**Algorithms**

Programming Assignment #3

Graph: DFS/BFS/Dijkstra

**Introduction:**

Our algorithms class wants to produce a commercial software to solve several important graph problems: *graphlab ®*. As the chief programmer of this software, you are asked to implement DFS/BFS to traverse an undirected graph. Also, we need to implement Dijkstra’s algorithms to find out the shortest path from the labeled vertex to root. To be user friendly, graphlab has a nice command line interface. To understand this interface, please run the command manager tutorial first.

**Input Files:**

In this PA, we use the *DOT[[1]](#footnote-1)* language to describe our graph. Below is an example undirected graph (gn4.dot) that has four vertices (*v*0, *v*1, *v*2, *v*3) and 5 edges. The first line ‘//’ is a comment. The second line starts with a keyword ‘*graph*’, followed by its name ‘*gn4*’. The description of this graph is enclosed in a pair of curly brackets {}. Each line between the curly brackets represents an edge. For example, ‘*v*0--*v*1’ represents an edge between vertices *v0* and *v1* and *etc.* In the square bracket, the edge weight is described as a label, which will be shown in the figure. For example, the edge between edge *v*0--*v*1 is two. The order of the edges does not matter.

|  |
| --- |
| // THIS IS COMMENT: example undirected graph with edge weight  **graph** gn4 **{**  v0 -- v1 **[label = "**2**"];** // edge weight is shown as text label  v1 -- v2 **[label = "**5**"];**  v0 -- v3 **[label = "**4**"]**;  v2 -- v3 **[label = "**3"**];**  v1 -- v3 **[label = "**1"**];**  **}**  // from v0 to v3 |

Figure 1. Original Graph

Last line comes for an argument, which tells you that we want to find the shortest path from which source node to which end node. For example, in Fig. 1, we want to use Dijkstra’s algorithm to find the shortest path from *v0* to *v*.

We have a nice tool available in Linux to render this graph into pictures in many formats, like ps or jpg or png. In the linux command line, please type:

dot -Tpng inputs/gn4.dot -o outputs/gn4.png

This generates an *outputs/gn4.png* file that shows graph *gn4*. To view this *png* file, you can type the command, which brings up an x-window to show graph *G*.

display outputs/gn4.png &

**Output File Format:**

1. For a DFS/BFS, when there are multiple vertices to choose, you should always choose the vertex with minimal vertex index. For example, Figure 2 shows the example output of the command:

write\_tree\_dfs –s v0 –o outputs/gn4\_dfs.dot

Starting from the source vertex *v0*, the first edge is *v0*--*v1* and the second edge is *v0*--*v3*. No point is given if these two edges are swapped. (see *gn4\_bfs.dot*)

|  |
| --- |
| // BFS tree produced by graphlab  graph gn4\_bfs {  v0 -- v1 [label = "2"];  v0 -- v3 [label = "4"];  v1 -- v2 [label = "5"];  }  // vertices = 4  // edges = 3  // source = v0  // runtime = 0.1 sec  // memory = 1.0 MB |

Figure 2 BFS Tree of Fig.1

2. When you write an edge, please always put the predecessor vertex in front of the successor vertex. For example, in Figure 2, *v*0--*v*1 is correct, but *v*1--*v*0 is incorrect. No point is given if these two vertices are swapped.

3. Please print the edges in the order of traversal. For example, the BFS tree in Figure 2, please print *v*0--*v*1 before *v*0--*v*3; print *v0*--*v3* before *v1*--*v2.* No point is given if the edges are swapped.

You should refer to the golden sample output files in the *outputs/* directory: *gn4\_bfs.dot*, *gn4\_dfs.dot,* and *gn4\_dijkstra.dot* files. Please write your own .dot output file in the same format. If your output format mismatches the golden output files, you may lose some scores.

4. Suppose we use the Dijkstra‘s algorithm to generate an output file (gn4\_dijkstra.dot) in the same DOT format.

Figure 3 shows an example output for this commend:

write\_dijkstra –o outputs/gn4\_dijkstra.dot –from v3 –to v0

The new graph *gn4\_dijkstra* shows the shortest path from source *v*3 to end *v*0. You should print the edges in the order from source to end node. For example, the answer for our input file is the following path: v3 to v1, v1 to v0. Be careful, you should print the edge in the order from source node to end node. For example, *v*3--*v*1 and *v*1--*v*0 is the correct order, but *v*3--*v*1 and *v*0--*v*1 is the wrong order.

At the end, we also show the corresponding information, including the number of vertices, edges, source node, end node, total\_weight in the comments. Total\_weight means the summation of the edges weights on the shortest path. The last two lines show the runtime (user) in seconds and memory usage (peak) in MB. Note: Please show your results (red parts) in correct format because our automatic grading tool will check these numbers.

|  |
| --- |
| // Dijkstra produced by graphlab  graph gn4\_dijkstra {  v3 – v1 [label = “1”];  v1 –– v0 [label = “2”];  }  // vertices = 3  // edges = 2  // source = v3  // end = v0  // total\_weight = 3  // runtime = 0.1 sec  // memory = 1.0 MB |

Figure 3. Dijkstra Shortest Path

5. . To simplify your work, you can assume that the file name and the graph name are always in the format of ‘*gn’* followed by the total number of vertices |*V*|. The vertex names are always in the format of ‘*vx*’, where *x* is an integer number from 0 to |*V*|-1.

