Drop-It! (dropit)

Limite di tempo: 1.0 secondi
Limite di memoria: 256 MiB

William ha scaricato il gioco *Drop-It!* per il suo cellulare. In questo gioco alcuni blocchi di varia lunghezza vengono fatti cadere da una gru uno sull'altro, e cadendo si comportano nel seguente modo:

1. Se un blocco cade su uno lungo uguale, si ferma distruggendosi assieme a quello su cui è caduto;
2. Se un blocco cade su uno più lungo, si ferma lì e sopra di esso se ne crea uno lungo quanto la differenza tra i due che si sono scontrati;
3. Se un blocco cade su uno più piccolo, si riduce della lunghezza di quello più piccolo, distrugge il più piccolo e continua a cadere su quello successivo.

Per aiutare William a pianificare meglio la strategia, scrivi un programma che calcoli la situazione della pila dopo la caduta di N blocchi con lunghezze L_i per $i = 0, \ldots, N$.

Implementazione

Dovrai sottoporre esattamente un file con estensione `.c`, `.cpp` o `.pas`.

> ☞ Tra gli allegati a questo task troverai un template (`dropit.c`, `dropit.cpp`, `dropit.pas`) con un esempio di implementazione da completare.

Se sceglierai di utilizzare il template, dovrai implementare la seguente funzione:

C/C++	`int cadi(int N, int L[], int P[]);`
Pascal	`function cadi(N: longint; var L, P: array of longint): longint;`

In cui:

- L'intero N rappresenta il numero di blocchi che vengono fatti cadere dalla gru.
- L'array `L`, indicizzato da 0 a $N-1$, contiene le lunghezze dei pezzi fatti cadere dalla gru nell'ordine in cui vengono fatti cadere.
- L'array `P`, indicizzato da 0 a $M-1$, dovrà essere riempito dalla funzione con le lunghezze dei blocchi che rimangono impilati alla fine del procedimento.
- La funzione dovrà restituire il numero M di blocchi rimasti alla fine, che verrà stampato sul file di output assieme all'array `P`.

Dati di input

Il file `input.txt` è composto da due righe. La prima riga contiene l'unico intero N. La seconda riga contiene gli N interi L_i separati da uno spazio.

Dati di output

Il file `output.txt` è composto da due righe. La prima riga contiene l'unico intero M. La seconda riga contiene gli N interi P_i separati da uno spazio.

Assunzioni

- $1 \leq N \leq 100\,000$.
- $1 \leq L_i \leq 10\,000$ per ogni $i = 0 \ldots N-1$.

Assegnazione del punteggio

Il tuo programma verrà testato su diversi test case raggruppati in subtask. Per ottenere il punteggio relativo ad un subtask, è necessario risolvere correttamente tutti i test relativi ad esso.

- **Subtask 1** [**10 punti**]: Casi d'esempio.
- **Subtask 2** [**20 punti**]: $N \leq 10$.
- **Subtask 3** [**40 punti**]: $N \leq 1000$.
- **Subtask 4** [**30 punti**]: Nessuna limitazione specifica.

Esempi di input/output

input.txt	output.txt
4 42 42 42 20	3 42 20 22
4 17 13 4 15	3 17 2 15

Spiegazione

Nel **primo caso di esempio**, si applica prima la regola (1) svuotando la pila finora formata e infine la regola (2) creando un ulteriore blocco lungo 22.

Nel **secondo caso di esempio**, si applicano le regole (2), (1), (3) e (2) ottenendo la successione di stati:

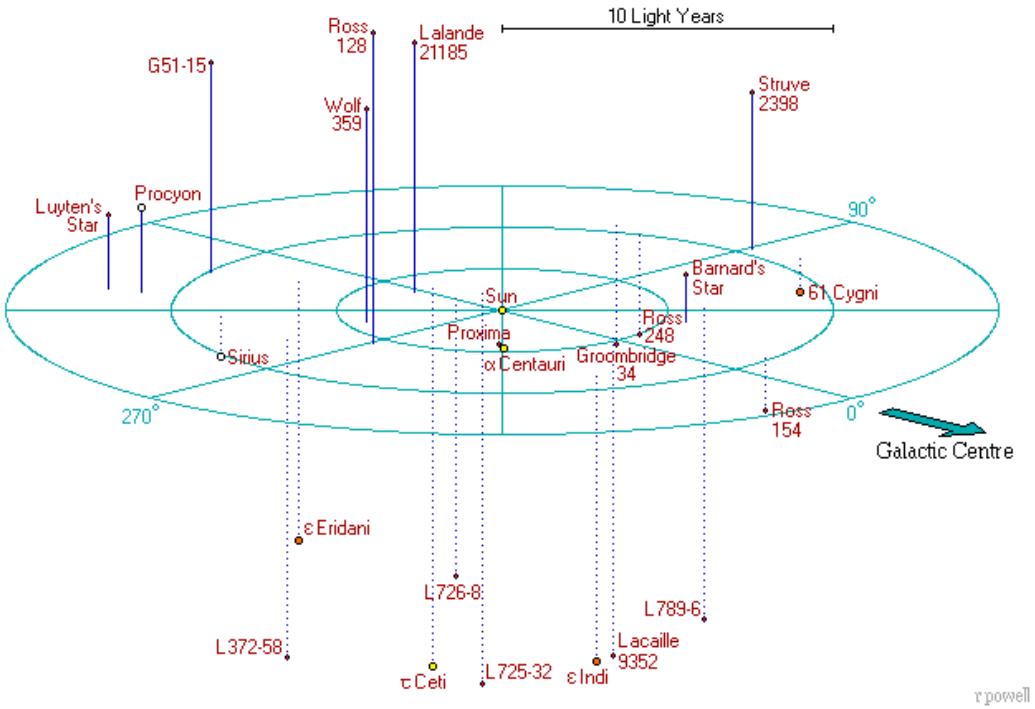
```
17 <-- 13
17 13 4 <-- 4
17 13 <-- 15
17 <-- 2
17 2 15
```

