Гамильтоновы циклы

Перебор вершин с возвратом

Определение

Граф называется гамильтоновым, если он содержит цикл, включающий все вершины графа.

Этот цикл тоже называется гамильтоновым.

Не все связные графы гамильтоновы.

Не найдено ни одного необходимого и достаточного условия существования гамильтонового цикла в произвольном графе…

Постановка задачи

Дан связный неориентированный граф.
Найти все гамильтоновы циклы
(если они есть).

“Простой” способ поиска:

Сгенерируем все перестановки вершин графа.

Дальше можно просто проверить каждую, не является ли она гамильтоновым циклом.

Так ли это просто?

Рассмотрим пример:

Пусть у графа, скажем, 20 вершин. Сколько существует перестановок вершин?

Число перестановок N предметов равно N!

$20! = 2432902008176640000 \approx 2.4 \cdot 10^{18}$

А если вершин 100?

Число перестановок будет равно:

933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000

$$\approx 9.3 \cdot 10^{157}$$

Это не все…

Чтобы проверить каждую из этих перестановок на “гамильтоновость” нужно затратить еще N операций.

Таким образом, “лобовое” решение требует

N•N! операций.

Это число растет с ростом N очень быстро…

Конечно, “лобовое” решение крайне нерационально. Ведь при построении всех перестановок вершин не используется информация о том, какие из вершин связаны друг с другом.

Есть более рациональный алгоритм…

Структуры данных:

Будем использовать два массива целых:

Arr[N] – в этом массиве будет находиться последовательность вершин;

Nnew[N] – если *i*-й элемент этого массива есть 0, значит на текущем шаге *i*-я вершина графа еще не посещалась.

Для задания структуры графа будем использовать матрицу смежности Matr[N][N]

Алгоритм

Будем предполагать, что поиск циклов мы ведем с вершины 1.

На очередном шаге в массиве Arr находится последовательность связанных друг с другом вершин, которые, возможно, являются началом гамильтонова цикла.

Алгоритм

Центральной процедурой алгоритма является рекурсивная функция Step.

Эта функция принимает один целый параметр – номер шага.

Алгоритм

1. Функция берет последнюю добавленную в массив Arr вершину, делает ее текущей, и ищет (в цикле по матрице смежности) вершины, связанные с текущей.

2. Если номер шага = N+1, а текущая вершина связана с первой вершиной, то в Arr находится гамильтонов цикл. Его можно вывести, а затем выйти из Step.

Алгоритм

3. Если найдена вершина, связанная с текущей и еще **непосещенная**, то:
   - Эта **новая** вершина добавляется в “хвост” **Arr**;
   - Новая вершина отмечается как **посещенная**;
   - Вызывается процедура **Step** со значением параметра, увеличенным на 1;
   - **После возврата новая вершина вновь отмечается, как непосещенная**.

4. Если все вершины, связанные с текущей уже посещались, то осуществляется выход из **Step**.

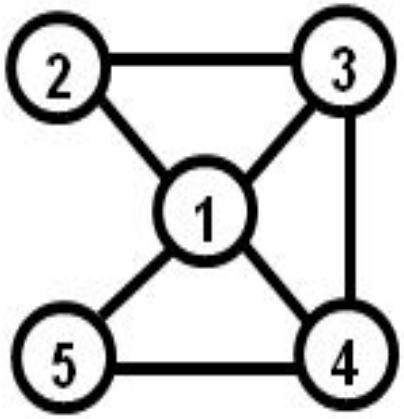
На каждом шаге к массиву Arr добавляется еще не посещенная вершина. Поскольку число вершин конечно, то процесс должен рано или поздно закончиться.

Смущает то, что после возврата из функции Step, последняя вершина помечается как непосещённая.

Не приведет ли это к зацикливанию?..

```
for (j=1; j <= gN; j++)
{
   if (isBound(j,v))
    {
      …
    if (Nnew[j]==0) // сюда управление
    {               // попадет при каждом j
      Arr[k]=j;       // не более 1 раза!
      Nnew[j]=1;
          Step(k+1);
      Nnew[j]=0;
    }
```

Рассмотрим граф:



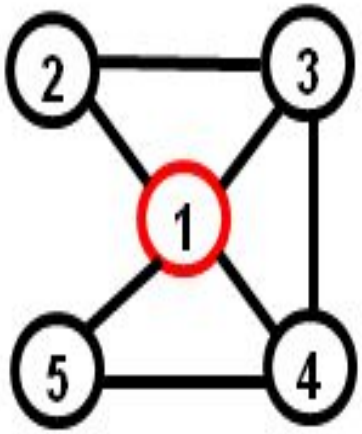
Этот граф имеет два гамильтоновых цикла:

(1,2,3,4,5)

и

(1,5,4,2,1)

