目描述：

现需要在某城市进行 5G 网络建设，已经选取 N 个地点设置 5G 基站，编号固定为 1 到 N，接下来需要各个基站之间使用光纤进行连接以确保基站能互联互通，不同基站之间架设光纤的成本各不相同，且有些节点之间已经存在光纤相连，请你设计算法，计算出能联通这些基站的最小成本是多少。

注意：基站的联通具有传递性，入基站 A 与基站 B 架设了光纤，基站 B 与基站 C 也架设了光纤，则基站 A 与基站 C 视为可以互相联通

输入描述：

第一行输入表示基站的个数 N，其中 0<N<=20

第二行输入表示具备光纤直连条件的基站对的数目 M，其中 0<M<N*(N-1)/2

从第三行开始连续输入 M 行数据，格式为 X Y Z P，其中 X Y 表示基站的编号，0<X<=N，0<Y<=N 且 x 不等于 y，Z 表示在 X Y 之间架设光纤的成本，其中 0<Z<100，P 表示是否已存在光纤连接，0 表示未连接，1 表示已连接

输出描述：

如果给定条件，可以建设成功互联互通的 5G 网络，则输出最小的建设成本；

如果给定条件，无法建设成功互联互通的 5G 网络，则输出 -1

示例 1

输入：

```
3

3

1 2 3 0

1 3 1 0
```

```
2 3 5 0
```

输出：

```
4
```

说明：

只需要在 1,2 以及 2,3 基站之间铺设光纤，其成本为 3+1=4

示例 2

输入：

```
3

1

1 2 5 0
```

输出：

```
-1
```

说明：

3 基站无法与其他基站连接，输出 -1

示例 3

输入：

```
3
3
1 2 3 0
1 3 1 0
2 3 5 1
```

输出：

```
1
```

说明：

2,3 基站已有光纤相连，只有要再 1,3 站点之间铺设光纤，其成本为 1

```
import java.util.Scanner;
import java.util.*;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main {
    public static int findRoot(int x, int[] parent) {
        while (x != parent[x]) {
            parent[x] = parent[parent[x]];
            x = parent[x];
        }
        return x;
    }
    public static boolean getUnion(int a, int b, int[] parent) {
        a = findRoot(a, parent);
        b = findRoot(b, parent);
        if (a == b)return false;
        parent[a] = b;
        return true;
    }
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int N = in.nextInt();
        int M = in.nextInt();
        int x, y, z, p;
        int[] parent = new int[N + 1];
        List<int[]>roads = new ArrayList<>();
        for (int i = 1; i <= N; i++) {
            parent[i] = i;
        }
        for (int i = 0; i < M; i++) {
            x = in.nextInt();
            y = in.nextInt();
            z = in.nextInt();
            p = in.nextInt();
            if (p == 1) {
                getUnion(x, y, parent);
            } else {
                int [] node = new int[3];
                node[0] = x;
                node[1] = y;
                node[2] = z;
                roads.add(node);
```

```
                }
            }
            int f1 = -1;
            boolean flag = true;
            for (int i = 1; i <= N; ++i) {
                int root = findRoot(i, parent);
                if (f1 == -1) {
                    f1 = root;
                } else if (f1 != root) {
                    flag = false;
                }
            }
            int result = 0;
            if (flag) {
                System.out.println(0);
                return;
            }
            for (int [] edge : roads) {
                if (getUnion(edge[0], edge[1], parent)) {
                    result += edge[2];
                }
                boolean flag1 = true;
                f1=-1;
                for (int i = 1; i <= N; ++i) {
                    int root = findRoot(i, parent);
                    if (f1 == -1) {
                        f1 = root;
                    } else if (f1 != root) {
                        flag1 = false;
                    }
                }
                if (flag1) {
                    flag = true;
                    break;
                }
            }
            if (flag) {
                System.out.println(result);
            } else {
                System.out.println(-1);

            }
        }
    }
```