



Congettura di Lollatz (lollatz)

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

Da qualche tempo, Giorgio si è interessato alla congettura di *Lollatz*. Questa congettura afferma che, dato un quadrato perfetto N , ripetendo i seguenti due passaggi:

- moltiplichi il numero per la sua (cifra delle unità $- 1$)
- dividi per due, arrotondando per difetto

prima o poi si arriva a un multiplo di 10. Giorgio, tuttavia, non riesce ad interpretare i passaggi della dimostrazione del dottor Lollatz, che sembra essere composta unicamente da abbreviazioni poco leggibili, e quindi si fida poco di questo risultato. Aiutalo a dirimere i suoi dubbi scrivendo un programma che verifichi questa congettura!

Implementazione

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

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

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

C/C++	<code>int afaikdiyrotfl(int N);</code>
Pascal	<code>function afaikdiyrotfl(N: longint): longint;</code>

In cui:

- L'intero N rappresenta il quadrato perfetto di partenza.
- La funzione dovrà restituire -1 se la congettura non è vera per N , altrimenti dovrà restituire il multiplo di 10 che si ottiene alla fine del procedimento. Il valore restituito verrà stampato sul file di output.

Dati di input

Il file `input.txt` è composto da un'unica riga contenente l'unico intero N .

Dati di output

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

Assunzioni

- $1 \leq N \leq 1\,000\,000$.



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]:** $N \leq 100$.
- **Subtask 3 [40 punti]:** $N \leq 1000$.
- **Subtask 4 [30 punti]:** Nessuna limitazione specifica.

Esempi di input/output

input.txt	output.txt
4	30

input.txt	output.txt
100	100

Spiegazione

Nel **primo caso di esempio**, dopo il primo passaggio otteniamo $4 \times (4 - 1)/2 = 6$. Proseguiamo quindi ottenendo $6 \times (6 - 1)/2 = 15$. Procediamo ancora ottenendo $15 \times (5 - 1)/2 = 30$, e qui ci possiamo fermare.

Nel **secondo caso di esempio**, il numero è già un multiplo di 10 quindi l'algoritmo si ferma subito.



Allineamento magnetico (magneti)

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

Gabriele, nonostante la sua età, ama moltissimo giocare con i magneti. Una delle sue costruzioni preferite è il cosiddetto “serpente di magneti”, cioè una lunga linea di magneti. Ovviamente, affinché i magneti rimangano accostati, è necessario che al polo positivo di un magnete corrisponda sempre il polo negativo del successivo, e viceversa. Ad esempio, sono serpenti di magneti validi le successioni di magneti

$(+-)(+-)(+-)(+-)(+-)$ e $(-+)(-+)(-+)$,

mentre non lo sono

$(+-)(-+)(-+)(-+)$ e $(-+)(+-)(-+)$.

Data una generica successione (non necessariamente stabile) di magneti, Gabriele si chiede quale è il minimo numero di magneti da girare affinché la successione risultante sia un serpente di magneti.

Implementazione

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

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

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

C/C++	<code>int gira(int N, char descrizione[]);</code>
Pascal	<code>function gira(N: longint; var descrizione: array of char): longint;</code>

In cui:

- L'intero N rappresenta il numero di caratteri della stringa `descrizione`.
- La stringa `descrizione`, array di caratteri indicizzato da 0 a $N - 1$, contiene la descrizione della situazione iniziale dei magneti, nello stesso formato degli esempi appena fatti (si veda la sezione **Esempi di input/output** per due ulteriori esemplificazioni del formato della stringa).
- La funzione dovrà restituire il minimo numero di magneti che è necessario girare, che verrà stampato sul file di output.

Dati di input

Il file `input.txt` è composto da due righe. La prima riga contiene l'unico intero N . La seconda riga contiene la stringa `descrizione`, composta da N caratteri.

Dati di output

Il file `output.txt` è composto da un'unica riga contenente un unico intero, il minimo numero di magneti che è necessario ruotare per rendere la successione un serpente di magneti.



Assunzioni

- $4 \leq N \leq 100\,000$.
- La stringa **descrizione** è formata solo dai caratteri $(,+,-,)$ ed è una stringa ben formata. Questo implica tra le altre cose che N è **multiplo di 4**.
- Nel caso in cui la successione iniziale di magneti sia già stabile e non ci sia quindi bisogno di girare alcun magnere, rispondere il valore 0.

