



**ABC 2018**

Testi dei problemi

## Massima latenza II (latenza2)

Limite di tempo: 3.0 secondi  
 Limite di memoria: 256 MiB

Dopo il precedente tentativo di copiare durante la maturità - rivelatosi un flop -, le nuove classi quinte dell'ITIS Paleocapa vogliono migliorare il sistema ideato da Luca tre anni fa per ridurre ulteriormente la latenza tra due smartphone.

Ricapitoliamo brevemente la situazione: i commissari esterni hanno spento la rete Wi-Fi dell'istituto ma non hanno ritirato gli smartphone ai candidati. Quindi, per riuscire a scambiarsi informazioni segretamente, gli studenti vogliono sfruttare la comunicazione via Bluetooth tra i loro telefoni. Come è ben noto, il protocollo Bluetooth richiede che due dispositivi vengano prima accoppiati.

Nel 2015 la classe di Luca aveva eseguito il numero minimo di accoppiamenti tra smartphone affinché nessuno restasse escluso. Poiché il sistema si è rivelato parecchio fragile (bastava infatti che uno smartphone si spengesse dopo aver esaurito la batteria per rendere inservibile tutta l'infrastruttura), quest'anno la nuova classe di maturandi ha eseguito qualche accoppiamento in più.


Gli  $N$  studenti della classe, ciascuno con il proprio smartphone, hanno preparato  $M$  accoppiamenti Bluetooth. Una volta completata questa operazione, due telefoni accoppiati possono comunicare scambiandosi dati con una *latenza* (ovvero, con un ritardo tra l'invio e la ricezione) pari a un certo numero di millisecondi, che dipende dalla velocità dei due telefoni.

Qualora due smartphone non possano comunicare direttamente perché non accoppiati, la latenza tra essi è data dalla somma delle varie latenze lungo il "percorso" di smartphone accoppiati che è necessario attraversare. La comunicazione transita sempre per il percorso migliore, ovvero quello che minimizza la latenza complessiva.

La preoccupazione di Luca, che è stato contattato dagli attuali maturandi come consulente, è sempre la stessa: due smartphone potrebbero essere molto distanti e quindi potrebbero esserci significativi ritardi nella comunicazione. Aiutalo a valutare la situazione: quanto vale la *massima latenza* in millisecondi tra due smartphone?

### Implementazione

Dovrai sottoporre un unico file con estensione `.cpp` o `.c`.

 Tra gli allegati a questo task troverai un template (`latenza2.cpp` e `latenza2.c`) con un esempio di implementazione.

Dovrai implementare la seguente funzione:

#### ■ Funzione `latenza`

```
C/C++ | int latenza(int M, int M, int A[], int B[], int L[]);
```

- L'intero  $N$  rappresenta il numero di studenti.
- L'intero  $M$  rappresenta il numero di accoppiamenti Bluetooth effettuati.
- Gli array  $A$ ,  $B$  e  $L$ , indicizzati da  $0$  a  $M - 1$ , contengono alla posizione  $i$  la seguente informazione: gli smartphone  $A[i]$  e  $B[i]$  sono accoppiati e la latenza tra loro vale  $L[i]$  millisecondi.
- La funzione dovrà restituire la massima latenza in millisecondi tra due smartphone.

Il grader chiamerà la funzione `latenza` e ne stamperà il valore restituito sul file di output.

## Grader di prova

Allegata a questo problema è presente una versione semplificata del grader usato durante la correzione, che potete usare per testare le vostre soluzioni in locale. Il grader di esempio legge i dati da `stdin`, chiama la funzione che dovete implementare e scrive su `stdout`, secondo il seguente formato.

Il file di input è composto da  $M + 1$  righe, contenenti:

- Riga 1: gli interi  $N$  e  $M$ , separati da uno spazio.
- Righe 2,  $\dots$ ,  $M + 1$ : i tre interi  $A[i]$ ,  $B[i]$  e  $L[i]$  per  $i = 0, \dots, M - 1$ .

Il file di output è composto da un'unica riga, contenente:

- Riga 1: il valore restituito dalla funzione `latenza`.

