Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Информационных технологий и управления
Кафедра Интеллектуальных информационных технологий

К защите допустить:
Заведующий кафедрой
_____ Д. В. Шункевич

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе
по дисциплине «Проектирование программ в интеллектуальных системах»
на тему

Реализация алгоритма поиска гамильтонова графа

БГУИР КРЗ 1–40 03 01 01 006 ПЗ

Выполнил: П. А. Белоус
Студент группы
921702

Руководитель: Д. В. Шункевич

Минск 2020

Содержание

Перечень условных обозначений

В курсовой работе используются следующие условные обозначения:
ИСС — интеллектуальная справочная система;
БЗ — база знаний;
SC — Semantic Code;
SCs — Semantic Code String;
SCg — Semantic Code Graphical;
SCn — Semantic Code Natural;
SCp — Semantic Code Programming;
OSTIS — Открытая семантическая технология проектирования интеллектуальных систем (Open Semantic Technology for Intelligent Systems);

Введение

Целью курсовой работы в этом семестре было решение поставленной теоретико-графовой задачи по поиску неориентированного гамильтонова графа, ее детальное исследование и разработка БЗ для ИСС по диагностике автомобилей.

Для достижения заданной цели были поставлены следующие задачи:

– Реализация алгоритма и тестовых примеров для теоретико-графовой задачи на языке SCp.

– Реализация алгоритма и тестовых примеров для теоретикографовой задачи на языке C++ с использованием SC-Memory.

– Наполнение новыми понятиями раздела "Тормозная система".

1 Теоретико-графовая задача

1.1 Список понятий

1. Графовая структура (абсолютное понятие) - это такая одноуровневая реляционная структура, объекты которой могут играть роль либо вершины, либо связки:

1.1 Вершина (относительное понятие, ролевое отношение);

1.2 Связка (относительное понятие, ролевое отношение).

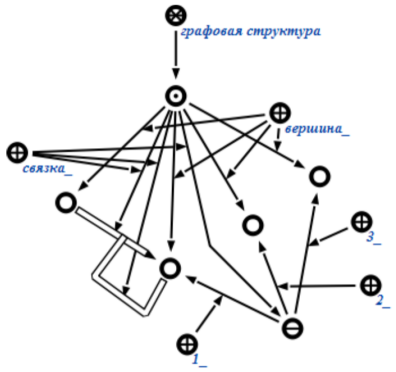


Рисунок *1.1* – Графовая структура

2. Графовая структура с неориентированными связками (абсолютное понятие)

2.1 Неориентированная связка (относительное понятие, ролевое отношение) – связка, которая задается неориентированным множеством.

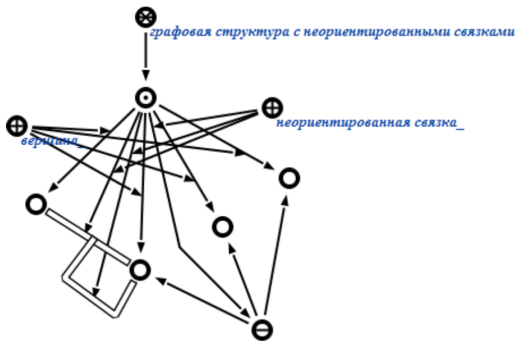


Рисунок *1.2* – Графовая структура с неор. связками

3. Граф (абсолютное понятие) – это такой мультиграф, в котором не может быть кратных связок, т.е. связок у которых первый и второй компоненты совпадают:

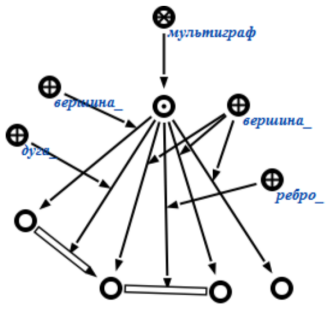


Рисунок *1.3* – Граф

4. Неориентированный граф (абсолютное понятие) –это такой граф, в котором все связки являются ребрами:

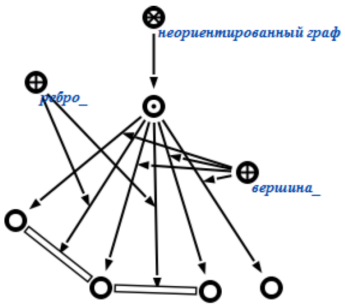


Рисунок *1.4* – Неориентированный граф

5. Связный граф (абсолютное понятие) – это такой граф, который содержит только одну компоненту связности:

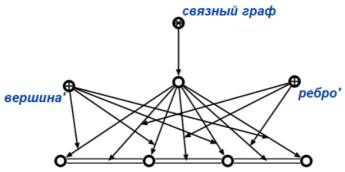


Рисунок *1.5* – Связный граф

6. Цикл - последовательность вершин, начинающаяся и заканчивающаяся в одной и той же вершине. В данном примере показан цикл 2-4-3-2.

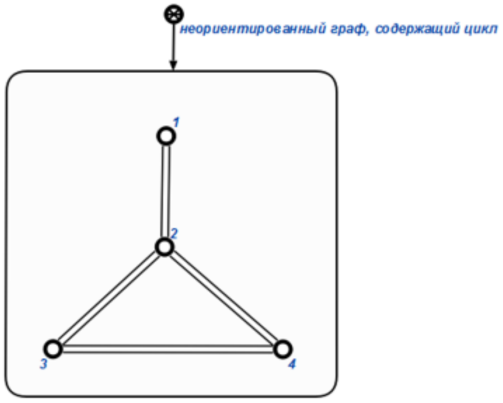


Рисунок *1.6* – Цикл

7. Гамильтонов граф — граф, содержащийв цикл, который проходит через каждую вершину данного графа ровно по одному разу:

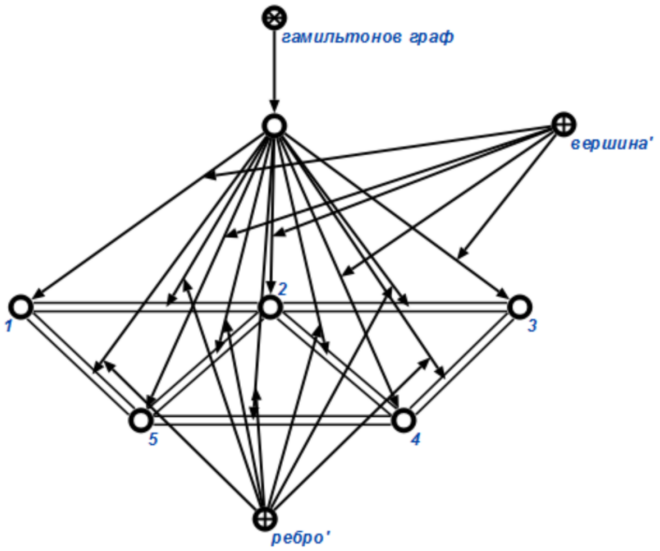


Рисунок *1.7* – Гамильтонов граф

1.2 Разработка алгоритма

Алгоритм поиска радиуса графа заключается в следующем:

1. Мы приходим в произвольную вершину нашего графа.

2. Заносим данную вершину в список начальных вершин.

3. Заносим данную вершину в список посещённых вершин.

4. Заносим данную вершину в список вершин-предков.

5. Ищем все вершины инцидентные текущей, проверяя их на принадлежность к множеству посещённых вершин.

6. Если непосещённая вершина найдена переходим с ней в пункт 3, а если не найдена переходим в пункт 7

7. Если количество посещенных вершин равно количеству вершин графа преходим к пункту 8, если нет - переходим в пункт 9

8. Если текущая вершина инцидентна начальной вершине - Граф Гамильтонов , если нет - переходим к пункту 9.

9. Удаляем текущую вершину из посещенных и рекурсивно возвращаемся в вершину предка и переходим в пункт 5.

10. Конец алгоритма.

1.3 Тестирование алгоритма

Для проверки разработанного алгоритма были составлены тестовые примеры, которые позволят проверить правильность решения поставленной задачи. Во всех тестах графы будут приведены в сокращенной форме со скрытыми ролями элементов графа.

Тест 1

Вход:

Необходимо определить является ли данный граф гамильтоновым.

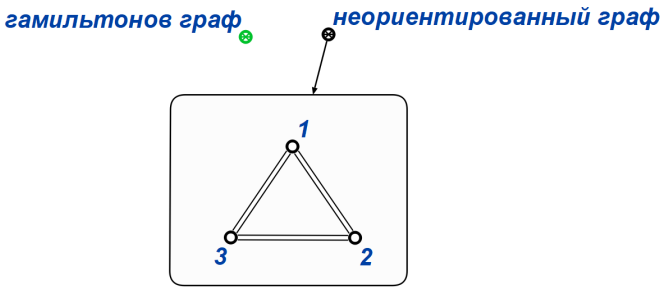


Рисунок *1.8* – Вход теста 1

Выход:

Граф был определен как гамильтонов.

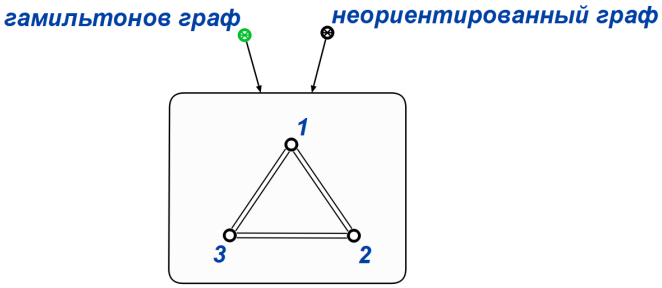


Рисунок *1.9* – Выход теста 1

Граф был определен как гамильтонов.

Тест 2
Вход:
Необходимо определить является ли данный граф гамильтоновым.

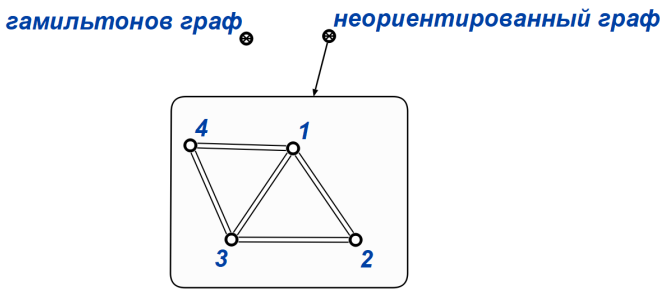


Рисунок *1.10* – Вход теста 2

Выход:
Граф был определен как гамильтонов.

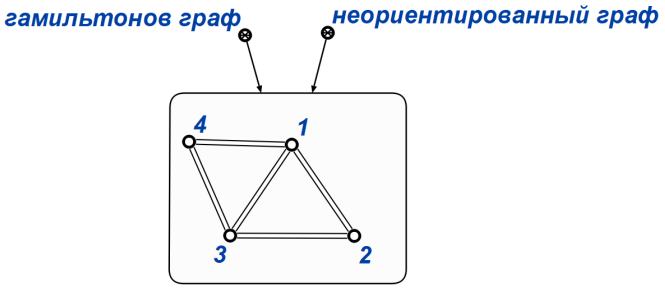


Рисунок *1.11* – Выход теста 2

Тест 3
Вход:
Необходимо определить является ли данный граф гамильтоновым.

