Глава 3.
Обходы графов

Существует ряд задач на графах, в которых требуется найти маршрут, который содержит все вершины или ребра графа – ***обход.***

Часто требуется, чтобы длина этого маршрута была минимальной (для взвешенных графов), или ограничивается число проходов по одному и тому же элементу графа.

Одна из задач заключается в том, чтобы обойти все вершины графа и в каждой из них только один раз выполнить какое-либо действие: в простейшем случае пронумеровать или пометить вершину; в более сложных случаях решить какую-либо задачу, связанную с этой вершиной. Эту задачу часто называют *поиском* на графе.

3.1. Поиск в ширину (breadth-first search, BFS)

1. Поиск начинается с некоторой фиксированной вершины s.
2. Последовательно просматриваются и помечаются вершины, находящиеся на расстоянии 1 от s (т.е. смежные с s).
3. k-й шаг. Просматриваются и помечаются все вершины, находящиеся на расстоянии k от s.

Процесс поиска продолжается до тех пор, пока все вершины не будут просмотрены и помечены. □

При представлении графа списком смежности сложность алгоритма составляет $O(|V|+|E|)$.

Пример 3.1. Такая разметка применялась в алгоритме Мура – Ли.

3.2. Поиск в глубину (*depth-first search, DFS*).

В процессе поиска будем различать вершины
- непросмотренные;
- просмотренные, но не помеченные;
- помеченные.

Перед началом просмотра все вершины считаются непросмотренными и непомеченными.
Общая идея метода состоит в следующем.
1. Поиск начинается с некоторой фиксированной вершины , которой назначаем номер 0. Вершина x_0 теперь просмотрена, но не помечена. Затем выбираем любую вершину , смежную с x_0, нумеруем ее единицей 1, и процесс поиска продолжается от нее. Считаем теперь вершину x_1 , которая просмотрена, но не помечена, текущей.

2. Пусть x_i – текущая просмотренная вершина. Если существует непросмотренная вершина x_{i+1}, смежная с x_i, то поиск продолжается с вершины x_{i+1}. Если нет непросмотренной вершины, смежной с x_i , то вершина x_i получает очередную метку и считается помеченной.
Теперь делается *возврат* – (*бэктрекинг, backtracking*) в вершину x_{i-1}, ту, из которой в процессе поиска мы попали в x_i.
Вершина x_{i-1} становится текущей и поиск продолжается из нее.
Поиск продолжается до тех пор, пока снова не вернемся в вершину x_0, и x_0 окажется помеченной. ☒
Метод поиска в глубину очевидным образом переносится на орграфы – переход к следующей вершине происходит по направлению ориентации дуги, а возврат – в противоположном направлении.

Таким образом, при обходе графа мы стремимся проникнуть в глубину графа так далеко, как это возможно, затем отступаем на шаг назад, снова стремимся пройти вперед и т.д. Алгоритм поиска в глубину на графе имеет сложность порядка $O(n + m)$, где n – число вершин, а m – число ребер графа.

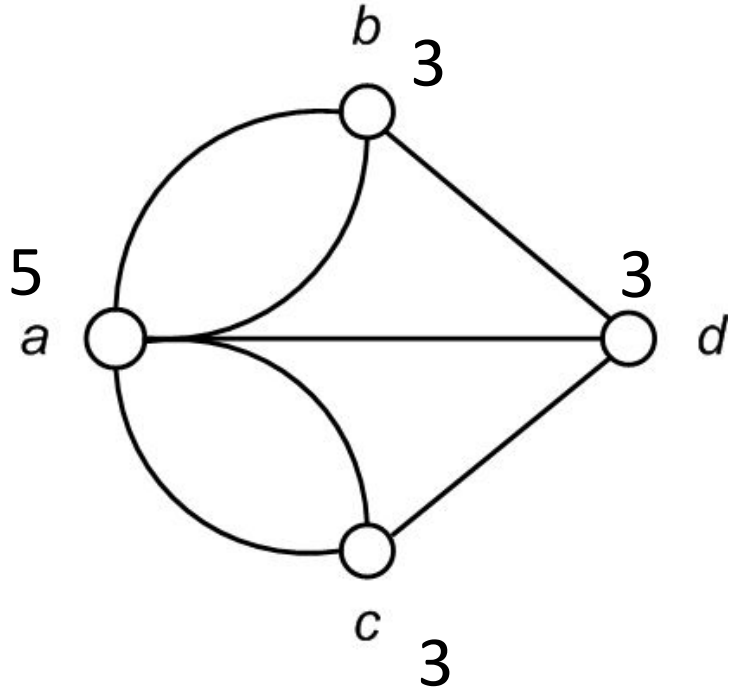
Методы поиска в ширину и в глубину применяются для широкого класса переборных задач и имеют много разновидностей и модификаций.

3.1. Эйлеровы цепи и циклы

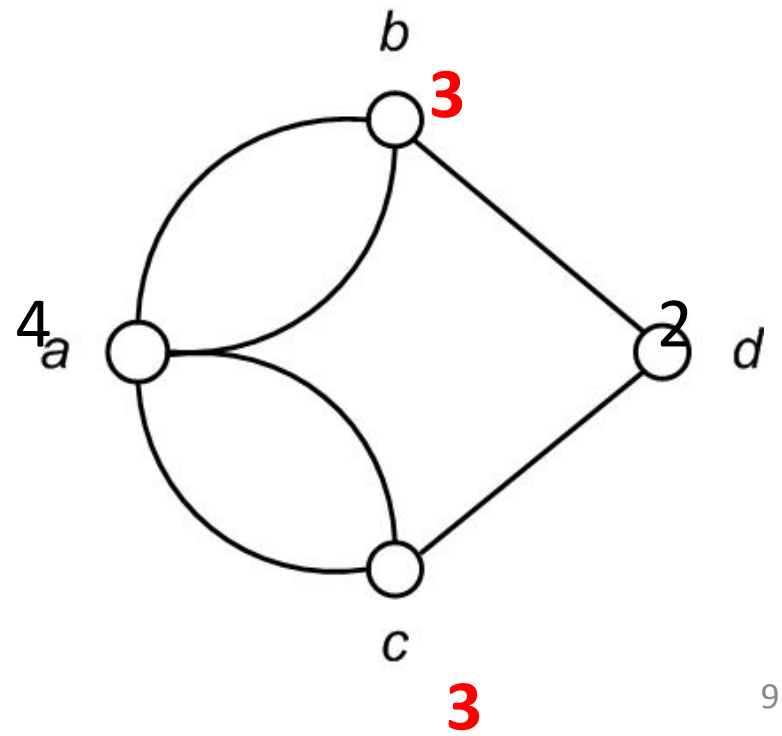
Постановка задачи

Определение. ***Эйлеровой*** **цепью (циклом)** называется цепь (цикл), содержащая (содержащий) все ребра графа. Если в графе существует эйлерова цепь (цикл), то граф называется ***эйлеровым***.

