Lietišķie algoritmi – 1. mājas darbs: Atrisinājumi

1.uzdevums: Aritmētiskais kods. Dota ziņojumu kopa $S = \{A, B, C, D\}$ ar attiecīgajām varbūtībām $\{0.2, 0.5, 0.2, 0.1\}$.

(a) Parādīt, kā iegūt aritmētisko kodu 6 ziņojumu virknei `CBAABD` – uzkonstruēt tai atbilstošo intervālu $[l_6; l_6 + s_6) \in [0; 1]$ un atrast īsāko bitu virkni $d_1 d_2 \dots d_\ell$ (visi $d_k \in \{0, 1\}$, kur pierakstot binārā pieraksta daļskaitlim $D = 0.d_1 d_2 \dots d_\ell \dots$ galā jebkuru turpinājumu ar ciparem 0 vai 1, iegūtais skaitlis $d + \varepsilon$ pieder intervālam $[l_6; l_6 + s_6)$.

(b) Noteikt, kādu ziņojumu virkni alfabētā S kodē skaitlis $D'' = 0.0011101011_2$.

Piezīme: Praksē lietojamie aritmētiskie kodi vai nu pievieno ziņojumu alfabētam EOT (teksta beigu) ziņojumu vai arī kaut kur iekodē ziņojumu virknes garumu, lai atkodētājs zinātu, kur apstāties.

2.uzdevums: Lempela-Ziva algoritms.

(a) Ar LZ78 metodi nokodēt tekstu "abracadabra, abracadabra".

(b) Atkodēt ar LZ78 metodi nokodētu tekstu $a, b, c, d, 2, 5, a, 6$, kur a, b un c apzīmē atbilstošos burtus, bet skaitļi – vārdnīcas virkņu numurus.

(c) Nokodēt (a) punkta tekstu "abracadabra, abracadabra" ar LZ77 metodi, kā logu lietojot visu nokodēto/atkodēto tekstu.

(a)

3.uzdevums: Berouza-Vīlera transformācija.

(a) Kāds ir rezultāts (transformētā simbolu virkne un sākotnējās virknes pozīcija), lietojot Berouza-Vīlera transformāciju 14 simbolu virknei `alusariirasula`?

(b) Kāds ir iepriekšējā piemērā iegūtās transformētās simbolu virknes pieraksts, izmantojot Move-to-Front kodēšanu?

(c) Pēc BW transformācijas pielietošanas tika iegūta simbolu virkne `mmmrvvauuuiibbbri`. Kāda bija simbolu virkne pirms transformācijas (ņemot 4. virkni no atjaunotās tabulas)?

(a) Ja lietojam to Berouza-Vīlera transformāciju, kāda tā aprakstīta Vikipēdijā: `https://en.wikipedia.org/wiki/Burrows%E2%80%93Wheeler_transform` –piemērā ar vārdu `^BANANA|`, tad izrakstām visas 14 cikliskās permutācijas vārdam `alusariirasula` ($0, 1, 2, \dots, 13$ burtus no šī vārda beigām pārceļam uz tā sākumu). Pēc tam šīs permutācijas (rindiņas zemāk attēlotajā tabulā) sakārtojam leksikogrāfiski: vienkārši sakot, pēc alfabēta: Vispirms izrakstām visas permutācijas, kam pirmais burts ir "a" (tās sašķiro pēc 2.burta; ja arī tas sakrīt, pēc 3.burta, utt.).

Pēc tam ievērojam, ka pēdējā tabulas kolonnā rakstīts `lasrriuaiauasl`. Mūsu sākotnējās virknes `alusariirasula` indekss ir 1 (šī virkne redzama tabulas 2.rindiņā), bet indeksus sāk numurēt, sākot no 0: 2.rindiņas indekss tāpēc ir 1.

```
aalusariirasul
alusariirasula
ariirasulaalus
asulaalusariir
iirasulaalusar
irasulaalusari
laalusariirasu
lusariirasulaa
rasulaalusarii
riirasulaalusa
```

`sariirasulaalu`
`sulaalusariira`
`ulaalusariiras`
`usariirasulaal`

(b) Kāds kods rodas no Move-to-Front, ja to lieto virknei `lasrriuaiauasl`, kur alfabētiskais sakārtojums ir `a` < `i` < `l` < `r` < `s` < `u`, kur burtu indeksus sāk skaitīt, sākot no 0.indeksa?
Rakstām tabulu, kurai augšējā rindiņā ir kodējamais vārds, bet alfabētus rakstām vertikāli uz leju:

Alfabēts↓	l	a	s	r	r	i	u	a	i	a	u	a	s	l
a	l	a	s	r	r	i	u	a	i	a	u	a	s	l
i	a	l	a	s	s	r	i	u	a	i	a	u	a	s
l	i	i	l	a	a	s	r	i	u	u	i	i	u	a
r	r	r	i	l	l	a	s	r	r	r	r	r	i	u
s	s	s	r	i	i	l	a	s	s	s	s	s	r	i
u	u	u	u	u	u	u	l	l	l	l	l	l	l	r
Izeja:	2	1	4	4	0	4	5	4	2	1	2	1	4	6

(a) un (b) citam Berouza-Vīlera transformācijas variantam:
Ja izmantojam Guy E.Blelloch grāmatu "Introduction to Data Compression", tad Berouza-Vīlera transformāciju iegūst, sakārtojot cikliskās permutācijas leksikogrāfiski pēc priekšpēdējā burta, ja sakrīt, tad pēc priekšpriešpēdējā, utt. Transformācijas rezultāts arī šajā gadījumā ir pēdējā kolonna.

`usariirasulaal`
`lusariirasulaa`
`ulaalusariiras`
`iirasulaalusar`
`asulaalusariir`
`rasulaalusarii`
`sariirasulaalu`
`alusariirasula`
`irasulaalusari`
`sulaalusariira`
`laalusariirasu`
`riirasulaalusa`
`aalusariirasul`
`ariirasulaalus`

Berouza-Vīlera transformācijas rezultāts ir pēdējā kolonna `lasrriuaiauals`.

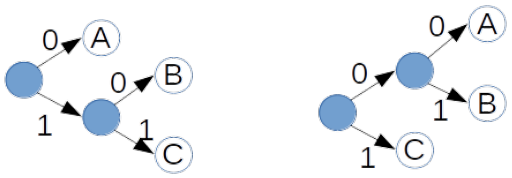
4.uzdevums: I-iespēja (atzīmei 10). Pieņemsim, ka ziņojumu kopai $S = \{x_1, x_2, \ldots, x_n\}$ ar izveidots optimāls prefiksu kodējums. Šis kodējums jāpārraida, izmantojot minimālu bitu skaitu.
Pierādīt vai apgāzt šādu apgalvojumu: Jebkuru optimālu prefiksu kodējumu šai n ziņojumu kopai var nosūtīt, izmantojot ne vairāk kā $2n - 1 + n\lceil log_2 n\rceil$ bitus. Šeit $\lceil x\rceil$ apzīmē noapaļošanu uz augšu jeb mazāko veselo skaitli, kas nav mazāks par x. Piemēram $\lceil 17\rceil = 17$ un $\lceil 3.14\rceil = 4$.
Ieteikums. Izmantojot $2n - 1$ bitus, var attēlot kodējumu koka virsotņu apstaigāšanas secību.

Pamatosim, ka prefiksu koku var nosūtīt, izmantojot ne vairāk kā $2n - 1 + n\lceil log_2 n\rceil$ bitus.

Tad, ja būtu jānosūta vienkārši n alfabēta simbolu permutācija (viens no $n!$ iespējamajiem šo ziņojumu izkārtojumiem, kurā katrs ziņojums parādās tieši vienu reizi), tad pietiktu ar $n\lceil log_2 n\rceil$

bitiem. Teiksim, 3 simbolu permutācijai (kādai no virknēm `ABC`, `ACB`, `BAC`, `BCA`, `CAB` vai `CBA`) iztērējas $3 \cdot 2 = 6$ biti, jo katru no trim burtiem var uzrakstīt ar 2 bitu virknīti.

Tomēr jāatceras, ka mūsu mērķis ir nosūtīt nevis simbolu permutāciju, bet bināru koku, kurā šie simboli parādās kā "koka lapas" (balti aplīši zīmējumā).



Zīmējums: 2 prefiksu koki ziņojumu alfabētam $S = \{A, B, C\}$.

Lai šos kokus varētu atšķirt, var izmantot bināro operatoru $*$ un iekavas. Piemēram:

$$(\texttt{A} * (\texttt{B} * \texttt{C})) \text{ un } ((\texttt{A} * \texttt{B}) * \texttt{C}).$$

Šajā sintaksē simbols $*$ apzīmē divu bināru koku salīmēšanas darbību. Ir vēl taupīgāks pieraksta veids – "apgrieztais poļu pieraksts" (*Reverse Polish Notation*), kurā $(\texttt{A} * \texttt{B})$ aizstāj ar `AB`$*$. Šajā sintaksē iekavas nav vajadzīgas, bet koku vienalga var atjaunot. Zīmējumā attēlotie prefiksu koki apgrieztajā poļu pierakstā būs šādi:
`ABC`$**$ un `AB`$*$`C`$*$

Vispārīgajā gadījumā, lai izveidotu šādu poļu pieraksta izteiksmi no n simboliem, jāpieraksta šie simboli kaut kādā secībā un vēl $n-1$ zvaigznītes (binārā kokā ar n lapām būs tieši $n-1$ iekšējas virsotnes – lai no n gabaliņiem izveidotu vienu koku, jālīmē kopā tieši $n-1$ reizes). Tā kā zvaigznīšu ir daudz, tās kodēsim vienkārši ar bitu "1", bet katram ziņojumu alfabēta S simbolam x_i rakstīsim priekšā bitu "0".

Kodējuma piemēri: Aplūkosim jau minēto prefiksu koku `(A*(B*C))`, kurš zīmējumā redzams pa kreisi: Tajā simbolu `A` kodē ar virknīti "0", simbolu `B` kodē ar "10", bet simbolu `C` kodē ar "11".

Šī prefiksu koka apgrieztais poļu pieraksts ir `ABC`$**$. Tā kā saņēmējs vēl nezina, kā tiks kodēti ziņojumu alfabēta simboli, tad kodējam tos vienkārši pēc kārtas ar vienāda garuma virknēm, kas visas sākas ar bitu "0". Simbola `A` fiksētā garuma kods (prefiksu koka nosūtīšanai, nevis Hafmana kodēšanai ar mainīgā garuma kodu) būs virkne "000", `B` kods būs virkne "001", `C` kods būs "010". Tā kā $n = 3$ nav divnieka pakāpe, tad virkne "011" paliek neizmantota.

Pierakstu `ABC`$**$ tātad kodē virkne "000 001 010 1 1" (atkodēšanai atstarpes nevajag). Savukārt citu bināro koku `AB`$*$`C`$*$, kas zīmējumā redzams pa labi, apzīmē virkne "000 001 1 010 1".

Patvaļīgam n prefiksu koka kodējums sastāv no n dažādiem ziņojuma simboliem – katru no tiem kodē garumā $\lceil \log_2 n \rceil + 1$ biti. Vajadzīgas arī $n-1$ zvaigznītes. Katru no tiem kodē ar 1 bitu. Kopīgais bitu skaits:

$$n \cdot (\lceil \log_2 n \rceil + 1) + (n-1) \cdot 1 = 2n - 1 + n \lceil \log_2 n \rceil .$$

■