

Графы в Wolfram Mathematica

1. Неориентированные графы: задание

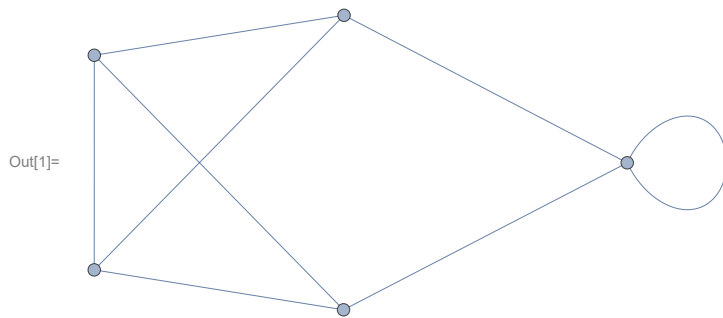
Задание графа: граф задается перечислением связей вершин (ниже пример графа из 5 вершин)

↔

UndirectedEdge []

↪ неориентированное ребро

In[1]:= **Graph [{1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 5, 3 ↔ 5, 1 ↔ 5, 2 ↔ 4, 1 ↔ 1}]**
↪ граф



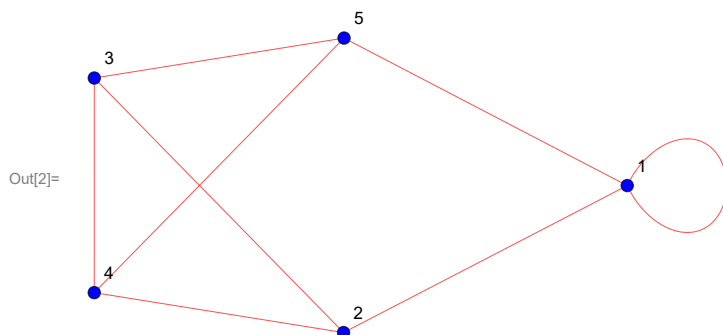
Значок неориентированного ребра ↔ ставится комбинацией "Esc", "u", "e", "Esc" или можно использовать функцию UndirectedEdge : UndirectedEdge[1, 2] = 1 ↔ 2

Настройка отображения: VertexStyle - цвет вершин, EdgeStyle - цвет ребер, VertexLabels - подписи вершин (в данном случае автоматически)

In[2]:= **Graph [{1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 5, 3 ↔ 5, 1 ↔ 5, 2 ↔ 4, 1 ↔ 1} ,**
↪ граф

VertexStyle → Blue, EdgeStyle → Red, VertexLabels → Automatic]

↪ стиль вершины ↪ синий ↪ стиль ребра ↪ красн... ↪ метки для вершин ↪ автоматически

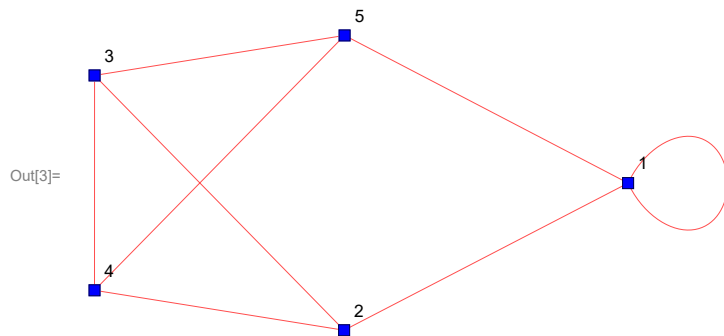


Настройка отображения : VertexShapeFunction задает форму вершин

```

In[3]:= Graph[{1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 5, 3 ↔ 5, 1 ↔ 5, 2 ↔ 4, 1 ↔ 1}, VertexStyle → Blue,
  EdgeStyle → Red, VertexLabels → Automatic, VertexShapeFunction → "Square"]

```

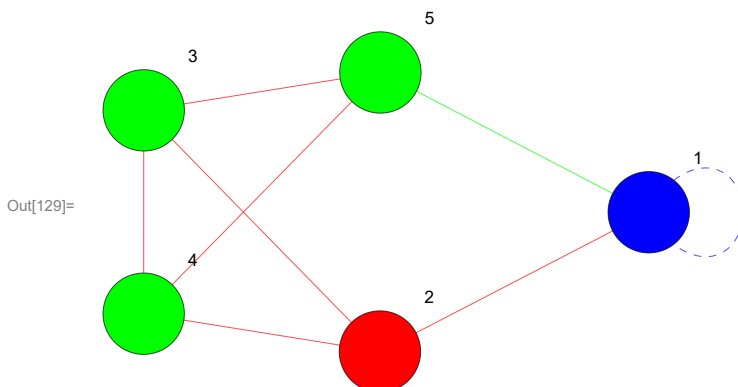


Настройка отображения : Пример закрашки отдельных вершин 1-ая синей, 2 красной, остальные -зеленые; закрашка ребер разными цветами и стилями: ребро 1-5 зеленое, ребро 1-1 синее пунктирное, остальные - красные; VertexSize - размер вершин

```

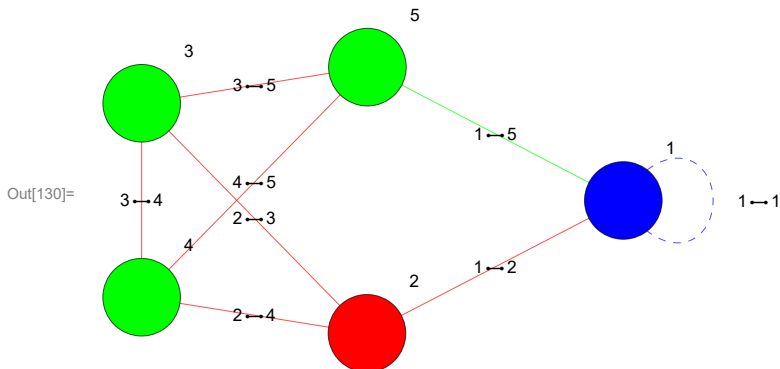
In[129]:= g = Graph[{1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 5, 3 ↔ 5, 1 ↔ 5, 2 ↔ 4, 1 ↔ 1},
  VertexStyle → {1 → Blue, 2 → Red, Green}, VertexLabels → Automatic,
  VertexSize → Large, EdgeStyle → {1 ↔ 5 → Green, 1 ↔ 1 → {Blue, Dashed}, Red}]

```



Настройка отображения : EdgeLabels включает подписи ребер (в данном случае автоматически)

```
In[130]:= Graph[g, VertexLabels → Automatic, EdgeLabels → "Name"]
[граф [метки для вершин [автоматиче... [пометки для рёбер
```

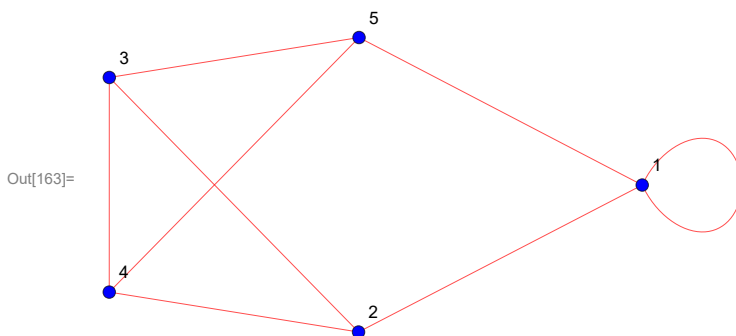


2. Матрицы графов


Матрица смежности графа g (Матрица смежности графа g с конечным числом вершин n (пронумерованных числами от 1 до n) — это квадратная матрица A размера n , в которой значение элемента a_{ij} равно числу рёбер из i -й вершины графа в j -ю вершину)

```
In[163]:= g1 = Graph[{1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 5, 3 ↔ 5, 1 ↔ 5, 2 ↔ 4, 1 ↔ 1},
[граф
```

```
VertexStyle → Blue, EdgeStyle → Red, VertexLabels → Automatic]
[стиль вершины [синий [стиль ребра [кр... [метки для вершин [автоматический
```



```
In[164]:= A = AdjacencyMatrix[g1]
[матрица смежности
```

Out[164]= SparseArray[ Specified elements: 15
Dimensions: {5, 5}]

```
In[165]:= A // MatrixForm
[матричная форма
```


Out[165]/MatrixForm=

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

SparseArray - хранение сильно разреженной матрицы в компактном виде ("зачем хранить кучу нулей?")

