Transició de fase i components connexes ens grafs aleatoris

Generated by Doxygen 1.8.13

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

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

Chapter 3

Class Documentation

3.1 Graph Class Reference

Stores the information of a Grpah and all its nodes using adjacency lists.

`#include <Graph.h>`

Public Member Functions

- Graph (int maxID)

  Creates and empty Graph.
- void setRandomCoordinates (int maxX, int maxY)

  Set all the nodes of the Graph to random coordinates with the specified upper boundary.
- void addEdge (int id1, int id2)

  Adds and edge between the two specified vertices if these exists and an edge does not already exist.
- set< int > getVertices ()

  Returns all the vertices of the GraphR.
- set< pair< int, int > > getEdges ()

  Returns all the edges of the Graph.
- bool isActive (int id)

  Verifies if there's a vertex with the specified ID.
- bool adjacent (int id1, int id2)

  Verifies if two vertices are adjacent.
- set< int > getAdjacencies (int id)

  Returns the adjacency list of a given vertex.
- vector< vector< int > > getConnectedComponents ()

  Checks the connectivity of the Graph.
- int getNumberVertices ()

  Returns the number of vertices of the Graph.
- void print ()

  print via the standard output the Graph

Static Public Member Functions

- static Graph generateERGraph (int n, int m)

  Creates a random Graph with a given number of vertices following the Erdos-Renyi model.
- static Graph generateRGGraph (int n, float r)

  Creates a Random Geometric Graph with a given number of vertices and edges between those vertices that their distance is in the given range 'r'.

3.1.1 Detailed Description

Stores the information of a Grpah and all its nodes using adjacency lists.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Graph()

```
Graph::Graph (
            int maxID )
```

Creates and empty Graph.

Constructors =1mm

spread 0pt [l]|X[-1,l]|X[-1,l]**Parameters**

Parameters

maxID Maximum ID value to assign to the Graph nodes

3.1.3 Member Function Documentation

3.1.3.1 addEdge()

```
void Graph::addEdge (
            int id1,
            int id2 )
```

Adds and edge between the two specified vertices if these exists and an edge does not already exist.

Setters =1mm

spread 0pt [l]|X[-1,l]|X[-1,l]**Parameters**

Parameters

id1 ID of one of the nodes

id2 ID of the other node

Creates a random Graph with a given number of vertices following the Erdos-Renyi model.

=1mm

spread 0pt [l]|X[-1,l]|X[-1,l]**Parameters**

Parameters

---

n Number of vertices of the generated Graph

---

m Number of edges of the resulting Graph

Returns

Random Graph with n vertices generated following the Erdos-Renyi model

3.1.3.4 generateRGGraph()

```
Graph Graph::generateRGGraph (
            int n,
            float r )  [static]
```

Creates a Random Geometric Graph with a given number of vertices and edges between those vertices that their distance is in the given range 'r'.

=1mm

spread 0pt [l]|X[-1,l]|X[-1,l]**Parameters**

Parameters

---

n Number of vertices of the generated Graph

---

r Neighbour radius parameter

Returns

Random Geometric Graph with n vertices and edges between those vertices that their distance is in the given range 'r'

3.1.3.5 getAdjacencies()

```
set< int > Graph::getAdjacencies (
            int id )
```

Returns the adjacency list of a given vertex.

=1mm

spread 0pt [l]|X[-1,l]|X[-1,l]**Parameters**

Parameters

---

id Vertex ID to consult its adjacency list

Returns

Adjacency list of vertex 'id'

3.1.3.6 getConnectedComponents()

```
vector< vector< int > > Graph::getConnectedComponents ( )
```

Checks the connectivity of the Graph.

Returns

Vertices that make up each of the connected components

3.1.3.7 getEdges()

```
set< pair< int, int > > Graph::getEdges ( )
```

Returns all the edges of the Graph.

Returns

Edges of the Graph

3.1.3.8 getNumberVertices()

```
int Graph::getNumberVertices ( )
```

Returns the number of vertices of the Graph.

Returns

Number of vertices of the Graph

3.1.3.9 getVertices()

```
set< int > Graph::getVertices ( )
```

Returns all the vertices of the GraphR.

Getters

Returns

Vertices IDs of the Graph

3.1.3.10 isActive()

```
bool Graph::isActive (
            int id )
```

Verifies if there's a vertex with the specified ID.

=1mm

spread 0pt [l]|X[-1,l]|X[-1,l]**Parameters**

Parameters

---

id Vertex ID to check

Returns

True if the Grpah has a vertex with the specified ID. False otherwise

3.1.3.11 setRandomCoordinates()

```
void Graph::setRandomCoordinates (
            int maxX,
            int maxY )
```

Set all the nodes of the Graph to random coordinates with the specified upper boundary.

=1mm

spread 0pt [l]|X[-1,l]|X[-1,l]**Parameters**

Parameters

---

maxX Maximum X-axis coordinate

---

maxY Maximum Y-axis coordinate

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

- Graph.h
- Graph.cpp

3.2 GreatGraph Class Reference

This class tests the phase transition of the expected size of the greatest connected component.

