INDIAN INSTITUTE OF TECHNOLOGY ROPAR
Lab on Graphs

You need to implement in C/C++ - 1) BFS, 2) DFS, 3) Topological sorting, 4) Dijkstra using adjacency list representation, and 5) Bellman Ford using adjacency list representation

Input Format

First Line contains **T**, the number of Test cases/Queries

First line of each Test-case mentions the Query id, **Q** i.e. either 1 or 2 or 3 or 4 or 5. Here 1 is for BFS, ..., 4 is for Shortest path using Dijkstra, and 5 is Bellman Ford

If query id **Q** is **3** , the Second line of test-case mentions **N** i.e. number of nodes in directed graph. This is then followed by **N** lines each containing **N** integers (basically **N**x**N** Adjacency matrix representation of directed graph). Note that Graph certainly would be directed graph for Topological sorting query

If query id **Q** is **1, 2, 4 or 5**, the Second line of test-case mentions **N D s** where N denotes the number of nodes in graph, **D** denotes whether graph is directed or not (D=1 means graph is directed, else undirected) and **s** indicates the label of source node. This is then followed by **N** lines each containing **N** integers (basically **N**x**N** Adjacency matrix representation of undirected/directed graph).

Note the following for Shortest path problems i.e when **Q=4 or 5,**
the NxN matrix values / integers denote the edge-weights, there are no self loops in the graph, and weight=999999 indicates there is no edge/path between those concerned vertices.

Output Format

T rows (a row for a test-case/query)

For Q=1 and D=1, print (single space separated)
No. of nodes identified at distance 1, 2, ... until you mention 0 (from vertex s using BFS),
then number of BFS tree edges, no. of backward edges, forward edges and cross edges.
For Q=1 and D=0, print (single space separated)
No. of nodes identified at distance 1, 2, ... until you mention 0 (from vertex s using BFS),
then number of BFS tree edges and no. of cross edges.

For Q=2 and D=1, print (single space separated)
Finishing time of source vertex s (assume discovery times begins from 1 and Adj list vertices explored in increasing label values),then number of DFS tree edges, no. of backward edges, forward edges and cross edges.
For Q=2 and D=0, print (single space separated)
Finishing time of source vertex s (assume discovery times begins from 1 and Adj list vertices explored in increasing label values), then number of DFS tree edges and no. of backward edges

For Q=3, print (single space separated)
list the topological sorted sequence (lexicographically smallest one) if it exists, else print -1

For Q=4, print (single space separated)
N entries with each entry indicating shortest path to that vertex from the given source vertex.
print 999999 if path don't exist and print -1 if dijkstra not applicable or shortest path not defined.

For Q=5, print (single space separated)
-1 if there is found negative weight cycle ELSE print
N+2 entries where first N entries indicate shortest path to that vertex from the given source vertex (print 999999 if path don't exist)
and then last two entries that correspond to (i) No. of relax_edge operations performed and (ii) Number of relax_edge operations that brought any modification. (Algorithms that minimize these entries would be more preferred)

Constraints

0 ≤ T ≤ 2000
1 ≤ Q ≤ 5
0 ≤ N ≤ 200
0 ≤ D ≤ 1
1 ≤ s ≤ N
-999999 ≤ Adj matrix values ≤ 999999 (for Q = 5)

You cannot use any inbuilt data structure – heap, queue, stack…Implement on your own. Your code shall be compatible to run on linux system.

---- Sample Input -----

```
13
1
3 0 1
0 1 1
1 0 1
1 1 0
1
3 0 1
0 1 0
1 0 1
0 1 0
1
5 0 1
0 1 0 1 0
1 0 1 0 1
0 1 0 0 1
1 0 0 0 1
0 1 1 1 0
1
9 0 1
0 1 0 0 0 0 1 1 0
1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0
0 0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 0 1
0 0 0 1 1 0 1 0 0
1 0 1 0 0 1 0 0 0
1 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0
1
4 1 1
0 1 1 0
0 0 0 1
0 0 0 0
0 0 1 0
2
4 1 1
```

0 1 1 0
0 0 0 1
0 0 0 0
0 0 1 0
2
3 0 1
0 1 0
1 0 1
0 1 0
2
3 0 1
0 1 1
1 0 1
1 1 0
3
4
0 1 1 0
0 0 0 1
0 0 0 0
0 0 1 0
4
4
0 2 10 999999
999999 0 999999 4
999999 999999 0 999999
999999 999999 2 0
4
4
0 2 999999 4
999999 0 999999 1
4 2 0 999999
999999 1 999999 0
4
4 1 1
0 2 10 999999
999999 0 999999 4
999999 999999 0 999999
-10 999999 2 0
4
3 0 1
0 1 1
1 0 1
1 1 0

----- Sample Output----------

2 0 2 1
1 1 0 2 0

2 2 0 4 2
3 2 2 1 0 8 0
2 1 0 3 0 0 1
8 3 0 1 0
6 2 0
6 2 1
1 2 4 3
0 2 8 6
0 2 999999 3
-1
0 1 1

Amongst correctly functioning submitted programs, better performing ones will get more marks