Anno luce (`annoluce`)

Limite di tempo:	1.0 secondi
Limite di memoria:	256 MiB

Grazie ad una recente ricerca che ha confermato l'esistenza delle onde gravitazionali, sempre più persone si stanno interessando allo spazio. Purtroppo però, lo spazio è ancora una realtà poco accessibile alle persone comuni. Sebbene sia un po' demoralizzato da questo fatto, William è convinto che sia possibile sfruttare la recente attenzione mediatica delle onde gravitazionali per pubblicizzare un business: ha deciso infatti di aprire una startup di viaggi interstellari.

C'è da dire però che, a parte il *Sole* che è la stella a noi più vicina, le altre stelle sono piuttosto distanti. *Proxima Centauri*, la "seconda stella più vicina", dista dal Sole ben 4.24 anni luce: questo vuol dire che sarebbero necessari più di quattro anni per raggiungere questa stella! (supponendo di poter viaggiare alla velocità della luce).



Le 33 stelle che distano al massimo 12.5 anni luce dal Sole
(Richard Powell CC BY-SA 2.5 via Wikimedia Commons)

William pensa di riuscire a costruire un'astronave in grado di viaggiare alla velocità della luce (ha trovato un tutorial su YouTube che gli sembra piuttosto convincente) e ha perciò acquistato un telescopio per tracciare una mappa 3D della Via Lattea. Ogni stella è indicata nella mappa 3D mediante un punto (x, y, z) dello spazio. Il Sole è sempre presente nella mappa ed è sempre identificato dal punto $(0, 0, 0)$.

Scrivi un programma che data la mappa stellare sia in grado di rispondere a Q query: ogni query fornisce un numero intero D e chiede quante sono le stelle raggiungibili avendo a disposizione D anni di viaggio.

Implementazione

Dovrai sottoporre esattamente un file con estensione `.c`, `.cpp` o `.pas`.

> ☞ Tra gli allegati a questo task troverai un template (`annoluce.c`, `annoluce.cpp`, `annoluce.pas`) con un esempio di implementazione da completare.

Se sceglierai di utilizzare il template, dovrai implementare le seguenti funzioni:

C/C++	`void mappatura(int N, int X[], int Y[], int Z[]);` `int query(int D);`
Pascal	`procedure mappatura(N: longint; var X, Y, Z: array of longint);` `function query(D: longint) : longint;`

In cui:

- L'intero N rappresenta il numero di stelle nella mappa.
- Gli array `X`, `Y` e `Z`, indicizzati da 0 a $N-1$, contengono le coordinate delle N stelle. In particolare la stella i-esima si trova nelle coordinate (`X[i]`, `Y[i]`, `Z[i]`).
- La funzione `mappatura` verrà richiamata una sola volta all'inizio del programma.
- Ogni chiamata alla funzione `query(D)` dovrà restituire il numero di stelle che distano al massimo D anni luce dal Sole.

Dati di input

Il file `input.txt` è composto da $N+Q+2$ righe. La prima riga contiene l'unico intero N. Le successive N righe contengono le coordinate X_i, Y_i, Z_i dell'i-esima stella, separate da spazio. La successiva riga contiene l'unico intero Q. Le successive Q righe contengono i valori di D delle relative query.

Dati di output

Il file `output.txt` è composto da Q righe contenente un intero ciascuna: la risposta alla relativa query.

Assunzioni

- $1 \leq N, Q \leq 100\,000$.
- $0 \leq X_i, Y_i, Z_i < 2^{30}$ per ogni $i = 0 \dots N-1$.
- L'unità degli assi x, y, z è l'anno luce.
- $0 \leq D < 2^{31}$ per ogni chiamata a `query(D)`.
- Il valore D è espresso in anni luce.

Assegnazione del punteggio

Il tuo programma verrà testato su diversi test case raggruppati in subtask. Per ottenere il punteggio relativo ad un subtask, è necessario risolvere correttamente tutti i test relativi ad esso.

- **Subtask 1** [**10 punti**]: Casi d'esempio.
- **Subtask 2** [**20 punti**]: `Y[i]` $= 0$ per ogni i. Invece che nello spazio tridimensionale, le stelle sono tutte su una retta!
- **Subtask 3** [**20 punti**]: `Z[i]` $= 0$ per ogni i. Invece che nello spazio tridimensionale, siamo su un piano bidimensionale!
- **Subtask 4** [**10 punti**]: $N, Q, D \leq 10$.
- **Subtask 5** [**10 punti**]: $N \leq 100$; $D < 10\,000$.
- **Subtask 6** [**10 punti**]: $Q \leq 100$; $D < 10\,000$.
- **Subtask 7** [**20 punti**]: Nessuna limitazione specifica.

