EIO õppesessioon. Pakkimine

Konstantin Tretjakov

29. aprill 2006. a.

Soovi korral võib lahendusi, küsimusi või kommentaare saata aadressile `kt@ut.ee`.

Informatsioon ja kodeerimine

1. Selgita, kuidas Sina saad aru mõistest *informatsioon*. Kuidas on arvutid seotud informatsiooniga? Kuidas saab informatsioon arvutisse?

2. Mida tähendavad mõisted *kodeerimine* ja *kood*? Too näiteid, kus ühte ja sama informatsiooni kodeeritakse erinevalt (vihje: failid). Leia vähemalt 6 erinevat viisi mõiste *arv 42* kodeerimiseks.

3. Miks kasutatakse erinevaid koode ning ei lepita kokku ühes kodeerimisviisis? Milliseid omadusi võib koodilt nõuda? Kas nad on vastuolulised?

4. Selgita mõistet *pakkimine*, kasutades terminit *kodeerimine*.

5. Kuidas esitada mõistet *arv 42* võimalikult lühidalt? Mitu bitti selleks kulub?

6. Eelmisest küsimusest peaks välja tulema, et pole väga mõttekas rääkida üheainsa mõiste kodeerimisest. Millest on siis mõttekas rääkida? Selgita veelkord mõistet *kood*, seekord täpsemalt.

7. Olgu meil nüüd kolm mõistet. Tähistame neid nüüd lühiduse mõttes *tähtedega* `A`, `B`, `C`. Leia mingi kood nende kodeerimiseks.

8. Vaatleme järgmist koodi: `A` → `0`, `B` → `1`, `C` → `01`. Kui me soovime edastada järjest rohkem kui ühte tähte (mõistet), on sellel koodil üks viga. Milline? Arva ära, mida tähendab *üheselt dekodeeritav kood*.

Definitsioon: *Prefikskood* — selline kood, mille puhul mitte ühegi tähe kodeering (*koodisõna*) pole ühtegi teise koodisõna prefiksiks (ehk oleks seda õigem nimetada hoopis *prefiksivabaks* koodiks, aga ajaloolistel põhjustel on levinum eelnev termin).

9. Leia mõni prefikskood hulgale {A,B,C}. Kas prefikskood on üheselt dekodeeritav? Too näide mitte-prefikskoodist hulgale {A,B,C}, mis on ka üheselt dekodeeritav. Miks prefikskood on mugavam? Esita prefikskood puuna. Selgita, miks teda saab alati niimoodi esitada.

Teoreem: Iga üheselt dekodeeritava koodi puhul leidub samade koodisõnade pikkustega prefikskood.

Huffmani kood

10. Vaatleme mingit hulga {A,B,C} koodi. Kas ta on pakkimiseks hea? Kuidas seda headust mõõta?

11. Olgu tähtede esinemise tõenäosused järgmised: A → 50%, B → 25%, C → 25%. Leia *lühim* võimalik prefikskood. Põhjenda, et ta tõesti on optimaalne.

12. Olgu tähtede esinemise tõenäosused järgmised: A → 50%, B → 25%, C → 12,5%, D → 12,5%. Vaatleme "teksti" ADABAABACDABACBA. Asendame selles kõik tähed C ja D uue tähega Z. Milline on tähe Z tõenäosus? Milline on optimaalne kood saadud teksti jaoks? Kui palju "kaotatud" informatsiooni (bittides) "peitub" iga tähe Z taga? Näita, kuidas saaks tähtedest {A,B,Z} koosneva teksti jaoks optimaalse koodi teisendada algse, tähtedest {A,B,C,D} koosneva teksti jaoks optimaalseks koodiks.

Algoritm: Eelnevas läbi tehtud protseduuri saab üldistada: selleks, et konstrueerida optimaalne kood mingisugusele tähtede jaotusele, leia kaks vähima tõenäosusega tähte x ja y, "ühenda" need ajutiselt üheks täheks z, konstrueeri optimaalne kood saadud väiksemale tähestikule ja siis laienda saadud kood algsele tähestikule: lisa z koodisõnale bitt 1, et saada koodisõna x jaoks, ning bitt 0, et saada koodisõna y jaoks. Saadud koodi nimetataksegi *Huffmani koodiks*.

13. Olgu antud tekst YAYBANANABANANA. Arvuta tähtede tõenäosused (sagedused). Konstrueeri tähtede kodeerimiseks Huffmani kood, kodeeri tekst saadud koodiga tähekaupa. Arvuta, mitu bitti kulus ja võrdle seda "tavalise" esitusega (2 bitti tähe kohta). Veendu, et lahtipakkimine on lihtsasti teostatav.

14. Kui me lihtsalt kirjutaks eelnevalt saadud bitijada faili, ei saaks me sellest teksti tagasi. Miks? Kuidas seda lahendada?

