Filmati e canzoni (download)

Descrizione del problema

Monica ha regalato un nuovo hard disk a Mojito, il suo cagnolino. Si sa, Mojito è un grande amante di filmati e canzoni, quindi ha intenzione di riempire tutto lo spazio del suo hard disk scaricando quanti più filmati e canzoni possibili!

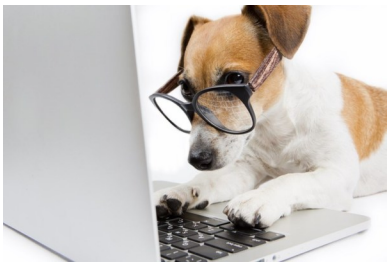


Figure 1: Mojito che sceglie filmati e canzoni da scaricare

È importante menzionare che Mojito preferisce di gran lunga i filmati alle canzoni, quindi comincerà subito a scaricare quanti più filmati possibile. Nel caso in cui lo spazio rimanente sull'hard disk non gli desse altra scelta, Mojito ripiegherà sulle canzoni e comincerà quindi a scaricarne fino a riempire completamente l'hard disk.

I gusti di Mojito non sono molto vari, i filmati che gli interessano hanno tutti la stessa dimensione, e lo stesso vale per le canzoni. Per l'esattezza, tutti i filmati hanno una dimensione di F byte e tutte le canzoni hanno una dimensione di C byte.

Sapendo che il nuovo hard disk ha una capacità di N byte, scrivi un programma che calcoli il **numero di filmati** ed il **numero di canzoni** che Mojito scaricherà, sapendo che il cane darà preferenza ai filmati.

Dati di input

La prima riga del file di input contiene un intero T, il numero di casi di test. Seguono T casi di test, numerati da 1 a T. Ogni caso di test è preceduto da una riga vuota.

Ciascun caso di test è composto da una sola riga, contenente tre numeri interi N, F, C separati da uno spazio, rispettivamente: la capacità in byte dell'hard disk, la dimensione in byte dei filmati e la dimensione in byte delle canzoni.

Dati di output

Il file di output deve contenere la risposta ai casi di test che sei riuscito a risolvere. Per ogni caso di test che hai risolto, il file di output deve contenere una riga con la dicitura

`Case #t: nf nc`

dove `t` è il numero del caso di test (a partire da 1) e i valori `nf`, `nc` sono il numero di filmati e canzoni che Mojito scaricherà.

Assunzioni

- $T = 6$, nei file di input che scaricherai saranno presenti esattamente 6 casi di test.
- $1 \leq N, F, C \leq 10\,000$.
- Non è detto che i filmati occupino più spazio delle canzoni.

Esempi di input/output

Input:

```
2

1000 300 10

1000 30  50
```

Output:

```
Case #1: 3 10
Case #2: 33 0
```

Spiegazione

Nel **primo caso d'esempio**, Mojito può scaricare 3 filmati e 10 canzoni per un occupazione totale di $3 \cdot 300 + 10 \cdot 10 = 900 + 100 = 1000$ byte e riempire così l'hard disk completamente. Non può invece scaricare 4 filmati, perché il totale verrebbe di $4 \cdot 300 = 1200$ che eccede la capacità dell'hard disk.

Nel **secondo caso d'esempio**, Mojito può scaricare 33 filmati per un occupazione totale di $33 \cdot 30 = 990$ byte. Nello spazio rimanente non è possibile scaricare canzoni.

Tornello olimpico (tornello)

Descrizione del problema

Questo settembre, all'*IIS G.B. Pentasuglia* di Matera, si terrà la fase finale delle Olimpiadi di Informatica. Come ogni anno Monica sarà presente e si occuperà di aspetti organizzativi legati alla logistica degli studenti come: assicurarsi che tutti siano presenti quando c'è da prendere un autobus, ritrovare e restituire la valigia perduta di qualche distratto, radunare gli studenti prima della gara in modo che nessuno rimanga per sbaglio in hotel, e così via.

