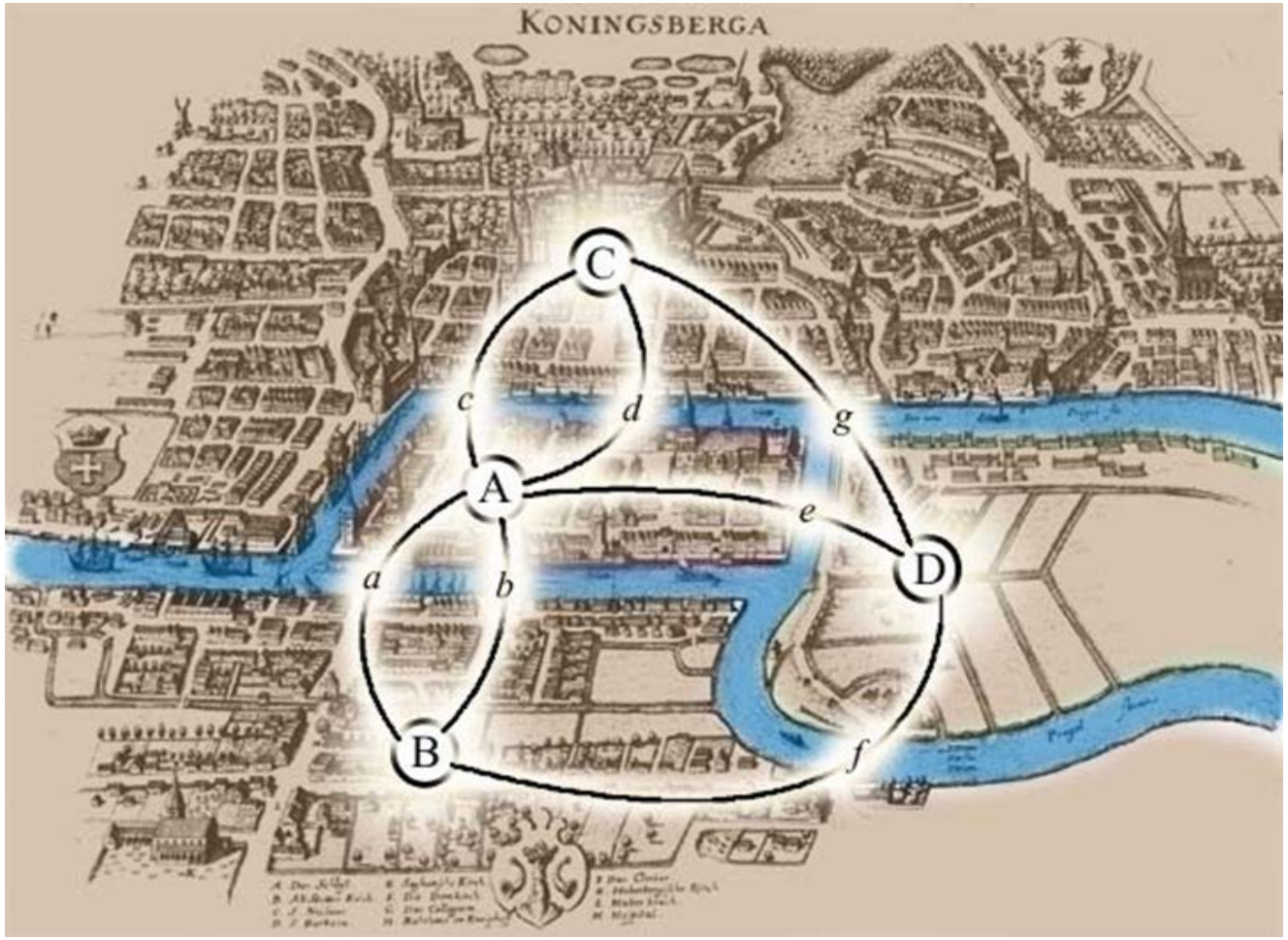
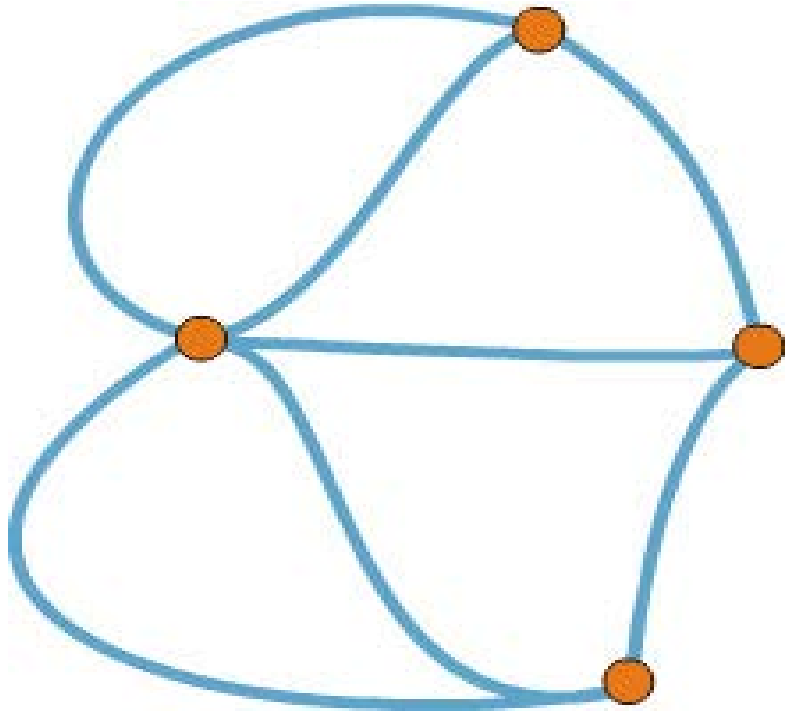
ЭЙЛЕРОВЫ ПУТИ И ЦИКЛЫ

Граф Кёнигсбергских мостов

Задача о семи кёнигсбергских мостах — старинная математическая задача, в которой спрашивалось, как можно пройти по всем семи мостам Кенигсберга, не проходя ни по одному из них дважды. Впервые была решена в 1736 году математиком Леонардом Эйлером, доказавшим, что это невозможно, и изобретшим таким образом эйлеровы циклы.





Произвольный путь в графе, проходящий через каждое ребро графа точно один раз, называется ЭЙЛЕРОВЫМ ПУТЕМ.

(По вершинам путь может проходить неоднократно, то есть путь может быть не простой). Т.е., в пути v(1)-v(2)-...-v(m+1) каждое ребро графа появляется один и только один раз как { v(i),v(i+1) } для некоторого i.

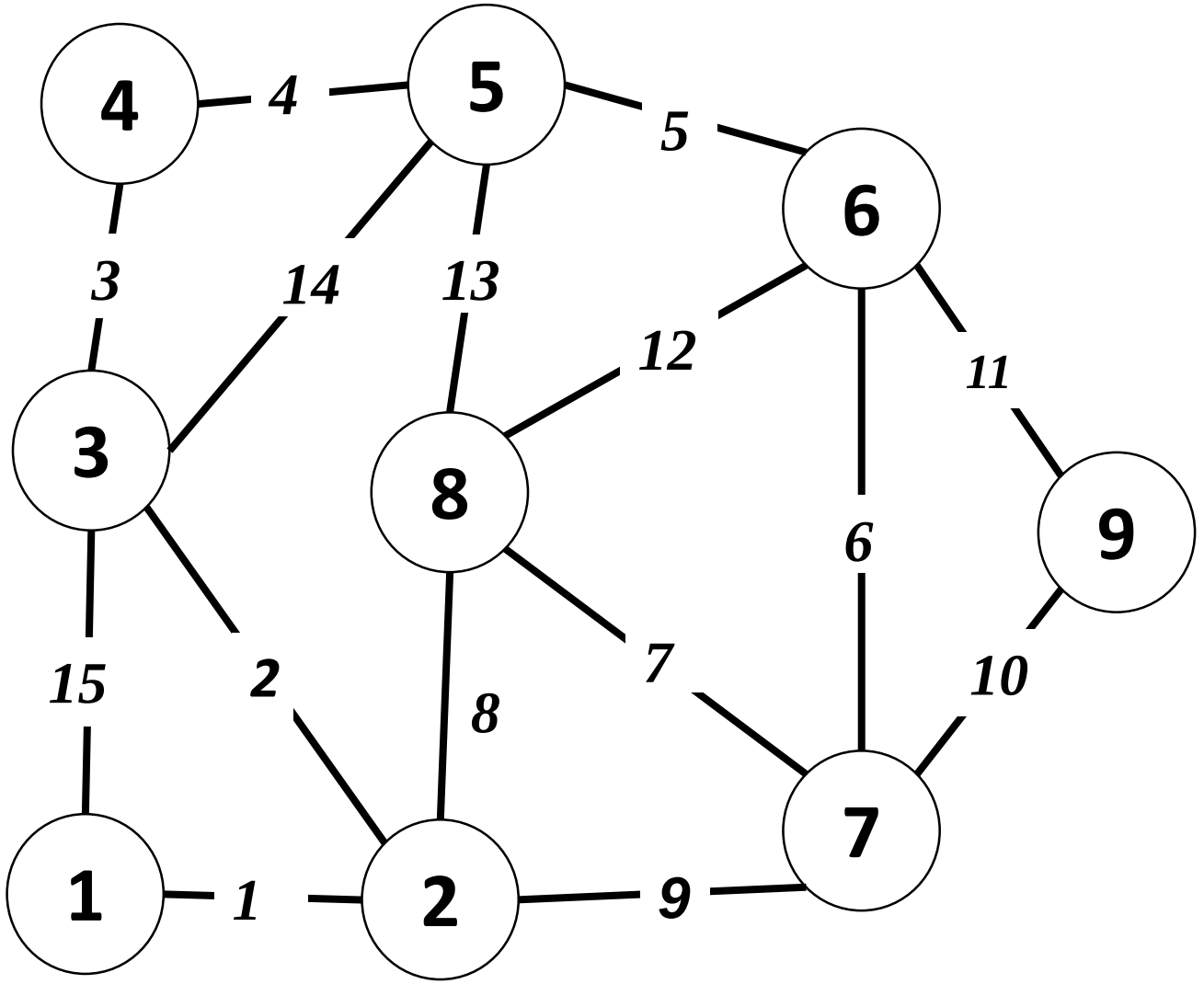
Если при этом v(1)=v(m+1), то имеем ЭЙЛЕРОВ ЦИКЛ.

- Для существования эйлерова **пути** в связном графе необходимо и достаточно, чтобы граф содержал **не более 2** вершин нечетной степени.
- Связный граф является эйлеровым (***обладает эйлеровым циклом***) ***тогда и только тогда***, когда каждая вершина имеет четную степень.

В алгоритме поиска эйлерова цикла связный граф G=<V,E> описан двунаправленным списками ZAP[v], а эйлеров цикл представлен последовательностью вершин в рабочем стеке ST; стек SE содержит последовательность полностью обработанных вершин. Перед поиском цикла необходимо убедиться, что он существует. То есть все вершины графа четные.

