ZDAJA DRUŠTVO MATEMATIKOV, FIZIKOV IN ASTRONOMOV SLOVENIJE

1993
Letnik 40
2

OBZORNIK ZA MATEMATIKO IN FIZIKO

OBZORNIK ZA MATEMATIKO IN FIZIKO

LJUBLJANA, MAREC 1993

letnik 40, številka 2, strani 33–64

Glasilo Društva matematikov, fizikov in astronomov Slovenije, 61111 Ljubljana, Jadranska c. 19, p.p. 64, telefonska št. (061) 265-061/53, žiro račun 50101-678-47233, devizna nakazila SKB banka d.d. Ljubljana, val-27621-42961/9, Ajdovščina 4, Ljubljana.

Člani društva prejemajo Obzornik brezplačno. Celoletna članarina 1.500,– SIT. Naročnina v knjigarnah in za ustanove 3.000,– SIT, za tujino 40 DEM. Posamezna številka za člane 300,– SIT, dvojna številka 600,– SIT, stare številke 160,– SIT.

Tisk: KURIR print d.o.o. Naklada 1450 izvodov.

Revijo sofinancirata Ministrstvo za znanost in tehnologijo ter Ministrstvo za šolstvo in šport.

Po mnenju MZT št. 415-52/92 z dne 5.2.1992 šteje revija med proizvode iz 13. točke tarifne št. 3 zakona o prometnem davku, za katere se plačuje 5% davek od prometa proizvodov.

 Poštnina plačana na pošti 61102 Ljubljana

VSEBINA — CONTENTS

Članki — Articles Str.–Page

Vesti — News

Na ovitku: karikatura k članku na strani 49

Spoštovana kolegica - kolega,

Sporočamo Vam, da je naše društvo postalo polnopravni član Evropskega matematičnega društva (EMS), Mednarodne matematične zveze (IMU) in Evropskega fizikalnega društva (EPS). Omenjena društva vzpodbujajo številne aktivnosti, s katerimi želijo dvigniti nivo osnovnih in aplikativnih raziskav ter nivo študija naravoslovja na univerzah ter na srednjih in osnovnih šolah. Pospešujejo izmenjavo informacij. EPS si prizadeva tudi olajšati mobilnost fizikov v Evropi. Če ne bomo uspeli dobiti dovolj finančne podpore, bo zaradi članstva v teh društvih članarina DMFAS v prihodnje narasla za nekaj sto SIT.

V zvezi z navedenimi članstvi potrebujemo nekaj dodatnih podatkov o naših članih, zato vam pošiljamo anketni list. Prosimo vas, da ga izpolnite in v nekaj dneh vrnete na naš naslov (DMFAS, Jadranska 19, 61000 Ljubljana).

Fizike opozarjamo na vprašanje ali želijo postati člani EPS. Članstvo v tem združenju je namreč poimensko. Član bo imel vse pravice, ki so jih doslej imeli le individualni člani EPS. Prejemal bo mesečnik Europhysics News. Morda bo za te člane v prihodnje članarina DMFAS večja za nekaj sto SIT.

Prosimo, da kopijo anketnega lista posredujete tudi znancem, ki bi radi postali člani DMFAS.

Ljubljana, 22. 4. 93

Predsednik F. Cvelbar

.....

I Z J A V A (občrtajte ustrezni odgovor)

Ime in priimek ..... rojen(a) .....
kraj ..... poklic .....
akademski naziv .....
Stanujem: kraj: ..... ulica ....., h.št. .....
Poštna številka ..... pošta .....
Telefon: ..... E-mail: .....

Sem, želim postati, član DMFAS v skupini matematika, fizika, astronomija.
Sem: - raziskovalec osnovnih - aplikativnih raziskav
- učitelj na srednji - osnovni šoli
- študent
- upokojenec
- drugo

POSPLOŠENI RIEMANNOV INTEGRAL, 2. del

SERGEJA LIPUŠČEK IN MILAN HLADNIK

Math. Subj. Class. (1985): 28 C 05

V članku predstavimo definicijo in osnovne lastnosti splošnega integrala realne funkcije, kot ga je vpeljal R. Henstock. Ta posplošitev zajema kot posebni primer klasični Riemannov integral, razne vrste izlimitiranih integralov in Lebesguov integral.

GENERALIZED RIEMANN INTEGRAL, PART II

In this paper definition and basic properties of a generalized Riemann integral (for real functions) introduced by R. Henstock are presented. The generalization includes classical Riemann integral, various kinds of improper integrals and Lebesgue integral.

3. Izlimitirani integrali

Kakšna je zveza posplošenega Riemannovega integrala z izlimitiranim? Preden odgovorimo na to vprašanje, se moramo vprašati nekaj drugega. Posplošeni Riemannov integral smo definirali tako na omejenih kot na neomejenih intervalih. V slednjem primeru neomejeni delilni podinterval na Riemannovo vsoto pravzaprav ni vplival, ker smo „dolžino" takega podintervala izenačili z 0. Po drugi strani je na intervalu integrabilna funkcija integrabilna tudi na vsakem podintervalu. Torej je posplošeni Riemannov integral določen že z integrali na podintervalih. Vprašanje je, ali je integral na neomejenem intervalu, če seveda obstaja, enak limiti integralov na omejenih podintervalih. Odgovor daje naslednji izrek.

Izrek 3. *Realna funkcija* $f : [a,b] \to \mathbb{R}$ *je na zaprtem intervalu* $[a,b] \subset \mathbb{R}^*$ *integrabilna natanko takrat, ko je integrabilna na vsakem zaprtem podintervalu* $[a,s]$, $s \in (a,b)$, *in obstaja limita* $\lim_{s\to b} \int_a^s f \in \mathbb{R}$. *Če je integrabilna, velja* $\int_a^b f = \lim_{s\to b} \int_a^s f$.

Dokaz. Naj bo funkcija f integrabilna na $[a,b]$ in $\varepsilon > 0$. Potem obstaja družina okolic τ na intervalu $[a,b]$, da je $|\int_a^b f - S(f,D)| < \varepsilon$ za poljubno τ-delitev D intervala $[a,b]$. Za vsak $s \in \tau(b)$ pa obstaja družina okolic τ_s na intervalu $[a,s]$ in taka τ_s-delitev D_s intervala $[a,s]$, da je $|\int_a^s f - S(f,D_s)| < \varepsilon$. Pri tem lahko vzamemo, da je $\tau_s(z) \subset \tau(z)$ za vsak $z \in [a,s]$. Delitvi D_s intervala $[a,s]$ dodajmo par $b[s,b]$ in dobimo delitev D intervala $[a,b]$, ki je usklajena tudi s τ zaradi inkluzije $\tau_s(z) \subset \tau(z)$. Potem je

$$|\int_a^b f - \int_a^s f| \le |\int_a^b f - S(f,D)| + |f(b)|L([s,b]) + |S(f,E) - \int_a^s f| < 3\varepsilon\,,$$

če le vzamemo $s \in \tau(b)$ dovolj blizu b. To pomeni, da limita $\lim_{s\to b} \int_a^s f$ obstaja in je enaka $\int_a^b f$.

Obratno, naj bo funkcija f integrabilna na vsakem podintervalu $[a,s]$, limita $\lim_{s\to b} \int_a^s f$ pa naj obstaja in naj bo enaka A. Izberimo $\varepsilon > 0$ in

poljubno naraščajoče zaporedje $(c_n)_{n>0}$ z lastnostjo $c_0 = c_1 = a$, $c_n < c_{n+1}$ za $n \leq 1$ in $\lim_{n\to\infty} c_n = b$. Ker je funkcija f integrabilna tudi na poljubnem zaprtem podintervalu, obstaja za vsak $n = 1, 2, 3, \ldots$ na intervalu $[c_{n-1}, c_{n+1}]$ družina okolic τ_n, da velja $|\int_{c_{n-1}}^{c_{n+1}} f - S(f, D)| < \varepsilon/2^n$ za poljubno τ_n-delitev D intervala $[c_{n-1}, c_{n+1}]$. Definirajmo novo družino okolic τ na intervalu $[a, b)$, tako da bo veljalo $\tau(z) \subset \tau_1(z) \cap [-\infty, c_1)$ in $\tau(z) \subset \tau_n(z) \cap (c_{n-1}, c_{n+1})$ za $z \in [c_{n-1}, c_n)$ in $n = 2, 3, \ldots$.

Naj bo E katerakoli τ-delitev podintervala $[a, s]$, kjer je $s \in (a, b)$ poljubno število. Pokazali bomo, da je $|\int_a^s f - S(f, E)| \leq \varepsilon$. Za vsak $n = 1, 2, 3, \ldots$ naj množica E_n vsebuje tiste pare zJ delitve E, za katere je $z \in [c_{n-1}, c_n)$. Podintervali v množici E_n sestavljajo interval, ki ga označimo z J_n. Množice E_n so med seboj disjunktne in samo končno mnogo je nepraznih. Poleg tega za vsak $n \geq 1$ po konstrukciji velja $J_n \subset [c_{n-1}, c_n)$, množica E_n pa je τ_n-delitev intervala J_n. Zaradi opombe na koncu prejšnjega razdelka tudi za podintervale $J_n \subset [c_{n-1}, c_{n+1}]$ in τ_n-delitve E_n velja $|\int_{J_n} f - S(f, E_n)| \leq \varepsilon/2^n$. Upoštevajmo še $S(f, E) = \sum_{n=1}^{\infty} S(f, E_n)$, pa dobimo

$$|\int_a^s f - S(f, E)| \leq \sum_{n=1}^{\infty} |\int_{J_n} f - S(f, E_n)| \leq \sum_{n=1}^{\infty} \varepsilon/2^n = \varepsilon.$$

Nazadnje definirajmo še $\tau(b)$, in sicer tako, da bo veljalo $|f(b)|L([s, b]) < \varepsilon$ in $|A - \int_a^s f| < \varepsilon$ za vsak $s \in \tau(b)$. Potem lahko za poljubno τ-delitev intervala $[a, b]$ dobimo zahtevano oceno

$$|A - S(f, D)| \leq |A - \int_a^s f| + |\int_a^s f - S(f, E)| + |f(b)|L([s, b]) < \varepsilon.$$

Podoben izrek seveda velja tudi za primer, ko limitiramo s proti levemu krajišču a. Dokazani izrek 3 med drugim pove, da je (v nasprotju s klasičnim primerom) nesmiselno definirati izlimitirani posplošeni Riemannov integral. Limitni proces nam ne dá nič novega.

Poleg tega lahko sedaj odgovorimo na vprašanje, zastavljeno takoj na začetku razdelka.

Izrek 4. *Vsaka funkcija, za katero obstaja izlimitirani Riemannov integral, je integrabilna in njen posplošeni Riemannov integral je enak izlimitiranemu.*

Dokaz. Naj za funkcijo f obstaja izlimitirani Riemannov integral na končnem ali neskončnem intervalu $[a, b)$. Tedaj je po definiciji f Riemannovo integrabilna na vsakem (omejenem) podintervalu $[a, s)$, po izreku 2 torej tudi (posplošeno) integrabilna. Poleg tega obstaja tudi limita $\lim_{s\to b} \int_a^s f = \lim_{s\to b} \int_a^s f(x)dx$, zato funkcija f, ki jo v točki b definiramo kakorkoli, zadošča pogojem izreka 3 in je torej integrabilna v posplošenem smislu tudi na intervalu $[a, b]$ z integralom, enakim dani limiti.

Zgled 2. Izlimitirani Riemannov integral $\int_1^\infty x^{-2}dx$ je enak 1, zato po izreku 4 obstaja tudi posplošeni Riemannov integral $\int_1^\infty x^{-2} = 1$. Izlimitirani Riemannov integral $\int_0^1 x^{-2}dx$ pa ne obstaja, saj je $\lim_{s\to 0}\int_s^1 x^{-2}dx = \lim_{s\to 0}(-1+s^{-1}) = \infty$. Ker je funkcija $f(x) = x^{-2}$ na intervalu $(0,1]$ zvezna, je Riemannovo in zato tudi (posplošeno) integrabilna na vsakem podintervalu $[s,1]$, $s > 0$. Po različici izreka 3 potem ni integrabilna na $[0,1]$, saj ustrezna limita ne obstaja.

4. Osnovni izrek infinitezimalnega računa in konvergenčni izreki

Riemannov integral zvezne funkcije f največkrat izračunamo tako, da najprej poiščemo njeno *primitivno funkcijo*, se pravi odvedljivo funkcijo F z lastnostjo $F' = f$. Potem je $\int_a^b f(x)dx = F(b) - F(a)$.

Kadar je izpolnjena enakost

$$\int_a^b F'(x)dx = F(b) - F(a),$$

pravimo, da za funkcijo F velja *osnovni izrek infinitezimalnega računa*. Na kratko rečemo, da je funkcija F integral svojega odvoda. V Lebesguovi teoriji je dovolj, da odvod F' obstaja skoraj povsod (glede na Lebesguovo mero) in je Lebesguovo integrabilen (to je očitno tudi potrebno). Takim funkcijam F rečemo absolutno zvezne funkcije. Vendar niti v Riemannovi niti v Lebesguovi teoriji ne velja osnovni izrek za *vse* odvedljive funkcije F na intervalu $[a,b]$. Lahko se npr. prepričamo, da je funkcija $F(x) = x^2\cos(1/x^2)$ za $0 < x \le 1$ in $F(0) = 0$ povsod na intervalu $[0,1]$ odvedljiva, njen odvod $F'(x) = 2x\cos(1/x^2) + \frac{2}{x}\sin(1/x^2)$, $F'(0) = 0$, pa ni niti Riemannovo niti Lebesguovo integrabilen na $[0,1]$. Prvi sumand odvoda je namreč na intervalu $(0,1]$ zvezna in omejena funkcija in ne dela težav, za drugega pa velja

$$\int_0^1 \frac{2}{x}\sin\frac{1}{x^2}dx = \int_1^\infty \frac{\sin t}{t}dt.$$

Kot vemo, slednji integral ni absolutno konvergenten.

