Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ
Факультет информационных технологий и управления
Кафедра интеллектуальных информационных технологий

РАСЧЕТНАЯ РАБОТА

по дисциплине «Представление и обработка информации в интеллектуальных системах»
на тему «Задача поиска графа замыкания неориентированного графа »

Выполнил: Витковская С. И.

Студент группы
121702
Проверил: Бутрин С. В.

Минск
2022

Содержание

Постановка задачи

Задача поиска графа замыкания неориентированного графа.

Цель

Получить навыки формализации и обработки информации с использованием семантических сетей.

1. Список понятий

1.1. Графовая структура

Это такая одноуровневая реляционная структура, объекты которой могут играть роль либо вершины, либо связки:

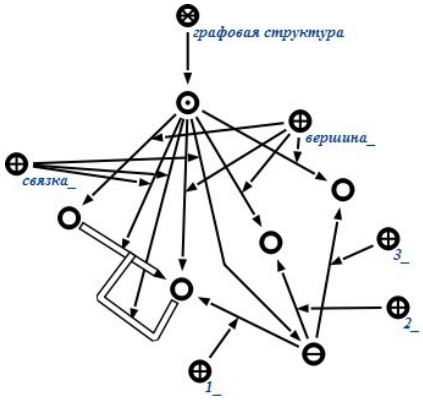

Рисунок 1.1. Графовая структура

1.2. Граф

Математическая абстракция реальной системы объектов любой природы, обладающих парными связями.

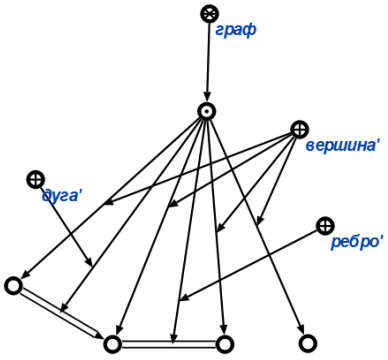

Рисунок 1.2. Граф

1.3. Неориентированный граф

Граф, ни одному ребру которого не присвоено направление.

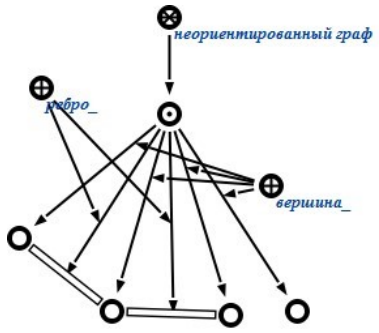

Рисунок 1.3. Неориентированный граф

1.4. Ребро

Линия, соединяющая пару смежных вершин графа.

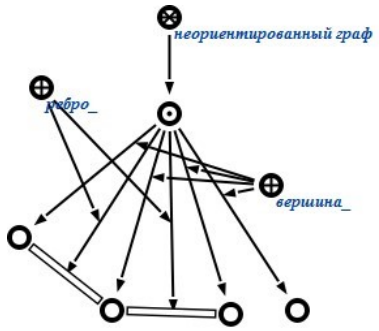

Рисунок 1.4. Ребро

1.5. Замыкание

Операция удаления пары транзитивных вершин и замена их новой.

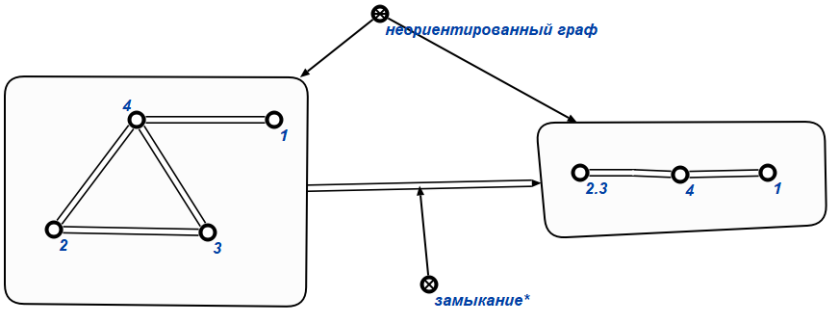

Рисунок 1.5. Замыкание

1.6. Транзитивные вершины

Вершины называются транзитивными, если из наличия ребер из x в y и из y в z, следует наличие ребра из x в z.