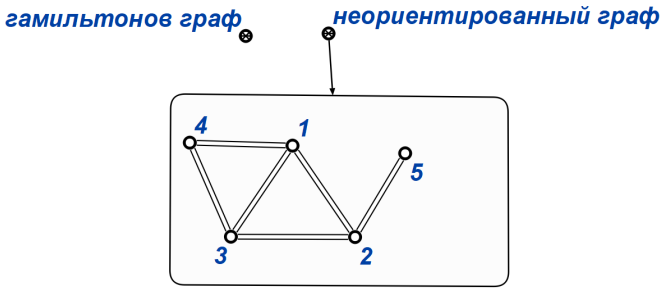


Рисунок *1.12* – Вход теста 3

Выход:
Граф был определен как негамильтонов.

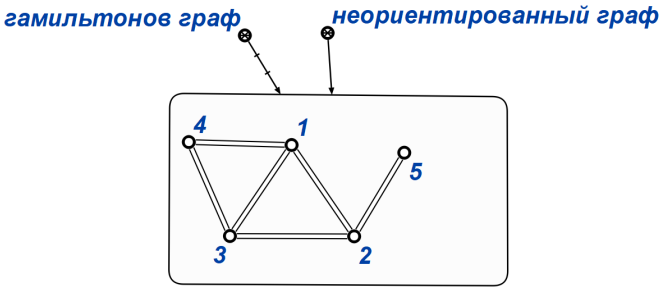


Рисунок *1.13* – Выход теста 3

Тест 4
Вход:
Необходимо определить является ли данный граф гамильтоновым.

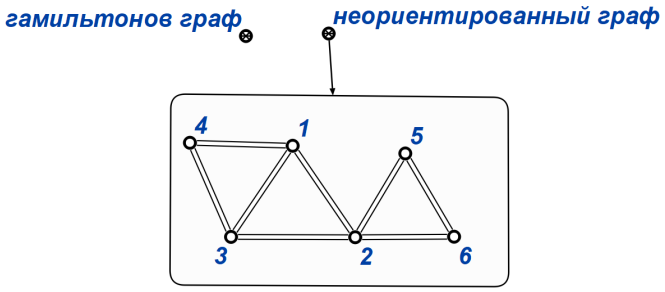


Рисунок *1.14* – Вход теста 4

Выход:
Граф был определен как негамильтонов.

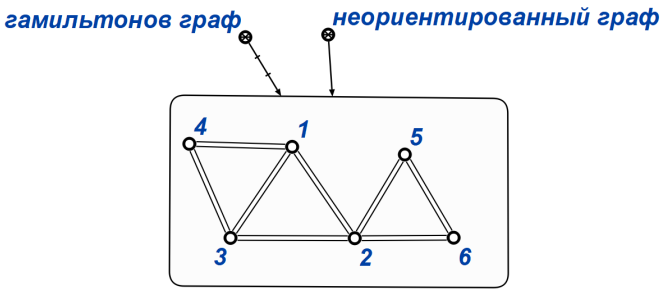


Рисунок *1.15* – Выход теста 4

Тест 5
Вход:
Необходимо определить является ли данный граф гамильтоновым.

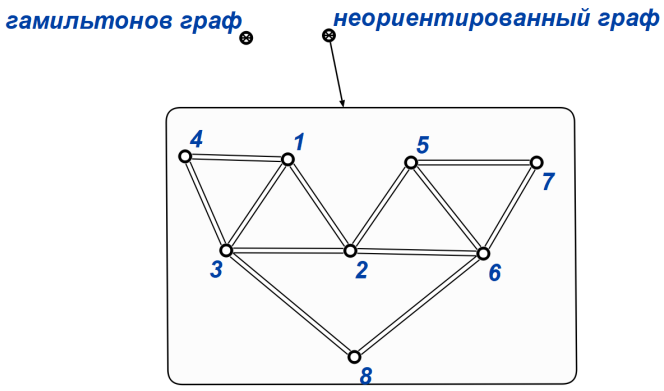


Рисунок *1.16* – Вход теста 5

Выход:
Граф был определен как гамильтонов.

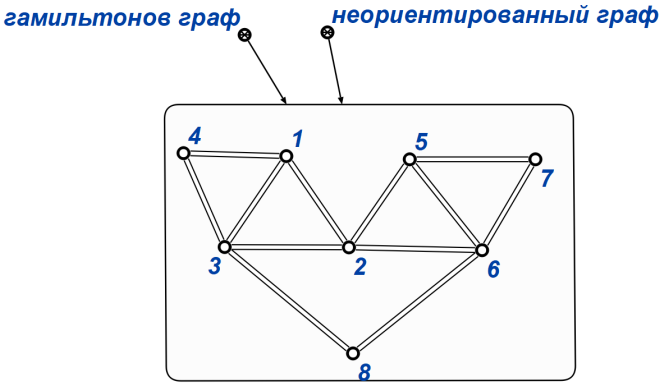


Рисунок *1.17* – Выход теста 5

1.4 Реализация алгоритма и демонстрация:

_vertex - Множество вершин графа.

_notchecked - Множество непросмотренных вершин.

_prev - множество, содержащее пердыдущие посещенные вершины графа.

_begin - переменная, которая получит в качестве значения начальную вершину 1.

_vertexcount - переменная, в которой будет находиться число посещенных вершин.

_current - переменная, в которой будет находиться текущая рассматриваемая вершина.

_repeat - Множество неповторяющихся узлов, связанных с определенным узлом.

