БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ
И ИНФОРМАТИКИ
Кафедра компьютерных технологий и систем

В. Б. Таранчук

ПОСТРОЕНИЕ, ВИЗУАЛИЗАЦИЯ, ПРИМЕРЫ АНАЛИЗА ГРАФОВ В СИСТЕМЕ *MATHEMATICA*

Учебные материалы
для студентов факультета
прикладной математики и информатики

МИНСК
2023

УДК 519.67(075.8)
ББК 22.19я73
Т19

Рекомендовано советом
факультета прикладной математики и информатики БГУ
29 июня 2023 г., протокол № 12

Р е ц е н з е н т
доктор физико-математических наук *Н. Н. Гринчик*

Таранчук, В. Б.

Т19 Построение, визуализация, примеры анализа графов в системе *Mathematica* : учеб. материалы для студентов фак. прикладной математики и информатики / В. Б. Таранчук. – Минск : БГУ, 2023. – 51 с.

Изложены основные функции, опции и директивы, используемые при построении, визуализации, оформлении графов в системе *Mathematica*. Включены примеры, которые иллюстрируют возможности получения и вывода комбинаторных свойств, модификации, анализа, сравнения графов, а также примеры решений нескольких типовых задач.

Предназначено для студентов факультета прикладной математики и информатики.

УДК 519.67(075.8)
ББК 22.19я73

ПРЕДИСЛОВИЕ

В учебных материалах изложены методические вопросы, примеры решения возникающих в практике интеллектуального анализа данных задач построения, анализа и визуализации графов с использованием инструментов системы компьютерной алгебры Wolfram *Mathematica*: Graph, Basic Properties, Structural Properties, Graph Isomorphism, Basic Measures, Distance Measures, Connectivity Measures, Centrality Measures, Similarity, GraphComplement, GraphDifference, FindClique, RandomGraph, FindShortestPath. Приведенные тексты, коды программных модулей, иллюстрации рекомендуются студентам при изучении (в формате управляемой самостоятельной работы) дисциплин специализации: «Когнитивная визуализация» (учебная дисциплина для специальности 1-31 03 03 «Прикладная математика», направление специальности 1-31 03 03-01 «Прикладная математика, научно-производственная деятельность»), «Компьютерный анализ и визуализация» (учебная дисциплина для специальности 1-31 03 07 «Прикладная информатика», направление специальности 1-31 03 07-01 «Прикладная информатика, программное обеспечение компьютерных систем»), «Технологии интерактивной визуализации» (учебная дисциплина для специальности 1-31 03 04 «Информатика»), «Интерактивные вычисления и визуализация» (учебная дисциплина для специальности 1-31 03 03 «Прикладная математика», направление специальности 1-31 03 03-01 «Прикладная математика, научно-производственная деятельность»).

В материалах акценты сделаны на:

- развитие навыков функционального программирования средствами системы Wolfram *Mathematica*;
- освоение обучаемыми инструментов создания в *Mathematica* интерактивных программных модулей с возможностями точных символьных вычислений, интерактивного графического представления результатов математических преобразований и расчётов.

Советы и критические замечания по изданию просьба направлять на taranchuk@bsu.by.

ПОСТРОЕНИЕ И ВИЗУАЛИЗАЦИЯ ГРАФОВ

Считается, что читатели, использующие материалы данного пособия, знают основы теории графов, и целью текущего изложения является ознакомление с широкими возможностями СКА Wolfram *Mathematica*, в основном, для тестирования собственных алгоритмических и программных реализаций путем сопоставления, т.к. все примеры и решения в *Mathematica* всегда тщательно проверены, оптимизированы по быстродействию выполнения.

Графы являются представительными элементами языка Wolfram Language, могут использоваться в качестве входных и выходных данных в программах и документах. Неориентированные и ориентированные графы обрабатываются в системе единообразно и поддерживают ряд стандартных свойств для вершин и ребер. Существенно, что графы в *Mathematica* также допускают пользовательские свойства для обеспечения гибкости моделирования или вычислений, могут быть преобразованы для различных представлений, включая матрицы, есть возможности экспорта в различные форматы файлов.

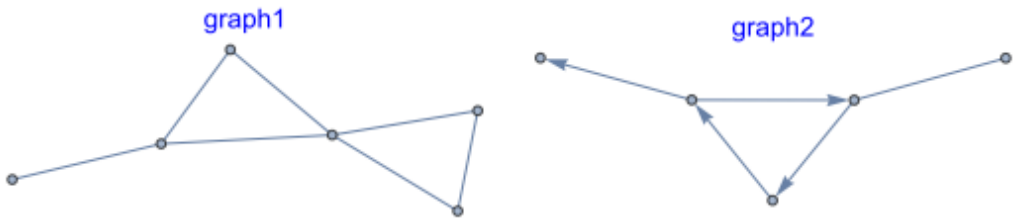
Объем функционала в *Mathematica* по графам, их приложениям громадный – несколько разделов с подробными описаниями и методическими рекомендациями, конкретных статей по функциям и специальным опциям в подсистеме помощи более тысячи, примеров и упражнений – более 50 тысяч. Отметим несколько основных разделов: Программирование графов (guide/GraphProgramming), Построение и представление графов (guide/GraphConstructionAndRepresentation), Графы и матрицы (guide/GraphsAndMatrices), Свойства графов и измерения (guide/GraphPropertiesAndMeasurements), Графы и сети (guide/Graphs AndNetworks).

Графы в *Mathematica* можно формировать различными способами, в частности, из вершин и ребер непосредственно в символьной форме. Они могут быть приняты и использованы, модифицированы из встроенных коллекций теоретических или эмпирических графов (детали ниже); специальные графы могут быть сгенерированы на основе параметрических спецификаций. Случайные графы, формируемые по различным распределениям, позволяют создавать имитируемые интернет-сети или графы цитирования и тестировать алгоритмы. Графы могут быть полностью заданы несколькими типами матриц или импортированы из множества поддерживаемых форматов файлов; сложные смешанные графы также могут быть построены последовательными действиями пользователя путем выполнения операций над имеющимися графами [1].

Примеры построения графов, их простейшая визуализация

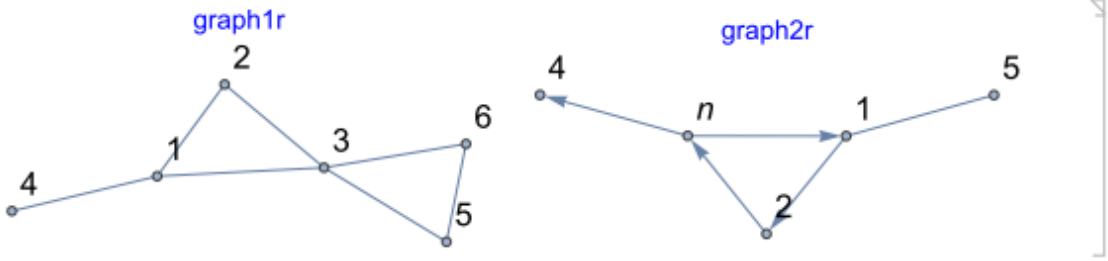
В системе *Mathematica* существует возможность создавать ориентированные и неориентированные графы, проводить над ними множество операций анализа. Для создания, как ориентированного, так и неориентированного графа используется функция ***Graph***. Примеры задаваемых пользователем графов graph1 и graph2, когда их изображения выводятся с установками по умолчанию (нет никаких пользовательских уточнений), показаны в следующих секциях ввода и результата:

```
graph1 = Graph[{1 ↔ 2, 2 ↔ 3, 3 ↔ 1, 1 ↔ 4, 3 ↔ 5, 5 ↔ 6, 6 ↔ 3},
    PlotLabel → Style["graph1", 14, Blue],
    ImageSize → 260];
graph2 = Graph[{1 ↔ 2, 2 ↔ n, n ↔ 1, n ↔ 4, 1 ↔ 5},
    PlotLabel → Style["graph2", 14, Blue],
    ImageSize → 260];

Row[{graph1, graph2}, Spacer[10]]
```



В коде выше использованы символы *Mathematica* ↔ и ↔. При необходимости можно для ненаправленных и направленных ребер записать полные формы ***UndirectedEdge*** и ***DirectedEdge*** или ASCII-символы <->, ->. В примерах graph1r и graph2r так определены ребра 1-2 и 2-n. Также добавлены опции ***VertexLabels*** вывода имен (меток) вершин с уточнениями стиля ***VertexLabelStyle***:

```
graph1r = Graph[{UndirectedEdge[1, 2],
     2 ↔ 3, 3 ↔ 1, 1 ↔ 4, 3 ↔ 5, 5 ↔ 6, 6 ↔ 3},
    VertexLabels → "Name", VertexLabelStyle → {Black, 17},
    PlotLabel → Style["graph1r", 14, Blue],
    ImageSize → 260];
graph2r = Graph[{DirectedEdge[1, 2], 2 -> n, n ↔ 1, n ↔ 4, 1 ↔ 5},
    VertexLabels → "Name", VertexLabelStyle → {Black, 17},
    PlotLabel → Style["graph2r", 14, Blue],
    ImageSize → 260];

Row[{graph1r, graph2r}, Spacer[10]]
```

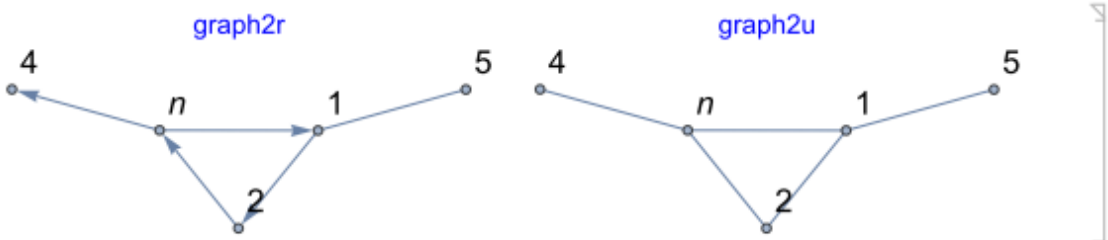


Если граф создан или импортирован, но необходимо изменение, есть много функций, позволяющих преобразовать сам граф (изменить связи, свойства), модифицировать ребра и вершины. Полный перечень возможностей редактирования содержится в системе помощи в разделе Graph Operations and Modifications. Отметим только наиболее часто используемые: ***UndirectedGraph*** (ненаправленный граф) – преобразует направленный граф в ненаправленный, ***DirectedGraph*** (ориентированный граф) – преобразует ненаправленный граф в направленный, ***ReverseGraph*** (обратный орграф) – дает обратный для текущего ориентированного, ***IndexGraph*** (индексный граф) – выполняет замену текущих номеров вершин их индексами; ***EdgeAdd*** (добавить ребро), ***EdgeDelete*** (удалить ребро), ***EdgeContract*** (стянуть ребро), ***VertexAdd*** (добавить вершину), ***VertexDelete*** (удалить вершину), ***VertexReplace*** (заменить вершину), ***VertexContract*** (стянуть вершины в одну).

Несколько примеров.

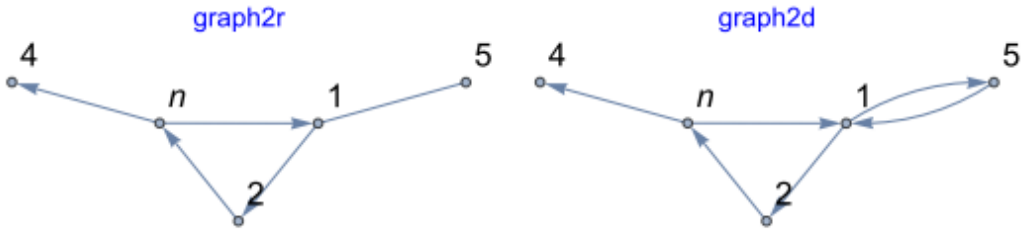
Преобразование исходного графа graph2 в ненаправленный (все ребра не имеют ориентации) иллюстрирует пример graph2u, результат обеспечивает функция ***UndirectedGraph***:

```
graph2u = UndirectedGraph[graph2,
    VertexLabels → "Name", VertexLabelStyle → {Black, 17},
    PlotLabel → Style["graph2u", 14, Blue], ImageSize → 260];

Row[{graph2r, graph2u}, Spacer[10]]
```



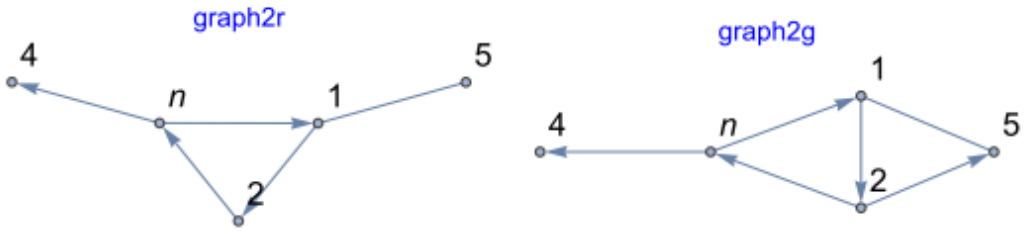
Преобразование графа в ориентированный (все ребра имеют направление) иллюстрирует пример graph2d, когда для графа graph2 единственному ребру 1-5 без указанного направления присвоены оба направления:

```
graph2d = DirectedGraph[graph2,
    VertexLabels → "Name", VertexLabelStyle → {Black, 17},
    PlotLabel → Style["graph2d", 14, Blue], ImageSize → 260];

Row[{graph2r, graph2d}, Spacer[10]]
```



Изменение графа graph2r добавлением ребра 2-5, причем, с указанием направления 2→5 иллюстрирует пример graph2g:

```
graph2g = EdgeAdd[graph2r, 2 → 5];
graph2gN =
  Show[graph2g, PlotLabel → Style["graph2g", 14, Blue]];

Row[{graph2r, graph2gN}, Spacer[10]]
```

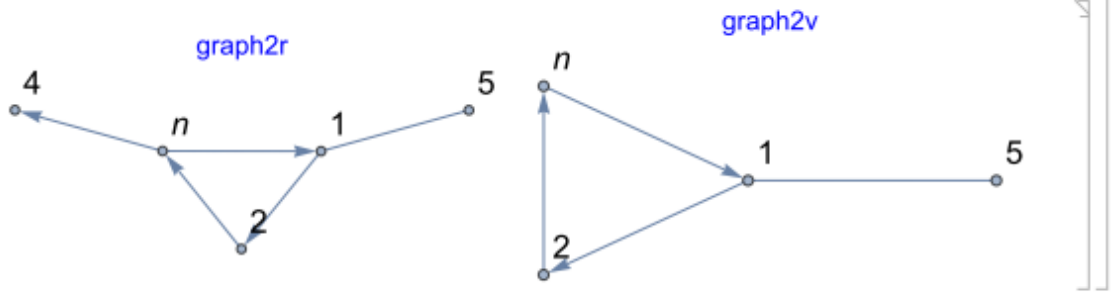


Пример graph2e иллюстрирует изменение, когда для ребра с указанным направлением добавлено дополнительное направление:

```
graph2e = EdgeAdd[graph2r, n ↔ 2];
graph2eN =
  Show[graph2e, PlotLabel → Style["graph2e", 14, Blue]];

Row[{graph2r, graph2eN}, Spacer[10]]
```

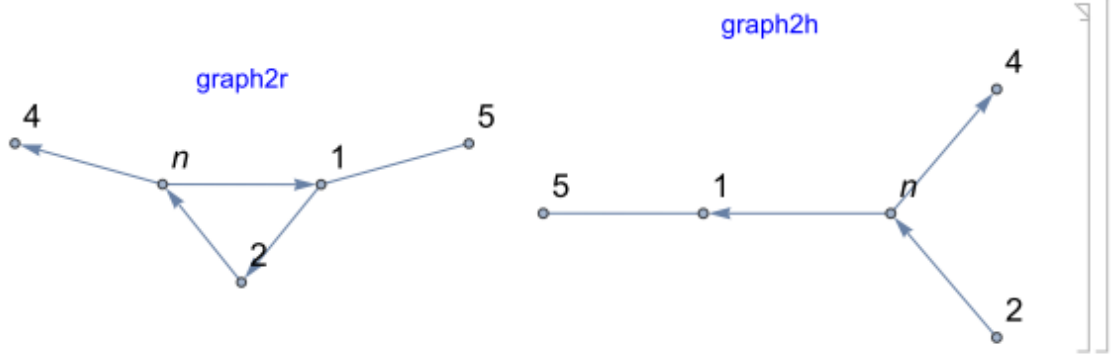
Получение из имеющегося graph2r другого графа graph2v удалением вершины 4 иллюстрирует пример ниже:

```
graph2v = VertexDelete[graph2r, 4];
graph2vN =
  Show[graph2v, PlotLabel → Style["graph2v", 14, Blue]];
Row[{graph2r, graph2vN}, Spacer[10]]
```



Изменение графа graph2r удалением ребра 1-2 иллюстрирует пример graph2h:

```
graph2h = EdgeDelete[graph2r, 1 ↔ 2];
graph2hN =
  Show[graph2h, PlotLabel → Style["graph2h", 14, Blue]];
Row[{graph2r, graph2hN}, Spacer[10]]
```



Функции Wolfram *Mathematica* автоматического построения графов с заданными свойствами описаны ниже.

Показатели, комбинаторные свойства графов

Многие алгоритмы и процедуры требуют наличия графов с определенными свойствами. Это могут быть базовые свойства, такие как неориентированность, или более глубокие свойства топологии, такие как связность или ацикличность. В некоторых областях ключевой проблемой является определение того, являются ли два графа одинаковыми, если заменить имена вершин, т.е. проверить, являются ли они изоморфными [2].

Язык Wolfram поддерживает широкий спектр показателей, характеризующих графы, от простых показателей, таких как количество вершин и ребер, которые определяют размер и разреженность графа, до степеней вершин, которые показывают, насколько локально хорошо

связана каждая вершина. Другие показатели включают геодезические расстояния на графе или показатели центральности, которые дают представление о том, насколько центральной в общем графе является каждая вершина; например, PageRank и HITS – показатели, используемые для упорядочивания важности веб-страницы по результатам, возвращаемым поисковой системой.

В разделах [2] можно ознакомиться с описаниями и примерами использования запросов ***Basic Properties, Structural Properties, Graph Isomorphism, Basic Measures, Distance Measures, Connectivity Measures, Centrality Measures, Reciprocity and Transitivity, Homophily, Assortative Mixing, Similarity***.

Используя графы или другие объекты (экспортированные, сформированные по результатам вычислений), в системе *Mathematica* можно проверять/уточнять их свойства.

Включить в текущее пособие полное изложение поддерживаемых, проверяемых в Wolfram *Mathematica* свойств графов не представляется возможным из-за объема соответствующих материалов, которые подробно описаны в документации системы и сопровождаются примерами с уровнями сложности начинающих, уверенных, продвинутых пользователей. В подтверждение перечислим только приведенные в [3] категории свойств графов: основные, индекса, локального графа, глобального графа, связанные с кликой, связанные с покрытием, связанные с набором независимые, связанные с подключаемостью графа, связанные с отображением графа, связанные с раскраской, связанные с выводом типа списка аннотируемые, дающие представляющие графовые многочлены чистые функции.

