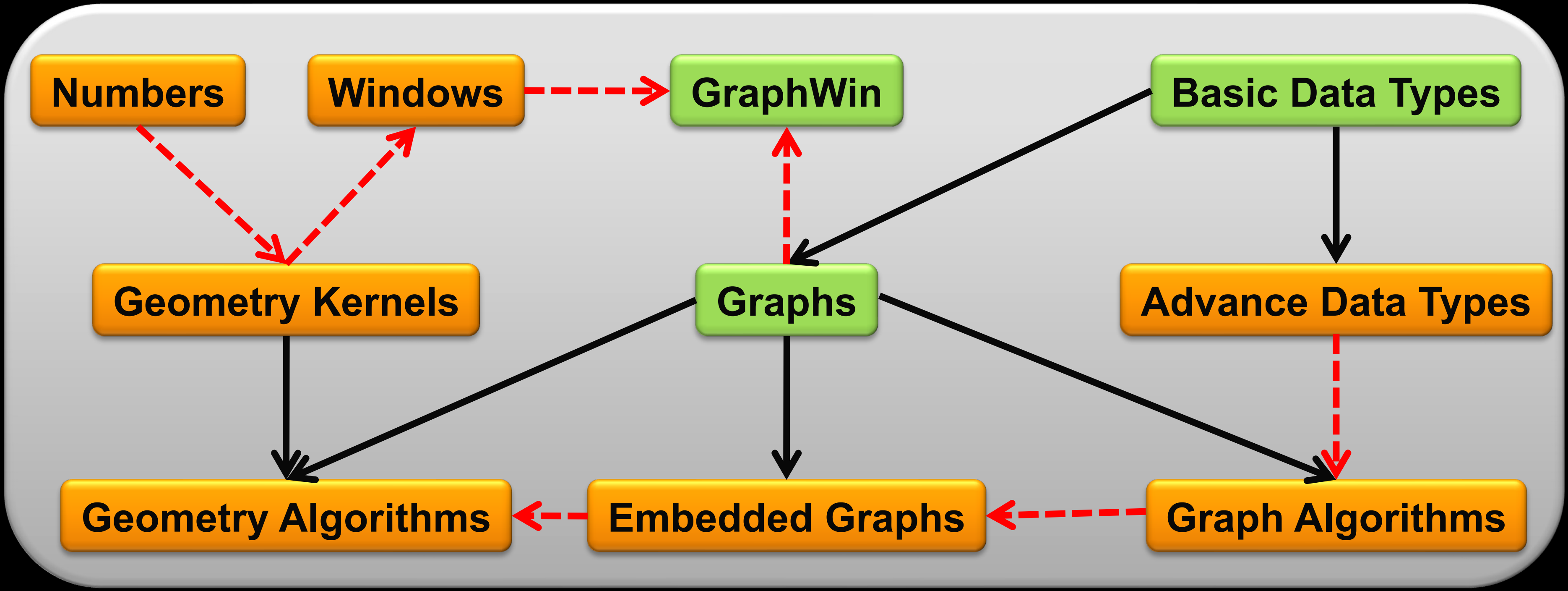
Library of Efficient Data types and Algorithms (LEDA)

Outline

- **Introduction to LEDA**
  - Basic data type
  - Graphs
  - GraphWin
- **Resources of LEDA**

LEDA Overview

- **C++ class library for efficient data types and algorithms**
  - Graph and network problems, geometric computations, combinatorial optimization



Ref: "*LEDA A Platform for Combinatorial and Geometric Computing*" CH.0 P.14 Fig.A

Basic Data Type

- **String**
- **Tuple**

```
#include <LEDA/core/tuple.h>
#include <LEDA/core/string.h>

using namespace leda;

int main()
{
  three_tuple<int,string,double>
      triple(17,"triple",3.1413);
  std::cout << triple << std::endl;
  return 0;
}
```

Container

- **Array**
- **Dictionary Array**

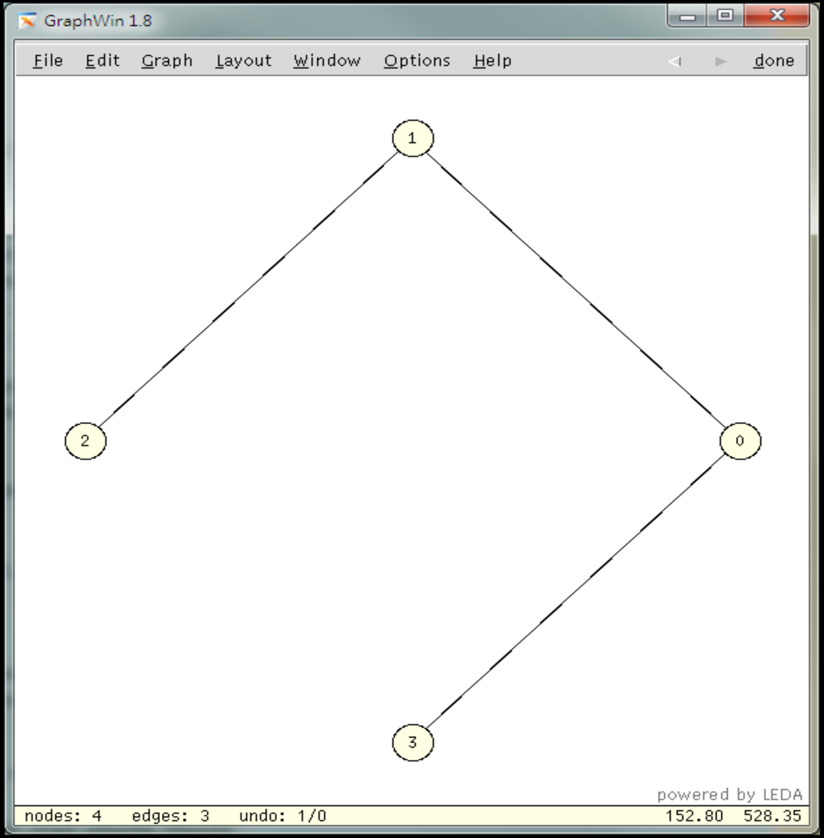
```
d_array<string,string> D;
//objects of type string, keys of type string

D["hello"]="hallo"; D["world"]="Welt"; D["book"]="Buch";
string s;
forall_defined(s,D) std::cout << s <<
    " " << D[s] << std::endl;
```

