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 «Традиционные и интеллектуальные информационные технологии»

на тему

Задача нахождения гамильтонова графа

Выполнил: П. А. Белоус

Студент группы
921702

Проверил: Д. В. Шункевич

Минск 2020

Цель: Получить навыки формализации и обработки информации с использованием семантических сетей

Задача: Определить, является ли граф гамильтоновым.

1 СПИСОК ПОНЯТИЙ

1. Графовая структура (абсолютное понятие) - это такая одноуровневая реляционная структура, объекты которой могут играть роль либо вершины, либо связки:
   a. Вершина (относительное понятие, ролевое отношение);
   b. Связка (относительное понятие, ролевое отношение).

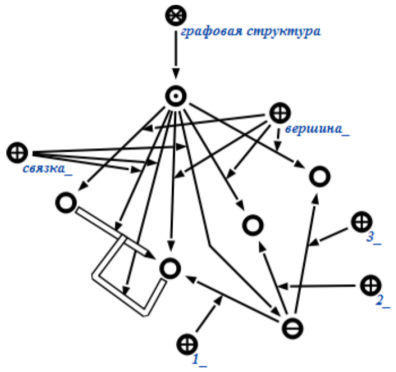

Рисунок *1.1* – Графовая структура

2. Графовая структура с неориентированными связками (абсолютное понятие)
   a. Неориентированная связка (относительное понятие, ролевое отношение) – связка, которая задается неориентированным множеством.