`#include <GreatGraph.h>`

Public Member Functions

- GreatGraph ()
  Class constructor.
- void Test ()
  Carry out 'GreatGraph' Experiment.

3.2.1 Detailed Description

This class tests the phase transition of the expected size of the greatest connected component.

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

- GreatGraph.h
- GreatGraph.cpp

3.3 Node Struct Reference

Stores the information of one Node of the Graph.

`#include <Graph.h>`

Public Attributes

- bool active
  Indicates if the node ID is the same as the vector position of the Graph where this node belongs.
- set< int > adjacencies
  collection of IDs of the adjacent nodes
- pair< int, int > coordinates
  If the node belongs to a RGG stores the coordinates of the node. Otherwise the values will be (-1, -1).

3.3.1 Detailed Description

Stores the information of one Node of the Graph.

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

- Graph.h

3.4 probConnex Class Reference

This class tests the phase transition of the probability of being a connected Graph.

`#include <probConnex.h>`

Public Member Functions

- probConnex ()
  Class constructor.
- void Experiment ()
  Carry out 'probConnex' Experiment.

3.4.1 Detailed Description

This class tests the phase transition of the probability of being a connected Graph.

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

- probConnex.h
- probConnex.cpp

3.5 UFnode Struct Reference

Stores the information of one Node of Union-Find disjoint set.

`#include <UnionFind.h>`

Public Attributes

- bool active
  Indicates if the node ID is the same as the vector position of the Graph where this node belongs.
- int parent
  Stores the ID of the node's parent.
- int rank
  Rank associated with this node.

3.5.1 Detailed Description

Stores the information of one Node of Union-Find disjoint set.

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

- UnionFind.h

3.6 UnionFind Class Reference

Stores the information of a Union-Find disjoint set and its related information.

`#include <UnionFind.h>`

Public Member Functions

- UnionFind (int maxId)

  UnionFind class constructor.
- void makeSet (int id)

  Create an equivalence class with its representative being the given vertex ID.
- void makeUnion (int id1, int id2)

  Merges the equivalence classes of the nodes with the given IDs if they exists and are from different classes.
- string toString ()

  Transforms the UnionFind into a string.
- int find (int id)

  Get the vertex ID of the representative node of the equivalence class of the given node.
- int getNClasses ()

  Get the number of different equivalence classes of the UnionFind set.
- bool hasNode (int id)

  Checks if the UnionFind set has a node with the given ID.
- bool equivalent (int id1, int id2)

  Check if two nodes belong to the same equivalence class.

3.6.1 Detailed Description

Stores the information of a Union-Find disjoint set and its related information.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 UnionFind()

```
UnionFind::UnionFind (
            int maxId )  [explicit]
```

UnionFind class constructor.

Constructors =1mm

spread 0pt [l]|X[-1,l]|X[-1,l]**Parameters**

Parameters

---

maxId Maximum vertex ID a node of the UnionFind set can have

3.6.3 Member Function Documentation

3.6.3.1 equivalent()

```
bool UnionFind::equivalent (
            int id1,
            int id2 )
```

Check if two nodes belong to the same equivalence class.

=1mm

spread 0pt [l]|X[-1,l]|X[-1,l]**Parameters**

Parameters

---

id1 ID of one of the nodes

---

id2 ID of the other node

Returns

True if the two given nodes belong two the same equivalence class. Otherwise, or if any of the IDs does not exists, false

3.6.3.2 find()

```
int UnionFind::find (
            int id )
```

Get the vertex ID of the representative node of the equivalence class of the given node.