Пример выполнения алгоритма в sc-памяти:

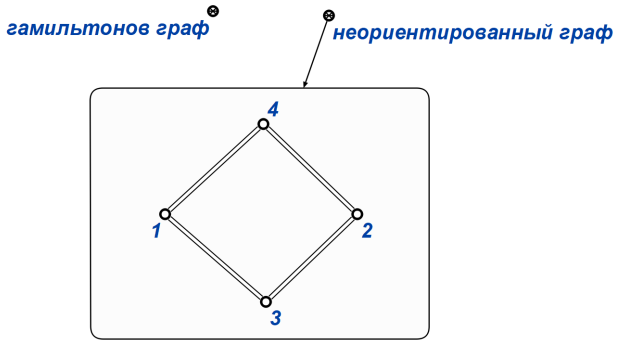


Рисунок *1.18* – Исходный граф

Шаг 1. Создаем множества _vertex, _notchecked, _prev;

Шаг 2. Создаем переменные _begin, _current и счетчик _vertexcount;

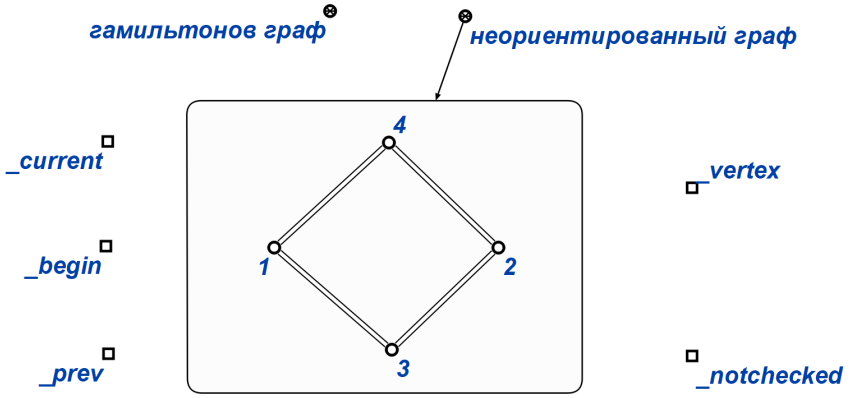


Рисунок *1.19* – Создание переменных _begin, _current, счетчика _vertexcount и множеств _vertex, _notchecked, _prev

Шаг 3. Внесем все вершины во множество _vertex и _notchecked;

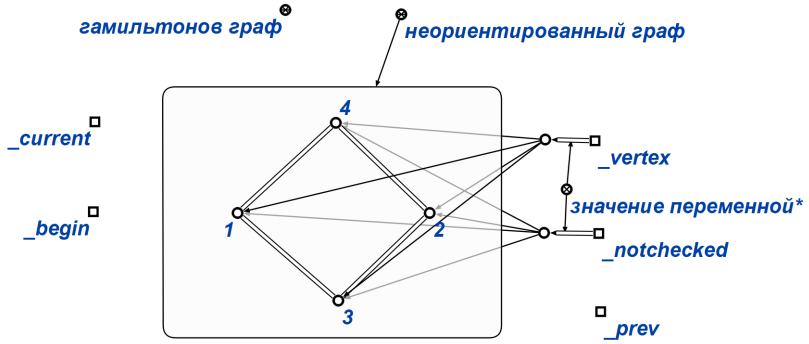


Рисунок *1.20* – Внесение всех вершин во множества _vertex и _notchecked

Шаг 4. Если количество вершин _vertex < 3 переходим к шагу 20;

Шаг 5. Внесем начальную вершину во множество _begin;

Шаг 6. Заносим вершину равную _begin в переменную _current и во множество _prev;

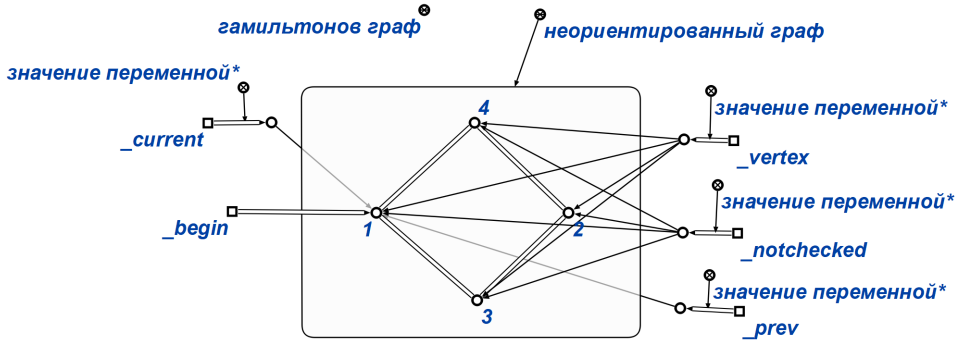


Рисунок *1.21* – Занесение вершины _begin в переменную _current и во множество _prev

Шаг 7. Если существует следующая вершина из _vertex, смежная с текущей вершиной _current и она принадлежит _notchecked переходим к шагу 8, иначе - к шагу 13.

Шаг 8. Если текущей вершины _current нет во множестве _prev, внесем текущую вершину _current в это множество _prev, иначе переходим к шагу 9;

Шаг 9. Вносим следующую вершину из множества _vertex смежную с вершиной _current в переменную _current;

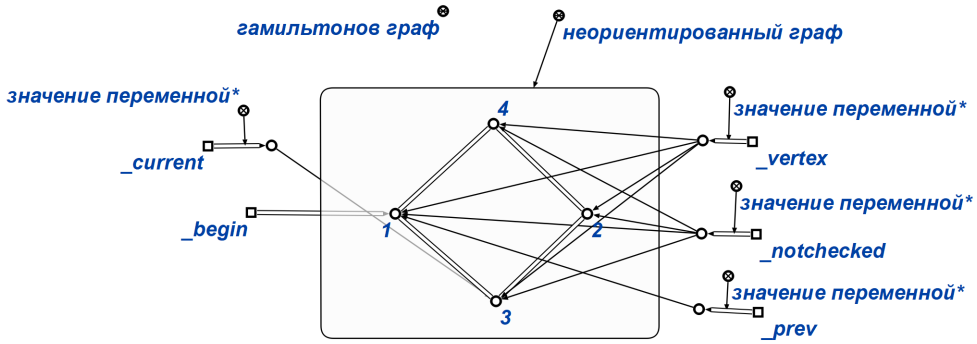


Рисунок *1.22* – Внесение вершины смежной с вершиной _current в переменную _current