Per ottimizzare l'efficienza con la quale svolgerà queste mansioni Monica ha pianificato di installare un **tornello**, simile a quello rappresentato in figura, agli ingressi che le interessa controllare (come l'ingresso della sede di gara, dell'hotel, e così via). Un tornello permette a Monica di essere informata in modo automatizzato di quando uno studente **entra** o **esce** da una stanza: in risposta a questi due tipi di eventi, il tornello manderà rispettivamente un segnale `+1` e un segnale `-1` sul PC di Monica. Naturalmente, il tornello non fornisce alcuna informazione su **chi** effettivamente è entrato/uscito.

Al file di prevedere i movimenti degli studenti, Monica ha scelto una stanza da monitorare e sta cercando di dedurre (a partire dalla sequenza di segnali ± 1 ricevuti dal tornello posto al suo ingresso) quale sia la risposta alla seguente domanda:

«Qual è il ***numero minimo di studenti diversi*** che hanno visitato la stanza?»

Nota bene: quando il tornello viene acceso (e comincia a "contare" gli ingressi e le uscite) è possibile che nella stanza *siano già presenti* alcuni studenti. Lo stesso vale quando il tornello viene spento: potrebbero *rimanere alcuni studenti* all'interno. Inoltre, per lo scopo di questo problema, supporremo che non ci siano ingressi/uscite secondarie per accedere alla stanza monitorata, e che nella stanza in questione sia installato un solo tornello.

Figure 1: Un esempio di tornello

Supponiamo per esempio che nel lasso di tempo in cui il tornello è operativo Monica riceva i segnali `-1 -1 -1`. In un caso di questo tipo, sarebbe possibile dedurre con certezza che **almeno 3 studenti** hanno visitato la stanza: infatti, se ne sono usciti 3, vuol dire che dentro ce n'erano almeno 3. È possibile che nella stanza fossero presenti inizialmente più di 3 studenti, ma con i dati a disposizione non possiamo dirlo con certezza.

Analogamente, nel caso in cui Monica ricevesse i segnali `+1 +1 +1`, si potrebbe affermare di nuovo che almeno 3 studenti hanno visitato la stanza. Infatti, dal momento che il tornello ha girato sempre nello stesso verso, non è pensabile che uno stesso studente sia entrato più volte (sarebbe dovuto uscire, prima di poter rientrare!) quindi sappiamo che il tornello è stato attivato sempre da persone diverse.

Più complicato è invece il caso in cui Monica riceve i segnali `+1 -1 +1`. In questo caso infatti è possibile che uno **stesso studente** sia entrato, poi uscito e successivamente rientrato, attivando il tornello 3 volte. Quindi in questa situazione il minimo numero di studenti diversi che hanno visitato la stanza è 1.

Aiuta Monica scrivendo un programma che, dati i segnali ottenuti da un tornello posizionato all'ingresso di una stanza, calcoli il **minimo numero di studenti diversi** che hanno visitato quella stanza.

Dati di input

La prima riga del file di input contiene un intero T, il numero di casi di test. Seguono T casi di test, numerati da 1 a T. Ogni caso di test è preceduto da una riga vuota.

Ciascun caso di test è composto da due righe. La prima contiene il numero intero N: il numero di segnali ricevuti dal tornello. La seconda contiene N interi (ciascuno uguale a `+1` o `-1`) separati da spazio: la sequenza di segnali ricevuti dal tornello.

Dati di output

Il file di output deve contenere la risposta ai casi di test che sei riuscito a risolvere. Per ogni caso di test che hai risolto, il file di output deve contenere una riga con la dicitura

`Case #t: c`

dove `t` è il numero del caso di test (a partire da 1) e il valore `c` è il minimo numero di studenti che hanno visitato la stanza.

Assunzioni

- $T = 12$, nei file di input che scaricherai saranno presenti esattamente 12 casi di test.
- $1 \leq N \leq 1\,000\,000$.