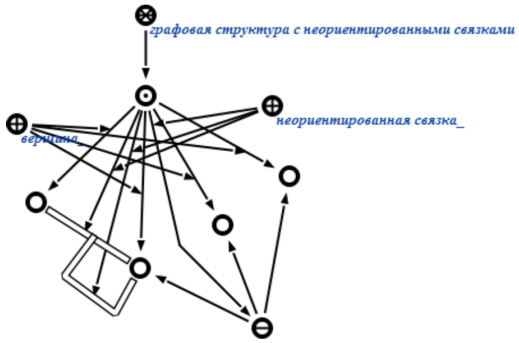

Рисунок *1.2* – Графовая структура с неор. связками

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

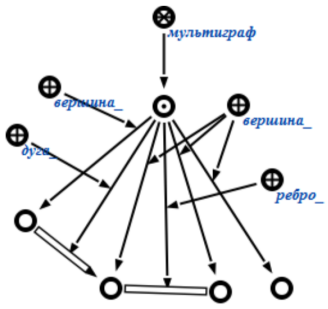

Рисунок *1.3* – Граф

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

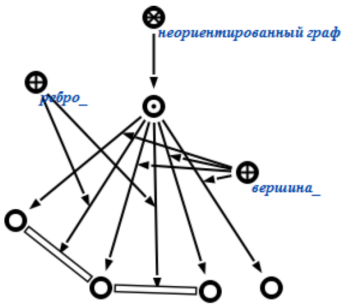

Рисунок *1.4* – Неориентированный граф

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

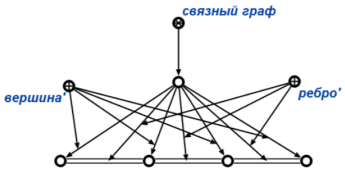

Рисунок *1.5* – Связный граф

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

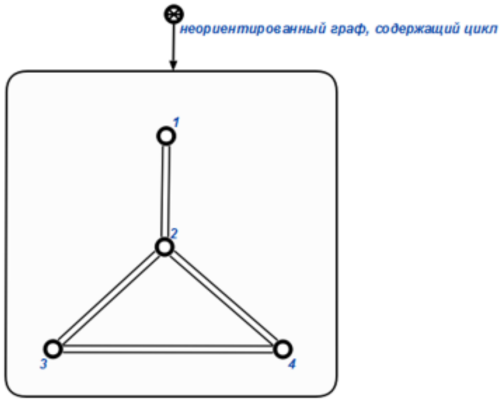

Рисунок *1.6* – Цикл

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

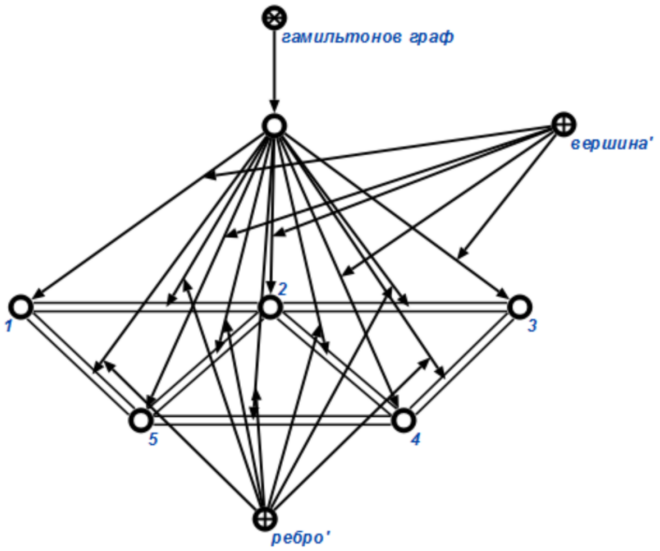

Рисунок *1.7* – Гамильтонов граф

2 ТЕСТОВЫЕ ПРИМЕРЫ

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

2.1 Тест 1

Вход:
Необходимо определить является ли данный граф гамильтоновым.

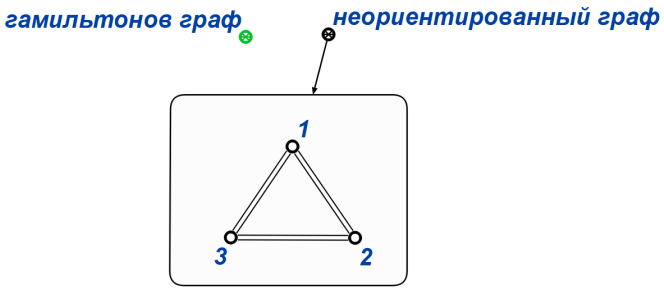

Рисунок *2.1* – Вход теста 1

Выход:

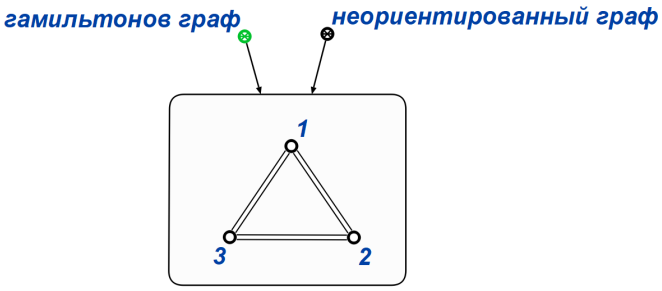

Рисунок *2.2* – Выход теста 1

Граф был определен как гамильтонов.

2.2 Тест 2

Вход:
Необходимо определить является ли данный граф гамильтоновым.

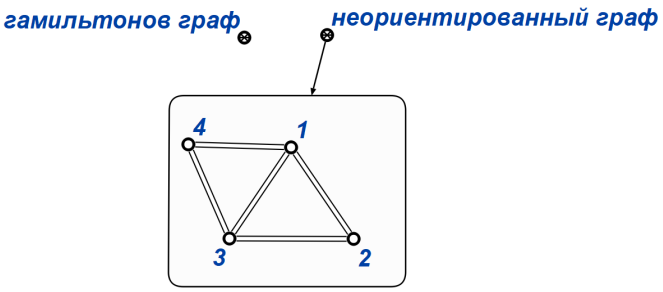

Рисунок *2.3* – Вход теста 2

Выход:
Граф был определен как гамильтонов.

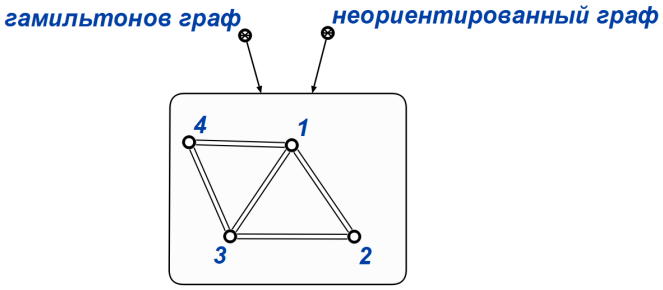

Рисунок *2.4* – Выход теста 2

2.3 Тест 3

Вход:
Необходимо определить является ли данный граф гамильтоновым.