Посмотрим, как будет работать описанный алгоритм с возвратами…

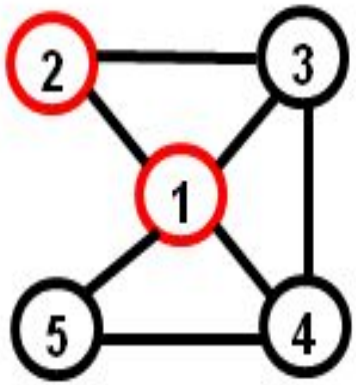


Arr=1

1,0,0,0,0

Вызов Step(2)

На желтом поле показывается массив Arr, на зеленом – признаки прохождения вершин



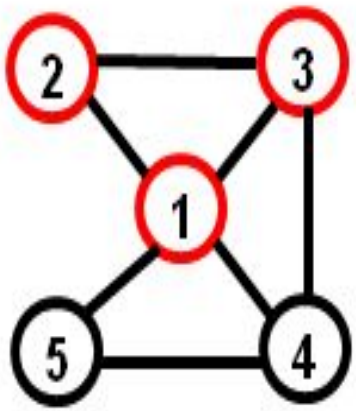
Arr=1,2

1,1,0,0,0

Вызов Step(2) [2]

Вызов Step(3)

В прямых скобках показан параметр цикла в соответствующем вызове при поиске вершины



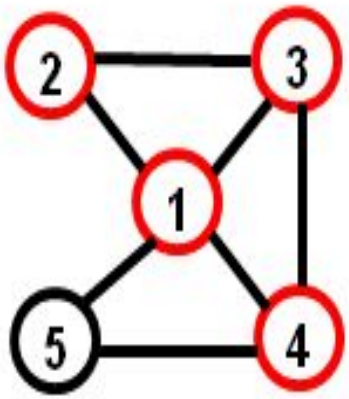
Arr=1,2,3

1,1,1,0,0

Вызов Step(2) [2]

Вызов Step(3) [3]

Вызов Step(4)



Arr=1,2,3,4

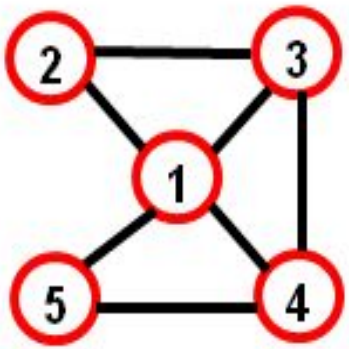
1,1,1,1,0

Вызов Step(2) [2]

Вызов Step(3) [3]

Вызов Step(4) [4]

Вызов Step(5)



Arr=1,2,3,4,5 1,1,1,1,1

Вызов Step(2) [2]

Вызов Step(3) [3]

Вызов Step(4) [4]

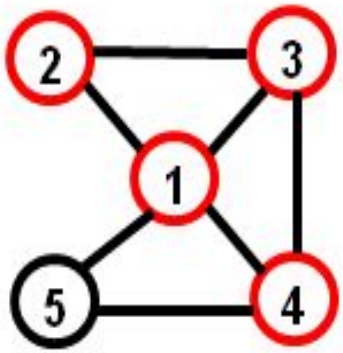
Вызов Step(5) [5]

Вызов Step(6)

Вывод цикла

Возврат

Параметр вызова=N+1 и из последней вершины достижима первая – выполнено условие цикла.



Arr=1,2,3,4

1,1,1,1,0

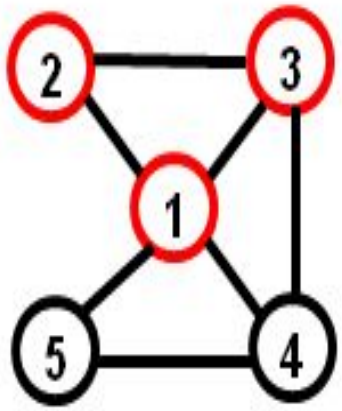
Вызов Step(2) [2]

Вызов Step(3) [3]

Вызов Step(4) [4]

Вызов Step(5) [5]

Возврат



Arr=1,2,3

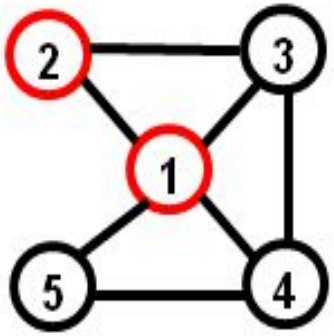
1,1,1,0,0

Вызов Step(2) [2]

Вызов Step(3) [3]

Вызов Step(4) [4]

Возврат



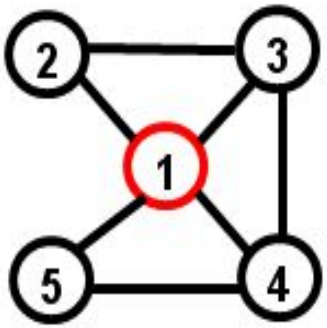
Arr=1,2

1,1,0,0,0

Вызов Step(2) [2]

Вызов Step(3) [3]

Возврат

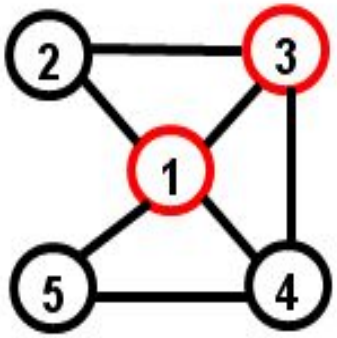


Arr=1

1,0,0,0,0

Вызов Step(2) [3]

Какую вершину будем добавлять?

