

Social_Graph

1.0

Generated by Doxygen 1.8.16

1 Namespace Index	2
1.1 Namespace List	2
2 Hierarchical Index	2
2.1 Class Hierarchy	2
3 Class Index	2
3.1 Class List	2
4 File Index	2
4.1 File List	2
5 Namespace Documentation	3
5.1 DiGraph Namespace Reference	3
5.1.1 Detailed Description	3
5.1.2 Function Documentation	3
5.2 SocialGraph Namespace Reference	4
5.2.1 Detailed Description	4
5.2.2 Function Documentation	5
6 Class Documentation	5
6.1 DiGraph.DiGraph Class Reference	5
6.1.1 Detailed Description	7
6.1.2 Constructor & Destructor Documentation	7
6.1.3 Member Function Documentation	8
6.1.4 Member Data Documentation	14
6.2 DiGraph.Edge Class Reference	15
6.2.1 Detailed Description	16
6.2.2 Constructor & Destructor Documentation	16
6.2.3 Member Function Documentation	17
6.2.4 Member Data Documentation	19
6.3 SocialGraph.SocialGraph Class Reference	19
6.3.1 Detailed Description	20
6.3.2 Constructor & Destructor Documentation	20
6.3.3 Member Function Documentation	21
6.3.4 Member Data Documentation	22
7 File Documentation	23
7.1 DiGraph.py File Reference	23
7.2 SocialGraph.py File Reference	23
Index	25

1 Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

DiGraph	3
SocialGraph	4

2 Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

object	
DiGraph.DiGraph	5
DiGraph.Edge	15
SocialGraph.SocialGraph	19

3 Class Index

3.1 Class List

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

DiGraph.DiGraph	
Class implementing a directed graph structure using a dictionary for holding vertices and a set for holding edges	5
DiGraph.Edge	
An edge holds the vertex it points to and its cost (or weight)	15
SocialGraph.SocialGraph	
Create a graph to model a social network of friendships	19

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

DiGraph.py	23
----------------------------	----

5 Namespace Documentation

5.1 DiGraph Namespace Reference

Classes

- class [DiGraph](#)
Class implementing a directed graph structure using a dictionary for holding vertices and a set for holding edges.
- class [Edge](#)
An edge holds the vertex it points to and its cost (or weight).

Functions

- def [cmp](#) (x, y)
Replacement for built-in function cmp that was removed in Python 3.
- def [main](#) (argv=None)
Main method.

5.1.1 Detailed Description

A very simple directed graph class.

Author

Paulo Roma.

Since

12/09/2018

See also

https://www.python-course.eu/graphs_python.php

5.1.2 Function Documentation

5.1.2.1 `cmp()`

```
def DiGraph.cmp (  
    x,  
    y )
```

Replacement for built-in function cmp that was removed in Python 3.

Compare the two objects x and y and return an integer according to the outcome.

Parameters

<i>x</i>	first object.
<i>y</i>	second object.

Returns

a negative value if $x < y$, zero if $x == y$ and strictly positive if $x > y$.

Referenced by `DiGraph.Edge.cmpCost()`.

5.1.2.2 main()

```
def DiGraph.main (
    argv = None )
```

Main method.

Creates a simple graph.

5.2 SocialGraph Namespace Reference

Classes

- class [SocialGraph](#)
Create a graph to model a social network of friendships.

Functions

- def [main](#) (args=None)
Create an empty graph aGraph.

5.2.1 Detailed Description

Manages a social network of friendships.

Author

Paulo Roma.

Since

13/09/2018

See also

<https://graphviz.readthedocs.io/en/stable/manual.html>
<http://www.it.usyd.edu.au/~shhong/directed.htm>

5.2.2 Function Documentation

5.2.2.1 main() `def SocialGraph.main (`
 `args = None)`

Create an empty graph aGraph.

Parse each line, print the entire line on the console, and call the corresponding method.

