Министерство образования Республики Беларусь

Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЕНИЯ

Кафедра интеллектуальных информационных технологий

Отчет

По дисциплине: Проектирование программного обеспечения интеллектуальных систем
На тему: Проектирование программ, ориентированных на обработку знаний в семантической памяти

Выполнил: Заломов Р.А. , 121702

Проверил: Бутрин С.В.

Минск 2022

Задача: найти пересечение множества неориентированных графов

Используемые понятия:

Неориентированный граф G(V, E) – совокупность двух множеств – непустого множества V и множества E неупорядоченных пар различных элементов множества V. Множество V называется множеством вершин графа, множество E называется множеством рёбер графа.

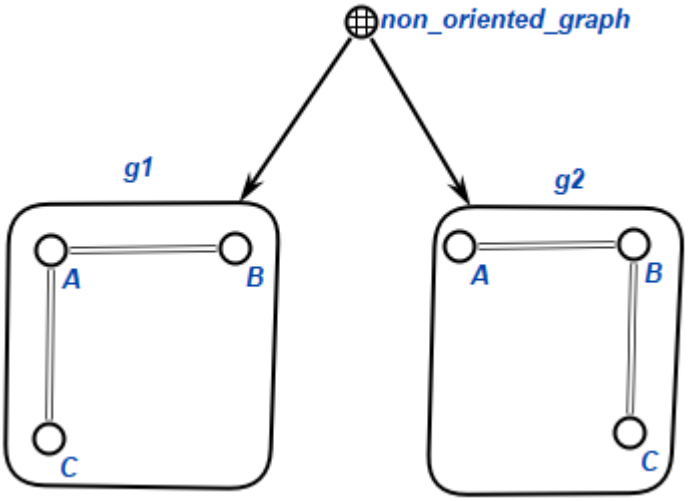
Пересечение графов – операция над графами, результатом которой является граф, множество вершин которого есть пересечение множеств вершин всех исходных графов и множество рёбер которого есть пересечение множеств рёбер всех исходных графов.

Алгоритм работы агента:

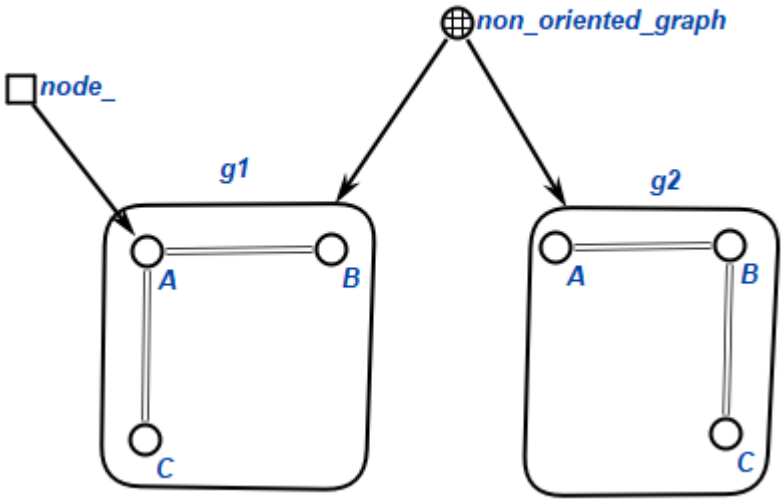
1. Берём граф из множества входных графов. Относительно его и будет формироваться результирующий граф
2. Для каждой вершины из взятого графа проверяем, присутствует ли она в каждом из входных графов. Если да, то подобная вершина будет одной из вершин выходного графа.
3. Для каждой вершины взятого ранее графа проверяем все инцидентные ей рёбра. Если какое-то из таких рёбер присутствует в каждом из графов, то оно будет в результирующем графе.

Пример работы алгоритма (не учитывая особенности C++ API):

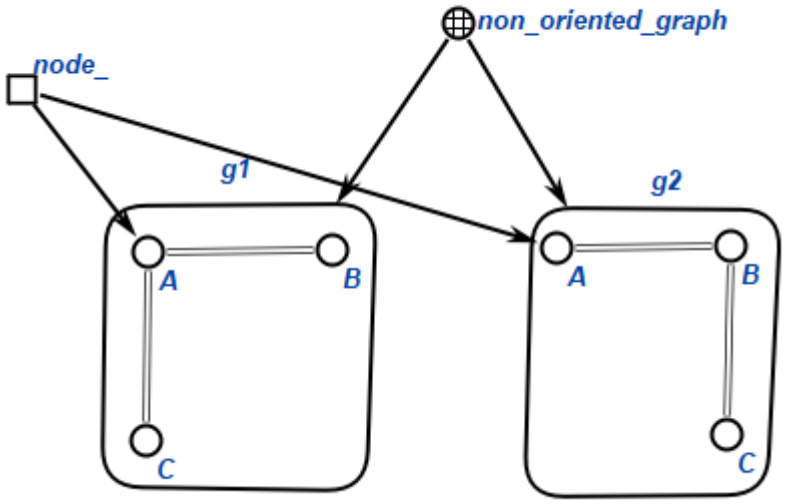
1. Даны следующие два неориентированных графа: g1 и g2