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]**: $N \leq 20$.
- **Subtask 3 [40 punti]**: $N \leq 1000$.
- **Subtask 4 [30 punti]**: Nessuna limitazione specifica.

Esempi di input/output

input.txt	output.txt
8 (+-)(-+)	1
input.txt	output.txt
12 (+-)(+-)(-+)	1

Spiegazione

Nel **primo caso di esempio** è sufficiente ruotare il primo magnete.

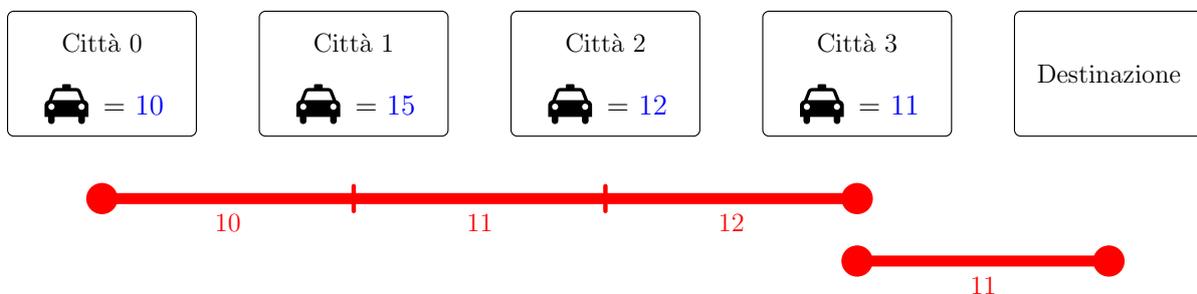
Nel **secondo caso di esempio** è sufficiente ruotare l'ultimo magnete.

Viaggio in taxi (taxi)

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

Gabriele deve andare a trovare Giorgio per mettere a punto la finale delle *OIS*. La strada da Brescia a Pinerolo è lunga, e attraversa $N + 1$ città (numerata da 0 a N) tutte alla stessa distanza di 1 tantometro. Dato che Gabriele ha recentemente racimolato un po' di liquidità, sta valutando se viaggiare comodo spostandosi di città in città in taxi. In ogni città c'è una stazione di taxi, ma la situazione è resa complicata dal fatto che i prezzi dei taxi variano notevolmente di città in città. Prendere un taxi nella città i ha un prezzo base di C_i euro per il primo tantometro percorso, e poi questo prezzo aumenta di uno per ogni ulteriore tantometro percorso (i tassisti sono reticenti ad allontanarsi dalla loro città natia!).

Per esempio, nella seguente situazione:



Gabriele inizia prendendo il taxi a Brescia, la città numero 0, con un prezzo base di 10 euro al tantometro. Quindi lo utilizza per andare fino alla città numero 3, pagando $10 + 11 + 12 = 33$ euro. A questo punto, preferisce cambiare taxi e per arrivare a Pinerolo spende ancora 11 euro, per un totale di 44.

Aiuta Gabriele a valutare se può permettersi di viaggiare in taxi, calcolando quanto costerebbe al minimo un tragitto da Brescia (città numero 0) a Pinerolo (città numero N) in taxi!

Implementazione

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

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

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

C/C++	<code>int viaggia(int N, int C[]);</code>
Pascal	<code>function viaggia(N: longint; var C: array of longint): longint;</code>

In cui:

- L'intero N rappresenta il numero di città tra Brescia e Pinerolo (incluse).
- L'array C , indicizzato da 0 a $N - 1$, contiene i prezzi base dei taxi nelle varie città. Non è riportato il prezzo dei taxi a Pinerolo, che non è rilevante per la soluzione di questo problema.
- La funzione dovrà restituire il costo minimo in euro per un tragitto in taxi da 0 a N , che verrà stampato sul file di output.



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 C_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

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

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]:** $N \leq 100$.
- **Subtask 4 [30 punti]:** Nessuna limitazione specifica.

Esempi di input/output

input.txt	output.txt
4 10 15 12 11	44
input.txt	output.txt
12 27 21 99 35 71 23 64 5 10 44 1 1	184

Spiegazione

Nel **primo caso di esempio**, Gabriele prende i taxi nelle città 0 e 3.

Nel **secondo caso di esempio**, Gabriele prende i taxi nelle città 0, 1, 5, 7, 10, 11.