The command is case sensitive. Assume that the file format is correct.

add arg1 arg2 arg3

- call `aGraph.addEdge(arg1, arg2, int(arg3))`

showFriends arg1

- call `aGraph.adjacentTo(arg1)`

remove arg1

- call `aGraph.remove(arg1)`

recommendFriends arg1 arg2 arg3

- call the `recommendFriends(arg1, arg2, int(arg3))`, where:
 - arg1 is the name of the person to recommend new friends for.
 - arg2 is either "dist" or "weightedDist" indicating the method to select people to recommend as new friends.
 - int(arg3) is the maximum number of new friends to recommend.

Parameters

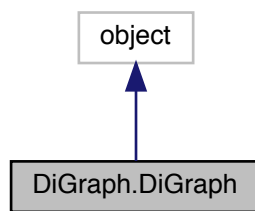
<code>args</code>	args[1] Input filename with all the commands and arguments.
-------------------	---

6 Class Documentation

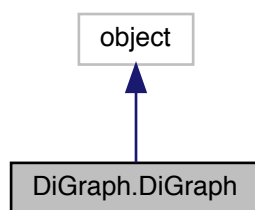
6.1 DiGraph.DiGraph Class Reference

Class implementing a directed graph structure using a dictionary for holding vertices and a set for holding edges.

Inheritance diagram for DiGraph.DiGraph:



Collaboration diagram for DiGraph.DiGraph:



Public Member Functions

- def `__init__` (self)
Create an empty graph.
- def `getInfinity` (self)
Returns the largest edge cost.
- def `addEdge` (self, src, dst, c=1)
Add a directed edge from the source node to the destination node.
- def `addVertex` (self, vertex)
Add a vertex to the graph with an empty set of edges associated with it.
- def `numVertices` (self)
Returns the number of vertices in this graph.
- def `vertices` (self)
Returns all vertices in this graph.
- def `adjacentTo` (self, vertex)
Gets all vertices adjacent to a given vertex.
- def `hasVertex` (self, vertex)
Checks whether a given vertex is in the graph.
- def `numEdges` (self)

- Returns the number of edges in this graph.*
- def [getEdge](#) (self, src, dst)
Gets the edge from src to dst, if such an edge exists.
- def [hasEdge](#) (self, src, dst)
Check whether an edge from src to dst exists.
- def [removeVertex](#) (self, vertex)
Remove this vertex from the graph if possible and calculate the number of edges in the graph accordingly.
- def [incomingEdges](#) (self, vertex)
Return a set of nodes with edges coming to this given vertex.
- def [__repr__](#) (self)
Return a representation of the graph as a string.
- def [Dijkstra](#) (self, source)
Compute Dijkstra single source shortest path from the source node.
- def [Dijkstra2](#) (self, source, dest)
Compute Dijkstra shortest path from the source to the destination node.

Public Attributes

- [graph](#)
A dictionary that stores an entry of a node, as the key, and a set of outgoing edges (destination node, weight) from the node, as its value.

Private Member Functions

- def [__generate_edges](#) (self)
A method generating the edges of the graph.

Private Attributes

- [__numEdges](#)
Total number of edges in the graph.
- [__infinity](#)
The largest edge distance.
- [__pathToNode](#)
Holds the path from a source node to a given node.
- [__dist](#)
Accumulated distance from source to a node.

6.1.1 Detailed Description

Class implementing a directed graph structure using a dictionary for holding vertices and a set for holding edges.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 `__init__()` `def DiGraph.DiGraph.__init__ (`
`self)`

Create an empty graph.

6.1.3 Member Function Documentation

6.1.3.1 `__generate_edges()` `def DiGraph.DiGraph.__generate_edges (`
`self) [private]`

A method generating the edges of the graph.

Edges are represented as tuples with one (a loop back to the vertex) or two vertices.

References `DiGraph.DiGraph.graph`.

Referenced by `DiGraph.DiGraph.__repr__()`.

6.1.3.2 `__repr__()` `def DiGraph.DiGraph.__repr__ (`
`self)`

Return a representation of the graph as a string.

References `DiGraph.DiGraph.__generate_edges()`, and `DiGraph.DiGraph.graph`.

