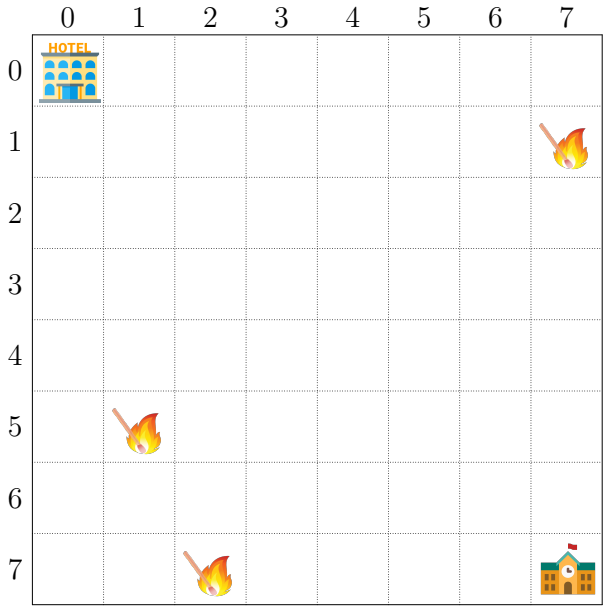
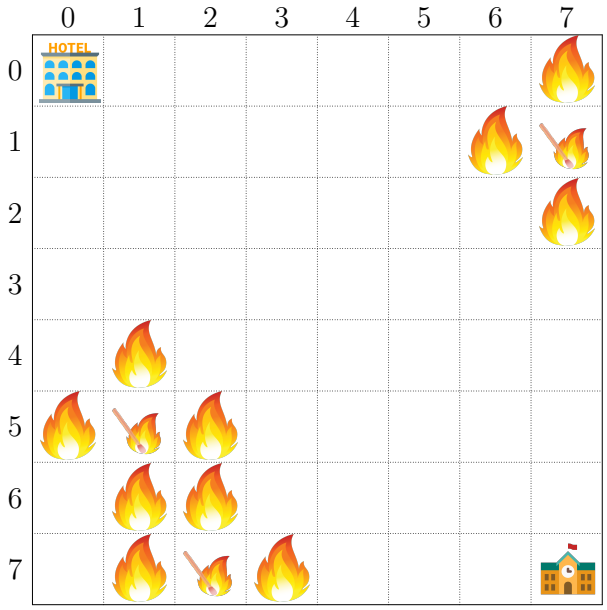
Attenti all'incendio (`incendio`)

Il Comitato delle Olimpiadi di Informatica teme che i problemi della finale nazionale di Campobasso siano troppo facili. Per questo motivo, la sede di gara e l'hotel sono stati posizionati in una griglia $N \times N$ in cui ogni cella è altamente infiammabile: l'hotel alle coordinate $(0, 0)$, la sede di gara alle coordinate $(N-1, N-1)$. Quest'anno i partecipanti, oltre a risolvere i task, dovranno anche evitare gli incendi che incontreranno nel percorso che va dall'hotel alla sede di gara!

Il comitato ha ottenuto dal comune il permesso per appiccare *contemporaneamente* M incendi, numerati da 0 a $M-1$. L'i-esimo incendio viene appiccato nella cella di coordinate $(X[i], Y[i])$. Per ogni minuto trascorso a partire da quel momento, le celle infuocate scateneranno altri incendi nelle rispettive 4 celle adiacenti (nord, sud, est, ovest) rendendole anch'esse infuocate. Gli incendi si diffondono nella griglia fin quando questa non è completamente incendiata, ma sono controllati in modo da non poter uscire dalla griglia (per motivi di sicurezza cittadina).



Situazione iniziale



Situazione dopo un minuto

Ai partecipanti, naturalmente, conviene partire il prima possibile per avere più possibilità di raggiungere la sede di gara illesi. Chupito invece, come la maggior parte dei gatti, è patologicamente pigro e non ha intenzione di svegliarsi presto se non assolutamente necessario.

Chupito può spostarsi solo verso una delle 4 celle adiacenti alla sua posizione attuale, senza mai uscire dalla griglia e senza mai attraversare celle infuocate. Per dormire il più possibile però, Chupito vuole partire il più tardi possibile.

Aiuta Chupito calcolando **dopo quanti minuti al massimo** dall'appiccamento degli incendi ci sarà ancora un qualche percorso con il quale raggiungere la sede di gara!

