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 поиска наибольшего пути во взвешенном ориентированном графе

Выполнил Старостин П.Г.
студент группы
021704

Проверил Витязь В.С

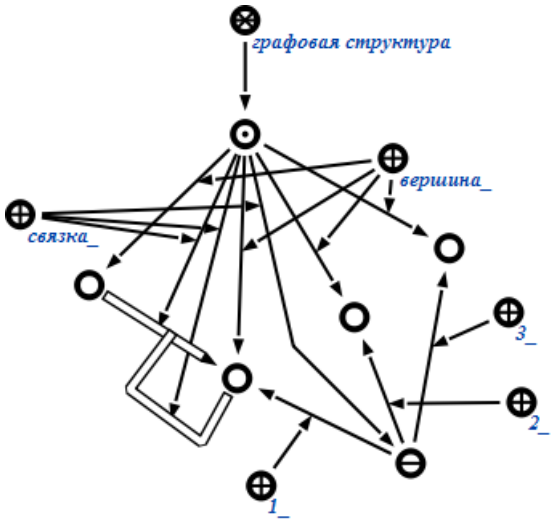
Минск 2021

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

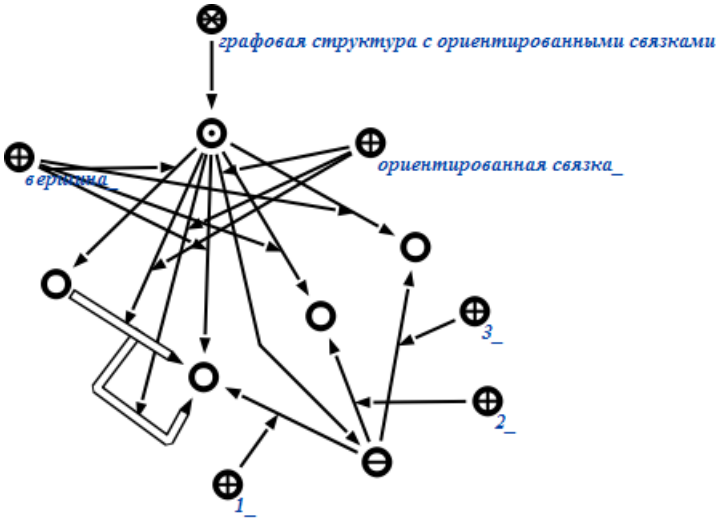
Задача: поиск наибольшего пути во взвешенном ориентированном графе

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).

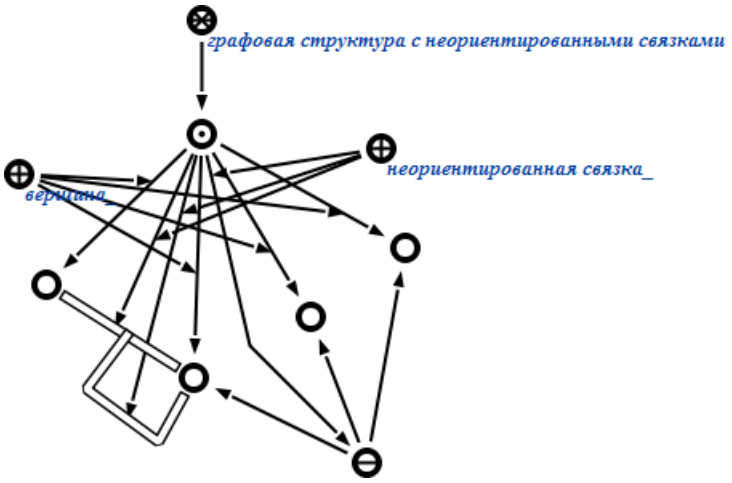


2. Графовая структура с ориентированными связками (абсолютное понятие)
    a. Ориентированная связка (относительное понятие, ролевое отношение) – связка, которая задается ориентированным множеством.

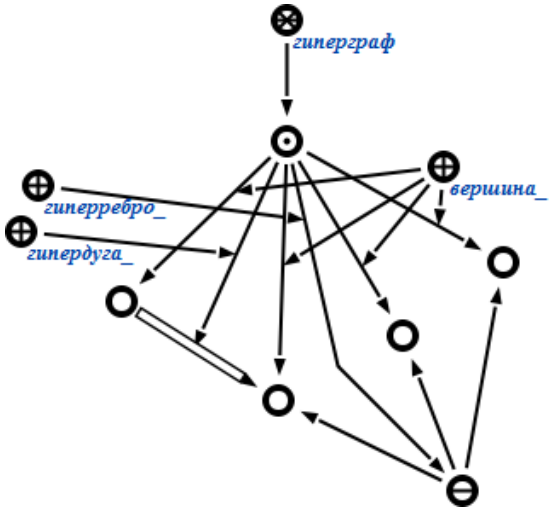


3. Графовая структура с неориентированными связками (абсолютное понятие)

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

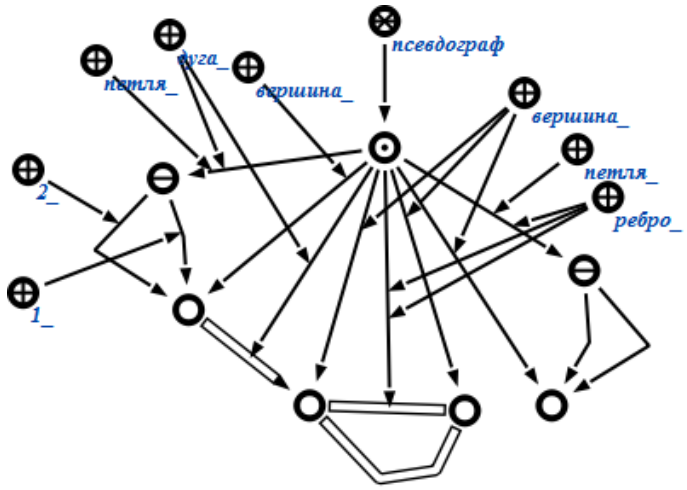


4. Гиперграф (абсолютное понятие) – это такая графовая структура, в которой связки могут связывать только вершины:
   a. Гиперсвязка (относительное понятие, ролевое отношение);
   b. Гипердуга (относительное понятие, ролевое отношение) – ориентированнаягиперсвязка;
   c. Гиперребро (относительное понятие, ролевое отношение) – неориентированнаягиперсвязка.

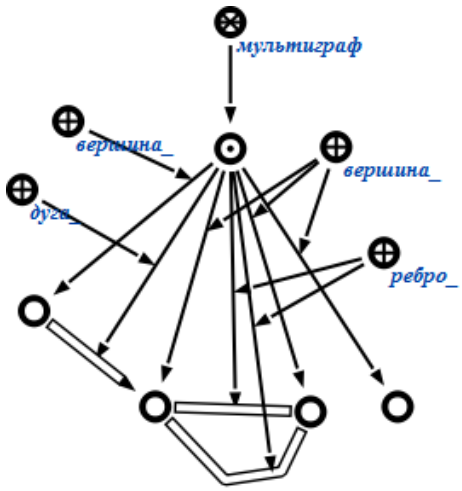


5. Псевдограф (абсолютное понятие) – это такой гиперграф, в котором все связки должны быть бинарными:
   a. Бинарная связка (относительное понятие, ролевое отношение) –гиперсвязка арности 2;
   b. Ребро (относительное понятие, ролевое отношение) – неориентированнаягиперсвязка;
   c. Дуга (относительное понятие, ролевое отношение) – ориентированная гиперсвязка;

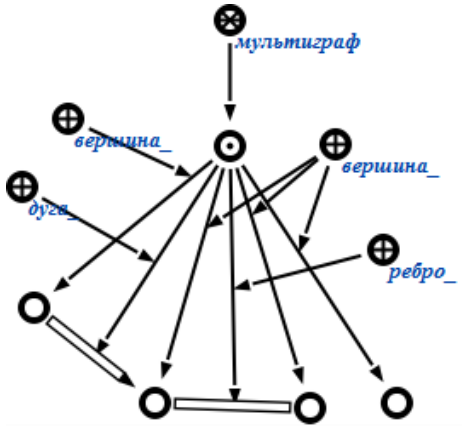
d. Петля (относительное понятие, ролевое отношение) – бинарная связка, у которой первый и второй компоненты совпадают.