6.1.3.3 `addEdge()` `def DiGraph.DiGraph.addEdge (`
`self,`
`src,`
`dst,`
`c = 1)`

Add a directed edge from the source node to the destination node.

If there is already existing edge from `src` to `dst`, replace the existing weight with the new weight.

Parameters

<i>src</i>	Source node.
<i>dst</i>	Destination node.
<i>c</i>	Weight of the edge.

Returns

- False if `src` or `dst` is None,

- or $c \leq 0$,
- or $src == dst$,
- True if a new edge from src to dst is added with the weight.

References `DiGraph.DiGraph.__numEdges`, `DiGraph.DiGraph.addVertex()`, `DiGraph.DiGraph.getEdge()`, `DiGraph.DiGraph.graph`, and `DiGraph.DiGraph.hasVertex()`.

6.1.3.4 addVertex() `def DiGraph.DiGraph.addVertex (`
 `self,`
 `vertex)`

Add a vertex to the graph with an empty set of edges associated with it.

Parameters

<i>vertex</i>	Vertex to be added.
---------------	---------------------

Returns

- False if $vertex$ is None or $vertex$ is already in the graph.
- True otherwise.

References `DiGraph.DiGraph.graph`, and `DiGraph.DiGraph.hasVertex()`.

Referenced by `DiGraph.DiGraph.addEdge()`.

6.1.3.5 adjacentTo() `def DiGraph.DiGraph.adjacentTo (`
 `self,`
 `vertex)`

Gets all vertices adjacent to a given vertex.

Parameters

<i>vertex</i>	
---------------	--

Returns

A set of vertices in which there is an edge from the given vertex to each of these vertices.

- An empty set is returned if there is no adjacent node,
- or the vertex is not in the graph,
- or vertex is None.

References `DiGraph.DiGraph.graph`, and `DiGraph.DiGraph.hasVertex()`.

Referenced by `DiGraph.DiGraph.Dijkstra()`.

6.1.3.6 Dijkstra() `def DiGraph.DiGraph.Dijkstra (`
 self,
 source)

Compute Dijkstra single source shortest path from the source node.

Parameters

<i>source</i>	Source node.
---------------	--------------

Returns

- Empty dictionary if the source is None,
- or it is not a vertex in the graph,
- or it does not have any outgoing edges.
- Otherwise, return a dictionary of entries, each having a vertex and smallest cost going from the source node to it.

See also

[Dijkstra's algorithm](#)

<https://docs.python.org/2/library/heapq.html>

<https://www.pythoncentral.io/priority-queue-beginners-guide/>

References `DiGraph.__dist`, `DiGraph.__pathToNode`, `DiGraph.adjacentTo()`, `DiGraph.getEdge()`, `DiGraph.getInfinity()`, `DiGraph.hasVertex()`, and `DiGraph.vertices()`.

Referenced by `DiGraph.Dijkstra2()`.

6.1.3.7 Dijkstra2() `def DiGraph.DiGraph.Dijkstra2 (`
 self,
 source,
 dest)

Compute Dijkstra shortest path from the source to the destination node.

Parameters

<i>source</i>	Source node
<i>dest</i>	Destination node

Returns

- Empty list if the source or dest are None,
- or they are not a vertex in the graph,
- or source does not have any outgoing edges.
- Otherwise, return a list of edges for reaching dest from source with the smallest cost.

See also

[Dijkstra's algorithm](#)

References `DiGraph.DiGraph.__pathToNode`, `DiGraph.DiGraph.Dijkstra()`, and `DiGraph.DiGraph.hasVertex()`.

6.1.3.8 getEdge() `def DiGraph.DiGraph.getEdge (`
 `self,`
 `src,`
 `dst)`

Gets the edge from `src` to `dst`, if such an edge exists.

Parameters

<i>src</i>	source vertex.
<i>dst</i>	target vertex.

Returns

- an edge if there exists an edge from `src` to `dst` regardless of the weight.
- None otherwise (including when either `src` or `dst` is None or `src` or `dst` is not in the graph).

References `DiGraph.DiGraph.graph`, and `DiGraph.DiGraph.hasVertex()`.

Referenced by `DiGraph.DiGraph.addEdge()`, `DiGraph.DiGraph.Dijkstra()`, `DiGraph.DiGraph.hasEdge()`, and `DiGraph.DiGraph.removeVertex()`.

6.1.3.9 getInfinity() `def DiGraph.DiGraph.getInfinity (`
 `self)`

Returns the largest edge cost.

References `DiGraph.DiGraph.__infinity`.

Referenced by `DiGraph.DiGraph.Dijkstra()`.

6.1.3.10 hasEdge() `def DiGraph.DiGraph.hasEdge (`
 `self,`
 `src,`
 `dst)`

Check whether an edge from `src` to `dst` exists.

Parameters

<i>src</i>	source vertex.
<i>dst</i>	target vertex.

Returns

- True if there exists an edge from src to dst regardless of the weight, and
- False otherwise (including when either src or dst is None or src or dst is not in the graph).

References `DiGraph.DiGraph.getEdge()`.

Referenced by `DiGraph.DiGraph.incomingEdges()`.

6.1.3.11 hasVertex() `def DiGraph.DiGraph.hasVertex (`
 self,
 vertex)

Checks whether a given vertex is in the graph.

Parameters

<i>vertex</i>	given vertex.
---------------	---------------

Returns

- True if the given vertex is in the graph.
- False otherwise, including the case of a None vertex.

References `DiGraph.DiGraph.graph`.

Referenced by `DiGraph.DiGraph.addEdge()`, `DiGraph.DiGraph.addVertex()`, `DiGraph.DiGraph.adjacentTo()`, `DiGraph.DiGraph.Dijkstra()`, `DiGraph.DiGraph.Dijkstra2()`, `DiGraph.DiGraph.getEdge()`, `DiGraph.DiGraph.incomingEdges()`, and `DiGraph.DiGraph.removeVertex()`.

6.1.3.12 incomingEdges() `def DiGraph.DiGraph.incomingEdges (`
 self,
 vertex)

Return a set of nodes with edges coming to this given vertex.

Parameters

<i>vertex</i>	given vertex.
---------------	---------------

Returns

- empty set if the vertex is None or the vertex is not in this graph.
- Otherwise, return a non-empty set consists of nodes with edges coming to this vertex.

References `DiGraph.DiGraph.hasEdge()`, `DiGraph.DiGraph.hasVertex()`, and `DiGraph.DiGraph.vertices()`.

Referenced by `DiGraph.DiGraph.removeVertex()`.

6.1.3.13 numEdges() `def DiGraph.DiGraph.numEdges (
 self)`

Returns the number of edges in this graph.

Returns

Total number of edges in this graph

References `DiGraph.DiGraph.__numEdges`.

6.1.3.14 numVertices() `def DiGraph.DiGraph.numVertices (
 self)`

Returns the number of vertices in this graph.

Returns

Number of vertices (nodes) in the graph.

References `DiGraph.DiGraph.graph`.

6.1.3.15 removeVertex() `def DiGraph.DiGraph.removeVertex (
 self,
 vertex)`

Remove this vertex from the graph if possible and calculate the number of edges in the graph accordingly.

For instance, if the vertex has 4 outgoing edges and 2 incoming edges, the total number of edges after the removal of this vertex is subtracted by 6.

Parameters

<code>vertex</code>	Vertex to be removed.
---------------------	-----------------------

Returns

- False if the vertex is None, or there is no such vertex in the graph.
- True if removal is successful.

References `DiGraph.DiGraph.__numEdges`, `DiGraph.DiGraph.getEdge()`, `DiGraph.DiGraph.graph`, `DiGraph.DiGraph.hasVertex()`, and `DiGraph.DiGraph.incomingEdges()`.

Returns

References DiGraph.DiGraph.graph.

Referenced by `DiGraph.DiGraph.Dijkstra()`, and `DiGraph.DiGraph.incomingEdges()`.

6.1.4.1 `__dist` `DiGraph.DiGraph.__dist` [private]

Referenced by `DiGraph.DiGraph.Dijkstra()`.

