4. mājasdarbs

Lietišķie algoritmi, 2020.g. rudens
Termiņš: 2021-01-04

Atrisinājumus lūdzam pārveidot par vienu PDF.

1.uzdevums (Bojera-Mūra algoritms).
(A) Uzrakstīt Bojera-Mūra algoritmā lietotās tabulas, ja meklējamā apakšvirkne P ir `abcbcab`.
(B) Nodemonstrēt Bojera-Mūra algoritma darbību šīs apakšvirknes meklēšanai tekstā $T =$ `abcabbcabcbcababababcbcab`.

2.uzdevums (Vispārināta prefiksu funkcija).
Knuta-Morisa-Prata algoritms izmanto Galīgu Determinētu Automātu (*Deterministic Finite Automaton*, DFA), lai iegaumētu garāko iespējamo prefiksu meklējamam paraugam S ievades tekstā T. Pārejas šajā DFA automātā var attēlot ar *prefiksu funkciju* jeb tabulu `memo[i]`, kur $i \in \{0, \ldots, \ell - 1\}$. Šie jēdzieni definēti `https://bit.ly/3fXWnWT`, Section 1.
Aplūkosim šī klasiskā uzdevuma variantu: Joprojām ievadē saņemam tekstu T, kuru lasām no kreisās uz labo pusi. Bet šoreiz jāmeklē jebkurš no diviem stringiem – vai nu $S_1 =$ `1110111101`, vai arī $S_2 =$ `10101`. (Ja izdodas atrast pirmo atbilstību, kur **vai nu** S_1 **vai arī** S_2 ietilpst tekstā T, tad automāts sasniedz savu beigu stāvokli un apstājas.)
Cik stāvokļi nepieciešami DFA automātā un kā pielāgot datu struktūru (līdzīgu tabulai `memo[i]`), lai iekodētu DFA stāvokļu pārejas?

3.uzdevums (Teksts ar zvaigznītēm).
Dots ievades teksts $T[0..n-1]$ garumā n, kas sastāv no diviem simboliem `a` un `b`. Pieņemsim arī, ka dots meklējamais paraugs $S[0..\ell-1]$ garumā ℓ (kur ℓ daudz mazāks par n), kas satur simbolus `a`, `b`, un arī $*$, kuru vajadzēs meklēt tekstā T. Šeit simbols $*$ ir "visēdājs" (*wild card*), kuram atbilst viens burts – gan `a`, gan `b`. (Parauga S simboliem, kuri nav zvaigznītes, jāsakrīt ar tiem, kas ir tekstā.)
Uzdevums: Izvadīt sakārtotu sarakstu M, kurā ir sarakstītas visas derīgās nobīdes, kā paraugu S var ievietot tekstā T tā, lai tas sakristu. Piemēram, ja $T =$ `ababbab` un $S =$ `ab`$*$, tad izvade ir $[0, 2]$. Aplūkojam Knuta-Morisa-Prata algoritma variantu, kas lasa meklējamo tekstu T no kreisās uz labo pusi un (katrā reizē, kur teksts T neatbilst paraugam S) nobīda paraugu S uz priekšu par mazāko iespējamo burtu skaitu (tā, lai jau saņemtie teksta T simboli nebūtu pretrunā ar jauno parauga stāvokli). Mēs vēlamies meklēt paraugu $S =$ `11`$*$`0111`$*$`01` jebkurā tekstā.
(A) Uzrakstiet šādi modificēta KMP algoritma pseidokodu, ja tas atšķiras no lekcijā dotā (sk. `https://bit.ly/3oQGt58`).
(B) Aizpildiet tabulu ar nobīdēm (ko varētu saukt arī par *prefiksu funkciju* jeb tabulu `memo[i]`), kas parāda, par cik jālec uz priekšu, ja T neatbilst paraugam $S =$ `11`$*$`0111`$*$`01` meklēšanas pozīcijā i.
Šī ir parodija par Uzdevumu 2.1, sk. `https://bit.ly/2XKX5AB`, 1.lappusi.

4.uzdevums (Ripojošais hešings).
Aplūkojam sekojošu funkciju, kas no stringiem S (garumā ℓ) iegūst veselus skaitļus.

$$k(S) = \Big(S[0] \cdot b^{\ell-1} + S[1] \cdot b^{\ell-2} + \ldots \\ \ldots + S[\ell-1] \cdot b^1 + S[\ell-1] \Big) \mod q. \tag{1}$$