Implementazione

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

> ☞ Tra gli allegati a questo task troverai un template `incendio.c` e `incendio.cpp` con un esempio di implementazione.

Dovrai implementare la seguente funzione:

C/C++	`int alzati(int N, int M, int X[], int Y[]);`

- L'intero N rappresenta la dimensione della griglia della città.
- L'intero M rappresenta il numero di incendi presenti all'inizio.
- Gli array `X` e `Y` (indicizzati da 0 a $M-1$) contengono le coordinate iniziali degli M incendi, per cui l'i-esimo incendio parte dalle coordinate $(X[i], Y[i])$.
- La funzione deve restituire un intero corrispondente al massimo numero di minuti entro il quale è possibile raggiungere la sede di gara in sicurezza.

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

Grader di prova

Nella directory relativa 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.

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

- Riga 1: gli interi N e M separati da spazio.
- Successive M righe: i valori `X[`i`]` e `Y[`i`]`, separati da spazio, per ogni $i = 0 \dots M-1$.

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

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

Assunzioni

- $2 \leq N \leq 1\,000\,000\,000$.
- $1 \leq M \leq 12\,000$.
- $0 \leq X[i], Y[i] \leq N-1$ per ogni $i = 0, \dots, M-1$.
- Tutti gli incendi iniziano da coordinate distinte.
- È inizialmente possibile raggiungere la sede di gara.
- Non sono inizialmente presenti incendi nelle coordinate $(0,0)$ e $(N-1, N-1)$.

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 che lo compongono.

- **Subtask 1 [0 punti]**: Casi d'esempio.
- **Subtask 2 [7 punti]**: $M = 1$.
- **Subtask 3 [10 punti]**: $X[i] = X[j]$ per ogni $0 \leq i, j < M$.
- **Subtask 4 [20 punti]**: $N \leq 500$.

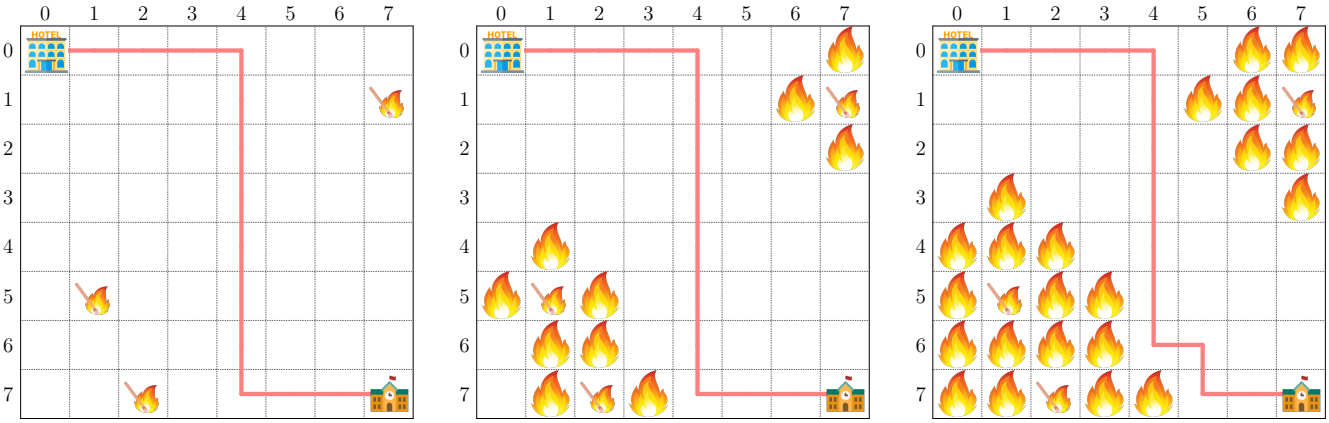
- **Subtask 5 [22 punti]**: $N \leq 1000$.
- **Subtask 6 [24 punti]**: $M \leq 2000$ e $X[i] + Y[i]$ è pari per ogni $0 \leq i < M$.
- **Subtask 7 [9 punti]**: $M \leq 10\,000$.
- **Subtask 8 [8 punti]**: Nessuna limitazione specifica.

