q.empty())

{

int cur\_node = q.front();

q.pop();

for (int i : G[cur\_node])

if (!marks[i])

{

q.push(i);

marks[i] = true;

}

}

}

int main()

{

setlocale(LC\_ALL, "Russian");

ifstream fin("test.txt");

ofstream fout("res.txt");

int n, m;

fin >> n >> m;

bool result = true;

if (m != n - 1)

{

fout << "0";

return 0;

}

vector<vector<int>> Graph(n);

int a, b;

for (int i = 0; i < m; i++)

{

fin >> a >> b;

Graph[a - 1].push\_back(b - 1);

Graph[b - 1].push\_back(a - 1);

}

marks.assign(n, false);

bfs(0, Graph);

for (int i = 0; i < n; i++)

if (!marks[i])

result = false;

fout << result;

return 0;

}

**Код программы проверки на дерево с обходом в глубину**

#include <iostream>

#include <fstream>

#include <vector>

using namespace std;

vector<bool> marks;

void dfs(int cur\_node, const vector<vector<int>> G)

{

marks[cur\_node] = true;

for (int i = 0; i < G[cur\_node].size(); i++)

if (!marks[G[cur\_node][i]])

dfs(G[cur\_node][i], G);

}

int main()

{

setlocale(LC\_ALL, "Russian");

ifstream fin("test.txt");

ofstream fout("res.txt");

int n, m;

fin >> n >> m;

bool result = true;

if (m != n - 1)

{

fout << "0";

return 0;

}

vector<vector<int>> Graph(n);

int a, b;

for (int i = 0; i < m; i++)

{

fin >> a >> b;

Graph[a - 1].push\_back(b - 1);

Graph[b - 1].push\_back(a - 1);

}

marks.assign(n, false);

dfs(0, Graph);

for (int i = 0; i < n; i++)

if (!marks[i])

result = false;

fout << result;

return 0;

}

**Тесты для поиска ошибок в программе**

|  |  |  |
| --- | --- | --- |
| **test.txt** | **res.txt** | **Пояснение** |
| 3 1  1 2 | 0 | m ≠ n - 1, несвязный граф с точкой 3 |
| 4 3  1 2  2 3  1 3 | 0 | Наличие цикла в графе + несвязный граф с точкой 4 |
| 4 3  1 2  2 3  3 4 | 1 | Простое дерево в форме линейного списка |
| 4 3  1 1  1 2  2 3 | 0 | Петля в точке 1 + несвязный граф с точкой 4 |
| 10 9  1 2  2 3  3 4  3 6  2 5  2 7  1 8  8 9  1 10 | 1 | Дерево с ветвлениями в точках 1, 2, 3 |

**Время, затраченное на проверку с обходом в ширину**

|  |  |
| --- | --- |
| **N, кол-во случайных вершин** | **Время, с** |
| 100 | 0,000415 |
| 1000 | 0,007973 |
| 10000 | 0,082696 |
| 100000 | 0,515285 |
| 1000000 | 5,137693 |

**Время, затраченное на проверку с обходом в глубину**

|  |  |
| --- | --- |
| **N, кол-во случайных вершин** | **Время, с** |
| 100 | 0,000736 |
| 1000 | 0,007493 |
| 10000 | 0,073853 |
| 100000 | 0,473859 |
| 1000000 | 4,952129 |

После сравнительного анализа результатов выполненной работы стоит отметить, что время работы программ как с обходом в ширину, так и в глубину, – линейно зависит от N. Так, при увеличении количества случайных вершин в 10 раз, время работы программы соответственно тоже увеличивается в 10 раз. Следовательно, сложность алгоритма программы равно O(N). Это видно и по написанному коду программы: используются только одинарные циклы длительностью N (либо числу, эквивалентному N), то есть функция содержит примерно столько же операций, сколько и всего вершин в дереве. Таким образом, полученная по тестам сложность равна сложности, определенной теоретически.