6.1.4.2 `__infinity` `DiGraph.DiGraph.__infinity` [private]

The largest edge distance.

```
self. infinity = sys.maxint
```

Referenced by `DiGraph.DiGraph.getInfinity()`.

6.1.4.3 numEdges DiGraph.DiGraph.__numEdges [private]

Total number of edges in the graph.

Referenced by `DiGraph.DiGraph.addEdge()`, `DiGraph.DiGraph.numEdges()`, and `DiGraph.DiGraph.removeVertex()`.

6.1.4.4 `__pathToNode` `DiGraph.DiGraph.__pathToNode` [private]

Holds the path from a source node to a given node.

Referenced by `DiGraph.DiGraph.Dijkstra()`, and `DiGraph.DiGraph.Dijkstra2()`.

6.1.4.5 graph `DiGraph.DiGraph.graph`

A dictionary that stores an entry of a node, as the key, and a set of outgoing edges (destination node, weight) from the node, as its value.

Referenced by `DiGraph.DiGraph.__generate_edges()`, `DiGraph.DiGraph.__repr__()`, `DiGraph.DiGraph.addEdge()`, `DiGraph.DiGraph.addVertex()`, `DiGraph.DiGraph.adjacentTo()`, `DiGraph.DiGraph.getEdge()`, `DiGraph.DiGraph.hasVertex()`, `DiGraph.DiGraph.numVertices()`, `DiGraph.DiGraph.removeVertex()`, and `DiGraph.DiGraph.vertices()`.

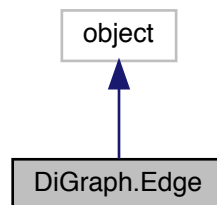
The documentation for this class was generated from the following file:

- [DiGraph.py](#)

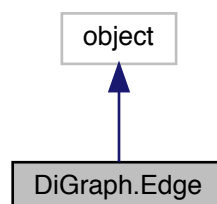
6.2 DiGraph.Edge Class Reference

An edge holds the vertex it points to and its cost (or weight).

Inheritance diagram for `DiGraph.Edge`:



Collaboration diagram for `DiGraph.Edge`:



Public Member Functions

- `def __init__ (self, n, c)`
Constructor from a node and its cost.
- `def getVertex (self)`
Get the target vertex.
- `def getCost (self)`
Get this edge cost.
- `def setCost (self, c)`
Set the cost of this edge.
- `def cmpCost (self, other)`
Compare two edges based on the cost only, not on the vertex.
- `def __repr__ (self)`
An unambiguous representaion of this edge.
- `def __hash__ (self)`
An [Edge](#) object must be hashable.
- `def __eq__ (self, obj)`
Operator ==.
- `def __contains__ (self, obj)`
Operator in.

Private Attributes

- `__node`
Node/vertex the edge points to.
- `__cost`
[Edge](#) cost.

6.2.1 Detailed Description

An edge holds the vertex it points to and its cost (or weight).

6.2.2 Constructor & Destructor Documentation

6.2.2.1 `__init__()` `def DiGraph.Edge.__init__ (`
`self,`
`n,`
`c)`

Constructor from a node and its cost.

Parameters

<code>n</code>	node/vertex.
<code>c</code>	weight or cost associated with the edge.

6.2.3 Member Function Documentation

6.2.3.1 `__contains__()` `def DiGraph.Edge.__contains__ (`
 `self,`
 `obj)`

Operator in.

Only compare the nodes but not the cost of the nodes.

6.2.3.2 `__eq__()` `def DiGraph.Edge.__eq__ (`
 `self,`
 `obj)`

Operator ==.

Only compare the nodes but not the cost of the nodes.

Parameters

<i>obj</i>	the edge for comparing this edge to.
------------	--------------------------------------

Returns

`(self.__node == obj.__node)`

References `DiGraph.Edge.__node`.

6.2.3.3 `__hash__()` `def DiGraph.Edge.__hash__ (`
 `self)`

An [Edge](#) object must be hashable.

Returns

`hash((self.__node, self.__cost))`

References `DiGraph.Edge.__cost`, and `DiGraph.Edge.__node`.

