I. Эйлеровы пути и циклы

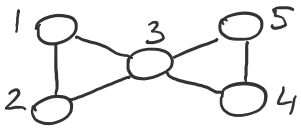
Эйлеров путь – путь в графе, проходящий по каждому ребру графа ровно один раз

Полуэйлеров граф – граф, в котором есть эйлеров путь

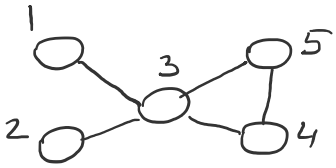
Эйлеров цикл – замкнутый эйлеров путь

Эйлеров граф – граф, в котором есть эйлеров цикл

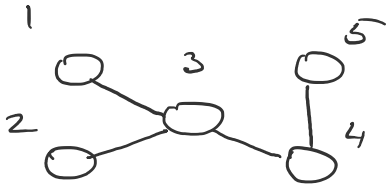
Примеры:



ЭЙЛЕРОВ
(1→3→4→5→3→2→1)



ПОЛУЭЙЛЕРОВ
(1→3→4→5→3→2)



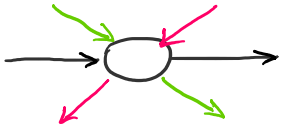
А это кто?

II. Критерии эйлеровости и полуэйлеровости для неориентированных графов

Теорема 2.1 (критерий эйлеровости)

Связный граф G эйлеров ⇔ Степени всех вершин связного графа G четны

▫ ⇒ Следует из баланса входящих и исходящих ребер при обходе по эйлеровому циклу (если по какому-то ребру попали в вершину, то должно найтись другое, по которому эту вершину покидаем)



⇐ Докажем конструктивно – предъявим алгоритм и докажем его корректность (см. Алгоритм) ▪

Теорема 2.2 (критерий полуэйлеровости)

Связный граф G полуэйлеров ⇔ В связном графе G 0 или 2 вершины имеют нечетную степень

▫ ⇒ Аналогично предыдущей теореме – сколько раз входили в вершину, столько же раз и выходили. Исключениями могут быть только стартовая и конечная вершины пути. Эти две вершины и будут иметь нечетную степень

⇐ Добавим ребро между вершинами с нечетными степенями. Тогда по предыдущей теореме найдем эйлеров цикл в графе. Удалим из него добавленное ребро – получим требуемый путь. ▪

III. Критерии эйлеровости и полуэйлеровости для ориентированных графов

Теорема 3.1 (критерий эйлеровости)

Слабосвязный граф G эйлеров ⇔ У любой вершины слабосвязного графа G входящая степень вершины равна исходящей степени

▫ Доказывается аналогично теореме 2.1 ▪

Теорема 3.2 (критерий полуэйлеровости)

Слабовязный граф G полуэйлеров ⇔ У любой вершины слабосвязного графа G входящая степень вершины равна исходящей степени, кроме, возможно, двух, у одной из которых входящая степень на 1 больше исходящей, а у второй – на 1 меньше.

▫ Доказывается аналогично теореме 2.2 ▪

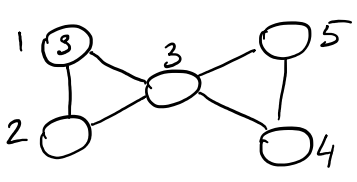
IV. Алгоритм

Задача решается с помощью алгоритма похожего на поиск в глубину. Ясно, что обычным DFS обойтись нельзя, так как DFS посещает каждую вершину ровно 1 раз, в то время как эйлеров цикл может посещать одну и ту же вершину несколько раз за обход.

Изменим DFS так, что при обходе графа будем помечать не вершины графа, а ребра (нам требуется по каждому ребру пройтись ровно один раз). Будем рекурсивно продвигаться вглубь графа по еще не посещенным ребрам до тех пор, пока не упремся в вершину, у которой все ребра посещены. Всякий раз, когда такое происходит, будем записывать последнее пройденное ребро в начало списка. Полученный список и будет искомым циклом.

```
1: DfsEulerCycle(G, v, answer): // Find EulerCycle in graph G, visit Node v, answer contains path of edges
2:   for (v, u) in G.E:          // for all edges with ending at node v
3:     G.remove((v, u))          // mark (v, u) as visited
4:     DfsEulerCycle(G, u, answer)
5:     answer.push_front((v, u))
```

Пример:



1) Начинаем в (1)
2) Идем по ребру (1,2)
3) Идем по ребру (2,3)
4) Идем по ребру (3,1)
5) Из (1) нет ребер
6) Возвращаемся в (3), выписываем (3,1)
7) Идем по ребру (3,4)
8) Идем по ребру (4,5)
9) Идем по ребру (5,3)
10) Из (3) нет ребер
11) Откатываемся назад, выписываем последовательно все ребра

ОТВЕТ: [(1,2), (2,3), (3,4), (4,5), (5,3), (3,1)]

Теорема 4.1 (о корректности)

В связном графе с четными степенями вершин алгоритм находит эйлеров цикл

▫ Для доказательства этого факта необходимо доказать 2 утверждения:

1) Алгоритм проходит по каждому ребру ровно один раз (все ребра будут записаны в ответе)

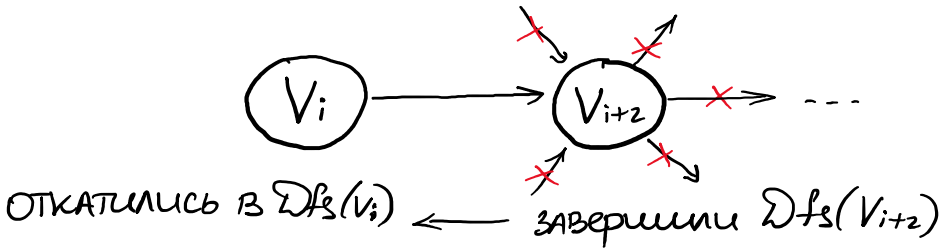
Проведем те же рассуждения, что мы применяли ранее для доказательства леммы о белом пути (предполагаем, что посещенные вершины тоже как-то помечаем). Тогда получим, что процедура DfsEulerCycle обойдет все вершины графа (в связном графе все вершины достижимы из стартовой). При вызове DfsEulerCycle просматриваются все инцидентные вершине ребра. Таким образом, будут посещены все ребра графа причем ровно один раз (так как при прохождении ребра тут же его удаляем).

2) Построенная последовательность ребер образует корректный путь

Рассмотрим в построенной последовательности два соседних ребра: …, (v_i, v_{i+1}), (v_{i+2}, v_{i+3}), …

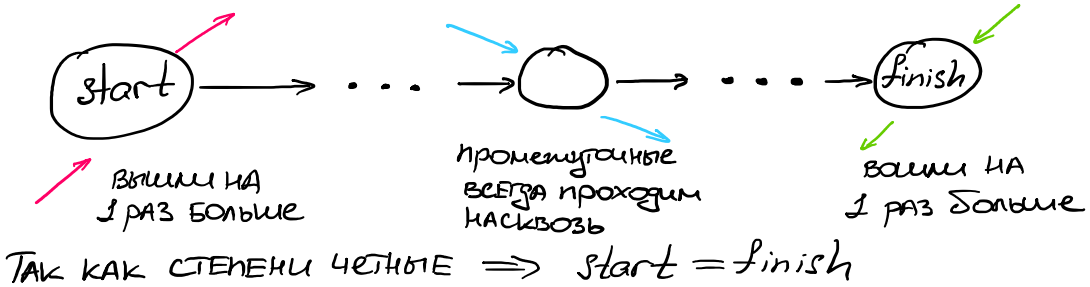
Покажем, что $v_{i+1} == v_{i+2}$. Рассмотрим момент, когда ребро (v_{i+2}, v_{i+3}) оказалось в списке (строка 5). В этот момент идет выполнение функции DfsEulerCycle(G, v_{i+2}, answer). Возможны две ситуации:

1. Эта итерация цикла была последней (из вершины v_{i+2} больше нет доступных ребер). Тогда вызов завершается и выполнение откатывается в DfsEulerCycle от вершины, которая вызывала Dfs от v_{i+2} (кто-то сделал этот вызов в строке 4). Но, что произошло сразу после этого? В следующий момент (после завершения DfsEulerCycle(G, v_{i+2}, answer), строка 5) мы кладем новое ребро в список, причем у этого ребра второй конец равен v_{i+2}! То есть $v_{i+1} == v_{i+2}$



2. Не все ребра из вершины v_{i+2} были просмотрены, то есть просматриваем новое ребро (строка 2) с началом в вершине v_{i+2} . Докажем, что путь, построенный этим вызовом Dfs (последовательность вызовов) закончится в вершине v_{i+2}. Для этого стоит понять, а в какой момент происходит ситуация, в которой вызов Dfs завершается (идти больше некуда).

Когда мы запускаем процедуру поиска цикла степень стартовой вершины уменьшается на 1, а у всех остальных при каждом проходе на 2 (при входе уменьшаем на 1 и при выходе тоже на 1). Таким образом, в какую бы вершину ни пришли – всегда из нее можем выбраться (так как степень любой вершины остается четной). Это верно для всех вершин кроме стартовой – в нее мы не приходили, мы в ней были изначально, соответственно у нее входящая степень на 1 больше, чем исходящая (то есть степень нечетна – в случае неориентированных графов). А это значит, что закончить свой путь можем только в ней! Далее, начинаем откатываться назад по вызовам Dfs и возвращаемся в первую вершину, из которой просмотрены не все ребра. Что мы знаем про свойства графа в этот момент?-У него все степени вершин четные (у каждой вершины уменьшали степень на 2 + у стартовой вершины уменьшили на 1 при выходе и еще на 1 при финальном возврате в нее). То есть попадаем в ту же ситуацию – начинаем обход из новой вершины, у которой степень становится нечетной, а у всех остальных вершин степень уменьшается каждый раз на 2 => вызовы Dfs прекратятся в этой же вершине.



Вернемся к первому абзацу. Мы показали, что завершение внутреннего вызова Dfs означает, что в этот момент степени всех вершин четны. А вызов новой последовательности Dfs впервые остановится, когда мы вернемся в вершину v_{i+2} (причем необязательно в первый раз, мы вполне можем в процессе обхода проходить несколько раз «сквозь» нее, но завершимся обязательно в ней). А это значит, что следующим в ответ добавится ребро с концом v_{i+2}. ▪

Замечание: внимательный читатель может заметить, что доказательство первого утверждения не проходит в случае ориентированных графов, так как из слабой связности в общем случае не следует достижимость всех вершин из стартовой. Однако легко показать, что из того, что полустепени вершин равны, следует достижимость любой вершины из любой другой. Это предлагается проделать в качестве упражнения (рассмотрите множество достижимых вершин и множество недостижимых, рассмотрите ребра между множествами, куда они направлены, придите к противоречию с равенством полустепеней).