Task 3: Sending Graph

Time Limit: 2 seconds
Memory Limit: 1024 MB

Problem Statement

Alice has a undirected simple graph[1] that has N nodes and M edges. Alice wants to send this graph G to Bob. She can do so by sending another simple, undirected graph G with V nodes and U edges to Bob.
Note that $V \leq 1500$, i.e. you can only send at most a graph with 1500 nodes.

However, during the sending process, the graph is relabelled. In particular, this means that

1. The labels of all V nodes are shuffled
2. The order of the U edges are shuffled
3. If the edge was given as $a \to b$, it may be given in the order of $b \to a$
4. Note that the graph Bob receives is isomorphic to G. I.e. the shape of the graph remains the same

You may refer to the appendix for a formal description of this procedure.

Implementation

You should submit the file `alicebob.cpp` including the header `lib.h`. You have to implement two functions, `Alice` and `Bob`.

```
void Alice( int N, int M, int A[], int B[] )
```

- For each test case, this function is called once.
- The parameter N is the number of nodes of the original graph
- The parameter M is the number of edges in the original graph
- The parameters $A[\,]$, $B[\,]$ are sequences of length M describing edges of the original graph

Using the following functions, the function Alice outputs information of the graph G sent by Alice.

`void InitG( int V, int U )` This function specifies the number of nodes of G and the number of edges of G. Call this once.

- The parameter V is the number of vertices of G. The parameter V should be an integer between 1 and 1500, inclusive. If the call to this function has parameters outside this range, your program is considered as **Wrong Answer[1]**.

[1] A simple graph is one with no self loops or multiple edges

- The parameter U is the number of edges of G. The parameter U should be an integer between 0 and $\frac{V(V-1)}{2}$, inclusive. If the call to this function has parameters outside this range, your program is considered as **Wrong Answer[2]**.

`void MakeG( int pos, int C, int D )` This function specifies the edges of G.

- The parameter *pos* is the number of the edge specified by the call. The parameter *pos* should be an integer between 0 and $U-1$, inclusive. If the call to this function has parameters outside this range, your program is considered as **Wrong Answer[3]**. This function not be called more than once with the same parameter *pos*. If this function is called more than once with the same parameter, your program is considered as **Wrong Answer[4]**.
- The parameters C and D are the vertices of the edge *pos* of the graph G. C and D should be integers between 0 and $V-1$, inclusive. Also, $C \neq D$ should be satisfied. If C or D does not satisfy these conditions, your program is considered as **Wrong Answer[5]**.
- Here U and V are the integers specified by `InitG`.

In the function `Alice`, after calling the function `InitG` once, the function `MakeG` should be called exactly U times. If the function `InitG` is caled twice, your program is considered as **Wrong Answer [6]**. If the function `MakeG` is called before the function `InitG` is called, your program is considered as **Wrong Answer[8]**. When the function `Alice` terminates, if the graph G described by `Alice` is not a simple graph, your program is considered as **Wrong Answer[9]**.

If the call to the function `Alice` is considered as Wrong Answer, your program is terminated immediately.

The second function to implement:

`void Bob( int V, int U, int C[], int D[] )`

- For each test case, this function is called once.
- The parameter V is the number of nodes of the graph G.
- The parameter U is the number of edges of the graph G.
- The parameters $C[\,], D[\,]$ are sequences of length U describing the edges of the graph G.

He should output his answer using the following functions:

`void InitMap( int N, int M )` This function answers the number of nodes N and the number of edges M of the original graph.

- The parameter N is an integer which should be equal to the actual number of nodes in the original graph. If they are not equal, your program is considered as **Wrong Answer[10]**.
- The parameter M is an integer which should be equal to the actual number of edges in the original graph. If they are not equal, your program is considered as **Wrong Answer[11]**.

`void MakeMap( int A, int B )` This function answers the edges that are present in the original graph.