1. Выбрать произвольную вершину V.
2. Поместить V в рабочий стек ST.
3. Пока стек ST не пуст делать:
   - Снимаем из стека вершину X
   - Если существует непройденное ребро {xy} тогда помечаем {xy} как пройденное и добавляем вершину Y в рабочий стек ST.
   - Иначе перемещаем вершину X из рабочего стека ST в стек результата SE

ZAP	ST	SE
1: 2, 3	1	4
2: 1, 3, 4	2	2
3: 1, 2, 4, 5	3	7
4: 3, 5	4	9
5: 3, 4, 6, 8	5	6
6: 5, 7, 8, 9	6	8
7: 2, 6, 8, 9	7	5
8: 2, 5, 6, 7	8	3
2: 1, 3, 7, 8	2	1
7: 2, 6, 8, 9	7	
9: 6, 7	9	
6: 5, 7, 8, 9	6	
8: 2, 5, 6, 7	8	
5: 3, 4, 6, 8	5	
3: 1, 2, 4, 5	3	
	1	



Для поиска Элерова пути:

Сначала выбирается произвольная вершина **v**. Если есть нечетные, то это одна из них.

Вершины с нечетной степенью должны быть конечными (можно их перенумеровать, не более 2).

В цикле по условию опустошения стека **ST** строится путь с началом в **v**.

Вершины строящегося пути размещаются в **ST**, ребра, входящие в путь, удаляются из графа (соответствующие элементы удаляются из списков **ZAP**).

Двунапрнавленность списков инцидентности **ZAP[v]** позволяет устранить ребро **{v,u}** за время, ограниченное константой.

Общая сложность алгоритма: **O(m),** где **m** – количество ребер графа.

Для графа:

1 –> 2, 3

2 -> 1, 3, 7, 8

3 -> 1, 2, 4, 5

4 -> 3, 5

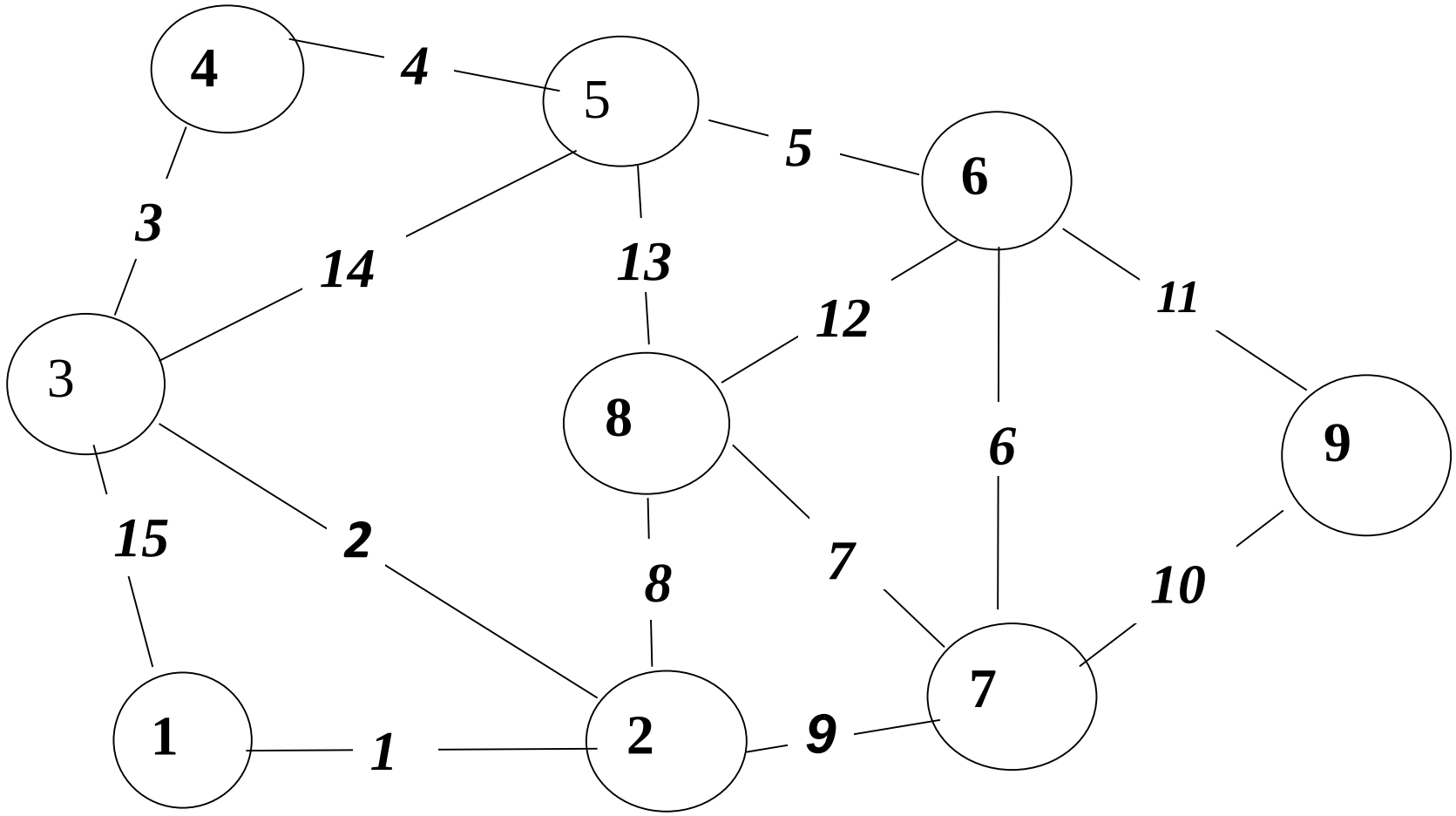
5 -> 3, 4, 6, 8

6 -> 5, 7, 8, 9

7 -> 2, 6, 8, 9

8 -> 2, 5, 6, 7

9 -> 6, 9



один из найденных эйлеровых циклов представлен последовательностью вершин:
1–2 –3 – 4 – 5 – 6 –7– 8 – 2 – 7 – 9 – 6 – 8 – 5 –3 – 1. ST – Эйлеров путь
SE 4 – 2 – 7 - 9 – 6 – 8 – 5 – 3 – 1 - последов. обработанных вершин

Гамильтоновы пути в графе

(алгоритмы с возвратом)

В 1857 г. ирландский математик ***Гамильтон*** предложил игру, названную "путешествие по додекаэдру".

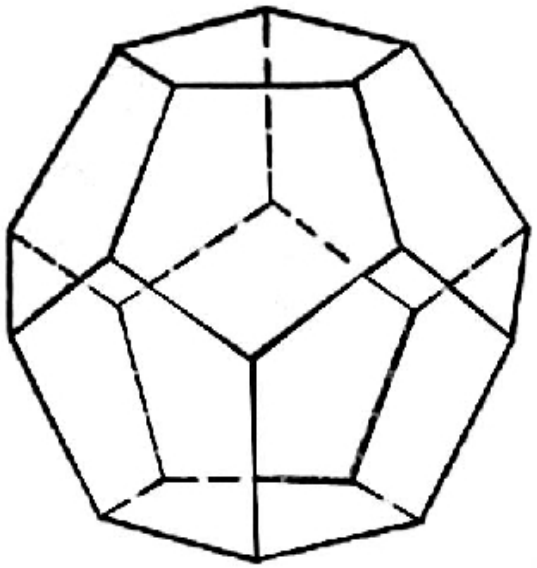
Игра сводилась к обходу по ребрам всех вершин правильного додекаэдра при условии, что ни в одну из вершин нельзя заходить более одного раза.

Додекаэдр - это многогранник, гранями которого служат 12 правильных пятиугольников. У него 20 вершин и 30 ребер.

Вершины и ребра додекаэдра составляют некоторый плоский граф.

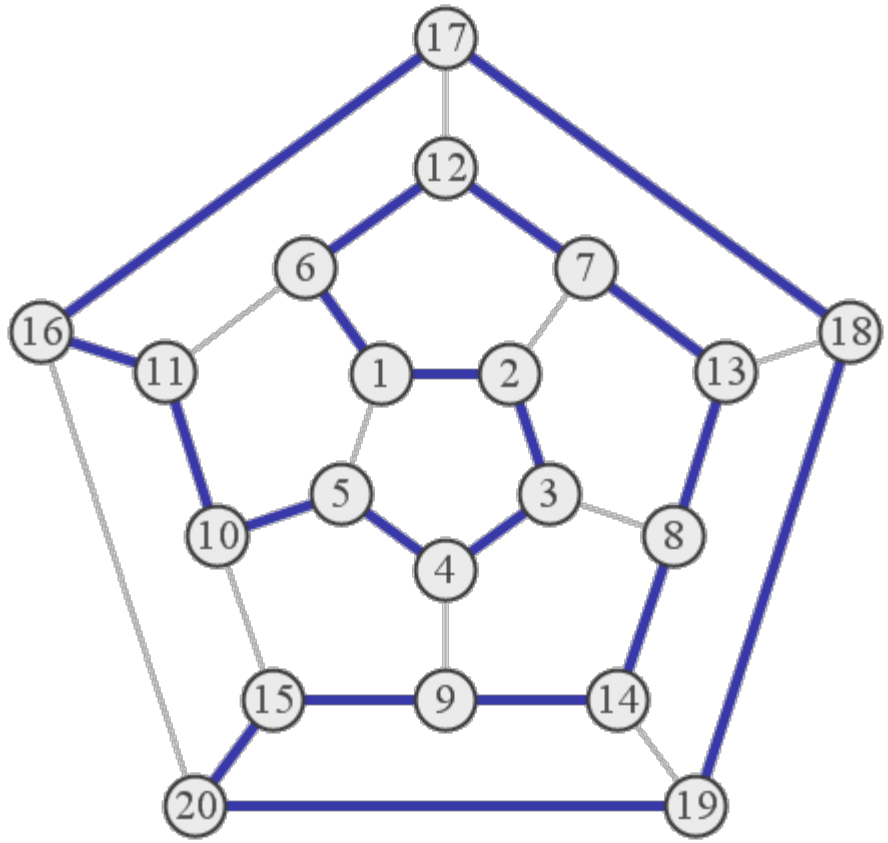
- Гамильтоновы путь, цикл и граф названы в честь ирландского математика Гамильтон, который впервые определил эти классы, исследовав задачу «кругосветного путешествия» (1857) по додекаэдру. В этой задаче вершины додекаэдра символизировали известные города, такие как Брюссель, Амстердам, Пекин, Прага, Дели, Прага и др., а рёбра — соединяющие их дороги. Путешествующий должен пройти «вокруг света», найдя путь, который проходит через все **вершины** ровно один раз· Чтобы сделать задачу более интересной, порядок прохождения городов устанавливался заранее. А чтобы было легче запомнить, какие города уже соединены, в каждую вершину додекаэдра был вбит гвоздь, и проложенный путь отмечался небольшой верёвкой, которая могла обматываться вокруг гвоздя. Однако такая конструкция оказалась слишком громоздкой, и Гамильтон предложил новый вариант игры, заменив додекаэдр плоским графом, изоморфным (количество компонентов (вершин и ребер) одинаково и их пограничное соединение сохраняется) графу, построенному на рёбрах додекаэдра.