6.2.3.4 `__repr__()` `def DiGraph.Edge.__repr__ (`
 `self)`

An unambiguous representaion of this edge.

Returns

a string representation of this edge.

References `DiGraph.Edge.__cost`, and `DiGraph.Edge.__node`.

6.2.3.5 cmpCost() `def DiGraph.Edge.cmpCost (`
 self,
 other)

Compare two edges based on the cost only, not on the vertex.

Parameters

<i>other</i>	the edge for comparing this edge to.
--------------	--------------------------------------

Returns

-cmp(self.getCost(), other.getCost())

References DiGraph.cmp(), and DiGraph.Edge.getCost().

6.2.3.6 getCost() `def DiGraph.Edge.getCost (`
 self)

Get this edge cost.

Returns

the cost of this edge.

References DiGraph.Edge.__cost.

Referenced by DiGraph.Edge.cmpCost().

6.2.3.7 getVertex() `def DiGraph.Edge.getVertex (`
 self)

Get the target vertex.

Returns

the node this edge points to.

References DiGraph.Edge.__node.

6.2.3.8 setCost() `def DiGraph.Edge.setCost (`
 self,
 c)

Set the cost of this edge.

Parameters

<code>c</code>	given cost.
----------------	-------------

References `DiGraph.Edge.__cost`.

6.2.4 Member Data Documentation

6.2.4.1 `__cost` `DiGraph.Edge.__cost` [private]

[Edge](#) cost.

Referenced by `DiGraph.Edge.__hash__()`, `DiGraph.Edge.__repr__()`, `DiGraph.Edge.getCost()`, and `DiGraph.↔Edge.setCost()`.

6.2.4.2 `__node` `DiGraph.Edge.__node` [private]

Node/vertex the edge points to.

Referenced by `DiGraph.Edge.__eq__()`, `DiGraph.Edge.__hash__()`, `DiGraph.Edge.__repr__()`, and `DiGraph.↔Edge.getVertex()`.

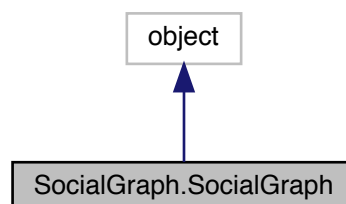
The documentation for this class was generated from the following file:

- [DiGraph.py](#)

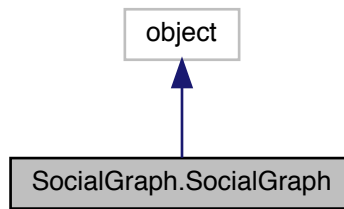
6.3 SocialGraph.SocialGraph Class Reference

Create a graph to model a social network of friendships.

Inheritance diagram for `SocialGraph.SocialGraph`:



Collaboration diagram for SocialGraph.SocialGraph:



Public Member Functions

- `def __init__ (self, file)`
Constructor from a file of commands.
- `def readGraphFile (self, file)`
Create a graph from a file.
- `def __str__ (self)`
Return a string representation of this graph.
- `def __repr__ (self)`
Return a graph description in the DOT language and save a png file with a graphical representation of this graph.
- `def recommendFriends (self, personOfInterest, option, topK)`
Recommend topK (e.g., 5) best friend candidates who are not already a friend of personOfInterest.

Public Attributes

- `dg`
Graph of friendship.

6.3.1 Detailed Description

Create a graph to model a social network of friendships.

See the homework description.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 __init__() `def SocialGraph.SocialGraph.__init__ (`
`self,`
`file)`

Constructor from a file of commands.

6.3.3 Member Function Documentation

6.3.3.1 `__repr__()` `def SocialGraph.SocialGraph.__repr__ (`
`self)`

Return a graph description in the DOT language and save a png file with a graphical representation of this graph.

References SocialGraph.SocialGraph.dg.

6.3.3.2 `__str__()` `def SocialGraph.SocialGraph.__str__ (`
`self)`

Return a string representation of this graph.

References SocialGraph.SocialGraph.dg.