- The parameters A and B mean there is an edge connecting the nodes A and B. A and B are integers between 0 and $N-1$, inclusive. Also, $A \neq B$ should be satisfied. If A or B does not satisfy these condition, your program is considered as **Wrong Answer [12]**. If there does not exist an edge connecting the nodes A and B in the original graph, your program is considered as **Wrong Answer [13]**. The edge described by a call to this function should be different from the edges of previous calls. When `MakeMap( A, B )` is called, if either `MakeMap( A, B )` is called or `MakeMap( B, A )` was already called before, your program is considered as **Wrong Answer [14]**.
- Here, N is the integer specified by `InitMap`.

In the function `Bob`, after calling the function `InitMap` once, the function `MakeMap` should be called exactly M times. If the function `InitMap` is called twice, your program is considered as **Wrong Answer [15]**. If the function `MakeMap` is called before the function `InitMap` is called, your prorgam is considered as **Wrong Answer [16]**. If `InitMap` is not called when the function `Bob` terminates, or the function `MakeMap` is not called M times, your program is considered as **Wrong Answer [17]**. Here, M is the integer value specified by `InitMap`.

If the call to function `Bob` is considered Wrong Answer, your program is terminated immediately.

Also, if Alice calls Bob's functions or vice versa, it will considered as **Wrong Answer [18]**. A summary of wrong answer numbers can be found in the appendix

Grading Procedure

The grading is done in the following way. If your program is considered as Wrong answer it is terminated immediately.

1. The function `Alice` is called once whose parameters describe information of the original graph.
2. Let G be the graph specified by the function `Alice`. The function `Bob` is called once whose parameters are the shuffled numbers of the vertices of G and the shuffled numbers on the edges of G.
3. Your program is graded.

Important Notices

- To ensure integrity, you must not use any global variables that both Alice and Bob can access. If you wish to use global variables, you can include them in a namespace, and access only the appropriate namespaces. The sample code provided gives a template on how to do so. This is a communication so you should send the information over by the functions only.
- We will be manually checking the submitted codes during and at the end of the contest, submissions that solves the problem by abusing information that should not be available will be voided. We trust that you know what to do, and if you're unsure if what you're doing is allowed, feel free to clarify.

- Your program can implement other functions for internal use. Submitted files will be compiled with the grader, and become a single executable file. All global variables and internal functions should be declared `static` to avoid confliction with other files. When it is graded, it will be executed as two processes of `Alice` and `Bob`. The process of `Alice` and `Bob` can not share global variables.
- Your program should not use the standard input and the standard output. Your program should not communicate with other files by any methods. But, your program may output debugging information to the standard error.

Compilation and Test Run

You can download a `.zip` file from the contest webpage which contains the sample grader to test your program. The `.zip` file also contains a sample source file of your program.

The sample grader consists of one source file, which is `samplegrader.cpp`. If your program is `alicebob.cpp`, to test them, put these files (`samplegrader.cpp`, `alicebob.cpp`, `lib.h`, in the same directory and run the following command to compile your prorgams.

```
g++ -std=c++17 -O2 -o grader samplegrader.cpp alicebob.cpp
```

When the compilation succeeds, the executable file `grader` is generated.

Note that the actual grader is different from the sample grader.

Input for the Sample Grader

The sample grader reads the following data from the standard input.
The first line contains 2 integers, N, M, the number of nodes and edges in the original graph.
The following M lines contain information of the original graph. The $(i+1)$-th line $(0 \leq i \leq M-1)$ of the M lines contains 2 integers, A_i, B_i that an edge of the original graph.

Output of the Sample Grader

When the program terminates successfully, the sample grader write the following information to the standard output.

If your program is considered as Wrong Answer, the sample grader writes its type in the following form "`Wrong Answer [1]`" and terminates.

Otherwise, the sample grader writes "`Accepted`". It also outputs the value of $V - N$ for subtask 3.

Constraints

$1 \leq N \leq 1000$ and the original graph is simple.

Subtasks

0. (0 points) Sample testcases
1. (13 points) $N \leq 10$
2. (7 points) $N \leq 40$
3. (80 points) No additional constraints.

For subtask 1 and 2, you will get all the points as long as you satisfy all the constraints.

For subtask 3, the scoring parameter is $d = V - N$, which represents difference in the number of nodes in the sent graph and the original graph.

- If $101 \leq d$, your points is 0.
- If $21 \leq d \leq 100$, your points is $\frac{(x-100)^2}{200} + 9.99$.