Esempi di input/output

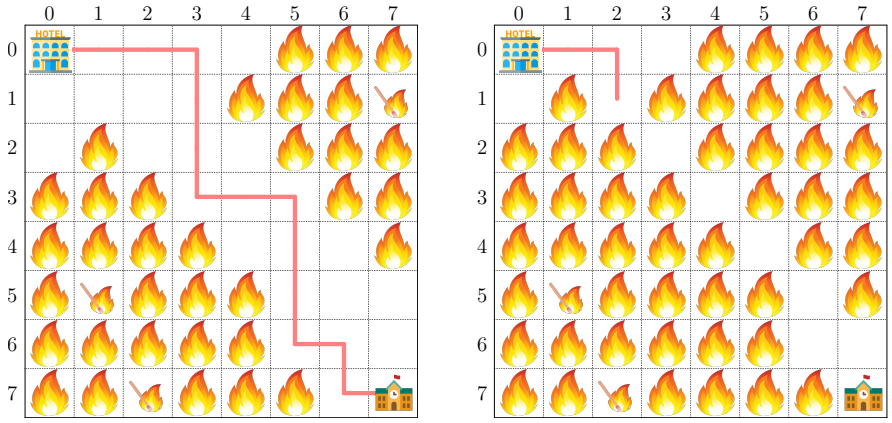
stdin	stdout
8 3 1 7 5 1 7 2	3
7 3 3 4 0 2 6 3	1

Spiegazione

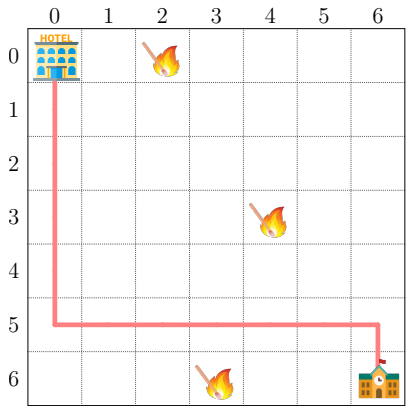
Nel **primo caso di esempio** abbiamo una griglia 8×8 nella quale sono presenti inizialmente 3 incendi, l'evoluzione della griglia è la seguente:

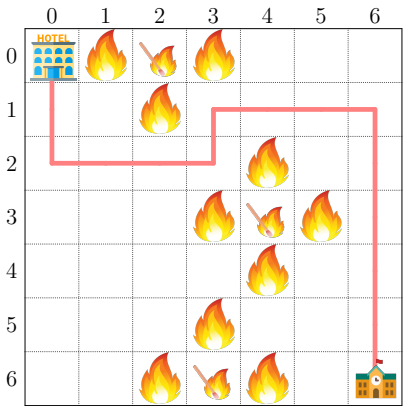


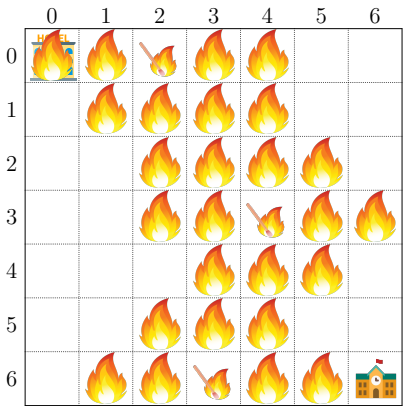
A partire da $t = 4$ non sarà più possibile raggiungere in sicurezza la sede di gara, $t = 3$ è quindi l'ultimo minuto utile che Chupito ha per alzarsi.



Nel **secondo caso di esempio** già da dopo $t = 1$ non sarà più possibile raggiungere in sicurezza la sede di gara:







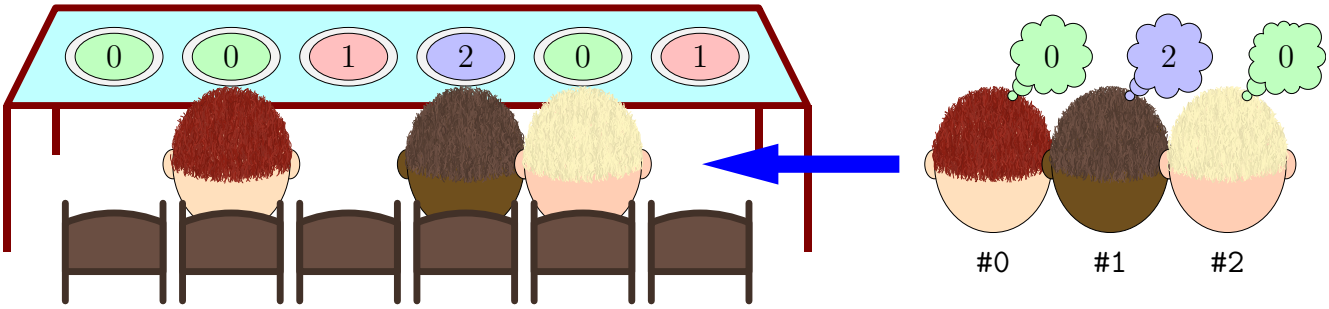
Cena di gala (cena)