=1mm

spread 0pt [l]|X[-1,l]|X[-1,l]**Parameters**

Parameters

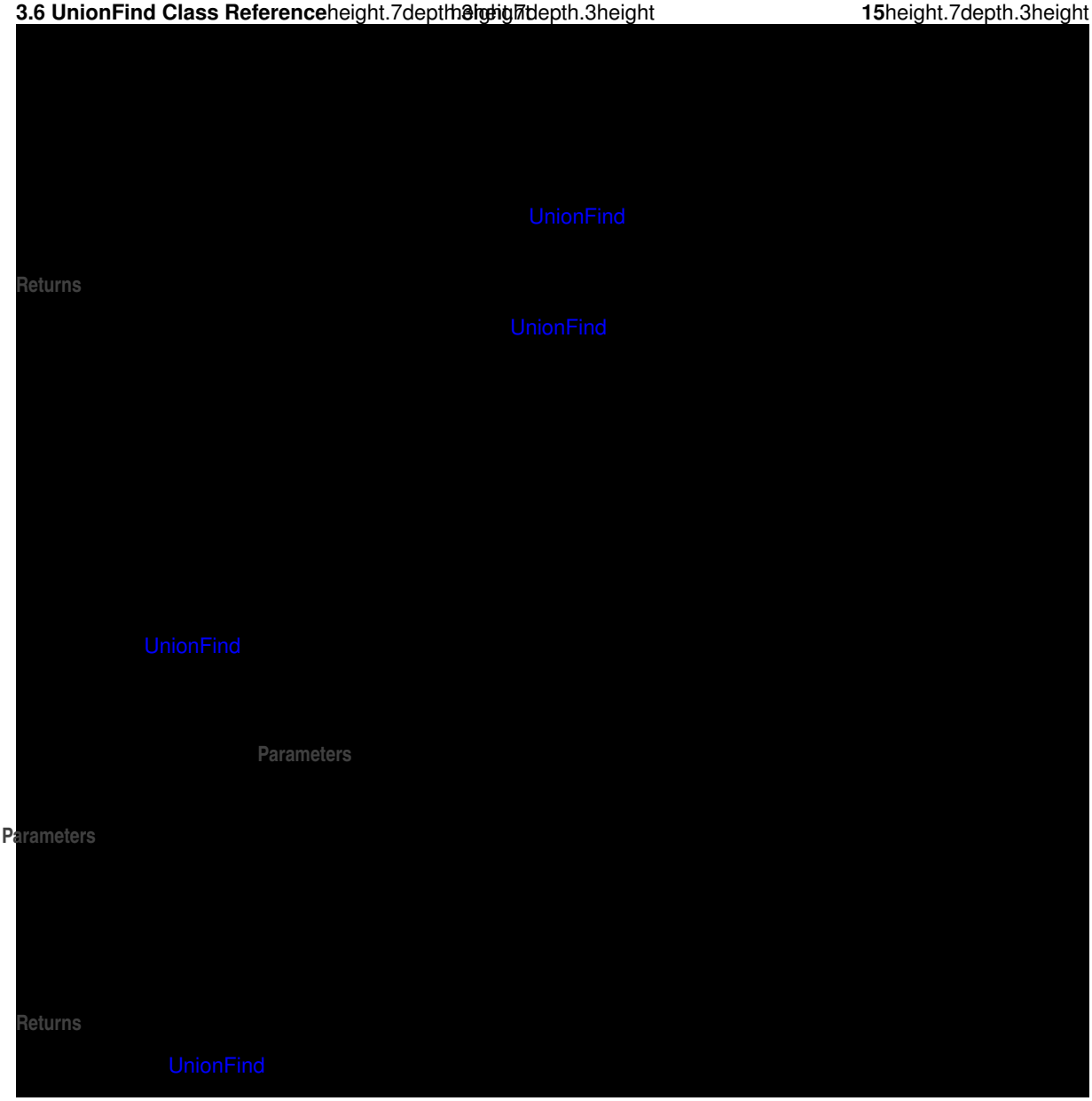
---

id Vertex ID to find the representative of its equivalence class

---

Returns

ID of the representative node of the equivalence class from the UnionFind node with the given id



3.6.3.5 makeSet()

```
void UnionFind::makeSet (
            int id )
```

Create an equivalence class with its representative being the given vertex ID.

Setters =1mm

spread 0pt [l]|X[-1,l]|X[-1,l]

Parameters

Parameters

---

id Vertex ID of the representative of the newly created equivalence class

---

3.6.3.6 makeUnion()

```
void UnionFind::makeUnion (
            int id1,
            int id2 )
```

Merges the equivalence classes of the nodes with the given IDs if they exists and are from different classes.

=1mm

spread 0pt [l]|X[-1,l]|X[-1,l]**Parameters**

Parameters

---

id1 Vertex ID of the first UnionFind node

---

id2 Vertex ID of the first UnionFind node

3.6.3.7 toString()

```
string UnionFind::toString ( )
```

Transforms the UnionFind into a string.

Getters

Returns

string with the UnionFind content

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

- UnionFind.h
- UnionFind.cpp

Chapter 4

File Documentation

4.1 Graph.h File Reference

Graph class specification.

```
#include <iostream>
#include <vector>
#include <set>
#include <string>
#include <cmath>
#include <ctime>
#include "UnionFind.h"
```

Include dependency graph for Graph.h:

4.2 GreatGraph.h File Reference

GreatGraph class specification.

```
#include <iostream>
#include <vector>
#include <set>
#include <fstream>
#include <cmath>
#include <ctime>
```

Include dependency graph for GreatGraph.h:

Classes

- class GreatGraph

  This class tests the phase transition of the expected size of the greatest connected component.

4.2.1 Detailed Description

GreatGraph class specification.

4.3 probConnex.h File Reference

probConnex class specification

```
#include <iostream>
#include <vector>
#include <set>
#include <fstream>
#include <cmath>
```

Include dependency graph for probConnex.h:

Classes

- class probConnex

  This class tests the phase transition of the probability of being a connected Graph.

4.3.1 Detailed Description

probConnex class specification

4.4 UnionFind.h File Reference

UnionFind class specification.

```
#include <iostream>
#include <vector>
#include <string>
```

Include dependency graph for UnionFind.h: This graph shows which files directly or indirectly include this file:

Classes

- struct UFnode

  Stores the information of one Node of Union-Find disjoint set.
- class UnionFind

  Stores the information of a Union-Find disjoint set and its related information.

4.4.1 Detailed Description

UnionFind class specification.