Esempi di input/output

input.txt	output.txt
3 0 0 0 1 1 1 2 2 2 5 0 1 2 3 4	1 1 2 2 3
5 1 2 3 4 5 6 0 0 0 9 9 9 7 1 1 3 20 8 9	5 3 4

Spiegazione

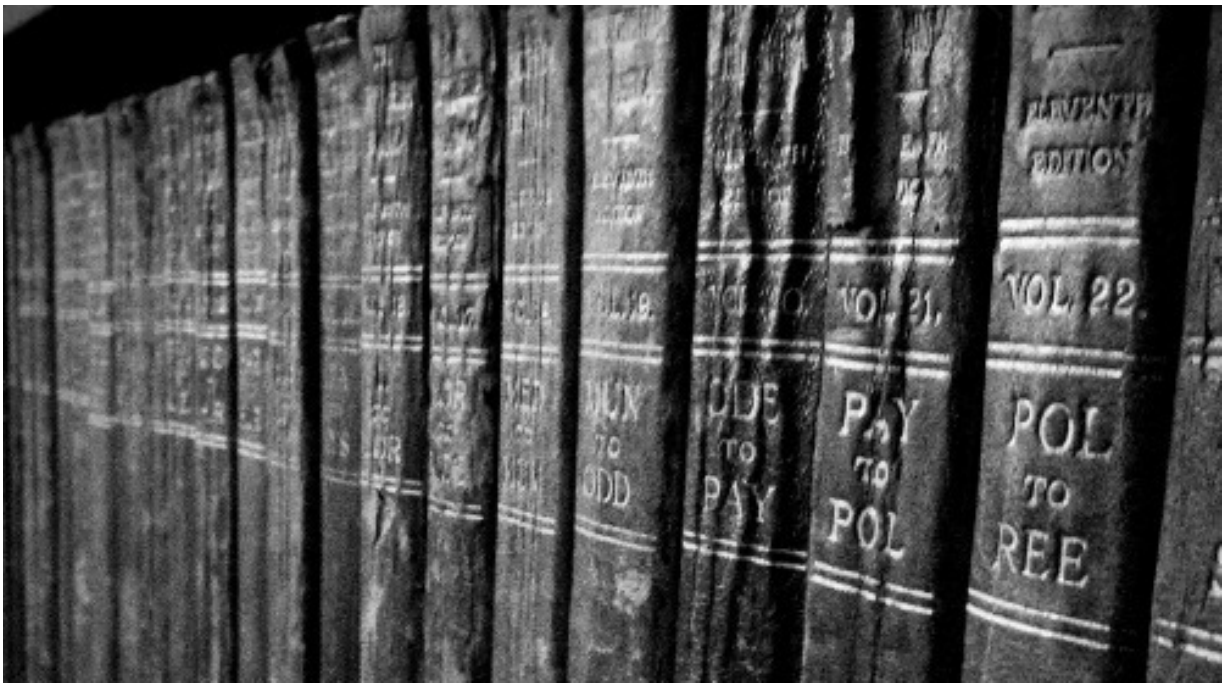
Nel **primo caso di esempio**, le 3 stelle distano dal sole rispettivamente: 0 anni luce, $\sqrt{3}$ anni luce e $2\sqrt{3}$ anni luce.

Nel **secondo caso di esempio**, le 5 stelle distano dal sole rispettivamente: $\sqrt{14}$ anni luce, $\sqrt{77}$ anni luce, 0 anni luce, $9\sqrt{3}$ anni luce e $\sqrt{51}$ anni luce.

Enciclopedia olimpica (`enciclopedia`)

Limite di tempo:	1.0 secondi
Limite di memoria:	256 MiB

Giorgio è sempre stato appassionato di enciclopedie. Nella sua vasta collezione si trovano alcune delle più rare enciclopedie esistenti. L'enciclopedia più famosa di sempre è probabilmente l'undicesima edizione dell'Enciclopedia Britannica. Ovviamente, Giorgio ha tutti e 29 i volumi:



Encyclopaedia Britannica - undicesima edizione
(CC BY-SA 3.0 via Wikimedia Commons)

Giorgio ha deciso di creare una nuova enciclopedia che (al posto delle parole) utilizzi i *nomi dei task* delle Olimpiadi di Informatica. La "definizione" di ogni termine sarà una dettagliata spiegazione della soluzione del task relativo. Ad esempio, alla voce "poldo", Giorgio scriverà una spiegazione dettagliata del problema "La dieta di Poldo" delle *selezioni regionali 2004* delle Olimpiadi di Informatica.

Sul dorso di ciascun volume va scritta un'*indicazione* di quali termini si troveranno al suo interno. Ai fini di questo task, per un dato volume che inizia con una parola p e finisce con una parola q, definiremo tale indicazione come la coppia $(p^\star, q^\star)$ *più corta possibile* e che soddisfa i seguenti criteri. Innanzitutto $p^\star$ deve essere un prefisso della parola p che *non* è prefisso di q, $q^\star$ un prefisso della parola q che *non* è prefisso di p. Se una di queste due cose non dovesse essere possibile (ad esempio con $p =$ "ciao" e $q =$ "ciaone", avremmo $q^\star =$ "ciaon" mentre $p^\star$ non esiste) allora diremo che il prefisso inesistente è impostato uguale all'intera parola (nell'esempio: $p^\star = p$).

Per esempio, se il primo e l'ultimo task contenuti nell'i-esimo volume fossero rispettivamente `caldaia` e `carrucola`, allora nel dorso dell'i-esimo volume sarebbe necessario scrivere `CAL-CAR`. In questo modo, se cercassimo il task `cantina`, sapremmo di doverlo andare a prendere dal volume i-esimo.

La coppia $(p^\star, q^\star)$ però dovrà anche rispettare un ulteriore criterio: il prefisso $p^\star$ dovrà anche *non* essere prefisso dell'ultima parola q dell'eventuale volume precedente, ed il prefisso $q^\star$ dovrà anche *non* essere prefisso della prima parola p dell'eventuale volume successivo.

