

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет Информатика и вычислительная техника

Кафедра Кибербезопасность информационных систем

**ПРАКТИЧЕСКАЯ РАБОТА № 3**

на тему «Компоненты связности графа»

Выполнила обучающаяся гр. ВКБ32

Попова Алина Петровна

Проверил

Савельев Василий Александрович

Ростов-на-Дону

2022

Проверить, что две вершины принадлежат одной компоненте связности (есть путь, их соединяющий)

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  typedef pair < int , int > ipair;  const int N = 100321;  // СНМ  int dsuP[N], dsuR[N];  // В этот массив записываются все изменения СНМ, чтобы их можно откатить  // При изменении какого-то значения в СНМ в hist записывается пара < указатель, старое значение >  vector < pair < int\*, int > > hist;  // Для элемента из СНМ возвращает корень дерева, в котором он находится  int dsuRoot(int v)  {  while (dsuP[v] != -1)  v = dsuP[v];  return v;  }  // Объединяет два множества. Используется ранговая эвристика.  // При любом изменении содержимого массивов dsuP и dsuR  // в hist записывается адрес и старое значение  void dsuMerge(int a, int b)  {  a = dsuRoot(a);  b = dsuRoot(b);  if (a == b)  return;  if (dsuR[a] > dsuR[b])  {  hist.emplace\_back(&dsuP[b], dsuP[b]);  dsuP[b] = a;  } else if (dsuR[a] < dsuR[b])  {  hist.emplace\_back(&dsuP[a], dsuP[a]);  dsuP[a] = b;  } else  {  hist.emplace\_back(&dsuP[a], dsuP[a]);  hist.emplace\_back(&dsuR[b], dsuR[b]);  dsuP[a] = b;  ++dsuR[b];  }  }  struct Query  {  int t, u, v;  bool answer;  };  int n, m;  Query q[N];  // Дерево отрезков, в каждой вершине которого хранится список рёбер  vector < ipair > t[N\*4];  // Эта функция добавляет ребро на отрезок  // [l r] - отрезок, на который добавляется ребро  // uv - ребро, c - текущая вершина дерева отрезков,  // [cl cr] - отрезок текущей вершины дерева отрезков  void addEdge(int l, int r, ipair uv, int c, int cl, int cr)  {  if (l > cr || r < cl)  return;  if (l <= cl && cr <= r)  {  t[c].push\_back(uv);  return;  }  int mid = (cl + cr) / 2;  addEdge(l, r, uv, c\*2+1, cl, mid);  addEdge(l, r, uv, c\*2+2, mid+1, cr);  }  // Обход дерева отрезков в глубину  void go(int c, int cl, int cr)  {  int startSize = hist.size();  // Добавляем рёбра при входе в вершину  for (ipair uv : t[c])  dsuMerge(uv.first, uv.second);  if (cl == cr)  {  // Если эта вершина - лист, то отвечаем на запрос  if (q[cl].t == 3)  q[cl].answer = (dsuRoot(q[cl].u) == dsuRoot(q[cl].v));  } else {  int mid = (cl + cr) / 2;  go(c\*2+1, cl, mid);  go(c\*2+2, mid+1, cr);  }  // Откатываем изменения СНМ  while ((int)hist.size() > startSize)  {  \*hist.back().first = hist.back().second;  hist.pop\_back();  }  }  int main()  {  ios::sync\_with\_stdio(false);  // Формат входных данных:  // n и m, затем в m строках запросы: по три числа t, u, v  // t - тип (1 - добавить ребро, 2 - удалить, 3 - принадлежат ли одной компоненте)  // Нумерация вершин с нуля  cin >> n >> m;  for (int i = 0; i < n; ++i) // Инициализация СНМ  dsuP[i] = -1;    // В этом массиве для каждого ещё не удалённого ребра хранится  // на каком запросе оно было создано  set < pair < ipair, int > > edges;  for (int i = 0; i < m; ++i)  {  cin >> q[i].t >> q[i].u >> q[i].v;  // Поскольку рёбра неориентированные, u v должно означать то же самое, что и v u  if (q[i].u > q[i].v) swap(q[i].u, q[i].v);  // При добавлении ребра кладём его в set  if (q[i].t == 1)  edges.emplace(ipair(q[i].u, q[i].v), i);  // При удалении ребра берём из set время его добавления - так мы узнаём отрезок заросов,  // на котором оно существует. Если есть несколько одинаковых рёбер, можно брать любое.  else if (q[i].t == 2)  {  auto iter = edges.lower\_bound(make\_pair(ipair(q[i].u, q[i].v), 0));  addEdge(iter->second, i, iter->first, 0, 0, m - 1);  edges.erase(iter);  }  }  // Обрабатываем рёбра, которые не были удалены  for (auto e : edges)  addEdge(e.second, m - 1, e.first, 0, 0, m - 1);    // Запускаем dfs по дереву отрезков  go(0, 0, m - 1);  // Выводим ответ.  // При обходе дерева отрезков запросы обрабатываются в том же порядке, в котором они даны,  // поэтому ответ можно выводить прямо в go без заполнения answer  for (int i = 0; i < m; ++i)  if (q[i].t == 3)  {  if (q[i].answer)  cout << "YES\n";  else  cout << "NO\n";  }  return 0;  } |