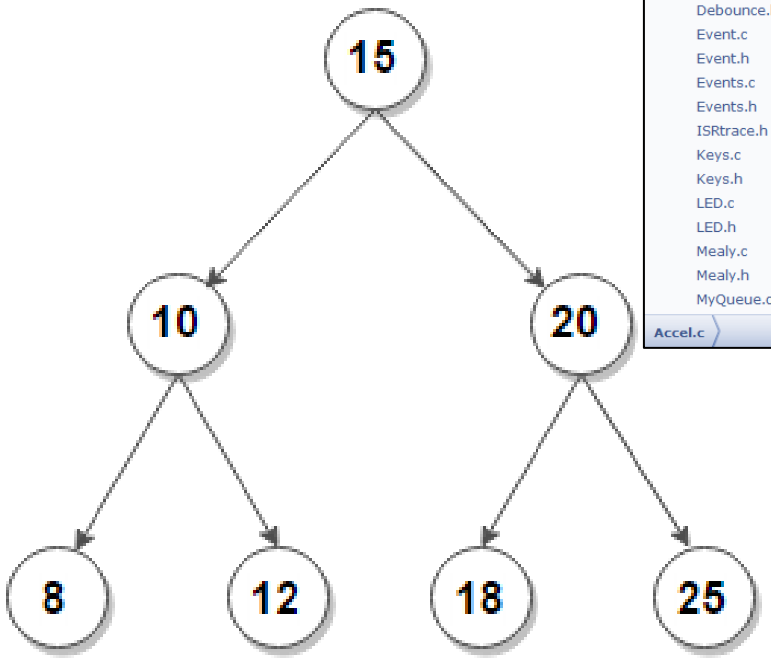
МИЭТ
Национальный исследовательский университет «МИЭТ»
Кафедра ПКИМС

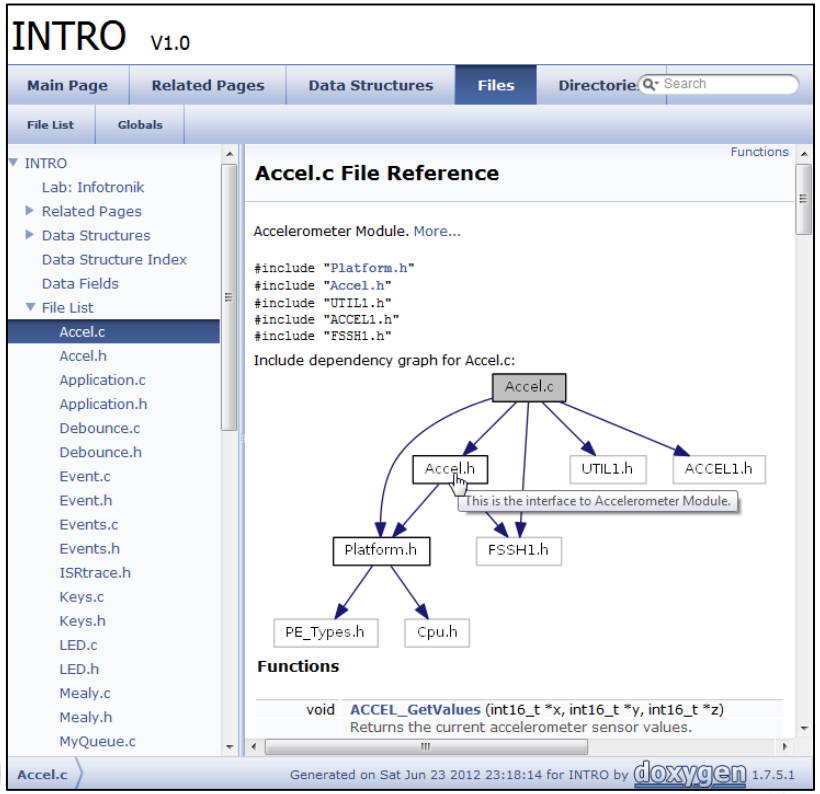
Компьютерные технологии в научных исследованиях

