***Toll Tax***

C371\_Coding\_Febraury2023

**Topic**: Queue

**Difficulty Level:** Hard

**Question / Problem Statement**:

Eleanor is studying the city's road system to find ways of improving its traffic conditions. The road system consists of **n** junctions connected by **e** bidirectional toll roads, where the ith toll road connects junctions **Xi** and **Yi.** Some junctions may not be reachable from others and there may be multiple roads connecting the same pair of junctions.

Each toll road has a toll rate that's paid each time it's used. This rate varies depending on the direction of travel:

* If travelling from **Xi** to **Yi**, then the toll rate is **ri.**
* If travelling from **Yi** to **Xi**, then the toll rate is 1000-**ri**. It is guaranteed that 0 < **ri** < 1000.

For each digit **d** ∈ {0,1,...,9}, Eleanor wants to find the number of ordered pairs of **(x, y)** junctions such that **x != y** and a path exists from **x** to **y** where the total cost of the tolls (i.e., the sum of all toll rates on the path) ends in digit **d**. Given a map of the city, for each digit **d** from 0 to 9, return the number of valid ordered pairs.

Write a program to return the number of valid ordered pairs for each digit **d** from 0 to 9.

**Note**

Each toll road can be traversed an unlimited number of times in either direction.

**Function Description**

In the provided code snippet, implement the provided **tollTax(...)** method using the variables to to return the number of valid ordered pairs for each digit **d** from 0 to 9. You can write your code in the space below the phrase **“WRITE YOUR LOGIC HERE”**.

There will be multiple test cases running so the Input and Output should match exactly as provided.  
The base Output variable **result** is set to a default value of **-404** which can be modified. Additionally, you can add or remove these output variables.

**Input Format**

The first line contains two space-separated integers describing the respective values of **n (**the number of junctions) and **e** (the number of roads).

Each line **i** of the **e** subsequent lines describes a toll road in the form of three space-separated integers, **Xi, Yi**, and **ri**.

**Sample Input**

3 3

1 3 602

1 2 256

2 3 411

**Constraints**

1 <= **n** <= 10^5.

1 <= **e** <= 2\*10^5.

1 <= **Xi, Yi** <= **n.**

**Xi != Yi.**

0 < **ri** < 1000.

**Output Format**

Print ten lines of output. Each line j (where 0 <= j <= 9) must contain a single integer denoting the answer for **d = j**. For example, the first line must contain the answer for **d** = 0, the second line must contain the answer for **d** = 1, and so on.

**Sample Output**

0

2

1

1

2

0

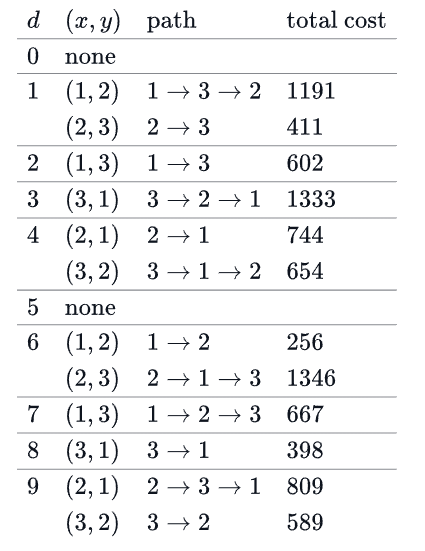
2

1

1

2

**Explanation**



Note the following:

* There may be multiple paths between each pair of junctions.
* Junctions and roads may be traversed multiple times. For example, the path 2 => 3 => 1 => 2 => 3 is also valid, and it has a total cost of 411+398+256+411=1476.
* An ordered pair can be counted for more than one d. For example, the pair (2, 3) is counted for d=1 and d=6.
* Each ordered pair must only be counted once for each d. For example, the paths 2=>1=>3 and 2 => 3 => 1 => 2 => 3 both have total costs that end in d=6, but the pair (2, 3) is only counted once.

**Solution Steps**

1. First, let's assume that all costs are reduced to modulo 10, including costs of whole paths. This allows us to restate the problem into simpler terms:

For every digit d ∈ {0,1,..,9}. How many ordered pairs of nodes (x,y) are there such that x!= y and there is a path from x to y with total cost d?