Эйлеров граф можно нарисовать одним росчерком пера.



Кенигсбергские мосты



Эйлеров

Теорема Эйлера (1736 г.)

Теорема. Эйлерова цепь (цикл) существует тогда и только тогда, когда число вершин с нечетной валентностью равно 2 (0).

Доказательство.

1. Необходимость.

Пусть эйлерова цепь существует и проходит из вершины *x* в вершину *y*. Если эта цепь – цикл $(x = y)$, то в каждую вершину цепь должна зайти столько же раз, сколько и выйти (т.е. валентности всех вершин – четные). Если $x \neq y$, то из x цепь должна выйти на один раз больше, чем зайти; в y цепь должна зайти на один раз больше, чем выйти (т.е. валентности вершин x и y должны быть нечетными).

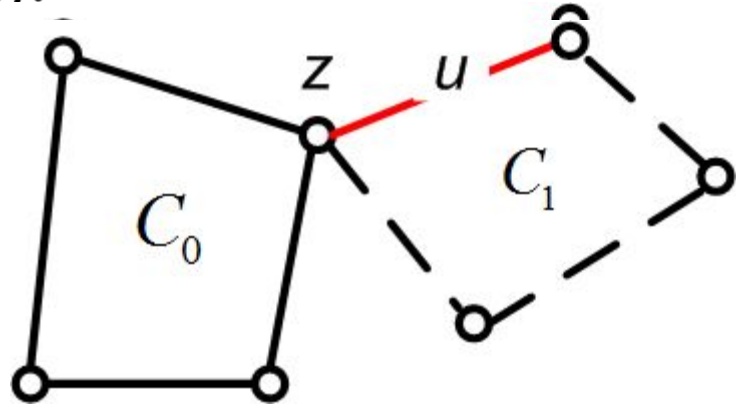
2. Достаточность (конструктивное доказательство).

Пусть валентности всех вершин, за исключением может быть вершин x и y, четные. Докажем существование эйлерова цикла, построив его.

Если все валентности четные: начнем цепь из некоторой произвольной вершины x и пойдем по некоторому еще не пройденному ребру к следующей вершине и так до тех пор, пока не вернемся в вершину x и не замкнем цикл.
Такой цикл обязательно найдется вследствие связности и четности валентностей вершин.
Если пройдены все ребра, то искомый эйлеров цикл C_0 построен.

Достаточность (продолжение).

Если в C_0 входят не все ребра графа, то цикл C_0 должен проходить через некоторую вершину z, инцидентную некоторому ребру u, не вошедшему в C_0 (т.к. граф связен).



Если удалить все ребра цикла C_0, то в оставшемся суграфе вершины будут по-прежнему иметь четные валентности. Начиная теперь с вершины z построим цикл C_1, начинающийся и кончающийся в z.

Достаточность (продолжение).

Если в C_1 пройдены все оставшиеся ребра, то процесс закончен. Нужный нам эйлеров цикл будет образован частью цикла C_0 до z, затем циклом C_1 и, наконец, частью цикла C_0 от z.

Если циклы C_1 и C_1 содержат не все ребра графа, то строится следующий цикл, и так далее, до тех пор, пока не будет найден эйлеров цикл.

Достаточность (окончание).

Пусть теперь вершины x и y имеют нечетные валентности. Если они смежны, то удалим ребро, соединяющее x и y. Теперь валентности всех вершин станут четными и, значит, существует эйлеров цикл. Построим эйлеров цикл из вершины x, а затем добавим ребро (x,y). В результате получим эйлерову цепь, соединяющую x и y.

Если вершины x, y несмежны, то добавим ребро (x,y). Снова валентности всех вершин станут четными. Построим эйлеров цикл с началом (и концом) в вершине y, а затем удалим ребро (x,y). Останется эйлерова цепь из x в y.

На этом доказательстве основан алгоритм нахождения эйлеровой цепи.

В 1736 г. Л.Эйлер опубликовал эту теорему, носящую в литературе его имя, без доказательства

e, f et g. Circa hos pontes iam ista proponebatur quaestio, num quis cursum ita instituere queat, ut per singulos pontes semel et non plus quam semel transeat. Hocque fieri posse, mihi dictum est, alios negare alios dubitare; neminem vero affirmare. Ego ex hoc mihi sequens maxime generale formavi problema: quaecunque sit fluvii figura et distributio in ramos atque quicunque fuerit numerus pontium, invenire, utrum per singulos pontes semel tantum transiri queat an vero secus.

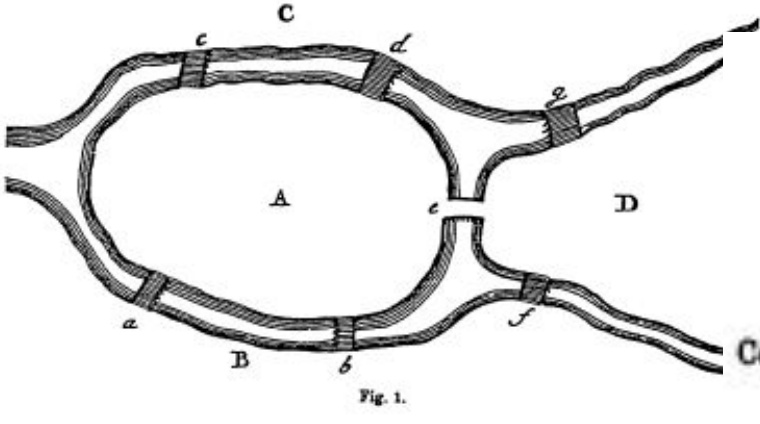


Fig. 1.

3. Quod quidem ad problema Regiomontanum de septem pontibus attinet, id resolvi posset facienda perfecta enumeratione omnium cursuum, qui institui possunt; ex his enim innotesceret, num quis cursus satisfaceret an vero nullus. Hic vero solvendi modus propter tantum combinationum numerum et nimis esset difficilis atque operosus et in aliis quaestionibus de multo pluribus pontibus ne quidem adhiberi posset. Hoc porro modo si operatio ad finem perducatur, multa inveniuntur, quae non erant in quaestione; in quo procul dubio tantae difficultatis causa consistit. Quamobrem missa hac me-

Фрагмент статьи Эйлера

SOLUTIO PROBLEMATIS AD GEOMETRIAM SITUS PERTINENTIS

Commentatio 53 indicis Enestroemiani

Commentarii academiae scientiarum Petropolitanae 8 (1736), 1741, p. 128—140

Алгоритм Хирхольцера

В 1873 г. вышла статья немецкого математика К. Хирхольцера
(**Carl Hierholzer,** 1840 – 1871), уже после смерти автора, содержащая конструктивное доказательство теоремы Эйлера.

C. Hierholzer: *Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren*.
Mathematische Annalen VI (1873), 30–32.

https://en.wikipedia.org/wiki/Carl_Hierholzer
https://www.youtube.com/watch?v=UgSytgwiM_A

Алгоритм Хирхольцера выполняется в точном соответствии
с доказательством достаточности теоремы Эйлера.

Рассмотрим обыкновенный связный граф

$$G = (V, E), \quad |V| = n, \quad |E| = m,$$

без вершин с нечетной степенью (в обыкновенном графе валентность вершины равна ее степени), в котором маршрут
определяется последовательностью вершин.

Алгоритм 1.

Обозначения.

C – список, эйлеров цикл, результат работы алгоритма;

S – список, промежуточный цикл;

s – начальная вершина каждого промежуточного цикла;

Шаг 0.

$C := \varnothing;\ \ S := \varnothing;$

Выбрать начальную вершину *s*;

$C \Leftarrow s.$ /* Поместить *s* в *C* */

Шаг 1. Выбрать в *C* вершину *s*, инцидентную еще не пройденному ребру.
Если таких вершин нет, **то** {**Выход –** *C* искомый эйлеров цикл}.

$S \Leftarrow s;\ \ x := s.$

/* Поместить *s* в *S* */

Шаг 2. /* Поиск цикла */

$\{x, y\}$ – /* выбрать не пройденное ребро*/.

$S := S \cup \{y\};$ /* Добавить *y* в конец списка *S* */

$x := y;$

$S := \varnothing.$

Если $x \neq s$ **то на Шаг 2**.

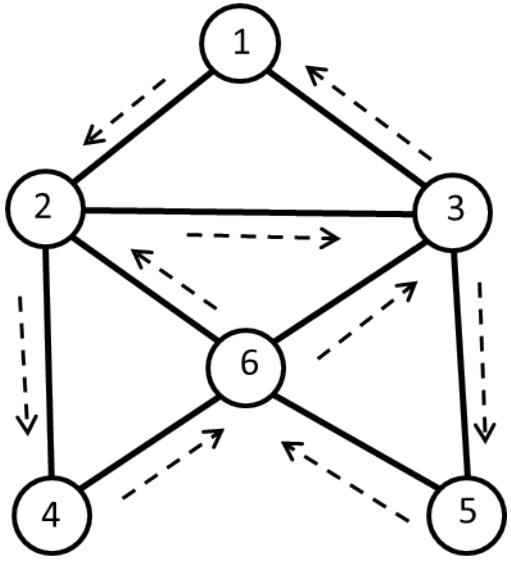
Шаг 3. /* найден цикл *S* */

В списке *C* заменить вершину *s* на список *S*;

$S := \varnothing;$

На Шаг 1.

Пример.



Выберем начальную вершину $s = 1$ и поместим ее в список *C*: $C = [1].$ Выполнив нужное число раз **Шаг 2**, найдем цикл $S = [1, 2, 3, 1]$. В списке *C* заменим вершину 1 на список *S*: $C = [1, 2, 3, 1]$ и очистим список *S*: $S := \varnothing$.

В списке *C* есть вершина 2, инцидентная еще не пройденным ребрам. Строим новый цикл *S* с начальной вершиной 2: $S = [2, 4, 6, 2]$. В списке *C* заменим вершину 2 на список *S*: $C = [1, 2, 4, 6, 2, 3, 1]$ и очистим список *S*: $S := \varnothing$. В списке *C* есть вершина 6, инцидентная еще не пройденным ребрам. Новый цикл с начальной вершиной 6 $S = [6, 3, 5, 6]$ поместим в список *C*: $C = [1, 2, 4, 6, 3, 5, 6, 2, 3, 1]$. В цикле *C* больше нет вершин, инцидентных еще не пройденным ребрам. Это и есть нужный эйлеров цикл.

На поиск вершин, инцидентных еще не пройденным ребрам, понадобится самое большее n просмотров; на включение ребер в цикл понадобится еще m просмотров. Таким образом, сложность алгоритма имеет порядок $O(nm)$

Описание и визуализацию алгоритма 1 Хирхольцера можно найти на сайте Мюнхенского технического университета

http://www-m9.ma.tum.de/graph-algorithms/hierholzer/index_en.html

Другой вариант реализации алгоритма Хирхольцера приводится в книге
Липский В. Комбинаторика для программистов: Пер. с польск.– М.: Мир, 1988. – 213 с. с илл.

Алгоритм 2 [Липский].

Задан обыкновенный связный граф $G = (V, E),\ |V| = n,\ \ |E| = m,$ без вершин с нечетной степенью.

Граф задан списками смежности вершин $\Gamma[x],\ x \in V.$

C – стек, эйлеров цикл, результат работы алгоритма;

S – вспомогательный стек, промежуточные циклы;

x, *y* – текущие вершины.

Шаг 0. /* Инициализация*/

$C := \varnothing;\ S := \varnothing;$

x:= произвольная вершина графа;

$S \Leftarrow x;$ /* поместить *x* в стек *S**/

Шаг 1.

если $S = \varnothing$ **то** {**Выход** – *C* искомый эйлеров цикл}.