Kot bomo videli, teh težav pri posplošenem Riemannovem integralu ni. Zanj velja osnovni izrek infinitezimalnega računa: vsaka povsod odvedljiva funkcija je integral svojega odvoda.

Za dokaz osnovnega izreka potrebujemo naslednjo preprosto lemo.

Lema. *Naj bo funkcija* $F : [a,b] \to \mathbb{R}$ *odvedljiva v točki* $z \in [a,b]$. *Potem za vsak* $\varepsilon > 0$ *obstaja tak* $\delta(z) > 0$, *da velja*

$$|F(v) - F(u) - F'(z)(v-u)| \le \varepsilon(v-u)\,,$$

kakor hitro je $u \le z \le v$ *in* $[u,v] \subset [a,b] \cap (z-\delta(z), z+\delta(z))$.

Dokaz. Ker je funkcija F odvedljiva v točki z, za vsak $\varepsilon > 0$ obstaja $\delta(z) > 0$, tako da je $|(F(x)-F(z))/(x-z)-F'(z)| < \varepsilon$ za $0 < |x-z| < \delta(z)$ in $x \in [a,b]$. Naj bo $u < z < v$ (za $z = u$ ali $z = v$ lema takoj sledi). Tedaj imamo

$$|F(v) - F(u) - F'(z)(v-u)| \leq$$
$$\leq |F(v) - F(z) - F'(z)(v-z)| + |F(z) - F(u) - F'(z)(z-u)| <$$
$$< \varepsilon(v-z) + \varepsilon(z-u) = \varepsilon(v-u)\,.$$

Izrek 5 (Osnovni izrek infinitezimalnega računa). *Če je funkcija* $F : [a,b] \to \mathbb{R}$ *odvedljiva na intervalu* $[a,b]$, *je* F' *integrabilna funkcija na* $[a,b]$ *po definiciji 1 in velja* $\int_a^b F' = F(b) - F(a)$.

Dokaz. Vzemimo $\varepsilon > 0$ in za $z \in [a,b]$ definirajmo $\tau(z) = (z - \delta(z), z + \delta(z))$, kjer je število $\delta(z)$ tako kot v prejšnji lemi. Tedaj za poljubno τ-delitev $D = \{z_1[x_0,x_1], \ldots, z_n[x_{n-1},x_n]\}$ po lemi velja

$$|S(F',D) - (F(b) - F(a))| = |\sum_{i=1}^{n}[F'(z_i)(x_i - x_{i-1}) - (F(x_i) - F(x_{i-1}))| <$$
$$< \sum_{i=1}^{n} \varepsilon(x_i - x_{i-1}) = \varepsilon(b-a)\,.$$

Ta izrek se da celo malo posplošiti na primer, ko funkcija F ni odvedljiva povsod.

Izrek 6. *Naj bo funkcija* $F : [a,b] \to \mathbb{R}$ *odvedljiva povsod razen kvečjemu v številno mnogo točkah intervala* $[a,b]$ *in naj bo* $f : [a,b] \to \mathbb{R}$ *taka funkcija, da je* $F'(x) = f(x)$, *če je* F *odvedljiva v točki* x. *Če je funkcija* F *zvezna, je* f *integrabilna na intervalu* $[a,b]$ *in velja* $\int_a^b f = F(b) - F(a)$.

Dokaz je podoben kot pri izreku 5, le malo bolj tehnično zapleten, saj je treba upoštevati izjemno množico točk, v katerih funkcija F ni odvedljiva (glej npr. [6], Theorem 5). Naj omenimo, da je zahteva o zveznosti funkcije F potrebna, sicer že preprosti primeri pokažejo, da osnovni izrek infinitezimalnega računa ne more držati (glej npr. funkcijo $F(x)$, ki je za $-1 \leq x < 0$ enaka $-x$, za $0 \leq x \leq 1$ pa $x+1$).

Preprosta posledica izreka 5 je naslednji

Izrek 7 (o integraciji po delih). *Naj bosta* F *in* G *na intervalu* $[a,b]$ *odvedljivi funkciji z odvodoma* $F' = f$ *in* $G' = g$. *Tedaj je funkcija* fG *integrabilna na* $[a,b]$ *natanko takrat kot funkcija* Fg *in velja*

$$\int_a^b fG = F(b)G(b) - F(a)G(a) - \int_a^b Fg.$$

Tudi tu bi lahko glede odvedljivosti dopustili štev no mnogo izjem.

Klasični Riemannov integral je pri uporabi precej neroden, kadar je potrebno opraviti limitni proces pod integralskim znakom oziroma zamenjati vrstni red integriranja in limitiranja, saj zanj velja le izrek o enakomerni konvergenci, ne pa npr. tudi izreka o monotoni in dominantni konvergenci. Pri tem ga, kot je znano, močno prekaša Lebesguov integral (glej npr. [5]). Zanimivo pa je, da tudi posplošeni Riemannov integral v tem pogledu nič ne zaostaja za Lebesguovim. V praktično isti obliki veljajo vsi trije konvergenčni izreki. Njihovo izpeljevanje bi zahtevalo nekoliko več priprave, zato jih samo navedimo (dokaz glej v [1] ali [2]).

Izrek 8 (o enakomerni konvergenci). *Zaporedje integrabilnih funkcij f_n naj na zaprtem intervalu $I \subset \mathbb{R}^*$ enakomerno konvergira k funkciji f. Tedaj je tudi funkcija f integrabilna in velja $\int_I f = \lim_{n\to\infty} \int_I f_n$.*

Izrek 9 (o monotoni konvergenci). *Zaporedje realnih integrabilnih funkcij f_n naj na zaprtem intervalu $I \subset \mathbb{R}^*$ monotono naraščajoče (ali monotono padajoče) konvergira k funkciji f. Tedaj je f integrabilna funkcija na I natanko takrat, ko obstaja končna limita $\lim_{n\to\infty} \int_I f_n$. V tem primeru je $\int_I f = \lim_{n\to\infty} \int_I f_n$.*

Izrek 10 (o dominantni konvergenci). *Zaporedje integrabilnih funkcij f_n naj na zaprtem intervalu $I \subset \mathbb{R}^*$ konvergira k funkciji f in naj bo g taka integrabilna funkcija na intervalu I, da velja $|f_n(x)| \leq g(x)$ za vsak $x \in I$. Tedaj je tudi funkcija f integrabilna na intervalu I in velja $\int_I f = \lim_{n\to\infty} \int_I f_n$.*

5. Lebesguov integral

Na kratko in zgolj informativno si oglejmo še zvezo posplošenega Riemannovega integrala z Lebesguovim. V ta namen potrebujemo nekaj pojmov, ki so značilni za Lebesguovo teorijo.

Rekli bomo, da je množica $C \subset \mathbb{R}$ *integrabilna*, če je karakteristična funkcija χ_C integrabilna na $\mathbb{R}$ (oziroma na $\mathbb{R}^*$, če jo kakorkoli razširimo tudi v neskončnost). Integrabilen je npr. vsak kompakten interval in sploh vsaka kompaktna podmnožica v $\mathbb{R}$. Množica $C \subset \mathbb{R}$ pa je *merljiva*, če je množica $C \cap J$ integrabilna za vsak kompakten interval $J \subset \mathbb{R}$. Vsaka integrabilna množica je merljiva, ker je presek integrabilnih množic spet integrabilna množica. Vse merljive podmnožice na realni osi sestavljajo σ-algebro množic. Na njej definiramo s predpisom $\mu(C) = \int_{\mathbf{R}} \chi_C$ *pozitivno mero.* Brez dokaza (glej [2]) navedimo pomemben izrek, ki te pojme povezuje z Lebesguovo teorijo.

Izrek 11. *Množica C je merljiva natanko takrat, ko je Lebesguovo merljiva. Mera μ se ujema z Lebesguovo mero.*

Merljive funkcije definiramo na običajen način: Funkcija f je *merljiva*, če je $f^{-1}(G)$ merljiva množica za vsako odprto množico $G \subset \mathbb{R}$. Vsaka (posplošeno) integrabilna funkcija ni nujno merljiva, pač pa je to res za poseben razred t.i. absolutno integrabilnih funkcij. Pravimo, da je funkcija f *absolutno integrabilna*, če sta tako funkcija f kot njena absolutna vrednost $|f|$ integrabilni po definiciji 1. Absolutno integrabilne funkcije so prava podmnožica integrabilnih funkcij (primer integrabilne funkcije, ki ni absolutno integrabilna, je funkcija $\sin x/x$ na intervalu $(0, \infty)$). Najbolj zanimivo pa je, da se pojem absolutne integrabilnosti natanko ujema s pojmom Lebesguove integrabilnosti.

Izrek 12. *Posplošeni Riemannov integral se na množici absolutno integrabilnih funkcij ujema z Lebesguovim integralom.*

Dokaz zahteva nekaj dela in se mu v podrobnostih odpovejmo (glej [2]). Prehoditi je pač treba pot, znano iz Lebesguove teorije. Preveriti je treba enakost obeh integralov na karakterističnih funkcijah intervalov, nato na karakterističnih funkcijah odprtih množic (ki so limite naraščajočega zaporedja karakterističnih funkcij končnih unij odprtih intervalov), nato na karakterističnih funkcijah merljivih množic (ki so limite padajočega zaporedja karakterističnih funkcij odrtih množic), na enostavnih funkcijah (ki so linearna kombinacija karakterističnih) in na pozitivnih Lebesguovo integrabilnih funkcijah (ki so limite naraščajočega zaporedja enostavnih funkcij). Večkrat je teba uporabiti izreka o monotoni in o dominantni konvergenci, ki veljata za obe vrsti integralov.

Kot vidimo, je posplošeni Riemannov integral še močnejši od Lebesguovega, saj slednjega vključuje.

Podobno pot kot v tem sestavku je pri uvedbi integrala ubral E. J. McShane, o čemer je pisal pred leti tudi Obzornik [3]. Razlika je samo ta, da McShane dopušča evaluacijsko točko pri obeleženi delitvi tudi zunaj delilnega podintervala, kar je za delitve manj, za integrabilnost pa bolj stroga zahteva kot pri posplošenem integralu, ko mora biti evaluacijska točka znotraj intervala. Končni rezultat je ta, da je integrabilnih funkcij manj in da so, kot se presenetljivo izkaže, to ravno Lebesguovo integrabilne funkcije. McShaneov integral se ujema z Lebesguovim (glej [3]).

LITERATURA

[1] S. Lipušček, *Posplošeni Riemannov integral* (diplomsko delo), Ljubljana 1991.

[2] R. M. McLeod, *The generalized Riemann integral*, Carus Math. Monographs **20**, MAA 1980.

[3] N. Prijatelj, *Nova pot do Lebesgua*, Obzornik mat. fiz. **34** (1987) 23–30.

[4] W. Rudin, *Principles of mathematical analysis*, McGraw-Hill, New York 1964.

[5] W. Rudin, *Real and Complex Analysis*, McGraw-Hill, London 1970.

[6] Ch. Swartz, B S. Thomson, *More on the fundamental theorem of calculus*, Amer. Math. Monthly **95** (1988) 644-648.

[7] I. Vidav, *Višja matematika I*, DZS, Ljubljana 1978.

VERJETNOSTNI ALGORITEM OHLAJANJE

JANEZ ŽEROVNIK

Math. Subj. Class. (1991) 68 R 05

Kot primer verjetnostnega algoritma podrobneje obravnavamo algoritem ohlajanje (angl. Simulated Annealing, oznaka $\mathcal{SA}$), ki ga lahko definiramo kot algoritem za reševanje splošnega problema kombinatorične optimizacije. Pokažemo presenetljivi rezultat, da verjetnost uspeha algoritma $\mathcal{SA}$ raste počasneje kakor verjetnost uspeha algoritma lokalna optimizacija.

THE PROBABILISTIC ALGORITHM SIMULATED ANNEALING

An example of a randomized algorithm, the Simulated Annealing algorithm ($\mathcal{SA}$), is considered as an algorithm for solving the general problem of combinatorial optimization. It is shown that the probability of success of the algorithm $\mathcal{SA}$ grows slower than the same probability for the local optimization.

1. Uvod

Med verjetnostnimi algoritmi je bil v zadnjem času deležen precej pozornosti algoritem ohlajanje ($\mathcal{SA}$). V pregledu [10] med literaturo navajamo prek 30 referenc o tem algoritmu, bibliografija pa s tem še zdaleč ni izčrpana, saj se članki še vedno pojavljajo. (Za ilustracijo omenimo, da v oglasu za eno od knjig omenjajo preko 300 člankov o $\mathcal{SA}$!) O popularnosti priča tudi veliko število imen, ki jih navajajo v uvodu [6]: Monte Carlo annealing, statistical cooling, probabilistic hill climbing, stochastic relaxation ali probabilistic exchange algorithm. Leta 1953 je Metropolis s sodelavci uporabil računalnik za simulacijo fizikalnega modela ohlajanja snovi. Kasneje so Kirckpatrick s sodelavci [4] in neodvisno Černy (Czerny) opazili analogijo med reševanjem problema kombinatorične optimizacije in Metropolisovo simulacijo fizikalnega modela. Verjetno je prav zaradi duhovite analogije algoritem hitro postal široko znan in ‚moderen'. Večina avtorjev kot vir za $\mathcal{SA}$ citira članek [4], zaradi pravičnosti pa velja omeniti, da so analogijo med statistično mehaniko in optimizacijo (zveznih) funkcij opazili že prej Khačaturjan, Semenovskaja, Vainstein [5] in Pincus [8], vendar sta članka ostala brez širšega odmeva. V nadaljevanju bomo povzeli izrek, da algoritem ‚konvergira'. Točneje: pri določenih pogojih (dovolj počasno ohlajanje) za pripadajočo (časovno) nehomogeno markovsko verigo obstaja limitna porazdelitev, v kateri imajo pozitivno verjetnost natanko globalno optimalna stanja (optimalne rešitve). V praksi je ohlajanje seveda vedno ‚prehitro'. V velikem številu poročil o poskusih (glej pregled v [10]) poročajo o dobrih rezultatih. To sicer včasih pomeni, da so dobljene rešitve dotlej najboljše znane za referenčne primerke nekaterih problemov, vendar gre to vedno na račun zelo velikega računskega časa. V literaturi najdemo celo vrsto teoretičnih analiz algoritma. Tukaj ob knjigi [6] in članku [7],

---

Članek je nastal na osnovi avtorjeve disertacije pod mentorstvom Tomaža Pisanskega. Marko Petkovšek in Milan Hladnik sta skrbno pregledala rokopis in opozorila na mnoge napake in pomanjkljivosti. Vsem se najlepše zahvaljujem.

po katerem povzemamo izreke o konvergenci algoritma, omenimo še en rezultat. Sasaki in Hajek sta pokazala, da je za problem maksimalnega prirejanja (angl. maximum matching) zahtevnost celega razreda algoritmov tipa $\mathcal{SA}$ gotovo večja kot polinomska [9].

