DA2324_PRJ1_G184

Generated by Doxygen 1.9.1

Chapter 1

DA2324_PRJ1_G184

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 files with brief descriptions:

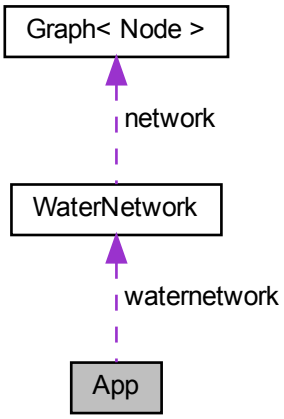
Chapter 4

Class Documentation

4.1 App Class Reference

`#include <App.hpp>`

Collaboration diagram for App:



Public Member Functions

- App (WaterNetwork &waternetwowk)
- std::string upperCase (const std::string &str)
- void mainMenu ()
- void goBackMainMenu ()
- void statisticsMenu ()
- void maxFlowMenu ()
- void waterNeedsMenu ()
- void reservoirImpactMenu ()
- void stationImpactMenu ()
- void pipeImpactMenu ()

Private Attributes

- WaterNetwork waternetwork

4.1.1 Constructor & Destructor Documentation

4.1.1.1 App()

```
App::App (
            WaterNetwork & waternetwowk )
```

4.1.2 Member Function Documentation

4.1.2.1 goBackMainMenu()

```
void App::goBackMainMenu ( )
```

4.1.2.2 mainMenu()

```
void App::mainMenu ( )
```

4.1.2.3 maxFlowMenu()

```
void App::maxFlowMenu ( )
```

4.1.2.4 pipeImpactMenu()

```
void App::pipeImpactMenu ( )
```

4.1.2.5 reservoirImpactMenu()

```
void App::reservoirImpactMenu ( )
```

4.1.2.6 stationImpactMenu()

```
void App::stationImpactMenu ( )
```

4.1.2.7 statisticsMenu()

```
void App::statisticsMenu ( )
```

4.1.2.8 upperCase()

```
std::string App::upperCase (
            const std::string & str )
```

4.1.2.9 waterNeedsMenu()

```
void App::waterNeedsMenu ( )
```

4.1.3 Member Data Documentation

4.1.3.1 waternetwork

```
WaterNetwork App::waternetwork  [private]
```

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

- inc/App.hpp
- src/App.cpp

4.2 Edge< T > Class Template Reference

```
#include <Graph.hpp>
```

Public Member Functions

- Edge (Vertex< T > *orig, Vertex< T > *dest, double w)
- Vertex< T > * getDest () const
- double getWeight () const
- bool isSelected () const
- Vertex< T > * getOrig () const
- Edge< T > * getReverse () const
- double getFlow () const
- void setWeight (double weight)
- void setSelected (bool selected)
- void setReverse (Edge< T > *reverse)
- void setFlow (double flow)

Protected Attributes

- Vertex< T > * dest
- double weight
- bool selected = false
- Vertex< T > * orig
- Edge< T > * reverse = nullptr
- double flow

4.2.1 Constructor & Destructor Documentation

4.2.1.1 Edge()

```
template<class T >
Edge< T >::Edge (
            Vertex< T > * orig,
            Vertex< T > * dest,
            double w )
```

4.2.2 Member Function Documentation

4.2.2.1 getDest()

```
template<class T >
Vertex< T > * Edge< T >::getDest
```

4.2.2.2 getFlow()

```
template<class T >
double Edge< T >::getFlow
```

4.2.2.3 getOrig()

```
template<class T >
Vertex< T > * Edge< T >::getOrig
```

4.2.2.4 getReverse()

```
template<class T >
Edge< T > * Edge< T >::getReverse
```

4.2.2.5 getWeight()

```
template<class T >
double Edge< T >::getWeight
```

4.2.2.6 isSelected()

```
template<class T >
bool Edge< T >::isSelected
```

4.2.2.7 setFlow()

```
template<class T >
void Edge< T >::setFlow (
            double flow )
```

4.2.2.8 setReverse()

```
template<class T >
void Edge< T >::setReverse (
            Edge< T > * reverse )
```

4.2.2.9 setSelected()

```
template<class T >
void Edge< T >::setSelected (
            bool selected )
```

4.2.2.10 setWeight()

```
template<class T >
void Edge< T >::setWeight (
            double weight )
```

4.2.3 Member Data Documentation

4.2.3.1 dest

```
template<class T >
Vertex<T>* Edge< T >::dest  [protected]
```

4.2.3.2 flow

```
template<class T >
double Edge< T >::flow  [protected]
```

4.2.3.3 orig

```
template<class T >
Vertex<T>* Edge< T >::orig  [protected]
```

4.2.3.4 reverse

```
template<class T >
Edge<T>* Edge< T >::reverse = nullptr  [protected]
```

4.2.3.5 selected

```
template<class T >
bool Edge< T >::selected = false  [protected]
```

