Graph Sets

Warning!!! Read Moodle's A14 PDF for a more complete description. The below description might be incomplete.

You are given a set of graphs G_0, G_1, ... , G_{K-1} where G_k = (V_k , E_k), k = {0, ... , K-1} is either a weighted,directed graph(Graph class) with edge weights w^k_{uv} or an unweighted directed graph(BooleanGraph). The nodes in the graph can have arbitrary node ids (any valid non-negative integer).

Classes

You have to implement a Graph class with the following functions/operators:

• Operator overloading for binary operator + : G = G_i + G_j
Given G_i = (V_i , E_i) and G_j = (V_j , E_j).
After the operation, G = (V_i ∪ V_j , E) , where E = { (u, v, w_{uv}) : ∀u, v ∈ V_i ∪ V_j , w_{uv} = w^i_{uv} + w^j_{uv} }.
An edge exists between nodes u and v in the new graph, only if $w_{uv} \neq 0$.

• Operator overloading for binary operator −: G = G_i − G_j
Given G_i = (V_i , E_i) and G_j = (V_j , E_j).
After the operation, G = (V_i ∪ V_j , E) , where E = { (u, v, w_{uv}) : ∀u, v ∈ V_i ∪ V_j , w_{uv} = w^i_{uv} - w^j_{uv} }.
An edge exists between nodes u and v in the new graph, only if $w_{uv} \neq 0$.

• Operator overloading for unary operator −: G = − G_i
Given G_i = (V_i , E_i).
After the operation, G = (V_i , E) , where E = { (v, u, w_{uv}) : ∀(u, v, w_{uv})) ∈ E_i }, i.e., reverse the direction of the edges in G_i .

• Operator overloading for operator [] : l = $G_i[u]$
Given G_i = (V_i , E_i), $G_i[u]$ should return the list of nodes adjacent to u.

NOTE: For the binary operators above, in case an edge exists only in one of them, you may assume zero weight for the other graph for that edge (see illustrations later).

From the general Graph class, derive a new class called BooleanGraph. The property of this class is that all the edges are directed but unweighted.

You have to implement the following functions/operators for this class:

• Operator overloading for binary operator + : G = G_i + G_j
Given G_i = (V_i , E_i) and G_j = (V_j , E_j).
After the operation, G = (V_i ∪ V_j , E_i ∪ E_j), i.e., an edge exists between nodes u and v in the new graph, if there is an edge between them in any of G_i or G_j.

• Operator overloading for binary operator − : G = G_i − G_j
Given G_i = (V_i , E_i) and G_j = (V_j , E_j).
After the operation, G = (V_i ∪ V_j , E_i − E_j), i.e., remove all edges from G_i which were also present in G_j(similar to set difference)

Queries

There are two types of queries, viz. operation query and print query. For operation query you do not have to print. But, you will have to modify your data structures. For print query you need to output accordingly as below.

Operation Queries

• ADD i j k : It means $G_k = G_i + G_j$. Perform "+" operation on G_i & G_j and replace graph G_k.

• SUB i j k : It means $G_k = G_i - G_j$. Perform "-" operation on G_i & G_j and replace graph G_k.

• NEG i k : It means $G_k = -G_i$. Perform unary "-" operation on G_i and replace graph G_k.

Note: There is a guarantee that G_i , G_j and G_k are same type of graphs.

Print Queries

• WEIGHT u v i : Print edge weight w^i_{uv} i.e edgeweight of (u, v) in graph G_i. If G_i is a BooleanGraph,then print 1 if the edge exists otherwise 0

• ADJOUT u i : In G_i, print the outward adjacent nodes(in sorted order) for the node u.

• DEGSEQ u i : In G_i, for all the out-adjacent nodes(in sorted order) of u , print its out degree.

• DFS u i : In G_i, start the DFS from u and print the nodes of the DFS tree as you visit.
Whenever choosing the neighbour nodes choose the least node id first and so on.

• NCOMP i : (To simplify matters and with slight abuse of definition) Print the number of weakly connected components in graph G_i. A Weakly connected component is a maximal subgraph of a directed graph such that every pair (u, v) , there is an undirected path from u to v . In simple words, the directed graph is weakly connected if it's underlying graph is connected.

Note: Refer sample testcases and diagrams for more clarity. In each of the print queries, use the current G_i . Graph G_i may have undergone changes during operation query.

