Il bianco e il nero (`brothers_in_arms` [88 punti])

Nella battaglia di Campaldino (11 giugno 1289) Dante il sommo poeta era disposto nella prima linea dei cavalieri. Vinse la sua parte, quella dei guelfi, con circa 300 caduti, sepolti in fosse comuni in prossimità del Convento di Certomondo. Ben più grave il bilancio dei ghibellini che parteggiavano per il primato dell'imperatore, con circa 1700 morti entro sera e oltre un migliaio finiti prigionieri, dei quali alcune centinaia che non vennero riscattati dai parenti perirono in fretta nelle prigioni fiorentine e furono sepolti a lato della via di Ripoli, a Firenze, in un luogo che ancora oggi si chiama "Canto degli aretini". Quando non molti anni dopo i guelfi si scissero in bianchi e neri dividendosi su fino a che punto il papa dovesse avere l'egemonia, Dante fu meno fortunato nello scegliere la parte da cui stare: non finì falciato nel campo ma l'esilio che ne conseguì lo amareggiò fino all'ultimo dei suoi giorni.

In letto di morte gli apparve una fila di n cavalieri guelfi in battaglia, alcuni neri e altri bianchi, il tutto così sparpagliatamente da ritenere non ve ne fosse criterio nè scopo alcuno. E a guardare bene, più da vicino, nemmeno v'era relazione alcuna tra il colore di un cavaliere e la di lui nobiltà.

Numerati coi numeri $0, \ldots, n-1$ i guelfi della fila come presi da sinistra a destra, la situazione nel sogno di Dante è catturata scrivendo $c_i =$"B" se i è di colore bianco, $c_i =$"N" se i è nero, e $c_i =$"D" se i è defunto, che la morte cancella i colori cui credono gli uomini. Due cavalieri vivi i e j, $i < j$, si considerano limitrofi se ogni cavaliere k compreso tra di loro ($i < k < j$) è già morto. In ogni momento della battaglia, due cavalieri limitrofi di opposto colore potrebbero essere il prossimo conflitto a scoppiare. Questa guerra è un processo randomico che procede per eventi del seguente tipo:

> due cavalieri ancora vivi e di opposto colore che attualmente si vedono limitrofi (perchè di nome i e $j = i + 1$, oppure perchè tutti quelli nell'intervallo aperto (i, j) sono già caduti) si affrontano in aperto duello eliminandodosi vicendevolmente. Come effetto collaterale della loro eliminazione reciproca diventano ora limitrofi il cavaliere $i' < i$ che precedentemente era limitrofo a i da sinistra ed il cavaliere $j' > j$ che era limitrofo a j da destra (sempre che i non fosse il primo o j l'ultimo ad essere rimasti vivi nella fila).

Benchè la guerra non possa terminare finchè sono ancora vivi due cavalieri con (lo stesso identico umore ma) la divisa di altro colore, essa resta comunque un processo destinato a terminare (monovariante: il numero di vivi cala ad ogni evento) e non deterministico in quanto è difficile prevedere chi saranno i prossimi due per cui suona la campana, come anche chi sarà tra i superstiti a guerra finita.

Vi è però certezza su chi ha speranze di salvarsi e chi è condannato già in bella partenza comunque si svolga l'intero processo. Sai rispondere, e con che efficienza, alle seguenti domande?

[**doomed**]: Data la sequenza $c = c_0, \ldots, c_{n-1}$ dei colori, sai dire quali cavalieri sono condannati in bella partenza (1) e quali no (0)?

[**save_Ryan**]: E se ti viene indicato il nome di un milite che può sopravvivere, sai descrivere uno scenario di guerra che lo vede tra i superstiti?

Affidiamoci ora alle suggestioni ricercando un linguaggio evocativo: "Nel caos della pungna siamo comunque governati da forze più grandi di noi, che l'occhio umano tende ad ignorare ma possono scrivere la nostra salvezza o la nostra condanna."

