Assignment 05
Graph Data Structure and Algorithms

DS286.Aug16 Data Structures and Programming

October 28, 2016

Submission is due on or before **Monday, 14 November, 2016, 11:59pm IST**.
The assignment carries **100 points**, which is 10% of the course weightage.

1 Question

This assignment requires you to design a graph data structure based on an adjacency list and perform graph algorithms on that for the given input datasets. You will learn to design an abstract data type (ADT) from the beginning and reuse *C++ Standard Template Library (STL)* in implementing this Graph ADT.

Your tasks are as follows:

1. Design an abstract data type `Graph` to represent a data structure for an *undirected graph*. Vertices of the graph are uniquely identified using an *integer* ID field, and a templatized *value* field. The edges of the graph should have a *value* field that is templatized to accept any class, and the edge is uniquely identified by the two vertex IDs it is incident upon. The graph ADT should allow you to add, remove and list vertices and edges. The edges incident on a vertex should be maintained as an adjacency list. The ADT should allow you to traverse to incident edges and adjacent vertices from a given vertex. You should use C++ STL where possible.

   The header file for the graph ADT should be placed in `graph.h` and its implementation placed in `graph.cpp`. These files should only contain the signature and implementation of the graph ADT, respectively, and should be well-documented.

2. Using the graph ADT you have defined above, implement the following graph algorithms and place them in the file `algos.cpp` with a corresponding header file `algos.h`.

   (a) *Depth First Search (DFS)*: The method `dfs(Graph g, int vid)` should be implemented to traverse the graph g in a depth-first manner from the source vertex identified by *vid*. At each step of the DFS,

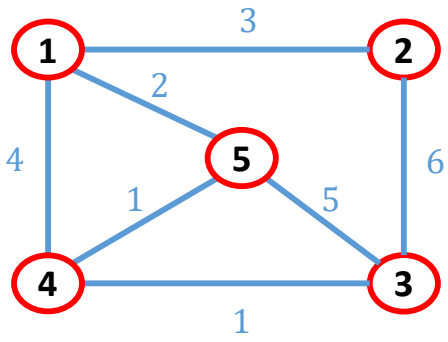

Figure 1: Sample Graph

you should pick the outgoing edge to *the adjacent non-visited vertex with the smallest ID* to traverse on.

The DFS traversal should print the IDs of the vertices in the order traversed, on a single line separated by a single space between the IDs, and prefixed with the string `DFS`. E.g. the DFS traversal of the graph in Fig. 1 from a source vertex ID `5` should produce the line:
`DFS 5 1 2 3 4`
terminated by a newline.

(b) *Single Source Shortest Path (SSSP)*: The method `sssp(Graph g, int vid)` should take the identifier *vid* of a source vertex and implement Dijkstra's shortest path algorithm to find the shortest distances from the source vertex to every vertex in the graph g, including the source. Assume that the edge weights are integers for the templated `edge` class.

For each vertex in the connected component that the source vertex is present in, the SSSP method call should print one line of output with the `vid distance`, preceded by a separate line having the string `SSSP`, e.g., running SSSP on the graph in Fig. 1 from a source vertex ID `5` should produce:

```
SSSP
5 0
4 1
1 2
3 2
2 5
```

The order of the lines does not matter.

(c) *K-means Clustering:* The method `kmeans(Graph g, int k, int maxcuts)` should take the number of clusters k to be identified in the graph g, and the maximum of the sum of the undirected edge cuts

that are allowed between clusters, *maxcuts*. The algorithm should pick k vertices as centers at random and color neighbors using a uniform BFS traversal initiated from each center until all vertices in the connected component are colored with one of the centers. Ignore the weights of the edges during traversal. Sum the number of edge cuts between vertices with different colors, and if strictly greater than the *maxcuts* threshold, repeat the process with a new set of k random center vertices. Else, if less than equal to the threshold, stop and print the clusters as output.

Print each cluster in a single line as a list of vertex IDs separated by space followed by the number of cuts in a separate line. These should be preceded by a separate line having the string `KMEANS`. E.g., with $k = 3$ for a given graph having 10 vertices, you will print three lines having the vertex IDs of each cluster as follows, and a separate line with the sum of the number of edge cuts between clusters, 4:

```
KMEANS
3 4 6 2
1 7 9
5 8
4
```

3. The `main()` method should be present in the `main.cpp` file, and accept as input the parameters listed below. The method should call the DFS, SSSP and K-Means clustering methods in that specific order with the input parameters and print the results exactly as listed above. *There should be no extra newline or whitespace or debugging output that is printed.* You should write the `Makefile` yourself to produce an executable named `GraphAlgos.out`.