Algoritma $\mathcal{SA}$ in $\mathcal{RLS}$ sta definirana dovolj splošno, da ju lahko uporabimo za reševanje kateregakoli problema kombinatorične optimizacije, zato ju je smiselno primerjati. Glavna zamera algoritmu $\mathcal{RLS}$ (oziroma notranji zanki algoritma) je to, da se ‚ujame' v lokalnih optimumih. Ko začnemo z novo začetno rešitvijo, v nekem smislu zavržemo vso informacijo, ki smo jo prigarali z dotedanjim računanjem (razen seveda dotlej najboljše dobljene rešitve). Če dovolimo tudi korake ‚navzgor' (v smeri povečanja stroškovne funkcije), se algoritem ne ustavi več v lokalnih optimumih. Tako dobimo tako imenovane ‚plezalne' (angl. hill climbing) algoritme, med katere uvrščamo tudi algoritem $\mathcal{SA}$. Na račun možnosti plezanja navzgor pa se sicer navidezno še vedno zelo enostavni algoritem precej zaplete, saj je potrebno poiskati prave vrednosti parametrov, ki kontrolirajo, kdaj in koliko dovolimo korake ‚navzgor'. Pri algoritmu $\mathcal{SA}$ parametru običajno rečemo temperatura. Problem, kako temperaturo zniževati, da bomo dobili dobre rezultate, ni enostaven in vse kaže, da je odgovor precej odvisen tudi od tipa problema, ki ga želimo reševati.

Primerjava algoritmov $\mathcal{SA}$ in $\mathcal{RLS}$ je po [6] odprt problem. Iz pregleda poročil o eksperimentih z obema algoritmoma vidimo, da je v nekaterih primerih boljši prvi, v drugih pa drugi algoritem [10]. Tu bomo pokazali (izrek 1), da je algoritem $\mathcal{SA}$ asimptotsko slabši od lokalne optimizacije in tako deloma odgovorili na gornje vprašanje.

2. Splošni problem kombinatorične optimizacije

Mnoge diskretne optimizacijske probleme lahko gledamo kot posebne primere *splošnega problema kombinatorične optimizacije*. Spomnimo se definicije iz [11]:

Definicija 1. *Splošni problem kombinatorične optimizacije je družina primerkov. Primerek je par* $(\mathbf{S}, c)$, *kjer je* $\mathbf{S}$ *množica (stanj, dopustnih rešitev),* c *pa je preslikava* $c : \mathbf{S} \to \mathbb{C}$ *(stroškovna funkcija). Treba je poiskati tak* $x_{min} \in \mathbf{S}$, *da velja*

$$c_{min} = c(x_{min}) = \min_{x \in \mathbf{S}} c(x)\,.$$

Problem iskanja maksimuma definiramo analogno, je pa trivialno ekvivalenten problemu minimuma na isti množici $\mathbf{S}$ s stroškovno funkcijo $\tilde{c} = -c$. Tu se bomo omejili na problem iskanja minimuma.

Pri obravnavi splošnega problema kombinatorične optimizacije $(\mathbf{S}, c)$ običajno privzamemo, da sta dana verjetnostna algoritma $\mathcal{R}$ $(\mathcal{R} : \emptyset \to \mathbf{S})$ in $\mathcal{N}$ $(\mathcal{N} : \mathbf{S} \to \mathbf{S})$. $\mathcal{R}$ slučajno generira začetno rešitev, $\mathcal{N}$ pa dani dopustni

rešitvi (hitro) najde eno od „sosednjih" dopustnih rešitev. Množico $\mathrm{N}(x) = \{y|P(\mathcal{N}(x) = y) > 0\}$ imenujemo *soseščina* točke $x \in \mathbf{S}$.

Lokalna optimizacija. Za reševanje splošnega problema kombinatorične optimizacije je najbolj znan naslednji algoritem, ki ga bomo imenovali algoritem *lokalna optimizacija* (angl. randomised local search $\mathcal{RLS}$, neighborhood search, iterative improvement, včasih tudi multistart algorithm):

Algoritem lokalna optimizacija ($\mathcal{RLS}$)
ponavljaj
  generiraj slučajno začetno stanje $x := \mathcal{R}$
  1:ponavljaj
    poišči naslednjega soseda x'
    če $c(x') < c(x)$ **potem** $x :='$ (in skoči na 1)
  dokler ni množica sosedov izčrpana
dokler ni preveč korakov

Enostaven premislek nas pripelje do ugotovitve, da se notranja zanka ustavi v lokalnih minimumih. Za kasnejšo rabo preštejmo, koliko največ poskusov prehoda iz stanja v stanje lahko algoritem $\mathcal{RLS}$ naredi, preden se ujame v lokalnem minimumu. Uporabili bomo oceno, da je čas za računanje enega poskusa prehoda iz stanja v stanje približno enak času, ki ga porabimo za en klic algoritma $\mathcal{N}$. Privzemimo, da je dan primerek I problema P. Potem je d, zgornja meja za število sosedov dopustne rešitve, določena z $d = \max_{x \in \mathbf{S}} |\mathrm{N}(x)|$. Označimo z R dolžino najdaljše poti, ki gre samo navzdol (v smeri zmanjšanja stroškovne funkcije). Dogovorimo se, da bomo kasneje, pri primerjavi z algoritmom $\mathcal{SA}$, za korak algoritma vzeli en (morebitni) prehod k novi rešitvi. Z drugimi besedami, za osnovno enoto za merjenje časa, ki ga algoritem porabi, bomo šteli en klic algoritma $\mathcal{N}$. Označimo z w razmerje med časom, potrebnim za generiranje začetne dopustne rešitve (algoritem $\mathcal{R}$), in med časom, potrebnim za generiranje novega stanja, če kakšno stanje že poznamo (algoritem $\mathcal{N}$). Ker lahko algoritem $\mathcal{RLS}$ napreduje le navzdol in ker v vsakem stanju lahko pregleda največ d sosedov, je največje število korakov, ki jih lahko algoritem $\mathcal{RLS}$ naredi, preden se ujame v lokalnem minimumu, omejeno z $Rd + w$.

Algoritem $\mathcal{SA}$. V nasprotju z lokalno optimizacijo so pri algoritmu *ohlajanje* možni tudi premiki v smeri poslabšanja stroškovne funkcije. Verjetnost takega koraka je odvisna od pozitivnega parametra T, ki mu po analogiji s fizikalnim modelom rečemo temperatura.

Algoritem ohlajanje ($\mathcal{SA}$):
generiraj slučajno začetno stanje $x := \mathcal{R}$
nastavi začetno temperaturo T
ponavljaj
  znižaj temperaturo T
  slučajno izberi soseda $x' :=\mathrm{N}\ (x)$
  če $c(x') < c(x)$ **potem** $x' := x$
  sicer $x' := x$ z verjetnostjo $p = p(T)$
dokler ni preveč korakov

Označimo s p verjetnost dogodka, da je v algoritmu sprejet korak, ki poslabša vrednost stroškovne funkcije. Ta verjetnost je odvisna od trenutnega stanja (trenutne dopustne rešitve) in od trenutne vrednosti parametra T. Predpostavili bomo, da je p naraščajoča funkcija parametra T, se pravi, da p pada z zniževanjem temperature T. Pri danem primerku problema in dani temperaturi T bomo zahtevali, da naj bo p omejena s konstanto $p < P < 1$ (P je torej neodvisna od trenutnega stanja).

Funkcijska odvisnost je običajno oblike

$$p = \begin{cases} 1, & \Delta C < 0 \\ \exp(-\Delta C/T), & \Delta C \geq 0 \end{cases} \tag{1}$$

kjer smo z ΔC označili razliko stroškovnih funkcij novega in starega stanja. Edina avtorju znana izjema je definicija Boltzmanovega stroja [1], kjer uporabljajo $p = (1 + \exp(\Delta C/T))^{-1}$.

Definicija algoritma $\mathcal{SA}$ je res enostavna, vendar pri implementaciji hitro naletimo na netrivialne probleme. Ni jasno na primer, kako je treba zmanjševati temperaturo, da bomo dosegli dobre rezultate. Vemo, da prehitro zniževanje temperature ne zagotavlja več ‚konvergence' algoritma [7]. Ali je mogoče zniževanje temperature dobro določiti vsaj za kakšen poseben problem? V [6] na primer predlagajo polinomske strategije ohlajanja, ki se dobro odrežejo na nekaterih preizkušenih primerih. (Pri tem se seveda morajo zadovoljiti z ‚dobrimi' namesto z optimalnimi rešitvami.) Tu se s tem problemom ne bomo podrobneje ukvarjali.

Zniževanje temperature lahko definiramo tako, da povemo, kako je odvisna od števila korakov. Poseben (ekvivalenten) primer je, da povemo, koliko korakov naredimo pri kaki temperaturi. Odvisnost temperature od časa v tem primeru podamo z zaporedjem parov (temperatura, število korakov), na primer

$$(T_0, m_0), (T_1, m_1), \ldots, (T_k, m_k), \ldots,$$

kar pomeni: naredi m_0 korakov pri temperaturi T_0,..., naredi m_k korakov pri temperaturi T_k,... itd. Kot že omenjeno, mora temperatura padati počasi, če želimo, da algoritem z verjetnostjo 1 najde optimalno rešitev (v končnem, toda ne omejenem številu korakov) [7].

V nadaljevanju bomo predpostavili, da temperatura T konvergira proti 0 z naraščajočim številom korakov: $\lim_{n\to\infty} T_n = 0$.

3. Konvergenca algoritma $\mathcal{SA}$

Med prvimi, ki so obravnavali konvergenco algoritma $\mathcal{SA}$, so bili Geman, Geman in Gidas, Hajek in Sasaki, Gelfand in Mitter, Haario in Saksman, Tsitsiklis in drugi. V tem razdelku bomo povzeli po članku [7] izrek o konvergenci algoritma $\mathcal{SA}$. Model izvajanja algoritma je (časovno) nehomogena markovska veriga, prehodne verjetnosti se namreč s časom spreminjajo. Izreke za nehomogene markovske verige, ki bi jih potrebovali za dokaz, dobimo na primer v knjigi [3].

Najprej navedimo nekaj definicij in potrebne predpostavke. Najenostavneje bi bilo privzeti, da je generiranje kateregakoli soseda danega stanja enako verjetno. Ker pa je v nekaterih uporabah pomembno, da se nekateri sosedi generirajo bolj verjetno kakor drugi, tu predpostavimo, da je verjetnost generiranja soseda j iz stanja i dana z

$$\frac{g(i,j)}{g(i)},$$

kjer so $g(i,j) > 0$ uteži in je $g(i) > 0$ normalizacijski faktor, določen z

$$\frac{1}{g(i)} \sum_{j \in N(i)} g(i,j) = 1,$$

kjer je $N(i)$ soseščina točke i. Privzeli bomo samo, da velja

$$g(i,j) = g(j,i) \qquad \text{za vsak} \quad i,j \in \mathbf{S}. \tag{2}$$

Poleg tega je smiselno predpostaviti, da je graf, ki ga dobimo iz množice $\mathbf{S}$ tako, da povežemo sosednja stanja, (krepko) povezan.

Verjetnosti prehodov algoritmu $\mathcal{SA}$ pripadajoče markovske verige so dane z

$$P_{ij}(T) = \begin{cases} 0, & \text{če } j \notin N(i) \text{ in } j \neq i \\ \dfrac{g(i,j)}{g(i)} \min\{1, \exp(-(c(j)-c(i))/T)\}, & \text{če } j \in N(i) \end{cases} \tag{3}$$

in

$$P_{ii} = 1 - \sum_{j \in N(i)} P_{ij}(T)\,. \tag{4}$$

Markovska veriga je nehomogena, saj se prehodne verjetnosti lahko s časom spreminjajo, če se spreminja vrednost parametra T.

Če parameter T fiksiramo, dobimo homogeno verigo. Stacionarne porazdelitve homogenih verig pri različnih vrednostih temperature T imenujemo (po [7]) *kvazi-stacionarne porazdelitve* originalne nehomogene verige.

Za $i \in S$ definirajmo

$$\pi_i(T) = \frac{g(i)\exp(-c(i)/T)}{G(T)}, \tag{5}$$

kjer je $G(T)$ faktor, s katerim normiramo vektor π tako, da velja

$$||\pi(T)|| = \sum_{i=1}^{s} \pi_i(T) = 1.$$

Tu je s moč množice S, $s = |S|$, in $\pi(T) = (\pi_1(T), \pi_2(T), \ldots \pi_s(T))$.