/* Поиск цикла*/

$x := top(S);$ /* := верхний элемент стека *S* */

если $\Gamma[x] = \varnothing,$ **то на Шаг 2.** /* Найден конец текущего цикла */

$y := top(\Gamma[x]);$ */ и/или нет ребер инцидентных x */

$S \Leftarrow y;$ /* y:=верхний элемент списка $\Gamma[x]$ */

/* Поместить y в стек *S* */

/* Удалить ребро $\{x, y\}$ из графа */

$\Gamma[x] := \Gamma[x] \setminus \{y\};$

$\Gamma[y] := \Gamma[y] \setminus \{x\};$ /* Удалить вершину y из списка $\Gamma[x]$ */

/* Удалить вершину x из списка $\Gamma[y]$ */

на Шаг 1;

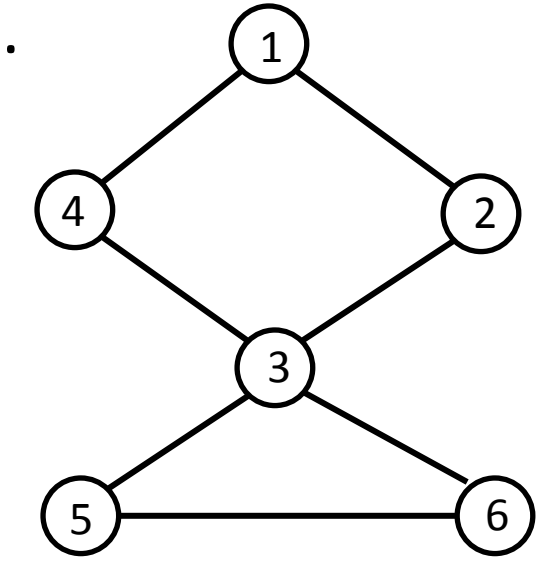
Шаг 2; /* если $\Gamma[x] = \Gamma$ */

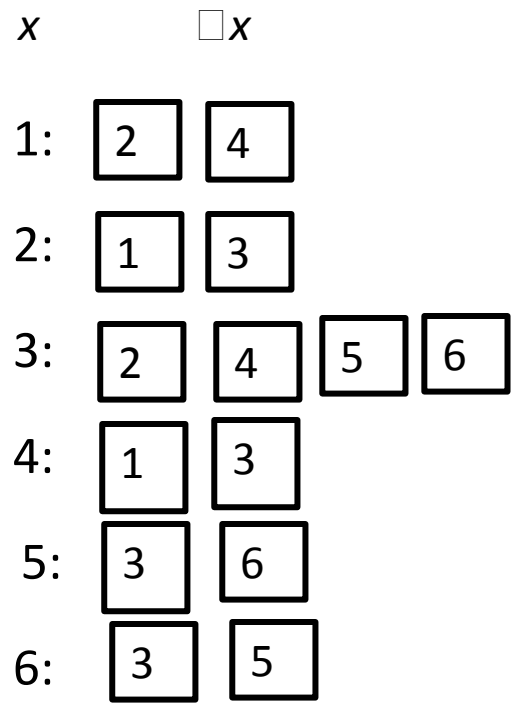
$x \Leftarrow S; C \Leftarrow x;$

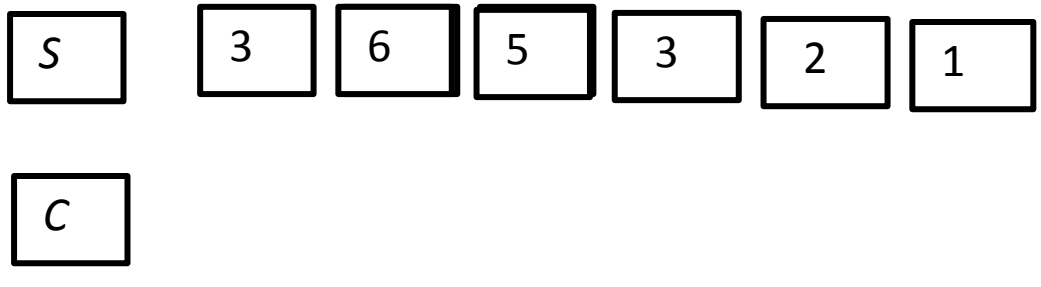
/*Удалить вершину x из стека S и поместить в стек C

на Шаг 1.

Пример








```
begin
 S:=∅; C:=∅;
 x:=произвольная вершина графа
 S ← x; /* поместить x в стек S*/
 while S≠∅ do
   begin x:=top(S); /*x=верхний
элемент S*/
      if  Γ[x] ≠∅ then
      begin  y:=top(Γ[x] );
      S ← y; /* поместить y в стек S*/
     /*удалить ребро {x,y} из графа*/
       Γ[x]:= Γ[x]\{y};
       Γ[y]:= Γ[y]\{x};
      end
       else  /* Γ[x] =∅*/
/*удалить x из S  и поместить в C */
       begin x ← S;  C ← x; end
   end
end
```

Эйлеровы пути.

Введем следующие обозначения:

$$v^+(x) = s^+(x) + s^\circ(x);$$

$$v^-(x) = s^-(x) + s^\circ(x).$$

v^+ и $v^-(x)$ – *полувалентности* вершины .

Путь в орграфе , содержащий все его дуги, называется *эйлеровым* путем. (Напомним, что путь – это ориентированная цепь.)

Теорема. Эйлеров путь из вершины s в вершину t существует тогда и только тогда, когда

1. $x \in V \setminus \{s,t\}$ $[v^+(x) = v^-(x)]$.
2. $s \neq t : v^+(s) - v^-(s) = 1; v^-(t) - v^+(t) = 1.$

(для всех вершин, кроме s и t, полувалентности исхода и полувалентности захода равны; для вершины s полувалентность исхода на единицу больше,
чем полувалентность захода;
для вершины t полувалентность захода на единицу больше,
чем полувалентность исхода.)

Доказательство полностью аналогично предыдущему.

Задача китайского почтальона

В графе с неотрицательными весами ребер найти циклический маршрут наименьшей длины, проходящий через каждое ребро графа по крайней мере один раз.

Почтальон называется китайским – первоначально эту задачу изучал китайский математик Мэй-Ку Куан в 1962 году.

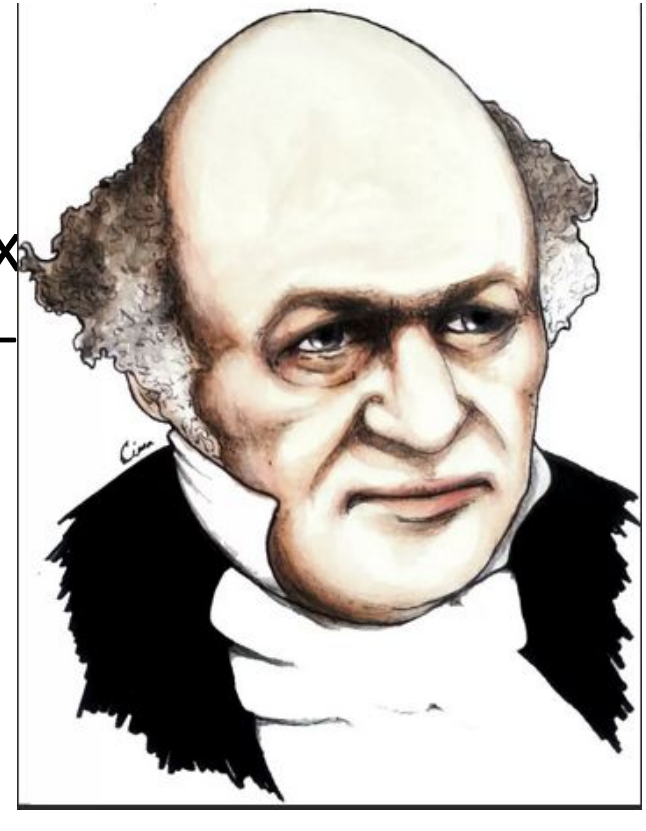
Очевидно, если граф эйлеров, то эйлеров цикл и будет оптимальным.

Если граф не эйлеров, то задача становится весьма трудоемкой. Для ее решения имеются специальные алгоритмы, сокращающие число перебираемых вариантов.

3.1. Гамильтоновы цепи и циклы

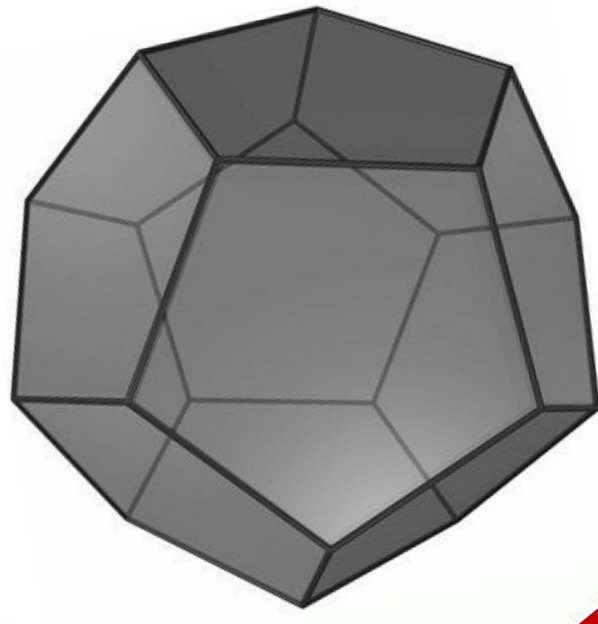
Постановка задачи

Определение. ***Гамильтоновой цепью (циклом)*** графа называется простая цепь (простой цикл), содержащая (содержащий) все вершины графа.

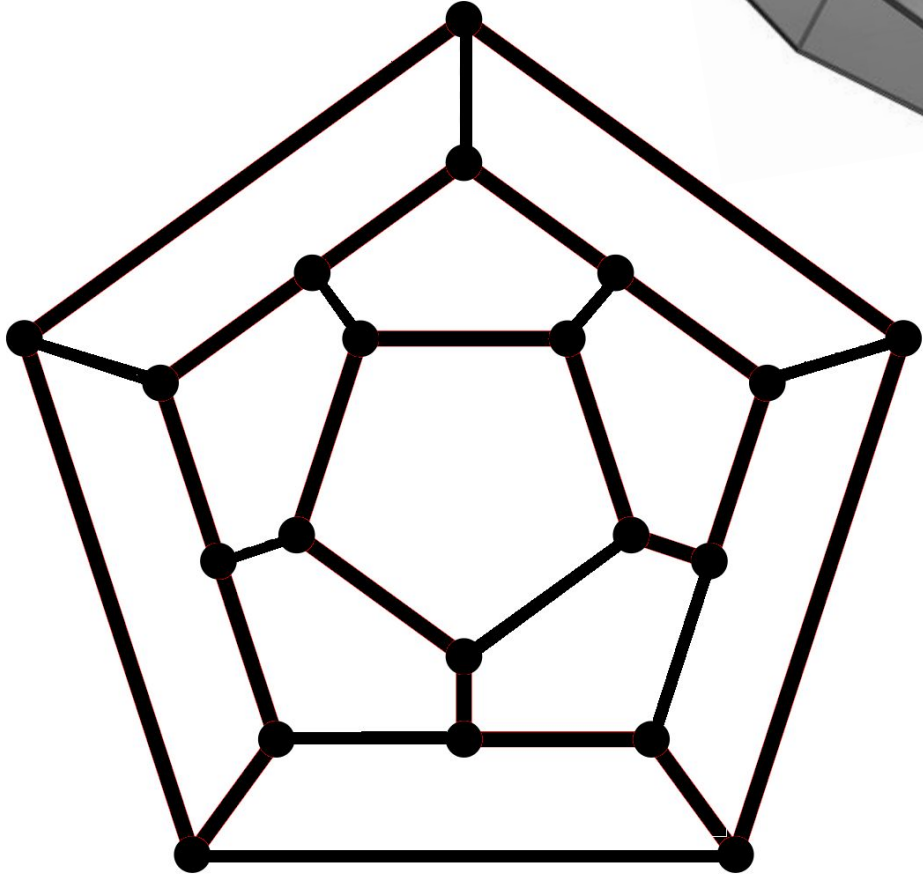


Сэр **Уильям Роуэн Гамильтон** (1805-1865) – самый известный ирландский математик, один из лучших математиков 19 века. Известен фундаментальными открытиями в математике , механике, физике (векторный анализ, вариационное исчисление, общий прин-цип наименьшего действия в механике, теория распространения света и др.)

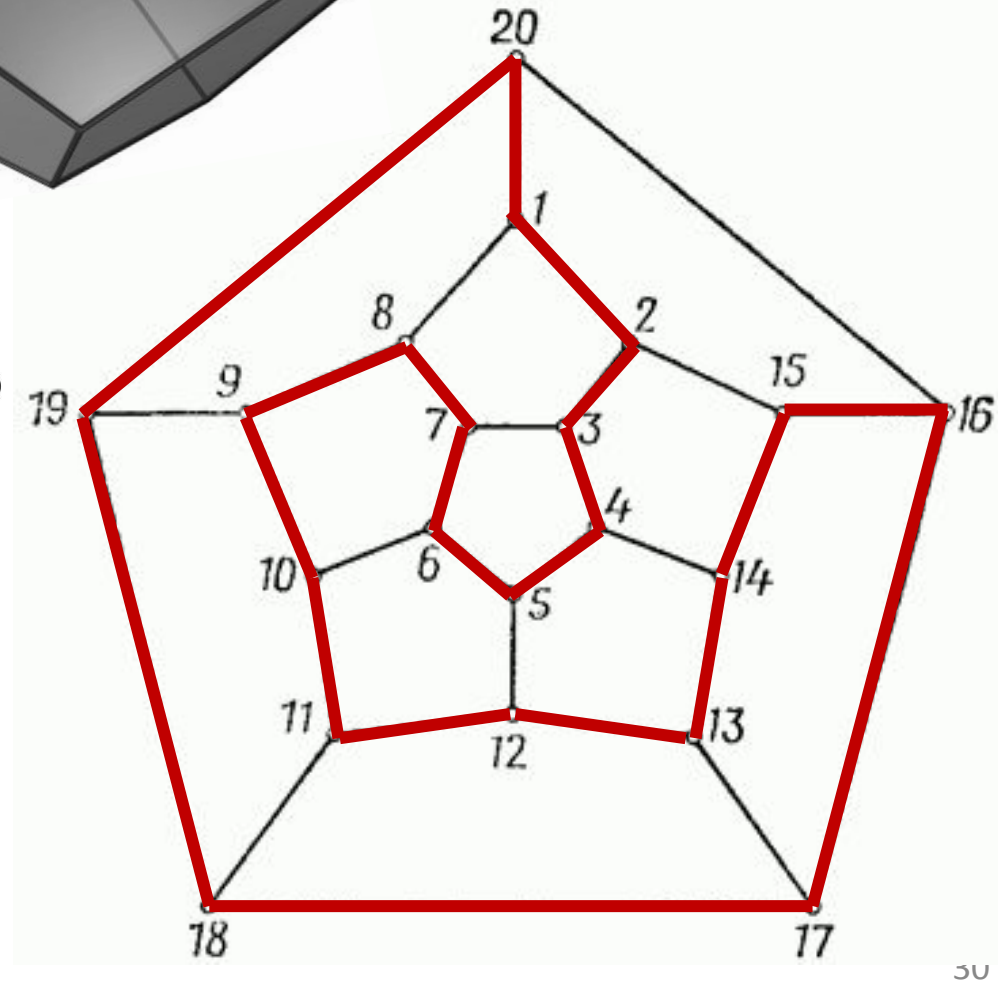
В 1859 г. Гамильтон изобрел головоломку обхода вершин додекаэдра



Додекаэдр:
12 граней,
20 вершин,
30 ребер



Граф Гамильтона



Гамильтонов цикл

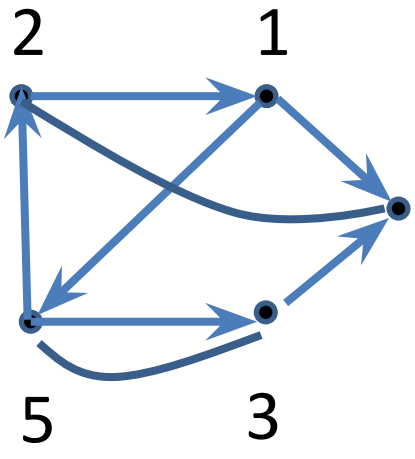
Понятия гамильтоновой цепи и цикла естественным образом обобщаются на случай ориентированных графов: двигаться в очередную вершину можно только по направлениям, указанным стрелками.

Пока неизвестен какой-либо достаточно простой критерий необходимости и достаточности существования гамильтоновой цепи (цикла) для произвольного графа G.

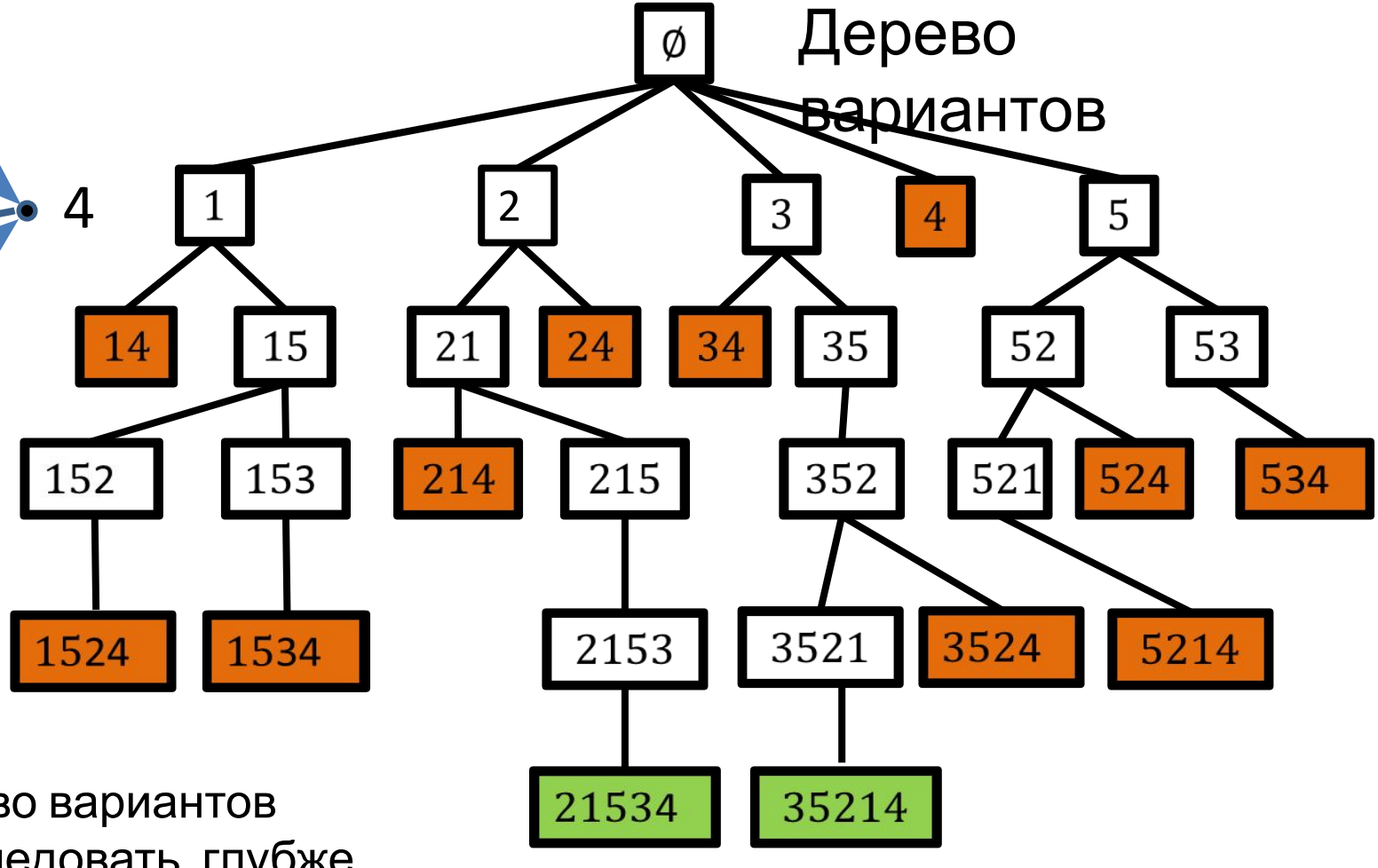
Неизвестен также алгоритм нахождения гамильтоновой цепи (цикла) полиномиальной сложности.

Мы рассмотрим два простейших переборных алгоритма поиска гамильтоновой цепи (цикла), пригодных для графов малой размерности: поиск в ширину и поиск в глубину.

Поиск в ширину



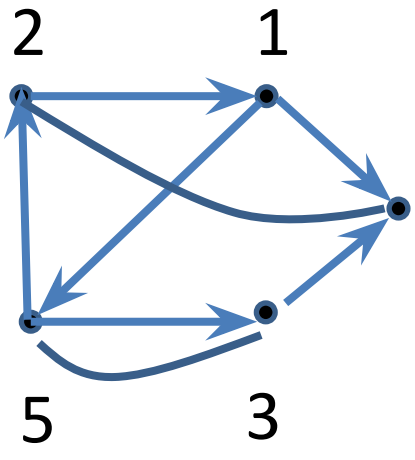
Дерево вариантов

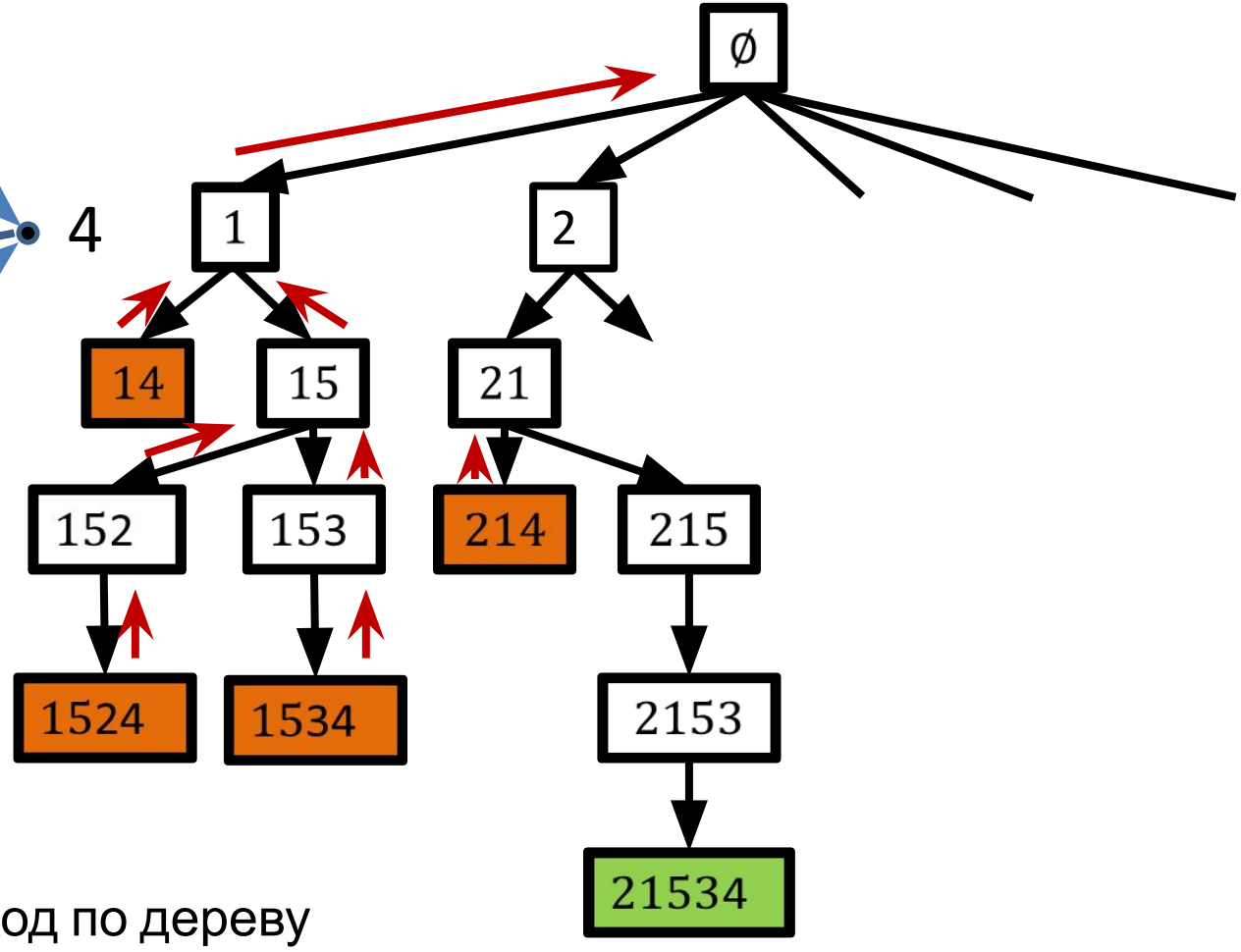


- множество вариантов стоит исследовать глубже

- бесперспективное (тупиковое) множество вариантов

Поиск в глубину





- прямой ход по дереву поиска

- обратный ход **(back tracking)**

Задача коммивояжера

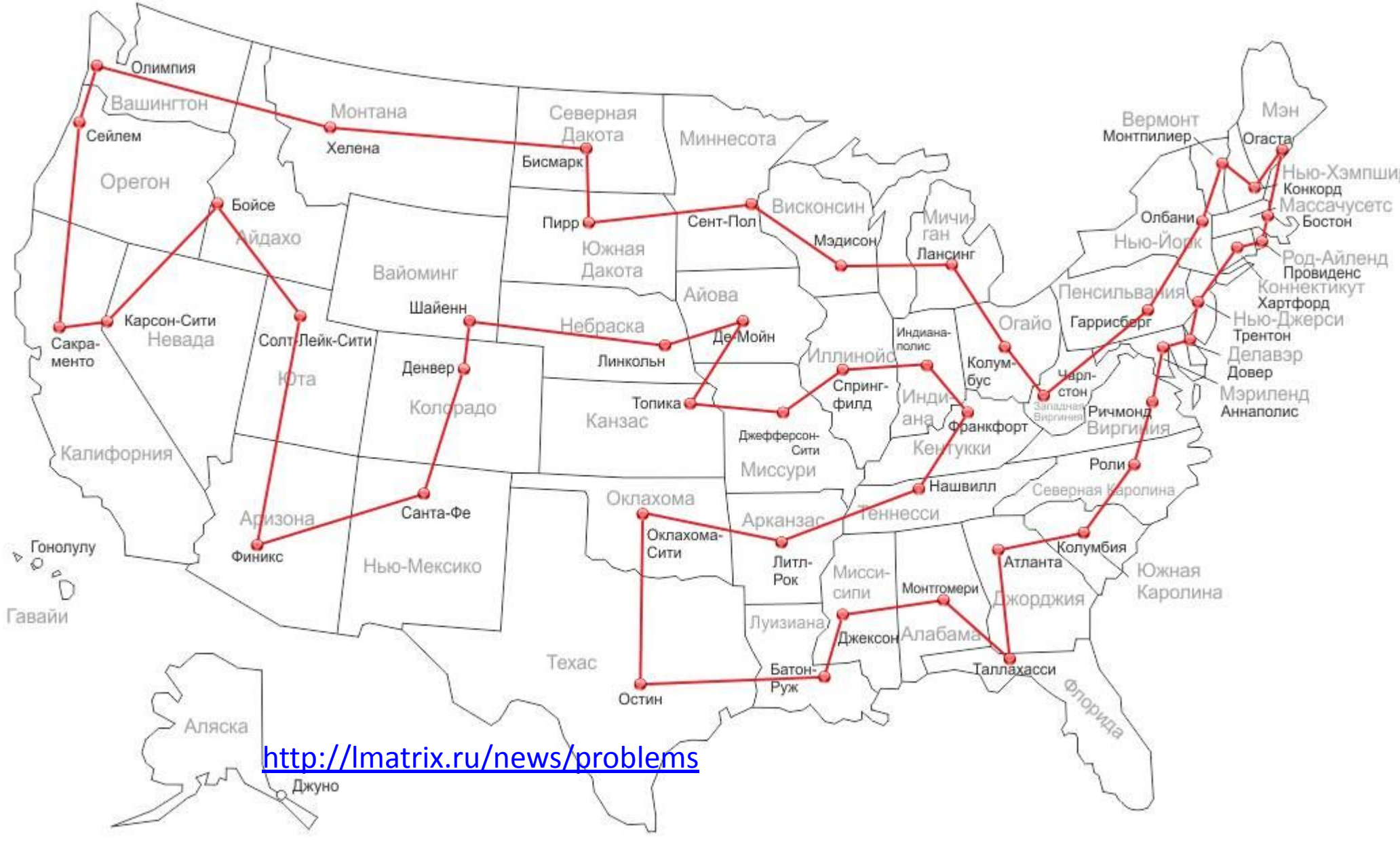
Задача коммивояжера (***travelling seller problem, TSP***) формулируется следующим образом: во взвешенном графе (обычно – полном) найти гамильтонову цепь (цикл) наименьшей длины (с наименьшим суммарным весом ребер).

Название задачи происходит от следующей формулировки: имеется n городов и известны расстояния между каждой парой городов; коммивояжер (бродячий торговец), выходящий из какого-либо города, должен посетить $n - 1$ других городов и вернуться в исходный. В каком порядке ему нужно посещать города, чтобы общее пройденное расстояние было минимальным?

Самый простой способ – перебрать все $(n-1)!$ гамильтоновых циклов (если граф полный), однако трудоемкость этого перебора имеет порядок n^n. Уже при $\mathrm{n} > 60$ перебор невозможен никакими теоретически мыслимыми компьютерами за время, меньшее нескольких миллиардов лет. Но это – единственный способ найти **точное** решение.

Существует множество алгоритмов нахождения **приближенных** решений. Задача коммивояжера из-за простоты постановки и наглядности является прекрасной моделью для разработки новых алгоритмов дискретной оптимизации. К ним относятся алгоритмы. реализующие **метод ветвей и границ, генетические алгоритмы** и др.

Пример. Столицы 48 штатов



http://lmatrix.ru/news/problems

Длина пути 16675 миль. Алгоритм LMatrix

Решение задачи коммивояжера по городам Золотого
Кольца. Длина маршрута: 1643 км. Алгоритм Lmatrix.