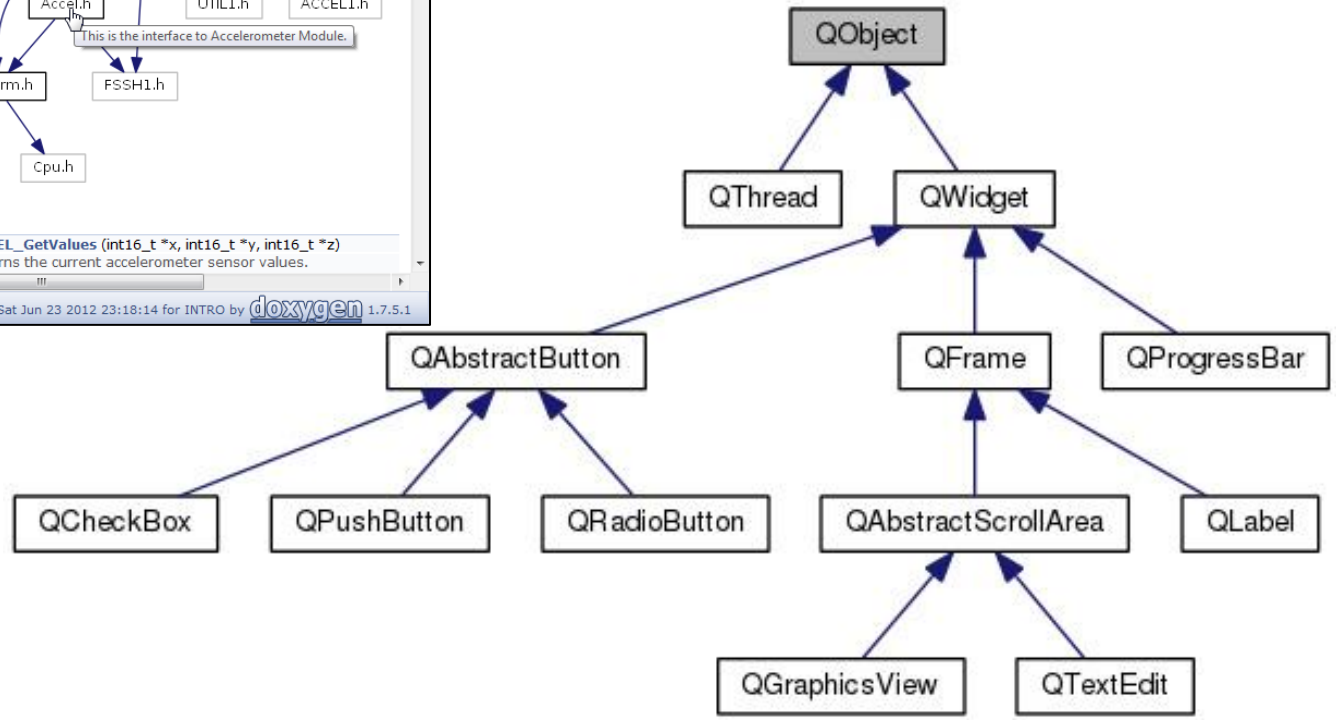
Семинар №5

Работа с пакетом graphviz

Примеры из области программирования

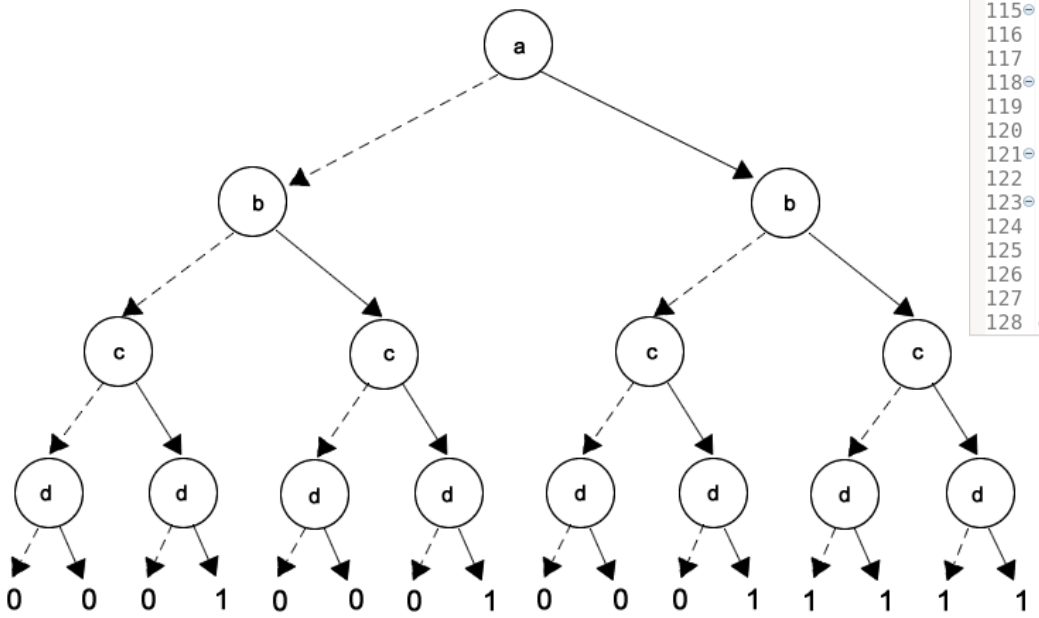






Примеры из области проектирования

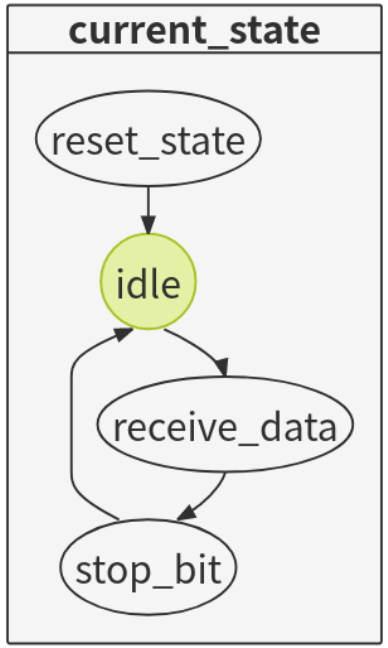
Sigasi;



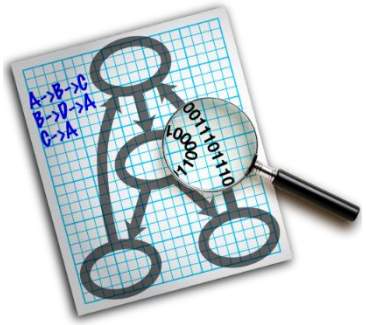
uart_rx.vhd

```
95      rx_control : process (current_state, rx_filtered, data_counter)
96      begin
97          case current_state is
98              when reset_state =>
99                  out_valid <= '0';
100
101                 next_state <= idle;
102             when idle =>
103                 out_valid <= 'X';
104
105                 if (rx_filtered = '0') then
106                     next_state <= receive_data;
107                 else
108                     next_state <= idle;
109                 end if;
110             when receive_data =>
111                 out_valid <= '0';
112
113                 if (data_counter = 7) then
114                     next_state <= stop_bit;
115                 else
116                     next_state <= receive_data;
117                 end if;
118             when stop_bit =>
119                 out_valid <= '1';
120
121                 if (rx_filtered = '1') then
122                     next_state <= idle;
123                 else
124                     next_state <= stop_bit;
125                 end if;
126         end case;
127     end process rx_control;
128 end architecture behavioural;
```

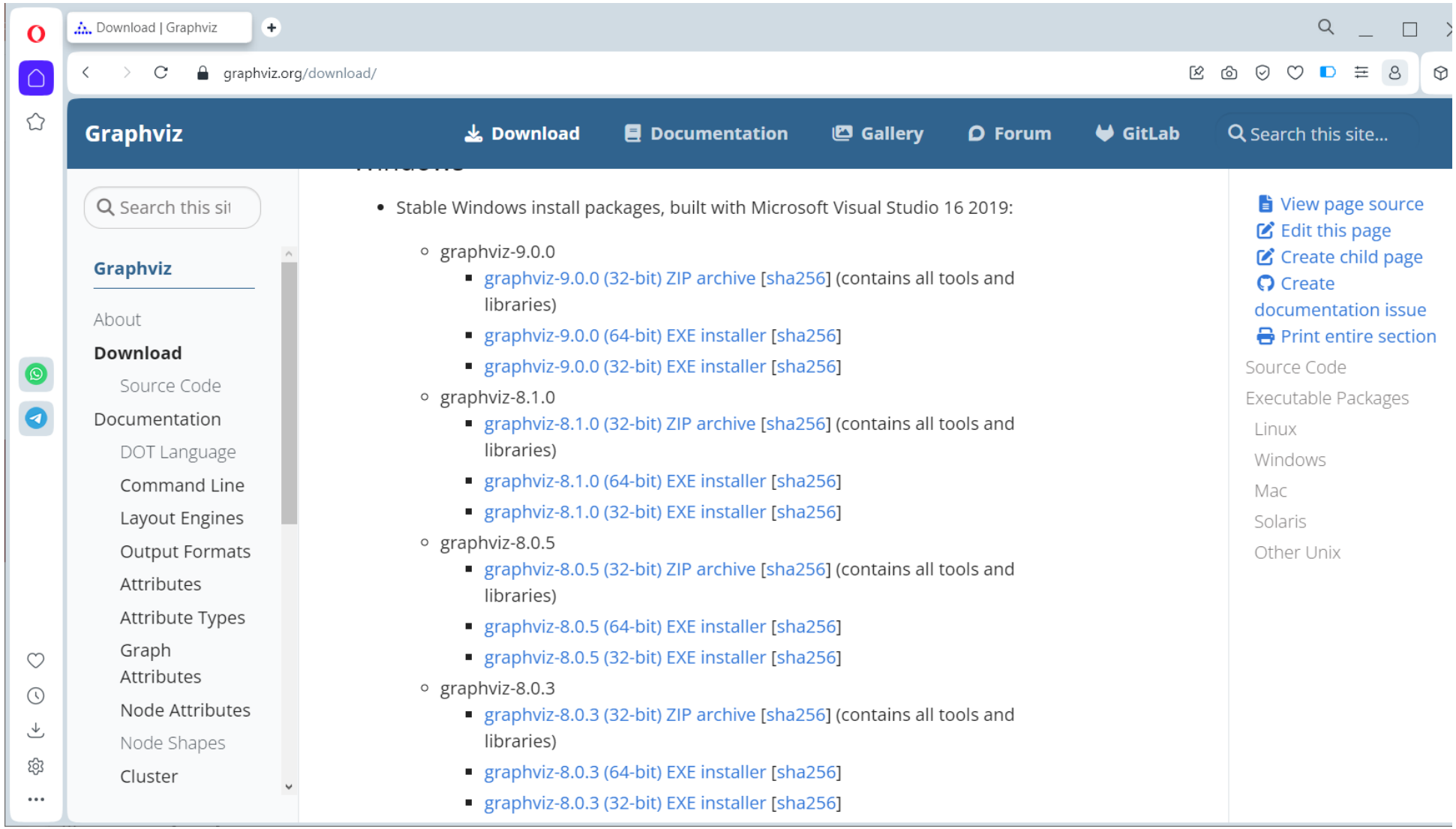
State Machines



Программа Graphviz



URL: https://graphviz.org



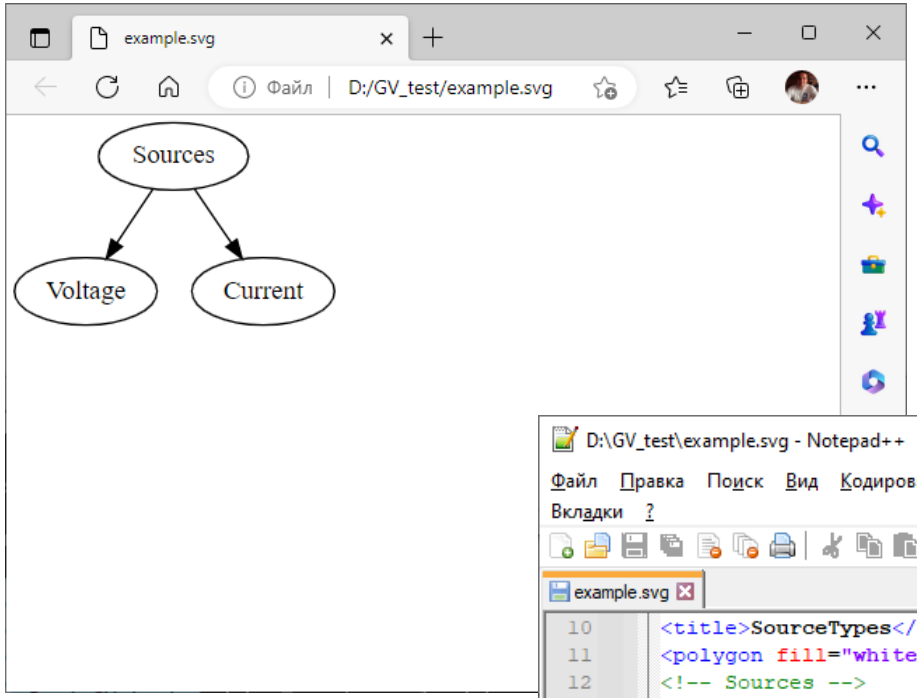
Выходные форматы GraphViz (1)