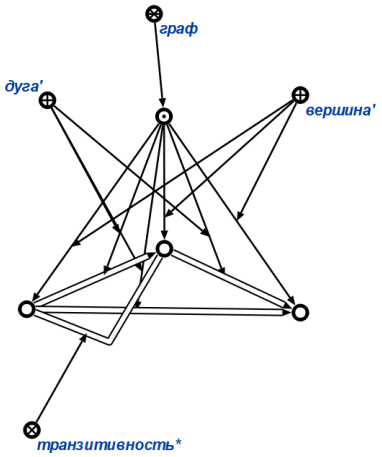

Рисунок 1.6. Транзитивные вершины

2. Алгоритм

```cpp
#include <sc-memory/sc_memory.hpp>
#include <sc-memory/sc_link.hpp>
#include <sc-memory/sc_common_templ.hpp>

#include <sc-agents-common/utils/GenerationUtils.hpp>
#include <sc-agents-common/utils/AgentUtils.hpp>
#include <sc-agents-common/utils/IteratorUtils.hpp>
#include <sc-agents-common/utils/CommonUtils.hpp>

#include "CourseWorkAgent.hpp"

using namespace std;
using namespace utils;

namespace exampleModule
{

std::string GetStringNodeIdtf(std::unique_ptr<ScMemoryContext> &ms_context, ScAddr node)
{

    ScIterator3Ptr it3 = ms_context->Iterator3(node, ScType::EdgeDCommonConst, ScType::LinkConst);
    while(it3->Next())
    {
        return CommonUtils::getLinkContent(ms_context.get(), it3->Get(2));
    }
    return CommonUtils::getLinkContent(ms_context.get(), it3->Get(2));
}

SC_AGENT_IMPLEMENTATION(TransitiveAddition)
{

    ScLog *logger = ScLog::GetInstance();
    ScAddr questionNode = ms_context->GetEdgeTarget(edgeAddr);
    ScAddr param = IteratorUtils::getAnyFromSet(ms_context.get(), questionNode);
```

```
ScAddr pair_nodes = ms_context->CreateNode(ScType::NodeConstClass);
ScAddr pair_nodes2 = ms_context->CreateNode(ScType::NodeConstClass);
ScIterator3Ptr it3 = ms_context->Iterator3(param, ScType::EdgeAccessConstPosPerm, ScType::NodeConst);
while(it3->Next()){

    ScIterator3Ptr it5 = ms_context->Iterator3(param, ScType::EdgeAccessConstPosPerm, ScType::NodeConst);
    int length_av2;int length_av1;
    ScAddr available_2;
    ScAddr available;

    while(it5->Next())
    {
    if (ms_context->HelperCheckEdge(it5->Get(2), it3->Get(2), ScType::EdgeDCommonConst))
    {

        available = ms_context->CreateNode(ScType::NodeConstClass);
        ScIterator3Ptr it4 = ms_context->Iterator3(param, ScType::EdgeAccessConstPosPerm, ScType::NodeConst);
        length_av1 = 0;
        while(it4->Next())
        {
        if (ms_context->HelperCheckEdge(it3->Get(2), it4->Get(2), ScType::EdgeDCommonConst))
        {
            ms_context->CreateEdge(ScType::EdgeAccessConstPosPerm, available, it4->Get(2));
            length_av1++;
        }}


        available_2 = ms_context->CreateNode(ScType::NodeConstClass);
        ScIterator3Ptr it6 = ms_context->Iterator3(param, ScType::EdgeAccessConstPosPerm, ScType::NodeConst);
        length_av2 = 0;
        while(it6->Next())
        {
```

```
        {
            if ((ms_context->HelperCheckEdge(it5->Get(2), it6->Get(2), ScType::EdgeDCommonConst))) {
                ms_context->CreateEdge(ScType::EdgeAccessConstPosPerm, available_2, it6->Get(2));
                length_av2++;
            }
        }

    ms_context->CreateEdge(ScType::EdgeAccessConstPosPerm, pair_nodes, it5->Get(2));
    ScIterator3Ptr it5_del = ms_context->Iterator3(available, ScType::EdgeAccessConstPosPerm, ScType::NodeConst);
    while(it5_del->Next())
    {
    if (it5_del->Get(2) == it5->Get(2)){
        ms_context->EraseElement(it5_del->Get(1));
        length_av1--;
    }
    }


    ms_context->CreateEdge(ScType::EdgeAccessConstPosPerm, pair_nodes2, it3->Get(2));