4.2.3.6 weight

```
template<class T >
double Edge< T >::weight  [protected]
```

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

- inc/Graph.hpp

4.3 Graph< T > Class Template Reference

```
#include <Graph.hpp>
```

Public Member Functions

- Vertex< T > * findVertex (const T &in) const
- bool addVertex (const T &in)
- bool removeVertex (const T &in)
- bool addEdge (const T &sourc, const T &dest, double w)
- bool removeEdge (const T &source, const T &dest)
- bool addBidirectionalEdge (const T &sourc, const T &dest, double w)
- int getNumVertex () const
- std::vector< Vertex< T > * > getVertexSet () const
- std::vector< T > dfs () const
- std::vector< T > dfs (const T &source) const
- void dfsVisit (Vertex< T > *v, std::vector< Vertex< T > * > &res) const
- std::vector< T > bfs (const T &source) const
- bool isDAG () const
- bool dfsIsDAG (Vertex< T > *v) const
- std::vector< T > topsort () const

Protected Member Functions

- int findVertexIdx (const T &in) const

Protected Attributes

- std::vector< Vertex< T > * > vertexSet

4.3.1 Member Function Documentation

4.3.1.1 addBidirectionalEdge()

```
template<class T >
bool Graph< T >::addBidirectionalEdge (
            const T & sourc,
            const T & dest,
            double w )
```

4.3.1.2 addEdge()

```
template<class T >
bool Graph< T >::addEdge (
            const T & sourc,
            const T & dest,
            double w )
```

4.3.1.3 addVertex()

```
template<class T >
bool Graph< T >::addVertex (
            const T & in )
```

4.3.1.4 bfs()

```
template<class T >
std::vector< T > Graph< T >::bfs (
            const T & source ) const
```

4.3.1.5 dfs() [1/2]

```
template<class T >
std::vector< T > Graph< T >::dfs
```

4.3.1.6 dfs() [2/2]

```
template<class T >
std::vector< T > Graph< T >::dfs (
            const T & source ) const
```

4.3.1.7 dfsIsDAG()

```
template<class T >
bool Graph< T >::dfsIsDAG (
            Vertex< T > * v ) const
```

Auxiliary function that visits a vertex (v) and its adjacent, recursively. Returns false (not acyclic) if an edge to a vertex in the stack is found.

4.3.1.8 dfsVisit()

```
template<class T >
void Graph< T >::dfsVisit (
            Vertex< T > * v,
            std::vector< Vertex< T > * > & res ) const
```

4.3.1.9 findVertex()

```
template<class T >
Vertex< T > * Graph< T >::findVertex (
            const T & in ) const
```

4.3.1.10 findVertexIdx()

```
template<class T >
int Graph< T >::findVertexIdx (
            const T & in ) const  [protected]
```

4.3.1.11 getNumVertex()

```
template<class T >
int Graph< T >::getNumVertex
```

4.3.1.12 getVertexSet()

```
template<class T >
std::vector< Vertex< T > * > Graph< T >::getVertexSet
```

4.3.1.13 isDAG()

```
template<class T >
bool Graph< T >::isDAG
```

4.3.1.14 removeEdge()

```
template<class T >
bool Graph< T >::removeEdge (
            const T & source,
            const T & dest )
```

4.3.1.15 removeVertex()

```
template<class T >
bool Graph< T >::removeVertex (
            const T & in )
```

4.3.1.16 topsort()

```
template<class T >
std::vector< T > Graph< T >::topsort
```

4.3.2 Member Data Documentation

4.3.2.1 vertexSet

```
template<class T >
std::vector<Vertex<T> *> Graph< T >::vertexSet  [protected]
```

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

- inc/Graph.hpp

4.4 Node Class Reference

Represents a node in a network.

`#include <Node.hpp>`

Public Member Functions

- Node ()

  Default constructor.
- Node (const unsigned char &type, const std::string &code, const std::string &reservoir, const std::string &municipality, const int &maxDelivery)

  Constructor for a reservoir node.
- Node (const unsigned char &type, const std::string &code)

  Constructor for a station node.
- Node (const unsigned char &type, const std::string &code, const std::string &municipality, const int &demand)

  Constructor for a city node.
- Node (const std::string &code)

  Constructor to find nodes.
- size_t getType () const

  Get the type of the node.
- std::string getCode () const

  Get the code associated with the node.
- std::string getReservoir () const

  Get the name of the reservoir.
- std::string getMunicipality () const

  Get the name of the municipality.
- int getMaxDelivery () const

  Get the maximum delivery of the reservoir.
- int getDemand () const

  Get the demand of the city.
- int getPopulation () const

  Get the population of the city.
- bool operator== (const Node &other)

  Overloaded equality operator.

Private Attributes

- unsigned char type
- std::string code
- std::string reservoir
- std::string municipality
- int maxDelivery
- int demand
- int population

4.4.1 Detailed Description

Represents a node in a network.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Node() [1/5]

```
Node::Node ( )
```

Default constructor.

4.4.2.2 Node() [2/5]

```
Node::Node (
            const unsigned char & type,
            const std::string & code,
            const std::string & reservoir,
            const std::string & municipality,
            const int & maxDelivery )
```

Constructor for a reservoir node.

Parameters

type	The type of the node.
code	The code associated with the node.
reservoir	The name of the reservoir.
municipality	The name of the municipality.
maxDelivery	The maximum delivery of the reservoir.

4.4.2.3 Node() [3/5]

```
Node::Node (
            const unsigned char & type,
            const std::string & code )
```

Constructor for a station node.

Parameters

type	The type of the node.
code	The code associated with the node.

4.4.2.4 Node() [4/5]

```
Node::Node (
            const unsigned char & type,
            const std::string & code,
            const std::string & municipality,
            const int & demand )
```

Constructor for a city node.