Шаг 10. Инкрементируем счетчик _vertexcount;

Шаг 11. Если вершина _current совпадает с переменной _begin, переходим к шагу 12, иначе - к шагу 16;

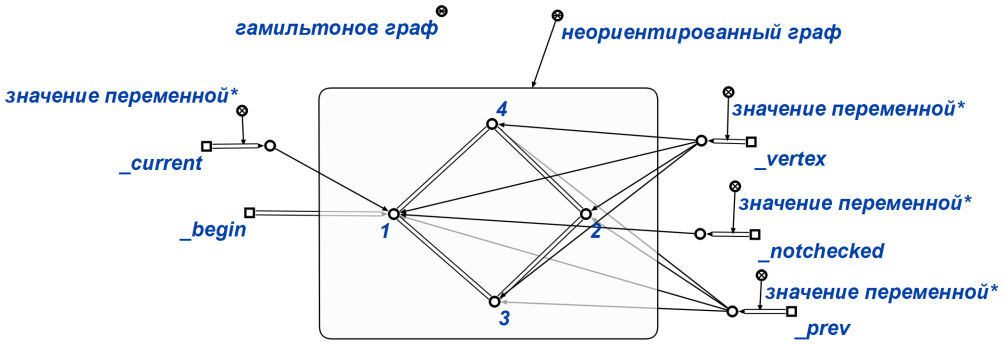


Рисунок *1.23* – Проверка, совпадает ли вершина _current с переменной _begin

Шаг 12. Если количество вершин из _vertex совпадает с переменной _vertexcount, переходим к шагу 19, иначе - к шагу 13;

Шаг 13. Если вершины _current нет в _notchecked, добавить текущую верщину _current в _notchecked;

Шаг 14. Возвращаемся к последней вершине из _prev и заносим ее в _current;

Шаг 15. Удаляем текущую вершину _current из _prev, деинкриминтируем _vertexcount и переходим к шагу 7.

Шаг 16. Если текущая вершина _current присутствует в _notchecked, переходим к шагу 17, иначе - к шагу 7.

Шаг 17. Если текущей вершины _current нет в _prev, внесем текущую вершину _current в _prev, иначе переходим к шагу 18.

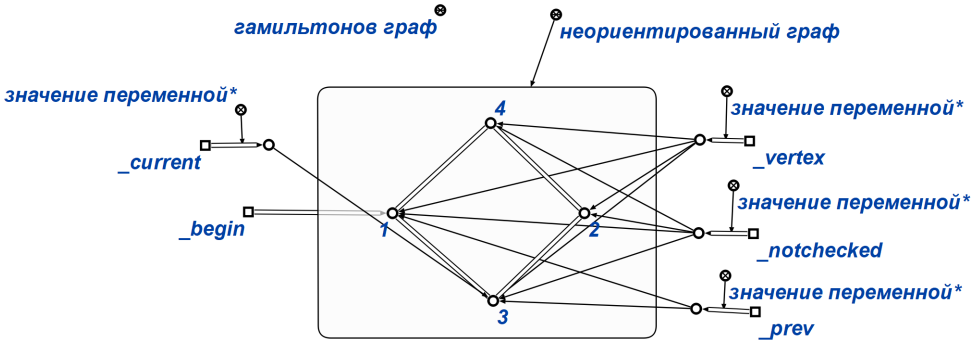


Рисунок *1.24* – Внесение текущей вершины _current в _prev

Шаг 18. Удаляем текующую вершину _current из _notchecked и переходим к шагу 7;

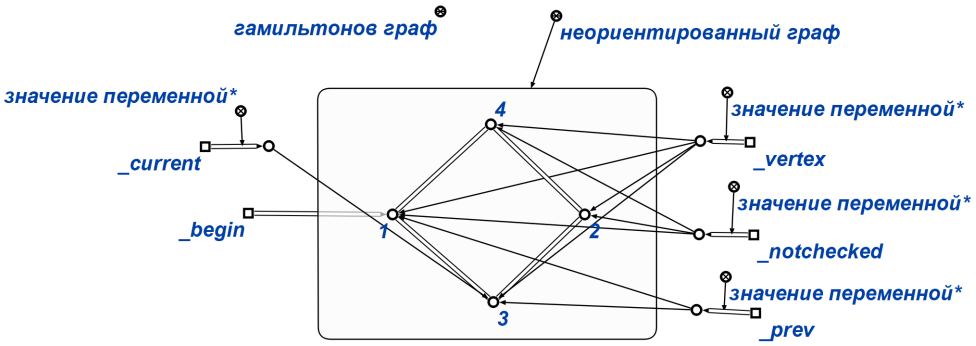


Рисунок *1.25* – Удаление текующей вершины _current из _notchecked

Шаг 19. Гамильтонов цикл найден.
Шаг 20. Граф является гамилтоновым.

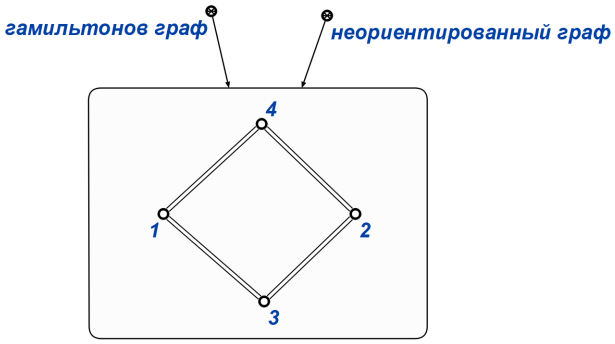


Рисунок *1.26* – Гамильтонов граф

Шаг 21. Завершение алгоритма.

