

## Tecla e l'Ape Maia (Appetito Aracnide) (tecla [ 87 punti ])

Ape Maya è rimasta intrappolata in un nodo della tela di Tecla, un ragno molto temuto tra le api dell'alveare. Tecla si affretta ad afferrarla ma, quando giunge su quel nodo, si accorge di non avere appetito, e dice BLEAH''. Va detto che l'appetito dei ragni è molto particolare: ogni volta che percorrono un filamento della loro rete, essi invertono lo stato del loro stomaco tra SLURP'' e BLEAH''. Tecla deve quindi farsi un giretto nella rete sperando di tornare da Maya in stato SLURP''.

La tela di Tecla è composta da  $n$  nodi (numerati da 0 a  $n - 1$ ) connessi tra loro da  $m$  filamenti. Tecla e Ape Maya all'inizio si trovano entrambe nel nodo 0, e ogni filamento può essere attraversato da Tecla in entrambe le direzioni. Aiuta Tecla ad individuare una passeggiata funzionale al buon appetito!

### Input

Si legga l'input da `stdin`. La prima riga contiene  $T$ , il numero di testcase (istanze) da risolvere. Seguono  $T$  istanze del problema, dove ogni istanza è un diverso grafo  $G = (V, E)$ . Per ogni istanza, la prima riga contiene i due numeri interi  $n$  (numero di nodi) ed  $m$  (numero di filamenti della tela), separati da spazi. Seguono  $m$  righe, l' $i$ -esima delle quali riporta, separati da spazio, i due interi  $u$  e  $v$  che identificano i due nodi ai capi del filamento  $i$ -esimo.

### Output

Per ciascuna istanza, prima di leggere l'istanza successiva, scrivi su `stdout` il tuo output composto da due righe, contenenti:

Riga 1: il numero di spostamenti  $L$  che Tecla deve compiere nella sua passeggiata.

Riga 2: i nodi come visitati da Tecla nell'ordine (una sequenza di  $L + 1$  numeri separati da spazi, di cui il primo e l'ultimo sono 0 e più in generale un nodo verrà ripetuto se visitato più volte).

**Nota:** qualora non esista alcuna soluzione la vecchia e pigra Tecla rinuncia al pasto e decide di non muoversi (si scriva quindi uno 0 sia nella prima riga che nella seconda).

Prevediamo tre goal:

**goal 1 [decide]:** basta saper distinguere se esista ( $L > 0$ ) o meno ( $L = 0$ ) una qualche soluzione

**goal 2 [any\_sol]:** serve anche produrre una soluzione ogniqualvolta essa esista

**goal 3 [opt\_sol]:** fornire la più breve soluzione possibile (minimizzare  $L$ )

Oltre alle dimensioni delle istanze, ogni subtask precisa quanti punti competono ai vari goal. Il punteggio ottenuto per la generica istanza di quel subtask sarà la somma dei punti dei goal a risposta corretta (purchè si rispetti almeno il formato in tutte le righe di output, incluse quelle che competono agli altri goal – altrimenti salta il protocollo di comunicazione tra il tuo programma risolutore e il server).

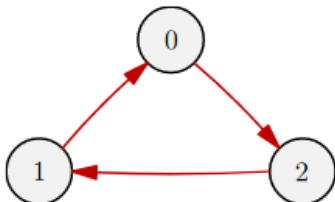
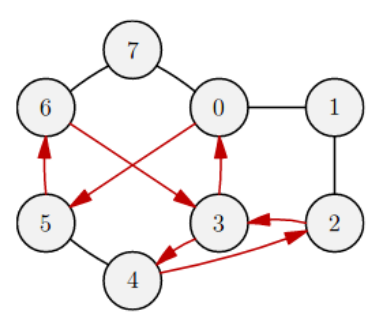
## Esempio di Input/Output

Input da `stdin`

-start-	8 12
3	0 1
3 3	1 2
0 1	2 3
1 2	3 0
2 0	2 4
6 5	3 4
0 1	4 5
0 3	5 6
2 1	6 7
4 5	7 0
2 3	0 5
-more-	6 3
	-end-

Output su `stdout`

3
0 2 1 0
0
0
5
0 3 2 4 3 0

Testcase 1: tela con percorso ottimo in rosso	Testcase 3: un percorso valido ma non ottimo
	

**Nota:** il Testcase 2 non consente invece alcuna soluzione, è quindi ottimo oltrechè corretto rispondere 0 su entrambe le righe.

## Subtask

Il tempo limite per istanza (ossia per ciascun testcase) è sempre di 2 secondi.

I testcase sono raggruppati nei seguenti subtask.

- [ 9 pts ← 3 istanze da 1 + 1 + 1 punti] **esempi\_testo:** i tre esempi del testo
- [ 6 pts ← 2 istanze da 1 + 1 + 1 punti] **hub\_0:** il nodo 0 è adiacente ad ogni altro nodo (con  $n \leq 100,000$ ,  $m \leq 200,000$ )
- [ 6 pts ← 2 istanze da 1 + 1 + 1 punti] **hub\_x:** qualche nodo è adiacente ad ogni altro nodo (con  $n \leq 100,000$ ,  $m \leq 200,000$ )
- [12 pts ← 4 istanze da 1 + 1 + 1 punti] **small:**  $n \leq 10$ ,  $m \leq 50$
- [18 pts ← 6 istanze da 1 + 1 + 1 punti] **medium:**  $n \leq 100$ ,  $m \leq 500$
- [18 pts ← 6 istanze da 1 + 1 + 1 punti] **big:**  $n \leq 1,000$ ,  $m \leq 5,000$
- [18 pts ← 6 istanze da 1 + 1 + 1 punti] **large:**  $n \leq 100,000$ ,  $m \leq 200,000$

In generale, quando si richiede la valutazione di un subtask vengono valutati anche i subtask che li precedono, ma si evita di avventurarsi in subtask successivi fuori dalla portata del tuo programma

che potrebbe andare in crash o comportare tempi lunghi per ottenere la valutazione completa della sottomissione. Ad esempio, chiamando<sup>1, 2</sup> :

```
rtal -s <URL> connect -x <token> -a size=hub_x  
tecla -- python my_solution.py
```

vengono valutati, nell'ordine, i subtask:

esempi\_testo, hub\_0, hub\_x.

Il valore di default per l'argomento size è large che include tutti i testcase.

---

<sup>1</sup><URL> server esame: [wss://ta.di.univr.it/esame](https://ta.di.univr.it/esame)

<sup>2</sup><URL> server esercitazioni e simula-prove: [wss://ta.di.univr.it/algo](https://ta.di.univr.it/algo)