GraphWin

- **The GraphWin combines Graphs and Windows**
- **Applications**
  - An Interactive GUI
  - Construct and display graphs
  - Visualize graphs and the results of graph algorithms

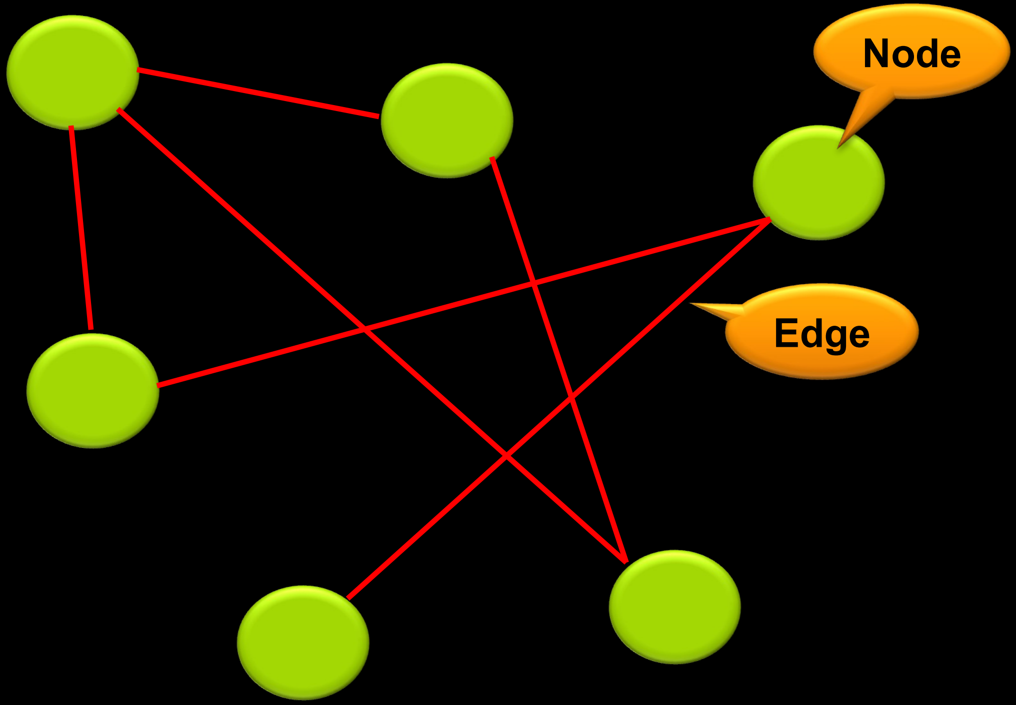
Create a GraphWin

```
GRAPH<int, int> G;
GraphWin gw;
//initial the graph
random_simple_undirected_graph(G, 4, 3);
Make_Connected(G);

//set graphwin
gw.set_graph(G);
gw.set_edge_direction(undirected_edge);

//show graphwin
gw.display(window::center,window::center);
gw.edit();
```

