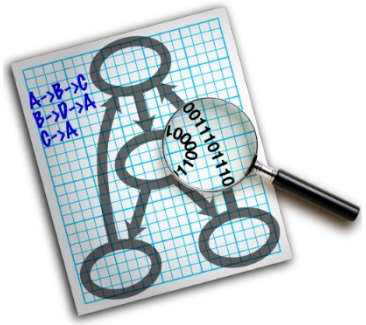
Разработка графических интерфейсов

Лабораторная работа 1

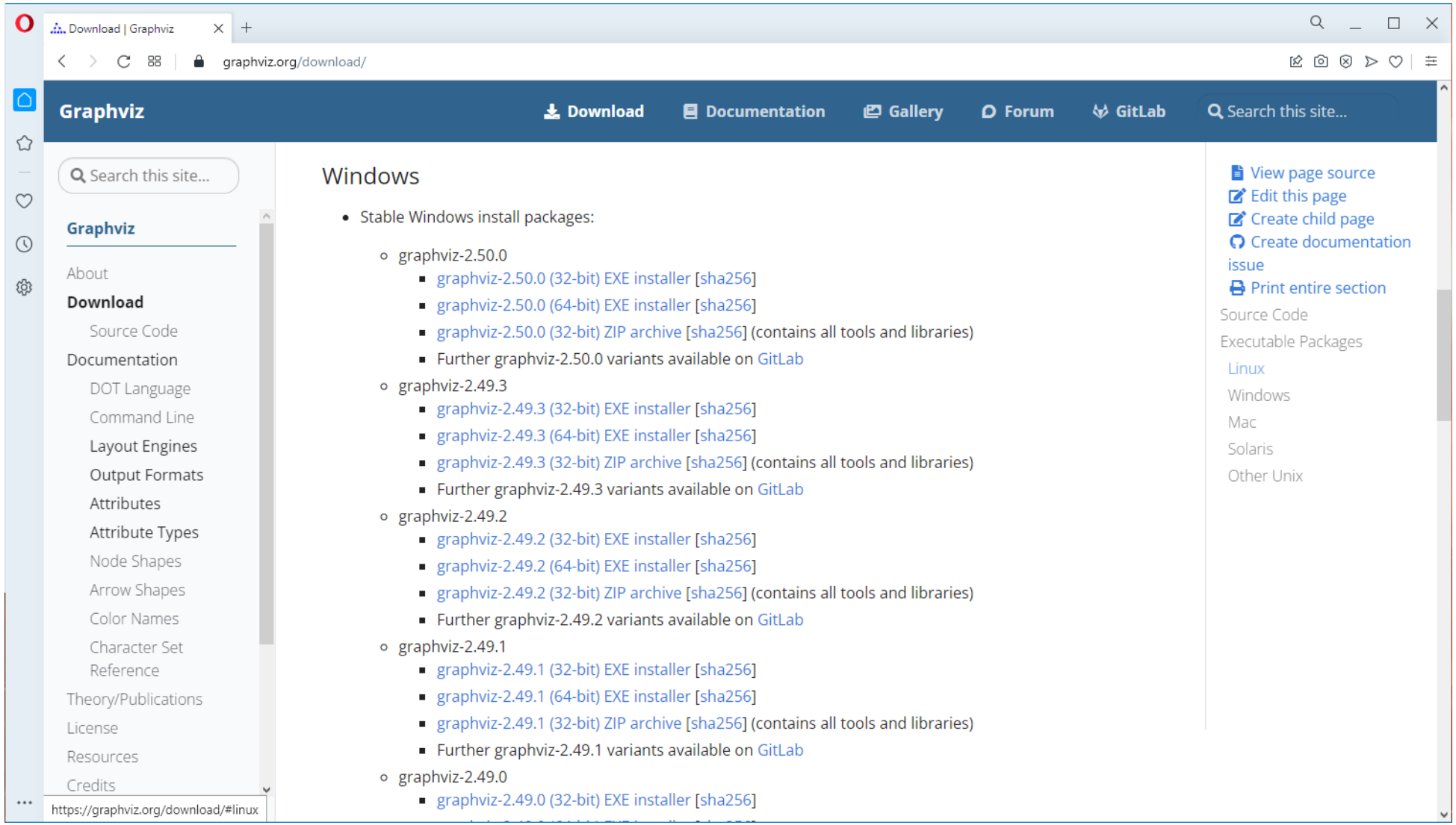
Визуализация графовых объектов.

Пакет Graphviz.

Программа Graphviz



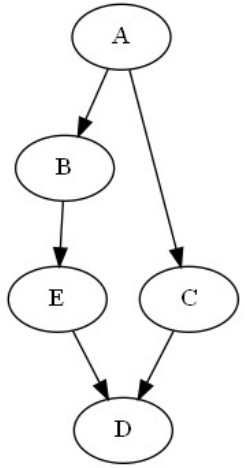
URL: https://graphviz.org



Синтаксис формата dot

граф	::=	[strict] (graph|digraph) [имя_графа] '{' список_объявлений '}'
список_объявлений	::=	[объявление [';'] список_объявлений]
объявление	::=	подграф | узел | ребро | атрибут | имя '=' имя
подграф	::=	[subgraph [имя_подграфа]] '{' список_объявлений '}'
узел	::=	имя_узла [список_атрибутов]
ребро	::=	(имя_узла | имя подграфа) данные_ребра [список_атрибутов]
данные_ребра	::=	тип_связи (имя_узла | имя подграфа) [данные ребра]
тип_связи	::=	(-- | ->)

```
digraph {
  A -> B;
  A -> C -> D;
  B -> E -> D;
}
```

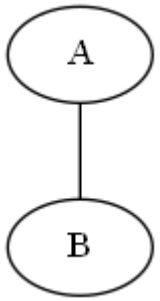


Типы графов: направленный и ненаправленный (1)

graph

Создаёт неориентированный граф:

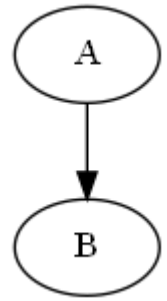
```
graph {
    A -- B
}
```



digraph

Создаёт ориентированный граф

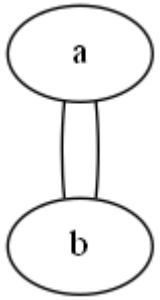
```
digraph {
    A -> B
}
```



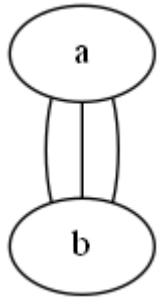
Описание графов

```
graph {
  a -- b
  a -- b
}
```

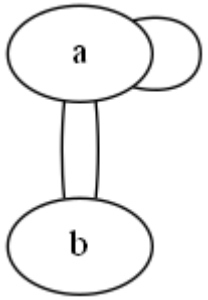
```
graph {
  a -- b
  b -- a
}
```




```
graph {
  a -- b
  a -- b
  b -- a
}
```

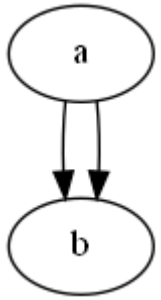



```
graph {
  a -- b
  a -- a
  b -- a
}
```

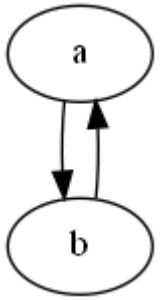


Описание орграфов

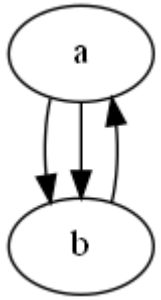
```
digraph {
  a -> b
  a -> b
}
```



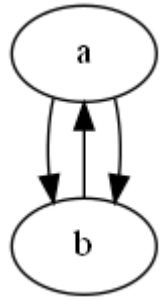

```
digraph {
  a -> b
  b -> a
}
```



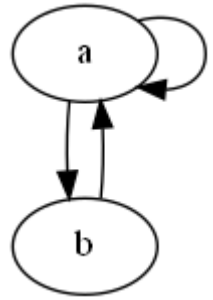

```
digraph {
  a -> b
  a -> b
  b -> a
}
```




```
digraph {
  a -> b
  b -> a
  a -> b
}
```



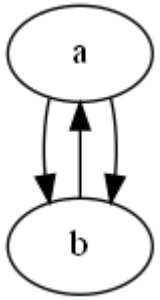

```
digraph {
  a -> b
  b -> a
  a -> a
}
```



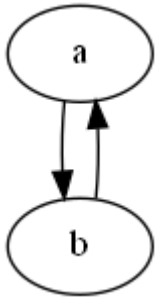
Ограничение числа рёбер: ключевое слово strict

