GHS Algorithm

0.1

Generated by Doxygen 1.9.1

Chapter 1

This is the documentation for the implemented code of GHS Algorithm.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 GHSNode Class Reference

Public Member Functions

- Graph< int, int > ∗ **run** ()
- **GHSNode** (int nid, std::unordered_map< int, int > neighbors)

4.1.1 Detailed Description

Definition at line 52 of file GHSNode.h.

The documentation for this class was generated from the following files:

- GHSNode.h
- GHSNode.cpp

4.2 Graph< T, U > Class Template Reference

Stores Undirected Weighted Graphs. Provides Undirected Weighted Graph ADT and provides some graph probabilities.

```
#include <Graph.h>
```

Public Member Functions

- bool **Equal** (Graph< T, U > ∗obj)
- Graph (int n, int m, std::vector< std::tuple< T, T, U > > weights_labels)

    Graph Constructor to take in the graph in given format.
- std::set< std::tuple< U, T, T > > **GetEdgeSet** ()
- void DrawGraph (std::ofstream &ofs)

    Puts the graph into ofs file.
- void PrintGraph ()

    Prints The various data structures of the graph.
- void PrintOutput ()

    Prints The output as requested.
- bool IsConnected ()

    Checks If the graph is connected.
- Graph< T, U > ∗ MST_Kruskal ()

    Gives the MST for the given graph.

4.2.1 Detailed Description

template<typename T, typename U>
class Graph< T, U >

Definition at line 36 of file Graph.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Graph()

```
template<typename T , typename U >
Graph< T, U >::Graph (
            int n,
            int m,
            std::vector< std::tuple< T, T, U > > weight_labels )
```

Graph constructor for initializing graphs.

Parameters

n	Number of Nodes
m	Number of Edges @para weight_labels Edges in form of tuple vector

Definition at line 60 of file Graph.cpp.

4.2.3 Member Function Documentation

4.2.3.1 DrawGraph()

```
template<typename T , typename U >
void Graph< T, U >::DrawGraph (
            std::ofstream & ofs )
```

Makes the .dot files for Graphviz library.

Parameters

ofs	Output .dot file

Definition at line 129 of file Graph.cpp.

4.2.3.2 IsConnected()

```
template<typename T , typename U >
bool Graph< T, U >::IsConnected
```

Checks if the graph is connected.

Definition at line 207 of file Graph.cpp.

4.2.3.3 MST_Kruskal()

```
template<typename T , typename U >
Graph< T, U > * Graph< T, U >::MST_Kruskal
```

Returns the Minimum Spanning Tree for the current graph.

Definition at line 233 of file Graph.cpp.

4.2.3.4 PrintGraph()

```
template<typename T , typename U >
void Graph< T, U >::PrintGraph
```

Prints various graph Data Structures.

Definition at line 139 of file Graph.cpp.

4.2.3.5 PrintOutput()

```
template<typename T , typename U >
void Graph< T, U >::PrintOutput
```

Prints Graph in the output format specified.

Definition at line 195 of file Graph.cpp.

The documentation for this class was generated from the following files:

- Graph.h
- Graph.cpp

4.3 GraphException Class Reference

Public Member Functions

- GraphException ()

  Generic Graph Exceptions.
- GraphException (int code)

  Specific Graph Exceptions.

4.3.1 Detailed Description

Definition at line 24 of file Graph.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 GraphException()

```
GraphException::GraphException (
            int code )
```

Parameters

code	error code for the graph

Definition at line 20 of file Graph.cpp.

The documentation for this class was generated from the following files:

- Graph.h
- Graph.cpp

4.4 GraphVz< T, U > Class Template Reference

The class plots the graph.

```
#include <dot_graph.h>
```

Public Member Functions

- GraphVz (std::ofstream &ofs, const std::vector< std::pair< T, T >> &edges, const std::vector< U > &labels, T root, bool has_labels=false, bool is_directed=false)

  Constructor for taking in parameters of the graph and file.