## Assunzioni

- $2 \leq N \leq 2000$ .
- $N - 1 \leq M \leq 100\,000$ .
- $1 \leq A[i], B[i] \leq N$  per ogni  $i = 0 \dots M - 1$ .
- L'accoppiamento coinvolge sempre due smartphone diversi:  $A[i] \neq B[i]$  per ogni  $i = 0 \dots M - 1$ .
- Ogni accoppiamento è riportato solo una volta ed è per sua natura bidirezionale.
- $1 \leq L[i] \leq 1000$  per ogni  $i = 0 \dots M - 1$ .

## 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 [ 0 punti]**: Casi d'esempio.
- **Subtask 2 [15 punti]**:  $N \leq 2000$  e gli accoppiamenti sono stati eseguiti in modo tale che gli smartphone sono connessi "in linea" come nello schema seguente:



- **Subtask 3 [55 punti]**:  $N \leq 500$  e  $M \leq 5000$ .
- **Subtask 4 [15 punti]**:  $N \leq 1000$ .
- **Subtask 5 [15 punti]**:  $M \leq 2000$ .

## Esempi di input/output

stdin	stdout
3 3 2 1 4 3 2 7 1 3 9	9
4 5 2 1 6 3 1 9 3 2 3 4 2 8 1 4 5	11

## Spiegazione

Nel **primo caso di esempio**, la latenza massima si ha tra gli smartphone 1 e 3: 9 millisecondi.

Nel **secondo caso di esempio**, la latenza massima si ha tra gli smartphone 3 e 4: il miglior percorso per i pacchetti è transitare attraverso lo smartphone 2. Questo porta la latenza totale della trasmissione a  $3 + 8 = 11$  millisecondi.

## Criptovalute (mining)

Limite di tempo: 2.0 secondi  
 Limite di memoria: 256 MiB

Per finanziarsi l'agognata vacanza estiva che avrà luogo in una località segretissima, Luca ha deciso di seguire con attenzione il fiorente mercato del mining di criptovalute. Ha scoperto infatti che nei laboratori della sua università sono presenti parecchie GPU che ben si prestano al mining di criptovalute.

Luca ha già ideato il suo piano nei dettagli: seguendo attentamente l'andamento del valore sul mercato di  $N$  valute, ha calcolato per ciascuna quanti minuti di calcolo occorrono, utilizzando una GPU, per minarne un'unità. Al prezzo di cambio corrente, è certo che se riuscirà a minare un'unità di tutte queste criptovalute, guadagnerà abbastanza per potersi permettere la vacanza. Bisogna solo fare attenzione a rispettare un vincolo un po' particolare, dovuto alla frenesia con cui le nuove criptovalute vengono via via abilitate al mining: Luca le ha già messe in ordine per momento in cui avverrà l'ICO <sup>1</sup>. Prerequisito affinché sia possibile minare una certa valuta è possedere (oppure aver avviato il mining) almeno un'unità di tutte quelle aventi l'ICO in precedenza.

Da adesso alla data prescelta per il viaggio mancano  $M$  minuti. Resta da compiere l'ultima missione cruciale: impossessarsi di alcuni computer in università per poterne utilizzare abusivamente le GPU. Naturalmente, meno GPU utilizzerà, meno è probabile che qualcuno scopra il suo piano e gli rovini l'estate. Aiuta Luca a non farsi scoprire: quante GPU dovrà utilizzare *al minimo* per riuscire a minare un'unità di tutte le criptovalute prima del viaggio?

## Implementazione

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

 Tra gli allegati a questo task troverai un template (`mining.c`, `mining.cpp`) con un esempio di implementazione.

Dovrai implementare la seguente funzione:

### ■ Funzione `gpu`

```
C/C++ | int gpu(int N, int M, int T[]);
```

- L'intero  $N$  rappresenta il numero di criptovalute che devono essere minate.
- L'intero  $M$  rappresenta il numero di minuti che mancano al viaggio.
- L'array  $T$ , indicizzato da 0 a  $N - 1$ , contiene il tempo in minuti necessario per minare un'unità dell' $i$ -esima criptovaluta.
- La funzione dovrà restituire il numero minimo di GPU necessarie per minare tutte le criptovalute prima del viaggio.