The graph in this problem is directed, but has the property that for every edge (x, y) with cost f, there is a corresponding edge (y, x) with cost -f. (Remember we're working modulo 10, so -f is really -f mod 10.) Also, in this problem, a path can pass through an edge multiple times.

We will also assume that the graph is connected, because otherwise, we can simply perform the same procedure for every connected component and add the numbers up.

2. Let's ignore the constraints first and find a simple working solution. Let's fix the node x, and say we're trying to count the number of nodes y that satisfy the problem requirements for each digit d. Rephrasing it slightly, we are trying to answer the following question: For each node y and digit d ∈ [0, 10), is there a path from x to y with a cost d?

We have the following useful facts:

* There's a path from x to x with a cost 0.
* If there's a path from x to y with cost d, and there's an edge from y and z with cost f, then there's a path from x to z with cost d + f. (Remember we're working modulo 10.)

This gives us a solution with breadth-first search (BFS) / depth-first search (DFS). We will perform a BFS/DFS where our states will be (y, d) for a node y and d ∈ [0, 10). We begin the search at the state (x,0) and then end the search if there are no more reachable states. We're guaranteed to end because there are a finite number of states. (In fact, exactly 10N.) After the search, we now have enough information to compute the answer: for each digit d, simply count the number of reached states of the form (y, d)! (Except y = x.)

3. Notice that performing a BFS/DFS starting at (x, 0) gives us the following information:

For each digit d, the number of nodes y such that (x,0) and (y, d) is connected. Let's call this

number Cx(d).

If we can compute Cx (d) for every node x and digit d, then we can compute the answers by simply adding the numbers with corresponding ds (with some minor adjustments which will be discussed later on). However, as explained earlier, we cannot perform a search for all nodes x because that would be slow. Thus, our goal here is to compute all the Cx(d) without doing too many searches.

Here are some simple but important observations:

1. If there is a path from x to y with cost d, then there is a path from y to x with cost -d. (Why?)

2. If there is a path from x to y with cost d, and there is a path from y to z with cost e, then there is a path from x to z with cost d + e.

Now, let's try to perform a BFS/DFS starting with some arbitrary node, say x. This gives us access to the values Cx (d) for all digits d.

How do we compute Cx’ (d) for other nodes x'? We can do so if we perform another BFS/DFS starting on (x',0), though this will be slow. But the following claim will give us another way of computing Cx’(d):

Assume (x, 0) and (x',d') are connected. (x and x' are connected in the original graph, so (x,0)

must be connected to (x' ,d') for some d'.) Then (x',0) and (y, d) are connected if and only if

(x,0) and (y, d + d') are connected.

We can prove this with the observations above:

* Assume (x, 0) and (x',d') are connected.
* By definition, there is a path from x to x' with cost d'
* By observation 1, there is a path from x' to x with cost -d'
* Now, suppose that (x, 0) and (y, d + d') are connected.
* By definition, there is a path from x to y with cost d + d'
* By observation 2, there is a path from x' to y with cost (d+d')-d' = d.
* Thus, by definition, (x',0) and (y, d) are connected.
* This proves the backward implication. A similar argument proves the forward implication.

This claim has a very useful consequence: Cx’(d) = Cx ((d + d') mod 10). This allows us to

compute Cx’ (d) without performing another BFS/DFS! (Notice that the d' values can also be

computed as a byproduct of the search.) This makes the running time of the overall algorithm simply O(N + E), which passes the time limit.

4. As a final note, we mentioned that minor adjustments must be made. This is because Cx(d)

doesn't take into account the constraint x != y. Thankfully, this is much easier to account for. We'll describe how to do that here:

The value ∑x’ is a node Cx’ (d) is the answer for the digit d, not taking into account the constraint x != y. In other words, it includes the paths from (x, 0) to (x, d), which we need to exclude from the sum. Let's define the following quantity:

Vx’ (d) = 1 if there is a path from (x', ,0) to (x', d)

Vx’ (d) = 0 otherwise

5. Using this quantity, we can now adjust the answer for a digit d: the adjusted answer is

∑x’ is a node (Cx' (d) -Vz (d)). Thus, our goal now is to compute Vx’ (d).

Notice that, by performing a search starting with the node x, we also naturally compute the array vis [x'][d], which is 1 if (x', ,d) is visited, and 0 otherwise. Now, suppose we want to know if there is a path from (x', ,0) to (x', ,d). But by the claim above, (x',0) and (x',d) are connected if and only if (x, 0) and (x' ,d + d’) are connected. This simply means that Vx' (d) is just vis[x'][(d + d') mod 10]. Thus, the V1' (d) can be computed with no significant additional running time.

**Running Solution in C++** :

#include <bits/stdc++.h>

using namespace std;

const int N = 1e5 + 6;

bool vis[N][10];

vector< pair< int, int > > adj[N];

long long ans[10], cnt[10];

bool mark[N];

int main() {

int n, m;

scanf("%d %d", &n, &m);

for (int i = 0; i < m; i++) {

int u, v, w;

scanf("%d %d %d", &u, &v, &w);

u--; v--;

w %= 10;

adj[u].emplace\_back(v, w);

adj[v].emplace\_back(u, (10 - w) % 10);

}

for (int r = 0; r < n; r++) if (!mark[r]) {

queue<pair< int, int >> que;

que.push(make\_pair(r, 0));

vis[r][0] = 1;

vector< int > visited;

while (!que.empty()) {

int u = que.front().first, w = que.front().second;

if (!mark[u]) {

mark[u] = 1;

visited.push\_back(u);

}

que.pop();

for (auto it : adj[u]) {

int v = it.first, nw = w + it.second;

if (nw >= 10)

nw -= 10;

if (vis[v][nw]) continue;

que.push(make\_pair(v, nw));

vis[v][nw] = 1;

}

}

memset(cnt, 0, sizeof cnt);

for (int i : visited)

for (int j = 0; j < 10; j++)

cnt[j] += vis[i][j];

for (int i : visited) {

for (int j = 0; j < 10; j++) if (vis[i][j]) {

int k = 10 - j;

if (k >= 10) k -= 10;

for (int z = 0; z < 10; z++) {

int nz = k + z;

if (nz >= 10) nz -= 10;

ans[nz] += cnt[z] - vis[i][z];

}

break;

}

}

}

// ans[0] -= n;

for (int i = 0; i < 10; i++)

printf("%lld\n", ans[i]);

return 0;

}

Input:

2 2

1 2 5

2 2 3

Output:

2

2

2

2

2

2

2

2

2

2

**Test Cases [ Qty: 12 ]**

| **Test Case No** | **Input** | **Output** | **Score** |
| --- | --- | --- | --- |
| 1 | 3 3  1 3 602  1 2 256  2 3 411 | 0  2  1  1  2  0  2  1  1  2 | 0 |
| 2 | 2 2  1 2 5  2 2 3 | 2  2  2  2  2  2  2  2  2  2 | 0 |
| 3 | 99883 5  94305 89594 833  96260 51462 36  19772 52266 764  93603 35855 723  66861 50225 657 | 0  0  0  3  2  0  2  3  0  0 | 1 |
| 4 | 10 6  1 2 5  2 3 10  3 4 25  2 4 2  1 4 20  5 6 10 | 14  12  12  12  12  12  12  12  12  12 | 1 |
| 5 | 99840 19  29573 93419 495  37716 67815 565  1393 31880 335  12892 31622 525  56829 70850 5  58307 60331 680  85615 98551 480  49794 65441 525  79420 25490 655  55776 3676 410  61806 26691 70  99770 19344 880  24457 58437 855  5103 82660 805  83972 98940 165  21370 91130 30  78480 84639 900  66856 694 700  20349 34703 375 | 16  0  0  0  0  22  0  0  0  0 | 1 |
| 6 | 15 10  1 2 5  2 3 4  3 4 8  4 5 7  5 6 12  6 7 18  7 8 12  8 9 15  9 10 25  10 12 85 | 10  11  10  10  10  18  10  10  10  11 | 1 |
| 7 | 100000 30  5024 9678 535  90530 89703 206  20467 17299 494  13952 86682 228  85702 18312 728  67467 28811 939  12344 49979 782  21629 47655 485  71813 75133 255  25247 99188 663  47172 86904 580  84584 23288 774  5236 41651 857  7604 28128 628  32447 85254 291  41072 99853 949  21848 8608 234  28557 23286 131  28168 70499 644  56382 36456 965  12462 23661 290  47389 63834 295  83964 30353 291  30839 64912 763  98803 9633 617  31098 45210 208  47199 93990 39  99625 97444 273  38236 48498 697  21061 14344 874 | 4  6  5  6  6  10  6  6  5  6 | 1 |
| 8 | 99840 40  65106 21555 485  414 66322 180  22365 80657 755  39608 69794 555  32595 39461 880  5967 94728 740  36000 66371 550  10714 22434 245  98778 99020 45  46117 32432 445  79481 92224 455  66860 70449 460  45223 59884 415  63539 8900 705  96220 7246 310  77496 48626 400  63292 20015 670  64824 43922 745  63139 56360 275  12298 26 710  57542 22519 15  35929 74876 565  6551 70774 210  42106 22342 980  43897 75687 20  33512 63834 685  65161 92758 775  83605 44633 30  60836 49379 780  99712 10100 55  4877 11942 105  69488 78421 195  41263 46480 590  33355 76671 730  14988 21262 555  74290 2289 10  65581 61844 730  68492 14819 840  67302 55393 965  94253 64152 150 | 40  0  0  0  0  40  0  0  0  0 | 1 |
| 9 | 99999 50  30998 87435 597  60516 3611 617  85384 10931 204  89669 49804 451  98670 64529 408  64172 64757 422  92603 96294 756  53089 38018 946  19851 69177 569  93159 61842 945  42876 3004 472  54210 72904 169  29495 85444 991  64 3087 213  35272 97708 599  94777 94304 544  63581 78698 658  33880 88949 32  77412 4575 246  97389 84475 992  58843 27095 880  73429 71394 221  62883 87488 544  15515 52379 443  92579 31972 236  61718 77831 251  33567 74091 919  83869 66247 539  26017 1726 833  89964 69929 645  26866 65447 331  70723 87495 94  16573 91547 122  88087 24549 403  94317 90631 366  66424 42675 905  96933 17369 820  19947 97624 353  3625 17433 954  77387 2894 594  36958 33541 246  44306 90725 915  21551 71249 194  40146 38853 692  51486 11354 939  61609 54851 208  27298 16350 56  22152 13124 184  5459 92064 524  41895 77019 629 | 4  12  9  7  16  8  16  7  9  12 | 1 |
| 10 | 99999 9  36958 33541 246  44306 90725 915  21551 71249 194  40146 38853 692  51486 11354 939  61609 54851 208  27298 16350 56  22152 13124 184  5459 92064 524 | 0  1  2  0  5  2  5  0  2  1 | 1 |
| 11 | 99999 30  36958 33541 246  44306 90725 915  21551 71249 194  40146 38853 692  51486 11354 939  61609 54851 208  27298 16350 56  22152 13124 184  5459 92064 524  41895 77019 629  35272 97708 599  94777 94304 544  63581 78698 658  33880 88949 32  77412 4575 246  97389 84475 992  58843 27095 880  73429 71394 221  62883 87488 544  15515 52379 443  61718 77831 251  33567 74091 919  83869 66247 539  26017 1726 833  89964 69929 645  26866 65447 331  70723 87495 94  16573 91547 122  88087 24549 403  94317 90631 366 | 2  8  6  3  10  4  10  3  6  8 | 1 |
| 12 | 100000 30  28168 70499 644  56382 36456 965  12462 23661 290  47389 63834 295  83964 30353 291  30839 64912 763  98803 9633 617  31098 45210 208  47199 93990 39  99625 97444 273  38236 48498 697  21061 14344 874  5024 9678 535  90530 89703 206  20467 17299 494  13952 86682 228  85702 18312 728  67467 28811 939  12344 49979 782  21629 47655 485  71813 75133 255  25247 99188 663  47172 86904 580  84584 23288 774  5236 41651 857  7604 28128 628  32447 85254 291  41072 99853 949  21848 8608 234  28557 23286 131 | 4  6  5  6  6  10  6  6  5  6 | 1 |