Specification

We shall define a class named **DirectedGraph** representing a *directed graph*.

The class **DirectedGraph** will provide the following methods:

DirectedGraph(int size = 0)
Constructs a graph with *size* vertices, the default size is 0

get_vertices(): int
Returns the number of vertices of the graph

get_edges(): int
Returns the number of edges of the graph

add_edge(int from_vertex, int to_vertex, int cost)
Adds an edge to the graph

parse_vertices(): list
Returns an iterable list of vertices

is_edge_defined(int having_start, int having_end): boolean
Returns true if an edge is defined by the start and end vertices and false otherwise

get_vertex_in_out(int vertex): (int, int)
Returns a pair of total in degree and out degree vertices for a given vertex

parse_vertex_in(int vertex): list
Returns an iterable list of in degree vertices for a given vertex

parse_vertex_out(int vertex): list
Returns an iterable list of out degree vertices for a given vertex

get_edge_cost(int having_start, int having_end): int
Returns the cost of a defined edge for a given start and end vertex

set_edge_cost(int having_start, int having_end, int cost)
Sets a given cost to a given edge defined by a start and end vertex

add_vertex()
Adds a new vertex

remove_edge(int having_start, int having_end)

Removes an edge defined by the given start and end of a vertex

remove_vertex(int vertex)

Removes the given vertex

get_copy(): DirectedGraph

Returns a copy of the **DirectedGraph** as a **DirectedGraph**

Implementation

The implementation uses 3 maps:

```
def __init__(self, size=0):
  self.store_from = {}
  self.store_to = {}
  self.store_cost = {}

  for v in range(size):
     self.store_from[v] = []
     self.store_to[v] = []
```

One map is used to store a list of “out vertices”, one is used to store a list of “in vertices” and one stores the cost of a pair of vertices. In the constructor, the vertices are initialized for the given size.

store_from:

key: vertex
value: list of vertices

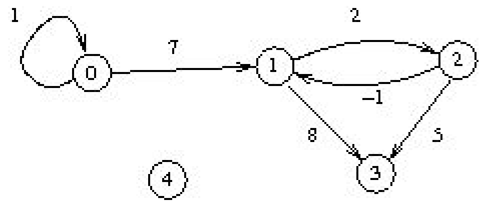
store_to:

key: vertex
value: list of vertices

store_cost:

key: pair of 2 vertices
value: cost

Example



store_from:

0: [0, 1]
1: [2, 3]
2: [1, 3]
3: []
4: []

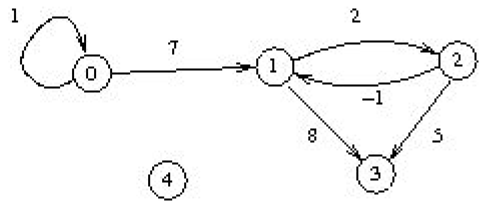
store_to:

0: [0]
1: [0, 2]
2: [1]
3: [1, 2]
4: []

store_cost:

(0, 0): 1
(0, 1): 7
(1, 2): 2
(1, 3): 8
(2, 1): -1
(2, 3): 5

Assignment 2



Example for exercise 2

Let’s suppose that there are 2 more connections: 0 to 4 and 4 to 1
If we call the function *breadth_first_search(source, target)* with source = 0 and target = 3, then the algorithm will start with node 3. It will search for all the inbound edges (1, 2 and 4), it will mark them as visited, compute the distance and add them to the queue. The next element from the queue is 1 and the same steps repeat (inbound edges being 0 and 2). The next element is 2

with the inbound edge 1. And the next one is 0 which is the number that we want to compute and the algorithm stops here.