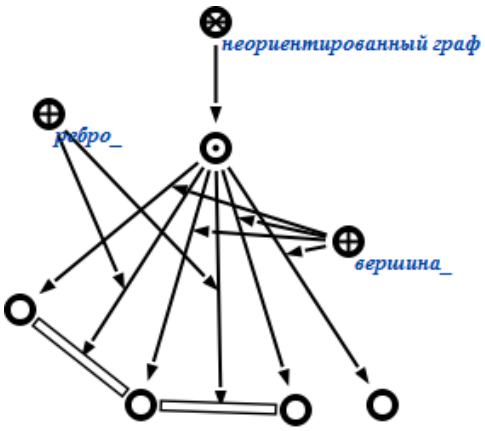
6. Мультиграф (абсолютное понятие) – это такой псевдограф, в котором не может быть петель:



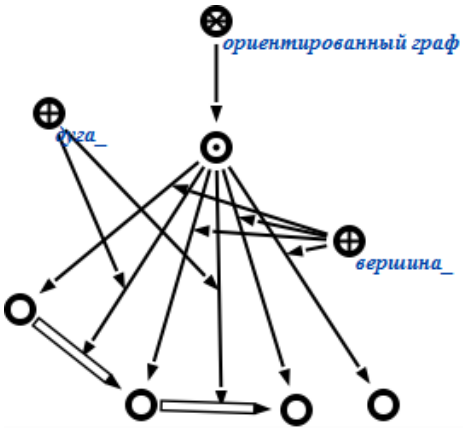
7. Граф (абсолютное понятие) – это такой мультиграф, в котором не может быть кратных связок, т.е. связок у которых первый и второй компоненты совпадают:



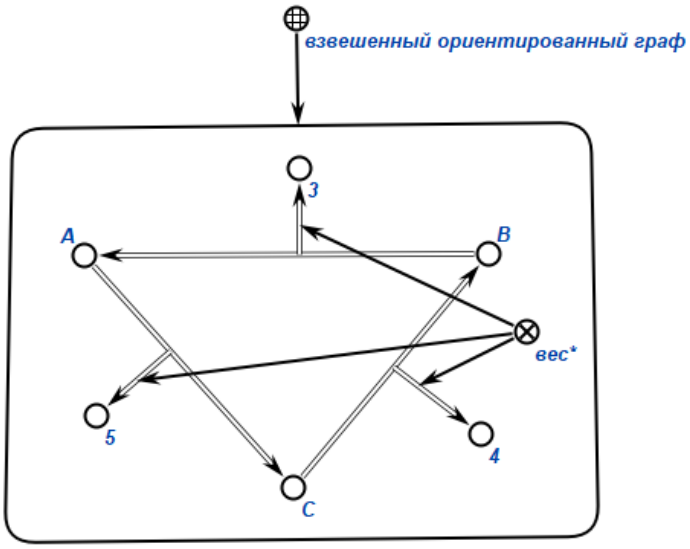
8. Неориентированный граф (абсолютное понятие) –это такой граф, в котором все связки являются ребрами:



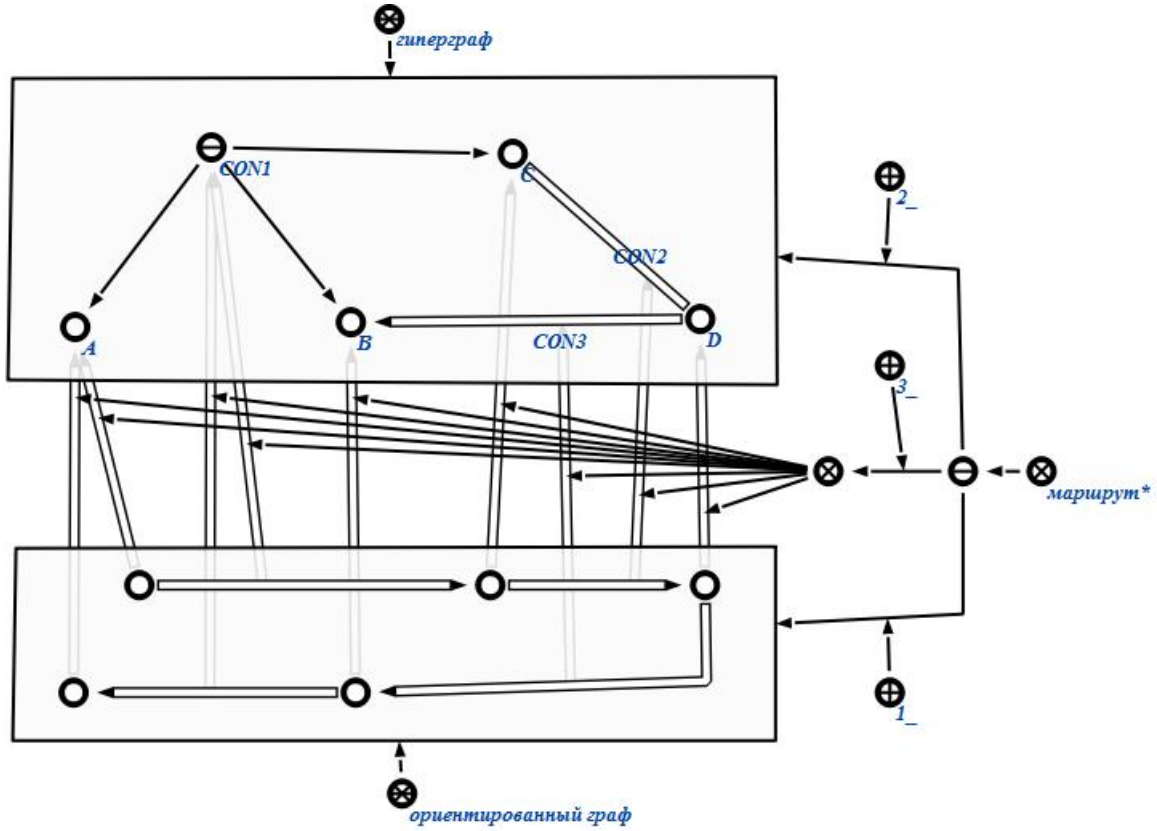
9. Ориентированный граф (абсолютное понятие) - это такой граф, в котором все связки являются дугами:



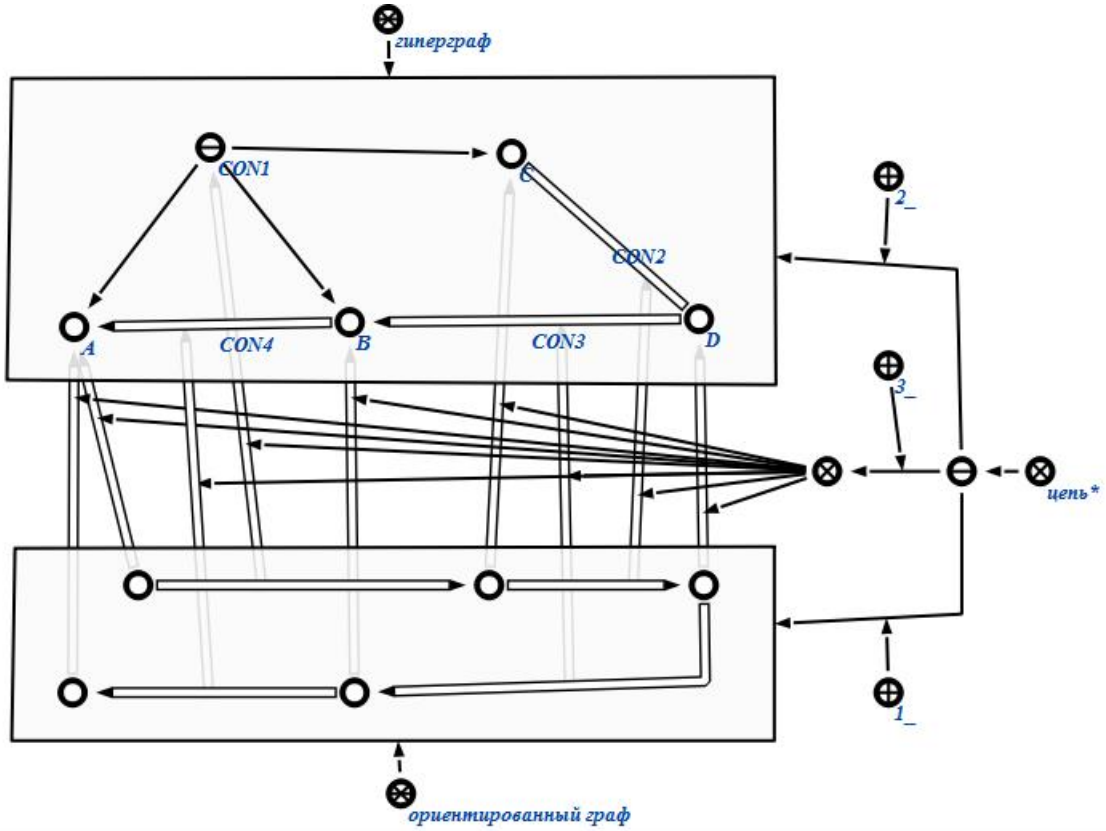
10. Взвешенный граф (абсолютное понятие) - это такой граф, в котором каждому ребру поставлено в соответствие некое значение (вес ребра):