Пример работы с SparseArray : зададим матрицу s в которой элементы $s_{1,2} = 1$, $s_{5,7} = 2$, $s_{3,14} = 3$:

```
In[9]:= s = SparseArray[{{1, 2} → 1, {5, 7} → 2, {3, 14} → 3}]
```

```
Out[9]= SparseArray[ Specified elements: 3  
Dimensions: {5, 14}]
```

Посмотрим матрицу в нормальном виде :

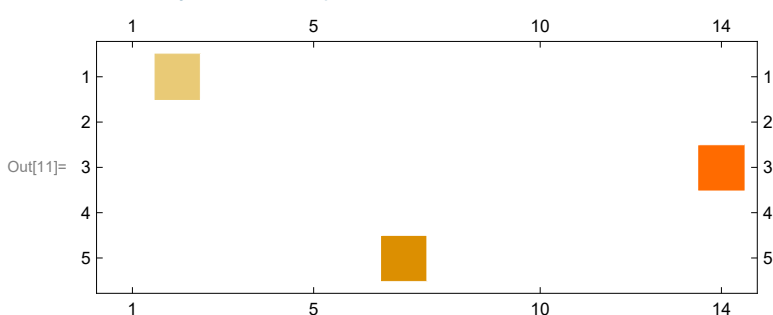
```
In[10]:= s // MatrixForm
```

```
Out[10]//MatrixForm=
```

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Есть очень удобное представление в графическом виде, которое позволяет понять структуру матрицы (элементы - цветные квадратики (желтый - самый маленький элемент, оранжевый - самый большой), пустота - нули)

```
In[11]:= s // MatrixPlot
```



Почему это важно? Разреженные матрицы встречаются очень часто. Такое хранение позволяет экономить память и использовать быстрые алгоритмы для операций над ними. Отвлечемся от графов и рассмотрим пример : создадим две произвольные вещественные трехдиагональные матрицы s1 и s2 размера 1000 на 1000

```
In[150]:= s1 = SparseArray[{i_, j_} /; Abs[i - j] ≤ 1 → RandomReal[], {1000, 1000}];
```

Разберем эту синтаксическую конструкцию, которая позволяет нам не использовать классические циклы программирования

1) /; " (условие) " – легко понять из примера ниже

```
In[135]:= f[x_] := Sin[x] /; x > 0
```

```
In[136]:= f[2]
```

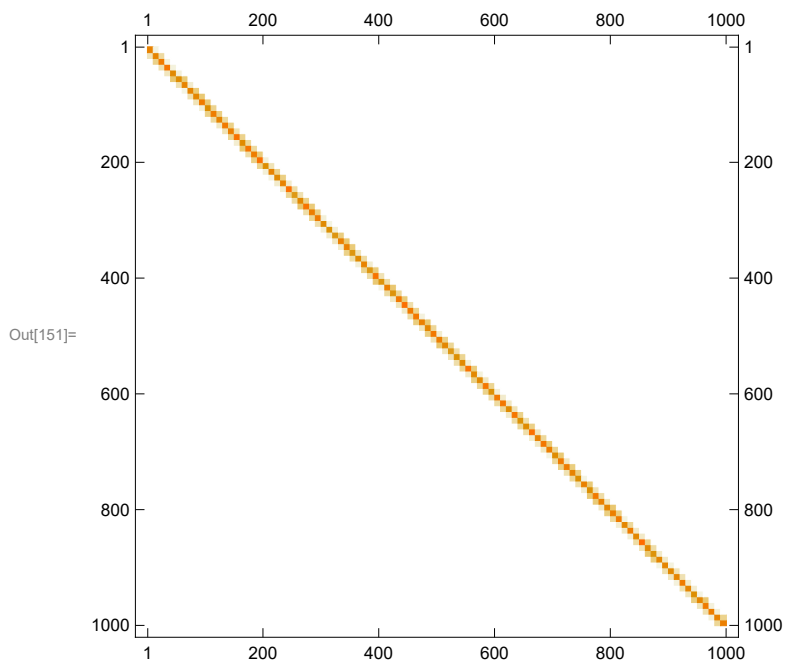
```
Out[136]= Sin[2]
```


[illegible]

[illegible]

```
In[151]:= s1 // MatrixPlot
```

Визуализация матрицы



Матрицу из разреженного вида к обычному списку можно привести функцией `Normal`

```
In[21]:= Normal[s1]
```

нормальное выражение

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

Создадим вторую трехдиагональную матрицу 100 на 100

```
In[149]:= s2 = SparseArray[{i_, j_] /; Abs[i - j] ≤ 1 -> RandomReal[], {1000, 1000}];
```

разреженный массив

абсолютное знач...

случайное действительное число

Посмотрим сколько времени занимает вычисления их произведения в виде `SparseArray` :

```
In[154]:= qt1 = AbsoluteTiming[s1.s2]
```

длительность по настенным

```
Out[154]:= {0.000209, SparseArray[
```

Specified elements: 4994
Dimensions: {1000, 1000}

А теперь посмотрим сколько времени займет эта операция если перевести их в обычные списки :

```
In[158]:= ss1 = Normal[s1];
```

нормальное вы

```
In[159]:= ss2 = Normal[s2];
```

нормальное вы

```
In[160]:= qt2 = AbsoluteTiming[ss1.ss2]
```

длительность по настенным час

```
Out[160]:= {0.0160455, {{0.343648, 0.274581, ... 996 ... , 0., 0.}, ... 998 ... , { ... 1 ... }}}}
```

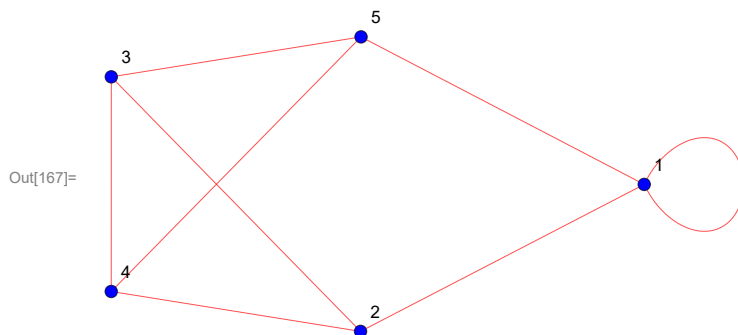
large output show less show more show all set size limit...

```
In[161]:= qt2[[1]] / qt1[[1]]
```

```
Out[161]:= 76.7727
```

Вернемся к графам. Матрицу смежности уже нашли, теперь найдем матрица инцидентности того же графа g1

```
In[167]:= g1
```



Матрица инцидентности — одна из форм представления графа, в которой указываются связи между инцидентными элементами графа (ребро и вершина). Столбцы матрицы соответствуют ребрам, строки—вершинам. Ненулевое значение в ячейке матрицы указывает связь между вершиной и ребром (их инцидентность).

```
In[168]:= IncidenceMatrix[g1] // MatrixForm (*матрица инцидентности*)
```

матрица инцидентий матричная форма

```
Out[168]//MatrixForm=
```

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 2 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{pmatrix}$$

Цифра "2" означает петлю. Важно : порядок ребер определяется при задании графа

```
In[170]:= EdgeList[g1] (*Список ребер, в том же порядке в котором и задавался*)
           ↳список рёбер
```

```
Out[170]= {1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 5, 3 ↔ 5, 1 ↔ 5, 2 ↔ 4, 1 ↔ 1}
```

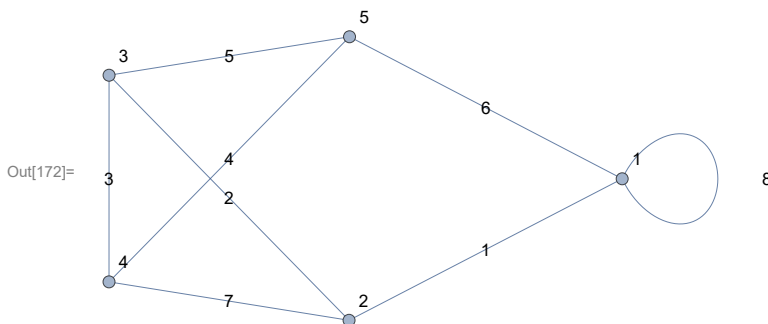
```
In[30]:= (*Ребро 1-1 задавалось последним!*)
```

```
In[171]:= VertexList[g1] (*список вершин*)
           ↳список вершин графа
```

```
Out[171]= {1, 2, 3, 4, 5}
```

Матрица весов. Каждому ребру графа можно сопоставить некоторое число - "вес" ребра (его интерпретация может быть разной, в зависимости от задачи: длина, время, пропускная способность и т.д.) Для этого используется EdgeWeight в примере ниже (ребру 1-2 сопоставляется вес 1, ребру 1-1 --- вес 8, т.е. вес указывается в том же порядке что и ребра)

```
In[172]:= g1 =
  Graph[{1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 5, 3 ↔ 5, 1 ↔ 5, 2 ↔ 4, 1 ↔ 1}, VertexLabels → Automatic,
        ↳граф                                     ↳метки для вершин ↳автоматически
        EdgeLabels → "EdgeWeight", EdgeWeight → {1, 2, 3, 4, 5, 6, 7, 8}]
        ↳пометки для рёбер ↳вес ребра           ↳вес ребра
```



```
In[173]:= WeightedAdjacencyMatrix[g1] // MatrixForm
           ↳матрица весов графа                               ↳матричная форма
           (*Матрица весов = матрица смежности с указанием веса ребра*)
```

```
Out[173]//MatrixForm=
  ⎛ 8  1  0  0  6 ⎞
  ⎜ 1  0  2  7  0 ⎟
  ⎜ 0  2  0  3  5 ⎟
  ⎜ 0  7  3  0  4 ⎟
  ⎜ 6  0  5  4  0 ⎟
```

Построение графа по заданным матрицам

Зададим матрицу смежности A

```
In[174]:= A = {{0, 1, 1}, {1, 0, 0}, {0, 1, 1}};
```

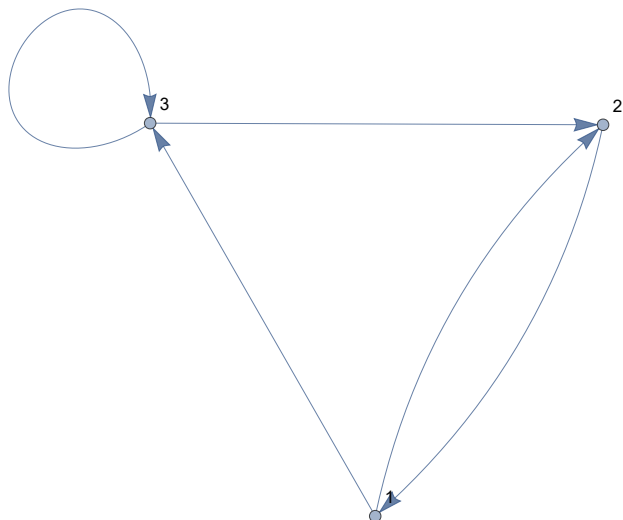
```
In[175]:= A // MatrixForm
           ↳матричная форма
```

```
Out[175]//MatrixForm=
  ⎛ 0  1  1 ⎞
  ⎜ 1  0  0 ⎟
  ⎜ 0  1  1 ⎟
```