Пример для ориентированных графов

```
digraph {
  a -> b
  b -> a
  a -> b
}
```

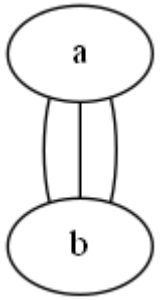



```
strict digraph {
  a -> b
  b -> a
  a -> b
}
```

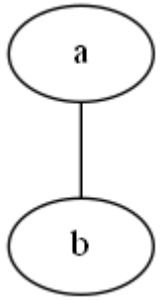


Пример для неориентированных графов

```
graph {
  a -- b
  a -- b
  b -- a
}
```

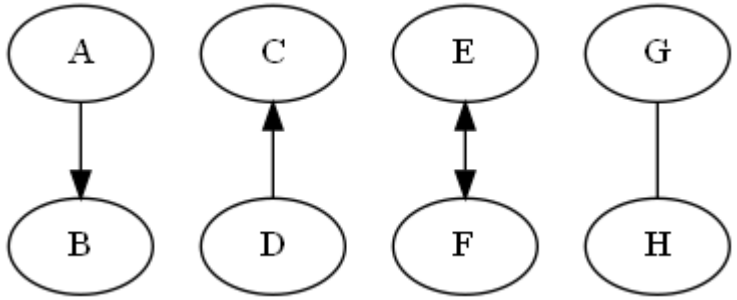



```
strict graph {
  a -- b
  a -- b
  b -- a
}
```

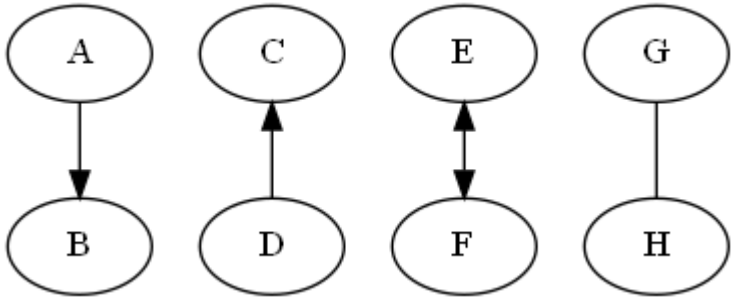


Явное задание ориентации ребра (3)

```
digraph {
  a -> b [dir=forward]
  c -> d [dir=back]
  e -> f [dir=both]
  g -> h [dir=none]
}
```




```
graph {
  a -- b [dir=forward]
  c -- d [dir=back]
  e -- f [dir=both]
  g -- h [dir=none]
}
```



Возможные значения атрибута dir (применим только для рёбер):

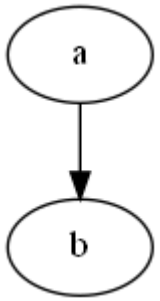
`forward` - значение по умолчанию для орграфов
`back`
`both`
`none` - значение по умолчанию для неориентированных графов

Задание вида рёбер (1)

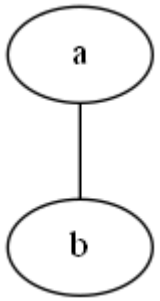
Форма указателя стрелки ребра

Применимо только для стрелок с атрибутом dir, равным *forward* или *both*

```
digraph {
  a -> b
}
```



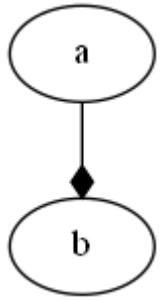

```
digraph {
  a -> b [arrowhead=none]
}
```



Форма указателя хвоста ребра

Применимо только для стрелок с атрибутом dir, равным *back* или *both*

```
digraph {
  a -> b [arrowhead=diamond]
}
```



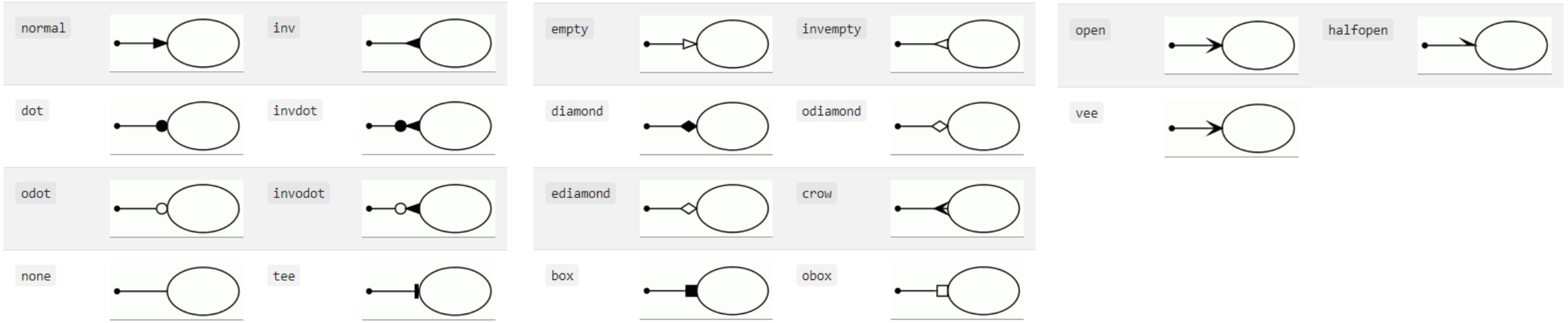
Задание вида рёбер (2)

Форма указателя стрелки ребра

Применимо только для стрелок с атрибутом dir, равным *forward* или *both*

Форма указателя хвоста ребра

Применимо только для стрелок с атрибутом dir, равным *back* или *both*



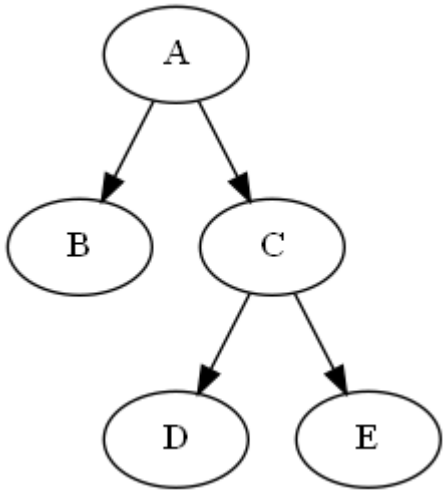
*Документация: http://www.graphviz.org/docs/attr-types/arrowType/

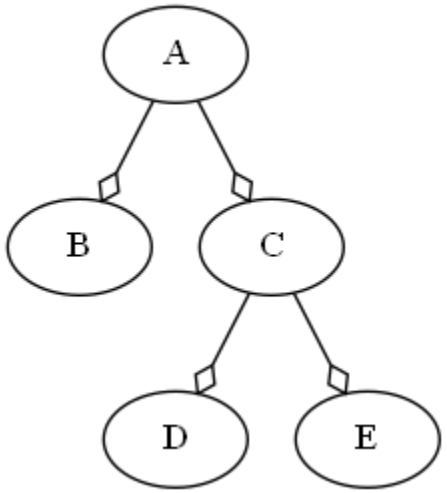
Описание для группы рёбер

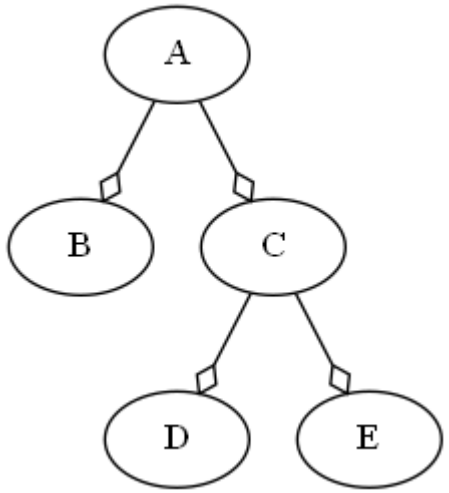
```
digraph {
  A -> B
  A -> C
  C -> D
  C -> E
}
```

```
digraph {
  A -> B [arrowhead=odiamond]
  A -> C [arrowhead=odiamond]
  C -> D [arrowhead=odiamond]
  C -> E [arrowhead=odiamond]
}
```

```
digraph {
  edge   [arrowhead=odiamond]
  A -> B
  A -> C
  C -> D
  C -> E
}
```