Espressione di parentesi (parentesi)

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

Giorgio ha scritto un programma contenente una serie di espressioni molto elaborate, formate ciascuna da un gran numero di parentesi di tutti i tipi, e cioè:

- angolate: '<' e '>'
- tonde: '(' e ')'
- quadrate: '[' e ']'
- graffe: '{' e '}'

Purtroppo, quando ha provato ad eseguirlo, il compilatore gli ha detto che c'è un errore senza nemmeno dirgli in quale espressione si trova! Aiutalo controllando quali espressioni sono ben formate e quali no.

Implementazione

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

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

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

C/C++	<code>int controlla(int N, char E[]);</code>
Pascal	<code>function controlla(N: longint; var E: array of char): longint;</code>

In cui:

- L'intero N rappresenta la lunghezza dell'espressione da controllare.
- L'array E , indicizzato da 0 a $N - 1$, contiene i caratteri di cui l'espressione è composta.
- La funzione dovrà restituire 0 se l'espressione è corretta, -1 altrimenti. Nel primo caso, sul file di output verrà stampata la stringa `'corretta'`, nel secondo caso la stringa `'malformata'`.

Dati di input

Il file `input.txt` è composto da due righe. La prima riga contiene l'unico intero N . La seconda riga contiene la stringa E .

Dati di output

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



Assunzioni

- $1 \leq N \leq 10\,000$.
- E_i è uno tra i caratteri ' $\{[(\langle\rangle)]\}$ '.

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]:** Tutte le parentesi sono tonde.
- **Subtask 3 [20 punti]:** Le espressioni corrette sono formate da parentesi ordinate gerarchicamente (vedi quarto caso di esempio).
- **Subtask 4 [30 punti]:** $N \leq 30$.
- **Subtask 5 [20 punti]:** Nessuna limitazione specifica.

Esempi di input/output

input.txt	output.txt
4 ([])	malformata
input.txt	output.txt
5 <({})	malformata
input.txt	output.txt
12 ([] {(<>)})	corretta
input.txt	output.txt
20 {(<><>)}{({) [<>] <>}}	corretta

Spiegazione

Nel **primo caso di esempio**, la parentesi tonda più esterna viene chiusa prima della parentesi quadra (che è più interna), e quindi l'espressione è malformata.

Nel **secondo caso di esempio**, la parentesi angolata all'inizio non viene mai chiusa.

Nel **terzo e quarto caso di esempio**, tutte le parentesi sono chiuse correttamente. Inoltre nell'ultimo viene rispettata la gerarchia tra le parentesi: le angolate sono tutte più all'interno, a seguire tonde, quadre e infine graffe più all'esterno. In altre parole, una parentesi graffa non può essere aperta all'interno di un'altro tipo di parentesi; una parentesi quadra non può essere aperta all'interno di una parentesi tonda o angolata, e una parentesi tonda non può essere aperta all'interno di una parentesi angolata.

Gioco del tris (tris)

Limite di tempo: 1.0 secondi

Limite di memoria: 256 MiB

Quasi tutti conoscono il gioco del tris, ma per completezza riportiamo qui sotto le regole:

- Il gioco si svolge su una griglia 3×3 , inizialmente vuota.
- A turno, i giocatori scelgono una cella vuota e vi disegnano il proprio simbolo (una X o una O).
- Vince il giocatore che riesce a disporre tre dei propri simboli in linea retta orizzontale, verticale o diagonale. Se la griglia viene riempita senza che nessuno dei giocatori sia riuscito a completare una linea retta di tre simboli, il gioco finisce in parità e nessuno dei giocatori vince.

Gabriele è stanco di perdere in continuazione contro Giorgio, abilissimo giocatore di tris, perciò decide di scrivere un programma in grado di predire, conoscendo lo stato corrente della griglia, se Gabriele riuscirà a vincere indipendentemente dalle mosse di Giorgio. Gabriele gioca sempre per primo col simbolo X, mentre giorgio gioca sempre per secondo con il simbolo O.

A titolo di esempio, consideriamo la griglia

	O	
	X	X

Contando il numero di simboli, si deduce che è il turno di Giorgio. Se Giorgio non blocca il tris sulla seconda riga, Gabriele avrà vittoria facile; supponiamo perciò che Giorgio blocchi il tris:

	O	
O	X	X

A questo punto Gabriele posiziona il suo simbolo nell'angolo in alto a destra.

