F.

Xét một cặp đỉnh (i, j) bất kỳ: Ta biết rằng input phải thỏa mãn d(i, j) <= d(i, k) + d(k, j) với mọi k khác i và j (bất đẳng thức tam giác). d(i, k) + d(k, j) là một đường đi từ i qua k rồi tới j. Rõ ràng đường đi này không thể tốt hơn hẳn đường đi ngắn nhất là d(i, j). Do đó d(i, j) > d(i, k) + d(k, j) là vô lý

Trong bài toán này, với cặp đỉnh (i, j) nếu tồn tại một đỉnh k sao cho d(i, j) = d(i, k) + d(k, j) thì cạnh từ i đến j không cần phải có. Bởi đã có một cách để đảm bảo đường đi ngắn nhất là đi thông qua k. Ngược lại, nếu với mọi k ta đều có d(i, j) < d(i, k) + d(k, j) thì rõ ràng phải dựng ra một cạnh nối i và j; bởi việc đi qua bất kỳ đỉnh trung gian nào đều không khả thi.

E.

Subtask 1

Đầu tiên dùng Floyd (hoặc Dijkstra n lần) để xây dựng mảng độ dài đường đi ngắn nhất giữa hai đỉnh bất kỳ. Duyệt từng cạnh và thử xóa cạnh này khỏi đồ thị. Sau khi xóa khỏi đồ thị, lại tiếp tục Dijkstra n lần đề xác định mảng độ dài đường đi ngắn nhất mới sau khi xóa cạnh. Khi đó từ mảng đường đi ngắn nhất cũ và mảng đường đi ngắn nhất mới, ta sẽ đếm được số cặp đỉnh (u, v) mà bị ảnh hưởng bởi việc xóa đi cạnh này.

Subtask 2

Đếm số đường đi ngắn nhất trên đồ thị.

Cách đếm số đường đi ngắn nhất trên đồ thị bằng thuật toán Dijkstra:

```
struct Edge {
	int u, v, cost;

	Edge(int _u = 0, int _v = 0, int _cost = 0) {
		u = _u; v = _v; cost = _cost;
	}
};

int numNode, numEdge;
Edge edges[MAX]; // mảng lưu các cạnh
vector<int> adj[MAX]; // mảng lưu chỉ số của các cạnh trong danh sách kề

// đọc đồ thị
cin >> numNode >> numEdge;
for (int i = 1; i <= numEdge; i++) {
	int u, v, c; cin >> u >> v >> c;
	edges[i] = Edge(u, v, c);
	adj[u].push_back(i); // vector danh sách kề lưu chỉ số của cạnh
	adj[v].push_back(i);
}
```

```
// Dijkstra đếm số đỉnh và số cạnh của đồ thị
long long dist[MAX][MAX]; // dist(u, v) = độ dài đường đi ngắn nhất từ u đến v
int numPath[MAX][MAX]; // numPath(u, v) = số đường đi ngắn nhất từ u đến v
// chạy thuật toán dijkstra tính độ dài và đếm số đường đi ngắn nhất từ s tới mọi đỉnh khác
trong đồ thị
void dijkstra(int s) {
	priority_queue<pair<long long, int>, vector<pair<long long, int>>, greater...> q;
	dist[s][s] = 0;
	numPath[s][s] = 1; // chỉ có 1 đường đi ngắn nhất từ s đến s
	q.push(make_pair(0, s));

	while (!q.empty()) {
		long long d = q.top().first;
		int u = q.top().second;
		q.pop();
		if (d != dist[s][u]) continue;

		// duyệt qua danh sách kề của u để tối ưu cho các nhãn khác
		for (int i : adj[u]) {
			int v = edges[i].u + edges[i].v - u;
			int c = edges[i].cost;
			if (minimize(dist[s][v], dist[s][u] + c)) {
				numPath[s][v] = numPath[s][u];
				q.push(make_pair(dist[s][v], v));
			} else if (dist[s][v] == dist[s][u] + c) {
				numPath[s][v] += numPath[s][u];
			}
		}
	}
}

// gọi thuật toán dijkstra từ tất cả các đỉnh từ 1 đến n
for (int i = 1; i <= n; i++) dijkstra(i);
```

Quay lại bài toán

Xét một cạnh bất kỳ trên đồ thị, giả sử là cạnh (u, v) với trọng số c. Để việc xóa cạnh này đi làm thay đổi độ dài đường đi ngắn nhất từ x đến y, nó phải thuộc mọi đường đi ngắn nhất từ x đến y.

Điều kiện để **cung có hướng u -> v trọng số c** thuộc **ít nhất một đường đi** ngắn nhất từ x đến y:
dist(x, y) = dist(x, u) + c + dist(v, y).

Điều kiện để **cung có hướng u -> v trọng số c** thuộc **mọi đường đi** ngắn nhất từ x đến y:
numPath(x, y) = numPath(x, u) * numPath(v, y).

Sau khi ta đã biết hai điều kiện này, lời giải của subtask 2 chính là duyệt qua tất cả M cạnh của đồ thị, với mỗi cạnh, duyệt qua mọi cặp đỉnh (x, y) và kiểm tra điều kiện trên:

```
for (int i = 1; i <= numEdge; i++) {
	int u = edges[i].u, v = edges[i].v, cost = edges[i].cost;

	int result = 0;
	for (int x = 1; x <= numNode; x++) for (int y = 1; y <= numNode; y++) {
		// đi theo chiều u -> v
		if (dist[x][y] == dist[x][u] + cost + dist[v][y] &&
			numPath[x][y] == numPath[x][u] * numPath[v][y]) result++;

		// đi theo chiều v -> u
		if (dist[x][y] == dist[x][v] + cost + dist[u][y] &&
			numPath[x][y] == numPath[x][v] * numPath[u][y]) result++;
	}
	cout << result << " ";
}
cout << endl;
```

Subtask 3

Cây Dijkstra: Được xây dựng như sau: Gốc của cây dijkstra chính là điểm xuất phát. Cha một nút khác gốc trên cây Dijkstra là đỉnh giúp đỉnh đó tối ưu. Nói cách khác, giả sử trong thuật toán Dijkstra, khi bạn dùng đỉnh u tối ưu cho đỉnh v (điều kiện dist(s, v) > dist(s, u) + c) xảy ra và ta thêm đỉnh v vào priority_queue, thì ta đồng thời gán cha của v là u.

Nhận xét: Một cạnh không thuộc cây Dijkstra chắc chắn không ảnh hưởng đến độ dài đường đi ngắn nhất của cặp đỉnh đó.

Mô hình thuật toán của subtask 3 như sau:
+ Thay vì dùng một vòng for để duyệt m cạnh như ở subtask 1 hay 2, ở subtask 3 ta không duyệt theo từng cạnh. Thay vào đó ta dùng một mảng để lưu lại kết quả của các cạnh và in ra sau.

```
for (int s = 1; s <= numNode; s++) {
	<Chạy thuật toán Dijkstra xuất phát từ đỉnh s để dựng được cây Dijkstra>
	for (int t = 1; t <= numNode; t++) {
		Duyệt qua tất cả các cạnh trên đường đi từ t tới s trên cây Dijkstra.
		Với mỗi cạnh, dùng điều kiện ở subtask 2 để kiểm tra và nếu điều kiện thỏa mãn,
tăng kết quả của cạnh này trên mảng kết quả thêm 1.
	}
}
```