## 1390. Автогонки

В городе N в ближайшее время состоится этап чемпионата мира по автогонкам среди автомобилей класса Формула-0. Поскольку организаторы не успели построить специальный автодром для этих соревнований, было принято решение провести гонки на улицах города.

В городе N имеется *n* перекрёстков, некоторые из которых соединены дорогами с двусторонним движением. При этом любые два перекрёстка соединены не более чем одной дорогой, и по существующим дорогам можно добраться от любого перекрёстка до любого другого.

Трасса для соревнований должна быть круговой, то есть начинаться и заканчиваться на одном и том же перекрёстке, при этом ни один перекрёсток не должен встречаться на пути более одного раза.

На предварительном этапе подготовки оргкомитет составил список всех дорог города, и теперь пришло время его использовать. Первый вопрос, который необходимо решить, – это вопрос о существовании в городе необходимой круговой трассы. Разумеется, если ответ окажется отрицательным, организаторам придётся срочно построить несколько новых дорог. Однако существует проблема: организаторы подозревают, что некоторые дороги в списке указаны более одного раза, так как он был составлен не слишком тщательно.

Напишите программу, которая по заданному списку дорог определит, возможна ли организация в городе требуемой круговой трассы.

**Вход.** Первая строка содержит два целых числа: количество перекрёстков *n* (1 ≤ *n* ≤ 1000) в городе N и количество дорог *m* (0 ≤ *m* ≤ 105) в списке.

Следующие *m* строк описывают дороги. Каждая дорога задается двумя числами *u* и *v* (1 ≤ *u*, *v* ≤ *n*, *u* ≠ *v*) – номера перекрёстков, которые она соединяет. Поскольку дороги двусторонние, то пары чисел (*u*, *v*) и (*v*, *u*) описывают одну и ту же дорогу.

**Выход.** Выведите “YES”, если в городе можно организовать круговую трассу для соревнований, и “NO” в противном случае.

|  |  |
| --- | --- |
| **Пример входа 1** | **Пример выхода 1** |
| 3 4  1 2  2 3  3 1  3 2 | YES |
|  |  |
| **Пример входа 2** | **Пример выхода 2** |
| 2 3  1 2  2 1  2 1 | NO |

## РЕШЕНИЕ

**графы – циклы**

# Анализ алгоритма

В задаче задан неориентированный связный граф. Необходимо проверить, содержит ли он цикл (который можно превратить в трассу Формулы-0).

Неориентированный граф содержит цикл, если существует обратное ребро. То есть ребро, ведущее в уже пройденную вершину.

# Пример

Приведенные в примере графы имеют вид:



# Первый граф содержит цикл, второй нет.

# Реализация алгоритма

Граф храним в матрице смежности *g*. Массив *used* используем для отмечания пройденных вершин.

#define MAX 1010

int g[MAX][MAX], used[MAX];

Функция ***dfs*** реализует поиск в глубину из вершины *v*. Необходимо отсечь случай, когда из *v* мы направляемся в предка: предок уже пройден, но цикла нет. Для этого в функции ***dfs*** введем второй параметр *p* – предка вершины *v*.

void dfs(int v, int p = -1)

{

Отмечаем вершину *v* пройденной.

used[v] = 1;

Перебираем вершины *i*, куда можно пойти из *v*.

for (int i = 1; i <= n; i++)

{

Рассматриваем все ребра, кроме того, которое ведет к предку *p*.

if (i == p) continue;

Если имеется ребро из *v* в *i*, причем *i* уже пройдена (used[*i*] = 1), то в графе имеется цикл. Устанавливаем *flag* = 1.

if (g[v][i] == 1)

{

if (used[i] == 1) flag = 1;

Иначе продолжаем поиск в глубину из вершины *i*.

else dfs(i, v);

}

}

}

Основная часть программы. Читаем входные данные.

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

memset(g,0,sizeof(g));

memset(used,0,sizeof(used));

Читаем входной неориентированный граф. Граф сохраняем в матрице смежности, поэтому повторяющиеся дороги будут учитываться только один раз.

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

{

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

g[u][v] = g[v][u] = 1;

}

Установим *flag* = 0, что означает отсутствие цикла в графе. Если цикл будет найден, значение переменной *flag* изменится на 1.

flag = 0;

Запускаем поиск в глубину из вершины 1 (по условию задачи граф связный).

dfs(1);

В зависимости от значения переменной *flag* выводим ответ.

if (flag) printf("YES\n");

else printf("NO\n");

# Java реализация

**import** java.util.\*;

**public** **class** Main

{

**static** **int** *g*[][], *used*[];

**static** **int** *n*, *m*, *flag*;

**static** **void** dfs(**int** v, **int** prev)

{

*used*[v] = 1;

**for**(**int** i = 1; i <= *n*; i++)

**if** ((i != prev) && *g*[v][i] == 1)

**if** (*used*[i] == 1) *flag* = 1; **else** *dfs*(i,v);

}

**public** **static** **void** main(String[] args)

{

Scanner con = **new** Scanner(System.***in***);

*n* = con.nextInt();

*m* = con.nextInt();

*g* = **new** **int**[*n*+1][*n*+1];

*used* = **new** **int**[*n*+1];

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

{

**int** u = con.nextInt();

**int** v = con.nextInt();

*g*[u][v] = *g*[v][u] = 1;

}

*flag* == 0;

*dfs*(1,-1);

**if** (*flag* == 1) System.***out***.println("YES");

**else** System.***out***.println("NO");

}

}

# Python реализация

Функция ***dfs*** реализует поиск в глубину из вершины *v*. Необходимо отсечь случай, когда из *v* мы направляемся в предка: предок уже пройден, но цикла нет. Для этого в функции ***dfs*** введем второй параметр *p* – предка вершины *v*.

**def** dfs(v, p = -1):

**global** flag

Отмечаем вершину *v* пройденной.

used[v] = 1

Перебираем вершины *i*, куда можно пойти из *v*.

**for** i **in** range(1, n + 1):

Рассматриваем все ребра, кроме того, которое ведет к предку *p*.

**if** i == p:

**continue**

Если имеется ребро из *v* в *i*, причем *i* уже пройдена (used[*i*] = 1), то в графе имеется цикл. Устанавливаем *flag* = 1.

**if** g[v][i] == 1:

**if** used[i] == 1:

flag = 1

Иначе продолжаем поиск в глубину из вершины *i*.

**else**:

dfs(i, v)

Основная часть программы. Читаем входные данные.

n, m = map(int, input().split())

g = [[0] \* (n + 1) **for** \_ **in** range(n + 1)]

used = [0] \* (n + 1)

Читаем входной неориентированный граф. Граф сохраняем в матрице смежности, поэтому повторяющиеся дороги будут учитываться только один раз.

**for** \_ **in** range(m):

a, b = map(int, input().split())

g[a][b] = g[b][a] = 1

Установим *flag* = 0, что означает отсутствие цикла в графе. Если цикл будет найден, значение переменной *flag* изменится на 1.

flag = 0

Запускаем поиск в глубину из вершины 1 (по условию задачи граф связный).

dfs(1)

В зависимости от значения переменной *flag* выводим ответ.

**if** flag == 1:

print(**"YES"**)

**else**:

print(**"NO"**)