Приведем код получения списка классов графов *Mathematica*, числа элементов списка и результаты выполнения:

```
GraphData["Classes"]
Length[%]
```

```
{Acyclic, AlmostHamiltonian, AlternatingGroup, Andrasfai,
 Antelope, Antiprism, Apex, Apollonian, Archimedean,
 ArchimedeanDual, ArcTransitive, Arrangement, Asymmetric,
 BananaTree, Barbell, Beineke, Bicolorable, Biconnected,
 Bicubic, Bipartite, BipartiteKneser, Bishop, BlackBishop,
 Book, Bouwer, Bridged, Bridgeless, Cactus, Cage, Caterpillar,
 Caveman, Cayley, Centipede, Chang, Chordal, Chordless,
 ChromaticallyNonunique, ChromaticallyUnique, Circulant, Class1,
 Class2, ClawFree, CocktailParty, Complete, CompleteBipartite,
```

```
CompletelyRegular, CompleteTree, CompleteTripartite, Cone,
 Conference, Connected, CriticalNonplanar, CrossedPrism,
 Crown, CubeConnectedCycle, Cubic, Cycle, Cyclic, Cyclotomic,
DeterminedByResistance, DeterminedBySpectrum, Disconnected,
 DistanceRegular, DistanceTransitive, Doob, DutchWindmill,
 EdgeTransitive, Empty, Eulerian, Fan, FibonacciCube,
 Firecracker, Fiveleaper, FoldedCube, Forest, Fullerene,
 Fusene, Gear, GeneralizedPetersen, GeneralizedPolygon,
 Grid, Haar, Hadamard, Halin, HalvedCube, HamiltonConnected,
 HamiltonDecomposable, Hamiltonian, HamiltonLaceable, Hamming,
 Hanoi, Harary, Helm, HoneycombToroidal, HStarConnected,
 Hypercube, Hypohamiltonian, Hypotraceable, Identity,
 IGraph, Imperfect, Incidence, Integral, Johnson, Keller,
 KempeCounterexample, King, Kneser, Knight, Kuratowski, Ladder,
 LadderRung, LCF, Line, Lobster, Local, LocallyPetersen,
 Lollipop, Matchstick, MaximallyNonhamiltonian, Median,
 MengerSponge, Metelsky, MoebiusLadder, MongolianTent,
 Moore, Mycielski, Noncayley, Nonempty, Noneulerian,
 Nonhamiltonian, Nonplanar, Nonsimple, NoPerfectMatching,
 NotDeterminedByResistance, NotDeterminedBySpectrum,
 Nuciferous, Octic, Odd, Ore, Paley, Pan, Pancyclic, Path,
 Paulus, Perfect, PerfectMatching, PermutationStar, Planar,
 Platonic, Polyhedral, Polyiamond, Polyomino, Prism,
 Pseudoforest, Pseudotree, Quartic, Queen, Quintic, Regular,
 RegularPolychoron, Rook, RookComplement, SelfComplementary,
 SelfDual, Semisymmetric, Septic, Sextic, SierpinskiCarpet,
 SierpinskiSieve, SierpinskiTetrahedron, Simple, Snark,
 Spider, SquareFree, StackedBook, StackedPrism, Star,
 StronglyPerfect, StronglyRegular, Sun, Sunlet, Symmetric,
 Tadpole, Taylor, Tetrahedral, Toroidal, TorusGrid, Traceable,
 Transposition, Tree, TriangleFree, Triangular, TriangularGrid,
 TriangularHoneycombAcuteKnight, TriangularHoneycombBishop,
 TriangularHoneycombKing, TriangularHoneycombObtuseKnight,
 TriangularHoneycombQueen, TriangularHoneycombRook,
 Triangulated, Tripod, Turan, TwoRegular, Unicyclic,
 UnitDistance, Untraceable, VertexTransitive,
 WeaklyPerfect, WeaklyRegular, Web, WellCovered, Wheel,
 WhiteBishop, Windmill, Wreath, ZeroSymmetric, ZeroTwo}

218
```

Видим по результату в секции выше, что только классов 218.

Содержание любого класса из приведенного списка дает выполнение секции типа:

```
GraphData["Platonic"]
```

```
{CubicalGraph, DodecahedralGraph,
 IcosahedralGraph, OctahedralGraph, TetrahedralGraph}
```

```
GraphData["Tetrahedral"]
```

```
{{Tetrahedral, 6}, {Tetrahedral, 7},
 {Tetrahedral, 8}, {Tetrahedral, 9}, {Tetrahedral, 10}}
```

```
GraphData["SierpinskiCarpet"]
```

```
{{Cycle, 8}, {SierpinskiCarpet, 2},
 {SierpinskiCarpet, 3}, {SierpinskiCarpet, 4}}
```

При этом в отдельных классах число графов может составлять несколько тысяч. В примере ниже в списке более 6 тысяч:

```
GraphData["Hamiltonian"]
Length[%]
```

```
{{5, 31}, {6, 86}, {6, 92}, {6, 95},
 {6, 96}, {6, 108}, {6, 109}, {6, 110},
 ··· 5992 ···, {ZeroTwoNonbipartite, {7, 49}},
 {ZeroTwoNonbipartite, {7, 50}},
 {ZeroTwoNonbipartite, {7, 52}},
 {ZeroTwoNonbipartite, {7, 53}},
 {ZeroTwoNonbipartite, {7, 54}},
 {ZeroTwoNonbipartite, {7, 55}},
 {ZeroTwoNonbipartite, {7, 56}}}

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

```
6007
```

В подобных результатах иногда выводится большое количество элементов типа, как в секции выше – {n,k}. Поэтому целесообразно прежде просмотреть число элементов списка (ниже в примере Snark выполнение Length[%] дает 367), а следующим шагом выводить только те, которые содержат имена элементов. Ниже – получение числа элементов класса Snark:

```
GraphData["Snark"];
Length[%]
```

```
367
```

В следующей секции записана функция Cases с директивой _String, вывод элементов, имеющих конкретные имена:

```
Cases[GraphData["Snark"], _String]
```

```
{CelminsSwartSnark1, CelminsSwartSnark2, CrossingNumberGraph3E,
 DoubleStarSnark, FlowerSnarkJ5, FlowerSnarkJ7,
 GoldbergSnark3, GoldbergSnark5, GoldbergSnark7,
 LoupekinesSnark1, LoupekinesSnark2, PetersenGraph,
 SzekeresSnark, TietzeGraph, TriangleReplacedPetersenGraph,
 WatkinsSnark, ZamfirescuGraph36}
```

Поиск и извлечение конкретных элементов из GraphData возможно путем перечисления нужных свойств. Например, можно получить список элементов Гамильтонов граф с конкретным числом вершин (ниже после списка с результатами выборки дополнительно выводится число элементов):

```
GraphData["Hamiltonian", 6]
Length[%]
```

```
{{6, 86}, {6, 92}, {6, 95}, {6, 96}, {6, 108}, {6, 109}, {6, 110},
 {6, 111}, {6, 112}, {6, 114}, {6, 124}, {6, 125}, {6, 128},
 {6, 129}, {6, 130}, {6, 135}, {6, 136}, {6, 137}, {6, 138},
 {6, 139}, {6, 140}, {6, 141}, {6, 144}, {6, 148}, {6, 149},
 {6, 150}, {BiggestLittlePolygon, 6}, {BlackBishop, {3, 4}},
 {Complete, 6}, {CompleteTripartite, {1, 2, 3}}, {Cone, {3, 3}},
 {Cycle, 6}, DominoGraph, {Fan, {1, 5}}, {Fan, {2, 4}},
 {HamiltonLaceable, {6, 1}}, {Hexahedral, 3}, {Hexahedral, 4},
 {Hexahedral, 5}, {King, {2, 3}}, OctahedralGraph,
 {Polyiamond, {4, 1}}, {Prism, 3}, {Queen, {2, 3}},
 {TriangularGrid, 2}, {Turan, {6, 5}}, UtilityGraph, {Wheel, 6}}

48
```

Также можно уточнить список дополнением запроса с указанием. например, основного класса, в которому, следуя терминологии системы, относятся: Bipartite, Nonplanar, Nonsimple, Planar, Simple, Tree.

В примере ниже основной запрос Hamiltonian уточняется дополнением – Планарный граф (Planar):

```
GraphData[{"Hamiltonian", "Planar"}, 6]
Length[%]
```

```
{{6, 86}, {6, 92}, {6, 95}, {6, 96}, {6, 108},
 {6, 109}, {6, 110}, {6, 111}, {6, 112}, {6, 124},
 {6, 125}, {6, 128}, {6, 129}, {6, 130}, {6, 135},
 {6, 136}, {6, 137}, {6, 139}, {6, 140}, {6, 141},
 {6, 144}, {BlackBishop, {3, 4}}, {Cycle, 6}, DominoGraph,
 {Fan, {1, 5}}, {Fan, {2, 4}}, {HamiltonLaceable, {6, 1}},
 {Hexahedral, 3}, {Hexahedral, 4}, {Hexahedral, 5},
 {King, {2, 3}}, OctahedralGraph, {Polyiamond, {4, 1}},
 {Prism, 3}, {TriangularGrid, 2}, {Wheel, 6}}
```

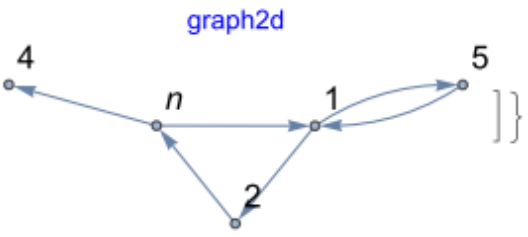
```
36
```

Отметим только несколько наиболее часто используемых команд проверки свойств. Так например, можно запрашивать по имеющемуся/определенному в системе объекту является ли он графом, и конкретно *получать подтверждение свойств*: базовых (GraphQ – граф?, DirectedGraphQ – ориентированный граф?, UndirectedGraphQ – ненаправленный граф?, MultigraphQ – мультиграф?, MixedGraphQ – смешанный граф?, EmptyGraphQ – пустой граф?, WeightedGraphQ – граф с весами?, CompleteGraphQ – полный граф?), структурных (SimpleGraphQ – простой граф?, AcyclicGraphQ – ацикличный граф?, LoopFreeGraphQ – граф без петель?, BipartiteGraphQ – двудольный граф?, ConnectedGraphQ – связный граф?, EulerianGraphQ – эйлеров граф?, HamiltonianGraphQ – гамильтонов граф?, PathGraphQ – граф маршрута?, PlanarGraphQ – планарный граф?, TreeGraphQ – граф дерево?), изоморфизма (IsomorphicGraphQ – изоморфные графы?).

Простейшие примеры.

Относительно базовых свойств графов. Вывод списков вершин и дуг обеспечивают ***VertexList*** и ***EdgeList***. При этом можно в качестве аргумента использовать идентификатор (имя) графа или его изображение:

```
{VertexList[graph2d], EdgeList[
```




```
]}
```

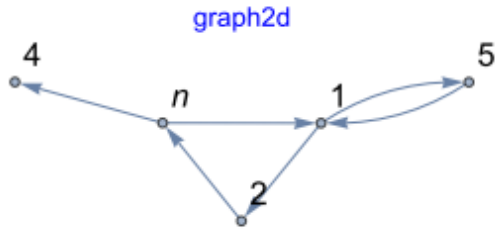
```
{{1, 2, n, 4, 5}, {1 ↔ 2, 2 ↔ n, n ↔ 1, n ↔ 4, 1 ↔ 5, 5 ↔ 1}}
```

Свойства графов категорий индексы и списки можно выводить, используя: ***VertexIndex*** – индекс вершины, ***EdgeIndex*** – индекс ребра, ***AdjacencyList*** – список смежных вершин, ***IncidenceList*** – список инциденций, ***EdgeRules*** – список рёбер в виде правил. (Детали, включая свойства

EdgeWeight, VertexWeight, EdgeCapacity, VertexCapacity, изложены отдельно).

В секции ниже использованы несколько из перечисленных выше функций Wolfram *Mathematica*, в частности, для определенного выше графа graph2d выводится индекс вершины с именем n, подтверждается имя (VertexList[graph2d][[%]] обеспечивает вывод имени вершины с индексом предыдущего действия); выводятся: название 3-го ребра списка (EdgeList[graph2d][[3]]), список рёбер в виде правил, список инциденций:

```
graph2d
VertexIndex[graph2d, n]
VertexList[graph2d][[%]]
EdgeList[graph2d][[3]]
EdgeRules[graph2d]
IncidenceList[graph2d, n]
```




```
3
n
n ↔ 1
{1 → 2, 2 → n, n → 1, n → 4, 1 → 5, 5 → 1}
{2 ↔ n, n ↔ 1, n ↔ 4}
```

Проверка и подтверждение свойств ненаправленный, ориентированный, без петель, ацикличный графа graph2d выполняются функциями секции ниже:

```
{UndirectedGraphQ[graph2d], DirectedGraphQ[graph2d],
 LoopFreeGraphQ[graph2d], AcyclicGraphQ[graph2d]}
```

```
{False, True, True, False}
```

Свойства планарный, связный, простой, двудольный графа graph2d проверяются и подтверждаются в результате выполнения функций секции:

```
{PlanarGraphQ[graph2d], ConnectedGraphQ[graph2d],
 SimpleGraphQ[graph2d], BipartiteGraphQ[graph2d]}
```

```
{True, False, True, False}
```

Графы и матрицы. Матричные представления обеспечивают связь с алгоритмами вычисления графов, основанными на линейной алгебре. Описание графов с помощью матриц часто используется до сих пор, и в некоторых областях оно является единственным (предпочтительным) способом. Матрицы смежности представляют смежные вершины, а матрица инцидентности – отношения вершин к ребрам. Обе способны представлять неориентированные и ориентированные графы.

Приведем несколько примеров использования функций ***AdjacencyMatrix*** – матрица смежности, ***AdjacencyGraph*** – граф по матрице смежности.

В следующей секции формируется матрица смежности – разрежённый массив, который выводится в виде матричной формы:

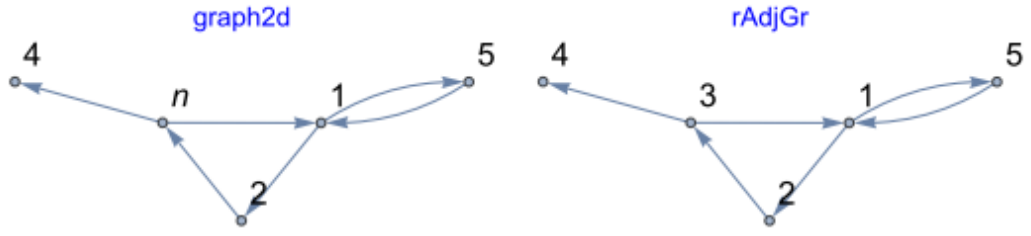
```
rAdjMatr = AdjacencyMatrix[graph2d]
MatrixForm[rAdjMatr]
```

```
SparseArray[ <Specified elements: 6
Dimensions: {5, 5}
Default: 0
Density: 0.24
Elements:
{1, 2} → 1
{1, 5} → 1
{2, 3} → 1
{3, 1} → 1
⋮ > ]
```

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

Построение графа по матрице смежности rAdjMatr обеспечивает ***AdjacencyGraph***:

```
rAdjGr = AdjacencyGraph[rAdjMatr,
    VertexLabels → "Name", VertexLabelStyle → {Black, 17},
    PlotLabel → Style["rAdjGr", 14, Blue],
    ImageSize → 260];
Row[{graph2d, rAdjGr}, Spacer[10]]
```



Заметим, что имя 3-ей вершины в результате проделанного отличается от исходного, но других отличий графов graph2d и rAdjGr нет, что подтверждает проверка запросом ***IsomorphicGraphQ***:

```
IsomorphicGraphQ[graph2d, rAdjGr]
```

```
True
```

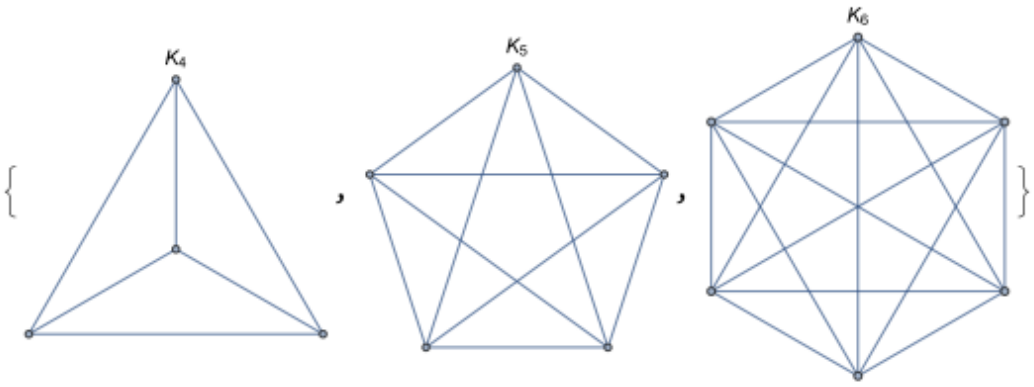
Простейшие функции построения графов

Язык Wolfram предоставляет много функций для создания новых графов на основе старых и генерирования графов с задаваемыми свойствами. Граф с требуемыми свойствами, как правило, можно построить, отталкиваясь от имеющегося графа. Новый граф может быть частью более крупного, или результатом последовательных изменений исходного путем удаления или добавления элементов. Также новые графы в *Mathematica* могут быть построены путем объединения нескольких графов с использованием логических операций.

Примеры создания графов конкретных типов.

Полный граф (простой неориентированный граф, в котором каждая пара различных вершин смежна) с заданным числом вершин формируется в системе функцией ***CompleteGraph***. Код следующей секции обеспечивает формирование трех графов с 4, 5 и 6 вершинами:

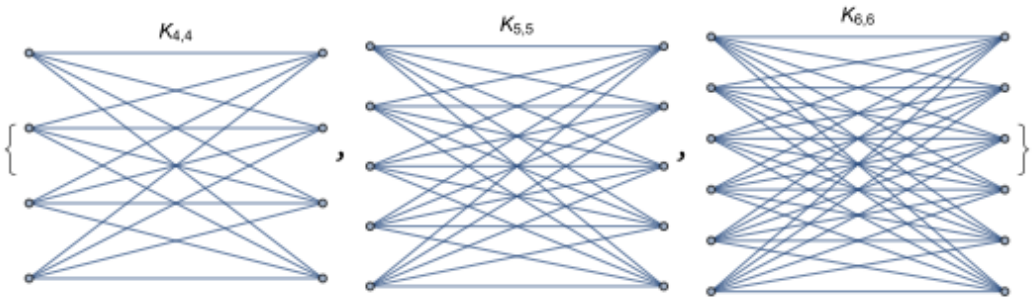
```
Table[CompleteGraph[i,
  PlotLabel → Subscript[K, i], ImageSize → 165], {i, 4, 6}]
```



В приведенном варианте вывода в заголовках графов применена нотация Subscript – с нижним индексом.

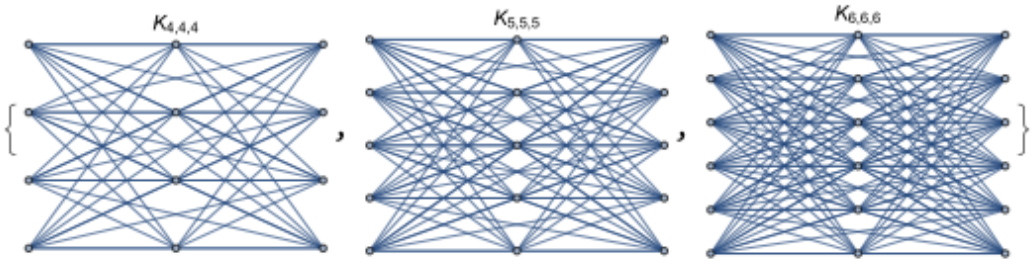
Двудольный граф или биграф – граф, множество вершин которого можно разбить на две части таким образом, что каждое ребро графа соединяет какую-то вершину из одной части с какой-то вершиной другой части, то есть не существует ребра, соединяющего две вершины из одной и той же части. Двудольные графы $K_{i,i}$:

```
Table[CompleteGraph[{i, i},
  PlotLabel → Ki,i, ImageSize → 165], {i, 4, 6}]
```



Полный k-дольный граф – граф, множество вершин которого можно разбить на k независимых множеств, k-дольный – такой граф, что любые две вершины, входящие в разные доли, смежны. Пример $K_{i,i,i}$:

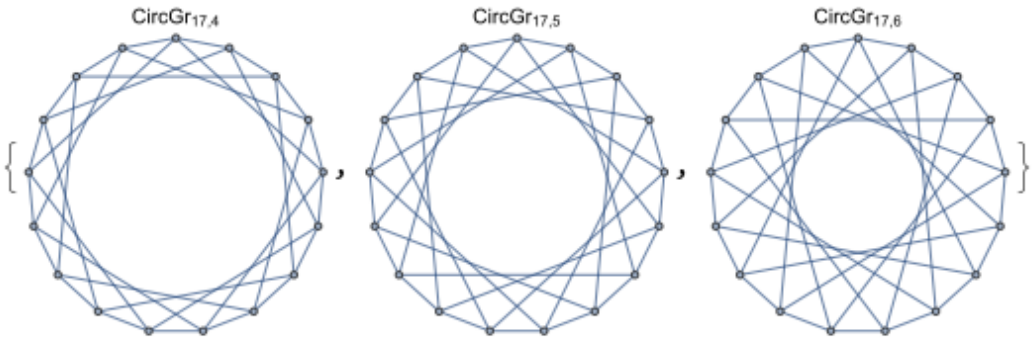
```
Table[CompleteGraph[{i, i, i},
  PlotLabel → Ki,i,i, ImageSize → 165], {i, 4, 6}]
```



Граф-цикл с задаваемым числом вершин, соединённых замкнутой цепью, формируется в системе функцией ***CycleGraph***, можно посмотреть вариант Table[CycleGraph[i,PlotLabel→Subscript[K,i],ImageSize→165],{i,4,6}].

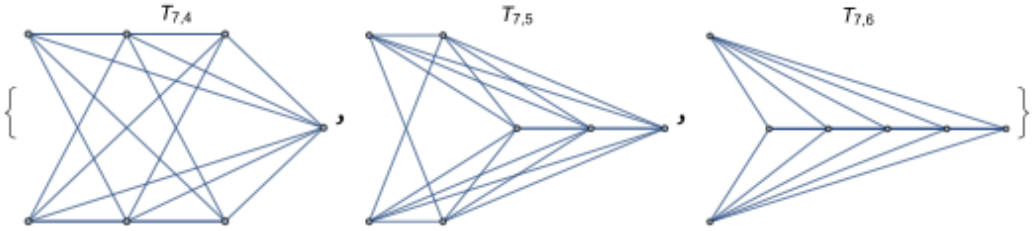
Циркулянтный граф – неориентированный граф, имеющий циклическую группу симметрий, которая включает симметрию, переводящую любую вершину в любую другую вершину. Формируется в *Mathematica* функцией ***CirculantGraph***:

```
Table[CirculantGraph[17, {1, k},
  PlotLabel → Subscript[CircGr, 17, k],
  ImageSize → 165], {k, {4, 5, 6}}]
```



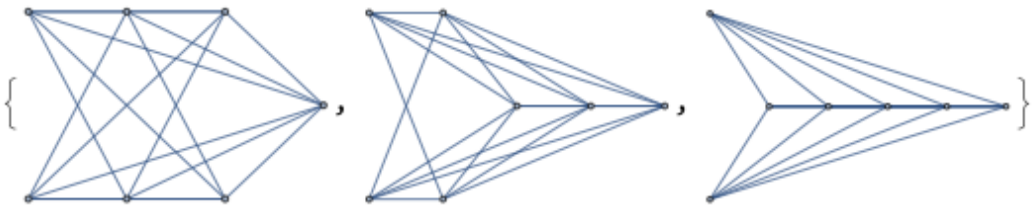
Граф Турана $T(n,k)$ – граф, образованный путем разбиения набора из n вершин на k подмножеств с максимально равными размерами и соединения двух вершин ребром тогда и только тогда, когда они принадлежат разным подмножествам. Формируется в системе функцией ***TuranGraph***:

```
Table[
 TuranGraph[7, i, PlotLabel -> T7,i, ImageSize → 165], {i, 4, 6}]
```



Граф Турана $T(n,k)$ эквивалентен полному k-дольному графу (полный k-дольный граф – это многодольный граф, такой, что любые две вершины, входящие в разные доли, смежны), подтверждение для примера 7 вершин дано визуализацией ниже:

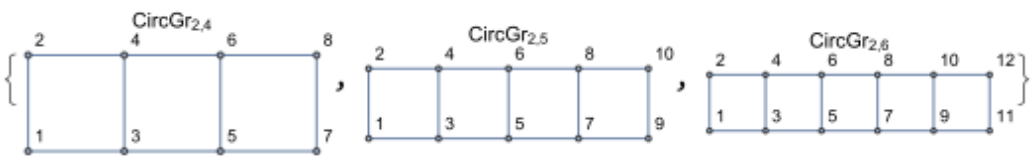
```
{CompleteGraph[{2, 2, 2, 1}, ImageSize → 165],
 CompleteGraph[{2, 2, 1, 1, 1}, ImageSize → 165],
 CompleteGraph[{2, 1, 1, 1, 1, 1}, ImageSize → 165]}
```



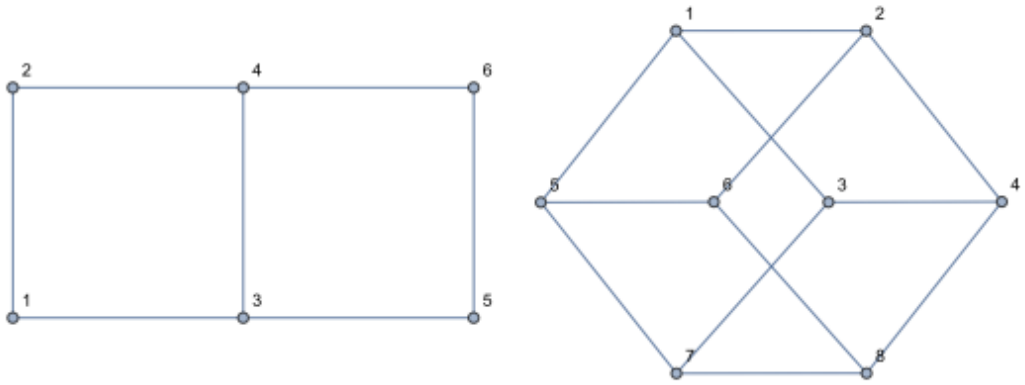
Решётчатый граф (ячеистый, сеточный) – граф, рисунок которого, вложенный в некоторое евклидово пространство образует правильную мозаику, его вершины располагаются в узлах многомерной целочисленной решетки, размерность равна количеству циклов в гнезде. ***GridGraph***[{m,n}] строит сеть (набор) графов $m \times n$ вершин $G_{m,n}$, ***GridGraph***[$\{n_1, n_2, \ldots, n_k\}$] строит сеть (набор) графов $n_1 \times n_2 \times \cdots \times n_k$ вершин $G_{n_1,n_2,\ldots,n_k}$.

Примеры:

```
Table[GridGraph[{2, i}, VertexLabels → "Name",
  PlotLabel → Subscript[CircGr, 2, i], ImageSize → 165], {i, 4, 6}]
```



```
Row[{
  GridGraph[{2, 3},
   VertexLabels → "Name", ImageSize → 260],
  GridGraph[{2, 2, 2},
   VertexLabels → "Name", ImageSize → 260]
 }, Spacer[10]]
```



Повторим отмеченное выше – библиотека графов системы Wolfram *Mathematica* включает все известные именованные графы, любой из них можно использовать для модификаций. Далее приведены соответствующие примеры, причем, чтобы модификации были понятными, изложение дается после описания настроек вывода и оформления.

Примеры оформления графов. Укладка

Для продуктивной работы с графами важно визуализировать их наиболее наглядным образом, особенно это важно в случае больших графов. В системе *Mathematica* имеется ряд встроенных функций рисования графов на плоскости или в трехмерном пространстве, когда расположение вершин и форма ребер вычисляется так, чтобы оптимизировать тот или иной "эстетический" функционал.

В иллюстрациях ниже используются приведенные выше, а также графы из [4]. Варианты оформления даются в следующей последовательности: укладка графов, оформление вершин, подписи ребер, вывод значений весов.

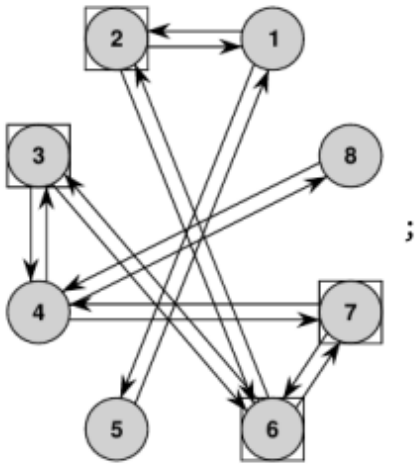
Следуя терминологии математической энциклопедии под укладкой графа понимают отображение вершин и ребер графа соответственно в точки и непрерывные кривые некоторого пространства такое, что вершины, инцидентные ребру, отображаются в концы кривой, соответствующей этому ребру. Правильной называется укладка, при которой разным вершинам соответствуют различные точки, а кривые, соответствующие ребрам

(исключая их концевые точки), не проходят через точки, соответствующие вершинам, и не пересекаются. Любой граф допускает правильную укладку в трехмерное пространство. Граф, допускающий правильную укладку на плоскости, называется плоским.

Wolfram *Mathematica* предлагает пользователям типовые схемы укладки, а также возможности индивидуального дизайна, что предполагает использование для настройки вида графа следующих способов: размещения вершин, прочерчивания ребер, размещения связанных компонент вершин – компоновка упаковки.

Начальную серию примеров приведем для графа из [4] leftPLA1:

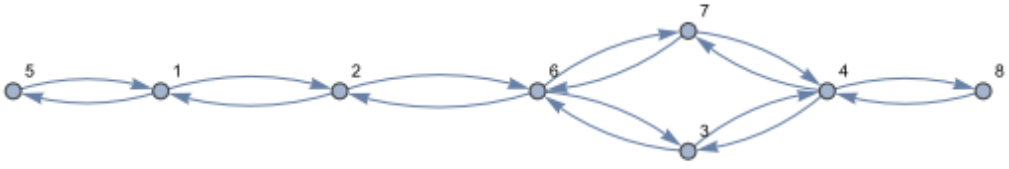
```
leftPLA1 =
```




```
;
```

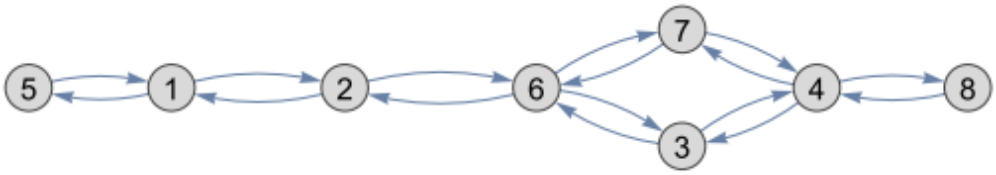
Построение такого графа, его представление без задания опций оформления, когда выводятся только номера вершин, реализовано в секциях ниже. Простое перечисление вершин и ребер, вывод с установками по умолчанию дают результат:

```
lstGrPLA = {1 ↔ 2, 2 ↔ 1, 1 ↔ 5, 5 ↔ 1, 2 ↔ 6, 6 ↔ 2, 3 ↔ 4, 4 ↔ 3,
    3 ↔ 6, 6 ↔ 3, 4 ↔ 7, 7 ↔ 4, 4 ↔ 8, 8 ↔ 4, 6 ↔ 7, 7 ↔ 6};
graphT0 = Graph[lstGrPLA, VertexLabels → "Name", ImageSize → 540]
```



В следующих иллюстрациях приведены несколько типовых вариантов укладки и, чтобы сохранить дизайн оригинала в части вида вершин, заданы стиль вершин и их номеров подобные leftPLA1. Ниже, в стартовом представлении графа в коде способ укладки не задан, используется вариант по умолчанию:

```
graphT1a = Graph[lstGrPLA,
  VertexStyle → LightGray, VertexSize → 0.4,
  VertexLabels → Placed["Name", Center],
  VertexLabelStyle → Directive[Black, 16],
  ImageSize → 540]
```



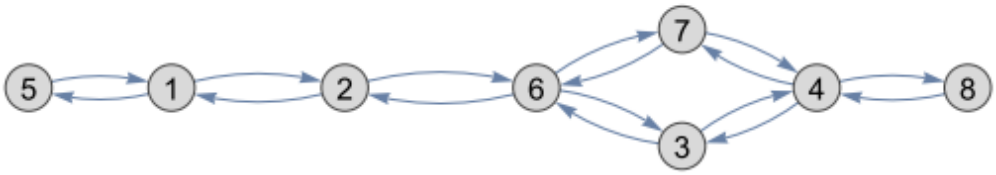
Дополнительно отметим, проверим запросом для графа graphT1a свойства планарный, связный, простой, двудольный, мультиграф, смешанный:

```
{PlanarGraphQ[graphT1a], ConnectedGraphQ[graphT1a],
 SimpleGraphQ[graphT1a], BipartiteGraphQ[graphT1a],
 MultigraphQ[graphT1a], MixedGraphQ[graphT1a]}
```

```
{True, True, True, True, False, False}
```

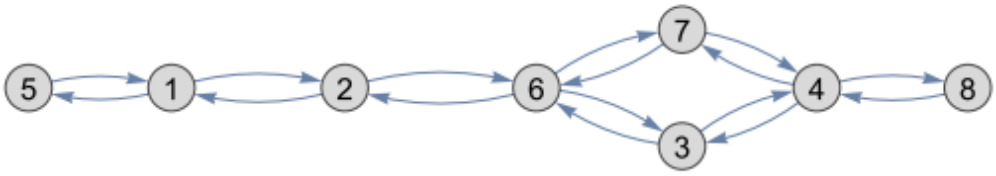
Код и результат выполнения в следующих секциях показывают, что вариант по умолчанию получается, в частности, при использовании опции ***GraphLayout*** с директивой ***SpringElectricalEmbedding***:

```
graphT1b = Graph[lstGrPLA,
  VertexStyle → LightGray, VertexSize → 0.4,
  VertexLabels → Placed["Name", Center],
  VertexLabelStyle → Directive[Black, 16],
  GraphLayout → "SpringElectricalEmbedding",
  ImageSize → 540]
```



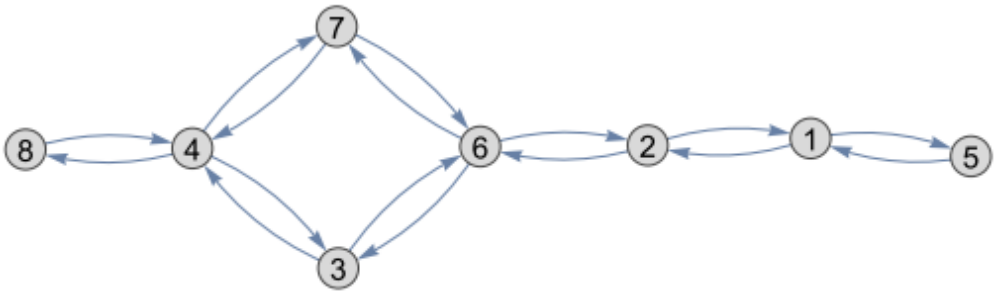
или такой же вид получается при сочетании ***PackingLayout*** и ***ClosestPacking*** (приблизительная ближайшая упаковка слева вверху):

```
graphT1c = Graph[lstGrPLA,
  VertexStyle → LightGray, VertexSize → 0.4,
  VertexLabels → Placed["Name", Center],
  VertexLabelStyle → Directive[Black, 16],
  GraphLayout → {"PackingLayout" → "ClosestPacking"},
  ImageSize → 540]
```



Похожий вариант (как бы перевернутый справа налево) получается при использовании опции ***SpringEmbedding***:

