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: Е. Д. Васильков

Студент группы
121703
Проверил: С. В. Бутрин

Минск 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Двусвязный неориентированый граф — это связный граф, в котором отсутвуют точки сочленения и удаление любой вершины не приводит к потере связности. Граф называется связным, если между каждой парой вершин существует ребро. Вершина в неориентированном связном графе является точкой сочленения, если ее удаление разъединяет граф. По соглашению две вершины, соединенные ребром, образуют двусвязный граф. Для графа с более чем двумя вершинами указанные выше свойства должны присутствовать, чтобы он был двусвязным.

Цель расчетной работы: Реализовать агента для определения двух-связного графа с помощью C++ API.

Задача: Определить, является ли неориентированный граф двусвязным.

1 ЛИСТИНГ

```
bool isArticulationExist(ScMemoryContext *ms_context, vector<ScAddr>& vertexes, int current,
                         vector<bool> &v, vector<int> &p, vector<int> &d, vector<int> &l) {
    v[current] = true;
    int dfsChild = 0;
    d[current] = l[current] = ++counter;

    for (int v_current = 0; v_current < vertexes.size(); v_current++) {
      if (ms_context->HelperCheckEdge(vertexes[current], vertexes[v_current], ScType(0)) ||
          ms_context->HelperCheckEdge(vertexes[v_current], vertexes[current], ScType(0))) {
        if (!v[v_current]) {
          dfsChild++;
          p[v_current] = current;

          if(isArticulationExist(ms_context, vertexes, v_current, v, p, d, l))
            return true;

          l[current] = (l[current] < l[v_current]) ? l[current] : l[v_current];

          if (p[current] == -1 && dfsChild > 1) {
            return true;
          }
          if(p[current] != -1 && l[v_current] >= d[current])
            return true;
        } else if (v_current != p[current]) {
          l[current] = (l[current] < d[v_current]) ? l[current] : d[v_current];
        }
      }
    }
    return false;
}

static int counter = 0;

bool isBiconnected(ScMemoryContext *ms_context, ScAddr graph) {
    vector<ScAddr> vertexes = IteratorUtils::getAllWithType(&(*ms_context), graph, ScType::NodeConst);

    vector<bool> v(vertexes.size(), false);
    vector<int> p(vertexes.size(), -1);
    vector<int> l(vertexes.size());
    vector<int> d(vertexes.size());

    if(vertexes.size() <= 2 || isArticulationExist(ms_context, vertexes, 0, v, p, d, l)) {
      return false;
    }
```

```
44
45      for (auto i : v) {
46        if (!i) {
47          return false;
48        }
49      }
50
51      return true;
52  }
53
54
55  SC_AGENT_IMPLEMENTATION(ASearchBiconnectedGraph)
56  {
57      if (!edgeAddr.IsValid())
58        return SC_RESULT_ERROR;
59
60      ScAddr question = ms_context->GetEdgeTarget(edgeAddr);;
61      ScAddr structure;
62
63      ScIterator3Ptr iter = ms_context->Iterator3(question,
            ScType::EdgeAccessConstPosPerm, ScType::NodeConst);
64
65      if (iter->Next())
66        structure = iter->Get(2);
67      else
68        return SC_RESULT_ERROR_INVALID_PARAMS;
69
70      ScAddr node = ms_context->CreateNode(ScType::NodeConst);
71      string graphIdtf = ms_context->HelperGetSystemIdtf(structure);
72
73      if(isBiconnected(&(*ms_context), structure)) {
74        ms_context->HelperSetSystemIdtf("graph '" + graphIdtf + "' is
              biconnected", node);
75        SC_LOG_COLOR(ScLog::Type::Debug, "graph '" + graphIdtf + "' is
              biconnected", ScConsole::Color::Blue);
76      } else {
77        ms_context->HelperSetSystemIdtf("graph '" + graphIdtf + "' is not
              biconnected", node);
78        SC_LOG_COLOR(ScLog::Type::Debug, "graph '" + graphIdtf + "' is not
              biconnected", ScConsole::Color::Red);
79      }
80
81      vector<ScAddr> answer = {node};
82      AgentUtils::finishAgentWork(&(*ms_context), question, answer, true);
83
84      return SC_RESULT_OK;
85  }
```

2 ТЕСТОВЫЕ ПРИМЕРЫ

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

Тест 1

Вход:

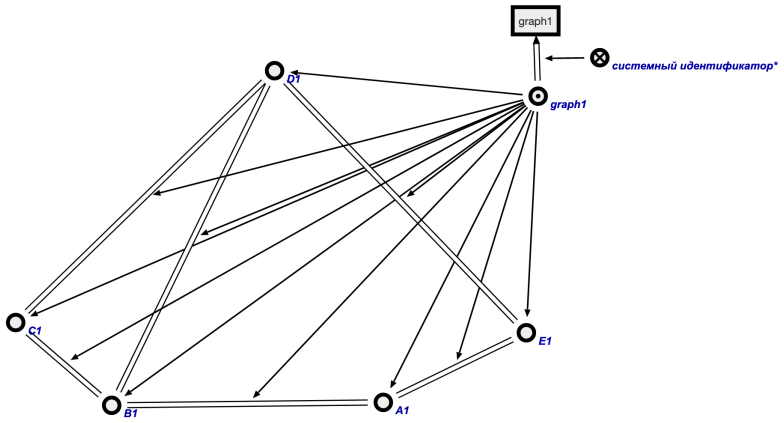

Рисунок *2.1* – Вход теста 1

Выход:

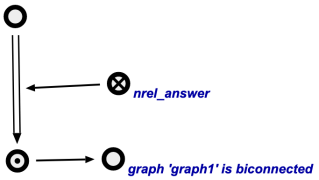

Рисунок *2.2* – Выход теста 1

Тест 2

Вход:

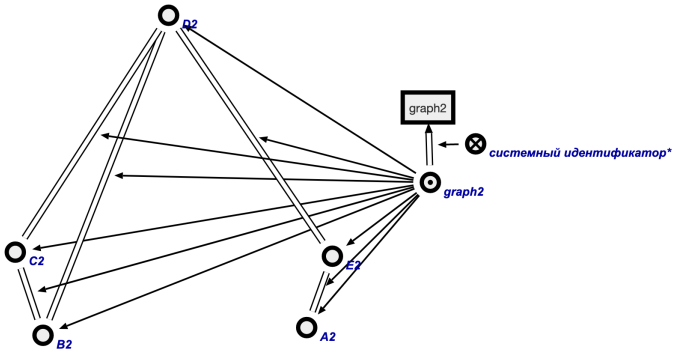

Рисунок *2.3* – Вход теста 2

Выход:

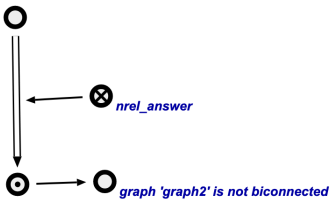

Рисунок *2.4* – Выход теста 2

Тест 3

Вход:

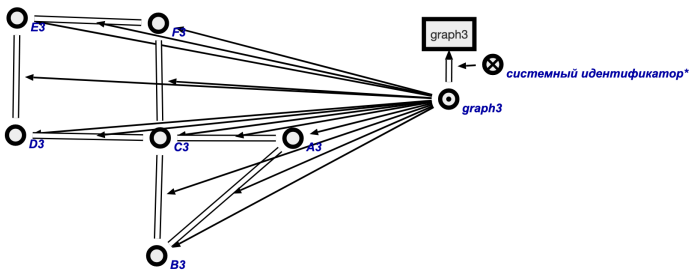

Рисунок *2.5* – Вход теста 3

Выход:

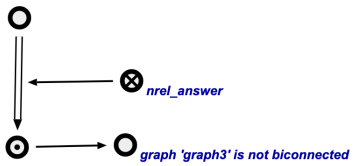

Рисунок *2.6* – Выход теста 3

Тест 4

Вход:

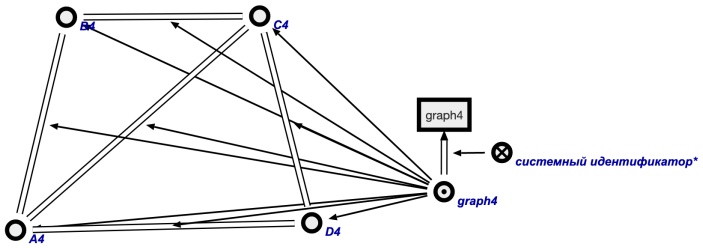

Рисунок *2.7* – Вход теста 4

Выход:

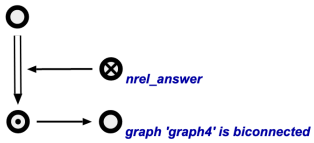

Рисунок *2.8* – Выход теста 4

Тест 5

Вход:

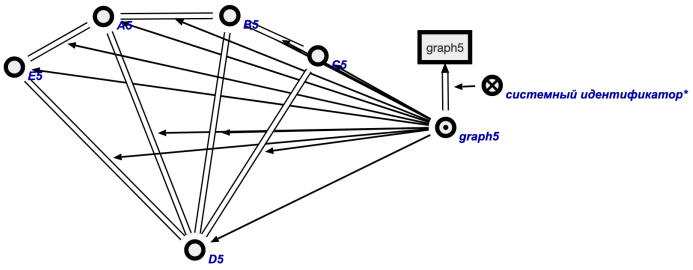

Рисунок *2.9* – Вход теста 5

Выход:

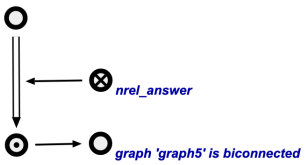

Рисунок *2.10* – Выход теста 5

ЗАКЛЮЧЕНИЕ

В заключении отчета сделаем краткие выводы по результатам проделанной работы:

– Получили навыки формализации и обработки информации с использованием семантических сетей.В частности формализавали различные типы графовых структур.

– Рассмотрели актуальную задачу, является ли неориентированный граф двусвязным.

– Получили навыки работы 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.