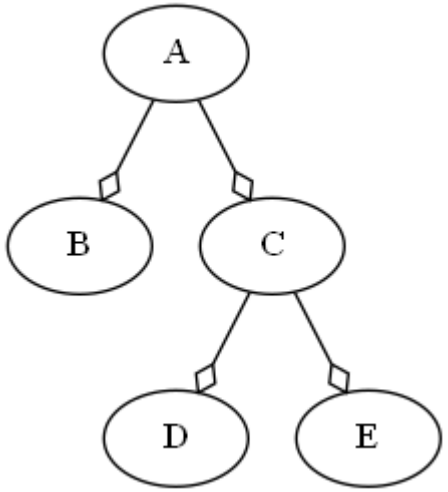




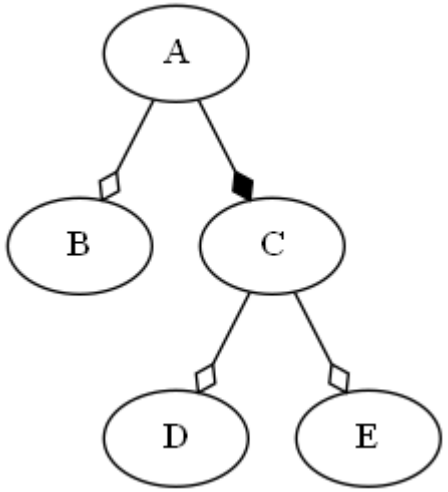


Описание рёбер (1)

```
digraph {
  edge   [arrowhead=odiamond]
  A -> B
  A -> C
  C -> D
  C -> E
}
```

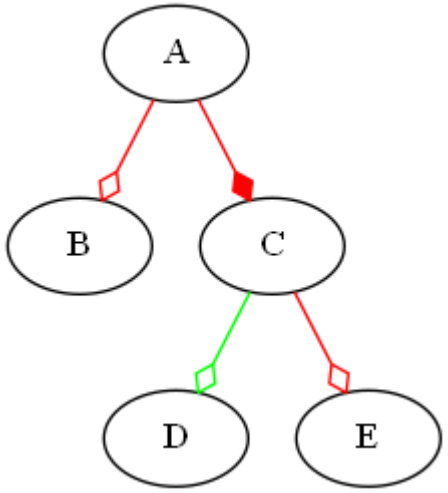



```
digraph {
  edge   [arrowhead=odiamond]
  A -> B
  A -> C [arrowhead=diamond]
  C -> D
  C -> E
}
```

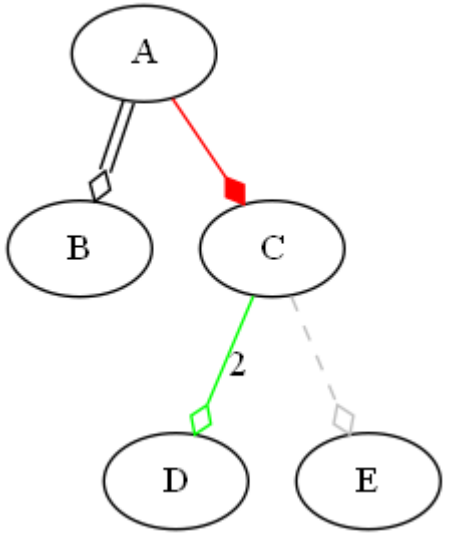


Описание рёбер (2)

```
digraph {
  edge    [arrowhead=odiamond, color=red]
  A -> B
  A -> C [arrowhead=diamond]
  C -> D [color=green]
  C -> E
}
```

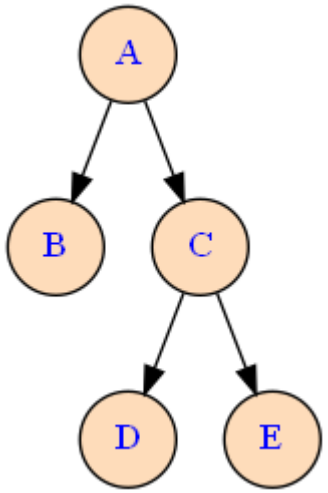



```
digraph {
  edge    [arrowhead=odiamond, color=red]
  A -> B [color="black:invis:black"]
  A -> C [arrowhead=diamond]
  C -> D [color=green, label="2"]
  C -> E [style=dashed, color=grey]
}
```



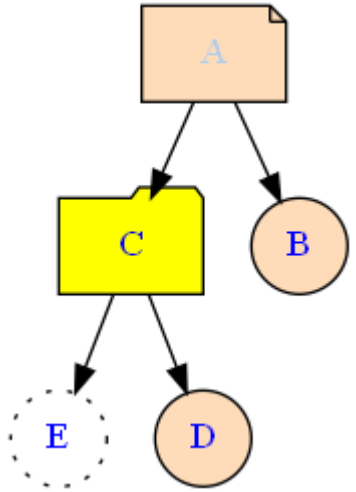
Задание вида узлов (1)

```
digraph {
  node [fontcolor=blue shape=circle style=filled fillcolor="#fedcba"]
  A -> B
  A -> C
  C -> D
  C -> E
}
```



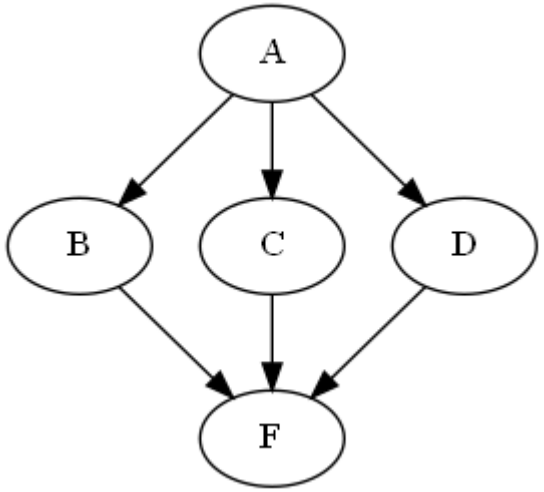
Задание вида узлов (2)

```
digraph {
  node [fontcolor=blue shape=circle style=filled fillcolor="#fedcba"]

  A [fontcolor="#abcdef" shape=note]
  C [shape=folder fillcolor="#ffff00"]
  E [style=dotted]

  A -> B
  A -> C
  C -> D
  C -> E
}
```

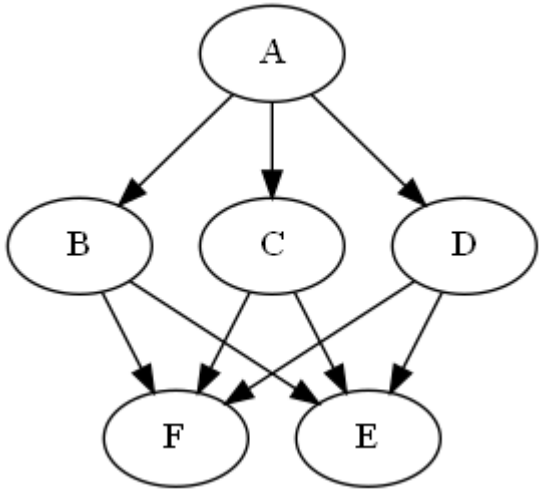


Группирование узлов (1)

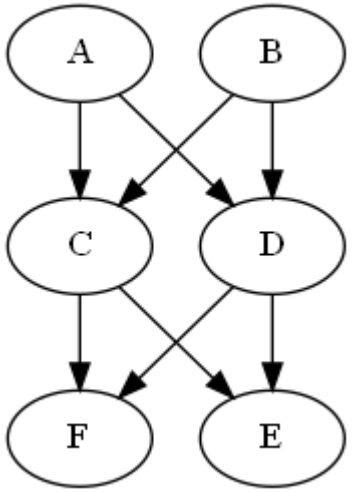
```
digraph {

  A -> {B, C, D} -> F

}
```




```
digraph {

  A -> {B, C, D} -> {F, E}

}
```

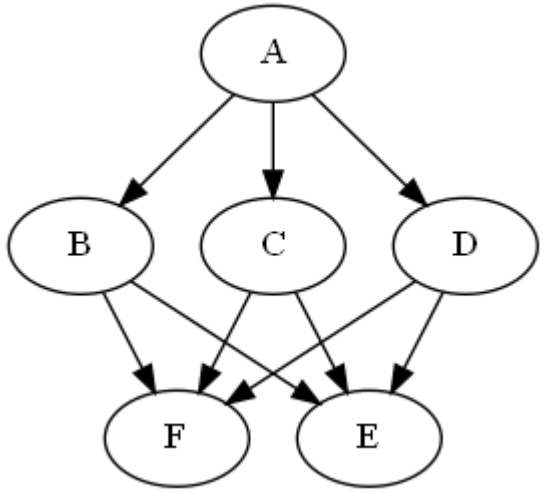



```
digraph {

  {A, B} -> {C, D} -> {F, E}

}
```

