1. mājasdarbs

Lietišķie algoritmi, 2019.g. rudens
Termiņš: 2019-09-30

Ieteicamā lasāmviela. Guy Blelloch. Introduction to Data Compression. 2013, pp.16–19. Sk. `https://bit.ly/3f9jzDD`.

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 $D'' = 0.0011101011_2$.

(a) Sākotnējais intervāls $[l_0; l_0 + s_0] = [0; 1]$. Uzrakstām tālākos sešus intervālus:

$$\begin{aligned} I_1 &= [0.70000, 0.90000) \\ I_2 &= [0.74000, 0.84000) \\ I_3 &= [0.74000, 0.76000) \\ I_4 &= [0.74000, 0.74400) \\ I_5 &= [0.74080, 0.74280) \\ I_6 &= [0.74260, 0.74280) \end{aligned}$$

Pēdējo intervālu I_6 kodē bitu virkne `1011111000100`, jo skaitlis $0.1011111000100_2 = \frac{6084}{2^{13}}$ un

$$\left[\frac{6084}{2^{13}}; \frac{6085}{2^{13}}\right] \subseteq [0.74260, 0.74280).$$

Kodiem garumā 13 citus skaitļus (izņemot jau atrastos 6064/8192 un 6065/8192) izmantot nevar: intervāls pilnībā neietilps I_6.
Piezīme: Ja esat ņēmuši kodu, kurš garāks par 13, tad iespējami arī citi atrisinājumi. Tos neuzskatām par nepareiziem, ja vien Jūsu kodam atbilstošais intervāls pilnībā ietilpst intervālā I_6. Praksē lietojamie aritmētiskie kodi vai nu pievieno ziņojumu alfabētam EOF (faila beigu) ziņojumu vai nu arī kaut kur iekodē ziņojumu virknes garumu, lai atkodētājs zinātu, kur apstāties.
Īpašs "pseido EOF" simbols mēdz būt aktuāls pat Hafmana koda vingrinājumos. Atšķirībā no bināriem skaitļiem, Hafmana prefiksu kodu virknē it kā ir redzams, kur vajag apstāties, bet praksē tas bieži nenostrādā, jo failam jāsatur vesels skaits baitu; pēc visu Hafmana kodavārdu izrakstīšanas var gadīties daži lieki biti, kurus negribam atkodēt. Plašāku diskusiju par "pseido EOF" sk. Stenfordas universitātes uzdotajā programmēšanas mājasdarbā:
`https://web.stanford.edu/class/archive/cs/cs106b/cs106b.1172/assn/huffman.html`.
(b) Sākam ar intevālu

$$[0.0011101011_2; 0.0011101100_2) = [235/1024; 236/1024) \approx [0.22949218750.2304687500).$$

Turpmākajos soļos noskaidrojam, kurā no ziņojumiem $\{A, B, C, D\}$ atbilstošajiem intervāliem ($[0; 0.2)$, $[0.2; 0.7)$, $[0.7; 0.9)$, $[0.9; 1.0)$) šis intervāls ietilpst. Tiklīdz kā esam to atraduši,

tad pierakstām atkodēto ziņojuma burtu un intervālu (kādu no $[0;0.2)$, $[0.2;0.7)$, $[0.7;0.9)$, $[0.9;1.0)$) "izstiepjam" tā, lai tas būtu garumā $[0;1]$. Pēc tam šo soli atkārto tikmēr, kamēr sākotnējais intervāls vairs nav viennozīmīgi ievietojams nevienā no tiem. Tad atkodēšana beidzas.

- 1.solis: $I_1 = [0.22949218750.2304687500) \subseteq [0.2;0.7)$. Atkodēts `B`.
- 2.solis: No abiem I_1 galapunktiem atņem 0.2 un dala ar $[0.2;0.7)$ garumu jeb 0.5: $I_2 = [0.0589843750.060937500) \subseteq [0;0.2)$. Atkodēts `A`.
- 3.solis: No abiem I_2 galapunktiem atņem 0 un dala ar $[0;0.2)$ garumu jeb 0.2: $I_3 = [0.2949218750.304687500) \subseteq [0.2;0.7)$. Atkodēts `B`.
- 4.solis: Atņem 0.2, dala ar 0.5: $I_4 = [0.189843750.20937500)$. Intervāls vēl ir pietiekami īss, bet tas vairs nav viennozīmīgi atkodējams, jo šķeļas gan ar $[0;0.2)$, gan $[0.2;0.7)$.

Atkodētais ziņojums ir `BAB`.
Piezīme: Praksē aritmētiskā koda atkodēšanas algoritms tiek pārcelts no peldošā punkta aritmētikas uz veselo skaitļu aritmētiku. Šajā gadījumā atkodē nevis visu intervālu, bet tikai tā kreiso galapunktu - piemēram, skaitli 235/1024. Viens skaitlis allaž nonāks tieši vienā no 4 apakšintervāliem un allaž varēs atkodēt. Atkodēšanu turpina tikmēr, kamēr sasniegts "pseido EOF" simbols, par kuru rakstījām augstāk. Ja par "pseido EOF" uzskata ziņojumu `D`, tad, atkodēšana rada citu atbildi `BABAD`. Ja kādam bija šāda atbilde, tad arī tā atbilst citam aritmētiskā koda dialektam.
Ja atbilde ir cita (nevis `BAB` vai `BABAD`), tad Jūsu aprēķinos ir kļūda.

□

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) Teksts satur alfabēta simbolus "`a`", "`b`", "`c`","`d`","`r`",","," ", ieskaitot komatu un tukšumu.

Solis	w	k	Izvade	Pievieno vārdnīcai
1	`a`	`"b"`	`a`	`ab`
2	`b`	`"r"`	`b`	`br`
3	`r`	`"a"`	`r`	`ra`
4	`a`	`"c"`	`a`	`ac`
5	`c`	`"a"`	`c`	`ca`
6	`a`	`"d"`	`a`	`ad`
7	`d`	`"a"`	`d`	`da`
8	`ab`	`"r"`	`ab` $\to$ 1	`abr`
9	`ra`	`","`	`ra` $\to$ 3	`ra,`
10	`","`	`" "`	`","`	`", "`
11	`" "`	`a`	`" "`	`" a"`
12	`abr`	`a`	`abr` $\to$ 8	`abra`
13	`ac`	`a`	`ac` $\to$ 4	`aca`
14	`ad`	`a`	`ad` $\to$ 6	`ada`
15	`abra`	`EOF`	`abra` $\to$ 12	–

Izvade: a.b.r.a.c.a.d.1.3.”,”.” ”.8.4.6.12.
Piezīme. Komatu un atstarpi likām pēdiņās un koda izvades atdalījām ar punktiem. (Reālā algoritma izvadē kodus atdala citādi - ar baitu robežām; tāpēc punkti un pēdiņas ir tikai pieraksta ērtība, nevis izvades sastāvdaļa.) Atkarībā no tā, vai risinātājs tukšumu starp vārdiem pamanīja un uzskatīja par ziņojumu, iespējams arī cits, līdzīgs kodējums, par kuru vērtējums netiek samazināts.
(b)

Solis	w	k	Izvade	Pievieno vārdnīcai
1	a	b	a	ab
2	b	c	b	bc
3	c	d	c	cd
4	d	b	d	db
5	bc	b	bc $\to$ 2	bcb
6	bcb	a	bcb $\to$ 5	bcba
7	bcba	EOF	bcba $\to$ 6	–

Atkodējums: a.b.c.d.2.5.a.6 $\to$ a.b.c.d.bc.bcb.a.bcba $\to$ abcdbcbcbabcba.
(c) Katrā LZ77 algoritma solī apzīmējam kursora stāvokli ar mazu pasvītrojumzīmi zem burta. Garāko kopīgo apakšvirkni meklēšanas logā krāsojam sarkanu. Iekavas, pēdiņas un semikoli mūsu ziņojumu alfabētā neietilpst, tie ir metasimboli, lai vieglāk uztvert LZ77 algoritma izvadi.
a̲bracadabra, abracadabra – izvade $(0; 0; \text{”a”})$
ab̲racadabra, abracadabra – izvade $(0; 0; \text{”b”})$
abr̲acadabra, abracadabra – izvade $(0; 0; \text{”r”})$
abra̲cadabra, abracadabra – izvade $(3; 1; \text{”c”})$
abraca̲dabra, abracadabra – izvade $(5; 1; \text{”d”})^*$
abracada̲bra, abracadabra – izvade $(7; 4; \text{”,”})$
abracadabra,̲abracadabra – izvade $(0; 0; \text{” ”})$
abracadabra, a̲bracadabra – izvade $(13; 11; -)$

Piezīme. Dažreiz logā (jau nokodētajā teksta gabalā) atrodas vairāki visgarākie apakšstringi; var izvēlēties starp diviem identiskiem apakšstringiem, kuru kopēt. Vēlāk kursā aplūkosim Bojera-Mūra garākās kopīgās apakšvirknes meklēšanas algoritmu; tas (līdzīgi citiem līdzīga veida algoritmiem) sāk virkņu salīdzināšanu no kreisās uz labo pusi. Tāpēc aplūkotajā piemērā, 5.solī izvadītais kods ir $(5, 1, \text{d})^*$ (kaut arī varēja rakstīt arī $(2, 1, \text{d})$ jeb skatīties atpakaļ nevis piecas pozīcijas, bet tikai divas). No atkodēšanas viedokļa nekādas atšķirības nav; tāpēc šajā piemērā (lai būtu konsekventi) rakstām pirmo atrasto pozīciju no kreisās puses. Bet par kļūdu neuzskatām arī jebkuru citu pozīciju, ja arī tā ir visgarākā. Diskusiju par šo apakšvirkņu meklēšanas metodēm sk. https://bit.ly/2pD7m2Y (*Longest-match String Searching for Ziv–Lempel Compression* by Timothy Bell, David Kulp).

□

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://bit.ly/3x6rojq` –piemērā ar vārdu `^BANANA|`, tad izrakstām visas 14 cikliskās permutācijas vārdam `alusariirasula` $(0, 1, 2, \ldots, 13$ 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. Tas ir cits vārds nekā lasrriuaiauasl (pēdējie 2 burti ir pretējā secībā). Arī Move-to-Front kods izskatās citādi:

21440454212166 nevis 21440454212146

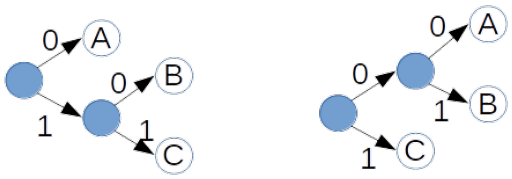
kā iepriekšējā variantā.

□

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:

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

Šajā sintaksē simbols $*$ apzīmē divu bināru koku salīmēšanas darbību. Ir vēl taupīgāks pieraksta veids – "apgrieztais poļu pieraksts" (*Reverse Polish Notation*), kurā $(\mathtt{A} * \mathtt{B})$ 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:

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

□