Graph Manipulation Tool

Robert Bakarić
rbakaric@irb.hr
bakaric@evolbio.mpg.de
03.04.2015
GraphMan-1.0

Abstract

Graph manipulation tool is a program designed to utilize operations on graphs including vertex relocation, removal, addition and deletion. As an input the program takes two integer vectors one containing the list of successor vertices and the other, the list of predecessor ones. The program is written in C++.

Contents

1 Installation

The simplest way to compile this program is to:

1. Unpack the GraphMan package (`graphman-XXX.tar.gz`):

   ```
   tar -xvzf graphman-XXX.tar.gz
   ```

2. Change the current directory to `graphman-XXX`:

   ```
   cd graphman-XXX/
   ```

3. Configure the program for your system (–bindir is optional):

   ```
   ./configure --bindir=/absolute/directory/path/graphman-xxx/bin
   ```

4. Compile the program:

   ```
   make
   ```

5. Install the program:

   ```
   make install
   ```

Your binaries should be located in your local bin directory if `--bindir` option has been set. Otherwise installation needs to be carried out with root privileges in order to be installed into `/usr/local/bin` directory.

2 Input files

The GraphMan takes a simple two column (tab separated) file containing integers. First column contains successor (child) vertexes and the second column their predecessors (parent vertex). An example of the input file can be found in `./graphman-xxx/demo` and it should look like this:

`Graph`:

```
2     1
4     1
8     2
5     1
67    2
6     3
14    4
15    6
68    3
3     2
11    67
17    67
```

3 Program options

I order to see program options type:

```
./bin/GraphMan -h
```

Expected output:

```
Usage: ./program [options]

********************************************************************************
                          GraphMan - Graph Manipulation Tool
                                        by
                                   Robert Bakaric

CONTACT:
 Code written and maintained by Robert Bakaric,
 email: rbakaric@irb.hr , bakaric@evolbio.mpg.de

ACKNOWLEDGEMENT:


LICENSE:
 The program is distributed under the GNU General Public License. You should have
 received a copy of the licence together  with this software. If not, see
 http://www.gnu.org/licenses/
********************************************************************************


Allowed options:
  -h [ --help ]           produce help message
  -v [ --version ]        print version information
  -i [ --input-file ] arg input file
```

4 Functions and classes

Graph class :

PRIVATE :

`MakeGraph` : Function converts two given vectors into an appropriate data structure.

PUBLIC :

`make` : Explicit constructor.

`destroy` : Explicit destructor.

`DepthFirstTrav` : Given a starting vertex, function conducts a depth first search traversal. The result is a set of vertices as they are visited during the traversal and their distance (depth) values. Both can be retrieved through **GetDepthVec()** and **GetVerticesVec()** functions.

`Clean` : Function is used for erasing results computed with `Traverse` function

`GetVertexIdVec` : Function returns the computed vertices as they were visited during the traversal.

`GetDepthVec` : Function returns the computed distance values of vertices as they were visited during the traversal.

`AddVertex` : Given a vertex and a location, function creates an edge between the two.

`RemoveVertex` : Given a vertex function removes it together with all of its successor (child) vertices from the graph by relocating it to a new, null-routed one

`EraseVertex` : Given a vertex function removes and incoming edges and reatteches all outgoing ones to its parrent(s) vertex(ices)

`RelocateVertex` : Given two vertices function creates an edge between them and removes any existing ones between the relocated vertex and its previous parent vertex

`JoinVertex` : Given two vertices function creates an edge between them

`CreateVertex` : Given a graph, the function computes the next free label (number) for the vertex

`DmpGraph` : Function returns the graph in the same format as it received it (two vector format).

5 Example

5.1 GraphMan.cpp

A minimal example demonstrating the usage of GraphMan demo program:

```
./bin/GraphMan -i ./demo/Graph
Graph:
1 1 2 1  2 3  4  6  3 2 67 67 <- parent vertex
2 4 8 5 67 6 14 15 68 3 11 17 <- child vertex

Changes:
CreateVertex:69
CreateVertex:70
AddVertex(69,8)
AddVertex(70,8)
EraseVertex(4)
RemoveVertex(67)

Traversal:
1 2  3  3 2 3  4  3 1  1 <- distance(depth)
2 8 69 70 3 6 15 68 5 14 <- vertex labels

New Graph:
 0 67 67 1 1  1 2 2 3  3  4  6  8  8 <- parent vertex
67 11 17 2 5 14 8 3 6 68 14 15 69 70 <- child vertex
```

5.2 GraphMan.hpp

Adding the `GraphMan.hpp` header file to your program will allow you to include all the functions described in section 4. A minimal example:

```
    #include<vector>
    #include<GraphMan.hpp>

    vector<int|long|unsigned|double> parent {1,1,2,1,2,3,4,6,3,2,67,67};
    vector<int|long|unsigned|double> child  {2,4,8,5,67,6,14,15,68,3,11,17};

/* Make Graph */

    /* Construction */
    Graph<int|long|unsigned|double> graph(parent,child);
    /*  OR  */
    Graph<int|long|unsigned|double> graph;
    graph.make(parent,child);
```

```
/*  Functions */

    graph.AddVertex(CreateVertex(),8)  // creates a new vertex and adds it to vertex 8
                                       //    result: 8 -- 69 edge

    graph.EraseVertex(4);  // it erases vertex 4 and makes an edge between 14 and 1

    graph.RemoveVertex(67);// vertex 67 and all its child nodes are relocated to graph
                           // rotted at 0
                                       //    result: 0 -- 67 +- 11
                                                            +- 17

    graph.RelocateVertex(5,6)          // assigns vertex 5 to vertex 6
                                             //    result:   1 +- 4 -- 14
                                                              +- 2 +- 8
                                                                   +- 3 +- 68
                                                                        +- 6 +- 15
                                                                             +- 5
    vector<int|long|unsigned|double> p-dash;
    vector<int|long|unsigned|double> c-dash;

    graph.DmpGraph(p-dash,c-dash);      // creates two vectors
                 //results: p-dash:  0 67 67 1  1 2 2 3  3  4  6 6  8  8
                 //         c-dash: 67 11 17 2 14 8 3 6 68 14 15 5 69 70

    graph.GetDepthVec()      // returns the depth vector
                             //    result for 2: 1 2  3  3 2 3  4 1  3  1

    graph.GetVertexIdVec()   // returns the set of vertices (vector)
                             //    result for 2: 2 8 69 70 3 6 15 5 68 14
```

6 Acknowledgement

7 Future work

1. Implement the breadth first search traversal.
2. Error management!