Построим граф:

```
In[176]:= gA = AdjacencyGraph[A, VertexLabels -> Automatic]
          |граф по матрице смежности |метки для вершин |автоматически
```

Out[176]=



```
In[177]:= EdgeList[gA]
          |список рёбер
```

Out[177]= {1 → 2, 1 → 3, 2 → 1, 3 → 2, 3 → 3}

```
In[178]:= VertexList[gA]
          |список вершин графа
```

Out[178]= {1, 2, 3}

```
In[179]:= A1 = {{0, 1, 1}, {1, 0, 0}, {1, 0, 1}};
```

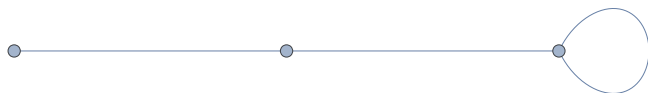
```
In[181]:= A1 // MatrixForm
          |матричная форма
```

Out[181]//MatrixForm=

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

```
In[180]:= AdjacencyGraph[A1]
          |граф по матрице смежности
```

Out[180]=



Зададим матрицу инцидентности i

```
In[182]:= i = {{2, 0, 1, 1}, {0, 1, 0, 1}, {0, 1, 1, 0}};
```

```
In[42]:= i // MatrixForm
```

матричная форма

```
Out[42]//MatrixForm=
```

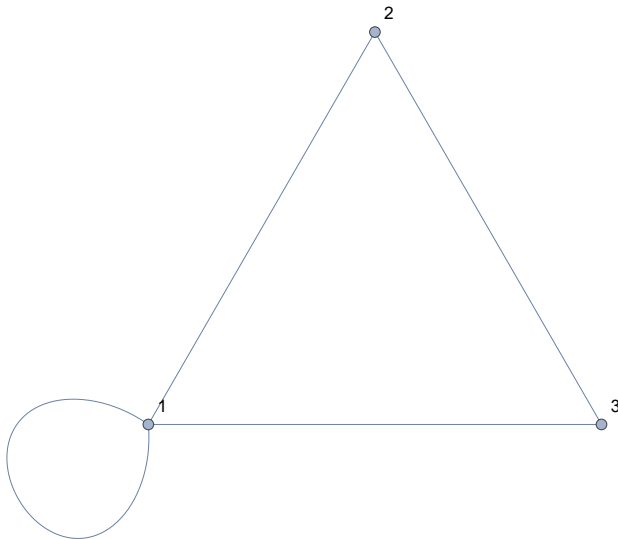
$$\begin{pmatrix} 2 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

Построим граф по ней

```
In[183]:= IncidenceGraph[i, VertexLabels → Automatic]
```

граф по матрице инц... метки для вершин автоматически

```
Out[183]=
```



Зададим весовую матрицу w : (Внимание : если между вершинами нет связи то нужно указать вес равный бесконечности)

```
In[184]:= w = {{∞, 5, 3}, {5, ∞, ∞}, {3, ∞, 8}};
```

```
In[47]:= w // MatrixForm
```

матричная форма

```
Out[47]//MatrixForm=
```

$$\begin{pmatrix} \infty & 5 & 3 \\ 5 & \infty & \infty \\ 3 & \infty & 8 \end{pmatrix}$$

```
In[185]:= WeightedAdjacencyGraph[w, VertexLabels → Automatic, EdgeLabels → "EdgeWeight"]
```

граф по матрице весов метки для вершин автоматически пометки для рёбра вес ребра

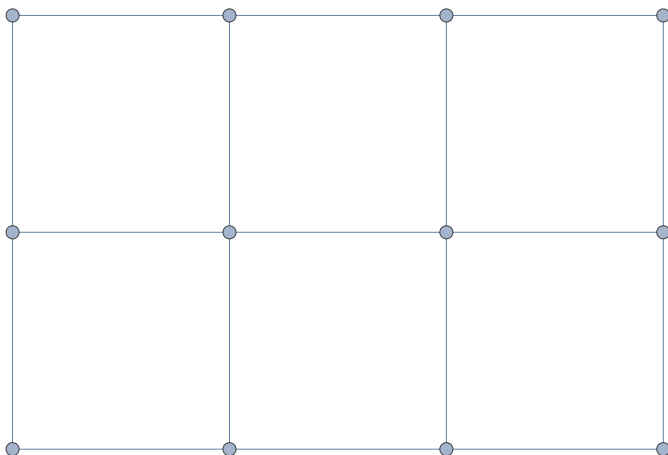
```
Out[185]=
```



Некоторые примеры встроенных графов

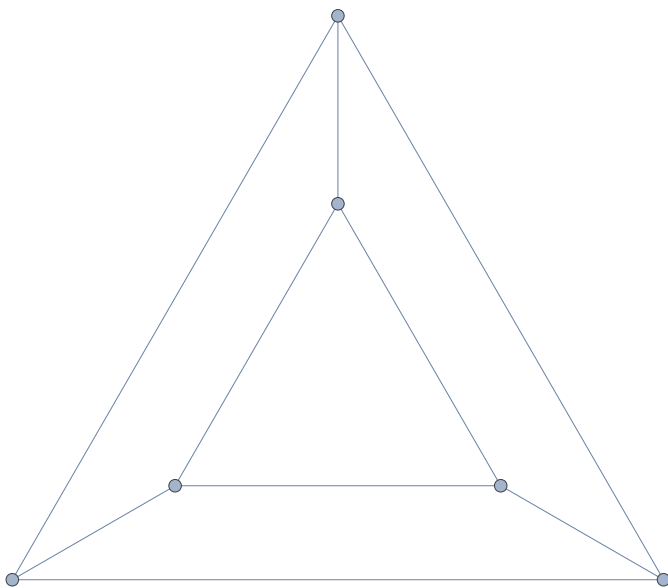
In[49]:= **GridGraph**[{3, 4}] (*Решетчатый граф – сетка из 3*4 вершин*)
|решётчатый граф

Out[49]=



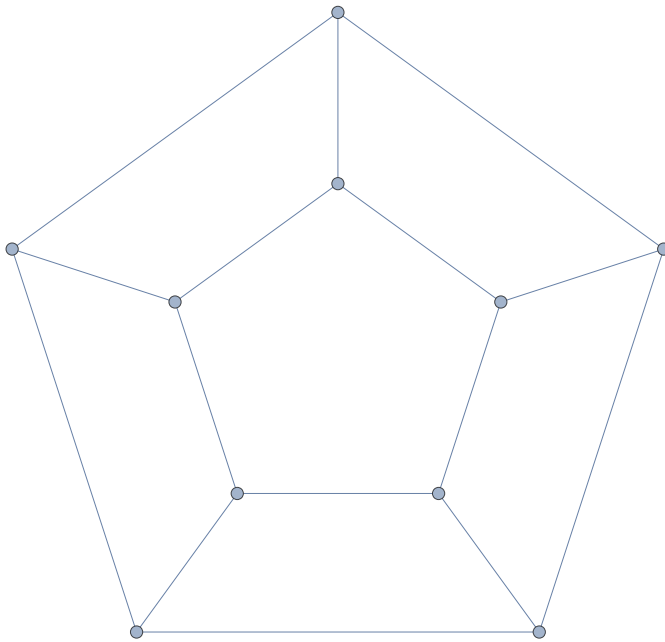
In[50]:= **PetersenGraph**[3, 2] (*Граф Петерсена $P_{3,2}$ *)
|граф Петерсена

Out[50]=



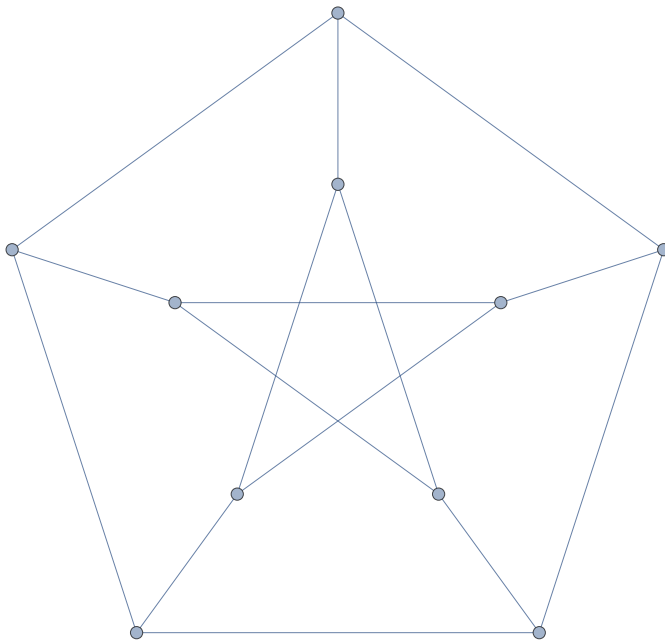
In[51]:= **PetersenGraph[5, 1]** (*Граф Петерсена $P_{5,1}$ *)
Граф Петерсена

Out[51]=



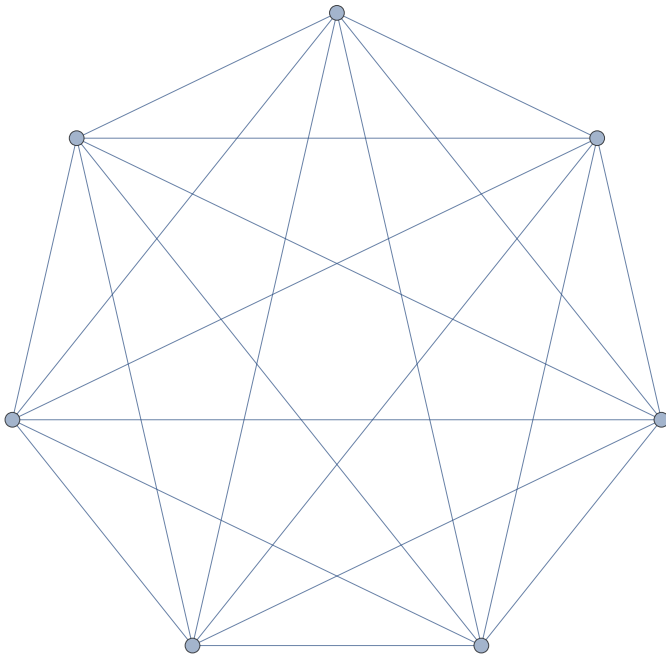
In[52]:= **PetersenGraph[5, 2]** (*Граф Петерсена $P_{5,2}$ *)
Граф Петерсена