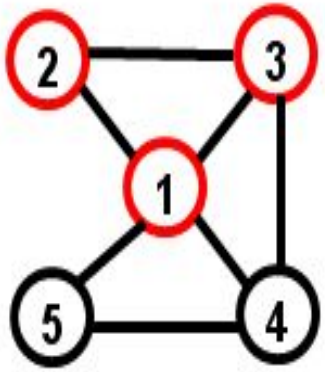


Arr=1,3

1,0,1,0,0

Вызов Step(2) [3]

Вызов Step(3) [2]



Arr=1,3,2

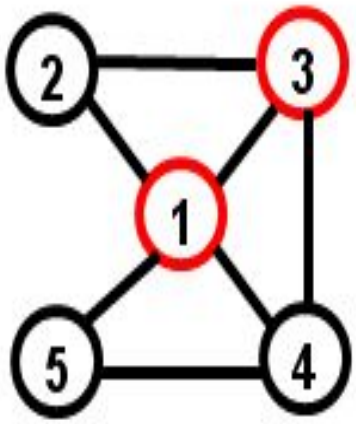
1,1,1,0,0

Вызов Step(2) [3]

Вызов Step(3) [2]

Вызов Step(4)

Возврат



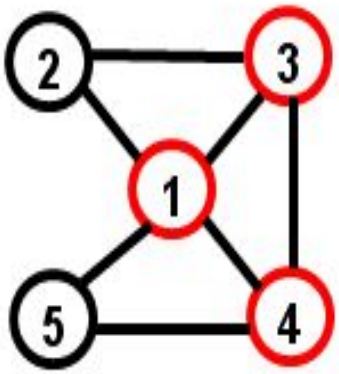
Arr=1,3

1,0,1,0,0

Вызов Step(2) [3]

Вызов Step(3) [4]

Какую вершину будем добавлять?



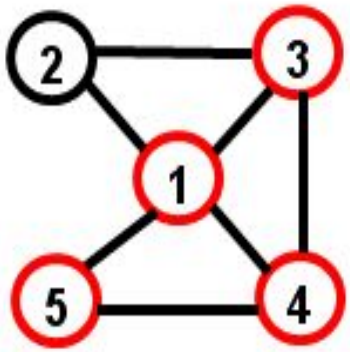
Arr=1,3,4

1,0,1,1,0

Вызов Step(2) [3]

Вызов Step(3) [4]

Вызов Step(4) [5]



Arr=1,3,4,5

1,0,1,1,1

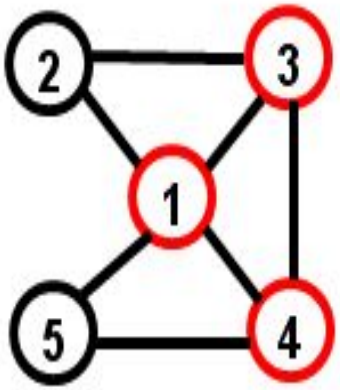
Вызов Step(2) [3]

Вызов Step(3) [4]

Вызов Step(4) [5]

Вызов Step(5)

Возврат



Arr=1,3,4

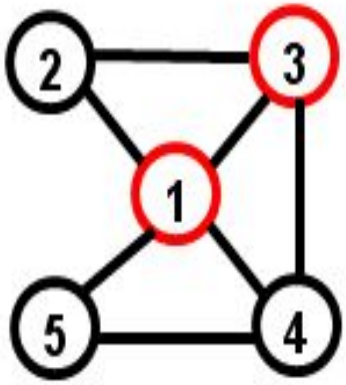
1,0,1,1,0

Вызов Step(2) [3]

Вызов Step(3) [4]

Вызов Step(4) [5]

Возврат



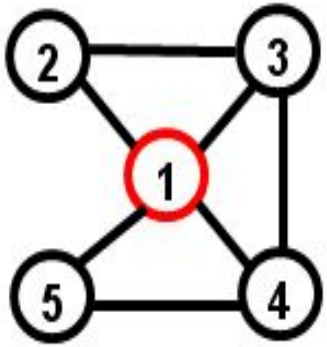
Arr=1,3

1,0,1,0,0

Вызов Step(2) [3]

Вызов Step(3) [4]

Возврат

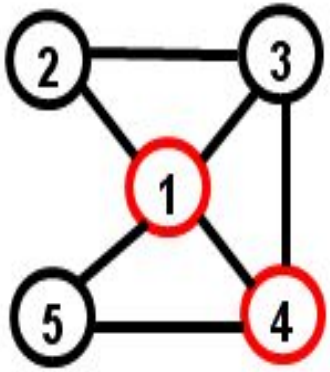


Arr=1

1,0,0,0,0

Вызов Step(2) [4]

Какую вершину будем добавлять?

