DOT语言 GUIDE

By cloudygoose

第一部分　**设置点和线的形状**和**颜**颜色

//先来看一个例子，我们创建一个文档　graph1.dot：

//digraph 是有向图，　graph是无向图，要注意，　->用在有向图中，　--用在无向图中表示一条边，不能混用。

```
1: digraph G {
2: main -> parse -> execute;
3: main -> init;
4: main -> cleanup;
5: execute -> make_string;
6: execute -> printf
7: init -> make_string;
8: main -> printf;
9: execute -> compare;
10: }
```

第一行给出了图的类型和名字

当一个点第一次出现，它就被创建了

用->标示符创建一条边

//然后在 cmd下用这个文件运行　dot

```
dot -Tps graph1.dot -o graph1.ps
```

//这是 ps格式，你也可以改成　jpg 等格式。

//-Tps 选择了 postscript output,

//就画出了这个图。

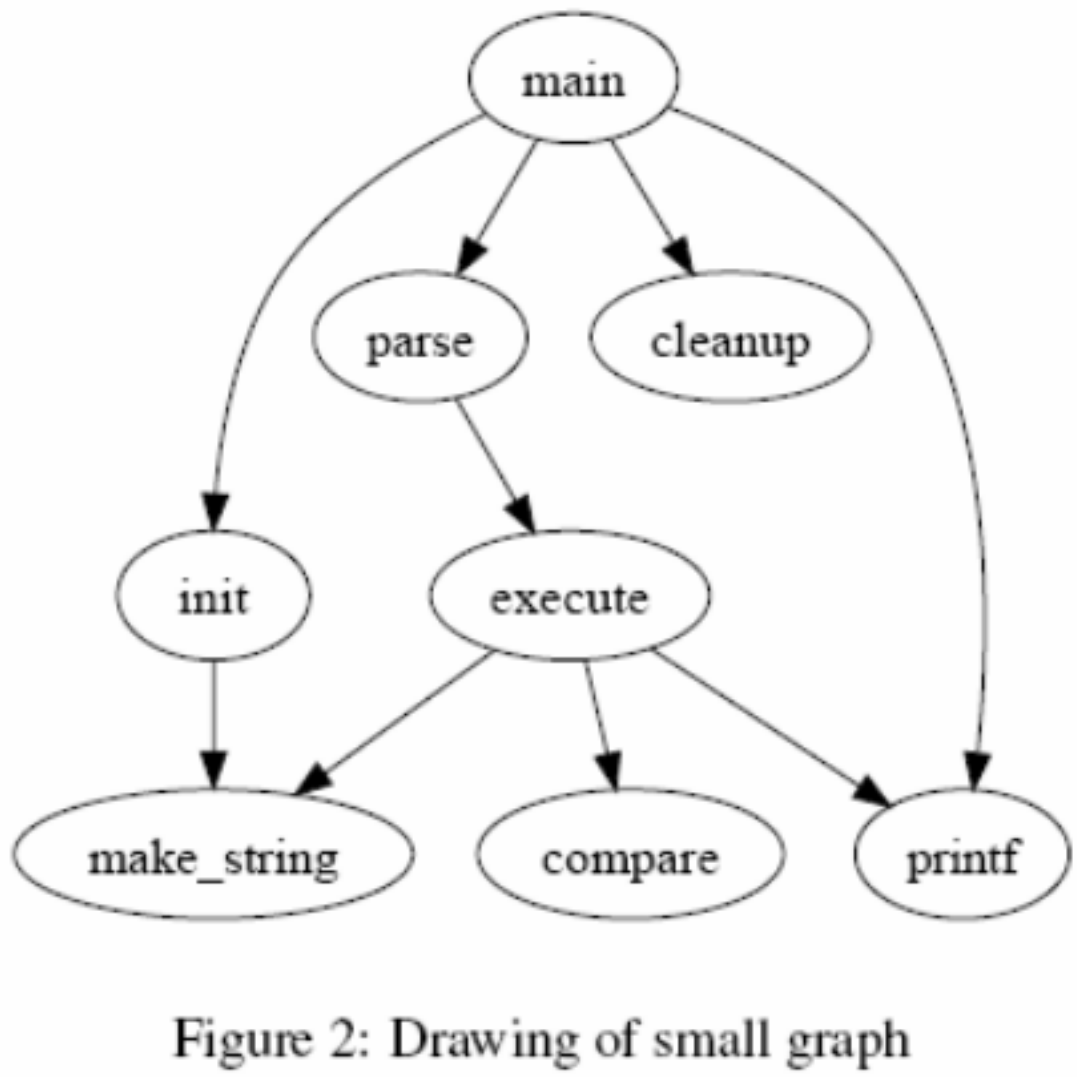


Figure 2: Drawing of small graph

//来看下一个稍微复杂点的例子，我们开始手动的设置一下图的属性。可以给点设置属性，也可以给边设置属性。　先来讲讲怎么设置边的属性，在每条边后面的双括号里设置边的属性。也可以在用　edge设置边的默认值。

//而给点设置属性就必须给每个点单独的设置一个属性，　node表示点的默认值。

// 点的默认参数是 shape=ellipse, width=.75, height=.5 and labeled by the node name.
// 一些点的形状在 appendix.h 中，一些常用的形状有 bos,circle,record,plaintext 。

```
1: digraph G {
2: size ="4,4";                          把图的尺寸设为 4 inch, 4 inch
3: main [shape=box]; /* this is a comment */   把main点的形状设为方形
4: main -> parse [weight=8];             weight是设置了这条边的重要
程度，默认是 1。
5: parse -> execute;
6: main -> init [style=dotted];          让这条线是点状的
7: main -> cleanup;
8: execute -> { make_string; printf}     这条语句一次连了两条线
9: init -> make_string;
10: edge [color=red]; // so is this      把边的默认颜色设为了 red
11: main -> printf [style=bold,label="100 times"];   label就是在边上写了一行字
12: make_string [label="make a\nstring"];   让make_string变成了一个两
行的字符串（注意那个 \n）。
13: node [shape=box,style=filled,color=".7 .3 1.0"];   设置了一下点的默认参数，蓝
色，这个被用在了 compare中。
14: execute -> compare;
15: }
```

连接2个node

画出以下图形：

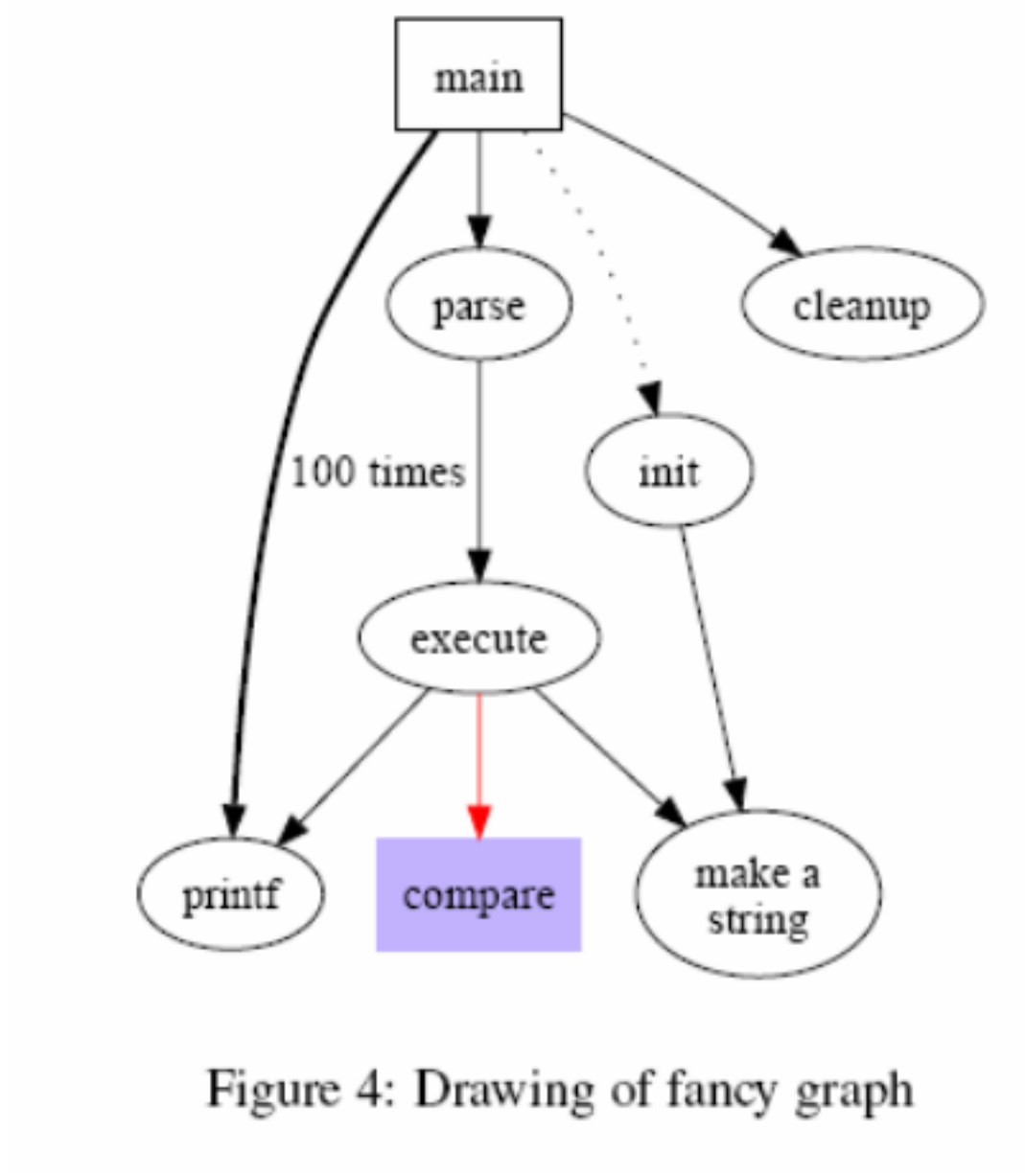


Figure 4: Drawing of fancy graph

// 可以设置每条边箭头的方向，用 dir，有 forward(default) ， back， both， none 四种。

```
digraph html {
```

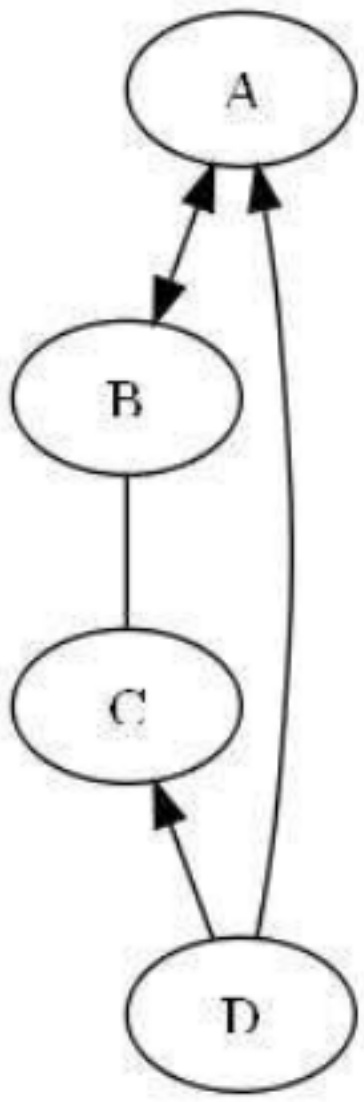
A -> B[dir = both];

B -> C[dir = none];

C -> D[dir = back];

D -> A[dir = forward];

}



//点的 shape 除了 record 和 Mrecord 这两种之外，其他的形状都是多边形，而我们可以对多边形进行一下属性上的设置， shape = polygon。Sides 用于设置它的边数， peripheries 用于设置多边形的外框的层数， regular = true 可以让你的多边形是一个规则的多边形， orientation = * ，可以让你的多边形旋转一个角度，如 orientation = 15 就是转了 15 度。 Skew 后面跟一个（-1.0~1.0）的小数，能让你的图形斜切一个角度， distortion 是让你的图形产生透视效果。

1: digraph G {

2: a -> b -> c;

3: b -> d;

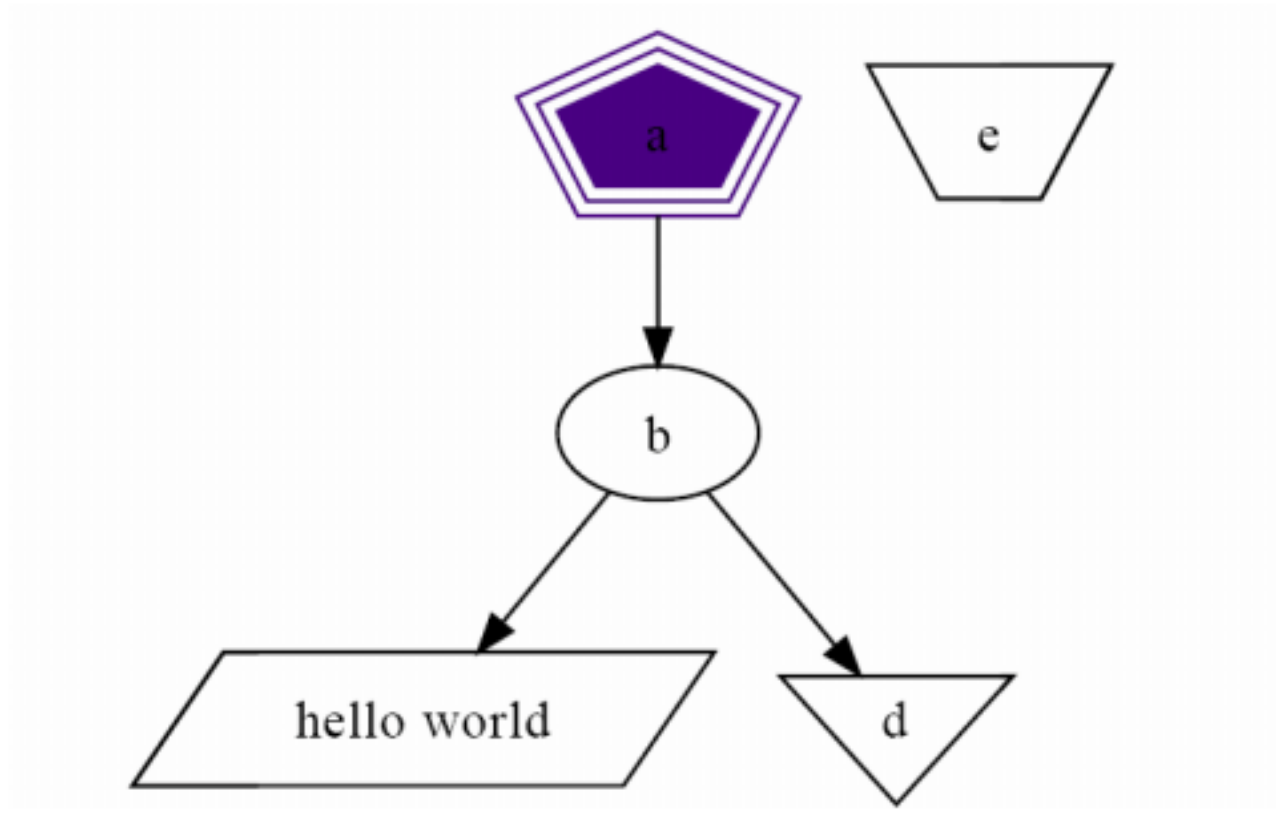
4: a [shape=polygon,sides=5,peripheries=3,color=lightblue,style=filled];

5: c [shape=polygon,sides=4,skew=.4,label="hello world"]

6: d [shape=invtriangle];

7: e [shape=polygon,sides=4,distortion=.7];

8: }



digraph A{

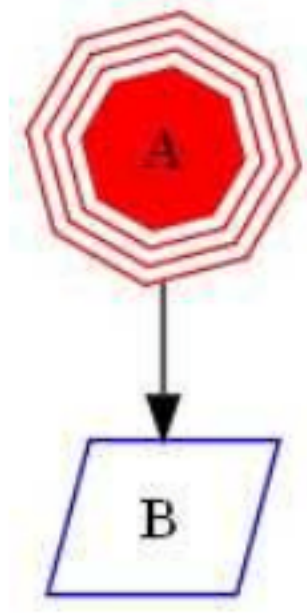
A -> B;

A[orientation = 15, regular = true, shape = polygon, sides = 8, peripheries = 4, color

= red style = filled];

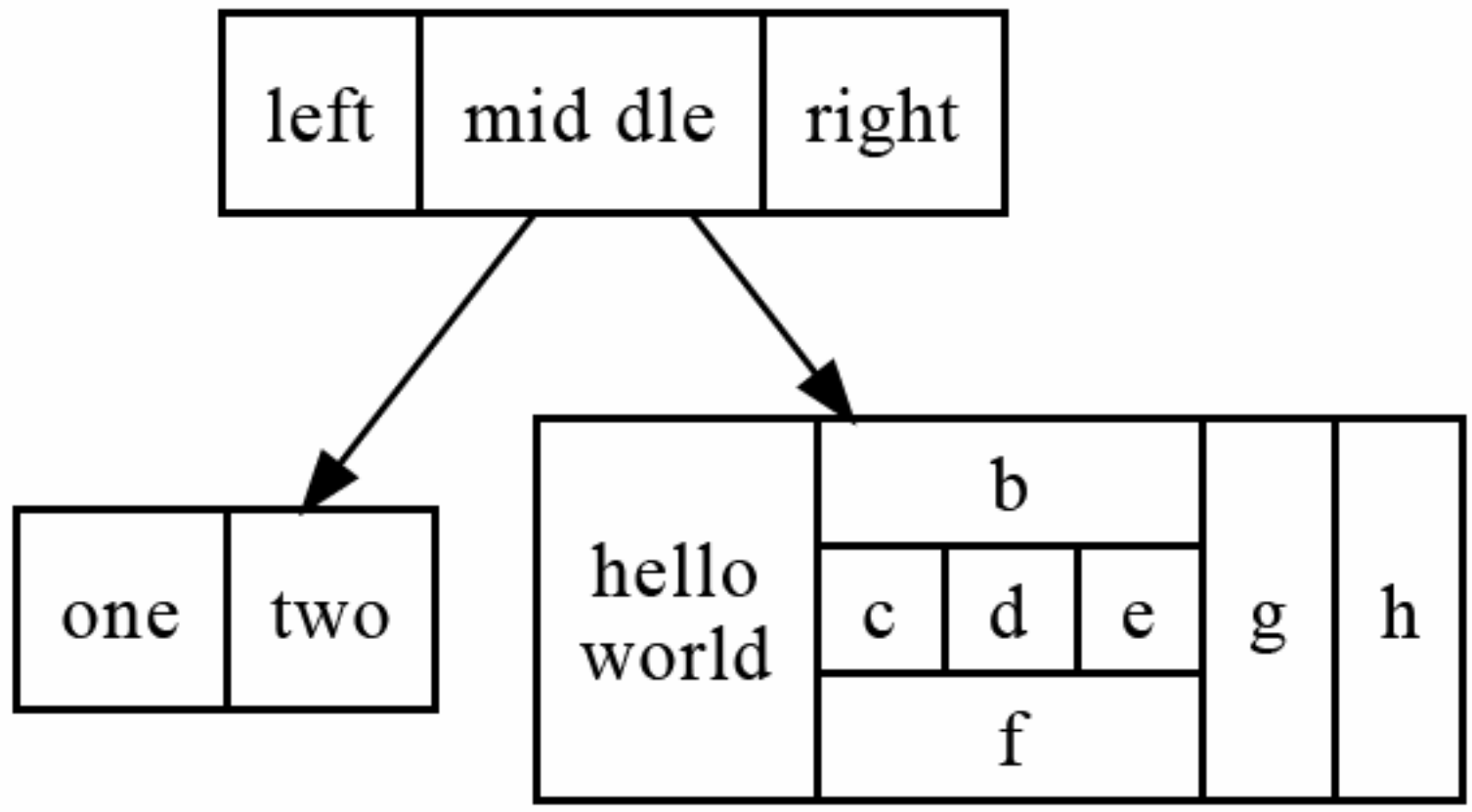
B[shape = polygon, sides = 4, skew = 0.5, color = blue];

}



//record 和 Mrecord 的区别就是 Mrecord 的角是圆的。 Record 就是由衡的和竖的矩形组成的图形。

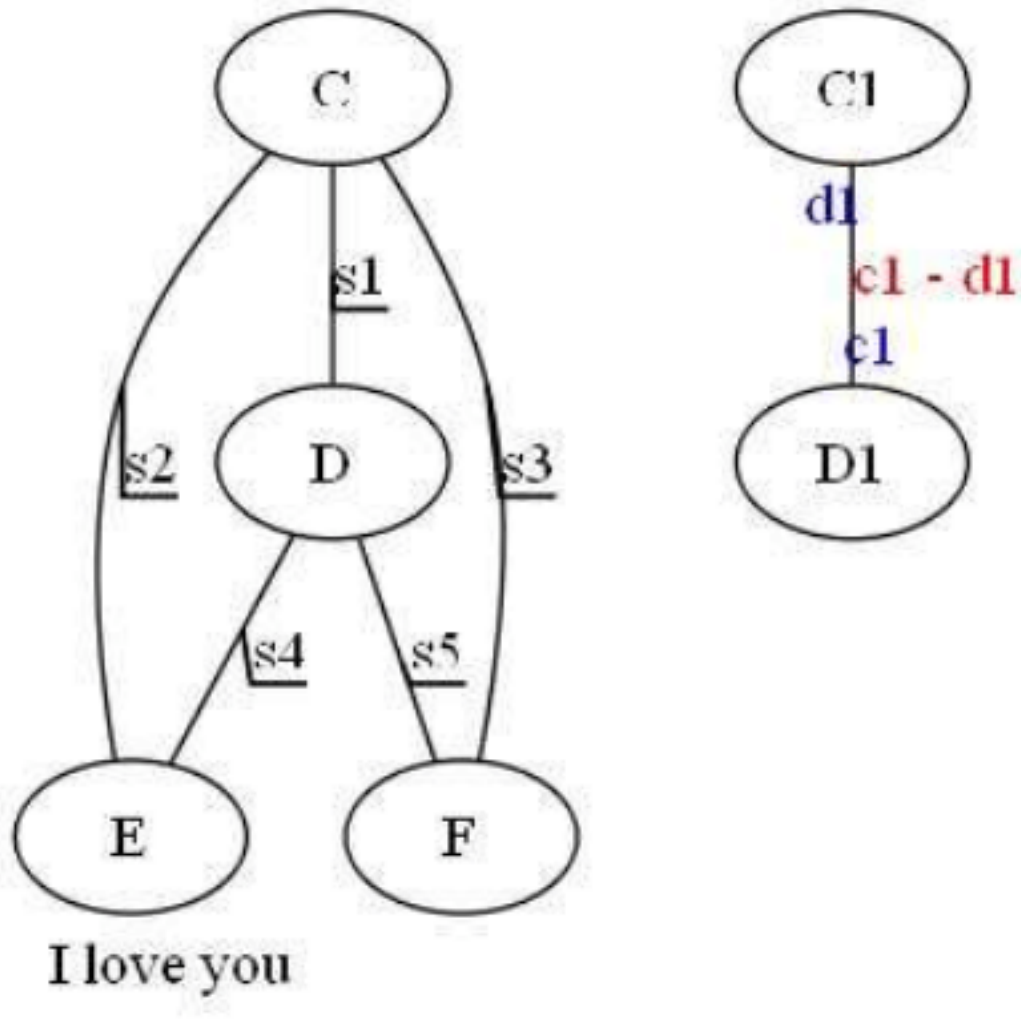
```
1: digraph structs {
2: node [shape=record];
3: struct1 [shape=record,label="<f0> left|<f1> mid\ dle|<f2> right"];
4: struct2 [shape=record,label="<f0> one|<f1> two"];
5: struct3 [shape=record,label="hello\nworld |{ b |{c|<here> d|e}| f}| g | h"];
6: struct1 -> struct2;
7: struct1 -> struct3;
8: }
```



//当你的线和线 label 比较多时，可以给线的属性 decorate = true，使得每条线的 label 与所属线之间连线。你还可以给每条线加上 headlabel 和 taillabel，给每条线的起始点和终点加上 label，他们的颜色由 labelfontcolor 来决定，而 label 的颜色由 fontcolor 来决定。

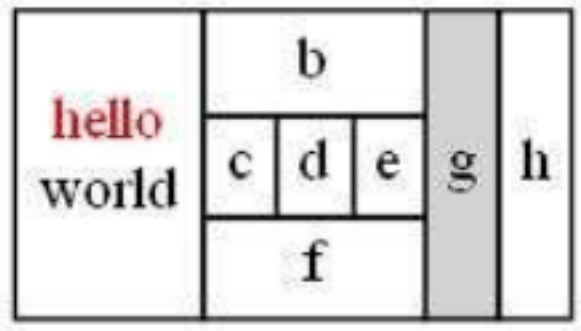
```
graph A{
label = "I love you";                          //给这幅图设置，名字
labelloc = b;                                  //图名字的位置在 bottom，也可以是 t
labeljust = l;                                 //图名字的位置在 left，也可以是 r

edge[decorate = true];
C -- D[label = "s1"];
C -- E[label = "s2"];
C -- F[label = "s3"];
D -- E[label = "s4"];
D -- F[label = "s5"];
edge[decorate = false, labelfontcolor = blue, fontcolor = red];
C1 -- D1[headlabel = "c1", taillabel = "d1", label = "c1 - d1"];
}
```



//在 dot 中我们可以用　html 语言写一个　table。在 label 后用 < >而不是 ” 就能引入 html 语言。

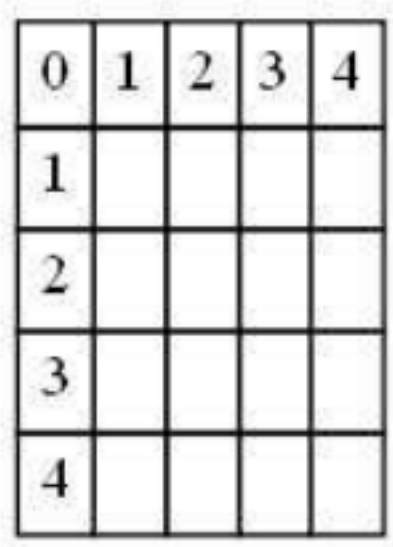
```
1: digraph html {
2: abc [shape=none, margin=0, label=<
3: <TABLE BORDER="0" CELLBORDER="1" CELLSPACING="0" CELLPADDING="4">
4: <TR><TD ROWSPAN="3"><FONT COLOR="red">hello</FONT><BR/>world</TD>
5: <TD COLSPAN="3">b</TD>
6: <TD ROWSPAN="3" BGCOLOR="lightgrey">g</TD>
7: <TD ROWSPAN="3">h</TD>
8: </TR>
9: <TR><TD>c</TD>
10: <TD PORT="here">d</TD>
11: <TD>e</TD>
12: </TR>
13: <TR><TD COLSPAN="3">f</TD>
14: </TR>
15: </TABLE>>];
16: }
```



//这样创造了一个　5 行 5 列的表格，我们可以在表格中打字。

```
digraph html {
abc [shape=none, margin=0, label=<
<TABLE BORDER="0" CELLBORDER="1" CELLSPACING="0" CELLPADDING="4">
<TR><TD>0</TD><TD>1</TD><TD>2</TD><TD>3</TD><TD>4</TD>
```

```
</TR>
<TR><TD>1</TD><TD></TD><TD></TD><TD></TD><TD></TD>
</TR>
<TR><TD>2</TD><TD></TD><TD></TD><TD></TD><TD></TD>
</TR>
<TR><TD>3</TD><TD></TD><TD></TD><TD></TD><TD></TD>
</TR>
<TR><TD>4</TD><TD></TD><TD></TD><TD></TD><TD></TD>
</TR>
</TABLE>>];
}
```



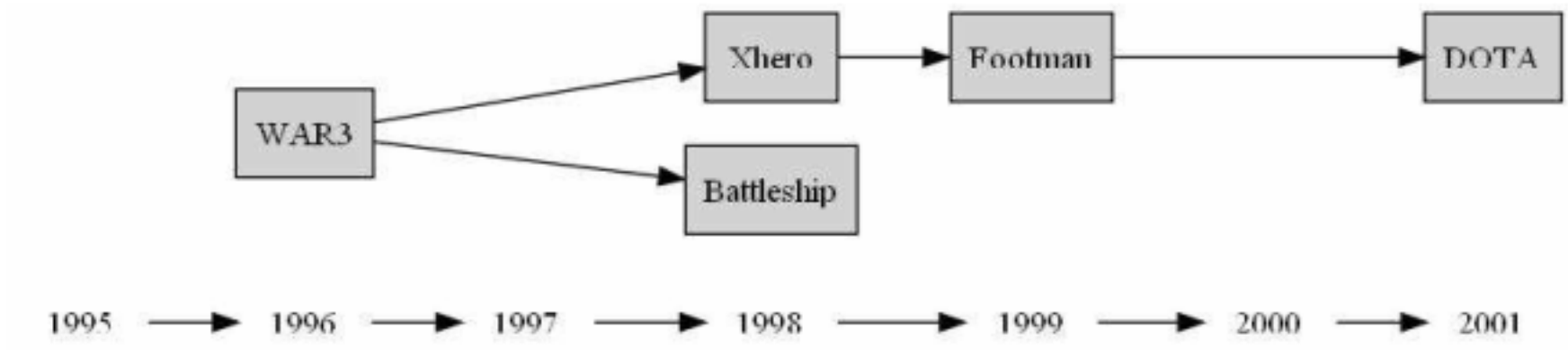
0	1	2	3	4
1				
2				
3				
4				

第二部份　设置点和线的位置,, 子图的概念

// 默认时图中的线都是从上到下的，我们可以将其改为从左到右，在文件的最上层打入 rankdir=LR 就是从左到右，默认是 TB(top -> bottom) ，也可以是 RL， BT。

// 当图中时间表之类的东西时，我们会需要点能排在一行（列） ，这时要用到 rank，用花括号把 rank=same，然后把需要并排的点一次输入。

```
digraph html {
rankdir = LR;
{
    node[shape = plaintext];
    1995 -> 1996 -> 1997 -> 1998 -> 1999 -> 2000 -> 2001;
}
{
    node[shape = box, style = filled];
    WAR3 -> Xhero -> Footman -> DOTA;
    WAR3 -> Battleship;
}
{rank = same; 1996; WAR3;}
{rank = same; 1998; Xhero; Battleship;}
{rank = same; 1999; Footman;}
{rank = same; 2001; DOTA;}
}
```

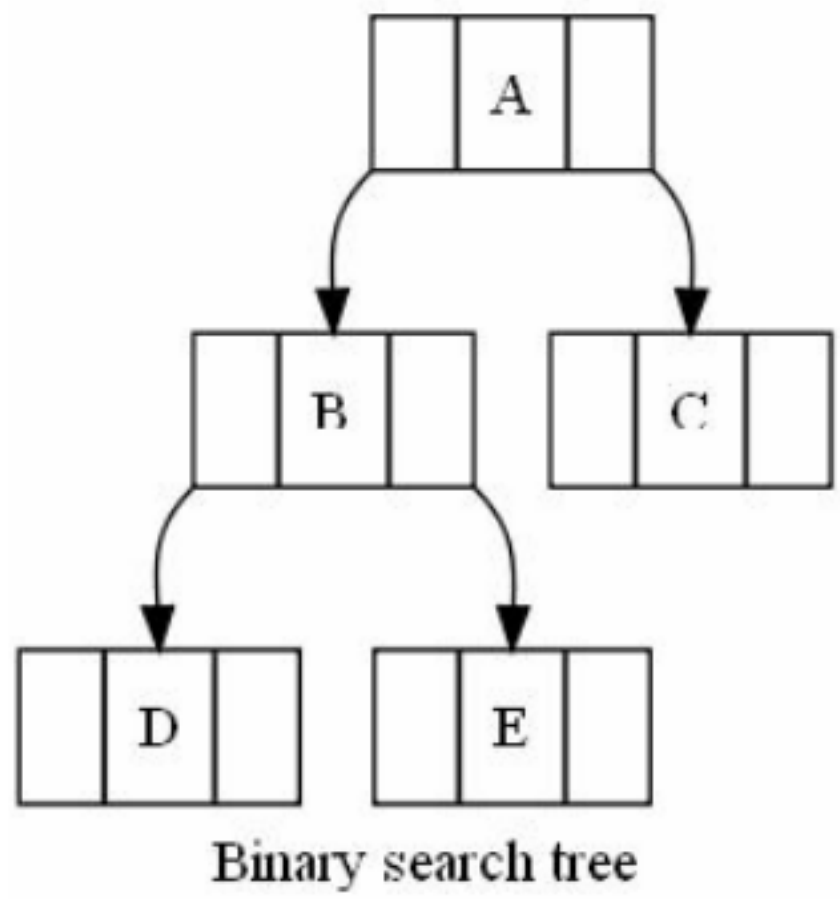


设立一条边时， 我们可以制定这条边从起点的那个位置射出和从哪个位置结束。 控制符有 "n", "ne","e", "se", "s", "sw", "w" 和 "nw"，具体效果见下：

```
digraph html {
node[shape = box];
c:n -> d[label = n];
c1:ne -> d1[label = ne];
c2:e -> d2[label = e];
b:se -> a[label = se];
c3:s -> d3[label = s];
c4:sw -> d4[label = sw];
c5:w -> d5[label = w];
c6:nw -> d6[label = nw];
}
```

我们也可以在 record 中给点定义一些 port，因为 record 类型中都是一个个格子。

```
digraph html {
label = "Binary search tree";
node[shape = record];
A[label = "<f0> | <f1> A |<f2> "];
B[label = "<f0> | <f1> B |<f2> "];
C[label = "<f0> | <f1> C |<f2> "];
D[label = "<f0> | <f1> D |<f2> "];
E[label = "<f0> | <f1> E |<f2> "];
A:f0:sw -> B:f1;
A:f2:se -> C:f1;
B:f0:sw -> D:f1;
B:f2:se -> E:f1;
}
```



Binary search tree

//构造一个 HASH 表

```
1: digraph G {
2: nodesep=.05;
3: rankdir=LR;
4: node [shape=record,width=.1,height=.1];
5:
6: node0 [label = "<f0> |<f1> |<f2> |<f3> |<f4> |<f5> |<f6> | ",height=2.5];
7: node [width = 1.5];
8: node1 [label = "{<n> n14 | 719 |<p> }"];
9: node2 [label = "{<n> a1 | 805 |<p> }"];
10: node3 [label = "{<n> i9 | 718 |<p> }"];
11: node4 [label = "{<n> e5 | 989 |<p> }"];
12: node5 [label = "{<n> t20 | 959 |<p> }"] ;
13: node6 [label = "{<n> o15 | 794 |<p> }"] ;
14: node7 [label = "{<n> s19 | 659 |<p> }"] ;
15:
16: node0:f0 -> node1:n;
17: node0:f1 -> node2:n;
18: node0:f2 -> node3:n;
19: node0:f5 -> node4:n;
20: node0:f6 -> node5:n;
21: node2:p -> node6:n;
22: node4:p -> node7:n;
23: }
```

Figure 17: Hash table graph file

n14 719

a1 805

i9 718

e5 989

t20 959

o15 794
s19 659
Figure 18: Drawing of hash table

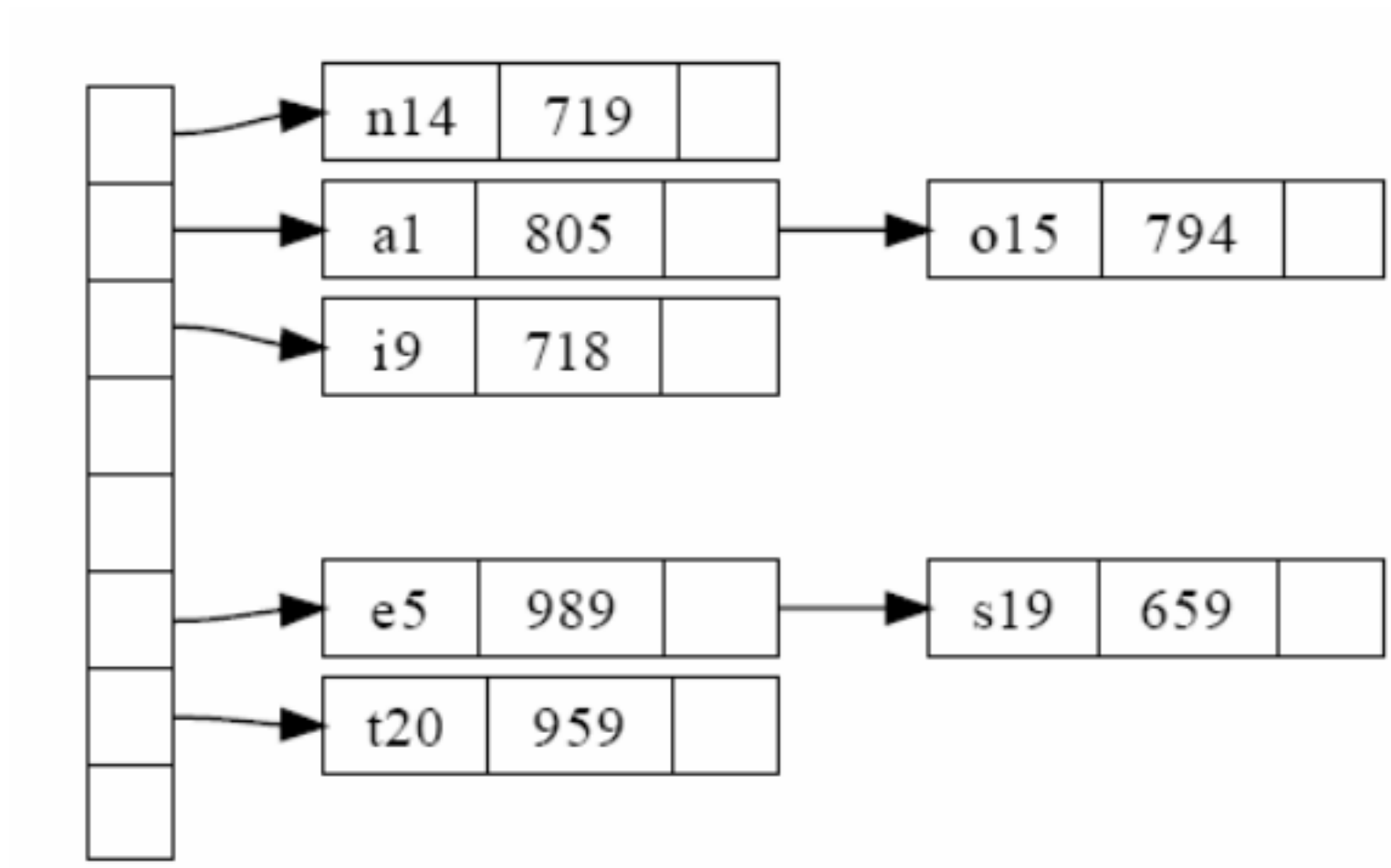


Figure 18: Drawing of hash table

画一个子图就是　subgraph cluster#，必须有　cluster 前缀。

```
digraph G {
    subgraph cluster0 {
        node [style=filled,color=white];
        style=filled;
        color=lightgrey;
        a0 -> a1 -> a2 -> a3;
        label = "process #1";
    }

    subgraph cluster1 {
        node [style=filled];
        b0 -> b1 -> b2 -> b3;
        label = "process #2";
        color=blue
    }
    start -> a0;
    start -> b0;
    a1 -> b3;
    b2 -> a3;
    a3 -> a0;
    a3 -> end;
    b3 -> end;

    start [shape=Mdiamond];
    end [shape=Msquare];
}
```

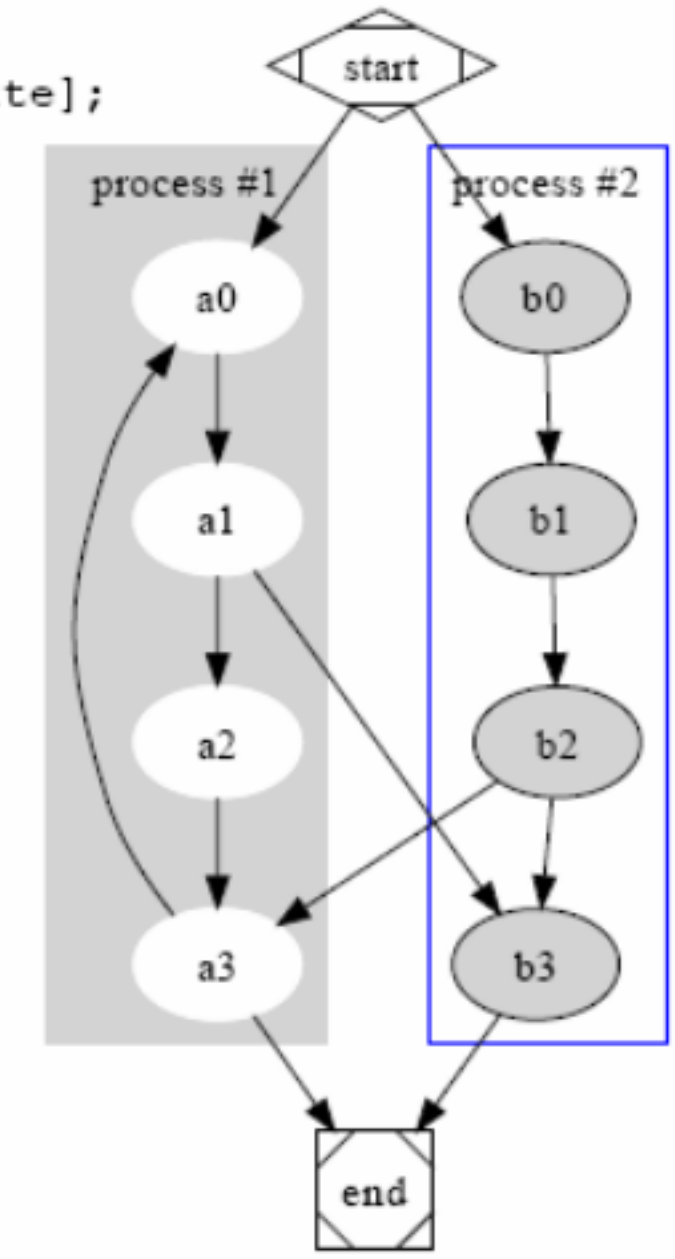
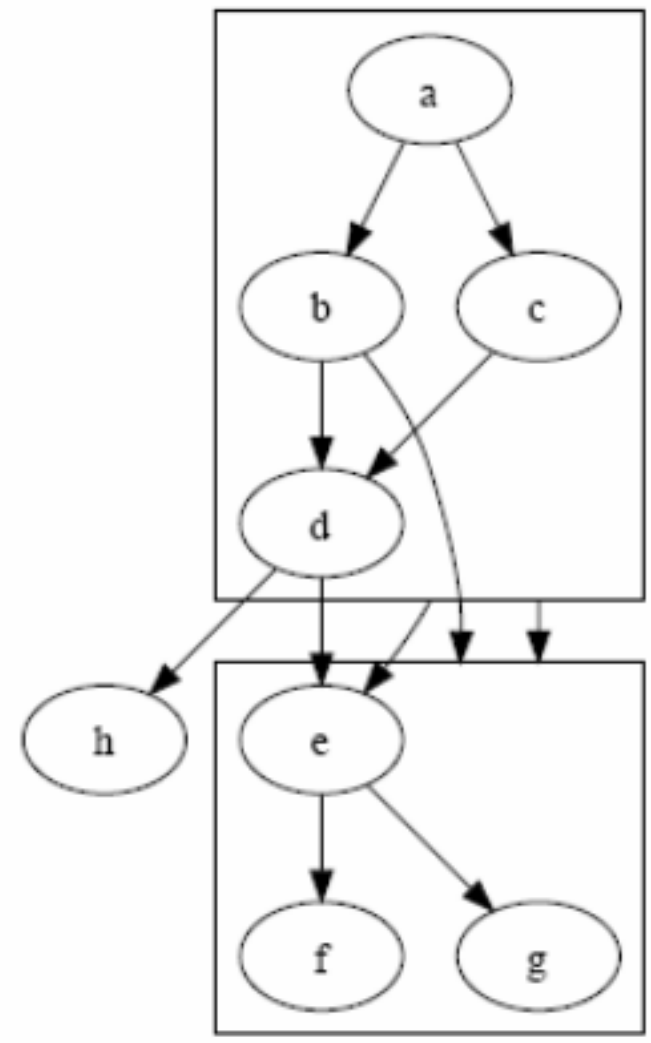


Figure 19: Process diagram with clusters

当你想把一条边连到一个子图的边界上，先输入 compound = true，然后就能用 lhead 和 ltail 来设置连接的子图了。

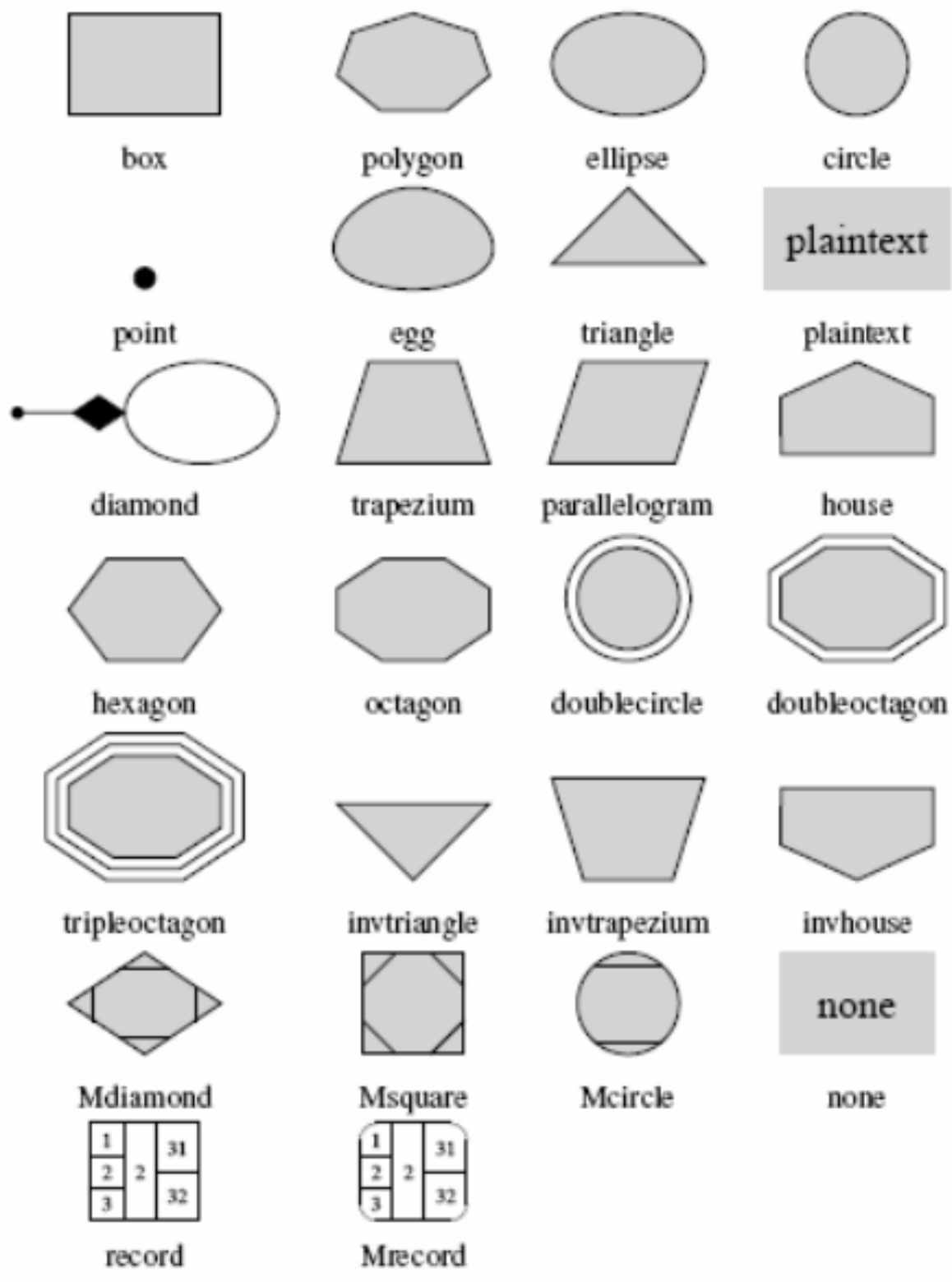
```
digraph G {
    compound=true;
    subgraph cluster0 {
      a -> b;
      a -> c;
      b -> d;
      c -> d;
    }
    subgraph cluster1 {
      e -> g;
      e -> f;
    }
    b -> f [lhead=cluster1];
    d -> e;
    c -> g [ltail=cluster0,
            lhead=cluster1];
    c -> e [ltail=cluster0];
    d -> h;
}
```



H Node Shapes

These are the principal node shapes. A more complete description of node shapes can be found at the web site

`www.graphviz.org/doc/info/shapes.html`



J Color Names

Here are some basic color names. More information about colors can be found at

`www.graphviz.org/doc/info/colors.html`
`www.graphviz.org/doc/info/attrs.html#k:color`

Whites
antiquewhite[1-4]
azure[1-4]
bisque[1-4]
blanchedalmond
cornsilk[1-4]
floralwhite
gainsboro
ghostwhite
honeydew[1-4]
ivory[1-4]
lavender
lavenderblush[1-4]
lemonchiffon[1-4]
linen
mintcream
mistyrose[1-4]
moccasin
navajowhite[1-4]
oldlace
papayawhip
peachpuff[1-4]
seashell[1-4]
snow[1-4]
thistle[1-4]
wheat[1-4]
white
whitesmoke

Greys
darkslategray[1-4]
dimgray
gray
gray[0-100]
lightgray
lightslategray
slategray[1-4]

Blacks
black

Reds
coral[1-4]
crimson
darksalmon
deeppink[1-4]
firebrick[1-4]
hotpink[1-4]
indianred[1-4]
lightpink[1-4]
lightsalmon[1-4]
maroon[1-4]
mediumvioletred
orangered[1-4]
palevioletred[1-4]
pink[1-4]
red[1-4]
salmon[1-4]
tomato[1-4]
violetred[1-4]

Browns
beige
brown[1-4]
burlywood[1-4]
chocolate[1-4]
darkkhaki
khaki[1-4]
peru
rosybrown[1-4]
saddlebrown
sandybrown
sienna[1-4]
tan[1-4]

Oranges
darkorange[1-4]
orange[1-4]
orangered[1-4]

Yellows
darkgoldenrod[1-4]
gold[1-4]
goldenrod[1-4]
greenyellow
lightgoldenrod[1-4]
lightgoldenrodyellow
lightyellow[1-4]
palegoldenrod
yellow[1-4]
yellowgreen

Greens
chartreuse[1-4]
darkgreen
darkolivegreen[1-4]
darkseagreen[1-4]
forestgreen
green[1-4]
greenyellow
lawngreen
lightseagreen
limegreen
mediumseagreen
mediumspringgreen
mintcream
olivedrab[1-4]
palegreen[1-4]
seagreen[1-4]
springgreen[1-4]
yellowgreen

Cyans
aquamarine[1-4]
cyan[1-4]
darkturquoise
lightcyan[1-4]
mediumaquamarine
mediumturquoise
paleturquoise[1-4]
turquoise[1-4]

Blues
aliceblue
blue[1-4]
blueviolet
cadetblue[1-4]
cornflowerblue
darkslateblue
deepskyblue[1-4]
dodgerblue[1-4]
indigo
lightblue[1-4]
lightskyblue[1-4]
lightslateblue[1-4]
mediumblue
mediumslateblue
midnightblue
navy
navyblue
powderblue
royalblue[1-4]
skyblue[1-4]
slateblue[1-4]
steelblue[1-4]

Magentas
blueviolet
darkorchid[1-4]
darkviolet
magenta[1-4]
mediumorchid[1-4]
mediumpurple[1-4]
mediumvioletred
orchid[1-4]
palevioletred[1-4]
plum[1-4]
purple[1-4]
violet
violetred[1-4]