Out[52]=



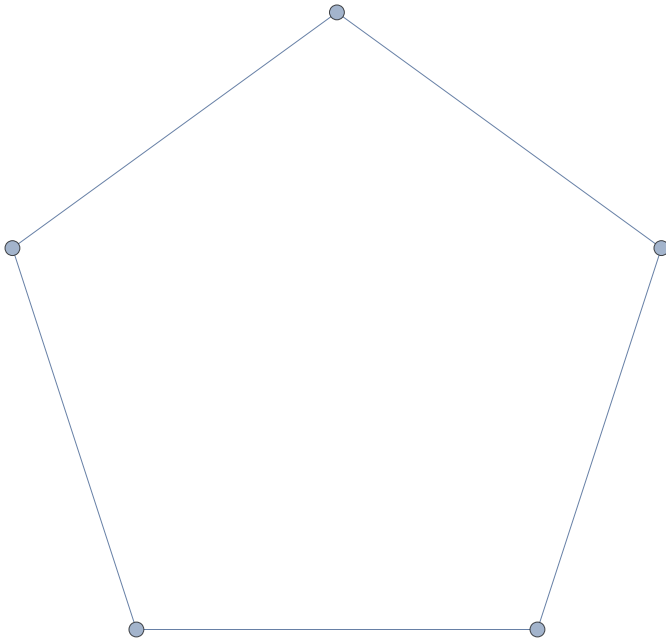
```
In[53]:= CompleteGraph[7] (*Полный граф  $K_7$ *)  
|полный граф
```

Out[53]=



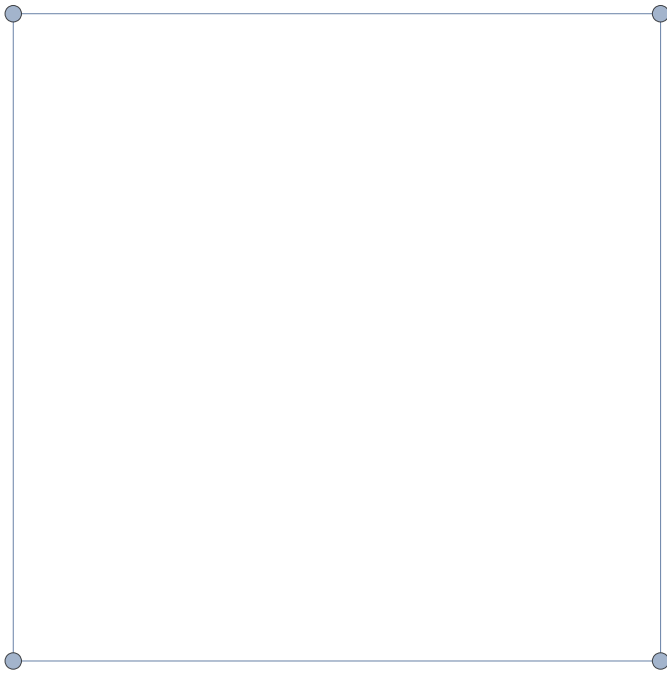
```
In[54]:= CycleGraph[5] (*Цикл из 5 вершин*)  
|граф-цикл
```

Out[54]=



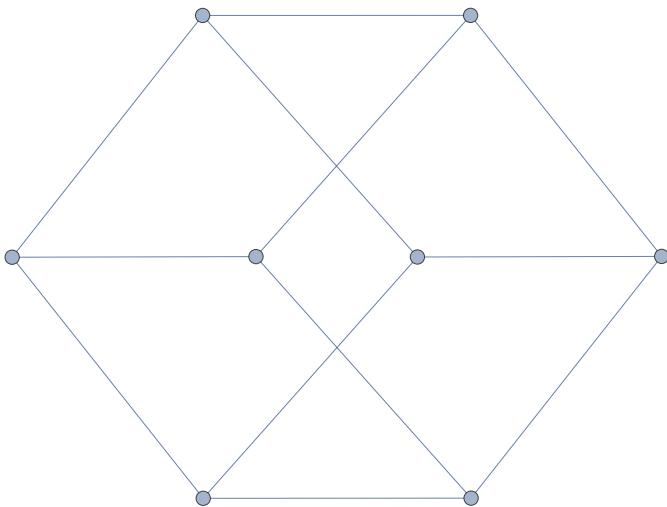
In[186]:= **HyperscubeGraph[2]** (*Граф гиперкуба Q_2 *)
_граф гиперкуба

Out[186]=



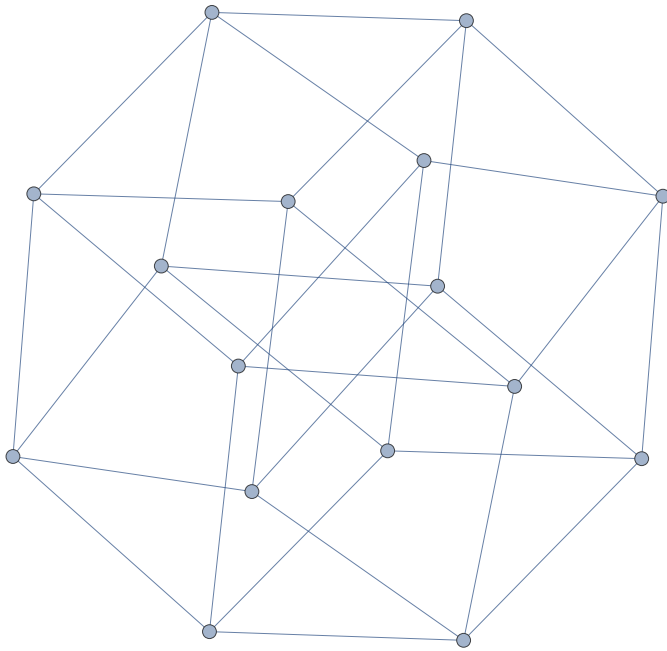
In[187]:= **HyperscubeGraph[3]** (*Граф гиперкуба Q_3 *)
_граф гиперкуба

Out[187]=



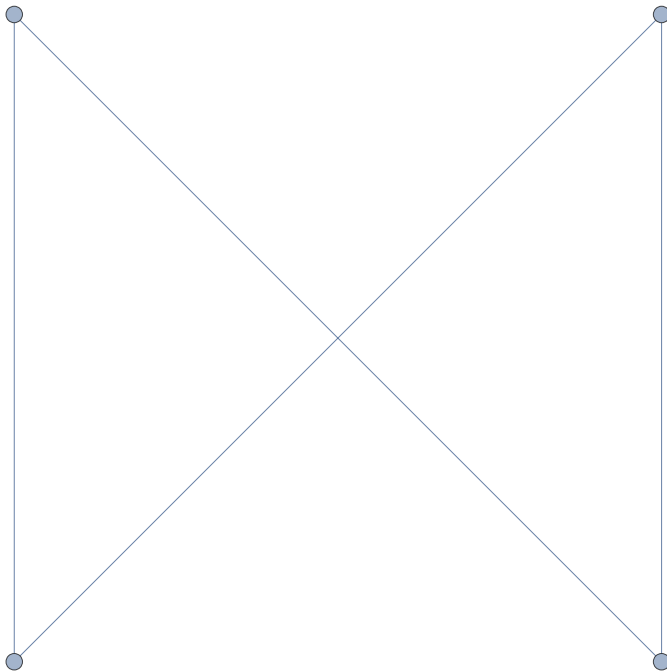
In[188]:= **HypercubeGraph[4]** (*Граф гиперкуба Q_4 *)
|граф гиперкуба

Out[188]=



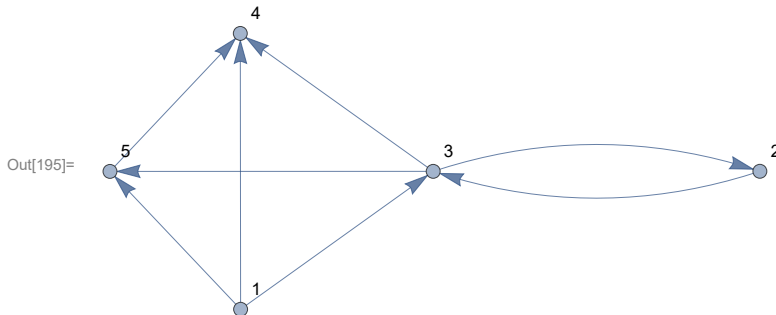
In[56]:= **ButterflyGraph[1]** (*Граф "бабочка"*)
|граф песочные часы

Out[56]=



3. Ориентированные графы

```
In[195]:= g1 =
  Graph[{1 → 5, 2 → 3, 3 → 5, 3 → 2, 1 → 4, 5 → 4, 1 → 3, 3 → 4}, VertexLabels → Automatic]
  |граф |метки для вершин |автоматически
```



Задается точно также, только вместо ненаправленного ребро ставится направленное \rightarrow :

"Esc", "d", "e", "Esc" или DirectedEdge[i, j] : DirectedEdge[1, 2] = 1 \rightarrow 2

```
In[58]:=
```

