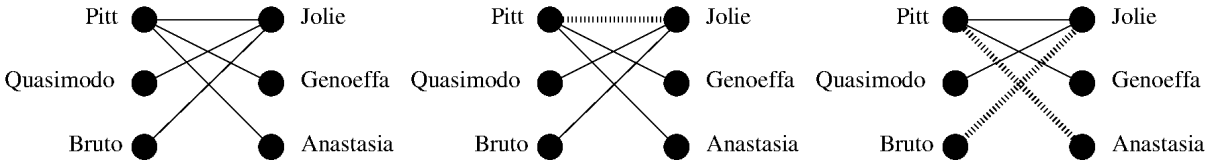
Max Bipartite Matching/Min Node Cover (`matching_bipartito` [54 punti])

Un *matching* in un grafo $G = (V, E)$ è un sottoinsieme $M \subseteq E$ degli archi tale che non vi siano due archi in M incidenti in uno stesso nodo.

Dato G, il problema del matching di massima cardinalità chiede di trovare un matching M di G il cui numero di archi $|M|$ sia massimo.

Ove sia inoltre fornito un valore $\text{val}(e)$ per ciascun arco $e = uv \in E$, il problema del matching di massimo valore chiede di trovare un matching M di G il cui valore totale $\sum_{e \in E} w_e$ sia massimo.

Come esempio del problema del matching di massima cardinalità su grafo bipartito, vorremmo trovare un massimo numero di matrimoni celebrabili nel paese in figura, dove gli archi indicano coppie di individui che sarebbero compatibili tra di loro.



Il paese istanza soluzione di Don Abbondio quella di Perpetua.

Si noti che il matching su cui si era intestardito Don Abbondio era massimale (nessun ulteriore matrimonio poteva essere aggiunto) senza essere di cardinalità massima (come comprovato dalla soluzione suggerita da Perpetua).

In generale il problema del matching su grafo bipartito $G = (U, V; E)$ trova espressione nella seguente formulazione di Programmazione Lineare Intera (PLI), dove si è introdotta una variabile x_{uv} per ogni arco $uv \in E$:

```
max sum_(uv in E) x_uv
   sum_(uv in delta(u)) x_uv <= 1 per ogni u in U
   sum_(uv in delta(v)) x_uv <= 1 per ogni v in V
   x_uv >= 0 e intera per ogni uv in E
```

Il *rillassamento continuo* di un problema di PLI come quello sopra è il problema di PL che si ottiene ignorando i vincoli di interezza. Nel caso del modello sopra (una formulazione viene chiamata modello quando non è specifica ad un'istanza ma vale per tutta una famiglia di istanze o problema), il suo rilassamento continuo è il seguente modello di PL:

```
max sum_(uv in E) x_uv
   sum_(uv in delta(u)) x_uv <= 1 per ogni u in U
   sum_(uv in delta(v)) x_uv <= 1 per ogni v in V
   x_uv >= 0 per ogni uv in E
```

Anche se in generale il valore ottimo del rilassamento può eccedere il valore ottimo della formulazione PLI, è noto che nel caso del problema del Bipartite Matching ogni soluzione ottima di base del rilassamento è comunque intera di suo. Questo significa che attraverso la formulazione intera viene in realtà risolto il problema di PLI in cui siamo in origine interessati.

Per l'istanza di Bipartite Matching del nostro paesino, si ottiene la seguente formulazione di PL (o di PLI), dove ciascun nodo è labellato dall'iniziale del nome della persona che esso rappresenta:

```
max x_(P J) + x_(P G) + x_(P A) + x_(Q A) + x_(B A)
   x_(P J) + x_(P G) + x_(P A) <= 1
   x_(P J)                                    <= 1
```

```
            x_(P G)                                 <= 1
                      x_(P A) + x_(Q A) + x_(B A) <= 1
                                x_(Q A)           <= 1
                                                x_(B A) <= 1
   x_(P J),  x_(P G),  x_(P A),  x_(Q A),  x_(B A) >= 0 (e intera, nella formulazione
di PLI) per ogni uv in E
```

Nel caso del grafo in figura il certificato che non esista un matching di cardinalità 3 è costituito dal node cover costituito dai due nodi P e J. Infatti, se ci fosse 3 archi senza estremi in comune allora non sarebbe possibile colpire tutti gli archi sparando solo su due nodi.

Input

Si legga l'input da `stdin`. La prima riga contiene T, il numero di testcase (istanze) da risolvere. Ogni istanza offre un diverso grafo bipartito $G = (U, V, E)$ come segue: nella prima rica sino scritti i numeri $|U|, |V|, |E|$ separati da spazio. Seguono $|E|$ righe, ciascuna a codificare un diverso arco in E. Il generico arco uv, con $u \in U := \{0, 1, ..., |U| - q\}$ e $v \in V := \{0, 1, ..., |V| - 1\}$ viene codificato nella sua riga stampando i numeri u e v, nell'ordine e separati da spazio.

Output

Per ciascuna istanza, prima di leggere l'istanza successiva o per concludere la sessione col server, scrivi su `stdout` il tuo output così strutturato:

in una prima riga si scrive `opt_val`, la cardinalità di un matching ottimo M^*. Seguono `opt_val` righe che codificano gli `opt_val` archi di G inclusi in M^*. Infine, in un ultima riga, si riportano i nodi di un minimo node cover, separati da spazi.

Esempio di Input/Output

Input da `stdin`

```
2
0 5
1 3
0 3
4
0 1 2 3
0 5
1 7
2 4
3 6
1 7
1 6
2 4
2 7
2 5
3 5
3 6
3 7
```

Output su `stdout`

```
2
0 5
1 3
0 3
4
0 5
1 7
2 4
3 6
0 1 2 3
```

Subtask

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

I testcase sono raggruppati nei seguenti subtask.

1. **[6 pts← 2 istanze da 1 + 1 + 1 punti] `esempi_testo`:** i due esempi del testo
2. **[15 pts← 5 istanze da 1 + 1 + 1 punti] `small`:** $m \leq 15$
3. **[12 pts← 4 istanze da 1 + 1 + 1 punti] `medium`:** $m \leq 50$
4. **[9 pts← 3 istanze da 1 + 1 + 1 punti] `big`:** $m \leq 100$
5. **[6 pts← 2 istanze da 1 + 1 + 1 punti] `large`:** $m \leq 1000$
6. **[6 pts← 2 istanze da 1 + 1 + 1 punti] `extra_large`:** $m \leq 10000$

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=medium
  matching_bipartito  -- python my_solution.py
```

vengono valutati, nell'ordine, i subtask:

`esempi_testo`, `small`, `medium`.

Il valore di default per l'argomento `size` è `extra_large` 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`