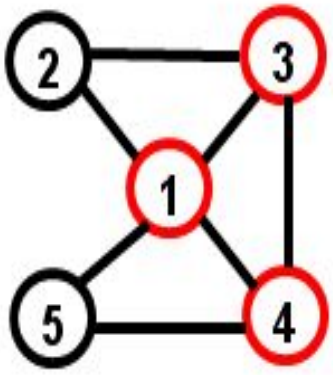


Arr=1,4

1,0,0,1,0

Вызов Step(2) [4]

Вызов Step(3) [3]



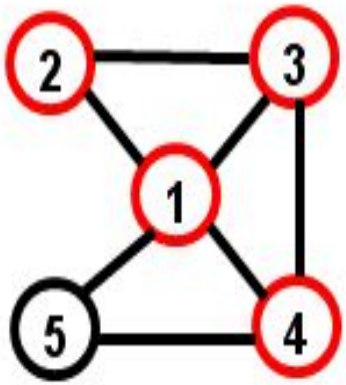
Arr=1,4,3

1,0,1,1,0

Вызов Step(2) [4]

Вызов Step(3) [3]

Вызов Step(4) [2]



Arr=1,4,3,2

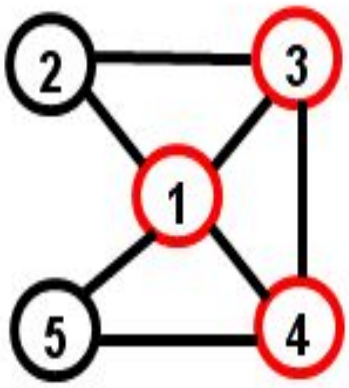
1,1,1,1,0

Вызов Step(2) [4]

Вызов Step(3) [3]

Вызов Step(4) [2]

Возврат



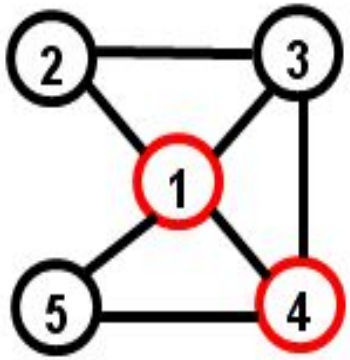
Arr=1,4,3

1,0,1,1,0

Вызов Step(2) [4]

Вызов Step(3) [3]

Возврат



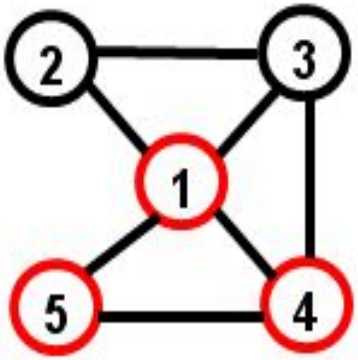
Arr=1,4

1,0,0,1,0

Вызов Step(2) [4]

Вызов Step(3) [4]

Вызов Step(4)



Arr=1,4,5

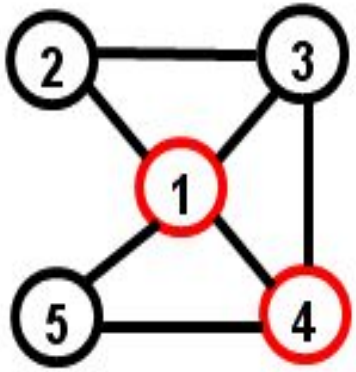
1,0,0,1,1

Вызов Step(2) [4]

Вызов Step(3) [4]

Вызов Step(4)

Возврат



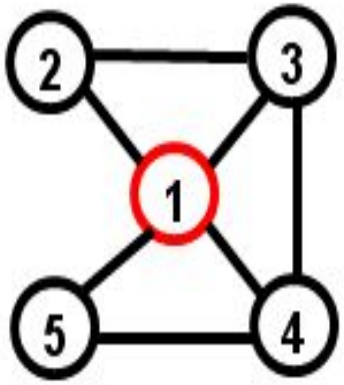
Arr=1,4

1,0,0,1,0

Вызов Step(2) [5]

Вызов Step(3) [4]