```
graphT1d = Graph[lstGrPLA,
  VertexStyle → LightGray, VertexSize → 0.25,
  VertexLabels → Placed["Name", Center],
  VertexLabelStyle → Directive[Black, 16],
  GraphLayout → "SpringEmbedding",
  ImageSize → 540]
```



В системе помощи *Mathematica* приведены более 50 типовых вариантов укладки. К возможным подходам компоновки графов относят: способ, как расставлять вершины (VertexLayout), как проводить ребра (EdgeLayout), как размещать связанные компоненты вершин (PackingLayout) [5]. Способы расстановки вершин: автоматический (Automatic), без вычисления (None) – воспроизводится представление пользователя. Методы вывода ребер: Связывание разделенных ребер в пакет сегментов (DividedEdgeBundling), объединение ребер в соответствии с иерархической древовидной структурой (HierarchicalEdgeBundling), прямые линии между ребрами (StraightLine). Методы компоновки упаковки (PackingLayout) включают реализации: приблизительная ближайшая упаковка слева вверху (ClosestPacking), приблизительное расстояние до ближайшей упаковки от центра (ClosestPackingCenter), расположение слоями, начиная с верхнего левого угла (Layered), расположение слоями, начиная слева (LayeredLeft), расположение слоями, начиная с самого верха (LayeredTop), упорядочить во вложенной сетке (NestedGrid).

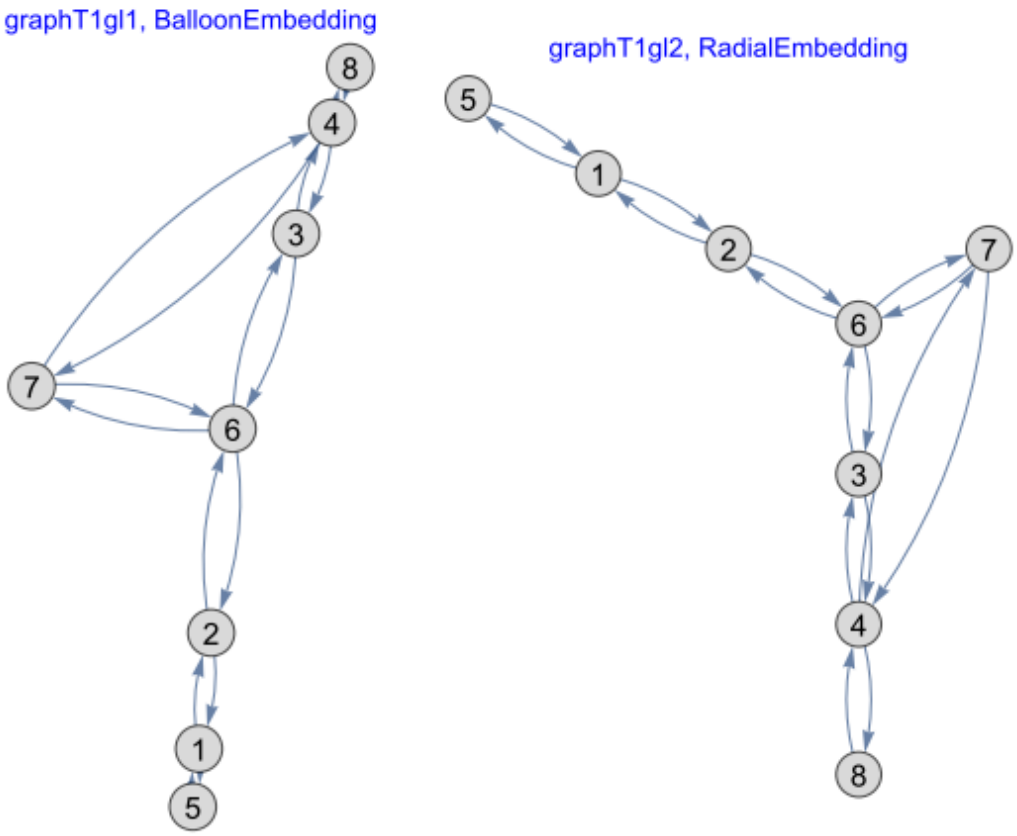
Базовые специальные укладки (виды, встраивания) включают: Двудольное вложение (BipartiteEmbedding) вершин на двух параллельных

прямых, Круговое вложение (CircularEmbedding) вершин на окружности, Круговое многостороннее вложение (CircularMultipartiteEmbedding) вершин на сегментах окружности, Дискретное спиральное встраивание (DiscreteSpiralEmbedding) вершин в дискретную спираль, Сетчатое встраивание (GridEmbedding) вершин в сетку, Линейное встраивание (LinearEmbedding) вершин в линию, Многостороннее встраивание (MultipartiteEmbedding) вершин в несколько параллельных линий, Спиральное встраивание (SpiralEmbedding) вершин в спроецированную в 2D трехмерную спираль, Звездообразное встраивание (StarEmbedding) вершин в окружность с центром.

Выбранные ниже для демонстрации способы укладки не систематизированы, просто иллюстрируется разнообразие.

Приведенные ниже графы graphT1gl1, graphT1gl2 (построены по списку lstGrPLA) отнесены в терминологии *Mathematica* к категориям, которые пояснены в комментариях:

```
(*Possible structured embeddings for layered graphs such as trees
  and directed acyclic graphs include:... BalloonEmbedding -
 vertices on a circle with the center at the parent vertex*)
graphT1gl0 = Graph[lstGrPLA,
   VertexStyle → LightGray,
   VertexLabels → Placed["Name", Center],
   VertexLabelStyle → Directive[Black, 16],
   ImageSize → 200];

(*Possible structured embeddings for layered graphs such as trees
  and directed acyclic graphs include:... RadialEmbedding -
 vertices on a circular segment*)
graphT1gl1 = Graph[graphT1gl0, VertexSize → 0.8,
   GraphLayout → "BalloonEmbedding",
   PlotLabel →
    Style["graphT1gl1, BalloonEmbedding", 14, Blue]];

(*Possible structured embeddings for layered graphs such as trees
  and directed acyclic graphs include:... RadialEmbedding -
 vertices on a circular segment*)
graphT1gl2 = Graph[graphT1gl0, VertexSize → 0.3,
   GraphLayout → "RadialEmbedding",
   PlotLabel → Style["graphT1gl2, RadialEmbedding", 14, Blue],
   ImageSize → 310];

Row[{graphT1gl1, graphT1gl2}, Spacer[20]]
```

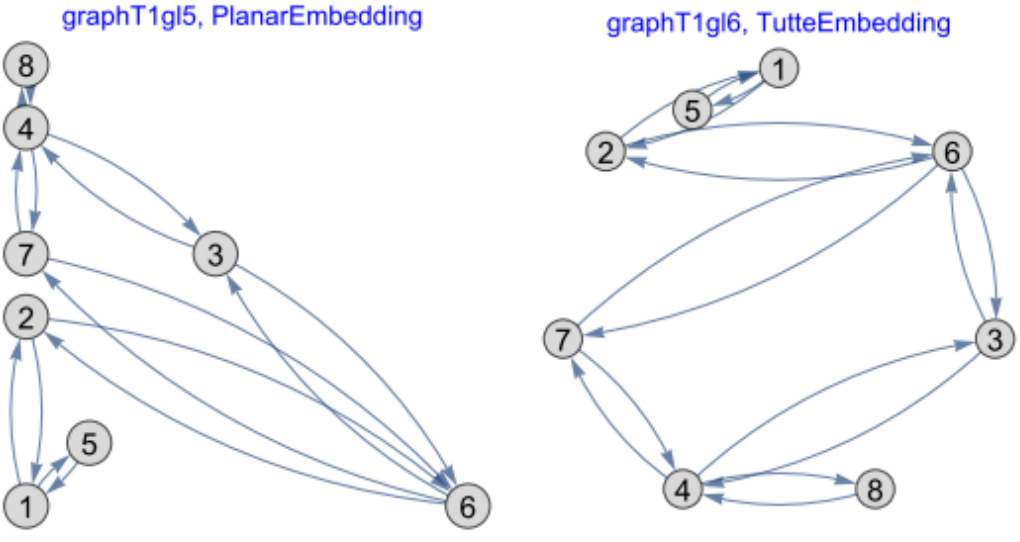


Дополнительно выделены классы: Возможные структурированные укладки многоуровневых графов, таких как деревья и направленные ациклические графы, Возможные оптимизирующие встраивания, которые сводят к минимуму количество, в частности: “Высокоразмерное встраивание”, “Плоское встраивание”, “Спектральное встраивание”, “Встраивание Tutte”.

Графы graphT1gl5 и graphT1gl6 отнесены в системе помощи *Mathematica* к категории возможные оптимизирующие встраивания сводят к минимуму количества ... – уточняется опцией, в текущем примере – ***PlanarEmbedding***:

```
(* Possible optimizing embeddings all minimize a quantity
 and include: HighDimensionalEmbedding,PlanarEmbedding,
SpectralEmbedding, SpringElectricalEmbedding,
SpringEmbedding, TutteEmbedding *)

graphT1gl5 = Graph[graphT1gl0, VertexSize → 0.7,
   GraphLayout → "PlanarEmbedding", (* PlanarEmbedding
    minimize a quantity number of edge crossings *)
   PlotLabel → Style["graphT1gl5, PlanarEmbedding", 14, Blue],
   ImageSize → 270];
```

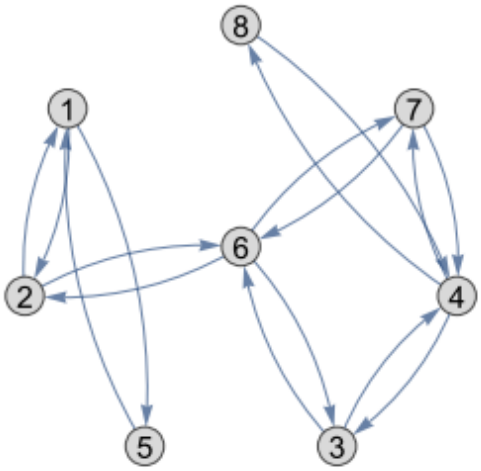
```
graphT1gl6 = Graph[graphT1gl0, VertexSize → 0.4,
    GraphLayout → "TutteEmbedding",
    (* TutteEmbedding minimize a number
     of edge crossings and distance to neighbors *)
    PlotLabel →
     Style["graphT1gl6, TutteEmbedding", 14, Blue],
    ImageSize → 260];

Row[{graphT1gl5, graphT1gl6}, Spacer[10]]
```



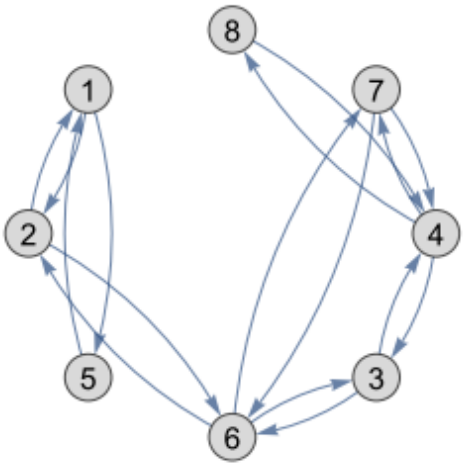
Приведенные ниже графы graphT1gl7 и graphT1gl8 иллюстрируют варианты укладки ***StarEmbedding*** и ***CircularEmbedding***:

```
graphT1gl7 = Graph[graphT1gl0, VertexSize → 0.2,
    GraphLayout → "StarEmbedding"
    (* StarEmbedding -
     vertices on a circle with a center. Другие варианты
      этой группы: BipartiteEmbedding, CircularEmbedding,
     DiscreteSpiralEmbedding, GridEmbedding, LinearEmbedding,
     MultipartiteEmbedding, SpiralEmbedding *),
    PlotLabel →
     Style["graphT1gl7, StarEmbedding", 14, Blue],
    ImageSize → 270];

graphT1gl8 = Graph[graphT1gl0, VertexSize → 0.3,
    GraphLayout → "CircularEmbedding",
    PlotLabel →
     Style["graphT1gl8, CircularEmbedding", 14, Blue],
    (* CircularEmbedding - vertices on a circle *)
    ImageSize → 260];

Row[{graphT1gl7, graphT1gl8}, Spacer[10]]
```

graphT1gl7, StarEmbedding



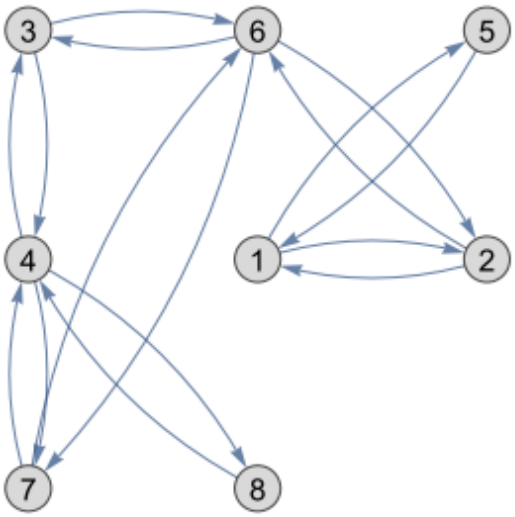
graphT1gl8, CircularEmbedding



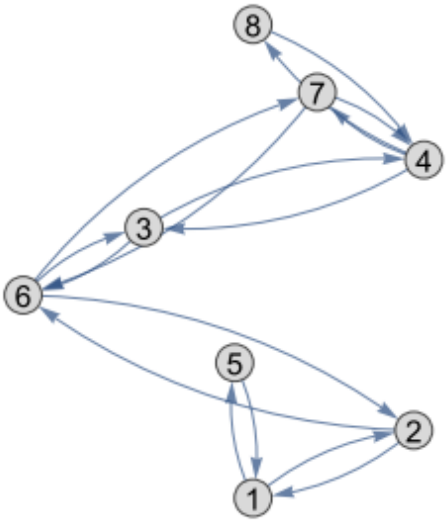
Следующие графы graphT1gl9 и graphT1glA иллюстрируют варианты укладки ***DiscreteSpiralEmbedding*** и ***SpiralEmbedding***:

```
graphT1gl9 = Graph[graphT1gl0, VertexSize → 0.2,
    GraphLayout → "DiscreteSpiralEmbedding", PlotLabel →
     Style["graphT1gl9, DiscreteSpiralEmbedding", 14, Blue],
    (* DiscreteSpiralEmbedding works best for path graphs *)
    ImageSize → 280];
graphT1glA = Graph[graphT1gl0, VertexSize → 0.4,
    GraphLayout → "SpiralEmbedding",
    PlotLabel → Style["graphT1glA, SpiralEmbedding", 14, Blue],
    (* With the setting OptimalOrder, vertices are
     reordered so that they lie on the spiral nicely *)
    ImageSize → 240];

Row[{graphT1gl9, graphT1glA}, Spacer[10]]
```

graphT1gl9, DiscreteSpiralEmbedding

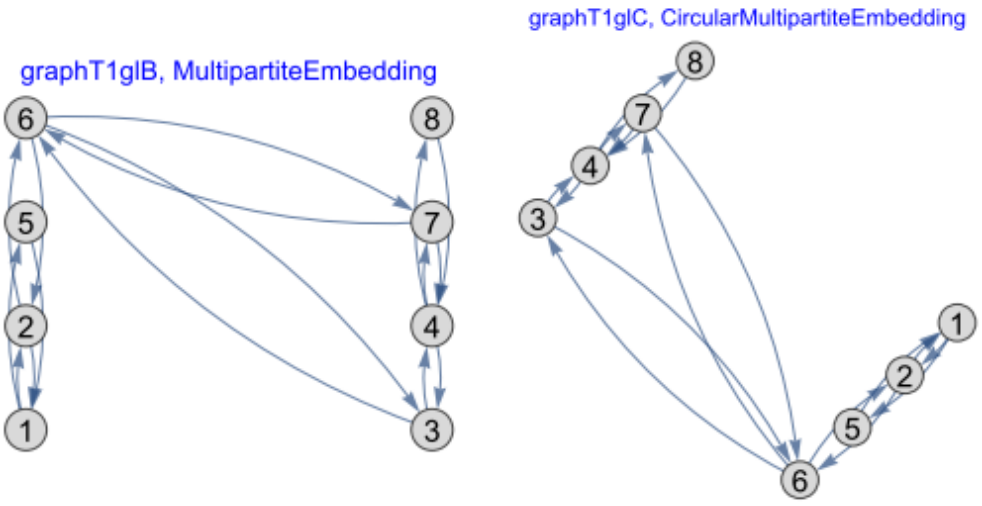


graphT1glA, SpiralEmbedding



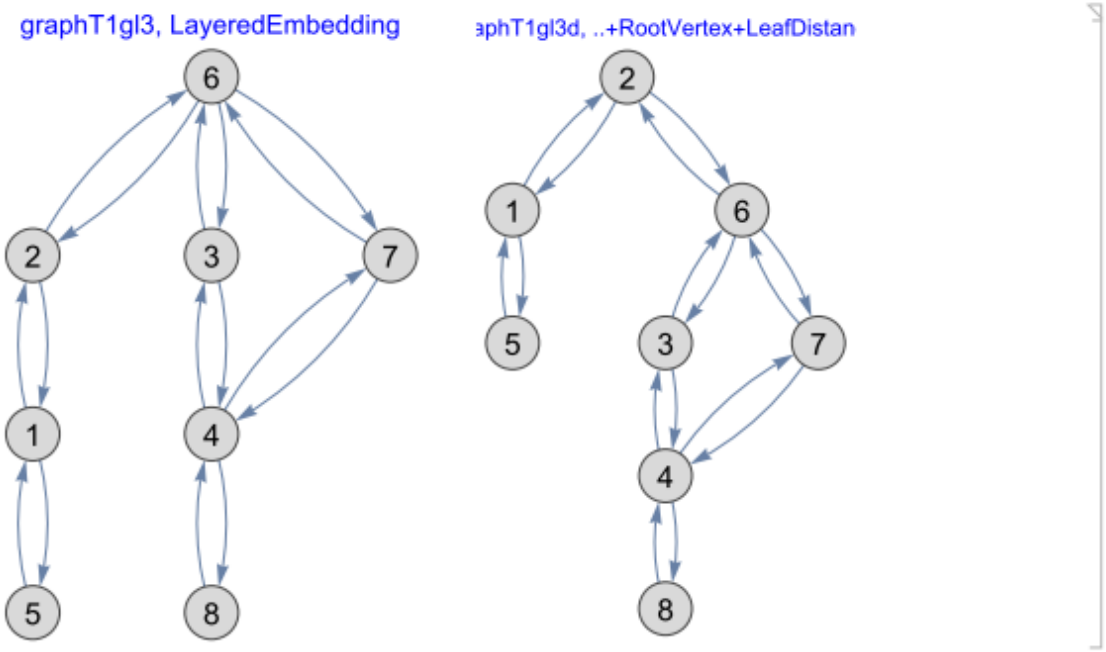
Графы graphT1glB и graphT1glC иллюстрируют варианты укладки ***MultipartiteEmbedding*** и ***CircularMultipartiteEmbedding***:

```
graphT1glB = Graph[graphT1gl0, VertexSize → 0.4,
    GraphLayout → "MultipartiteEmbedding",
    (* Use "VertexPartition" -
     partition to specify a partition of vertices *)
    PlotLabel →
     Style["graphT1glB, MultipartiteEmbedding", 14, Blue],
    ImageSize → 260];
graphT1glC = Graph[graphT1gl0, VertexSize → 0.5,
    GraphLayout → "CircularMultipartiteEmbedding",
    PlotLabel → Style[
      "graphT1glC, CircularMultipartiteEmbedding", 12, Blue],
    ImageSize → 250];