A gara terminata, i P partecipanti delle Olimpiadi Italiane di Informatica (numerati da 0 a $P-1$) parteciperanno ad una cena di gala in piena tradizione molisana.

Il banchetto consiste di S piatti numerati da 0 a $S-1$, ciascuno contenente una tra le "migliori 100 pietanze molisane" scelte personalmente dallo chef: la *composta molisana*, lo *spezzatino di pecora*, l'*agnello con cicoria* e altre 97 succulente specialità.

Al momento dell'arrivo alla cena di gala i partecipanti si recheranno uno alla volta verso l'inizio del banchetto, da destra verso sinistra (dal piatto $S-1$ verso il piatto 0) e potranno scegliere se sedersi o se passare al piatto successivo. Ogni partecipante ha una pietanza preferita, per cui si siederà al banchetto solo in uno dei posti dove è apparecchiato un piatto contenente tale pietanza. I partecipanti arrivano al banchetto in ordine dal primo all'ultimo classificato.

Non è permesso scavalcarsi, quindi quando un partecipante decide di sedersi di fronte al piatto i-esimo, tutti i piatti in posizione $j < i$ saranno inaccessibili ai partecipanti seguenti. Una volta seduti, quindi, i partecipanti avranno lo stesso ordine in cui sono arrivati. È possibile che ci siano dei "buchi" dove nessuno è seduto. I partecipanti non sono obbligati a scegliere il primo piatto "preferito" che incontrano: tra i piatti contenenti la loro pietanza preferita, infatti, *possono sceglierne uno qualsiasi* come in figura.



Purtroppo però è sorto un problema: Mojito e Chupito, rispettivamente il cane ed il gatto di Monica, hanno eluso la sicurezza e si sono intrufolati nella sala del banchetto prima dell'arrivo dei partecipanti. I due monelli sono saliti sulla lunga tavolata con l'intenzione di papparsi alcuni dei piatti, in particolare:

- Mojito è salito dal lato *sinistro* del tavolo, per papparsi i piatti partendo dal numero 0.
- Chupito è salito dal lato *destro* del tavolo, per papparsi i piatti partendo dal numero $S-1$.

Monica, allertata dall'assenza dei suoi due pelosetti, è corsa verso il banchetto e deve ora decidere come e quando fermare i due affamati. È possibile fermare Mojito e Chupito in qualunque punto, anche prima che i due comincino a mangiare. Tuttavia, intenerita dai suoi cuccioli, Monica vorrebbe lasciarli lì a mangiare almeno un po' se possibile...

Indichiamo con A e B rispettivamente il numero di piatti mangiati da Mojito e da Chupito. Aiuta Monica calcolando **quante coppie diverse** (A, B) esistono tali che, anche "sacrificando" i primi A piatti e gli ultimi B piatti della lunga tavolata, **c'è almeno una strategia** che permette ai partecipanti di sedersi tutti davanti ad un piatto contenente la propria preferenza.

✍ **Nota bene:** i partecipanti delle OII sono curiosi di "sperimentare" le pietanze molisane. Infatti, indipendentemente dal valore di P, il numero di partecipanti *con una stessa pietanza preferita* non sarà mai superiore a **1000**.

Implementazione

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

> ☞ Tra gli allegati a questo task troverai un template `cena.c` e `cena.cpp` con un esempio di implementazione.

Dovrai implementare la seguente funzione:

C/C++	`long long conta(int S, int s[], int P, int p[]);`

- L'intero S indica il numero di piatti nella tavola.
- L'array s (indicizzato da 0 a $S-1$) indica il tipo di cibo nell'i-esimo piatto.
- L'intero P indica il numero di partecipanti.
- L'array p (indicizzato da 0 a $P-1$) indica la preferenza dell'i-esimo partecipante.