Возврат

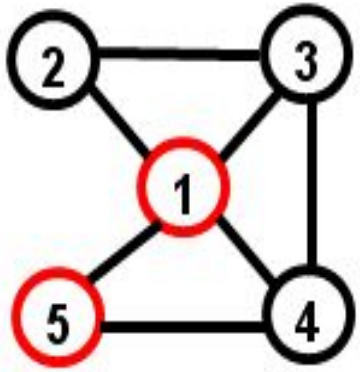


Arr=1

1,0,0,0,0

Вызов Step(2) [5]

Следующей будет добавлена 5-я вершина

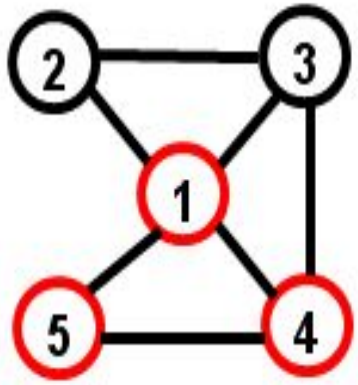


Arr=1,5

1,0,0,0,1

Вызов Step(2) [5]

Вызов Step(3) [4]



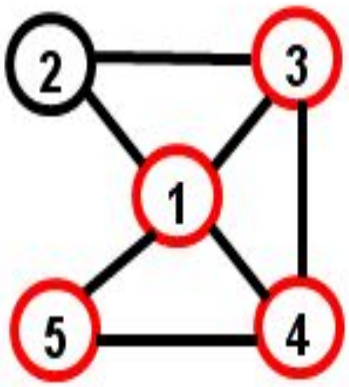
Arr=1,5,4

1,0,0,1,1

Вызов Step(2) [5]

Вызов Step(3) [4]

Вызов Step(4) [3]



Arr=1,5,4,3

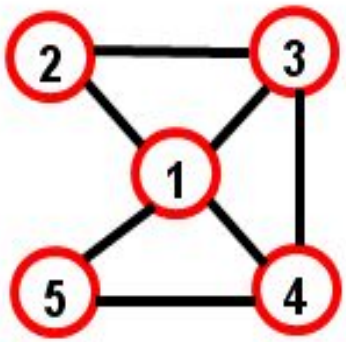
1,0,1,1,1

Вызов Step(2) [5]

Вызов Step(3) [4]

Вызов Step(4) [3]

Вызов Step(5) [2]



Arr=1,5,4,3,2

1,1,1,1,1

Вызов Step(2) [5]

Вызов Step(3) [4]

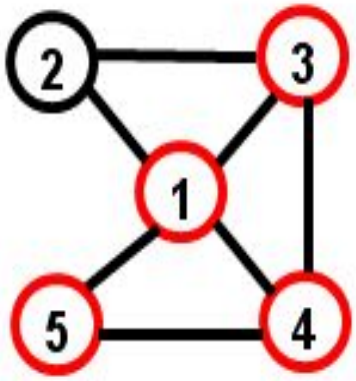
Вызов Step(4) [3]

Вызов Step(5) [2]

Вызов Step(6)

Вывод цикла

Возврат



Arr=1,5,4,3

1,0,1,1,1

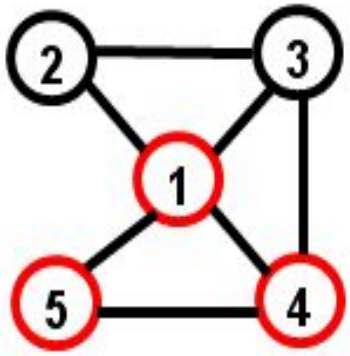
Вызов Step(2) [5]

Вызов Step(3) [4]

Вызов Step(4) [3]

Вызов Step(5) [2]

Возврат



Arr=1,5,4

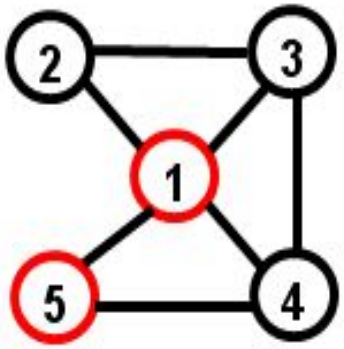
1,0,0,1,1

Вызов Step(2) [5]

Вызов Step(3) [4]

Вызов Step(4) [3]

Возврат



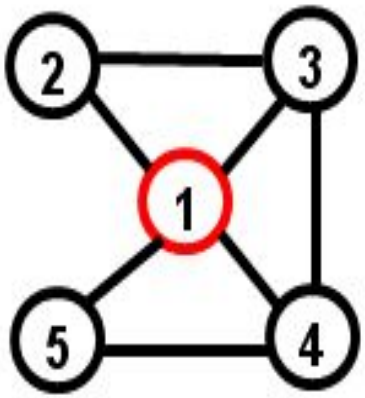
Arr=1,5

1,0,0,0,1

Вызов Step(2) [5]

Вызов Step(3) [4]

Возврат



Arr=1

1,0,0,0,0

Вызов Step(2) [5]