Правильный додекаэдр



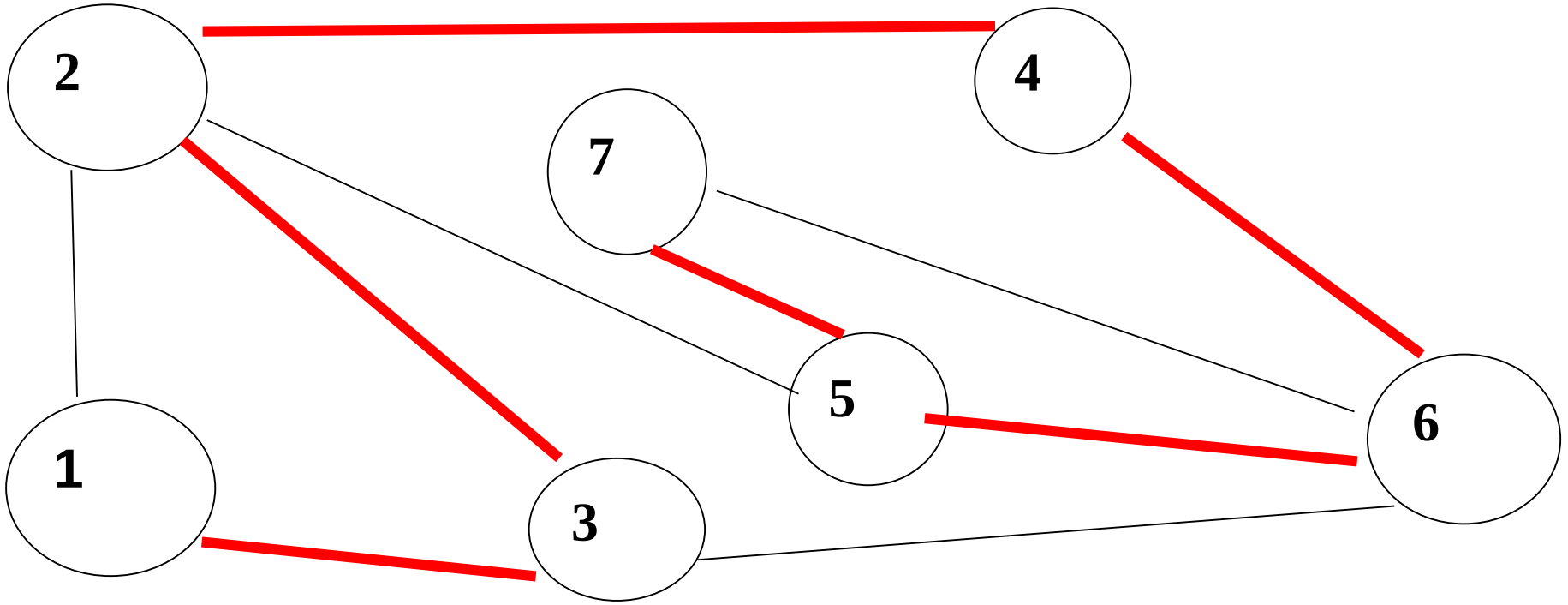
Составлен из двенадцати правильных пятиугольников. Каждая вершина додекаэдра является вершиной трёх правильных пятиугольников. Следовательно, сумма плоских углов при каждой вершине равна *324º*.

12 граней, 30 ребер, 20 вершин



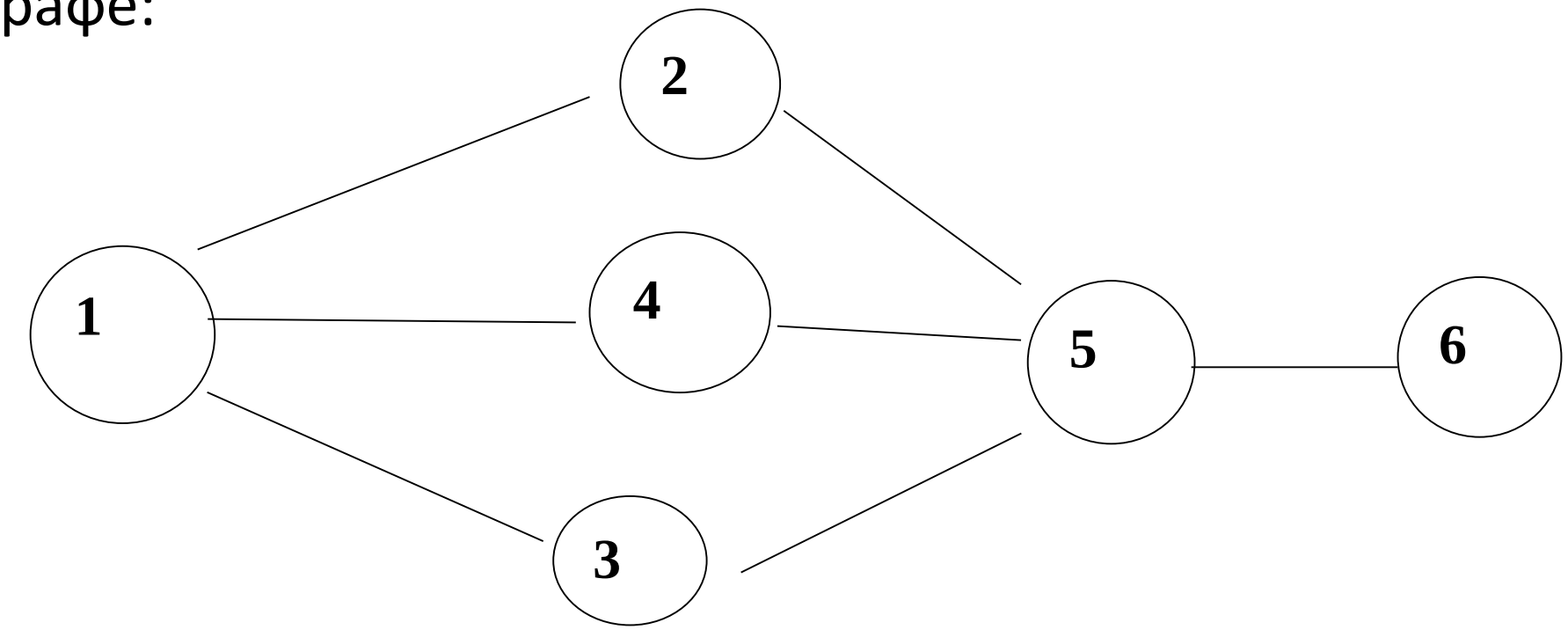
- Пусть $G = [A, B]$ - некоторый граф. Он называется ***гамильтоновым,*** если в нем существует простой цикл, содержащий **все вершины** графа. Т.е. в нем возможно посещение каждой вершины ровно 1 раз.
- Напр., каждый полный граф – гамильтонов, т.к. в нем проведены все возможные ребра и, в частности, те, благодаря которым возможен обход по всем вершинам.
- Общих и легко осуществляемых действий, с помощью которых можно было бы достоверно выяснить, является ли данный граф гамильтоновым, **не существует**. Но, имеются **достаточные** условия на гамильтоновость, которые проверить легко.
- Недостаток - даже если ни одно из этих условий не выполняется, граф может быть гамильтоновым

- Рассмотрим построение путей в графе, проходящих в точности один раз через каждую вершину (а не каждое ребро) графа. Такой путь и соответствующий цикл называются Гамильтоновыми и существуют не для каждого графа, как и Эйлеров путь. Например, в графе:
- 1 –> 2, 3
- 2 –> 1, 3, 4, 5
- 3 –> 1, 2, 6
- 4 –> 2, 6
- 5 -> 2, 6, 7
- 6 -> 3, 4, 5, 7
- 7 -> 5, 6



есть гамильтонов путь 1, 3, 2, 4, 6, 5, 7.

- В то же время в графе:
- 1 -> 2, 3, 4
- 2 -> 1, 5
- 3 -> 3, 5
- 4 -> 1, 5
- 5 -> 2, 3, 4, 6
- 6 -> 5



- гамильтонова пути не существует.

- В отличие от эйлеровых путей неизвестно ни одного простого **необходимого** и **достаточного** условия для существования гамильтоновых путей.

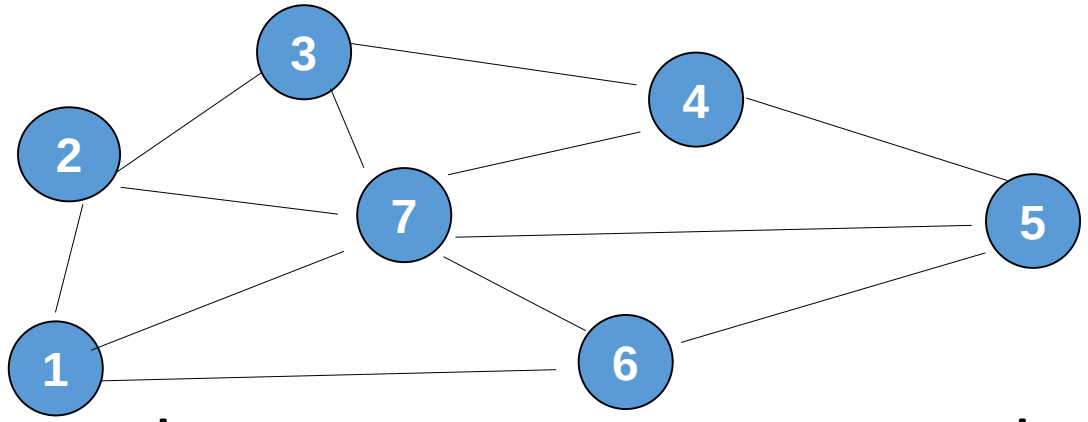
Условия Дирака, Оре и Поша, гарантирующие существование в графе гамильтонова цикла

условие Дирака

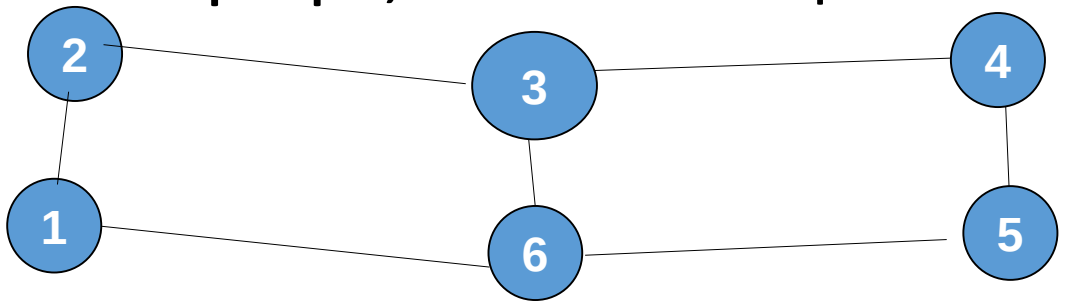
- Пусть p - число вершин в данном графе; если степень
- каждой вершины не меньше, чем целое от $\frac{p}{2}$, то граф называется ***графом Дирака***.
- Можно доказать, что ***каждый граф Дирака обязательно гамильтонов.***
- пример графа Дирака:

очевидно, этот граф

- гамильтонов.



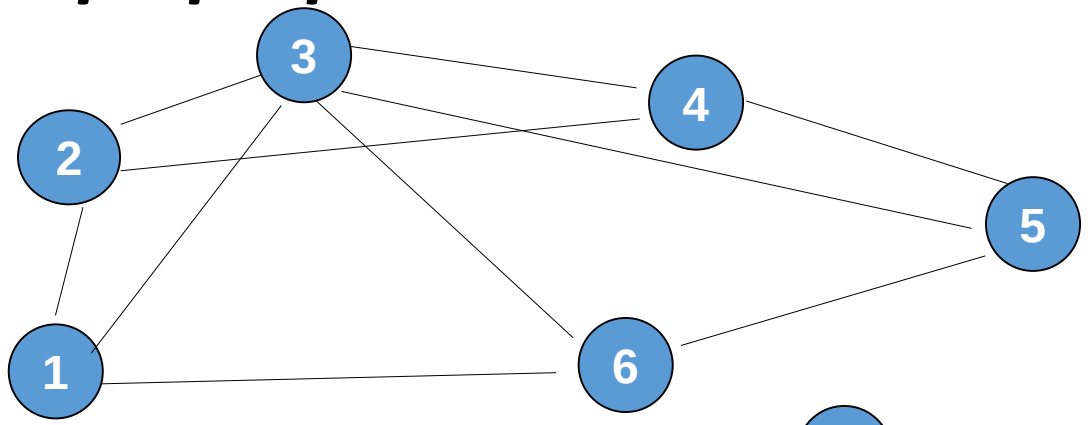
- А вот пример гамильтонова графа, не являющегося графом Дирака:



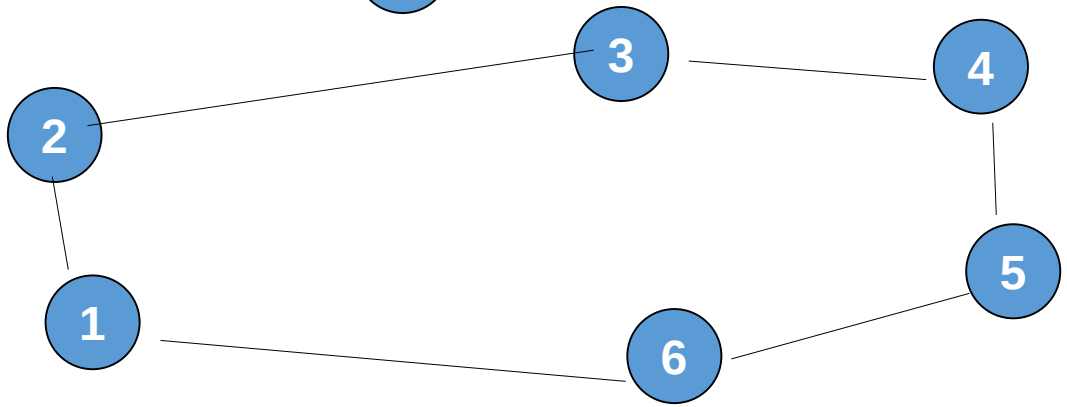
условие Оре

Если для любой пары несмежных вершин x, y выполнено неравенство $d(x) + d(y) \geq p$, то граф называется ***графом Оре*** (т.е. сумма степеней несмежных вершин >= числу вершин графа)

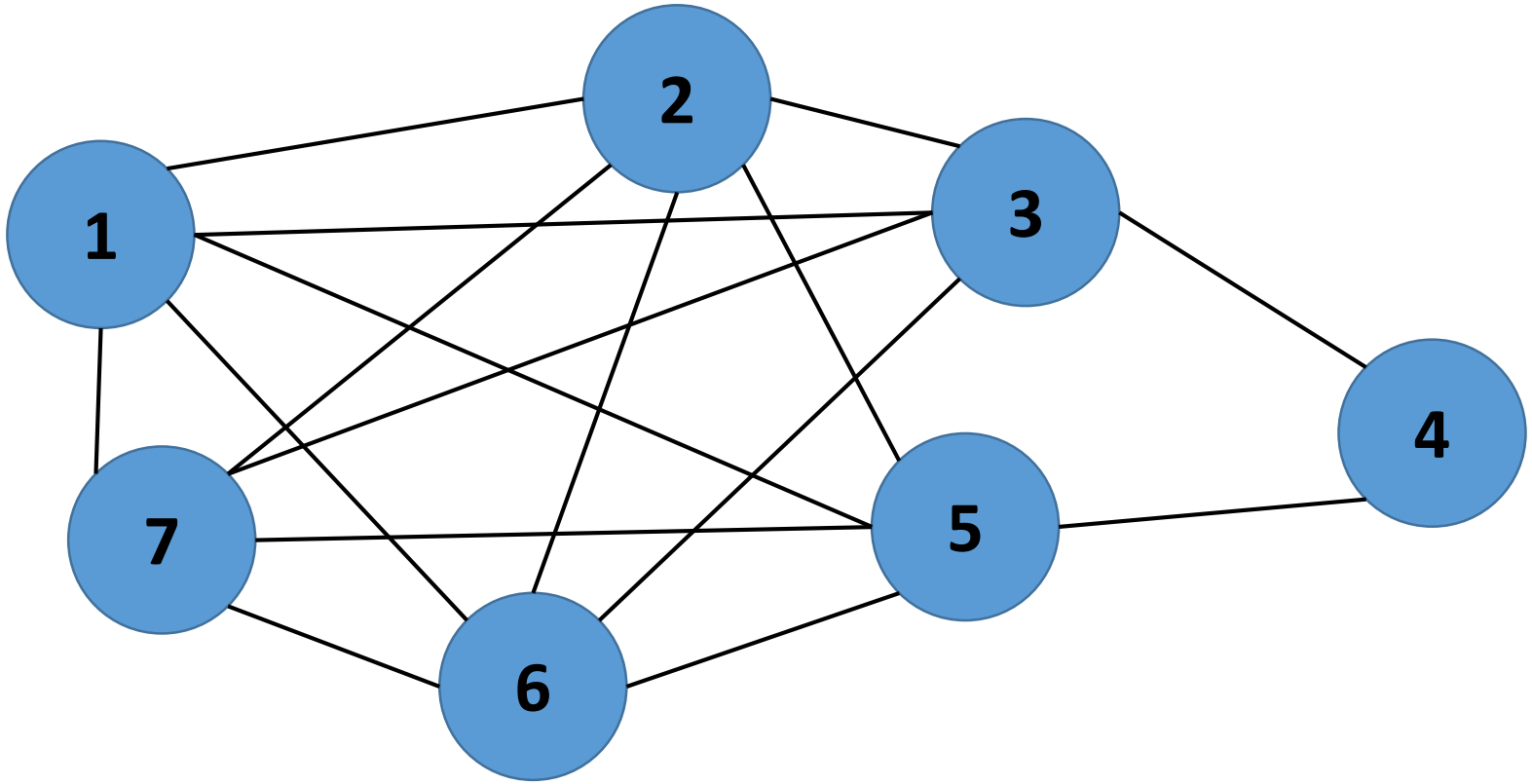
- Можно доказать, что ***всякий граф Оре обязательно гамильтонов***.
- пример графа Оре:
- очевидно, этот граф
- гамильтонов.



- пример гамильтонова графа
- но не граф Оре:



- Нетрудно заметить, что всякий граф Дирака автоматически является графом Оре. Обратное – неверно! Вот пример графа Оре, не являющегося графом Дирака:



условие Поша

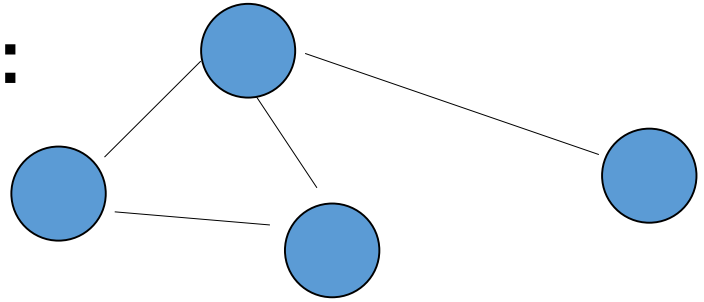
Введем функцию $f(x)$ целого неотрицательного аргумента X. Для графа $G=[A,B]$ запишем определение формулой, по которой строится функция $f(x)$

$$f(x)=\left|\left\{a\in A \mid d(a)\le x\right\}\right|$$

Т.е. функция $f(x)$ в каждом целом неотрицательном X принимает значение, равное количеству вершин указанного графа ***G***, степень которых не превосходит X.

Такую функцию ***F(x)*** называют *функцией Поша* для графа ***G*** .

построим функцию Поша следующего графа:



Функция будет описана таблично:

X	0	1	2	3	4	5	…
f (x)	0	1	3	4	4	4	…

Количество вершин, степень которых не больше x

сформулируем условие Поша:

Графом Поша называется граф ***G = [A,B]***, удовлетворяющий следующим условиям (число вершин графа обозначим через ***p***, его функцию Поша обозначим через ***f(x)***, ***x*** – целое число)

1) для $1 \le x < \frac{p-1}{2}$ **выполняется неравенство** $f(x) < x$

2) если $\frac{p-1}{2}$ **- целое число, то при** $x = \frac{p-1}{2}$

имеет место неравенство: $f(x) \le \frac{p-1}{2}$

Другими словами:

Пусть граф G имеет $\boldsymbol{p} \geqslant 3$ вершин и выполнены следующие два условия:

1) *Для* $1 \leqslant \boldsymbol{x} < (\boldsymbol{p}-1)/2$, число вершин со степенями, не превосходящими $\boldsymbol{X}$, меньше чем $\boldsymbol{X}$
2) для нечетного $\boldsymbol{p}$ число вершин степени $(\boldsymbol{p}-1)/2$ не превосходит $(\boldsymbol{p}-1)/2$

тогда G — гамильтонов граф

- Можно доказать, что *каждый граф Поша обязательно гамильтонов*. Легко заметить, что простой цикл на большом числе вершин, графом Поша не является, но является гамильтоновым графом.

Кроме того, нетрудно заметить, что ***всякий граф Дирака является графом Поша, каждый граф Оре является графом Поша***. Обратное в обоих последних случаях неверно.

- Неизвестен даже алгоритм полиномиальной сложности, проверяющий существование гамильтонова пути в произвольном графе. Проблема существования гамильтонова пути принадлежит к классу т.н. **NP**-полных задач.
- Т.е. полный перебор, т.е. **n!*n** шагов. Для сокращения перебора был разработан класс алгоритмов «с возвратом» (backtracking).