Prva trditev pravi, da je (5) stacionarna porazdelitev homogene markovske verige.

Trditev 1. (Mitra et al., 1986). *Če velja (2), potem za števila* $\pi_i(T)$, *definirana s (5), velja*

$$\pi(T)P(T) = \pi(T), \tag{6}$$

kjer je $P(T)$ *matrika prehodnih verjetnosti, definirana s (3) in (4).*

Ker je dokaz kratek, ga povzemimo.

Dokaz. Zaradi (5), (3) in (2) velja za poljubna soseda i in j iz S, pa tudi za $i = j$

$$\frac{\pi_i(T)}{\pi_j(T)} = \frac{g(i)}{g(j)} \exp((c(j) - c(i))/T) = \frac{P_{ji}(T)}{P_{ij}(T)},$$

ne glede na predznak izraza $c(j) - c(i)$. Torej velja

$$\pi_i(T)P_{ij}(T) = \pi_j(T)P_{ji}(T). \tag{7}$$

Za pare nesosednjih i, j sta obe strani enaki 0, torej enakost velja za poljubne pare $i, j \in S$.

Če seštejemo enačbe (7) za vse $i \in S$ in upoštevamo (4), dobimo (6). ■

Pred formulacijo naslednjega izreka definirajmo še nekaj oznak. Pri poljubnem $m \geq 1$ bodi $P(m, m)$ identična matrika in

$$P(m, m+n) = \prod_{i=0}^{n-1} P(T_{m+i})$$

za $n \geq 1$. Označimo z

$$\nu(m) = [\nu_1(m), \nu_2(m), \ldots, \nu_s(m)]$$

vektor verjetnosti stanj po m prehodih v markovski verigi, tako da je

$$\nu(m+n) = \nu(m)P(m, m+n).$$

Izkaže se (glej [7]), da kvazistacionarne porazdelitve $\pi(T_m)$ po komponentah konvergirajo k vektorju e^*, ki je določen z

$$e_i^* = \begin{cases} g(i)/g(*), & i \in S^* \\ 0, & i \notin S^* \end{cases}$$

kjer je S^* množica globalno optimalnih stanj, in je

$$g(*) = \sum_{j \in S^*} g(j)\,.$$

Najmočnejši konvergenčni izrek v [7] trdi, da je markovska veriga, ki pripada algoritmu $\mathcal{SA}$ s temperaturo

$$T_m = \frac{\gamma}{\log(m + m_0 + 1)}, \quad m = 0, 1, 2, \ldots$$

kjer sta γ in m_0 primerno izbrani konstanti, krepko ergodična. Namesto definicije krepke ergodičnosti rajši navedimo, kaj to pomeni v našem primeru:

Izrek 2. (Mitra et al., 1986). *Za vse m velja*

$$\lim_{n \to \infty} \sup_{\nu(0)} ||\nu(0) P(m, n) - e^*|| = 0.$$

Če torej parameter T pada dovolj počasi, potem iz kakršnegakoli začetnega stanja algoritem $\mathcal{SA}$ z verjetnostjo 1 po neskončno mnogo korakih konča v enem od globalnih optimumov.

4. Primerjava algoritmov $\mathcal{SA}$ in $\mathcal{RLS}$

V tem razdelku nas bo zanimalo, kako s časom (z naraščajočim številom korakov) raste verjetnost uspešnosti danega algoritma. Definirajmo

$$P_{\mathcal{A}}(n) \quad = \quad \Pr\{\textit{algoritem } \mathcal{A} \textit{ je uspel vsaj v } n \textit{ korakih}\}. \qquad (8)$$

Če za algoritma $\mathcal{A}$ in $\mathcal{A}'$ obstaja tako naravno število n_0, da je za vsak $n > n_0$

$$P_{\mathcal{A}}(n) > P_{\mathcal{A}'}(n),$$

potem rečemo, da je algoritem $\mathcal{A}'$ *asimptotsko slabši* od algoritma $\mathcal{A}$. Pokazali bomo, da je algoritem $\mathcal{SA}$ asimptotsko slabši od algoritma $\mathcal{RLS}$. Še več, iz dokaza bo sledilo, da je celo enostavno ponavljanje neodvisnih generiranj dopustnih rešitev asimptotsko uspešnejši algoritem kakor algoritem $\mathcal{SA}$. Tu samo omenimo, da je mogoče izrek posplošiti tudi na primer, ko dovolimo vzporedno izvajanje obeh algoritmov [2].

Predpostavimo, da je dan primerek I problema P. Označimo z N moč množice dopustnih rešitev. Pri danem primerku I je določeno tudi število stanj, iz katerih vse poti, ki gredo samo navzdol, končajo v enem od globalnih minimumov. Označimo to število s K_1.

Privzeli bomo, da algoritem $\mathcal{R}$ generira vse dopustne rešitve z enako verjetnostjo. (Privzetek deloma poenostavi dokaz, vendar ni bistven za resničnost izrekov.)

Najprej poiščimo spodnjo mejo za verjetnost, da algoritem lokalna optimizacija najde globalno optimalno rešitev v n korakih.

Ko prvič generiramo slučajno rešitev, je verjetnost, da se bo lokalna optimizacija končala v globalnem optimumu, enaka K_1/N. Za eno lokalno optimizacijo porabimo največ $Rd+w$ enot časa. Ker so naslednja generiranja začetnih rešitev neodvisna, imamo opravka z Bernoullijevim zaporedjem poskusov. Označimo $c = \lfloor \frac{n}{Rd+w} \rfloor$ (celi del), potem je

$$P_{\mathcal{RLS}}(n) \geq \frac{K_1}{N} + \left(\frac{N-K_1}{N}\right)\frac{K_1}{N} + \left(\frac{N-K_1}{N}\right)^2\frac{K_1}{N} + \ldots\left(\frac{N-K_1}{N}\right)^c\frac{K_1}{N} \geq$$

$$\geq \frac{K_1}{N}\sum_{i=0}^{c}\left(\frac{N-K_1}{N}\right)^i = \frac{K_1}{N}\left(\frac{1-(\frac{N-K_1}{N})^{c+1}}{(1-\frac{N-K_1}{N})}\right) =$$

$$= 1 - \left(\frac{N-K_1}{N}\right)^{c+1}.$$

Videli smo torej:

Lema 1. $P_{\mathcal{RLS}}(n) \geq 1 - \left(\frac{N-K_1}{N}\right)^{\lfloor \frac{n}{Rd+w} \rfloor + 1}$. (9)

Poiščimo še zgornjo mejo za verjetnost, da algoritem $\mathcal{SA}$ najde globalni optimum v n korakih. Pri danem primerku I problema P so določene tudi naslednje konstante:

- K označuje število stanj, iz katerih obstaja pot, ki gre samo navzdol in se konča v enem od globalnih minimumov. Očitno $K \geq K_1 > 0$. Da se izognemo trivialnim primerom, privzemimo še $N > K$.
- $1-p_i$ je verjetnost, da algoritem ne bo sprejel slabšega stanja v algoritmu $\mathcal{SA}$ pri temperaturi T_i. Lahko bi tudi rekli: verjetnost, da algoritem ne bo šel ‚gor'. (Vrednost p_i je odvisna tudi od trenutnega stanja.)
- $q_i = \min\{1 - p_i\}$, kjer minimum izračunamo po vseh možnih stanjih danega primerka problema. Temperatura je tu fiksirana (T_i). q_i je torej spodnja meja za verjetnost dogodka, da pri temperaturi T_i pri danem primerku problema algoritem $\mathcal{SA}$ ne bo sprejel koraka ‚gor', v smeri poslabšanja stroškovne funkcije. V dokazu izreka 2 bomo predpostavili $q_i \to 1$, ko gre $i \to \infty$. Če vzamemo (1) za definicijo p in predpostavimo $T_i \to 0$, sledi $q_i \to 1$, torej je za algoritem $\mathcal{SA}$ predpostavka izreka izpolnjena.

Lema 2. *Če je* $n = \tilde{m}_i + \sum_{j=1}^{i-1} m_j$ *in* $0 < \tilde{m}_i \leq m_i$, *velja*

$$P_{\mathcal{SA}}(n) \leq 1 - q_i^{\tilde{m}_i} q_{i-1}^{m_{i-1}} \ldots q_2^{m_2} q_1^{m_1}\left(1 - \frac{K}{N}\right), \quad (10)$$

Dokaz. Ker iščemo zgornjo mejo, smo lahko velikodušni. Rekli bomo, da je algoritmu $\mathcal{SA}$ uspelo (najti globalni optimum), brž ko je dosegel stanje, iz katerega obstaja vsaj ena pot, ki gre samo navzdol in se konča v globalnem optimumu.

Z x_i označimo verjetnost dogodka, da je algoritem SA uspel pri temperaturi, ki ni nižja od T_i.

Očitno je

$$x_i \leq x_{i-1} + (1 - x_{i-1})(1 - q_i^{m_i}) = 1 - q_i^{m_i}(1 - x_{i-1}), \qquad (11)$$

saj velja naslednje: v primeru, ko algoritem $\mathcal{SA}$ ni uspel, preden je temperatura padla na T_i, mora algoritem (po definiciji ‚uspeha') narediti vsaj en korak ‚gor', če naj uspe pri temperaturi T_i.

Za začetno temperaturo velja

$$x_1 \leq \frac{K}{N} + (1 - \frac{K}{N})(1 - q_1^{m_1}) = 1 - q_1^{m_1}\left(1 - \frac{K}{N}\right). \qquad (12)$$

Enostaven račun nam iz (11) in (12) da

$$x_i \leq 1 - q_i^{m_i} q_{i-1}^{m_{i-1}} \ldots q_2^{m_2} q_1^{m_1}\left(1 - \frac{K}{N}\right).$$

Torej za $n = \sum_{k=1}^{i-1} m_k + \tilde{m}_i$, kjer je $0 < \tilde{m}_i \leq m_i$, dobimo

$$P_{\mathcal{SA}}(n) \leq 1 - q_i^{\tilde{m}_i} q_{i-1}^{m_{i-1}} \ldots q_2^{m_2} q_1^{m_1}\left(1 - \frac{K}{N}\right). \quad \blacksquare$$

Zdaj se prepričajmo, da spodnja meja za verjetnost uspeha algoritma $\mathcal{RLS}$ raste hitreje kot zgornja meja verjetnosti uspeha algoritma $\mathcal{SA}$. Definirajmo

$$C := \left(1 - \frac{K_1}{N}\right)^{1/(Rd+w)}.$$

Očitno je $0 < C < 1$. Uporabimo privzetek $q_i \to 1$. Najprej predpostavimo, da obstaja tak k, da je $q_k \leq C$ in $q_{k'} > C$ za vsak $k' > k$. Vpeljimo konstanti:

$$D := q_{k-1}^{m_{k-1}} \ldots q_2^{m_2} q_1^{m_1}\left(1 - \frac{K}{N}\right),$$

$$m := \sum_{i=1}^{k-1} m_i .$$

Lahko izberemo tak (dovolj velik) $\tilde{n}$, da velja

$$\left(\frac{C}{q_k}\right)^{\tilde{n}} < C^{-m} D .$$

Zapisano drugače,

$$C^{\tilde{n}} < q_k^{\tilde{n}} C^{-m} q_{k-1}^{m_{k-1}} \dots q_2^{m_2} q_1^{m_1} \left(1 - \frac{K}{N}\right) .$$

Če vzamemo $n = \tilde{n} + m$, dobimo

$$C^n = C^{\tilde{n}+m} < q_k^{\tilde{n}} q_{k-1}^{m_{k-1}} \dots q_2^{m_2} q_1^{m_1} \left(1 - \frac{K}{N}\right) <$$

$$< q_i^{\tilde{m}_i} q_{i-1}^{m_{i-1}} \dots q_k^{m_k} q_{k-1}^{m_{k-1}} \dots q_2^{m_2} q_1^{m_1} \left(1 - \frac{K}{N}\right) .$$

Torej je

$$1 - C^n > 1 - q_i^{\tilde{m}_i} q_{i-1}^{m_{i-1}} \dots q_2^{m_2} q_1^{m_1} \left(1 - \frac{K}{N}\right)$$

za dovolj velike n.

Če je $q_i > C$ za vsak i, je dokaz te neenakosti še lažji. Ker iz leme 1 sledi $1 - C^n < P_{\mathcal{RLS}}(n)$, smo dokazali

Izrek 2. *Obstaja konstanta n_0, tako da za vsak $n > n_0$ velja $P_{\mathcal{SA}}(n) < < P_{\mathcal{RLS}}(n)$.*

LITERATURA

[1] E. H. L. Aarts, J. H. M. Korst, *Boltzmann Machines and Their Applications*, PARLE Parallel Architectures and Languages Europe, Lecture Notes in Comp. Sci. 258, Springer, Berlin 1987, 34–50.

[2] A. G. Ferreira, J. Žerovnik, *On the Behavior of Simulated Annealing*, (sprejeto v Jour Journal of computers & mathematics with applications).

[3] D. L. Isaacson, R. W. Madsen, *Markov Chains, Theory and Applications*, John Wiley and Sons, New York 1976.

[4] S. Kirckpatrick, C. D. Gellat Jr., M. P. Vecchi, *Optimization by Simulated Annealing*, Science **220** (1983) 671–680.

[5] A. Khačaturjan, S. Semenovskaja and B. Vainstein, *The Thermodynamical Approach to the Structure Analysis of Chrystals*, Acta Cryst **A37** (1981) 742–754.

[6] P. J. M. Laarhoven, E. H. L. Aarts, *Simulated Annealing, Theory and Applications*, D. Reidel Publishing Company, Dordrecht 1987.

[7] D. Mitra, F. Romeo, A. Sangiovanni-Vincentelli, *Convergence and Finite Time Behavior of Simulated Annealing*, Adv. Appl. Prob. **18** (1986) 747–771.