Parameters

type	The type of the node.
code	The code associated with the node.
municipality	The name of the municipality.
demand	The demand of the city.

4.4.2.5 Node() [5/5]

```
Node::Node (
            const std::string & code )
```

Constructor to find nodes.

Parameters

code	The code associated with the node.

4.4.3 Member Function Documentation

4.4.3.1 getCode()

```
std::string Node::getCode ( ) const
```

Get the code associated with the node.

Returns

The code associated with the node.

4.4.3.2 getDemand()

```
int Node::getDemand ( ) const
```

Get the demand of the city.

Returns

The demand of the city.

4.4.3.3 getMaxDelivery()

```
int Node::getMaxDelivery ( ) const
```

Get the maximum delivery of the reservoir.

Returns

The maximum delivery of the reservoir.

4.4.3.4 getMunicipality()

```
std::string Node::getMunicipality ( ) const
```

Get the name of the municipality.

Returns

The name of the municipality.

4.4.3.5 getPopulation()

```
int Node::getPopulation ( ) const
```

Get the population of the city.

Returns

The population of the city.

4.4.3.6 getReservoir()

```
std::string Node::getReservoir ( ) const
```

Get the name of the reservoir.

Returns

The name of the reservoir.

4.4.3.7 getType()

```
size_t Node::getType ( ) const
```

Get the type of the node.

Returns

The type of the node.

4.4.3.8 operator==()

```
bool Node::operator== (
            const Node & other )
```

Overloaded equality operator.

Parameters

other	The other node to compare.

Returns

True if the nodes codes are equal, false otherwise.

4.4.4 Member Data Documentation

4.4.4.1 code

```
std::string Node::code  [private]
```

4.4.4.2 demand

```
int Node::demand  [private]
```

4.4.4.3 maxDelivery

```
int Node::maxDelivery  [private]
```

4.4.4.4 municipality

```
std::string Node::municipality  [private]
```

4.4.4.5 population

```
int Node::population  [private]
```

4.4.4.6 reservoir

```
std::string Node::reservoir  [private]
```

4.4.4.7 type

```
unsigned char Node::type  [private]
```

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

- inc/Node.hpp
- src/Node.cpp

4.5 Vertex< T > Class Template Reference

```
#include <Graph.hpp>
```

Public Member Functions

- Vertex (T in)
- T getInfo () const
- std::vector< Edge< T > * > getAdj () const
- bool isVisited () const
- bool isProcessing () const
- unsigned int getIndegree () const
- double getDist () const
- Edge< T > * getPath () const
- std::vector< Edge< T > * > getIncoming () const
- double getCurrentFlow () const
- void setCurrentFlow (const double &cur)
- double getUsedDelivery () const
- void setUsedDelivery (const double &used)
- void setInfo (T info)
- void setVisited (bool visited)
- void setProcesssing (bool processing)
- void setIndegree (unsigned int indegree)
- void setDist (double dist)
- void setPath (Edge< T > *path)
- Edge< T > * addEdge (Vertex< T > *dest, double w)
- bool removeEdge (T in)
- void removeOutgoingEdges ()

Protected Member Functions

- void deleteEdge (Edge< T > *edge)

Protected Attributes

- T info
- std::vector< Edge< T > * > adj
- bool visited = false
- bool processing = false
- unsigned int indegree
- double dist = 0
- double currentFlow = 0
- double usedDelivery = 0
- Edge< T > * path = nullptr
- std::vector< Edge< T > * > incoming

4.5.1 Constructor & Destructor Documentation

4.5.1.1 Vertex()

```
template<class T >
Vertex< T >::Vertex (
            T in )
```

4.5.2 Member Function Documentation

4.5.2.1 addEdge()

```
template<class T >
Edge< T > * Vertex< T >::addEdge (
            Vertex< T > * dest,
            double w )
```

4.5.2.2 deleteEdge()

```
template<class T >
void Vertex< T >::deleteEdge (
            Edge< T > * edge )  [protected]
```

4.5.2.3 getAdj()

```
template<class T >
std::vector< Edge< T > * > Vertex< T >::getAdj
```

4.5.2.4 getCurrentFlow()

```
template<class T >
double Vertex< T >::getCurrentFlow
```

4.5.2.5 getDist()

```
template<class T >
double Vertex< T >::getDist
```

4.5.2.6 getIncoming()

```
template<class T >
std::vector< Edge< T > * > Vertex< T >::getIncoming
```

4.5.2.7 getIndegree()

```
template<class T >
unsigned int Vertex< T >::getIndegree
```

4.5.2.8 getInfo()

```
template<class T >
T Vertex< T >::getInfo
```

4.5.2.9 getPath()

```
template<class T >
Edge< T > * Vertex< T >::getPath
```

4.5.2.10 getUsedDelivery()

```
template<class T >
double Vertex< T >::getUsedDelivery
```

4.5.2.11 isProcessing()

```
template<class T >
bool Vertex< T >::isProcessing
```

4.5.2.12 isVisited()

```
template<class T >
bool Vertex< T >::isVisited
```

4.5.2.13 removeEdge()

```
template<class T >
bool Vertex< T >::removeEdge (
            T in )
```

4.5.2.14 removeOutgoingEdges()

```
template<class T >
void Vertex< T >::removeOutgoingEdges
```

4.5.2.15 setCurrentFlow()

```
template<class T >
void Vertex< T >::setCurrentFlow (
            const double & cur )
```