Row[{graphT1glB, graphT1glC}, Spacer[10]]
```



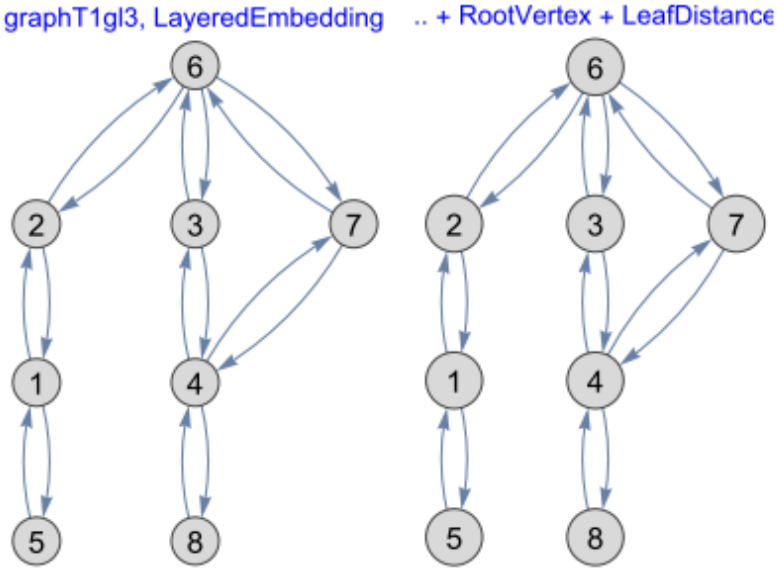
Граф graphT1gl3d иллюстрирует возможность указания ***RootVertex*** вершины (следуя терминологии mathworld.wolfram ”Root Vertex: A special graph vertex that is designated to turn a tree into a rooted tree or a graph into a rooted graph. The root is sometimes also called “eve,” or an “endpoint” /Saaty and Kainen 1986, p. 30/”). Также в примере уточняется расстояние между слоями (***LeafDistance***):

```
graphT1gl3d = Graph[graphT1gl0, VertexSize → 0.4,
    GraphLayout → {"LayeredEmbedding",
      "RootVertex" → 2, "LeafDistance" → 1.15},
    PlotLabel →
     Style["graphT1gl3d, ..+RootVertex+LeafDistance", 12, Blue],
    ImageSize → 240];
Row[{graphT1gl3, graphT1gl3d}, Spacer[20]]
```



Следует отметить, что назначение ***RootVertex*** предполагает, какая вершина будет вверху изображения, и, конечно же, в некоторых случаях вид графа заметно меняется. В примере graphT1gl3e вершиной Root Vertex назначена 6-я (обратите внимание, что такой вариант соответствует выводу, когда опция ***RootVertex*** не прописана – граф graphT1gl3). Также опцией ***LeafDistance*** уточняется расстояние между слоями (по высоте):

```
graphT1gl3e = Graph[graphT1gl0, VertexSize → 0.4,
    GraphLayout →
     {"LayeredEmbedding", "RootVertex" → 6, "LeafDistance" → 0.9},
    PlotLabel → Style[".. + RootVertex + LeafDistance", 14, Blue],
    ImageSize → 240];
Row[{graphT1gl3, graphT1gl3e}, Spacer[10]]
```



Примеры оформления графов. Визуализация вершин, ребер

В текущей серии примеров иллюстрации подготовлены для варианта укладки ***DiscreteSpiralEmbedding***.

В отличие от примера graphT1gl9 в варианте graphT1gl9a изменены цвет фона (***VertexStyle***→Yellow), стиль и цвет номеров вершин (***VertexLabelStyle***→Directive[Red,Italic,16]); в варианте graphT1gl9b для конкретных вершин заданы индивидуальные установки, которые перечисляются ({***Style***[1,Green], ...), причем, для остальных действуют общие установки (***VertexStyle***→Yellow).

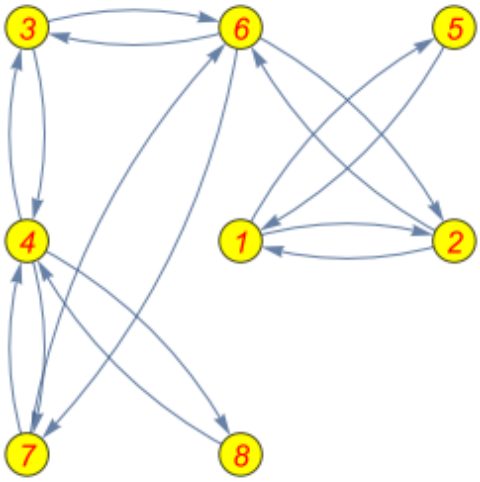
В вариантах graphT1gl9c, graphT1gl9d изменены не только цвета фона (***VertexStyle***). В отличие от примера graphT1gl9b в варианте graphT1gl9c перечисление вершин сделано в другом порядке, что меняет вид укладки. В варианте graphT1gl9d перечисление вершин и цвета фона подобраны так, чтобы идентичными по виду были 1-й и 2-й ряды графа; дополнительно при выводе вершины 8 использование опции ***VertexShapeFunction*** иллюстрирует возможность указания другой формы выводимых вершин (***Square***); в комментарии перечислены более 10 вариантов примитивов вершин из встроенной коллекции (полный перечень выводится функцией ***GraphElementData***[“Vertex”], можно, используя ***Inset***, вставлять собственные растровые или векторные примитивы).

В вариантах graphT1gl9c, graphT1gl9d изменены не только цвета фона (***VertexStyle***). В отличие от примера graphT1gl9b в варианте graphT1gl9c перечисление вершин сделано в другом порядке, что меняет вид укладки. В варианте graphT1gl9d перечисление вершин и цвета фона подобраны так, чтобы идентичными по виду были 1-й и 2-й ряды графа; дополнительно при выводе вершины 8 использование опции ***VertexShapeFunction*** иллюстрирует возможность указания другой формы выводимых вершин (***Square***). В замечании ниже перечислены более 10 вариантов примитивов вершин из встроенной коллекции (полный перечень выводится функцией ***GraphElementData***[“Vertex”], также можно, используя ***Inset***, вставлять собственные растровые или векторные примитивы):

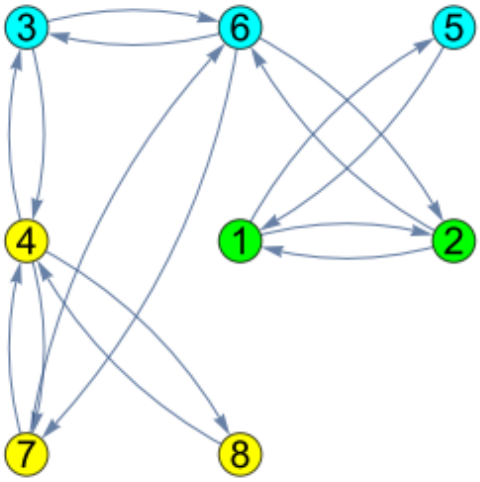
```
graphT1gl9a = Graph[lstGrPLA,
   VertexSize → 0.2, VertexLabels → Placed["Name", Center],
   VertexStyle → Yellow,
   VertexLabelStyle → Directive[Red, Italic, 16],
   GraphLayout → "DiscreteSpiralEmbedding",
   PlotLabel → Style["graphT1gl9a, VertexStyle+..LabelStyle",
     14, Blue], ImageSize → 265];
```

```
graphT1gl9b = Graph[
    {Style[1, Green], Style[2, Green], Style[5, Cyan],
     Style[6, Cyan], Style[3, Cyan]}, lstGrPLA,
    VertexSize → 0.2, VertexLabels → Placed["Name", Center],
    VertexStyle → Yellow,
    VertexLabelStyle → Directive[Black, 20],
    GraphLayout → "DiscreteSpiralEmbedding", PlotLabel →
     Style["graphT1gl9b,..Style(1,2,5,6,3)..LabelStyle",
      13, Blue], ImageSize → 265];
Row[{graphT1gl9a, graphT1gl9b}, Spacer[10]]
```

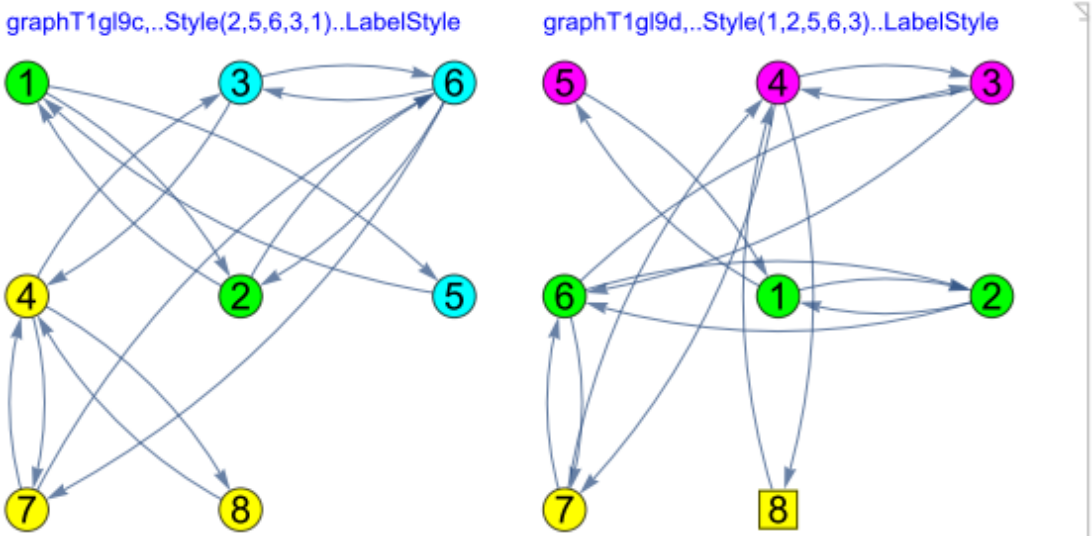
graphT1gl9a, VertexStyle+..LabelStyle



graphT1gl9b,..Style(1,2,5,6,3)..LabelStyle



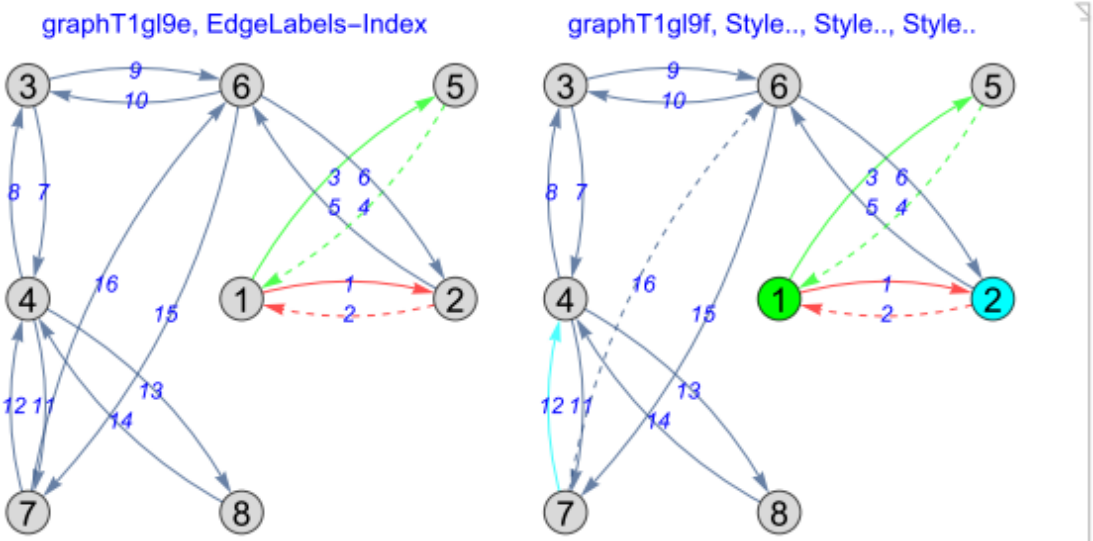

```
graphT1gl9c = Graph[Graph[
     {Style[2, Green], Style[5, Cyan], Style[6, Cyan], Style[3, Cyan]
      Style[1, Green]}, lstGrPLA, VertexSize → 0.2,
     VertexStyle → Yellow, VertexLabels → Placed["Name", Center],
     VertexLabelStyle → Directive[Black, 20],
     GraphLayout → "DiscreteSpiralEmbedding", PlotLabel →
      Style["graphT1gl9c,..Style(2,5,6,3,1)..LabelStyle",
       13, Blue], ImageSize → 265]];
graphT1gl9d = Graph[
    {Style[1, Green], Style[2, Green], Style[3, Magenta],
     Style[4, Magenta], Style[5, Magenta], Style[6, Green]},
    lstGrPLA, VertexSize → 0.2,
    VertexStyle → Yellow, VertexLabels → Placed["Name", Center],
    VertexLabelStyle → Directive[Black, 20],
    VertexShapeFunction → {8 → "Square"},
    GraphLayout → "DiscreteSpiralEmbedding", PlotLabel →
     Style["graphT1gl9d,..Style(1,2,5,6,3)..LabelStyle",
      13, Blue], ImageSize → 265];
Row[{graphT1gl9c, graphT1gl9d}, Spacer[10]]
```



В примерах выше можно использовать и другие примитивы, в частности: “Triangle”, “Square”, “Rectangle”, “Pentagon”, “Hexagon”, “DownTrapezoid”, “UpTrapezoid”, “Parallelogram”, “FiveDown”, “Circle”, “Diamond”, “Star”, “Capsule”

В отличие от примера graphT1gl9 в варианте graphT1gl9e дополнительно выводятся индексы ребер (***EdgeLabels***→”***Index***”), причем, указанным стилем (***EdgeLabelStyle***); перечисляются конкретные ребра 1↔2, 2↔1, 1↔5, 5↔1 и опцией ***EdgeStyle*** задается стиль их отображения. Вариант graphT1gl9f – пример уточнения с использованием функции ***HighlightGraph*** (граф с подкраской) стиля конкретных выводимых вершин (***Style***[1,Green], ***Style***[2,Cyan]) и перечисленных ребер (***Style***[7↔6,Dashed], ***Style***[7↔4,Cyan]):

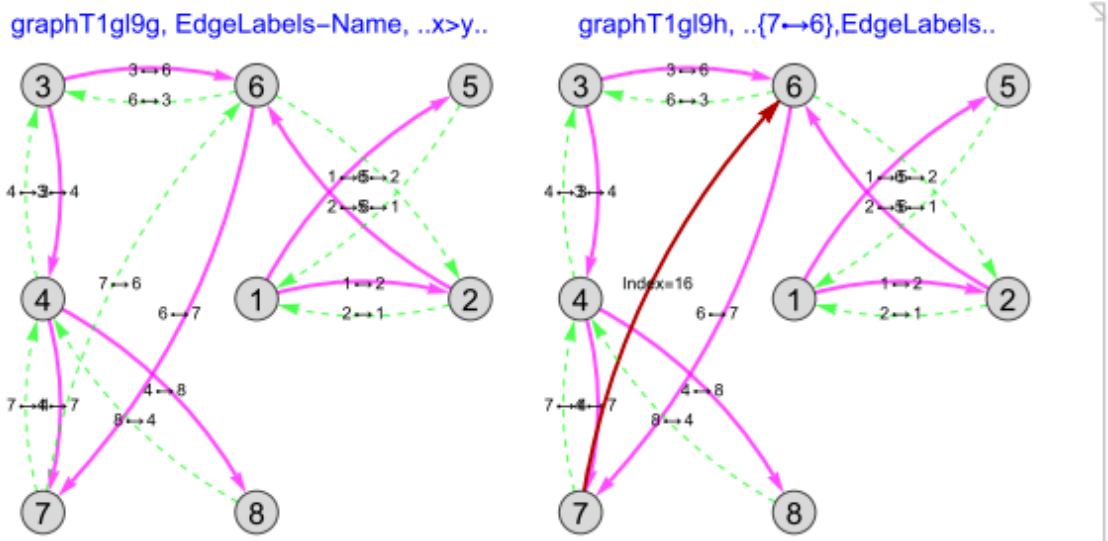
```
graphT1gl9e = Graph[lstGrPLA, VertexSize → 0.2,
    VertexStyle → LightGray, VertexLabels → Placed["Name", Center],
    VertexLabelStyle → Directive[Black, 18],
    GraphLayout → "DiscreteSpiralEmbedding", EdgeLabels → "Index",
    EdgeLabelStyle → Directive[Blue, Italic, 12],
    EdgeStyle → {1 ↔ 2 → Red, 2 ↔ 1 → {Red, Dashed},
      1 ↔ 5 → Green, 5 ↔ 1 → {Green, Dashed}},
    PlotLabel → Style["graphT1gl9e, EdgeLabels-Index", 14, Blue],
    ImageSize → 265];

graphT1gl9f =
  HighlightGraph[graphT1gl9e, {Style[1, Green], Style[2, Cyan],
    Style[7 ↔ 6, Dashed], Style[7 ↔ 4, Cyan]}, PlotLabel →
    Style["graphT1gl9f, Style.., Style.., Style..", 14, Blue]];

Row[{graphT1gl9e, graphT1gl9f}, Spacer[10]]
```



Дополнительно отметим, что в приведенном варианте graphT1gl9f ***Highlight-Graph*** обеспечивает изменение вида графа graphT1gl9e только по перечисленным позициям.

Вариант graphT1gl9g – пример уточнения стиля выводимых ребер, когда оформление определяется задаваемыми условиями – все дуги от узла с большим номером к узлу с меньшим выводятся зелеными пунктирными тонкими линиями, а остальные малиновыми сплошными толстыми, стилем по умолчанию выводятся подписи (***EdgeLabels***→"*Name*"). Вариант graphT1gl9h – пример уточнения вида графа graphT1gl9g стиля и подписи конкретного выводимого ребра:

```
graphT1gl9g = Graph[lstGrPLA, VertexSize → 0.2,
    VertexStyle → LightGray,
    VertexLabels → Placed["Name", Center],
    VertexLabelStyle → Directive[Black, 16],
    GraphLayout → "DiscreteSpiralEmbedding",
    PlotLabel →
     Style["graphT1gl9g, EdgeLabels-Name, ..x>y..", 14, Blue],
    EdgeLabels → "Name",
    EdgeStyle → {x_ ↔ y_ /; x > y → Directive[Green, Dashed, Thin],
      Directive[Thick, Magenta]},
    ImageSize → 265];

graphT1gl9h = HighlightGraph[graphT1gl9g,
    {7 ↔ 6}, EdgeLabels → "Index=16",
    PlotLabel →
     Style["graphT1gl9h, ..{7↔6},EdgeLabels..", 14, Blue]];

Row[{graphT1gl9g, graphT1gl9h}, Spacer[10]]
```



Примеры оформления с учетом задаваемых весов

Внимание. Надо отслеживать нумерацию вершин у создаваемых в Wolfram *Mathematica* графов. У системы это не совсем то, что иногда кажется (не те номера, которые мы вписываем, определяя граф, если делаем это самостоятельно). ***Важно***. Номера (индексы) вершин графа определяются не той цифрой (именем), которую прописываем, задавая связи. Коды ниже обеспечивают сопоставление списка индексов вершин используемого в примерах графа lstGrPLA с числовым рядом номеров от 1 до числа вершин (в рассматриваемом случае – до 8):