Esempi di input/output

Input:

```
4

3
-1 -1 -1

3
+1 +1 +1

3
+1 -1 +1

10
-1 +1 +1 +1 -1 +1 +1 +1 -1 +1
```

Output:

```
Case #1: 3
Case #2: 3
Case #3: 1
Case #4: 5
```

Gerarchie di tutor (gerarchie)

Descrizione del problema

Le Olimpiadi Italiane di Informatica sono gestite da Luigi con l'aiuto di N ragazzi e ragazze che prendono il nome di **tutor**. Questi membri dello staff (che in passato hanno partecipato alle OII come voi!) con grande impegno e dedizione mettono a beneficio delle OII le loro conoscenze tecniche. In così tanti però è difficile organizzarsi, per questo già da molti anni è stata definita una gerarchia molto precisa che, se rispettata, massimizza l'efficienza della squadra.

Al tutor i-esimo è stato associato un **tutor di riferimento** R_i, che è in pratica un altro tutor incaricato di supervisionare il suo lavoro. In cima alla gerarchia si trova il **tutor leader**, al quale non è assegnato alcun tutor di riferimento.

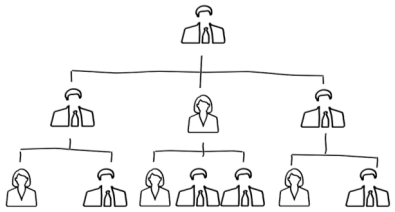


Figure 1: Esempio di gerarchia

L'efficiacia di questa politica è massima quando ogni tutor di riferimento risulta *più competente di tutti i tutor che è incaricato di supervisionare*. Negli ultimi anni però, Luigi si è accorto che qualcosa non va: sembra infatti che i livelli di competenza C_i dei vari tutor siano cambiati e che quindi non ci sia più il bilanciamento necessario. Urge una riorganizzazione! Però, essendo tra amici, ed essendo la burocrazia molto costosa, sarà necessario prestare molta attenzione alle promozioni fatte.

La promozione di un tutor i consiste nello scambio di ruolo tra i ed il suo tutor di riferimento, di fatto facendo salire i nella gerarchia e facendo scendere il suo supervisore. Ognuna di queste operazioni ha un costo e, per ribilanciare l'intera struttura, Luigi vorrebbe farne il minor numero possibile.

Essendo tra amici, Luigi vuole evitare che un tutor venga continuamente promosso e declassato durante la riorganizzazione. Per evitare ogni problema si impone quindi la seguente regola: se un tutor viene *promosso*, tutti i **successivi** scambi di ruolo che lo coinvolgono dovranno essere ancora delle *promozioni*.

Aiuta Luigi a calcolare il numero di minimo di scambi che deve fare per riorganizzare la gerarchia senza però mai violare la sua regola!

Dati di input

La prima riga del file di input contiene un intero T, il numero di casi di test. Seguono T casi di test, numerati da 1 a T. Ogni caso di test è preceduto da una riga vuota.

Ciascun caso di test è composto da $N + 1$ righe: la prima contiene un numero intero N, il numero di tutor. Ciascuna delle seguenti N righe contiene due interi R_i e C_i separati da spazio: rispettivamente l'indice del tutor di riferimento ed il livello di competenza dell'i-esimo tutor (con i che va da 0 a $N - 1$).

Dati di output

Il file di output deve contenere la risposta ai casi di test che sei riuscito a risolvere. Per ogni caso di test che hai risolto, il file di output deve contenere una riga con la dicitura

```
Case #t: s
```

dove `t` è il numero del caso di test (a partire da 1) e il valore `s` è il minimo numero di scambi per bilanciare la gerarchia rispettando le regole.

Assunzioni

- $T = 19$, nei file di input che scaricherai saranno presenti esattamente 19 casi di test.
- $1 \leq N \leq 1000$.
- $0 \leq C_i \leq N - 1$.
- $0 \leq R_i \leq N - 1$ se i non è il tutor leader, $R_i = -1$ se i è il tutor leader.
- I valori di competenza dei tutor sono numeri da 0 a $N - 1$ e sono tutti distinti.
- Non è consentito declassare un tutor **dopo** una sua promozione ma è possibile declassarlo **prima**.
- Esiste uno ed un solo tutor leader.
- Risalendo la catena di supervisori di un qualsiasi tutor, si raggiunge sempre il tutor leader.