Выведем матрицы графа :

```
In[196]:= AdjacencyMatrix[g1] // MatrixForm
  |матрица смежности |матричная форма
```

```
Out[196]//MatrixForm=
```

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

```
In[197]:= IncidenceMatrix[g1] // MatrixForm
  |матрица инцидентов |матричная форма
```

```
Out[197]//MatrixForm=
```

$$\begin{pmatrix} -1 & 0 & 0 & 0 & -1 & 0 & -1 & 0 \\ 1 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & -1 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Еще одна характеристика графа : степень вершины .

```
In[199]:= VertexDegree[g1, 1] (*степень 1ой вершины: из нее исходит 3 ребра*)
  |степень вершины
```

```
Out[199]= 3
```

```
In[200]:= VertexDegree[g1, 3]
  |степень вершины
```

```
Out[200]= 5
```

Получим список степеней всех вершин :

```
In[201]:= VertexDegree[g1]
```

степень вершины

```
Out[201]= {3, 3, 2, 5, 3}
```

Также есть возможность получить полустепени захода и исхода каждой вершины :

```
In[202]:= VertexInDegree[g1]
```

входящая степень вершины

```
Out[202]= {0, 2, 1, 2, 3}
```

```
In[203]:= VertexOutDegree[g1]
```

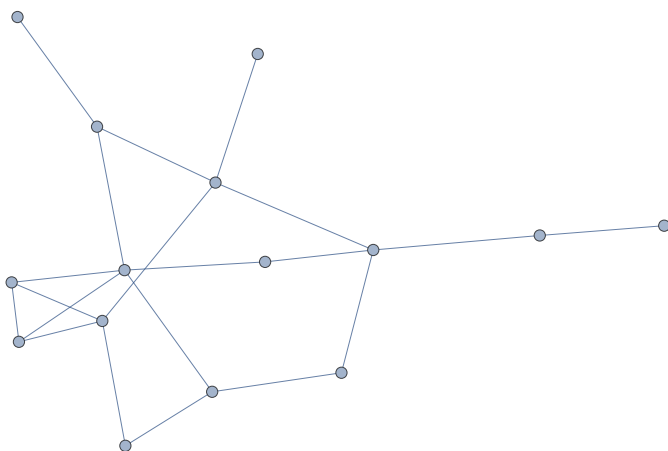
выходящая степень вершины

```
Out[203]= {3, 1, 1, 3, 0}
```

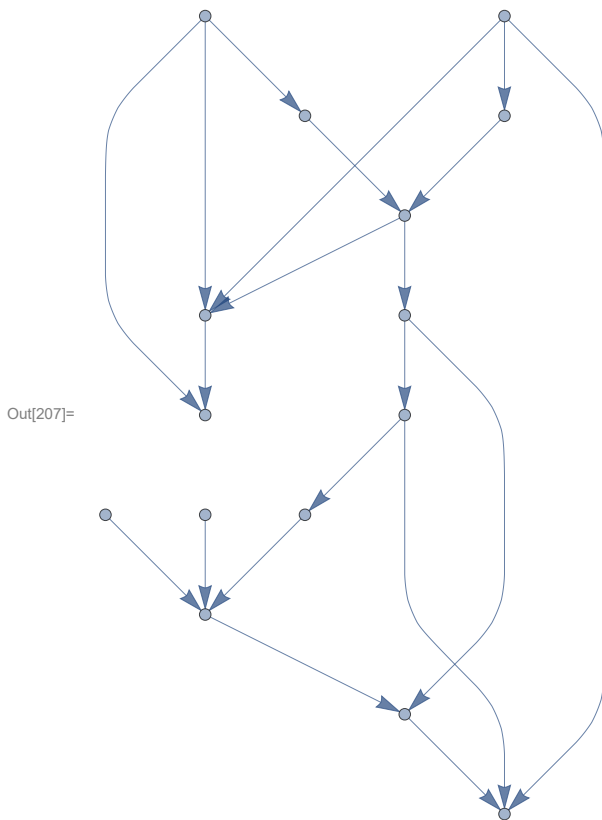
```
In[206]:= RandomGraph[{15, 20}]
```

случайный граф

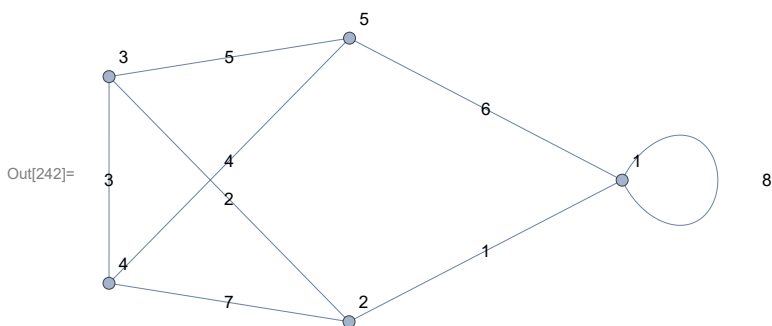
```
Out[206]=
```



```
In[207]:= RandomGraph[{15, 20}, DirectedEdges → True]
```



```
In[242]:= g1
```



```
In[241]:= GraphDistanceMatrix[g1] // MatrixForm
```

матрица расстояний на графе
матричная форма

Out[241]//MatrixForm=

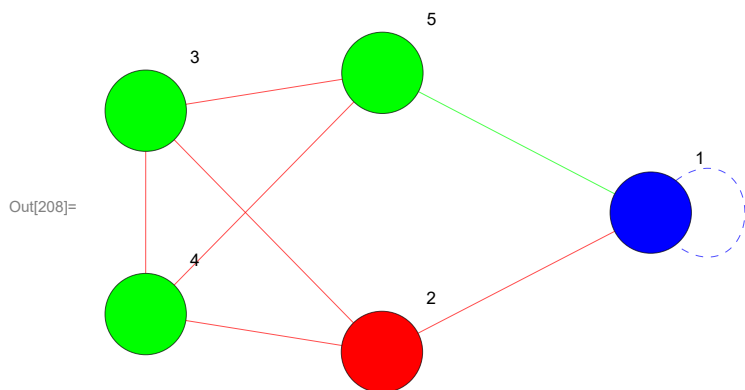
$$\begin{pmatrix} 0. & 1. & 3. & 6. & 6. \\ 1. & 0. & 2. & 5. & 7. \\ 3. & 2. & 0. & 3. & 5. \\ 6. & 5. & 3. & 0. & 4. \\ 6. & 7. & 5. & 4. & 0. \end{pmatrix}$$

-> EdgeWeight

4. Операции с графами

Наш граф g

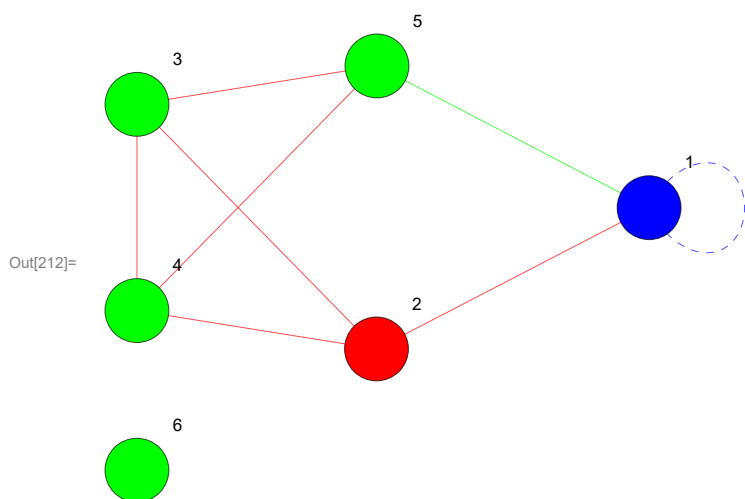
In[208]:= **g**



Добавление вершины :

In[212]:= **h1 = VertexAdd[g, 6]**

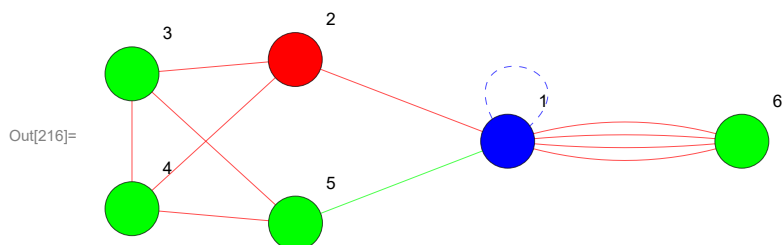
добавить вершину



Добавление ребра

In[216]:= **h1 = EdgeAdd[h1, {6 ↔ 1}]**

добавить ребро



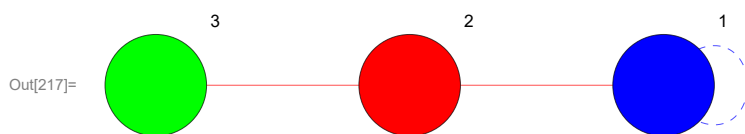
EdgeDelete, VertexDelete

удалить ребро удалить вершину

Выделение подграфа

```
In[217]:= q1 = Subgraph[h1, {3, 2, 1}, VertexLabels -> Automatic]
```

подграф метки для вершин автоматически



```
In[218]:= q2 = Subgraph[h1, {3, 6, 4}, VertexLabels -> Automatic]
```

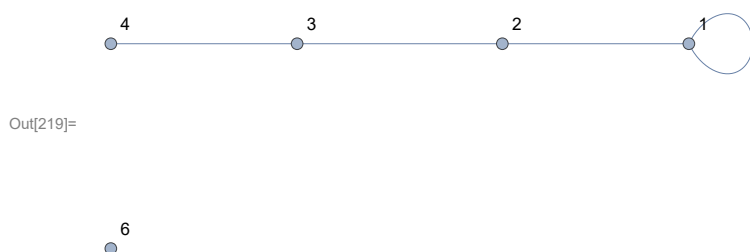
подграф метки для вершин автоматически



Объединение графов

```
In[219]:= GraphUnion[q1, q2, VertexLabels -> Automatic]
```