Per esempio, se il volume $(i+1)$-esimo cominciasse con `cartella`, sarebbe necessario cambiare l'indicazione `CAL-CAR` del volume i-esimo con `CAL-CARR` per evitare ambiguità. Allo stesso modo, se `calamaro` fosse l'ultima parola del volume $(i-1)$-esimo, allora sarebbe necessario cambiare `CAL-CARR` con `CALD-CARR`.

Giorgio ha una lista di N nomi dei task di tutte le scorse edizioni delle Olimpiadi di Informatica (è garantito che tutti i nomi sono diversi), e ha già deciso il numero K di volumi che comporranno la sua enciclopedia (è garantito che N è divisibile per K). Aiuta Giorgio a decidere le indicazioni da stampare sul dorso di ciascun volume in modo che (nel frattempo che le spiegazioni dettagliate di ciascun task vengono preparate) possa almeno prenotare le K copertine dal suo artigiano di fiducia.

Implementazione

Dovrai sottoporre esattamente un file con estensione `.c`, `.cpp` o `.pas`.

> ☞ Tra gli allegati a questo task troverai un template (`enciclopedia.c`, `enciclopedia.cpp`, `enciclopedia.pas`) con un esempio di implementazione da completare.

Se sceglierai di utilizzare il template, dovrai implementare le seguenti funzioni:

C/C++	`void rilega(int N, int K, char* parole[]);` `void volume(char* S);`
Pascal	`procedure rilega(N, K: longint; var parole: array of ansistring);` `procedure volume(var S: ansistring);`

In cui:

- L'intero N rappresenta il numero di task che finiranno nell'enciclopedia.
- L'intero K rappresenta il numero di volumi dai quali l'enciclopedia sarà composta.
- L'array `parole`, indicizzato da 0 a $N-1$, contiene i nomi dei task.
- La funzione non restituirà alcun valore, ma dovrà bensì richiamare K volte la funzione `volume` per stampare l'indicazione di un volume sul file di output.
- Il parametro S della funzione `volume` è l'indicazione da stampare sul dorso del volume. Deve essere composta da due gruppi di lettere alfabetiche maiuscole separati da un carattere "`-`" nel mezzo. Tali gruppi di lettere devono essere "lessicograficamente crescenti", ovvero, il primo gruppo deve essere lessicograficamente inferiore del secondo.

Dati di input

Il file `input.txt` è composto da $N+1$ righe. La prima riga contiene due interi N e K separati da spazio. Le seguenti N righe contengono le parole da inserire nell'enciclopedia, una parola per riga.

Dati di output

Il file `output.txt` è composto da K righe contenenti le indicazioni da far stampare sul dorso dei volumi dell'enciclopedia olimpica.

Assunzioni

- $4 \leq N \leq 10\,000$.
- K è un divisore proprio di N (quindi $1 \leq K < N$).
- Gli N nomi sono composti da soli caratteri alfabetici minuscoli: niente spazi.
- Gli N nomi vengono forniti già ordinati lessicograficamente e senza ripetizioni.
- $1 \leq |\texttt{parola}[\texttt{i}]| \leq 30$ per ogni $i = 0 \ldots N-1$. Ovvero la lunghezza di ciascuna parola è ≤ 30.

Assegnazione del punteggio

Il tuo programma verrà testato su diversi test case raggruppati in subtask. Per ottenere il punteggio relativo ad un subtask, è necessario risolvere correttamente tutti i test relativi ad esso.

- **Subtask 1** [**10 punti**]: Casi d'esempio.
- **Subtask 2** [**40 punti**]: $K = 2$.
- **Subtask 3** [**20 punti**]: $N \leq 10$.
- **Subtask 4** [**30 punti**]: Nessuna limitazione specifica.

Esempi di input/output

input.txt	output.txt
9 3 anello barca calamaro caldaia cantina carrucola cartella dinosauri energia	A-CALA CALD-CARR CART-E
4 2 xyzxx xzqh yay zzzzzzzzz	XY-XZ Y-Z
4 2 a ab abc abcd	A-AB ABC-ABCD

Codici interessanti (`interessante`)

Limite di tempo:	1.0 secondi
Limite di memoria:	256 MiB

Giorgio sta ora studiando i *codici interessanti*, che sono sequenze di N cifre 0 o 1 tali che per ogni possibile ragione $x = 1, \ldots, 10$ e valore iniziale $i = 0, \ldots, N - 3x - 1$, le cifre nelle posizioni individuate dalla corrispondente progressione aritmetica $(i, i + x, i + 2x, i + 3x)$ *non sono tutte uguali*. Per esempio, i seguenti codici di 10 cifre sono interessanti:

```
0001001000
0001100011
1010110111
```

Mentre questi non lo sono:

```
1011101011
1000010110
1101000100
```

per via delle progressioni aritmetiche (i, x) rispettivamente $(0, 3)$, $(1, 1)$, $(2, 2)$. Quanti sono i codici interessanti di N cifre e che contengono esattamente K cifre 1 al loro interno?

Implementazione

Dovrai sottoporre esattamente un file con estensione `.c`, `.cpp` o `.pas`.

> ☞ Tra gli allegati a questo task troverai un template (`interessante.c`, `interessante.cpp`, `interessante.pas`) con un esempio di implementazione da completare.

Se sceglierai di utilizzare il template, dovrai implementare la seguente funzione:

C/C++	`int conta(int N, int K);`
Pascal	`function conta(N, K: longint): longint;`

In cui:

- L'intero N rappresenta il numero di cifre totali.
- L'intero K rappresenta il numero di cifre che devono essere pari a 1.
- La funzione dovrà restituire il numero R di codici interessanti di N cifre contenenti K cifre 1, che verrà stampato sul file di output.