Форматы изображений:

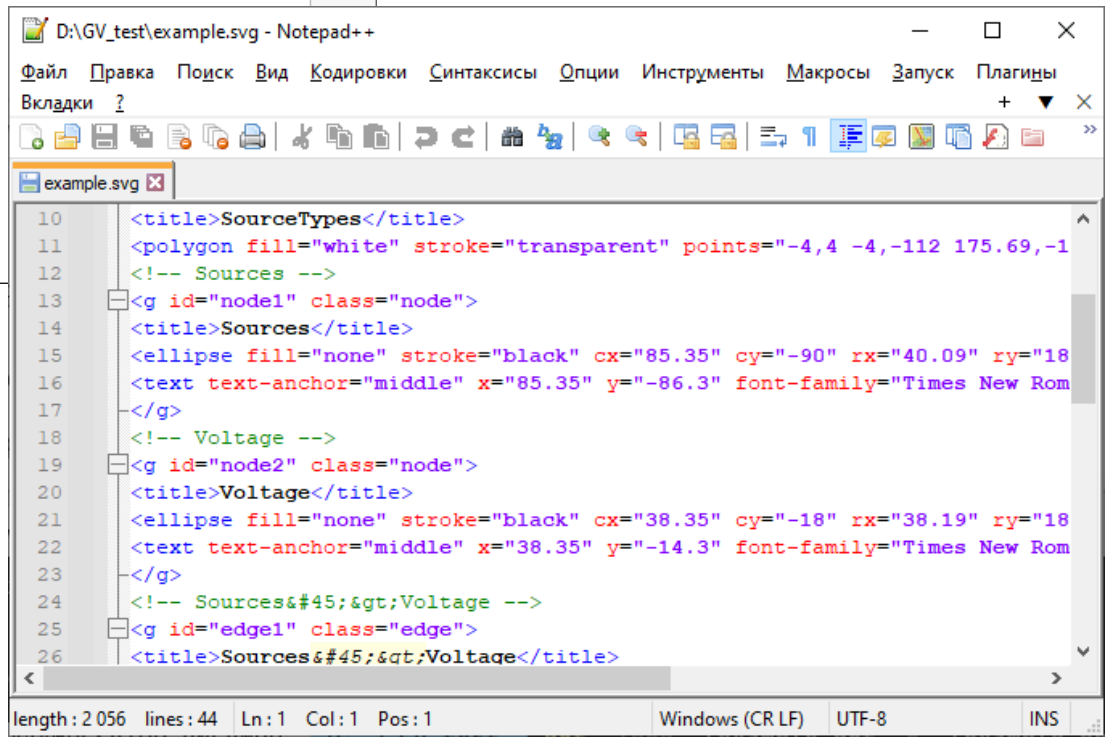
- bmp
- gif
- ico
- jpeg
- png
- psd
- tga
- tiff
- webp

Текстовые форматы изображений:

- fig
- pic
- ps
- svg



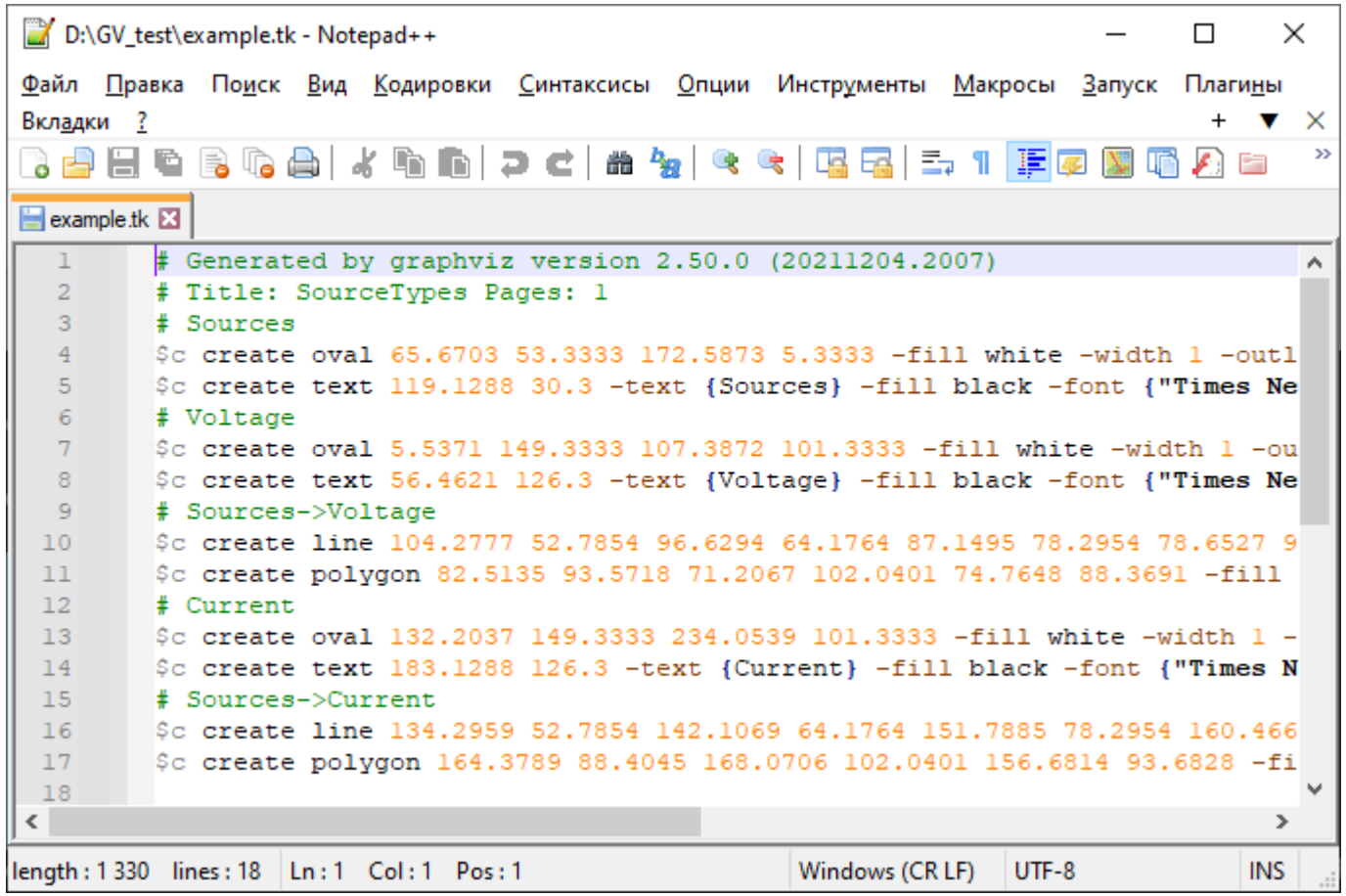

```
dot
-Tpng
<имя файла с графом>
-o
<имя файла с результатом>
```



Выходные форматы GraphViz (2)

Форматы кода:

- ~~gtk~~
- json
- tk

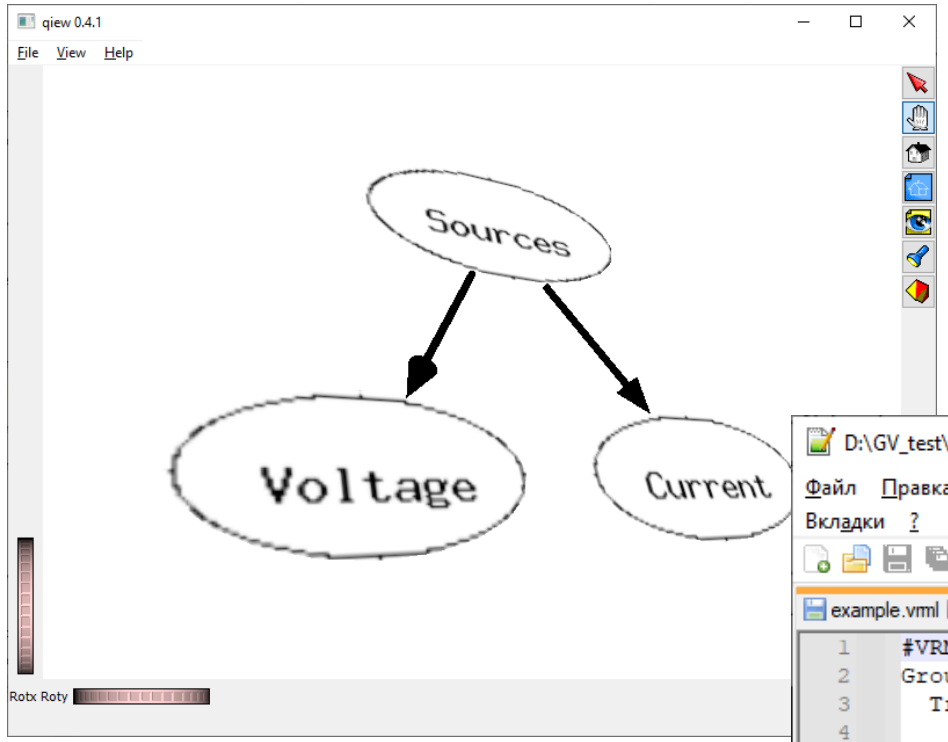



```
dot
-Ttk
<имя файла с графом>
-o
<имя файла с результатом>
```

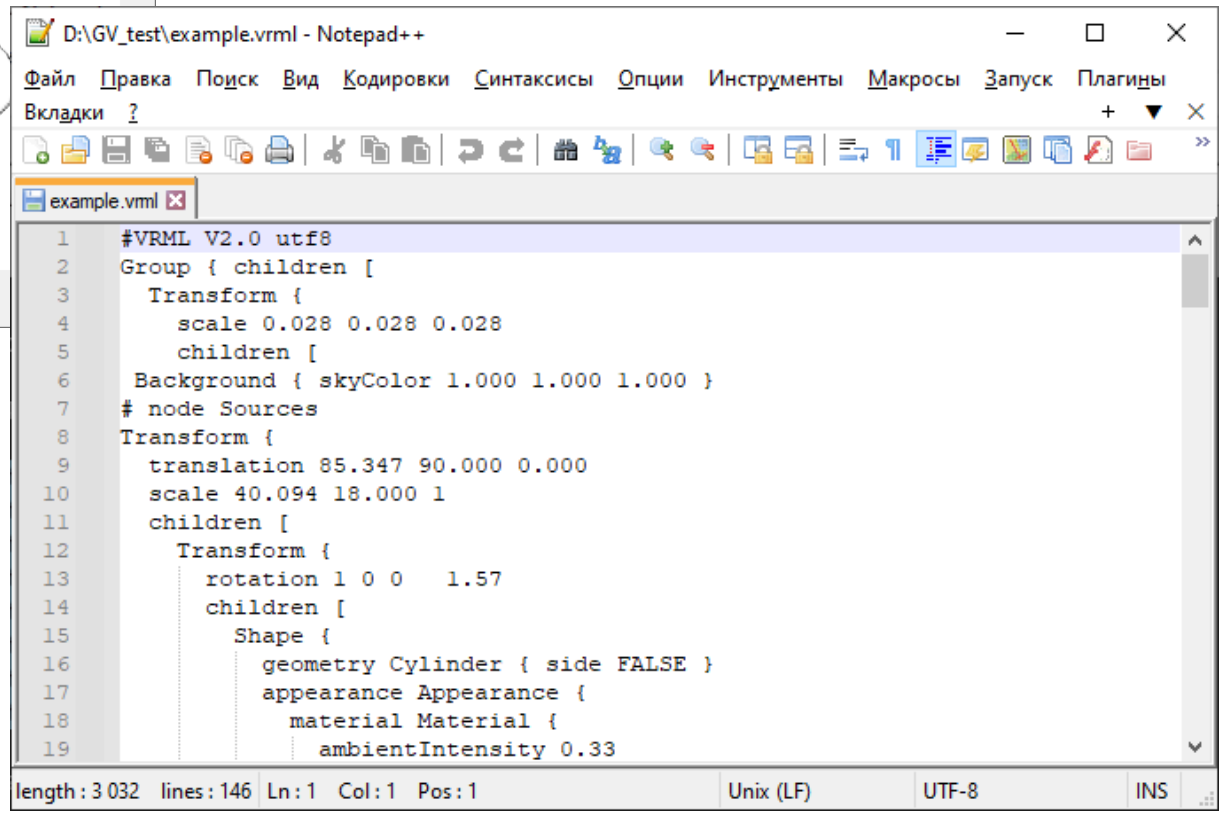
Выходные форматы GraphViz (3)

Форматы кода:

- ~~gtk~~
- json
- tk
- vrml



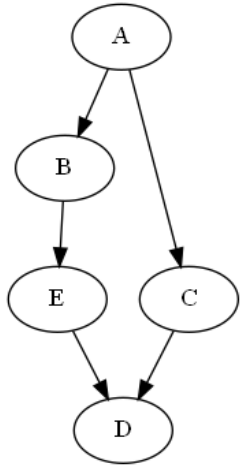

```
dot
-Tvrml
<имя файла с графом>
-o
<имя файла с результатом>
```



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;
}
```

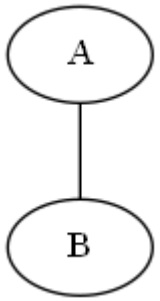


Типы графов: направленный и ненаправленный

graph

Создаёт неориентированный граф:

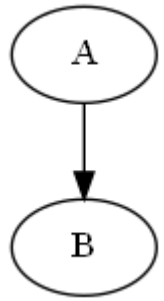
```
graph {
    A -- B
}
```



digraph

Создаёт ориентированный граф

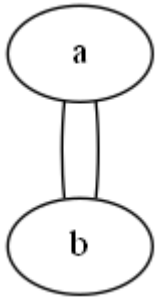
```
digraph {
    A -> B
}
```



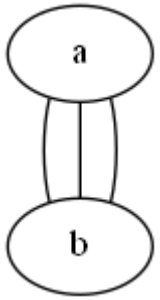
Описание графов

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

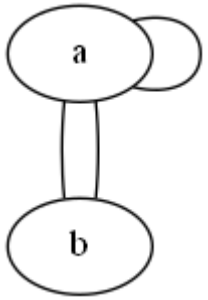
```
graph {
  a -- b
  b -- a
}
```




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

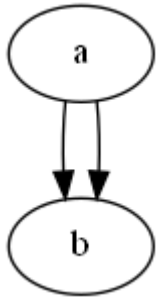



```
graph {
  a -- b
  a -- a
  b -- a
}
```

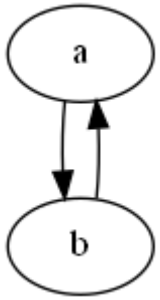


Описание орграфов

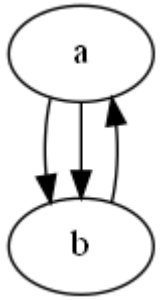
```
digraph {
  a -> b
  a -> b
}
```



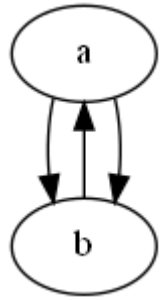

```
digraph {
  a -> b
  b -> a
}
```



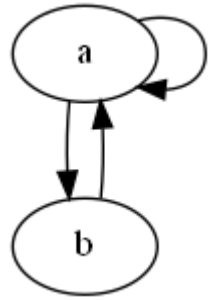

```
digraph {
  a -> b
  a -> b
  b -> a
}
```




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

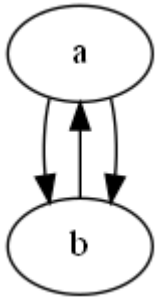