	O	X
O	X	X

Giorgio si trova ora spiazzato, in quanto dovrebbe coprire due potenziali tris, e perde la partita. Nell'esempio appena mostrato, è evidente che Giorgio perde a prescindere dalle sue mosse: se non avesse coperto il tris sulla seconda riga avrebbe perso, ma anche coprendolo viene sconfitto in poche mosse.



Implementazione

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

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

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

C/C++	<code>int vincente(char griglia[3][3]);</code>
Pascal	<code>function vincente(var griglia: array[0..2, 0..2] of char): longint;</code>

In cui:

- L'array `griglia` è una matrice 3×3 , in cui le righe e le colonne sono indicizzate a partire da 0, che rappresenta lo stato della griglia. Ogni cella della matrice contiene un punto se la corrispondente cella della griglia è vuota, una `X` se è occupata da un simbolo di Gabriele e una `O` se è occupata da un simbolo di Giorgio.
- La funzione dovrà restituire 1 se Gabriele può vincere indipendentemente dalle mosse di Giorgio, e 0 altrimenti. Il valore ritornato verrà stampato sul file di output.

Dati di input

Il file `input.txt` è composto da tre righe, ognuna formata da 3 caratteri, rappresentanti la situazione della griglia.

Dati di output

Il file `output.txt` è composto da un'unica riga contenente il valore 1 se Gabriele può vincere indipendentemente dalle mosse di Giorgio, e 0 altrimenti.

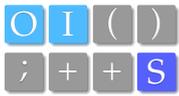
Assunzioni

- Esiste almeno una cella vuota.
- La griglia rappresenta una partita ancora in corso: non sono presenti tris.
- Il pareggio non è considerata una vittoria di Gabriele.

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 [30 punti]:** Esiste esattamente una cella vuota nella griglia.
- **Subtask 3 [30 punti]:** Il numero di celle vuote è ≤ 4 .
- **Subtask 4 [30 punti]:** nessuna limitazione specifica.



Esempi di input/output

input.txt	output.txt
.0. .XX ...	1
input.txt	output.txt
..0 .XX ...	0
input.txt	output.txt
..0 .XX X00	1

Spiegazione

Il **primo caso di esempio** corrisponde all'esempio del testo.

Nel **secondo caso di esempio** se Giorgio gioca bene può pareggiare.

Nel **terzo caso di esempio** è sufficiente che Gabriele completi il tris sulla seconda riga.



Inviti alla festa (festa)

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

Gabriele e Giorgio hanno deciso di organizzare una festa. Per questo hanno raccolto una lista di tutti i loro amici, e ora vogliono invitarli. Affinché gli amici di Gabriele e Giorgio non si annoino alla festa, è necessario che ogni invitato conosca almeno altri due degli amici invitati.

Conoscendo il grafo delle conoscenze reciproche tra gli amici di Giorgio e Gabriele, stabilisci qual è il massimo numero di persone che Gabriele e Giorgio possono invitare alla festa.

Implementazione

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

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

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

C/C++	<code>int invita(int N, int M, int conoscenzeA[], int conoscenzeB[]);</code>
Pascal	<code>function invita(N, M: longint; var conoscenzeA, conoscenzeB: array of longint): longint;</code>

In cui:

- L'intero N rappresenta il numero di amici di Gabriele e Giorgio.
- L'intero M rappresenta il numero di archi nel grafo delle conoscenze (coppie di amici che si conoscono).
- Gli array `conoscenzeA` e `conoscenzeB`, indicizzati da 0 a $M - 1$, contengono le informazioni sugli archi del grafo: per ogni $0 \leq i < M$, gli amici `conoscenzeA[i]` e `conoscenzeB[i]` si conoscono. Gli amici sono indicizzati con valori $0, 1, \dots, N - 1$.
- La funzione dovrà restituire il massimo numero di persone che è possibile invitare rispettando la condizione che ogni invitato conosca almeno altri due invitati. Tale valore verrà stampato sul file di output.

Dati di input

Il file `input.txt` è composto da $M + 1$ righe. La prima riga contiene gli interi N e M separati da uno spazio. Le righe successive contengono la descrizione degli archi del grafo delle conoscenze: sulla i -esima di queste righe sono presenti gli interi `conoscenzeA[i]` e `conoscenzeB[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

- $1 \leq N \leq 10\,000$.
- $0 \leq M \leq 100\,000$.
- Il grafo delle conoscenze è un grafo non diretto (cioè la conoscenza è sempre reciproca). Tutti gli archi sono validi ($0 \leq \text{conoscenzeA}[i], \text{conoscenzeB}[i] \leq N-1$ e $\text{conoscenzeA}[i] \neq \text{conoscenzeB}[i]$), e non vengono ripetuti.
- Gabriele e Giorgio non vanno contati nel numero di amici conosciuti dagli invitati.
- Nel caso in cui non sia possibile invitare alcun amico, stampare il valore 0.