Возврат

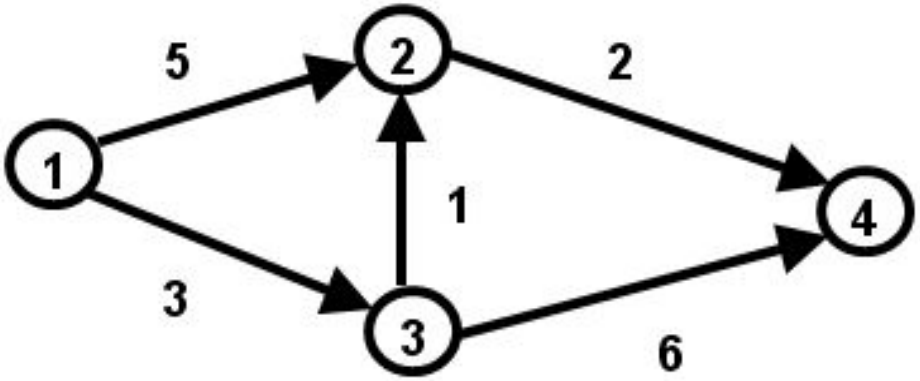
Кратчайшие пути в графах

Постановка задачи

- Дан ориентированный граф;
- Каждой дуге приписана “длина” – вес дуги;
- Веса дуг хранятся в матрице смежности, причем, если i-я и j-я вершины не связаны дугой, то A[i,j]=∞
- “Длина” или оценка пути = сумме весов дуг, составляющих путь.

Требуется находить пути с минимальной оценкой (т.е. кратчайшие).

Пример:

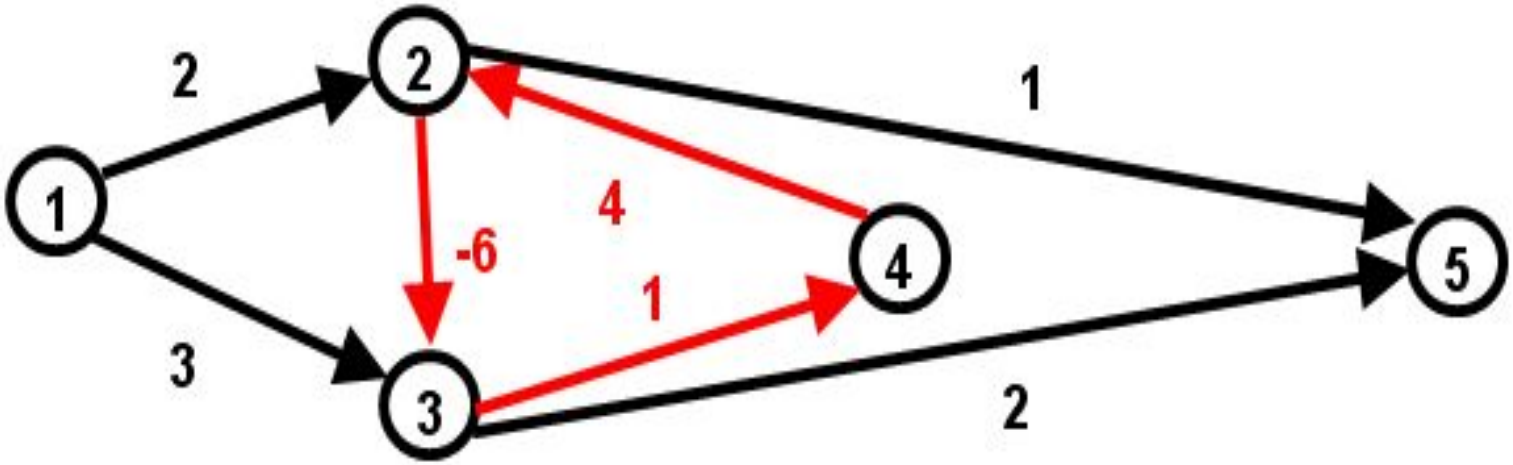


Каков будет кратчайший путь из 1 в 4?

Путь (1-3-2-4). Его длина = 6. Другие пути длиннее.

Контуры отрицательного веса

Если граф содержит контуры отрицательного веса, то поиск минимальной длины пути теряет смысл



Если мы ищем кратчайший путь из 1 в 5, то проходя контур 3-4-2-3 несколько раз, можно сделать путь 1-5 меньше любой наперед заданной величины.

Соглашение:

Мы далее будем рассматривать только графы без контуров отрицательного веса.

(но дуги с отрицательной длиной могут существовать)

Обозначим длину минимального пути между i-й и j-й вершинами через D(i,j).

К настоящему времени неизвестен алгоритм, позволяющий найти минимальный путь только для пары вершин.

Все алгоритмы требуют определения оценки минимального пути для всех вершин графа.

Для некоторой вершины p обозначим массив кратчайших расстояний до всех остальных вершин через D_p.

