

### Task 1. Hint

Doc Brown a reușit să își transforme DeLorean-ul într-o mașină a timpului. El are acum în vizor o problemă și mai mare: cel mai lung subșir comun. Când se dau două secvențe de numere  $A$  și  $B$  de lungimi  $N$  și  $M$ , el vrea să găsească cea mai lungă (sau una dintre cele mai lungi) secvențe  $C$ , astfel încât toate elementele din  $C$  apar atât în  $A$ , cât și în  $B$  în aceeași ordine (dar nu neapărat consecutiv) ca și în  $C$ . El a reușit să scrie un program oarecum lent care va rula mai multe zile până ce găsește o soluție. Cu toate acestea, el are nevoie de un răspuns cât mai curând posibil. Planul lui inițial era să lase programul să ruleze și după câteva zile să trimită datele de ieșire ale programului înapoi în timp la sinele lui prezent. Problema este că deplasările în timp necesită o cantitate enormă de energie, deci trimiterea întregii soluții înapoi în timp ar fi extrem de costisitoare.

Acum Doc are un nou plan, dar are nevoie de ajutorul tău ca să-l pună în aplicare. El vrea să trimită un scurt indiciu despre soluția din viitor către prezent și după să folosească indiciul pentru a reconstrui o soluție optimă pe baza acestuia. Reține că nu trebuie să fie neapărat aceeași soluție optimă ca cea din viitor.

Tu ar trebui să scrii un program `hint.cpp` care implementează două funcții: `genHint` și `solve`, care realizează planul lui Doc.

#### Detalii de implementare

Funcția ta `genHint` trebuie să aibă următorul antet:

```
std::vector<bool> genHint(const std::vector<int>& a, const std::vector<int>& b,  
    const std::vector<int>& sol);
```

Va fi apelată doar o dată și va primi ca parametrii cele două secvențe date și o soluție optimă. Ar trebui să returneze un indiciu, care va fi trimis către cealaltă funcție pe care o vei implementa.

Funcția ta `solve` trebuie să aibă următorul antet:

```
std::vector<int> solve(const std::vector<int>& a, const std::vector<int>& b,  
    const std::vector<bool>& hint);
```

Va fi apelată doar o dată și va primi ca parametrii cele două secvențe date și indiciul pe care cealaltă funcție l-a returnat. Ar trebui să returneze o soluție optimă.

În afară de aceste două funcții, programul tău poate să aibă orice fel de funcții auxiliare, clase, variabile, etc. Cu toate, **nu trebuie** să conțină o funcție `main` și **trebuie** să includă fișierul header `hint.h`. **Rețineți că pe platforma de evaluare cele două funcții implementate de tine vor fi apelate în instanțe separate ale programului tău, rulând în procese separate.** Asta înseamnă că nu veți putea partaja nicio stare globală între rulările celor două funcții.

#### Testare locală

Ți se va da un fișier `Lgrader.cpp`, pe care îl poți compila împreună cu programul tău pentru a-l testa. El va citi  $N$  și  $M$ , urmat de  $A$  și  $B$ . După aceea va citi lungimea soluției optime  $K$  și soluția optimă  $C$ . Va afișa lungimea indiciului din funcția ta `hint`, precum și soluția pe care funcția ta `solve` a returnat-o. Rețineți că, spre deosebire de platforma de evaluare, grader-ul local nu rulează funcțiile în procese separate.

#### Restricții

$$1 \leq N, M \leq 10^5$$
$$0 \leq A_i, B_j < \min(N, M)$$

**Subtask-uri**

Subtask	Puncte	$N, M$
1	10	$\leq 10^4$
2	90	$\leq 10^5$

Soluția ta va primi punctele pentru un subtask doar dacă trece toate testele acestuia.

**Punctare**

Scorul pe care îl vei primi pentru un subtask depinde de lungimea maximă  $L$  a indiciului pe care programul tău l-a generat pentru un test în acel subtask. Proporția de puncte pe care îl vei primi pentru subtask va fi:

$$\min\left(\left(\frac{640}{L+1}\right)^{0.3}, 1\right)$$