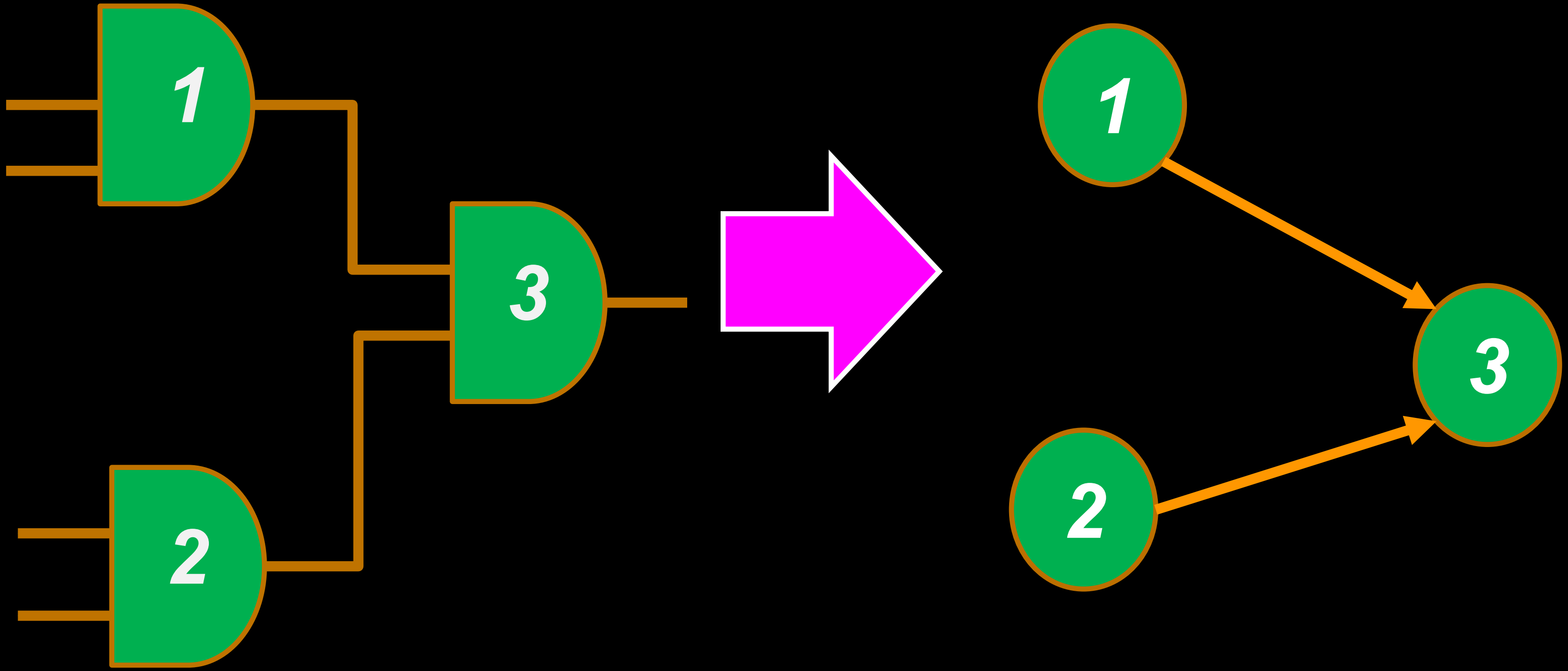

GraphWin 1.8
File Edit Graph Layout Window Options Help done
1
2
0
3
powered by LEDA
nodes: 4 edges: 3 undo: 1/0 152.80 528.35

Graph

- **Elements in a Graph**
  - Node set
  - Edge set



Graph Representation



Graph Data Structure

- **Node**
  - Node name
  - Neighbor
  - Serial number
  - Weight
- **Edge**
  - Edge name
  - Serial number
  - Weight
  - Source
  - Sink

```
class NODE{
		string name;
		vector<NODE> neighbor;
		int sn;
		int weight;
};
```

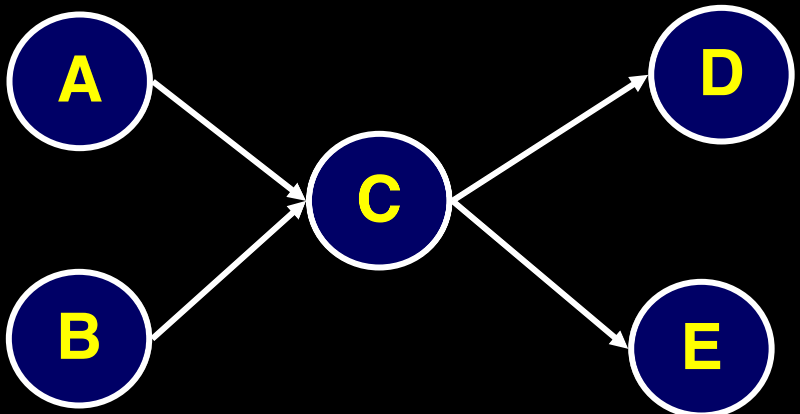
```
class EDGE {
		string name;
		int sn;
		int weight;
		NODE source;
		NODE sink;
};
```

Basic Graph Operation

- Insert a node
- Delete a node
- Insert an edge
- Delete an edge

Graphs

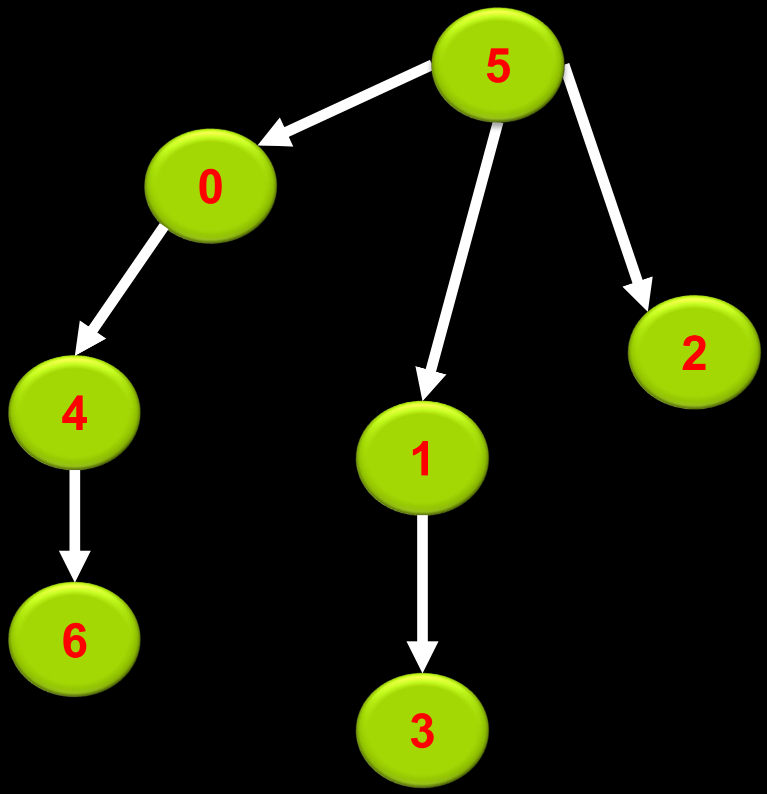
```
GRAPH<string, string> G;
node n_temp1, n_temp2, n_temp3, n_temp4, n_temp5;

n_temp1 = G.new_node("A");
n_temp2 = G.new_node("B");
n_temp3 = G.new_node("C");
n_temp4 = G.new_node("D");
n_temp5 = G.new_node("E");

G.new_edge(n_temp1, n_temp2);
G.new_edge(n_temp2, n_temp3);
G.new_edge(n_temp3, n_temp4);
G.new_edge(n_temp3, n_temp5);
```