    ScIterator3Ptr it3_del = ms_context->Iterator3(available_2, ScType::EdgeAccessConstPosPerm, ScType::NodeConst);
    while(it3_del->Next()){
    if (it3_del->Get(2) == it3->Get(2)){
        ms_context->EraseElement(it3_del->Get(1));
        length_av2--;
    }
    }

    if(length_av1 == length_av2)
    {
        bool av1_av2 ;
        bool av2_av1 ;
```

```
    int exist1=0;
    int exist2=0;

    ScIterator3Ptr from_av1 = ms_context->Iterator3(available, ScType::EdgeAccessConstPosPerm, ScType::NodeConst);
    while(from_av1->Next())
    {
        av1_av2 = 0;
        ScIterator3Ptr from_av2 = ms_context->Iterator3(available_2, ScType::EdgeAccessConstPosPerm, ScType::NodeConst);
        while(from_av2->Next())
        {
        if(from_av1->Get(2) == from_av2->Get(2))
        {av1_av2 = 1;}
        }
        if(av1_av2){exist1++;}

    }

    ScIterator3Ptr from_av2_2 = ms_context->Iterator3(available_2, ScType::EdgeAccessConstPosPerm, ScType::NodeConst);
    while(from_av2_2->Next())
    {
        av2_av1 = 0;
        ScIterator3Ptr from_av1_2 = ms_context->Iterator3(available, ScType::EdgeAccessConstPosPerm, ScType::NodeConst);
        while(from_av1_2->Next())
        {
        if(from_av1_2->Get(2) == from_av2_2->Get(2))
        {av2_av1 = 1;}
        }
        if(av2_av1){exist2++;}
    }

    if((exist1 == length_av1) && (exist2 == length_av1))
    {
        ScAddr visited = ms_context->CreateNode(ScType::NodeConstClass);

        ScAddr new_n = ms_context->CreateNode(ScType::NodeConstClass);
        ScAddr NewNode = ms_context->CreateNode(ScType::NodeConst);
        ms_context->CreateEdge(ScType::EdgeAccessConstPosPerm, new_n, NewNode);
```

```
            ms_context->CreateEdge(ScType::EdgeAccessConstPosPerm, new_n, NewNode);
            ms_context->CreateEdge(ScType::EdgeAccessConstPosPerm, param, NewNode);
            ScIterator3Ptr temp_new = ms_context->Iterator3(new_n, ScType::EdgeAccessConstPosPerm, ScType::NodeConst);
            ScIterator3Ptr from_av = ms_context->Iterator3(available_2, ScType::EdgeAccessConstPosPerm, ScType::NodeConst);
            while(from_av->Next())
            {
                ms_context->CreateEdge(ScType::EdgeDCommonConst, from_av->Get(2), temp_new->Get(2));
                ms_context->CreateEdge(ScType::EdgeDCommonConst, temp_new->Get(2), from_av->Get(2));
            }
            ms_context->CreateEdge(ScType::EdgeAccessConstPosPerm, visited, NewNode);
            std::string ans = "";
            ans += GetStringNodeIdtf(ms_context, it3->Get(2)) + "  " + GetStringNodeIdtf(ms_context, it5->Get(2));
            ScIterator3Ptr new_it3 = it3;
            it3 = ms_context->Iterator3(visited, ScType::EdgeAccessConstPosPerm, ScType::NodeConst);
            logger->Message(ScLog::Type::Info, "Merged transitive vertices {" + ans + "}");
            ms_context->EraseElement(it3->Get(2));
            ms_context->EraseElement(it5->Get(2));
        }
        else{SC_LOG_ERROR( "netransitiv");}
    }

    }
    }
}
utils::AgentUtils::finishAgentWork(ms_context.get(), questionNode, param);
return SC_RESULT_OK;
```