```
lstGrPLA = {1 ↔ 2, 2 ↔ 1, 1 ↔ 5, 5 ↔ 1, 2 ↔ 6, 6 ↔ 2, 3 ↔ 4, 4 ↔ 3,
    3 ↔ 6, 6 ↔ 3, 4 ↔ 7, 7 ↔ 4, 4 ↔ 8, 8 ↔ 4, 6 ↔ 7, 7 ↔ 6};
nVertex = VertexCount[lstGrPLA]
```

```
8
```

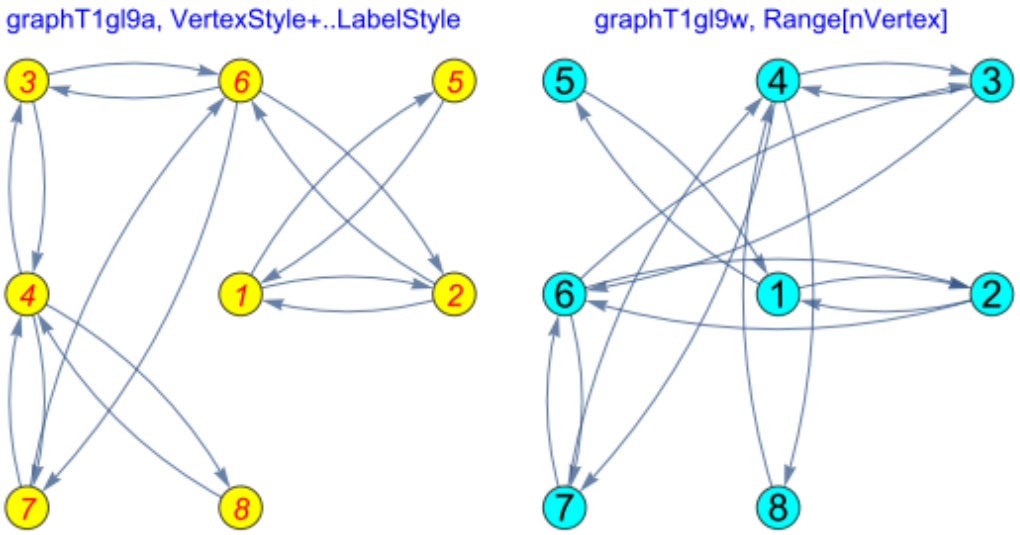
```
Range[nVertex]
VertexList[lstGrPLA]
```

```
{1, 2, 3, 4, 5, 6, 7, 8}
```

```
{1, 2, 5, 6, 3, 4, 7, 8}
```

Из результатов выше видим, что, например, 3-ей является вершина 5, 4-ой – 6, ... Подобное следует обязательно учитывать при задании весов в списке VertexWeight. До рассмотрения примеров визуализации значений весов дополнительно приведем пример “упорядочивания” номеров(индексов) вершин. Чтобы избегать недоразумений “номера вершин”, следует описывать ребра и вершины явно. Простейший вариант, обеспечивающий “упорядочить номера вершин” графа lstGrPLA и формирование уточненного графа graphT1gl9w:

```
graphT1gl9w = Graph[Range[nVertex], lstGrPLA,
    VertexSize → 0.2, VertexLabels → Placed["Name", Center],
    VertexStyle → Cyan,
    VertexLabelStyle → Directive[Black, 20],
    GraphLayout → "DiscreteSpiralEmbedding",
    PlotLabel → Style["graphT1gl9w, Range[nVertex]", 14, Blue],
    ImageSize → 265];

Row[{graphT1gl9a, graphT1gl9w}, Spacer[10]]
```



Обратить внимание, что после приведения в соответствие номеров, при отрисовке графа в варианте DiscreteSpiralEmbedding имеем вершины идут подряд.

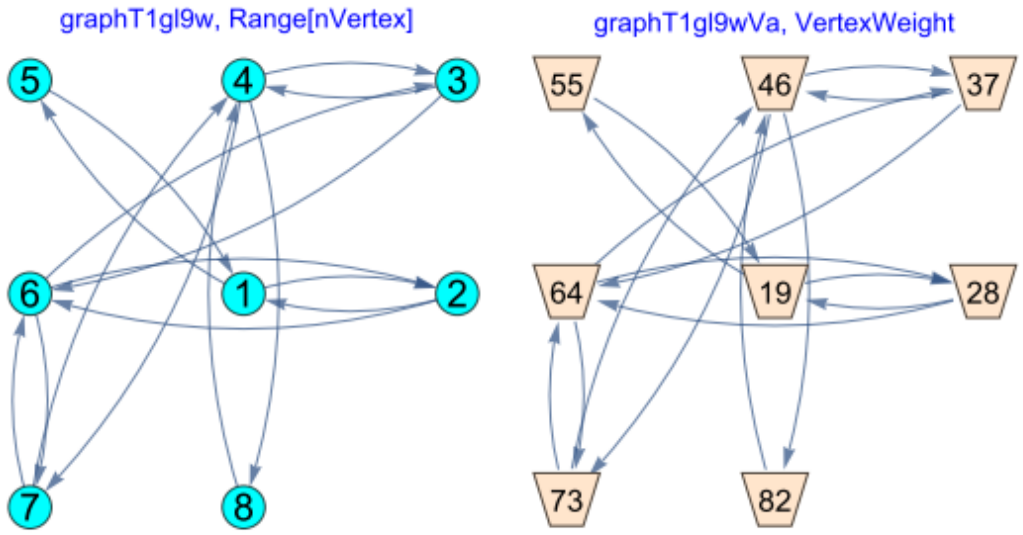
В предыдущих примерах в рассмотренных графах не были заданы веса вершин, ребер. Приведем примеры, визуализации весов, предварительно определив их, используя опции ***VertexWeight*** и ***EdgeWeight***.

При формировании графа graphT1gl9wVa опцией ***VertexWeight*** задаются веса, и для наглядности они определены двузначной цифрой, в которой первая цифра – индекс вершины, а вторая цифра – 9, 8, ..., 2 (чтобы при выводе весов по первой цифре было понятно, какая это вершина).

В иллюстрации графа graphT1gl9wVa изменены цвет (***VertexStyle***) и примитив вершин (***VertexShapeFunction***) – круги заменены трапециями; выводятся не индексы, а веса. Также напомним другие возможные варианты вместо использованного в примере DownTrapezoid: “Triangle”, “Square”, “Rectangle”, “Pentagon”, “Hexagon”, “DownTrapezoid”, “UpTrapezoid”, “Parallelogram”, “FiveDown”, “Circle”, “Diamond”,”Star”,”Capsule”.

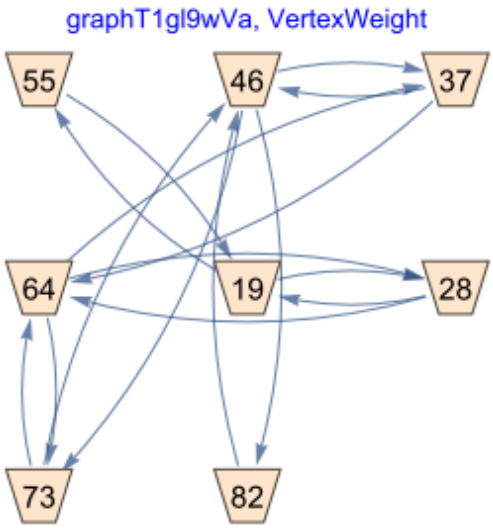
Обратить внимание, что в случаях задания (! ошибка) числа весов больше числа вершин система создает и при визуализации выводит “пустые” вершины.

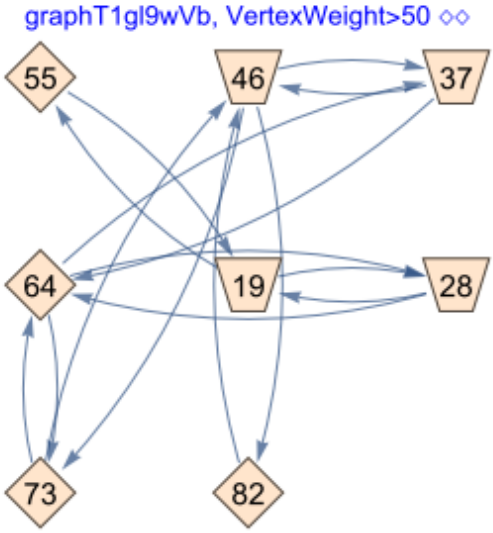
```
graphT1gl9wVa = Graph[graphT1gl9w, VertexSize → 0.3,
    VertexWeight → {19, 28, 37, 46, 55, 64, 73, 82},
    VertexShapeFunction → "DownTrapezoid",
    VertexLabels → Placed["VertexWeight", Center],
    VertexStyle → LightOrange,
    VertexLabelStyle → Directive[Black, 16],
    GraphLayout → "DiscreteSpiralEmbedding",
    PlotLabel → Style["graphT1gl9wVa, VertexWeight", 14, Blue],
    ImageSize → 265];

Row[{graphT1gl9w, graphT1gl9wVa}, Spacer[10]]
```



Пример ниже graphT1gl9wVb – иллюстрация, как выводить вершины, оформляя их с учетом задаваемого правила по весу вершины. В примере вершины с весом более 50 выводятся примитивом ромб:

```
graphT1gl9wVb = Graph[graphT1gl9wVa, VertexSize → 0.3,
    VertexShapeFunction → Table[v →
        If[PropertyValue[{graphT1gl9wVa, v},
            VertexWeight] > 50, "Diamond", "DownTrapezoid"],
      {v, VertexList[graphT1gl9wVa]}],
    VertexLabels → Placed["VertexWeight", Center],
    VertexLabelStyle → Directive[Black, 16],
    GraphLayout → "DiscreteSpiralEmbedding",
    PlotLabel →
     Style["graphT1gl9wVb, VertexWeight>50 ◇◇", 14, Blue],
    ImageSize → 265];

Row[{graphT1gl9wVa, graphT1gl9wVb}, Spacer[10]]
```





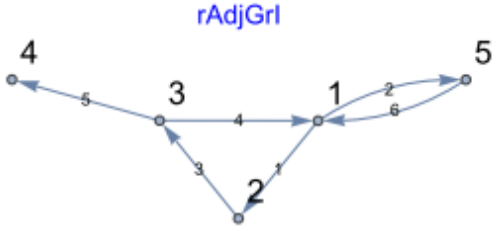
Работа с весами ребер, вывод подписей выполняются аналогично, для этого следует использовать ***EdgeWeight, EdgeLabels, EdgeShapeFunction***.

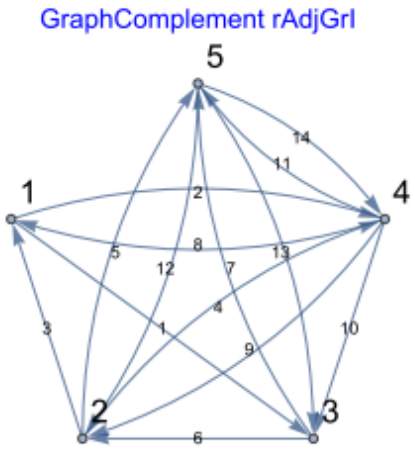
Функции формирования, модификации, анализа, сравнения графов

Приведем несколько простейших функций построения и манипуляций, анализа, сравнения графов.

Функция ***GraphComplement*** строит дополнения неориентированного графа до полного. В примерах ниже иллюстрации даются для (построенного выше) графа rAdjGrI, используется rAdjGr, а не начальный graph2d, чтобы не было нестандартного имени вершины n:

```
rAdjGrGC = GraphComplement[rAdjGrI,
    VertexLabels → "Name", VertexLabelStyle → {Black, 17},
    EdgeLabels → "Index",
    PlotLabel → Style["GraphComplement rAdjGrI", 14, Blue],
    ImageSize → 260];
Row[{rAdjGrI, rAdjGrGC}, Spacer[10]]
```





```
{VertexCount[rAdjGrI], EdgeCount[rAdjGrI]}
{VertexCount[rAdjGrGC], EdgeCount[rAdjGrGC]}
```

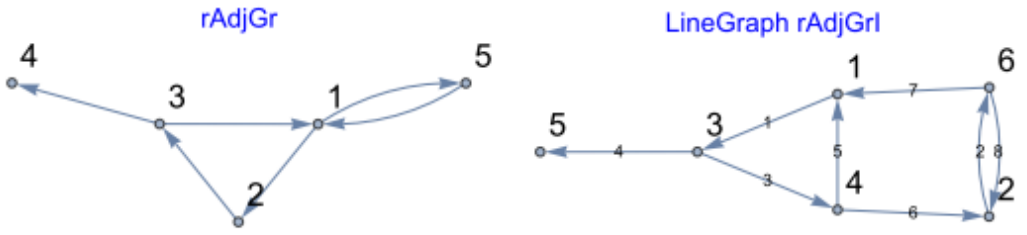
```
{5, 6}
```

```
{5, 14}
```

Для наглядности дополнительно опцией ***EdgeLabels***→"Index" в иллюстрациях rAdjGrI и rAdjGrLG выводятся индексы ребер. Также отдельно для графов rAdjGrI и rAdjGrGC выводятся числа вершин и ребер (используются функции ***VertexCount*** и ***EdgeCount***).

Линейный граф (ребра становятся вершинами и наоборот; количество ребер в графе равно количеству вершин в его линейном графе) формируется в *Mathematica* функцией ***LineGraph***:

```
rAdjGrLG = LineGraph[rAdjGrI,
    VertexLabels → "Name", VertexLabelStyle → {Black, 17},
    PlotLabel → Style["LineGraph rAdjGrI", 14, Blue],
    EdgeLabels → "Index", ImageSize → 260];
Row[{rAdjGr, rAdjGrLG}, Spacer[10]]
```

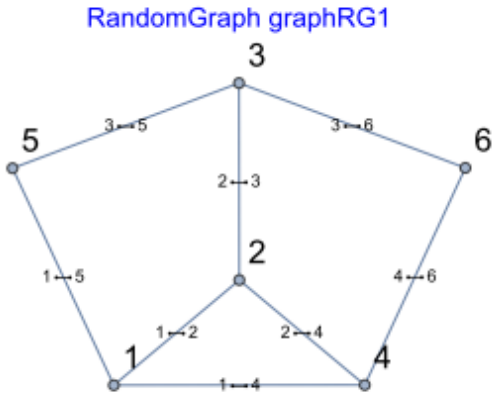


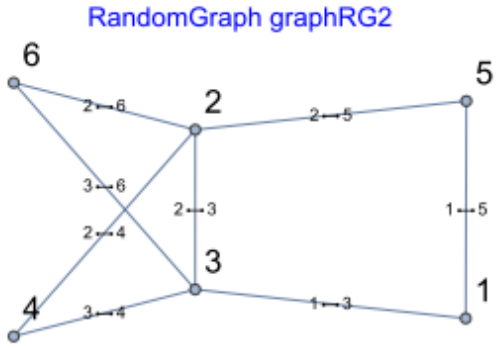

```
{VertexCount[rAdjGrLG], EdgeCount[rAdjGrLG]}
```

```
{6, 8}
```

Функция ***RandomGraph***[{*n*,*e*}] создает случайный граф, который формируется из задаваемого множества *n* изолированных вершин и последовательного случайного добавления соединяющих вершины *e* ребер. Примеры графов 6 вершин и 8 ребер:

```
graphRG1 = RandomGraph[{6, 8}, VertexLabels → "Name",
    EdgeLabels → "Name", VertexLabelStyle → {Black, 17},
    PlotLabel → Style["RandomGraph graphRG1", 14, Blue],
    ImageSize → 260];
graphRG2 = RandomGraph[{6, 8}, VertexLabels → "Name",
    EdgeLabels → "Name", VertexLabelStyle → {Black, 17},
    PlotLabel → Style["RandomGraph graphRG2", 14, Blue],
    ImageSize → 260];
Row[{graphRG1, graphRG2}, Spacer[10]]
```





Формирование графов выше выполняется одним и тем же кодом, чтобы показать существенную разницу результатов; числа вершин и ребер также заданы не случайно, а они такие же, как в примере rAdjGrLG. Отметим, что каждое выполнение секции выше дает другие графы.

Примеры использования функции ***RandomGraph***[{*n*,*e*}] выше иллюстрируют построение псевдослучайного графа с *n* вершинами и *e* ребрами без уточнения функции распределения. Отметим, что в качестве такового принимается (по умолчанию) ***UniformGraphDistribution*** – равномерное распределение. В *Mathematica* возможен вариант использования ***RandomGraph*** с инициализацией конкретного генератора псевдослучайных чисел, например с распределениями: ***BernoulliGraphDistribution, DegreeGraphDistribution, SpatialGraphDistribution, PriceGraphDistribution, BarabasiAlbertGraphDistribution, WattsStrogatzGraphDistribution***. Случайные графы применяются во многих областях, особенно при моделировании сложных сетей, причем, для многих из них созданы и постоянно обновляются библиотеки моделей графов, отражающих разнообразные типы. При изучении разных случайных графов можно рекомендовать многократное выполнение с целью подобрать конфигурацию (вид) предпочтительного, его запротоколировать, в частности с использованием ***EdgeList***, а затем экспериментировать с конкретным предпочтительным (понимая, что любое выполнение ***RandomGraph*** даст что-то другое).

Приведем несколько простейших функций анализа, сравнения графов.

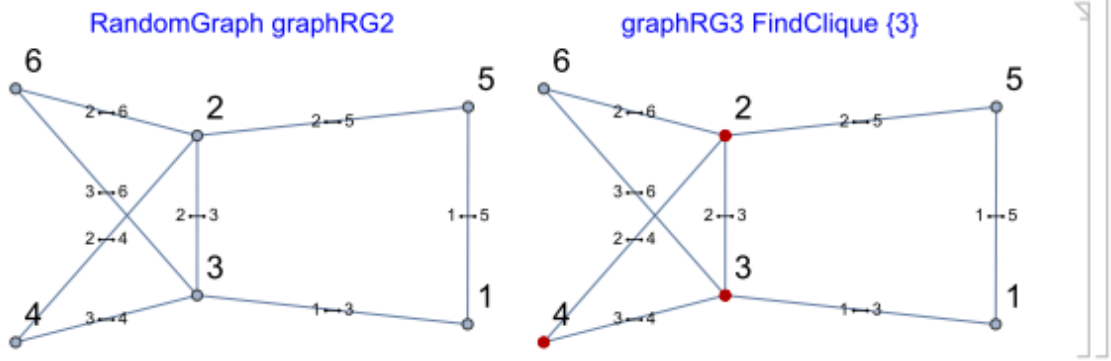
Функция ***GraphDifference*** строит те ребра каждого из двух графов, которые отсутствуют в другом.