2 Детальное исследование теоретико-графовой задачи

2.1 Проверка на некорректные входные данные

В алгоритме реализованном на C++ введён ряд проверок, на некорректный ввод данных. Для примера расмотрим код предостерегающий работу алгоритма с единичной вершиной(Рисунок 2.1):

```
if(it_3->Next())
    hinges = dfs(context, all_arcs, it_3->Get(2), hinges);
else std::cout<<"Вершина не имеет инцидентных вершин!!!";
```

Рисунок *2.1* – Проверка вершины на инцидентность с другими вершинами

В коде на scp существует проверка(Рисунок 2.2) на содержимое графа. Если вершина предположительно содержащая граф окажется пустой, алгоритм выведет сообщение(Рисунок 2.3) закончит работу:

```
//Проверка содержимого графа
-> ..check_value_of_graph(*
    <- ifVarAssign;;
    -> rrel_1: _graph;;
    => nrel_then: ..print_error;;
    =>nrel_else: ..get_all_begin_vertices;;
*);;

-> ..print_error(*
    <-printNl;;
    -> rrel_1: rrel_fixed: rrel_scp_const: [Граф не найден!!!];;
    => nrel_goto: ..hinges_end;;
*);;
```

Рисунок *2.2* – Представление проверки в коде

```
>>printNl
Граф не найден!!!
Client connected:  QHos
```

Рисунок *2.3* – Вывод сообщения об ошибке

2.2 Документация программы

При написании программы псевдоязыке scp(1 часть курсовой работы),я вел документирование программы. Оставлял комментарии к функциям программы, которые нуждались в дополнительном пояснении об их функциональности и цели конкретно созданной функции. Также мной были оставлены комментарии при написании программы на языке scp. Вся программа разделена на три составные части (функции).Первая функция отвечает за вывод графовых примеров и их поочередный запуск для функции. В комментировании первая функция не нуждается,так как она несет в себе алгоритм нахождения вершинной связности. Так же можно сказать и про третью часть программы, которая формирует все необходимые переменные для запуска второй функции (алгоритма нахождения гамильтонова цикла).

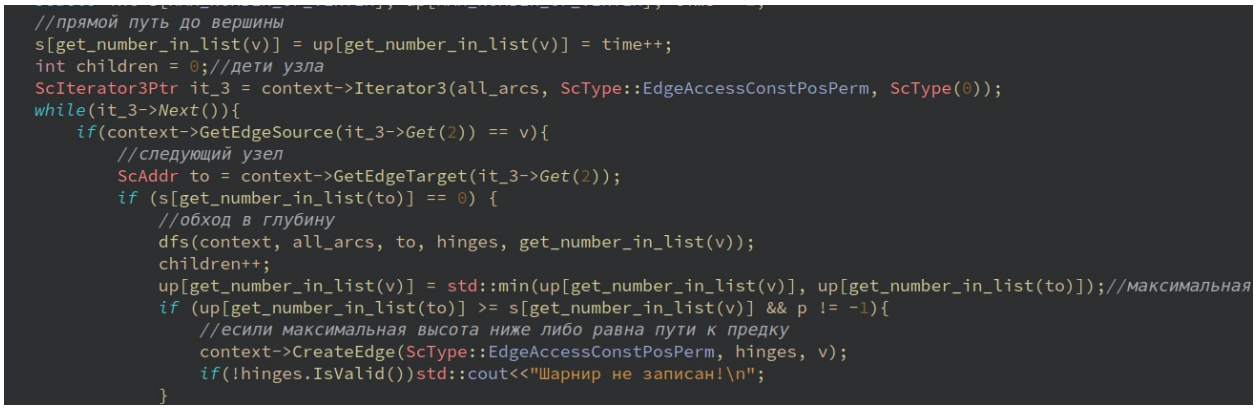
Описание алгоритма на scp(Рисунок 2.4) :

```
//Проверка на случай если мы уже были в _next_vertex
-> ..check_visited(*
    <- searchElStr3;;
    -> rrel_1: rrel_fixed: rrel_scp_const: _visited_verteces;;
    -> rrel_2: rrel_assign: rrel_scp_var: _visit_arc;;
    -> rrel_3: rrel_fixed: rrel_scp_var: _next_vertex;;
    => nrel_then: ..erase_visited_vertex;;
    => nrel_else: ..sturtsquhuygoyiy;;
*);;

//Если мы уже были в данной вершине,
//мы в неё не заходим, а лишь удаляем из списка смежных
-> ..erase_visited_vertex(*
    <- eraseEl;;
    -> rrel_1: rrel_fixed: rrel_erase: rrel_scp_var: _incident_arc;;
    => nrel_goto: ..try;;
*);;
```

Рисунок *2.4* – Описания алгоритма на scp

Описание алгоритма на C++(Рисунок 2.5) :



```
//прямой путь до вершины
s[get_number_in_list(v)] = up[get_number_in_list(v)] = time++;
int children = 0;//дети узла
ScIterator3Ptr it_3 = context->Iterator3(all_arcs, ScType::EdgeAccessConstPosPerm, ScType(0));
while(it_3->Next()){
    if(context->GetEdgeSource(it_3->Get(2)) == v){
        //следующий узел
        ScAddr to = context->GetEdgeTarget(it_3->Get(2));
        if (s[get_number_in_list(to)] == 0) {
            //обход в глубину
            dfs(context, all_arcs, to, hinges, get_number_in_list(v));
            children++;
            up[get_number_in_list(v)] = std::min(up[get_number_in_list(v)], up[get_number_in_list(to)]);//максимальная
            if (up[get_number_in_list(to)] >= s[get_number_in_list(v)] && p != -1){
                //если максимальная высота ниже либо равна пути к предку
                context->CreateEdge(ScType::EdgeAccessConstPosPerm, hinges, v);
                if(!hinges.IsValid())std::cout<<"Шарнир не записан!\n";
            }
```