[8] M. Pincus, *A Monte Carlo Method for the Approximate Solutions of Certain Types of Constrained Optimization Problems*, Oper. Res **18** (1970) 1225–1228.

[9] G. H. Sasaki, B. Hajek, *The time Complexity of Maximum Matching by Simulated Annealing*, Jour. ACM **35** (1988) 387–403.

[10] J. Žerovnik, *Algoritem Ohlajanje (Simulated Annealing)*, Informatica **2** (1991) 40–48.

[11] J. Žerovnik, *Verjetnostni algoritmi*, Obzornik mat. fiz. **39** (1992) 999–999.

DELI IN VLADAJ Z OBJEKTNIM PROGRAMIRANJEM

JERNEJ KOZAK

Math. Subj. Class. (1991): 68N15, 68P05, 68Q25

V tem sestavku pokažemo, da nam objekti omogočajo programiranje na bistveno višjem abstraktnem nivoju, kot smo ga poznali brez njih. Za zgled si vzamemo strategijo *deli in vladaj*, ki jo v narečju programskega jezika pascal (turbo pascal 6.0) prepeljemo od zasnove do rešitev konkretnih problemov.

OBJECT VIEW OF DIVIDE AND CONQUER

In this note it is shown that objects admit programming at much higher abstract level as it was possible before. As an example the *divide and conquer* strategy is developed in a dialect of the programming langauge pascal (turbo pascal 6.0) from the basic concept to solutions of some particular problems.

1. Uvod

Ko je nastajala knjiga [6], sem se s kolegi večkrat posvetoval, kako bi v njej zapisal algoritme. Mnenja so bila kaj različna, odločil pa sem se za ohlapnejši, neformalni pristop. To je zapis poenostavilo in skrajšalo. Temu nasproti je stalo mnenje, da bi moral biti opis izpeljan v (slovnično neoporečnem) programskem jeziku, npr. v pascalu. Le tako bi algoritem bil zares prepričljiv. Tudi v knjigi [3] zasledimo podobne dvome: ali je abstraktno zapisan algoritem zares algoritem? Jedro dvomov tiči v tem, komu je algoritem namenjen. Če je zapisan ohlapno, je namenjen posredniku-človeku. Ta ga lahko nepravilno razume ali napačno prenese v programski jezik. Ker mora biti algoritem natančno določen, abstraktno zapisani postopek torej to ni! Vendar v praksi vse zahtevnejše probleme rešujemo tako, da začnemo kar se da abstraktno, v naravnem jeziku, in nadaljujemo z vse bolj natančnimi koraki, ki jih na koncu prepišemo v programski jezik. Na tej poti, polni pasti, se oddahnemo šele, ko algoritem prepustimo računalniku v avtomatično dodelavo (prevajanje, izvedba).

V tem članku se k zapisu algoritma vračamo ponovno, a z novim programerskim prijemom — objekti. Naš namen je pokazati, da nam objekti omogočajo prepis algoritmov iz naravnega v programski jezik na bistveno višjem abstraktnem nivoju, kot je bilo to mogoče doslej. V programskem jeziku začnemo lahko že z opisom same metode razvoja algoritmov določenega razreda. Pokazalo se bo, da gre koncept objekta z roko v roki z razvojem postopkov od osnovnih strategij do rešitev konkretnih problemov.

Strategija je abstrakten postopek, ki predpostavlja nekaj bistvenih značilnosti problema, ki ga rešuje. Na osnovi teh lastnosti je izpeljana rešitev problema. Običajno jo opišemo bolj z besedami, v naravnem jeziku. S pravim programiranjem začnemo šele ob posameznih problemih. In seveda s konkretno predstavitvijo podatkov. V teh vrsticah si bomo za zgled vzeli strategijo *deli in vladaj*. Najdemo jo opisano na mnogih mestih, med drugim v [6, str. 164–166]. Primerna je za reševanje problemov, pri katerih lahko

rešitev celotnega problema sestavimo iz rešitve podproblemov (ki so iste narave kot prvotni problem).

In zdaj nekaj besed o objektnem programiranju. Ta pristop uvaja koncept *objekta*, strukture, ki združuje *podatke* in *operacije* (procedure in funkcije) nad njimi. Operacijam rečemo *metode*. Dve vrsti metod sta posebne narave: **constructor**, ki spremenljivko danega objektnega tipa sestavi, in **destructor**, ki jo uniči. Podatke in operacije nad njimi združimo na enem mestu in s tem v objektu pregledno ohranimo odnose med sestavnimi deli. Tej prvi pomembni lastnosti rečemo *enkapsulacija*. Pomaga nam zagrešiti kar se da malo napak.

Druga pomembna lastnost objektov je *dedovanje*. Enega lahko gradimo kot naslednika drugega. Pri tem potomec deduje vse podatke in operacije, ki jih pozna prednik, in jim seveda doda svoje posebnosti. Metode, ki jih želi izpeljati po svoje, preprosto prekliče, ponovno sprogramira. Če popravimo prednika (odkrijemo programersko napako ali pospešimo posamezne metode), se popravijo tudi vsi njegovi potomci.

Tretja zelo pomembna lastnost je *polimorfizem* objektov. Že beseda sama pove, da objekt ni toga, do vseh podrobnosti dorečena struktura, ampak opisuje veliko potencialnih oblik hkrati. Polimorfne so v resnici skoraj vse strukture v programskem jeziku, a še zdaleč ne v tej meri kot objekti. Že npr. obseg celih števil se lahko loči od računalnika do računalnika, pojem celoštevilčne vrednosti pa ostaja en sam.

Polimorfizem objektov se kaže predvsem na dveh mestih: pri sami najavi objekta in pri njegovi uporabi. Pri najavi tipa objekta naštejemo in nato sprogramiramo vse metode objekta. Metode delimo na statične (običajne) in virtualne. *Statična metoda* je metoda, ki jo sprogramiramo enkrat za vselej. Potomci je ne morejo zamenjati z svojimi izpeljankami. Če najavijo metodo z enakim imenom, s prvotno ne bo imela nič skupnega. V nasprotju s tem so *virtualne metode* polimorfne. Potomci jih lahko nadomestijo (ne prekrijejo!) s svojimi izvedbami. V trenutku, ko zapišemo klic takšne metode, v resnici pokličemo njo ali katerokoli od metod, ki jo bo v potomcih objekta nasledila. Razumljivo je, da se morajo virtualne metode, ki jih objekt ponovno sprogramira, ujemati v številu in tipih parametrov z najavo metode v predniku.

V uporabi spremenljivk objektnega tipa nam polimorfizem omogoča, da lahko potomca vedno uporabimo namesto prednika. Če npr. v najavi dane procedure kot parameter navedemo kak objekt, lahko na mestu dejanskega parametra uporabimo tako objekt navedenega tipa kot vse njegove naslednike.

V tem sestavku se bomo naslonili na objektno programiranje v turbo pascalu 6.0 ([1]). To narečje pascala je široko dostopno, sam pascal pa osnovni programski jezik v srednjih šolah. Toliko v opravičilo zagovornikom jezika C^{++} ali smalltalk. Vsi v članku zajeti zgledi so preneseni neposredno iz delujočih, preverjenih programov. Opogumljamo bralca, da se z dodajanjem svojih potomcev tudi sam prepriča, da objektni pristop, opisan v

članku, ni le pedagoško orodje, ampak je tudi praktično uporaben. Začetna, časovno zahtevna naložba v pripravo osnovnih objektov se obilno povrne s preprostim programiranjem potomcev — rešitev konkretnih problemov.

2. Strategija *deli in vladaj*

Začnimo z osnovnim objektom TODinV, ki bo povzel strategijo *deli in vladaj* v smislu [6, str. 164-166] kar se da splošno. Na kratko opišemo metodo takole: če je problem, ki ga rešujemo, dovolj majhen, da ga lahko preprosto rešimo, to storimo. Drugače ga razdelimo v dva podproblema enake narave (še splošneje v več podproblemov), po potrebi rešimo podproblema in delne rešitve združimo v rešitev osnovnega problema.

Prvi korak v sestavi objekta je izbira podatkov, ki opisujejo posamezni problem. Ta opis mora biti tako splošen, da bodo potomci TODinV lahko obdelovali kaj različne podatke, npr. urejali števila, besedila, zapise, iskali simbole ipd. In to le z majhno spremembo prvotnih metod!

Majhnemu problemu pripada le skupina nekaj podatkov, večjim problemom več takšnih skupin. Narediti je torej treba dva koraka: dogovoriti se, kako opisati podatke majhnih problemov, in predpisati združevanje teh podatkov v podatke večjih problemov. Pri majhnih problemih ni kaj izbirati. Osnovnih podatkov ne poznamo, torej jih lahko predstavimo (kar se da abstraktno) le kot kazalec, ki kaže nanje. Zanj seveda še ni nič povedano, na kaj kaže. Ko bo čas, bodo to opredelili potomci osnovnega objekta. V turbo pascalu je tip takšnega kazalca **pointer**.

```
PVoz = ↑ TVoz;
TVoz = record                          { Vozlisce dvojiskega drevesa. }
         Pd : pointer;                   { Kazalec na pripadajoci podatek. }
         Ls : PVoz;                       { Kazalec na levega sina. }
         Ds : PVoz                      { Kazalec na desnega sina. }
       end;
TOpP = record                        { Podatki, ki dolocajo podproblem. }
         case integer of
           1: ( Z, K: integer ); { Linearno urejeni podatki v tabeli, problem Z:K. }
           2: ( R: integer );          { Dvojisko drevo v tabeli, koren R. }
           3: ( V: PVoz ) { Dinamicno predstavljeno dvojisko drevo, koren V↑. }
       end;
PTabela = ↑ TTabela;
TTabela = array[ 1..MaxInt div SizeOf(pointer) ] of pointer; { Genericna tabela. }
```

Slika 1. Najave podatkov za opis problema

Predstavitev podatkov večjih problemov sedaj pogojuje združevanje osnovnih kazalcev. Ta mora omogočati učinkovito delitev in združevanje. Najpreprosteje bo, če lahko podatke linearno uredimo in zložimo v tabelo. Za posamezni podproblem je v tem primeru treba povedati le, od kod do kod sega. Če to ne gre, uporabimo dvojiško drevo. Podatki, ki pripadajo danemu problemu, so kar tisti, ki jih pripadajoče drevo vsebuje. Če je

drevo levo poravnano, ga zložimo po nivojih v tabelo, sicer ga predstavimo dinamično. Slika 1 v najavi zapisa TOpP povzema te tri možnosti. Naj bo P tipa TOpP. V prvem primeru pripadajo P vsi podatki med P.Z in P.K. V drugem je P.R indeks korena, koren levega poddrevesa 2*P.R, desnega 2*P.R + 1 ipd. Dinamično predstavljeno drevo bo tu sestavljeno iz vozlišč tipa TVoz in polje TOpP.V v opisu problema kaže na koren drevesa. Za drugačne predstavitve bi morali zapis TOpP razširiti.

Objekt v turbo pascalu najavimo tako, da najprej opišemo sam tip: naštejemo prednika, podatke in metode. Nato slede sprogramirane metode. Objekte, namenjene večkratni uporabi, najpriročneje sprogramiramo v modulu (v turbo pascalu je to enota — **unit**). Najava objekta sodi v javni del modula (pred ločnico **implementation**) in sprogramirane metode v skriti del za to ločnico. V tem sestavku bomo najavo objektov in sprogramirane metode ločili po slikah.

Na sliki 2 je osnovni objekt. Sestavlja ga en sam podatek Konec in kopica metod. Spremenljivka Konec bo stražar, ki sporoči, če je rešitev osnovnega problema že najdena. Ker stražar ni namenjen za rabo zunaj objekta, smo ga skrili za zaveso **private**.

```
TODinV = object                    { Osnovni objekt metode Deli in vladaj. }
           constructor Init;                    { Inicializacija objekta. }
           procedure Resi( P: TOpP );
           function Majhen( P: TOpP ): Boolean; virtual;
           procedure ResiMP( P: TOpP ); virtual;
           procedure Deli( P: TOpP; var L, D : TOpP ); virtual;
           function Levo( L: TOpP ): Boolean; virtual;
           procedure Med( P: TOpP ); virtual;
           function Desno( D: TOpP ): Boolean; virtual;
           procedure Zdruzi( P, L, D: TOpP ); virtual;
           function Prazen( P: TOpP ): Boolean; virtual;
           private
           Konec: Boolean;              { Resnicno, ce je resitev ze najdena }
           procedure Zamenjaj( var pA, pB : pointer );
         end;
```

Slika 2. Najava praočeta strategije *deli in vladaj*

Med metodami objekta je najpomembnejša Resi, ki povzema celotno strategijo. Ker je nihče od potomcev ne bo več spreminjal, jo najavimo kot statično. Metoda Majhen(P) pove, kdaj je problem P majhen, ResiMP(P) reši majhen problem P, Deli(P,L,D) deli problem P v podproblema L,D (L-levi, D-desni). Levo(L) odloči, ali je potrebno rešitev v podproblemu L sploh iskati. Podobno velja za Desno(D). Oba podproblema, če je potrebno, reši ponovno Resi sam, seveda rekurzivno. Med(P) postori, če je sploh treba kaj, med iskanjem rešitve prvega in drugega podproblema in Zdruzi(P, L, D) združi rešitvi podproblemov L,D v rešitev problema P.

