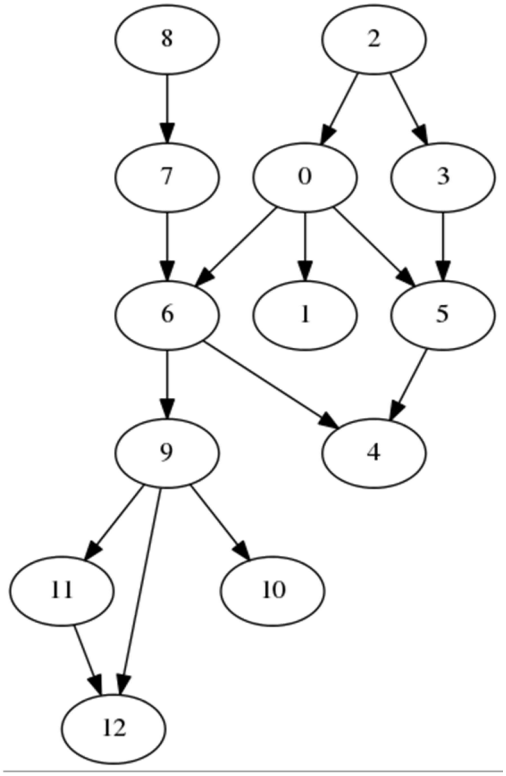
Computational Lab - Assignment 4: Directed Acyclic Graph (DAG) with STL-Plus Lib
By: Pankaj Azad, Roll Number - 14M517, M.Tech Ist Sem

Enter number of vertices...13
Enter the edges (E.g. 4 3 means a directed edge from 4 to 3). -1 -1 as input stops the process
0 5
0 1
0 6
2 0
2 3
3 5
5 4
6 4
6 9
7 6
8 7
9 11
9 12
9 10
11 12
-1 -1

Enter any of the following available choices
1 - Add Edge(s)
2 - Delete Edge(s)
3 - Print Adjacency list and draw graph (generates a png image)
4 - Depth First Search (DFS)
5 - Detect existence of cycle
9 - Exit
3
DAG: 13 Vertices and 15 Edges
[0] => 5 1 6
[1] =>
[2] => 0 3
[3] => 5
[4] =>
[5] => 4
[6] => 4 9
[7] => 6
[8] => 7
[9] => 11 12 10
[10] =>
[11] => 12
[12] =>

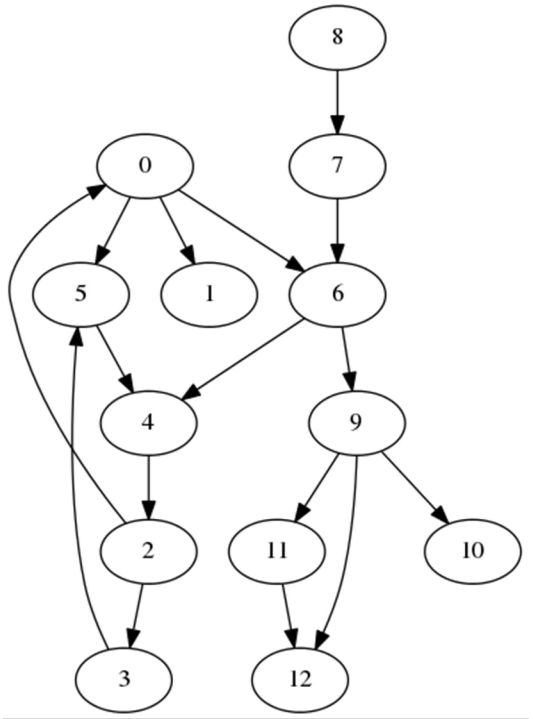


Enter any of the following available choices
1 - Add Edge(s)
2 - Delete Edge(s)

3 - Print Adjacency list and draw graph (generates a png image)
4 - Depth First Search (DFS)
5 - Detect existence of cycle
9 - Exit
5
No Cycle found!!

Enter any of the following available choices
1 - Add Edge(s)
2 - Delete Edge(s)
3 - Print Adjacency list and draw graph (generates a png image)
4 - Depth First Search (DFS)
5 - Detect existence of cycle
9 - Exit
1
Enter the edges (E.g. 4 3 means a directed edge from 4 to 3). -1 -1 as input stops the process
4 2
-1 -1

Enter any of the following available choices
1 - Add Edge(s)
2 - Delete Edge(s)
3 - Print Adjacency list and draw graph (generates a png image)
4 - Depth First Search (DFS)
5 - Detect existence of cycle
9 - Exit
3
DAG: 13 Vertices and 16 Edges
[0] => 5 1 6
[1] =>
[2] => 0 3
[3] => 5
[4] => 2
[5] => 4
[6] => 4 9
[7] => 6
[8] => 7
[9] => 11 12 10
[10] =>
[11] => 12
[12] =>



Enter any of the following available choices
1 - Add Edge(s)
2 - Delete Edge(s)
3 - Print Adjacency list and draw graph (generates a png image)
4 - Depth First Search (DFS)
5 - Detect existence of cycle

9 - Exit
5
Cycle Exits!!

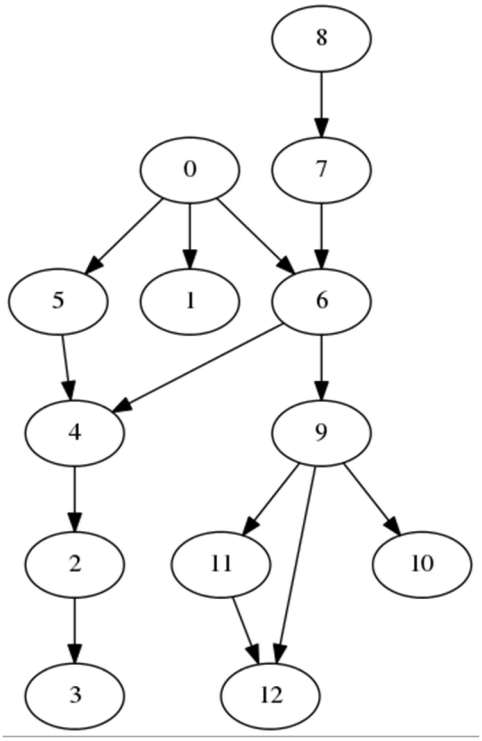
Enter any of the following available choices
1 - Add Edge(s)
2 - Delete Edge(s)
3 - Print Adjacency list and draw graph (generates a png image)
4 - Depth First Search (DFS)
5 - Detect existence of cycle
9 - Exit
2
Enter the edges (E.g. 4 3 means a directed edge from 4 to 3). -1 -1 as input stops the process
3 5
-1 -1

Enter any of the following available choices
1 - Add Edge(s)
2 - Delete Edge(s)
3 - Print Adjacency list and draw graph (generates a png image)
4 - Depth First Search (DFS)
5 - Detect existence of cycle
9 - Exit
5
Cycle Exits!!

Enter any of the following available choices
1 - Add Edge(s)
2 - Delete Edge(s)
3 - Print Adjacency list and draw graph (generates a png image)
4 - Depth First Search (DFS)
5 - Detect existence of cycle
9 - Exit
2
Enter the edges (E.g. 4 3 means a directed edge from 4 to 3). -1 -1 as input stops the process
2 0
-1 -1

Enter any of the following available choices
1 - Add Edge(s)
2 - Delete Edge(s)
3 - Print Adjacency list and draw graph (generates a png image)
4 - Depth First Search (DFS)
5 - Detect existence of cycle
9 - Exit
5
No Cycle found!!

Enter any of the following available choices
1 - Add Edge(s)
2 - Delete Edge(s)
3 - Print Adjacency list and draw graph (generates a png image)
4 - Depth First Search (DFS)
5 - Detect existence of cycle
9 - Exit
3
DAG: 13 Vertices and 14 Edges
[0] => 5 1 6
[1] =>
[2] => 3
[3] =>
[4] => 2
[5] => 4
[6] => 4 9
[7] => 6
[8] => 7
[9] => 11 12 10
[10] =>
[11] => 12
[12] =>



Enter any of the following available choices
1 - Add Edge(s)
2 - Delete Edge(s)
3 - Print Adjacency list and draw graph (generates a png image)
4 - Depth First Search (DFS)
5 - Detect existence of cycle
9 - Exit
4
Enter the source node for DFS...0
DFS from 0 : 0, 5, 4, 2, 3, 1, 6, 9, 11, 12, 10,

Enter any of the following available choices
1 - Add Edge(s)
2 - Delete Edge(s)
3 - Print Adjacency list and draw graph (generates a png image)
4 - Depth First Search (DFS)
5 - Detect existence of cycle
9 - Exit
9

SOURCE CODE
dag.h

```
#include <iostream>
#include <list>
#include <vector>
#include "digraph.hpp"
using namespace std;

enum Color { WHITE, GREY, BLACK };

typedef stlplus::digraph<int,bool> IntBoolGraph; // int Node Type & bool arc type
typedef std::vector<IntBoolGraph::iterator> NodeVector ;
typedef std::vector<IntBoolGraph::arc_iterator> ArcVector ;


class Dag
{
    public:
        Dag(int vertices);
        ~Dag();

        int vertices() { return numberOfVertices; }
        int edges() { return numberOfEdges; }

        bool addEdge(int src, int dst);
        void addEdges();

        bool deleteEdge(int src, int dst);
        void deleteEdges();

        vector<int>& performDfsFromGivenSource();
        void dfs(int sourceVertex);

        bool doesCycleExist();
        bool visit(int vertex);

        void print(ostream &out);

    private:
        int numberOfEdges;
        int numberOfVertices;
        list<int>* adjList;
        char* colorList;
        vector<int> dfsOutputList;

        IntBoolGraph graph;
        NodeVector nodes;
        ArcVector arcs;

};
```

dag.cpp

```
#include "dag.h"
```

```
#include <fstream>
#include <cstdlib>
#include <string>
#include <sstream>

using namespace std;

Dag::Dag(int vertices)
{
    numberOfVertices = vertices;

    // Add nodes/vertices
    IntBoolGraph::iterator node;
    for (int i=0;i<numberOfVertices;i++)
        {
        IntBoolGraph::iterator node = graph.insert(i);
        nodes.push_back(node);
        }
    numberOfEdges    = 0;

    // adjList = new list<int>[numberOfVertices];
    colorList = new char[numberOfVertices];
}

Dag::~Dag()
{
    // delete adjList;
    delete colorList;
}

bool Dag::addEdge(int src, int dst)
{
    bool ret = false;
    if( src < numberOfVertices && dst < numberOfVertices)
        {
        IntBoolGraph::arc_iterator edge =
graph.arc_insert(nodes[src],nodes[dst],true);
        arcs.push_back(edge);
        numberOfEdges++;
        ret = true;
        }
    return ret;
}

bool Dag::deleteEdge(int src, int dst)
{
    bool ret = false;
    if( src < numberOfVertices && dst < numberOfVertices)
        {
        //locate and remove the edge
        IntBoolGraph::arc_iterator edge;
        IntBoolGraph::iterator from;
        IntBoolGraph::iterator to;
        for (int i=0;i<numberOfEdges; i++)
            {
```

```
                from = graph.arc_from(arcs[i]);
                to = graph.arc_to(arcs[i]);
                if (*from == src && *to == dst)
                {
                    graph.arc_erase(arcs[i]);
                    numberOfEdges--;
                    ret = true;
                    break;
                }
            }
        }
    return ret;
}

void Dag::print(ostream &out)
{
    static int dotfileNumber=0;
    dotfileNumber++;

    out << "DAG: " << numberOfVertices << " Vertices and " << numberOfEdges << "
Edges " << endl;

    ofstream dotfile;
    ostringstream dotfileName;
    dotfileName <<"dag" << dotfileNumber << ".dot";
    dotfile.open(dotfileName.str().c_str(),ofstream::out);
    dotfile << "digraph G {" << endl;

    for (int index = 0; index < numberOfVertices; index++)
        {
        out << "["<< index << "] => " ;

            for(int j = 0; j< graph.fanout(nodes[index]); j++)
            {
                IntBoolGraph::arc_iterator edge = graph.output(nodes[index],j);
                IntBoolGraph::iterator toNode = graph.arc_to(edge);
                out << *toNode << " " ;
                dotfile << "\t "<< index << " -> " << *toNode << ";" << endl;
            }
        cout << endl;
        }

    dotfile << "}" << endl;
    dotfile.close();

    ostringstream dagGenerationCommand;
    dagGenerationCommand << "dot -Tpng dag" << dotfileNumber << ".dot -o dag" <<
dotfileNumber << ".png";
    system(dagGenerationCommand.str().c_str());

    ostringstream imageOpeningCommand;
    imageOpeningCommand << "ristretto dag" << dotfileNumber << ".png" << "&";
    system(imageOpeningCommand.str().c_str());
}
```

```
void Dag::addEdges()
{

    int src, dst;
    cout << "Enter the edges (E.g. 4 3 means a directed edge from 4 to 3). -1 -1 as
input stops the process" << endl;
    do
        {
            cin >> src >> dst;

            if(src!=-1 && !addEdge(src,dst))
                cout <<"Failed to add this edge, (Try again with vertex numbers
between range (0,TotalVertices -1))"  << endl;

        }while(src!=-1);
}


void Dag::deleteEdges()
{

    int src, dst;
    cout << "Enter the edges (E.g. 4 3 means a directed edge from 4 to 3). -1 -1 as
input stops the process" << endl;
    do
        {
            cin >> src >> dst;

            if(src!=-1 && !deleteEdge(src,dst))
                cout <<"Failed to add this edge, (Try again with vertex numbers
between range (0,TotalVertices -1))"  << endl;

        }while(src!=-1);
}

vector<int>& Dag::performDfsFromGivenSource()
{
    int src;
    cout << "Enter the source node for DFS...";
    cin >> src;

    dfsOutputList.clear();
    for(int i=0;i<numberOfVertices;i++)
    {
        colorList[i] = WHITE;
    }
    dfs(src);
    cout << "DFS from " << src << " : ";

    for(int i=0;i<dfsOutputList.size();i++)
        cout << dfsOutputList[i] << ", ";

    cout << endl;
    return dfsOutputList;
}
```

```
void Dag::dfs(int sourceVertex)
{
    colorList[sourceVertex] = BLACK;
    dfsOutputList.push_back(sourceVertex);

    for(int j = 0; j< graph.fanout(nodes[sourceVertex]); j++)
        {
        IntBoolGraph::arc_iterator edge = graph.output(nodes[sourceVertex],j);
        IntBoolGraph::iterator toNode = graph.arc_to(edge);
        if ( colorList[*toNode] != BLACK )
            dfs(*toNode);
        }
    cout << endl;
}

bool Dag::doesCycleExist()
{
    NodeVector topo = graph.dag_sort();
    return topo.empty();
}
```

main.cpp

```
// build command
// g++ -g -Istlplus3-03-11/containers -Istlplus3-03-11/persistence -Istlplus3-03-
11/portability -Lstlplus3-03-11 -lstlplus3-03-11 dag.cpp main.cpp -o dag

#include <iostream>
#include <cstdlib>
#include <cstdio>
#include "dag.h"

#ifdef __linux__
    #define CLEAR_SCREEN system("clear")
#elif _WIN32
    #define CLEAR_SCREEN system("cls")
#endif

using namespace std;

int main()
{
    int input=0,vertices=0;

    CLEAR_SCREEN;

    cout << "\nComputational Lab - Assignment 4: Directed Acyclic Graph (DAG) with
STL-Plus Lib";
    cout << "\nBy: Pankaj Azad, Roll Number - 14M517, M.Tech Ist Sem\n\n\n" << endl;

    cout << "Enter number of vertices...";
    cin >> vertices;

    Dag dag(vertices);
```

```
    dag.addEdges();

    do
    {
        cout << "\n\n\n";
        cout << "Enter any of the following available choices" << endl;
        cout << "1 - Add Edge(s)" << endl;
        cout << "2 - Delete Edge(s)" << endl;
        cout << "3 - Print Adjacency list and draw graph (generates a png image)" <<
endl;
        cout << "4 - Depth First Search (DFS)" << endl;
        cout << "5 - Detect existence of cycle" << endl;
        cout << "9 - Exit" << endl;

        cin >> input;
        switch(input)
        {
            case 1:
                dag.addEdges();
                break;

            case 2:
                dag.deleteEdges();
                break;

            case 3:
                dag.print(cout);
                break;

            case 4:
                dag.performDfsFromGivenSource();
                break;

            case 5: cout << ( dag.doesCycleExist()? "\nCycle Exits!!\n": "\nNo Cycle
found!!\n") << endl;
                break;

            case 9: exit(0);
                break;
        }

    }while(input!=9);

    return 0;
}
```