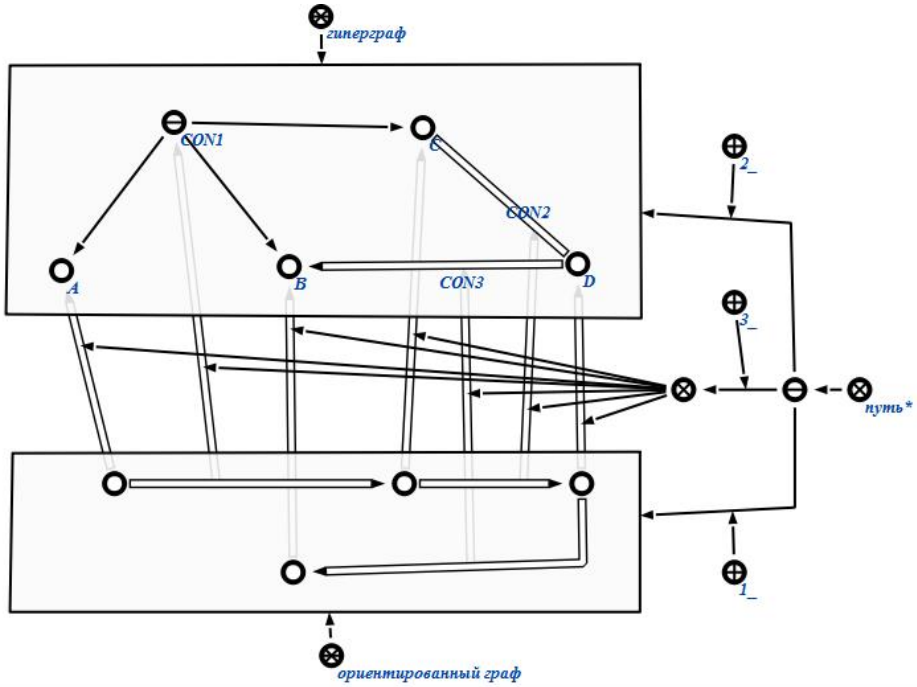
11. Маршрут (относительное понятие, бинарное ориентированное отношение) – это чередующаяся последовательность вершин и гиперсвязок в гиперграфе, которая начинается и кончается вершиной, и каждая гиперсвязка последовательности инцидентна двум вершинам, одна из которых непосредственно предшествует ей, а другая непосредственно следует за ней. В примере ниже показан маршрут *A*, *CON1*, *C*, *CON2*, *D*, *CON3*, *B*, *CON1*, *A* в гиперграфе.



12. Цепь (относительное понятие, бинарное ориентированное отношение) – это маршрут, все гиперсвязки которого различны. В примере ниже показана цепь *A*, *CON1*, *C*, *CON2*, *D*, *CON3*, *B*, *CON4*, *A* в гиперграфе.



13. Простая цепь, путь (относительное понятие, бинарное ориентированное отношение) – это цепь, в которой все вершины различны. В примере ниже показан путь *A*, *CON1*, *C*, *CON2*, *D*, *CON3*, *B* в гиперграфе.



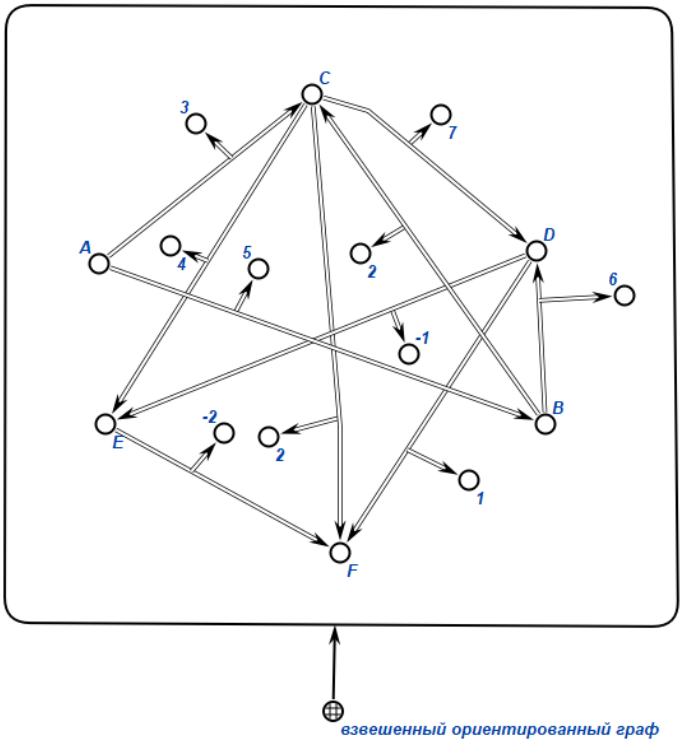
2 Тестовые примеры

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

2.1 Тест 1

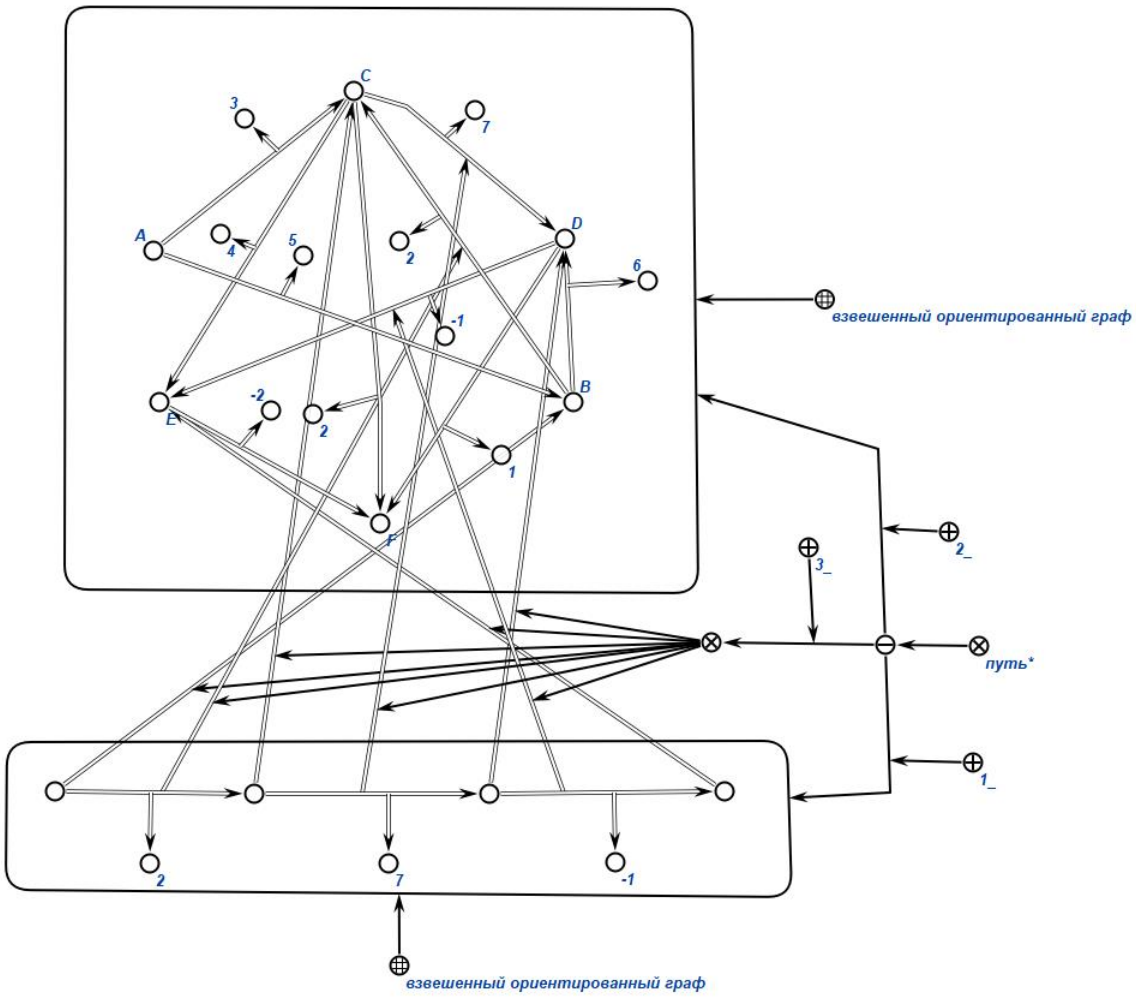
Вход:

Необходимо найти максимальный путь между вершинами B и E.