Večina metod je na tem mestu še povsem neopredeljenih in zato seveda virtualnih. Nekoliko si olajšajmo delo za pozneje in brez škode za splošnost

```
constructor TODinV.Init; { Inicializira osnovni objekt. } begin Konec := false; end;

procedure TODinV.Resi( P: TOpP );
{ Resi problem P. Jedro metode deli in vladaj. } var L, D: TOpP;
begin
  if Majhen( P ) then ResiMP( P )              { Ce je P majhen, ga resi. }
  else
    begin
      Deli( P, L, D );                    { Deli P v podproblema L in D. }
      if Levo ( L ) then Resi( L );       { Po potrebi resi levi podproblem L. }
      Med( P );                           { Postori, kar je treba po resitvi L. }
      if Desno( D ) then Resi( D );       { Po potrebi resi desni podproblem D. }
      Zdruzi( P, L, D );                  { Zdruzi resitev L in D v resitev P. }
    end;
end;

function TODinV.Majhen( P: TOpP ): Boolean;
{ Ali je problem P majhen? } begin Majhen := P.Z = P.K end;

procedure TODinV.ResiMP( P: TOpP );   { Resi majhen problem. } begin end;

procedure TODInV.Deli( P: TOpP; var L, D: TOpP );
{ Deli problem P v podproblema L in D. }
begin
  L.Z := P.Z; L.K := (P.Z + P.K) div 2; D.Z := L.K + 1; D.K := P.K
end;

function TODinV.Levo( L: TOpP ): Boolean;
{ Ali potrebujemo resitev levega podproblema L? }
begin Levo := (not Konec) and (not Prazen(L)) end;

procedure TODInV.Med( P: TOpP );    { Med resitvama. } begin end;

function TODinV.Desno( D: TOpP ): Boolean;
{ Ali potrebujemo resitev desnega podproblema? }
begin Desno := (not Konec) and (not Prazen(D)) end;

procedure TODInV.Zdruzi( P, L, D: TOpP );
{ Zdruzi resitve podproblemov L in D v resitev celotnega problema P. } begin end;

function TODinV.Prazen( P: TOpP ): Boolean;
{ Ali je P prazen problem? } begin Prazen := P.Z > P.K end;

procedure TODinV.Zamenjaj( var pA, pB: pointer );
{ Zamenja pA in pB. } var p: pointer;
begin p := pA; pA := pB; pB := p; end;
```

Slika 3. Metode objekta TODinV

privzemimo naslednje: podatki so predstavljeni s tabelo kazalcev, problem je majhen za zaporedja, dolga en podatek, zaporedje podatkov se deli na pol. Na sliki 3 so sprogramirane vse metode osnovnega objekta. K osnovnim metodam smo zaradi udobnosti dodali še metodo Prazen(P), ki pove, kdaj je problem P prazen, in metodo, ki zamenja dva kazalca (Zamenjaj — potomcem skrita metoda!). Lep zgled za uporabo virtualnih metod! Statična metoda Resi jih kliče celo vrsto. Kaj te metode v resnici pomenijo,

```
constructor TDinVD.Init( dK: PVoz ); begin TODinV.Init; K := dK end;

function TDinVD.Prazen( P: TOpP ): Boolean; begin Prazen := P.V = nil end;

function TDinVD.Majhen( P: TOpP ): Boolean; begin Majhen := Prazen( P ); end;

procedure TDinVD.Deli( P: TOpP; var L, D: TOpP );
begin L.V := P.V↑.Ls; D.V := P.V↑.Ds end;

procedure TDinVD.Poisci; var P: TOpP; begin P.V := K; Resi( P ) end;
```

Slika 7. Metode objekta TDinVD

bodo dorekli šele potomci.

V objektu TODinV še nismo povedali, kako so podatki o osnovnem problemu sploh podani. Opravimo korak naprej in dodajmo izpeljanki osnovnega objekta (TDinVT – slika 4 in TDinVD – slika 5).

```
TDinVT = object( TODInV )                          { Deli in vladaj s podatki v tabeli. }
           nd: integer;                                  { Stevilo podatkov v tabeli. }
           Pd: PTabela;                  { Kazalec na tabelo podatkov (kazalcev). }
           constructor Init( n : integer; P : PTabela );
           function Manjsi( pA, pB: pointer ): Boolean; virtual;
           function Enak( pA, pB: pointer ): Boolean; virtual;
           procedure Poisci;                                      { Poisci resitev. }
         end;
```

Slika 4. *Deli in vladaj* s podatki v tabeli

```
TDinVD = object( TODinV )
           K: PVoz;                          { Koren drevesa podatkov. }
           constructor Init( dK : PVoz );
           function Prazen( P: TOpP ): Boolean; virtual;
           function Majhen( P: TOpP ): Boolean; virtual;
           procedure Deli( P: TOpP; var L, D: TOpP ); virtual;
           procedure Poisci;                          { Poisci resitev. }
         end;
```

Slika 5. *Deli in vladaj* z drevesno predstavitvijo podatkov

```
constructor TDinVT.Init( n: integer; P: PTabela );
begin TODinV.Init; nd := n; Pd := P end;

function TDinVT.Manjsi( pA, pB: pointer ): Boolean; {  pA↑ < pB↑?  } begin end;

function TDinVT.Enak( pA, pB: pointer ): Boolean; {  pA↑ = pB↑?  } begin end;

procedure TDinVT.Poisci; var P:TOpP; begin P.Z := 1;P.K := nd; Resi(P) end;
```

Slika 6. Metode objekta TDinVT

Prva naj za predstavitev podatkov uporablja tabelo, druga dinamično predstavljeno dvojiško drevo. Tako v prvem kot drugem primeru moramo v osnovni objekt dodati podatke, ki opredelijo osnovni problem. Zato v prvo izpeljanko dodamo velikost tabele nd in kazalec nanjo (Pd), v drugo kazalec na koren dvojiškega drevesa (K). Te osnovne podatke naj v objekt zapiše konstruktor!

Za TDinVT sam je to povsem dovolj. Ker pa mnogo problemov, ki jih rešimo z metodo *deli in vladaj*, zahteva primerjavo podatkov, smo zaradi priročnosti dodali še metodi Manjsi in Enak, ki znata primerjati dva podatka. Prva naj vrne *resnično*, če je podatek, na katerega kaže prvi kazalec, manjši od podatka, na katerega kaže drugi kazalec. Druga naj preverja enakost. Zaenkrat sta obe metodi še nemi.

V TDinVD so podatki predstavljeni v dinamičnem dvojiškem drevesu. Osnovni objekt je predpostavljal predstavitev podatkov v tabeli. Bežen pregled metod objekta TODinV pove, da je treba reprogramirati Prazen, Majhen in Deli. Problem P naj bo sedaj majhen tedaj, ko je prazen, in prazen tedaj, ko P.V kaže v prazno. Deli naj deli problem v problema, ki pripadata levemu in desnemu poddrevesu.

Metode objekta TDinVT so povzete v sliki 6, objekta TDinVD pa v 7. Obema objektoma smo zaradi priročnosti dodali še metodo Poisci, ki poišče rešitev začetnega problema. V naslednjih treh razdelkih bomo oba objekta dogradili v rešitve konkretnih problemov.

3. Urejanje

S problemom urejanja podatkov se srečamo kaj pogosto. Pred časom je celo veljalo, da urejanje pojé kar četrtino vseh računalniških zmogljivosti. Najpopolnejši pregled algoritmov urejanja najdemo v [5], poučno je tudi branje v [7]. Čeprav je bilo o urejanju prelitega že ogromno črnila, je pri tem pogosto zamolčano preprosto dejstvo: algoritmi, ki urejajo, temelje na strategiji *deli in vladaj*. To velja tako za tiste, ki urejajo s primerjavami, kot tudi za tiste, ki urejajo po distribuciji.

Podkrepimo to z zgledoma. Predpostavimo, da je zaporedje podatkov, ki ga je treba urediti, predstavljeno s tabelo. Sestavimo dva objekta, ki znata preurediti zaporedje v urejeno, vsak seveda na svoj način. Osvežimo osnovno strategijo: zaporedje, ki ni majhno, uredimo tako, da ga razdelimo v dve podzaporedji, ti uredimo in dobljeni urejeni podzaporedji združimo. Razmišljati moramo le o tem, kako izpeljati metodi Deli in Zdruzi. Vse druge metode očeta TDinVT so uporabne takšne, kot so. Zahtevnost urejanja zaporedja raste hitreje kot sama velikost zaporedja. Zato bo postopek (verjetno) učinkovitejši, če bosta oba podproblema, v katera delimo problem, približno enako velika. Naj bo to vodilo pri iskanju metode!

Vztrajajmo najprej, da naj bo delitev preprosta, npr. kar delitev na pol. Prednikov Deli je že pravi, treba je sprogramirati le Zdruzi. Pri tem ni nobene svobode več. Kaj naj Zdruzi stori, je očitno: dve urejeni podzaporedji (rešitvi podproblemov) mora zliti v urejeno celotno zaporedje.

```
THitroUredi =
   object( TDinVT ) procedure Deli( P: TOpP; var L, D: TOpP ); virtual; end;

TUrediZZlivanjem =
   object( TDinVT ) procedure Zdruzi( P, L, D: TOpP ); virtual; end;
```

Slika 8. Objekta za hitro urejanje in urejanje z zlivanjem

```
procedure THitroUredi.Deli( P: TOpP; var L, D : TOpP );
{  Prerazporedi P.Z:P.K v L.Z:L.K−1 manjsih in D.Z:D.K vecjih od L.K:L.K.  }
var i, j : integer;
begin
   if Manjsi( Pd↑[P.K], Pd↑[P.Z] ) then Zamenjaj( Pd↑[P.Z], Pd↑[P.K] );
   i := P.Z; j := P.K;
   while i < j do
      begin
         repeat i := i + 1 until not Manjsi( Pd↑[i], Pd↑[P.Z] );
         repeat j := j − 1 until not Manjsi( Pd↑[P.Z], Pd↑[j] );
         if i < j then Zamenjaj( Pd↑[i], Pd↑[j] );
      end;
   Zamenjaj( Pd↑[P.Z], Pd↑[j] );
   L.Z := P.Z; L.K := j−1; D.Z := j+1; D.K := P.K
end;

procedure TUrediZZlivanjem.Zdruzi( P, L, D: TOpP );
{  Zlije urejeni podzaporedji L.Z:L.K in D.Z:D.K v urejeno zaporedje P.Z:P.K.  }
var il,id,ik: integer; pDt: PTabela;
begin
   GetMem( pDt, nd*SizeOf(TOpP));          {  Pripravimo delovno tabelo.  }
   il := L.Z; id := D.Z; ik := P.Z;
   while (il ≤ L.K) and (id ≤ D.K) do  {  Dokler obe podzaporedji nista izcrpani.  }
      begin
         if Manjsi( Pd↑[il], Pd↑[id] ) then
            begin pDt↑[ik] := Pd↑[il]; il := il + 1 end
         else
            begin pDt↑[ik] := Pd↑[id]; id := id + 1 end;
         ik := ik + 1;
      end;
   for id := L.K downto il do Pd↑[id+ik−il] := Pd↑[id];    {  Prelozimo levi rep.  }
   for il := P.Z to ik−1 do pD↑[il] := pDt↑[il];      {  Prelozimo zlito zaporedje.  }
   FreeMem( pDt, nd*SizeOf(TOpP) );          {  Sprostimo delovno tabelo.  }
end;
```

Slika 9. Metodi za obe urejanji

Izpeljali smo urejanje z zlivanjem!

Zahtevajmo sedaj, da Zdruzi (ki je v predhodnem primeru kar zapleten) ni potreben. To pomeni, da mora Deli razdeliti osnovno zaporedje tako, da so podatki v enem delu vsi manjši ali enaki kot v drugem. Tu imamo na izbiro več poti. Preproste vodijo v kvadratno zahtevne postopke ([6, str. 179–180]), Hoarova ideja pa v enega najučinkovitejših postopkov ureja-

nja. Ta izbere delilni podatek, npr. prvi element zaporedja. Nato pregleda zaporedje. Podatke, ki so večji od njega, postavi desno (D.Z:D.K), manjše levo od njega (L.Z:D.K-1). Mogoče je pokazati, da sta obe podzaporedji pri slučajno izbranih podatkih v povprečju približno enako veliki. Izpeljali smo hitro urejanje!

Objekta sta najavljena na sliki 8 in metodi sprogramirani na sliki 9. Seveda objekta THitroUredi in TUrediZZlivanjem še ne urejata zares, saj še nismo opredelili, kaj naj urejata. Prav v tem, da smo odložili odločitev o tem, kaj urejamo, povsem do konca, se ponovno pokaže pomembnost virtualnih metod. S trivialnim korakom objekta dopolnimo v potomce, ki urejajo karkoli.

```
TMojHitroUredi =
   object( THitroUredi ) function Manjsi( pA,pB: pointer): Boolean; virtual; end;

function TMojHitroUredi.Manjsi( pA, pB: pointer ): Boolean;
begin Manjsi := PReal( pA )↑ < PReal( pB )↑ end;
```

Slika 10. Objekt, ki hitro ureja realna števila

Predpostavimo, da je tip PReal kazalec na realno in PInteger kazalec na celo število. Na sliki 10 je objekt, ki zna hitro urejati realna števila, na sliki 11 pa objekt, ki zna z zlivanjem urejati cela števila.

```
TMojUrediZZlivanjem =
   object( TUrediZZlivanjem ) function Manjsi(pA,pB:pointer): Boolean; virtual; end;

function TMojUrediZZlivanjem.Manjsi( pA, pB: pointer ): Boolean;
begin Manjsi := PInteger( pA )↑ < PInteger( pB )↑ end;
```

Slika 11. Objekt, ki z zlivanjem uredi cela števila

Reprogramirati je bilo treba le metodo Manjsi. V njej smo najavili, na kaj posamezni kazalec kaže, in s tem omogočili računalniku, da primerjavo tudi opravi. Če predpostavimo, da je spremenljivka UrHit tipa TMojHitro-Uredi in PTabPtr kaže na tabelo kazalcev na zaporedje realnih števil, bosta klica

```
UrHit.Init( n, PTabPtr ); UrHit.Poisci
```

uredila zaporedje na mestu.