Предположим,что есть алгоритм определения этого массива для любой вершины графа.

Алгоритм определения пути

Ищем кратчайший путь из i-й вершины в j-ю.

Можно найти такую вершину k, что:

$$D_i(j)=D_i(k)+A[k,j]$$

Таким свойством обладает предпоследняя вершина кратчайшего пути из i в j.

Запомним вершину k в стеке и ищем вершину l, для которой:

$$D_i(k)=D_i(l)+A[l,j]$$

На каждом шаге мы приближаемся к исходной вершине, и, в конце концов, дойдем до нее.

В стеке будет искомый путь (последовательность вершин).

Таким образом, если для каждой вершины известен массив кратчайших расстояний до всех вершин графа, можно построить кратчайший путь.

Но как определить массив кратчайших расстояний?

Общая схема такова:

Пусть зафиксирована i-я вершина, для которой мы ищем массив кратчайших расстояний.

Для каждой вершины j вычисляем верхние ограничения $D_i(j)$ на расстояние (i – j).

Далее стараемся улучшить эти ограничения.

Если для вершины k нашли вершину q, такую, что:

$$D_i(k) > D_i(q)+A[k,q],$$

то заменяем $D_i(k)$ на сумму $D_i(q)+A[k,q]$.

Процесс завершается, когда дальнейшее улучшение невозможно.

Описанной схеме не хватает существенного момента – порядка выбора вершин k и q.

В реальности этот порядок важен, т.к. от него существенно зависит эффективность алгоритма.

Алгоритм Форда-Беллмана

Этот алгоритм применим к любому ориентированному графу без контуров отрицательной длины

Исходные данные алгоритма:

1) **Орграф без контуров отрицательной длины;**
2) **Матрица весов дуг;**
3) **Фиксированная вершина i**

Результат:

- **Расстояние (кратчайшее) от всех вершин графа до фиксированной вершины i.**

1) Первоначально присваиваем всем элементам массива $D_i[k]$ значения $A[i,k]$; Элементу $D_i[i]$ присваиваем значение 0.
2) (N-2) раза повторяем следующие действия:
   - Для всех вершин q (кроме i)
   - Для всех вершин w вычисляем:
     $D_i[q]=\min(D_i[q],D_i[w]+A[w,q])$

Чему равна временная сложность этого алгоритма?

N-кратное повторение трех вложенных циклов дает порядок $O(N^3)$

Если веса всех дуг графа неотрицательны, то алгоритм Форда-Беллмана можно улучшить до производительности $O(N^2)$.

Алгоритм Дейкстры

Исходные данные алгоритма:

1) **Орграф с дугами неотрицательной длины;**
2) **Матрица весов дуг;**
3) **Фиксированная вершина i**

Результат:

- **Расстояние (кратчайшее) от всех вершин графа до фиксированной вершины i.**

1) Первоначально присваиваем всем элементам массива $D_i[k]$ значения $A[i,k]$; Элементу $D_i[i]$ присваиваем значение 0.
2) Обозначим через T совокупность вершин графа без вершины i.
3) Выполнять, пока T не пусто, следующие действия:

   - искать в T вершину u, для которой величина $D_i[u]$ минимальна;

   - исключить u из T;

   - для всех оставшихся вершин w из T вычислить:

$$D_i[w]=\min(D_i[w],D_i[w]+A[u,w]$$

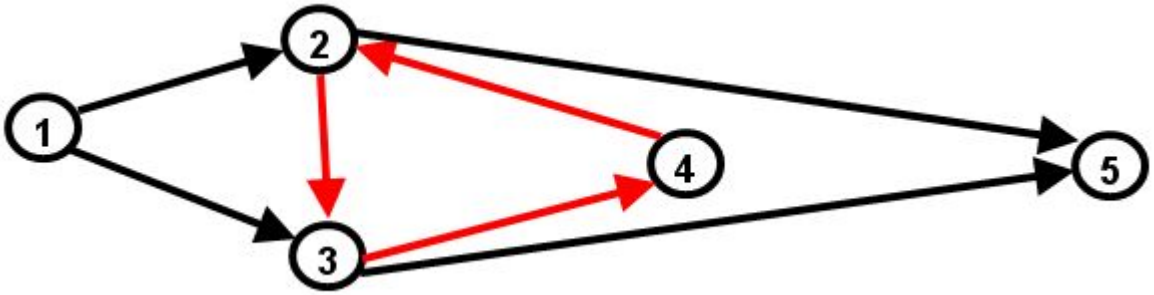
Алгоритм Дейкстры имеет сложность $O(N^2)$

Имеется еще один частный вид графов, для которых вычисление расстояний имеет сложность $O(N^2)$.