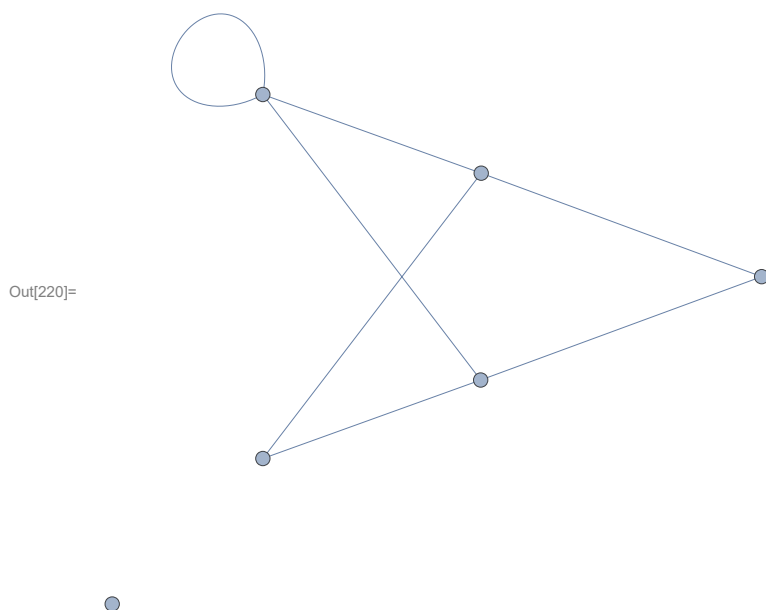
объединение графов метки для вершин автоматически



Разность графов

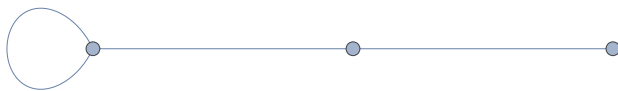
```
In[220]:= GraphDifference[g, q2]
```

разность графов



Пересечение графов

In[221]:= **GraphIntersection[g, q1]**
 [пересечение графов]



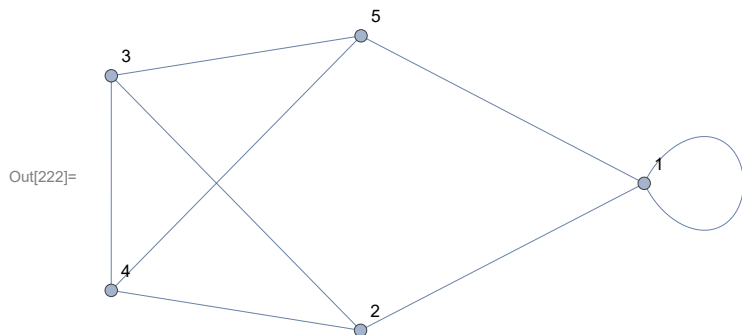
Out[221]=



5. Некоторые задачи на графах

Наш граф g

In[222]:= **g =**
Graph[{1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 5, 3 ↔ 5, 1 ↔ 5, 2 ↔ 4, 1 ↔ 1}, VertexLabels → Automatic]
 [граф] [метки для вершин] [автоматически]

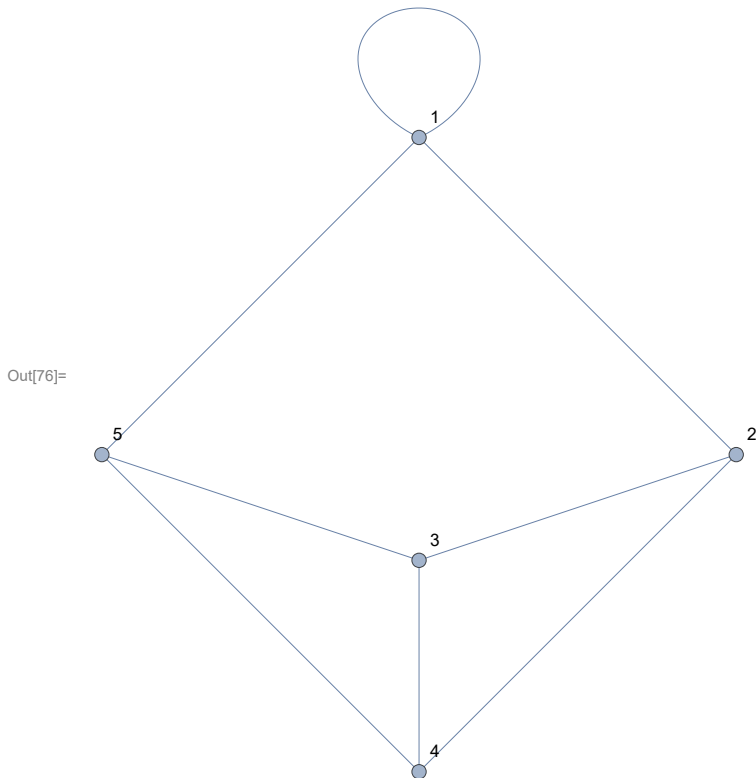


Out[222]=

In[75]:= **PlanarGraphQ[g] (*Проверка на планарность*)**
 [планарный граф?]

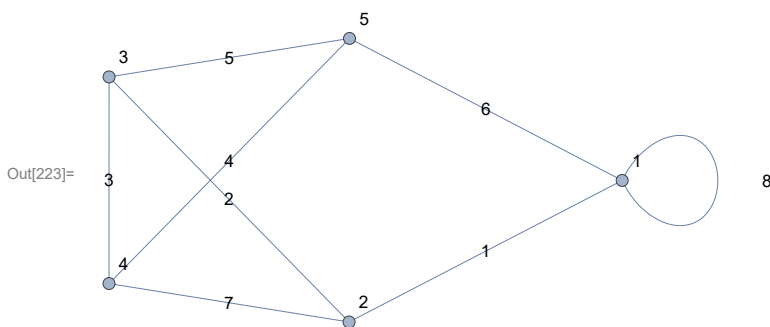
Out[75]= **True**

In[76]:= **PlanarGraph[g]** (*Укладка графа на плоскость без самопересечений*)
 ↳планарный граф



Наш весовой граф g1

In[223]:= **g1 =**
Graph[{1 ↔ 2, 2 ↔ 3, 3 ↔ 4, 4 ↔ 5, 3 ↔ 5, 1 ↔ 5, 2 ↔ 4, 1 ↔ 1}, VertexLabels → Automatic,
 ↳граф ↳метки для вершин ↳автоматически
EdgeLabels → "EdgeWeight", EdgeWeight → {1, 2, 3, 4, 5, 6, 7, 8}]
 ↳пометки для рё... ↳вес ребра ↳вес ребра



Поиск короткого пути между двумя вершинами (1 и 4)

In[226]:= **f1 = FindShortestPath[g1, 1, 4]** (*Сам путь*)
 ↳найти кратчайший путь

Out[226]= {1, 2, 3, 4}

In[224]:= **GraphDistance[g1, 1, 4]** (*Его вес*)
 ↳расстояние на графе

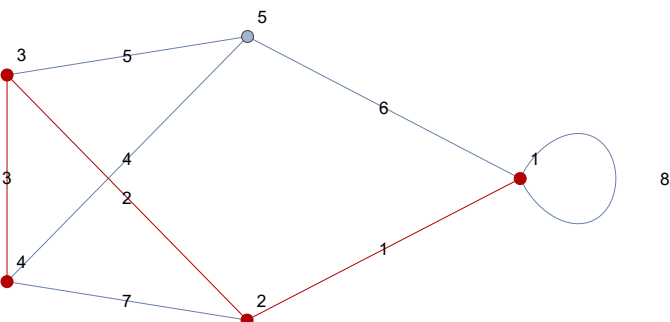
Out[224]= 6.

In[82]:= (*Выделим его на графе*)

In[227]:= PathGraph[f1]
 |граф маршрута

Out[227]= 

In[83]:= HighlightGraph[g1, PathGraph[f1]]
 |граф с подкраской |граф маршрута

Out[83]= 

In[84]:=

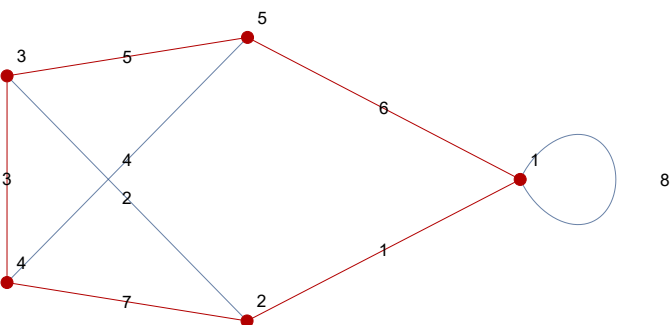
Поиск Гамильтонова цикла

In[86]:= f2 = FindHamiltonianCycle[g1]
 |найти гамильтонов цикл

Out[86]= { {1 ↔ 2, 2 ↔ 4, 4 ↔ 3, 3 ↔ 5, 5 ↔ 1} }

In[87]:= HighlightGraph[g1, PathGraph[f2[[1]]]
 |граф с подкраской |граф маршрута

(*[[1]] выдает первый подсписок, т.к. циклов может быть несколько*)