4. Bisekcija

Z bisekcijo npr. poiščemo podatek v urejenem zaporedju, simbol v dvojiškem slovarju, ničlo zvezne funkcije ipd. ([6, str. 169, 255]). Osnovno vodilo bisekcije je: problem razdelimo na tri dele, na majhen problem in dva druga. Če rešitev najdemo v malem problemu, je postopek končan. Drugače nadalje iščemo v enem ali drugem podproblemu. Podatki morajo biti

vnaprej urejeni tako, da vemo, v katerem od podproblemov naj nadaljujemo. Ponovno torej problem, ki sodi pod okrilje naše strategije.

Za zgled sestavimo dva objekta: TBisekcija in TIsciKtega. Prvi naj ugotovi, ali v urejenem zaporedju dani podatek obstaja ali ne. Drugi naj poišče tistega, ki je k-ti po velikosti.

```
TBisekcija = object( TDinVT )
                pX: pointer;                    { Kazalec na podatek, ki ga iscemo. }
                Indeks: Integer;                { Indeks najdenega elementa. }
                function Poisci( apX: pointer ): integer;
                procedure ResiMP( P: TOpP ); virtual;
                procedure Deli( P: TOpP; var L, D: TOpP ); virtual;
                function Levo( L: TOpP ): Boolean; virtual;
                function Desno( D: TOpP ): Boolean; virtual;
             end;

TIsciKtega = object( THitroUredi )
                kS: integer;                    { Iscemo kS-ti zaporedni podatek. }
                procedure Poisci( k: integer ); virtual;
                procedure ResiMP( P: TOpP ); virtual;
                function Levo( L: TOpP ): Boolean; virtual;
                function Desno( D: TOpP ): Boolean; virtual;
             end;
```

Slika 12. Objekta za iskanje z bisekcijo

Slika 12 vsebuje najavo obeh objektov, slika 13 pa sprogramirane metode.

Objekt TBisekcija bo potomec TODinV. Dodamo dva podatka: pX, kazalec na podatek, ki ga v zaporedju iščemo, in Indeks, ki označi, kje smo ga v zaporedju našli. Vrednost pX določi funkcija Poisci, ki tudi vrne odgovor. Opazimo, da ta funkcija, ki je statična, nima nič skupnega z enako imenovano očetovo proceduro. Le to, da jo s pridom uporabi. Ponovno so sprogramirane še tele metode: ResiMP: zabeleži, ali je to, kar iščemo, najdeno; Levo,Desno: pove, ali dana veja sploh pride v poštev za iskanje; Deli, ki najprej z očetovo metodo razdeli zaporedje na pol, nato od levega zaporedja odlušči zadnji podatek in pogleda, ali rešitev tiči v njem.

Za iskanje k-tega podatka uporabimo prednika THitroUredi. Njegov Deli razmesti zaporedje okoli delilnega podatka. Če je njegov indeks že pravi, končamo, sicer nadaljujemo v enem ali drugem podzaporedju. Rezultat iskanja je podatek na k-tem mestu. Zgled na sliki 14 določi osebo, ki je k-ta po letu rojstva, in znotraj tega po abecedi.

5. Dvojiška drevesa

Prvi zgled tu naj bo kopica, dvojiško, levo poravnano drevo. Podatek, ki pripada vsakemu vozlišču, mora biti večji od podatkov, ki pripadajo sinovom. Ker je drevo poravnano, ga učinkovito predstavimo v tabeli. Položaj sinov očeta i da račun $i \longmapsto 2i, i \longmapsto 2i+1$, položaj očeta vozlišča

```
function TBisekcija.Poisci( apX : pointer ): integer;
begin    { Vrne indeks iskanega podatka. −1 pomeni, da ga v zaporedju ni. }
   Indeks := −1; Konec := false; pX := apX; TDinVT.Poisci; Poisci := Indeks
end;

procedure TBisekcija.ResiMP( P: TOpP );
begin   { Je P iskani podatek?  }
   if Enak( Pd↑[P.Z], pX ) then begin Indeks := P.Z; Konec := true end;
end;

procedure TBisekcija.Deli( P: TOpP; var L, D: TOpP );
begin    { Deli problem P v L.Z:L.K−1, L.K:L.K in D.Z:D.K.  }
   TDinVT.Deli( P,L,D ); P.Z := L.K; P.K := L.K; L.K := L.K − 1; ResiMP( P );
end;
```

```
function TBisekcija.Levo( L: TOpP ): Boolean;
begin Levo := TDinVT.Levo( L ) and (not Manjsi( Pd↑[L.K], pX )) end;

function TBisekcija.Desno( D: TOpP ): Boolean;
begin Desno := TDinVT.Desno( D ) and (not Manjsi( pX, Pd↑[D.Z] )) end;

procedure TIsciKtega.ResiMP( P: TOpP );
begin if kS = P.Z then Konec := true end;

function TIsciKtega.Levo( L: TOpP ): Boolean;
begin Levo := THitroUredi.Levo( L ) and (kS ≤ L.K) end;

function TIsciKtega.Desno( D: TOpP ): Boolean;
begin Desno := THitroUredi.Desno( D ) and (kS ≥ D.Z) end;

procedure TIsciKtega.Poisci( k: integer );
begin kS := k; Konec := false; THitroUredi.Poisci end;
```

Slika 13. Metode v bisekciji

```
PStavek = ↑TStavek;
TStavek = record
                ime : string[30];                    { Priimek in ime  }
                leto : integer;                      { Leto rojstva  }
                                          { ... in druga polja ...  }
          end;
TMojIsciKTega =
   object( TIsciKTega ) function Manjsi( pA, pB: pointer ): Boolean; virtual; end;

function TMojIsciKtega.Manjsi( pA, pB: pointer ): Boolean;
begin
   if PStavek( pA )↑.Leto = PStavek( pB )↑.Leto then
      Manjsi := PStavek( pA )↑.Ime < PStavek( pB )↑.Ime
   else Manjsi := PStavek( pA )↑.Leto < PStavek( pB )↑.Leto
end;
```

Slika 14. Poišči $k-$to osebo

i pa $i \longmapsto \lfloor \frac{i}{2} \rfloor$. Naj objekt TKopica sestavi kopico: deli dano drevo v levo poddrevo, koren in desno poddrevo. Obe poddrevesi sestavi v kopico. Združi obe kopici s korenom tako, da bo celotno drevo tudi kopica.

```
TKopica = object( TDinVT )
            function Prazen( P: TOpP ): Boolean; virtual;
            procedure Deli( P: TOpP; var L, D: TOpP ); virtual;
            procedure Zdruzi( P, L, D: TOpP ); virtual;
          end;
```

Slika 15. Objekt, ki sestavi kopico

TKopica bo potomec TDinVT.

V opisu problema P tu vodimo indeks korena P.R. Povejmo, kdaj kaže v prazno: P.R $\notin$ $[1 : nd]$. To stoji v metodi Prazen. Metoda Deli opravi delitev tako, da izračuna korena levega in desnega poddrevesa. Metodo Zdruzi poenostavljeno povzemamo po [4]. Podatek iz korena spustimo do lista, tako da ga ves čas zamenjujemo z večjim od sinov. Nato ga po isti poti dvigujemo toliko časa, da trči v večjega od sebe.

```
function TKopica.Prazen( P: TOpP ): Boolean;
begin Prazen := (P.R ≤ 0) or (P.R > nd) end;

procedure TKopica.Deli( P: TOpP; var L, D: TOpP );
begin L.R := 2*P.R; D.R := 2*P.R + 1 end;

procedure TKopica.Zdruzi( P, L, D: TOpP );
var tP, O, S: TOpP;
begin
  tP := P;
  while not Prazen( L ) do
    begin
      S := L;
      if not Prazen( D ) then
        if Manjsi( Pd↑[L.R], Pd↑[D.R] ) then S := D;
      Zamenjaj( Pd↑[tP.R], Pd↑[S.R] );
      tP := S; L.R := 2*tP.R; D.R := 2*tP.R + 1;
    end;
  O.R := tP.R div 2;
  while tP.R <> P.R do
    begin
      if Manjsi( Pd↑[tP.R], Pd↑[O.R] ) then exit;
      Zamenjaj( Pd↑[tP.R], Pd↑[O.R] );
      tP := O; O.R := tP.R div 2;
    end;
end;
```

Slika 16. Metode objekta, ki sestavi kopico

Za konec navedimo še zgled, ki temelji na nadgradnji objekta TDinVD. Podatki o problemu so tu zbrani v dinamičnem dvojiškem drevesu. Privzemimo, da je to drevo slovar. To pomeni, da je kazalec TVoz.Pd kazalec na besedilo, npr. PString. Besedila so razvrščena tako, da so v vsakem levem poddrevesu po abecedi pred in v vsakem desnem poddrevesu za besedilom v korenu.

```
TMojelskaInoDrevo = object( TDinVD ) procedure Med( P: TOpP ); virtual; end;
procedure TMojelskaInoDrevo.Med(P:TOpP); begin writeln(PString(P.V↑.Pd)↑) end;
```

Slika 17. Objekt, ki izpiše slovar po abecedi

Kako slovar izpišemo v abecednem vrstnem redu? Očitno spet znana naloga: deli slovar v dva dela, besedila, ki so pred besedilom v korenu, in besedila, ki mu slede. Izpiši levo drevo po abecedi, izpiši podatek v korenu, izpiši desno poddrevo po abecedi. Združevanja rešitev torej ni, edina metoda, ki jo moramo sprogramirati, je Med, ki je v vseh dosedanjih zgledih bila le nema metoda. Tu izpiše podatek v korenu, celotni objekt TMDrevo pa kaže slika 17.

Izpis slovarja po abecedi je le poseben primer pregleda dvojiškega drevesa ([6, str. 96–104]). Gornji premislek lahko brez težav prenesemo na kateregakoli od pregledov.

LITERATURA

[1] *Priročnik za Turbo pascal 6.0*, Programmer's Guide, Borland 1990.
[2] *Priročnik za Turbo pascal 6.0*, Turbo Vision Guide, Borland 1990.
[3] E. Horowitz, S. Sahni, *Fundamentals of Data Structures*, Computer Science Press, Inc., Maryland, 1977.
[4] S. Klavžar, J. Marinček, *Izboljšano urejanje s kopicami — Bottom-up heapsort*, Obzornik mat. fiz. **38** (1991) 39–44.
[5] D. E. Knuth, *The Art of Computer Programming, vol. 3, Sorting and Searching*, Addison-Wesley Publishing Company, Reading Massachusetts, 1972.
[6] J. Kozak, *Podatkovne strukture in algoritmi*, DMFA, Ljubljana 1986.
[7] N. Wirth, *Algorithms + Data Structures = Programs*, Prentice-Hall, Englewood Cliffs, 1976.

VESTI

PREŠERNOVE NAGRADE ŠTUDENTOM MATEMATIKE FEBRUARJA 1993

Prešernovo nagrado Univerze v Ljubljani je letos dobilo 13 študentov, med njimi tudi študent matematike Bor Plestenjak za delo Optimalni cirkulantski operatorji.

Na Fakulteti za naravoslovje in tehnologijo so Prešernove nagrade dobili samo študentje matematike:

Gregor Cigler	Linearno rekurzivna zaporedja
Jaka Cimprič	Simultana diagonalizacija matrik
Barbara Drinovec	Razširitve Jordanskih baz
Sašo Strle	Pioncaréjeva homološka sfera
Matjaž Željko	Eksotični faktorji prostora $\mathbb{R}^4$
Arjana Žitnik	Risanje grafov na ploskve

IGRE NARAVE 8.

OKO IN UHO IMATA ENAK PRAG ZA ZAZNAVANJE ENERGIJSKEGA TOKA

MITJA ROSINA

PACS 43.50.+y, 43.66.+y, 42.66.Lc

Prag vidnosti je komaj za velikostno stopnjo nad kvantnim šumom, prag slušnosti pa le za velikostno stopnjo nad termičnim šumom.

SOME TRICKS OF NATURE 8.

THE THRESHOLDS FOR THE EYE AND THE EAR ARE AT THE SAME ENERGY CURRENT DENSITY

The threshold of vision is only one order of magnitude above quantum noise, while the threshold of hearing is only one order of magnitude above thermal noise.

Igra narave

Oko in uho sta zelo različna inštrumenta. Zato nas preseneti, da imata oba približno isti prag gostote energijskega toka $j = 10^{-12}\ \mathrm{W\,m^{-2}}$, ki ga še zaznata (pri optimalnih valovnih dolžinah) [1], [2]. To ustreza sveči 30 km daleč, zvezdi sedme magnitude ter zvočniku 10 μW 1 km od nas.

Kot bomo videli, se oko zelo približa kvantnemu šumu, uho pa termičnemu šumu. Če bi bila oba inštrumenta še za dobro velikostno stopnjo občutljivejša, bi nam v temnem in tihem prostoru bliskalo pred očmi in šumelo v ušesih. Ker se to ne izplača, narava tega ni naredila.

Da slučajno ustreza obema šumoma približno isti energijski tok, je igra narave.

Prag vidnosti

Izračunajmo, koliko fotonov pride v oko pri omenjeni najmanjši gostoti moči v blisku, ki traja 1/20 s. Vzemimo svetlobo z valovno dolžino 555 nm (fotone z energijo 2,2 eV) ter odprto zenico s prerezom 0,5 cm^2.

$$E = jSt = 10^{-12}\,\mathrm{W\,m^{-2}} \cdot 0,5 \cdot 10^{-4}\,\mathrm{m^2} \cdot 0,05\,\mathrm{s} =$$
$$= 2,5 \cdot 10^{-18}\ \mathrm{J} = 16\,\mathrm{eV} \rightarrow 7\ \text{fotonov}\,.$$

Očitno potrebujemo vsaj en foton, saj svetloba prihaja v kvantih in ne v poljubno majhnih obrokih. Dejansko jih potrebujemo vsaj 10 ali 100, da aktiviramo vsaj nekaj sosednjih čutnic. Če se aktivira ena sama čutnica, ne zaznamo ničesar. Posamezna čutnica bi se lahko sprožila slučajno, zato je narava ubrala zanesljivejšo pot, ko primerja sosednje signale. Tudi izkoristek fotonov ni popoln, je pa presenetljivo odličen.