Выход:

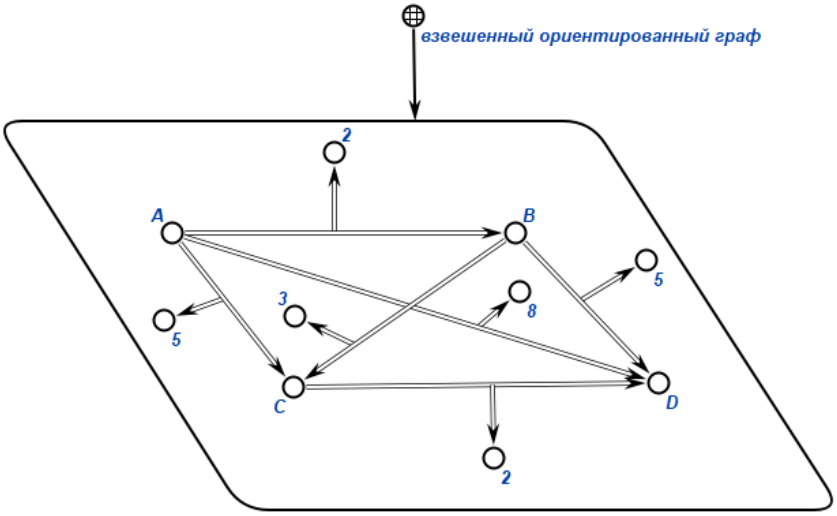
Будет найден путь *B*, *C*, *D*, *E*:



2.2 Тест 2

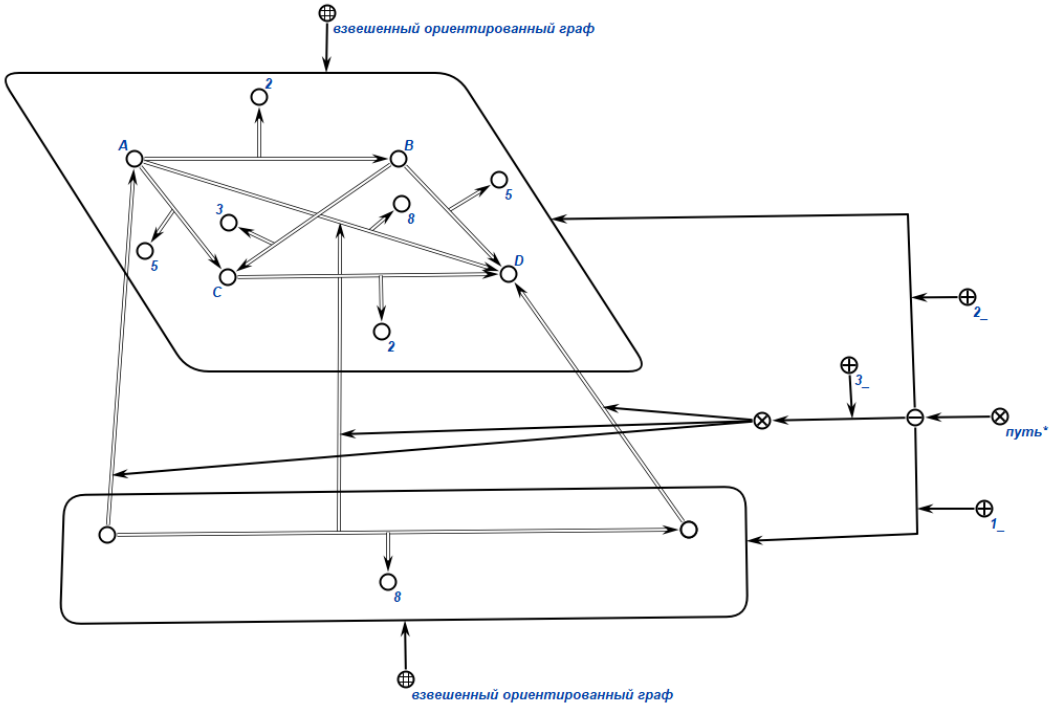
Вход:

Необходимо найти максимальный путь между вершинами A и D.



Выход:

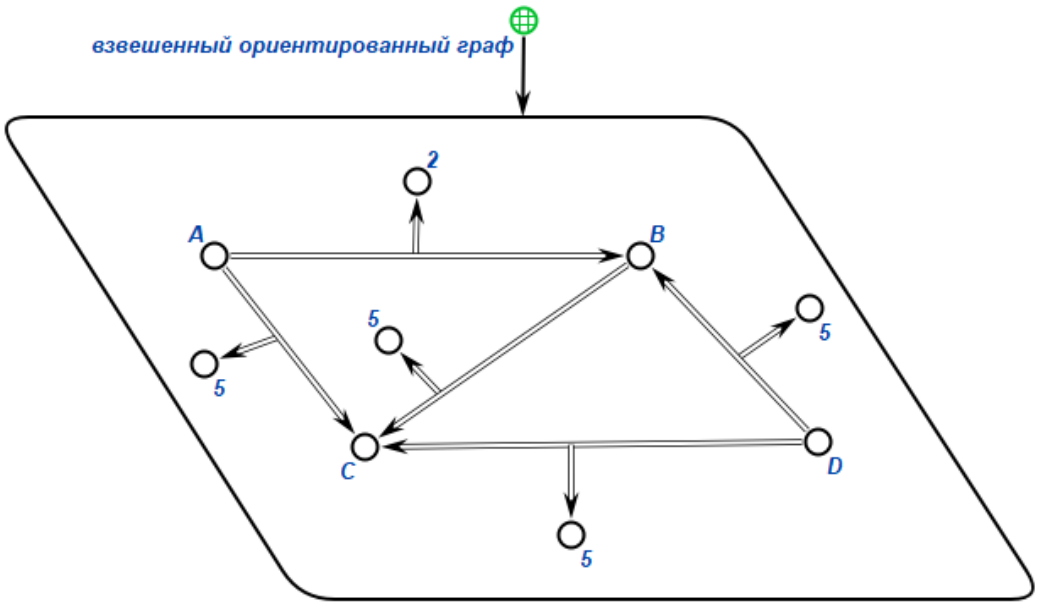
Будет найден путь *A,D*:



2.3 Тест 3

Вход:

Необходимо найти максимальный путь между вершинами A и D.



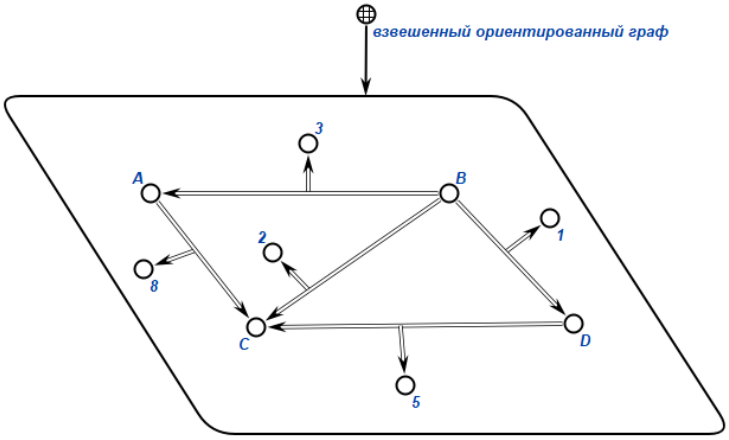
Выход:

Пути между вершинами A и D не существует. Программа должна вернуть ошибку вызывающему контексту.

2.4 Тест 4

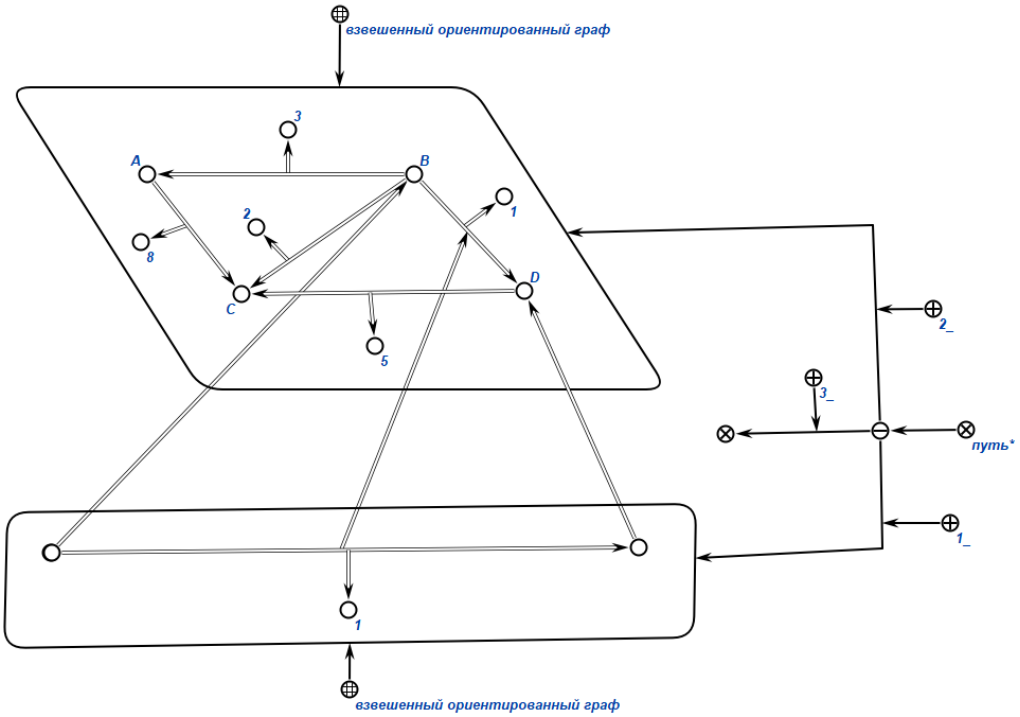
Вход:

Необходимо найти максимальный путь между вершинами B и D.