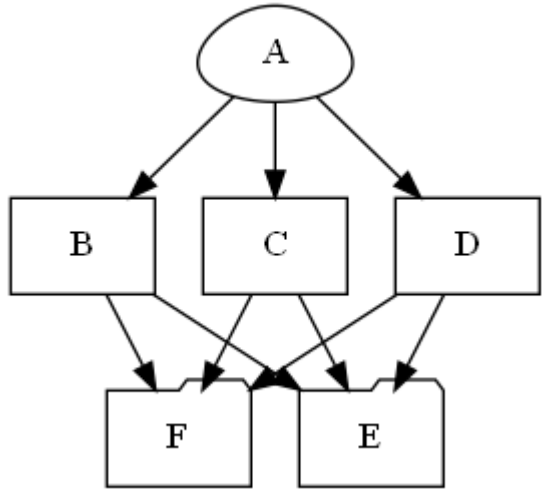


Группирование узлов (2)

```
digraph {

  A -> {B, C, D} -> F

}
```

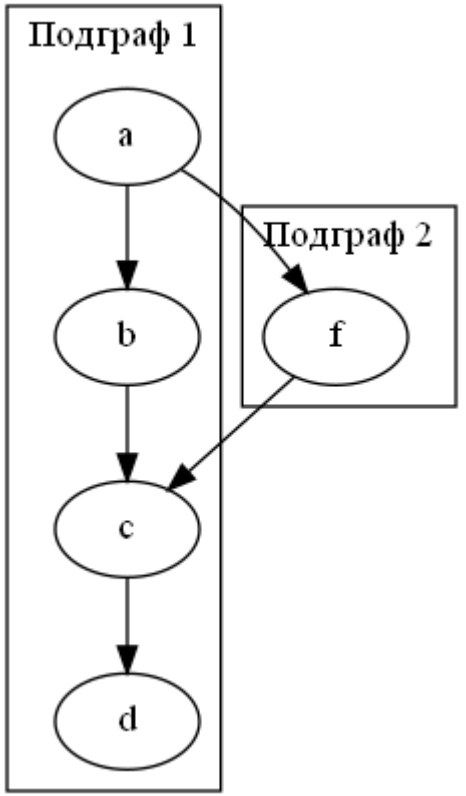



```
digraph {

  node [shape=egg]
  A
  node [shape=box]
  B; C; D
  node [shape=folder]
  F; E

  A -> {B, C, D} -> {F, E}

}
```



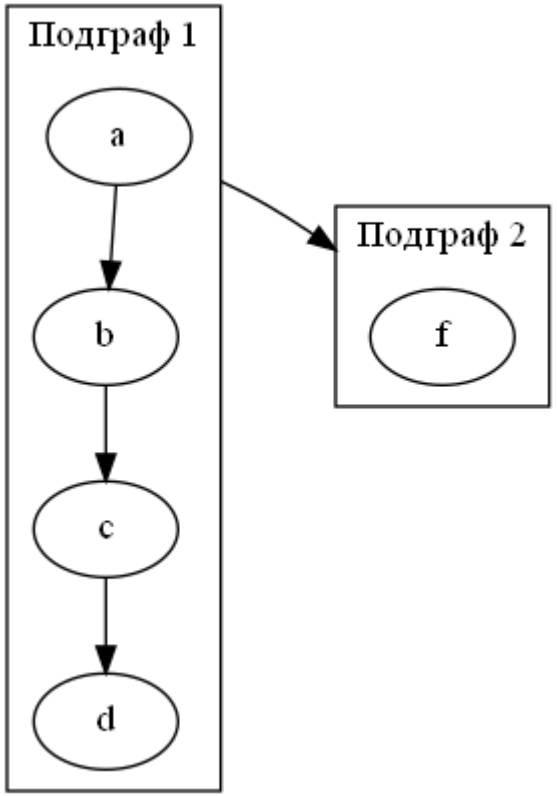
Группирование узлов подграфами (1)

```
digraph {
  subgraph cluster_0 {
   label="Подграф 1";
   a -> b;
   b -> c;
   c -> d;
  }

  subgraph cluster_1 {
   label="Подграф 2";
   a -> f;
   f -> c;
  }
}
```



Группирование узлов подграфами (2)

```
digraph {
  graph [compound=true nodesep=1 ranksep=0.5]
  subgraph cluster_0 {
    label="Подграф 1";
    a -> b;
    b -> c;
    c -> d;
  }

  subgraph cluster_1 {
    label="Подграф 2";
    f;
  }
  a -> f [ltail=cluster_0 lhead=cluster_1]
}
```



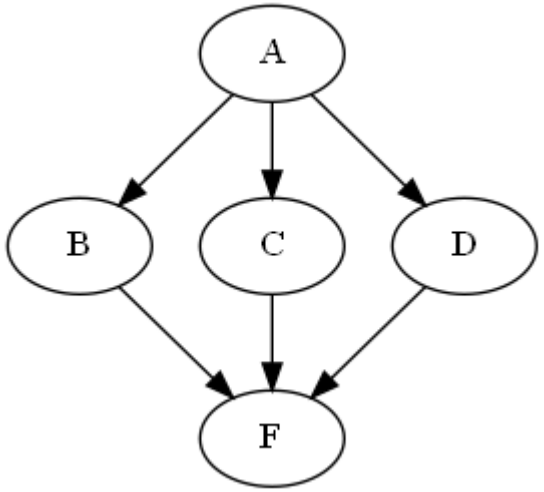
`compound=true` - разрешить связывать подграфы

`nodesep=1` - длина ребра между подграфами

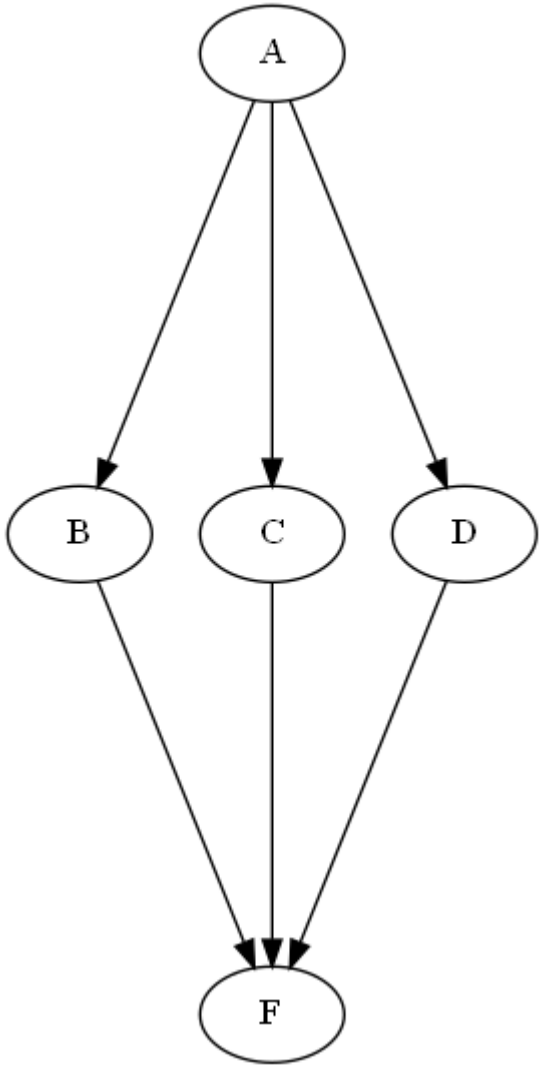
`ranksep=0.5` - длина ребра внутри подграфа

Задание свойств графа

```
digraph {

  A -> {B, C, D} -> F

}
```



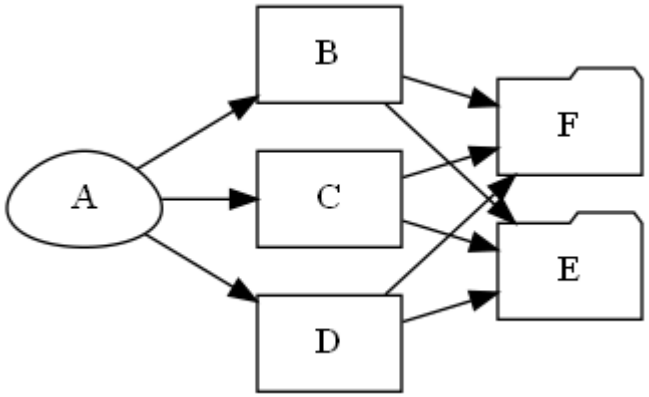

```
digraph {
  graph [ranksep=2]

  A -> {B, C, D} -> F

}
```



