Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Departement of Computer Science
Markus Püschel, David Steurer

21. October 2018

Datenstrukturen & Algorithmen Blatt P5 HS 17

Please remember the rules of honest conduct:

- Programming exercises are to be solved alone
- Do not copy code from any source
- Do not show your code to others

Hand-in: Sunday, 04. November 2018, 23:59 clock via Online Judge (source code only).
Questions concerning the assignment will be discussed as usual in the forum.

Exercise P5.1 *Skier.*

Sarah is a passionate skier, visiting Zermatt for a weekend trip. The ski resort at Zermatt has multiple mountain peaks, many ski trails and fast ski-lifts. By looking at the map, Sarah realizes that the mountain peaks are interconnected, in two ways: either there is a ski-trail that goes downhill from one mountain peak to the other, or a ski-lift that goes uphill. But unfortunately, not all two mountain peaks are directly connected with a ski-trail or a ski-lift at all, and in some cases two mountain peaks are directly connected both ways - with a ski-trail downhill and a ski-lift uphill.

To be able to take the best of the weekend trip and see much of Zermatt, Sarah likes to find whether there is at least one mountain peak that can reach all other mountain peaks.

Note that in this context, we define reachability as either having a direct connection from one peak to the other, or indirect connection though a set of one or more peaks in between. Both the direct and indirect connections can be either though a ski-lift or ski-trail, or any combination of the two.

Input The input consists of a set of instances, or *test-cases*, of the previous problem. The first line of the input contains the number T of test-cases. The first line of each test-case are the integers V and E, where V corresponds to number of mountain peaks in the ski-resort, and E corresponds to the number of connections in the ski-resort. The next E lines describe the connection in the ski resort. In particular, the $(n + 2)$-th line of the test-case contains two integers i and j, that describe a directed connection from mountain peak i to j. Note that mountain peaks do not have names, but are enumerated with numbers such that the first mounting peak starts at 0: $0, 1, \dots V - 1$. We constrain V and E such that $1 \leq V \leq 1000$ and $0 \leq E \leq \frac{V \cdot (V-1)}{2}$

Output The output consists of T lines, each containing either *Yes* or *No*, depending on whether there exist a mounting peak that can reach all other mountain peaks.

Grading You get 3 bonus points if your program works for all inputs. Your algorithm should have an asymptotic time complexity of $O(V + E)$ with reasonable hidden constants. Submit your `Main.java` at `https://judge.inf.ethz.ch/team/websubmit.php?cid=25012&problem=AD18H5P1`. The enrolment password is “`asymptotic`”.

Example

Input:

```
2
7 8
0 1
0 2
1 3
4 1
6 4
6 0
5 2
5 6
3 2
0 1
2 1
```

Output:

```
Yes
No
```

Notes For this exercise we provide an archive on the lecture website, available at `https://www.cadmo.ethz.ch/education/lectures/HS18/DA/uebungen/AD18H5P1.Skier.zip` containing a program template that will load the input and write the output for you. The archive also contains additional test cases (which differ from the ones used for grading). Importing any additional Java class is **not allowed** (with the exception of the already imported `java.util.{Scanner, LinkedList, Iterator, Stack}`, as well as `java.io.{InputStream, OutputStream}` classes).

Solution In this problem, we are looking for a mountain peak that can reach all other mountain peaks. In other words, we are looking for a vertex v_i that can reach all other vertices $v_1, v_2, \ldots, v_V$ assuming there is a path from v_i to all $v_1, v_2, \ldots, v_V$. This is the well-known problem of finding a *mother vertex* in a graph. We can solve this problem using depth first search (DFS) algorithm, as shown below:

```
// Create a structure to keep track of visited vertices
boolean visited[] = new boolean[V];
for (int i = 0; i < V; i += 1) visited[i] = false;
// Spawn a DFS from each unvisited vertex and keep track of the ones visited
int v = -1;
for (int i = 0; i < V; i += 1) if (!visited[i]) {
  DFS(adj, visited, i);
  v = i;
}
// v is now candidate for a mother vertex, so spawn another DFS to validate
for (int i = 0; i < V; i += 1) visited[i] = false;
DFS(adj, visited, v);
// check whether all other vertices have been visited
boolean isMotherVertex = true;
for (int i = 0; i < V; i++) { isMotherVertex = isMotherVertex && visited[i]; }
```

The algorithm works in two steps:

1. Performs DFS traversal of the given graph. While doing traversal, it keeps track of the visited vertices and also keeps track of the last finished vertex v. This step takes $O(V + E)$ time, as the algorithm is not visiting the vertices that have already been visited before.
2. If there exist mother vertex (or vertices), then v must be one (or one of them). Check if v is a mother vertex by doing DFS starting from v. This step also takes $O(V + E)$ time.