Zaradi slikovite (ne)analogije z ušesom smo zrnatost svetlobe imenovali „kvantni šum".

Prag slušnosti

Privzemimo temperaturo $T = 290$ K, ki ji ustreza $kT = 0,025$ eV. Pri svetlobi je termični šum [3] $h\nu/(\exp(h\nu/kT)-1) = 1,3\cdot10^{-38}$ eV zanemarljiv v primerjavi s „kvantnim šumom" $h\nu = 2,2$ eV, pri ušesu pa je obratno. Pri zvoku je termični šum $kT = 0,025$ eV, „kvantni šum" pa je pri frekvenci 1000 Hz le $h\nu = 4\cdot10^{-12}$ eV.

Povprečna termična energija oscilatorja je kT, (ker je $kT \gg h\nu$). Moč, s katero energija prihaja in odhaja, ocenimo tako, da to energijo delimo z značilnim časom prehajanja $\Delta t = 1/\Delta\nu$ (bolje: $\Delta t = 1/4\Delta\nu$), kjer je $\Delta\nu$ opazovani frekvenčni interval.

Ocenimo termični šum v ušesu s prerezom bobniča $S = 1$ cm^2, če nas moti frekvenčni interval $\Delta\nu = 5000$ Hz.

$$j = P/S = 4kT\Delta\nu/S = 500\,\mathrm{eV\,s^{-1}\,cm^{-2}} = 0,8\cdot10^{-12}\,\mathrm{W\,m^{-2}}.$$

Enakovredno oceno lahko napravimo tudi z nazornim mikroskopskim razmislekom. V zraku je številčna gostota molekul $n = 25\cdot10^{24}\,\mathrm{m^{-3}}$. V času $\Delta t = 1/5000\,\mathrm{s}$ udari $N = n\bar{v}S\Delta t/4 = 0,55\cdot10^{20}$ molekul ob bobnič ($\bar{v} = 440\,\mathrm{m\,s^{-1}}$) in izvaja tlak 1 bar. Statistična fluktuacija tega števila je $N \pm \sqrt{N}$, sorazmerna je tudi statistična fluktuacija tlaka $\Delta p = p/\sqrt{N} = 10^5\,\mathrm{Pa}/0,74\cdot10^{10} = 1,3\cdot10^{-5}\,\mathrm{Pa}$. Temu ustreza gostota moči

$$j' = (\Delta p)^2/\rho_{\mathrm{zraka}}c_{\mathrm{zvoka}} = 0,40\cdot10^{-12}\,\mathrm{W\,m^{-2}}.$$

Ker nismo zelo pazljivo ocenili števila molekul N, ki sodelujejo pri fluktuacijah, vrednosti j in j' nista isti, vendar sta zadovoljivo blizu. Bralec se lahko prepriča, da sta algebrajska izraza za j in j' do numeričnega koeficienta ista, če vstavi [3] za $\bar{v} = \sqrt{8kT/\pi M}$, $c_{\mathrm{zvoka}} = \sqrt{\kappa kT/M}$, $\kappa = c_p/c_v = 1,4$ in $\Delta t = 1/\Delta\nu$:

$$j' = (\Delta p)^2/\rho_{\mathrm{zraka}}c_{\mathrm{zvoka}} = \frac{p^2}{n\bar{v}S\Delta t/4}\,\frac{1}{Mnc_{\mathrm{zvoka}}} = \sqrt{\frac{\pi}{8\kappa}}\,\frac{4kT\Delta\nu}{S}.$$

LITERATURA

[1] J. Strnad, *Fizika, 1. del*, Državna založba Slovenije, Ljubljana 1977, 174.
[2] J. Strnad, *Fizika, 2. del*, Državna založba Slovenije, Ljubljana 1978, 523.
[3] I. Kuščer, S. Žumer, *Toplota*, DMFA Slovenije, Ljubljana 1987, 154, 201.

NOVE KNJIGE

F. KRIŽANIČ, Linearna algebra in linearna analiza (LAILA), Državna založba Slovenije, Ljubljana 1993, 544 str.

Leta 1990 je DZS izdala obsežen Križaničev visokošolski učbenik *Temelji realne matematične analize* (TRMA). Avtor ga postavlja na začetek svojega zaokroženega tečaja matematične analize, na naslednje mesto pa v uvodu TRME uvrsti knjigo LAILA, ki je leta 1966 izšla pri Mladinski knjigi. Slednjo je zdaj temeljito prenovil in razširil. Nastalo je novo delo s starim imenom: LAILA. Po avtorjevih besedah je to učbenik linearne algebre s skromnim uvodom v začetne pojme funkcionalne analize. Bežno jo prelistajmo.

V obširnem prvem poglavju nas pisec popelje v svet končno razsežnih vektorskih prostorov, linearnih preslikav in matrik. Seznani nas tudi s polilinearnimi formami in se med poševno-simetričnimi najdlje posveti determinanti. V drugem poglavju obravnava bilinearne in kvadratne forme ter z njimi poveže geometrije na vektorskih prostorih. V tem naravnem okolju zagleda tudi evklidske in unitarne prostore. Naslednje poglavje posveti endomorfizmom končno razsežnih vektorskih prostorov. Zanima ga Jordanova matrika endomorfizma, ki jo najde s pomočjo funkcijskega računa na kvadratni matriki. Zajame tudi posebne endomorfizme v unitarnem in evklidskem prostoru, poglavje pa sklene z uporabo kvadratnih form pri klasifikaciji ploskev drugega reda. V poglavju z naslovom Polilinearna algebra podrobno obravnava polilinearne in še posebej poševno-simetrične forme. Vanj vključi tenzorski produkt vektorskih prostorov in linearnih preslikav, vpelje pa tudi Grassmannovo algebro vektorskega prostora, ki jo posebej prouči v primeru evklidskega prostora. V petem poglavju nas iz linearne algebre popelje v analizo. Osrednja tema tega poglavja so Hilbertovi prostori funkcij in Fourierove vrste v njih – bistveno vlogo odigrajo v nadaljevanju knjige. V šestem poglavju avtor pripravi gradivo iz splošne teorije linearnih operatorjev v Hilbertovem prostoru, ki ga koristno uporabi pri podrobni obravnavi linearnih diferencialnih operatorjev drugega reda v naslednjem poglavju. V zadnjem, osmem poglavju nam postreže s kopico zgledov iz matematične fizike.

Knjiga je napisana v značilnem Križaničevem slogu, živo in barvito, duhovito in jedrnato, z izostrenim posluhom za slovenski jezik. Avtor z občutkom vodi bralca skozi zahtevno gradivo, ki ga splete v čvrsto povezano celoto. Na tej poti večkrat postane in se razgleda, opozori na pomembnejši korak in ves čas usmerja bralca k bistvu. Knjiga je pisana na zelo visoki ravni in zahteva posluh za matematični jezik. Naj ta misel zveni kot vabilo in ne kot svarilo.

LAILA je sodoben visokošolski učbenik, koristen vir vsem, ki se resneje srečajo z linearno algebro ali matematično fiziko. Mednje prav gotovo sodijo študentje matematike in fizike (tudi podiplomski), ki jim lahko služi kot izvrstna dopolnitev k predavanjem iz linearne algebre in analize. Za

osvežitev in poglobitev znanja bo knjiga dobrodošla marsikomu, še posebej pa učiteljem matematike.

Naj sklenem z vabilom iz Križaničevega gimnazijskega učbenika: Ne **boj se in ne sovraži!**

Boris Lavrič

VESTI

POROČILO KOMISIJE ZA TISK

V letu 1992 je Komisija za tisk v sodelovanju z Oddelkom za matematiko in mehaniko in Oddelkom za fiziko na FNT, Inštitutom za matematiko, fiziko in mehaniko ter Slovenskim društvom raziskovalcev šolskega polja izdala naslednje publikacije.

Obzornik za matematiko in fiziko, 39 (1992) 1, 2, 3, 4, 5, 6 (1093, 1099, 1105, 1119, 1133, 1134)

Presek – list za mlade matematike, fizike astronome in računalnikarje, **19** (1991/92) številke 4, 5, 6 ter **20**(1992/93) številke 1, 2, 3 (1094, 1097, 1101, 1115, 1127, 1137)

Presekova knjižnica:

- Ranzinger P., Presekova zvezdna karta, 3. natis (1117)

Priročniki in učbeniki:

- Željko M., Šviligoj B., PiCTeX CAD in navodila za uporabo programa (1110)
- Štalec I., Zbirka vaj iz aritmetike, algebre in analize za 1. razred srednjih šol, 15. natis (1121)
- Štalec I., Zbirka vaj iz aritmetike, algebre in analize za 2. razred srednjih šol, 14. natis (1122)
- Avsec F., Cokan A., Molinaro I., Pucelj I., Roblek B., Štalec I., Vagaja M., Zbirka vaj iz aritmetike, algebre in analize za 3. razred srednjih šol, 12. natis (1123)
- Štalec I., Zbirka vaj iz aritmetike, algebre in analize za 4. razred srednjih šol, 10. natis (1124)
- Zbirka testnih nalog druge mednarodne raziskave znanja matematike z rešitvami (1103)
- Uršič S., Matematične tabele in formule, 3. natis (1118)
- Lokar M., Turbo pascal 6.0 (Račkova knjižica 8) (1125)

Knjižnica Sigma:

- Grasselli J., Diofantski približki (1138)
- Prijatelj N., Logika I (4. natis) (1130)
- Strnad J., Mala kvantna fizika, 2. natis (1132)

Izbrana poglavja iz matematike in računalništva:

- Hladnik M., Povabilo v harmonično analizo (1095)
- Batagelj V., Klavžar S., **DS 2** Algebra in teorija grafov. Naloge (1096)
- Dobovišek M., Hladnik M., Omladič M., Rešene naloge iz Analize I, 9. natis (1129)
- Juvan M., Lokar M., 121 nalog iz pascala (1128)

Zbirka izbranih poglavij iz fizike:

- Pahor J., Ponikvar D., Elektronski praktikum za fizike, 2. natis (1098)
- Likar A., Osnove fizikalnih merjenj in merilnih sistemov (1104)
- Likar A., Naloge iz fizikalnih merjenj in merilnih sistemov (1114)

Fizika – Matematika:

- Strnad J., Fizika, 2. del, Elektrika, Optika, 4. natis (1100)

- Strnad J., Fizika, 3. del, Posebna teorija relativnosti, Kvantna fizika, Atomi, 4. natis (1112)

Matematika – Fizika:

- Jamnik R., Matematika, 5. natis (1109)
- Prijatelj N., Osnove matematične logike, 2. del, Formalizacija (1113)

Seminar za računalniško matematiko:

- Seminar za računalniško matematiko 600 (1120)
- Legat D., Modeliranje sistemov na osnovi teorije množične strežbe (1102)

Tekmovanja (Bilteni):

- 36. republiško tekmovanje iz matematike SŠ
- 30. republiško tekmovanje iz fizike SŠ
- 7. republiško tekmovanje iz logike SŠ
- 11. predtekmovanje iz fizike – Koper OŠ
- 12. republiško tekmovanje iz fizike OŠ

Poslovanje komisije za tisk se je v letošnjem letu prilagajalo novim predpisom in zakonom. Med drugim smo morali začeti plačevati tudi davek na dobiček, 5% prometni davek in preiti na dvostransko knjigovodstvo. Marca je odšla v pokoj gospa Marija Hrovath, ki je vodila našo prodajalno. Na vseh področjih se je bolj ali manj zapletlo tudi delo z g. Cirilom Velkovrhom. Konec junija mu je potekla reelekcija kot vodji centra za strokovni tisk na Oddelku za matematiko in mehaniko pri FNT. Ponovno na to mesto ni bi izvoljen. Rešitev smo v sodelovanju z Oddelkom za matematiko in upravnim odborom DMFA našli v upokojitvi ob dokupu let. Ob polletju je odstopil blagajnik prof. Markelj. Vodenje knjig smo prepustili za to specializirani firmi.

Na tem mestu bi se vsem trem rad zahvalil za ves vloženi trud, entuziazem in potrpljenje, ki so ga vložili v svoje delo.

O finančnem poslovanju napišimo le naslednje. Suhoparne številke ob letni bilanci povedo naslednje: Skupni prihodki so bili 20.916.945,50 SIT od tega subvencije 4.911.336,10 SIT. Odhodkov pa je bilo za 20.879.294,30 SIT.

Mirko Dobovišek

NOVI ČLANI DMFA SLOVENIJE V LETU 1992

Lani se je v društvo matematikov, fizikov in astronomov Slovenije včlanilo 31 novih članov, izstopilo pa je 30 članov. Novi člani društva so:

1600. Andraž Aber
1601. Dušan Babič
1602. Jaka Bonča
1603. Marko Bračko
1604. Klavdija Brecelj
1605. Angel Česnik
1606. Marija Dornik
1607. Hubert Fröhlich
1608. Monique Gradolato
1609. Franci Jerič
1610. Veronika Jerše
1611. Andrej Kobe
1612. Žiga Kranjec
1613. Živka Krmelj
1614. Marija Kšela
1615. Matej Leskovar
1616. Jani Melik
1617. Mičo Mrkaič
1618. Matej Orešič
1619. Andrej Pangeršič
1620. Boštjan Podobnik
1621. Andrej Podpečan
1622. Vlado Pogač
1623. Jasna Prezelj
1624. Irena Remec
1625. Marko Robnik
1626. Robert Sraka
1627. Marko Škriba
1628. Matjaž Šlibar
1629. Karmen Špacapan
1630. Primož Ziherl