4.5.2.16 setDist()

```
template<class T >
void Vertex< T >::setDist (
            double dist )
```

4.5.2.17 setIndegree()

```
template<class T >
void Vertex< T >::setIndegree (
            unsigned int indegree )
```

4.5.2.18 setInfo()

```
template<class T >
void Vertex< T >::setInfo (
            T info )
```

4.5.2.19 setPath()

```
template<class T >
void Vertex< T >::setPath (
            Edge< T > * path )
```

4.5.2.20 setProcesssing()

```
template<class T >
void Vertex< T >::setProcesssing (
            bool processing )
```

4.5.2.21 setUsedDelivery()

```
template<class T >
void Vertex< T >::setUsedDelivery (
            const double & used )
```

4.5.2.22 setVisited()

```
template<class T >
void Vertex< T >::setVisited (
            bool visited )
```

4.5.3 Member Data Documentation

4.5.3.1 adj

```
template<class T >
std::vector<Edge<T> *> Vertex< T >::adj  [protected]
```

4.5.3.2 currentFlow

```
template<class T >
double Vertex< T >::currentFlow = 0  [protected]
```

4.5.3.3 dist

```
template<class T >
double Vertex< T >::dist = 0  [protected]
```

4.5.3.4 incoming

```
template<class T >
std::vector<Edge<T> *> Vertex< T >::incoming  [protected]
```

4.5.3.5 indegree

```
template<class T >
unsigned int Vertex< T >::indegree  [protected]
```

4.5.3.6 info

```
template<class T >
T Vertex< T >::info  [protected]
```

4.5.3.7 path

```
template<class T >
Edge<T>* Vertex< T >::path = nullptr  [protected]
```

4.5.3.8 processing

```
template<class T >
bool Vertex< T >::processing = false  [protected]
```

4.5.3.9 usedDelivery

```
template<class T >
double Vertex< T >::usedDelivery = 0  [protected]
```

4.5.3.10 visited

```
template<class T >
bool Vertex< T >::visited = false  [protected]
```

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

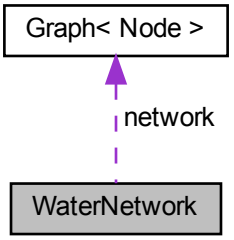
- inc/Graph.hpp

4.6 WaterNetwork Class Reference

Class representing a water supply network.
.

`#include <WaterNetwork.hpp>`

Collaboration diagram for WaterNetwork:



Public Member Functions

- WaterNetwork ()

  Default constructor for the WaterNetwork class.
- WaterNetwork (const std::string reservoirs_filename, const std::string stations_filename, const std::string cities_filename, const std::string pipes_filename)

  Constructor for the WaterNetwork class.
  This constructor initializes a water network graph by parsing data from provided files.
- ~WaterNetwork ()

  Destructor for the WaterNetwork class.
- Graph< Node > * getNetworkGraph () const

  Getter for the network graph.
- double singleSinkMaxFlow (const std::string &city_code) const
- std::vector< std::pair< std::string, double > > multiSinkMaxFlow () const
- std::vector< std::pair< std::string, double > > multiWaterNeeds (Graph< Node > *g, const bool &flag) const
- std::vector< std::pair< std::string, double > > evaluateReservoirImpact (const std::string &reservoir_code) const
- void evaluateAllReservoirImpact () const
- void evaluateAllPumpingStationImpact () const
- void evaluatePipelineImpact (const std::string &city_code) const

Private Attributes

- Graph< Node > * network

4.6.1 Detailed Description

Class representing a water supply network.
.

This class provides functionality to model and analyze a water supply network, including methods for calculating maximum flow, evaluating network metrics, and managing network components.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 WaterNetwork() [1/2]

```
WaterNetwork::WaterNetwork ( )
```

Default constructor for the WaterNetwork class.

4.6.2.2 WaterNetwork() [2/2]

```
WaterNetwork::WaterNetwork (
            const std::string reservoirs_filename,
            const std::string stations_filename,
            const std::string cities_filename,
            const std::string pipes_filename )
```

Constructor for the WaterNetwork class.
This constructor initializes a water network graph by parsing data from provided files.

Parameters

reservoirs_filename	The filename of the CSV file containing reservoir data.
stations_filename	The filename of the CSV file containing station data.
cities_filename	The filename of the CSV file containing city data.
pipes_filename	The filename of the CSV file containing pipe data.

4.6.2.3 ~WaterNetwork()

```
WaterNetwork::~WaterNetwork ( )
```

Destructor for the WaterNetwork class.

4.6.3 Member Function Documentation

4.6.3.1 evaluateAllPumpingStationImpact()

```
void WaterNetwork::evaluateAllPumpingStationImpact ( ) const
```

4.6.3.2 evaluateAllReservoirImpact()

```
void WaterNetwork::evaluateAllReservoirImpact ( ) const
```

4.6.3.3 evaluatePipelineImpact()

```
void WaterNetwork::evaluatePipelineImpact (
            const std::string & city_code ) const
```

4.6.3.4 evaluateReservoirImpact()

```
vector< pair< string, double > > WaterNetwork::evaluateReservoirImpact (
            const std::string & reservoir_code ) const
```

4.6.3.5 getNetworkGraph()

```
Graph< Node > * WaterNetwork::getNetworkGraph ( ) const
```

Getter for the network graph.

