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»:

Реализация алгоритма поиска гамильтонова графа

БГУИР КРЗ 1–40 03 01 02 81 ПЗ

Студентка: А. А. Рабушка
Группа: 021703

Руководитель: Д. В. Шункевич

Минск 2021

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ

БЗ — база знаний;
ИИС — интеллектуальная справочная система;
SC — Semantic Code;
SCg — Semantic Code Graphical;
SCn — Semantic Code Natural;
SCP — Semantic Code Programming.

ВВЕДЕНИЕ

Целью курсовой работы является решение поставленной теоретико-графовой задачи по поиску неориентированного гамильтонова графа и разработка БЗ для ИИС по кинофильмам.

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 теоретико-графовой задачи на языке C++ с использованием SC-Memory

– Создание и наполнение новыми понятиями разделов "Фильмы", "Актеры", "Режиссеры", "Мультфильмы", "Киностудии", "Персонажи".

1 ТЕОРЕТИКО-ГРАФОВАЯ ЗАДАЧА

1.1 Список понятий

а) Граф (абсолютное понятие) - совокупность непустого множества вершин и множества пар вершин (связей между вершинами), где каждая связь установлена ровно между двумя вершинами.

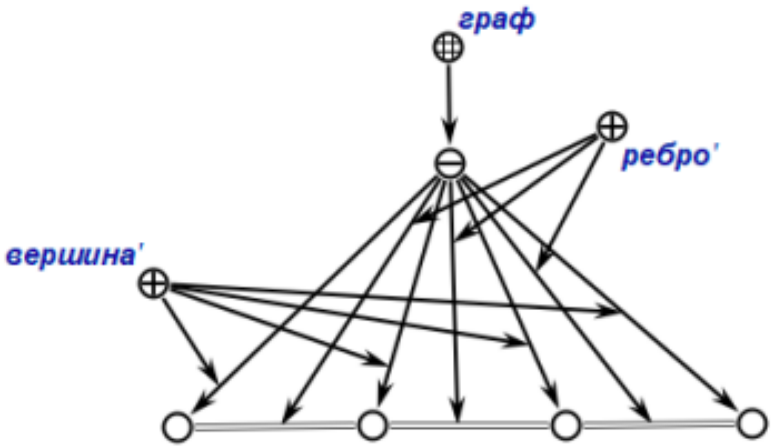


Рисунок *1.1* – Граф

б) Неориентированный граф (абсолютное понятие) – совокупность непустого множества вершин и множества неупорядоченных пар различных вершин, называемых рёбрами.

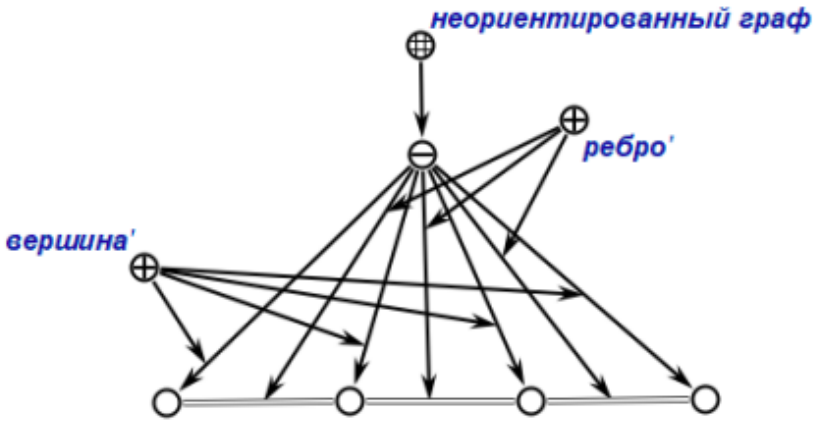


Рисунок *1.2* – Неориентированный граф

в) Граф-цикл - граф, состоящий из единственного цикла, или некоторого числа вершин, соединенных замкнутой цепью.

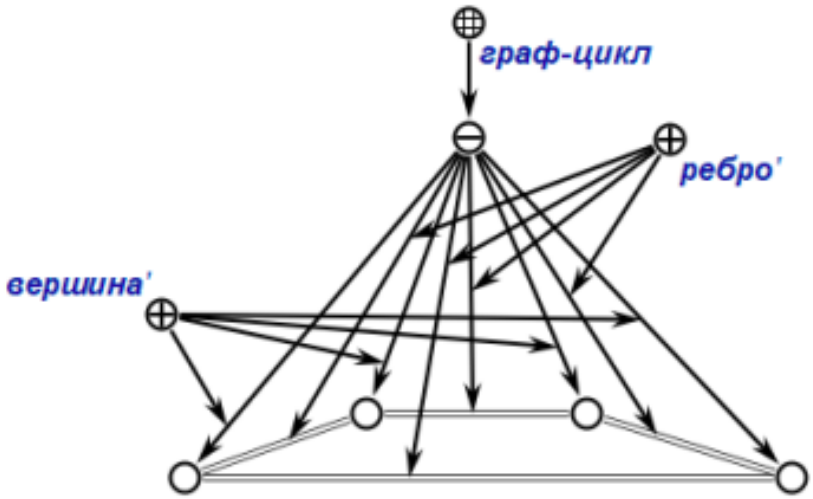


Рисунок *1.3* – Граф-цикл

г) Гамильтонов граф - граф, в котором есть простой цикл, содержащий все вершины графа ровно по одному разу(гамильтонов цикл).