Graph Traversal Example

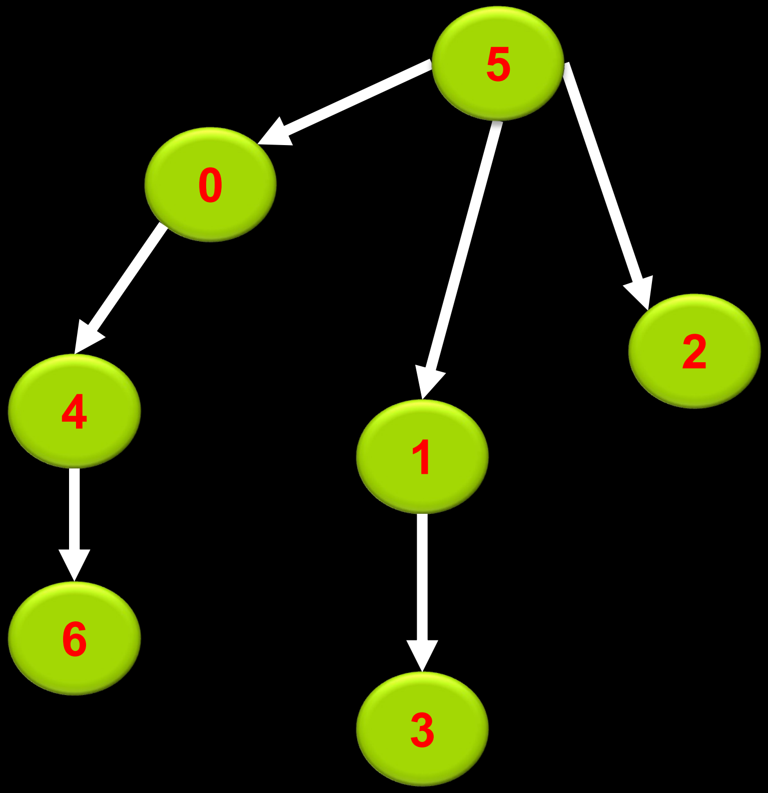
- **Depth-First Search**

5→0→4→6→1→3→2

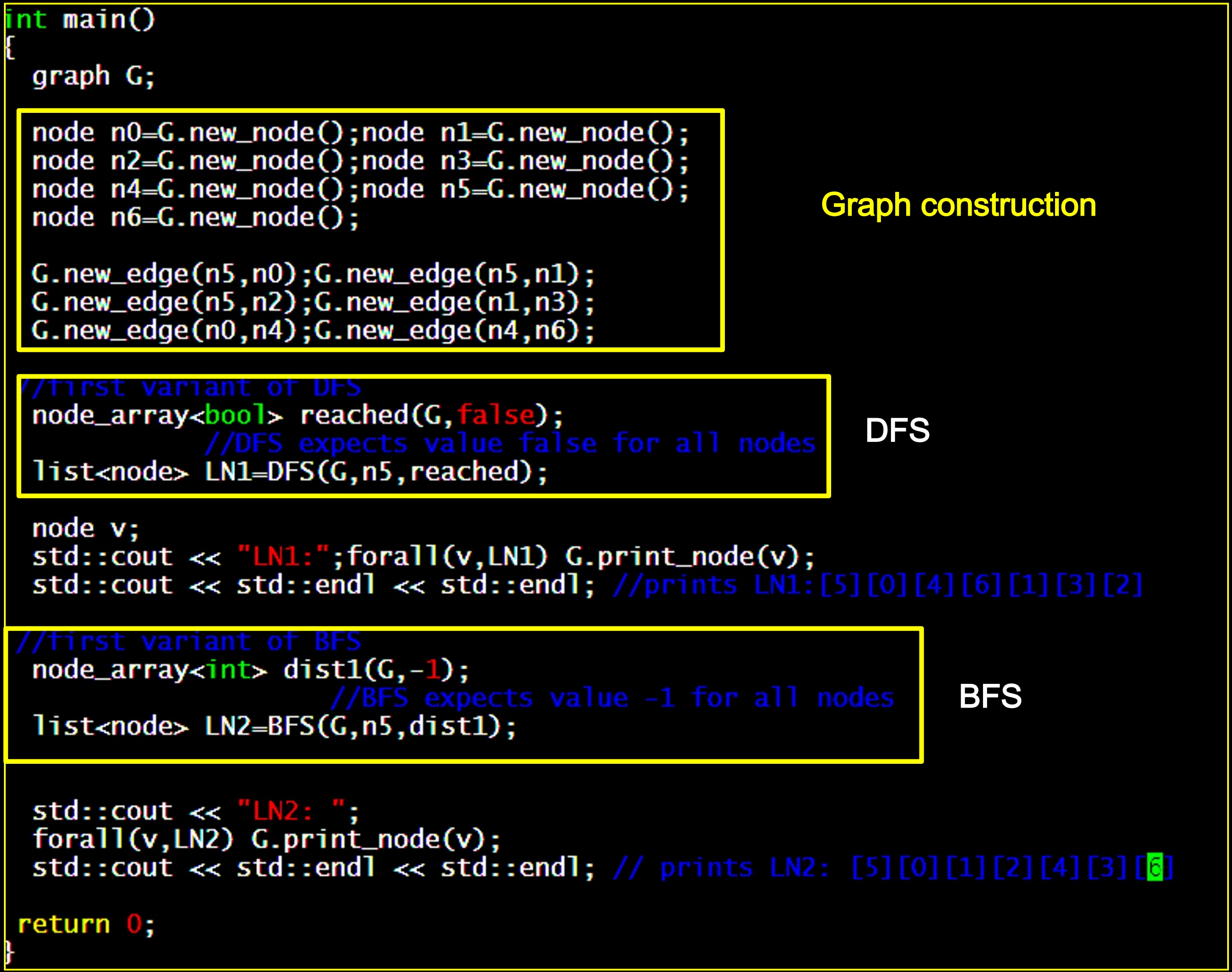


Bread-First Search

5→0→1→2→4→3→6



Example Code



```
int main()
{
  graph G;

  node n0=G.new_node();node n1=G.new_node();
  node n2=G.new_node();node n3=G.new_node();
  node n4=G.new_node();node n5=G.new_node();
  node n6=G.new_node();

  G.new_edge(n5,n0);G.new_edge(n5,n1);
  G.new_edge(n5,n2);G.new_edge(n1,n3);
  G.new_edge(n0,n4);G.new_edge(n4,n6);

 //first variant of DFS
  node_array<bool> reached(G,false);
              //DFS expects value false for all nodes
  list<node> LN1=DFS(G,n5,reached);

  node v;
  std::cout << "LN1:";forall(v,LN1) G.print_node(v);
  std::cout << std::endl << std::endl; //prints LN1:[5][0][4][6][1][3][2]

 //first variant of BFS
  node_array<int> dist1(G,-1);
                      //BFS expects value -1 for all nodes
  list<node> LN2=BFS(G,n5,dist1);


  std::cout << "LN2: ";
  forall(v,LN2) G.print_node(v);
  std::cout << std::endl << std::endl; // prints LN2: [5][0][1][2][4][3][6]

 return 0;
}
```

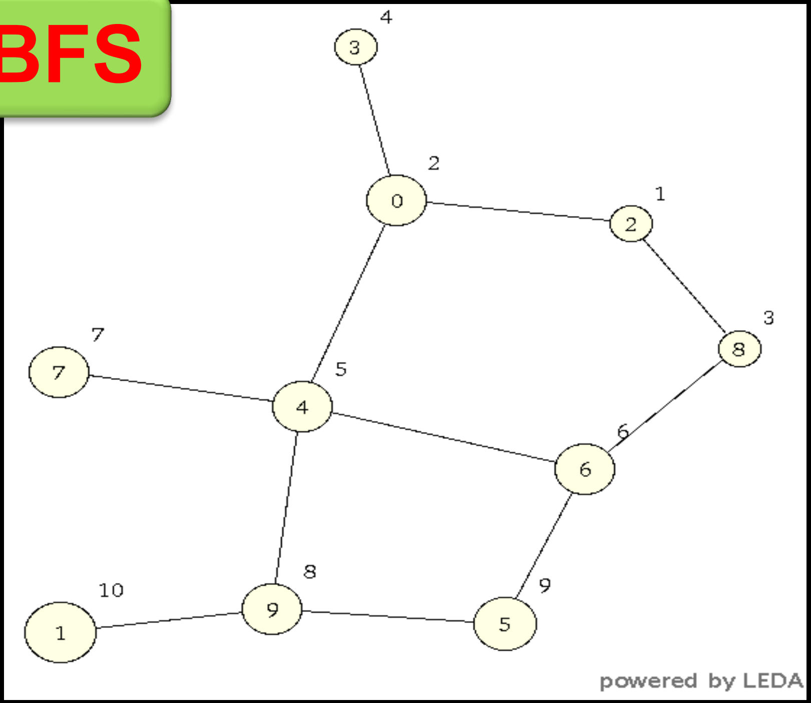
Graph construction

DFS

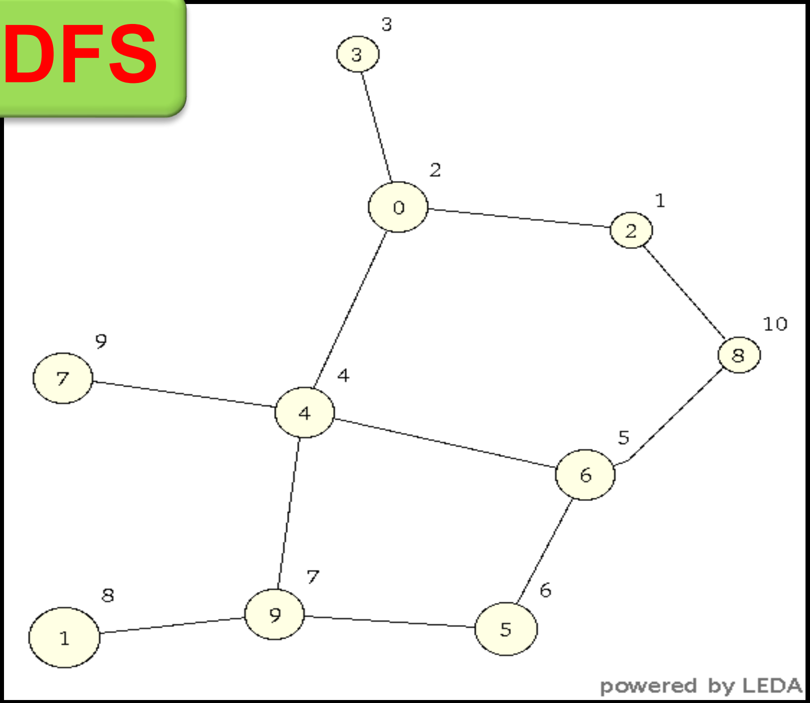
BFS

Graph Traversal Visualization

BFS



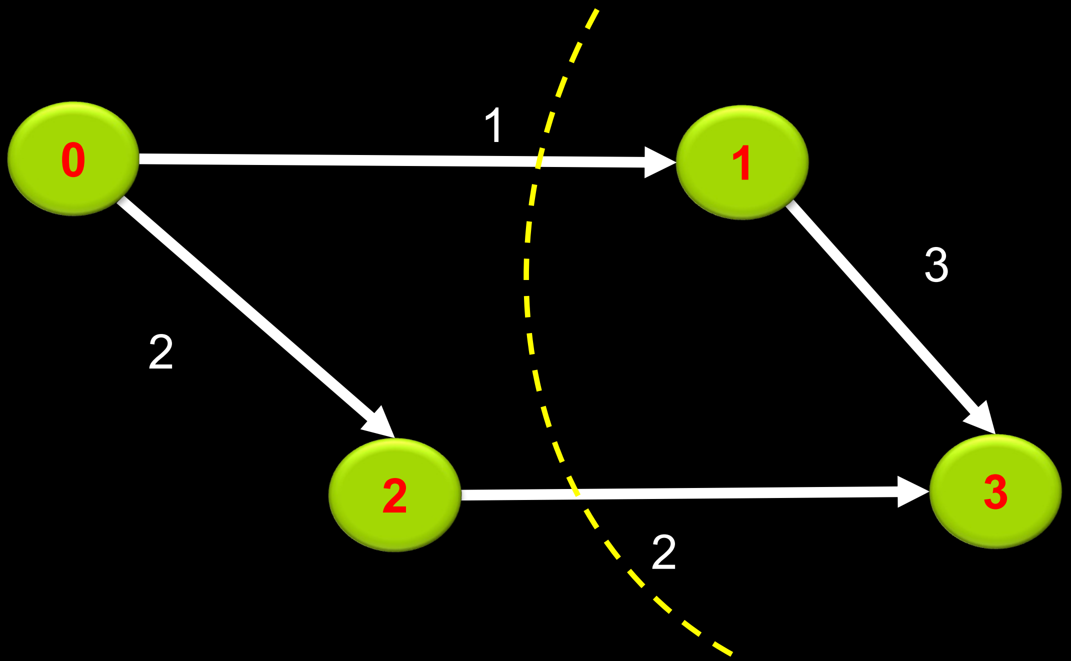
DFS



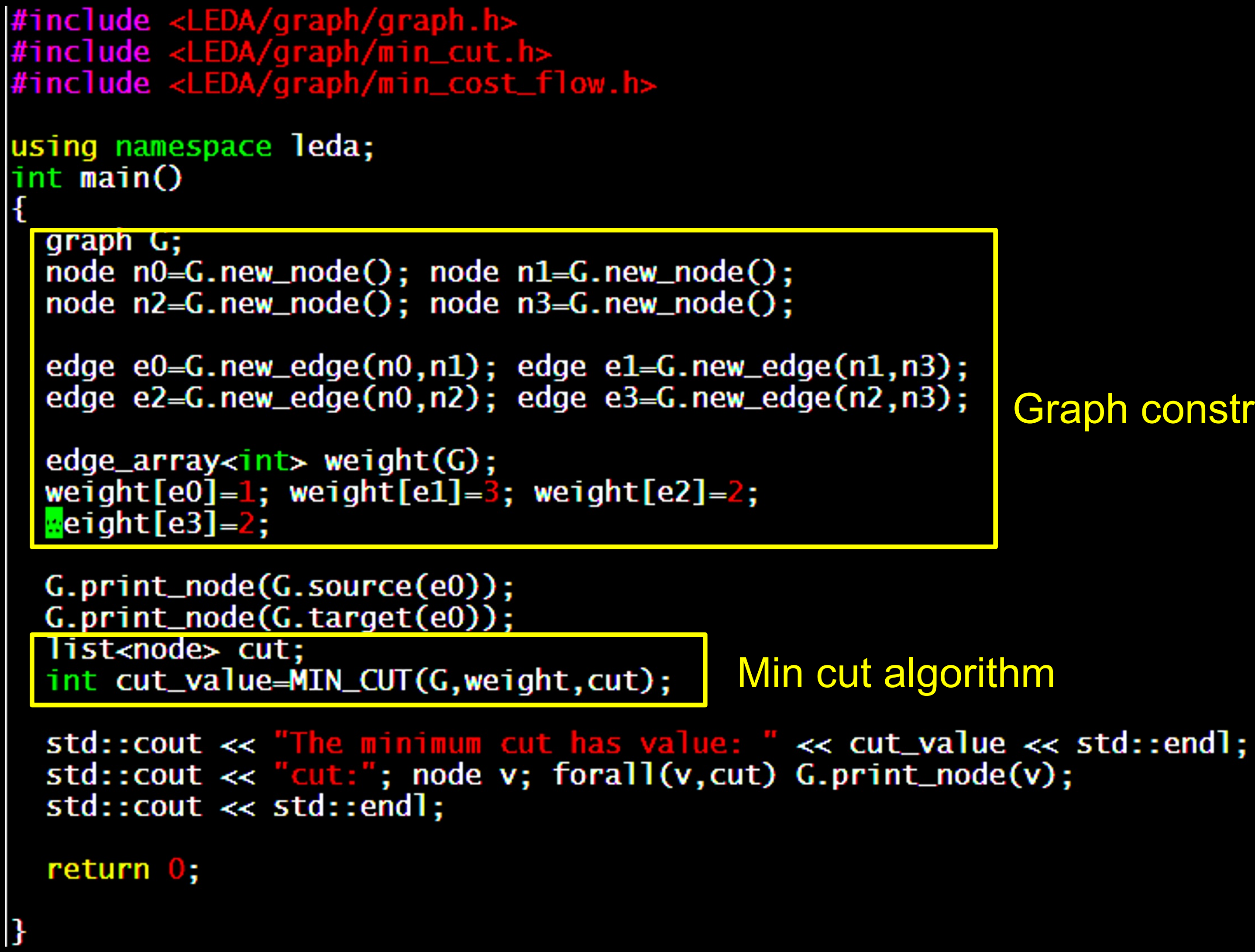
Min Cut Example

The minimum cut has value: 3

cut:[3][1]



Example Code



```
#include <LEDA/graph/graph.h>
#include <LEDA/graph/min_cut.h>
#include <LEDA/graph/min_cost_flow.h>

using namespace leda;
int main()
{
  graph G;
  node n0=G.new_node(); node n1=G.new_node();
  node n2=G.new_node(); node n3=G.new_node();

  edge e0=G.new_edge(n0,n1); edge e1=G.new_edge(n1,n3);
  edge e2=G.new_edge(n0,n2); edge e3=G.new_edge(n2,n3);

  edge_array<int> weight(G);
  weight[e0]=1; weight[e1]=3; weight[e2]=2;
  weight[e3]=2;

  G.print_node(G.source(e0));
  G.print_node(G.target(e0));
  list<node> cut;
  int cut_value=MIN_CUT(G,weight,cut);

  std::cout << "The minimum cut has value: " << cut_value << std::endl;
  std::cout << "cut:"; node v; forall(v,cut) G.print_node(v);
  std::cout << std::endl;

  return 0;

}
```

Graph construction

Min cut algorithm

Outline

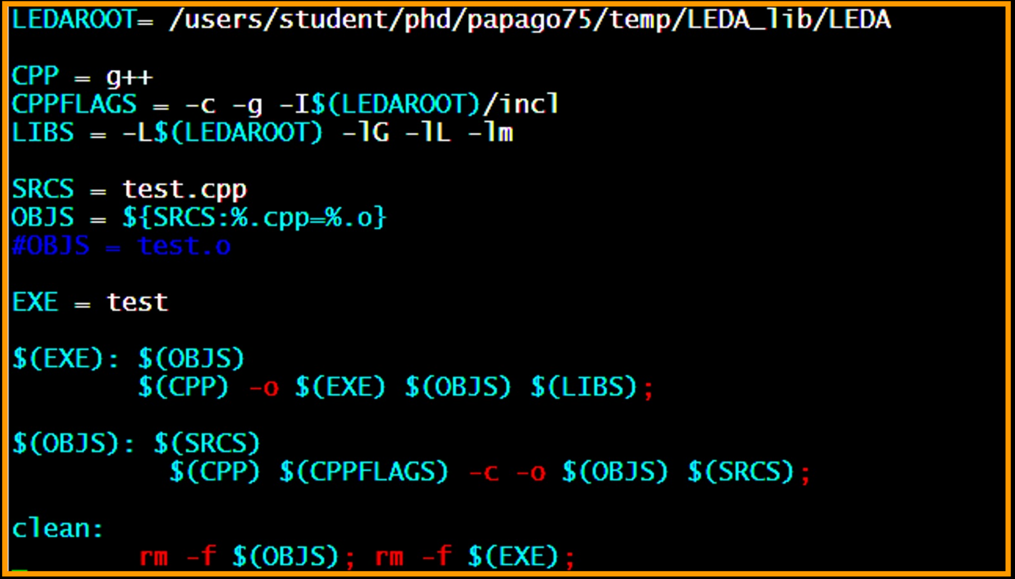
- **Introduction to LEDA**
  - Basic data type
  - Graphs
  - GraphWin
- **Resources of LEDA**

Resource of LEDA

- **LEDA Office Page**
  - http://www.algorithmic-solutions.com/leda/
- **LEDA User Manual**
  - http://www.algorithmic-solutions.info/leda_manual/manual.html
- **LEDA Guide**
  - http://www.algorithmic-solutions.info/leda_guide/Index.html
- **The LEDA Platform of Combinatorial and Geometric Computing**
  - http://www.mpi-inf.mpg.de/~mehlhorn/LEDAbook.html

Compilation on Workstation

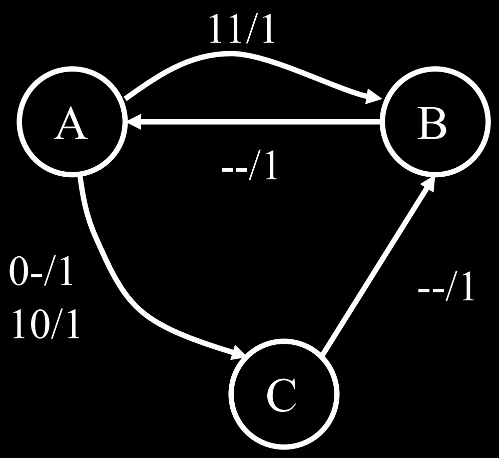
- **In NTHU-CAD**
  - g++ -c –g -I/users/student/yourid/LEDA_lib/LEDA/incl -c -o test.o test.cpp
  - g++ -o test test.o -L/users/student//yourid/LEDA_lib/LEDA -lG -lL -lm;
- **g++ parameters**
  - -I: location of the LEDA header files
  - -L: location the LEDA library files

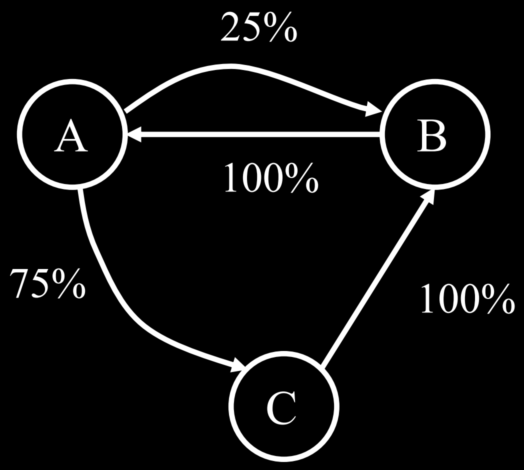


```
LEDAROOT= /users/student/phd/papago75/temp/LEDA_lib/LEDA

CPP = g++
CPPFLAGS = -c -g -I$(LEDAROOT)/incl
LIBS = -L$(LEDAROOT) -lG -lL -lm

SRCS = test.cpp
OBJS = ${SRCS:%.cpp=%.o}
#OBJS = test.o

EXE = test

$(EXE): $(OBJS)
        $(CPP) -o $(EXE) $(OBJS) $(LIBS);

$(OBJS): $(SRCS)
         $(CPP) $(CPPFLAGS) -c -o $(OBJS) $(SRCS);

clean:
        rm -f $(OBJS); rm -f $(EXE);
```

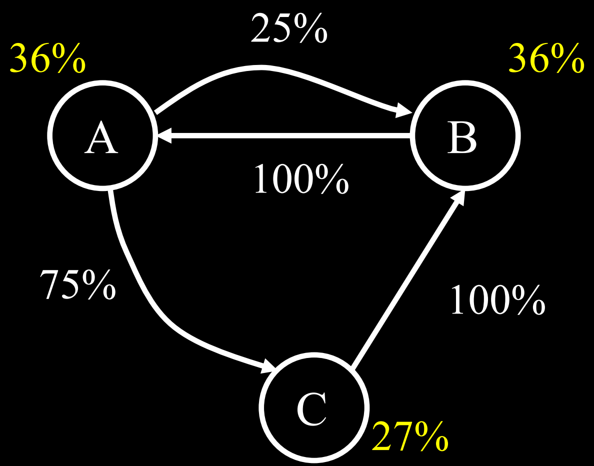
Appendix

State Probability Calculation



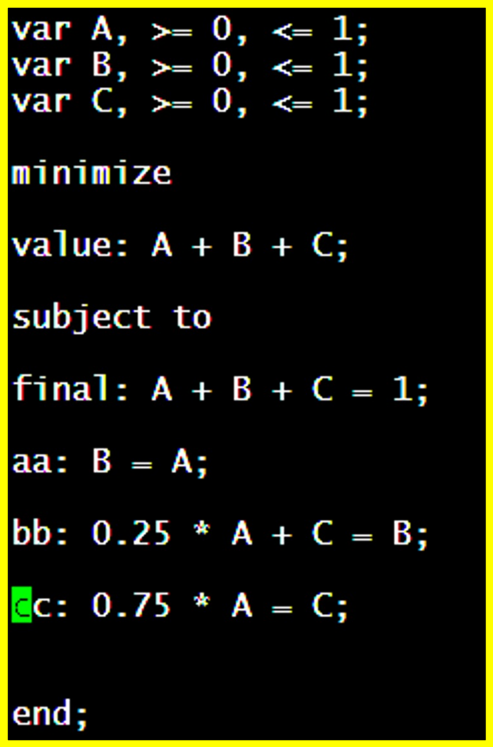


$$A + B + C = 1$$
$$B = A$$
$$0.25 * A + C = B$$
$$0.75 * A = C$$



Linear Programming Solver

sample.mod



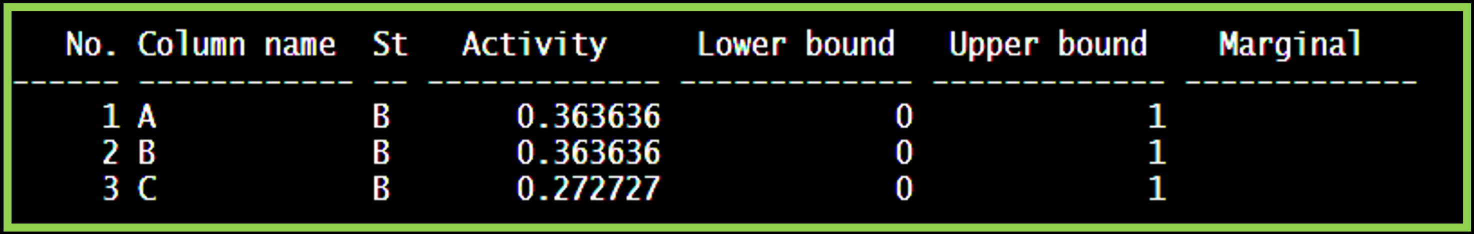
```
var A, >= 0, <= 1;
var B, >= 0, <= 1;
var C, >= 0, <= 1;

minimize

value: A + B + C;

subject to

final: A + B + C = 1;

aa: B = A;

bb: 0.25 * A + C = B;

cc: 0.75 * A = C;


end;
```

$A + B + C = 1$
$B = A$
$0.25 * A + C = B$
$0.75 * A = C$

./glpsol -m sample.mod -o sample.out

GLPK (GNU Linear Programing Kit)

sample.out



```
   No. Column name  St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- -------------
     1 A            B       0.363636             0             1
     2 B            B       0.363636             0             1
     3 C            B       0.272727             0             1
```