Il grader chiamerà la funzione `gpu` e ne stamperà il valore restituito sul file di output.

## Grader di prova

Allegata a questo problema è presente una versione semplificata del grader usato durante la correzione, che potete usare per testare le vostre soluzioni in locale. Il grader di esempio legge i dati da `stdin`, chiama le funzioni che dovete implementare e scrive su `stdout`, secondo il seguente formato.

<sup>1</sup>[https://en.wikipedia.org/wiki/Initial\\_coin\\_offering](https://en.wikipedia.org/wiki/Initial_coin_offering)

Il file di input è composto da 2 righe, contenenti:

- Riga 1: gli interi  $N$  e  $M$ , separati da uno spazio.
- Riga 2:  $N$  interi  $T[i]$  per  $i = 0, \dots, N - 1$ .

Il file di output è composto da un'unica riga, contenente il valore restituito dalla funzione `gpu`.

## Assunzioni

- $1 \leq N \leq 100\,000$ .
- $1 \leq M \leq 1\,000\,000$ .
- $1 \leq T[i] \leq M$  per ogni  $i = 0, \dots, N - 1$ .
- Quando una GPU completa il mining di una certa criptovaluta, può subito iniziare a minarne un'altra al minuto successivo.
- Prese le criptovalute in ordine, è consentito avviare il mining della  $i$ -esima solamente se tutte le precedenti (ovvero, tutte le  $j$ -esime con  $j < i$ ) sono già state minate oppure sono in corso di mining.
- Non è consentito dividere il lavoro di mining di una certa criptovaluta su più GPU.
- Tutte le criptovalute devono essere minate entro il minuto  $M$  (incluso).

## 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 [ 0 punti]**: Casi d'esempio.
- **Subtask 2 [10 punti]**:  $T[0] + \dots + T[N - 1] \leq M$ .
- **Subtask 3 [35 punti]**:  $N \leq 1\,000$ .
- **Subtask 4 [30 punti]**:  $N \leq 5\,000$ .
- **Subtask 5 [25 punti]**: Nessuna limitazione specifica.

## Esempi di input/output

stdin	stdout
3 10 5 2 3	1
4 100 50 70 25 10	2

## Spiegazione

Nel **primo caso di esempio**, possiamo far minare tutte le tre criptovalute ad un'unica GPU, che terminerà proprio al minuto 10, giusto in tempo per la vacanza.

Nel **secondo caso di esempio**, occorrono due GPU. Una possibile distribuzione del lavoro è attribuire il mining della prima e della terza criptovaluta alla prima GPU (che può terminare al più presto al minuto 75), mentre la seconda GPU può minare la seconda e la quarta (finendo non prima del minuto 80).

## Crittografia RSA (rsa)

Limite di tempo: 1.0 secondi

Limite di memoria: 256 MiB

Lemma Teorema

Per rendere ancora più sicure le comunicazioni segrete tra gli smartphone durante l'esame di maturità, gli studenti stanno ponderando l'eventualità di *cifrare* tutti i messaggi che si scambieranno. Nel corso dell'ultimo anno scolastico hanno infatti studiato l'algoritmo di cifratura asimmetrica RSA. L'idea è semplice: realizzare un'app che cifri i messaggi e un'altra che li decifri; in questo modo anche qualora i commissari dovessero intercettarli non riuscirebbero a leggerne il contenuto!

La maturità si avvicina e i lavori per l'app di decifratura sono ancora in alto mare. In estrema sintesi<sup>2</sup>, ti viene fornita una coppia di interi  $N$  e  $d$  che costituiscono la "chiave privata". Ogni intero  $c$ , che rappresenta un carattere cifrato, si decifra calcolando

$$c^d \bmod N$$

☞ L'operazione di modulo restituisce il resto della divisione intera tra due numeri. In C/C++, si calcola con l'operatore `%`. Questa operazione gode, inoltre, delle seguenti proprietà (molto utili per evitare *integer overflow* quando si vogliono calcolare numeri molto grandi):

- $(A + B) \bmod M = (A \bmod M + B \bmod M) \bmod M$
- $(A \cdot B) \bmod M = (A \bmod M \cdot B \bmod M) \bmod M$

Aiuta gli studenti a scrivere l'app per la decifratura di un intero messaggio lungo  $L$  caratteri, ciascuno da decifrare singolarmente per produrre il testo in chiaro e mettere al sicuro il buon esito dell'esame!