Dati di input

Il file `input.txt` è composto da un'unica riga contenente i due interi N e K.

Dati di output

Il file `output.txt` è composto da un'unica riga contenente un unico intero, la risposta a questo problema.

Assunzioni

- $1 \leq K \leq N \leq 2000$.
- N e K sono tali per cui $NR \leq 7\,000\,000$.

Assegnazione del punteggio

Il tuo programma verrà testato su diversi test case raggruppati in subtask. Per ottenere il punteggio relativo ad un subtask, è necessario risolvere correttamente tutti i test relativi ad esso.

- **Subtask 1** [**10 punti**]: Casi d'esempio.
- **Subtask 2** [**20 punti**]: $N \leq 10$.
- **Subtask 3** [**25 punti**]: $N \leq 22$.
- **Subtask 4** [**25 punti**]: $N \leq 100$.
- **Subtask 5** [**20 punti**]: Nessuna limitazione specifica.

Esempi di input/output

input.txt	output.txt
1 1	1
8 2	4

Spiegazione

Nel **primo caso di esempio**, l'unico codice interessante corrispondente è `1`.

Nel **secondo caso di esempio**, i codici interessanti corrispondenti sono:

```
00010010
00011000
00100100
01001000
```

Filiali bilanciate (`filiali`)

Limite di tempo:	1.0 secondi
Limite di memoria:	256 MiB

La *OIS S.p.A.* vuole aprire F nuove filiali scelte ciascuna tra N possibili città. Le N città considerate sono tutte disposte lungo l'*Autostrada del Sole*, ciascuna a un diverso chilometro K_i per $i = 0, \ldots, N$. Data una possibile scelta delle filiali, definiamo il suo *bilanciamento* come la minima distanza tra due qualsiasi delle filiali scelte (la distanza tra due filiali è pari alla differenza di chilometraggio delle relative città).

Qual è il massimo bilanciamento ottenibile per una scelta di F filiali tra le N città date?

Implementazione

Dovrai sottoporre esattamente un file con estensione `.c`, `.cpp` o `.pas`.

> ☞ Tra gli allegati a questo task troverai un template (`filiali.c`, `filiali.cpp`, `filiali.pas`) con un esempio di implementazione da completare.

Se sceglierai di utilizzare il template, dovrai implementare la seguente funzione:

C/C++	`int apri(int N, int F, int K[]);`
Pascal	`function apri(N, F: longint; var K: array of longint): longint;`

In cui:

- L'intero N rappresenta il numero di città possibili per le filiali.
- L'intero F rappresenta il numero di filiali da aprire.
- L'array `K`, indicizzato da 0 a $N - 1$, contiene il chilometro corrispondente a ciascuna città.
- La funzione dovrà restituire il miglior bilanciamento per F filiali, che verrà stampato sul file di output.

Dati di input

Il file `input.txt` è composto da due righe. La prima riga contiene i due interi N ed F. La seconda riga contiene gli N interi K_i separati da uno spazio.

Dati di output

Il file `output.txt` è composto da un'unica riga contenente un unico intero, la risposta a questo problema.

Assunzioni

- $2 \leq F \leq 1000$.
- $1 \leq N \leq 1\,000\,000$.
- $0 \leq K_i \leq K_{i+1} < 2^{31}$ per ogni $i = 0 \ldots N - 1$.

Assegnazione del punteggio

Il tuo programma verrà testato su diversi test case raggruppati in subtask. Per ottenere il punteggio relativo ad un subtask, è necessario risolvere correttamente tutti i test relativi ad esso.

- **Subtask 1** [**10 punti**]: Casi d'esempio.
- **Subtask 2** [**10 punti**]: $F = 3$.
- **Subtask 3** [**10 punti**]: $F = 4$.
- **Subtask 4** [**10 punti**]: $F \leq 7$.
- **Subtask 5** [**20 punti**]: $N, F \leq 100$.
- **Subtask 6** [**20 punti**]: $N \leq 5000$.
- **Subtask 7** [**20 punti**]: Nessuna limitazione specifica.

Esempi di input/output

input.txt	output.txt
5 2 11 27 32 32 53	42
6 3 0 40 60 85 90 100	40

Spiegazione

Nel **primo caso di esempio**, conviene aprire le filiali nella prima e nell'ultima città.

Nel **secondo caso di esempio**, due filiali vengono aperte nella prima e nell'ultima città e la rimanente filiale può essere aperta indifferentemente nella seconda o nella terza città.

Carta fedeltà (`miglia`)

Limite di tempo:	1.0 secondi
Limite di memoria:	256 MiB

Giorgio ha appena ottenuto una borsa con cui può andare a visitare una qualunque università straniera. Il viaggio di andata e ritorno gli verrà rimborsato a patto di non prendere più di K voli in totale, Giorgio vuole quindi sfruttare questa occasione per accumulare più *miglia* possibile sulla sua carta fedeltà *Alimpiadi*!

La compagnia aerea *Alimpiadi* effettua M tratte (unidirezionali) che collegano un totale di N aereoporti. Ciascuna tratta ha un suo valore in miglia, e non è detto che la tratta dall'aeroporto X all'aeroporto Y abbia lo stesso valore della tratta dall'aeroporto Y all'aeroporto X (o che entrambe queste tratte esistano). Inoltre, non è garantito che prendendo solo i voli della *Alimpiadi* sia possibile raggiungere qualunque aeroporto.