Returns

A pointer to the network graph.

4.6.3.6 multiSinkMaxFlow()

```
vector< pair< string, double > > WaterNetwork::multiSinkMaxFlow ( ) const
```

4.6.3.7 multiWaterNeeds()

```
vector< pair< string, double > > WaterNetwork::multiWaterNeeds (
            Graph< Node > * g,
            const bool & flag ) const
```

4.6.3.8 singleSinkMaxFlow()

```
double WaterNetwork::singleSinkMaxFlow (
            const std::string & city_code ) const
```

4.6.4 Member Data Documentation

4.6.4.1 network

```
Graph<Node>* WaterNetwork::network  [private]
```

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

- inc/WaterNetwork.hpp
- src/WaterNetwork.cpp

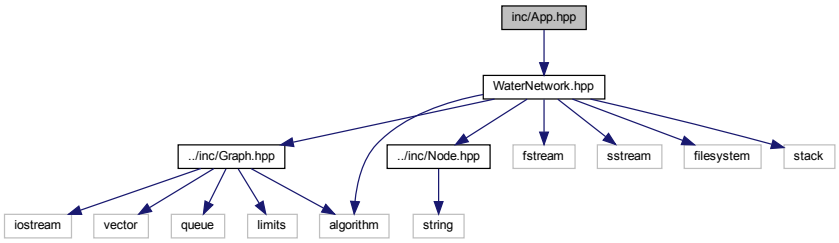
Chapter 5

File Documentation

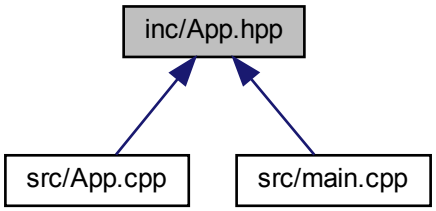
5.1 inc/App.hpp File Reference

```
#include "WaterNetwork.hpp"
```

Include dependency graph for App.hpp:



This graph shows which files directly or indirectly include this file:



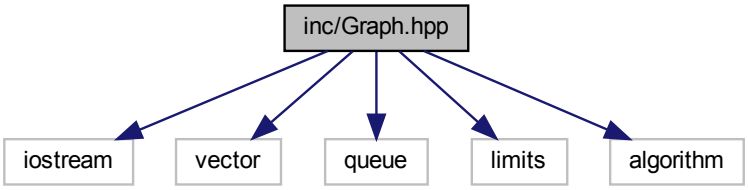
Classes

- class App

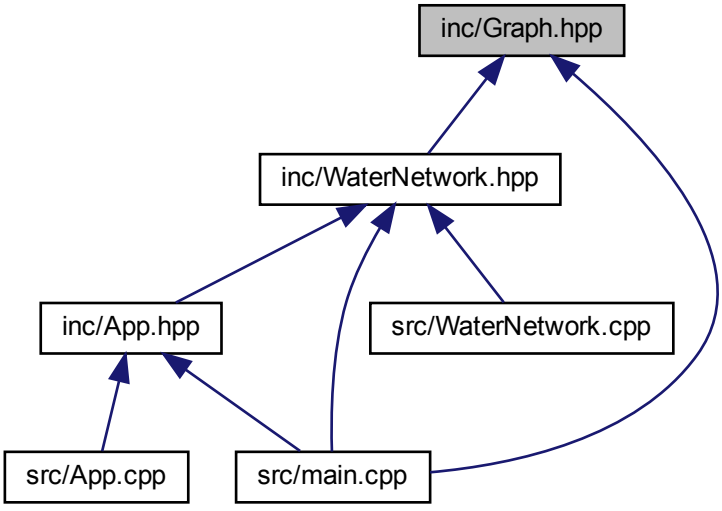
5.2 inc/Graph.hpp File Reference

```
#include <iostream>
#include <vector>
#include <queue>
#include <limits>
#include <algorithm>
```

Include dependency graph for Graph.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Vertex< T >
- class Edge< T >
- class Graph< T >

Macros

- #define INF std::numeric_limits<double>::max()

5.2.1 Macro Definition Documentation

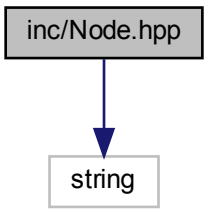
5.2.1.1 INF

```
#define INF std::numeric_limits<double>::max()
```

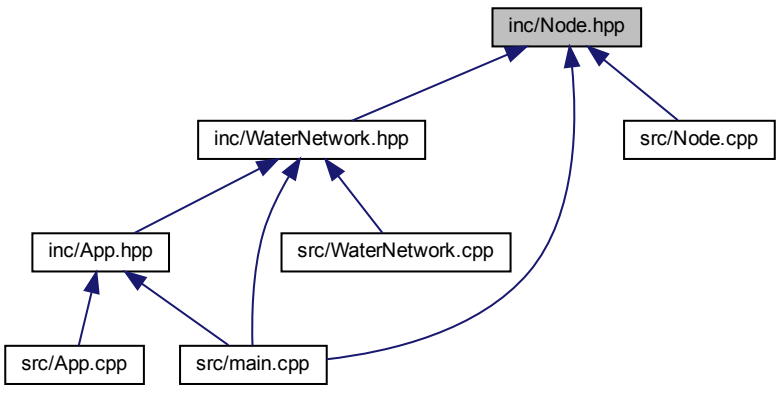
5.3 inc/Node.hpp File Reference

```
#include <string>
```