6.3.3.3 `readGraphFile()` `def SocialGraph.SocialGraph.readGraphFile (`
`self,`
`file)`

Create a graph from a file.

References SocialGraph.SocialGraph.dg, and SocialGraph.SocialGraph.recommendFriends().

6.3.3.4 `recommendFriends()` `def SocialGraph.SocialGraph.recommendFriends (`
`self,`
`personOfInterest,`
`option,`
`topK)`

Recommend topK (e.g., 5) best friend candidates who are not already a friend of personOfInterest.

- If dist option is used, find the shortest path from personOfInterest to all the other nodes in the graph using Dijkstra's single source shortest path algorithm and friendship distances. The smaller the distance means the closer the relationship.
- If weightedDist option is used, after computing the shortest path like in the dist option to all the other nodes in the graph, multiply each distance with the total number of edges in the graph less the number of incoming edges to that node.

For instance, suppose the graph has a total of 10 edges.

Suppose the shortest distance from personOfInterest to node A is 5 and there are 4 incoming edges to A, the weighted distance is $5 \cdot (10 - 4) = 30$.

- The lower the weighted distance, the better the candidate.

This method considers both distance and popularity. The person with a lot of incoming edges means that the person is likely more well-liked by other people and should be recommended.

- Sort the distance/weighted distance in increasing order.
- If there are less than topK candidates, return only those candidates.
- If there are more than topK candidates, return only the topK candidates, when there are no other candidates with the same distance/weighted distance as the last candidate in the topK list.
- If there are other candidates with the same distance/weighted distance as the last candidate in the topK list, return all the candidates with the same distance. In this case, more than topK candidates are included in the list.

Parameters

<i>personOfInterest</i>	Name of the person to recommend new friend candidates for.
<i>option</i>	Either dist or weightedDist, which indicates whether to use the friendship distance or the weighted friendship distance.
<i>topK</i>	Desirable maximum number of candidate friends to recommend.

Returns

List of candidate friends.

References SocialGraph.SocialGraph.dg.

Referenced by SocialGraph.SocialGraph.readGraphFile().

6.3.4 Member Data Documentation

6.3.4.1 dg SocialGraph.SocialGraph.dg

Graph of friendship.

Referenced by SocialGraph.SocialGraph.__repr__(), SocialGraph.SocialGraph.__str__(), SocialGraph.SocialGraph.readGraphFile(), and SocialGraph.SocialGraph.recommendFriends().

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

- [SocialGraph.py](#)

7 File Documentation

7.1 DiGraph.py File Reference

Classes

- class [DiGraph.Edge](#)
An edge holds the vertex it points to and its cost (or weight).
- class [DiGraph.DiGraph](#)
Class implementing a directed graph structure using a dictionary for holding vertices and a set for holding edges.

Namespaces

- [DiGraph](#)

Functions

- def [DiGraph.cmp](#) (x, y)
Replacement for built-in function cmp that was removed in Python 3.
- def [DiGraph.main](#) (argv=None)
Main method.

7.2 SocialGraph.py File Reference

Classes

- class [SocialGraph.SocialGraph](#)
Create a graph to model a social network of friendships.

Namespaces

- [SocialGraph](#)

Functions

- def [SocialGraph.main](#) (args=None)
Create an empty graph aGraph.