Aiuta Giorgio calcolando quante miglia al massimo può accumulare effettuando un percorso di *esattamente* K voli che parta e ritorni dalla sua Torino (la città numero 0)!

Implementazione

Dovrai sottoporre esattamente un file con estensione `.c`, `.cpp` o `.pas`.

> ☞ Tra gli allegati a questo task troverai un template (`miglia.c`, `miglia.cpp`, `miglia.pas`) con un esempio di implementazione da completare.

Se sceglierai di utilizzare il template, dovrai implementare la seguente funzione:

C/C++	`int vola(int K, int N, int M, int da[], int a[], int V[]);`
Pascal	`function vola(K, N, M: longint; var da, a, V: array of longint): longint;`

In cui:

- L'intero K rappresenta il numero di voli che Giorgio deve prendere.
- L'intero N rappresenta il numero di aeroporti collegati dalla *Alimpiadi*.
- L'intero M rappresenta il numero di diverse tratte effettuate dalla *Alimpiadi*.
- Gli array `da`, `a` e `V`, indicizzati da 0 a $M-1$, contengono rispettivamente gli aeroporti di partenza e di arrivo e il valore in miglia di ciascuna delle tratte.
- La funzione dovrà restituire il massimo numero di miglia accumulabili, che verrà stampato sul file di output.

Dati di input

Il file `input.txt` è composto da $M+1$ righe. La prima riga contiene i tre interi K, N, M. Le successive M righe contengono ciascuna i tre interi `da[`i`]`, `a[`i`]`, `V[`i`]` separati da uno spazio.

Dati di output

Il file `output.txt` è composto da un'unica riga contenente un unico intero, la risposta a questo problema.

Assunzioni

- $2 \leq K \leq 100$.
- $2 \leq N \leq 1000$.
- $2 \leq M \leq 10\,000$.
- $0 \leq \texttt{V[}i\texttt{]} \leq 100\,000$ per ogni $i = 0 \ldots N-1$.
- $0 \leq \texttt{da[}i\texttt{]}, \texttt{a[}i\texttt{]} \leq N-1$ e $\texttt{da[}i\texttt{]} \neq \texttt{a[}i\texttt{]}$ per ogni $i = 0 \ldots M-1$.
- Possono esistere più tratte che collegano esattamente gli stessi aeroporti, possibilmente con valore in miglia diverso.
- Esiste almeno un percorso chiuso di K voli che parte dall'aeroporto 0.
- I percorsi possono passare più volte per le stesse tratte e per gli stessi aeroporti.

Assegnazione del punteggio

Il tuo programma verrà testato su diversi test case raggruppati in subtask. Per ottenere il punteggio relativo ad un subtask, è necessario risolvere correttamente tutti i test relativi ad esso.

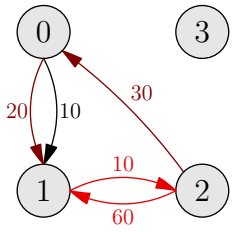
- **Subtask 1** [**10 punti**]: Casi d'esempio.
- **Subtask 2** [**20 punti**]: $K \leq 3$.
- **Subtask 3** [**20 punti**]: $K \leq 20$ e ogni aeroporto ha *al più due* voli in uscita.
- **Subtask 4** [**30 punti**]: $\texttt{V[}i\texttt{]} \leq 1$ per ogni $i = 0 \ldots N-1$.
- **Subtask 5** [**20 punti**]: Nessuna limitazione specifica.

Esempi di input/output

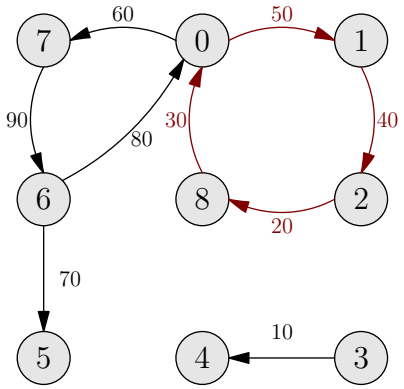
input.txt	output.txt
9 4 5 0 1 10 0 1 20 1 2 10 2 1 60 2 0 30	270
4 9 9 0 1 50 1 2 40 2 8 20 8 0 30 0 7 60 7 6 90 6 0 80 6 5 70 3 4 10	140

Spiegazione

Nel **primo caso di esempio**, sono possibili i due percorsi 0 – 1 – 2 – 0 – 1 – 2 – 0 – 1 – 2 – 0 oppure 0 – 1 – 2 – 1 – 2 – 1 – 2 – 1 – 2 – 0, ma il secondo offre il guadagno in miglia migliore:



Nel **secondo caso di esempio**, il percorso seguito (nonché l'unico accettabile) è 0 – 1 – 2 – 8 – 0:



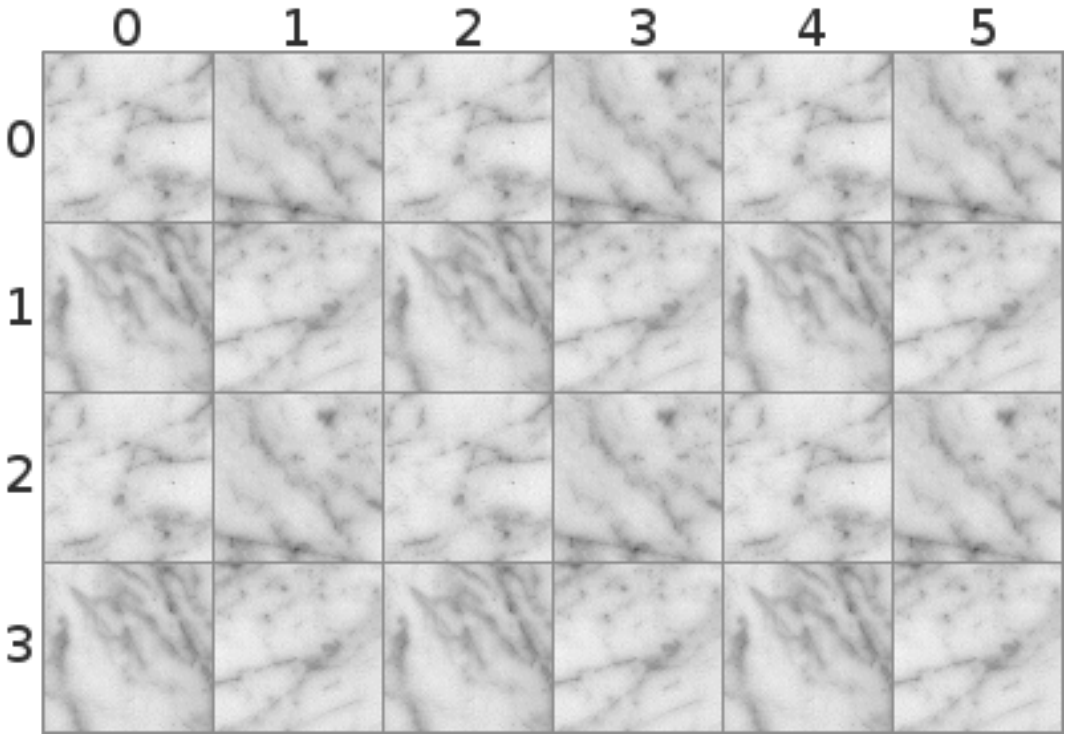
Labigriglia (`labigriglia`)

Limite di tempo:	2.0 secondi
Limite di memoria:	512 MiB

La quarantaduesima edizione del campionato annuale delle formiche robot sta per cominciare, e quest'anno è compito di Giorgio decidere la prova che i concorrenti dovranno affrontare.

Dato il pasticcio che hanno combinato gli organizzatori della scorsa edizione (chi potrebbe mai dimenticare l'*Arena con il formichiere*?), Giorgio ha deciso di puntare su qualcosa di più semplice: allestirà una griglia a mo' di labirinto (la *Labigriglia*) e premierà la formica che la percorrerà nel modo più efficiente.

Una labigriglia non è altro che una pavimentazione formata da $N \times M$ mattonelle quadrate. Ad esempio, una labigriglia valida con $N = 4$ e $M = 6$ è visibile nella seguente figura:



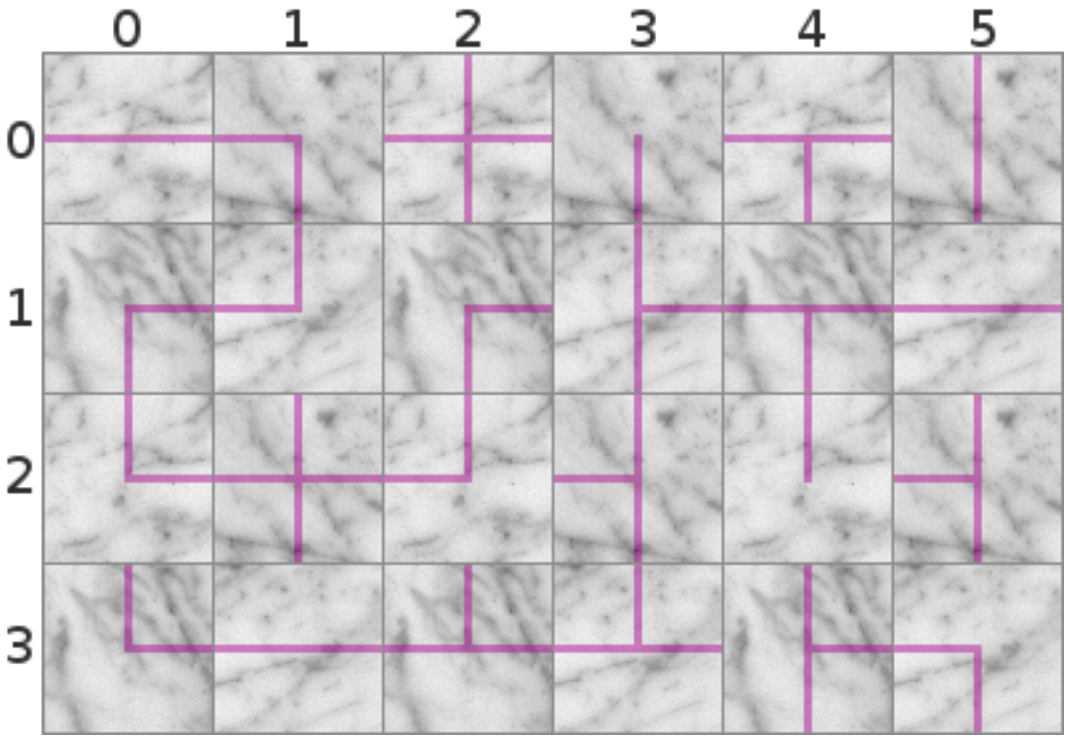
L'obiettivo delle formiche robot è quello di partire dallo spigolo in alto a sinistra della mattonella $(0, 0)$ e arrivare allo spigolo in basso a destra della mattonella $(N - 1, M - 1)$ cercando però di non "calpestare" troppe mattonelle. La formica vincente è quella che calpesta il minor numero di mattonelle.