If there a exist mother vertex (or vertices), then one of the mother vertices is the last finished vertex in DFS (a vertex is said to be finished in DFS if an iterative (or recursive) call for its DFS is over, i.e., all descendants of the vertex have been visited).

Correctness

Let the last finished vertex be v. We need to prove that there cannot be an edge from another vertex u to v if u is not another mother vertex (Or there cannot exist a non-mother vertex u such that $u \to v$ is an edge). There can be two possibilities.

1. DFS call is made for u before v. If an edge $u \to v$ exists, then v must have finished before u because v is reachable through u and a vertex finishes after all its descendants.
2. DFS call is made for v before u. In this case also, if an edge $u \to v$ exists, then either v must finish before u (which contradicts our assumption that v is finished at the end) or u should be reachable from v (which means u is another mother vertex).

Thus v must be a mother vertex.

Exercise P5.2 *Submatrix Sum.*

You are given a $n \times n$ matrix $M = (m_{i,j})$ in which each entry $m_{i,j}$ with $1 \le i, j \le n$ is an integer between 0 and 1000 (rows and columns are numbered from 1 to n, from top-left to bottom-right). Your task is to design a data structure that, after *preprocessing* the matrix M, is able to support the following *query* operation: *Given $a, b, c, d \in \mathbb{Z}$ with $1 \le a \le b \le n$ and $1 \le c \le d \le n$, return*

$$S(a, b, c, d) = \sum_{\substack{a \le i \le b \\ c \le j \le d}} m_{i,j}.$$

Input The first line of the input contains the integer n. Each of the following n lines is one row of M. More precisely, the $(i+1)$-th line of the input contains the n integers $m_{i,1}, \dots, m_{i,n}$. The $(n+2)$-th line of the input contains the number m of queries to be answered and the i-th of the following m lines $(1 \le i \le m)$ contains four integers a_i, b_i, c_i, d_i.

Output

The output consists of m lines, where the i-th line contains the answer to the i-th query, i.e., the number $S(a_i, b_i, c_i, d_i)$.

Grading This exercise rewards no bonus points. Your algorithm should require time $O(n^2)$ preprocessing time and it should answer each query in constant time. Submit your `Main.java` at `https://judge.inf.ethz.ch/team/websubmit.php?cid=25012&problem=AD18H5P2`. The enrollment password is “`asymptotic`”.

Example

Input (corresponding to the matrix in Table 1):

5	3	1	5	0
8	0	4	3	6
1	6	1	5	1
0	7	9	1	7
4	5	8	8	3

Table 1: Example of matrix M with $n = 5$.

```
5
5 3 1 5 0
8 0 4 3 6
1 6 1 5 1
0 7 9 1 7
4 5 8 8 3
3
1 4 2 5
4 5 2 4
2 2 3 3
```

Output:

```
59
38
4
```

Notes For this exercise we provide an archive on the lecture website, available at `https://www.cadmo.ethz.ch/education/lectures/HS18/DA/uebungen/AD18H5P2.SubMatrixSum.zip`. The archive also contains additional test cases (which differ from the ones used for grading). Importing any additional Java class is **not allowed** (with the exception of the already imported `java.util.Scanner` as well as `java.io.{InputStream, OutputStream}` class).

Solution We want to pre-compute in $O(n^2)$ time all the values $q_{i,j} = \sum_{h=1}^{i} \sum_{k=1}^{j} m_{i,j}$ for $0 \leq i, j \leq n$. This can be done by noticing that, for all $0 < i, j \leq n$:

$$q_{i,j} = a_{i,j} + q_{i-1,j} + q_{i,j-1} - q_{i-1,j-1},$$

where $q_{0,j}$ and $q_{i,0}$ are equal to 0 by definition.

Once all the values $q_{i,j}$ have been computed, it is possible to answer a query in constant time. Indeed, we have:

$$S(a, b, c, d) = \sum_{\substack{a \leq i \leq b \\ c \leq j \leq d}} m_{i,j} = q_{b,d} - q_{a-1,d} - q_{b,c-1} + q_{a-1,c-1}.$$