// using color and bfs both

#include<bits/stdc++.h>

using namespace std;

vector<vector<pair<int,int> > > graph;

void addEdge(int x, int y, int w){

graph[x].push\_back(make\_pair(y,w));

graph[y].push\_back(make\_pair(x,w));

}

void display(){

for(int i=0;i<graph.size();i++){

cout<<i<<"->";

for(auto it: graph[i]){

cout<<it.first<<" "<<it.second<<", ";

}

cout<<endl;

}

}

//2@2

// 1@21

// 3@23

// 0@210

// 4@234

// 5@2345

// 6@2346

typedef pair<int,int> pi;

bool bipartite(int src, vector<int> &visited){

queue<pi> q;

q.push(make\_pair(src,0));

while(!q.empty()){

pair<int,int> temp = q.front();

q.pop();

if(visited[temp.first]!=-1){

if(temp.first!=visited[temp.first]){

return false;

}

}

else{

visited[temp.first] = temp.second;

}

for(auto it: graph[temp.first]){

if(visited[it.first]==-1){

q.push(make\_pair(it.first,temp.second+1));

}

}

}

return true;

}

// using color

bool bipartite\_bfs(int src, vector<int> &color){

queue<int> q;

q.push(src);

color[src] = 0;

while(!q.empty()){

int node = q.front();

q.pop();

for(auto x: graph[node]){

if(color[x.first]==-1){

q.push(x.first);

color[x.first] = (color[node]==0)? 1: 0;

}

else{

if(color[x.first] == color[node]){

return false;

}

}

}

}

return true;

}

bool bipartite\_DFS(int src, vector<int> &color){

if(color[src]!=-1){

for(auto it: graph[src]){

if(color[it.first]!=-1){

if(color[src]==color[it.first]){

return false;

}

}

}

}

for(auto x: graph[src]){

if(color[x.first]==-1){

color[x.first] = 1 - color[src];

return bipartite(x.first,color);

}

}

return true;

}

int main(){

int v,e;

cin>>v>>e;

graph.resize(v);

while(e--){

int x,y,w;

cin>>x>>y>>w;

addEdge(x,y,w);

}

bool a = true;

vector<int> visited(v,-1);

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

if(visited[i]==-1){

bool flag = bipartite(i,visited);

if(flag==false){

a = false;

}

}

}

if(a){

cout<<"true";

}

else{

cout<<"false";

}

}