## Implementazione

Dovrai sottoporre un unico file con estensione `.cpp` o `.c`.

☞ Tra gli allegati a questo task troverai un template (`rsa.cpp` e `rsa.c`) con un esempio di implementazione.

Dovrai implementare la seguente funzione:

### ■ Funzione decifra

```
C/C++ void decifra(int N, int d, int L, int messaggio[], char plaintext[]);
```

- Gli interi  $N$  e  $d$  costituiscono la chiave privata necessaria per decifrare il messaggio.
- L'intero  $L$  rappresenta la lunghezza del messaggio che è stato cifrato e trasmesso.
- L'array `messaggio`, indicizzato da 0 a  $L - 1$ , contiene alla posizione  $i$  l'intero  $c$  che rappresenta l' $i$ -esima lettera cifrata.
- La funzione dovrà riempire l'array `plaintext` in modo che le posizioni da 0 a  $L - 1$  contengano ciascuna il risultato della decifrazione del messaggio contenuto a quella rispettiva posizione nell'array `messaggio`.  
La posizione  $L$  dell'array `plaintext` dovrà contenere il carattere di fine stringa `'\0'`.

<sup>2</sup>Un'introduzione all'algoritmo completo, *non necessaria per risolvere questo problema*, si trova per chi volesse saperne di più presso <https://it.wikipedia.org/wiki/RSA>.



Il grader chiamerà la funzione `decifra` e stamperà l'array `plaintext` sul file di output.

## Grader di prova

Allegata a questo problema è presente una versione semplificata del grader usato durante la correzione, che potete usare per testare le vostre soluzioni in locale. Il grader di esempio legge i dati da `stdin`, chiama la funzione che dovete implementare e scrive su `stdout`, secondo il seguente formato.

Il file di input è composto da 2 righe, contenenti:

- Riga 1: gli interi  $N$ ,  $d$  e  $L$ , separati da uno spazio.
- Riga 2: i numeri interi `messaggio[i]`, per  $i = 0, \dots, L - 1$ .

Il file di output è composto da un'unica riga, contenente:

- Riga 1: il contenuto dell'array `plaintext`, così come modificato dalla funzione `decifra`.

## Assunzioni

- $128 \leq N \leq 2^{31} - 1$  (ovvero  $N$  è rappresentabile con un intero, anche con segno, a 32 bit).
- $1 \leq d < N$ .
- $1 \leq L \leq 100$ .
- $0 \leq \text{messaggio}[i] < N$  per ogni  $i = 0 \dots L - 1$ .
- È garantito che una corretta decifratura del messaggio porta ad avere un plaintext costituito da numeri interi, in base 10, rappresentanti caratteri ASCII validi (in particolare, lettere minuscole).

## 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 [ 0 punti]:** Casi d'esempio.
- **Subtask 2 [25 punti]:**  $L = 1$ ,  $\text{messaggio}[0] \leq 1\,000$ ,  $d = 3$ .
- **Subtask 3 [25 punti]:**  $N \leq 1\,000\,000$ ,  $L = 1$ .
- **Subtask 4 [10 punti]:**  $N \leq 1\,000\,000$ .
- **Subtask 5 [40 punti]:** Nessuna limitazione specifica.

## Esempi di input/output

stdin	stdout
145 3 1 119	r
391 3 3 295 123 129	abc

## Spiegazione

Nel **primo caso di esempio** dobbiamo decifrare un messaggio composto da un unico carattere cifrato: l'intero 119. Calcoliamo quindi  $119^3 = 1\,685\,159$  e prendiamo il resto della divisione per  $N = 145$ . Il risultato è l'intero 114, che nella codifica ASCII rappresenta la lettera 'r'.

Nel **secondo caso di esempio** procediamo in modo analogo, decifrando singolarmente ciascun intero.