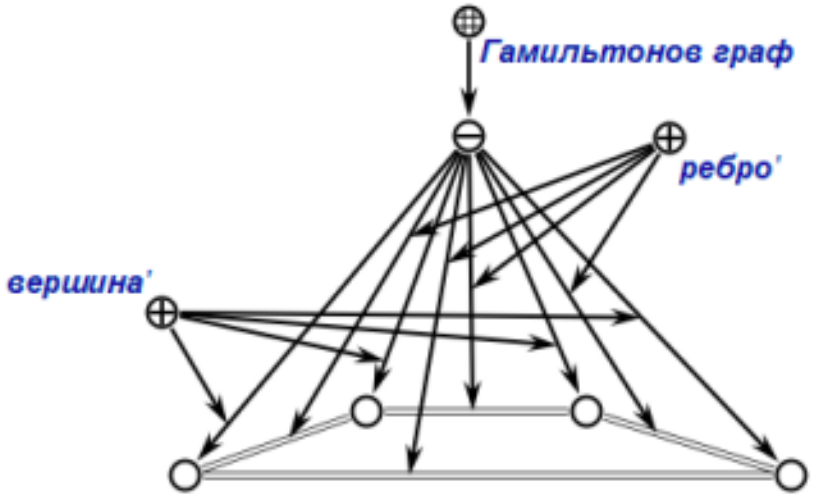


Рисунок *1.4* – Гамильтонов граф

1.2 Разработка алгоритма

Переменные, используемые для выполнения программы:

_not_checked_vertexes - переменная, принимающая значение множества непроверенных вершин

_path - переменная, принимающая значение множества, являющегося результатом выполнения программы

_rrel_parent - переменная, принимающая значение дуги от вершины к её родителю

Алгоритм поиска гамильтонова графа:
1. Находим все вершины графа и заносим их в _not_checked_vertexes.
2. Выбираем начальную вершину графа и удаляем её из _not_checked_vertexes
3. Добавляем данную вершину в _path
4. Ищем вершину, смежную данной
5. Если вершина найдена, переходим к пункту 6, в противном случае переходим к пункту 10
6. Если найденная вершина - начальная, переходим к пункту 10, в противном случае переходим к пункту 7
7. Если найденная вершина принадлежит _not_checked_vertexes, переходим к пункту 8, в противном случае к пункту 12
8. Добавляем найденную вершину в _path и проводим от неё к предыдущей вершине дугу, которую добавляем в _rrel_parent
9. Удаляем найденную вершину из _not_checked_vertexes, переходим к пункту 4
10. Если _not_checked_vertexes пустое (перебрали все вершины), переходим к пункту 13, в противном случае переходим к пункту 12
11. Добавляем данную вершину в _not_checked_vertexes, улаляем её из _path
12. Если текущая вершина начальная, переходим к пункту 14. В противном случае переходим по дуге, принадлежащей _rrel_parent, к предыдущей вершине и переходим к пункту 4
13. Граф является гамильтоновым. Добавляем начальную вершину во множество результата _path, переходим к пункту 15
14. Граф не является гамильтоновым
15. Завершение 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 сопределить является ли данный граф гамильтоновым (Рисунок 1.5)

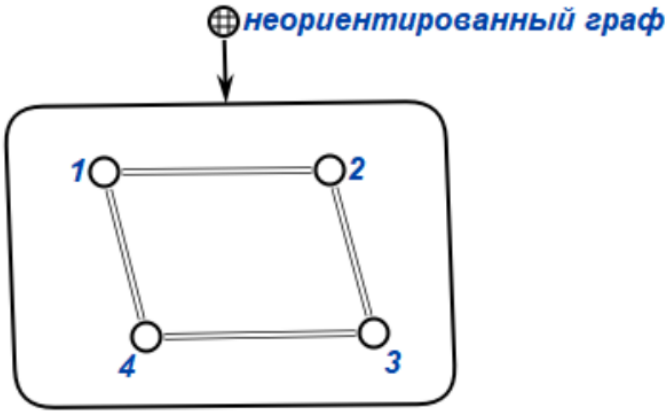


Рисунок *1.5* – Вход первого теста

Выход:

Граф является гамильтоновым (Рисунок 1.6).

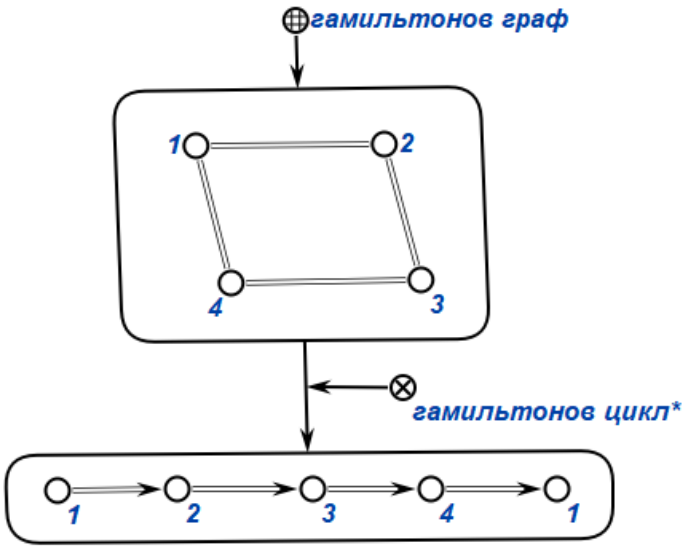


Рисунок *1.6* – Выход первого теста

Тест 2
Вход:
Граф является гамильтоновым (Рисунок 1.7)

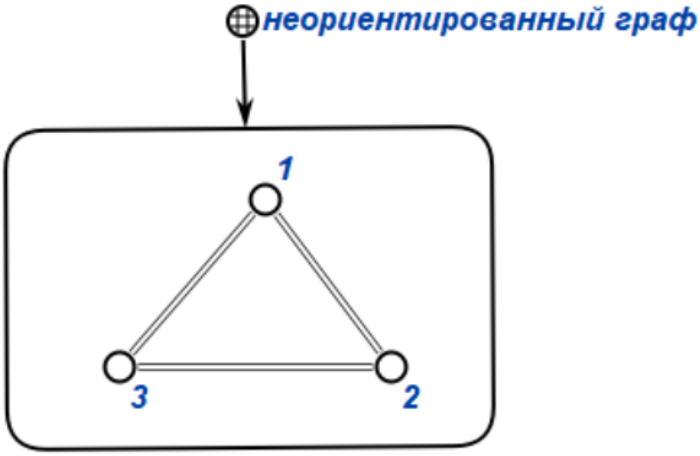


Рисунок *1.7* – Вход второго теста

Выход:
Граф является гамильтоновым (Рисунок 1.8).