Выход:

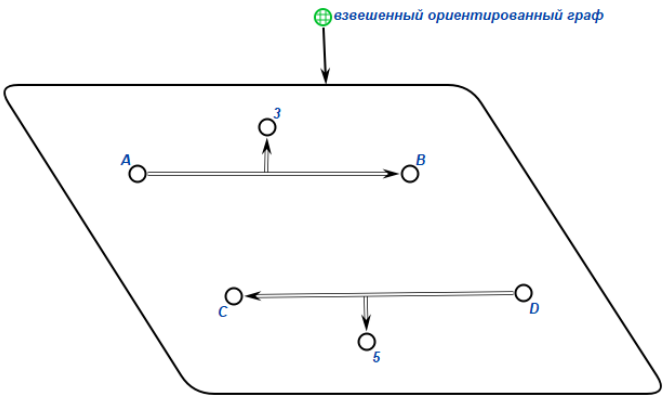
Будет найден путь *B,D*



2.5 Тест 5

Вход:

Необходимо найти максимальный путь между вершинами A и D.

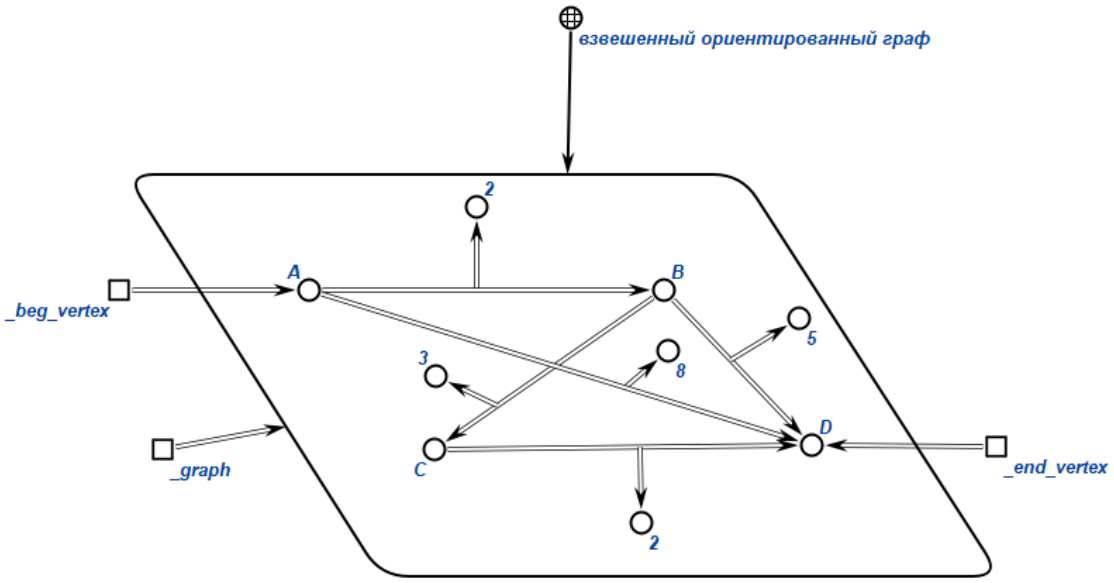


Выход:

Пути между вершинами A и D не существует. Программа должна вернуть ошибку вызывающему контексту.

3 Описание алгоритма

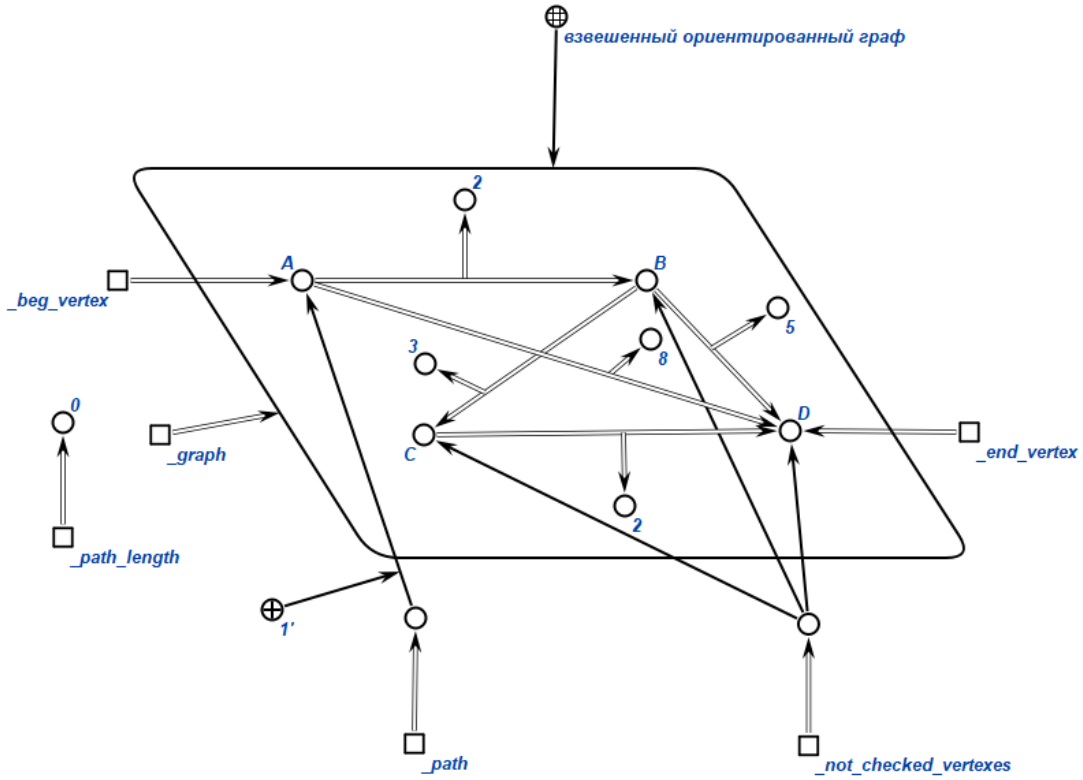
1. Задание входного графа, начальной и конечной вершины пути для работы алгоритма



Переменные изменятся следующим образом:

- _graph получит в качестве значения sc-узел ориентированного взвешенного графа;
- _beg_vertex получит в качестве значения вершину A, которая будет начальной для поиска максимального пути;
- _end_vertex получит в качестве значение вершину D, которая будет конечной для поиска максимального пути.

2. Создание ориентированного множества пути, переменной для хранения длины пути и создание множества неиспользованных вершин



Переменная _path получит в качестве значения будущее ориентированное множество вершин в порядке следования в текущем пути. Будем использовать его для проверки, используется ли уже проверяемая вершина, и для хранения текущего исследуемого пути.

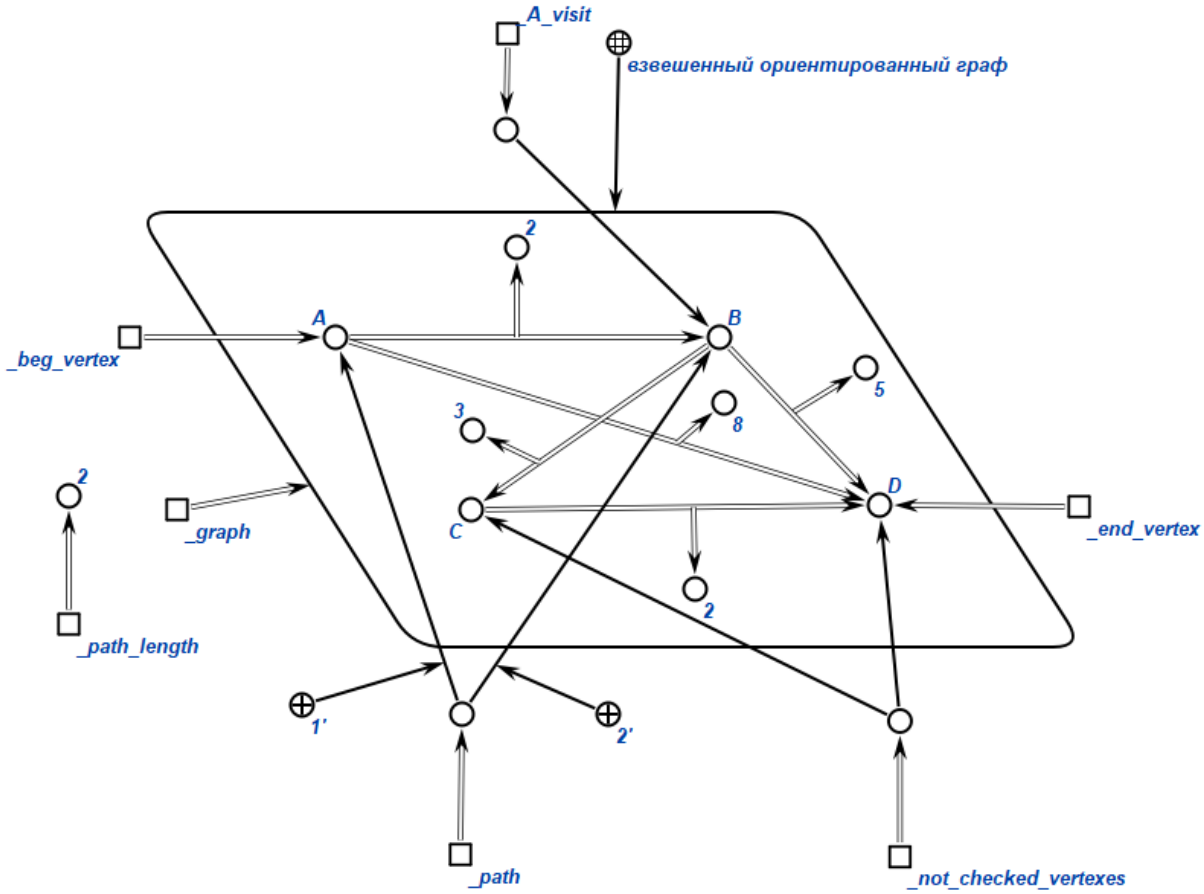
Добавляем первую вершину: это всегда будет вершина A, так как это начальная вершина. Указываем ее порядок следования во множестве _path.