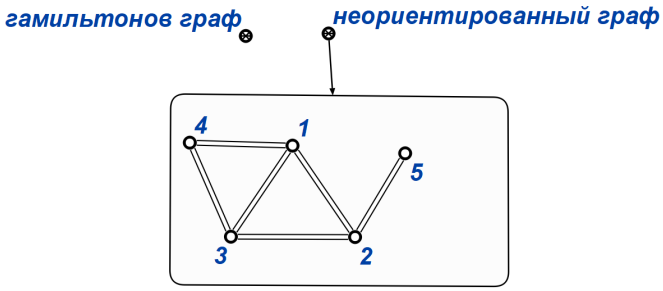

Рисунок *2.5* – Вход теста 3

Выход:
Граф был определен как негамильтонов.

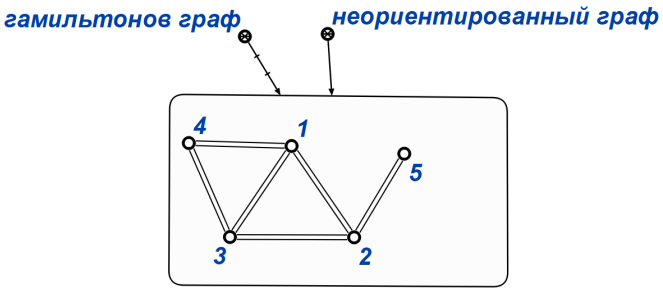

Рисунок *2.6* – Выход теста 3

2.4 Тест 4

Вход:
Необходимо определить является ли данный граф гамильтоновым.

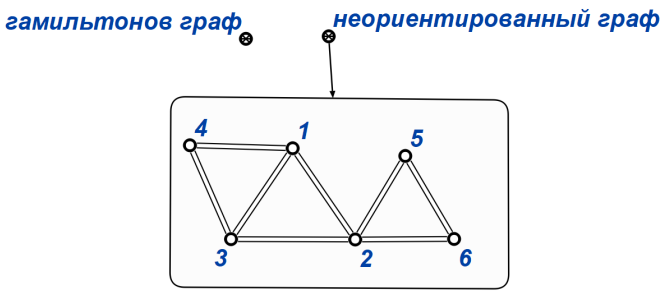

Рисунок *2.7* – Вход теста 4

Выход:
Граф был определен как негамильтонов.

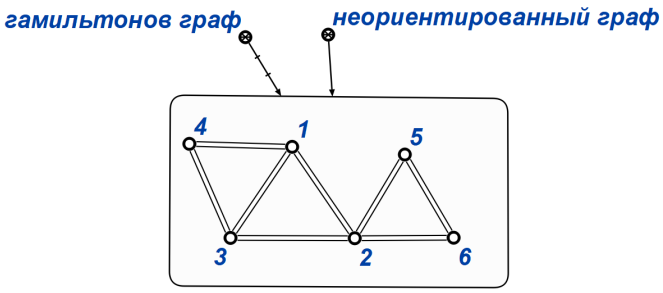

Рисунок *2.8* – Выход теста 4

2.5 Тест 5

Вход:
Необходимо определить является ли данный граф гамильтоновым.

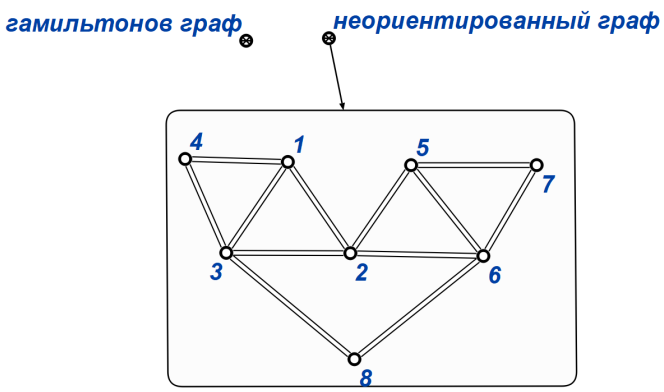

Рисунок *2.9* – Вход теста 5

Выход:
Граф был определен как гамильтонов.

