Adventure Plan

يستعد كيان لخوض مغامرة. هذه المغامرة عبارة عن رحلة من الرأس المبدئي إلى الرأس النهائي، موجودة على بيان موجه لا يحتوي على دورات (DAG). يحتوي الرسم على n رأسًا مرقمة من 0 إلى $n-1$ و m حافة. الرأس المبدئي هو الرأس 0، وجميع الرؤوس قابلة للوصول منه.

كل حافة موجهة من u_i إلى v_i لها خاصيتان: l_i و r_i. هما على الترتيب الزمن الأدنى والأقصى اللازم لاجتياز هذه الحافة بأمان. بالتالي، يمكنه اختيار زمن مخطط t_i لأي قيمة صحيحة ضمن المجال $[l_i, r_i]$.

الكثير من أصدقاء كيان أيضًا يستعدون للمغامرة. كل واحد منهم سيسلك مسارًا مختلفًا من الرأس المبدئي حتى النهاية. ولضمان السلامة، يرغب أن يلتقي الأصدقاء الذين يسلكون مسارات مختلفة عند كل رأس **في نفس الوقت تمامًا**. أي أنه يريد أن تعيّن الأزمنة لكل الحواف بحيث يكون: لكل رأس u، جميع المسارات من الرأس المبدئي حتى u لها نفس المجموع الكلي للزمن.

نسمي البيان **آمنًا** إذا تحقق هذا الشرط. يُضمن أن البيان الابتدائي آمن.

Task 1

يرغب كيان في إضافة بعض الحواف الجديدة إلى البيان لجعله أكثر إثارة. سيجري q عملية من نوع **"إضافة حافة جديدة"**. كل عملية تعطيه حافة جديدة (u_i, v_i, l_i, r_i). تعلم أنه بعد إضافة هذه الحافة، يبقى البيان بدون دورات (DAG)، لكنك ليس متأكدًا إن كان سيبقى آمنًا. مطلوب منه أن يحدد ما إذا كان الرسم لا يزال آمنًا بعد إضافة هذه الحافة. إذا كان آمنًا، يجب إضافة الحافة إلى البيان إذا لم يكن آمنًا، يجب تجاهل هذه العملية.

Task 2

بعد إنهاء جميع العمليات، يحتاج إلى إخراج القيم المخططة t_i لكل الحواف (بما فيها الحواف الجديدة المقبولة) كبرهان على أنك متأكد أن البيان الجديد آمن.

Implementation Details

.You need to implement two procedures

:The first procedure you need to implement is `add_roads`

```
boolean[] add_roads(int32 N, int32 M, int32 Q,
    int32[] U, int32[] V,
    int32[] L, int32[] R,
    int32[] U2, int32[] V2,
    int32[] L2, int32[] R2);
```

- N: عدد الرؤوس؛

- M: عدد الحواف الابتدائية؛
- Q: عدد العمليات؛
- U, V: مصفوفتان بطول M، حيث $(U[i], V[i])$ تمثل الحافة الموجهة رقم i؛
- L, R: مصفوفتان بطول M، حيث $[L[i], R[i]]$ هو مجال الزمن الممكن للحافة i؛
- $U2$, $V2$, $L2$, $R2$: مصفوفات بطول Q تصف الحواف الجديدة؛
- هذا الإجراء يُستدعى مرة واحدة فقط لكل حالة اختبار في بداية البرنامج.

هذا الإجراء يجب أن يُرجع مصفوفة بطول Q، حيث العنصر i هو `true` إذا تمت إضافة الحافة رقم i، أو `false` إذا تم تجاهلها.

The second procedure you need to implement is `assign_times`:

```
int32[] assign_times();
```

- This procedure is called exactly once for each test case after calling `add_roads`.
- يجب أن يُرجع مصفوفة تحتوي على القيم t_i المخططة لكل الحواف الموجودة في البيان النهائي (بأي ترتيب صحيح).

Constraints

- $3 \le n \le 500$
- $n - 1 \le m \le 10^5$
- $0 \le q \le 500$
- $0 \le u_i < v_i < n$
- $1 \le l_i \le r_i \le 10^9$

Scoring

1. Subtask 1 (7 points): $n \le 3$
2. Subtask 2 (21 points): $q = 0$
3. Subtask 3 (12 points): $v_i = u_i + 1$
4. Subtask 4 (11 points): $l_i = r_i$
5. Subtask 5 (24 points): $n \le 100, m \le 100, q \le 100$
6. Subtask 6 (25 points): No additional constraints

Examples

Example 1:

Consider the following call.

```
add_roads(4, 4, 2,
      [0, 1, 0, 0],
      [1, 3, 3, 2],
      [1, 3, 9, 6],
      [5, 7, 14, 8],
      [2, 2],
      [3, 3],
      [7, 5],
      [11, 7]);
```

.The procedure should return `[false, true]`

.Consider the following input after `add_roads` is called

```
assign_times();
```

.The procedure should return `[5, 7, 12, 6, 6]`

Sample Grader

:The sample grader reads the input in the following format

- q Line 1: Three integers n, m, and
- Next m lines: four integers u_i, v_i, l_i, r_i describing each edge
- Next q lines: four integers u_i, v_i, l_i, r_i describing each operation