Include dependency graph for Node.hpp:



This graph shows which files directly or indirectly include this file:



Classes

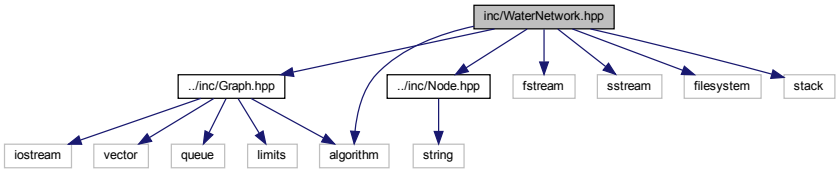
- class Node

  Represents a node in a network.

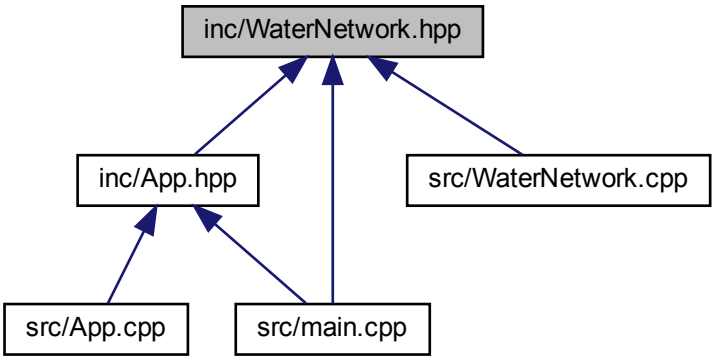
5.4 inc/WaterNetwork.hpp File Reference

```
#include "../inc/Graph.hpp"
#include "../inc/Node.hpp"
#include <algorithm>
#include <fstream>
#include <sstream>
#include <filesystem>
#include <stack>
```

Include dependency graph for WaterNetwork.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class WaterNetwork

  Class representing a water supply network.

  .

Functions

- void testAndVisit (std::queue< Vertex< Node > * > &q, Edge< Node > *e, Vertex< Node > *w, double residual)

  Helper function to test and visit vertices during BFS traversal.

  .
- bool findAugmentingPath (Graph< Node > *g, Vertex< Node > *s, Vertex< Node > *t)

  Function to find an augmenting path in the graph using BFS.

  .
- double findMinResidualAlongPath (Vertex< Node > *s, Vertex< Node > *t)

  Function to find the minimum residual capacity along an augmenting path.

  .
- void augmentFlowAlongPath (Vertex< Node > *s, Vertex< Node > *t, double f)

  Function to augment flow along an augmenting path.

  .
- void edmondsKarp (Graph< Node > *g, Node source, Node target)

  Function implementing the Edmonds-Karp algorithm for maximum flow.

  .
- void resetGraph (Graph< Node > *g, const Node &s, const Node &t)

  Function to reset the state of the graph.

  .
- Node createSuperSource (Graph< Node > *g)

  Function to create a super source node in the graph.

  .
- Node createSuperSink (Graph< Node > *g)

  Function to create a super sink node in the graph.

  .
- Graph< Node > * findConnectedComponent (Graph< Node > *g, const std::string &node_code)

5.4.1 Function Documentation

5.4.1.1 augmentFlowAlongPath()

```
void augmentFlowAlongPath (
            Vertex< Node > * s,
            Vertex< Node > * t,
            double f )
```

Function to augment flow along an augmenting path.

.

This function traverses the augmenting path from the target vertex to the source vertex and updates the flow values of edges and current flow values of vertices accordingly.

Parameters

s	The source vertex.
t	The target vertex.
f	The flow value to augment along the path.

@complexity O(V)

5.4.1.2 createSuperSink()

```
Node createSuperSink (
            Graph< Node > * g )
```

Function to create a super sink node in the graph.
.

This function creates a super sink node in the graph with maximum demand capacity.
It adds edges from all city nodes in the graph except itself to the super sink.

Parameters

g	The graph to add the super sink node.

Returns

The super sink node created.

@complexity O(V)

5.4.1.3 createSuperSource()

```
Node createSuperSource (
            Graph< Node > * g )
```

Function to create a super source node in the graph.
.

This function creates a super source node in the graph with maximum delivery capacity.
It adds edges from the super source to all reservoir nodes in the graph except itself.

Parameters

g	The graph to add the super source node.

Returns

The super source node created.

@complexity O(V)

5.4.1.4 edmondsKarp()

```
void edmondsKarp (
            Graph< Node > * g,
```

```
            Node source,
            Node target )
```

Function implementing the Edmonds-Karp algorithm for maximum flow.
.

This function finds the maximum flow in the graph from a source vertex to a target vertex using the Edmonds-Karp algorithm.
It repeatedly finds augmenting paths using BFS and augments flow along the paths until no more augmenting paths exist.

Parameters

g	The graph representing the water network.
source	The source node.
target	The target node.

@complexity O(V $*$ E^2)

5.4.1.5 findAugmentingPath()

```
bool findAugmentingPath (
            Graph< Node > * g,
            Vertex< Node > * s,
            Vertex< Node > * t )
```

Function to find an augmenting path in the graph using BFS.
.