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

Grader di prova

Nella directory relativa 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.

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

- Riga 1: gli interi S e P.
- Riga 2: i valori `s[`i`]` per $i = 0 \dots S-1$.
- Riga 3: i valori `p[`i`]` per $i = 0 \dots P-1$.

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

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

Assunzioni

- $1 \leq P \leq S \leq 100\,000$.
- $1 \leq P \leq 50\,000$.
- $0 \leq$ `s[`i`]`, `p[`i`]` < 100.
- Non esistono 1001 partecipanti con la stessa preferenza.

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 che lo compongono.

- **Subtask 1 [0 punti]**: Casi d'esempio.
- **Subtask 2 [13 punti]**: $S \leq 100$ e $P \leq 100$.
- **Subtask 3 [23 punti]**: $P \leq 1000$ e tutti i partecipanti hanno la stessa preferenza.
- **Subtask 4 [27 punti]**: $S \leq 10\,000$ e $P \leq 100$.
- **Subtask 5 [16 punti]**: $P \leq 100$.
- **Subtask 6 [21 punti]**: Nessuna limitazione specifica.

Esempi di input/output

stdin	stdout
6 3 0 0 1 2 0 1 0 2 0	4
9 3 0 1 0 1 1 0 1 0 1 0 0 0	8

Spiegazione

Il **primo caso di esempio** coincide con quello illustrato nel testo. Per garantire ai partecipanti di potersi sedere, ci sono 4 modi di fermare Mojito e Chupito:

- `0[01̲20̲]1` – Monica lascia mangiare 1 piatto a Mojito e 1 piatto a Chupito.
- `[00̲1̲201]` – Mojito 0 piatti, Chupito 0 piatti.
- `[00̲120̲]1` – Mojito 0 piatti, Chupito 1 piatto.
- `0[01̲201]` – Mojito 1 piatto, Chupito 0 piatti.

La zona "risparmiata" dai due affamati è indicata tra parentesi quadre. Inoltre, per dimostrare che tutti questi modi garantiscono ai partecipanti di trovare la propria pietanza preferita, è stata sottolineata una possibile disposizione che questi potrebbero scegliere.

Nel **secondo caso di esempio**, dato che i partecipanti hanno tutti la stessa preferenza, sono validi tutti i modi di fermare Mojito e Chupito che risparmiano almeno 3 piatti di tipo `'0'`:

- `[01̲0̲11̲0̲]101`
- `[01̲0̲11̲01̲]01`
- `[01̲011̲01̲0̲]1`
- `[01̲0̲11̲0̲1̲0̲1̲]`
- `0[1̲0̲11̲01̲0̲]1`
- `0[1̲0̲110̲1̲01̲]`
- `01[01̲1̲01̲0̲]1`
- `01[01̲1̲0̲1̲01̲]`

Circuiti bruciati (`circuiti`)

Il server delle Olimpiadi Italiane di Informatica si è guastato proprio il giorno della finale nazionale! Ce la faranno i sistemisti molisani a riportarlo online prima dell'inizio della gara?

Dopo un'attenta analisi, è venuto fuori che ci sono dei circuiti bruciati da sostituire. Fortunatamente, nel laboratorio del Marconi ci sono un saldatore ed abbastanza componenti per ricostruire da zero i circuiti mancanti. Dario, il tutor smanettone, si è offerto di eseguire la riparazione. Quello che gli manca però è il progetto (schema elettrico) dei circuiti. Aiuta Dario a progettare lo schema dei circuiti mancanti!

Un circuito legge un array di N interi tramite i *registri di input* `in`$[i]$ ($0 \leq i < N$), e può scrivere il risultato sui *registri di output* `out`$[i]$. Ci sono tre tipi di circuiti da sostituire, che devono realizzare le seguenti operazioni.

Somma	Scrivere su `out[0]` la somma degli N interi in input.
Prefissi	Scrivere sui registri di output l'array delle somme prefisse, definito come `out`$[i]$ = `in[0]` $+ \cdots +$ `in`$[i]$ con $0 \leq i < N$.
Massimo sottoarray	Scrivere su `out[0]` il più grande numero ottenibile come somma di elementi consecutivi dell'array di input (ovvero, che si possa scrivere come `in`$[i]$ $+ \cdots +$ `in`$[j]$ con $0 \leq i \leq j < N$). Anche il vettore vuoto vale come sottoarray, per cui il risultato di questa operazione è sempre almeno 0.