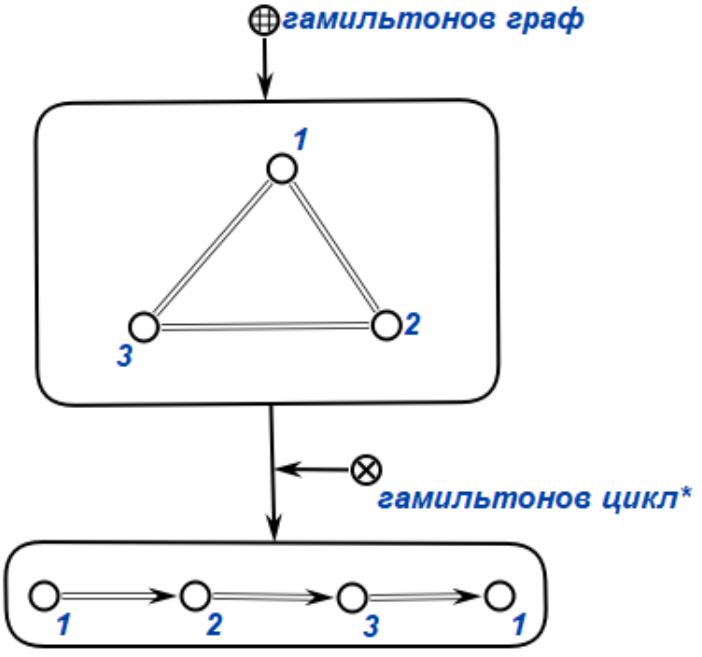


Рисунок *1.8* – Выход второго теста

Тест 3
Вход:
Граф является гамильтоновым (Рисунок 1.9)

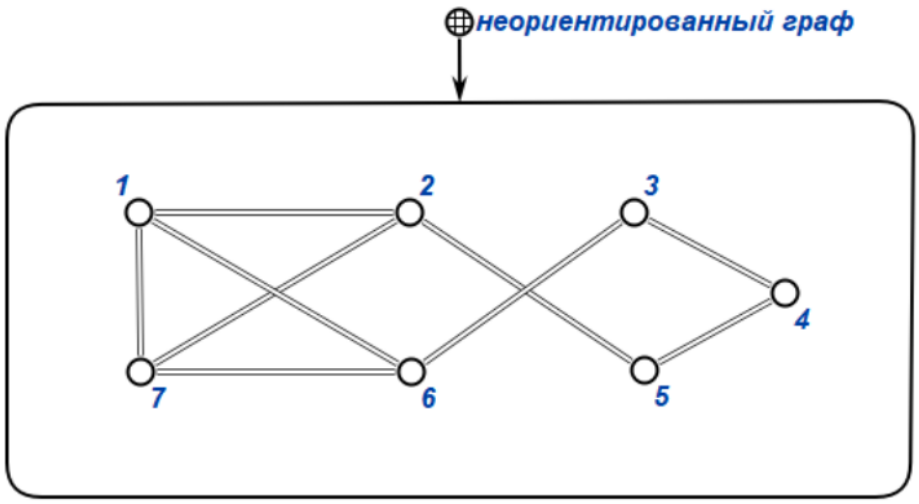


Рисунок *1.9* – Вход третьего теста

Выход:
Граф является гамильтоновым (Рисунок 1.10).

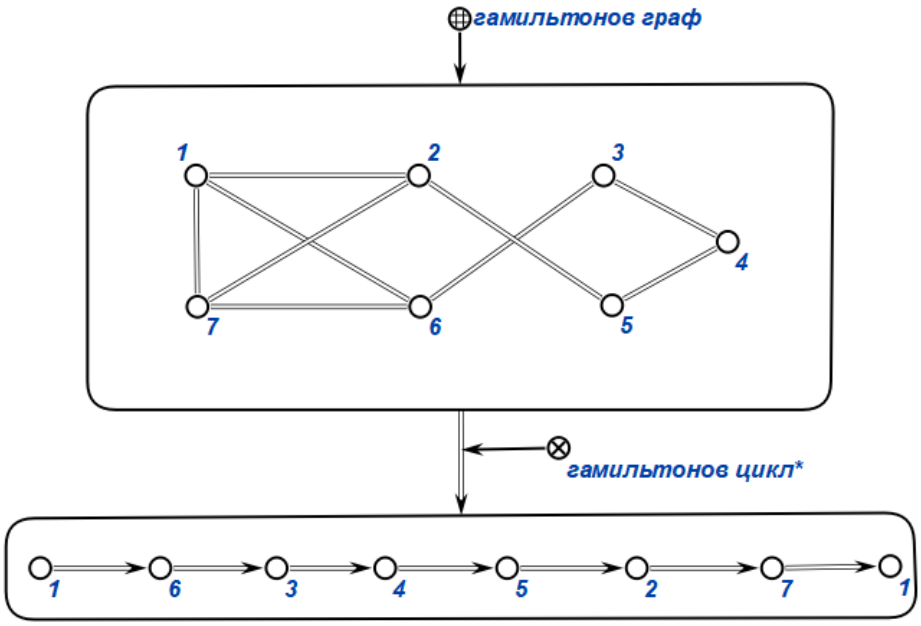


Рисунок *1.10* – Выход третьего теста

Тест 4

Вход:

Необходимо сопределить является ли данный граф гамильтоновым (Рисунок 1.11)

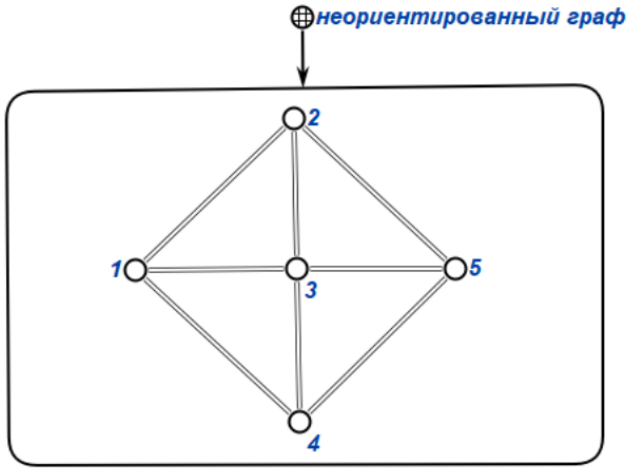


Рисунок *1.11* – Вход четвертого теста

Выход:

Граф является гамильтоновым (Рисунок 1.12).