Переменная _path_length получает текущую длину пути. Так как путь сейчас состоит из одной начальной вершины, значение длины равно нулю.

Переменная _not_checked_vertexes получит в качестве значения множество непроверенных вершин обрабатываемого графа (в это множество не включена начальная вершина пути A). Мы не можем использовать одинаковые вершины несколько раз.

По множеству неиспользованных вершин проверяем, с какими вершинами имеет связи вершина A. Это вершины B и D.

3. Добавление во множество используемых вершин вершины B



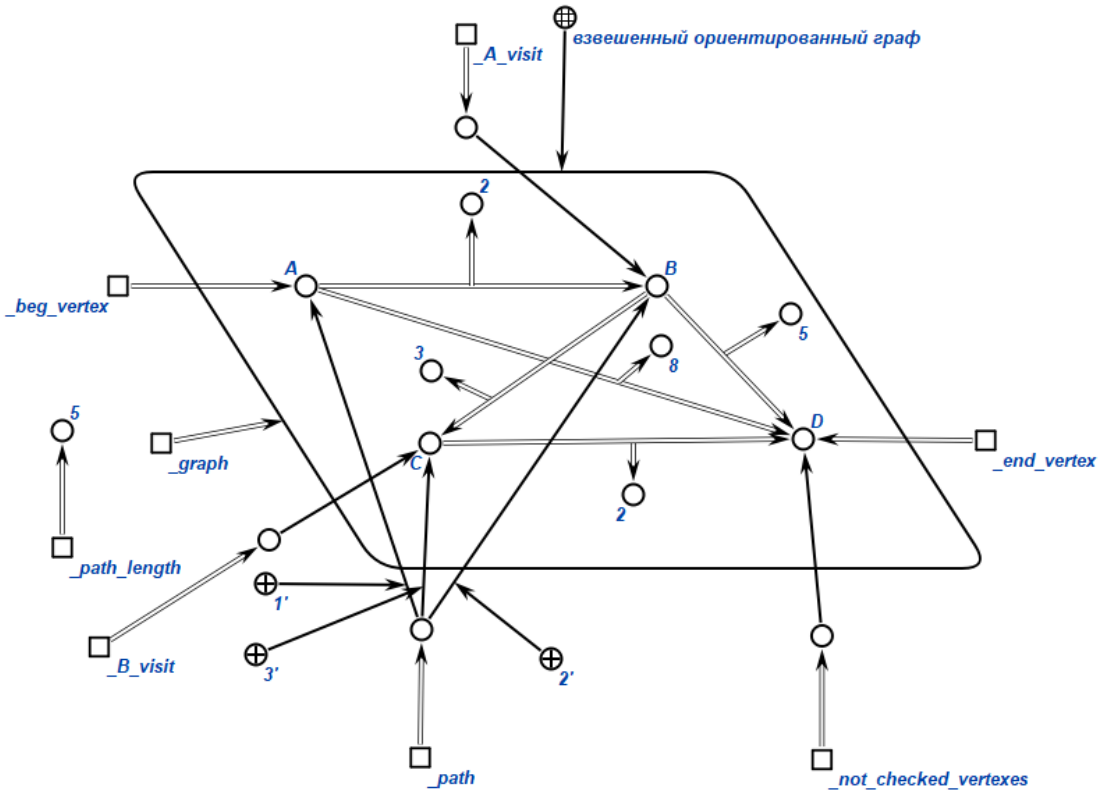
Из двух вершин, с которыми имеет связи выбираем первую - B. Добавляем ее во множество пути с порядковым номером 2. Создаем множество посещенных вершин для вершины A. Добавляем во множество посещенных вершин из вершины A вершину B.

Длину пути изменяем на значение веса ребра AB. Длина текущего пути – 2.

Удаляем из множества неиспользованных вершин вершину B.

Вершина C имеет связь с вершинами: C и D.

4. Добавление во множество используемых вершин вершины C



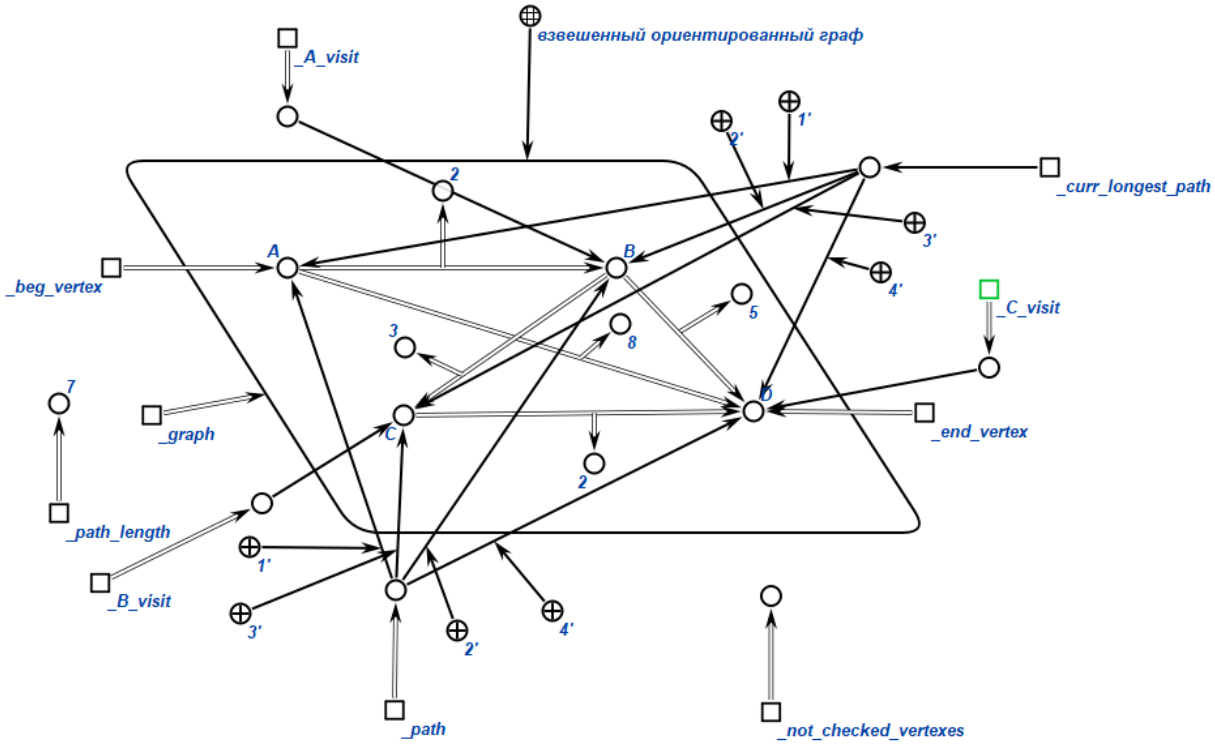
Добавляем вершину C во множество пути с порядковым номером 3. Создаем множество посещенных вершин для вершины B. Добавляем туда вершину C.

Удаляем из множества неиспользованных вершин вершину C.

Изменяем длину текущего пути на значение веса ребра BC. Длина текущего пути - 5.

Вершина C имеет связь с единственной вершиной: D.

5. Добавление во множество используемых вершин вершины D и создание ориентированного множества текущего максимального пути



Добавляем вершину D во множество пути с порядковым номером 4. Создаем множество посещенных вершин для вершины C. Добавляем туда вершину D.

Удаляем из множества неиспользованных вершин вершину D.

Изменяем длину текущего пути на значение веса ребра CD. Длина текущего пути - 7.

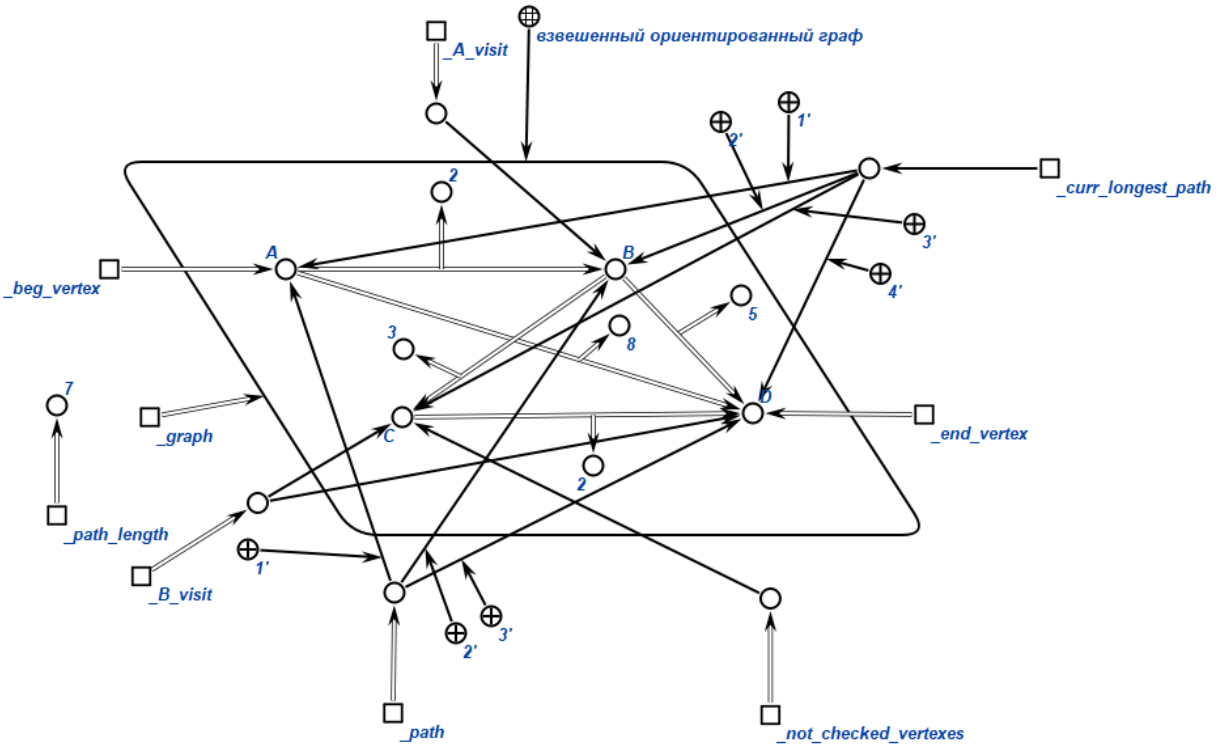
Мы пришли в финальную вершину, однако необходимо проверить все возможные пути из A до D, чтобы выбрать из них самый длинный.

Удаляем вершину D из множества пути и добавляем во множество неиспользованных вершин. Уменьшаем длину текущего пути на 2. _C_visit все еще хранит D. Неисследованных путей из C нет.

Удаляем вершину C из множества пути и добавляем во множество неиспользованных вершин. Уменьшаем длину текущего пути на 3. Удаляем _C_visit. _B_visit все еще хранит C.

Вершина B имеет еще одну неисследованную связь с вершиной D, так как_B_visit не хранит вершину D.

6. Добавление во множество используемых вершин вершины D и проверка, является ли новый путь длиннее имеющегося



Добавляем вершину D во множество пути с порядковым номером 3. В _B_visit добавляем вершину D.

Удаляем из множества неиспользованных вершин вершину D.

Изменяем длину текущего пути на значение веса ребра BD. Длина текущего пути - 7.

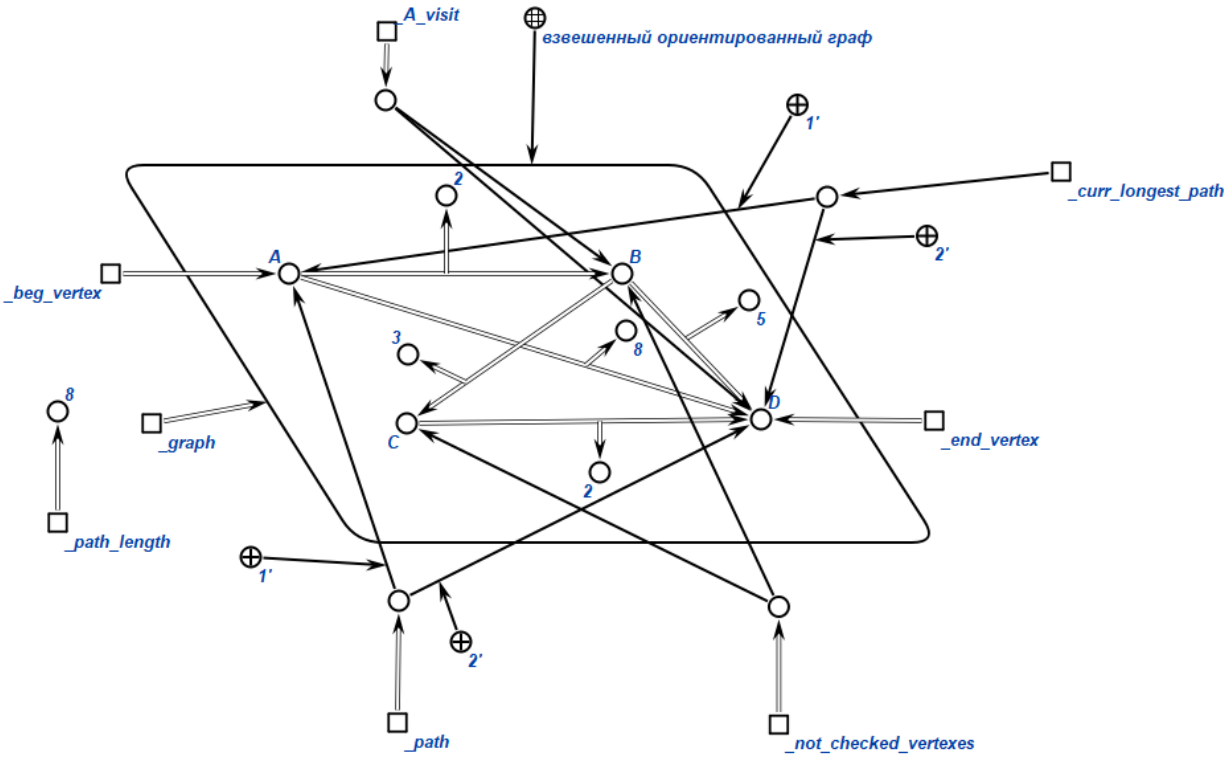
Мы пришли в финальную вершину, однако необходимо проверить все возможные пути из A до D, чтобы выбрать из них самый длинный. Длина текущего полученного пути равна длине текущего самого длинного пути. По этой причине не обновляем множество текущего самого длинного пути.

Удаляем вершину D из множества пути и добавляем во множество неиспользованных вершин. Уменьшаем длину текущего пути на 5. _B_visit все еще хранит C и D. Неисследованных путей из B нет.

Удаляем вершину B из множества пути и добавляем во множество неиспользованных вершин. Уменьшаем длину текущего пути на 2. _B_visit удаляется. _A_visit все еще хранит B.

Вершина A имеет еще одну неисследованную связь с вершиной D, так как_A_visit не хранит вершину D.

7. Добавление во множество используемых вершин вершины D



Добавляем вершину D во множество пути с порядковым номером 2. В _A_visit добавляем вершину D.

Удаляем из множества неиспользованных вершин вершину D.

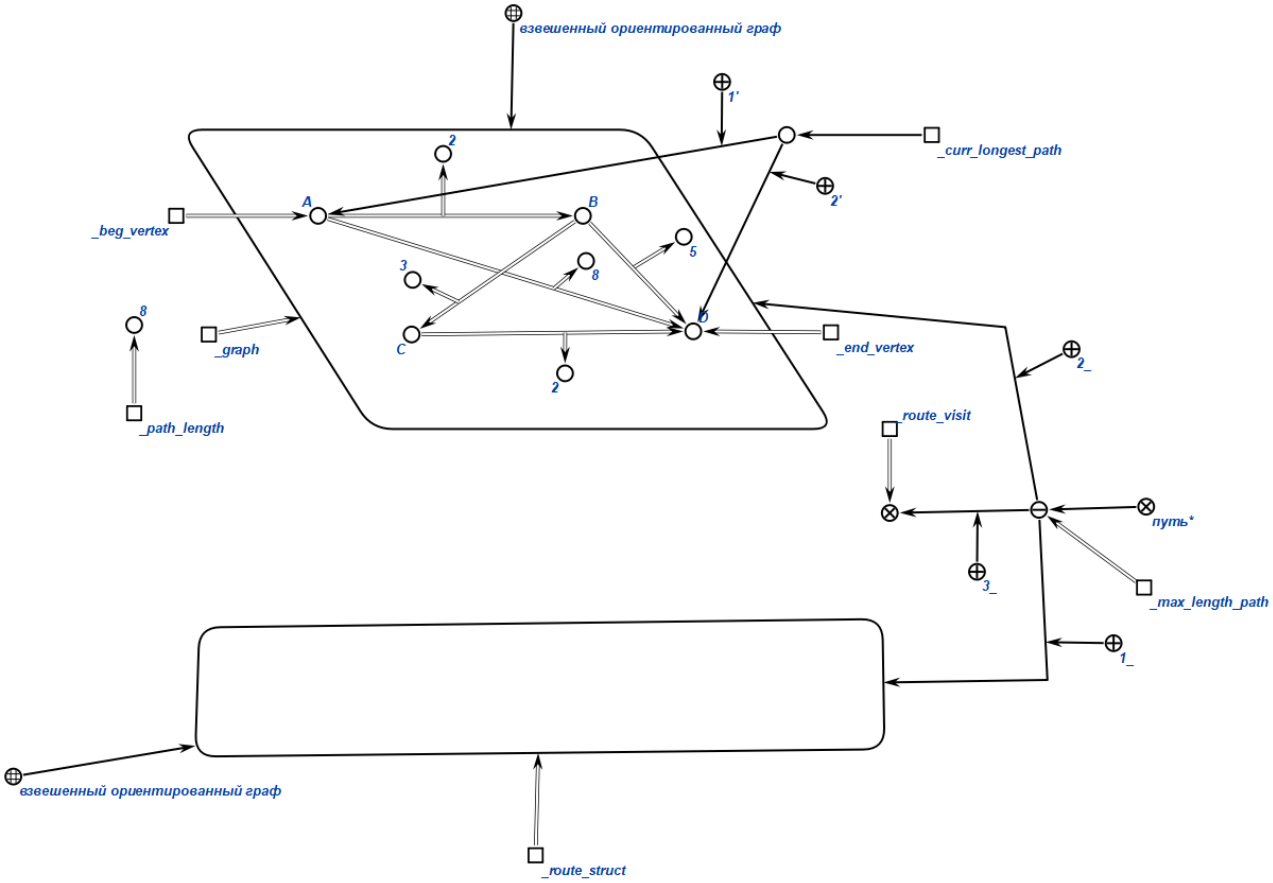
Изменяем длину текущего пути на значение веса ребра AD. Длина текущего пути - 8.

Мы пришли в финальную вершину и проверили все возможные пути из A до D, так как_A_visit хранит вершины D и B. Длина текущего полученного пути больше длины текущего самого длинного пути. По этой обновляем множество текущего самого длинного пути и записываем туда элементы из множества текущего пути с сохранением порядка следования.

Множество непроверенных вершин можем удалить в силу последующей ненадобности. Множество _A_visit можем также удалить.

Множество текущего пути можем удалить в силу последующей ненадобности. Интересующий нас путь хранится в переменной _curr_longest_path.

8. Создание связки отношения путь*



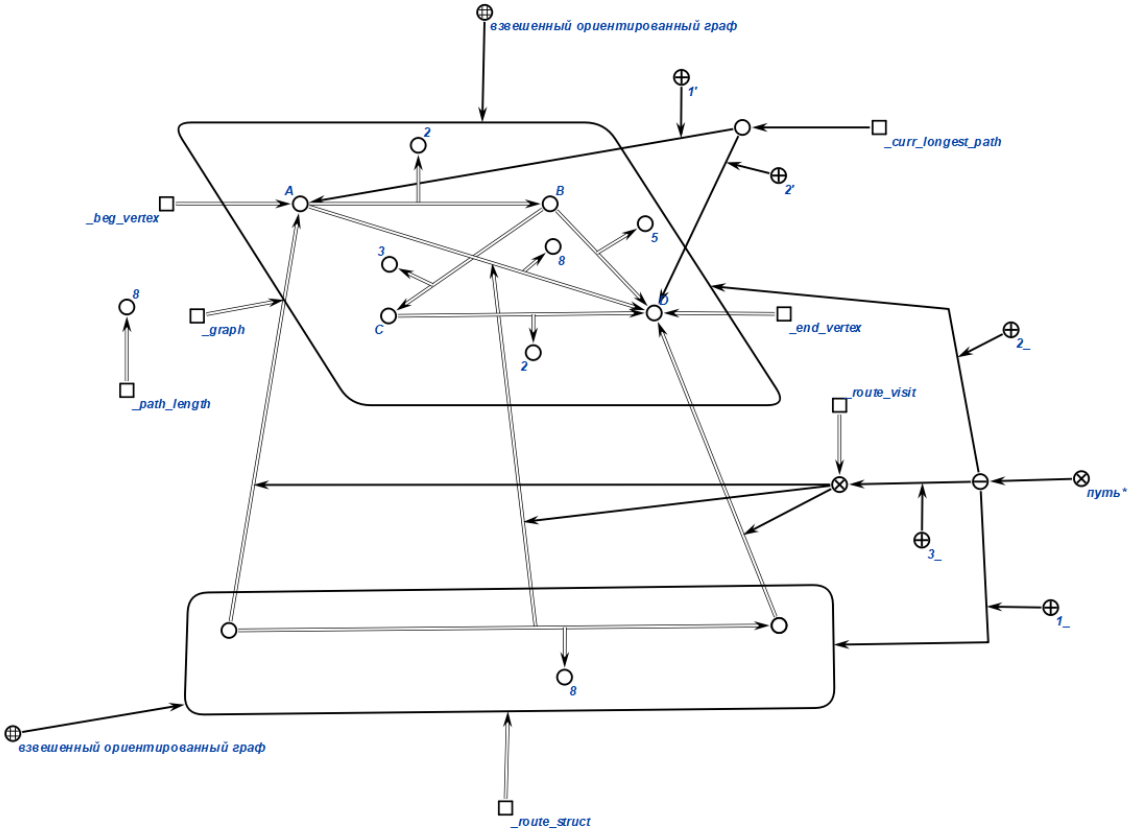
Можем получить ответ, так как мы исследовали все возможные пути в вершину, которая хранится в переменной _end_vertex и нашли самый длинный из них.

Начинаем генерацию самого длинного пути.

Создадим связку отношения путь* и установим ее в качестве значения переменной _max_length_path.

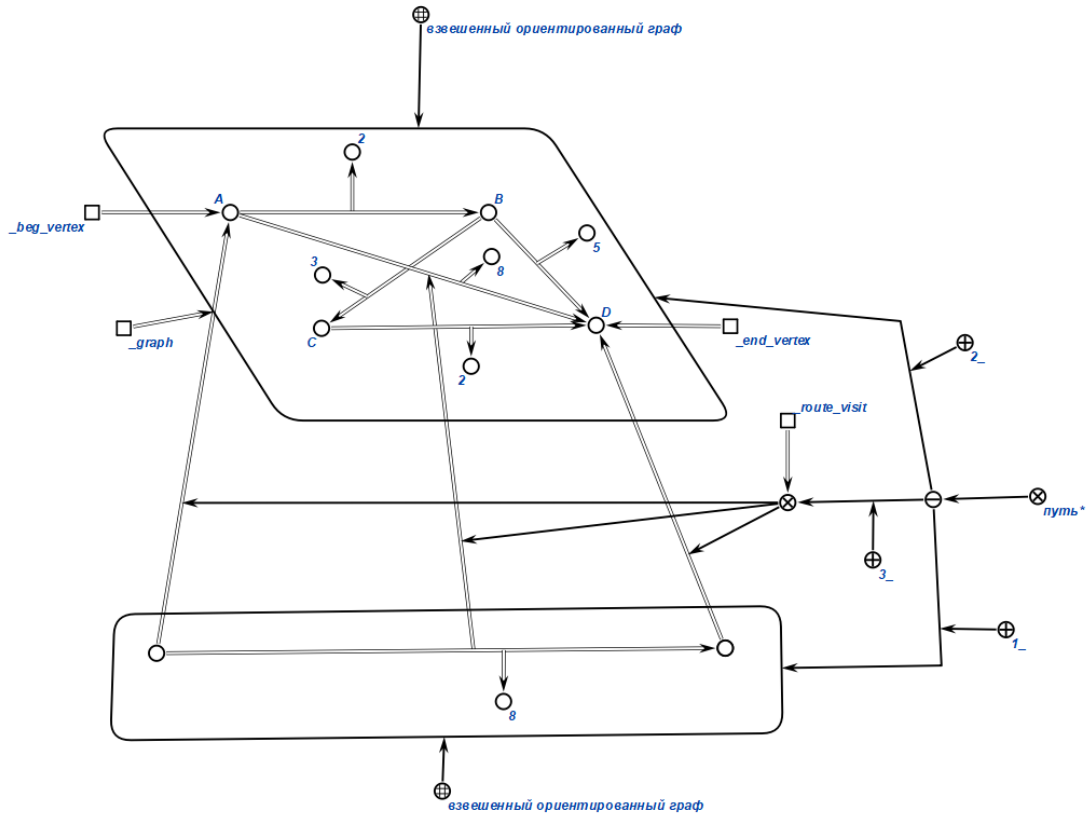
Переменная _route_struct получит в качестве значения ориентированный граф структуры пути, а переменная _route_visit – отношения посещения.

9. Добавление в структуру пути посещения начальной и конечной вершины генерируемого пути и связывающего ребра



Добавим в структуру генерируемого пути посещение начальной и конечной вершины. Создание посещения связывающего их ребра. Также указываем вес связующего ребра.

10. Удаление ориентированного множества _curr_longest_path. Результат работы алгоритма



4 Список литературы

OSTIS GT [В Интернете] // База знаний по теории графов OSTIS GT. - 2011 г.. - http://ostisgraphstheo.sourceforge.net/index.php/Заглавная_страница.

Харарри Ф. Теория графов [Книга]. - Москва : Едиториал УРСС, 2003.