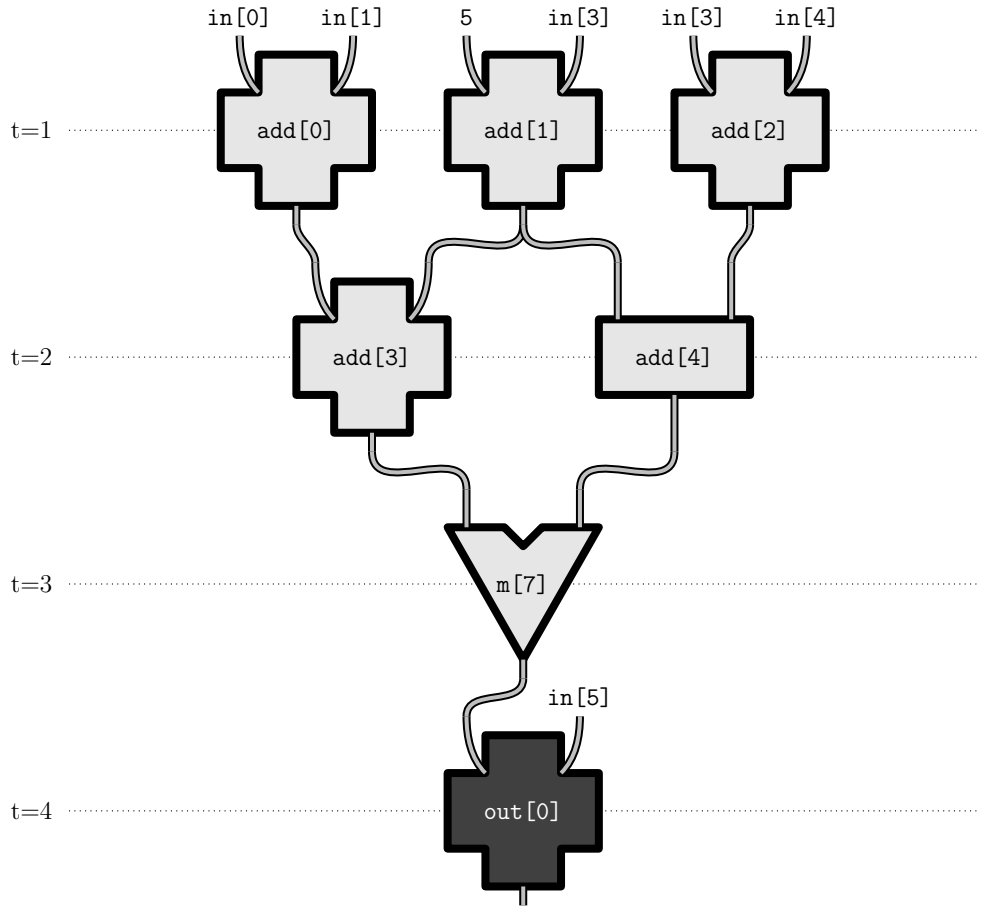
Oltre ai registri di input e output, un circuito può contenere *registri intermedi*. Inoltre esistono dei registri speciali che forniscono delle *costanti* che possono essere usate come ingresso per gli altri componenti.

Un circuito è formato da diversi *componenti*, come esemplificato in Figura 1. Ogni componente è collegata a due registri di ingresso e un registro di uscita, e implementa una fra le seguenti operazioni matematiche: somma (`+`), differenza (`-`) e massimo (`max`). Appena vengono scritti i valori di entrambi i registri di ingresso, il componente effettua il calcolo e, dopo un ciclo di clock, scrive il risultato dell'operazione scelta sul suo registro di uscita. Il registro di uscita può a sua volta essere collegato (come registro di ingresso) ad uno o più componenti, che eseguiranno il calcolo appena avranno disponibili entrambi i loro ingressi e così via.

I registri di input possono essere usati solo come ingressi, mentre i registri intermedi e i registri di output sono uscite di **esattamente una** componente. Il calcolo termina quando tutti i registri sono stati scritti.

Per ciascun circuito che deve essere progettato, conosci il tipo di circuito (somma, prefissi o massimo sottoarray), il numero N di registri di input, e il tempo di calcolo C desiderato (espresso in cicli di clock).