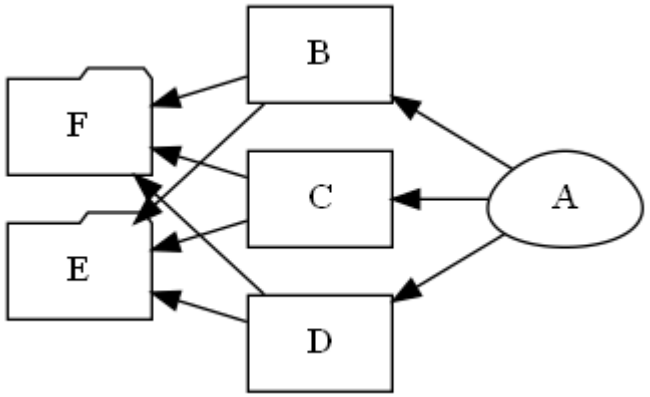
Изменение ориентации графа

```
digraph A {

  rankdir=LR;

  node [shape=egg]
  A
  node [shape=box]
  B; C; D
  node [shape=folder]
  F; E

  A -> {B, C, D} -> {F, E}

}
```

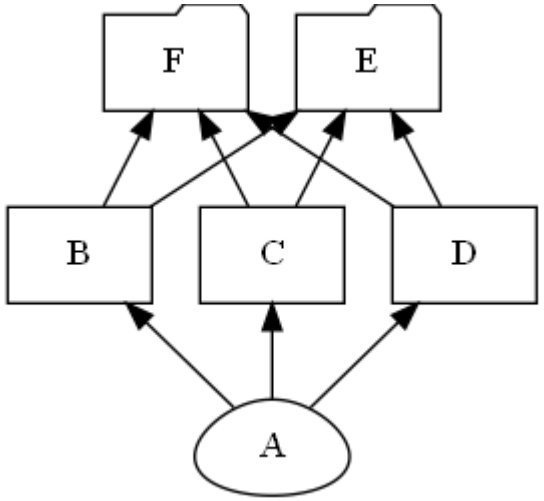
rankdir=LR;



rankdir=RL;

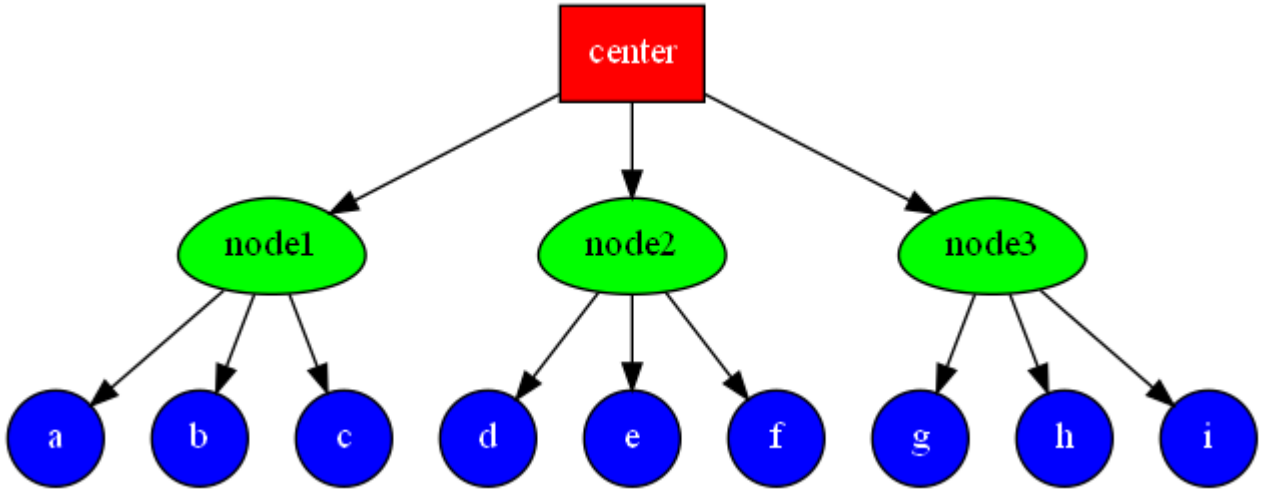


rankdir=BT;



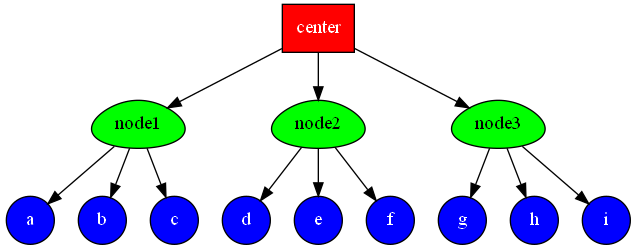
Алгоритмы визуализации графа (1)

```
digraph {
  node [style=filled fillcolor=red shape=box fontcolor=white]
  center
  node [style=filled fillcolor=green shape=egg fontcolor=black]
  node1; node2; node3
  node [style=filled fillcolor=blue shape=circle fontcolor=white]

  center -> node1
  center -> node2
  center -> node3

  node1 -> a
  node1 -> b
  node1 -> c

  node2 -> d
  node2 -> e
  node2 -> f

  node3 -> g
  node3 -> h
  node3 -> i
}
```

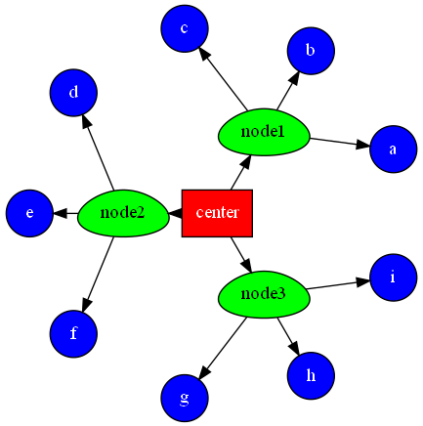


Алгоритмы визуализации графа (2)

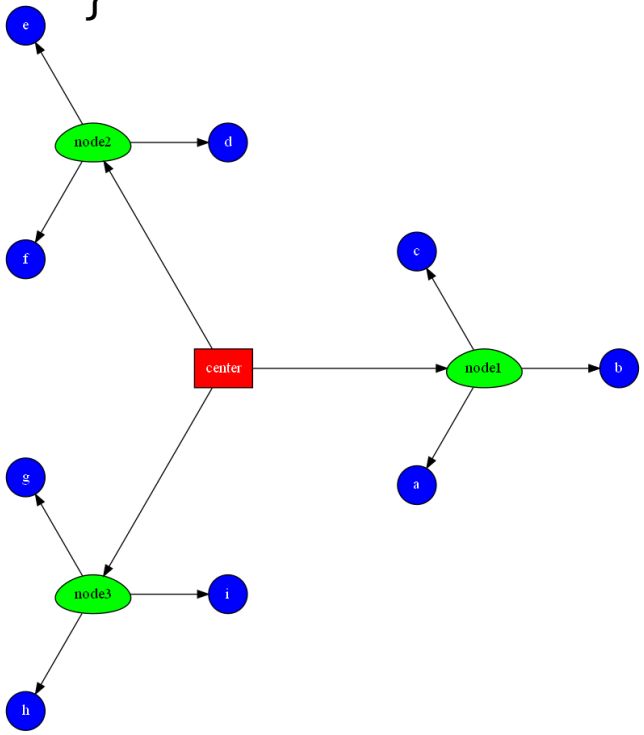
```
digraph {
  graph [layout=dot]
  ...
}
```



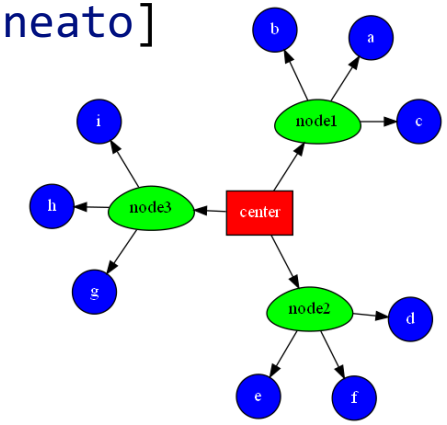

```
digraph {
  graph [layout=twopi]
  ...
}
```




```
digraph {
  graph [layout=circo]
  ...
}
```

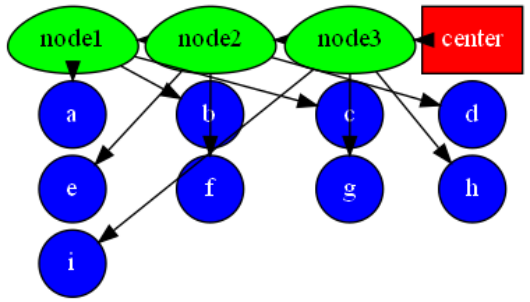



```
digraph {
  graph [layout=neato]
  ...
}
```

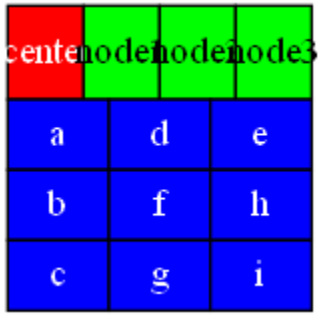


Алгоритмы визуализации графа (3)

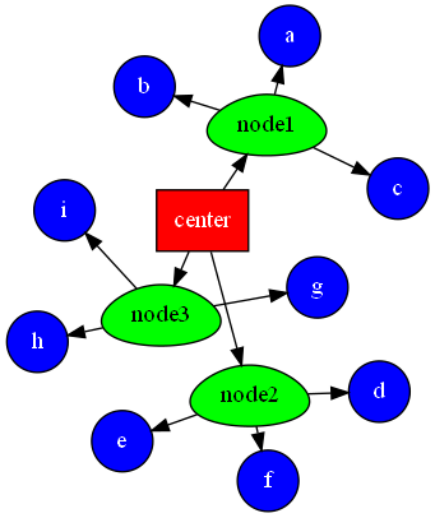
```
digraph {
  graph [layout=osage]
  ...
}
```




```
digraph {
  graph [layout=patchwork]
  ...
}
```

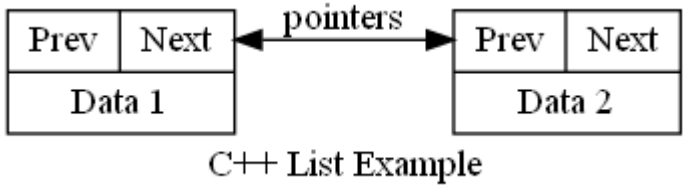



```
digraph {
  graph [layout=fdp]
  ...
}
```

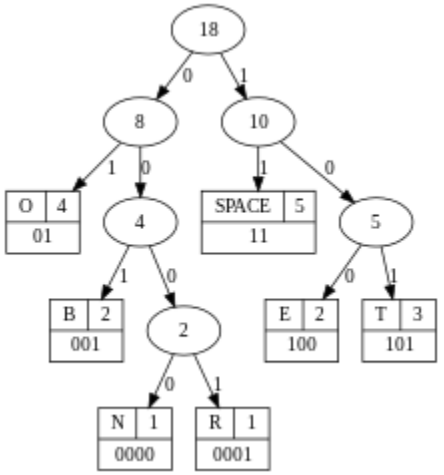


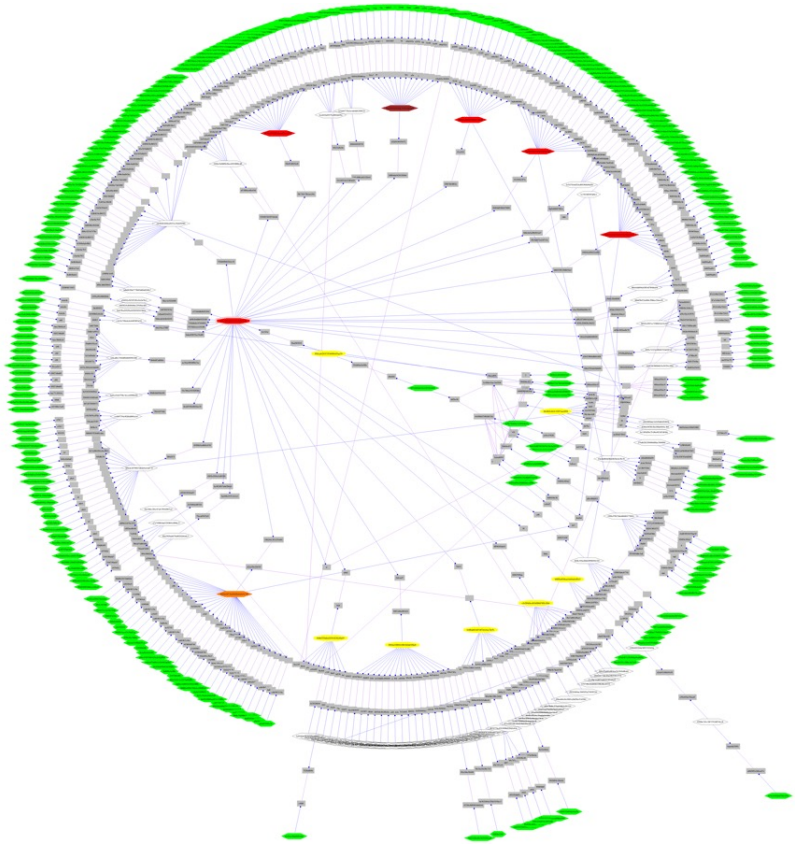
Более сложные примеры (1)

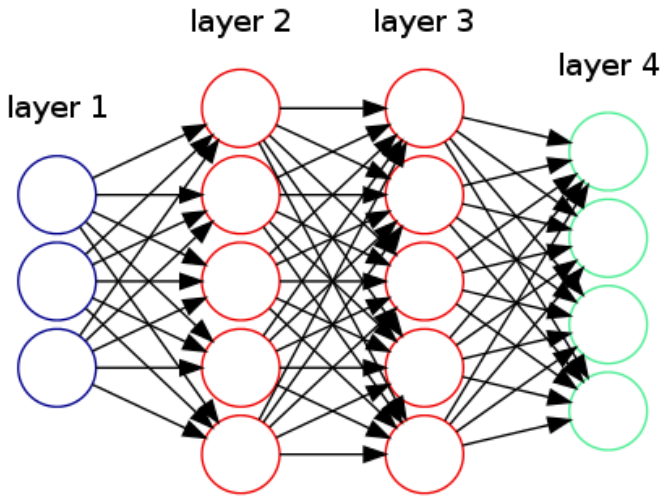
```
digraph {
  graph [label="C++ List Example" rankdir=LR]

  node [shape=record]
  dataA [
    label = "{<prev>Prev|<next>Next}|Data 1"
  ]

  dataB [
    label = "{<prev>Prev|<next>Next}|Data 2"
  ]

  dataA:next -> dataB:prev [label="pointers" dir=both]
}
```