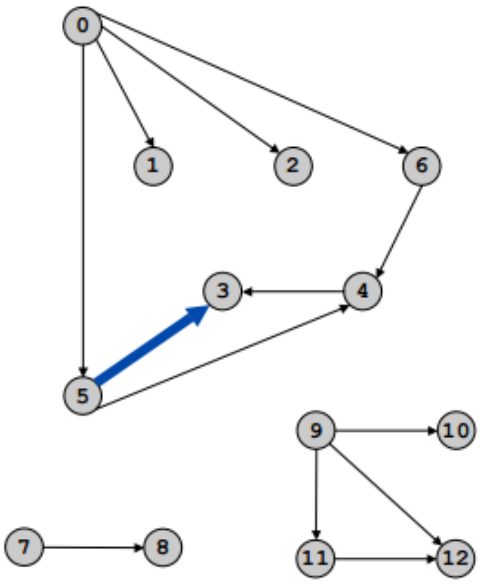
==========
Feel free to ask any clarification

Instructions regarding submission:
- Single .c file / .cpp file
-Clearly mention in Comments which query type work correctly and which one you fail to implement
- Input outputs that you tried or used for testing your code can be included in your program file at end within comments (using /* */)

http://www.cs.princeton.edu/courses/archive/fall07/cos226/lectures/digraph.pdf

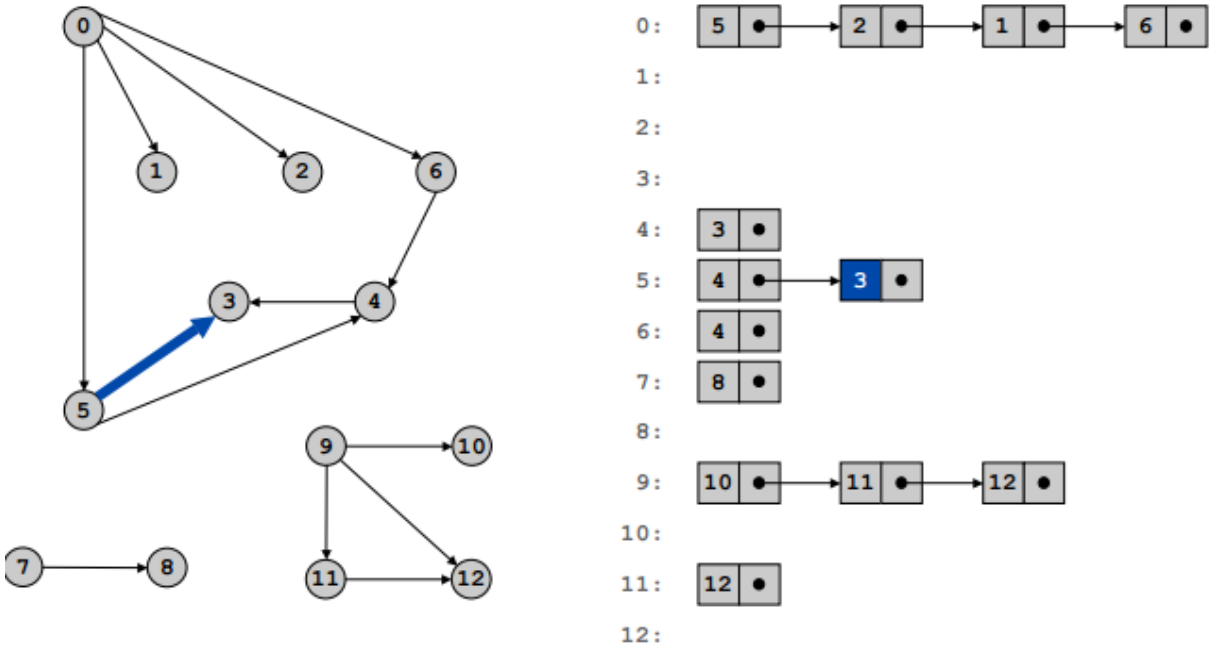
Adjacency matrix representation.

- Two-dimensional V × V boolean array.
- Edge v→w in graph: `adj[v][w] = true.`

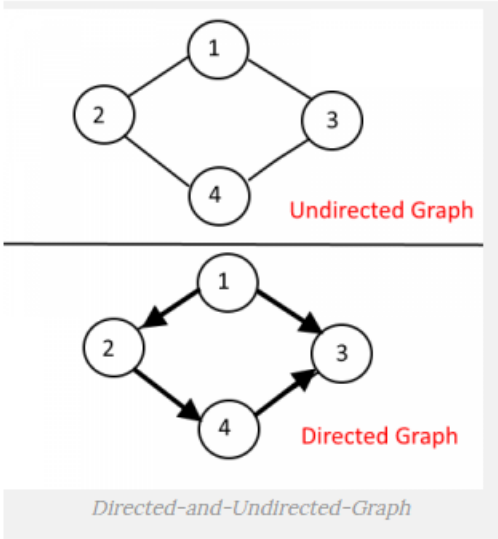


from \ to	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0	0	0	0	0
5	0	0	0	1	1	0	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	1
12	0	0	0	0	0	0	0	0	0	0	0	0	0

Adjacency list representation. Vertex indexed array of lists.