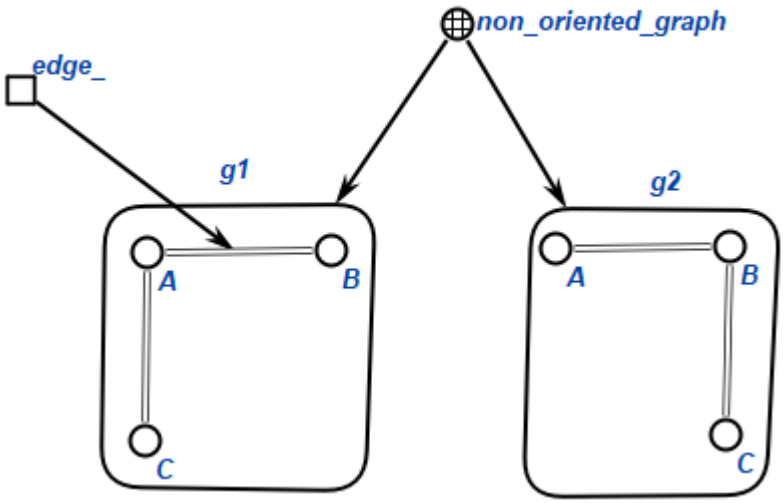
2. Формировать выходной граф будем на основе графа g1. Найдём все вершины, общие для обоих графов. Начнём с вершины A



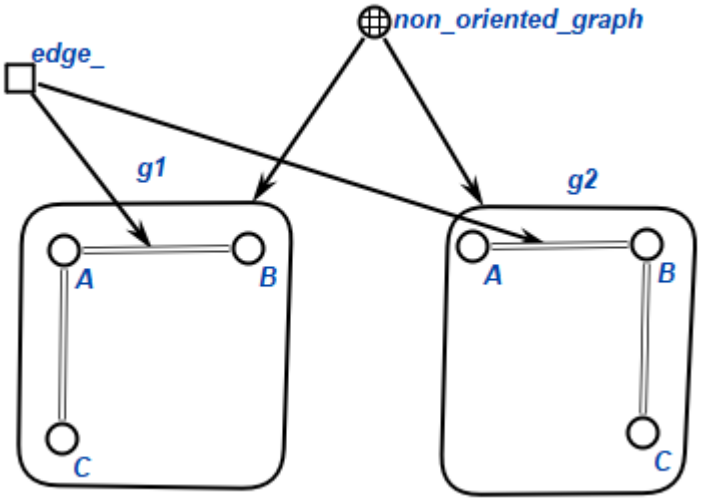
Как видно, она присутствует в обоих графах.



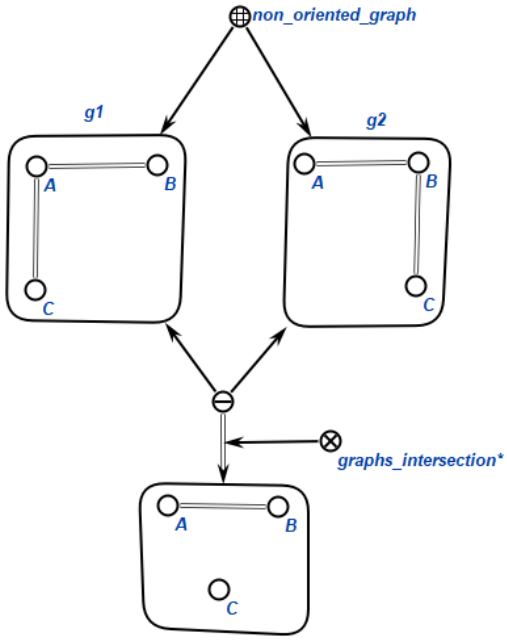
3. Повторяя подобное для всех вершин, получим, что в выходной граф попадут вершины A, B, C.
4. Перейдём к рёбрам. Возьмём одно из рёбер, инцидентных ребру A



Как видно, это ребро присутствует во всех графах



5. Повторяя это выше описанное действие для всех вершин, мы получим, что в результирующий граф войдёт только одно ребро – ребро между вершинами A и B.
6. Результат:



Особенности реализации агента на C++ API:

1. Т.к. итераторы на данный момент не способны работать с неориентированными дугами, каждая из неориентированных дуг графов заменена на две ориентированные.
2. Т.к. на данный момент операция равенства над элементами не реализована, равенством элементов (вершин, дуг) будем называть равенство их идентификаторов (для вершин) и равенство идентификаторов входных и выходных вершин (для дуг).