Aiuta Dario a progettare tutti i circuiti che mancano, in modo tale che funzionino **qualunque siano i dati in ingresso**, e che il loro tempo di calcolo sia il più possibile vicino a quello desiderato. Cerca inoltre di non usare troppi componenti: ogni circuito viene con 1100 componenti in regalo, ma ogni ulteriore componente va comprato!



Esempio di realizzazione della formula:

$$\texttt{out[0]} = \max(\texttt{in[0]} + \texttt{in[1]} + 5 + \texttt{in[3]}, (5 + \texttt{in[3]}) - (\texttt{in[3]} + \texttt{in[4]})) + \texttt{in[5]}$$

I circuiti che dovrai progettare sono i seguenti:

Input	**Tipo**	N	C
`input000.txt`	Somma	7	3
`input001.txt`	Somma	256	8
`input002.txt`	Somma	1093	11
`input003.txt`	Prefissi	7	5
`input004.txt`	Prefissi	128	13
`input005.txt`	Prefissi	371	15
`input006.txt`	Massimo sottoarray	7	15
`input007.txt`	Massimo sottoarray	64	31
`input008.txt`	Massimo sottoarray	110	35
`input009.txt`	Massimo sottoarray	124	35

Formato dell'input

I circuiti da progettare sono riportati nei file di input come nella tabella precedente. Ogni file consiste in tre interi T, N e C dove T indica il tipo di circuito da realizzare (1 per la somma, 2 per i prefissi, 3 per il massimo sottoarray).

Formato dell'output

Il circuito da te progettato va descritto sotto forma di componenti e corrispettive connessioni. Ogni riga del file di output rappresenta un componente, secondo uno dei tre seguenti formati possibili:

$$C[k] = A[i] \texttt{ op } B[j] \qquad C[k] = i \texttt{ op } B[j] \qquad C[k] = A[i] \texttt{ op } j$$

in cui:

- A, B e C sono nomi di array. Questi nomi possono essere composti da massimo 50 caratteri alfanumerici (maiuscoli o minuscoli), e non devono cominciare con una cifra.
- i, j, k sono costanti intere positive, che devono stare in un intero a 32 bit con segno.
- `op` è un operatore a scelta fra `+`, `-` o `max`.

Tale riga definisce un componente che ha $A[i]$ e $B[j]$ come registri di ingresso (o le costanti i e j rispettivamente), $C[k]$ come registro di uscita, e implementa l'operazione `op`.

I due array speciali `in` e `out` indicano rispettivamente i registri di input e output. È possibile usare a piacere altri array come registri intermedi. Tuttavia, prima di definire un componente che usa un certo registro intermedio come ingresso, è necessario che un componente **precedentemente definito** nel file lo abbia come uscita.

Assegnazione del punteggio

I file di output inviati verranno controllati e valutati da un correttore il quale verificherà la correttezza del formato e la correttezza del circuito realizzato, e assegnerà a ciascuno di essi un punteggio da 0 a 10.

Se il file inviato non è corretto riceverà punteggio 0. Altrimenti, gli verrà assegnato un punteggio P secondo la seguente formula:

$$P = 10 \cdot \min\left\{1, \frac{C_{\text{richiesto}}}{C_{\text{ottenuto}}}\right\} \cdot \min\left\{1, \frac{1100}{S_{\text{ottenuto}}}\right\}$$

dove C_{ottenuto} è il tempo di calcolo del tuo circuito, e S_{ottenuto} è il suo numero di componenti. Nota che il tuo file riceverà punteggio pieno 10 se e solo se C_{ottenuto} è pari o inferiore a $C_{\text{richiesto}}$ e S_{ottenuto} è pari o inferiore a 1100.

Esempi di input/output

Lo schema disegnato nel testo corrisponde al seguente file di output:

```
add[0] = in[0] + in[1]
add[1] = 5 + in[3]
add[2] = in[3] + in[4]
add[3] = add[0] + add[1]
add[4] = add[1] - add[2]
m[7] = add[3] max add[4]
out[0] = m[7] + in[5]
```

anche se non è la soluzione di nessun input valido. La visualizzazione di questo output, **o di qualunque altro da te prodotto**, può essere ottenuta cliccando sull'icona di un circuito presente nella barra delle applicazioni alla sinistra del tuo schermo. L'applicazione corrispondente ti chiederà di selezionare un file e ti mostrerà il PDF ad esso corrispondente (contenente il disegno di un circuito o eventuali errori presenti nel file). Il visualizzatore posiziona i componenti corrispondenti allo stesso ciclo di clock t nell'ordine in cui sono riportati nel file, da sinistra verso destra.