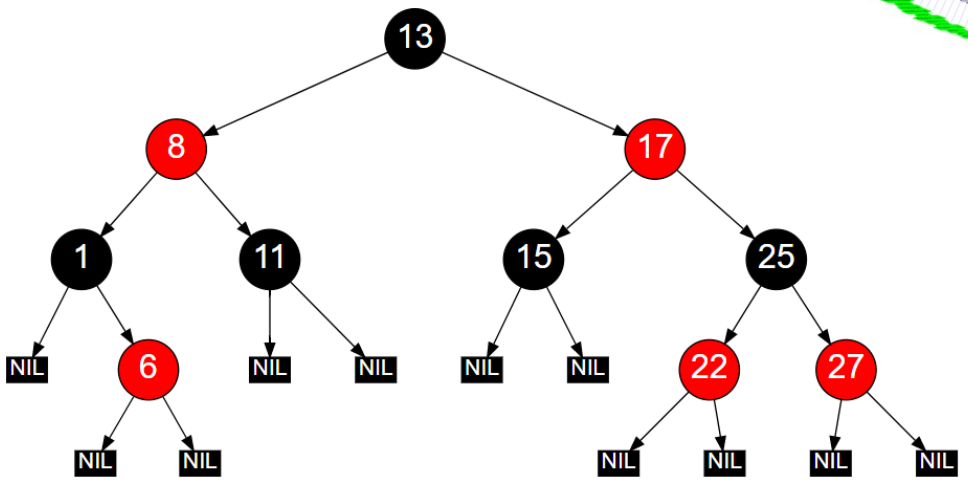


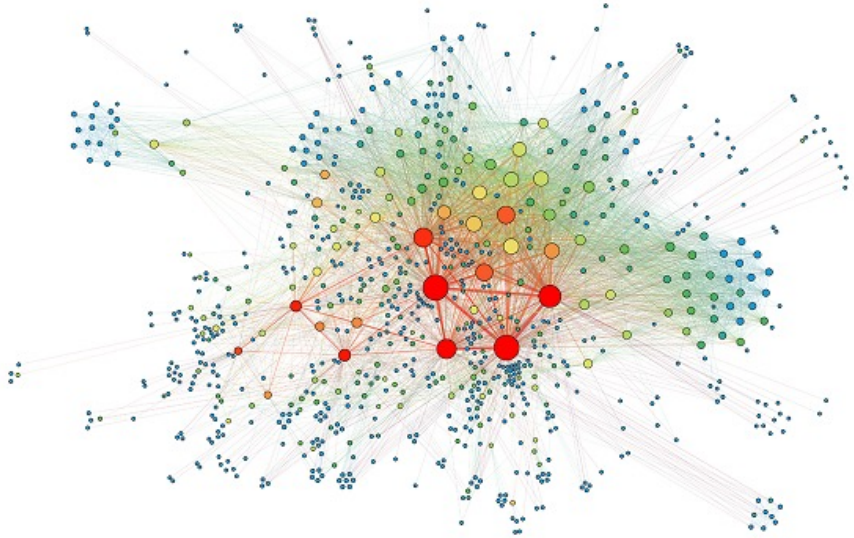
Более сложные примеры (2)





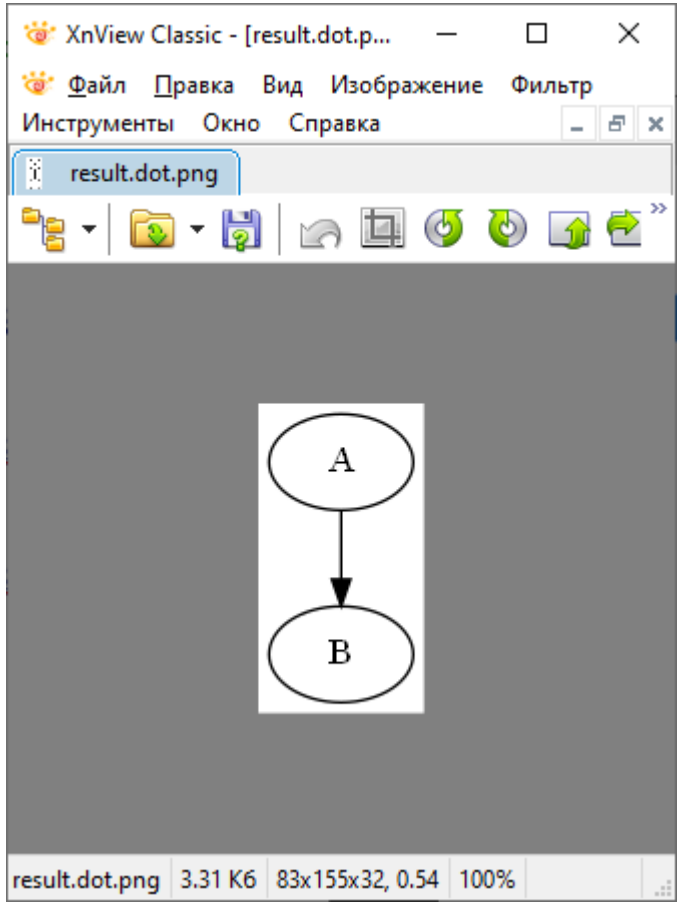






Python: модуль graphviz

```
#!/usr/bin/python

import graphviz

graph = graphviz.Digraph()

graph.edge('A', 'B')

graph.render('result.dot', format='png', view=True)
```



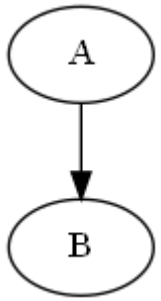
Как работает модуль graphviz

Код .py

```
#!/usr/bin/python

import graphviz

graph = graphviz.Digraph()

graph.edge('A', 'B')

graph.render('result.dot', format='png', view=True)
```

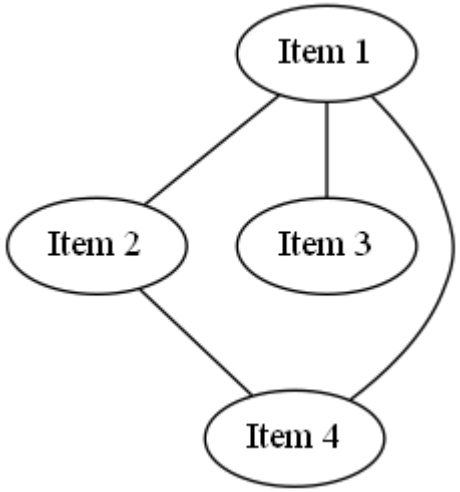
Код .dot

```
digraph {
    A -> B
}
```



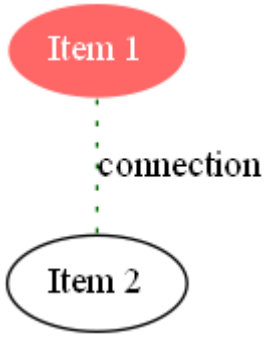
Работа с вершинами и рёбрами

```
#!/usr/bin/python

from graphviz import Graph

graph = Graph()

graph.node('a', 'Item 1')
graph.node('b', 'Item 2')
graph.node('c', 'Item 3')
graph.node('d', 'Item 4')

graph.edge('a', 'b')
graph.edge('a', 'c')
graph.edge('a', 'd')
graph.edge('b', 'd')

graph.render('result.dot', format='png', view=True)
```



Работа с атрибутами вершин в Python (1)

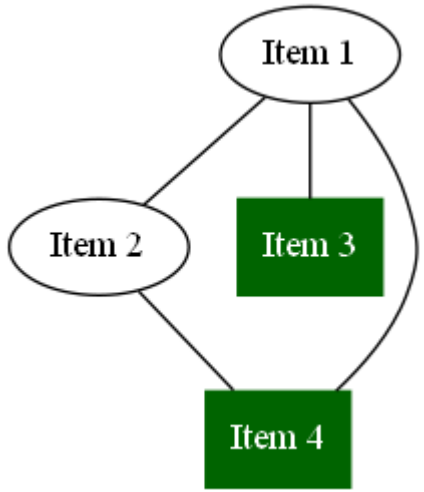
```
graph {
    a [label="Item 1" color=white
       fillcolor="#ff6666" fontcolor=white
       style=filled]

    b [label="Item 2"]
    a -- b [label=connection color=darkgreen
            style="dotted,"]
}
```




```
#!/usr/bin/python

from graphviz import Graph

graph = Graph()

graph.node('a', 'Item 1', style='filled', color='white',
           fontcolor='white', fillcolor='#ff6666')

graph.node('b', 'Item 2')

graph.edge('a', 'b', label="connection", style='dotted,', color='darkgreen')

graph.render('result.dot', format='png', view=True)
```

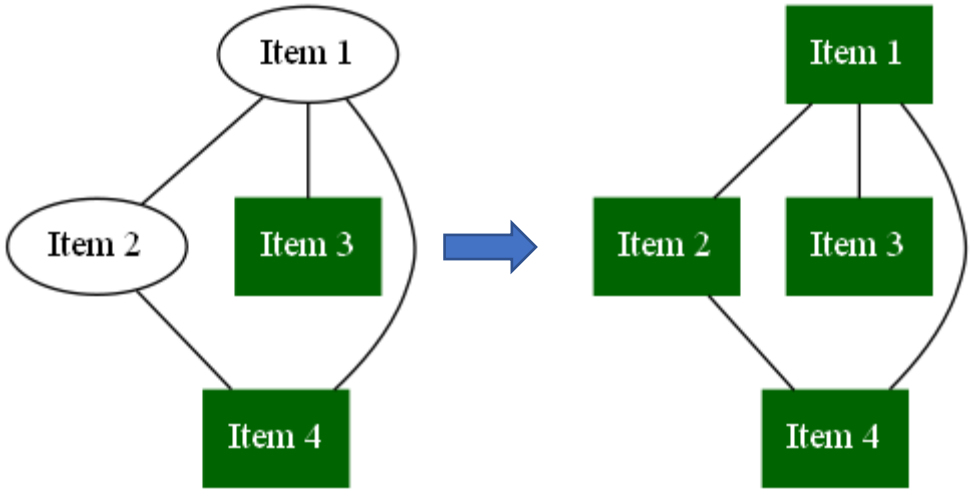
Работа с атрибутами вершин в Python (2)

```
#!/usr/bin/python

from graphviz import Graph

graph = Graph()

graph.node('a', 'Item 1')
graph.node('b', 'Item 2')

graph.attr('node', shape='box', style='filled', color='darkgreen', fontcolor='white')
graph.node('c', 'Item 3')
graph.node('d', 'Item 4')

graph.edge('a', 'b')
graph.edge('a', 'c')
graph.edge('a', 'd')
graph.edge('b', 'd')

graph.render('result.dot', format='png', view=True)
```