- Un-directed Graph — when you can traverse either direction between two nodes.
- Directed Graph — when you can traverse only in the specified direction between two nodes.



Directed-and-Undirected-Graph

	1	2	3	4
1	0	1	1	0
2	1	0	0	1
3	1	0	0	1
4	0	1	1	0

Undirected Graph

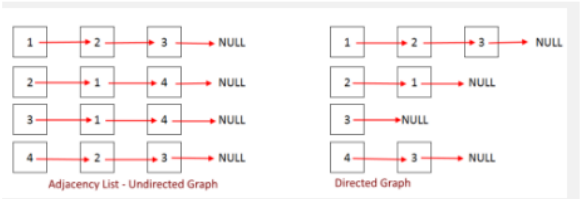
	1	2	3	4
1	0	1	1	0
2	0	0	0	1
3	0	0	0	0
4	0	0	1	0

Directed Graph

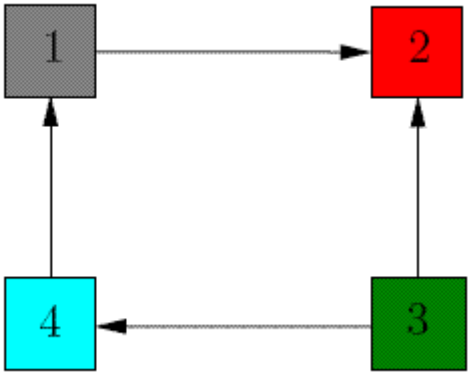
http://algorithms.tutorialhorizon.com/graph-representation-adjacency-matrix-and-adjacency-list/

Adjacency List:

Adjacency List is the Array[] of Linked List, where array size is same as number of Vertices in the graph. Every Vertex has a Linked List. Each Node in this Linked list represents the reference to the other vertices which share an edge with the current vertex. The weights can also be stored in the Linked List Node.



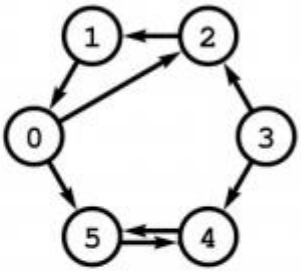
Adjacency-List



$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

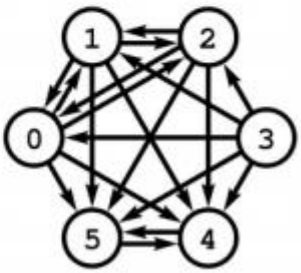
Transitive closure. Is there a directed path from v to w?

G



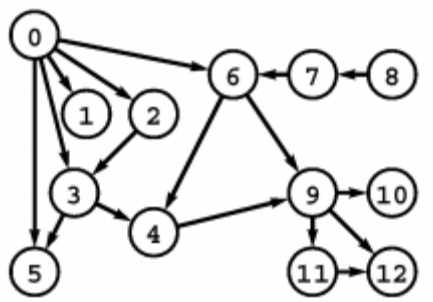
	0	1	2	3	4	5
0	1	0	1	0	0	1
1	1	1	0	0	0	0
2	0	1	1	0	0	0
3	0	0	1	1	1	0
4	0	0	0	0	1	1
5	0	0	0	0	1	1

Transitive closure

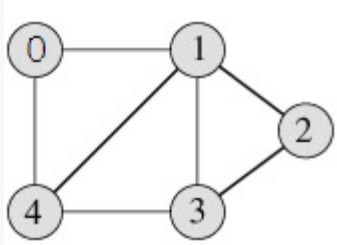


	0	1	2	3	4	5
0	1	1	1	0	1	1
1	1	1	1	0	1	1
2	1	1	1	0	1	1
3	1	1	1	1	1	1
4	0	0	0	0	1	1
5	0	0	0	0	1	1

tc[v][w] = 1 iff path from v to w

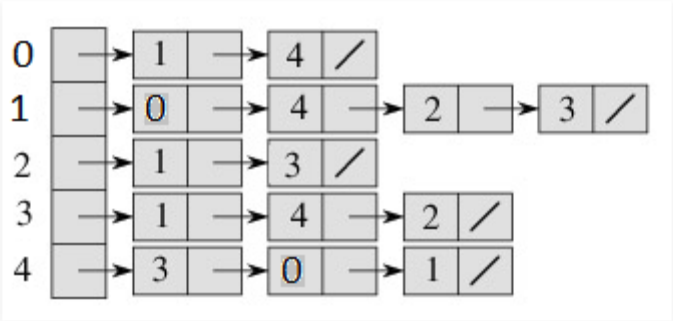


http://www.geeksforgeeks.org/graph-and-its-representations/



	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

Adjacency Matrix Representation of the above graph



Adjacency List Representation of the above Graph

http://www.sanfoundry.com/c-program-represent-graph-adjacency-matrix/

https://www.khanacademy.org/computing/computer-science/algorithms/graph-representation/a/representing-graphs