Highway Tolls

ביפן, הערים מחוברות ברשת כבישים. הרשת מכילה N ערים ו-M כבישים. כל כביש מחבר זוג ערים שונות. אין שני כבישים שמחברים בין אותו זוג ערים. הערים ממוספרות מ-0 עד $N-1$, והכבישים ממוספרים מ-0 עד $M-1$. בכל כביש אפשר לנסוע בשני הכיוונים. אפשר להגיע מכל עיר לכל עיר אחרת באמצעות הכבישים.

בכל כביש הנסיעה כרוכה בתשלום אגרה (toll). האגרה עבור כביש תלויה **בעומס התנועה** (**traffic**) באותו כביש. העומס בכביש יכול להיות **קל** (**light**) או **כבד** (**heavy**). כשהעומס קל, האגרה היא A ין (ין הוא המטבע היפני). כשהעומס כבד, האגרה היא B ין. מובטח שמתקיים $A < B$. שימו לב שהערכים A ו-B ידועים לכם.

יש לכם מכונה, שבהינתן מצב העומסים בכל הכבישים, מחשבת את האגרה הכוללת המינימלית שצריך לשלם כדי לנסוע בין זוג הערים S ו-T ($S \neq T$) תחת עומסים אלה.

אבל, המכונה היא רק אבטיפוס. הערכים של S ו-T קבועים (כלומר, מובנים בתוך המכונה) ואינכם יודעים אותם. אתם רוצים לגלות את S ו-T. כדי לעשות זאת, בכוונתכם לתת מצבי עומסים שונים למכונה, ולהשתמש באגרות שהיא פולטת כדי להסיק את S ו-T. מכיוון שהגדרת מצב עומסים היא פעולה יקרה, אינכם רוצים להשתמש במכונה פעמים רבות.

פרטי מימוש

עליכם לממש את הפונקציה הבאה:

```
find_pair(int N, int[] U, int[] V, int A, int B)
```

- N: מספר הערים.
- U ו-V: מערכים באורך M, כאשר M הוא מספר הכבישים שמחברים בין ערים. עבור כל i $(0 \leq i \leq M-1)$, הכביש i מחבר בין הערים U[i] ו-V[i].
- A: האגרה עבור כביש כאשר העומס בו קל.
- B: האגרה עבור כביש כאשר העומס בו כבד.
- פונקציה זו נקראת פעם אחת בדיוק בכל test case.
- שימו לב שהערך M הוא אורך המערכים, ואפשר לקבל אותו כפי שכתוב ב-implementation notice.

הפונקציה find_pair יכולה לקרוא לפונקציה הבאה:

```
int64 ask(int[] w)
```

- האורך של w חייב להיות M. המערך w מתאר את מצב העומסים.

- עבור כל i ($0 \le i \le M-1$), `w[i]` הוא מצב העומס בכביש i. הערך של `w[i]` חייב להיות 0 או 1.
  - `w[i] = 0` מסמן שהעומס בכביש i הוא קל.
  - `w[i] = 1` מסמן שהעומס בכביש i הוא כבד.
- פונקציה זו מחזירה את האגרה הכוללת המינימלית עבור נסיעה בין הערים S ו-T, תחת מצב העומסים שמוגדרים ב-w.
- מותר לקרוא לפונקציה זו לכל היותר 100 פעמים (בכל test case).

`find_pair` צריכה לקרוא לפונקציה הבאה כדי לדווח את התשובה:

```
answer(int s, int t)
```

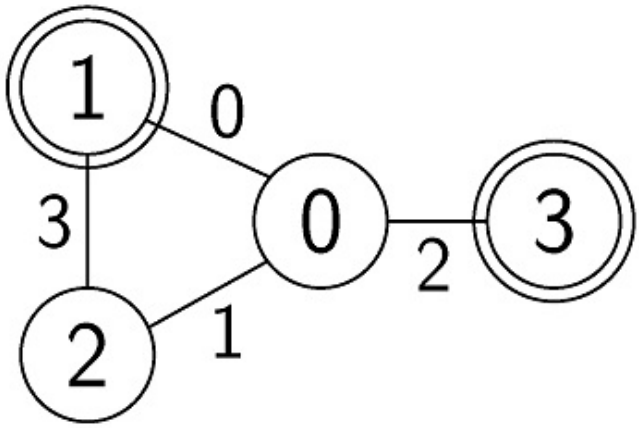
- `s` ו-`t` חייבים להיות הזוג S ו-T (הסדר לא משנה).
- פונקציה זו צריכה להיקרא פעם אחת בדיוק.

אם לא כל התנאים האלה מתקיימים, הפידבק על תוכניתכם יהיה **Wrong Answer**. אחרת, הפידבק יהיה **Accepted** והניקוד יחושב לפי מספר הקריאות ל-`ask` (ראו תת משימות).

דוגמה

נגדיר $N = 4, M = 4, U = [0, 0, 0, 1], V = [1, 2, 3, 2], A = 1, B = 3, S = 1, T = 3$.

הגריידר מבצע את הקריאה `find_pair(4, [0, 0, 0, 1], [1, 2, 3, 2], 1, 3)`.



באיור לעיל, הצלע עם המספר i מתאימה לכביש i. להלן קריאות אפשריות ל-`ask` וערכי ההחזרה המתאימים:

Call	Return
`ask([0, 0, 0, 0])`	2
`ask([0, 1, 1, 0])`	4
`ask([1, 0, 1, 0])`	5
`ask([1, 1, 1, 1])`	6

עבור הקריאה `ask([0, 0, 0, 0])`, העומס בכל כביש הוא קל והאגרה בכל כביש היא 1. המסלול הזול ביותר מ-$S = 1$ ל-$T = 3$ הוא $3 \to 0 \to 1$. האגרה הכוללת במסלול זה היא 2. לכן, הפונקציה מחזירה 2.

כדי שהתשובה תיחשב נכונה, הפונקציה `find_pair` צריכה לקרוא ל-`answer(1, 3)` או ל-`answer(3, 1)`.

הקובץ `sample-01-in.txt` ב-zip המצורף מכיל דוגמה זו. קיימים בו גם קלטים עבור דוגמאות נוספות.

מגבלות

- $2 \leq N \leq 90\,000$
- $1 \leq M \leq 130\,000$
- $1 \leq A < B \leq 1\,000\,000\,000$
- עבור כל $0 \leq i \leq M - 1$
  - $0 \leq U[i] \leq N - 1$
  - $0 \leq V[i] \leq N - 1$
  - $U[i] \neq V[i]$
- $(U[i], V[i]) \neq (U[j], V[j])$ וגם $(U[i], V[i]) \neq (V[j], U[j])$ $(0 \leq i < j \leq M - 1)$
- אפשר להגיע מכל עיר לכל עיר אחרת באמצעות הכבישים.
- $0 \leq S \leq N - 1$
- $0 \leq T \leq N - 1$
- $S \neq T$

בבעיה זו, הגריידר **אינו** אדפטיבי. כלומר S ו-T נקבעות בתחילת הריצה של הגריידר ואינן תלויות בשאילתות שאתם מבצעים.

תת משימות

1. (5 נקודות) אחד מבין S ו-T הוא 0, $N \leq 100$, $M = N - 1$
2. (7 נקודות) אחד מבין S ו-T הוא 0, $M = N - 1$
3. (6 נקודות) $M = N - 1$, $U[i] = i$, $V[i] = i + 1$ $(0 \leq i \leq M - 1)$
4. (33 נקודות) $M = N - 1$
5. (18 נקודות) $A = 1$, $B = 2$
6. (31 נקודות) ללא מגבלות נוספות

נניח שהפידבק על תוכניתכם הוא **Accepted**, והיא מבצעת X קריאות ל-ask. אז הניקוד P שתקבלו עבור ה-test case, כתלות בתת משימה, מחושב כך:

- תת משימה 1. $P = 5$.
- תת משימה 2. אם $X \leq 60$ אז $P = 7$. אחרת $P = 0$.
- תת משימה 3. אם $X \leq 60$ אז $P = 6$. אחרת $P = 0$.
- תת משימה 4. אם $X \leq 60$ אז $P = 33$. אחרת $P = 0$.
- תת משימה 5. אם $X \leq 52$ אז $P = 18$. אחרת $P = 0$.
- תת משימה 6.
  - אם $X \leq 50$ אז $P = 31$.
  - אם $51 \leq X \leq 52$ אז $P = 21$.
  - אם $53 \leq X$ אז $P = 0$.

שימו לב שהניקוד שלכם עבור כל תת משימה הוא הניקוד המינימלי מבין כל ה-test cases שבתוכה.

גריידר לדוגמה (Sample grader)

הגריידר לדוגמה קורא את הקלט בפורמט הבא:

- שורה 1: N ואחריו M ואחריו A ואחריו B ואחריו S ואחריו T.
- שורה $2 + i$ ($0 \leq i \leq M - 1$): $U[i]$ ואחריו $V[i]$.

אם הפידבק על תוכניתכם הוא **Accepted**, הגריידר לדוגמה מדפיס `Accepted: q`, כאשר q הוא מספר הקריאות ל-ask.

אם הפידבק על תוכניתכם הוא **Wrong Answer**, הוא מדפיס `Wrong Answer: MSG` כאשר MSG הוא אחד מהבאים:

- `answered not exactly once`: הפונקציה answer לא נקראה בדיוק פעם אחת.
- `w is invalid`: האורך של w שניתן ל-ask אינו M או ש-w[i] אינו 0 ואינו 1 עבור i כלשהו $(0 \leq i \leq M - 1)$.
- `more than 100 calls to ask`: הפונקציה ask נקראה יותר מ-100 פעמים.
- `{s, t} is wrong`: הפונקציה answer נקראה עם זוג שגוי של s ו-t.