C'è sempre un 2-set, con esattamente un 1-set nei DAG (`qkernels` [86 punti])

Un *digrafo* (o *grafo diretto*) è un qualsisi grafo $D = (V, A)$ dove ogni arco $a \in A$ sia una coppia *ordinata* di nodi, ossia distingua tra il suo nodo coda t_a e il suo nodo testa h_a. Per ogni nodo v, $N^+(v) := \{h_a : a \in A, t_a = v\}$ indica il *vicinato uscente di* v, e $N_1^+[v] := \{v\} \cup N^+(v)$ indica l'insieme di tutti quei nodi che possono essere raggiunti da v attraversando al più un arco (rispettandone la direzione, dal nodo coda al nodo testa). Più in generale, $N_k^+[v]$ indica l'insieme di tutti quei nodi che possono essere raggiunti da v attraversando al più k archi. Quindi $\{v\} = N_0^+[v] \subseteq N_1^+[v] \subseteq N_2^+[v] \subseteq \ldots$.

Un k-set è un qualsiasi sottoinsieme $S \subseteq V$ tale che:

1. per ogni $a \in A$, al più uno tra t_a ed h_a è contenuto in S
2. per ogni $v \in V$, c'è un $s \in S$ tale che $v \in N_k^+[s]$

Quindi, ogni k-set è anche un $(k+1)$-set. Il digrafo $(V, A) = (\{0, 1, 2\}, \{(0, 1), (1, 2), (2, 0)\})$, ossia il triangolo diretto, offre un esempio di digrafo privo di 1-set, mentre è invece vero che ogni digrafo ammette almeno un 2-set. La classica dimostrazione di questo fatto conferma la regola: "Quando un primo approccio ricorsivo non funziona al primo colpo, individuare meglio dove fallisce è la via per poterlo recuperare (ove possibile)."

Se hai dimestichezza nell'organizzare un ragionamento ricorsivo potrebbe riuscirti di ricostruire tale dimostrazione che ricorre su un sottografo indotto (ossia ottenuto dal grafo originale per eliminazione di un certo sottoinsieme di nodi). Tale dimostrazione si tradurrà in un algoritmo lineare che potrai parafrasare in un breve codice per totalizzare tutti i punti in palio in questo esercizio.

In caso contrario, o se per qualsivoglia ragione fatichi a mettere a punto il giusto contratto ricorsivo, potrai comunque ottenere gran parte dei punti dell'esercizio affrontando il problema su una classe particolari di digrafi: un **DAG** (acronimo per **d**irected **a**cyclic **g**raph) è un qualsiasi digrafo che sia privo di cicli diretti (ovvero di percorsi che riescano a rivisitare uno stesso nodo pur rispettando la direzione di ciascun arco). Un digrafo $D = (V, A)$ è un DAG se e solo se ammette un'ordinamento topologico dei suoi nodi ossia una funzione biunivoca $f : V \longmapsto \{0, 1, \ldots, n-1\}$ tale che $f(t_a) < f(h_a)$ per ogni $a \in A$. (Quì $n = |V|$ indica il numero di nodi.)

In realtà ogni DAG ammette non solo dei 2-sets ma anche un 1-set — potrebbe pertanto convenirti convincerti di questo risultato più forte, dove si è più vincolati e vi è maggiore struttura (al punto che in realtà ogni DAG ammette uno ed uno solo 1-set). Se persegui questo 1-set il corretto contratto induttivo si manifasterà da solo (di fatto non ti sarà nemmeno necessario ragionare esplicitamente in stile ricorsivo per ottenere la dimostrazione/algoritmo che lo produce).

Input

Si legga l'input da `stdin`. La prima riga contiene T, il numero di testcase (istanze) da risolvere. Seguono T istanze del problema. Ciascuna istanza è un digrafo, descritto in $m + 1$ righe dove m è il numero dei suoi archi: la prima riga riporta i numeri n ed m in questo ordine e separati da spazio, ogni altra riga riporta la testa e la coda di un diverso arco, in questo ordine e separati da spazio. (Gli n nodi sono etichettati coi primi n numeri naturali; ossia, $V = \{0, 1, \ldots, n-1\}$ e ogni arco è specificato da una coppia ordinata di numeri diversi in V.)

Output

Per ciascuna istanza, dopo aver trovato un 2-set S del digrafo in questione, e prima di leggere l'istanza successiva, scrivi su `stdout` una codifica di S strutturata su due righe: la prima riga riporta il numero

$|S|$ di nodi contenuti in S, la seconda riga riporta in un ordine qualsiasi e separati da spazi i nodi contenuti in S (ossia $|S|$ numeri diversi presi da $V = \{0, 1, ..., n-1\}$).

Esempio di Input/Output

Input da `stdin`

```
-start-
2
3 3
0 1
1 2
2 0
-more-
```

```
5 6
0 1
0 2
1 2
1 4
2 3
3 4

 -end-
```

Output su `stdout`

```
1
2
2
0 3
```

Subtask

Il tempo limite per istanza (ossia per ciascun testcase) è sempre di 1 secondo.

I testcase sono raggruppati nei seguenti subtask.

1. **[2 pts← 2 istanze da 1 punti] `esempi_testo`:** i due esempi del testo
2. **[9 pts← 9 istanze da 1 punti] `small_DAG_ordered`:** DAG con $0, 1, ..., n-1$ come ordinamento topologico, $n \leq 10$, $m \leq 25$
3. **[9 pts← 9 istanze da 1 punti] `medium_DAG_ordered`:** DAG con $0, 1, ..., n-1$ come ordinamento topologico, $n \leq 50$, $m \leq 200$
4. **[9 pts← 9 istanze da 1 punti] `big_DAG_ordered`:** DAG con $0, 1, ..., n-1$ come ordinamento topologico, $n \leq 300$, $m \leq 1000$
5. **[10 pts←10 istanze da 1 punti] `small_DAG`:** DAG con $n \leq 10$, $m \leq 25$
6. **[10 pts←10 istanze da 1 punti] `medium_DAG`:** DAG con $n \leq 50$, $m \leq 200$
7. **[10 pts←10 istanze da 1 punti] `big_DAG`:** DAG con $n \leq 300$, $m \leq 1000$
8. **[9 pts← 9 istanze da 1 punti] `small`:** digrafo qualsisi con $n \leq 10$, $m \leq 25$
9. **[9 pts← 9 istanze da 1 punti] `medium`:** digrafo qualsisi con $n \leq 50$, $m \leq 200$
10. **[9 pts← 9 istanze da 1 punti] `big`:** digrafo qualsisi con $n \leq 300$, $m \leq 1000$

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[1, 2] :

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

vengono valutati, nell'ordine, i subtask:

`esempi_testo`, `small_DAG_ordered`, `medium_DAG_ordered`, `big_DAG_ordered`, `small_DAG`.

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

[1] <URL> server esame: `wss://ta.di.univr.it/esame`

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