Shannon-Fano kood*

15. Mängime järgmist mängu: ma mõtlen mingi tähe peale, ja ütlen, et tõenäosusega 50% on see täht `A`, tõenäosusega 25% on see `B`, tõenäosusega 12,5% on see `C` ja tõenäosusega 12,5% on see `D`. Sina pead selle tähe ära arvama, esitades jah/ei küsimusi stiilis "Kas see on `A`?", "Kas see on `A` või `D`?", "Kas see on `B` või `C` või `D`?" jne. Mõtle välja strateegia, kuidas Sa saaksid minu tähe ära arvata keskmiselt minimaalse küsimuste arvuga. Visualiseeri see puuna. Pane tähele, et antud näite korral on see Huffmani puu.

Algoritm: *Shannon-Fano koodi* konstrueerimiseks jaga tähestik kaheks *võrdtõenäoliseks* osaks, konstrueeri kood kummagi osa jaoks eraldi ja lisa esimese osa koodisõnadele ette bitt 0 ning teise osa koodisõnadele bitt 1.

16. Shannon-Fano kood erineb Huffmani koodist ja pole üldiselt optimaalne. Leia tähestiku {`A`,`B`,`C`,`D`} jaoks jaotus, mille puhul Shannon-Fano kood erineb Huffmani koodist ning pole optimaalne.

17*. Näita, et kui kõikide tähtede tõenäosused on kujul $\frac{1}{2^n}$ (st 50%, 25%, 12,5% jne), siis on Shannon-Fano ja Huffmani koodide koodisõnade pikkused samad. Täpsemalt, tõenäosusega $p = \frac{1}{2^k}$ esineva tähe koodisõna pikkus on täpselt $k = \log_2 \frac{1}{p}$. Arvuta koodisõnade keskmine pikkus selle juhu jaoks.

Definitsioon*: Olgu antud tähestik $\{T_1, T_2, \ldots, T_n\}$ ja selle tõenäosusjaotus $(p_1, p_2, \ldots, p_n)$. Suurust

$$H(p_1, p_2, \ldots, p_n) = p_1 \log_2 \frac{1}{p_1} + p_2 \log_2 \frac{1}{p_2} + \cdots + p_n \log_2 \frac{1}{p_n}$$

nimetatakse jaotuse $(p_1, p_2, \ldots, p_n)$ *entroopiaks*. Entroopia mõõtühikuks on *bitt*.

Teoreem*: Nii Shannon-Fano kui ka Huffmani koodide keskmine koodisõna pikkus ℓ rahuldab võrratusi

$$H(p_1, p_2, \ldots, p_n) \leq \ell < H(p_1, p_2, \ldots, p_n) + 1.$$

18*. Seleta, kuidas võiks entroopia mõistet kasutada selleks, et hinnata, kui palju bitte võtaks üks või teine tekst tähehaaval kodeerituna. Entroopiat kasutataksegi tihti "informatsioonisisalduse" mõõduna. Põhjenda.

Lempel-Ziv-Welchi pakkimisalgoritm

19. Huffmani kood on (keskmise pikkuse mõttes) *optimaalne* viis kodeerida tekste tähthaaval. Kas see tähendab et teksti ei saagi paremini kodeerida? Selgita seda teksti `KUMMIKUTES KUMMITUS KUMMITAS KUMMUTIS` näitel.

20. Leia mingi üldine teksti pakkimise meetod, mis oleks üldiselt sama hea kui Huffmani kood, aga pakiks vähemalt kummikute näidet paremini.

21. Kas Sa eelmises ülesandes kasutasid oma algoritmis *sõnastikku*? Kui mitte, siis mõtle välja algoritm, mis seda kasutaks. Kust ja kuidas Su algoritm sõnastiku ehitab?

Algoritm: *Lempel-Ziv-Welchi algoritm* töötaks teksti `AABABB` pakkimisel järgmiselt:

- Alustame sõnastikuga, milles on ainult tühi sõne: $\{0 \to \text{“”}\}$.
- Leiame teksti pikima prefiksi, mis on olemas meie sõnastikus. See on tühi sõne. Lisame tühjale sõnele teksti järgmise tähe `A` ning lisame saadud sõne sõnastikku. Uus sõnastik on nüüd $\{0 \to \text{“”}, 1 \to \text{“A”}\}$. Väljastame (0,`A`).
- ... (Jätka ise.)

22. Paki eelmises ülesandes saadud kood lahti.

23. Kas peab alustama tühja sõnastikuga?

24. Kui pakkida eelmise algoritmiga liiga pika teksti, võib sõnastik kasvada liiga suureks. Kuidas seda probleemi lahendada?

Teoreem: LZW algoritm on *asümptootiliselt optimaalne*. Piisavalt pikkade tekstide puhul on koodi pikkus keskmiselt lähedal minimaalsele võimalikule.

25. Praktikas kasutatakse LZW ja Huffmani koodi kombinatsiooni. Selgita.

Kõige parem pakkimine?

26. Kus Sa näed pakkimise piire? Kas saaks välja mõelda Kõige Parema Pakkimisalgoritmi? Millised sõned peaksid olema hästi pakitavad, millised mitte?

Definitsioon: Sõne *Kolmogorovi keerukus* on lühima seda sõnet väljastava programmi pikkus.

Teoreem: Etteantud sõne Kolmogorovi keerukus ei ole algoritmiliselt arvutatav.