If $d \leq 20$, your points will be given by the following table.

d	20	19	18	17	16	15	14	13	≤ 12
points	43.2	45.3	47.4	49.5	52.6	55.7	62.8	65.9	80

Sample Communication

Here is a sample input for sample grader and the corresponding function calls.

Sample Input 1	Sample Calls			
	Call	Return	Call	Return
4 3	`Alice(...)`			
0 1			`InitG(4,3)`	
0 2				(none)
0 3			`MakeG(0,0,1)`	
				(none)
			`MakeG(1,0,2)`	
				(none)
			`MakeG(2,0,3)`	
				(none)
		(none)		
	`Bob(...)`			
			`InitMap(4,3)`	
				(none)
			`MakeMap(0,1)`	
				(none)
			`MakeMap(0,2)`	
				(none)
			`MakeMap(0,3)`	
				(none)
		(none)		

In this case, the parameters given to the function `Alice(...)`, `Bob(...)` are as follows.

Parameters	`Alice(...)`	`Bob(...)`
N	4	
M	3	
V		4
U		3
A	{0, 0, 0}	
B	{1, 2, 3}	
C		{2, 2, 2}
D		{3, 0, 1}

Appendix - shuffling method

More formally, the graph G sent by Alice can be transformed as follows.

1. Alice specifies a graph G with V nodes and U edges. The nodes are numbered $0, 1, \ldots, V-1$ and the edges are numbered from $0, 1, U-1$.

2. Let the i-th edge ($0 \leq i \leq U-1$) of G connect nodes C_i and D_i.

3. The numbers of the vertices of G are shuffled. First, a sequence $p[0], p[1], \ldots, p[V-1]$, which is a permutation of $0, 1, \ldots, V-1$, is generated. Then $C_0, C_1, \ldots, C_{U-1}$ are replaced with $p[C_0], p[C_1], \ldots p[C_{U-1}]$, and $D_0, D_1, \ldots, D_{U-1}$ are replaced with $p[D_0], p[D_1], \ldots, p[D_{U-1}]$.

4. Then, the order of the edges of G is shuffle. First, a sequence $q[0], q[1], \ldots, q[U-1]$, which is a permutation of $0, 1, \ldots, U-1$, is generated. Then $C_0, C_1, \ldots, C_{U-1}$ are replaced with $C_{q[0]}, C_{q[1]}, \ldots C_{q[U-1]}$, and $D_0, D_1, \ldots, D_{U-1}$ are replaced with $D_{q[0]}, D_{q[1]}, \ldots, D_{q[U-1}$.

5. Then, the direction of the edges of G are shuffled. First, a sequence $r[0], r[1], \ldots, r[U-1]$, $0 \leq r[i] \leq 1$ ($0 \leq i \leq U-1$), is generated. For all i ($0 \leq i \leq U-1$), if $r[0] = 0$, $(C'_i, D'_i) = (C_i, D_i)$, otherwise $(C'_i, D'_i) = (D_i, C_i)$.

6. Finally, the following data is sent to Bob: the values of V and U, and the values of the parameters $C'_0, C'_1, \ldots, C'_{U-1}$ and $D'_0, D'_1, \ldots, D'_{U-1}$.

Appendix - wrong answer[x]

There are a total of 18 wrong answer messages, they are:

1. V not within $[1, 1500]$
2. U not within $[0, \frac{V(V-1)}{2}]$
3. In MakeG, *pos* not within 0 and $U - 1$
4. In MakeG called with same *pos* more than once
5. In MakeG, C or D not in $[0, V - 1]$ or $C = D$
6. InitG called twice
7. MakeG called before InitG
8. MakeG not called exactly U times
9. G is not simple
10. InitMap answers wrong N
11. InitMap answers wrong M
12. In MakeMap, A or B not in $[0, N - 1]$ or $A = B$
13. In MakeMap, (A, B) is not an edge that exists in the original graph
14. MakeMap calls (A, B) when (A, B) or (B, A) has been called before
15. InitMap called twice
16. MakeMap called before InitG
17. MakeMap not called exactly M times
18. Alice calls InitMap or MakeMap, or Bob calls MakeG or InitG