> ☞ Le formiche robot possono spostarsi da una mattonella all'altra tramite i lati o gli spigoli. Una formica può quindi spostarsi al massimo verso 8 mattonelle (dallo stesso punto di partenza).

Per rendere più interessante il gioco, Giorgio ha comprato un insetticida per formiche robot, e ha intenzione di spruzzarlo su alcune mattonelle. Una formica robot non può calpestare l'insetticida.

Se il giorno della gara Giorgio sarà di buon umore l'insetticida non verrà spruzzato. Se invece Giorgio sarà di cattivo umore, allora sceglierà una o più mattonelle e comincierà ad avvelenarle seguendo questa tecnica: all'inizio dirigerà il getto dell'insetticida verso il centro della mattonella, dopodiché continuerà a spruzzare muovendosi in una delle 4 direzioni: verso l'alto, verso destra, verso il basso, verso sinistra.

Quindi, se Giorgio si dovesse sentire particolarmente malvagio, la "traccia" lasciata dall'insetticida potrà arrivare a formare un simbolo + nella mattonella. Ad esempio, nella seguente labigriglia ci sono due mattonelle posizionate in $(0, 2)$ e in $(2, 1)$ che sono state riempite di veleno!



Per far capire ai partecipanti dove si trova il veleno e dove no, Giorgio definisce il "fattore di velenosità" per ciascuna mattonella. Questo fattore è inizialmente pari a 0 per tutte le mattonelle. Durante l'applicazione dell'insetticida il fattore aumenta come segue:

- Dal centro *verso l'alto*: la velenosità della mattonella cresce di 1;
- Dal centro *verso destra*: la velenosità della mattonella cresce di 2;
- Dal centro *verso il basso*: la velenosità della mattonella cresce di 4;
- Dal centro *verso sinistra*: la velenosità della mattonella cresce di 8.

Nota: questo valore non indica quindi *quanto* è velenosa una mattonella, bensì *dove* è velenosa.

> ☞ In pratica, il fattore di velenosità è un numero binario di 4 bit. Se il bit meno significativo vale 1, allora l'insetticida è stato spruzzato dal centro verso l'alto; se il secondo bit meno significativo vale 1, allora l'insetticida è stato spruzzato dal centro verso destra; e così via.

Aiuta Giorgio a organizzare la prossima edizione del campionato annuale delle formiche robot! Scrivi un programma che determini se una data labigriglia può essere risolta da una formica robot e, in tal caso, che calcoli anche il *minimo numero di mattonelle* che essa dovrà calpestare per farlo.

Nota: se si passa più di una volta sulla stessa mattonella è necessario contarla di nuovo!

Implementazione

Dovrai sottoporre esattamente un file con estensione `.c`, `.cpp` o `.pas`.

> ☞ Tra gli allegati a questo task troverai un template (`labigriglia.c`, `labigriglia.cpp`, `labigriglia.pas`) con un esempio di implementazione da completare.

Se sceglierai di utilizzare il template, dovrai implementare la seguente funzione:

C/C++	`int cammina(int N, int M, int griglia[][MAXN]);`
Pascal	`type matrix = array of array of longint;` `function cammina(N, M: longint; var griglia: matrix): longint;`

In cui:

- Gli interi N e M rappresentano rispettivamente il numero di righe e di colonne della labigriglia.
- La matrice `griglia`, indicizzata da 0 a $N-1$ per le rige e da 0 a $M-1$ per le colonne, descrive la labigriglia. In particolare `griglia[i][j]` è il fattore di velenosità della mattonella (i, j).
- La funzione dovrà restituire il numero minimo di mattonelle che una formica robot deve calpestare per risolvere la labigriglia, oppure l'intero -1 se non fosse possibile risolverla.

Dati di input

Il file `input.txt` è composto da $N+1$ righe. La prima riga contiene gli interi N e M separati da spazio. Le successive N righe contengono ciascuna M interi, e descrivono la matrice `griglia`.

Dati di output

Il file `output.txt` è composto da un'unica riga contenente l'intero -1 qualora non esistesse una soluzione, oppure un intero positivo: la risposta al problema.

Assunzioni

- $1 \leq N, M \leq 1000$.
- $0 \leq$ `griglia[i][j]` ≤ 15 per ogni i, j.

Assegnazione del punteggio

Il tuo programma verrà testato su diversi test case raggruppati in subtask. Per ottenere il punteggio relativo ad un subtask, è necessario risolvere correttamente tutti i test relativi ad esso.

- **Subtask 1** [**10 punti**]: Casi d'esempio.
- **Subtask 2** [**10 punti**]: `griglia[i][j]` $= 0$ per ogni i, j. Niente veleno!
- **Subtask 3** [**35 punti**]: $\min(N, M) = 1$. Diciamo che è più un *labicorridoio*.
- **Subtask 4** [**20 punti**]: $N, M \leq 10$.
- **Subtask 5** [**25 punti**]: Nessuna limitazione specifica.

Esempi di input/output

input.txt	output.txt
4 6 10 12 15 4 14 5 6 9 6 7 14 10 3 15 9 13 1 13 3 10 11 11 7 12	14
4 6 10 12 15 4 14 5 6 9 6 7 14 10 3 15 9 13 1 13 2 10 11 11 7 12	13

Spiegazione

Il **primo caso di esempio** corrisponde alla seconda figura presente nel testo. Un percorso che passa per 14 mattonelle nella labigriglia in questione è il seguente: