



МИЭТ

Национальный исследовательский университет «МИЭТ»

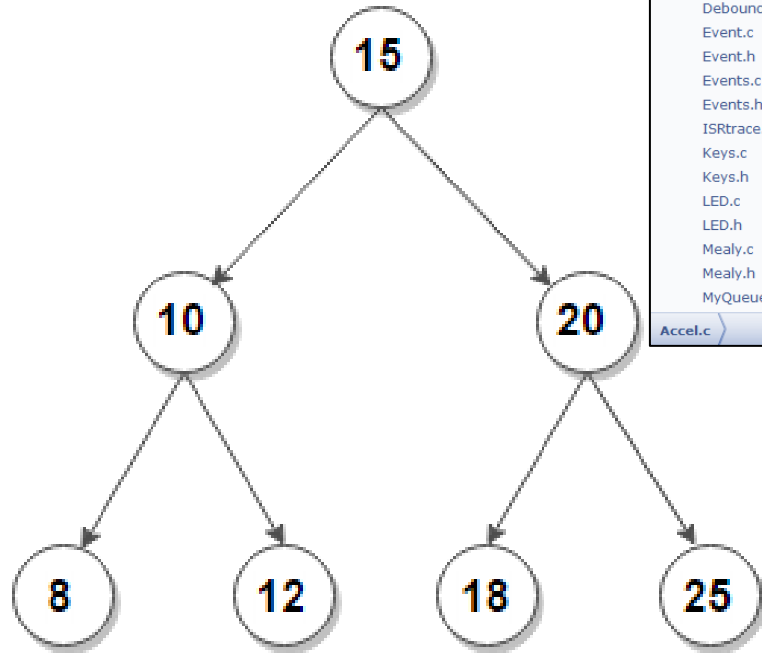
Кафедра ПКИМС

Компьютерные технологии в научных исследованиях

Семинар №5

Работа с пакетом graphviz

Примеры из области программирования



INTRO V1.0

Main Page Related Pages Data Structures **Files** Directorie Search

File List Globals

INTRO

- Lab: Infotronik
- Related Pages
- Data Structures
- Data Structure Index
- Data Fields
- File List
 - Accel.c**
 - Accel.h
 - Application.c
 - Application.h
 - Debounce.c
 - Debounce.h
 - Event.c
 - Event.h
 - Events.c
 - Events.h
 - ISRtrace.h
 - Keys.c
 - Keys.h
 - LED.c
 - LED.h
 - Mealy.c
 - Mealy.h
 - MyQueue.c

Accel.c File Reference

Accelerometer Module. More...

```
#include "Platform.h"
#include "Accel.h"
#include "UTIL1.h"
#include "ACCEL1.h"
#include "FSSH1.h"
```

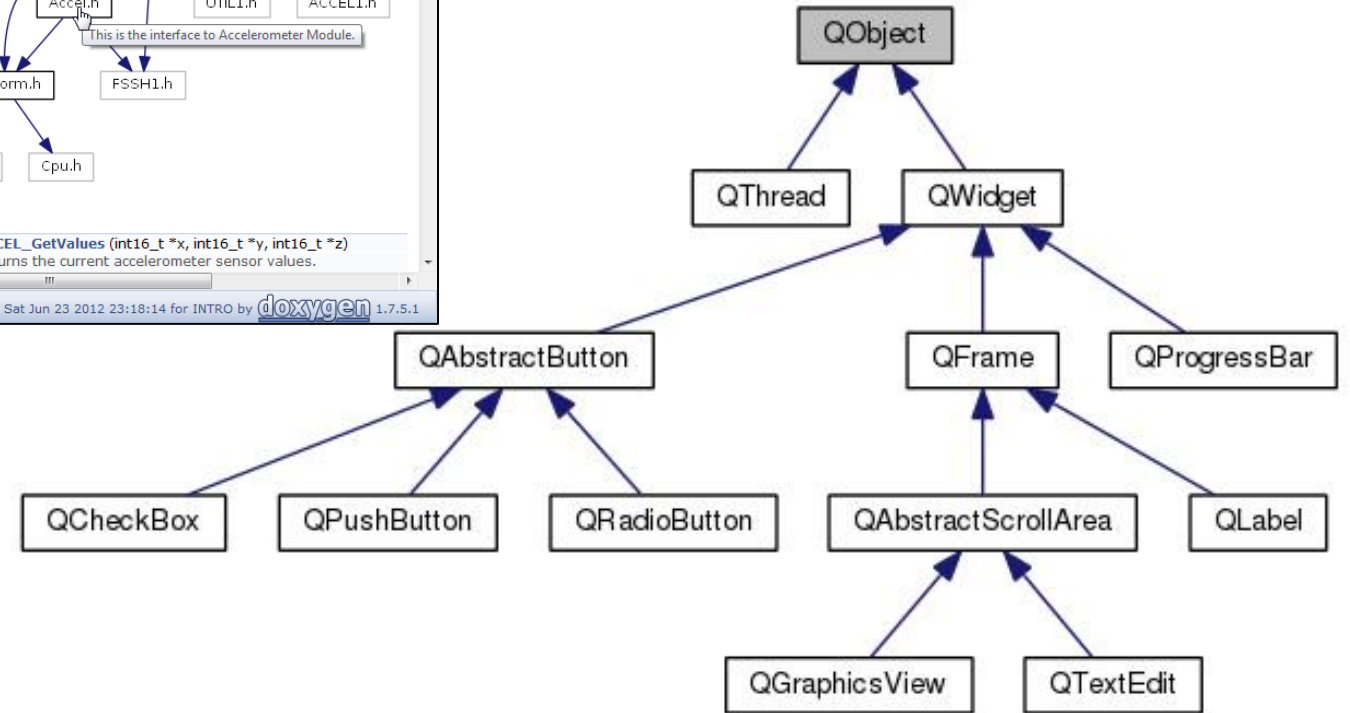
Include dependency graph for Accel.c:

```
graph TD; Accel.c --> Accel.h; Accel.c --> UTIL1.h; Accel.c --> ACCEL1.h; Accel.c --> Platform.h; Accel.c --> FSSH1.h; Platform.h --> PE_Types.h; Platform.h --> Cpu.h;
```

Functions

```
void ACCEL_GetValues (int16_t *x, int16_t *y, int16_t *z)
Returns the current accelerometer sensor values.
```

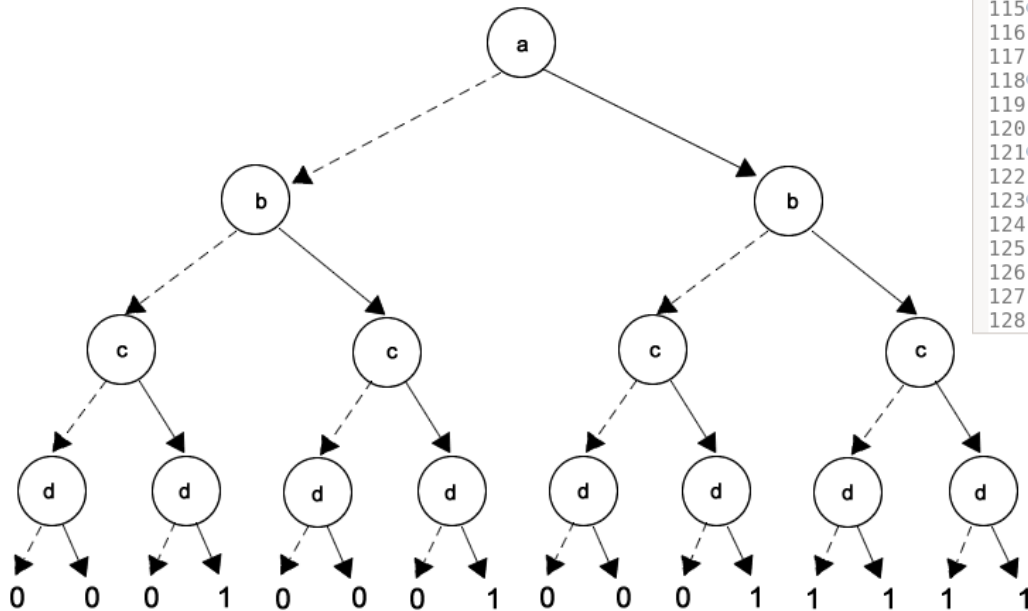
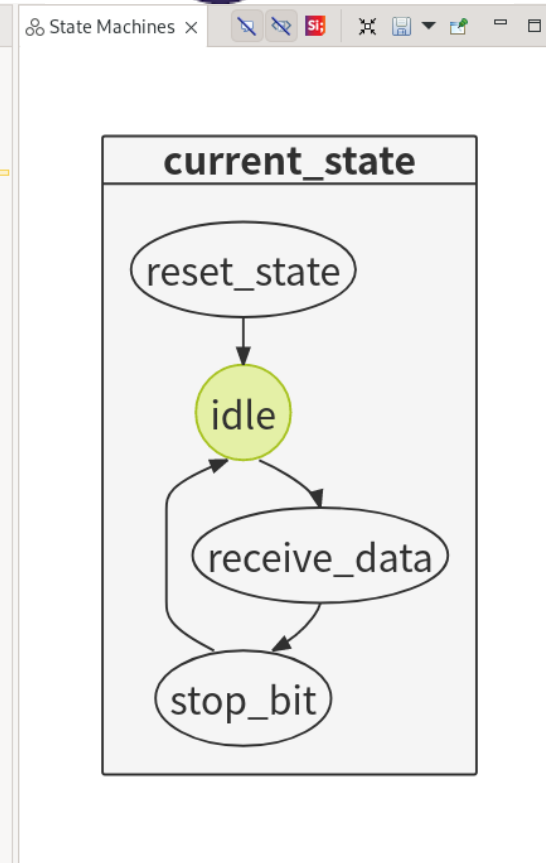
Generated on Sat Jun 23 2012 23:18:14 for INTRO by doxygen 1.7.5.1



Примеры из области проектирования

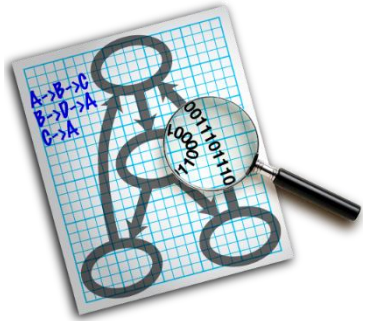


```
uart_rx.vhd x
95 rx_control : process (current_state, rx_filtered, data_counter)
96 begin
97   case current_state is
98     when reset_state =>
99       out_valid <= '0';
100
101       next_state <= idle;
102   when idle =>
103     out_valid <= 'X';
104
105     if (rx_filtered = '0') then
106       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;
```



Программа Graphviz

URL: <https://graphviz.org>



Download | Graphviz

graphviz.org/download/

Graphviz

Download Documentation Gallery Forum GitLab Search this site...

Search this site

Graphviz

About

Download

Source Code

Documentation

DOT Language

Command Line

Layout Engines

Output Formats

Attributes

Attribute Types

Graph

Attributes

Node Attributes

Node Shapes

Cluster

- Stable Windows install packages, built with Microsoft Visual Studio 16 2019:
 - graphviz-9.0.0
 - graphviz-9.0.0 (32-bit) ZIP archive [sha256] (contains all tools and libraries)
 - graphviz-9.0.0 (64-bit) EXE installer [sha256]
 - graphviz-9.0.0 (32-bit) EXE installer [sha256]
 - graphviz-8.1.0
 - graphviz-8.1.0 (32-bit) ZIP archive [sha256] (contains all tools and libraries)
 - graphviz-8.1.0 (64-bit) EXE installer [sha256]
 - graphviz-8.1.0 (32-bit) EXE installer [sha256]
 - graphviz-8.0.5
 - graphviz-8.0.5 (32-bit) ZIP archive [sha256] (contains all tools and libraries)
 - graphviz-8.0.5 (64-bit) EXE installer [sha256]
 - graphviz-8.0.5 (32-bit) EXE installer [sha256]
 - graphviz-8.0.3
 - graphviz-8.0.3 (32-bit) ZIP archive [sha256] (contains all tools and libraries)
 - graphviz-8.0.3 (64-bit) EXE installer [sha256]
 - graphviz-8.0.3 (32-bit) EXE installer [sha256]

View page source

Edit this page

Create child page

Create documentation issue

Print entire section

Source Code

Executable Packages

Linux

Windows

Mac

Solaris

Other Unix

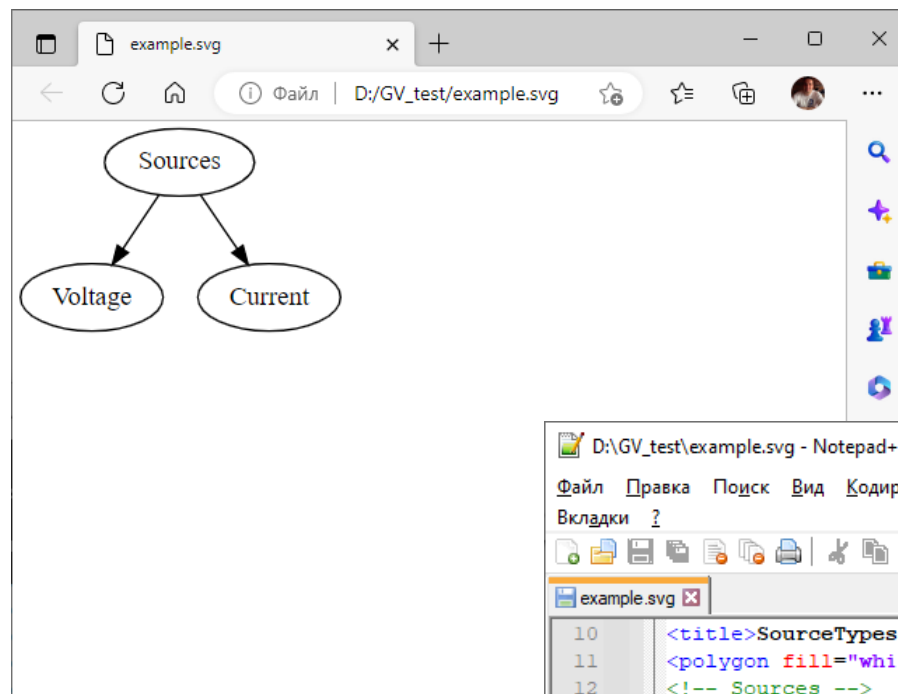
Выходные форматы GraphViz (1)

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>

```
D:\GV_test\example.svg - Notepad++
Файл  Правка  Поиск  Вид  Кодировки  Синтаксисы  Опции  Инструменты  Макросы  Запуск  Плагины
Вкладки  ?
example.svg x
10  <title>SourceTypes</title>
11  <polygon fill="white" stroke="transparent" points="-4,4 -4,-112 175.69,-1
12  <!-- Sources -->
13  <g id="node1" class="node">
14  <title>Sources</title>
15  <ellipse fill="none" stroke="black" cx="85.35" cy="-90" rx="40.09" ry="18
16  <text text-anchor="middle" x="85.35" y="-86.3" font-family="Times New Rom
17  </g>
18  <!-- Voltage -->
19  <g id="node2" class="node">
20  <title>Voltage</title>
21  <ellipse fill="none" stroke="black" cx="38.35" cy="-18" rx="38.19" ry="18
22  <text text-anchor="middle" x="38.35" y="-14.3" font-family="Times New Rom
23  </g>
24  <!-- Sources&#45;&gt;Voltage -->
25  <g id="edge1" class="edge">
26  <title>Sources&#45;&gt;Voltage</title>
```

Выходные форматы GraphViz (2)

Форматы кода:

- gtk
- json
- tk

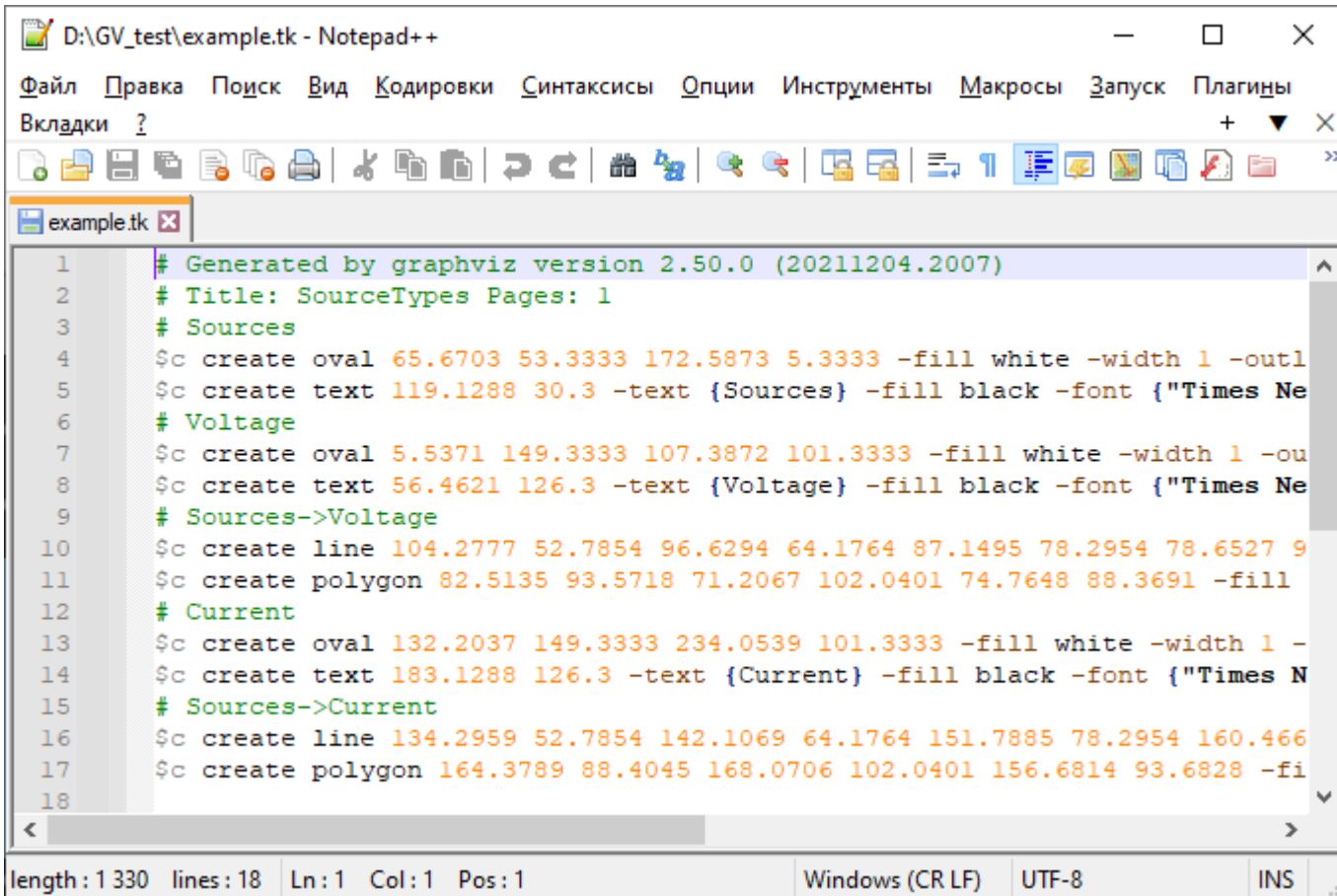
dot

-Ttk

<имя файла с графом>

-o

<имя файла с результатом>

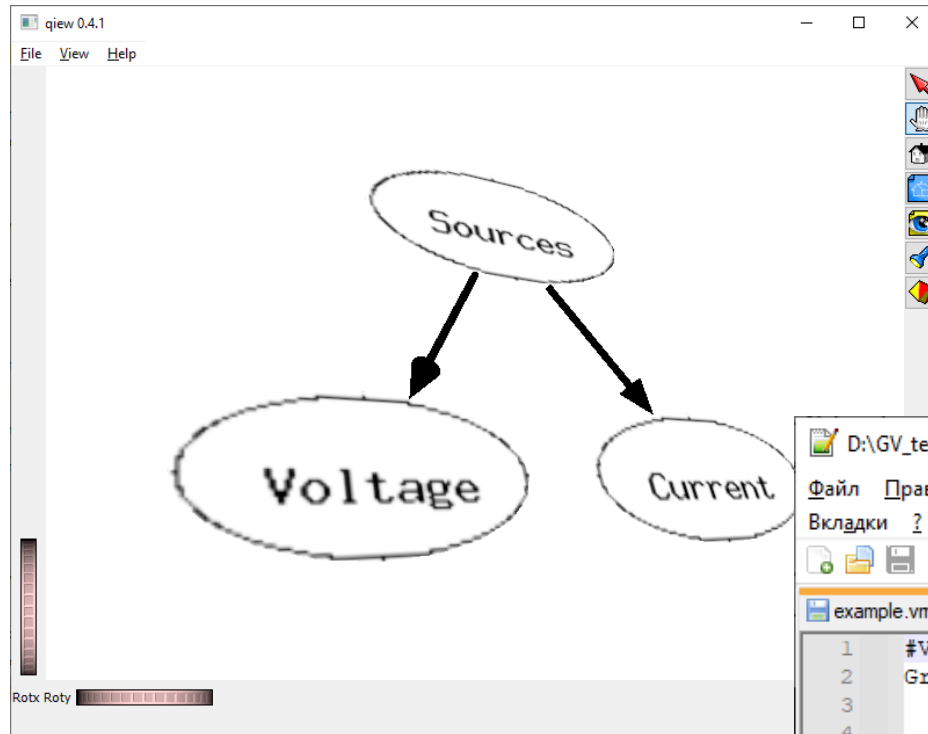


```
D:\GV_test\example.tk - Notepad++
Файл  Правка  Поиск  Вид  Кодировки  Синтаксисы  Опции  Инструменты  Макросы  Запуск  Плагины
Вкладки  ?
example.tk x
1  # Generated by graphviz version 2.50.0 (20211204.2007)
2  # Title: SourceTypes Pages: 1
3  # Sources
4  $c create oval 65.6703 53.3333 172.5873 5.3333 -fill white -width 1 -outl
5  $c create text 119.1288 30.3 -text {Sources} -fill black -font {"Times Ne
6  # Voltage
7  $c create oval 5.5371 149.3333 107.3872 101.3333 -fill white -width 1 -ou
8  $c create text 56.4621 126.3 -text {Voltage} -fill black -font {"Times Ne
9  # Sources->Voltage
10 $c create line 104.2777 52.7854 96.6294 64.1764 87.1495 78.2954 78.6527 9
11 $c create polygon 82.5135 93.5718 71.2067 102.0401 74.7648 88.3691 -fill
12 # Current
13 $c create oval 132.2037 149.3333 234.0539 101.3333 -fill white -width 1 -
14 $c create text 183.1288 126.3 -text {Current} -fill black -font {"Times N
15 # Sources->Current
16 $c create line 134.2959 52.7854 142.1069 64.1764 151.7885 78.2954 160.466
17 $c create polygon 164.3789 88.4045 168.0706 102.0401 156.6814 93.6828 -fi
18
length: 1330 lines: 18 Ln: 1 Col: 1 Pos: 1 Windows (CR LF) UTF-8 INS
```

Выходные форматы GraphViz (3)

Форматы кода:

- `gtk`
- `json`
- `tk`
- `vrml`



`dot`

`-Tvrml`

<имя файла с графом>

`-O`

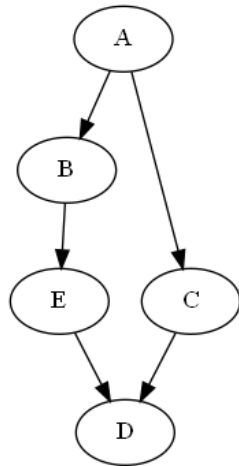
<имя файла с результатом>

```
1 #VRML V2.0 utf8
2 Group { children [
3   Transform {
4     scale 0.028 0.028 0.028
5     children [
6       Background { skyColor 1.000 1.000 1.000 }
7     # node Sources
8     Transform {
9       translation 85.347 90.000 0.000
10      scale 40.094 18.000 1
11      children [
12        Transform {
13          rotation 1 0 0 1.57
14          children [
15            Shape {
16              geometry Cylinder { side FALSE }
17              appearance Appearance {
18                material Material {
19                  ambientIntensity 0.33
```

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;  
}
```

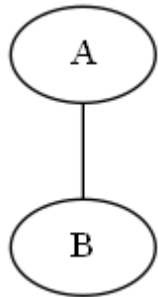


Типы графов: направленный и ненаправленный

graph

Создаёт неориентированный граф:

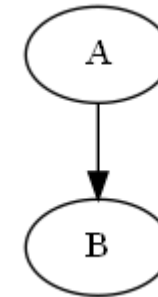
```
graph {  
    A -- B  
}
```



digraph

Создаёт ориентированный граф

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



Описание графов

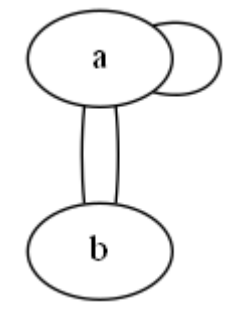
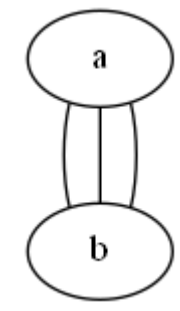
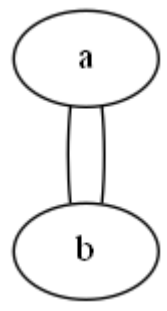
```
graph {  
  a -- b  
  a -- b  
}
```



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

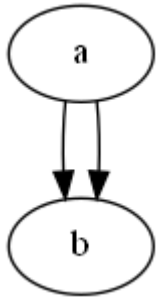
```
graph {  
  a -- b  
  a -- b  
  b -- a  
}
```

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

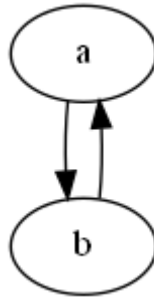


Описание орграфов

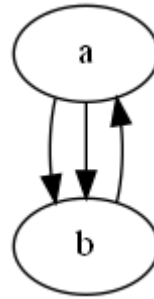
```
digraph {  
  a -> b  
  a -> b  
}
```



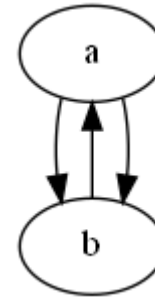
```
digraph {  
  a -> b  
  b -> a  
}
```



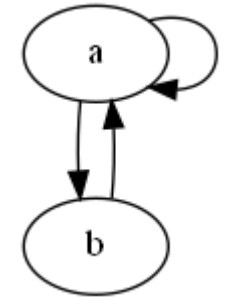
```
digraph {  
  a -> b  
  a -> b  
  b -> a  
}
```



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

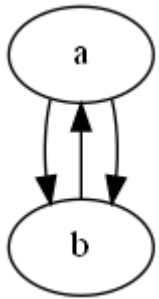


```
digraph {  
  a -> b  
  b -> a  
  a -> a  
}
```

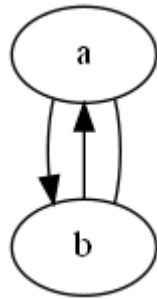


Атрибуты. Явное задание ориентации ребра

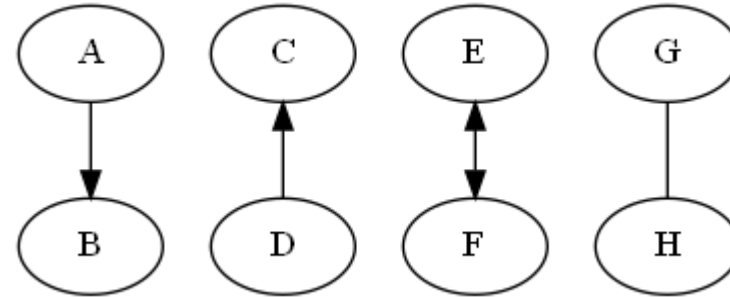
```
digraph {  
  a -> b  
  b -> a  
  a -> b  
}
```



```
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`

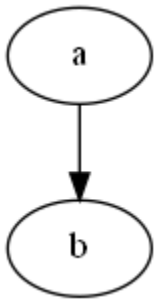
`none` - значение по умолчанию для неориентированных графов

Задание вида рёбер

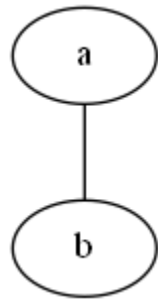
Форма указателя стрелки ребра

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

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



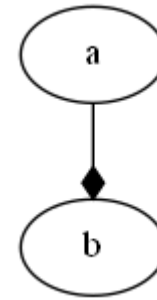
```
digraph {  
  a -> b [arrowhead=none]  
}
```



Форма указателя хвоста ребра

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

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



Описание для группы рёбер

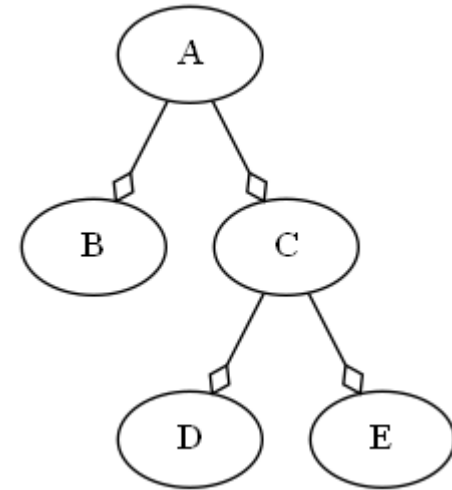
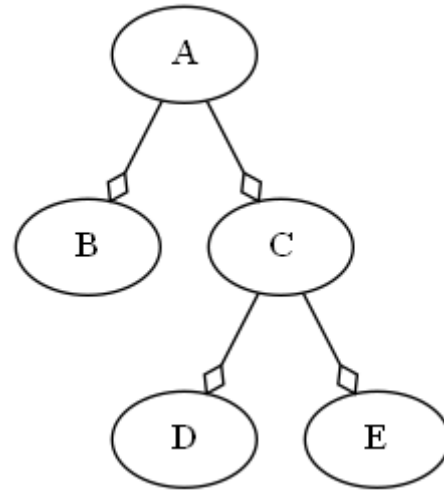
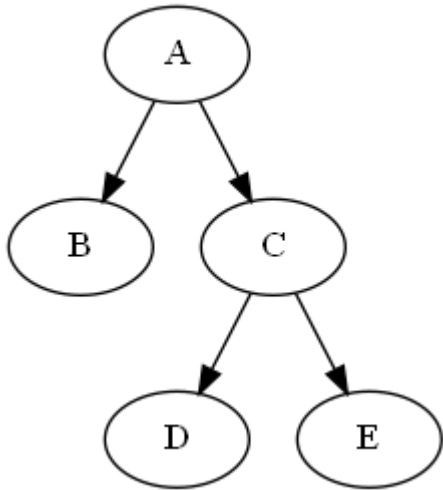
```
digraph {  
  A -> B  
  A -> C  
  C -> D  
  C -> E  
}
```



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

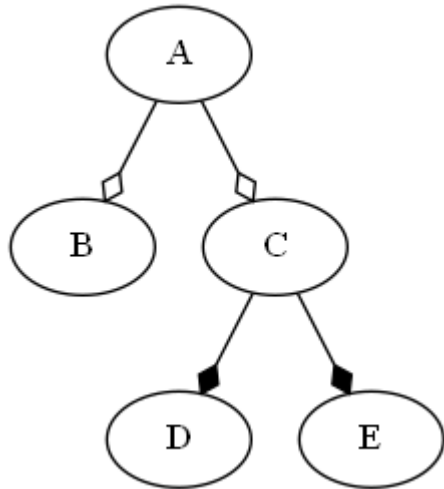


```
digraph {  
  edge [arrowhead=odiamond]  
  A -> B  
  A -> C  
  C -> D  
  C -> E  
}
```

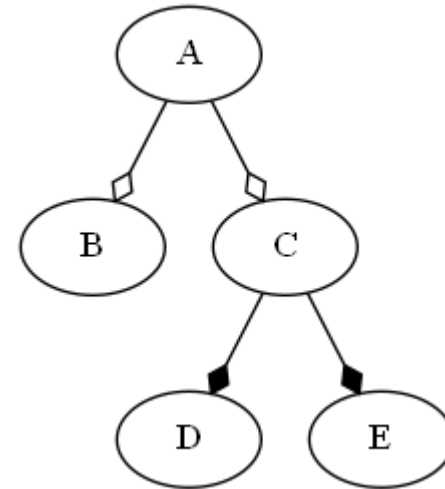


Описание рёбер (1)

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

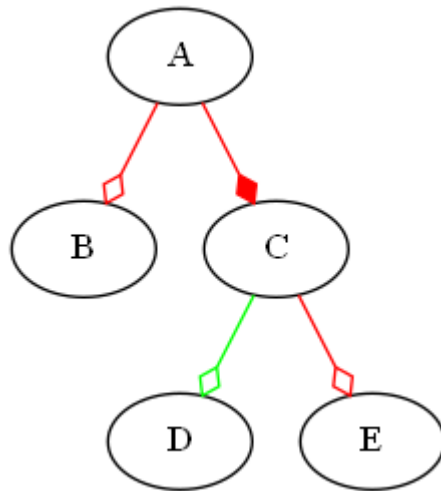


```
digraph {  
  edge [arrowhead=odiamond]  
  A -> B  
  A -> C  
  edge [arrowhead=diamond]  
  C -> D  
  C -> E  
}
```

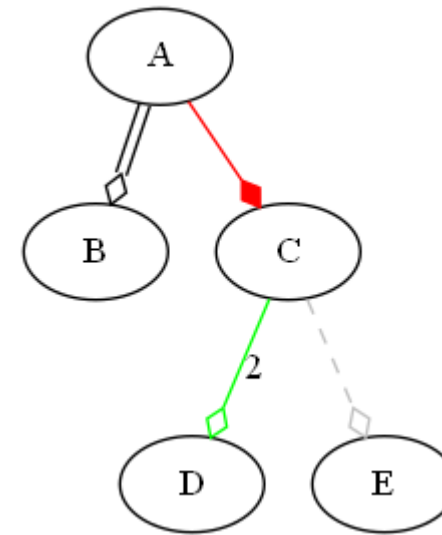


Описание рёбер (2)

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

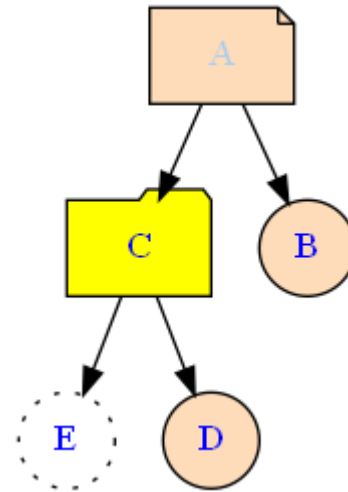


```
digraph {  
  edge [arrowhead=odiamond, color=red]  
  A -> B [color="black:invis:black"]  
  A -> C [arrowhead=diamond]  
  C -> D [color=green, label="2"]  
  C -> E [style=dashed, color=grey]  
}
```



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
```

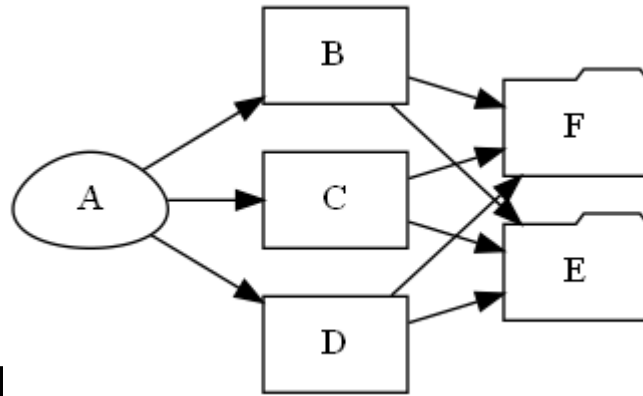
```
node [shape=folder]
```

```
F; E
```

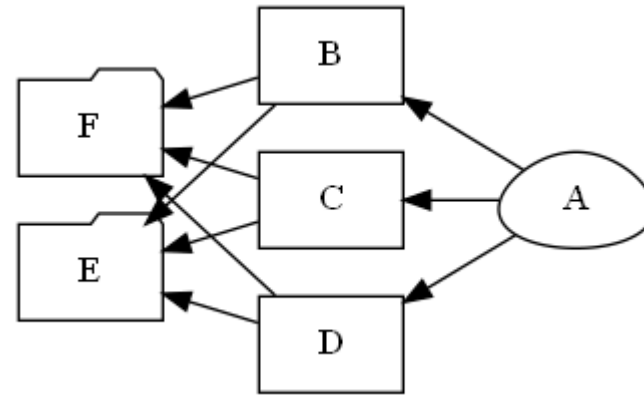
```
A -> {B, C, D} -> {F, E}
```

```
}
```

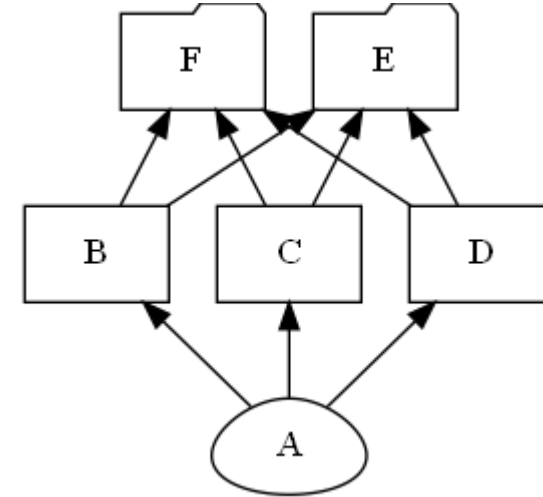
rankdir=LR;



rankdir=RL;

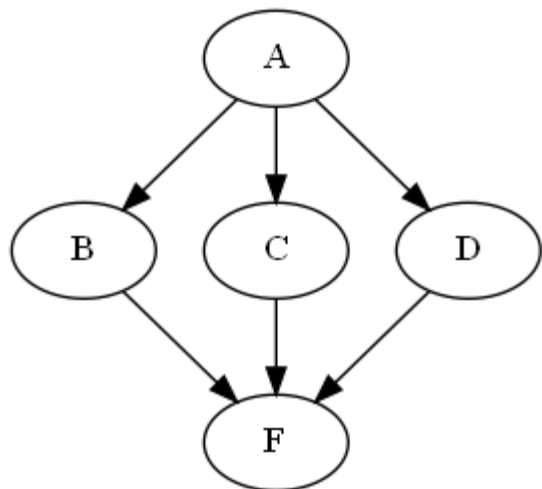


rankdir=BT;

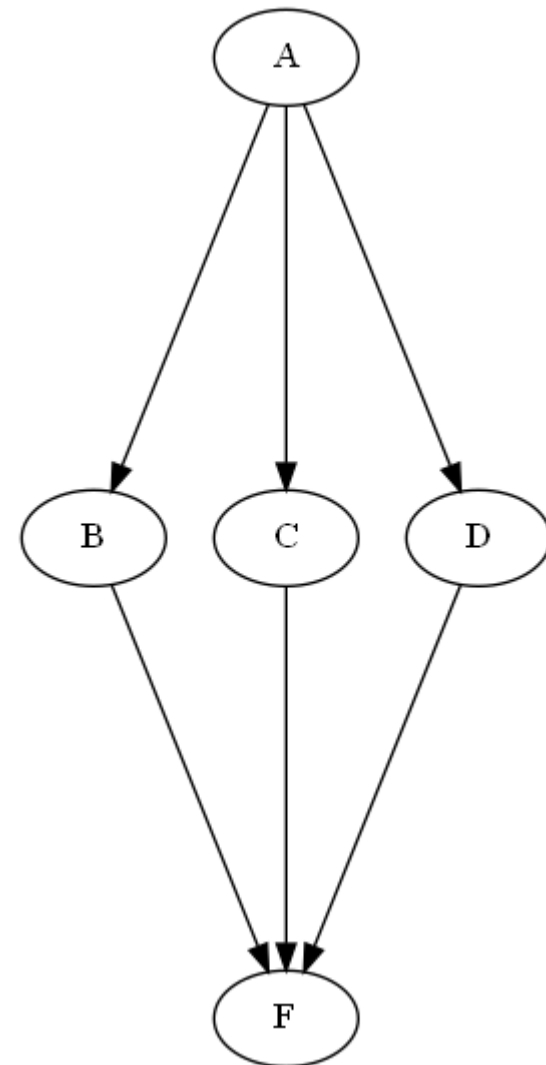


Задание длин рёбер графа

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



```
digraph {  
  graph [ranksep=2]  
  A -> {B, C, D} -> F  
}
```

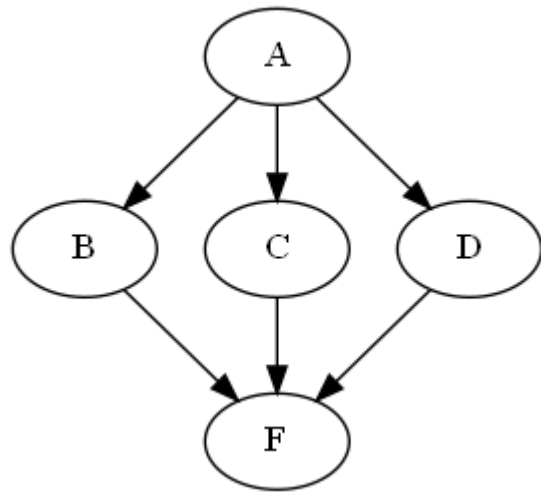


Группирование узлов

digraph {

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

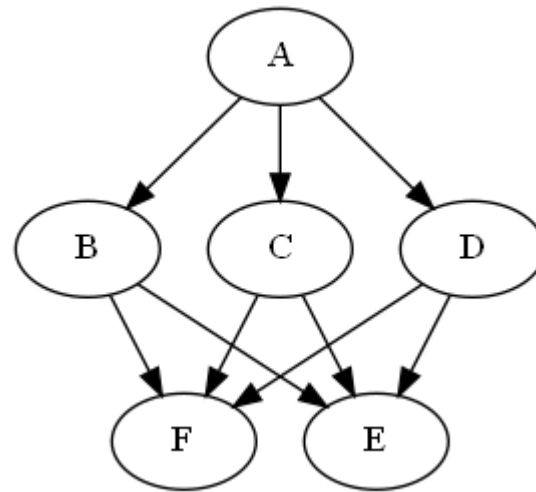
}



digraph {

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

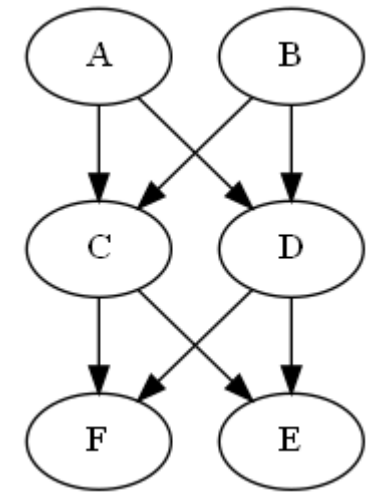
}



digraph {

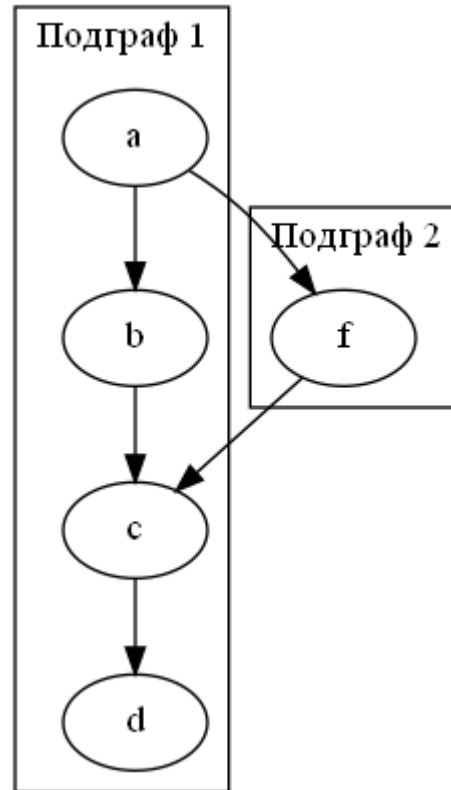
{A, B} -> {C, D} -> {F, E}

}



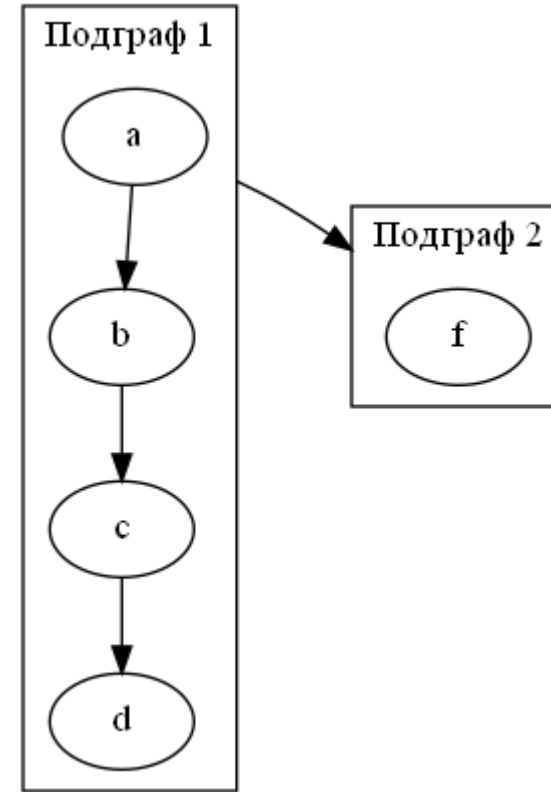
Группирование узлов подграфами (1)

```
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`

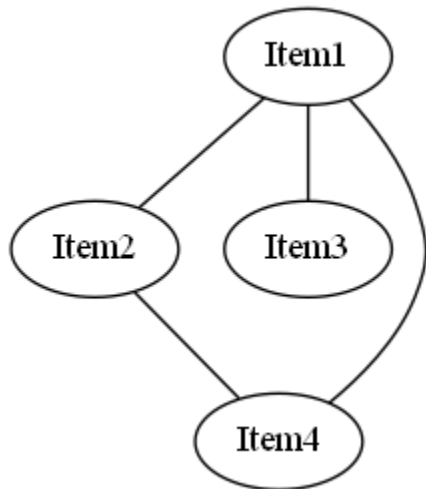
- длина ребра между подграфами

`ranksep=0.5`

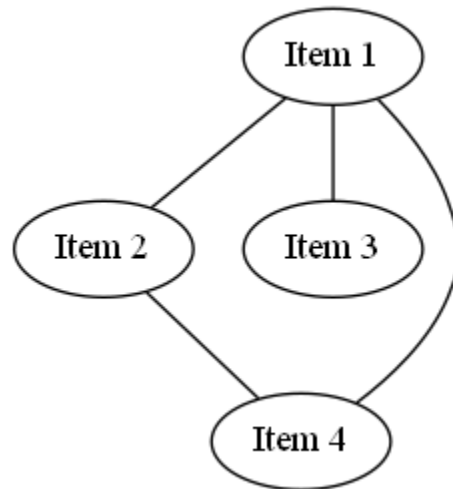
- длина ребра внутри подграфа

Именованние узлов

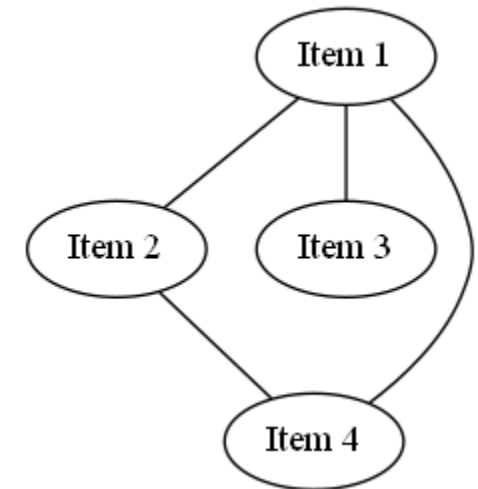
```
graph {  
  Item1  
  Item2  
  Item3  
  Item4  
  Item1 -- Item2  
  Item1 -- Item3  
  Item1 -- Item4  
  Item2 -- Item4  
}
```



```
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"  
}
```

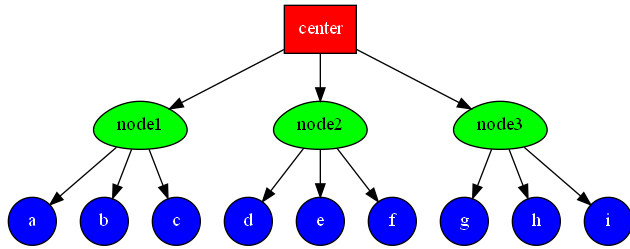


```
graph {  
  a [label="Item 1"]  
  b [label="Item 2"]  
  c [label="Item 3"]  
  d [label="Item 4"]  
  a -- b  
  a -- c  
  a -- d  
  b -- d  
}
```

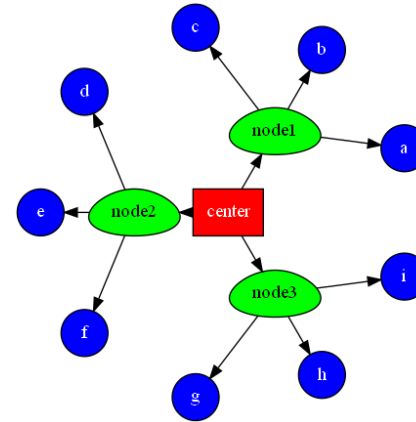


Алгоритмы визуализации графа (1)

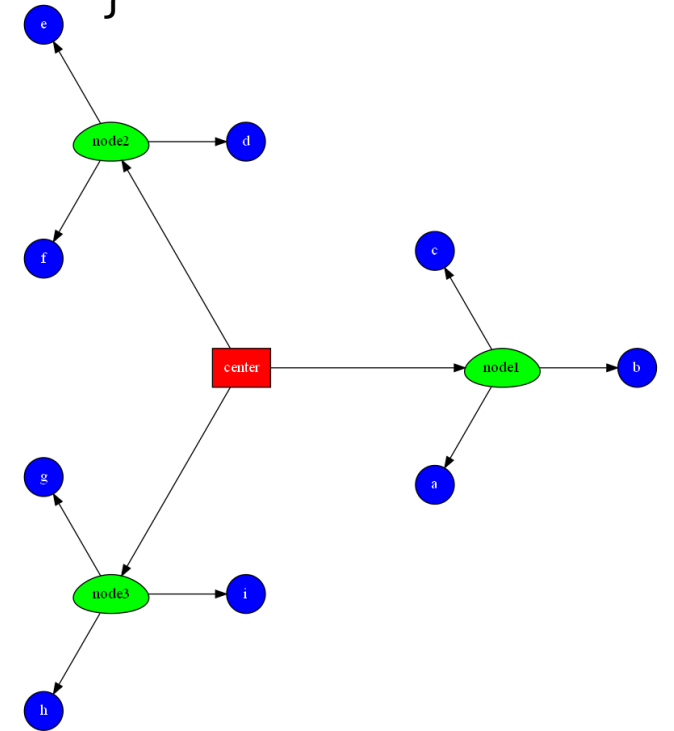
```
digraph {  
  graph [layout=dot]  
  ...  
}
```



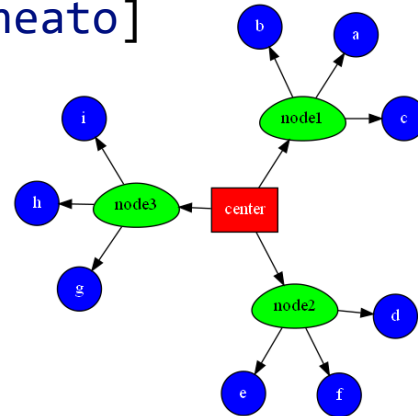
```
digraph {  
  graph [layout=twopi]  
  ...  
}
```



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

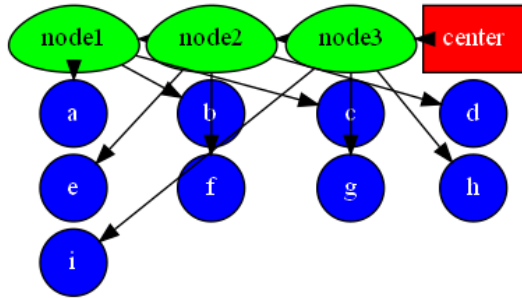


```
digraph {  
  graph [layout=neato]  
  ...  
}
```



Алгоритмы визуализации графа (2)

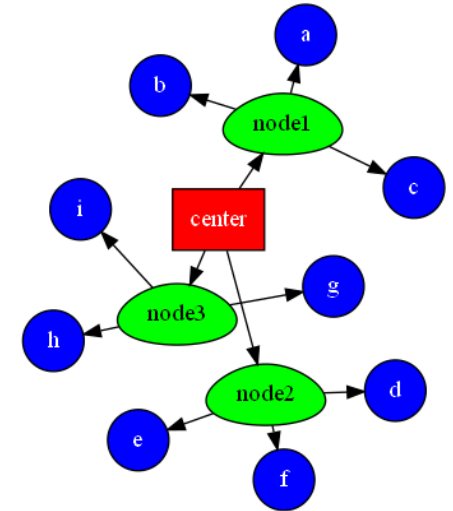
```
digraph {  
  graph [layout=osage]  
  ...  
}
```



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

| center | node1 | node2 | node3 |
|--------|-------|-------|-------|
| a | d | e | |
| b | f | h | |
| c | g | i | |

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



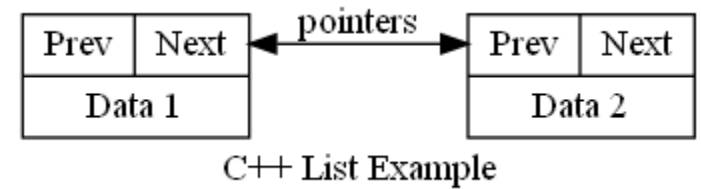
Более сложный пример

```
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

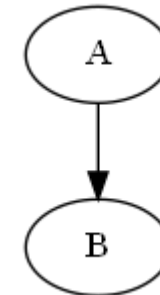
Код .py

```
#!/usr/bin/python  
  
import graphviz  
  
graph = graphviz.Digraph()  
  
graph.edge('A', 'B')  
  
graph.render('result.dot', format='png', view=True)
```



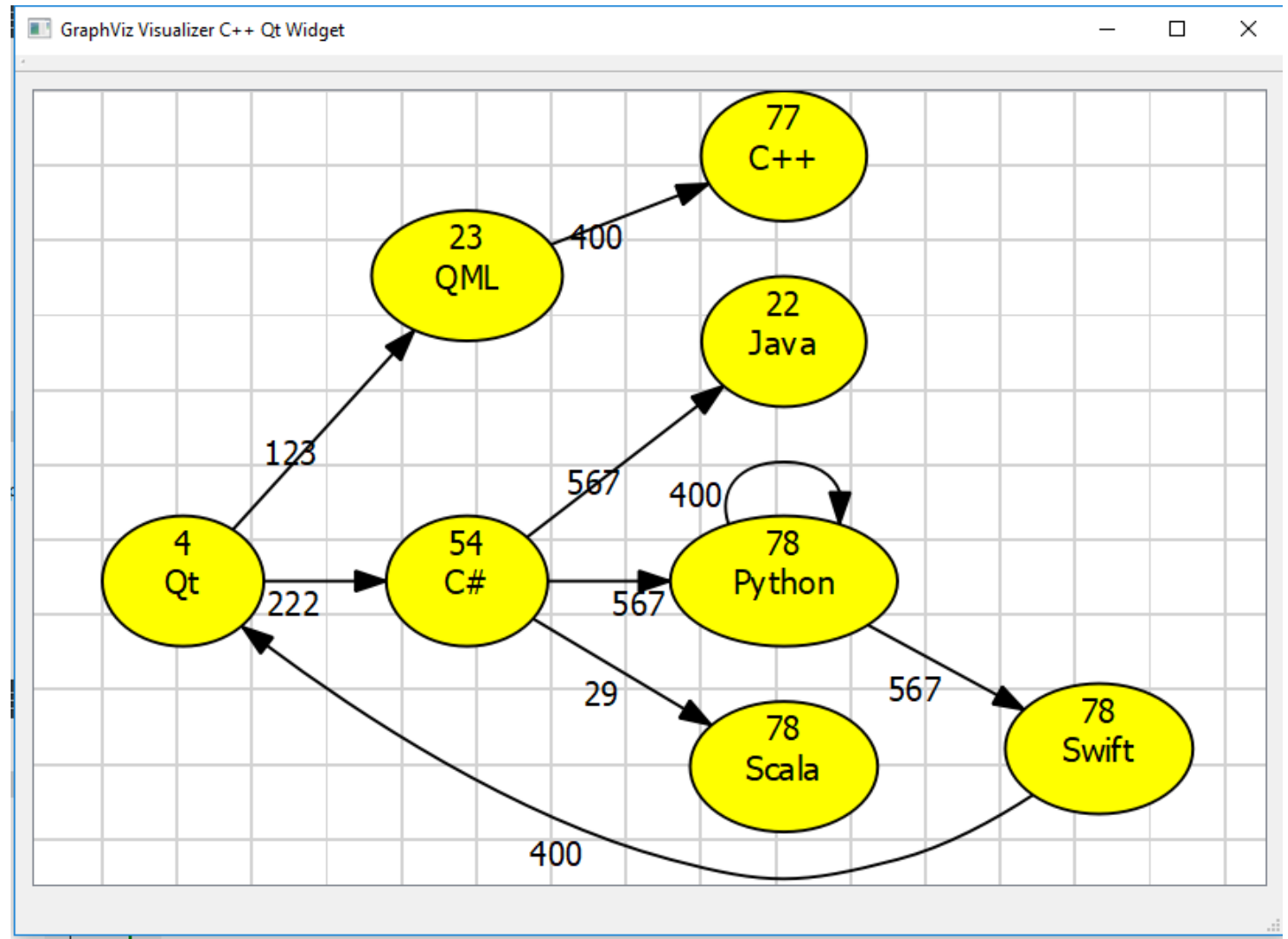
Код .dot

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



Вызов из других языков программирования: C++ и Qt

```
18 #include "graphvizwrapper.h"
19 #include <QApplication>
20
21 int main(int argc, char *argv[])
22 {
23     QApplication a(argc, argv);
24     GraphvizWrapper graph;
25
26     //Add Nodes Like This
27     graph.addNode("Qt" , 4);
28     graph.addNode("QML" , 23);
29     graph.addNode("C#" , 54);
30     graph.addNode("C++" , 77);
31     graph.addNode("Java" , 22);
32     graph.addNode("Python" , 78);
33     graph.addNode("Scala" , 78);
34     graph.addNode("Swift" , 78);
35
36     //Add Edges Like This
37     graph.addEdge("Qt","C#",222);
38     graph.addEdge("Qt","QML",123);
39     graph.addEdge("QML","C++",400);
40     graph.addEdge("C#", "Scala",29);
41     graph.addEdge("C#", "Python",567);
42     graph.addEdge("C#", "Java",567);
43     graph.addEdge("Python","Swift",567);
44     graph.addEdge("Python","Python",400);
45     graph.addEdge("Swift","Qt",400);
46
47     //Add Attributes Like This
48     graph.addGraphAttribute("rankdir", "LR");
49     graph.addNodeAttribute("style", "filled");
50     graph.addNodeAttribute("fillcolor", "yellow");
51     graph.addNodeAttribute("height", "0.6");
52     graph.addNodeAttribute("margin", "0.1");
53
54     graph.draw();
```



Создание графов онлайн

The screenshot shows the Graphviz Online web application. The left sidebar contains a code editor with the following code:

```
1 digraph G {  
2   a --> b  
3   a --> b  
4   a --> c  
5 }
```

The right pane displays the rendered graph. Node 'a' is at the top, with two arrows pointing down to node 'b'. A single arrow points from 'a' to node 'c' on the right. The interface includes a 'RESET' button and dropdown menus for 'Engine: dot' and 'Format: svg'.

The screenshot shows the Edotor web application. The left sidebar contains a code editor with the following code:

```
1 # Place the cursor inside "graph" to get some refactoring options  
2  
3 graph {  
4  
5   # To refactor nodes, place the cursor left to a node name  
6   a -- b  
7   a -- c;  
8   a -- d;  
9  
10  # Hover over color names to get a color picker  
11  b -- b [color=blue]  
12  b -- d [color=red];  
13  
14  # Get completion when assigning a shape or color  
15  b [shape=box, color=yellow];  
16  
17  a; # You can remove optional ; by placing the cursor left to a  
18  ; semicolon  
19 }
```

The right pane displays the rendered graph. Node 'a' is at the top, with arrows pointing to nodes 'b', 'c', and 'd'. Node 'b' is highlighted with a yellow box and has a blue self-loop and a red arrow pointing to 'd'. Node 'c' is in the middle, and node 'd' is at the bottom. The interface includes a 'Copy Share Link' button and dropdown menus for 'Engine: dot' and 'Graphviz Documentation'.