Šeit b (ko sauc par "skaitīšanas sistēmas bāzi") ir skaitlis, kas lielāks par alfabēta izmēru (simbolu skaitu, ar kuru uzrakstīts strings S).
To pašu funkciju var lietot arī garāka teksta T apakšstringiem (apstrādājot tieši ℓ simbolus):

$$\begin{aligned} & k(T[i..i+\ell-1]) = \\ = & \Big(T[i]\cdot b^{\ell-1} + T[i+1]\cdot b^{\ell-2} + \dots \\ & \dots + T[i+\ell-2]\cdot b^{1} + T[i+\ell-1]\Big) \bmod q. \end{aligned} \tag{2}$$

Aplūkojam arī šādu funkciju (sk. *reizināšanas metodi* no `https://bit.ly/2V8UfDF`)

$$h(k) = [(a\cdot k) \bmod 2^w] \;\gg\; (w-r), \tag{3}$$

Šajā izteiksmē

- $\gg$ apzīmē operatoru bitu nobīdei pa labi,
- $2^r = m$ ir heštabulas izmērs,
- w ir fiksēts "mašīnvārda garums" (4 baiti, 8 baiti vai jebkas cits, ko ērti glabāt masīvā Jūsu arhitektūras datoram).
- a ir nepāra skaitlis starp 2^{w-1} un 2^w.

Aplūkosim "ripojošo hešfunkciju" (*rolling hash*)

$$h(k(T[i..i+\ell-1]))$$

Rabina-Karpa stringu meklēšanas algoritmā. Vēlamies atrast paraugus S (garumā $\ell \le 100$) garā tekstā T. Gan S, gam T pierakstīti ASCII alfabētā, izmantojot visus 128 simbolus.
(A) Kā aprēķināt `r.append(c)` un `r.skip(c)` operācijas ripojošā hešinga ADT, ja hešingu definē izteiksmes (1), (2) un (3).
Ripojošā hešinga abstraktais datutips (ADT) un Rabina-Karpa algoritms ir izskaidroti video `https://bit.ly/2BxpNMu` (no 38:00).
(B) Uzrakstīt laika sarežģītību Rabina-Karpa algoritmam, izmantojot $O(\dots)$ asimptotiku ("lielā O apzīmējumu") šai konkrētajai hešinga metodei. Apzīmējiet teksta T garumu ar $n = |T|$, un meklējamā parauga $|S|$ garumu ar $\ell = |S|$.

Piezīme. Sk. 4-1 no `https://bit.ly/3dyNtgK`.

Piezīme. Pilna hešfunkcija ir šajā uzdevumā ir $h(k(S))$. Mainīgais a ir fiksēts. Tas ir arī iebūvētajās hešfunkcijās (valodās Java vai Python) lietots algoritms: Hešings nozīmē argumenta reizināšanu ar konstanti, pēc tam nosakot atlikumu pēc liela moduļa.

Piezīme. Mainīgais `r` ir objekts, kurā uzkrāta "tekošā" hešfunkcijas vērtība (un pēc Jūsu izvēles arī cita informācija, kas vajadzīga hešfunkcijas rēķināšanai). Objektam izsauc `r.append(c)`, tad hešfunkciju aprēķina no eksistējošā `r` stāvokļa, bet pievieno tam labajā pusē simbolu, kas atrodas mainīgajā "c" (tas ir viens ASCII simbols - no 0 līdz 127). Ja šim objektam izsauc `r.skip(c)`, tad hešfunkcijas objekts "r" atmet kreisajā pusē simbolu, kas atrodas mainīgajā "c".
Piemēram, ja objekts `r` inicializēts uz hešfunkcijas vērtību $h(k(\texttt{"ABCD"}))$, tad `r.append("E")` atradīs jau hešfunkcijas vērtību $h(k(\texttt{"ABCDE"}))$. Bet, pēc tam izsaucot `r.skip("A")`, objekts `r` ir jau uzstādīts uz hešfunkcijas vērtību $h(k(\texttt{"BCDE"}))$. Šo darbību rezultātā sākotnējais "teksta logs" "ABCD" pabīdījies uz "BCDE".
Rabina-Karpa algoritma būtība ir – atrast šos `r.skip(...)` `r.append(...)` ātri, konstantā laikā, lai hešvērtības nav jārēķina neatkarīgi viena no otras.

5.uzdevums (I-iespēja).

Dota $n \times m$ tabula, katrā rūtiņā ir ierakstīts tieši viens simbols. Teiksim, ka vārds ir paslēpts tabulā, ja to var nolasīt, sākot kādā rūtiņā un katru nākamo simbolu nolasot no rūtiņas, kurai ar iepriekšējo ir kopīga mala. Vārds, paslēpts tabulā, drīkst arī pārklāties pats ar sevi.
Piemēram, šajā tabulā ir paslēpti vārdi `SAULE` un `SOS`:

S	A	E
O	U	L

Aprakstīt algoritmu, kas noskaidro, vai dotais vārds garumā w paslēpts tabulā un pamatot, ka tas strādā laikā $O(wnm)$.