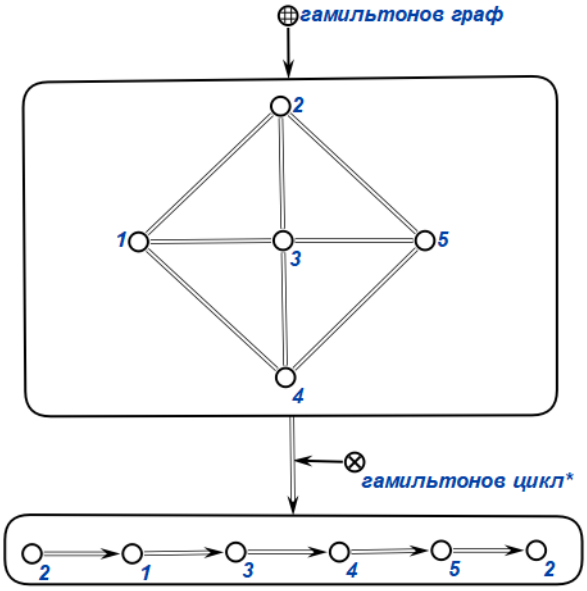


Рисунок *1.12* – Выход четвертого теста

Тест 5

Вход:

Необходимо сопределить является ли данный граф гамильтоновым (Рисунок 1.13)

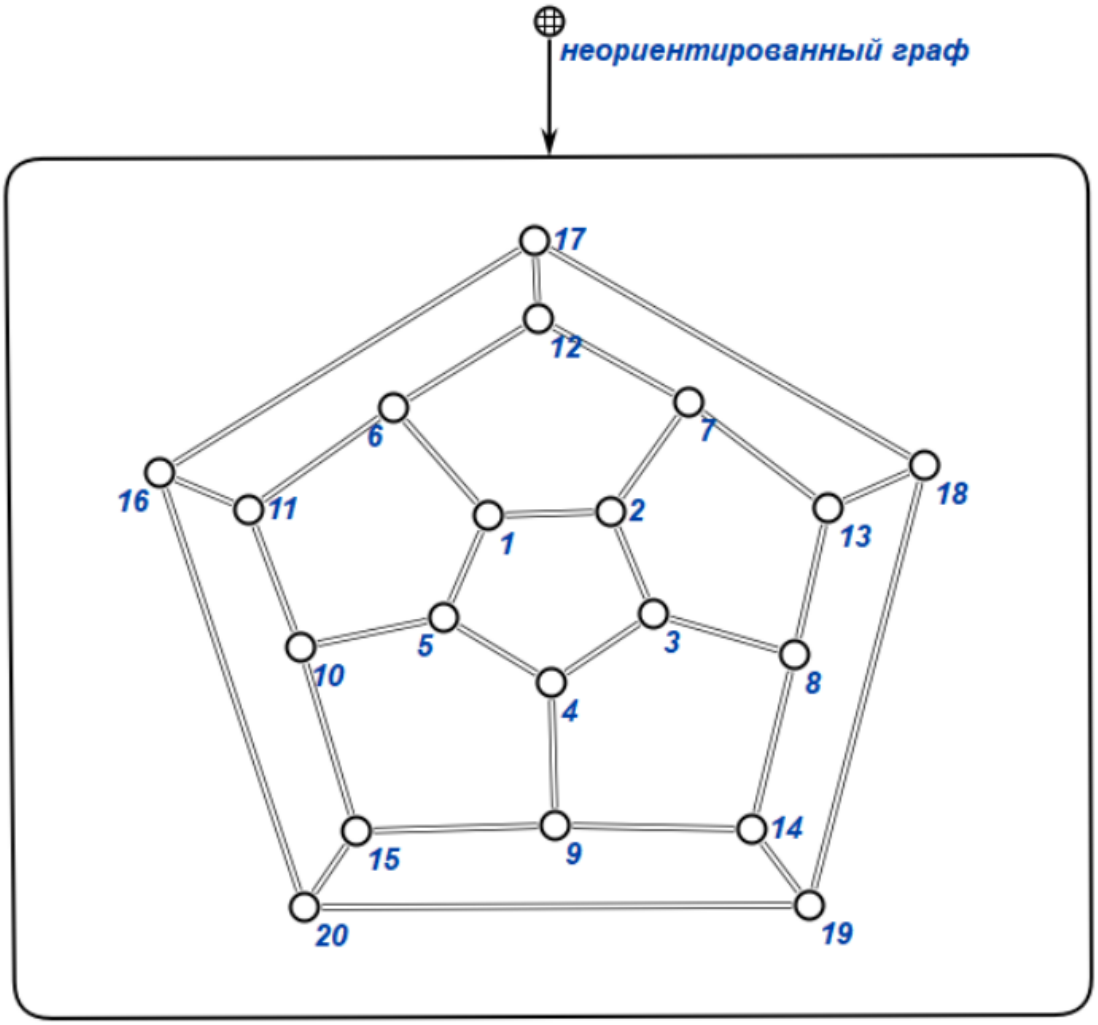


Рисунок *1.13* – Вход пятого теста

Выход:
Граф является гамильтоновым (Рисунок 1.14).