Line	source	target	visited	distance	queue	root	v
L238	3	1	None	None	None	None	None
L240	3	1	[False, False, False, False, False]	None	None	None	None
L241	3	1	[False, False, False, False, False]	[SearchPair(0, []), SearchPair(0, []), SearchPair(0, []), SearchPair(0, []), SearchPair(0, [])]	None	None	None
L243	3	1	[False, False, False, False, False]	[SearchPair(0, []), SearchPair(0, []), SearchPair(0, []), SearchPair(0, []), SearchPair(0, [])]	[3]	None	None
L244	3	1	[False, False, False, True, False]	[SearchPair(0, []), SearchPair(0, []), SearchPair(0, []), SearchPair(0, []), SearchPair(0, [])]	[3]	None	None
L246	3	1	[False,	[SearchP	[3]	None	None

			False, False, True, False]	air(0, []), SearchPair(0, []), SearchPair(0, [3]), SearchPair(0, []), SearchPair(0, [])]			
L249	3	1	[False, False, False, True, False]	[SearchPair(0, []), SearchPair(0, []), SearchPair(0, []), SearchPair(0, [3]), SearchPair(0, [])]	[]	3	None
L251	3	1	[False, False, False, True, False]	[SearchPair(0, []), SearchPair(0, []), SearchPair(0, []), SearchPair(0, [3]), SearchPair(0, [])]	[]	3	1
L252	3	1	[False, True, False, True, False]	[SearchPair(0, []), SearchPair(0, []), SearchPair(0, []), SearchPair(0, [3]), SearchPair(0, [])]	[]	3	1
L253	3	1	[False, True, False, True,	[SearchPair(0, []), SearchPair(1, [3,	[]	3	1

			False]	1]), SearchPair(0, []), SearchPair(0, [3]), SearchPair(0, [])]			
L253	3	1	[False, True, False, True, False]	[SearchPair(0, []), SearchPair(1, [3, 1]), SearchPair(0, []), SearchPair(0, [3]), SearchPair(0, [])]	[1]	3	1
L251	3	1	[False, True, False, True, False]	[SearchPair(0, []), SearchPair(1, [3, 1]), SearchPair(0, []), SearchPair(0, [3]), SearchPair(0, [])]	[1]	3	2
L252	3	1	[False, True, True, True, False]	[SearchPair(0, []), SearchPair(1, [3, 1]), SearchPair(0, []), SearchPair(0, [3]), SearchPair(0, [])]	[1]	3	2
L253	3	1	[False, True, True,	[SearchPair(0, []), SearchP	[1]	3	2

			True, False]	air(1, [3, 1]), SearchPair(1, [3, 2]), SearchPair(0, [3]), SearchPair(0, [])]			
L254	3	1	[False, True, True, True, False]	[SearchPair(0, []), SearchPair(1, [3, 1]), SearchPair(1, [3, 2]), SearchPair(0, [3]), SearchPair(0, [])]	[1, 2]	3	2
L251	3	1	[False, True, True, True, False]	[SearchPair(0, []), SearchPair(1, [3, 1]), SearchPair(1, [3, 2]), SearchPair(0, [3]), SearchPair(0, [])]	[1, 2]	3	4
L252	3	1	[False, True, True, True, True]	[SearchPair(0, []), SearchPair(1, [3, 1]), SearchPair(1, [3, 2]), SearchPair(0, [3]), SearchP	[1, 2]	3	4

				air(0, [])]			
L253	3	1	[False, True, True, True, True]	[SearchPair(0, []), SearchPair(1, [3, 1]), SearchPair(1, [3, 2]), SearchPair(0, [3]), SearchPair(1, [3, 4])]	[1, 2]	3	4
L254	3	1	[False, True, True, True, True]	[SearchPair(0, []), SearchPair(1, [3, 1]), SearchPair(1, [3, 2]), SearchPair(0, [3]), SearchPair(1, [3, 4])]	[1, 2, 4]	3	4
L249	3	1	[False, True, True, True, True]	[SearchPair(0, []), SearchPair(1, [3, 1]), SearchPair(1, [3, 2]), SearchPair(0, [3]), SearchPair(1, [3, 4])]	[2, 4]	1	None
L251	3	1	[False, True, True,	[SearchPair(0, []), SearchP	[2, 4]	1	0

