Tema Learn & Earn

Problema 1

Se citesc de la tastatura N elemente numere intregi care se stocheaza si reprezinta intr-o structura de date SD. Propuneti doi **algoritmi eficienti** care determina al k-lea cel mai mic element si al k-lea cel mai mare element din multimea data, fara sa se utilizeze memorie suplimentara (in afara de cea ocupata de SD) si fara sa se numere elementele.

Punctaj:
Alegerea corecta a structurii de date si justificarea alegerii (analiza complexității algoritmilor) – 10p
Implementarea operatiei de inserare a unui nou element in colectia reprezentata cu SD propusa – 10p
Determinare al k-lea cel mai mic element – 10p
Determinare al k-lea cel mai mare element – 10p
Eleganta implementarii – stil de codare clar si consistent, utilizare conventii consistente de numire a variabilelor/functiilor, documentarea codului, etc. – 10p

Problema 2

Fie T un şir de n caractere şi P un pattern de m caractere. Să se verifice dacă P apare sau nu ca subsecvenţă în şirul T.

Indicaţie:

Se va parcurge șirul T, caracter cu caracter, și, pentru fiecare poziție i se va verifica potrivirea șirului P cu șirul $T[i…i+m-1]$ folosind metoda descrisă în continuare (nu prin verificarea egalității caracter cu caracter).

Fie b numărul de caractere distincte care apar în T (acest număr b reprezintă cardinalul alfabetului folosit).

Un şir de lungime m poate fi considerat un număr în baza $b+1$ având m cifre (se folosesc cifre de la 1 la b).

Se asociază pattern-ului P o valoare p (şirul de caractere este considerat ca un număr întreg în baza $b+1$).

Pentru fiecare secvenţă de m caractere din şirul T se calculează t_i, valoarea în baza $b+1$ a subsecvenţei din T care începe la poziţia i ($T[i…i+m-1]$). Evident, $p=t_i$ dacă şi numai dacă $T[i…i+m-1]$.

Valorile t_i pentru i=1,… $i+m-1$ se pot calcula recurent astfel:

$$t_{i+1} = (b+1) * (t_i - (b+1)^{m-1} * T[i]) + T[i+m].$$

Dificultatea constă în faptul că p şi t_i pot fi numere foarte mari. Prin urmare, operaţiile cu aceste numere nu se execută în timp constant.

Se va lucra cu p şi t_i mod Q, unde Q este un număr prim convenabil ales, mai exact se foloseşte o funcţie de dispersie. Deoarece $p \bmod Q = t_i \bmod Q$ nu implică faptul că $p=t_i$, și cu atât mai puțin că $P=T[i \ldots i+m-1]$, apar coliziuni. Dar dacă $p \bmod Q \neq t_i \bmod Q$, atunci sigur $P \neq T[i \ldots i+m-1]$.

Aşadar, acest algoritm este probabilistic, eficienţa în detectarea potrivirii pattern-ului depinde de funcţia de dispersie. Pentru a elimina potrivirile *"false"*, se va aplica o verificare caracter cu caracter pentru fiecare potrivire detectată. Alternativ, se pot folosi mai multe funcţii de dispersie.

Precizări:

- Șirul T și pattern-ul P vor fi citite de la tastatură
- Programul va afișa:
    - În cazul în care pattern-ul este identificat: indexul din șirul de intrare la care se regăsește primul caracter din P; dacă sunt identificate mai multe instanțe ale lui P, se afișează toți acești indecși.
    - În cazul în care pattern-ul nu este identificat: un mesaj care să indice acest lucru.
- Precizați complexitatea timp pentru cazul mediu si pentru cazul cel mai defavorabil al algoritmului de mai sus. Cum se compara cu cautarea brute-force ?
- Pentru stabilirea valorii b, consultati https://www.asciitable.com/
- Relația de recurența de mai sus va fi adaptată pentru a lucra cu p şi t_i mod Q

Punctaj:
Implementarea rezolvării conform indicației de mai sus – 50p
Rezolvarea coliziunilor prin rehashing – 10p
Analiza complexității – 5p
Eleganta implementarii – stil de codare clar si consistent, utilizare conventii consistente de numire a variabilelor/functiilor, documentarea codului, etc. – 10p