**Commands to be implement:**

In the graphlab tool, you need to support the following system commands. The following commands have been already implemented in the cmdmgrTutorial.

|  |  |
| --- | --- |
| **System Commands** | **Description** |
| **ls** | List directory contents |
| **exit** or **quit** | Exit graphlab |
| **help** | Show all commands |
| **source** <batch\_filename> | Execute commands in batch mode |
| **dot** –Tpng <dot\_filename> –o <png\_filename> | Convert dot file into png file |
| **display** <png\_filename> | Show picture in X-window |

You need to implement the following user commands.

|  |  |
| --- | --- |
| **Commands** | **Description** |
| **read\_graph** <dot\_filename> | Read the graph in dot format |
| **write\_tree\_dfs** –s <sourcenode> –o <dot\_filename> | Perform depth first search starting from source node. Then write to a dot file. |
| **write\_tree\_bfs** –s <sourcenode> –o <dot\_filename> | Perform breadth first search starting from source node. Then write to a dot file. |
| **write\_dijkstra** –o <dot\_filename> -from <sourcenode> -to <endnode> | Perform Dijkstra’s algorithm to find the shortest path from sourcenode to endnode. Then write to a dot file. |

To simplify our problem, we assume that the given graphs for BFS/DFS/Dijkstra are always undirected and fully connected (there must be a path from a node to every other node). All weights are positive integers.

Your graphlab will be tested in *batch mode* (that means, many command in a single file). An example command file (*gn4.bat)* is provided in the input directory. Test your program using this command:

bin/graphlab –f inputs/gn4.bat

|  |
| --- |
| read\_graph inputs/gn4.dot  dot –Tpng inputs/gn4.dot –o outputs/gn4.png  display outputs/gn4.png  write\_tree\_dfs –s v0 –o outputs/gn4\_dfs.dot  dot –Tpng outputs/gn4\_dfs.dot –o outputs/gn4\_dfs.png  display outputs/gn4\_dfs.png  write\_tree\_bfs –s v0 –o outputs/gn4\_bfs.dot  write\_dijkstra –o outputs/gn4\_dijkstra.dot –from v3 –to v0  exit |

**Requirements:**

1. Your source code must be written in C or C++. The code must be executable on EDA union lab machines.

2. In your report, compare the results, running time, and memories different input sizes. Please fill in the following table and also plot figures showing the memory and running time. Please use –O2 optimization and turn off all debugging message.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | DFS | | BFS | | Dijkstra | | |
| input file | input size  vertices/edges | runtime  (s) | memory (MB) | runtime  (s) | memory (MB) | mini-weight | runtime  (s) | memory (MB) |
| gn4 | 4/5 |  |  |  |  |  |  |  |
| gn8 |  |  |  |  |  |  |  |  |
| gn10 |  |  |  |  |  |  |  |  |
| gn100 |  |  |  |  |  |  |  |  |
| gn1000 |  |  |  |  |  |  |  |  |
| gn10000 |  |  |  |  |  |  |  |  |

Please draw runtime of all cases of BFS in a figure. Draw runtime of all cases of DFS in another figure. Draw run time of all cases of Dijkstra in a figure. Do not need to draw memory.

**Submission:**

Please submit a hardcopy of your report in class. Also, please submit a single *\*.tgz* file to CEIBA system. Your submission must contain:

1. your source codes and a make file. By simply typing ‘make’, an executable binary is produced under the *bin* directory. Please use ‘-O2’ optimization. Please turn off all debugging messages.

2. a README file that explains your files.

3. a report in the *doc* directory. Please also submit a printed report in class.

4. We will use our own test cases so do NOT include the *inputs* and *outputs* directory.

The submission filename should be compressed in a single file <student\_id>\_<pa3>.tgz. (e.g. b90901000\_pa3.tgz). If you have a modified version, please add \_v2 as a postfix to the filename and resubmit it (e.g. b9090100 0\_pa3\_v2.tgz). You can use the following command to compress a whole directory:

tar zcvf <filename>.tgz <dir>

**checkSubmitPA3:**

You must use the checksubmitPA3 script, which is under the /*utility* directory, to check whether your result is correct or not. Please put your *.tgz* file in the */utility* directory. To use this checker, simply type

cd utility

./checkSubmitPA3.sh <filename>.tgz

For example:

./checkSubmitPA3.sh b99901000\_pa3.tgz

This checker will check: 1 decompression, 2. Compilation. Please check the correctness of your answers with other students.

We have so many students in the class so we need automatic grading. Any mistake in the submission will result in cost 20% off your score. Please be very careful in your submission.

**Grading:**

60% correctness

20% file format and location

20% report

**Bonus :**

5% bonus will be given to very good features that you add to make your program faster or smaller memory. Please write clearly in your report.

**NOTE:**

TA will check your source code carefully. Copying other source code can result in zero grade for all students involved.

**Reference:**

Details about DOT can be found in the Wikipedia

http://en.wikipedia.org/wiki/DOT\_language

1. More details about DOT can be found in the Wikipedia (<http://en.wikipedia.org/wiki/DOT_language>). [↑](#footnote-ref-1)