Esempi di input/output

Input:

```
2

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

8
3 6
2 4
-1 7
2 5
5 1
2 0
3 2
5 3
```

Output:

```
Case #1: 6
Case #2: 2
```

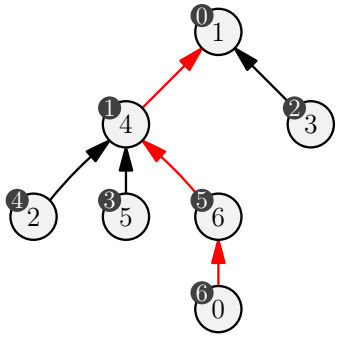
Spiegazione

Nel **primo caso d'esempio** sono presenti 7 tutor, una possibile sequenza di scambi di ruolo ottima è la seguente:

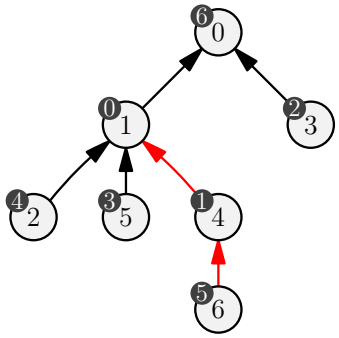
- Tre promozioni del tutor 0 facendolo diventare tutor leader e declassando prima il tutor 6, poi il tutor 4 e poi il tutor 1.
- Due promozioni del tutor 5, declassando il tutor 4 e poi il tutor 1.
- Una promozione del tutor 4, declassando il tutor 1.

I tutor che hanno subito una promozione (0, 5 e 4) non hanno *successivamente* subito alcun declassamento, la situazione finale risulta bilanciata in quanto tutti i tutor hanno più competenze di coloro che sorvegliano.

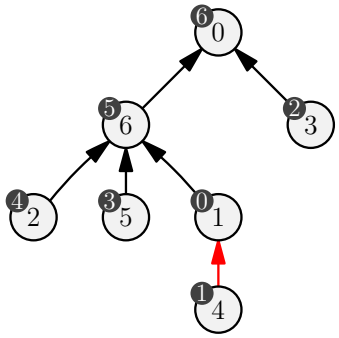
Nella figura seguente, il valore nel nodo **bianco** è l'**indice** del tutor, mentre il valore nel pallino **grigio** è il valore di **competenza** di quel tutor.



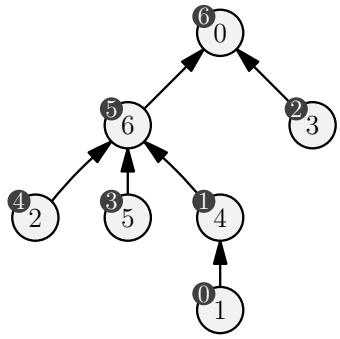
Tripla promozione di 0 declassando 6, 4 e 1.



Doppia promozione di 6 declassando 4 e 1.



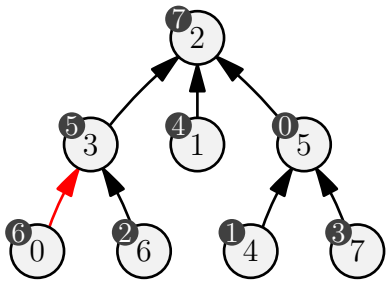
Promozione di 4 declassando 1.



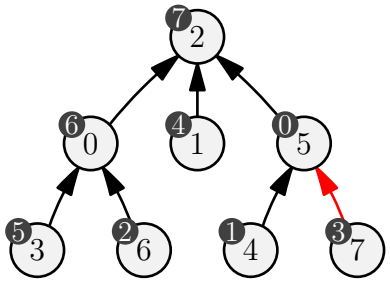
Situazione finale.

Figure 2: Primo caso d'esempio

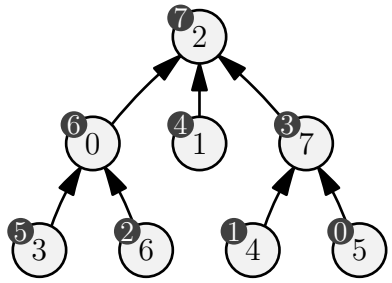
Nel **secondo caso d'esempio** sono sufficienti i seguenti due scambi di ruolo per bilanciare l'organizzazione.