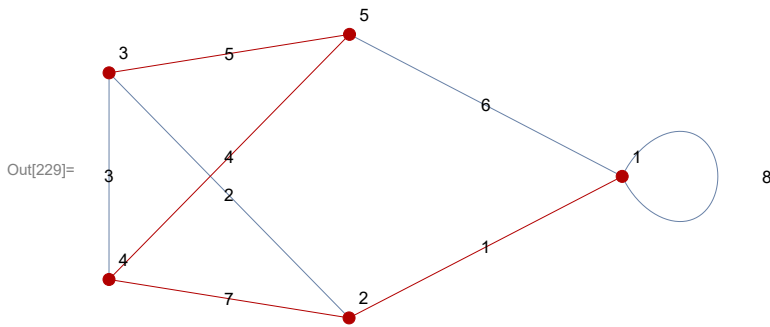
Out[87]= 

Поиск Гамильтонова пути

In[228]:= f3 = FindHamiltonianPath[g1, 3, 1]
 |найти гамильтонов путь

Out[228]= {3, 5, 4, 2, 1}

In[229]:= **HighlightGraph**[g1, PathGraph[f3]]
 [граф с подкраской] [граф маршрута]

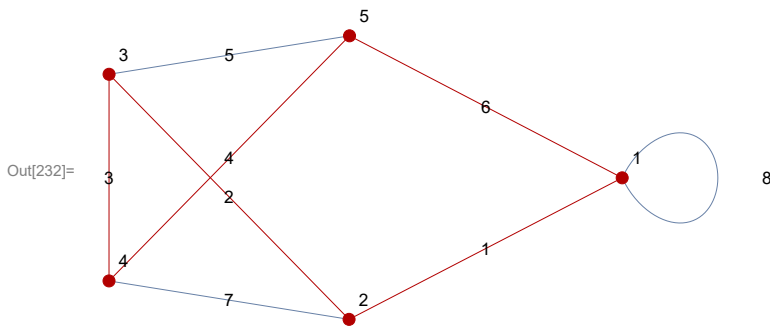


Поиск кратчайшего обхода по вершинам ("задача коммивояжера")

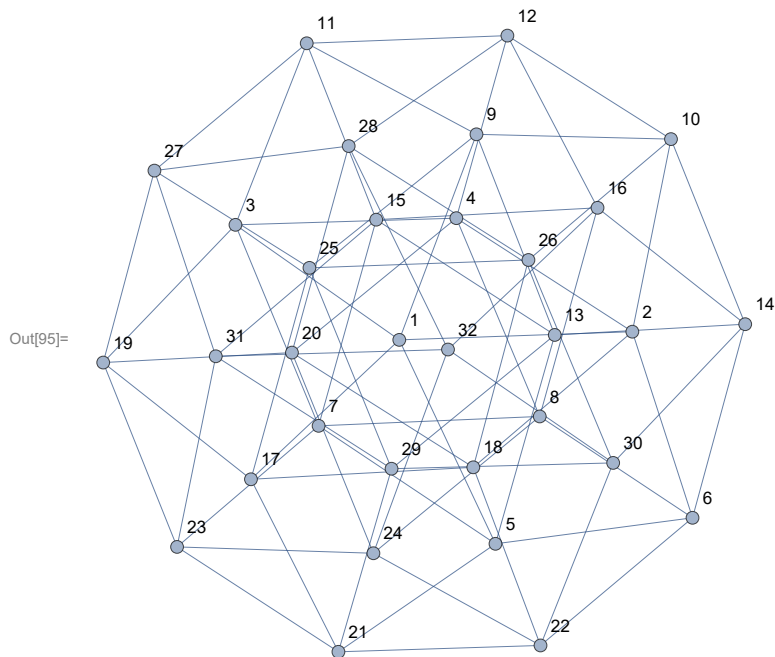
In[230]:= **q = FindShortestTour**[g1]
 [найти кратчайший тур]

Out[230]= {16, {1, 5, 4, 3, 2, 1}}

In[232]:= **HighlightGraph**[g1, PathGraph[q[[2]]]]
 [граф с подкраской] [граф маршрута]



In[95]:= **gg = HypercubeGraph**[5, VertexLabels -> Automatic]
 [граф гиперкуба] [метки для вершин] [автоматически]



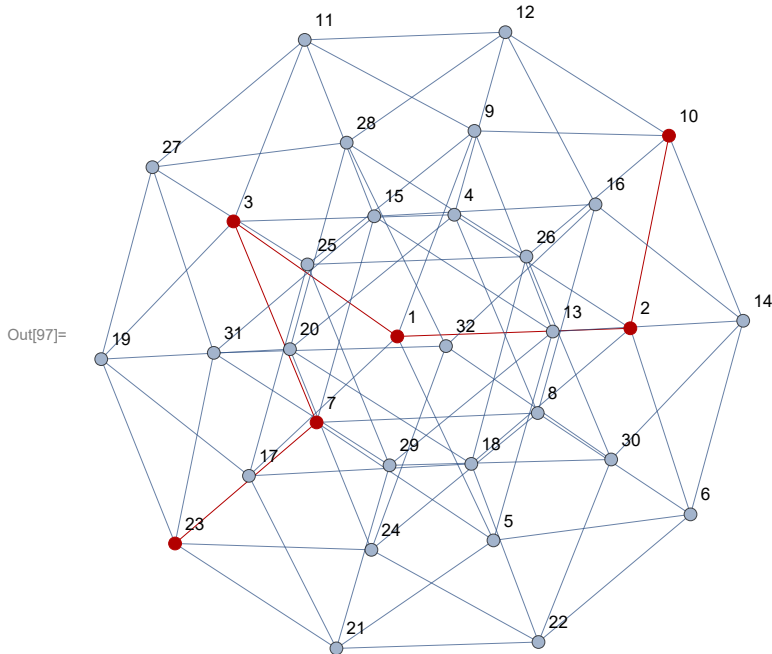
```
In[96]:= qq = FindShortestPath[gg, 10, 23]
```

найти кратчайший путь

```
Out[96]:= {10, 2, 1, 3, 7, 23}
```

```
In[97]:= HighlightGraph[gg, PathGraph[qq]]
```

граф с подкраской граф маршрута



```
In[98]:=
```

Интересный пример использования Wolfram Mathematica : задача коммивояжера по большим городам США

```
In[233]:= usa = CityData[{Large, "USA"}]
```

данные о г... крупный

```
Out[233]:= { New York City , Los Angeles , Chicago , Houston , Philadelphia , Phoenix ,
San Antonio , San Diego , Dallas , San Jose , Austin , Jacksonville ,
San Francisco , Indianapolis , Columbus , Fort Worth , Charlotte , Seattle ,
Denver , El Paso , Detroit , Washington , Nashville , Boston , Memphis ,
Portland , Oklahoma City , Las Vegas , Baltimore , Milwaukee , Albuquerque ,
Tucson , Fresno , Sacramento , Kansas City , Long Beach , Mesa , Atlanta ,
Colorado Springs , Virginia Beach , Raleigh , Omaha , Miami , Oakland ,
Minneapolis , Tulsa , Wichita , New Orleans , Arlington , Cleveland , Bakersfield ,
Tampa , Aurora , Honolulu , Anaheim , Santa Ana , Corpus Christi , Riverside ,
Saint Louis , Lexington , Stockton , Pittsburgh , Oyster Bay , Saint Paul ,
```

Anchorage , Cincinnati , Henderson , Greensboro , Plano , Newark , Toledo ,
 Lincoln , Orlando , Chula Vista , Jersey City , Chandler , Fort Wayne , Buffalo ,
 Durham , Saint Petersburg , Irvine , Louisville , Laredo , Lubbock , Madison ,
 Gilbert , Norfolk , Reno , Winston-Salem , Glendale , Hialeah , Garland ,
 Scottsdale , Irving , Chesapeake , North Las Vegas , Fremont , North Hempstead ,
 Baton Rouge , Paradise , Richmond , Boise , San Bernardino , Spokane ,
 Birmingham , Modesto , Des Moines , Rochester , Tacoma , Arlington , Fontana ,
 Oxnard , Moreno Valley , Fayetteville , Huntington Beach , Augusta , Yonkers ,
 Glendale , Aurora , Montgomery , Columbus , Amarillo , Little Rock , Akron ,
 Shreveport , Grand Rapids , Mobile , Salt Lake City , Huntsville , Tallahassee ,
 Sunrise Manor , Grand Prairie , Overland Park , Knoxville , Worcester ,
 Brownsville , Newport News , Santa Clarita , Port Saint Lucie , Providence ,
 Fort Lauderdale , Spring Valley , Chattanooga , Tempe , Oceanside , Garden Grove ,
 Rancho Cucamonga , Cape Coral , Santa Rosa , Vancouver , Sioux Falls ,
 Peoria , Ontario , Jackson , Elk Grove , Springfield , Pembroke Pines , Salem ,
 Corona , Eugene , McKinney , Fort Collins , Lancaster , Cary , Palmdale ,
 Hayward , Salinas , Frisco , Springfield , Pasadena , Alexandria , Pomona ,
 Lakewood , Sunnyvale , Escondido , Kansas City , Hollywood , Clarksville ,
 Torrance , Rockford , Joliet , Paterson , Bridgeport , Naperville , Savannah ,
 Mesquite , Syracuse , Pasadena , Orange , Fullerton , Killeen , Dayton ,
 McAllen , Bellevue , Metairie , Miramar , Hampton , West Valley City ,
 Warren , Ramapo , Olathe , Columbia , Thornton , Carrollton , Midland ,
 Charleston , Waco , Sterling Heights , Denton , Cedar Rapids , New Haven ,
 Roseville , Gainesville , Visalia , Coral Springs , Thousand Oaks , Elizabeth ,
 Stamford , Concord , Surprise , Lafayette , Topeka , Kent , Simi Valley ,
 East Los Angeles , Santa Clara , Murfreesboro , Amherst , Hartford , Victorville ,
 Abilene , Vallejo , Berkeley , Athens , Norman , Allentown , Evansville ,
 Columbia , Odessa , Fargo , Beaumont , Independence , Ann Arbor , El Monte ,
 Springfield , Round Rock , Wilmington , Arvada , Provo , Peoria , Lansing ,
 Downey , Carlsbad , Costa Mesa , Westminster , Clearwater , Fairfield ,

```

Rochester , Elgin , Temecula , West Jordan , Inglewood , Richardson ,
Lowell , Gresham , Antioch , Cambridge , High Point , Billings , Manchester ,
Murrieta , Centennial , San Buenaventura (Ventura) , Richmond , Jurupa Valley ,
Pueblo , Pearland , Waterbury , West Covina , Enterprise , North Charleston ,
Everett , College Station , Palm Bay , Pompano Beach , Boulder , Norwalk ,
West Palm Beach , Broken Arrow , Daly City , Sandy Springs , Burbank , Green Bay ,
Santa Maria , Wichita Falls , Lakeland , Clovis , Lewisville , Tyler , El Cajon ,
San Mateo , Brandon , Rialto , Edison , Davenport , Hillsboro , Las Cruces ,
South Bend , Spokane Valley , Vista , Greeley , Davie , San Angelo , Renton }