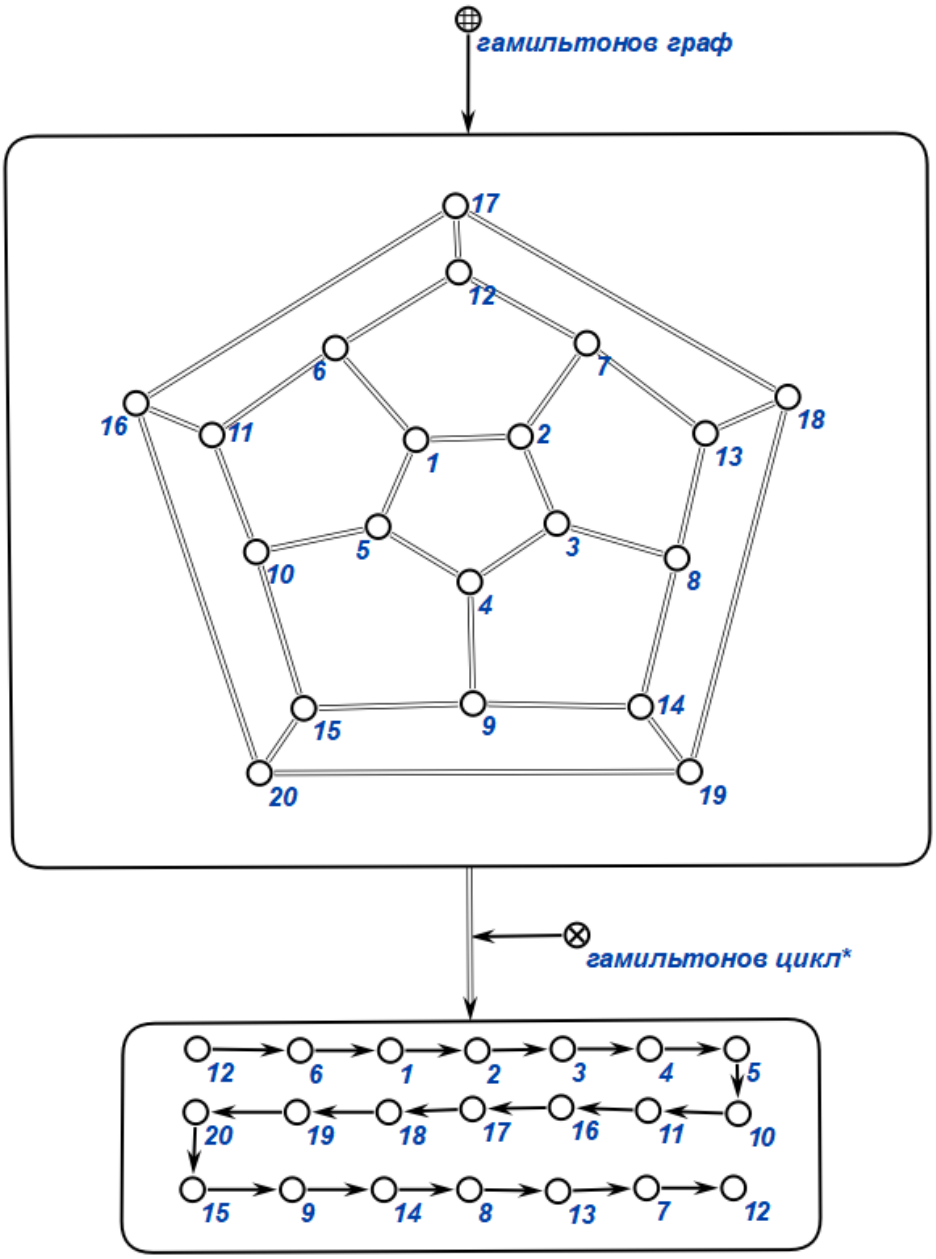


Рисунок *1.14* – Выход пятого теста

1.4 Реализация и демонстрация алгоритма

Для решения задачи необходимы следующие переменные:

1. Множество непроверенных вершин **_not_checked_vertexes**;

2. Множество, которое является результатом выполнения программы **_path**;

3. Переменная, в которой будет находиться рассматриваемая вершина **_current**;

4. Переменная, в которой будет находиться дуга от текущей вершины к предыдущей **_rrel_parent**;

5. Переменная, которая получает в качестве значения начальную вершину **_begin**

6. Переменная, которая получает в качестве значения предыдущую вершину **_prev**

Пример выполнения алгоритма:

Шаг 1. Создаем множества _not_checked_vertexes, _path

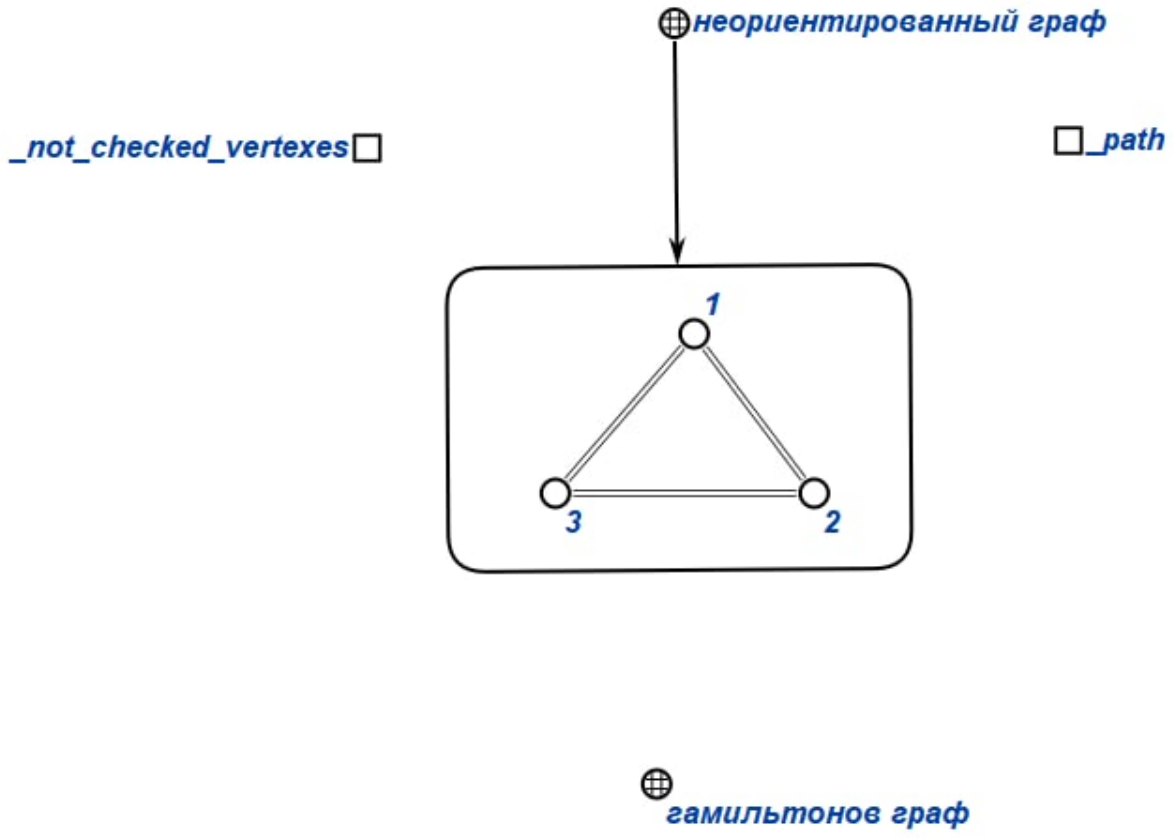


Рисунок *1.15* – Создаем множества _not_checked_vertexes, _path

Шаг 2. Создаем переменные _begin, _current, _prev

Шаг 3. Вносим все вершины графа в _not_checked_vertexes

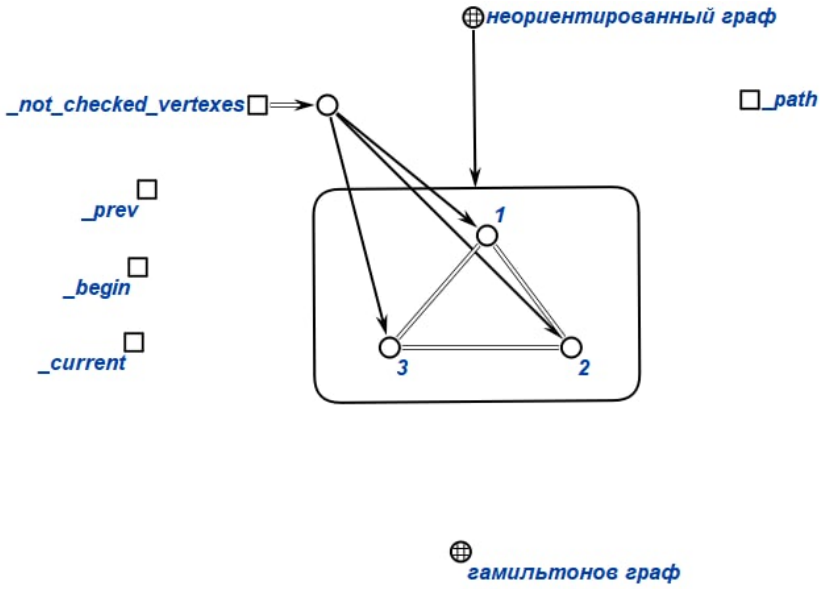


Рисунок *1.16* – Вносим все вершины графа в _not_checked_vertexes

Шаг 4. Вносим начальную вершину в _begin, _current и множество _path, удаляем её из множества _not_checked_vertexes

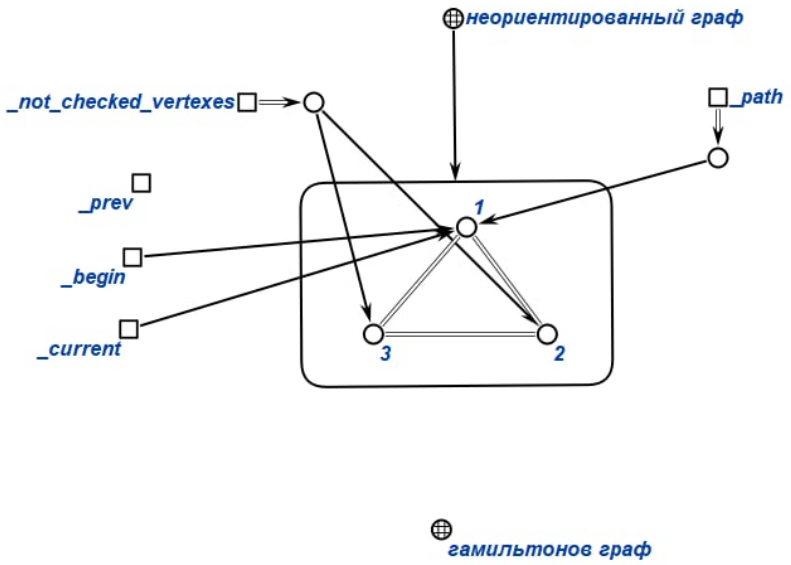


Рисунок *1.17* – Вносим начальную вершину в _begin, _current и множество _path, удаляем её из множества _not_checked_vertexes

Шаг 5. Если существует следующая вершина из множества _not_checked_vertexes, смежная с текущей вершиной _current, переходим к шагу 6, в противном случае переходим к шагу 9

Шаг 6. Если найденная вершина равна начальной _begin, переходим к шагу 10, в противном случае к шагу 7

Шаг 7. Вносим вершину _current в _prev, а найденную вершину в _current

Шаг 8. Вносим дугу между текущей вершиной _current и предыдущей _prev в _rrel_parent

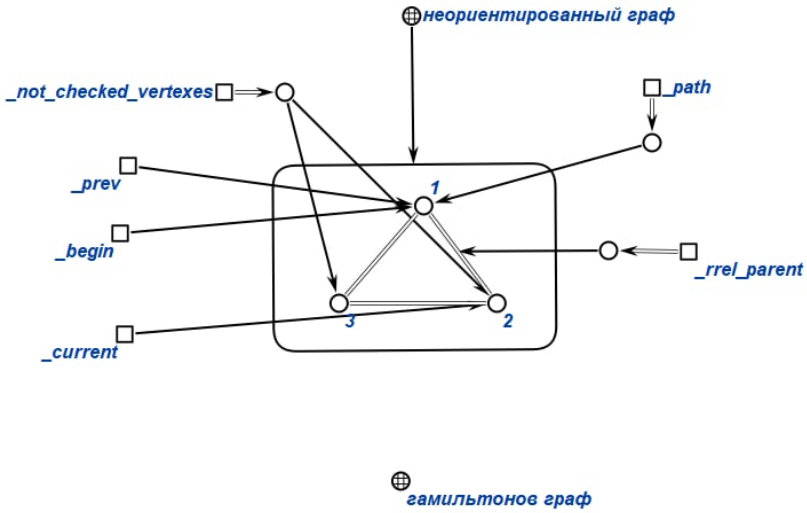


Рисунок *1.18* – Вносим вершину _current в _prev, а найденную вершину в _current и вносим дугу между текущей вершиной _current и предыдущей _prev в _rrel_parent

Шаг 9. Удаляем вершину _current из множества _not_checked_vertexes, добавляем её в _path и переходим к шагу 5

Шаг 10. Если множество _not_checked_vertexes пустое, переходим к шагу 11

Шаг 11. Граф является гамильтоновым.

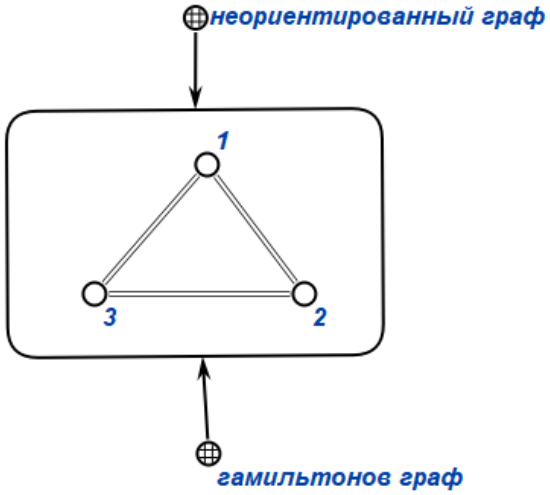


Рисунок *1.19* – Гамильтонов граф

Шаг 12. 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):

```
ScAddr other_vertex = get_other_vertex_incidence_edge(context, t_arc, vertex);

//если вершин нет
if (!other_vertex.IsValid()) {
    std::cout << "Вершина не имеет инцедентных вершин!" << std::endl;
}
```

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:

```
        //Оператор поиска всех вершин (данные вершины будут началом и концом всех циклов)
        //В случае, если не будет найдено ни одной вершины, выходим из программы (может
произойти в случае, если граф не имеет вершин)
        //Если найдено множество, то переходим к следующему пункту (получение вершины)
        -> ..get_all_vertices (*
                 <- searchSetStr5;;
                -> rrel_1: rrel_fixed: rrel_scp_var: _graph;;
                -> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc1;;
                -> rrel_3: rrel_assign: rrel_scp_var: _vertex_of_graph;;
                -> rrel_4: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc2;;
                -> rrel_5: rrel_fixed: rrel_scp_const: rrel_vertex;;
                ->rrel_set_3: rrel_assign: rrel_scp_var: _vertices;;
                => nrel_then: ..get_vertex;;
                => nrel_else: ..print_graph_is_not_founded;;
        *);;
```

Рисунок *2.2* – Представление проверки в коде

Рисунок *2.3* – Вывод сообщения об ошибке

2.2 Документация программы

При написании программы на псевдоязыке SCP я вела документирование программы. Я оставляла комментарии к функциям, где это было необходимо для более детального понимания работы алгоритма.

Описание алгоритма на C++(Рисунок 2.4) :

```
//множество всех дуг
ScAddr set_arcs = all_arcs;
//выбираем дугу из всего множества
ScIterator3Ptr it_arc = context->Iterator3(set_arcs, ScType::EdgeAccessConstPosPerm,  param3: ScType( type: 0));
while (it_arc->Next()) {
    //находим ту самую дугу связи вершинок
    ScAddr t_arc = it_arc->Get( idx: 2);
    //находим соединенную с нашей вершинку
    ScAddr other_vertex = get_other_vertex_incidence_edge(context, t_arc, vertex);
```

Рисунок *2.4* – Описание алгоритма на C++

Я также оставляла комментарии и при работе с языком SCP(Рисунок 2.5)

Вся программа была разделена на несколько составных частей (функций) для удобства работы. Эти функции включают вывод графовых примеров и их поочередный запуск, алгоритм нахождения гамильтонова цикла для определения гамильтонова графа.

```
        //Оператор создания множества всех непроверенных вершин (непроверенными вершинами будут
считаться все вершины графа)
        //Если такое множество создано, то переходим к созданию простого цикла, в которую
занесем начальную вершину
        //Условие, когда не будет создано множество непроверенных вершин, невыполнимо
        -> ..create_set_not_checked_vertices (*
                 <- searchSetStr5;;
                -> rrel_1: rrel_fixed: rrel_scp_var: _graph;;
                -> rrel_2: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc1;;
                -> rrel_3: rrel_assign: rrel_scp_var: _vertex_of_graph;;
                -> rrel_4: rrel_assign: rrel_pos_const_perm: rrel_scp_var: _arc2;;
                -> rrel_5: rrel_fixed: rrel_scp_const: rrel_vertex;;
                ->rrel_set_3: rrel_assign: rrel_scp_var: _not_checked_vertices;;
                => nrel_then: ..create_simple_cycle;;
                => nrel_else: ..return;;
        *);;
```

Рисунок *2.5* – Описание алгоритма на SCP

3 ЛИЧНЫЙ ВКЛАД В РАЗВИТИЕ ИИС ПО КИНОФИЛЬМАМ

3.1 Разработка БЗ для ИИС по кинофильмам

В разделах "Фильмы", "Актеры", "Режиссеры", "Мультфильмы", "Киностудии", "Персонажи" были формализованы следующие абсолютные понятия:
1. Роберт Паттинсон
2. Довод
3. Сумерки
4. Сумерки. Сага. Новолуние
5. Сумерки. Сага. Затмение
6. Сумерки. Сага. Рассвет: Часть 1
7. Сумерки. Сага. Рассвет: Часть 2
8. Эдвард Каллен
9. Белла Свон
10. Дэвид Вашингтон
11. Тимоти Шаламе
12. Флоренс Пью
13. Маленькие женщины
14. Дюна
15. Кэтрин Хардвик
16. Дэвид Слейд
17. Билл Кондон
18. Кристофер Вайц
19. Грета Гервиг
20. Дени Вильнёв
21. Король Англии
22. Черная вдова
23. Дэвид Мишо
24. Гарри Стайлс
25. Вечные
26. Дюнкерк
27. Унесенные призраками
28. Ходячий замок
29. Хаяо Миядзаки
30. студия Гибли

Рассмотрим реализованное на языке SCn абсолютное понятие "Роберт Паттинсон". На рисунке 3.1 изображен фрагмент базы знаний ИСС по кинофильмам, показывающий идентификаторы понятия "Роберт Паттинсон".

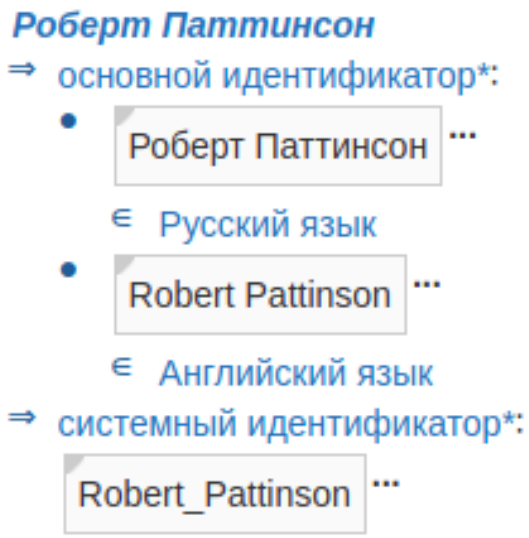
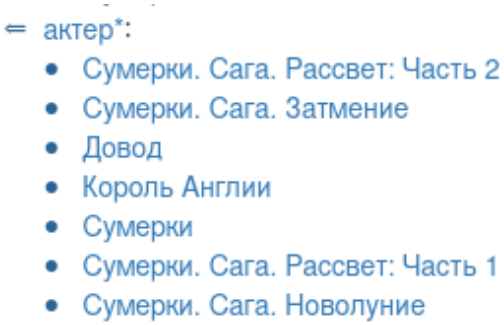


Рисунок *3.1* – Идентификаторы понятия "Роберт Паттинсон"

На рисунке 3.2 изображен фрагмент базы знаний ИСС по кинофильмам, показывающий теоретико-множественные связи понятия "Роберт Паттинсон".



(а)

Рисунок *3.2* – Теоретико-множественные связи понятия "Роберт Паттинсон"

На рисунке 3.3 изображен фрагмент базы знаний ИСС по кинофильмам, показывающий описание понятия "Роберт Паттинсон".

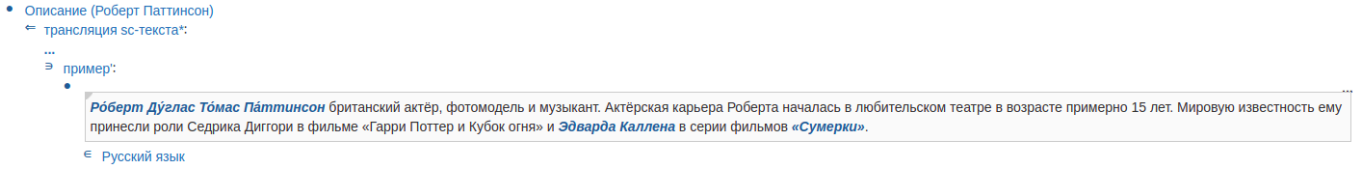


Рисунок *3.3* – Описание понятия "Роберт Паттинсон"

На рисунке 3.4 изображен фрагмент базы знаний ИСС по кинофильмам, показывающий изображение понятия "Роберт Паттинсон.

Рисунок *3.4* – Изображение понятия "Роберт Паттинсон"

ЗАКЛЮЧЕНИЕ

В рамках курсовой работы были решены следующие задачи:

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

— Дополнены разделы "Фильмы", "Актеры", "Режиссеры", "Мультфильмы", "Киностудии", "Персонажи" в ИИС по кинофильмам

— Формализовано 30 понятий в разделах "Фильмы", "Актеры", "Режиссеры", "Мультфильмы", "Киностудии", "Персонажи".

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] М.И. Нечипуренко В.К. Попков, С.М. Майнагашев и др. Алгоритмы и программы решения задач на графах и сетях / С.М. Майнагашев и др. М.И. Нечипуренко, В.К. Попков; Сиб. отд-ние. — Наука, 1990. — 515 с.

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

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

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

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

[8] О.П., Новожилов. Электротехника (теория электрических цепей) в 2 ч. Часть 1. Учебник для СПО / Новожилов О.П.; Издательство Юрайт. — (Профессиональное образование), 2019. — 403 с.

[9] Д.В., Карпов. Теория графов / Карпов Д.В. — 2017. — 525 с.

[10] А.А., Казанский. Дискретная математика. Краткий курс. Учебное пособие / Казанский А.А.; Издательство "Проспект". — 2015. — 163 с.