Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Информационных технологий и управления
Кафедра Интеллектуальных информационных технологий

ОТЧЁТ

по дисциплине «ППОИС»
на тему
Двусвязный граф

Выполнил: И. В. Якимович

Студент группы
121703
Проверил: С. В. Бутрин

Минск 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В теории графов двусвязный граф — это связный и неделимый граф, в том смысле, что удаление любой вершины не приведёт к потере связности. Теорема Уитни утверждает, в частности, что граф двусвязен тогда и только тогда, когда между любыми двумя его вершинами есть минимум два непересекающихся пути. Таким образом, двусвязный граф не имеет шарниров.

Это свойство особенно полезно при рассмотрении графов с двойным резервированием, чтобы избежать разрыва при удалении единственного ребра.

Использование двусвязных графов очень важно в области сетей ввиду их свойств резервирования.

Цель расчетной работы: Реализовать агента для определения двухсвязного графа в графовой памяти системы ostis с помощью C++ API.

Задача: Определить, является ли неориентированный граф двусвязным.

1 ЛИСТИНГ

Код агента:

```
void generateAnswer(const std::unique_ptr<ScMemoryContext>& context, ScAddr
    startNode, ScAddr structNode, bool isBiconnected) {
        ScAddr answer = context->CreateNode(ScType::NodeConst);
        ScAddr edge = context->CreateEdge(ScType::EdgeDCommonConst,
            startNode, answer);
        context->CreateEdge(ScType::EdgeAccessConstPosPerm,
            courseWorkNamespace::Keynodes::nrel_answer, edge);

        context->HelperSetSystemIdtf("'" +
            context->HelperGetSystemIdtf(structNode) + "' is " +
                    (isBiconnected ? "biconnecteed" : "is not biconnected"),
                        answer);
}


vector<ScAddr> getVertexes(const std::unique_ptr<ScMemoryContext>& context,
    ScAddr structNode) {
        vector<ScAddr> allVertexes;
        ScIterator3Ptr it = context->Iterator3(structNode,
            ScType::EdgeAccessConstPosPerm, ScType::NodeConst);

        while (it->Next()) {
              ScAddr node = it->Get(2);
              ScType node_type = context->GetElementType(node);

              if (node_type.IsNode() == ScType::Node) {
                  allVertexes.push_back(node);
              }
        }
        return allVertexes;
}


bool isArticulation(const std::unique_ptr<ScMemoryContext>& context,
    vector<ScAddr>& vertexes, int pos,
                    vector<bool> &visited, vector<int> &parent, vector<int>
                       &disc, vector<int> &low) {
    static int time = 0;
    int dfsChild = 0;
    visited[pos] = true;
    disc[pos] = low[pos] = ++time;

    int a;

    for (int v_pos = 0; v_pos < vertexes.size(); v_pos++) {
      if (context->HelperCheckEdge(vertexes[pos],vertexes[v_pos], ScType(0)))
          {
        if (!visited[v_pos]) {
          dfsChild++;
          parent[v_pos] = pos;
```

```
41
42              if(isArticulation(context, vertexes, v_pos, visited, parent, disc,
                    low))
43                return true;
44
45              low[pos] = (low[pos] < low[v_pos]) ? low[pos] : low[v_pos];
46
47              if (parent[pos] == -1 && dfsChild > 1) {
48                return true;
49              }
50
51              if(parent[pos] != -1 && low[v_pos] >= disc[pos])
52                return true;
53
54            } else if (v_pos != parent[pos]) {
55              low[pos] = (low[pos] < disc[v_pos]) ? low[pos] : disc[v_pos];
56            }
57          }
58        }
59        return false;
60    }
61
62
63    bool isBiconnected(const std::unique_ptr<ScMemoryContext>& context, ScAddr
        structNode) {
64        vector<ScAddr> vertexes = getVertexes(context, structNode);
65
66        SC_LOG_DEBUG("graph: " + context->HelperGetSystemIdtf(structNode));
67        SC_LOG_DEBUG("count: " + to_string(vertexes.size()));
68
69        vector<bool> visited(vertexes.size(), false);
70        vector<int> parent(vertexes.size(), -1);
71
72        vector<int> disc(vertexes.size());
73        vector<int> low(vertexes.size());
74
75        if(isArticulation(context, vertexes, 0, visited, parent, disc, low)) {
76          SC_LOG_COLOR(::utils::ScLog::Type::Debug, "IS NOT BICONNECTED",
                ScConsole::Color::Magneta);
77          return false;
78        }
79
80        for (auto v : visited) {
81          if (!v) {
82            SC_LOG_COLOR(::utils::ScLog::Type::Debug, "IS NOT BICONNECTED",
                    ScConsole::Color::Magneta);
83            return false;
84          }
85        }
86
87
88        SC_LOG_COLOR(::utils::ScLog::Type::Debug, "IS BICONNECTED",
            ScConsole::Color::Magneta);
89        return true;
90    }
```

```
93  SC_AGENT_IMPLEMENTATION(ASearchBiconnectedGraph)
94  {
95      if (!edgeAddr.IsValid())
96        return SC_RESULT_ERROR;
97
98      SC_LOG_DEBUG("INIT");
99
100     ScAddr start = ms_context->GetEdgeTarget(edgeAddr);;
101     ScAddr graph;
102
103     ScIterator3Ptr iter = ms_context->Iterator3(start,
            ScType::EdgeAccessConstPosPerm, ScType::NodeConst);
104
105     if (iter->Next())
106       graph = iter->Get(2);
107     else
108       return SC_RESULT_ERROR_INVALID_PARAMS;
109
110     generateAnswer(ms_context, start, graph, isBiconnected(ms_context,
            graph));
111
112     SC_LOG_DEBUG("DEINIT");
113     return SC_RESULT_OK;
114 }
```

2 ТЕСТОВЫЕ ПРИМЕРЫ

Во всех тестах графы будет приведены в сокращенной форме со скрытыми ролями элементов графа.

Тест 1

Вход:

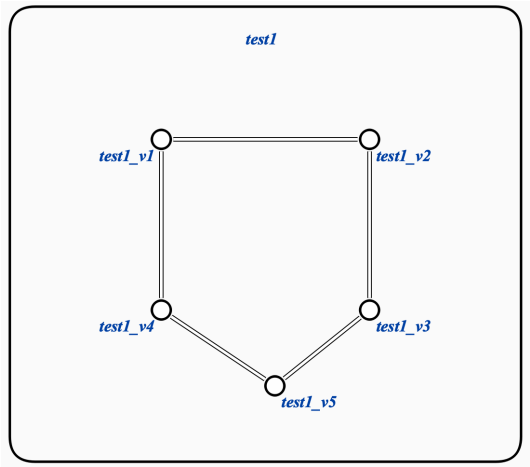

Рисунок *2.1* – Вход теста 1

Выход:

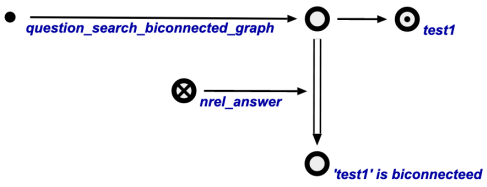

Рисунок *2.2* – Выход теста 1

Тест 2

Вход:

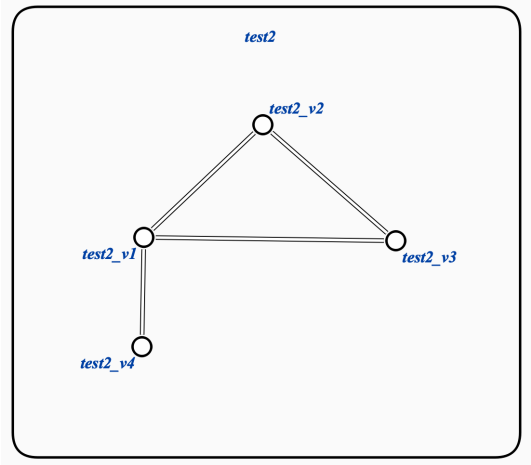

Рисунок *2.3* – Вход теста 2

Выход:

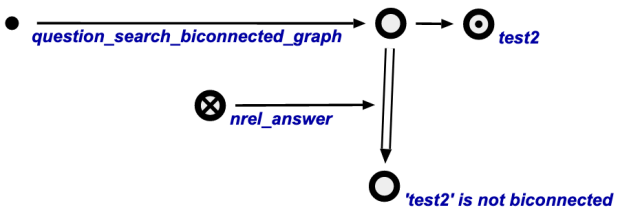

Рисунок *2.4* – Выход теста 2

Тест 3

Вход:

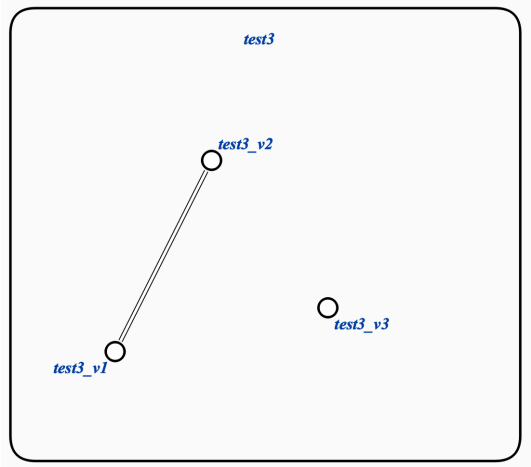

Рисунок *2.5* – Вход теста 3

Выход:

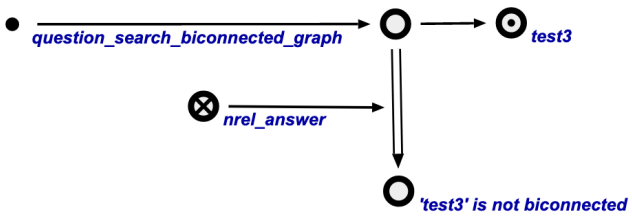

Рисунок *2.6* – Выход теста 3

Тест 4

Вход:

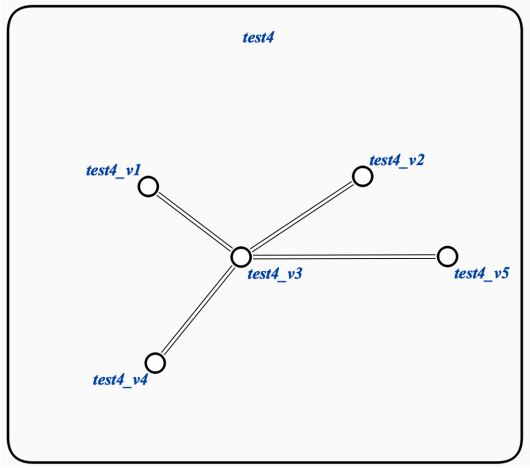

Рисунок *2.7* – Вход теста 4

Выход:

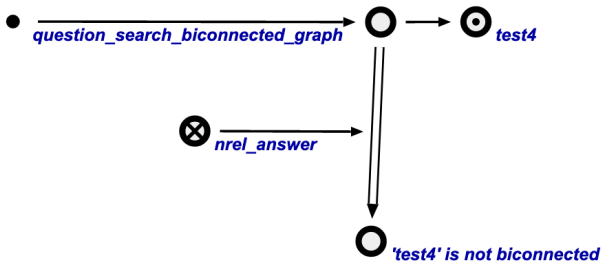

Рисунок *2.8* – Выход теста 4

Тест 5

Вход:

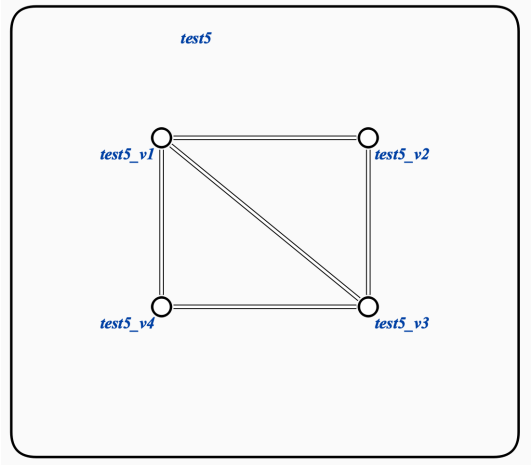

Рисунок *2.9* – Вход теста 5

Выход:

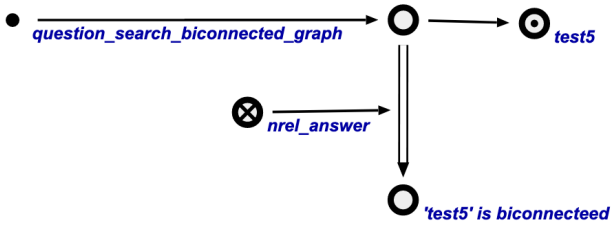

Рисунок *2.10* – Выход теста 5

ЗАКЛЮЧЕНИЕ

В ходе данной расчетной работе мы получили навыки формализации и обработки с использованием семантических сетей. А также научились работать с C++ API системы ostis. Реализовали агента для поиска двухсвязных графов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] База знаний по теории графов OSTIS GT [Электронный ресурс] / проект OSTIS, 2022. – Режим доступа:. http://ostisgraphstheo.sourceforge.net.

[2] Лазуркин, Д.А. Руководство к выполнению расчетной работы по курсам ОИИ и ППвИС / Д.А. Лазуркин. — 2013. — P. 126.

[3] Оре, О. Теория графов / О. Оре. — Наука, 1980. — P. 336.