			True, True]	air(1, [3, 1]), SearchPair(1, [3, 2]), SearchPair(0, [3]), SearchPair(1, [3, 4])]			
L252	3	1	[True, True, True, True, True]	[SearchPair(0, []), SearchPair(1, [3, 1]), SearchPair(1, [3, 2]), SearchPair(0, [3]), SearchPair(1, [3, 4])]	[2, 4]	1	0
L253	3	1	[True, True, True, True, True]	[SearchPair(2, [3, 1, 0]), SearchPair(1, [3, 1]), SearchPair(1, [3, 2]), SearchPair(0, [3]), SearchPair(1, [3, 4])]	[2, 4]	1	0
L253	3	1	[True, True, True, True, True]	[SearchPair(2, [3, 1, 0]), SearchPair(1, [3, 1]), SearchP	[2, 4, 0]	1	0

				air(1, [3, 2]), SearchPair(0, [3]), SearchPair(1, [3, 4])]			
L251	3	1	[True, True, True, True, True]	[SearchPair(2, [3, 1, 0]), SearchPair(1, [3, 1]), SearchPair(1, [3, 2]), SearchPair(0, [3]), SearchPair(1, [3, 4])]	[2, 4, 0]	1	2
L249	3	1	[True, True, True, True, True]	[SearchPair(2, [3, 1, 0]), SearchPair(1, [3, 1]), SearchPair(1, [3, 2]), SearchPair(0, [3]), SearchPair(1, [3, 4])]	[4, 0]	2	None
L251	3	1	[True, True, True, True, True]	[SearchPair(2, [3, 1, 0]), SearchPair(1, [3, 1]), SearchPair(1, [3, 2]),	[4, 0]	2	1

				SearchPair(0, [3]), SearchPair(1, [3, 4])]			
L249	3	1	[True, True, True, True, True]	[SearchPair(2, [3, 1, 0]), SearchPair(1, [3, 1]), SearchPair(1, [3, 2]), SearchPair(0, [3]), SearchPair(1, [3, 4])]	[0]	4	None
L251	3	1	[True, True, True, True, True]	[SearchPair(2, [3, 1, 0]), SearchPair(1, [3, 1]), SearchPair(1, [3, 2]), SearchPair(0, [3]), SearchPair(1, [3, 4])]	[0]	4	0
L249	3	1	[True, True, True, True, True]	[SearchPair(2, [3, 1, 0]), SearchPair(1, [3, 1]), SearchPair(1, [3, 2]), SearchPair(0, [3]),	[]	0	None

				SearchPair(1, [3, 4])]			
L251	3	1	[True, True, True, True, True]	[SearchPair(2, [3, 1, 0]), SearchPair(1, [3, 1]), SearchPair(1, [3, 2]), SearchPair(0, [3]), SearchPair(1, [3, 4])]	[]	0	0
L256	3	1	[True, True, True, True, True]	[SearchPair(2, [3, 1, 0]), SearchPair(1, [3, 1]), SearchPair(1, [3, 2]), SearchPair(0, [3]), SearchPair(1, [3, 4])]	[]	None	None

Outputs of algorithm:

1 -> 100 from graph1k: [100, 624, 699, 175, 487, 5, 1] having the length of 6
100 -> 1 from graph1k: [1, 109, 865, 354, 416, 100] having the length of 5

1 -> 100 from graph10k: [100, 4722, 739, 1494, 690, 2404, 4118, 7317, 1] having the length of 8
100 -> 1 from graph10k: [1, 4260, 528, 4997, 1451, 2781, 5568, 100] having the length of 7

1 -> 100 from graph100k: [100, 14973, 32753, 4156, 3075, 14969, 27471, 17024, 1] having the length of 8

100 -> 1 from graph100k: [1, 58288, 98655, 95930, 53263, 6606, 54527, 44340, 100] having the length of 8