```
grDiffer12 =
  GraphDifference[graphRG1, graphRG2, VertexLabels → "Name",
   PlotLabel → Style["Difference graphRG1 graphRG2", 14, Blue],
   ImageSize → 260];
```

```
grDiffer21 =
  GraphDifference[graphRG2, graphRG1, VertexLabels → "Name",
   PlotLabel → Style["Difference graphRG2 graphRG1", 14, Blue],
   ImageSize → 260];

Row[{grDiffer12, grDiffer21}, Spacer[10]]
```

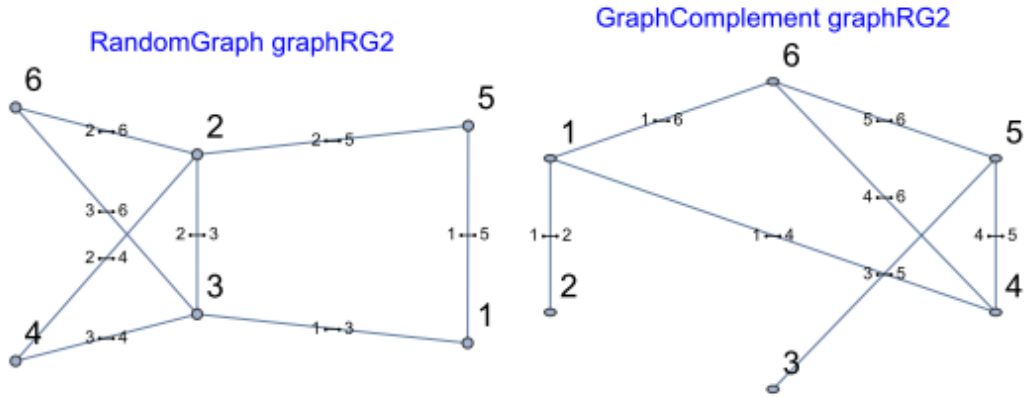
Функция ***FindClique***[graph,{n}] (найти максимальную клику) возвращает список вершин, которые принадлежат клике графа из *n* вершин (кликой неориентированного графа называется подмножество его вершин, любые две из которых соединены ребром). В примере ниже graphRG3 иллюстрация сделана для графа graphRG2:

```
graphRG3 = HighlightGraph[graphRG2,
    FindClique[graphRG2, {3}],
    PlotLabel → Style["graphRG3 FindClique {3}", 14, Blue],
    ImageSize → 260];

Row[{graphRG2, graphRG3}, Spacer[10]]
```



Уже использованная выше функция ***GraphComplement*** строит дополнения неориентированного графа до полного; пример применения для graphRG2:

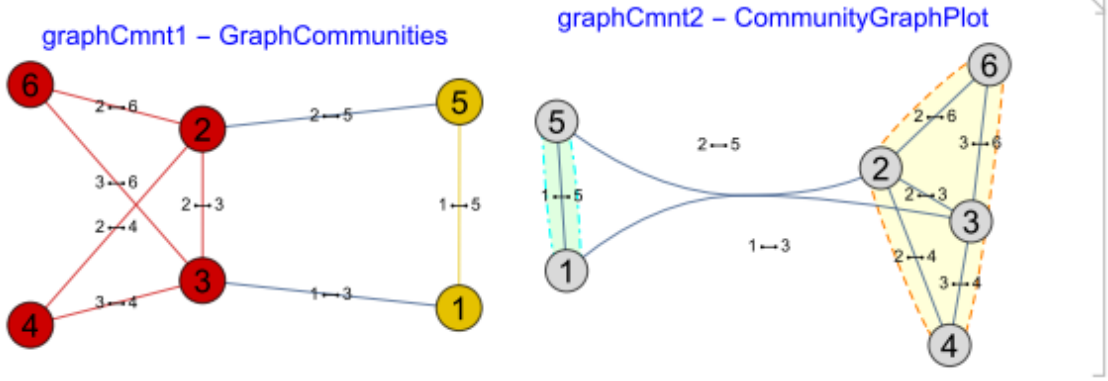
```
graphRG4 = GraphComplement[graphRG2,
    PlotLabel → Style["GraphComplement graphRG2", 14, Blue],
    VertexLabels → "Name",
    EdgeLabels → "Name", VertexLabelStyle → {Black, 17},
    AspectRatio → 0.7, ImageSize → 260];
Row[{graphRG2, graphRG4}, Spacer[10]]
```



Функция ***FindGraphCommunities*** возвращает списки вершин, которые принадлежат связным множествам графа. В примере graphCmnt1 предварительно функцией ***FindGraphCommunities*** определяется список вершин подграфа lstFGC, затем отобранные вершины списка и ребра оформляются (с установками по умолчанию) и выводятся с использованием функций ***HighlightGraph***, ***Map***, ***Subgraph***. Функция ***CommunityGraphPlot***[g,***FindGraphCommunities***[g]] служит для удобного отображения данных связных множеств. В примере graphCmnt2 дополнительно применены опции ***CommunityBoundaryStyle*** и ***CommunityRegionStyle***, которыми заданы атрибуты линий окаймления подмножеств и цвета заливки их площадей:

```
lstFGC = FindGraphCommunities[graphRG2]
graphCmnt1 = HighlightGraph[graphRG2,
    Map[Subgraph[graphRG2, #] &, lstFGC],
    VertexSize → 0.3, VertexStyle → LightGray,
    VertexLabels → Placed["Name", Center],
    PlotLabel → Style["graphCmnt1 - GraphCommunities", 14, Blue],
    ImageSize → 260];

graphCmnt2 = CommunityGraphPlot[graphRG2,
    FindGraphCommunities[graphRG2],
    VertexSize → 0.4, VertexStyle → LightGray,
    VertexLabels → Placed["Name", Center],
    CommunityBoundaryStyle → {Directive[{Orange, Dashed}],
      Directive[{Cyan, DotDashed}]},
    CommunityRegionStyle → {LightYellow, LightGreen},
    (* CommunityLabels→{"one","two"}, *)
    PlotLabel →
     Style["graphCmnt2 - CommunityGraphPlot", 14, Blue],
    ImageSize → 260];

Row[{graphCmnt1, graphCmnt2}, Spacer[10]]
```

```
{{2, 3, 4, 6}, {1, 5}}
```

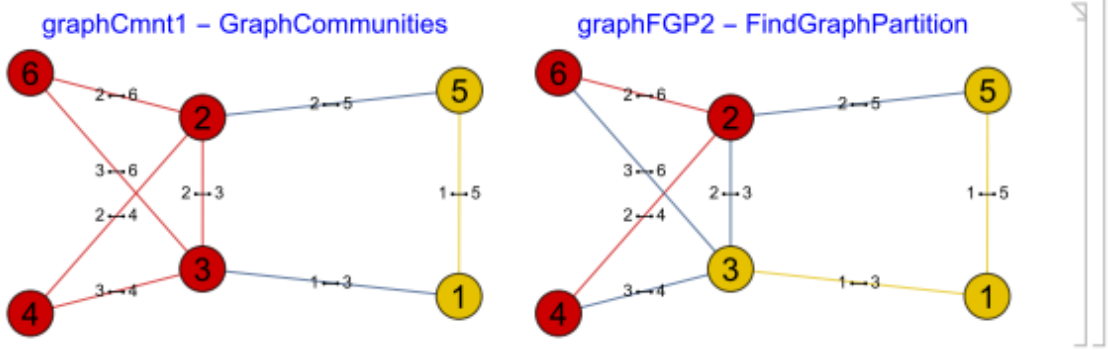


Повторим, что в примерах всегда код записывается максимально простым, хотя очевидно, что в ситуации выше, например, можно было бы использовать % вместо идентификатора предыдущего действия и, соответственно, его (lstFGC) не вводить.

В примере graphFGP2 предварительно функцией ***FindGraphPartition*** определяется список rFGP2 вершин подграфов, graphFGP2 – показывает разбиение графа graphRG2:

```
rFGP2 = FindGraphPartition[graphRG2]
graphFGP2 = HighlightGraph[graphRG2,
    Map[Subgraph[graphRG2, #] &, rFGP2],
    VertexSize → 0.3, VertexLabels → Placed["Name", Center],
    PlotLabel →
     Style["graphFGP2 - FindGraphPartition", 14, Blue]];

Row[{graphCmnt1, graphFGP2}, Spacer[10]]
```

```
{{2, 4, 6}, {1, 3, 5}}
```



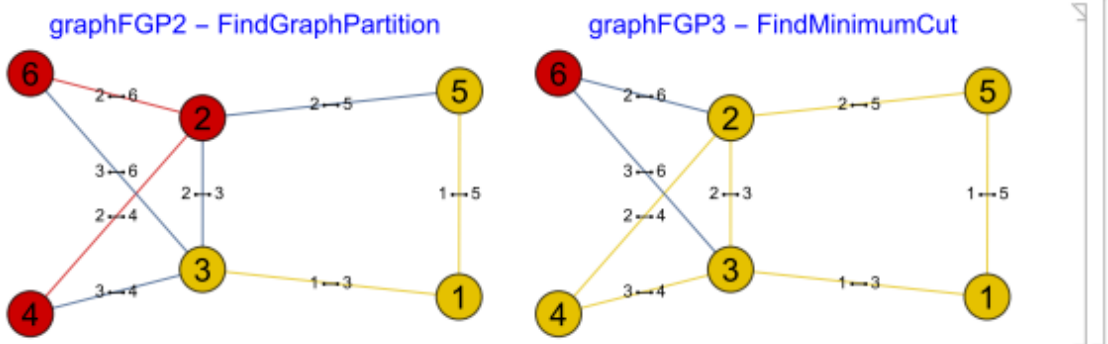
Разбиения графов используются, когда надо упростить задачу анализа сложной системы. Часто разбиение позволяет выполнить анализ графовой модели системы.

В примере graphFGP3 предварительно функцией ***FindMinimumCut*** определяется список rFGP3 вершин подграфа, graphRG3 – показывает

минимальный разрез графа graphRG2:

```
rFGP3 = FindMinimumCut[graphRG2]
graphFGP3 = HighlightGraph[graphRG2,
    Map[Subgraph[graphRG2, #] &, Last[rFGP3]],
    VertexSize → 0.3, VertexLabels → Placed["Name", Center],
    PlotLabel → Style["graphFGP3 - FindMinimumCut", 14, Blue]];

Row[{graphFGP2, graphFGP3}, Spacer[10]]
```

```
{2, {{6}, {1, 2, 3, 4, 5}}}
```



FindMinimumCut возвращает список вида $\{c_{min}, \{c_1, c_2, \ldots\}\}$, где c_{min} – значение найденного минимального разреза, и $\{c_1, c_2, \ldots\}$ – разбиение вершин, для которых оно найдено. По умолчанию вес каждого ребра принимается равным его значению ***EdgeWeightproperty***, если оно определено, в противном случае 1. Также можно использовать функцию ***FindVertexCut***.

Визуализация решений в примерах типовых задач

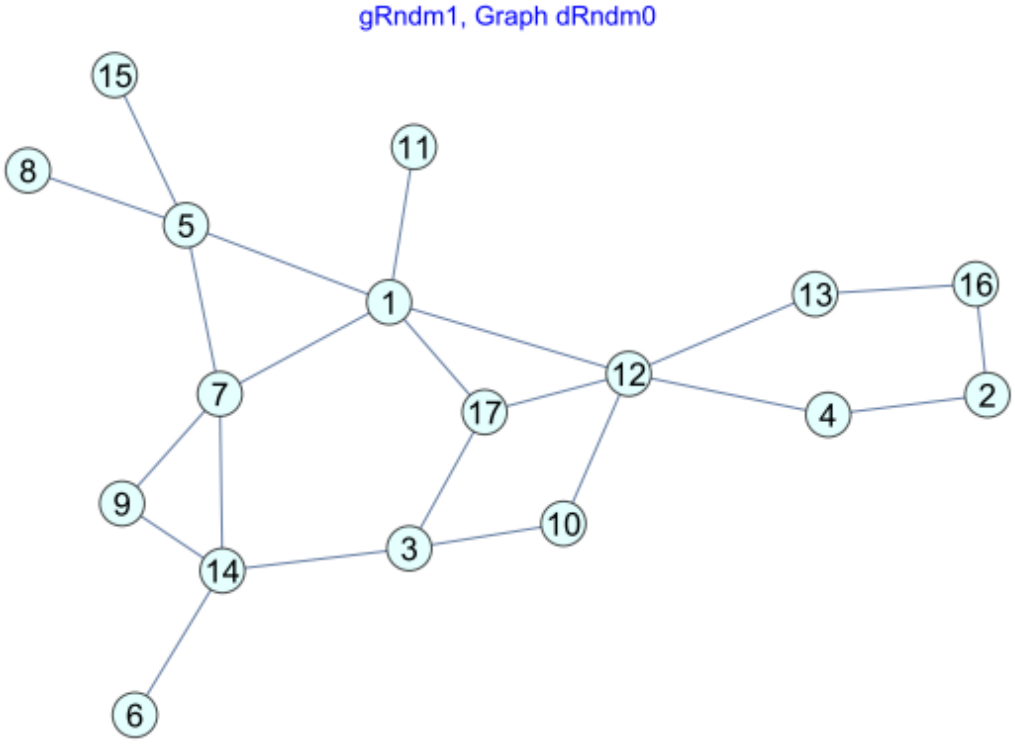
Приведем несколько примеров получения решений и их визуализации для двух вариантов задачи нахождения кратчайшего пути.

Исходные данные, графы сформируем с использованием генераторов случайных чисел, причем, сделаем это для нескольких вариантов, когда изменения вносятся путем модификации начального графа. Чтобы в упражнениях, вариантах визуализации можно было просто фиксировать отличия определим сформированный граф списком ребер, понимая, что получен таковой в результате выполнения функции ***RandomGraph***[{17, 22}]. Ниже будем использовать одну из отобранных реализаций.

Относительно основных параметров подготовленного графа – выведем его свойства (ориентированный, ненаправленный, мультиграф, смешанный, простой, связный, маршрута, полный, без петель):

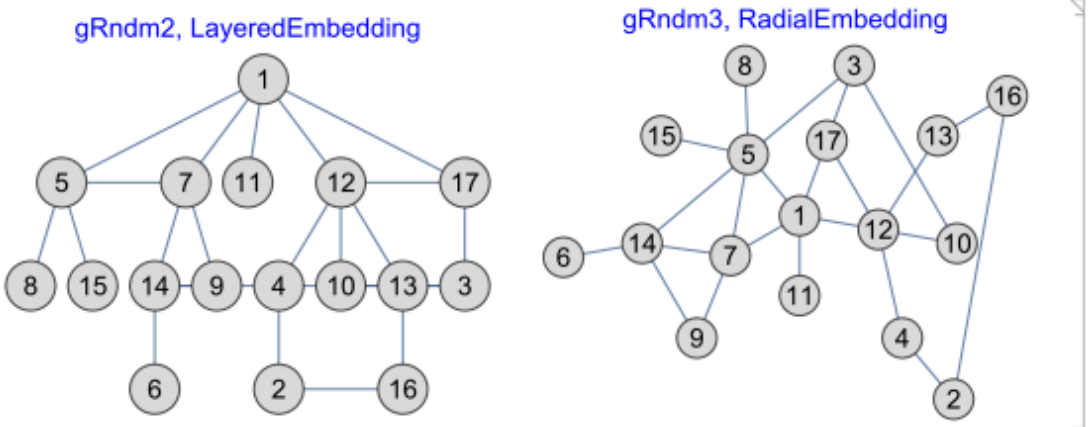
```
dRndm0 = {1 ↔ 5, 1 ↔ 7, 1 ↔ 11, 1 ↔ 12, 1 ↔ 17, 2 ↔ 4, 2 ↔ 16,
    3 ↔ 10, 3 ↔ 14, 3 ↔ 17, 4 ↔ 12, 5 ↔ 7, 5 ↔ 8, 5 ↔ 15, 6 ↔ 14,
    7 ↔ 9, 7 ↔ 14, 9 ↔ 14, 10 ↔ 12, 12 ↔ 13, 12 ↔ 17, 13 ↔ 16};

{nVertex = VertexCount[dRndm0], nEdge = EdgeCount[dRndm0]}
VertexList[dRndm0]

gRndm1 = Graph[Range[nVertex], dRndm0,
    VertexSize → 0.4, VertexStyle → LightCyan,
    VertexLabels → Placed["Name", Center],
    VertexLabelStyle → Directive[Black, 17],
    PlotLabel → Style["gRndm1, Graph dRndm0", 14, Blue],
    ImageSize → 550];

{DirectedGraphQ[gRndm1], UndirectedGraphQ[gRndm1],
 MultigraphQ[gRndm1], MixedGraphQ[gRndm1], SimpleGraphQ[gRndm1],
 ConnectedGraphQ[gRndm1], PathGraphQ[gRndm1],
 CompleteGraphQ[gRndm1], LoopFreeGraphQ[gRndm1]}

gRndm1
```

```
{17, 22}

{1, 5, 7, 11, 12, 17, 2, 4, 16, 3, 10, 14, 8, 15, 6, 9, 13}

{False, True, False, False, True, True, False, False, True}
```



Задачу нахождения кратчайшего пути рассмотрим на примере gRndm1, причем, сделаем это для нескольких вариантов, когда изменения вносятся путем модификации начального графа. Визуализацию dRndm1 в варианте укладки по умолчанию можно дополнить другими видами, например, с укладками ***LayeredEmbedding*** и ***RadialEmbedding***:

```
gRndm2 = Graph[dRndm0, VertexSize → 0.8,
    VertexStyle → LightGray, VertexLabels → Placed["Name", Center],
    VertexLabelStyle → Directive[Black, 14],
    GraphLayout → {"LayeredEmbedding", "LeafDistance" → 0.6},
    PlotLabel → Style["gRndm2, LayeredEmbedding", 14, Blue],
    ImageSize → 265];

gRndm3 = Graph[dRndm0, VertexSize → 0.5,
    VertexStyle → LightGray, VertexLabels → Placed["Name", Center],
    VertexLabelStyle → Directive[Black, 14],
    GraphLayout → "RadialEmbedding",
    PlotLabel → Style["gRndm3, RadialEmbedding", 14, Blue],
    ImageSize → 265];

Row[{gRndm2, gRndm3}, Spacer[10]]
```



Дополнительно можно получить основные параметры случайным образом сформированного графа, в частности, используя функции ***GraphAssortativity*** (ассортативность графа), ***GraphLinkEfficiency*** (эффективность соединённости графа), ***GraphDensity*** (плотность графа), ***GraphPeriphery*** (периферия графа), ***GraphDiameter*** (диаметр графа), ***GraphHub*** (ядро графа), ***GraphReciprocity*** (мера обратимости графа):