Input

Si legga l'input da `stdin`. La prima riga contiene T, il numero di testcase (istanze) da risolvere. Seguono T istanze del problema, dove ogni istanza è descritta in tre righe: la prima contiene il numero di cavalieri n, la seconda gli n caratteri $c_1, \ldots, c_{n-1}$ in questo ordine e separati da spazi, la terza contiene

un numero intero $y \in [0, n-1)$: esso è l'indice/nome del soldato Ryan. Garantiamo che Ryan possa ancora salvarsi e ti chiediamo di produrre un possibile svolgimento della guerra che a terminazione lo veda tra i superstiti.

Output

Per ciascuna istanza, prima di leggere l'istanza successiva, scrivi su `stdout` il tuo output così strutturato:

[**goal 1**]: la prima riga contiene, separate da spazio, le n cifre binarie x_i, $i = 0, ..., n-1$. Quì $x_i = 1$ se il soldato i è condannato, altrimenti $x_i = 0$.

[**riga riempitiva per facilitare la lettura**]: la seconda riga ripropone identica (a meno dell'entità delle spaziature) la seconda riga in input, quella coi colori degli n cavalieri. Questa riga fà da leggio assistendo nella comprensione sia della riga precedente (goal 1) e che di quella successiva (goal 2). Inoltre tiene allineati gli stream di input e di output (eccetto la prima riga di input col numero di testcase).

[**goal 2**]: la terza riga contiene, separati da spazio, gli n numeri $t_0, ..., t_{n-1}$. Quì t_i riporta l'istante di morte del soldato i. Se già defunto alla partenza (quando $c_i =$"D") allora $t_i = 0$, se si salva allora $t_i = -1$. Le regole da rispettare perchè questa riga sia valida nel certificare la salvabilità di Ryan sono:

1. $t_y = -1$ (il soldato Ryan ne esce vivo)
2. tutti gli $i \in [0, n)$ con $t_i = -1$ hanno lo stesso colore (i superstiti son tali solo a guerra conclusa)
3. $t_i = 0$ se e solo se $c_i =$"D".
4. sia $T := \max(t_i, i \in [0, n))$. Allora per ogni $t \in [1, T]$ vi sono precisamente due diversi cavalieri che muoiono nell'evento i-esimo e per meglio esprimere le ultime regole da rispettare li chiameremo con $a(t), b(t) \in [0, n)$ scegliendo questi nomi in modo che per maggior semplicità si abbia $a(t) < b(t)$. Per intanto abbiamo detto che sono ben definite due funzioni $a, b : [1, T] \to [0, n)$ per le quali $t_{a(t)} = t_{b(t)} = t$ e $a(t) < b(t)$ per ogni $t \in [1, T]$
5. $\left\{c_{a(t)}, c_{b(t)}\right\} = \{$"B", "N"$\}$, ossia $a(t)$ e $b(t)$ sono due cavalieri di opposto colore
6. $0 \leq t_i < t$ per ogni soldato i con $i \in (a(t), b(t))$ (ossia ogni soldato che nella linea sia infrapposto tra $a(t)$ e $b(t)$ è già deceduto prima dell'istante t, quando $a(t)$ e $b(t)$ si eliminano a vicenda)

Oltre alle dimensioni delle istanze, ogni subtask precisa quanti punti competono a ciascuno dei due goal. Per ogni istanza di quel subtask si otterranno tutti i punti dei goal correttamente evasi (purchè si rispetti almeno il formato in tutte le righe di output, incluse quelle che competono agli altri goal – altrimenti salta il protocollo di comunicazione tra il tuo programma risolutore e il server).

Esempio di Input/Output

Input da `stdin`

```
2
9
B B N N B B N N B
4
14
D B B N N N D B B N N B B D
8
```

Output su `stdout`

```
0  1  1  1  0  1  1  1  0
B  B  N  N  B  B  N  N  B
2  1  1  2 -1  3  3  4  4
1  0  1  1  1  1  1  1  0  1  1  1  0  1
D  B  B  N  N  N  D  B  B  N  N  B  B  D
0  2  1  1  2  3  0  3 -1  5  4  4  5  0
```

Assunzioni

1. $1 \leq n < 10^6$
2. in tutte le istanze che proporremo il soldato Ryan (y) può salvarsi

Subtask

Il tempo limite per istanza (ossia per ciascun testcase) è sempre di 1 secondo.