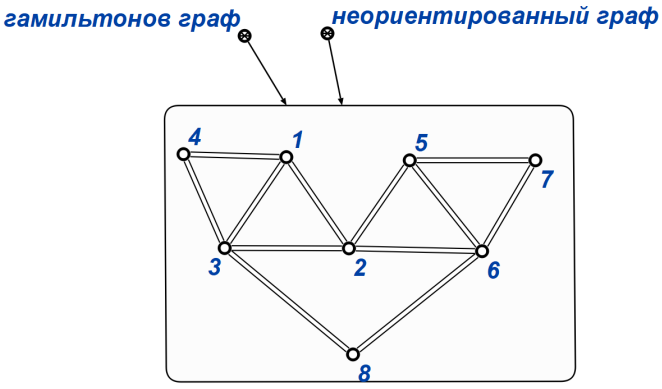

Рисунок *2.10* – Выход теста 5

3 ПРИМЕР РАБОТЫ АЛГОРИТМА В СЕМАНТИЧЕСКОЙ ПАМЯТИ

3.1 Переменные для решения задачи:

_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.

3.2 Пример выполнения алгоритма в sc-памяти:

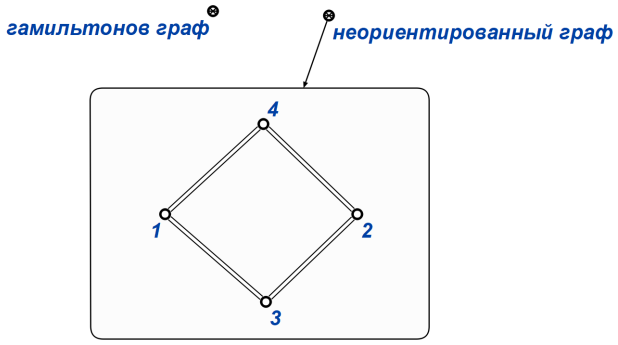

Рисунок *3.1* – Исходный граф

Шаг 1. Создаем множества _vertex, _notchecked, _prev;
Шаг 2. Создаем переменные _begin, _current и счетчик _vertexcount;

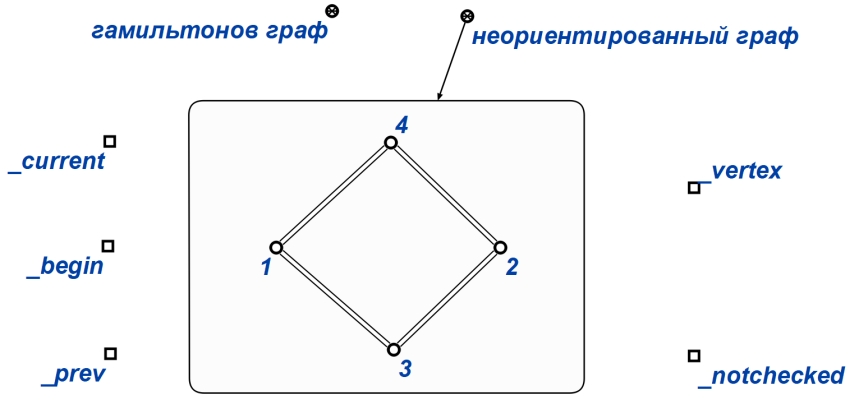

Рисунок *3.2* – Создание переменных _begin, _current, счетчика _vertexcount и множеств _vertex, _notchecked, _prev

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

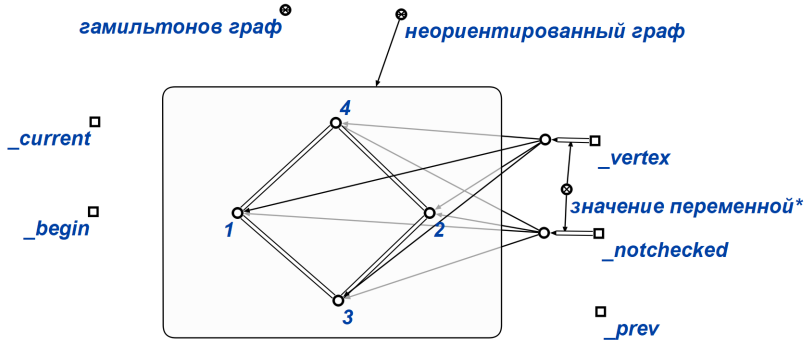

Рисунок *3.3* – Внесение всех вершин во множества _vertex и _notchecked

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;

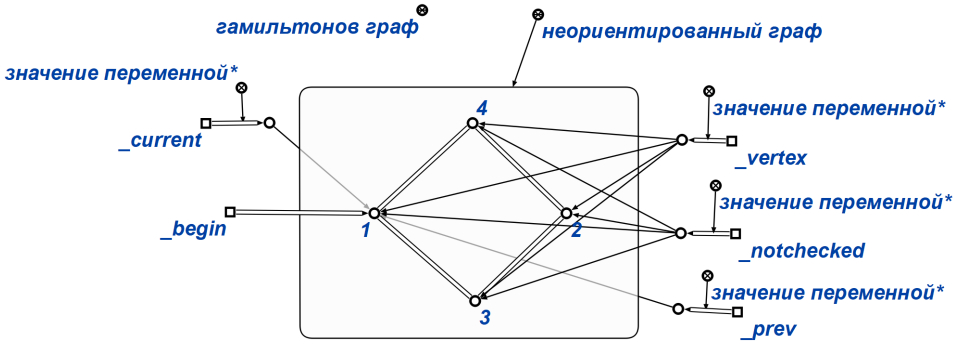

Рисунок *3.4* – Занесение вершины _begin в переменную _current и во множество _prev

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;

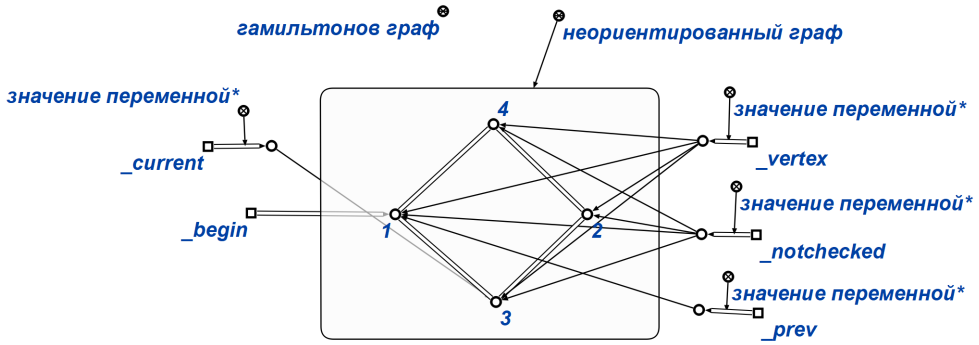

Рисунок *3.5* – Внесение вершины смежной с вершиной _current в переменную _current

Шаг 10. Инкрементируем счетчик _vertexcount;

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

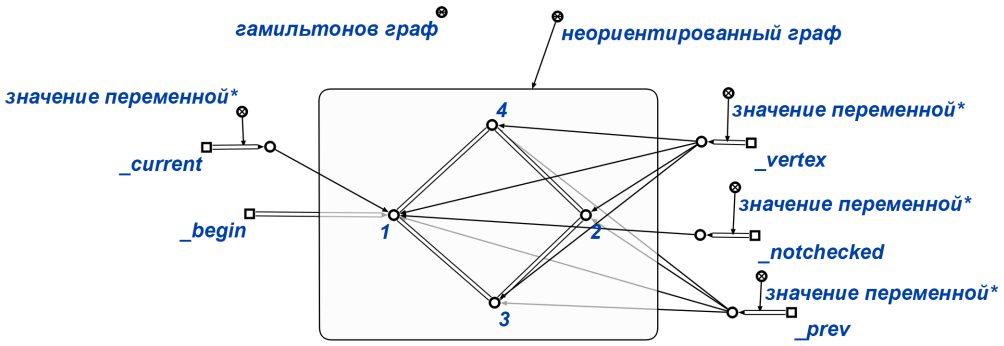

Рисунок *3.6* – Проверка, совпадает ли вершина _current с переменной _begin

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.

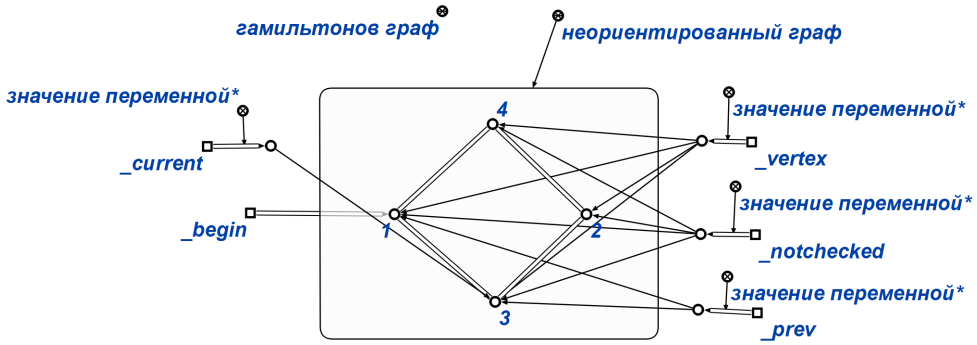

Рисунок *3.7* – Внесение текущей вершины _current в _prev

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

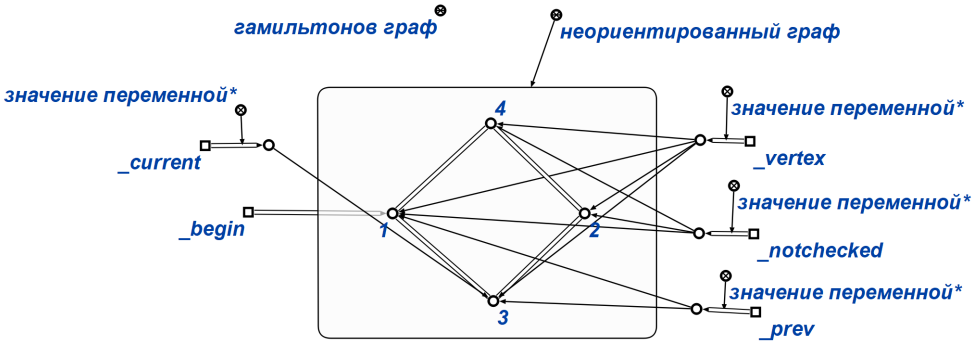

Рисунок *3.8* – Удаление текующей вершины _current из _notchecked

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

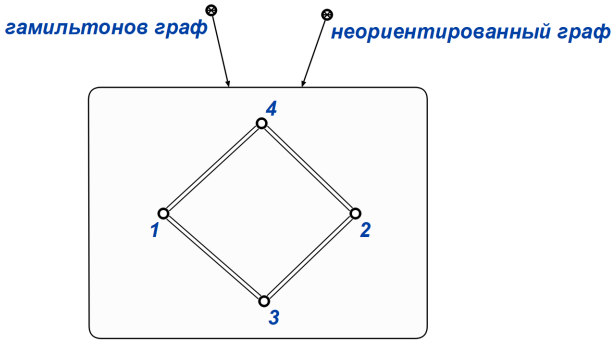

Рисунок *3.9* – Гамильтонов граф

Шаг 21. Завершение алгоритма.

4 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Емеличев В.А. — лекции по теории графов — глава 6.

[2] Diestel — Graph Theory 2005. — P. 83.

[3] Харарри, Ф. Теория графов / Ф. Харарри. — Эдиториал УРСС, 2018. — P. 126.

[4] Оре, О. Теория графов /О. Оре. — Наука, 1980. — P. 336.