алгоритм поиска всех гамильтоновых циклов в графе

- в глобальном векторе X строится последовательность вершин:
- X[1],…X[k-1] строящегося пути;
- логический вектор DOP показывает, можно ли вставить вершины в проектируемый путь,
- n – кол-во вершин графа

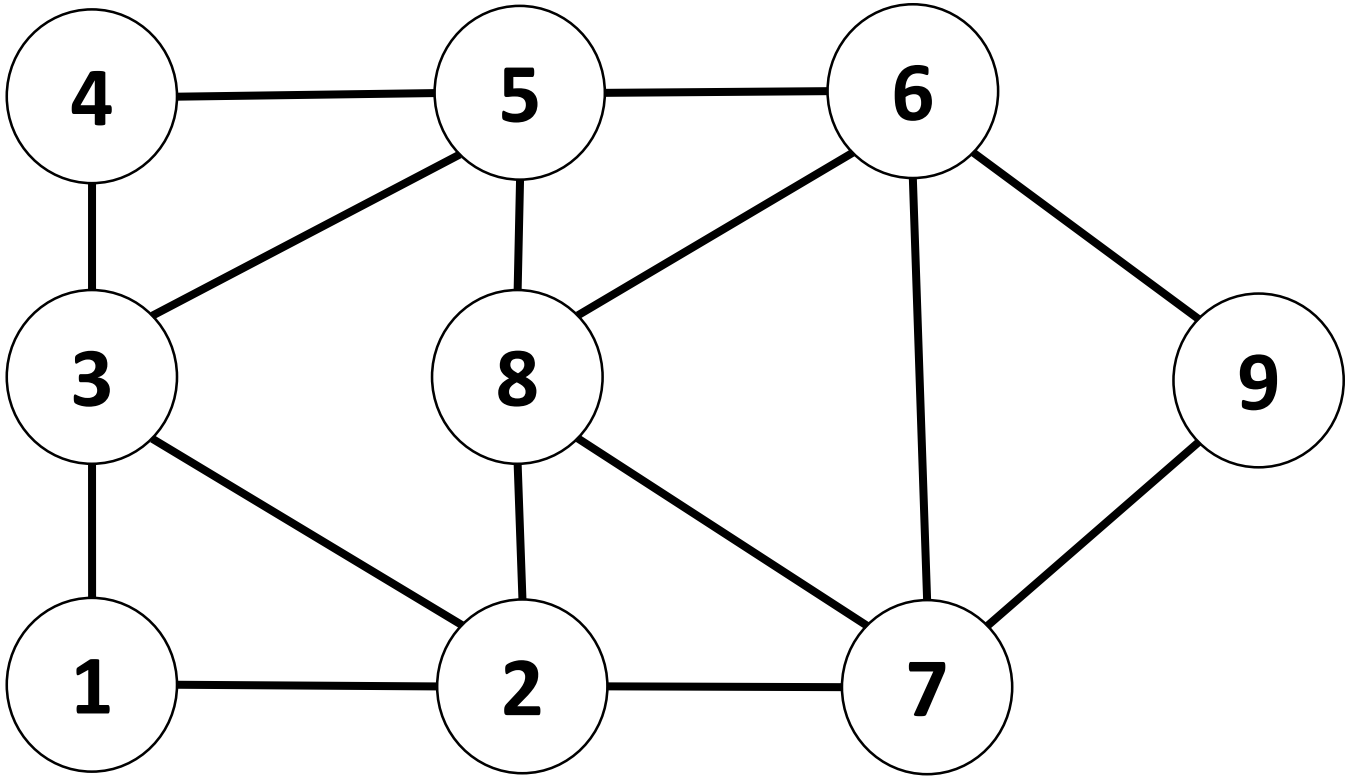
```
Procedure Ham(k);
  Begin
    For u∈ZAP[X[k-1]] do
      If  (k=n+1)                {если все вершины уже вставлены в путь}
            And (u=v0)    {и если последняя в-на в пути совпадает с 1-ой}
                                {т.е. это цикл}
          Then Write(X[1],…X[k-1],v0)    {печать пути и возможное окончание
                                          алгоритма}
          Else
            If DOP[u]                          {если вершину можно вставить в путь}
              Then
                Begin
                  X[k] ← u;  DOP[u] ← False;
                  If k<= n Then
                    Ham(k+1);    {очередное удлинение пути привело в тупик –}
                  Else
                    DOP[u]  ← True;
                                    {необходимо вернуться для следующей попытки}
                End;
  End; {Ham}
```

- Begin
- For v∈V do DOP[v] ← True; //инициализация}
- X[1] ← v0; //v0 – произвольная фиксированная вершина графа
- DOP[v0] ← False;
- Ham(2); //начало построения пути длиной 2
- End.

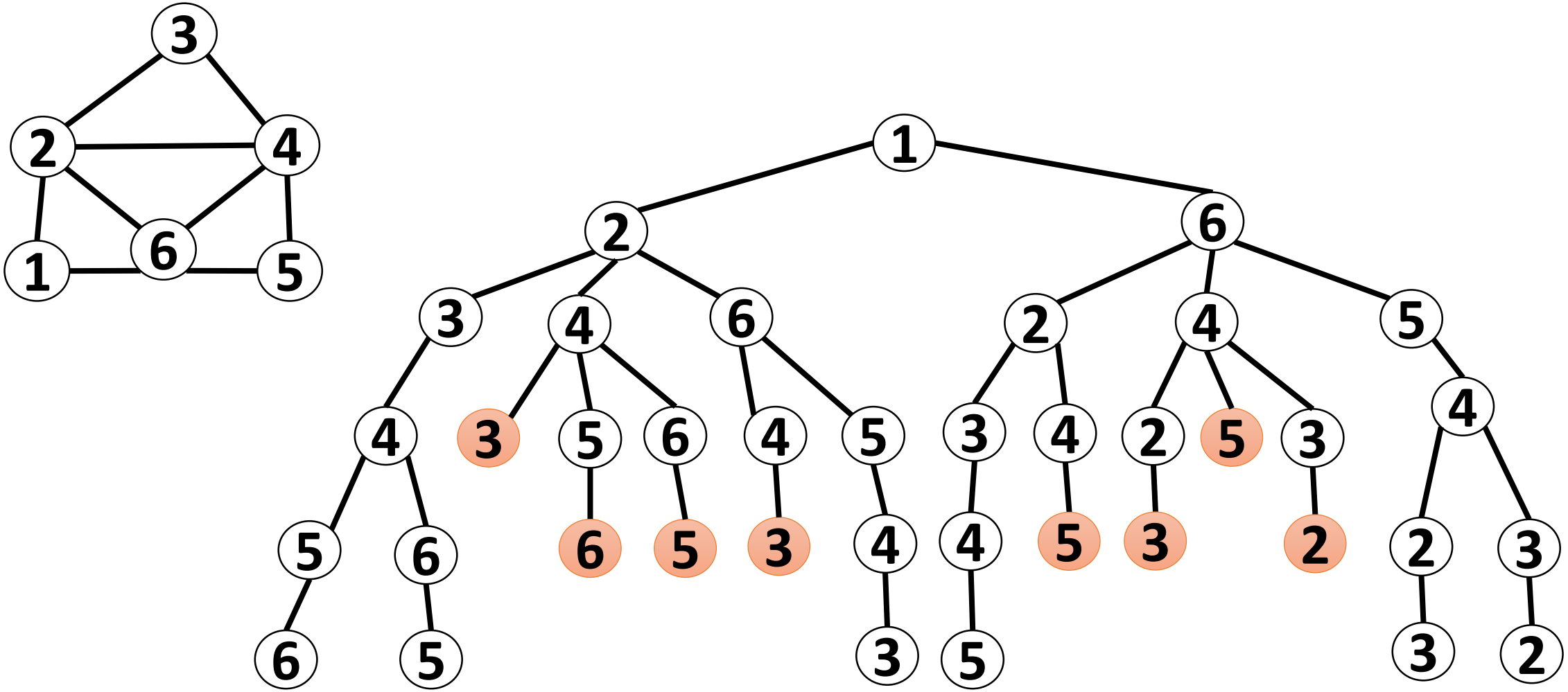
- Этот рекурсивный алгоритм имеет сложность не более чем **exp(n)**
- Работу этого алгоритма можно проиллюстрировать процессом поиска в некотором дереве, состоящем из всех возможных последовательностей <x1,…,xk> для графа.
- **(**Литтл, Мерти, Суини, Кэрел в 1963 г.)

Алгоритм поиска гамильтонова пути

PATH	Смежные вершины
1	1 : ~~2~~ 3
2	2 : 1 ~~3~~ 7 8
3	3 : 1 2 ~~4~~ 5
4	4 : 3 ~~5~~
5	5 : 3 4 5 ~~6~~ 8
6	6 : 5 ~~7~~ ~~8~~ 9
~~7~~	~~7 : 2 6 8 9~~
~~8~~	~~8 : 2 5 6 7~~
~~9~~	~~9 : 6 7~~
8	8 : 2 5 6 ~~7~~
7	7 : 2 6 8 9
9	9 : 6 7



Древовидное представление



Алгоритм Дейкстры

- Поиск в орграфе кратчайших путей от одной вершины до всех остальных при неотрицательных ребрах.
- Каждой вершине из множества вершин **V** сопоставим метку — минимальное известное расстояние от этой вершины до **a**. Алгоритм на каждом шаге «посещает» одну вершину и пытается уменьшать метки. Работа завершается, когда посетили все вершины.
- Инициализация. Метка вершины **a** полагается = 0, метки остальных — бесконечности (т.е. расстояния от **a** пока неизвестны, и они помечаются как непосещённые)
- В цикле по всем вершинам идет поиск «минимальной » вершины и релаксация из нее всех достижимых из нее вершин с сохранением результата.

инициализация

- Procedure Initz (G,S)

  For v $\in$ V [G] do — для всех вершин принадлежащих графу

- D[v] $\leftarrow \infty$ — устанавливаем верхнюю оценку веса кратчайшего пути в v
- Pred [v] $\leftarrow$ Nil — Pred – массив с вершинами, из которых был переход в v
- D[S] $\leftarrow$ 0 — S – начальная вершина

релаксация

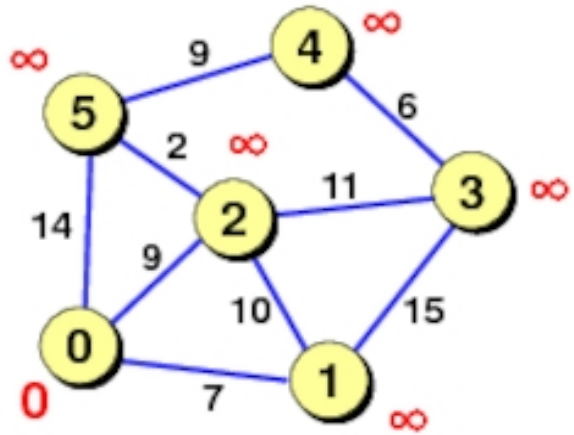
- Procedure Relax (u,v,w) - релаксация
- If D [v] > D [u] + w (u,v) если верхн. оценка больше чем сумма с ребром
- Then D [v] $\leftarrow$ D [u] + w (u,v) то она становится верхней оценкой
- Pred[v] $\leftarrow$ u сохраним предыдущую

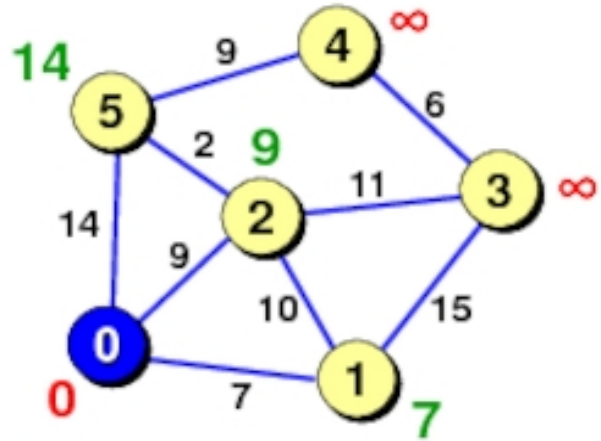
алгортм Дейкстра

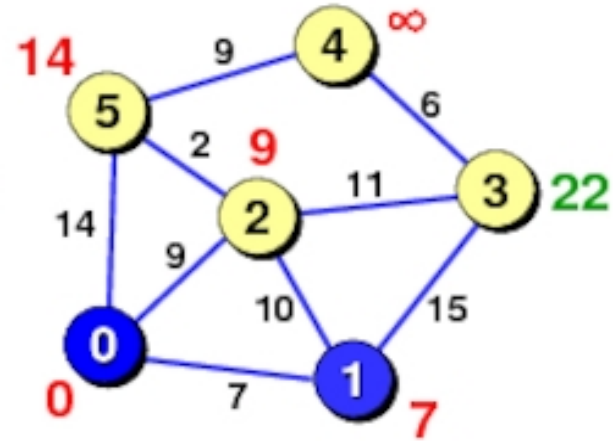
- **Procedure Dijkstra (G,w,S)** - **($O(v^2)$) ($O(n^2+m)$)**
- {в орграфе мин-ный путь из S ко всем (ребра >0)}
- **Initz (G,S)** инициализируем граф
- **S ← 0** S – пустое множество
- **Q ← V[G]** вершину V ставим в очередь
- **While Q <> 0 do** пока очередь не пуста
- **u ← r** в u ставим произвольную вершину r ,
- такую, что для нее D[r] = min {D[p]}
- **S ← S ∪ { u }** вершина u изымается из очереди и добавляется в S
- **For всех v смежных с u do**
- **Relax(u,v,w)** релаксируем ребра
- (если множество S представить 2-ичн. деревом, то: **O(E log v)**

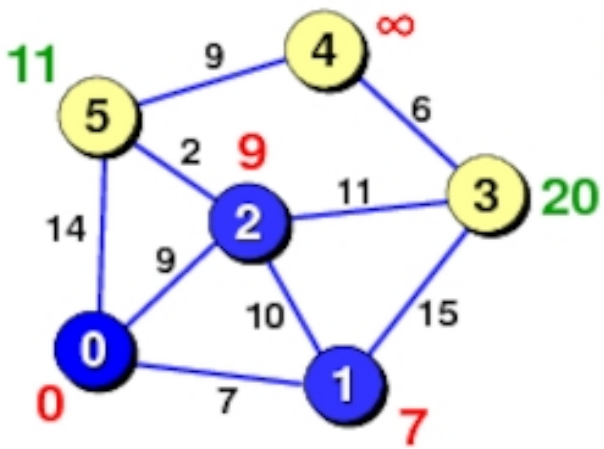
- Если граф представить списком ZAP[u], то цикл можно записать так:
- For v ∈ Zap [u] do для всех вершин принадлежащих графу
- If D [v] > D [u] + w (u,v) если верхн. оценка больше чем сумма ребром,
- Then D [v] = D [u] + w (u,v) то она становится этой суммой

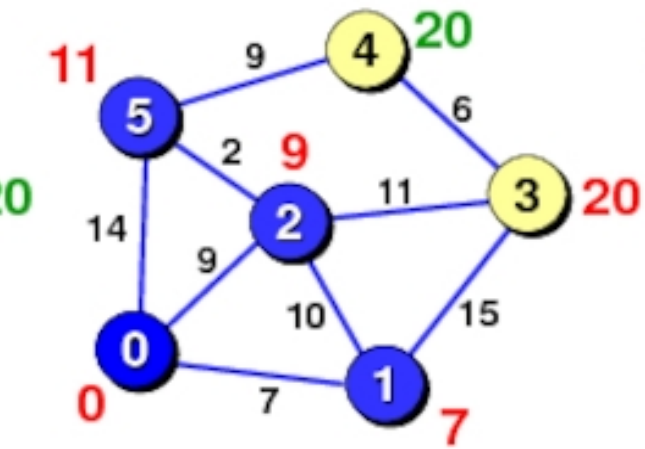
Алгоритм Дейкстры

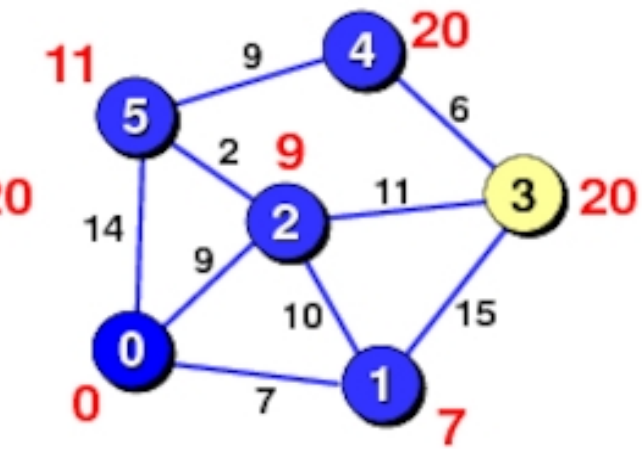












Алгоритм Беллмана–Форда

- Алгоритм поиска кратчайшего пути во взвешенном графе.
- За время O(|V| × |E|) алгоритм находит кратчайшие пути от **одной** вершины графа до всех остальных. В отличие от алг-ма Дейкстры, алг-тм Беллмана–Форда допускает рёбра с **отрицательным** весом.
- Ричарда Беллмана (Richard Bellman) (1958 г.) - статья, посвящённую конкретно задаче нахождения кратчайшего пути. Лестер Форд (Lester Ford) (1956 г.) - фактически изобрёл этот алгоритм при изучении другой математ. задачи.

- Алгоритм маршрутизации RIP (алгоритм Беллмана–Форда) был впервые разработан в 1969 году, как основной для сети ARPANET.

- Формулировка задачи

- Дан ор или неорграф **G** со взвешенными рёбрами. Длина пути - сумма весов рёбер, входящих в этот путь. Найти кратчайшие пути от выделенной вершины **s** до всех вершин графа (их может не существовать, т.к., в графе, содержащий цикл с отрицательным суммарным весом, существует сколь угодно короткий путь от одной вершины этого цикла до другой) Цикл, сумма весов рёбер которого отрицательна, называется отрицательным циклом.

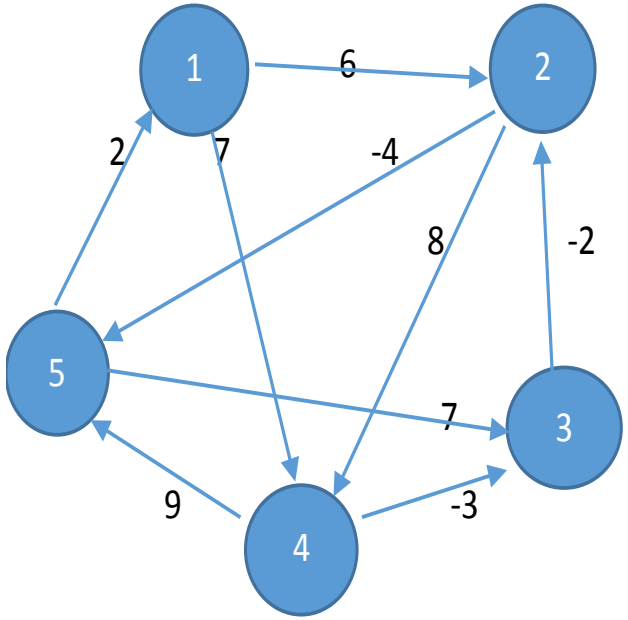
- А. Б.–Ф. позволяет очень просто определить, существует ли в графе **G** отрицательный цикл, достижимый из вершины **s**. Достаточно произвести внешнюю итерацию цикла не $|V| - 1$, а ровно $|V|$ раз.
- Если при исполнении последней итерации длина кратчайшего пути до к-л вершины строго уменьшилась, то в графе есть отрицат. цикл, достижимый из **s**. На основе этого можно предложить след. оптимизацию: отслеживать изменения в графе и, как только они закончатся, дальнейшие итерации будут бессмысленны. Значит можно сделать выход из цикла.

Решение задачи на графе без отрицательных циклов

- Для нахождения кратчайших путей от одной вершины до всех остальных, восп-ся методом динамического программирования. На входе - матрица **Aij**, содержащая длины дуг (ребер) между смежными вершинами. На выходе – вектор с кратчайшими расстояниями от выбранной вершины до всех остальных и указание об отсутствии отрицательных циклов.

A						k	D[1]	D[2]	D[3]	D[4]	D[5]
	1	2	3	4	5	от 1	**0**	**6**	**∞**	**7**	**∞**
1	**∞**	**6**	**∞**	**7**	**∞**	от 2-й и 4-й	1) **0**	**6**	**4**	**7**	**2**
2	**∞**	**∞**	**∞**	**8**	**-4**	от 3-й	2) **0**	**2**	**4**	**7**	**-2**
3	**∞**	**-2**	**∞**	**∞**	**∞**	от 5-й	3) **0**	**2**	**4**	**7**	**-2**
4	**∞**	**∞**	**-3**	**∞**	**9**						
5	2	**∞**	**7**	**∞**	**∞**						

- **Ребра перебираются по порядку**
- 1) (2 - 4) (2 - 5) (4 - 3) (4 - 5)
- 2) (3 - 2) (2 - 4) (2 – 5)
- 3) (5 - 1) (5 - 3)



- Похоже на поиск в ширину от всех смежных вершин
- Для заданного взвешенного ориентированного графа **G = (V, E)** с истоком **S** и весовой функцией
- **w : E —» R** алгоритм Беллмана-Форда возвращает лог. значение, указывающее содержится ли в графе цикл с отрицатательным весом, достижимый из истока. Если такой цикл существует, в алгоритме указывается, что решения не существует. Если же таких циклов нет, алгоритм выдает кратчайшие пути и их вес.

- **Procedure Bellman_Ford (G,w,S)** // алгоритм Беллмана –Форда **O(VE)** {в орграфе мин-ный путь из одной вершины ко всем. (ребра любые, без циклов <0)}
- **Initz (G,S)** //инициализируем граф
- **For i ←1 to |V[G]| - 1** //для всех вершин, кроме S
- **Do For каждого ребра (u,v) ∈ E|G|**
- **Do Relax(u,v,w)** //релаксируем каждое ребро
- **For каждого ребра (u,v) ∈ E|G|** //проверяем нет ли отрицат. //цикла
- **Do If D[v] > D[u]+ w(u,v) Then Return False**
  . // отрицательный цикл
- **Return True**

- В целом, а. Дейкстры, по сравнению с а. Беллмана-Форда, обеспечивает более реальную оценку ситуации в сети, более быструю реакцию на важные изменения в сети (такие, как включение новой линии связи) и уменьшает зацикливание пакетов; однако а. Дейкстры сложнее в реализации и требует в несколько раз больше памяти.

- Эффективность - O(V*E)

- Константа в оценке сложности может быть уменьшена за счёт :
- 1) если на очередной итерации не произошло ни одной успешной релаксации, то алгоритм завершает работу;
- 2) на очередной итерации рассматриваются не все рёбра, а только выходящие из вершин, для которых на прошлой итерации была выполнена успешная релаксация (на первой итерации – только рёбра, выходящие из источника).

Задача коммивояжера

Оптимизационная постановка задачи относится к классу NP-полных задач, впрочем, как и большинство её частных случаев. Версия «decision problem» (т.е. такая, в которой ставится вопрос, существует ли маршрут не длиннее, чем значение k). Задача коммивояжёра относится к числу трансвычислительных: уже при относительно небольшом числе городов (66 и более) она не может быть решена методом перебора вариантов никакими теоретически мыслимыми компьютерами за время, меньшее нескольких миллиардов лет.