Описание алгоритма:

Примечание: для удобства, вместо неориентированных ребер, проведем две ориентированные дуги

1. Выбираем случайную вершину графа;
2. Выбираем вторую случайную вершину графа;
   2.1 Если существует дуга между первой и второй вершиной, переходим к п.3;
   2.2 Переходим к п.2;
3. Заносим все вершины, смежные с первой вершиной, кроме второй вершины, в класс available;
4. Заносим все вершины, смежные со второй вершиной,кроме первой вершины, в класс available_2;
5. Сравниваем количество элементов классов available и available_2 ;
   5.1 Если они равны, переходим к п.6;
   5.2 Переходим к п.2;
6. Проверяем наличие все вершин из класса available в классе available_2;
   6.1 Если для любой вершины из класса available можно найти соответсвующую в

классе available_2, переходим к п. 7;

6.2 Переходим к п.2;

7. Проверяем наличие все вершин из класса available_2 в классе available;

7.1 Если для любой вершины из класса available_2 можно найти соответсвующую в классе available, переходим к п. 8;

7.2 Переходим к п.2;

8. Создаем новую вершину;

9. Удаляем первую и вторую вершины;

10. Проводим дуги от новой вершины к вершинам из класса available и обратно;

11. Проверяем, существуют ли еще непроверенные вершины;

11.1 Если да, переходим к п. 1;

11.2 Переходим к п.12;

12.Возвращение результата. Конец алгоритма.

3. Тестовые примеры

3.1. Тест 1

Вход: *Необходимо найти граф замыкания неориентированного графа.*

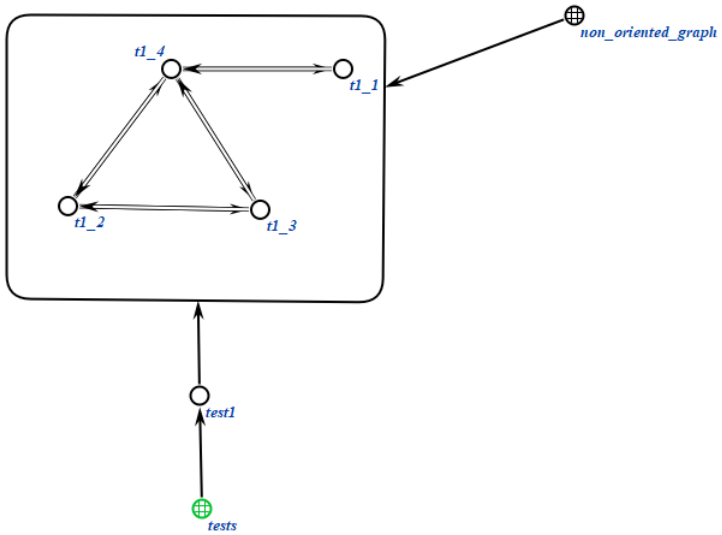

Рисунок 3.1. Вход теста 1

Шаг 1: Создаем итератор it3, который захвватывает случайную вершину графа, пусть это будет вершина t1_2.

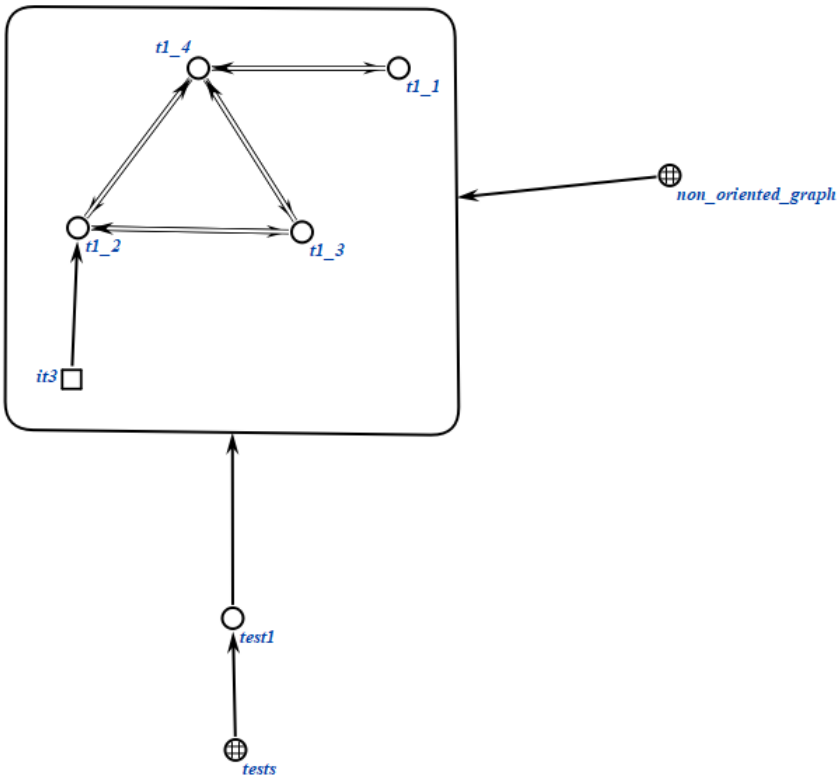

Рисунок 3.2. Шаг 1

Шаг 2: Создаем итератор it5, который захватывает случайную вершину графа, пусть это будет вершина t1_3.

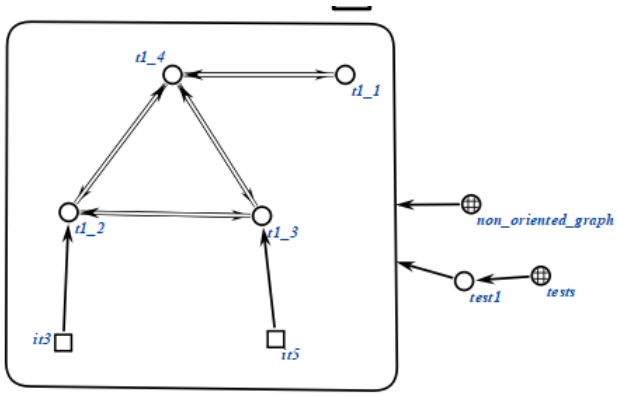

Рисунок 3.3. Шаг 2

Шаг 3: Между вершинами t1_3 и t1_2 есть дуги, соответсвенно условие смежности выполняется, поэтому создаем классы available и available_2 и переменные length и length_2, которые сразу инициализируем нулями.

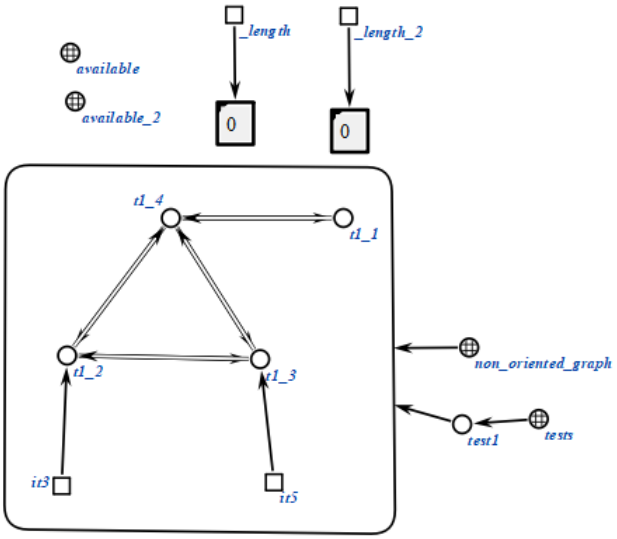

Рисунок 3.4. Шаг 3

Шаг 4: Заносим вершины, смежные с вершиной t1_2, в класс available и увеличиваем переменную length на соответствующее значение, а вершины, смежные с вершиной t1_3 заносим в класс available_2 и увеличиваем переменную length_2 на соответствующее значение.

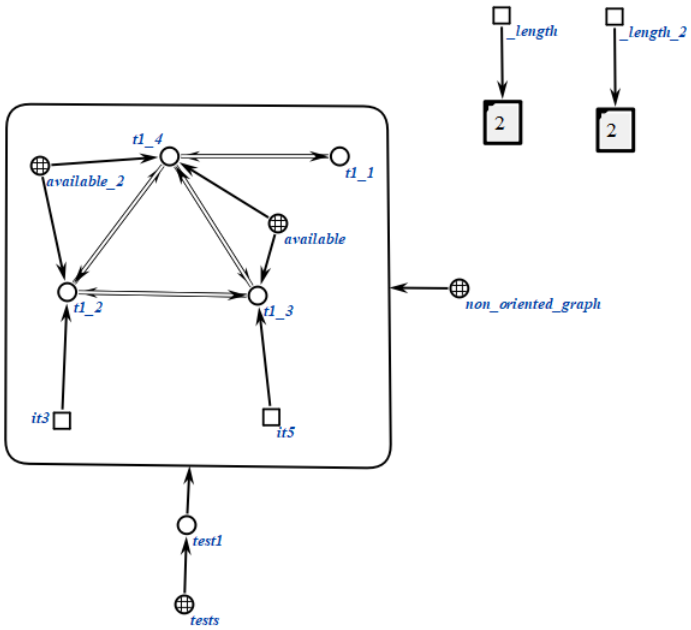

Рисунок 3.5. Шаг 4

Шаг 5: Удаляем вершину t1_2 из класса available_2 и вершину t1_3 из класса available. И соответсвенно уменьшаем переменные length и length_2 на 1.

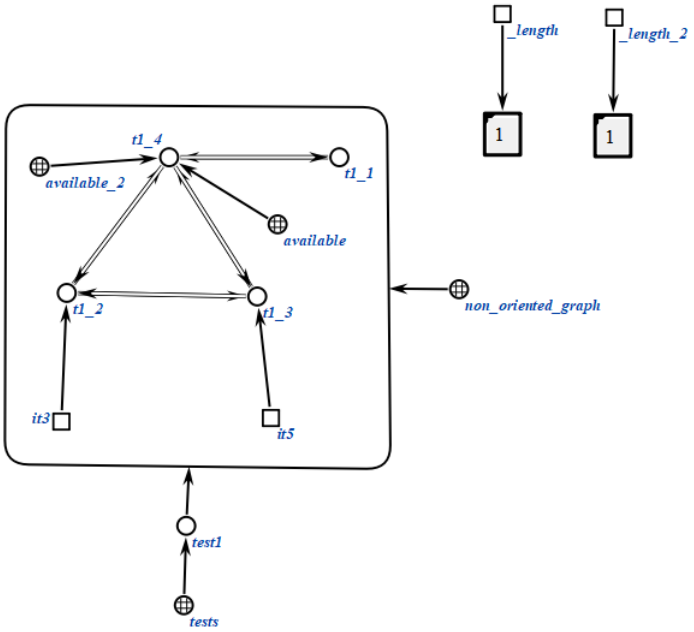

Рисунок 3.6. Шаг 5

Шаг 6: Т.к. переменные length и length_2 совпадают, так же, как и содержимое классов available и available_2, создаем новую вершину t1_2_t1_3.