4.4.1 Detailed Description

template<typename T, typename U>
class GraphVz< T, U >

Parameters

ofs	File Stream to write the graph dotfile into
edges	List of edges of the graph
labels	Weights of the corresponding edges
root	Root of the graph
has_labels	Flag to check whether graph is weighted
is_directed	Flag to check whether the graph is directed.

Definition at line 20 of file dot_graph.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 GraphVz()

```
template<typename T , typename U >
GraphVz< T, U >::GraphVz (
            std::ofstream & ofs,
            const std::vector< std::pair< T, T >> & edges,
            const std::vector< U > & labels,
            T root,
            bool has_labels = false,
            bool is_directed = false )
```

Parameters

ofs	File Stream to write the graph dotfile into
edges	List of edges of the graph
labels	Weights of the corresponding edges
root	Root of the graph
has_labels	Flag to check whether graph is weighted
is_directed	Flag to check whether the graph is directed.

Definition at line 20 of file dot_graph.cpp.

The documentation for this class was generated from the following files:

- dot_graph.h
- dot_graph.cpp

4.5 hash_pair Struct Reference

Provides Hashing for pair. Gives a Hash of Two objects of arbitrary type by using XOR.

```
#include <Graph.h>
```

Public Member Functions

- template<class T1 , class T2 >
  size_t **operator()** (const std::pair< T1, T2 > &p) const

4.5.1 Detailed Description

Definition at line 14 of file Graph.h.

The documentation for this struct was generated from the following file:

- Graph.h

4.6 Message Struct Reference

Public Member Functions

- **Message** (std::vector< std::string > m)

Public Attributes

- std::vector< std::string > **msg**

4.6.1 Detailed Description

Definition at line 9 of file GHSNode.h.

The documentation for this struct was generated from the following file:

- GHSNode.h

4.7 Network Struct Reference

Public Attributes

- std::unordered_map< int, Queue > **msg_queues**

4.7.1 Detailed Description

Definition at line 47 of file GHSNode.h.

The documentation for this struct was generated from the following file:

- GHSNode.h

4.8 Queue Struct Reference

Public Member Functions

- void **push** (Message m)
- Message **top** ()
- Message **pop** ()

Public Attributes

- std::mutex **mut**
- std::queue< Message > **q**

4.8.1 Detailed Description

Definition at line 20 of file GHSNode.h.

The documentation for this struct was generated from the following file:

- GHSNode.h

Chapter 5

File Documentation

5.1 dot_graph.cpp File Reference

Uses GraphViz Library to plot Graphs.

```
#include <bits/stdc++.h>
#include "dot_graph.h"
#include "Graph.h"
```

5.2 dot_graph.h File Reference

```
#include <bits/stdc++.h>
```

Classes

- class GraphVz< T, U >

  The class plots the graph.

5.2.1 Detailed Description

Header file for dot_graph

5.3 Graph.cpp File Reference

```
#include <bits/stdc++.h>
#include "dot_graph.h"
#include "Graph.h"
```

5.3.1 Detailed Description

Provides Implementation of the Graph Class.

5.4 Graph.h File Reference

```
#include <bits/stdc++.h>
#include "dot_graph.h"
```

Classes

- struct hash_pair

  Provides Hashing for pair. Gives a Hash of Two objects of arbitrary type by using XOR.
- class GraphException
- class Graph< T, U >

  Stores Undirected Weighted Graphs. Provides Undirected Weighted Graph ADT and provides some graph probabilities.

5.4.1 Detailed Description

Provides Signature for the Graph Class.

