International Journal of Management, IT & Engineering
Vol. 7 Issue 12, December 2017,
ISSN: 2249-0558 Impact Factor: 7.119
Journal Homepage: http://www.ijmra.us, Email: editorijmie@gmail.com
Double-Blind Peer Reviewed Refereed Open Access International Journal - Included in the International Serial Directories Indexed & Listed at: Ulrich's Periodicals Directory ©, U.S.A., Open J-Gage as well as in Cabell's Directories of Publishing Opportunities, U.S.A

LEXICOGRAPHICAL ORDER/SORTING USING BOOST LIBRARY IMPLEMENTATION

Jayant Kumar[*]

ABSTRACT:

Background:
the lexicographic or lexicographical order (also known as lexical order, dictionary order, alphabetical order or lexicographic(al) product) is a generalization of alphabetically ordered based on the alphabetical order of their component letters.

This generalization means that the order is not based on alphabetical order but based on relationship between two letters or entities.

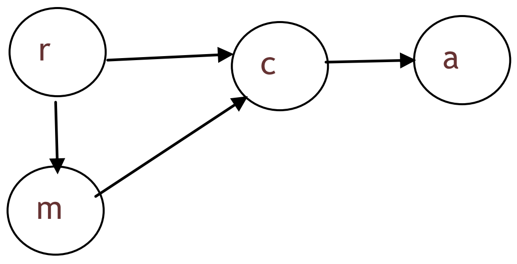
Example:
Given that r < c, c < a, r < m, m < c

So, the lexicographical order would be r,m,c,a

INTRODUCTION

Boost provides free peer-reviewed portable C++ source libraries. We can create an Adjacent graph by using boost adjacency list.

So, if we have relationship like r < c and further relationship to create graph as below.



We can apply topological sorting to get a ordered list.

Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge uv, vertex u comes before v in the ordering.

Since Boost is a C++ library, we will use C++ for implementation.

[*] Director of Platform Integration and Architecture, Bidtellect Inc., Delray Beach, USA - 33432

IMPLEMENTATION:

Let's implement the same by using a file which will provide relationship between letters.
The file is comprised of a sequence of words that are arranged in alphabetical order (for some arbitrary alternate alphabet)

File will have content like as below.

```
rcrtv
rcrmb
rcrdsfd
rcxrbw
rcxrbws
rcxrbwn
rcxrbkdz
rsqxbwarbw
rsqxbwarbws
rsqxbwarbwn
rsqxbwarbkdz
rsqxbwawa
rsqxbwawan
rsqxbwafqn
rsqxbwafqnxh
rjwdkbkwn
rjwdkbh
rjwdfaapwr
rjwaktkqj
rjkrcxw
rjktrcxw
rjktrcxh
rjks
rjksw
rjksnb
```

By reading each word one by one, we find the relationship between two letters and created a directed graph.
Once we are done reading all words, we will apply topological sort to get final output.

Implementation is divide into various steps.

STEP1:

Let's create a header file named "**LexcoSorting.h**" as below.
We will be declaring `labeled_graph` **`Graph`** which will be used to create graph while reading various words from give file and fnding relationship between two letters.
`vertex_iter` would be used to traversed through various vertex of the graph
`AddVertex` would be used to add new vertex if vertex doesn't exist in the graph
`AddEdge` would be used to add Edge between two give vertex

```
#ifndef LexcoSorting_H
                         #define LexcoSorting_H


                         #include "boost/graph/adjacency_list.hpp"
                         #include "boost/graph/labeled_graph.hpp"
                         #include "boost/graph/topological_sort.hpp"
                         #include <deque>
                         #include <iterator>
                         #include <iostream>
                         #include <fstream>
                         #include <string>




                         using namespace std;
                         using namespace boost;


                         class LexcoSorting
                         {
                             struct VertexProperty
                             {
                                 char c_literal;
                             };


                             typedef boost::labeled_graph<boost::adjacency_list< boost::vecS,
                         boost::vecS, boost::directedS,VertexProperty>,char> Graph;
                             typedef boost::adjacency_list<>::vertex_descriptor Vertex;
                             //typedef boost::labeled_graph<boost::setS,
                         boost::vecS,boost::directedS,VertexProperty> Graph;
                             typedef boost::graph_traits<Graph>::vertex_iterator vertex_iter;
                             typedef std::vector<Graph::vertex_descriptor>  Vcontainer;


                             Graph g;
                         //    list<Vertex> lVertex = new list<Vertex>(30);


                             void AddVertex(char c_temp);
                             void AddEdge(char sFirst,char sSecond);
                             void PrintOutPut();
public:
    string Compare(string sPrevious,string sCurrent);
    void Execute(string sInputFile);
};


#endif // LexcoSorting_H
```

STEP2:

Create Another file **LexcoSorting.cpp** with below code which will define all methods declared in **LexcoSorting.h**

AddVertex – Add a new vertex in the graph

PrintOutPut – Apply topological sort and prints the final sorted output

AddEdge – Add Edge between two letters passed as input

Compare – Compare two strings and find the relationship (order) between two letters by comparing two words as both words are in alphabetical order.

Execute: It's the main method which read word by word in the file and create Vertex if it doesn't exist or compare previous and current letters and create Edge between two if doesn't exist already.