Possibili soluzioni di input, diversi da quelli che dovrai progettare, sono presentati di seguito.

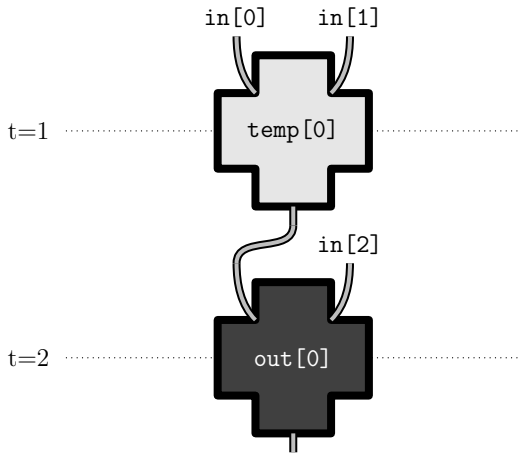
input	output
1 3 2	`temp[0] = in[0] + in[1]` `out[0] = temp[0] + in[2]`

2 3 4	`out[0] = in[0] + 0`
	`out[1] = in[0] + in[1]`
	`out[2] = in[2] + out[1]`

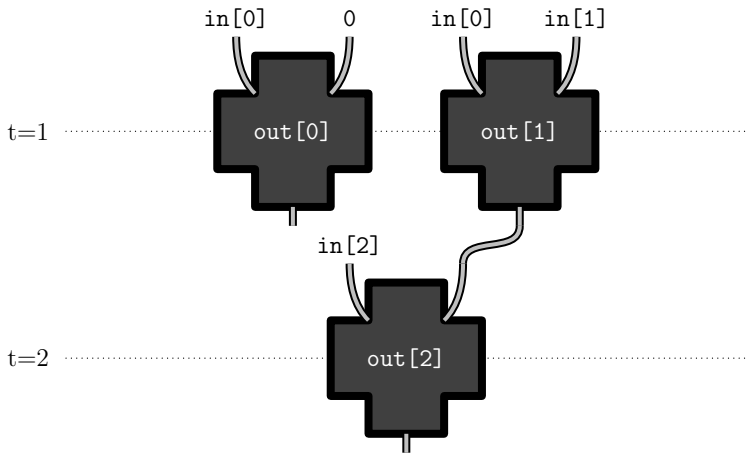
3 3 7	`somma[0] = 0 + 0`
	`somma[12] = in[0] + in[1]`
	`somma[23] = in[1] + in[2]`
	`mx[0] = somma[0] max in[0]`
	`somma[123] = somma[12] + in[2]`
	`mx[1] = mx[0] max in[1]`
	`mx[2] = mx[1] max in[2]`
	`mx[3] = somma[12] max mx[2]`
	`mx[4] = mx[3] max somma[23]`
	`out[0] = mx[4] max somma[123]`

Spiegazione

Nel **primo caso di esempio** si tratta di sommare 3 interi in input, per esempio, se l'input fosse `in[0]` $= -10$, `in[1]` $= 3$, `in[2]` $= 5$ allora il valore di `temp[0]` sarà -7 e `out[0]` varrà -2. Le due operazioni devono essere eseguite una dopo l'altra, impiegando quindi due cicli di clock a produrre il risultato.



Nel **secondo caso di esempio**, dato lo stesso input del caso precedente, l'output varrà: `out[0]` $= -10$, `out[1]` $= -7$, `out[2]` $= -2$. Nota che il terzo componente deve attendere il risultato del secondo, quindi anche in questo caso il circuito impiega 2 cicli di clock.



Nel **terzo caso di esempio**, dato sempre lo stesso input, i valori dei registri saranno:

`somma[0]` $= 0$
`somma[12]` $= -7$
`somma[23]` $= 8$
`somma[123]` $= -2$

`mx[0]` $= 0$
`mx[1]` $= 3$
`mx[2]` $= 5$
`mx[3]` $= 5$

`mx[4]` $= 8$
`out[0]` $= 8$