I testcase sono raggruppati nei seguenti subtask.

1. **[4 pts← 2 istanze da 1 + 1 punti] `esempi_testo`:** i due esempi del testo
2. **[8 pts← 4 istanze da 1 + 1 punti] `small_balanced`:** 7 guelfi contro 8
3. **[6 pts← 3 istanze da 1 + 1 punti] `medium_balanced`:** $n \leq 100$, $\lfloor \frac{n}{2} \rfloor$ guelfi contro $\lceil \frac{n}{2} \rceil$
4. **[6 pts← 3 istanze da 1 + 1 punti] `big_balanced`:** $n \leq 1000$, un colore ha un guelfo in più
5. **[8 pts← 4 istanze da 1 + 1 punti] `large_balanced`:** $n \leq 10^6$, una parte ha un guelfo in più
6. **[8 pts← 4 istanze da 1 + 1 punti] `small`:** $n \leq 15$
7. **[6 pts← 3 istanze da 1 + 1 punti] `medium`:** $n \leq 100$
8. **[6 pts← 3 istanze da 1 + 1 punti] `big`:** $n \leq 1000$
9. **[8 pts← 4 istanze da 1 + 1 punti] `large`:** $n \leq 10^6$
10. **[8 pts← 4 istanze da 1 + 1 punti] `small_in_the_heat`:** $n \leq 15$, con morti già all'avvio
11. **[6 pts← 3 istanze da 1 + 1 punti] `medium_in_the_heat`:** $n \leq 100$, con morti già all'avvio
12. **[6 pts← 3 istanze da 1 + 1 punti] `big_in_the_heat`:** $n \leq 1000$, con morti già all'avvio
13. **[8 pts← 4 istanze da 1 + 1 punti] `large_in_the_heat`:** $n \leq 10^6$, con morti già all'avvio

In generale, quando si richiede la valutazione di un subtask vengono valutati anche i subtask che li precedono, ma si evita di avventurarsi in subtask successivi fuori dalla portata del tuo programma che potrebbe andare in crash o comportare tempi lunghi per ottenere la valutazione completa della sottomissione. Ad esempio, chiamando[1, 2] :

```
rtal -s <URL>  connect -x <token> -a size=large_balanced
       brothers_in_arms  -- python my_solution.py
```

vengono valutati, nell'ordine, i subtask:

`esempi_testo`, `small_balanced`, `medium_balanced`, `big_balanced`, `large_balanced`.

Il valore di default per l'argomento `size` è `large_in_the_heat` che include tutti i testcase.

Consigli

1. Cercare di comprendere la struttura del problema puntando a buon caratterizzare quando un milite armato sia condannato o possa essere salvato. Nel fare questo potrai dover caratterizzare prima questioni più fondamentali ma ausiliarie che potrebbero sorgerti spontanee.
2. Un espediene classico per entrare nella struttura di un problema consta nell'isolare casi particolari e porsi buone domande. I subtask proposti suggeriscono casi particolari dove la struttura è più forte: ragionare su di essi può aiutare ad entrare nel problema per accendere la luce.
3. Questo problema gode di diverse invarianti ciascuna delle quali può facilitarti nell'ottenere comprensione come nel produrre una tua soluzione.
4. Anche il pensiero ricorsivo può aiutare nell'entrare nel problema e persino condurre ad una soluzione. Coniugato con la programmazione dinamica/memoizzazione si può pervenire ad algoritmi quadratici che cominciano a conquistare punti significativi. Il problema non si presta ad un algoritmo lineare di programmazione dinamica, ma a parte i punti che può già ottenere una dinamica quadratica può mettere sulla strada per cogliere le caratterizzazioni che poi conducano a soluzioni lineari.
5. In ogni caso ti conviene affrontare il problema per gradi, prima nelle comprensione e poi anche nella codifica.

[1] `<URL>` server esame: `wss://ta.di.univr.it/esame`

[2] `<URL>` server esercitazioni e simula-prove: `wss://ta.di.univr.it/algo`