Листинг агента:

Функция get_link_context для получения идентификатора элемента

```
std::string get_link_context(std::unique_ptr<ScMemoryContext> &ms_context, ScAddr element){
  ScIterator3Ptr link_itr = ms_context->Iterator3(element, ScType::EdgeDCommonConst, ScType::LinkConst);
  std::string content;
  while(link_itr->Next()){
    content = CommonUtils::getLinkContent(ms_context.get(), link_itr->Get(2));
  }
  return content;
}
```

Функция is_node_represented_in_graph для проверки присутствия вершины в графе

```
bool is_node_represented_in_graph(std::unique_ptr<ScMemoryContext> &ms_context, ScAddr &graph, ScAddr &node){
  ScIterator3Ptr graph_itr = ms_context->Iterator3(graph, ScType::EdgeAccessConstPosPerm, ScType::NodeConst);
  while(graph_itr->Next()){
    if(get_link_context(ms_context, graph_itr->Get(2)) == get_link_context(ms_context, node)) return true;
  }
  return false;
}
```

Функция is_edge_represented_in_graph для проверки присутствия ребра в графе

```
bool is_edge_represented_in_graph(std::unique_ptr<ScMemoryContext> &ms_context, ScAddr &graph, ScAddr &edge){
  ScIterator3Ptr graph_itr = ms_context->Iterator3(graph, ScType::EdgeAccessConstPosPerm, ScType::EdgeDCommonConst);
  ScAddr begin_node = ms_context->GetEdgeSource(edge);
  ScAddr end_node = ms_context->GetEdgeTarget(edge);
  while(graph_itr->Next()){
    ScAddr check_edge = graph_itr->Get(2);
    ScAddr compared_begin = ms_context->GetEdgeSource(check_edge);
    ScAddr compared_end = ms_context->GetEdgeTarget(check_edge);
    if((get_link_context(ms_context, begin_node) == get_link_context(ms_context, compared_begin)) &&
       (get_link_context(ms_context, end_node) == get_link_context(ms_context, compared_end))){
         return true;
       }
  }
  return false;
}
```

Листинг SC_AGENT_IMPLEMENTATION

```
SC_AGENT_IMPLEMENTATION(GraphIntersectionAgent)
{
  ScLog *logger = ScLog::GetInstance();

  logger->Message(ScLog::Type::Info, "Trying to launch GraphIntersectionAgent");

  if(!edgeAddr.IsValid()) return SC_RESULT_ERROR;

  ScAddr question_node = ms_context->GetEdgeTarget(edgeAddr);
  ScAddr first_graph = IteratorUtils::getAnyFromSet(ms_context.get(), question_node);
  std::string db = get_link_context(ms_context, first_graph);
  SC_LOG_DEBUG(db);

  if(!first_graph.IsValid()) return SC_RESULT_ERROR_INVALID_PARAMS;

  ScAddr answer = ms_context->CreateNode(ScType::NodeStruct);


  ScIterator3Ptr first_graph_node_itr = ms_context->Iterator3(first_graph, ScType::EdgeAccessConstPosPerm, ScType::NodeConst);
  while(first_graph_node_itr->Next()){
    bool is_node_represented_in_all_graphs = true;
    ScIterator3Ptr graphs_itr = ms_context->Iterator3(question_node, ScType::EdgeAccessConstPosPerm, ScType::NodeStruct);
    ScAddr checked_node = first_graph_node_itr->Get(2);
    while(graphs_itr->Next()){
      ScAddr graph = graphs_itr->Get(2);
      if(get_link_context(ms_context, graph) == get_link_context(ms_context, first_graph)) continue;
      if(!is_node_represented_in_graph(ms_context, graph, checked_node)){
        is_node_represented_in_all_graphs = false;
        break;
```

```
    if(is_node_represented_in_all_graphs){
      ms_context->CreateEdge(ScType::EdgeAccessConstPosPerm, answer, checked_node);
      std::string debug = "Added node with idtf " + get_link_context(ms_context, checked_node) + " to result graph";
      SC_LOG_DEBUG(debug);
    }
  }
```