```

```

In[101]:= pos = GeoPosition[CityData[#, "Coordinates"]] & /@ usa;
           [гео-координ... [данные о городах

```

```

In[119]:= {ny, la} = Flatten[Position[usa, #] & /@ {New York City , Los Angeles }];
           [уплостить [позиция по образцу

```

```

In[126]:= tour = FindShortestTour[pos, ny, la]
           [найти кратчайший тур

```

```

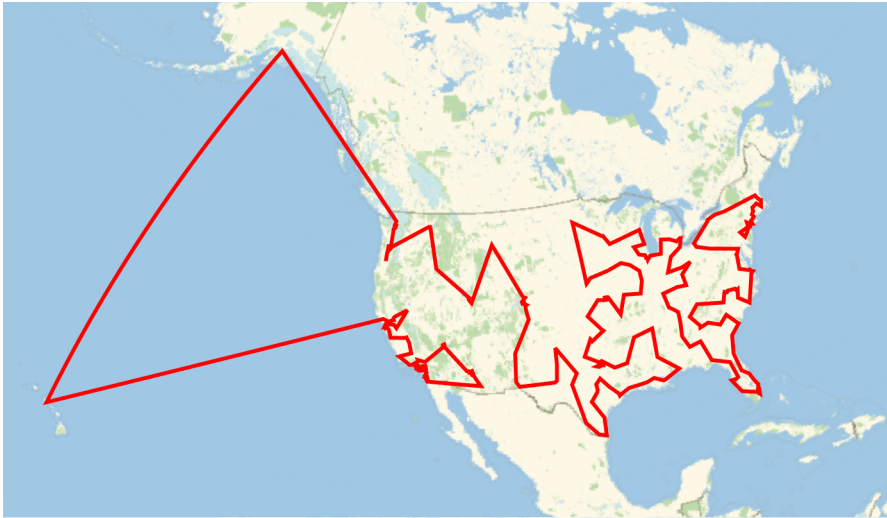
Out[126]= { 38 369.6 km , {1, 75, 217, 304, 5, 236, 70, 182, 200, 117, 98, 63, 218, 183, 211, 278,
229, 169, 135, 140, 24, 267, 264, 270, 187, 108, 228, 78, 50, 124, 62, 29, 22, 110,
171, 101, 137, 197, 87, 40, 95, 247, 114, 41, 164, 79, 68, 268, 89, 17, 134, 143,
291, 38, 234, 116, 202, 281, 206, 185, 12, 73, 284, 139, 288, 215, 285, 141, 177,
43, 91, 196, 312, 157, 148, 80, 256, 52, 302, 296, 213, 130, 121, 120, 105, 129,
227, 23, 178, 237, 82, 60, 66, 15, 192, 14, 77, 71, 21, 208, 199, 243, 251, 126,
308, 3, 181, 184, 119, 259, 180, 85, 30, 293, 258, 64, 45, 240, 151, 72, 42, 107,
210, 305, 250, 245, 59, 238, 156, 242, 35, 176, 133, 201, 222, 47, 46, 289, 235,
27, 295, 209, 298, 204, 94, 16, 49, 132, 9, 186, 92, 263, 69, 168, 161, 299, 125,
123, 25, 154, 127, 48, 195, 99, 221, 241, 170, 277, 4, 283, 207, 191, 246, 11, 7,
57, 136, 193, 83, 313, 231, 122, 84, 205, 239, 20, 307, 31, 276, 39, 272, 53, 19,
203, 255, 248, 173, 286, 311, 162, 269, 249, 261, 198, 128, 102, 309, 104, 265,
158, 160, 306, 26, 150, 109, 223, 314, 194, 18, 282, 65, 54, 149, 257, 155, 34,
212, 88, 106, 61, 266, 219, 232, 274, 233, 44, 290, 13, 301, 166, 97, 226, 174, 10,
167, 33, 297, 214, 51, 294, 273, 112, 216, 224, 138, 165, 163, 230, 280, 142, 28,
96, 131, 100, 67, 220, 152, 90, 6, 93, 37, 144, 76, 86, 32, 74, 8, 300, 175, 253,
310, 145, 260, 271, 113, 103, 303, 111, 147, 153, 275, 58, 159, 189, 56, 81, 254,
115, 146, 55, 190, 172, 279, 244, 188, 118, 292, 225, 252, 287, 36, 179, 262, 2} }

```

```
In[127]:= GeoGraphics[{Thick, Red, GeoPath[usa[tour[[2]]]]}]
```

гео-графика жирный кр... гео-кривая

Out[127]=



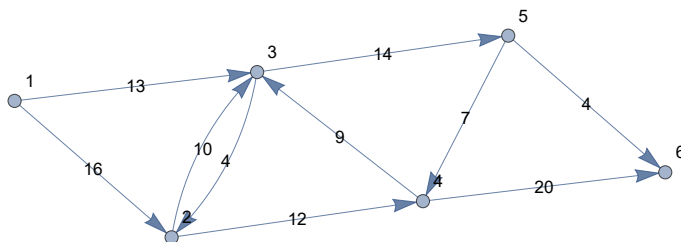
In[105]:=

Сети

```
In[244]:= ff = Graph[{1 → 2, 2 → 3, 3 → 2, 1 → 3, 2 → 4, 4 → 3, 4 → 6, 3 → 5, 5 → 4, 5 → 6},
  VertexLabels → Automatic, EdgeWeight → {16, 10, 4, 13, 12, 9, 20, 14, 7, 4},
  EdgeCapacity → {16, 10, 4, 13, 12, 9, 20, 14, 7, 4}, EdgeLabels → "EdgeWeight"]
```

граф метки для вершин автоматиче... вес ребра ёмкость ребра пометки для рё... вес ребра

Out[244]=



```
In[243]:= FindMaximumFlow[ff, 1, 6]
```

найти максимальный поток

Out[243]= 23

```
In[236]:= data = FindMaximumFlow[ff, 1, 6, "OptimumFlowData"]
```

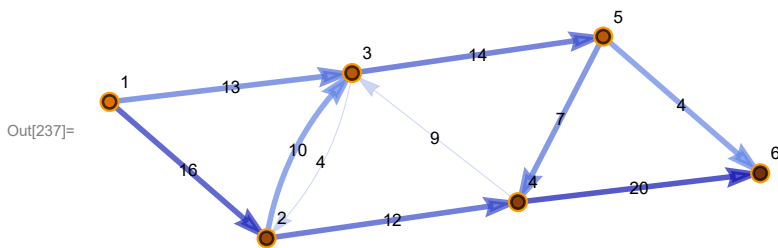
найти максимальный поток данные об оптимальном

Out[236]= OptimumFlowData[ Flow value: 23]

```
In[111]:= data["FlowValue"]
```

Out[111]= 23

```
In[237]:= data["FlowGraph"]
```



```
In[238]:= Grid[{#, data[#]} & /@ data["EdgeList"], Frame → All]
```

таблица список рёбер рамка всё

Out[238]=

1 ↔ 2	16
1 ↔ 3	7
2 ↔ 3	4
2 ↔ 4	12
3 ↔ 5	11
4 ↔ 6	19
5 ↔ 4	7
5 ↔ 6	4

```
In[239]:= data["FlowMatrix"] // MatrixForm
```

матричная форма

Out[239]//MatrixForm=

$$\begin{pmatrix} 0 & 16 & 7 & 0 & 0 & 0 \\ 0 & 0 & 4 & 12 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 11 \\ 0 & 0 & 0 & 0 & 19 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 4 & 0 \end{pmatrix}$$

```
In[245]:= ff1 = Graph[{1 → 2, 2 → 3, 3 → 2, 1 → 3, 2 → 4, 4 → 3, 4 → 6, 3 → 5, 5 → 4, 5 → 6},
```

граф

```
VertexLabels → Automatic, EdgeWeight → {16, 10, 4, 13, 12, 9, 20, 14, 7, 4},
```

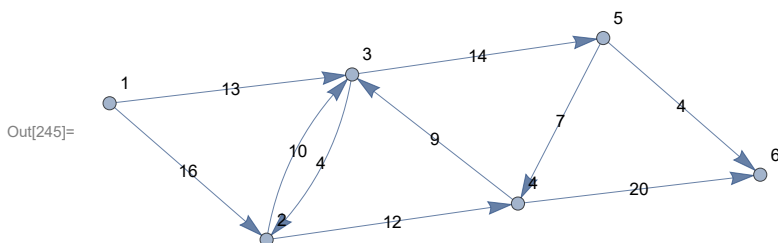
метки для вершин автоматиче... вес ребра

```
EdgeCost → {16, 10, 4, 13, 12, 9, 20, 14, 7, 4},
```

стоимость ребра

```
EdgeCapacity → {16, 10, 4, 13, 12, 9, 20, 14, 7, 4}, EdgeLabels → "EdgeWeight"]
```

ёмкость ребра пометки для рё... вес ребра



```
In[246]:= FindMinimumCostFlow[ff1, 1, 6]
```

найти поток минимальной стоимости

Out[246]= 802