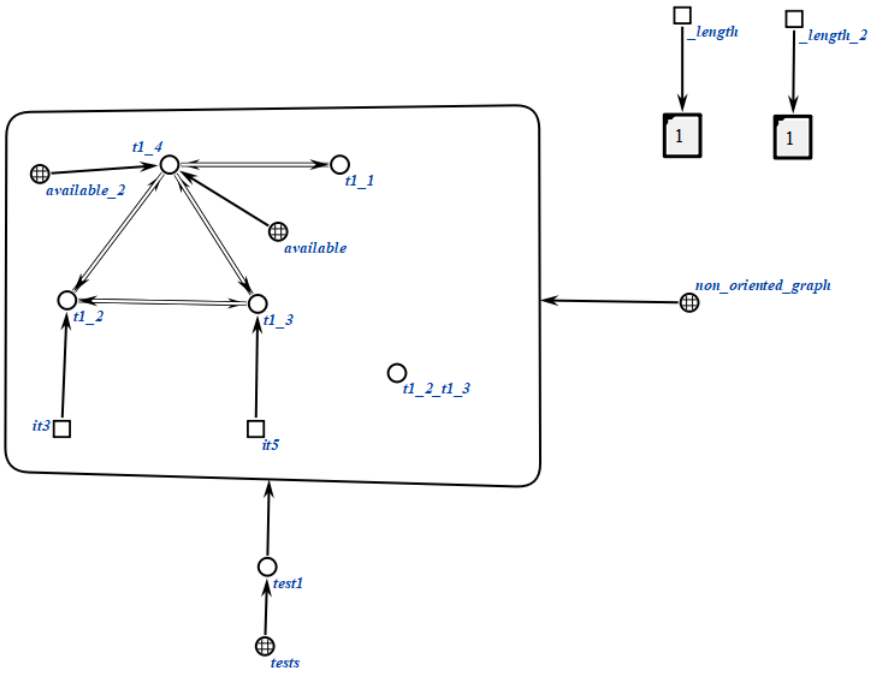

Рисунок 3.7. Шаг 6

Шаг 7: Переставляем итератор it3 на новую вершину и удаляем вершины t1_3 и t1_2,

проводим дуги от вершин, содержащихся в классе available к новой вершине и обратно.

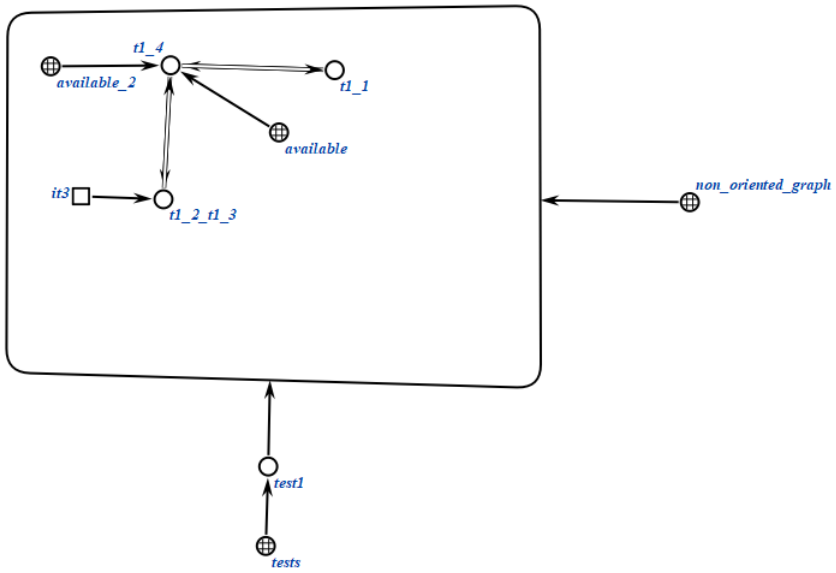

Рисунок 3.8. Шаг 7

Шаг 9: В консоль выводятся слитые вершины.

```
[22:15:29][Info]: Merged transitive vertices {t1_2  t1_3}
```

Рисунок 3.9. Вывод теста 1

3.2. Тест 2

Вход: *Необходимо найти граф замыкания неориентированного графа.*

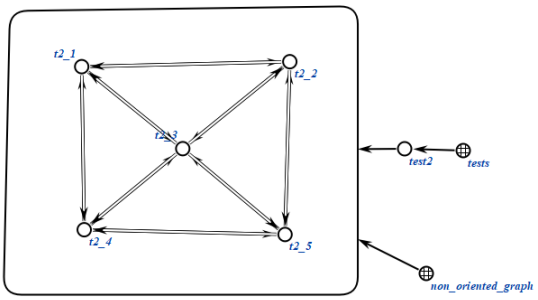

Рисунок 3.10. Вход теста 2

Выход: *В графе нет ни одной пары транзитивных вершин.*

```
[22:16:12][Error]: netransitiv
```

Рисунок 3.11. Выход теста 2

3.3. Тест 3

Вход: *Необходимо найти граф замыкания неориентированного графа.*

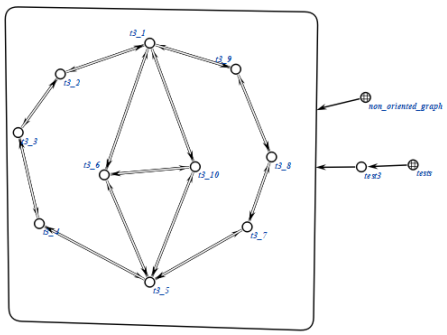

Рисунок 3.12. Вход теста 3

Выход: *Будут представлены слитые вершины:*

```
[22:16:41][Info]: Merged transitive vertices {t3_6  t3_10}
```

Рисунок 3.13. Выход теста 3

3.4. Тест 4

Вход: *Необходимо найти граф замыкания неориентированного графа.*

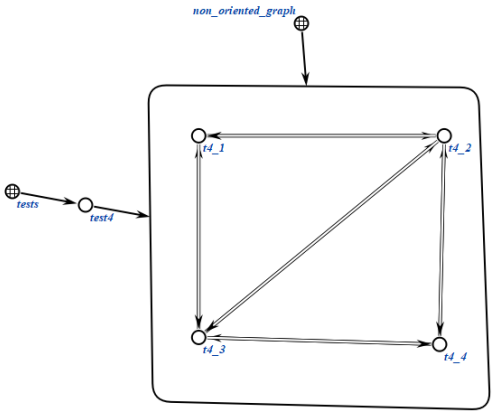

Рисунок 3.14. Вход теста 4

Выход: *Будут представлены слитые вершины:*

```
[22:17:03][Info]: Merged transitive vertices {t4_3  t4_2}
```

Рисунок 3.15. Выход теста 4

3.5. Тест 5

Вход: *Необходимо найти граф замыкания неориентированного графа.*

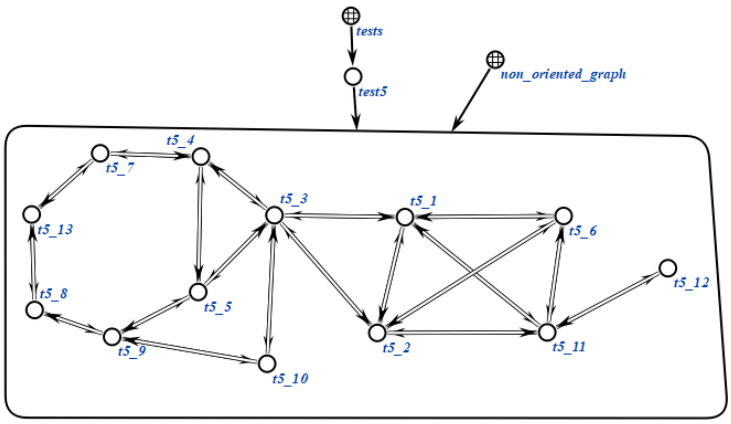

Рисунок 3.16. Вход теста 5

Выход: *Будут представлены слитые вершины:*

```
[22:14:58][Info]: Merged transitive vertices {t5_1  t5_2}
```

Рисунок 3.17. Выход теста 5

Вывод

В ходе выполнения работы изучили понятия графа, ребра, графовой структуры, неориентированного графа, операции транзитивного замыкания. Научились работать с базами знаний, создавать агентов в OSTIS. Рассмотрели алгоритм выполнения операции транзитивного замыкания и научились выполнять данную операцию на конкретном примере.

Список литературы

[1] OSTIS GT. База знаний по теории графов OSTIS GT. - 2011. [Электронный ресурс].- Режим доступа: http://ostisgraphstheo.sourceforge.net/index.php.- Дата доступа - 15.03.2022.

[2] Гладков Л.А, Курейчик В. В., Курейчик В. М. Дискретная математика: Теория множеств, алгоритмов, алгебры логики: учебное пособие/ Под ред. В.М. Курейчика. - Таганрог: Изд-во ТТИ ЮФУ, 2009. - 312 с.