README: MAX FLOW GRAPH ALGORITHMS

Above code contains sequential and parallel implementations of three graph algorithms which are,

1.) Edmonds Karp Algorithm

2.) Dinic's Algorithm

3.) Push Relabel Algorithm

HOW TO GENERATE GRAPHS:

1. Your own graph:

   (Note vertex number starts from 1)

- Open a blank text file.
- Format:
  - First line contains two space separated integers, number of vertices (n) and number of edges (m) respectively.
  - Second line contains two space separated integers, source vertex and sink vertex respectively.
  - Following m lines contains three space separated integers of the form: u v c denoting capacity 'c' of the edge from vertex 'u' to vertex 'v'.
- Save the file as "sample.txt"

2. How to generate some random graph:

   We have provided a graph generator file (graph_generator.cu) that can automatically generate graphs up to 10,000 vertices and 2*10000C2 directed edges but the number of edges and vertices can be set manually also.

   Default Setting: Random number of vertices (max 10,000) and edges will be generated with maximum capacity possible for an edge as 10.

   Manual Setting:

- Just open the graph_generator.cu file.

- Uncomment line number 25 and set the `nvertices` as whatever number of vertices you want.
    - Line number 25: `// nvertices = 5000;`
    - Uncomment this and set your value.

- Comment the line number 24.
    - Line number 24: `nvertices = (rand () % 10000) + 1;`
    - Comment this line.

- Uncomment line number 31 and set the `nedges` as whatever number of edges you want.
    - Line number 31: `// nedges = 10000;`
    - Uncomment this and set your value, but make sure (nedges <= nvertices * (nvertices - 1))

- Comment line number 30.
    - Line number 30: `nedges = rand () % ((nvertices*(nvertices-1)));`
    - Comment this line.

- Go to line number 35 and set the maximum capacity possible for an edge.
    - Line number 35: `maxcapacity = 10;`
    - Set your value, but make sure maxcapacity is in the range of short (not so large), so that overflow does not occur for large number of edges.

- Run the file and graph will be generated by file name “sample.txt”.

HOW TO RUN ALGORITHMS:

- By default, only parallel implementation of Push-Relabel is on (because number of vertices and number of edges can take large values in random graph generation, so other algorithm might take longer time to give output).

- If you want to run other algorithms such as parallel Edmond Karps or Dinic or sequential versions of them then,

    - Open the “main.cu” file.

- For Sequential Edmond-Karps: Uncomment the line number 1196 – 1204
- For Parallel Edmond-Karps: Uncomment the line number: 1210 – 1218
- For Sequential Dinic: Uncomment the line number: 1223 – 1231
- For Parallel Dinic: Uncomment the line number: 1237 – 1245
- For Sequential Push-Relabel: Uncomment the line number: 1250 – 1258

  We have uncommented as multiline comment, just remove them.

- Run the "main.py "from command line as:

  ```
  ./a sample.txt output.txt
  ```
  [For Windows users]

  ```
  ./a.out sample.txt output.txt
  ```
  [For Linux users]

  where "sample.txt "is the input file name (graph file) and "output.txt" is the output file name (file in which final maxflow and flow between edges will be printed in the form of (vertices * vertices) size adjacency matrix.

OTHER ADDITIONAL INFORMATION:

When we tried to run it on our GPU (GeForce GTX 1050, 3GB Memory), we could only allocate 10,000 vertices. On Google Colab, we were able to run 20,000 vertices graph.

So, if your PC has sufficient memory, then you can try to run graph with more vertices. If you want to generate graph of more than 10,000 vertices, then you have to make the following change in the "graph_generator.cu" file apart from the changes mentioned in manual settings above.

Line number 8: `int graph [10001] [10001];`

If you want to make n vertex graph, set above line as: `int graph [n + 1][n + 1] ;`