```
{GraphAssortativity[dRndm0],
 GraphLinkEfficiency[dRndm0], GraphDensity[dRndm0],
 GraphPeriphery[dRndm0], GraphDiameter[dRndm0],
 GraphHub[dRndm0], GraphReciprocity[dRndm0]}
```

$$\left\{-\frac{73}{422}, \frac{2617}{2992}, \frac{11}{68}, \{2, 16, 6\}, 6, \{1, 12\}, 1\right\}$$

Также можно рассчитать и выводить расстояния на графе, например:

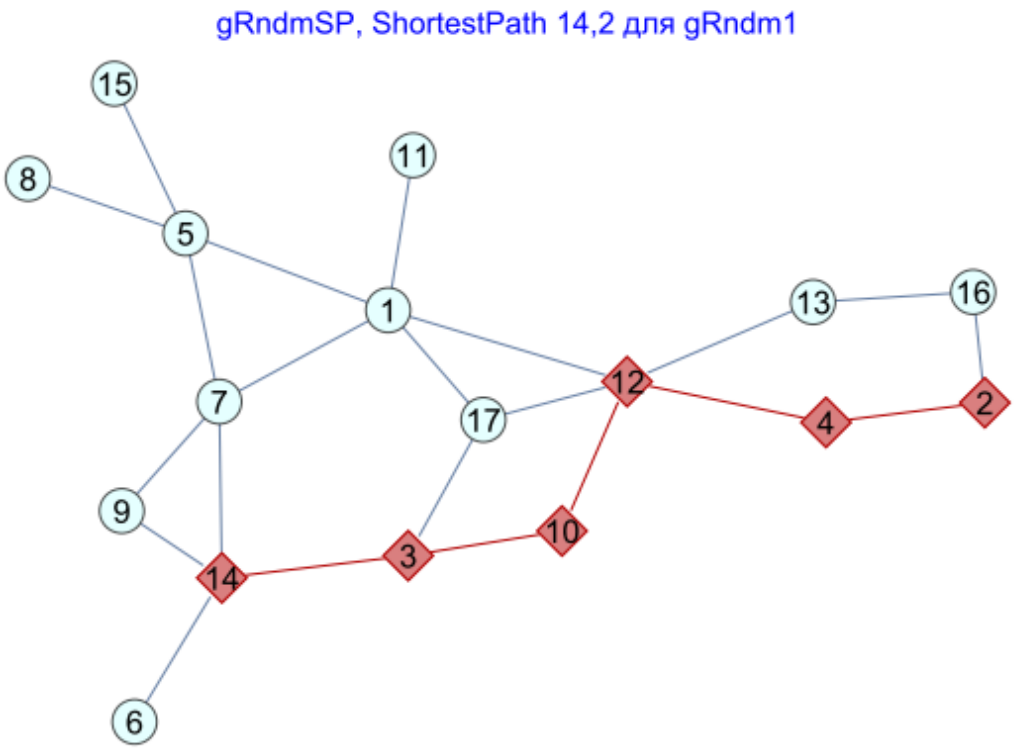
```
{GraphDistance[dRndm0, 1, 2], GraphDistance[dRndm0, 1, 3],
 GraphDistance[dRndm0, 17, 2], GraphDistance[dRndm0, 17, 6]}
```

```
{3, 2, 3, 3}
```

Пример решения задачи нахождения кратчайшего пути и его визуализации (функция ***FindShortestPath***$[g,n_1,n_2]$ возвращает кратчайший путь, по которому можно пройти из вершины n_1 в вершину n_2):

```
FindShortestPath[gRndm1, 14, 2]

gRndmSP = HighlightGraph[gRndm1, PathGraph[%],
  GraphHighlightStyle → "VertexDiamond",
  (* Другие возможные варианты вместо VertexDiamond:
    "VertexConcaveDiamond", "VertexTriangle",
  "DehighlightFade", "DehighlightGray", "DehighlightHide" *)
  PlotLabel → Style["gRndmSP, ShortestPath 14,2 для gRndm1",
    16, Blue]]
```

```
{14, 3, 10, 12, 4, 2}
```

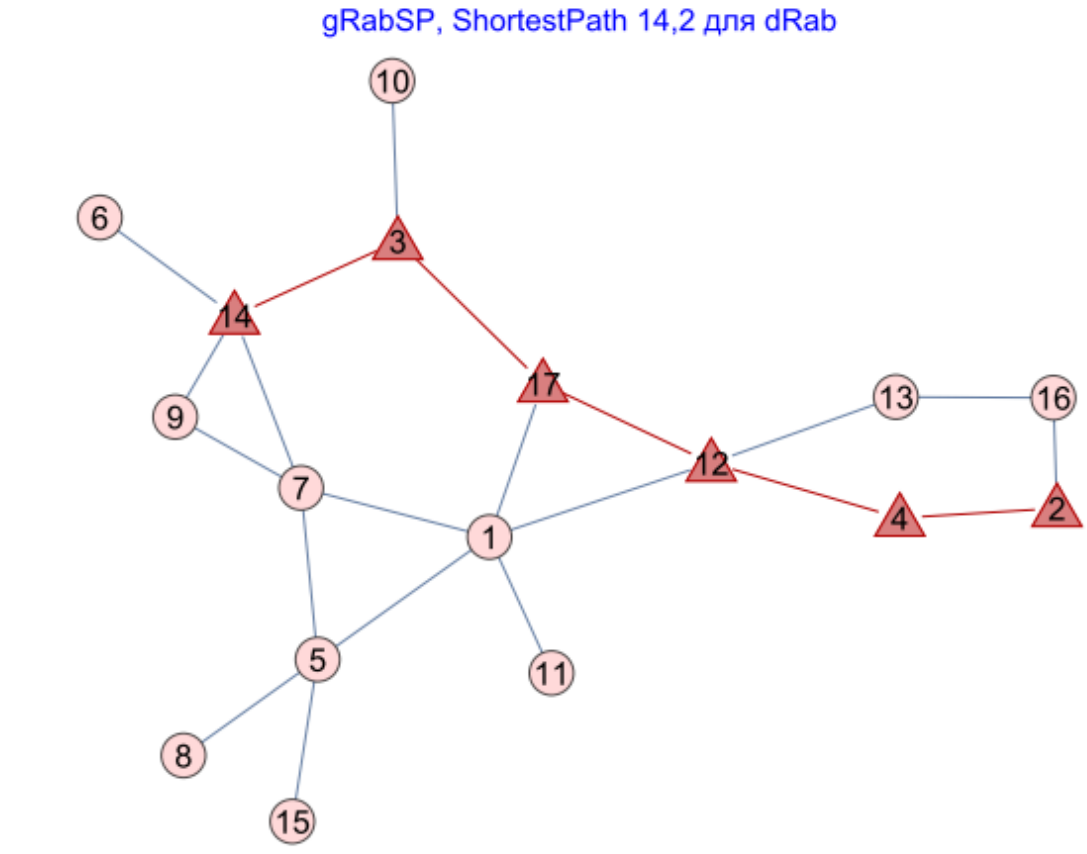


Заметим, что, используя ***FindShortestPath***, получать результаты и изучать варианты, задавая разные начальную и конечную вершины, очень

просто, но часто для сопоставления надо подбирать укладку сравниваемых графов. Поясним замечание примером. Сформируем граф gRab1, используя измененный список dRndm0, удалив в нем ребро 10⟷12, которое является частью кратчайшего пути для gRndmSP. Решение и результаты gRab1:

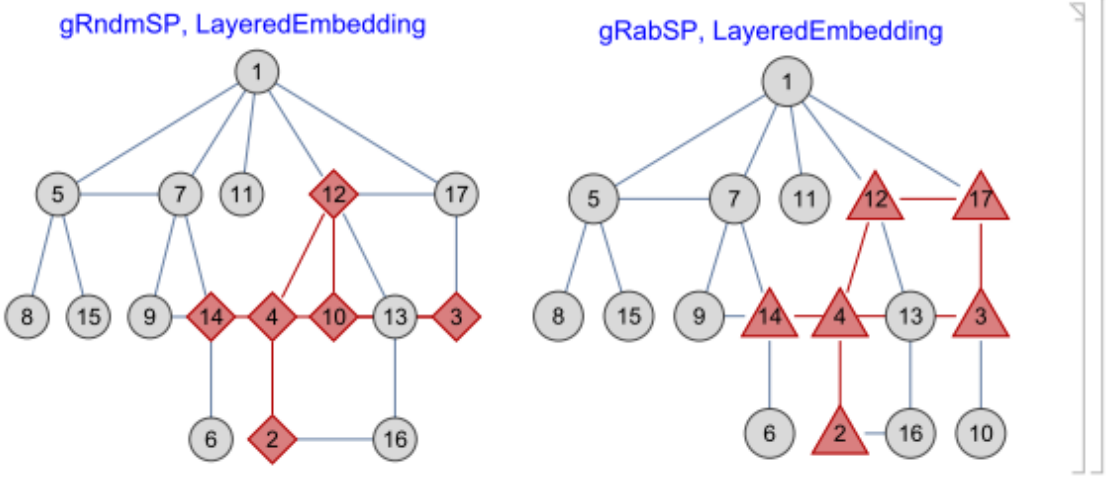
```
dRab = {1 ⟷ 5, 1 ⟷ 7, 1 ⟷ 11, 1 ⟷ 12, 1 ⟷ 17, 2 ⟷ 4, 2 ⟷ 16, 3 ⟷ 10,
    3 ⟷ 14, 3 ⟷ 17, 4 ⟷ 12, 5 ⟷ 7, 5 ⟷ 8, 5 ⟷ 15, 6 ⟷ 14, 7 ⟷ 9,
    7 ⟷ 14, 9 ⟷ 14, (*10⟷12,*) 12 ⟷ 13, 12 ⟷ 17, 13 ⟷ 16};

{nVertex = VertexCount[dRab], nEdge = EdgeCount[dRab]}

VertexList[dRab]
gRab1 = Graph[Range[nVertex], dRab,
    VertexSize → 0.4, VertexStyle → LightRed,
    VertexLabels → Placed["Name", Center],
    VertexLabelStyle → Directive[Black, 17],
    ImageSize → 550];
FindShortestPath[gRab1, 14, 2]
gRabSP = HighlightGraph[gRab1, PathGraph[%],
  GraphHighlightStyle → "VertexTriangle", PlotLabel →
    Style["gRabSP, ShortestPath 14,2 для dRab", 16, Blue]]
```

```
{17, 21}
{1, 5, 7, 11, 12, 17, 2, 4, 16, 3, 10, 14, 8, 15, 6, 9, 13}
{14, 3, 17, 12, 4, 2}
```



Очевидно, что сравнивать картинки gRndmSP и gRabSP затруднительно, так как в укладке по умолчанию виды графов совершенно разные. Как поступить? Применение подобранной укладки дает более зрелищное представление. Ниже для обоих вариантов дополнительно назначена укладка ***LayeredEmbedding***:

```
gRndmSPd = Graph[gRndmSP,
    VertexSize → 0.7,
    VertexStyle → LightGray,
    VertexLabels → Placed["Name", Center],
    VertexLabelStyle → Directive[Black, 12],
    GraphLayout → {"LayeredEmbedding", "LeafDistance" → 0.5},
    GraphHighlightStyle → "VertexDiamond",
    PlotLabel → Style["gRndmSP, LayeredEmbedding", 14, Blue],
    ImageSize → 260];

gRabSPd = Graph[gRabSP,
    VertexSize → 0.7,
    VertexStyle → LightGray,
    VertexLabels → Placed["Name", Center],
    VertexLabelStyle → Directive[Black, 12],
    GraphLayout → {"LayeredEmbedding", "LeafDistance" → 0.6},
    GraphHighlightStyle → "VertexTriangle",
    PlotLabel → Style["gRabSP, LayeredEmbedding", 14, Blue],
    ImageSize → 260];

Row[{gRndmSPd, gRabSPd}, Spacer[10]]
```



В примерах gRndmSP и gRabSP мы не указывали веса вершин, ребер, которые по умолчанию одинаковые (равны 1). Конечно же, веса следует

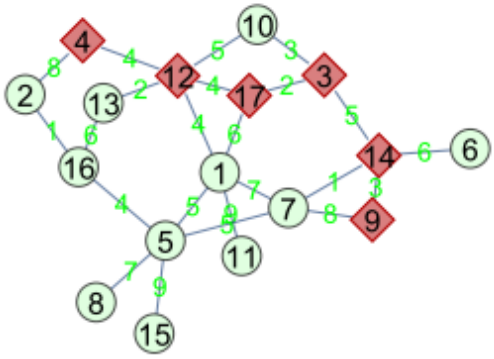
учитывать, и примеры ниже поясняют, как это делать. Для иллюстраций использованы результаты [5] – граф топологии сети и модифицированный, когда вес одного из ребер (17⟷3) изменен:

```
gExr1 = Graph[
    {1 ⟷ 5, 1 ⟷ 7, 1 ⟷ 11, 1 ⟷ 12, 1 ⟷ 17, 2 ⟷ 4, 2 ⟷ 16, 3 ⟷ 10,
     3 ⟷ 14, 3 ⟷ 17, 4 ⟷ 12, 5 ⟷ 7, 5 ⟷ 8, 5 ⟷ 15, 5 ⟷ 16, 6 ⟷ 14,
     7 ⟷ 9, 7 ⟷ 14, 9 ⟷ 14, 10 ⟷ 12, 12 ⟷ 13, 12 ⟷ 17, 13 ⟷ 16},
    EdgeWeight → {5, 7, 9, 4, 6, 8, 1, 3, 5, 2, 4,
      5, 7, 9, 4, 6, 8, 1, 3, 5, 2, 4, 6},
    VertexSize → 0.6, VertexStyle → LightGreen,
    VertexLabelStyle → Directive[Black, 16]];
Print[Style[Row[{"VertexCount[gExr1]=", VertexCount[gExr1],
     ", ", "EdgeCount[gExr1]=", EdgeCount[gExr1]}], 16]]

path = FindShortestPath[gExr1, 4, 9];
gExr1v = Graph[gExr1, EdgeLabels → "EdgeWeight",
    EdgeLabelStyle → Directive[Green, 14],
    VertexLabels → Placed[Automatic, Center]];

Exr1HG = HighlightGraph[gExr1v, path,
    GraphHighlightStyle → "VertexDiamond", PlotLabel →
     Style["Exr1HG, EdgeWeight→{5,7,10,...", 14, Blue],
    ImageSize → 265];

gExr2 = Graph[
    {1 ⟷ 5, 1 ⟷ 7, 1 ⟷ 11, 1 ⟷ 12, 1 ⟷ 17, 2 ⟷ 4, 2 ⟷ 16, 3 ⟷ 10,
     3 ⟷ 14, 3 ⟷ 17, 4 ⟷ 12, 5 ⟷ 7, 5 ⟷ 8, 5 ⟷ 15, 5 ⟷ 16, 6 ⟷ 14,
     7 ⟷ 9, 7 ⟷ 14, 9 ⟷ 14, 10 ⟷ 12, 12 ⟷ 13, 12 ⟷ 17, 13 ⟷ 16},
    EdgeWeight → {5, 7, 9, 4, 6, 8, 1, 3, 5, 9, 4,
      5, 7, 9, 4, 6, 8, 1, 3, 5, 2, 4, 6},
    VertexSize → 0.5, VertexStyle → LightRed,
    VertexLabelStyle → Directive[Black, 16]];
Print[Style[Row[{"VertexCount[gExr2]=", VertexCount[gExr2],
     ", ", "EdgeCount[gExr2]=", EdgeCount[gExr2]}], 16]]
path2 = FindShortestPath[gExr2, 4, 9];
gExr2v = Graph[gExr2, EdgeLabels → "EdgeWeight",
    EdgeLabelStyle → Directive[Red, 15],
    VertexLabels → Placed[Automatic, Center]];
Exr2HG = HighlightGraph[gExr2v, path2,
    GraphHighlightStyle → "VertexTriangle", PlotLabel →
     Style["Exr2HG, EdgeWeight→{5,3,10,...", 14, Blue],
    ImageSize → 265];

Row[{Exr1HG, Exr2HG}, Spacer[10]]
Print[Style[Row[{"path=", path, ", path2=", path2}], 16]]
```

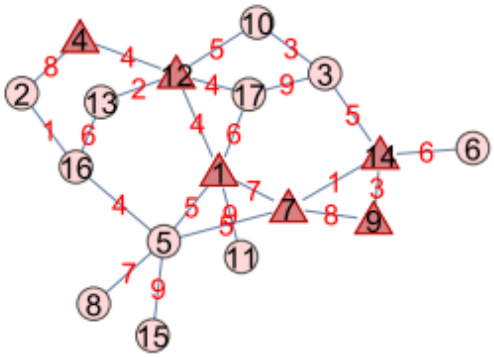
```
VertexCount[gExr1]=17, EdgeCount[gExr1]=23
```

```
VertexCount[gExr2]=17, EdgeCount[gExr2]=23
```

Exr1HG, EdgeWeight→{5,7,10,...



Exr2HG, EdgeWeight→{5,3,10,...



```
path={4, 12, 17, 3, 14, 9}, path2={4, 12, 1, 7, 14, 9}
```

Напоминание. Графы Exr1HG, Exr2HG имеют одинаковое число вершин и ребер, рассчитанные кратчайшие пути path, path2 определены с учетом весов.

Рекомендуемая литература

1. Теория графов. Основные понятия и виды графов. [Электронный ресурс]. URL : https://skysmart.ru/articles/mathematic/osnovnye-ponyatiya-teorii-grafov.

2. Graph Properties & Measurements. [Электронный ресурс]. URL : https://reference.wolfram.com/language/guide/GraphPropertiesAndMeasurements.html

3. Wolfram Language & System Documentation Center. GraphData. [Электронный ресурс]. URL : https://reference.wolfram.com/language/ref/GraphData.html.

4. *Пилипчук, Л. А.* Дробно-линейные экстремальные неоднородные задачи потокового программирования / Л.А. Пилипчук. - Минск: БГУ, 2013. - 235 с.

5. Wolfram Language & System Documentation Center. GraphLayout. [Электронный ресурс]. URL : https://reference.wolfram.com/language/ref/GraphLayout.html.

6. *Таранчук, В.Б.* Инструменты создания и сопровождения базы знаний путем интеграции системы Wolfram Mathematica и пакета NEVOD / В.Б. Таранчук, В.А. Савёнок // Материалы IX Международной научно-практической конференции «Системы управления, сложные системы: моделирование, устойчивость, стабилизация, интеллектуальные технологии» CSMSSIT-2023 (г. Елец, 24-25 апреля 2023 г.). Елец: ЕГУ им. И.А. Бунина, 2023. С. 201–205.

Содержание

Учебное издание

Таранчук Валерий Борисович

ПОСТРОЕНИЕ, ВИЗУАЛИЗАЦИЯ, ПРИМЕРЫ АНАЛИЗА ГРАФОВ В СИСТЕМЕ *MATHEMATICA*

Учебные материалы для студентов факультета прикладной математики и информатики

В авторской редакции

Ответственный за выпуск *В. Б. Таранчук*

Подписано в печать 19.12.2023. Формат 60×84/16. Бумага офсетная.
Усл. печ. л. 3,02. Уч.-изд. л. 2,94. Тираж 50 экз. Заказ

Белорусский государственный университет.
Свидетельство о государственной регистрации издателя, изготовителя, распространителя печатных изданий № 1/270 от 03.04.2014.
Пр. Независимости, 4, 220030, Минск.

Отпечатано на копировально-множительной технике
факультета прикладной математики и информатики
Белорусского государственного университета.
Пр. Независимости, 4, 220030, Минск.