```
digraph {
  a -> b
  b -> a
  a -> a
}
```

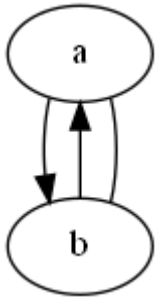


Атрибуты. Явное задание ориентации ребра

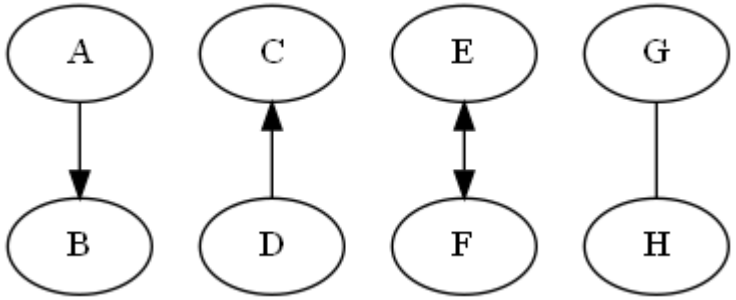
```
digraph {
  a -> b
  b -> a
  a -> b
}
```




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




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



Возможные значения атрибута dir (применим только для рёбер):

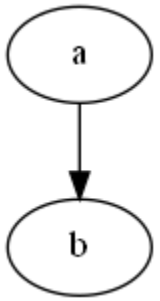
`forward` - значение по умолчанию для орграфов
`back`
`both`
`none` - значение по умолчанию для неориентированных графов

Задание вида рёбер

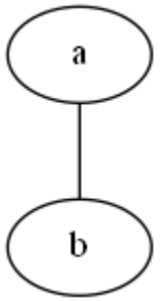
Форма указателя стрелки ребра

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

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



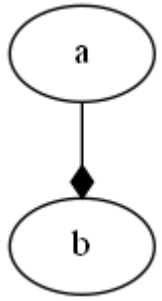

```
digraph {
  a -> b [arrowhead=none]
}
```



Форма указателя хвоста ребра

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

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

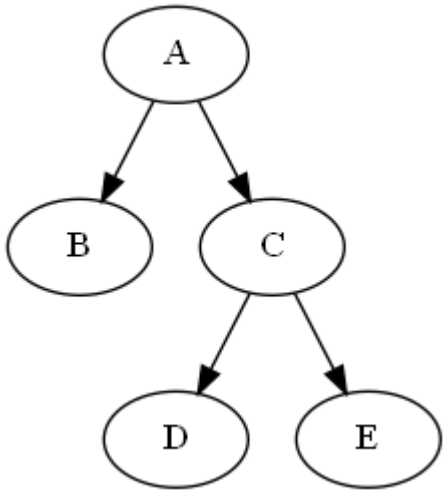


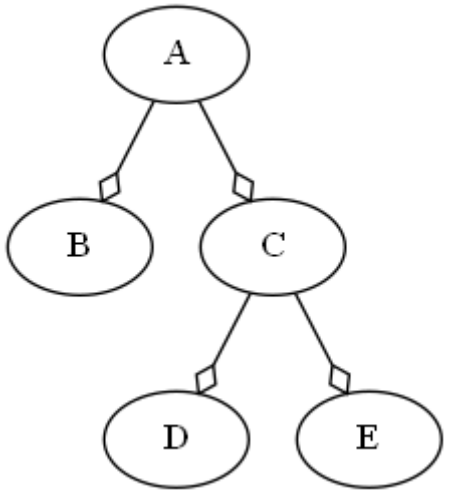
Описание для группы рёбер

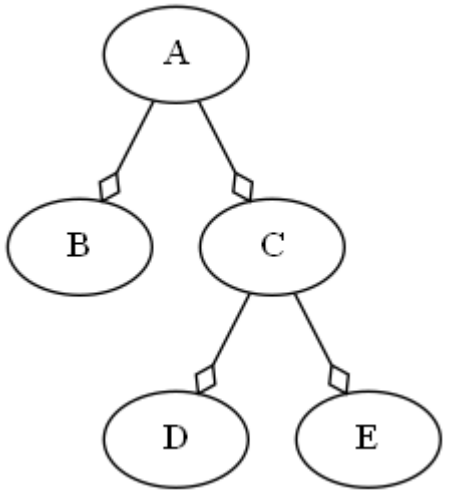
```
digraph {
  A -> B
  A -> C
  C -> D
  C -> E
}
```

```
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
}
```

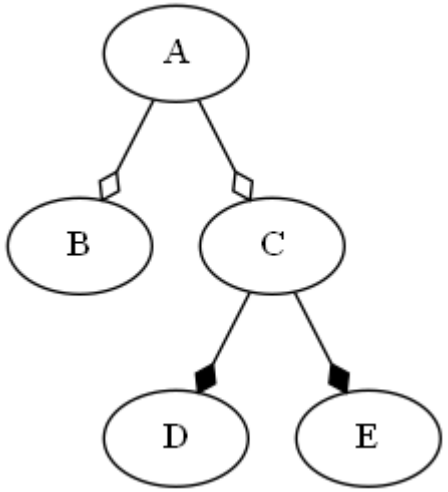




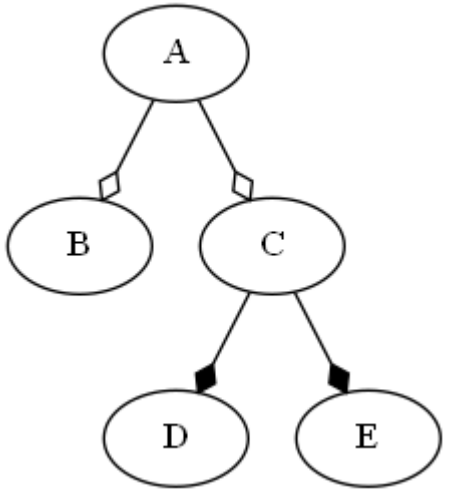


Описание рёбер (1)

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

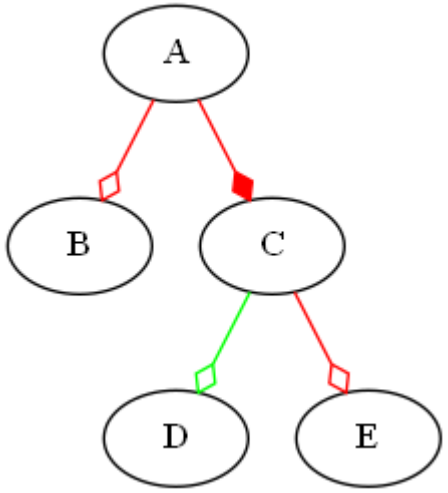



```
digraph {
  edge   [arrowhead=odiamond]
  A -> B
  A -> C
  edge   [arrowhead=diamond]
  C -> D
  C -> E
}
```

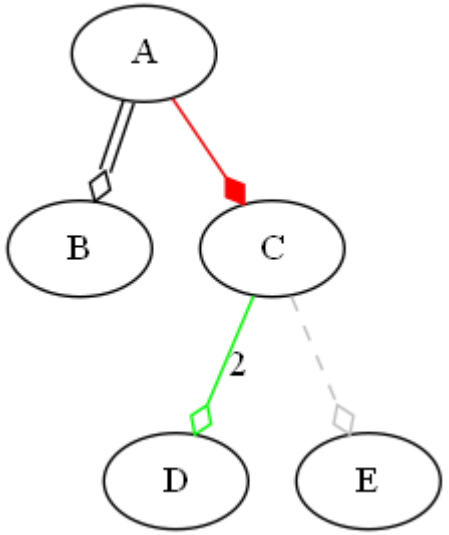


Описание рёбер (2)

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

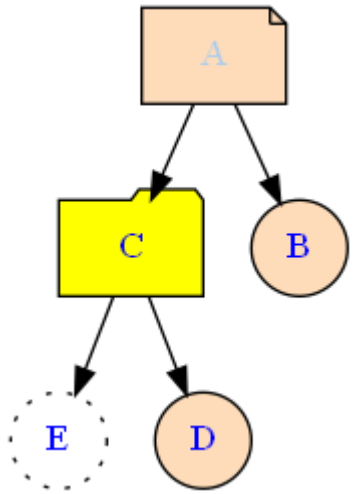



```
digraph {
  edge    [arrowhead=odiamond, color=red]
  A -> B [color="black:invis:black"]
  A -> C [arrowhead=diamond]
  C -> D [color=green, label="2"]
  C -> E [style=dashed, color=grey]
}
```



Задание вида узлов

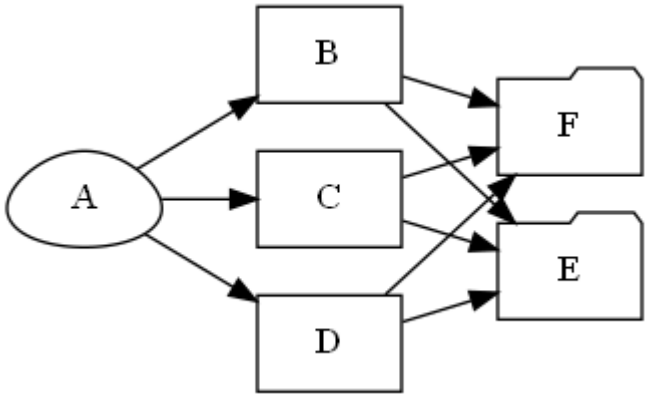
```
digraph {
  node [fontcolor=blue shape=circle style=filled fillcolor="#fedcba"]

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

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



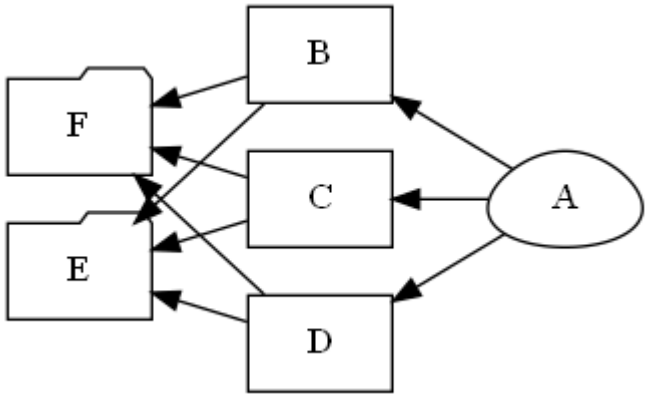
Изменение ориентации графа

```
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}

}
```

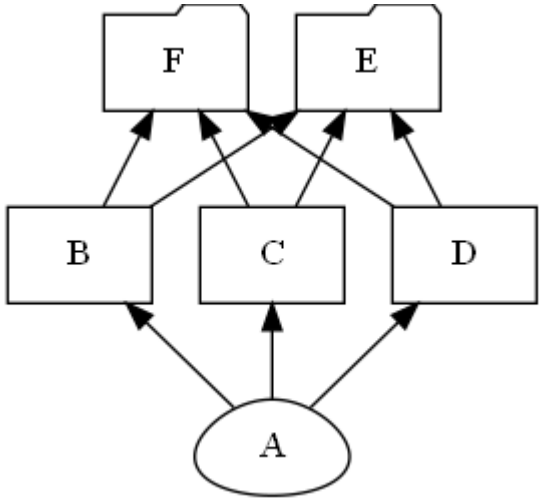
rankdir=LR;



rankdir=RL;

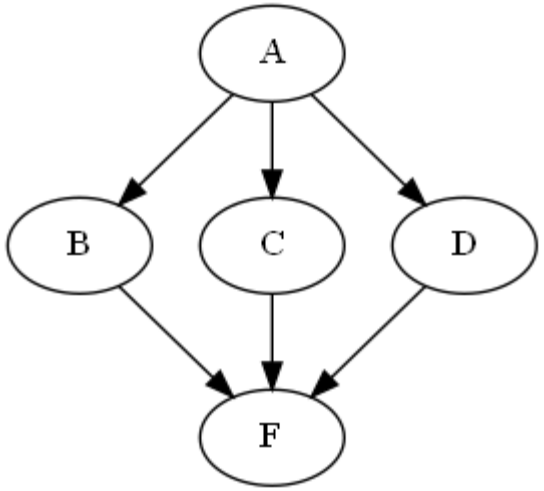


rankdir=BT;

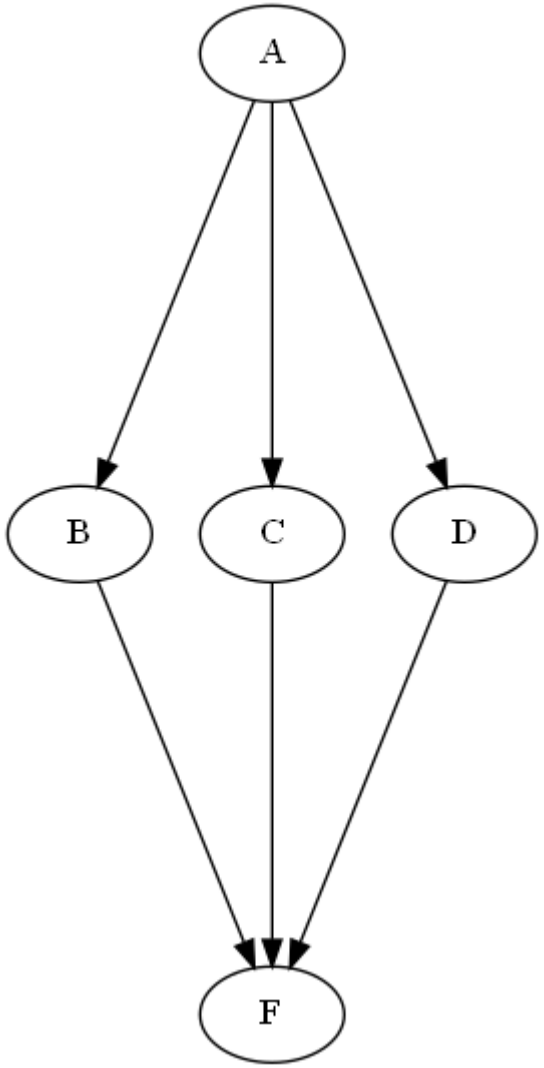


Задание длин рёбер графа

```
digraph {

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

}
```

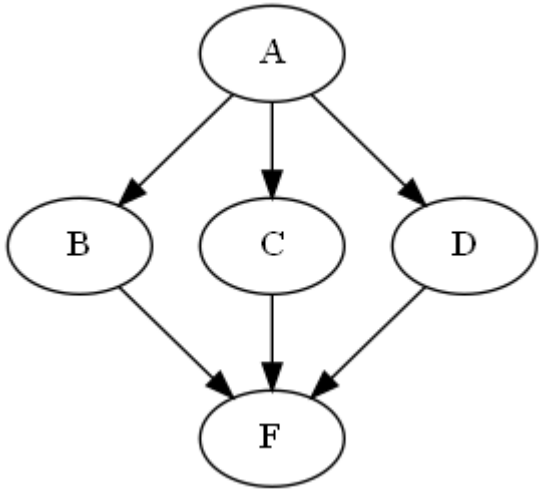



```
digraph {
  graph [ranksep=2]

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

}
```

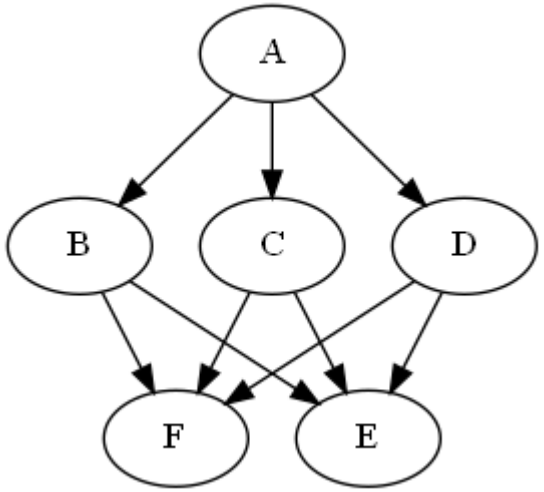


Группирование узлов

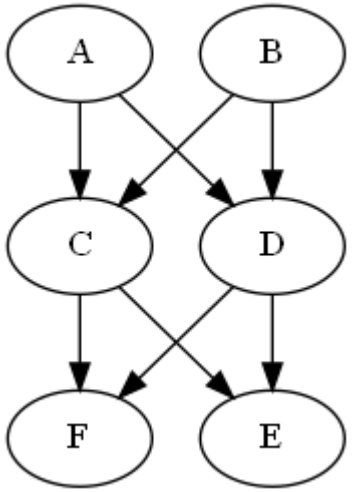
```
digraph {

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

}
```




```
digraph {

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

}
```

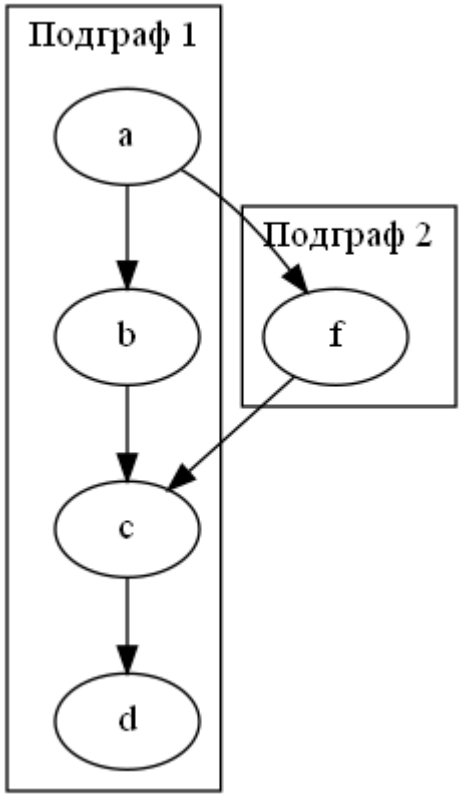



```
digraph {

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

}
```



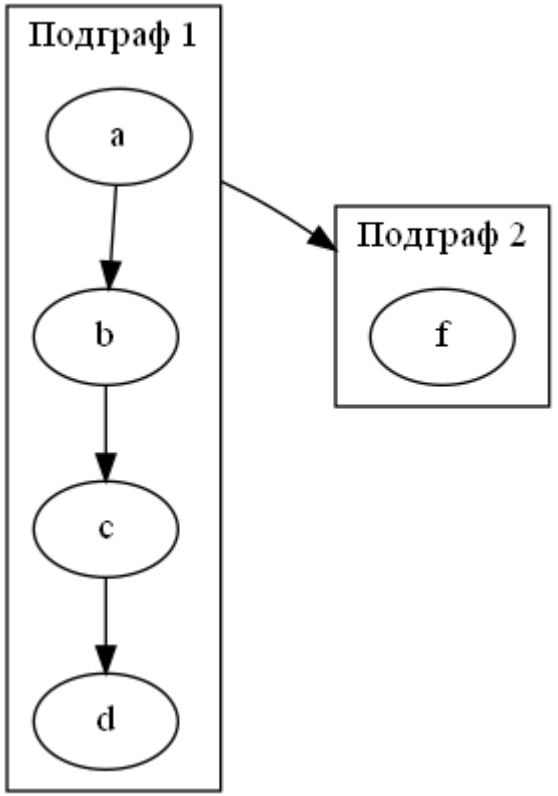
Группирование узлов подграфами (1)

```
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]
}
```



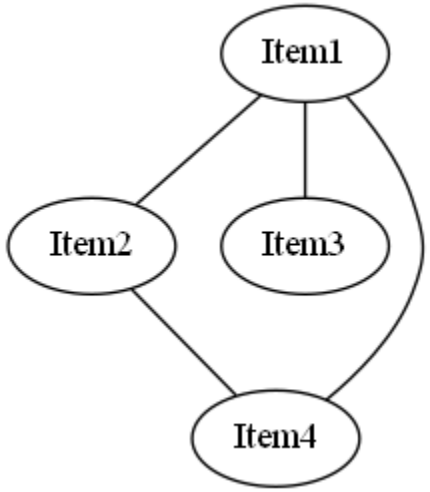
`compound=true` - разрешить связывать подграфы

`nodesep=1` - длина ребра между подграфами

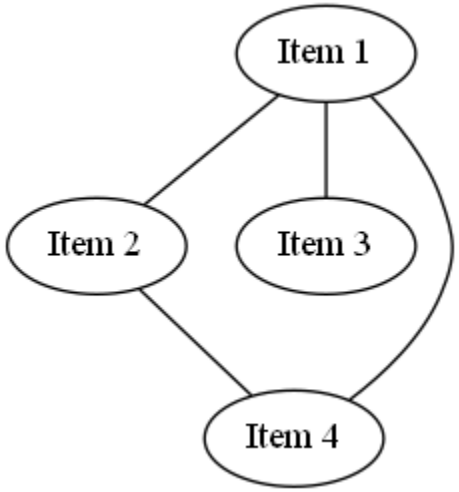
`ranksep=0.5` - длина ребра внутри подграфа

Именование узлов

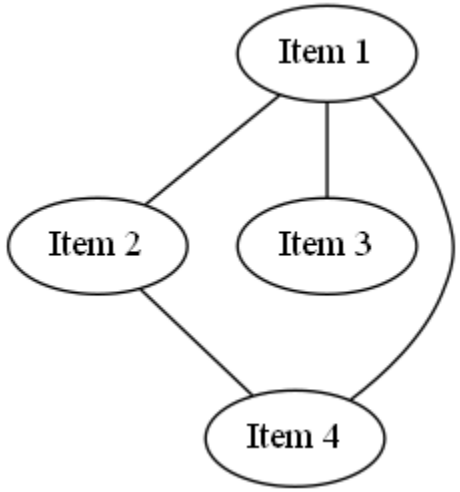
```
graph {
    Item1
    Item2
    Item3
    Item4
    Item1 -- Item2
    Item1 -- Item3
    Item1 -- Item4
    Item2 -- Item4
}
```




```
graph {
    "Item 1"
    "Item 2"
    "Item 3"
    "Item 4"
    "Item 1" -- "Item 2"
    "Item 1" -- "Item 3"
    "Item 1" -- "Item 4"
    "Item 2" -- "Item 4"
}
```

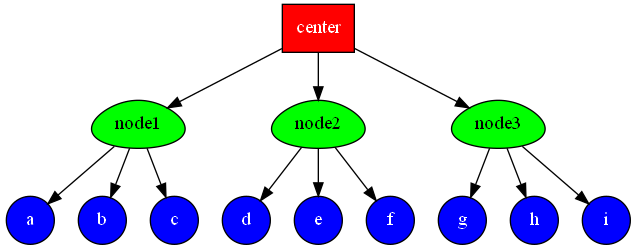



```
graph {
    a [label="Item 1"]
    b [label="Item 2"]
    c [label="Item 3"]
    d [label="Item 4"]
    a -- b
    a -- c
    a -- d
    b -- d
}
```

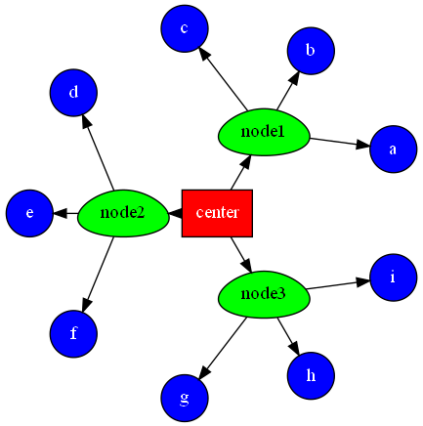


Алгоритмы визуализации графа (1)

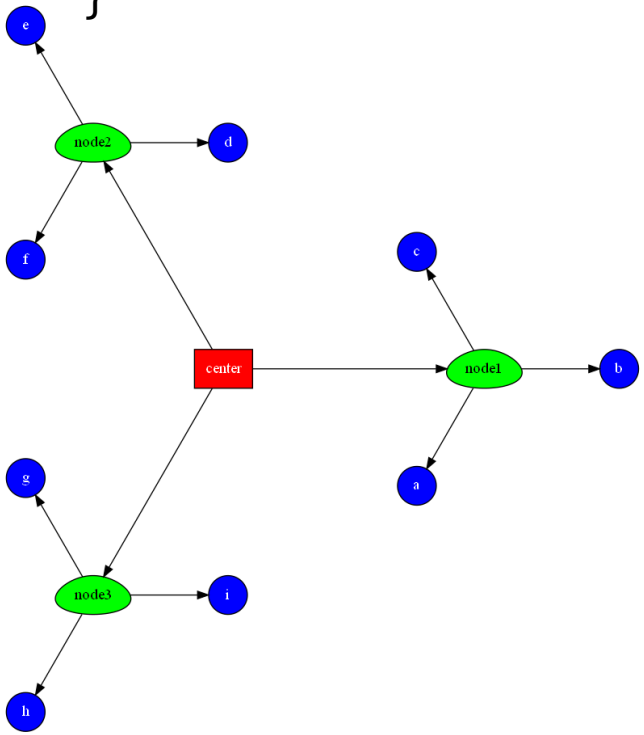
```
digraph {
  graph [layout=dot]
  ...
}
```



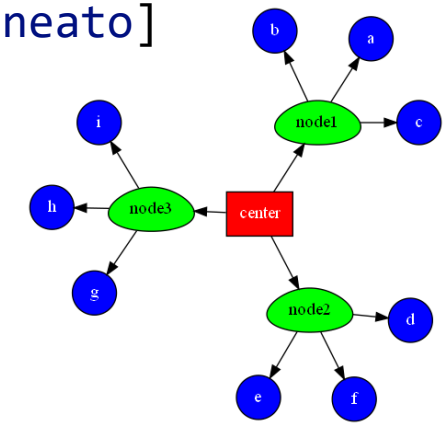

```
digraph {
  graph [layout=twopi]
  ...
}
```




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

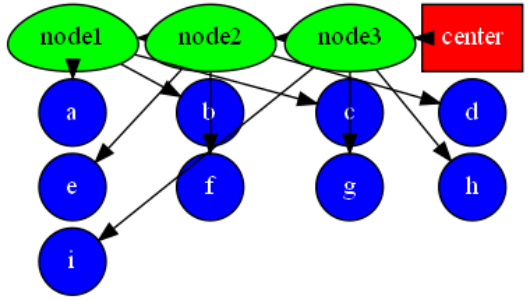



```
digraph {
  graph [layout=neato]
  ...
}
```

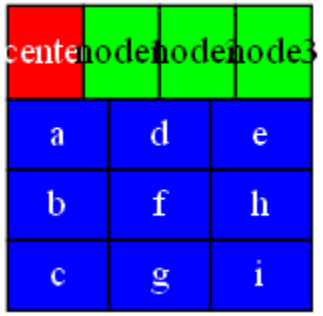


Алгоритмы визуализации графа (2)

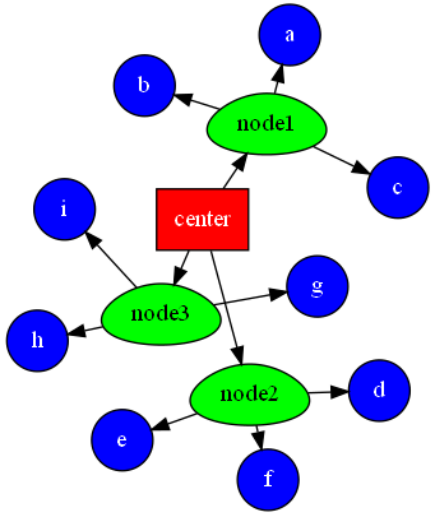
```
digraph {
  graph [layout=osage]
  ...
}
```




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

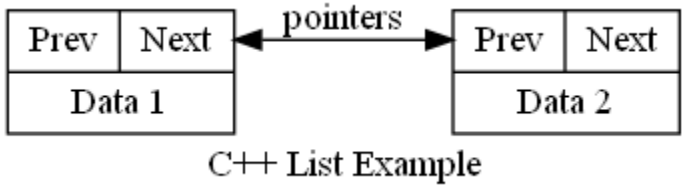



```
digraph {
  graph [layout=fdp]
  ...
}
```



Более сложный пример

```
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]
}
```



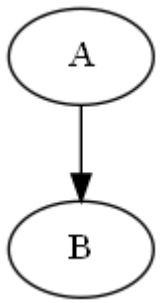
Вызов из других языков программирования: Python

Код .py

```python
#!/usr/bin/python

import graphviz

graph = graphviz.Digraph()

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

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

Код .dot

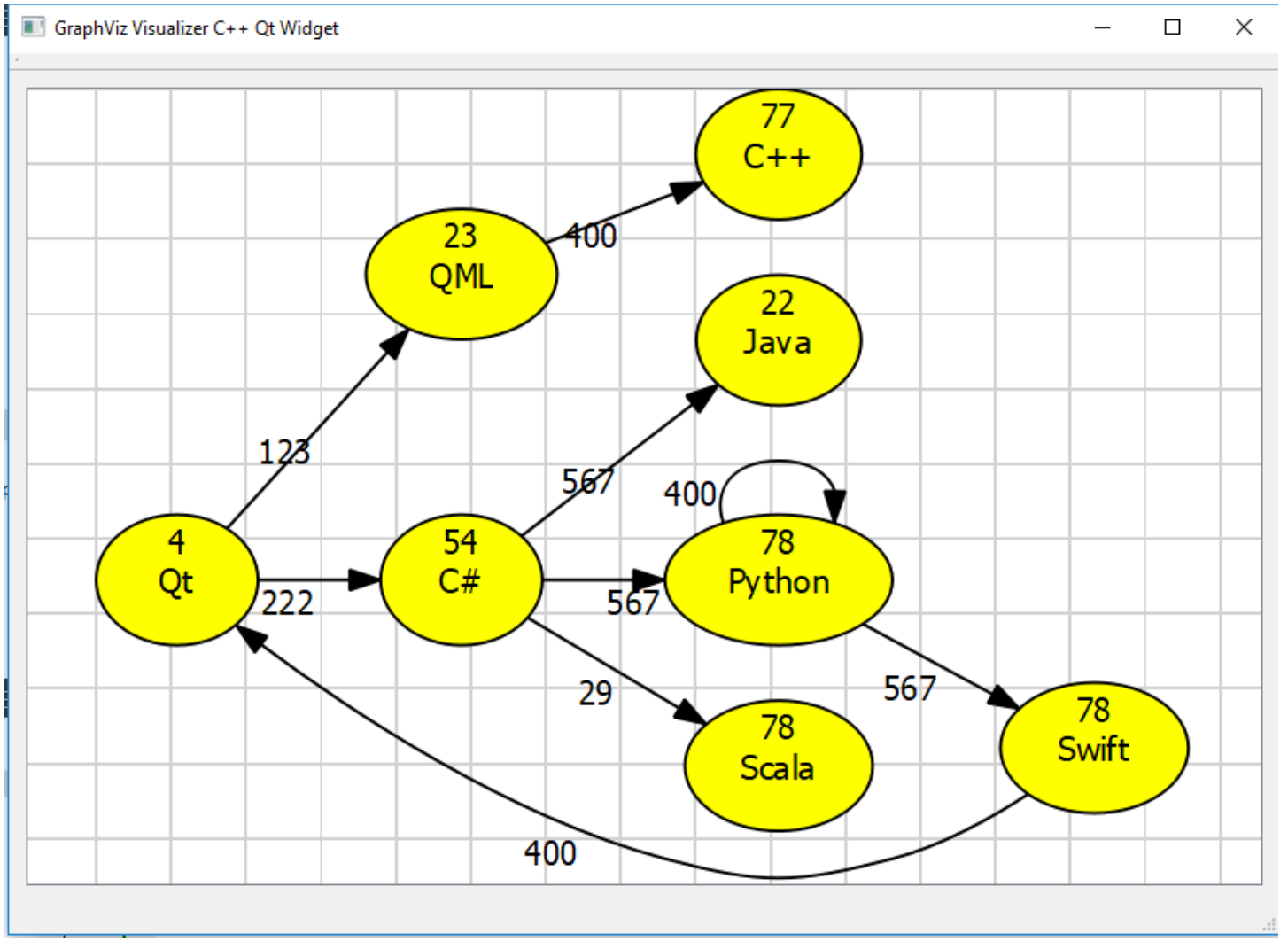
```
digraph {
    A -> B
}
```



Вызов из других языков программирования: C++ и Qt

```cpp
#include "graphvizwrapper.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    GraphvizWrapper graph;

    //Add Nodes Like This
    graph.addNode("Qt" , 4);
    graph.addNode("QML" , 23);
    graph.addNode("C#" , 54);
    graph.addNode("C++" , 77);
    graph.addNode("Java" , 22);
    graph.addNode("Python" , 78);
    graph.addNode("Scala" , 78);
    graph.addNode("Swift" , 78);

    //Add Edges Like This
    graph.addEdge("Qt","C#",222);
    graph.addEdge("Qt","QML",123);
    graph.addEdge("QML","C++",400);
    graph.addEdge("C#","Scala",29);
    graph.addEdge("C#","Python",567);
    graph.addEdge("C#","Java",567);
    graph.addEdge("Python","Swift",567);
    graph.addEdge("Python","Python",400);
    graph.addEdge("Swift","Qt",400);

    //Add Attributes Like This
    graph.addGraphAttribute("rankdir", "LR");
    graph.addNodeAttribute("style", "filled");
    graph.addNodeAttribute("fillcolor", "yellow");
    graph.addNodeAttribute("height", "0.6");
    graph.addNodeAttribute("margin", "0.1");

    graph.draw();
```

1 Issues 2 Search Results 3 Application Output 4 Compile Output 5 Debugger Console 6 General Messages 8 Test Results



Создание графов онлайн