Алгоритм Флойда - Уоршелла

- Алгоритм Флойда - Уоршелла — динамический алгоритм для нахождения кратчайших расстояний между всеми вершинами взвешенного ориентированного графа. Разработан в 1962 году Робертом Флойдом (Robert W Floyd,) и Стивеном Уоршеллом (Stephen Warshall).
- Вершины графа последовательно пронумерованы от 1 до **n.** Алгоритм Флойда использует матрицу **A** размера **n * n**, в кот. вычисляются длины кратчайших путей.
- Вначале **A[i, j] = D[i, j]** для всех **i != j**. Если дуга **i -» j** отсутствует, то **D[i, j] = ∞**. Каждый диагональный элемент матрицы A равен 0.

- Над матрицей **A** выполняется **n** итераций. После **k**-й итерации
- **A[i, j]** содержит значение наименьшей длины путей из вершины i в вершину **y**, которые не проходят через вершины с номером, большим **k**. Другими словами, между концевыми вершинами пути **i** и **y** могут находиться только вершины, номера которых меньше или равны **k**. На **k-й** итерации для вычисления матрицы **A** применяется следующая формула:
- $A_k[i,j]=\min(A_{k-1}[i,j], A_{k-1}[i,k]+A_{k-1}[k,j])$
- Время выполнения этой программы, очевидно, имеет порядок $O(V^3)$
- Доказательство "правильности" работы этого алгоритма также очевидно и выполняется с помощью математической индукции по **k**, показывая, что на **k**-й итерации вершина **k** включается в путь только тогда, когда новый путь короче старого.

- **Procedure Floyd_Warshall**
- // - алгоритм Флойда – Уоршелла **$O(n^3)$**
- // минимальный путь между всеми вершинами
- **N** ← размер матрицы A
- **D(0) ← A** // начальная матрица D содержит A
- **For K ←1 to N**
- **Do For I ← 1 to N**
- **Do For J ← 1 to N**
- Do d**(k) i,j** ←min (d **(k-1) I,j** , d **(k-1) i,k** + d **(k-1) k,j**)

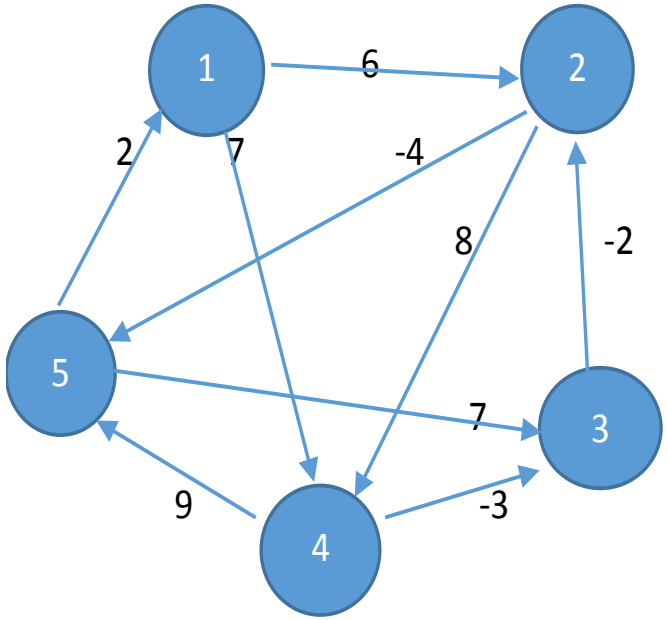
A

	1	2	3	4	5
1	0	6	∞	7	∞
2	∞	0	∞	8	∞
3	∞	-2	0	∞	∞
4	∞	∞	-3	0	9
5	2	∞	7	∞	0

k = 1

	1	2	3	4	5
1	0	2	4	7	2
2	2	0	4	8	-4
3	∞	-2	0	∞	∞
4	∞	∞	-3	0	9
5	2	∞	7	∞	0

	1	2	3	4	5
1	0	2(4,3)	4(4)	7	42(4,3,2)
2	2(5)	0	3(5)	8	-4
3	-4(2,5)	-2	0	3(2,5,1)	-6
4	-7(3,2,5)	-5	-3	0	-9
5	2	4(2,4,3)	6(1,4)	9(1)	0



- Алгоритм Флойда — Уоршелла является эффективным для расчёта всех кратчайших путей в плотных графах, когда имеет место большое количество пар рёбер между парами вершин.
- В случае разреженных графов с рёбрами неотрицательного веса лучшим выбором считается использование алгоритма Дейкстры для каждого возможного узла. При таком выборе сложность составляет O (VE Log V), что лучше, O(V^3) алгоритма Флойда — Уоршелла. Если граф разрежен, у него имеются рёбра с отрицательным весом и отсутствуют циклы с отрицательным суммарным весом, то используется алгоритм . Беллмана-Форда, который имеет ту же сложность, что и вариант с алгоритмом Дейкстры.

- Также являются известными алгоритмы с применением алгоритмов быстрого перемножения матриц, которые ускоряют вычисления в плотных графах, но они обычно имеют дополнительные ограничения (например, представление весов рёбер в виде малых целых чисел). Вместе с тем, из-за большого константного фактора времени выполнения преимущество при вычислениях над алгоритмом Флойда — Уоршелла проявляется только на больших графах.

Построение минимальных остовых деревьев

Жадные алгоритмы

Алгоритмы Прима и Крускала – **жадные**: на каждом их шаге делается локально оптимальный (жадный) выбор, который никогда не отменяется на последующих шагах.

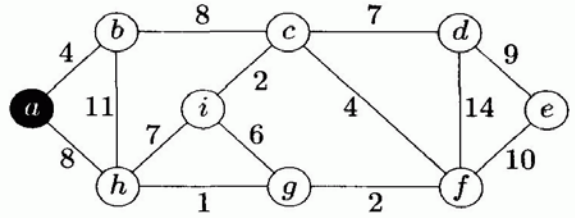
Жадный алгоритм можно использовать, если для задачи выполняются 2 условия:

- оптимальное решение задачи содержит в себе оптимальные решения подзадач (свойство оптимальности подзадач);
- последовательность локально оптимальных выборов дает глобально оптимальное решение (т.е. жадный выбор на каждом шаге не закрывает путь к оптимальному решению).

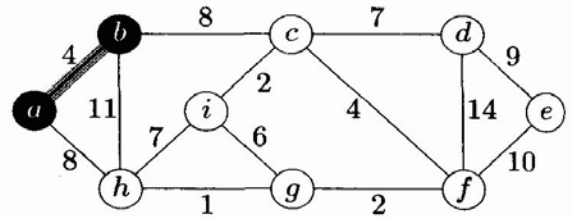
Алгоритм Прима

- А. Пр. — **алгоритм построения минимального остового дерева взвешенного связного неор. графа.** Алгоритм впервые был открыт в 1930 году чешским математиком Войцехом Ярником, позже переоткрыт Робертом Примом в 1957 году, и, независимо от них, Э. Дейкстрой в 1959 году.
- Построение начинается с дерева, включающего в себя одну (произвольную) вершину. В течение работы алгоритма дерево разрастается, пока не охватит все вершины исх. графа. На каждом шаге алгоритма к текущему дереву присоединяется самое лёгкое из рёбер, соединяющих вершину из построенного дерева, и вершину не из дерева.
- На входе - связный неориентированный граф G(V,E)
- На выходе - мн-во T рёбер минимального остового дерева

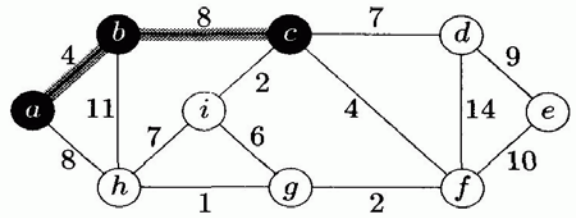
(а)


a
b
c
d
e
f
g
h
i
4
8
8
11
7
2
6
4
7
14
9
10
1
2

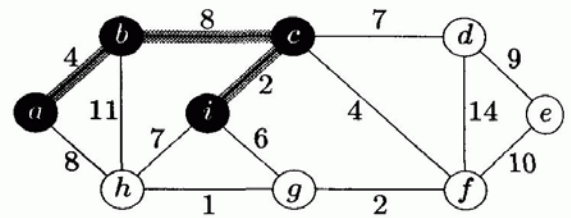
(б)


a
b
c
d
e
f
g
h
i
4
8
8
11
7
2
6
4
7
14
9
10
1
2

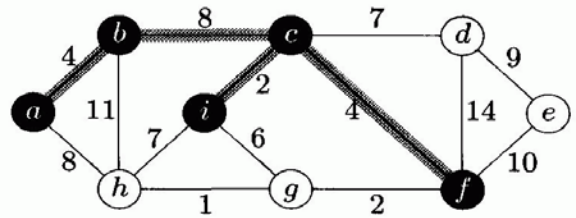
(в)


a
b
c
d
e
f
g
h
i
4
8
8
11
7
2
6
4
7
14
9
10
1
2

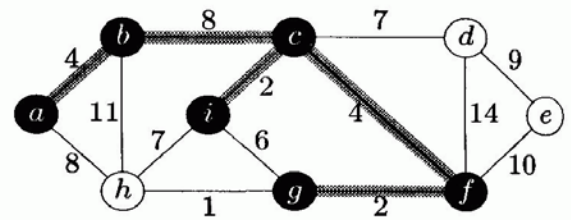
(г)


a
b
c
d
e
f
g
h
i
4
8
8
11
7
2
6
4
7
14
9
10
1
2

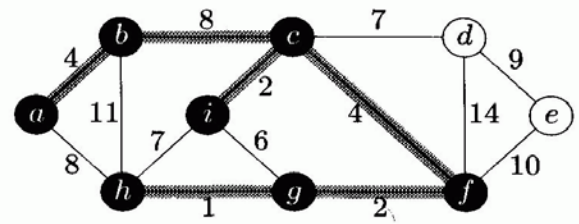
(д)


a
b
c
d
e
f
g
h
i
4
8
8
11
7
2
6
4
7
14
9
10
1
2

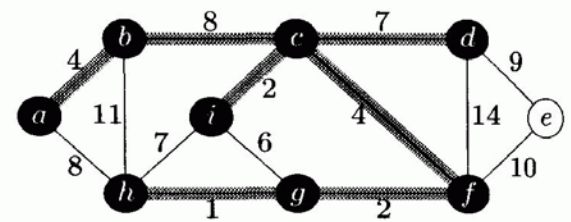
(е)


a
b
c
d
e
f
g
h
i
4
8
8
11
7
2
6
4
7
14
9
10
1
2

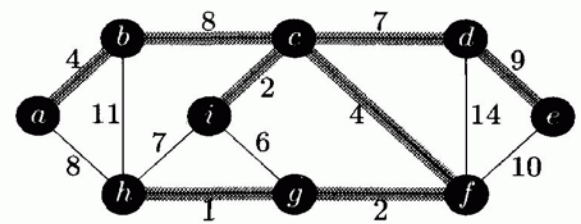
(ж)


a
b
c
d
e
f
g
h
i
4
8
8
11
7
2
6
4
7
14
9
10
1
2

(з)


a
b
c
d
e
f
g
h
i
4
8
8
11
7
2
6
4
7
14
9
10
1
2

(и)


a
b
c
d
e
f
g
h
i
4
8
8
11
7
2
6
4
7
14
9
10
1
2

Пример. Алгоритм Прима.