Работа с атрибутами вершин в Python (3)

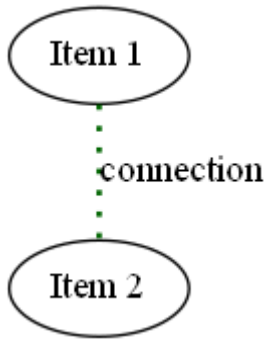
```
...
graph.node('b', 'Item 2')

graph.attr('node', shape='box', style='filled', color='darkgreen', fontcolor='white’)

graph.node('c', 'Item 3’)
...
```




```
...
graph.node('b', 'Item 2')

graph.node_attr['shape'] = 'box'
graph.node_attr['style'] = 'filled'
graph.node_attr['color'] = 'darkgreen'
graph.node_attr['fontcolor'] = 'white'

graph.node('c', 'Item 3’)
...
```

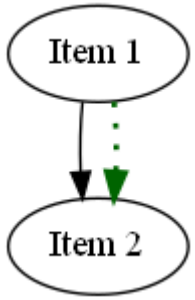
Работа с атрибутами рёбер в Python (1)

```
#!/usr/bin/python

from graphviz import Graph

graph = Graph()

graph.node('a', 'Item 1')
graph.node('b', 'Item 2')

graph.edge('a', 'b', label='connection', style='dotted, bold', color='darkgreen')

graph.render('result.dot', format='png', view=True)
```



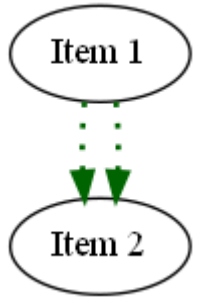
Работа с атрибутами рёбер в Python (2)

```
#!/usr/bin/python

from graphviz import Digraph

graph = Digraph()

graph.node('a', 'Item 1')
graph.node('b', 'Item 2')

graph.edge('a', 'b')
graph.attr('edge', style='dotted, bold', color='darkgreen')
graph.edge('a', 'b')

graph.render('result.dot', format='png', view=True)
```



Работа с атрибутами рёбер в Python (3)

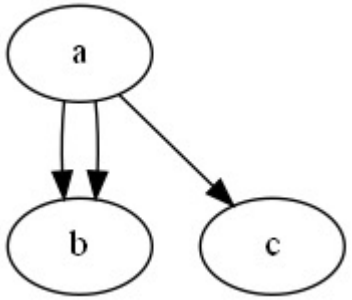
```
#!/usr/bin/python

from graphviz import Digraph

graph = Digraph()

graph.node('a', 'Item 1')
graph.node('b', 'Item 2')

graph.edge('a', 'b')
graph.edge_attr['style'] = 'dotted, bold'
graph.edge_attr['color'] = 'darkgreen'
graph.edge('a', 'b')

graph.render('result.dot', format='png', view=True)
```



Задание атрибутов графа

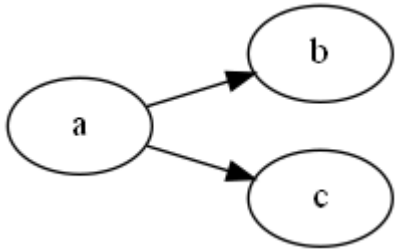
```python
from graphviz import Digraph

graph = Digraph()

graph.edge('a', 'b')
graph.edge('a', 'b')
graph.edge('a', 'c')

graph.render('result.dot', format='png', view=True)
```

```
digraph {
    a -> b
    a -> b
    a -> c
}
```




```python
from graphviz import Digraph

graph = Digraph(strict='true')
graph.graph_attr['rankdir'] = 'LR'

graph.edge('a', 'b')
graph.edge('a', 'b')
graph.edge('a', 'c')

graph.render('result.dot', format='png', view=True)
```

```
strict digraph {
    graph [rankdir=LR]
    a -> b
    a -> b
    a -> c
}
```



Задание на лабораторную работу (1)

Разработать программу на языке Python с использованием библиотеки graphviz, которая:

1. зачитает из входного файла нетлист на языке spice, среди элементов которого содержатся только двухполюсники и который не содержит иерархию (нет .subckt);
2. сформирует перечень элементов схемы с использованием библиотеки graphviz и выведет его;

Вся информация об элементах и узлах обрабатывается через регулярные выражения и списки.

Уровень «минимум»

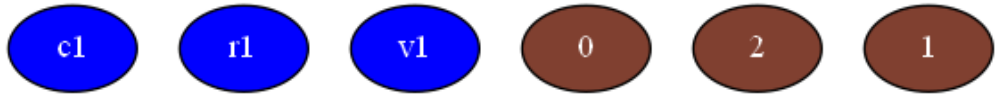
1. На результирующей картинке должны присутствовать вершины, соответствующие элементам (значения в узлах графов должны быть именами элементов);
2. На результирующей картинке должны присутствовать вершины, соответствующие узлам, (имена узлов также отображаются в узлах графа);
3. Вершины графа не соединены между собой, но разукрашены в разные цвета в зависимости от типа: элементы или узлы (см. пример).
4. Расположение элементов - как у меня (в одну строку, сначала элементы, затем узлы).

Пример

Нетлист:

```
v1 1 0 5
r1 1 2 1k
c1 2 0 1p
.tran 100n
.end
```

Результат:



Задание на лабораторную работу (2)

Разработать программу на языке Python с использованием библиотеки graphviz, которая:

1. зачитает из входного файла нетлист на языке spice, среди элементов которого содержатся только двухполюсники и который не содержит иерархию (нет .subckt);
2. сформирует перечень элементов схемы с использованием библиотеки graphviz и выведет его;

Вся информация об элементах и узлах обрабатывается через регулярные выражения и списки.

Уровень «норм»

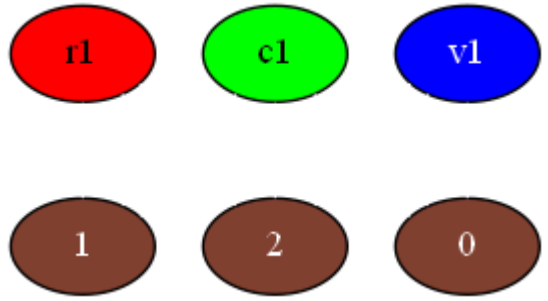
1. На результирующей картинке должны присутствовать вершины, соответствующие элементам (значения в узлах графов должны быть именами элементов);
2. На результирующей картинке должны присутствовать вершины, соответствующие узлам, (имена узлов также отображаются в узлах графа);
3. Вершины графа не соединены между собой, но узлы, соответствующие элементам, раскрашены в зависимости от типа элемента (см. пример).
4. Расположение элементов - как у меня (в две строки, вверху элементы, внизу узлы).

Пример

Нетлист:

```
v1 1 0 5
r1 1 2 1k
c1 2 0 1p
.tran 100n
.end
```

Результат:



Задание на лабораторную работу (3)

Разработать программу на языке Python с использованием библиотеки graphviz, которая:

1. зачитает из входного файла нетлист на языке spice, среди элементов которого содержатся только двухполюсники и который не содержит иерархию (нет .subckt);
2. сформирует перечень элементов схемы с использованием библиотеки graphviz и выведет его;

Вся информация об элементах и узлах обрабатывается через регулярные выражения и списки.

Уровень «крутяк»

1. На результирующей картинке должны присутствовать вершины, соответствующие узлам схемы (значения в вершинах графов должны быть именами узлов схемы);
2. На результирующей картинке должны присутствовать рёбра, соединяющие вершины-узлы (имена элементов отображаются как подписи к рёбрам).

Пример

Нетлист:

```
v1 1 0 5
r1 1 2 1k
c1 2 0 1p
.tran 100n
.end
```

Результат: