GraphViz

Seminar, VI Semester

February 26, 2018

Prepared by: KARTHIK M

Karthik M
15CO221
VI Sem
CSE

Introduction, set-up and installation

What is GraphViz?

Graphviz (short for Graph Visualization Software) is a package of open-source tools initiated by AT&T Labs Research for drawing graphs specified in DOT language scripts. Graph visualization is a way of representing structural information as diagrams of abstract graphs and networks. It has important applications in networking, bioinformatics, software engineering, database and web design, machine learning, and in visual interfaces for other technical domains.

What is DOT?

DOT is a plain text graph description language. It is a simple way of describing graphs that both humans and computer programs can read. DOT graphs are typically files with the file extension *dot*

Installing GraphViz on Linux

Run the following command in the terminal:

sudo apt-get install graphviz

Rendering a DOT script on command line

Various programs can process DOT files. Some, such as *dot, neato, twopi and circo* can read a DOT file and render it in graphical form.

The general method of running a DOT script is as follows:
toolname -T<output_format> filename -o output.<extension>

Eg:

Rendering in png format using dot

dot -Tpng prg1.dot -o dotPNG.png

Rendering in PostScript format using circo

circo -Tps prg1.dot -o circoPS.ps

Rendering in GIF format using neato

neato -Tgif prg1.dot -o neatoGIF.gif

Some Applications of Graphviz

- **ArgoUML**
  ArgoUML renders UML diagrams using Graphviz

- **Doxygen**
  Doxygen uses Graphviz for generating class hierarchies and collaboration diagrams for source code

- **Bison**
  Bison can output the grammar as dot for visualization of the language.

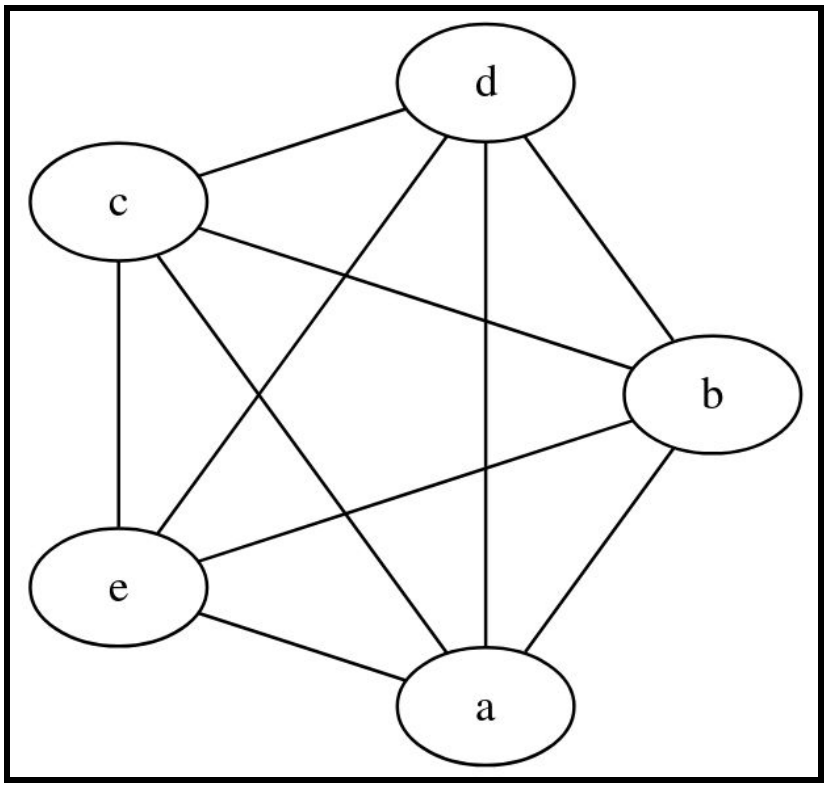
- **Sphinx**
  Sphinx is a documentation generator that can use Graphviz to embed graphs in documents.

Sample Programs in GraphViz

Undirected graphs

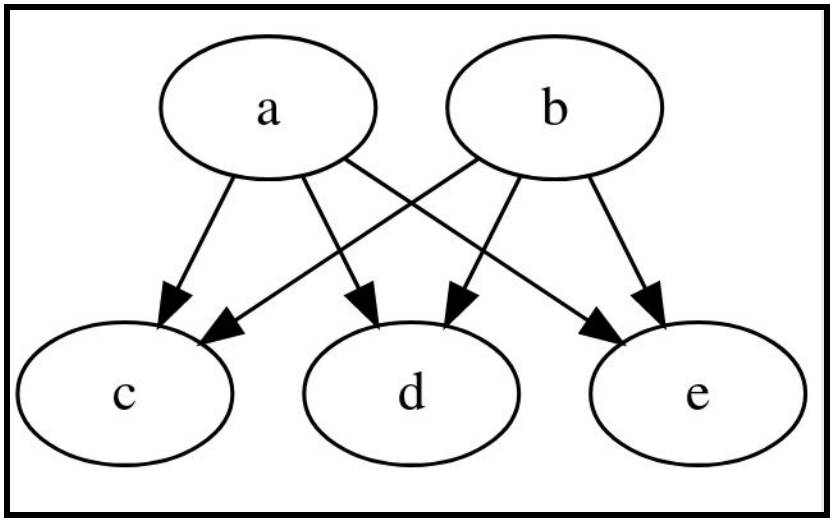
```
graph K5
{
        a--b
        a--c
        a--d
        a--e
        b--c
        b--d
```

```
        b--e
        c--d
        c--e
        d--e
}
```



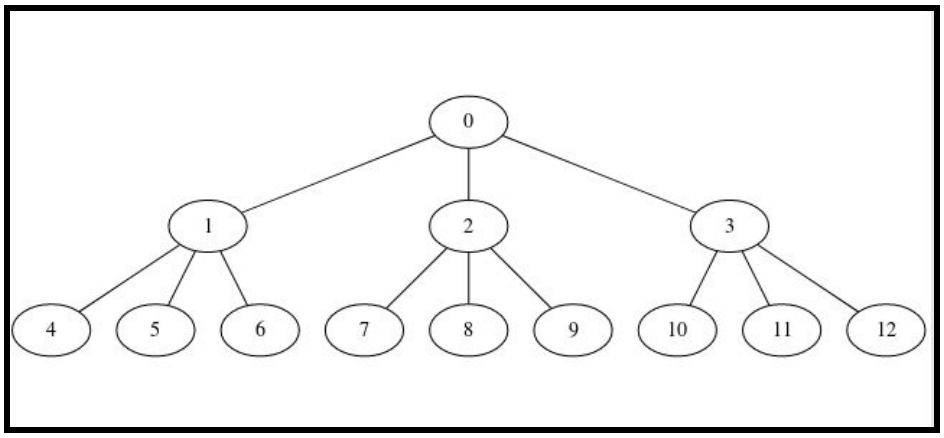
Directed graphs

```
digraph K_23
{
        a->c
        a->d
        a->e
        b->c
        b->d
        b->e
}
```



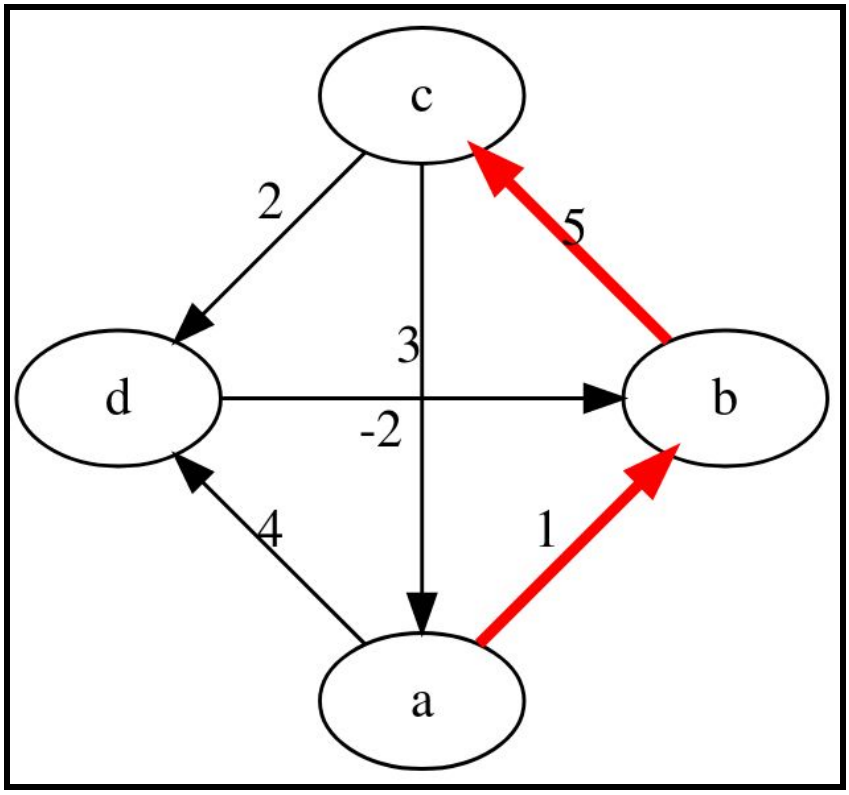
Hierarchy

```
graph tree4level
{
        0--{1 2 3}
        1--{4 5 6}
        2--{7 8 9}
        3--{10 11 12}
}
```



Weighted Graphs and Colored Edges

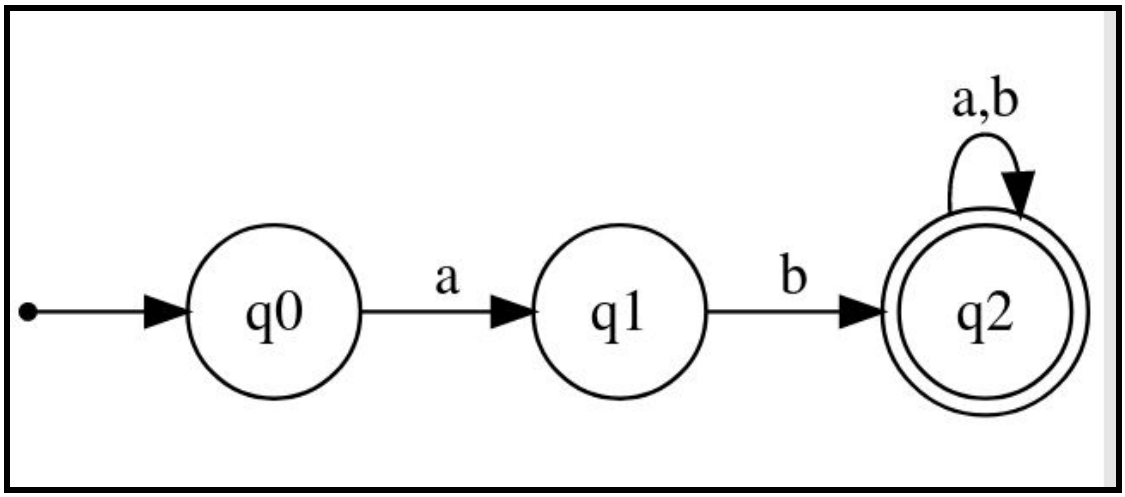
```
digraph shortest_path_weighted_graph
{
        a->b[label="1",color="red",penwidth=3];
        b->c[label="5",color="red",penwidth=3];
        c->d[label="2"];
        a->d[label="4"];
        d->b[label="-2"];
        c->a[label="3"];
}
```



Finite State Machine

```
digraph finite_state_machine
{
```

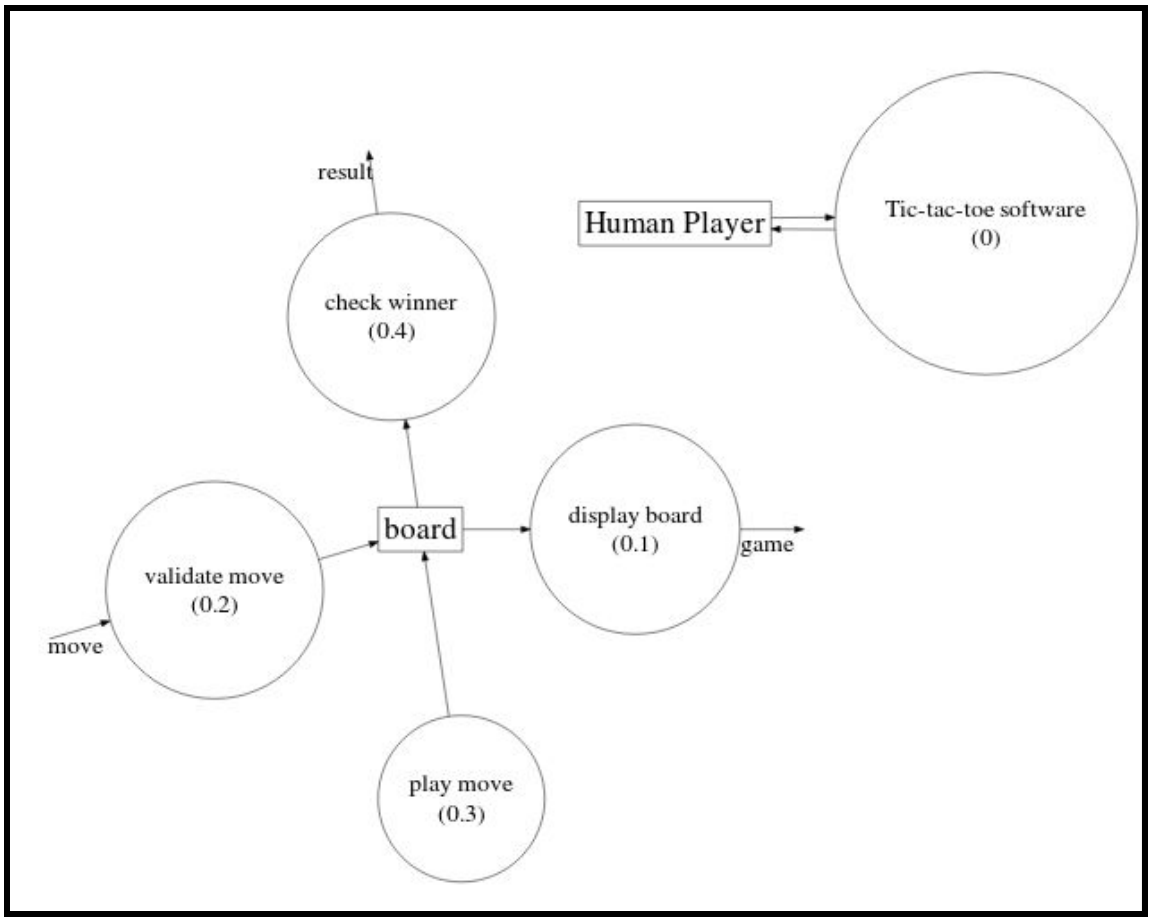
```
        rankdir=LR;

        node[shape=doublecircle];
        q2;
        node[shape=point];
        init;

        node[shape=circle];
        init->q0;
        q0->q1[label="a"];
        q1->q2[label="b"];
        q2->q2[label="a,b"];
}
```



Data Flow Diagram

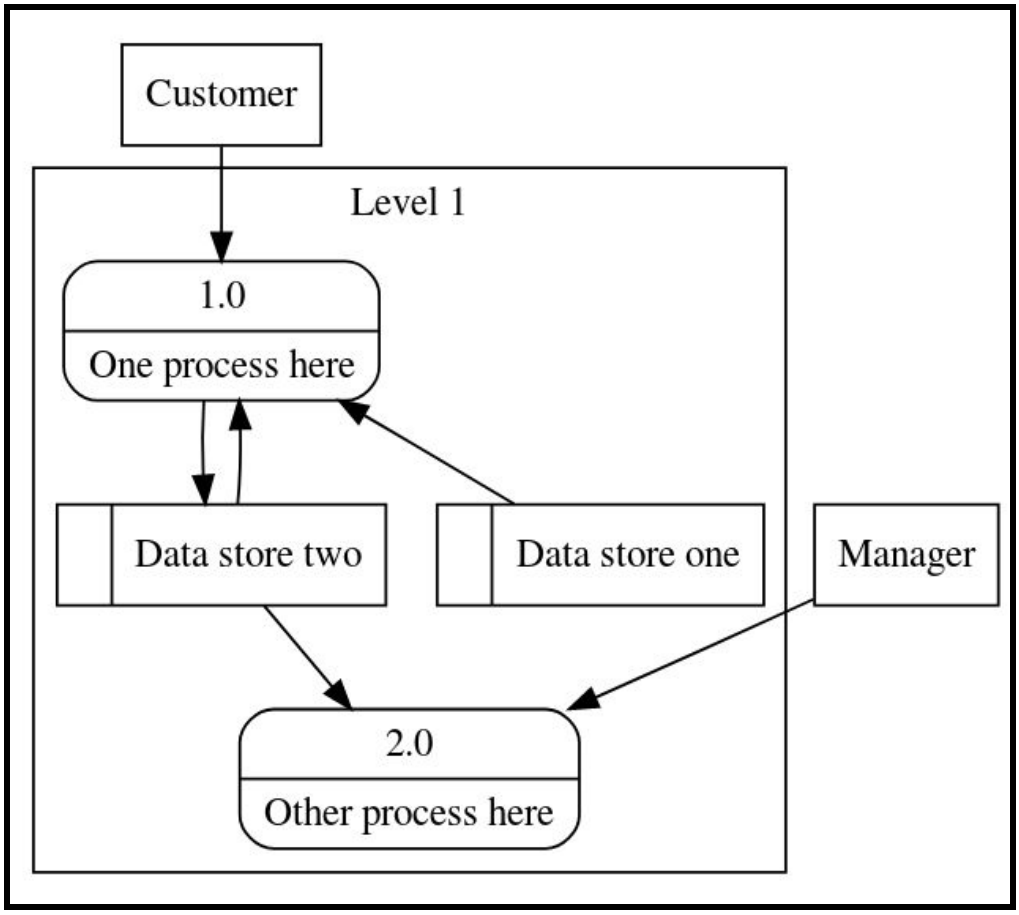
```
digraph Data_Flow_Diagram
{
        node[shape=rectangle,fontsize=28];
        HP[label="Human Player",fontsize=35];
        Board[label="board",fontsize=35];

        node[shape=circle,fontsize=28];
        0[label="Tic-tac-toe software\n(0)"];
        0.1[label="display board\n(0.1)"];
        0.2[label="validate move\n(0.2)"];
        0.3[label="play move\n(0.3)"];
        0.4[label="check winner\n(0.4)"];
```

```
        node[style=invis,fontsize=28];
        a,b,c;

        0->HP;
        HP->0;
        Board->{0.1,0.4};
        0.3->Board;
        0.2->Board;
        // Hp->;
        0.4->a[label="result",fontsize=28];
        0.1->b[label="game",fontsize=28];
        c->0.2[label="move",fontsize=28];
}
```



Collaboration Diagram

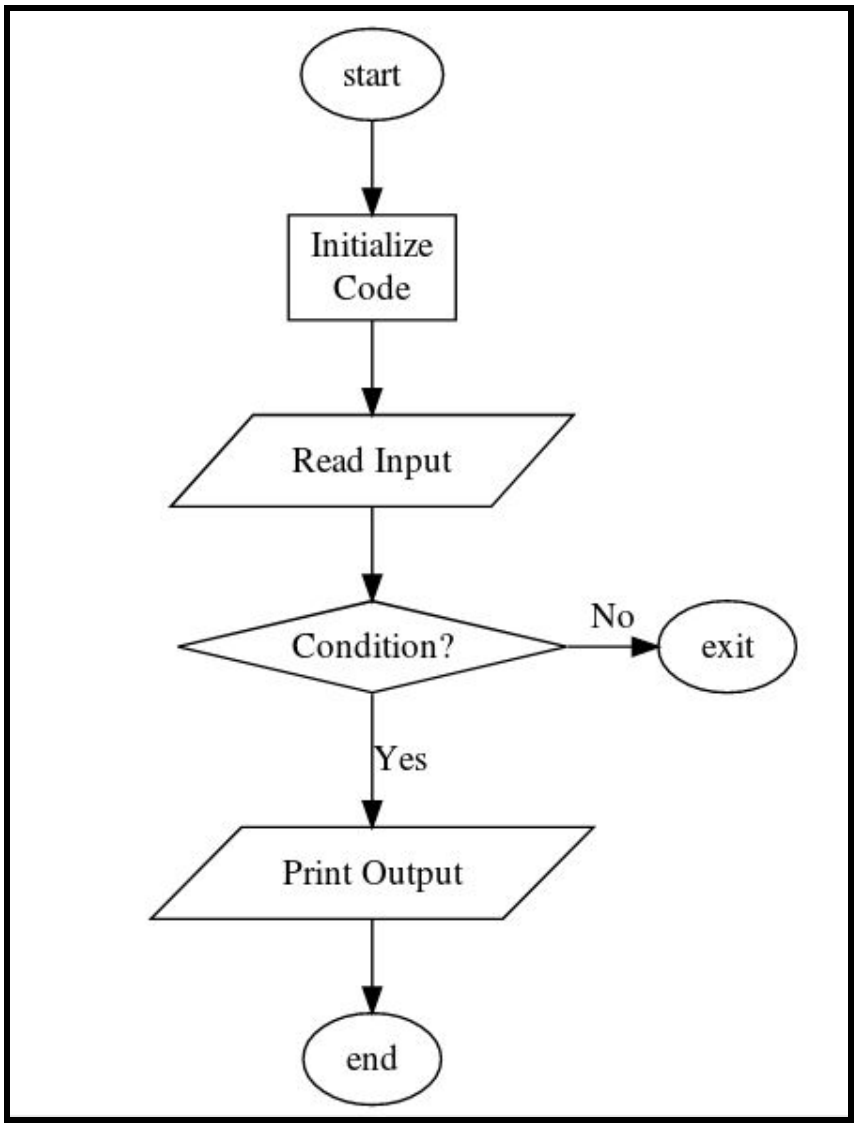
```
digraph CollaborationDiagram
{
        node[shape=record]

        customer [label="Customer" shape=box];
        manager [label="Manager" shape=box];


        subgraph cluster_level1
        {
        label ="Level 1";
        process1 [label="{1.0 | One process here}" shape=Mrecord];
        process2 [label="{ 2.0 | Other process here}" shape=Mrecord];
        store1 [label="| Data store one"];
        store2 [label="| Data store two"];
        {rank=same; store1, store2}

        }

        customer -> process1
        manager -> process2
        store1 -> process1
        store2 -> process2
        process1 -> store2
        store2 -> process1
}
```



Flow Chart

```
digraph flow_chart
{
        node [shape = oval] start end exit

        node [shape = parallelogram; label="Read Input"]; input;
        node [shape = parallelogram; label="Print Output"]; output;
        node [shape = diamond; label="Condition?"]; condition;
        node [shape = Rectangle; label="Initialize\nCode"]; init_process;


        start -> init_process;
        init_process -> input
        {rank = same; condition -> exit[label="No"]}

        input -> condition;
        condition -> output [label = "Yes"]
```

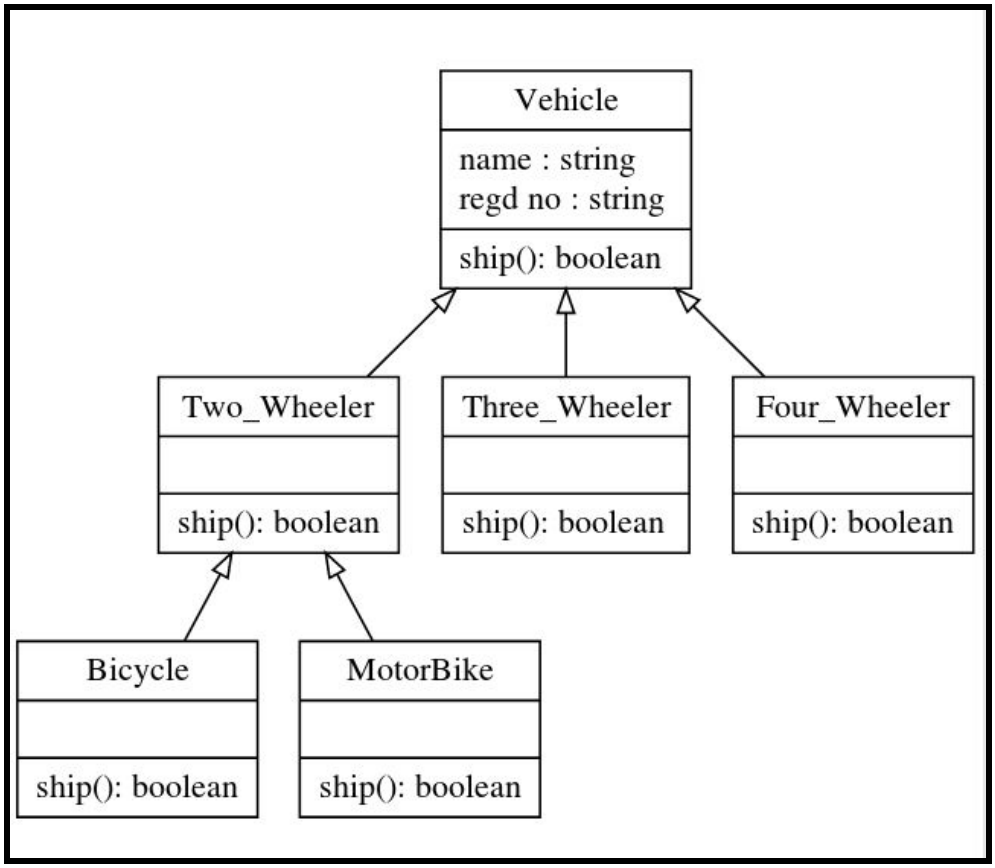
```
        output -> end
}
```



Class Diagram

```
digraph ClassDiagram
{
        node [shape = "record"]
        edge [arrowtail = "empty"]
```

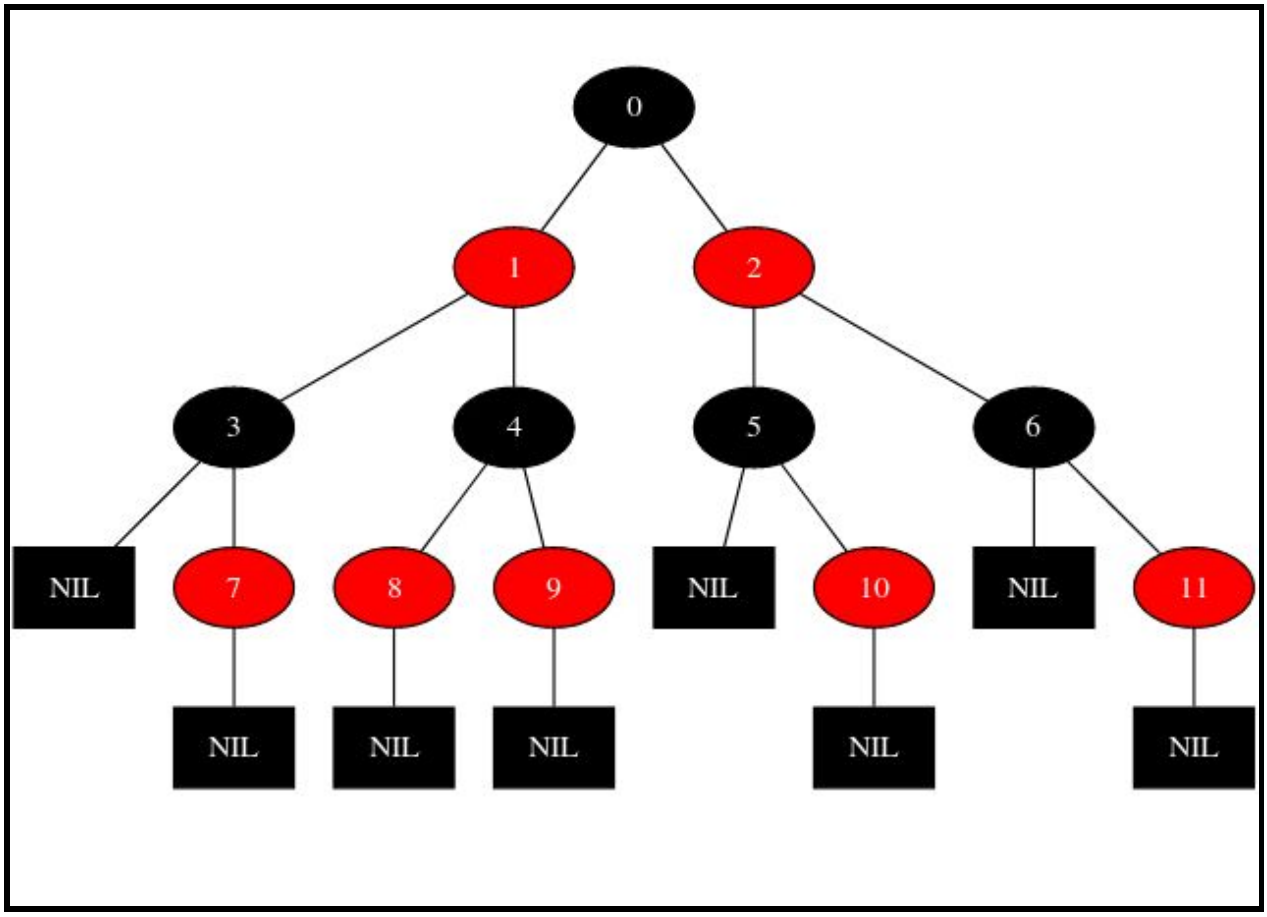
```
        Vehicle
        [
        label = "{Vehicle|name : string\l regd no : string\l | ship(): boolean\l}"
        ]

        Two_Wheeler
        [
        label = "{Two_Wheeler|\l|ship(): boolean\l}"
        ]

        Three_Wheeler
        [
        label = "{Three_Wheeler|\l|ship(): boolean\l}"
        ]

        Four_Wheeler
        [
        label = "{Four_Wheeler|\l|ship(): boolean\l}"
        ]

        Bicycle
        [
        label = "{Bicycle|\l|ship(): boolean\l}"
        ]

        MotorBike
        [
        label = "{MotorBike|\l|ship(): boolean\l}"
        ]

        Vehicle -> Three_Wheeler [dir=back]
        Vehicle -> Two_Wheeler [dir=back]
        Vehicle -> Four_Wheeler [dir=back]
        Two_Wheeler -> Bicycle [dir=back]
        Two_Wheeler -> MotorBike [dir=back]
}
```



Red Black Tree

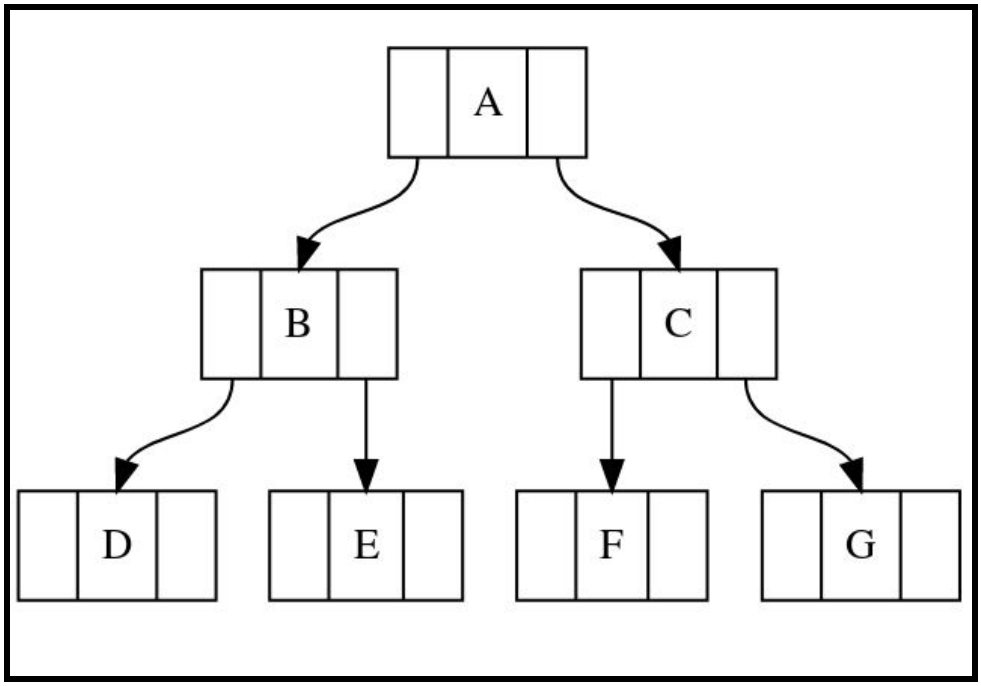
```
graph RedBlackTree
{
        {
        node[style=filled fillcolor=black fontcolor=white];
        0,3,4,5,6;
        }

        {
        node
        [
                style=filled
                shape=rectangle
                fillcolor=black
                fontcolor=white
                label=NIL
        ];
        nil1,nil2,nil3,nil4,nil5,nil6,nil7,nil8;
```

```
        }

        {
        node[style=filled fillcolor=red fontcolor=white];
        1,2,7,8,9,10,11;
        }

        0--{1 2}
        1--{3 4}
        2--{5 6}
        3--{nil1 7}
        4--{8 9}
        5--{nil5 10}
        6--{nil7 11}
        7--nil2
        8--nil3
        9--nil4
        10--nil6
        11--nil8
}
```



B Tree

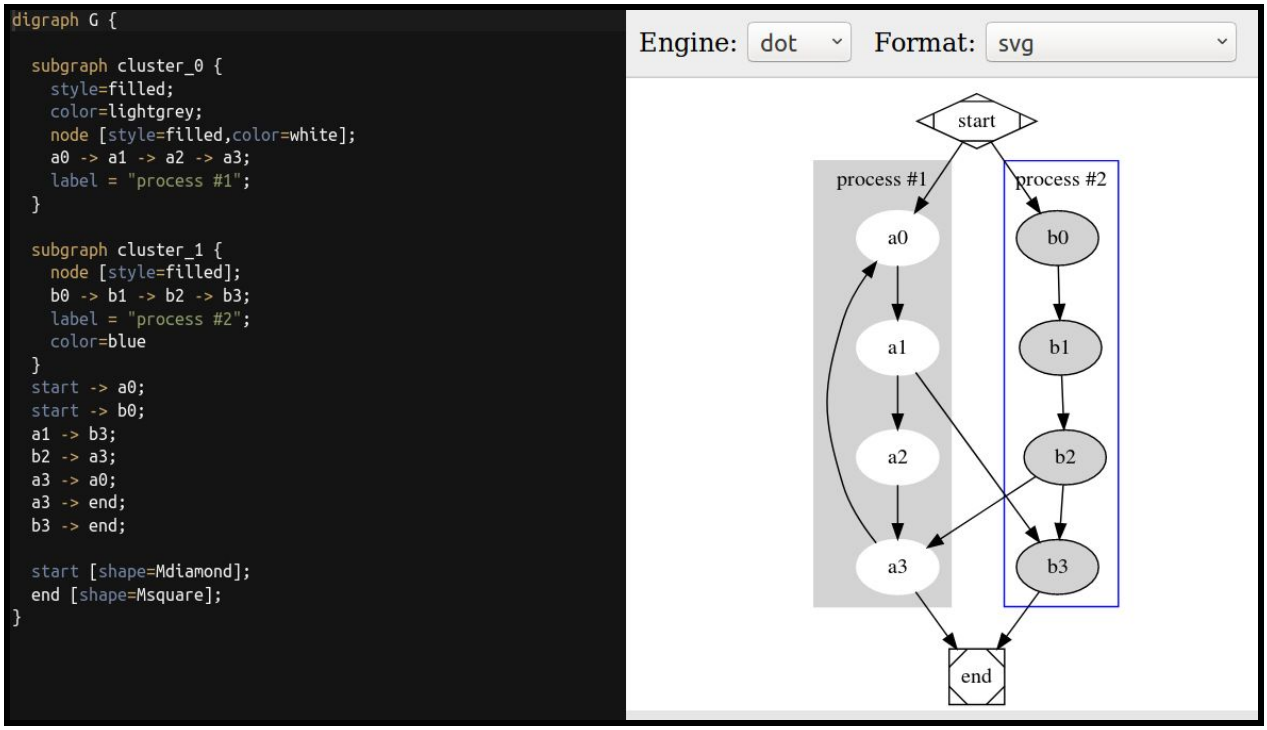
```
digraph B_Tree
{
	node [shape=record];

	node0 [label = "<l> | <m> A | <r>"];
	node1 [label = "<l> | <m> B | <r>"];
	node2 [label = "<l> | <m> C | <r>"];
	node3 [label = "<l> | <m> D | <r>"];
	node4 [label = "<l> | <m> E | <r>"];
	node5 [label = "<l> | <m> F | <r>"];
	node6 [label = "<l> | <m> G | <r>"];

	node0:l -> node1:m;
	node0:r -> node2:m;
	node1:l -> node3:m;
	node1:r -> node4:m;
	node2:l -> node5:m;
	node2:r -> node6:m;
}
```



References

1. https://dreampuf.github.io/GraphvizOnline/

   For online editor to see an instantaneous output



2. http://tonyballantyne.com/graphs.html#sec-8-2-3

   Tutorial on GraphViz

3. https://www.graphviz.org/

   Documentation of GraphViz