Происходит наращивание дерева

- Обозначения
- **d[i]** — расстояние (приоритет) от **i**-й вершины до построенного дерева
- **p[i]** — предок **i-**й вершины, то есть такая вершина **u**, что **(i,u)** легчайшее из всех рёбер соединяющее **i** с вершиной из построенного дерева.
- **w(i,j)** — вес ребра **(i,j)**
- **Q** — приоритетная очередь вершин графа, где ключ — **d[i]**
- **T** — множество ребер минимального остового дерева

- **Procedure Prim (G,w,r)**
- **Q ← V|G|** //инициализация очереди
- **For каждой в-ны u € Q**
- **do D[u] ← ∞**
- **D[r] ← 0** //корень → r
- **Pred[r] ← Nil**
- **While Q ≠ 0** //Пока Q не пуста
- **Do u ← ближайшая в-на (из очереди)**
- **For каждой в-ны v смежной с u**
- **Do if v € Q and w (u,v) < D[v]** //Если ребро от этой в-ны min
- **Then Pred[v] ← u** //И ее приоритет = этому ребру
- **D[v] ← w(u,v)**

- **Оценка**

- Асимптотика алгоритма зависит от способа хранения графа и способа хранения вершин, не входящих в дерево.

Способ представления графа и приоритетной очереди	**Асимптотика**
Массив **d**, списки смежности (матрица смежности)	$O(V^2)$
Бинарная пирамида, списки смежности	$O((V + E)\log V) = O(E \log V)$
Фибоначчиева пирамида, списки смежности	$O(E + V \log V)$

Алгоритм Крускала (или Краскала)

- Алгоритм построения минимального остового дерева взвешенного связного неор. графа. Алгоритм впервые описан Джозефом Крускалом (Kruskal)в 1956 году.
- Вначале текущее множество рёбер устанавливается пустым. Затем, пока это возможно, проводится след. операция: из всех рёбер, добавление которых к уже имеющемуся множеству не вызовет появление в нём цикла, выбирается ребро минимального веса и добавляется к уже имеющемуся множеству. Когда таких рёбер больше нет, алгоритм завершён. Подграф данного графа, содержащий все его вершины и найденное множество рёбер, является его остовым деревом минимального веса

- До начала работы алгоритма необходимо отсортировать рёбра по весу, это требует O(E*log(E)) времени. После чего компоненты связности удобно хранить в виде системы непересекающихся множеств. Эффективность можно принять за O(E * log(E)).
- *Идея алгоритма.* Искомые ребра соединяют вершины. Поэтому возможны две стратегии построения. Можно идти от вершин и для каждой из них искать мин-ное ребро (как это сделано в а. Прима), а м. для каждого ребра выяснять можно ли его включить в строящееся дерево.

- Алгоритм Крускала:

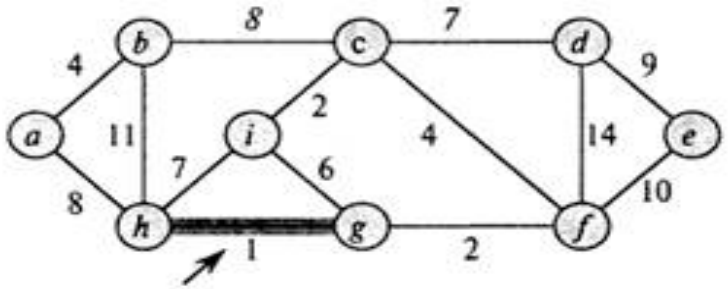
1. Ребра графа пронумеровываем в порядке возрастания весов
2. Затем для каждого ребра, начиная с первого, проверяем соединяет или нет оно две несвязные вершины, если да, то его можно включить в остовое дерево.

Если мы имеем V вершин, то работа алгоритма начинается с V несвязных компонент графа (пока из графа все ребра исключаем). Для того, чтобы их связать необходимо найти V-1 ребро.

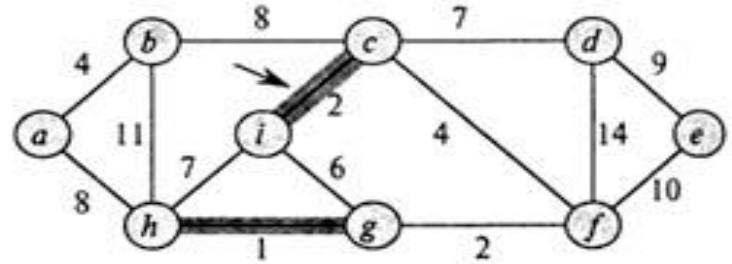
- Др. словами, алгоритм организует процесс роста компонент связности в процессе кот. он объединяются друг с другом до тех пор пока не останется одна являющаяся конечным результатом. Т.е., алгоритм Крускала строит лес ребер.

- Создаем список ребер по возрастанию.
- Создаем множество компонент связности каждая из кот. содержит ровно одну вершину
- Пока компонент связности больше чем одна
- Взять ребро из начала списка ребер
- Если ребро соединяет две разных компоненты связности то
- Компоненты связности объединить в одну

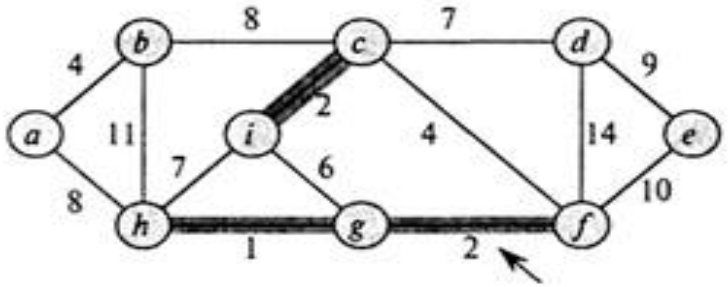
а)


a
b
c
d
e
f
g
h
i
4
8
8
11
7
2
6
4
7
14
9
10
1
2

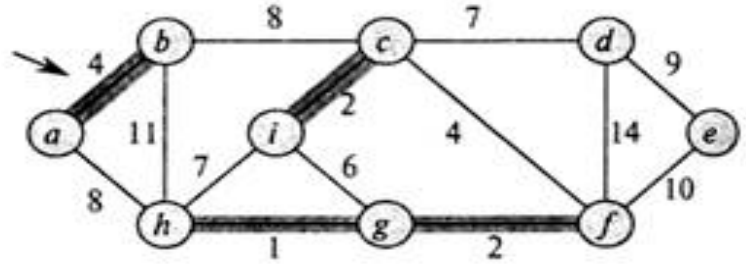
б)


a
b
c
d
e
f
g
h
i
4
8
8
11
7
2
6
4
7
14
9
10
1
2

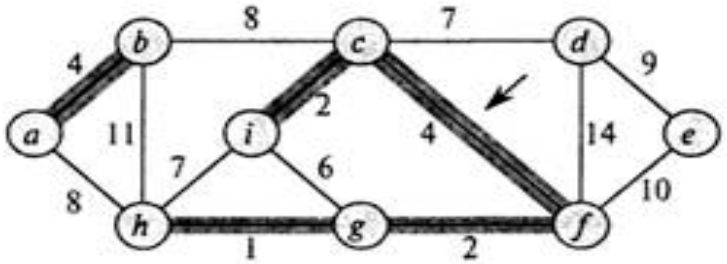
в)


a
b
c
d
e
f
g
h
i
4
8
8
11
7
2
6
4
7
14
9
10
1
2

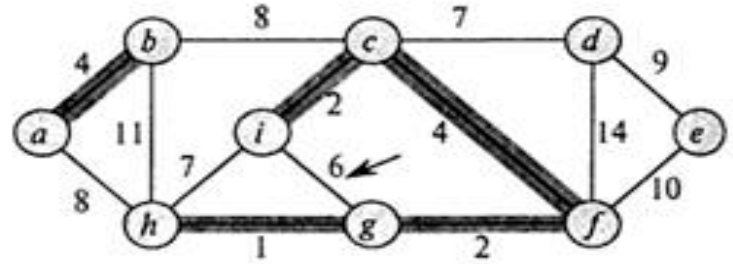
г)


a
b
c
d
e
f
g
h
i
4
8
8
11
7
2
6
4
7
14
9
10
1
2

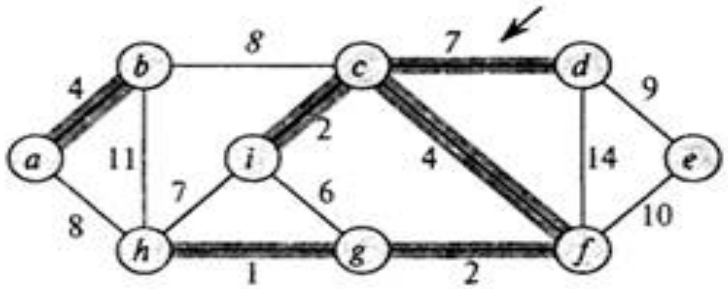
д)


a
b
c
d
e
f
g
h
i
4
8
8
11
7
2
6
4
7
14
9
10
1
2

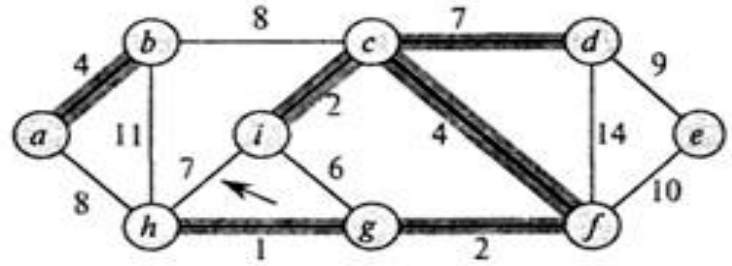
е)


a
b
c
d
e
f
g
h
i
4
8
8
11
7
2
6
4
7
14
9
10
1
2

ж)


a
b
c
d
e
f
g
h
i
4
8
8
11
7
2
6
4
7
14
9
10
1
2

з)


a
b
c
d
e
f
g
h
i
4
8
8
11
7
2
6
4
7
14
9
10
1
2

Алгоритм Краскала:

- **Procedure Kruskal (G,w)**
- **A← 0** //множесво - пусто
- **For каждой в-ны v ∈ E|G|**
- **Do создать |V| деревьев из одной в-ны**
- **Упорядочить ребра E по весам**
- **for (u,v) ∈ E (в порядке возрастания веса)**
- **Do if концы ребра не лежат в одном дереве**
- **Then A ← A (u,v) ∈ E|G| добавляем ребро, из множества ребер A, объединяем деревья**
- **return A**
- **Else нельзя добавить ребро, не получая цикла.**

Использование:

- Поиск суммарных дорог между городами, чтобы длина была минимальной.
- В этой задаче вершины графа (**V**i) – представляют собой города, а ребра E(**V**i, **V**j) – дороги между городами. Нам известны расстояния между парами городов (**V**i, **V**j) – это вес ребер w(E(**V**i, **V**j)). Нужно найти такое дерево в графе (без циклов: зачем строить лишние дороги), чтобы сумма ребер была минимальной и при этом все вершины были достижимы.