

<b>Started on</b>	Wednesday, 27 March 2024, 5:30 PM
<b>State</b>	Finished
<b>Completed on</b>	Wednesday, 27 March 2024, 6:14 PM
<b>Time taken</b>	43 mins 35 secs
<b>Grade</b>	<b>10.00</b> out of 10.00 ( <b>100%</b> )

## Question 1

Correct

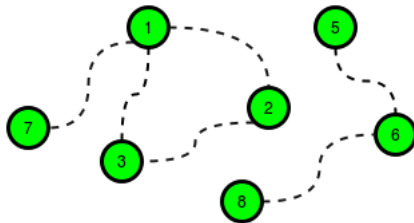
Mark 10.00 out of 10.00

Determine the minimum cost to provide library access to all citizens of HackerLand. There are  $n$  cities numbered from  $1$  to  $n$ . Currently there are no libraries and the cities are not connected. Bidirectional roads may be built between any city pair listed in ***cities***. A citizen has access to a library if:

- Their city contains a library.
- They can travel by road from their city to a city containing a library.

## Example

The following figure is a sample map of HackerLand where the dotted lines denote possible roads:



$c_{road} = 2$

$c_{lib} = 3$

***cities*** =  $[[1, 7], [1, 3], [1, 2], [2, 3], [5, 6], [6, 8]]$

The cost of building any road is  $c_{road} = 2$ , and the cost to build a library in any city is  $c_{lib} = 3$ . Build  $5$  roads at a cost of  $5 \times 2 = 10$  and  $2$  libraries for a cost of  $6$ . One of the available roads in the cycle  $1 \rightarrow 2 \rightarrow 3 \rightarrow 1$  is not necessary.

There are  $q$  queries, where each query consists of a map of HackerLand and value of  $c_{lib}$  and  $c_{road}$ . For each query, find the minimum cost to make libraries accessible to all the citizens.

## Function Description

Complete the function `roadsAndLibraries` in the editor below.

`roadsAndLibraries` has the following parameters:

- *int n*: integer, the number of cities
- *int c<sub>lib</sub>*: integer, the cost to build a library
- *int c<sub>road</sub>*: integer, the cost to repair a road
- *int cities[m][2]*: each ***cities***[*i*] contains two integers that represent cities that can be connected by a new road

## Returns

- *int*: the minimal cost

## Input Format

The first line contains a single integer  $q$ , that denotes the number of queries.

The subsequent lines describe each query in the following format:

- The first line contains four space-separated integers that describe the respective values of  $n$ ,  $m$ ,  $c_{lib}$  and  $c_{road}$ , the number of cities, number of roads, cost of a library and cost of a road.
- Each of the next  $m$  lines contains two space-separated integers,  $u[i]$  and  $v[i]$ , that describe a bidirectional road that can be built to connect cities  $u[i]$  and  $v[i]$ .

## Constraints

- $1 \leq q \leq 10$
- $1 \leq n \leq 10^5$
- $0 \leq m \leq \min(10^5, \frac{n \cdot (n-1)}{2})$
- $1 \leq c_{road}, c_{lib} \leq 10^5$
- $1 \leq u[i], v[i] \leq n$
- Each road connects two distinct cities.

For example:

Input	Result
2	4
3 3 2 1	12
1 2	
3 1	
2 3	
6 6 2 5	
1 3	
3 4	
2 4	
1 2	
2 3	
5 6	

Answer: (penalty regime: 0 %)

Reset answer

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  string ltrim(const string &);
6  string rtrim(const string &);
7  vector<string> split(const string &);
8
9  /*
10 * Complete the 'roadsAndLibraries' function below.
11 * The function is expected to return a LONG_INTEGER.
12 * The function accepts following parameters:
13 * 1. INTEGER n
14 * 2. INTEGER c_lib
15 * 3. INTEGER c_road
16 * 4. 2D_INTEGER_ARRAY cities
17 */
18
19
20 long roadsAndLibraries(int n, int c_lib, int c_road, vector<vector<int>> cities) {
21     if (c_lib <= c_road) {
22         // If building libraries is cheaper than roads, build library in each city
23         return (long)n * (long)c_lib;
24     }
25
26     vector<vector<int>> adj(n + 1);
27     vector<bool> visited(n + 1, false);
28     long cost = 0;
29
30     // Build adjacency list
31     for (auto& road : cities) {
32         adj[road[0]].push_back(road[1]);
33         adj[road[1]].push_back(road[0]);
34     }
35
36     // DFS function
37     function<void(int)> dfs = [&](int node) {
38         visited[node] = true;
39         for (int neighbor : adj[node]) {
40             if (!visited[neighbor]) {
41                 dfs(neighbor);
42             }
43         }
44     };
45
46     // Count the number of connected components
47     int components = 0;
48     for (int i = 1; i <= n; ++i) {
49         if (!visited[i]) {
50             ++components;
51             dfs(i);
52         }
53     }
54 }

```

	Input	Expected	Got	
✓	2 3 3 2 1 1 2 3 1 2 3 6 6 2 5 1 3 3 4 2 4 1 2 2 3 5 6	4 12	4 12	✓
✓	5 9 2 91 84 8 2 2 9 5 9 92 23 2 1 5 3 5 1 3 4 3 1 5 4 4 1 5 2 4 2 8 3 10 55 6 4 3 2 7 1 1 0 5 3 2 0 102 1	805 184 80 5 204	805 184 80 5 204	✓
✓	1 5 3 6 1 1 2 1 3 1 4	15	15	✓

Passed all tests! ✓

► [Show/hide question author's solution \(C++\).](#)

Correct

Marks for this submission: 10.00/10.00.