This function marks all vertices as not visited, marks the source vertex as visited, and performs BFS traversal to find an augmenting path from the source to the target vertex.
It returns true if a path to the target vertex is found, and false otherwise.

Parameters

g	The graph to search for augmenting paths.
s	The source vertex.
t	The target vertex.

Returns

True if an augmenting path is found, false otherwise.

@complexity O(V + E)

5.4.1.6 findConnectedComponent()

```
Graph<Node>* findConnectedComponent (
            Graph< Node > * g,
            const std::string & node_code )
```

5.4.1.7 findMinResidualAlongPath()

```
double findMinResidualAlongPath (
            Vertex< Node > * s,
            Vertex< Node > * t )
```

Function to find the minimum residual capacity along an augmenting path.
.

This function traverses the augmenting path from the target vertex to the source vertex to find the minimum residual capacity.
It considers the capacity of edges and the demand of cities.

Parameters

s	The source vertex.
t	The target vertex.

Returns

The minimum residual capacity along the augmenting path.

@complexity O(V)

5.4.1.8 resetGraph()

```
void resetGraph (
            Graph< Node > * g,
            const Node & s,
            const Node & t )
```

Function to reset the state of the graph.
.

This function resets the state of the graph by marking all vertices as not visited, clearing paths, current flow, and used delivery.
It also resets the flow of all edges to zero. Additionally, it removes the super source and super sink nodes from the graph.

Parameters

g	The graph to reset.
s	The super source node to remove.
t	The super sink node to remove.

@complexity O(V + E)

5.4.1.9 testAndVisit()

```
void testAndVisit (
            std::queue< Vertex< Node > * > & q,
```

```
Edge< Node > * e,
Vertex< Node > * w,
double residual )
```

Helper function to test and visit vertices during BFS traversal.
.

This function checks if the vertex 'w' is not visited and there is residual capacity.
If so, it marks 'w' as visited, sets the path through which it was reached, and enqueues it.

Parameters

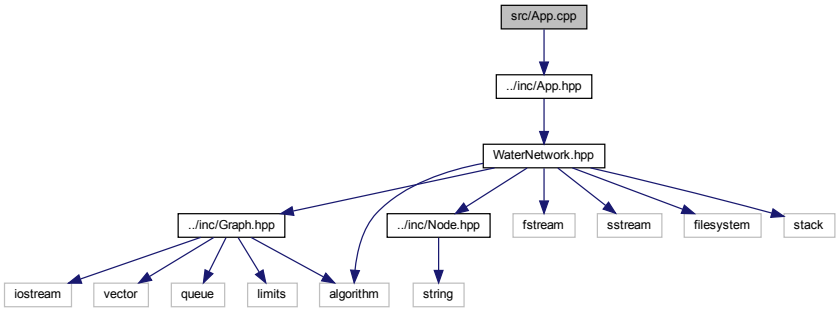
q	The queue of vertices to visit.
e	The edge connecting the current vertex to 'w'.
w	The vertex to visit.
residual	The residual capacity of the edge connecting the current vertex to 'w'.

@complexity O(1)

5.5 README.md File Reference

5.6 src/App.cpp File Reference

```
#include "../inc/App.hpp"
```
Include dependency graph for App.cpp:



Functions

- void clearScreen ()

5.6.1 Function Documentation

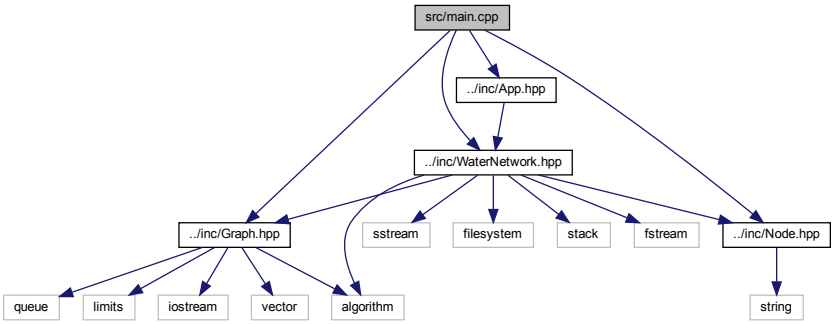
5.6.1.1 clearScreen()

```
void clearScreen ( )
```

5.7 src/main.cpp File Reference

```
#include "../inc/Graph.hpp"
#include "../inc/Node.hpp"
#include "../inc/WaterNetwork.hpp"
#include "../inc/App.hpp"
```

Include dependency graph for main.cpp:



Functions

- int main ()

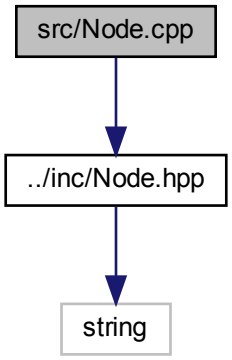
5.7.1 Function Documentation

5.7.1.1 main()

```
int main ( )
```

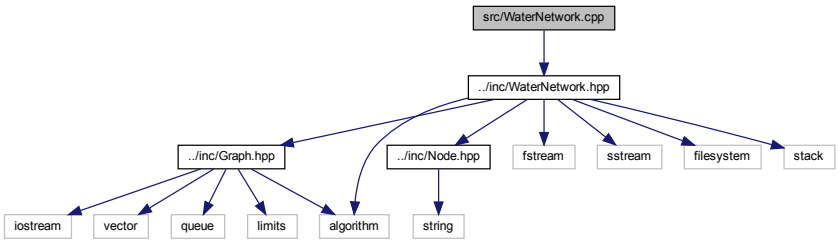
5.8 src/Node.cpp File Reference

`#include "../inc/Node.hpp"`
Include dependency graph for Node.cpp:



5.9 src/WaterNetwork.cpp File Reference

`#include "../inc/WaterNetwork.hpp"`
Include dependency graph for WaterNetwork.cpp:



Functions

- void testAndVisit (queue< Vertex< Node > * > &q, Edge< Node > *e, Vertex< Node > *w, double residual)
- bool findAugmentingPath (Graph< Node > *g, Vertex< Node > *s, Vertex< Node > *t)

  Function to find an augmenting path in the graph using BFS.