Promozione di 0 declassando 3.



Promozione di 7 declassando 5.



Situazione finale.

Figure 3: Secondo caso d'esempio

Processori multicore (multicore)

Descrizione del problema

Quest'anno Mojito, il cane di Monica, è stato incaricato di gestire il sistema di gara delle Olimpiadi di Informatica. Essendo molto preoccupato di fare bella figura, ha deciso di destinare tutto il budget possibile nelle CPU che verranno usate dal sistema di gara.

Mojito vuole puntare ad avere il maggior numero possibile di *core* a disposizione, così da gestire la gara nel migliore dei modi. Va notato infatti che, quando tanti partecipanti sottopongono le loro soluzioni al sistema di gara, avere tanti core disponibili è utile perché significa poter valutare *contemporaneamente* più soluzioni.

Per esempio: con una CPU da 8 core si possono valutare 8 soluzioni contemporaneamente. Non è detto però che una sola CPU da tanti core sia sufficiente a gestire l'intera gara: per questo motivo, Mojito vuole acquistare tante CPU in modo da sommare la loro capacità di calcolo (i core disponibili). Infatti, montando una CPU da 8 core ed un'altra da 4 core, si possono valutare contemporaneamente ben 12 soluzioni!

Figure 1: Alcune CPU che Mojito potrebbe comprare

Sapendo di avere un budget pari a B centesimi di euro, Mojito ha deciso di fare i suoi acquisti su un mercatino dell'usato online. In questo sito ha trovato N inserzioni: l'i-esima inserzione fa riferimento ad una specifica CPU con un certo prezzo P_i ed un certo numero di core C_i.

Mojito è interessato a trovare il **massimo numero di core** che si possono acquistare con il budget disponibile, ovvero: vuole massimizzare la somma dei valori C_i delle CPU selezionate facendo attenzione al fatto che la somma dei rispettivi valori P_i non superi B.

Aiuta Mojito scrivendo un programma che scelga quali CPU comprare!

Dati di input

La prima riga del file di input contiene un intero T, il numero di casi di test. Seguono T casi di test, numerati da 1 a T. Ogni caso di test è preceduto da una riga vuota.

Ciascun caso di test è composto da $N + 1$ righe. La prima riga contiene i due interi N, B separati da uno spazio: rispettivamente il numero di inserzioni ed il budget a disposizione. Ciascuna delle seguenti N righe contiene due interi C_i, P_i separati da uno spazio: rispettivamente il numero di core ed il prezzo dell'i-esima CPU.

Dati di output

Il file di output deve contenere la risposta ai casi di test che sei riuscito a risolvere. Per ogni caso di test che hai risolto, il file di output deve contenere una riga con la dicitura

`Case #t: nc`

dove `t` è il numero del caso di test (a partire da 1) e il valore `nc` è il numero di core che Mojito può acquistare.

Assunzioni

- $T = 27$, nei file di input che scaricherai saranno presenti esattamente 27 casi di test.
- $1 \leq N \leq 300$.
- $1 \leq C_i \leq 200$.
- $1 \leq P_i \leq B \leq 10^9$.

Esempi di input/output

Input:

```
2

1 100
2 10

4 10000
4 5000
6 7000
8 4000
10 8000
```

Output:

```
Case #1: 2
Case #2: 12
```

Spiegazione

Nel **primo caso d'esempio**, Mojito ha trovato un'unica inserzione di CPU dual core, e non può far meglio che comprare quella.

Nel **secondo caso d'esempio**, Mojito potrebbe comprare la CPU con 10 core, ma non gli rimarrebbe abbastanza budget per comprare altro (e similmente se comprasse la CPU con 6 core). In alternativa, può comprare le CPU con 4 e 8 core, per un totale di 12 core e un prezzo di $5000 + 4000 = 9000$ che è sotto il suo budget di 10 000.