Code:

```
#include "LexcoSorting.h"




void LexcoSorting::AddVertex(char c_temp)
{
    VertexProperty v1 ;
    v1.c_literal = c_temp;
  //Vertex vtemp = boost::add_vertex(c_temp,g);
  boost::add_vertex(c_temp,v1,g);
  //g[boost::add_vertex(c_temp,g)].c_literal = c_temp;
}


void LexcoSorting::AddEdge(char sFirst,char sSecond)
{
    boost::add_edge_by_label(sFirst, sSecond, g);
}
```

```
void LexcoSorting::PrintOutPut()
{
    Vcontainer c;
    Vcontainer::iterator ii;


    topological_sort(g.graph(), std::back_inserter(c));


    std::cout << "A topological ordering: ";
    for ( ii=c.begin(); ii!=c.end(); ++ii)
    {
        //cout << *ii << " ";
        //cout << *ii << " ";
        cout << g.graph()[*ii].c_literal << " ";
    }


}


string LexcoSorting:: Compare(string sPrevious,string sCurrent)
{
    string op;
    int cnt=0,i=0;
    while(sPrevious[i] !='\0' || sCurrent[i] !='\0')
    {
        if(sPrevious[i] == sCurrent[i]) {
                cnt = 1;
        }
        else {
            break;
        }
        i++;
     }
     if(cnt > 0) {
        op[0] = sPrevious[i];
        op[1] = sCurrent[i];
        op[3] = '\0';
    }
    return op;


}
```

```
void LexcoSorting::Execute(string sInputFile)
{
    /* //For Testing
    AddVertex('a');
    AddVertex('b');
    AddVertex('c');
    AddVertex('d');
    AddEdge('a','b');
    AddEdge('d','c');
    AddEdge('b','c');
    //AddEdge('a','b',*g);
    PrintOutPut();*/


    std::ifstream infile(sInputFile);
    std::string strcurrent,strprevious,strCompare;




    while (std::getline(infile, strcurrent))
    {
        if(!strcurrent.empty())
        {


          if(strprevious.empty())
          {
                for(char& c : strcurrent) {
                    AddVertex(c);
                }
          }
          else
          {
                strCompare = Compare(strprevious,strcurrent);
                if(!strCompare.empty())
                {
                    AddEdge(strCompare[0],strCompare[1]);
                }
                AddVertex(strCompare[0]);
                AddVertex(strCompare[1]);


          }
          strprevious = strcurrent;
          //cout << strcurrent;
          //file_contents.push_back('\n');
        }
```

```
                        }
                        PrintOutPut();


                    }
```

STEP3:

Edit/Create file main.cpp which would be the main file compiled and executed. It will take path of the input file which have alphabetical order of words.

Code:

```
#include <iostream>
                      #include "LexcoSorting.h"
                      #include <exception>


                      using namespace std;

                      int main()
                      {
                          try
                          {
                              LexcoSorting l;
                              cout << "Enter File Path and Name" << endl ;
                              char cfilename[200];
                              cin.getline(cfilename,sizeof(cfilename));
                              cout << "File Name : " << cfilename << endl;
                              l.Execute(cfilename);
                                    getchar();
                                       //"C:\\Users\\Jayant Kumar\\Documents\\alphabet.txt");
                          }
                          catch (const std::exception& e)
                          {
                              cout << e.what() << endl;
                          }


                          return 0;
                      }
```

STEP4:

Create a makefile with below code to compile the program which declare all dependencies.

Code:

```
# compiler:
              CC =    g++


              # compiler flags:
              CFLAGS  = -std=c++11 -g -Wall


              #Linking Flag
              LFLAGS =      -Wall


              INCLUDES =    -I C:/boost_1_59_0 -I C:/MinGW -I C:/boost_1_59_0/boost/graph
              LIBS =
              # the build target executable:
              TARGET = Lexicograph


              $(TARGET):    main.o LexcoSorting.o
                     $(CC) $(CFLAGS) -o $(TARGET) main.o LexcoSorting.o


              main.o:main.cpp LexcoSorting.h
                     $(CC) $(INCLUDES) $(CFLAGS) -o main.o -c main.cpp


              LexcoSorting.o:      LexcoSorting.cpp LexcoSorting.h
                     $(CC) $(INCLUDES) $(CFLAGS) -c LexcoSorting.cpp


              clean:
                     $(RM) $(TARGET) *.o *~
```

USE CASE AND IMPACT:

GENETIC SCIENCE:

It can be used to create genetic database as if we know relationship like FATHER -> SON, MOTHER-DAUGHTER we can get entire genetical order.

HEALTH CARE:

We can also create use the relationship between cause and symptom of a disease. That information can be used to order the symptoms of disease in a perfect order and we can keep track of hour health and discover any disease in early stages.

CRIMINOLOGY;

Same techinque can be applied to various crime cases and we can solve few complex criminal cases by ordering every aspect and story point of a crime.

REFERENCES

[1] https://en.wikipedia.org/wiki/Lexicographical_order
[2] http://www.boost.org/
[3] http://www.boost.org/doc/libs/master/libs/graph/doc/adjacency_list.html
[4] https://en.wikipedia.org/wiki/Glossary_of_graph_theory_terms#adjacent
[5] http://www.geeksforgeeks.org/topological-sorting/
[6]