Input Format

K Q // There are K graphs followed by Q queries.
V_0 E_0 $IsBOOL_0$ // $IsBOOL_i$ = 1 if G_i is a BooleanGraph.
v^0_1 v^0_2 ...$v^0_{v_0}$ // V_0 space-separated integers denoting the node ids of G_0.
u v W^0_{uv} // Next E_0 lines listing the weighted edges of the graph G_0 .
... // Weights will only be given if the graph is not a boolean graph.
... // and so on.
V_{K-1} E_{K-1} $IsBOOL_{K-1}$
... // V_{K-1} space-separated integers denoting the node ids for G_{K-1}.
... // Next E_{K-1} lines listing the weighted edges of the graph G_{K-1} .
...
// Final Q lines can be any of the following queries.
ADD i j k // $G_k = G_i + G_j$
SUB i j k // $G_k = G_i - G_j$
NEG i k // $G_k = - G_i$
WEIGHT u v i // Print weight W^i_{uv} in G_i
ADJOUT u i // $G_i[u]$
DEGSEQ u i // Print degree sequence of node u in G_i
DFS u i // Print the DFS traversal from node u on G_i
NCOMP i // Print number of weakly connected components in G_i

Constraints

1 ≤ K ≤ 1000
1 ≤ Q ≤ 1000
W^k_{uv} ∈ Z − { 0 }
0 ≤ u, v < INTMAX
1 ≤ i,j,k ≤ K

Output Format

Following is the expected output of the print queries:

• WEIGHT u v i : Print the current edge weight w^i_{uv} of G_i.

• ADJOUT u i : Print the current node ids (in sorted order) given by $G_i[u]$ with space separation in a single line. If there are no outward adjacent nodes for u in G_i, print an empty line.

• DEGSEQ u i : Print the out degree of nodes present in N(u) (following sorted order of node ids), with single space separation in a single line. N(u) is the list of adjacent nodes of u in G_i. If there are no outward adjacent nodes for u in G_i , print an empty line.

• DFS u i : Print the space-separated node ids of DFS traversal from node u on current G_i.

• NCOMP i : Print a single number, the number of components in current G_i as per our definition

Note: Graph G_i may have undergone changes during an operation query. So, use the current G_i in every step.

Sample Input 0

```
11 22
5 4 0
5 4 2 6 7
2 4 1
7 5 4
5 6 3
4 5 2
5 4 0
3 4 9 5 6
3 4 1
4 5 2
9 5 4
5 6 3
5 4 0
10 9 8 13 16
10 9 2
10 13 3
10 8 11
13 16 8
4 6 0
10 9 8 7
10 9 2
10 8 3
10 7 11
7 8 8
9 8 8
9 7 5
5 3 1
2 4 6 5 7
5 6
2 4
4 5
5 3 1
4 5 3 9 6
4 5
9 5
5 6
3 2 0
200 0 63
```

```
200 63 -9
200 0 -91
2 1 0
0 63
0 63 -9
1 0 0
3
3 0 1
0 3 56
2 1 1
3 56
56 3
NCOMP 10
NCOMP 9
ADD 4 5 9
SUB 4 5 10
NCOMP 10
NCOMP 9
NCOMP 7
ADD 0 1 6
SUB 0 1 7
NCOMP 7
WEIGHT 4 5 9
WEIGHT 4 5 0
NEG 1 8
WEIGHT 6 5 8
DEGSEQ 10 3
DEGSEQ 4 9
ADJOUT 10 2
ADJOUT 8 2
ADJOUT 2 10
ADJOUT 3 10
DFS 10 2
DFS 2 9
```

Sample Output 0

```
1
3
6
3
1
3
1
2
3
1 0 2
1
8 9 13

4

10 8 9 13 16
2 4 5 6
```

Sample Input 1

```
3 13
4 6 0
4 0 1 2
1 2 3
0 1 3
2 4 5
4 2 5
0 2 3
4 0 1
1 0 0
4
2 1 0
4 3
3 4 5
ADJOUT 4 2
DFS 4 1
DEGSEQ 1 0
DEGSEQ 4 2
DFS 4 1
ADD 1 2 0
DEGSEQ 4 1
```

```
ADJOUT 4 0
ADJOUT 3 2
WEIGHT 3 4 0
WEIGHT 3 4 0
WEIGHT 3 4 2
SUB 1 0 2
```

Sample Output 1

```
4
1

4


4
5
5
5
```

Sample Input 2

```
3 15
6 13 0
2 3 8 0 9 4
3 2 4
8 2 6
9 2 4
9 8 10
8 0 10
9 0 -5
3 0 6
4 2 4
0 3 4
8 4 2
3 4 5
0 2 10
4 0 2
3 3 0
7 0 8
7 0 -5
0 7 7
8 7 5
5 10 0
6 7 4 0 3
6 4 -2
7 0 3
4 6 -4
3 0 -7
7 6 2
6 0 -3
7 4 9
3 6 6
4 3 8
3 4 -2
SUB 1 0 2
DEGSEQ 0 1
WEIGHT 9 2 0
DEGSEQ 0 1
ADJOUT 0 2
DFS 0 1
ADD 1 2 0
ADJOUT 0 1
DFS 7 0
DFS 2 2
ADJOUT 4 0
DEGSEQ 8 2
DFS 3 0
WEIGHT 7 0 1
WEIGHT 3 4 2
```

Sample Output 2

```
1
4
1
```

```
2 3 7
0 7
7
7 0 2 3 4
2
0 2
3 0 2 1
3 0 2 7 4
-5
-5
```