Рисунок *2.5* – Описания алгоритма на C++

3 Личный вклад в развитие ИСС по диагностике автомобилей

3.1 Разработка БЗ для ИСС по диагностике автомобилей

Был наполнен новыми понятиями раздел «Тормозная система» и «Двигатель внутреннего сгорания», а так же дополнены некоторые существовавшие до этого понятия. В рамках этого раздела были формализованы следующие абсолютные и относительные понятия:
1. тормозная трубка
2. барабанная тормозная колодка
3. топливный фильтр
4. тормозной суппорт
5. мост
6. стяжная пружина
7. дисковая тормозная колодка
8. топливная система карбюраторного двигателя
9. топливный насос
10. ступица
11. гидравлический тормозной привод
12. форсунка
13. инжекторная система подачи топлива
14. монтажный комплект тормозной системы
15. штифт
16. поршень суппорта
17. ремкомплект вакуумных усилителей тормозов
18. ремкомплект суппортов
19. опорные пальцы колодки
20. механизм самоподвода
21. тип передних тормозов
22. колодочная распорка
23. пневматический тормозной привод
24. ремкомплект тормозной колодки
25. вакуумный усилитель тормозов
26. ремкомплект дискового тормоза
27. ремкомплект тормозной колодки
28. тормозной щит
29. ленточный тормоз
30. втулка тормозной колодки
31. тормозной путь*
32. давление тормозной жидкости*
33. давление*

Рассмотрим формализованное на языке SCn абсолютное понятие "штифт". На рисунке 3.1 изображен фрагмент базы знаний ИСС по диагностике, показывающий идентификаторы понятия "штифт".

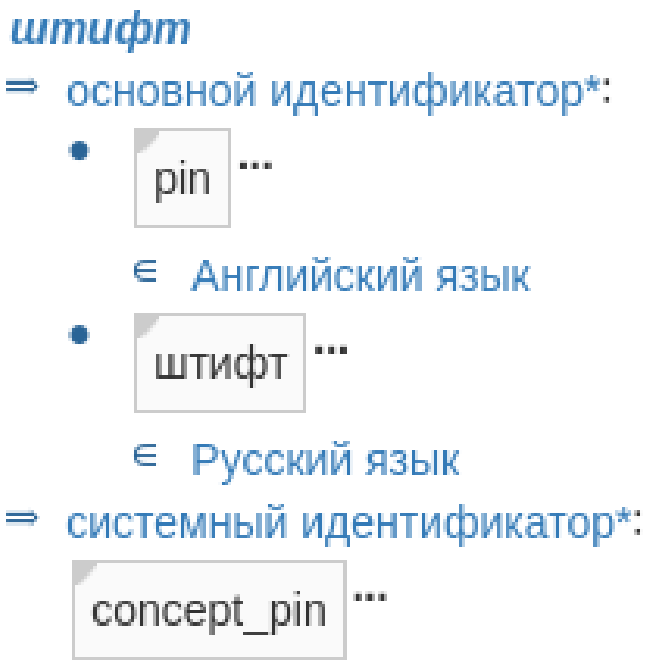


Рисунок *3.1* – Идентификаторы понятия "штифт"

На рисунке 3.2 изображен фрагмент базы знаний ИСС по диагностике автомобилей, показывающий теоретико-множественные связи понятия "штифт".

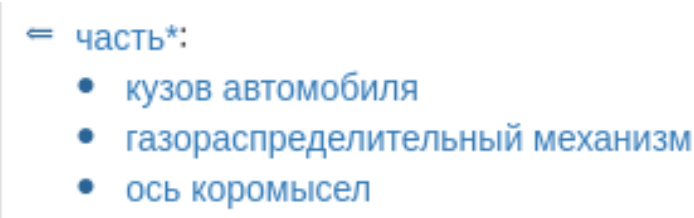


Рисунок *3.2* – Теоретико-множественные связи понятия "штифт"

На рисунке 3.3 изображен фрагмент базы знаний ИСС по диагностике автомобилей, показывающий определение понятия "штифт".

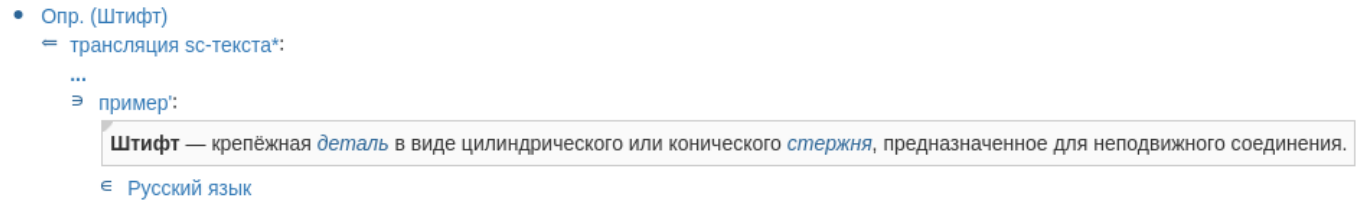


Рисунок *3.3* – Определение понятия "штифт"

На рисунке 3.4 изображен фрагмент базы знаний ИСС по диагностике автомобилей, показывающий утверждение с понятием "штифт".