   ```
   GraphAlgos.out <graph_file> <dsp_src> <sssp_src> <k> <maxcuts>
   ```

   where:
   `<graph_file>` : Name of file containing the *input graph* as an edge list. Each line of the file contains the triple for an edge, $\langle u\ v\ w \rangle$, where u and v are vertex IDs that the edge is incident upon, and w is the weight of the edge. Each undirected edge appears only once in the file. For e.g., the input file for the graph in Fig. 1 would be:

   ```
   1 2 3
   1 5 2
   1 4 4
   2 3 6
   3 4 1
   3 5 5
   4 5 1
   ```

`<dsp_src>` : The *source vertex ID* for the DFS traversal method.
`<sssp_src>` : The *source vertex ID* for the SSSP method.
`<k>` : The *number of clusters* for the K-Means clustering method.
`<maxcuts>` : The *maximum cuts* threshold for the K-Means clustering method.

For e.g., the following is a sample invocation of your executable:

```
GraphAlgos.out small.in 5 5 3 4
```

You will separately be given three graph input files `small.in, medium.in` and `large.in`, along with the input parameters for each. You should ensure that your application runs successfully for each of these inputs and generates the correct output. You should include the output from each of these runs in three text files `small.out, medium.out` and `large.out`.

Separately, we will test your application against additional input graphs as well and your evaluation will be based upon these.

2 Submission Instruction

Please follow these instructions carefully. We use automated scripts for evaluation. So a failure to follow these instructions will mean that your submission will not be evaluated.

- You are not provided with any skeleton code this time. You will need to write all code by yourself! Refer to the skeleton code given for earlier assignments as a sample.
- Only write your code in these following source and header files: `graph.h, graph.cpp, algos.h, algos.cpp` and `main.cpp`.
- Include the `Makefile` and compiled executable file `GraphAlgos.out` in your submission.
- For each of the three sample inputs and parameters, submit a text file containing the outputs from these executions in files named `small.out, medium.out` and `large.out`. These files should only contain the output printed from your application and not the commandline itself.
- Place all these files in a single folder whose name is determined as follows. My full name is "Prateeksha Varshney" where "Prateeksha" is my first name, so the folder name should be `05Prateeksha` for my submission. Please note the capitalization of first letter of the first name. The final contents of the folder would be as follows:

```
05Prateeksha
|- main.cpp
|- graph.h
|- graph.cpp
|- algos.h
|- algos.cpp
|- GraphAlgos.out
|- Makefile
|- small.out
|- medium.out
|- large.out
```

- This folder should be compressed using the `tar` program as follows:
  `tar -cvf 05Prateeksha.tar 05Prateeksha/`
  Note: Any other compression format *will not be accepted* and will be treated as no submission. Its your responsibility to check if the file can be properly uncompressed and all files inside are intact.
- Send a separate mail to the TA's email address `prateeksha@grads.cds.iisc.ac.in` with the subject line `DS286.Aug16.A05`. Do not write anything more or less in the subject line. Do add any text in the body. Do not send the assignment as a reply-email to any other mail.
- **Only one submission will be accepted**. If multiple emails or files are received, **only the first one** will be taken as the submission. Only the submission received **before the deadline** will be accepted.
- Use only the C++ language for completing this assignment. Make sure the code compiles and executes correctly on the head node of the `turing.cds.iisc.ac.in` server using `g++` command. You will need to pass the `-std=c++11` flag to the compiler to use STLs. The code will be compiled and tested on this machine during evaluation.
- Indent/format the code and add inline comments describing that the code is supposed to do. This will help you debug better, and give us an insight on the logic you are using.
- It is your responsibility to remove all debug statements you may have added during development and testing your code. The evaluation of your submission is done using automated scripts, and your console output will be tested against a predetermined correct output. If the outputs do not match exactly, it will be taken as wrong output.

3 Ethics

You should not get assistance from other students or external sources in directly solving the assignment. Getting help on generic C++ and data structures concepts, e.g., on using lists, strings, libraries, compilation, etc. is accepted. You

are encouraged to post questions to the course mailing list so that the TA, instructor or other students can respond. This also ensures you do not have an unfair advantage/disadvantage over other students. If you have received assistance from other sources, send a *separate email to the Instructor and the TA* disclosing the external sources and type of support received.

By making a submission, you are asserting that all code that you submit was designed and developed by you. Do NOT copy and paste code from anyone else! All code will be verified using a plagiarism checker, and *penalties will be imposed* if plagiarism is found from unattributed sources.