Это бесконтурные графы

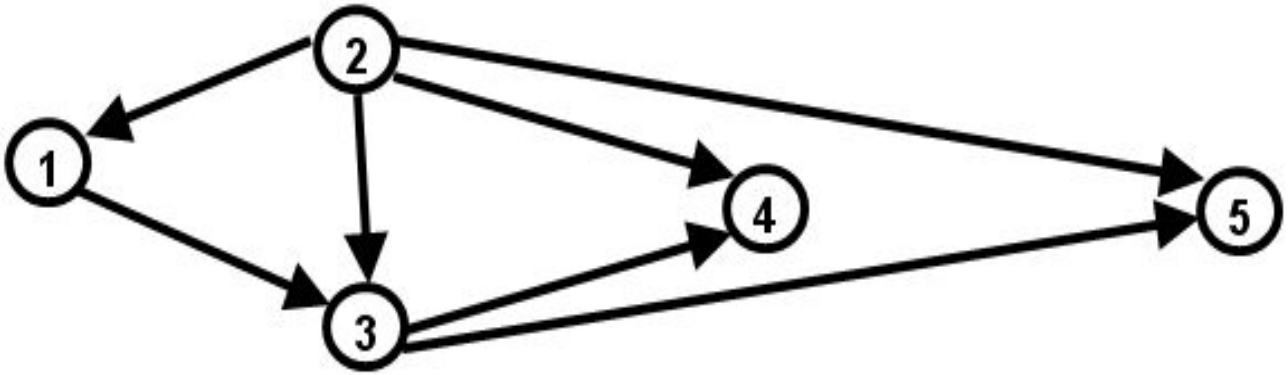
(ориентированные графы без циклов)

Является ли этот граф бесконтурным?



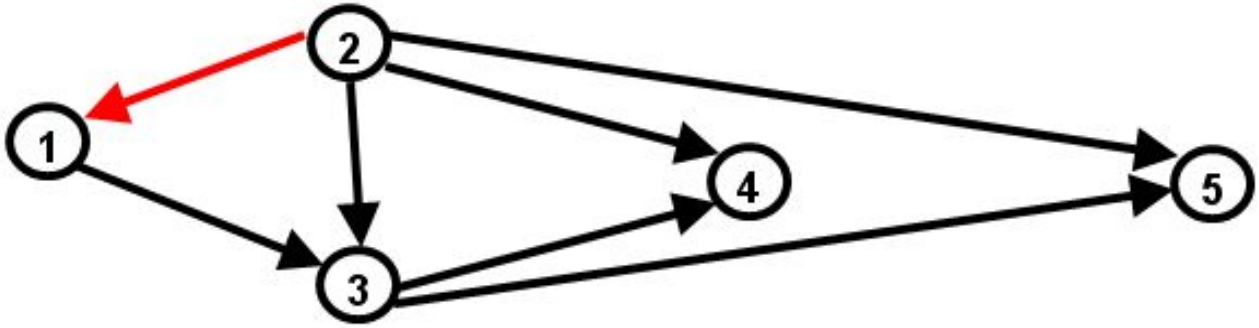
Нет. Контур выделен.

А этот?

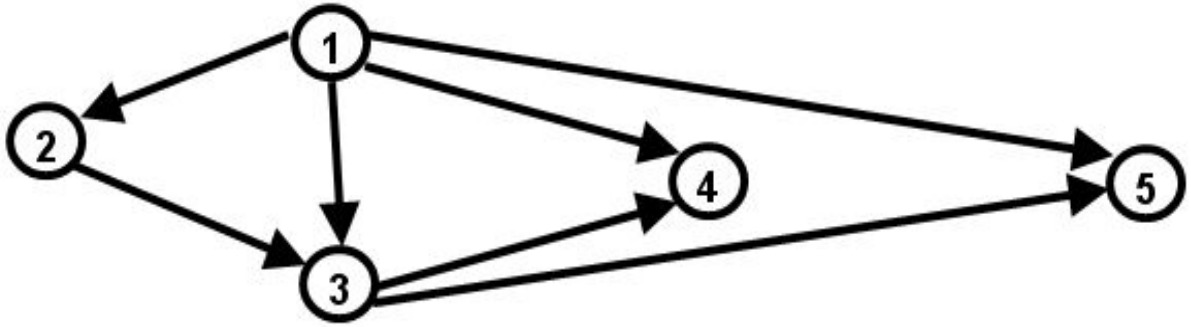


Бесконтурные графы обладают замечательным свойством: их вершины можно перенумеровать так, что для любой дуги (p,q) всегда будет q > p.

У нашего графа есть “неправильная” дуга (2-1):



Если сделать вершину 1 вершиной 2, а вершину 2 – вершиной 1, то граф станет “правильным”:



Существует эффективный алгоритм, позволяющий перенумеровать вершины бесконтурного графа и превратить его в “правильный” граф.

Сложность этого алгоритма = O(N+M).

Для “правильного” бесконтурного графа расстояния считаются очень просто.

http://catstail.narod.ru/lec/lec-12.zip