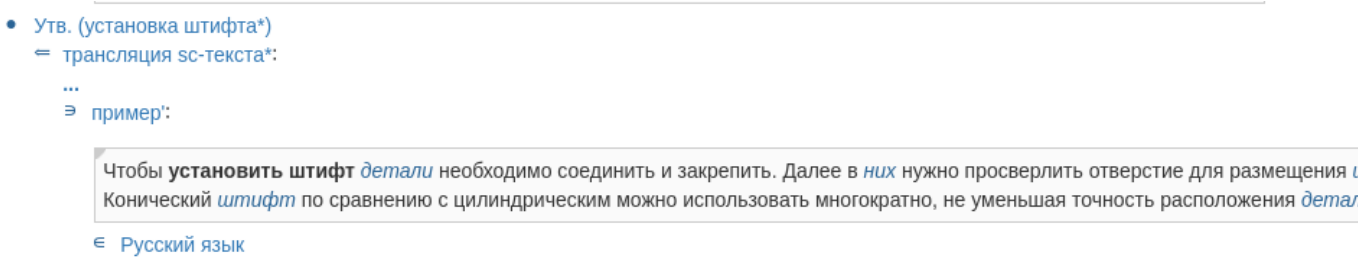


Рисунок *3.4* – Утверждение с понятием "штифт"

На рисунке 3.5 изображен фрагмент базы знаний ИСС по диагностике автомобилей, показывающий изображение понятия "штифт".

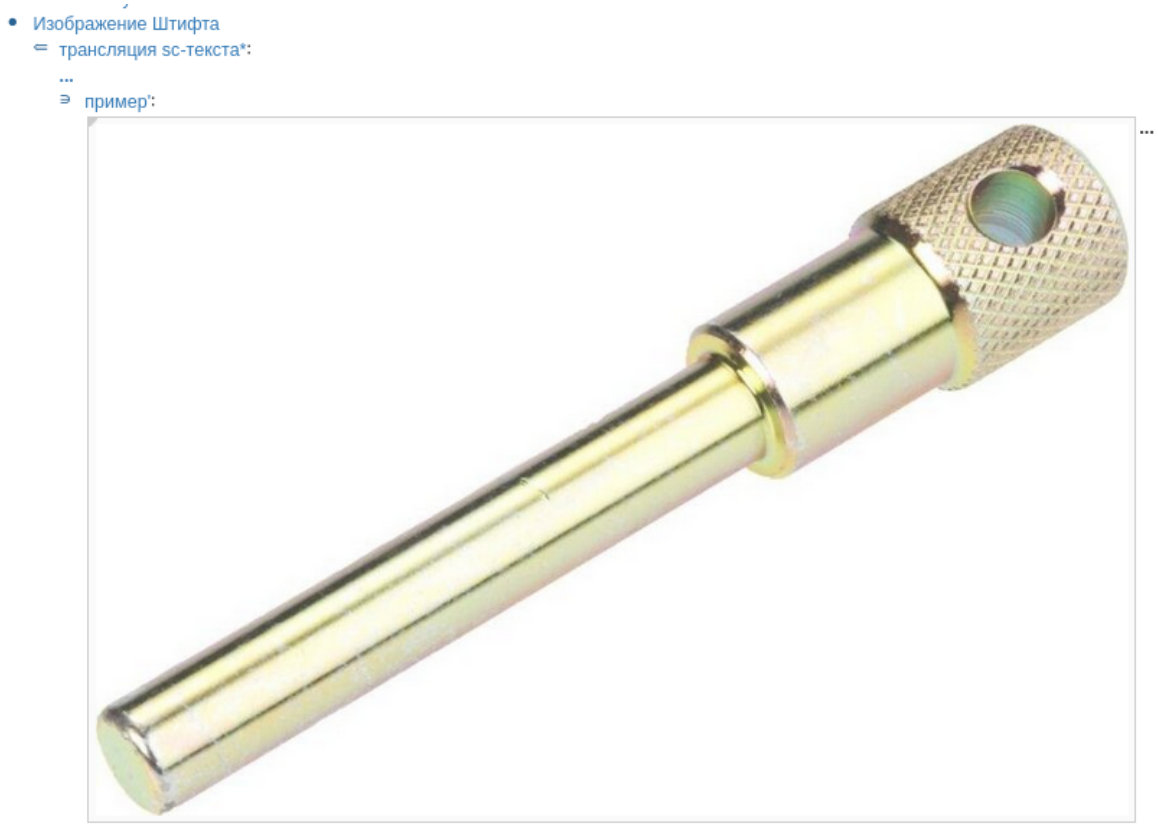


Рисунок *3.5* – Изображение понятия "штифт"

Заключение

В рамках курсовой работы были решены следующие задачи:

— Разработан алгоритм теоретико-графовой задачи: Определение гамильтонова графа.

— Проведено детальное исследование теоретико-графовой задачи.

— Дополнен раздел «Тормозная система» и «Топливная система» в ИСС по диагностике автомобилей.

— Формализовано 33 понятия в разделах «Тормозная система» и «Топливная система».

Список использованных источников

[1] Кузнецов, О. П. Дискретная математика для инженера / О. П. Кузнецов; Издательство «Лань». — 2009. — 394 с.

[2] Метасистема IMS. http://ims.ostis.net.

[3] Ф., Харари. Теория графов / Харари Ф. — КомКнига, 2006. — 296 с.

[4] В.А., Горбатов. Фундаментальные основы дискретной математики. Информационная математика / Горбатов В.А. — Наука, Физматлит, 2000. — 544 с.

[5] Ф.А., Новиков. Дискретная математика для программистов / Новиков Ф.А. — Питер, 2003. — 364 с.

[6] В.Б., Тарасов. От многоагентных систем к интеллектуальным организациям / Тарасов В.Б. — Изд-во УРСС, 2002.

[7] О., Оре. Теория графов / Оре О. — Наука, 1980. — 336 с.

[8] Головко, В. А. Нейроинтеллект: теория и применение. Книга 1: Организация и обучение нейронных сетей с прямыми и обратными связями / В. А. Головко. — Брестский политехнический институт., 1999. — 265 Р.

[9] Круглов В.В., Борисов В.В. Искусственные нейронные сети. Теория и практика / Борисов В.В. Круглов В.В. — Горячая линия-Телеком., 2002. — 383 Р.

[10] Зыков, А.А. Основы теории графов / А.А. Зыков. — Издательство «Наука», 1987. — 384 с.