  .
- double findMinResidualAlongPath (Vertex< Node > *s, Vertex< Node > *t)

  Function to find the minimum residual capacity along an augmenting path.

  .
- void augmentFlowAlongPath (Vertex< Node > *s, Vertex< Node > *t, double f)

Function to augment flow along an augmenting path.

.

- void edmondsKarp (Graph< Node > *g, Node source, Node target)

  Function implementing the Edmonds-Karp algorithm for maximum flow.

  .
- Node createSuperSource (Graph< Node > *g)

  Function to create a super source node in the graph.

  .
- Node createSuperSink (Graph< Node > *g)

  Function to create a super sink node in the graph.

  .
- void resetGraph (Graph< Node > *g, const Node &s, const Node &t)

  Function to reset the state of the graph.

  .
- Graph< Node > * findConnectedComponent (Graph< Node > *g, const string &node_code)

5.9.1 Function Documentation

5.9.1.1 augmentFlowAlongPath()

```
void augmentFlowAlongPath (
            Vertex< Node > * s,
            Vertex< Node > * t,
            double f )
```

Function to augment flow along an augmenting path.

.

This function traverses the augmenting path from the target vertex to the source vertex and updates the flow values of edges and current flow values of vertices accordingly.

Parameters

s	The source vertex.
t	The target vertex.
f	The flow value to augment along the path.

@complexity O(V)

5.9.1.2 createSuperSink()

```
Node createSuperSink (
            Graph< Node > * g )
```

Function to create a super sink node in the graph.

.

This function creates a super sink node in the graph with maximum demand capacity.
It adds edges from all city nodes in the graph except itself to the super sink.

Parameters

g	The graph to add the super sink node.

Returns

The super sink node created.

@complexity O(V)

5.9.1.3 createSuperSource()

```
Node createSuperSource (
            Graph< Node > * g )
```

Function to create a super source node in the graph.
.

This function creates a super source node in the graph with maximum delivery capacity.
It adds edges from the super source to all reservoir nodes in the graph except itself.

Parameters

g	The graph to add the super source node.

Returns

The super source node created.

@complexity O(V)

5.9.1.4 edmondsKarp()

```
void edmondsKarp (
            Graph< Node > * g,
            Node source,
            Node target )
```

Function implementing the Edmonds-Karp algorithm for maximum flow.
.

This function finds the maximum flow in the graph from a source vertex to a target vertex using the Edmonds-Karp algorithm.
It repeatedly finds augmenting paths using BFS and augments flow along the paths until no more augmenting paths exist.

Parameters

g	The graph representing the water network.
source	The source node.
target	The target node.

@complexity O(V ∗ E^2)

5.9.1.5 findAugmentingPath()

```
bool findAugmentingPath (
            Graph< Node > * g,
            Vertex< Node > * s,
            Vertex< Node > * t )
```

Function to find an augmenting path in the graph using BFS.
.

This function marks all vertices as not visited, marks the source vertex as visited, and performs BFS traversal to find an augmenting path from the source to the target vertex.
It returns true if a path to the target vertex is found, and false otherwise.

Parameters

g	The graph to search for augmenting paths.
s	The source vertex.
t	The target vertex.

Returns

True if an augmenting path is found, false otherwise.

@complexity O(V + E)

5.9.1.6 findConnectedComponent()

```
Graph<Node>* findConnectedComponent (
            Graph< Node > * g,
            const string & node_code )
```

5.9.1.7 findMinResidualAlongPath()

```
double findMinResidualAlongPath (
            Vertex< Node > * s,
            Vertex< Node > * t )
```

Function to find the minimum residual capacity along an augmenting path.
.

This function traverses the augmenting path from the target vertex to the source vertex to find the minimum residual capacity.
It considers the capacity of edges and the demand of cities.

Parameters

s	The source vertex.
t	The target vertex.

Returns

The minimum residual capacity along the augmenting path.

@complexity O(V)

5.9.1.8 resetGraph()

```
void resetGraph (
            Graph< Node > * g,
            const Node & s,
            const Node & t )
```

Function to reset the state of the graph.
.

This function resets the state of the graph by marking all vertices as not visited, clearing paths, current flow, and used delivery.
It also resets the flow of all edges to zero. Additionally, it removes the super source and super sink nodes from the graph.

Parameters

g	The graph to reset.
s	The super source node to remove.
t	The super sink node to remove.

@complexity O(V + E)

5.9.1.9 testAndVisit()

```
void testAndVisit (
            queue< Vertex< Node > * > & q,
            Edge< Node > * e,
            Vertex< Node > * w,
            double residual )
```

Index