5.5 input_generator.cpp File Reference

Generates a connected input graph for given number of nodes and probability of an edge between any two nodes.

```
#include <bits/stdc++.h>
```

Macros

- #define **MAX_NODES** 400
- #define **PRECISION** 1000000
- #define **MAX_WEIGHT** 50000000

Functions

- bool checkinputs (int N, double p)

  Checks validity of given inputs.
- void DFS (int node, int color_val, std::vector< std::set< int > > &adj_list, std::vector< int > &color, std↩ ::unordered_map< int, int > &colormap)

  Does DFS on the graph starting from a node.
- void DFS_Util (int N, std::set< int > &edge_weights, std::vector< std::tuple< int, int, int > > &edges, std↩ ::vector< std::set< int > > &adj_list, std::vector< int > &color, std::unordered_map< int, int > &colormap)

  Uses DFS to make the graph connected.
- int **main** ()

5.5.1 Detailed Description

Date

8/4/2021

Version

0.1

Author

Dhananjay Kajla

Vijay Meena

5.5.2 Function Documentation

5.5.2.1 checkinputs()

```
bool checkinputs (
             int N,
             double p )
```

Parameters

N	Total number of vertices
p	probability of an edge between two vertices Total number of vertices(N) should be less than MAX_NODES Probaility(p) should be between 0 and 1

Definition at line 24 of file input_generator.cpp.

5.5.2.2 DFS()

```
void DFS (
             int node,
             int color_val,
             std::vector< std::set< int > > & adj_list,
             std::vector< int > & color,
             std::unordered_map< int, int > & colormap )
```

DFS runs a dfs and colors the nodes into connected components recursively. At the end we have all nodes connected to the current node colored with the same color(color_val).

Parameters

node	index of the current node
color_val	color of the connected component of which node is a part
adj_list	adjacency set of the graph
color	color of connected components of various nodes
colormap	a map from color to one of its representative node

Definition at line 50 of file input_generator.cpp.

5.5.2.3 DFS_Util()

```
void DFS_Util (
             int N,
             std::set< int > & edge_weights,
             std::vector< std::tuple< int, int, int > > & edges,
             std::vector< std::set< int > > & adj_list,
             std::vector< int > & color,
             std::unordered_map< int, int > & colormap )
```

DFS_Util runs DFS for all nodes and puts them into connected components. All the connected components are then joined by edges linearly.

Parameters

N	index of the current node
edge_weights	Set of edge weights of the graph
edges	Set of edges of the graph point to point
adj_list	adjacency set of the graph
color	color of connected components of various nodes
colormap	a map from color to one of its representative node

Definition at line 81 of file input_generator.cpp.

5.6 main.cpp File Reference

This file contains the "main" function and does I/O and runs the GHS Algorithm.

```
#include <bits/stdc++.h>
#include <pthread.h>
#include "dot_graph.h"
#include "Graph.h"
```

Functions

- std::vector< int > int_extractor (std::string s)

  Given a comma seperated string, this returns a vector of integers.
- Graph< int, int > ∗ GraphInput (int &n, int &m, std::vector< std::tuple< int, int, int > > &edges)

  Take in the graph as per the assignment statement.
- std::vector< std::unordered_map< int, int > > ThreadAdjList (int &n, int &m, std::vector< std::tuple< int, int, int > > &edges)

  Breaks down the input into adjacency list.
- Graph< int, int > ∗ thread_runner (std::vector< std::unordered_map< int, int > > &adj_list)

  Initializes and runs all threads.
- int **main** ()

5.6.1 Function Documentation

5.6.1.1 int_extractor()

```
std::vector<int> int_extractor (
            std::string s )
```

Parameters

s	Comma seperated string consisting of 3 integers : 2 vertices and 1 edge

Definition at line 15 of file main.cpp.

5.6.1.2 ThreadAdjList()

```
std::vector<std::unordered_map<int, int> > ThreadAdjList (
            int & n,
            int & m,
            std::vector< std::tuple< int, int, int > > & edges )
```

Parameters

n	Number of nodes
m	Number of edges
edges	List of edges with their weights

Definition at line 91 of file main.cpp.