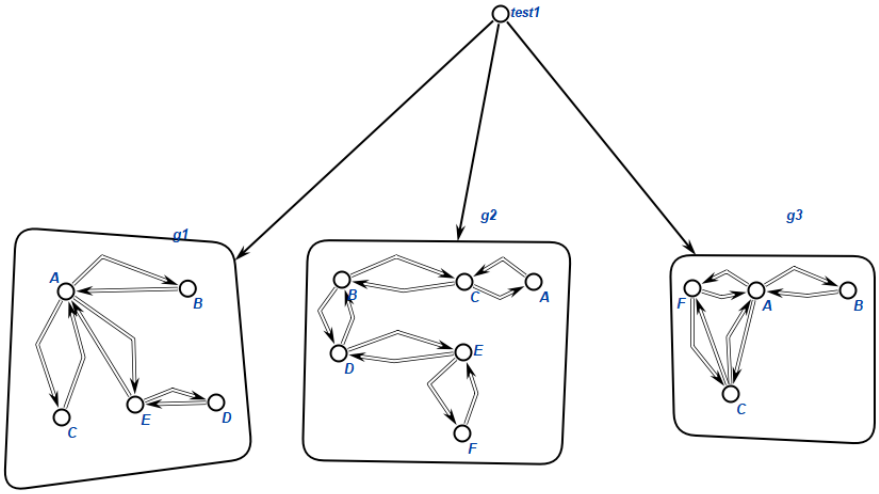
```
ScIterator3Ptr first_graph_edge_iterator = ms_context->Iterator3(first_graph, ScType::EdgeAccessConstPosPerm, ScType::EdgeDCommonConst);
while(first_graph_edge_iterator->Next()){
  bool is_edge_represented_in_all_graphs = true;
  ScAddr checked_edge = first_graph_edge_iterator->Get(2);
  ScIterator3Ptr graphs_itr = ms_context->Iterator3(question_node, ScType::EdgeAccessConstPosPerm, ScType::NodeStruct);

  while(graphs_itr->Next()){
    ScAddr graph = graphs_itr->Get(2);
    if(get_link_context(ms_context, graph) == get_link_context(ms_context, first_graph)) continue;

    if(!is_edge_represented_in_graph(ms_context, graph, checked_edge)){
      is_edge_represented_in_all_graphs = false;
      break;
    }
  }

  if(is_edge_represented_in_all_graphs){
    ScAddr begin_node = ms_context->GetEdgeSource(checked_edge);
    ScAddr end_node = ms_context->GetEdgeTarget(checked_edge);
    ScAddr new_edge = ms_context->CreateEdge(ScType::EdgeDCommonConst, begin_node, end_node);
    ms_context->CreateEdge(ScType::EdgeAccessConstPosPerm, answer, new_edge);
    SC_LOG_DEBUG("Added new edge!");
  }
}

utils::AgentUtils::finishAgentWork(ms_context.get(), question_node, answer);
```

```
  SC_LOG_DEBUG("GraphIntersectionAgent finished its work...");

  return SC_RESULT_OK;
}

}
```

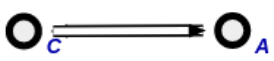
Тесты

Тесты проведены с учётом особенностей C++ API

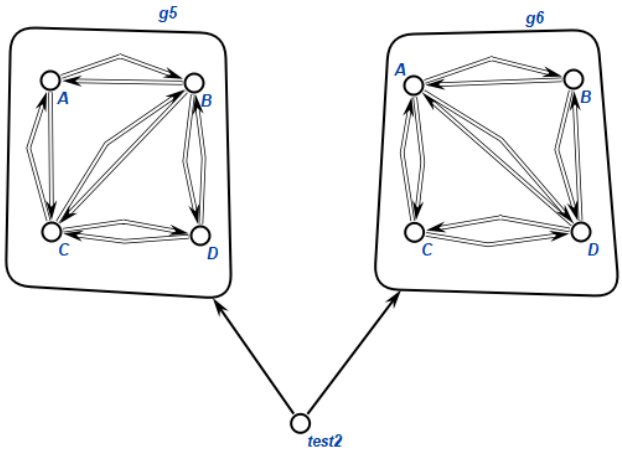
1. Вход:



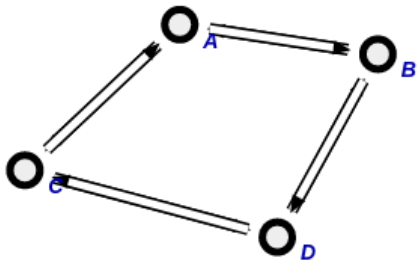
Выход:



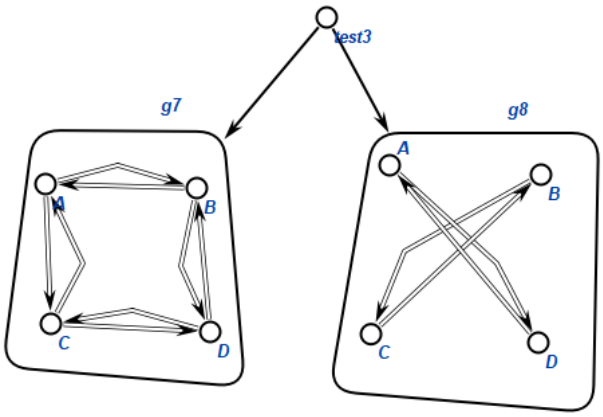
2. Вход



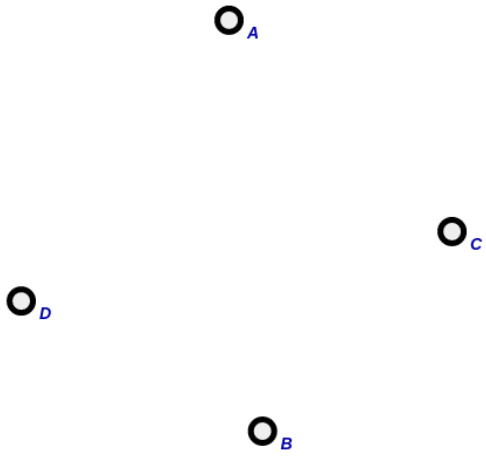
Выход



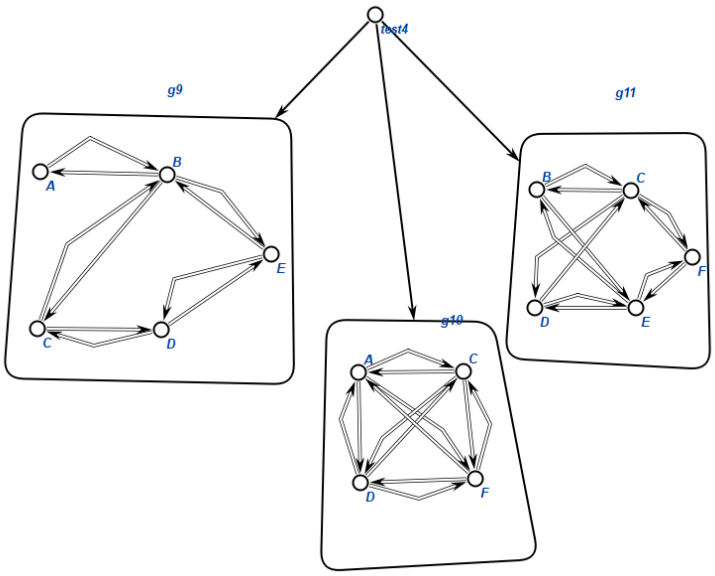
3. Вход



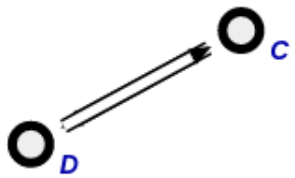
Выход



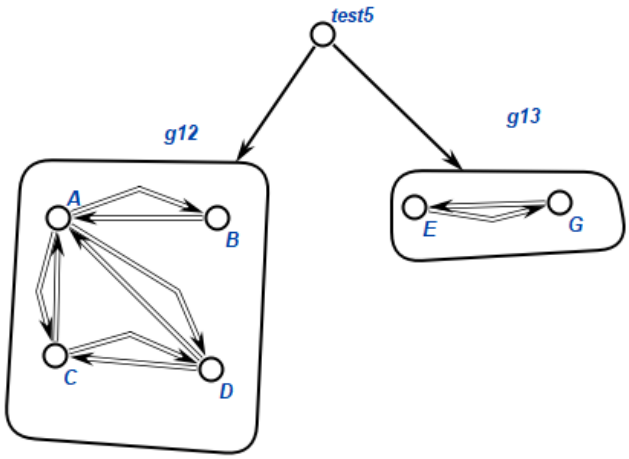
4. Вход



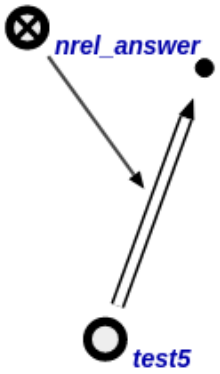
Выход



5. Вход



Выход:



Замечание: Выход теста 5 имеет подобный вид по причине того, что результирующий граф должен быть пустым (на входных графах видно, что у них нет общих вершин).