Index

- `__contains__`
 - `DiGraph.Edge`, [17](#)
 - `__cost`
 - `DiGraph.Edge`, [19](#)
 - `__dist`
 - `DiGraph.DiGraph`, [14](#)
 - `__eq__`
 - `DiGraph.Edge`, [17](#)
 - `__generate_edges`
 - `DiGraph.DiGraph`, [8](#)
 - `__hash__`
 - `DiGraph.Edge`, [17](#)
 - `__infinity`
 - `DiGraph.DiGraph`, [14](#)
 - `__init__`
 - `DiGraph.DiGraph`, [7](#)
 - `DiGraph.Edge`, [16](#)
 - `SocialGraph.SocialGraph`, [20](#)
 - `__node`
 - `DiGraph.Edge`, [19](#)
 - `__numEdges`
 - `DiGraph.DiGraph`, [14](#)
 - `__pathToNode`
 - `DiGraph.DiGraph`, [14](#)
 - `__repr__`
 - `DiGraph.DiGraph`, [8](#)
 - `DiGraph.Edge`, [17](#)
 - `SocialGraph.SocialGraph`, [21](#)
 - `__str__`
 - `SocialGraph.SocialGraph`, [21](#)
- `addEdge`
 - `DiGraph.DiGraph`, [8](#)
- `addVertex`
 - `DiGraph.DiGraph`, [9](#)
- `adjacentTo`
 - `DiGraph.DiGraph`, [9](#)
- `cmp`
 - `DiGraph`, [3](#)
- `cmpCost`
 - `DiGraph.Edge`, [17](#)
- `dg`
 - `SocialGraph.SocialGraph`, [22](#)
- `DiGraph`, [3](#)
 - `cmp`, [3](#)
 - `main`, [4](#)
- `DiGraph.DiGraph`, [5](#)
 - `__dist`, [14](#)
 - `__generate_edges`, [8](#)
 - `__infinity`, [14](#)
 - `__init__`, [7](#)
 - `__numEdges`, [14](#)
 - `__pathToNode`, [14](#)
 - `__repr__`, [8](#)
 - `addEdge`, [8](#)
 - `addVertex`, [9](#)
 - `adjacentTo`, [9](#)
 - `Dijkstra`, [9](#)
 - `Dijkstra2`, [10](#)
 - `getEdge`, [11](#)
 - `getInfinity`, [11](#)
 - `graph`, [14](#)
 - `hasEdge`, [11](#)
 - `hasVertex`, [12](#)
 - `incomingEdges`, [12](#)
 - `numEdges`, [12](#)
 - `numVertices`, [13](#)
 - `removeVertex`, [13](#)
 - `vertices`, [13](#)
- `DiGraph.Edge`, [15](#)
 - `__contains__`, [17](#)
 - `__cost`, [19](#)
 - `__eq__`, [17](#)
 - `__hash__`, [17](#)
 - `__init__`, [16](#)
 - `__node`, [19](#)
 - `__repr__`, [17](#)
 - `cmpCost`, [17](#)
 - `getCost`, [18](#)
 - `getVertex`, [18](#)
 - `setCost`, [18](#)
- `DiGraph.py`, [23](#)
- `Dijkstra`
 - `DiGraph.DiGraph`, [9](#)
- `Dijkstra2`
 - `DiGraph.DiGraph`, [10](#)
- `getCost`
 - `DiGraph.Edge`, [18](#)
- `getEdge`
 - `DiGraph.DiGraph`, [11](#)
- `getInfinity`
 - `DiGraph.DiGraph`, [11](#)
- `getVertex`
 - `DiGraph.Edge`, [18](#)
- `graph`
 - `DiGraph.DiGraph`, [14](#)
- `hasEdge`
 - `DiGraph.DiGraph`, [11](#)
- `hasVertex`
 - `DiGraph.DiGraph`, [12](#)
- `incomingEdges`
 - `DiGraph.DiGraph`, [12](#)
- `main`
 - `DiGraph`, [4](#)
 - `SocialGraph`, [5](#)
- `numEdges`

- DiGraph.DiGraph, [12](#)
- numVertices
 - DiGraph.DiGraph, [13](#)
- readGraphFile
 - SocialGraph.SocialGraph, [21](#)
- recommendFriends
 - SocialGraph.SocialGraph, [21](#)
- removeVertex
 - DiGraph.DiGraph, [13](#)
- setCost
 - DiGraph.Edge, [18](#)
- SocialGraph, [4](#)
 - main, [5](#)
- SocialGraph.py, [23](#)
- SocialGraph.SocialGraph, [19](#)
 - __init__, [20](#)
 - __repr__, [21](#)
 - __str__, [21](#)
 - dg, [22](#)
 - readGraphFile, [21](#)
 - recommendFriends, [21](#)
- vertices
 - DiGraph.DiGraph, [13](#)