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

input.txt	output.txt
3 2 0 2 1 2	0

input.txt	output.txt
9 10 0 1 2 0 5 2 4 5 3 4 2 3 7 2 6 5 0 5 1 8	5



Spiegazione

Nel **primo caso di esempio** il massimo numero di invitati si raggiunge invitando alla festa gli amici 3, 4 e 5.

Nel **secondo caso di esempio** non è possibile invitare alcun amico alla festa.

Nel **terzo caso di esempio** il massimo numero di invitati si raggiunge invitando alla festa gli amici 3, 4, 2, 0, 5.



Piano degli studi (pianostudi)

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

Gabriele vuole laurearsi il prima possibile! Ogni anno deve compilare il cosiddetto “piano degli studi”, dove indica i corsi che intende frequentare per quell’anno accademico. Ogni corso, oltre alla data di inizio e fine lezioni, ha associato anche un numero (intero) di crediti. Per potersi laureare, Gabriele sa che deve accumulare un certo numero di crediti, ottenuti superando gli esami dei corsi che sceglie di frequentare.

Per non essere troppo stressato, Gabriele stabilisce che tra i corsi che sceglierà non dovranno esserci sovrapposizioni, ovvero non dovrà succedere che i periodi di erogazione dei corsi abbiano intersezione, neanche eventualmente per un solo giorno. Data la lista dei corsi, aiuta Gabriele a scegliere il sottoinsieme di corsi che garantisce il massimo numero di crediti.

Implementazione

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

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

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

C/C++	<code>int pianifica(int N, int da[], int a[], int crediti[]);</code>
Pascal	<code>function pianifica(N: longint; var da, a, crediti: array of longint): longint;</code>

In cui:

- L’intero N rappresenta il numero totale di corsi erogati dall’università di Gabriele.
- Gli array `da` e `a`, indicizzati da 0 a $N - 1$, contengono le date di inizio e di fine dei corsi, rappresentate con dei numeri interi. Il periodo di erogazione del corso i è quindi quello che va dal giorno `da[i]` al giorno `a[i]`, estremi inclusi.
- L’array `credit`, indicizzato da 0 a $N - 1$, contiene alla posizione i il numero di crediti ottenuti dalla frequentazione del corso i .
- La funzione dovrà restituire il massimo numero di crediti che Gabriele può ottenere in totale, sotto la condizione che tra i corsi scelti non vi sia sovrapposizione. Il valore ritornato verrà stampato sul file di output.

Dati di input

Il file `input.txt` è composto da $N + 1$ righe. La prima riga contiene l’unico intero N . Le successive N righe contengono le informazioni sui corsi: la i -esima di queste contiene i tre interi `da[i]`, `a[i]`, `credit[i]`, separati da uno spazio.

Dati di output

Il file `output.txt` è composto da un’unica riga contenente un unico intero, il massimo numero di crediti a cui Gabriele può puntare.



Assunzioni

- $1 \leq N \leq 100\,000$.
- $1 \leq da[i], a[i] \leq 100\,000\,000$ per ogni $i = 0 \dots N - 1$.
- $1 \leq crediti[i] \leq 10\,000$ per ogni $i = 0 \dots N - 1$.

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]:** $N \leq 10$.
- **Subtask 3 [20 punti]:** $N \leq 1000$ e tutti i corsi valgono lo stesso numero di crediti.
- **Subtask 4 [30 punti]:** $N \leq 1000$.
- **Subtask 5 [30 punti]:** Nessuna limitazione specifica.

Esempi di input/output

input.txt	output.txt
2 5 8 1 1 5 2	2
input.txt	output.txt
3 3 9 30 2 4 10 5 6 15	30

Spiegazione

Nel **primo caso di esempio** i due corsi si sovrappongono nel giorno 5, quindi Gabriele può frequentarne solo uno, e sceglie il secondo corso, per un valore totale di 2 crediti.

Nel **secondo caso di esempio** la scelta ottima è frequentare solo il corso da 30 crediti.