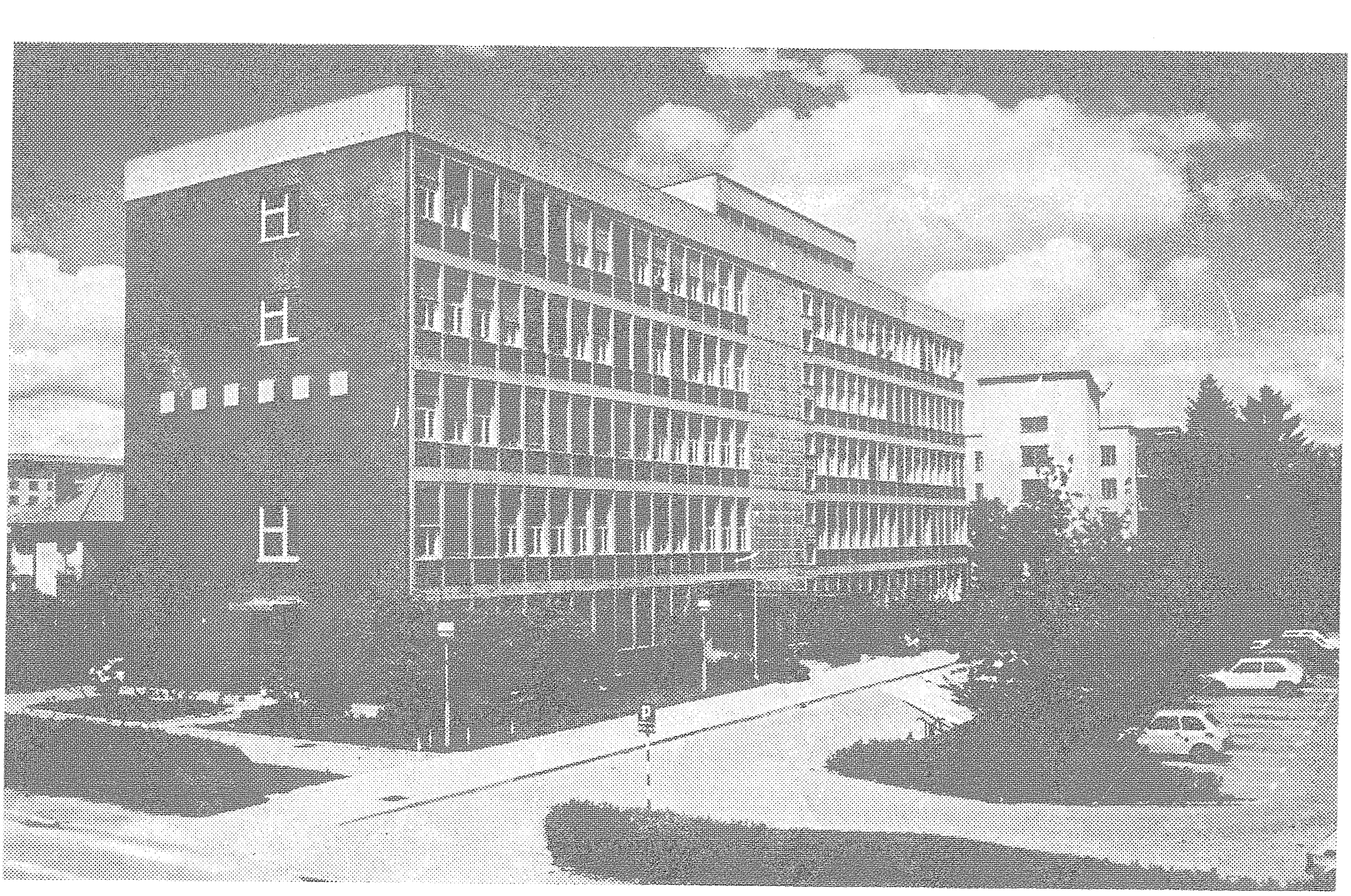
IZDAJA DRUŠTVO MATEMATIKOV, FIZIKOV IN ASTRONOMOV SR SLOVENIJE

YU ISSN 0473-7466

1984
Letnik 31
2

OBZORNIK ZA MATEMATIKO IN FIZIKO



LETNIK 31 — ŠTEVILKA 2

OBZORNIK ZA MATEMATIKO IN FIZIKO

LJUBLJANA, MAREC 1984

Uredniški odbor: Edvard Kramar — glavni urednik, Janez Strnad — urednik za fiziko, Gabrijel Tomšič — odgovorni urednik, Ciril Velkovrh — urednik, Jože Vrabec — urednik za matematiko.

Odbor svetovalcev: Robert Blinc, Alojz Kodre, Peter Legiša, Anton Moljk, Mitja Rosina, Tomaž Skulj, Anton Suhadolc, Ivan Vidav.

Jezikovni pregled Marija Janežič, slike Miha Štalec.

Naročnina: za posameznike 450.— din (za člane društva je že vračunana članarina), za dijake in študente 225.— din, za ustanove in podjetja 900.— din, za tujino 20 $, posamezna številka 150.— din, dvojna številka 300.— din.

Dopise pošiljajte in list naročajte na naslov: Društvo matematikov, fizikov in astronomov SRS — Podružnica Ljubljana — Obzornik za matematiko in fiziko, 61111 Ljubljana, Jadranska c. 19, p.p. 64, tel. št. (061) 265-061/53, žiro račun 50101-678-47233, devizni račun pri Ljubljanski banki št. 50100-620-107-257300-5694/4.

Tisk: Tiskarna Ljudske pravice v Ljubljani. Naklada 1500 izvodov. Izdajo revije sofinancirata ISS in RSS.

 Poštnina plačana na pošti 61102 Ljubljana

NAVODILO AVTORJEM ZA PRIPRAVO ROKOPISA

Rokopis mora biti natipkan v dveh izvodih (drugi izvod je lahko kseroks kopija) na belem papirju formata A-4, z dvojnim razmikom in vsaj 2 cm širokim robom na vseh štirih straneh. V tekstu morajo biti vse besede, ki naj bodo postavljene kurzivno, in vsi matematični simboli podčrtani z valovito črto. Besede in simboli, ki morajo biti stavljeni polkrepko, pa podčrtani z ravno črto. Podrobnejša navodila so v Obzorniku mat. fiz. **21** (1974) 62—64. Pri korekturah na krtačnih odtisih uporabljajte dogovorjene oznake.

VSEBINA — CONTENTS

Na ovitku: Zgradba Inštituta za matematiko, fiziko in mehaniko, kjer imajo svoje prostore tudi VTOZD Fizika ter VTOZD Matematika in mehanika FNT. (Foto Marjan Smerke)

Obzornik mat. fiz. 31 (1984) 2

NUMERIČNE METODE

ZVONIMIR BOHTE

Math. Subj. Class. (1980): 65—01, 65 H 05, 65 H 10

Članek obravnava pregled snovi, ki je predvidena za poglavje Numerične metode v okviru predmeta uporabna matematika v 4. letniku vzgojno izobraževalnega programa: naravoslovno-matematična tehnologija.

NUMERICAL METHODS

This is a survey of the material planned for the chapter Numerical Methods as part of the subject Applied Mathematics in the fourth year of the educational programme "naravoslovno-matematična tehnologija".

Uvod

V vzgojno-izobraževalnem programu: naravoslovno-matematična tehnologija je med drugimi matematičnimi predmeti poseben predmet uporabna matematika, ki ima za cilj seznaniti učence z nekaterimi področji matematike in jih navajati na uporabo pridobljenega znanja pri reševanju matematičnih problemov in problemov iz drugih strok. Ob tem naj bi si učenci razvijali sposobnost in spretnost za delo in spoznavali nekatere naloge, na katere utegnejo naleteti po končani šoli.

Ta predmet obsega poglavja

— metode pri reševanju problemov (15 ur)
— geometrija (35 ur)
— teorija grafov (20 ur)
— teorija števil (20 ur)
— numerične metode (20 ur)
— statistika (20 ur)
— matematični praktikum (28 ur)

Prva tri poglavja spadajo v 3. letnik, preostala štiri pa v 4. letnik.

V tem članku bomo na kratko pregledali vsebino poglavja Numerične metode, ki je v učnem načrtu začrtana takole:

Numerični postopki in napake pri numeričnem računanju. Numerične metode za računanje ničel funkcij: navadna iteracija, sekantna in tangentna metoda. Uporaba pri analizi funkcij.

Dodano je pojasnilo:

Učence navajamo na računske spretnosti in jih seznanimo s preprostejšimi numeričnimi postopki ter ob njih s pojmi: pogojenost problema, konvergenca metode in stabilnost računskega postopka. S primeri jim poskušamo privzgojiti občutek za natančnost in kritičnost pri numeričnem računanju.

Nekaj osnovnih pojmov o približkih in napakah učenci spoznajo že v 1. letniku, nekaj postopkov pa v 3. letniku (npr. Hornerjevo metodo). Seveda je bistveni del potrebnega znanja za to poglavje tudi pojem limite in odvoda ter nekaj osnovnih uporab infinitezimalnega računa.

Za uspešno obvladanje tega poglavja je treba imeti na razpolago vsaj preprost žepni kalkulator, zaželen pa je seveda dostop do računalnika z nekaj spomina in možnostjo programiranja.

Oglejmo si sedaj na kratko osnovne razdelke v pregledni obliki. Obširnejši tekst bo izšel kot posebna publikacija.

1. Napake pri numeričnem računanju

1.1 *Numerična matematika*

Numerična matematika je veja matematike, ki razvija metode za numerično reševanje matematičnih problemov. Med poglavji numerične matematike so tudi taka, ki so v zvezi z elementarno matematiko in osnovami višje matematike: numerično reševanje sistemov linearnih enačb, numerično reševanje algebraičnih in transcendentnih enačb, interpolacija ter numerično odvajanje in integriranje.

Pri numeričnem reševanju kakega problema izhajamo iz številskih podatkov, nato pa s končnim zaporedjem elementarnih operacij izračunamo rezultat v številski obliki. Elementarne operacije so navadno osnovne štiri aritmetične operacije in v računalnik vgrajene funkcije. Rezultati so praviloma le približki, zato je bistveni del numeričnega računanja tudi ocenjevanje napak v rezultatih.

Numerične metode uporabljamo predvsem takrat, kadar drugih metod ni, npr. pri algebraičnih enačbah višjih stopenj ($x^5 - 5x + 1 = 0$), za reševanje transcendentnih enačb ($x = \operatorname{tg} x$) ali za računanje integralov $\left(I = \int_0^1 e^{x^2}\, dx\right)$.

Včasih pa so numerične metode udobnejše od analitičnih, kot npr. pri enačbi $x^3 - 5x + 1 = 0$, ki jo sicer lahko rešimo s Cardanovimi formulami [2], a pri tem nastopi operacija tretji koren iz kompleksnega števila, ki ni elementarna. Včasih pa je analitična rešitev taka, da iz nje brez dodatnih transformacij ne moremo izračunati rešitve dovolj natančno. Zgled za to so integrali

$$I_n = \int_0^1 x^n e^{-x}\, dx, \qquad n = 0,\ 1,\ 2,\ \ldots$$

Z integracijo per partes dobimo formulo

$$I_n = e^{-1}\, n! \left(e - \sum_{k=0}^{n} 1/k!\right)$$

ki je v tej obliki neuporabna za količkaj velike n ali za nekoliko večjo natančnost.

1.2 *Izvori napak*

Numerično reševanje kakega matematičnega problema lahko razumemo kot računanje vrednosti neke funkcije

$$F : X \to Y$$

pri danem argumentu x iz X. Pri tem sta navadno X in Y normirana linearna prostora nad obsegom realnih števil. Brez škode za razumevanje si lahko

predstavljamo, da gre za računanje vrednosti dane funkcije ene spremenljivke, npr. funkcije $F(x) = \sin x$. Tedaj je npr. X kar interval $[0, \pi/4]$, Y pa interval $[0, \sqrt{2}/2]$.

Namesto prave vrednosti

$$y = F(x)$$

bomo izračunali njen približek y^*, ki se v splošnem razlikuje od y. Razliko

$$D = y - y^*$$

imenujemo *celotna napaka* vrednosti y^*. Praviloma se napaki ni mogoče izogniti. Res pa je tudi, da v praksi ne potrebujemo točnih rezultatov, saj jih lahko realiziramo le z določeno natančnostjo.

Izvore napak, ki nastanejo pri numeričnem računanju, delimo na tri skupine.

a) Podatki

Podatki za numerično reševanje so navadno dobljeni z meritvami ali pa so izračunani s predhodnimi računi. Tudi pri čitanju podatkov v računalnik pride do zamenjave podatkov z njihovimi približki v okviru osnovne natančnosti računalnika. Pri računanju zato ne poznamo točnega podatka x, ampak le njegov približek $\bar{x}$. Zato lahko poizkusimo izračunati le

$$\bar{y} = F(\bar{x})$$

Razliko

$$D_n = y - \bar{y}$$

imenujemo *neodstranljiva napaka*, saj računar navadno ne more vplivati na natančnost podatkov. Problem, pri katerem majhne spremembe podatkov povzročijo vedno le majhne spremembe rezultatov, je *neobčutljiv*. Če pa se lahko rezultati pri majhni spremembi podatkov močno spremenijo, je problem *občutljiv*. Stopnjo občutljivosti merimo z razmerjem med velikostjo neodstranljive napake in velikostjo napake v podatkih. Oceno za to razmerje imenujemo *občutljivost* problema.

Primeri neobčutljivih problemov so vrednost polinoma, vrednost zvezno odvedljive funkcije z absolutno majhnim odvodom, dobro separirana korena kvadratne enačbe, presečišče dveh skoraj pravokotnih premic, integral zvezne funkcije.

Primeri občutljivih problemov so vrednost hitro naraščajoče funkcije, dvojna ali bližnja korena kvadratne enačbe, presečišče dveh skoraj vzporednih premic, odvod odvedljive funkcije.

b) Numerična metoda

Pogosto se točna rešitev problema ne da dobiti s končnim številom aritmetičnih operacij, ampak le kot rezultat neskončnega procesa. Pri numeričnem reševanju smo prisiljeni vse neskončne procese nadomestiti s končnimi. Tako npr. limito zaporedja nadomestimo s poznim členom, neskončno vrsto s končno delno vsoto, odvod funkcije s končnim diferenčnim kvocientom, integral s končno vsoto.

Okrnitev neskončnega procesa lahko tolmačimo kot zamenjavo prave funkcije F z neko drugo funkcijo G, ki pa jo je mogoče numerično realizirati, to se pravi, da je mogoče dobiti njeno vrednost s končnim številom elementarnih operacij. Zato lahko izračunamo kvečjemu

$$\bar{\bar{y}} = G(\bar{x})$$

Razliko

$$D_m = \bar{y} - \bar{\bar{y}}$$

imenujemo *napaka metode.* Numerična metoda je *konvergentna,* če lahko dosežemo, da je absolutna vrednost napake metode poljubno majhna, če le dovolj pozno okrnemo ustrezni neskončni proces. Za uporabo metod je važna *hitrost konvergence,* ki vpliva na ekonomičnost metod, to je na računski čas ali ceno računanja.

c) Aritmetika

Vsi digitalni računalniki imajo zmožnost predstaviti le končno množico racionalnih števil. Aritmetične enote teh računalnikov so take, da pri posameznih aritmetičnih operacijah točen rezultat nadomestijo s približkom v množici predstavljivih števil. Zaradi napak pri izvajanju aritmetičnih operacij na posameznih korakih končnega računskega procesa tudi števila $\bar{\bar{y}}$ ne moremo izračunati točno, ampak lahko dobimo le njegov približek y^*, ki je končni izračunani rezultat.

Razliko

$$D_z = \bar{\bar{y}} - y^*$$

imenujemo *zaokrožitvena napaka.* Ta je odvisna od vrstnega reda operacij, od natančnosti računanja in načina zaokroževanja. *Osnovna zaokrožitvena napaka* je maksimalna napaka pri eni aritmetični operaciji. Računski proces imenujemo *stabilen,* če je zaokrožitvena napaka primerno omejena. Če pa so lahko zaokrožitvene napake poljubno velike, pravimo, da je računski proces *nestabilen.* Stopnjo nestabilnosti merimo z razmerjem med velikostjo zaokrožitvene napake in osnovno zaokrožitveno napako. Oceno za to razmerje imenujemo *nestabilnost* računskega procesa.

Primeri stabilnih računskih procesov so Hornerjev postopek za računanje vrednosti polinoma, seštevanje pozitivnih števil, dvočlenska rekurzivna relacija v pravi smeri.

Primeri nestabilnih računskih procesov so odštevanje približno enakih pozitivnih števil, dvočlenska rekurzivna relacija v napačni smeri.

Očitno velja med naštetimi napakami zveza

$$D = D_n + D_m + D_z$$

Celotna napaka je torej vsota posameznih napak. Pri tem se posamezne napake lahko med seboj delno ali v celoti uničijo. Za ocene pa velja

$$|D| \leqq |D_n| + |D_m| + |D_z|$$

Ocena za celotno napako je vsota absolutnih vrednosti posameznih napak, ali še več, vsota ocen za posamezne napake. Ocene se torej vedno le seštevajo. Zato je ocena za celotno napako dostikrat pesimistična.

Ponavadi izbiramo natančnost računanja tako, da so vsi trije prispevki v oceni celotne napake približno istega velikostnega reda.

Pri numeričnem reševanju se torej trudimo, da

— formuliramo probleme tako, da se izognemo veliki občutljivosti,

— izbiramo ekonomične metode, da se izognemo prevelikim stroškom ali preveliki izgubi časa in

— oblikujemo stabilne računske procese, da se izognemo prevelikim zaokrožitvenim napakam.

Vaja: Izračunaj vrednost funkcije $y = \sin x$ pri argumentu $x = \pi/10$ iz dveh členov Taylorjeve vrste in oceni celotno napako.

1.3 *Primeri računskih nalog iz elementarne matematike*

V elementarni matematiki je dosti nalog, ki zahtevajo numerično računanje pri realističnih (nešolskih) podatkih. Pri tem je treba učence opozarjati na kritičnost pri numeričnem računanju.

Naštejmo nekatere take naloge.

a) Linearna interpolacija

S prodorom sodobnih računalnikov v vsakdanje življenje je uporaba matematičnih tabel pri numeričnem računanju zastarela. Kljub temu pa je prav, da učenci že iz zgodovinskih razlogov spoznajo logaritme kot računski pripomoček in uporabo tabel.

Pri interpolaciji z linearno funkcijo izpeljemo najprej izraz za napako metode

$$f(x) - I(x) = f''(\xi)\,(x - x_k)\,(x - x_{k+1})/2$$

Tu je $f(x)$ tabelirana funkcija, za katero predpostavljamo dvakratno zvezno odvedljivost, $I(x)$ polinom prve stopnje, ki gre skozi točki $(x_k, f(x_k))$ in $(x_{k+1}, f(x_{k+1}))$, število ξ pa za vsak x med x_k in x_{k+1} tudi leži tam nekje vmes. Če ne poznamo pravih funkcijskih vrednosti $f(x_k)$, ampak le njihove približke y_k, potem nastane v interpolirani vrednosti neodstranljiva napaka, ki za noben x med x_k in x_{k+1} ne preseže večje od napak v vrednostih y_k in y_{k+1}. Tudi zaokrožitveno napako pri računanju vrednosti interpolacijskega polinoma lahko hitro ocenimo. Tako lahko dobimo oceno za celotno napako pri linearni interpolaciji. Na tej osnovi lahko tudi ocenjujemo razmik argumentov v tabeli, ki opravičuje uporabo linearne interpolacije. Vsa ta vprašanja so podrobno obdelana v [4].

b) Kvadratna enačba

Pri kvadratni enačbi lahko najprej analiziramo neodstranljivo napako v obeh korenih. Pri tem ugotovimo, da so dvojni koreni in koreni, ki so blizu skupaj, zelo občutljivi. To lahko podkrepimo s primeri tipa

$$(x - 1)^2 - \varepsilon = 0$$

in

$$(1/\sqrt{2})\,x^2 - (2/\sqrt{2})\,x + 1/\sqrt{2} = 0$$

Po drugi strani pa lahko analiziramo stabilnost računanja obeh korenov enačbe $ax^2 + bx + c = 0$ po formulah

$$x_1 = (-b + \sqrt{b^2 - 4ac})/(2a),\quad x_2 = (-b - \sqrt{b^2 - 4ac})/(2a)$$

Če je število $4ac$ po absolutni vrednosti znatno manjše od b^2, je priporočljiva samo tista formula, v kateri se prispevka v številcu seštejeta. Druga da namreč rezultat z veliko relativno napako. Zato moramo drugi koren izračunati iz enačbe $x_1 x_2 = c/a$.

Podobna težava nastopi pri računanju kvadratnega korena iz kompleksnega števila. Oboje je podrobno obdelano v [4].

c) Računanje skoraj nedoločenih izrazov

Odštevanje skoraj enakih približkov vedno vodi do velike relativne napake. Zato se je treba temu izogniti s predelavo izraza v drugačno obliko. Naštejmo nekaj takih primerov.

$$(a^n - b^n)/(a - b) = a^{n-1} + a^{n-2}\,b + \ldots + ab^{n-2} + b^{n-1} \text{ za } a \doteq b$$

$$(1 - \cos x)/x^2 = (\sin (x/2)/(x/2))^2/2 \text{ za } x \doteq 0$$

$$\sqrt{x+1} - \sqrt{x} = 1/(\sqrt{x+1} + \sqrt{x}) \quad \text{za} \quad x \gg 1$$

$$e - \sum_{k=0}^{n} 1/k! = \sum_{k=n+1}^{\infty} 1/k! \quad \text{za} \quad n \gg 1$$

d) Naloge iz trigonometrije

Naloge iz trigonometrije dajejo obilo priložnosti za privzgojitev občutka za numerično stabilno računanje. Naštejmo samo nekaj primerov.

i) Dani sta števili a in b. Izračunaj $\sin x$ in $\cos x$, če je $\operatorname{tg} x = a/b$.

ii) Dani sta števili a in b. Izračunaj $\sin x$ in $\cos x$, če je $\operatorname{tg}(2x) = a/b$.

iii) V trikotniku so dane vse tri stranice. Izračunaj vse druge količine.

iv) V trikotniku sta dana dva kota in ena stranica. Izračunaj vse druge količine.

v) V trikotniku sta dani dve stranici in kot, ki ga oklepata. Izračunaj vse druge količine.

vi) V trikotniku sta dani dve stranici in kot, ki leži večji stranici nasproti. Izračunaj vse druge količine.

2. Računanje ničel funkcij

To poglavje numerične matematike je najbolj elementarno, zato je primeren uvod za numerične metode. Nekoliko je obdelano v [4], več o tem piše v [5], podrobno pa je obravnavano v [1].

Naloga je poiskati enega ali več korenov enačbe

$$f(x) = 0 \tag{1}$$

kjer je $f(x)$ zvezna funkcija, pogosto tudi nekajkrat zvezno odvedljiva, na kakem končnem intervalu, včasih kar na celi realni osi.

Osnovni izrek o zveznih funkcijah nam omogoča ugotoviti, da leži na intervalu (a, b) vsaj en koren enačbe (1), če je $f(a)\,f(b) < 0$. Dokaz tega izreka z bisekcijo [8] je konstruktiven in to imamo lahko za najpreprostejšo metodo.

2.1 *Bisekcija*

Naj velja $AB < 0$, kjer je $f(a) = A$ in $f(b) = B$. Vzemimo, da želimo izračunati enega od korenov, ki ležijo na intervalu (a, b) z natančnostjo e. Metodo bisekcije najlaže opišemo z algoritmom.

1. Izračunaj $c = (a + b)/2$.
2. Če je $b - a < e$, postavi koren $= c$, stoj.
3. Izračunaj $C = f(c)$.
4. Če je $C = 0$, postavi koren $= c$, stoj.
5. Če je $AC < 0$, postavi $b = c$, $B = C$, pojdi na 1.
6. Postavi $a = c$, $A = C$, pojdi na 1.

Bisekcija je geometrijsko nazorna, numerično izredno stabilna, deluje v vseh primerih, morda je le premalo ekonomična. Če je npr. $b - a = 1$ in $e = 10^{-9}$, moramo narediti približno 30 korakov, torej izračunati 30 funkcijskih vrednosti.

2.2 *Sekantna metoda*

Sekantna metoda je le neznatna modifikacija bisekcije. Pri tej metodi tudi izhajamo iz pogoja $f(a)\,f(b) < 0$, le da za naslednji približek c ne vzamemo središča intervala (a, b), ampak presečišče sekante skozi točki $(a, f(a))$ in $(b, f(b))$ z osjo (x). Postopek nadaljujemo s tistim od obeh podintervalov (a, c) ali (c, b), v krajiščih katerega ima funkcija spet nasproten predznak. Zaključek je nekoliko drugačen. Približek za napako je širina manjšega od obeh podintervalov. Algoritem je enak algoritmu za bisekcijo, le prva dva koraka se spremenita:

1. Izračunaj $c = a - (b - a)A/(B - A)$.
2. Če je $\min(b - c,\ c - a) < e$, postavi koren $= c$, stoj.

Tudi sekantna metoda je geometrično nazorna, numerično stabilna in univerzalna. V primeri z bisekcijo je navadno bolj ekonomična, čeprav se dajo skonstruirati primeri, ko je bisekcija hitrejša.

2.3 *Navadna iteracija*

Enačbo (1) rešujemo z navadno iteracijo tako, da jo zapišemo v ekvivaletni obliki

$$x = g(x) \tag{2}$$

si izberemo začetni približek x_0 in nato računamo naslednje približke po pravilu

$$x_{r+1} = g(x_r),\ r = 0, 1, \ldots \tag{3}$$

Velja preprost izrek o konvergenci zaporedja približkov.

Če je α rešitev enačbe (2) in je na nekem intervalu $I = [\alpha - a, \alpha + a]$ okrog rešitve

$$|g'(x)| \leqq m < 1$$

potem zaporedje približkov (3) konvergira k rešitvi α za vsak začetni približek x_0 iz I. Koren α je edini na tem intervalu.

Čim manjši je m, tem hitrejša je konvergenca. Velja ocena za napako

$$|x_r - \alpha| \leqq m|x_r - x_{r-1}|/(1 - m)$$

Primer. Enačbo

$$x + \ln x = 0 \tag{4}$$

ki ima en sam realni koren v bližini števila 0·5, lahko zapišemo v oblikah

$$x = -\ln x$$
$$x = e^{-x}$$
$$x = (x + e^{-x})/2$$
$$x = (x + 2e^{-x})/3$$
$$x = (x^2 + e^{-x})/(1 + x)$$

Prva oblika sploh ni primerna, vse druge pa so. Vsaka naslednja da koren z manjšim številom iteracij. Zadnja ima lastnost, da je odvod iterativne funkcije $g(x)$ v rešitvi α enak 0.

2.4 *Tangentna metoda*

Tangentna metoda za reševanje enačbe (1) je poseben primer navadne iteracije, če vzamemo za iterativno funkcijo

$$g(x) = x - f(x)/f'(x)$$

Zaporedje približkov torej računamo po formulah

$$x_{r+1} = x_r - f(x_r)/f'(x_r),\ r = 0, 1, \ldots \tag{5}$$

Približek x_{r+1} pomeni presečišče osi (x) in tangente na krivuljo $y = f(x)$ v točki $(x_r, f(x_r))$. Če je α enostaven koren ($f(\alpha) = 0$, $f'(\alpha) \neq 0$), potem je $g'(\alpha) = 0$. Zato zaporedje približkov zelo hitro konvergira.

Pri enačbi (4) je vseeno, ali vzamemo za $f(x)$ funkcijo $x + \ln x$, ki da zaporedje

$$x_{r+1} = x_r - (x_r + \ln x_r)/(1 + 1/x_r) = x_r(1 - \ln x_r)/(1 + x_r)$$

ali pa $f(x) = x - e^{-x}$, ki da zaporedje

$$x_{r+1} = x_r - (x_r - e^{-x_r})/(1 + e^{-x_r}) = (1 + x_r)/(1 + e^{x_r})$$

torej nam ni treba ugibati, v katero ekvivalentno obliko je pametno enačbo predelati.

V bližini večkratnega korena je konvergenca tangentne metode počasna, daleč od korena pa zelo negotova.

Da se dokazati tale izrek o konvergenci tangentne metode.

Naj bo $f(x)$ dvakrat zvezno odvedljiva funkcija na intervalu $[a, b]$, naj bo $f(a)\,f(b) < 0$, $f'(x)$ in $f''(x)$ naj bosta na tem intervalu povsod različna od nič, x_0 pa naj bo tisto krajišče, v katerem ima funkcijska vrednost isti znak kot drugi odvod. Potem zaporedje (5) konvergira k rešitvi α.

Če le moremo, uporabimo za reševanje enačbe (1) tangentno metodo. Kadar pa bi neradi računali odvod funkcije $f(x)$, lahko tangentno metodo preoblikujemo

$$x_{r+1} = x_r - f(x_r)\,(x_r - x_{r-1})/(f(x_r) - f(x_{r-1})),\ r = 1, 2, \ldots \tag{6}$$

Tu smo odvod $f'(x_r)$ nadomestili z diferenčnim kvocientom

$$f'(x_r) \doteq (f(x_r) - f(x_{r-1}))/(x_r - x_{r-1})$$

Naslednji približek x_{r+1} je presečišče sekante skozi točki $(x_r, f(x_r))$ in $(x_{r-1}, f(x_{r-1}))$ z osjo (x). Zato je metoda (6) tudi varianta sekantne metode.

Do dobrih začetnih približkov pridemo navadno s tabeliranjem funkcije $f(x)$.

3. Reševanje sistema dveh nelinearnih enačb

Za reševanje sistema dveh nelinearnih enačb

$$\begin{aligned} f(x, y) &= 0 \\ g(x, y) &= 0 \end{aligned} \tag{7}$$

kjer sta $f(x, y)$ in $g(x, y)$ zvezni in odvedljivi funkciji dveh spremenljivk, imamo na razpolago podobne metode kot za reševanje ene enačbe z eno neznanko.

Vsaka od enačb implicitno določa y kot funkcijo spremenljivke x. Če približno narišemo grafa obeh funkcij, lahko vidimo, koliko je realnih rešitev, in odčitamo zanje začetne približke.

Primer. Rešitve sistema

$$x^2 + y^2 = 2$$
$$x^2 - 2xy + y = 1$$

so štiri presečišča med krogom in hiperbolo.

3.1 *Navadna iteracija*

Če sistem (7) zapišemo v ekvivalentni obliki

$$x = F(x, y)$$
$$y = G(x, y)$$

lahko računamo zaporedje približkov po pravilu

$$x_{r+1} = F(x_r, y_r)$$
$$y_{r+1} = G(x_r, y_r) \qquad r = 0, 1, 2, \ldots$$

To zaporedje konvergira k rešitvi (α, β), če so vsi štirje parcialni odvodi F_x, F_y, G_x, G_y po absolutni vrednosti v okolici rešitve majhni; zadošča namreč, da so manjši od 1/2.

3.2 *Newtonova metoda*

Posplošitev tangentne metode za dve enačbi je Newtonova metoda. Če namreč v primeru ene enačbe (1) razvijemo funkcijo $f(x)$ okrog tekočega približka x_r v Taylorjevo vrsto in zanemarimo člene od drugega dalje, dobimo ravno tangentno metodo:

$$f(x) = f(x_r) + (x - x_r) f'(x_r) + \ldots = 0$$

Rešitev ustrezne linearne enačbe je naslednji približek x_{r+1}.

Pri dveh enačbah ravnamo podobno:

$$f(x, y) = f(x_r, y_r) + (x - x_r) f_x(x_r, y_r) + (y - y_r) f_y(x_r, y_r) + \ldots = 0$$
$$g(x, y) = g(x_r, y_r) + (x - x_r) g_x(x_r, y_r) + (y - y_r) g_y(x_r, y_r) + \ldots = 0$$

Naslednji približek dobimo z rešitvijo sistema dveh linearnih enačb

$$f_x(x_r, y_r)\,\Delta x_r + f_y(x_r, y_r)\,\Delta y_r = -f(x_r, y_r)$$
$$g_x(x_r, y_r)\,\Delta x_r + g_y(x_r, y_r)\,\Delta y_r = -g(x_r, y_r) \qquad (8)$$
$$x_{r+1} = y_r + \Delta x_r$$
$$y_{r+1} = y_r + \Delta y_r$$

Konvergenca je v bližini enostavnih rešitev zelo hitra. Če pa začnemo daleč od rešitve, konvergenca približkov v splošnem ni zagotovljena.

4. Reševanje algebraičnih enačb

Vse naštete metode lahko uporabimo tudi za reševanje algebraične enačbe

$$p(x) = 0 \tag{9}$$

kjer je $p(x)$ polinom stopnje n

$$p(x) = a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x + a_n$$

Vzemimo, da so koeficienti a_i realna števila in da je $a_0a_n \neq 0$. Potem vemo, da ima enačba (9) natanko n realnih ali kompleksnih od nič različnih korenov, če jih štejemo po večkratnosti. Kompleksni koreni nastopajo v konjugiranih parih.

4.1 *Računanje realnih korenov*

Če si polinom $p(x)$ grafično predočimo, lahko dobimo vtis o številu realnih korenov in začetne približke zanje. Pri večjih stopnjah n je to kar nerodna naloga. Zato si dostikrat pomagamo kar s slepo izbiro začetnih približkov. Če vemo, da ima enačba same realne korene, je naloga lažja, saj lahko dobimo za začetni približek število, ki je večje od največjega korena. Vzemimo, da je $a_0 > 0$. Če ima polinom $p(x)$ samo nenegativne koeficiente, potem enačba (9) nima pozitivnih korenov in je število 0 zgornja meja realnih korenov. Če pa ima polinom $p(x)$ nekaj negativnih koeficientov in je a_k prvi izmed njih, A pa absolutna vrednost najmanjšega izmed njih, je zgornja meja realnih korenov število (glej [7])

$$M = 1 + (A/a_0)^{1/k} \tag{10}$$

Zaporedje približkov po tangentni metodi (5) z začetnim približkom (10) v primeru samih realnih korenov vedno konvergira k največjemu korenu.

Vrednost polinoma in njegovega odvoda pri realnem argumentu računamo po Hornerjevem postopku [6]. Ta postopek nam dostikrat daje tudi mejo realnih korenov. Če računamo vrednost polinoma $p(x)$ pri $x = c$ in dobimo v tretji vrsti Hornerjeve sheme sama nenegativna števila, je c meja pozitivnih korenov. S transformacijama $y = -x$ in $y = 1/x$ pa si pomagamo pri ugotavljanju spodnjih mej za negativne in pozitivne korene.

4.2 *Izločitev realnih korenov*

Naj bo a približek za realni koren enačbe (9). Če želimo izračunati še kak drug koren te enačbe, se nam lahko zgodi, da novo zaporedje skonvergira k istemu korenu, čeprav smo začeli z drugim začetnim približkom. Zato je dobro izločiti vpliv že izračunanega korena. Naravno je, da polinom $p(x)$ delimo z binomom $x - a$

$$p(x) = (x - a)\, q(x) + R$$

Ostanek R je enak $p(a)$, zato ga zanemarimo. Kvocient $q(x)$ je polinom stopnje $n - 1$. Rešitve enačbe

$$q(x) = 0$$

so približki za preostalih $n - 1$ korenov enačbe (9). Deljenje opravimo s Hornerjevim postopkom.

Izkaže se, da je redukcija enačbe numerično stabilna, če izločimo koren z najmanjšo absolutno vrednostjo.

4.3 *Računanje kompleksnih korenov*

Tangentno metodo moremo uporabiti tudi s kompleksnim začetnim približkom:

$$z_{r+1} = z_r - p(z_r)/p'(z_r), \quad r = 0, 1, 2, \ldots$$

Vrednost polinoma in njegovega odvoda pri kompleksnem argumentu računamo z razširjeno Hornerjevo metodo [6].

Kompleksni aritmetiki se izognemo, če iterativno računamo približek za trinom $x^2 + px + q$, ki ima za korena iskani konjugirano kompleksni par. Metoda Bairstowa in Hitchcocka je ekvivalent Newtonove metode za sistem dveh nelinearnih enačb. Podrobno je razložena v [1].

4.4 *Izločitev kompleksnih korenov*

Če smo izračunali približek $a + ib$ za kompleksni koren, potem delimo polinom $p(x)$ s trinomom $r(x) = x^2 - 2ax + (a^2 + b^2)$:

$$p(x) = r(x)\, q(x) + Sx + T$$

Koeficienta S in T zanemarimo in naprej rešujemo enačbo

$$q(x) = 0$$

ki je stopnje $n - 2$.

S primerno povezavo vseh opisanih postopkov lahko v principu izračunamo vse realne in kompleksne korene algebraične enačbe.

Zavedati se moramo, da so večkratni in bližnji koreni vedno zelo občutljivi in da so pri enačbah visokih stopenj lahko tudi dobro separirani koreni takšni.

5. Uporaba pri analizi funkcij

Navedene metode za računanje ničel funkcij lahko uporabljamo pri analizi funkcij, risanju njihovih grafov, računanju ekstremov, prevojev in drugih posebnih točk. Numerične metode pridejo še posebej do izraza pri analizi funkcije $y = y(x)$, ki je implicitno podana z enačbo

$$f(x, y) = 0$$

Zdaj moramo že za računanje vrednosti implicitne funkcije rešiti nelinearno enačbo. Tu lahko s pridom izkoristimo korene, ki smo jih izračunali pri nekem x za začetne približke korenov pri sosednjem x.

LITERATURA

[1] Z. Bohte, *Numerično reševanje enačb*, DZS, Ljubljana 1974.

[2] Z. Bohte, *Numerična matematika*, OMF **19** (1972), 97—102.

[3] Z. Bohte, *Analiza napak*, OMF **19** (1972), 103—107.

[4] Z. Bohte, *Pregled elementarne matematike s stališča numeričnega računanja*, OMF **19** (1972), 108—127.

[5] Z. Bohte, *Numerične metode*, DZS, Ljubljana 1978.

[6] Z. Bohte, *Uporaba in posplošitve Hornerjevega algoritma*, OMF **26** (1979), 129—140.

[7] F. Križanič, *Aritmetika, algebra in analiza za gimnazije*, III. del, DZS, Ljubljana 1975.

[8] I. Vidav, *Višja matematika I*, DZS, Ljubljana 1973.

Obzornik mat. fiz. **31** (1984) 2

RAČUNANJE RAZMERJA MED OBSEGOM IN PREMEROM KROGA

TOMAŽ PISANSKI

Math. Subj. Class. (1980) 68 — 01, 65 — 01

Obravnavamo nekaj metod za računanje decimalk števila π, predvsem Arhimedovo metodo s pravilnimi mnogokotniki in metode, ki temeljijo na razvoju funkcije arc tg (x) v potenčno vrsto. Posebej omenimo prispevek slovenskega matematika Jurija Vege, ki je svojčas prvi izračunal 137 decimalk števila π — ne zato, ker je bil bolj potrpežljiv od svojih sodobnikov, ampak ker je imel boljši algoritem.

CALCULATING THE RATIO BETWEEN THE CIRCUMFERENCE AND THE DIAMETER OF A CIRCLE

Methods for calculating decimal digits of π are discussed, mainly the old method of Archimedes using regular polygons and various methods based on series expansion of arc tg (x). The contribution of the Slovene mathematician Jurij Vega is mentioned. He was the first to compute 137 digits of π, not because he worked harder then others but because he had a better algorithm.

Število π je razmerje med obsegom in premerom kroga. Brez dvoma je to eno najbolj zanimivih števil v zgodovini človeštva. Vsak razume, kako ga definiramo. V šoli se že kmalu sprijaznimo z dejstvom, da je π približno 3,14. Če pa bi nas kdo vprašal, kako bi računali π poljubno natančno, bi bili prvi hip verjetno v zadregi. Vendar je že Arhimed poznal metodo (algoritem), ki mu je vsaj v načelu omogočala računati π poljubno natančno.

Preden prikažemo zgodovino računanja decimalk števila π, si oglejmo nekaj kvalitativnih rezultatov.

Eden od znanih »nerešljivih« problemov starih Grkov je problem *kvadrature kroga.* (Glej npr. [5].) Ta problem zahteva, da dani krog preoblikujemo v ploščinsko enak kvadrat. Nerešljiv pa je zaradi dodatne zahteve, da pri tem uporabljamo samo šestilo in ravnilo. Če si izberemo enoto, je problem kvadrature kroga ekvivalenten problemu načrtati daljico dolžine π. Nekatera pozitivna realna števila lahko načrtamo s šestilom in ravnilom.

V moderni obleki bi se problem kvadrature kroga glasil: ali je π mogoče konstruirati s šestilom in ravnilom? Že stari Grki so vedeli, da je mogoče s šestilom in ravnilom konstruirati vsako pozitivno racionalno število. Po drugi strani pa je res, da je število racionalno, če in samo če se mu v desetiškem zapisu decimalke periodično ponavljajo. Zato so z računanjem decimalk števila π upali odkriti tako zakonitost, ki bi morebiti pomagala dokazati, da je π racionalno število, in s tem rešiti problem kvadrature kroga. Leta 1761 je Nemec Lambert te špekulacije pretrgal, saj je dokazal, da je π iracionalno število. S tem pa seveda še ni pokazal, da je problem kvadrature kroga nerešljiv, saj obstaja kopica iracionalnih števil, ki jih lahko konstruiramo samo s šestilom in ravnilom. Mednje sodi na primer $\sqrt{2}$, diagonala kvadrata. Kasneje so matematiki realna števila razdelili na algebraična in transcendentna. Algebraična števila so vsi koreni polinomov s celoštevilskimi koeficienti. Brez težav lahko pokažemo, da s šestilom in ravnilom ne moremo konstruirati nobenega transcendentnega števila. Ko je leta 1882

F. Lindemann dokazal, da je število π transcendentno, je dokončno »rešil« problem kvadrature kroga. Na prvi pogled je po takem negativnem rezultatu računanje decimalk števila π precej nekoristna zadeva. In vendar ni čisto tako. Za število pravimo, da je *normalno*, če se v decimalnem zapisu vsaka decimalka pojavlja tako pogosto kot druge (z verjetnostjo 0,1) in se vsak par zaporednih decimalk pojavlja v povprečju enako mnogokrat (z verjetnostjo 0,01) itd. Zelo težak in doslej še nerešen problem je: ali je π normalno število? Eksperimentalni rezultati zdaj, ko poznamo že več kot milijon decimalk števila π, podpirajo domnevo, da π je normalno število. Zato računanje decimalk števila π ni popolnoma nesmiselno početje. Drug, bolj praktičen problem pa je tale. Poznamo več algoritmov za računanje decimalk števila π. Kateri med njimi je najbolj učinkovit? Ali bolj natančno povedano: če želim izračunati prvih n decimalk števila π, koliko osnovnih operacij moram izvesti? S takimi vprašanji se ukvarja teorija zahtevnosti, ki je del teoretičnega računalništva?

Zdaj pa si bomo ogledali nekaj postopkov za računanje števila π. S približki za π so se ukvarjali matematiki vseh starih kultur. Ohranila so se prizadevanja starih Indijcev, Babiloncev, Egipčanov in Kitajcev.

Načelno lahko krog aproksimiramo z očrtanimi in včrtanimi pravilnimi mnogokotniki. Arhimed je računal obsege včrtanih in očrtanih mnogokotnikov in tako dobival spodnje in zgornje približke za število π. Če krogu s polmerom 1 včrtamo pravilni n-kotnik, je njegov polovični obseg $p(n) = n \sin(\pi/n)$, polovični obseg krogu očrtanega pravilnega n-kotnika pa je $P(n) = n \operatorname{tg}(\pi/n)$. $P(2n)$ in $p(2n)$ lahko izrazimo s $p(n)$ in $P(n)$ takole:

$$P(2n) = \frac{2p(n)\,P(n)}{p(n) + P(n)}, \qquad p(2n) = \sqrt{P(2n)\,p(n)}$$

Torej je $P(2n)$ harmonična sredina števil $p(n)$ in $P(n)$, $p(2n)$ pa je geometrična sredina števil $P(2n)$ in $p(n)$. Definirajmo zaporedji $\{a_m = P(2^m n)\}$ in $\{b_m = p(2^m n)\}$. Prvo monotono pada, drugo pa monotono raste in imata skupno limito π.

Če pozabimo na geometrijo, lahko Arhimedovo metodo opišemo takole. Naj bosta a_0 in b_0 poljubni pozitivni realni števili. Zaporedji a_m in b_m definiramo takole:

$$a_{m+1} = 2a_m b_m/(a_m + b_m) \quad \text{in} \quad b_{m+1} = \sqrt{a_{m+1}\, b_m}$$

Označimo z $A(a_0, b_0)$ skupno limito teh zaporedij. Če vzamemo $a_0 = 6 \,.\, \operatorname{tg}(\pi/6) = 2\sqrt{3}$ in $b_0 = 6 \,.\, \sin(\pi/6) = 3$, tedaj je $A(a_0, b_0) = \pi$.

Tabela 1 prikazuje prvih 15 členov a_m in b_m, kot jih izračunamo z žepnim kalkulatorjem.

Leta 1800 sta Gauss in Pfaff obravnavala pare zaporedij x_n in y_n, ki ustrezajo relacijama:

$$x_{n+1} = (x_n + y_n)/2 \quad \text{in} \quad y_{n+1} = \sqrt{x_{n+1} \cdot y_n}$$

Spet se izkaže, da je prvo zaporedje padajoče in drugo naraščajoče in da imata isto limito.

Neodvisno od Gaussa in Pfaffa je Borchardt leta 1881 uporabil tak par zaporedij za računanje števila π. Če pri poljubno izbranih pozitivnih začetnih členih x_0, y_0 označimo z $B(x_0, y_0)$ skupno limito, zaporedij x_n in y_n in če vzamemo $x_0 = 1/a_0$ in $y_0 = 1/b_0$, brez težav uvidimo, da je $x_n = 1/a_n$ in $y_n = 1/b_n$

m	a_m	b_m
0	3,464 101 616	3,000 000 000
1	3,215 390 310	3,105 828 542
2	3,159 659 943	3,132 628 614
3	3,146 086 216	3,139 350 204
4	3,142 714 600	3,141 031 952
5	3,141 873 050	3,141 452 473
6	3,141 662 747	3,141 557 608
7	3,141 610 176	3,141 583 892
8	3,141 597 034	3,141 590 463
9	3,141 593 748	3,141 592 105
10	3,141 592 926	3,141 592 515
11	3,141 592 721	3,141 592 618
12	3,141 592 669	3,141 592 644
13	3,141 592 656	3,141 592 650
14	3,141 592 653	3,141 592 652
15	3,141 592 652	3,141 592 652

Tabela 1. Računanje števila π z Arhimedovo metodo

za vsak n in je zato tudi $A(a_0, b_0) = 1/B(1/a_0, 1/b_0)$. Posebej velja $\pi = A(2\sqrt{3}, \sqrt{3}) = 1/B(\sqrt{3}/6, \sqrt{3}/3)$.

Borchardtov postopek za računanje števila π je boljši od Arhimedovega, saj za n decimalk porabimo pri obeh isto število korakov, pri enem koraku pa zahteva Borchardtov postopek manj aritmetičnih operacij kot Arhimedov. Kljub vsemu pa sta oba precej zamudna, saj moramo računati kvadratne korene, to pa zahteva precej časa.

Arhimedovo metodo so uporabljali več kot tisoč let mnogi matematiki v najrazličnejših oblikah. Nekateri so namesto obsegov uporabljali ploščine pravilnih mnogokotnikov. Sprva so bili dobljeni približki precej nenatančni. Arhimed je na primer vedel, da je $22/7 = 3{,}143$ prevelik približek za π. Natančneje pa se številu π ni približal, saj Grki še niso poznali pozicijskega zapisa števil. Kitajec Zhang Heng je v prvem stoletju našega štetja našel približek $\sqrt{10} = 3{,}162$ za π. V petem stoletju pa je Zu Chong-Zhi odkril čudoviti ulomek 355/113, ki je π na šest decimalk natančno. Z Arhimedovo metodo so računali Indijanci in Arabci, v Evropi pa na primer Vieta, ki je leta 1579 z $(2^{16} . 6)$-kotnikom izrazil π na devet decimalk, Van Roomen (Romanus), ki je leta 1593 z 2^{30}-kotnikom izračunal 15 decimalk števila π, ter Ludolph Van Ceulen, ki je leta 1610 z 2^{62}-kotnikom izračunal 32 decimalk tega števila. Snell je bil zadnji, ki je računal π z Arhimedovo metodo; leta 1621 je izračunal 34 decimalk tega števila. Za tem pa je geometrijo pri računanju števila π nadomestila analiza. Angležž Wallis je leta 1659 izrazil $\pi/2$ v obliki neskončnega produkta:

$$\pi/2 = (2/1) . (2/3) . (4/3) . (4/5) . (6/5) . \ldots$$

ki pa žal prepočasi konvergira in ni uporaben za praktično računanje. Leta 1673 je Leibniz našel vrsto

$$\pi/4 = 1 - 1/3 + 1/5 - 1/7 + \ldots$$

a tudi ta ne konvergira nič hitreje.

Leibnizova vrsta je poseben primer splošne vrste za funkcijo arc tg (x)

$$\text{arc tg}\, x = x - x^3/3 + x^5/5 - x^7/7 + \ldots \; (-1 < x \leqq 1)$$

Določil jo je Gregory leta 1670. Če vanjo vstavimo $x = 1$, dobimo Leibnizovo vrsto, za $x = \sqrt{3}/3$ pa vrsto za $\pi/6$, s katero je Sharp leta 1700 izračunal π na 71 decimalk. Če vrsto malo preoblikujemo, dobimo

$$\pi = \frac{16\sqrt{3}}{3}\left[\frac{1}{2^2 - 1} + \frac{2}{9(6^2 - 1)} + \frac{3}{9^2(10^2 - 1)} + \ldots\right]$$

Tabela 2 prikazuje, kaj dobimo, če upoštevamo v tej vrsti prvih n členov.

n	približek za π
1	3,0
2	3,13
3	3,141 3
4	3,141 56
5	3,141 590
6	3,141 592 4
7	3,141 592 63
8	3,141 592 652
9	3,141 592 654

Tabela 2. Računanje π z vrsto za arc tg $(\sqrt{3}/3)$

Machin je leta 1706 našel zvezo

$$\pi/4 = \text{arc tg}\, 1 = 4\,\text{arc tg}\,(1/5) - \text{arc tg}\,(1/239)$$

Če razvijemo oba arc tg na desni v vrsto in rezultat malo preoblikujemo, dobimo:

$$\pi = 32\left[\frac{3 \cdot 12 + 1}{1 \cdot 3 \cdot 5^3} + \frac{7 \cdot 12 + 1}{5 \cdot 7 \cdot 5^7} + \frac{11 \cdot 12 + 1}{9 \cdot 11 \cdot 5^{11}} + \ldots\right] -$$
$$- 8\left[\frac{3 \cdot 28\,560 + 1}{1 \cdot 3 \cdot 239^3} + \frac{7 \cdot 28\,560 + 1}{5 \cdot 7 \cdot 239^7} + \ldots\right]$$

To pa nam da devet decimalk števila π, če upoštevamo tri člene prve in en člen druge vrste.

Kasneje je veliki slovenski matematik in balistik Jurij Vega izpopolnil Machinovo idejo. Uporabil je zvezo

$$\pi/4 = 5\,\text{arc tg}\,(1/7) + 2\,\text{arc tg}\,(3/79)$$

ki zagotavlja še hitrejšo konvergenco. Z njo je izračunal π na 137 mest. Račune je preveril s podobnim obrazcem

$$\pi/4 = 2\,\text{arc tg}\,(1/3) + \text{arc tg}\,(1/7)$$

Pri tem je arc tg (1/7) razvil v vrsto le enkrat. Vega je tako na 113 mestu popravil napako prejšnjega rekorderja de Lagnyja, ki je leta 1719 izračunal 127 decimalk števila π.

Šele leta 1844 je Dase izboljšal Vegov rezultat, ko je pod Gaussovim nadzorstvom z obrazcem

$$\pi/4 = \text{arc tg}\,(1/2) + \text{arc tg}\,(1/5) + \text{arc tg}\,(1/8)$$

izračunal 200 decimalk števila π. »Peš« sta π računala še Rutherford leta 1853 na 440 in Shanks leta 1873 na 526 pravilnih decimalk. Danes poznamo splošen postopek, ki nam omogoča iskati zveze, kot so jih našli Machin, Vega in drugi. Postopek je precej preprost.

1. Izberi $t \in [-1,1]$. Postavi $T_0 = 1$. Izberi naravno število n.
2. *Za* $i = 1, 2, \ldots, n$ ponovi:

$$T_i = (T_{i-1} - t)/(tT_{i-1} + 1)$$

Če velja $|T_n| < 1$ in je $tT_i > -1$ za $i = 1, 2, \ldots,$ n $- 1$, tedaj velja

$$\pi = 4n \text{ arc tg } t + 4 \text{ arc tg } T_n$$

Navadno izberemo za t recipročno naravno število, za n pa tisto število, za katerega je kót n arc tg t najbližje pravemu kotu. Po omenjenem postopku tedaj dobimo $T_n = \text{tg}\,(\pi/4 - n \text{ arc tg } t)$. Če izberemo $t = 1/5$ in $n = 4$, dobimo Machinov obrazec.

Vegov obrazec dobimo za $t = 1/7$ in $n = 5$. Tedaj pride $T_5 = 237/3116$, vendar je arc tg $T_5 = 2$ arc tg $(3/79)$. Drugo Vegovo formulo pa dobimo za $t = 1/3$ in $n = 2$. Iz $t = 1/8$ in $n = 6$ ter $t = 1/18$ in $n = 12$ lahko izpeljemo obrazca

$$\pi = 24 \text{ arc tg } (1/8) + 8 \text{ arc tg } (1/57) + 4 \text{ arc tg } (1/239)$$

in

$$\pi = 48 \text{ arc tg } (1/18) + 32 \text{ arc tg } (1/57) - 20 \text{ arc tg } (1/239)$$

ki sta ju leta 1961 uporabila Shanks in Wrench za računanje 100 000 decimalk števila π. Kasneje pa je s tema obrazcema in z dobrim računalnikom Guilloud iz Pariza izračunal π na 1,000 000 decimalk. Zanimivo bi bilo najti še boljše obrazce. Treba bi bilo tudi izdelati podrobno analizo časovne zahtevnosti omenjenih postopkov.

KASNEJŠI DOSTAVEK. Kmalu potem ko sem oddal ta rokopis, sem v reviji *BIT* 23 (1983) 538—540 zasledil tale postopek za računanje števila π: $\pi = \lim \pi_n$, kjer je zaporedje π_n podano — skupaj z dvema drugima zaporedjema x_n *in* y_n — rekurzivno takole

$$x_1 = \sqrt{2}, \qquad y_1 = (\sqrt[4]{2} + 1)/\sqrt{2}, \qquad \pi_1 = 2 + \sqrt{2},$$

$$x_{n+1} = \frac{x_n + 1/\sqrt{x_n}}{2} \qquad x_{n+1} = \frac{x_n y_n + 1/\sqrt{x_n}}{y_n + 1} \qquad \pi_{n+1} = \pi_n \frac{x_{n+1} + 1}{y_{n+1} + 1}$$

Avtorja J. M. Browien in P. B. Browien trdita, da za $n \geqslant 5$ velja ocena $|\pi_n - \pi| \leqslant 10^{-2^n}$.

LITERATURA

[1] Philip J. Davis, *The Lore of Large Numbers*, Random House, New York 1961.

[2] George Miel, *An algorithm for the calculation of* π, Amer. Math. Monthly **86** (1979) 694—697.

[3] George Miel, *Of calculations past and present: the Archimedean Algorithm*, Amer. Math. Monthly **90** (1983) 17—35.

[4] Jože Povšič, *Bibliografija Jurija Vege*, SAZU, Ljubljana 1974.

[5] Ivan Vidav, *Rešeni in nerešeni problemi matematike*, MK, Ljubljana 1975.

Obzornik mat. fiz. 31 (1984) 2

O ALGORITMU IN PODATKOVNI STRUKTURI

JERNEJ KOZAK

Math. Subj. Class. (1980): 68 B 10, 68 B 15

V članku se seznanimo z algoritmi in podatkovnimi strukturami. Posebej razložimo njihovo pomembno vlogo v računalniški znanosti.

ON ALGORITHM AND DATA STRUCTURE

In the article an introduction to algorithms and data structures is given. In particular, their fundametal role in the computer science is explained.

1. Uvod

Računalništvo je mlada veja znanosti. Toda ne bomo se dosti zmotili, če trdimo, da bo s sadovi svojih spoznanj temeljito spremenila svet. Čeprav njeno vidno rast lahko spremljamo šele od leta 1944, s svojimi novimi načini reševanja problemov že vrača začetno pomoč sestrskim vejam znanosti, kot so matematika, elektrotehnika, fizika in druge. Za pomembno novost štejemo, da človeka poudarjeno vodi k natančnemu algoritmičnemu načinu razmišljanja. Po izkušnjah sodeč trdimo, da je ta vpliv med mlajšimi že danes krepko opazen, pa naj gre zasluga za to malim žepnim računalnikom, osebnim računalnikom ali pa uvajanja pouka programiranja v srednje šole. Vzemimo primer. Pred desetimi leti je bila v začetnem tečaju programiranja na Fakulteti za naravoslovje in tehnologijo naloga Sestavi algoritem, ki v danem zaporedju poišče največje število velik problem. V posebnem primeru, na primer v zaporedju

17 , 16 , 13 , 48 , 24 , 35

je seveda vsakdo pokazal na 48, redkokdo pa je znal pojasniti, kako je do te vrednosti prišel na način, ki bi bil uporaben tudi v splošnem primeru. Kaj šele, da bi napotek dosledno zapisal, tako da bi bilo navodilu mogoče slepo, a pravilno slediti do rezultata. No, danes bi ob taki nalogi v istem razredu dobili več pravilnih rešitev.

Algoritmično razmišljanje človeku res ni prirojeno. O tem nas prepriča mnogo primerov iz vsakdanjega življenja. Čeprav je na primer jasno, KAJ je treba storiti, se pri izpeljavi, KAKO to storiti, stvari tako zapletejo, da skoraj gotovo obstaja vrsta poti, ki vodijo čisto drugam namesto proti zastavljenemu cilju. Srečujemo se s postopki, ki so izrazito nepregledni, imajo vrsto izjem, so v nekaterih delih premišljeni do nepomembnih detajlov, drugod pa povsem neizdelani.

Tudi z organiziranim učenjem se le počasi vživljamo v algoritme in njihovo sestavo. O tem nas prepriča presenetljiv zgled, ki ga srečamo kljub obširnemu pouku računalništva na matematiki. Študentje uporabne matematike med študijem poslušajo dva obvezna in en neobvezen računalniški predmet in se srečajo z obvezno uporabo računalnika pri več drugih predmetih. Pomemben del snovi v prvem letniku je študij postopkov urejanja. Tu se srečajo s preprostimi algoritmi, kot sta urejanje z (navadnim) izbiranjem in urejanje z vstavljanjem, pa tudi z bolj zapletenimi, kot so urejanje z zlivanjem, urejanje s kopicami in hitro urejanje. Spoznajo časovne zahtev-

nosti posameznih postopkov in se ob preizkušanju sami prepričajo o učinkovitosti hitrega urejanja v primerjavi z drugimi algoritmi (za večja zaporedja seveda). Toda ko je v drugem letniku v kaki domači nalogi treba urejati, skoraj vsak študent uporabi urejanje z izbiranjem, če je izbira postopka prepuščena njemu samemu. Hitro urejanje mu pač ni prišlo pod kožo in mu ne zaupa. S tem zgodbe še ni konec. Tudi po treh računalniških predmetih nekateri v diplomskem delu uporabijo urejanje z izbiranjem. Prav odločitev za urejanje z izbiranjem pove vse. To je naraven algoritem, ki ga je morda najpreprosteje razumeti. Za druge, bolj zapletene postopke se sicer zdi, da delujejo, ampak... Zato se je varneje držati postopka, ki ni učinkovit, a je domač. Pouk računalništva je tako pri takšnih študentih izzvenel v prazno.

2. **Algoritem...**

Pojem algoritma je za računalništvo tako pomemben, da bi smeli reči, da je računalniška znanost študij algoritmov. Ta študij razdelimo v štiri osnovna področja:

a) *Računski stroji, ki opravljajo algoritme.* V to področje sodi prozvodnja računalnikov s svojo pestro vsebino, od žepnih, osebnih, mikro, mini računalnikov pa do velikih drobilcev števil, kot jim v šali včasih rečemo.

b) *Jeziki, s katerimi opisujemo algoritme.* Kot vemo, zahteva avtomatično izvrševanje napotkov v računalniku natančen, vnaprej pripravljen opis naših zahtev. Še več, to navodilo mora biti tedaj, ko ga računalnik opravlja, zapisano v njemu razumljivem jeziku, na primer kodirano z vrednostma 0 in 1, v jeziku procesne enote. Takšen zapis je človeku popolnoma nerazumljiv, zato napotke pišemo v programskih jezikih, njihovo prevajanje v osnovni računalnikov jezik pa prepuščamo prevajalnikom. Jezik, v katerem sestavimo program, mora zato biti primeren za avtomatično prevajanje, hkrati pa dovolj zmogljiv, da omogoča preprost in pregleden zapis algoritma. Kot vemo, je prav razvoj takšnih jezikov bistveno prispeval k širjenju uporabe računalnikov, saj je prinesel lažje pisanje in predvsem prenosljivost programske opreme.

c) *Osnove algoritmov.* To je področje, kjer (bolj kot same algoritme) študiramo razrede problemov, ki jih z algoritmi rešujemo. Zanima nas, katerih problemov se z algoritmi sploh lahko lotimo, koliko bodo razviti algoritmi vsaj zahtevni ipd. Tu se srečamo s slovitimi nerešenimi vprašanji, kot P = = NP? in drugimi.

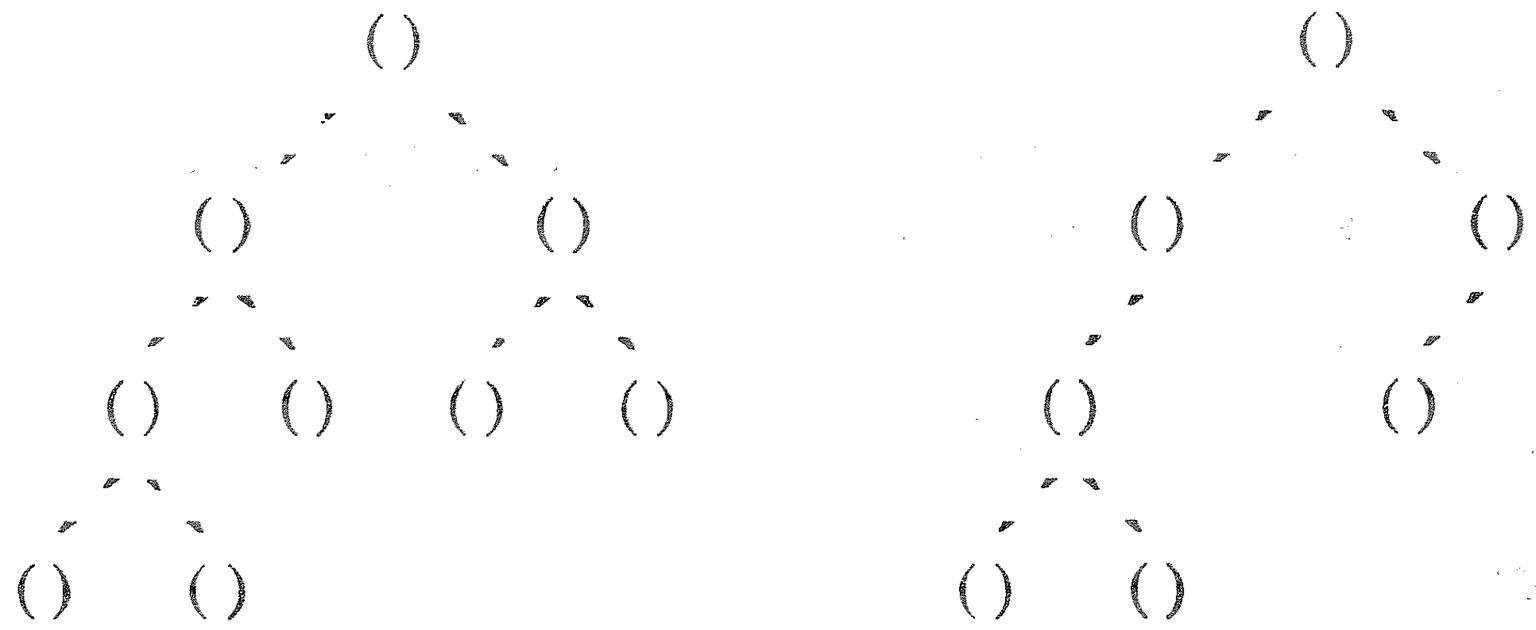
d) *Analiza algoritmov.* Algoritem je rešitev nekega problema. Vendar ga računalniku nikoli slepo ne prepustimo v izvajanje. Vnaprej ocenimo, koliko prostora bo zasedal v računalnikovem pomnilniku in koliko časa bo tekel. Pravimo, da analiziramo njegovo *prostorsko* in *časovno zahtevnost.* Mnogokrat zahtevnost podrobneje razčlenimo in ocenimo, koliko bo algoritem tekel v najslabšem, v najboljšem in v pričakovanem primeru, odvisno od vhodnih podatkov.

Beseda algoritem je izpeljana iz imena perzijskega matematika Abu Ja'far Mohamed ibn Musa al Khowarizmi (825 pr. n. š.) nekako takole

al-Khowarizmi → algorism → algorithm → algoritem,

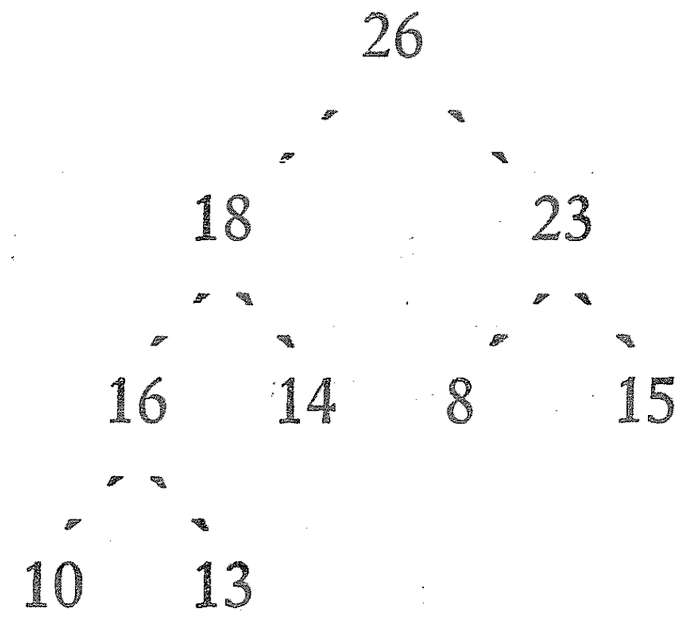
torej izvira iz davne preteklosti. Povejmo, kaj algoritem je. Da nam bo lažje, si za začetek pripravimo primer. Zapišimo algoritem (algoritem 1), ki vstavi

element v *kopico*. Kopica je *levo poravnano dvojiško drevo*, kjer vrednosti *sinov* nista večji od vrednosti *očeta*. Dvojiško drevo je ali prazno ali pa ga sestavlja končna množica *vozlišč*, od katerih je eno posebej odlikovano in mu pravimo *koren*, preostala pa razpadejo v dve disjunktni množici, ki sta prav tako dvojiški drevesi (levo in desno poddrevo). Najpreprosteje si dvojiško drevo predstavimo tako, da ga narišemo od korena navzdol. Dvojiško drevo je levo poravnano, če so vsi nivoji drevesa razen morda zadnjega zasedeni polno, na zadnjem nivoju pa vsa vozlišča pomaknjena v levo. Na sliki 1 sta primera dveh dvojiških dreves brez podatkov v vozliščih. Prvo je poravnano, drugo pa ne, saj na tretjem nivoju manjkata dve vozlišči.



Sl. 1. Primera dvojiških dreves

Iz dvojiškega levo poravnanega drevesa dobimo kopico, če v vozlišča razporedimo števila ali kake drugačne podatke tako, da vrednost v korenu ni manjša od vrednosti v korenu levega in desnega poddrevesa, obe poddrevesi pa sta prav tako kopici. Prazni kopici pripišemo vrednost, ki je manjša od vseh drugih. Zgled kopice, kjer so podatki kar naravna števila, vidimo na sliki 2.



Sl. 2. Primer kopice naravnih števil

Ker je kopica poravnano dvojiško drevo, jo preprosto vodimo v tabeli. Vrednosti zložimo zaporedoma po nivojih v globino, vrednosti istega nivoja od leve v desno. Pri takšnem dogovoru lahko brez težav obnovimo prvotno informacijo neposredno iz tabele. Za element na mestu i najdemo očeta na mestu i *div* 2, levega sina na mestu 2 * i in desnega sina na mestu 2 * i + 1. Tu operator *div* označuje celoštevilčno deljenje. Kopico s slike 2 prepišemo v tabelo takole

(26 , 18 , 23 , 16 , 14 , 8 , 15 , 10 , 13)

In sedaj k primeru algoritma. Sestavljanje kopice razvijmo v dveh korakih. Prvi naj dani kopici, na primer iz *n* podatkov, doda novo vrednost tako, da je razširjeno drevo tudi kopica. Drugi pa naj sestavi kopico v celoti.

Tudi bolj zapleteni, prvi algoritem, uženemo brez težav. Urejenost kopice kvari le dodani podatek na zadnjem mestu razširjene tabele, zato naj splava v višino, dokler ne trči na večjega očeta. Za natančen zapis postopka si izposodimo del slovnice programskega jezika pascal. Zapis bo dovolj preprost, da ne bo delal težav tudi tistim, ki s pascalom niso domači.

```
procedure vstavi (k , n , e);
{Vstavi podatke e v kopico k[1 : n] na tako mesto,
da je nova tabela k[1 : n := n + 1] tudi kopica.}
begin
    n := n + 1;
    j := n;
    i := j div 2;
    {j indeks kandidata za mesto podatka e, i njegov oče.}
    while (i > 0) and (k[i] < e) do
        begin
            k[j] := k[i];
            j := i;     i := j div 2
        end;
    k[j] := e;
end.
```

Algoritem 1. Vstavi nov podatek v kopico.

Celotno kopico sestavimo z algoritmom 2.

```
n := 0;
while je še kaj podatkov do
  begin
    e dobi vrednost;
    vstavi (k , n , e)
  end.
```

Algoritem 2. Zgradi kopico z zaporednim vstavljanjem.

Natančno definicijo algoritma bi sicer morali nasloniti na formalni model računalnika, kateremu je algoritem namenjen. Ta opis bi zajemal med drugim nabor podatkov, osnovne operacije, organizacijo dela itd. Za naš namen pa bo popolnoma dovolj intuitivna predstava o računalniku, ki smo si jo v dosedanji uporabi računalnika že pridobili. Med operacije, ki jih procesna enota pozna, štejemo aritmetične operacije (nad celimi in realnimi števili), primerjave ipd. Postavimo trditev.

Algoritem je končna množica ukazov, ki, če jih ubogamo, opravijo neko nalogo. Zanj velja:

Ima podatke. Množica podatkov je lahko tudi prazna. V algoritmu 1 so podatki

n , $k[1 : n]$, e

kar razberemo iz pojasnila v glavi procedure.

Vrne rezultat. Vrne nam vsaj eno izračunano vrednost ali kaj stori, kot na primer premakne papir na tiskalniku na novo stran. V našem primeru je rezultat

n , $k[1 : n]$

Je natančno določen. Vsak ukaz mora nedvoumno povedati, kaj naj tisti, ki opravlja algoritem, v danem trenutku stori. Stavki, kot je **while . . .do**,

for, prireditveni stavek imajo svoj natančno definirani pomen. Natančno povedo, kakšno zaporedje ukazov naj se opravi.

Se vedno konča. V algoritmu 1 je to lahko preveriti. Če se algoritem ne bi končal, bi se to lahko zgodilo le v zanki **while.** Toda pri vsaki ponovitvi se i, prvotno določen kot celo število n **div** 2, celoštevilčno deli z 2. Zato gotovo doseže 0 in vsaj pogoj $i > 0$ ni vedno izpolnjen.

Mogoče ga je opraviti. V načelu ga je mogoče izpeljati »peš«, s papirjem in svinčnikom. To v našem primeru ne bo težko. Vzemimo podatke k in jih po algoritmu 2 na mestu preuredimo v kopico

$$k = (10\ ,\ 16\ ,\ 8\ ,\ 23\ ,\ 14\ ,\ 26\ ,\ 15\ ,\ 13\ ,\ 18)$$

Naj mastno tiskani podatki označujejo kopico na tekočem koraku. Zaporedno vstavljanje prikazuje tabela 1.

Podatek, ki ga vstavljamo	Tekoča kopica								
10	**10,**	16,	8,	23,	14,	26,	15,	13,	18
16	**16,**	**10,**	8,	23,	14,	26,	15,	13,	18
8	**16,**	**10,**	**8,**	23,	14,	26,	15,	13,	18
23	**23,**	**16,**	**8,**	**10,**	14,	26,	15,	13,	18
14	**23,**	**16,**	**8,**	**10,**	**14,**	26,	15,	13,	18
26	**26,**	**16,**	**23,**	**10,**	**14,**	**8,**	15,	13,	18
15	**26,**	**16,**	**23,**	**10,**	**14,**	**8,**	**15,**	13,	18
13	**26,**	**16,**	**23,**	**13,**	**14,**	**8,**	**15,**	**10,**	18
18	**26,**	**18,**	**23,**	**16,**	**14,**	**8,**	**15,**	**10,**	**13**

Tabela 1. Primer sestavljanja kopice

Reševanje zahtevnejših problemov zahteva pravzaprav razvoj vse bolj podrobno razčlenjenih algoritmov. Začetni opis problema, na primer

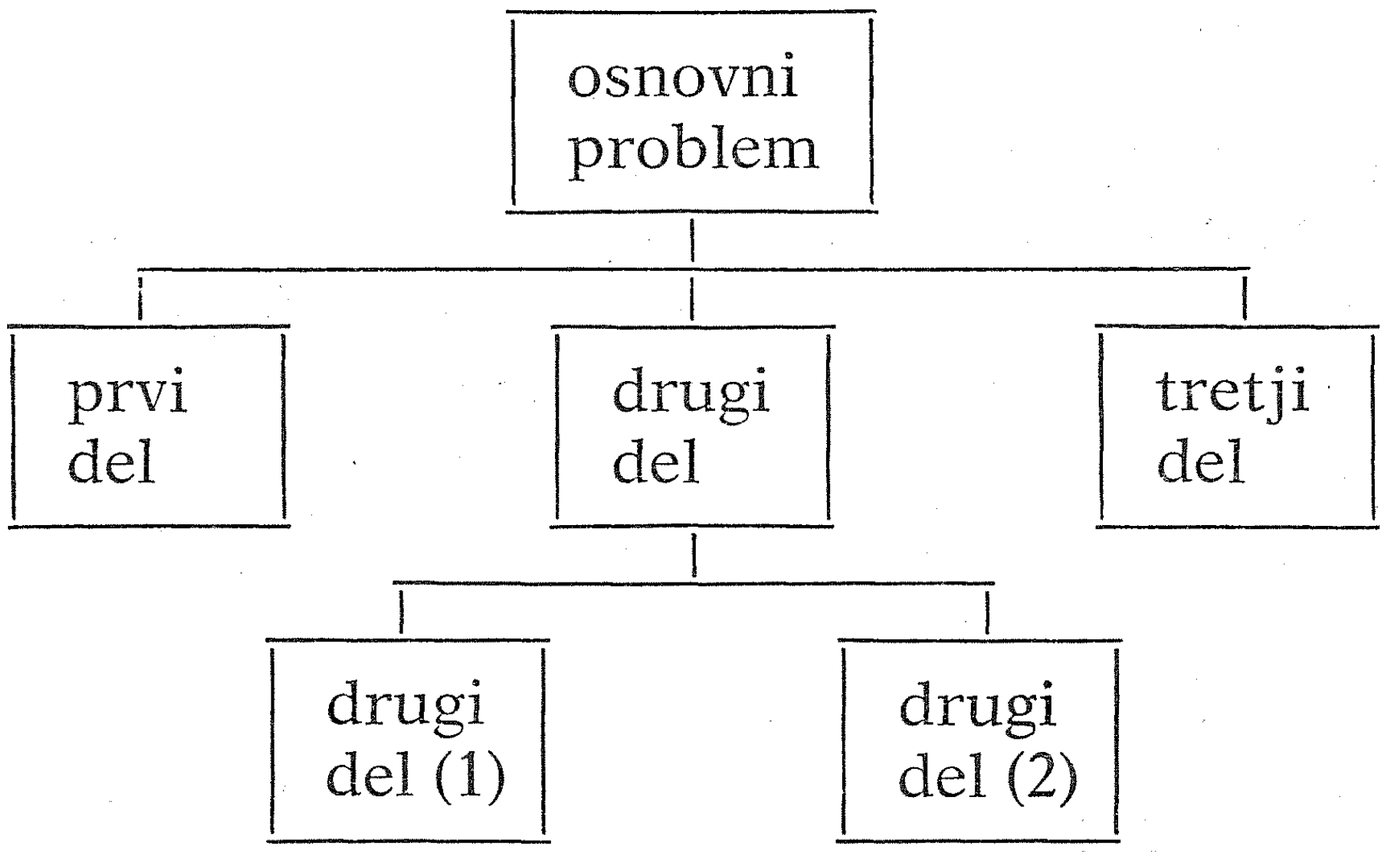
»Določi vrednost $I := \int_b^a f(t)\,dt$.«

ali

»Uredi zaporedje števil $a[1], a[2], \ldots, a[n]$ nepadajoče.«

je lahko popolnoma dobro opisan postopek za človeka, ne pa za računalnik, ki tako zapletenih operacij, kot »Določi vrednost integrala« ali »Uredi zaporedje«, pač ne pozna. Prvotni opis, ki pove predvsem natančno, KAJ je treba storiti, moramo razgraditi v preprostejše probleme, ki povedo, KAKO to storiti, tako da jim bo računalnik lahko sledil. V končni fazi razvoja je celotni algoritem zaporedje preprostih ukazov, kot so *seštej, odštej, zmnoži, deli, ali je vrednost* 0, *ali ni vrednost* 0 itd., ki so med seboj ustrezno povezani.

Osnovno pravilo v razvoju rešitve je spoznanje, da težko brez napak premišljamo o več stvareh hkrati. Zato zapleten problem razdelimo na preprostejše probleme, tako da vsakega od delnih problemov lahko rešujemo neodvisno. Povezava med deli rešitve so *podatki.* Postopek ponavljamo v globino, dokler osnovni problem ni dovolj preprost, da ga rešimo v enem koraku. Takšno sistematsko razgrajevanje najenostavneje ponazorimo z drevesom. Primer takega drevesa (razvoja algoritma) kaže slika 3.



Slika 3. Primer drevesa, ki prikazuje razvoj algoritma

Končni algoritem sestavljajo listi drevesa, katerih vsak reši svoj del problema neodvisno od drugih, in povezave med njimi. Algoritmi, ki rešujejo prvi del, drugi del (1), drugi del (2) in tretji del, imajo vsak svoje podatke in rezultate. Rezultate algoritma, ki reši prvi del, lahko uporabijo drugi deli kot podatke, rezultati drugega dela (1) so lahko vhodna informacija za drugi del (2) in tretji del itd. Zato je v razgrajevanju rešitve problema vse več podatkov, ki posredujejo informacijo med posameznimi deli rešitve.
Na zahtevni poti reševanja problema

KAJ → KAKO

se srečamo z vrsto vprašnj:

Kako zasnovati algoritem? Kot smo dejali, algoritmični način razmišljanja človeku po naravi ni domač in se ga mora priučiti. Te naloge se pogosto lotimo s splošnimi prijemi za razvoj algoritmov. Takšnim splošnim načinom pravimo *strategije.* Med pomembne strategije štejemo: *deli in vladaj, požrešna metoda, dinamično programiranje, sestopanje* ter *razveji in omeji.* Vsaka od strategij predpostavlja nekaj posebnih lastnosti problema, na katerem jo lahko uporabimo.

Kako preveriti algoritem? Algoritem sestavlja skupina ukazov, ki jim v izračunu sledimo slepo. Zato v algoritmu ne sme biti napak. Vse možne poti izvajanja algoritma, pa čeprav do njih pripeljejo le popolnoma nesmiselni podatki, moramo premisliti vnaprej. Preizkus algoritma ali programa za dane podatke kvečjemu odkrije napako, ne zagotavlja pa pravilnosti algoritma za poljuben nabor podatkov. Pomagamo si tako, da pravilnost postopka dokažemo.

Pri sestavljanju kopice bi dokaz tekel takole. Za začetek opazimo, da na obeh mestih, kjer j dobi vrednost, določimo tudi i z $i := j$ **div** 2. Torej i vseskozi označuje indeks očeta j. Trdimo:

Prvotni podatki $k[1 : n]$ so med tekom
procedure ves čas v

$k[l]$, $l = 1, 2, \ldots, j-1, j+1, \ldots, n := n+1$.

Na začetku se n poveča na $n := n + 1$, j dobi vrednost (novega) n in trditev res velja. Veljavo trditvi lahko izpodnesemo le tedaj, ko spremenimo j. Toda na tem mestu stavka

$k[j] := k[i]$; in $j := i$

kot celota trditev ohranita. Na mesto $k[j]$ zato upravičeno vstavimo podatek e, če je le to mesto pravo. Zanka **while** se gotovo konča, kot smo že ugotovili. Dokler zanka teče, ves čas velja $k[i] < e$. Če se izteče z $i = 0$, e sodi v koren ($j = 1$), če pa za dani i velja $k[i] \geqq e$, e sodi na mesto sina i, to je j. Dokaz je končan.

Kako analizirati algoritem? Kaj algoritem zahteva za svojo izvedbo, kolikšna je njegova časovna in prostorska zahtevnost? V splošnem kot zahtevnost razumemo tako imenovano količino računskih virov in sredstev, ki jih potrebujemo za rešitev problema po izbranem postopku. V zahtevnost sodi: količina potrebnega prostora v pomnilniku, na magnetnem disku in na magnetnem traku, potrebno število osnovnih aritmetičnih operacij ipd. Zahtevnost je seveda odvisna od števila podatkov, torej od obsežnosti problema v konkretnem primeru. Merimo jo v kaj različnih enotah. Ko govorimo o časovni zahtevnosti, sta običajni enoti čas teka algoritma in število osnovnih aritmetičnih operacij. Med osnovne operacije štejemo seštevanja, odštevanja, množenja, deljenja, primerjave, torej v računalniku vgrajene operacije. Za prostorsko zahtevnost je najpogosteje enota kar število besed ali zlogov, potrebnih za predstavitev vse informacije. Ker nas običajno zanima le velikostni red zahtevnosti, si pomagamo s funkcijama $O(x)$ in $o(x)$.

Vzemimo algoritma 1 in 2 ter ocenimo njuno časovno zahtevnost. Za posamezno vstavljanje o zahtevnosti odloča število ponovitev **while** zanke v proceduri *vstavi*. V najboljšem primeru se zanka nikoli ne izvrši, kar nam za sestavo kopice z n podatki da zahtevnost $O(n)$. V najslabšem primeru bo i-ti podatek vsakič splaval v koren. Zanimivo je, da ta najslabši primer dosežemo, če dobivamo podatke že urejene nepadajoče. Naj bo zaradi preprostosti računa drevo polno

$$n = 2^r - 1$$

Vozlišča nivoja i torej v najslabšem primeru prinesejo

(število podatkov na nivoju i) * (globina $-$ 1) =

(število podatkov na nivoju i) * ($i - 1$)

ponovitev zanke, kar nam skupno da

$$\sum_{i=1}^{r} (i - 1)\, 2^{i-1} = r\, 2^r - 2^{r+1} + 2 = O\,(n \log (n))$$

Ker zahtevnost kvečjemu raste z n, sklepamo, da algoritem tudi za splošni n zahteva $O\,(n \log (n))$ operacij.

Kako izraziti algoritem? Algoritem opišemo v jeziku, ki ga tisti, ki mu je namenjen, človek ali računalnik, enolično razume. Zapis, ki je namenjen tudi človeku, mora biti enostaven, lahko čitljiv in komentiran. Če je algoritem zapisan v programskem jeziku, torej namenjen računalniku, mu pravimo *program* in govorimo o čitljivosti programa, navodilih za uporabnika programa (ki natančno povedo, kaj programska enota dela, kaj so podatki, kaj rezultat in kje je programska enota shranjena) ter o navodilih za vzdrževalca programske enote (ki povedo ne le, KAJ programska enota dela, ampak tudi, KAKO to dela). V navodilih za vzdrževalca so posebej pomembni skrbno opisani vhodni in izhodni podatki programske enote, slovar lokalnih spremenljivk ter čitljivost programa samega, torej jasen, s komentarji pojasnjen programski tok. Zgled programske enote, ki smo ga zabeležili v tretjem letniku matematike, kaže, da nekaterim študentom jasno pisanje programov kljub pouku računalništva ni domače:

```
      SUBROUTINE HORNER (N, A, B, X)
C
C     POLJA
C     A-PARAMETER
C     B-PARAMETER
C     SPREMENLJIVKE
C     N-PARAMETER
C     X-PARAMETER
C
```

Na podobno slabost nas opozori nesmiselni komentar, ki ga srečamo v mnogih učbenikih v svarilo

```
C     I SE POVEČA ZA 1
C
      I = I + 1
```

V obeh primerih nam komentar ne pove nič novega. V prvem primeru kljub komentarju ne izvemo, kaj pomenijo parametri, v drugem pa komentar in stavek trdita isto, zato je komentar odveč.

2. . . . in podatkovna struktura

Pravzaprav je bil naš pogled na računalništvo precej enostranski. Res je algoritem zaporedje ukazov, ki nekaj stori z vhodnimi podatki in izračuna rezultat. Vendar se lahko postavimo tudi na stališče, da nam je popolnoma vseeno, kako nekaj izračunamo, dovolj je le, da vemo, v kakšni zvezi so rezultati z vhodnimi podatki. Če nas zanima le uporaba računalnika, ne pa tudi priprava postopkov samih, je tak pristop morda najnaravnejši. Poglejmo si, ali nam ta pot ponudi kaj novega v poznavanju računalništva. Definirajmo računalništvo kot študij podatkov. Štiri osnovna področja, na katera smo razdelili računalniško znanost, po gornjem premisleku opišemo takole:

a) *Računski stroji, ki obdelujejo podatke*
b) *Jeziki, s katerimi opisujemo obdelavo podatkov*
c) *Osnove obdelave podatkov*
d) *Analiza podatkovnih struktur*

Študija podatkovnih struktur in algoritmov sta torej med seboj tesno povezana in pravzaprav govorita o isti stvari iz dveh zornih kotov. Pojem podatka in sestavljenega podatka (strukture) srečamo že v uvodnih korakih spoznavanja programskih jezikov. Tam na primer spoznamo podatkovne tipe:

znak, celo število, realno število, število v dvojni natančnosti, množica, Boolova (logična) vrednost, itd.

Vsak podatek v računalniku pripada zalogi vrednosti svojega tipa. Tako na primer celim številom v računalniku s šestnajstbitno predstavitvijo celih števil pripada zaloga vrednosti

$$[-32767\ ,\ 32767]$$

V programskem jeziku poznamo vrsto stavkov, v katerih naštejemo spremenljivke, ki jih bomo uporabili, in povemo njihov tip. Tako stavki

```
INTEGER N
integer n
n : integer
dcl n bin fixed
```

povedo v različnih jezikih, da je n ime spremenljivke celoštevilčnega tipa, torej spremenljivke, katere vrednost bo, ko bo definirana, pripadala zalogi vrednosti celih števil. Za tipe podatkov, ki jih pozna že programski jezik sam, pravimo, da so vgrajeni. Sama zaloga vrednosti kakega podatka ne zasluži imena struktura, postane pa to, ko povemo, katere operacije nad danim podatkom lahko opravimo in katere lastnosti imajo te operacije. Pogosto se zgodi, da operacije prepletajo več različnih množic podatkov. Da lahko opišemo tudi takšne strukture, moramo v splošnem v strukturi združiti več množic. Podatkovno strukturo torej sestavlja nabor osnovnih množic

$$M_i,\ i \in I$$

in operacij (funkcij) oblike

$$f_j : \prod_{i \in L_j} M_{r_{i,j}} \rightarrow M_{n_j},\ j \in J$$

Za množice M_i včasih zahtevamo, da imajo svojo notranjo strukturo. V primeru podatkovne strukture, ki jo bomo vzeli za zgled, bo ena od množic linearno urejena, druga pa najpreprostejši primer Boolove algebre.

Podatkovna struktura je torej popolnoma algebraičen pojem. V računalništvu pa si pri njenem opisu in uporabi dovolimo nekaj nedoslednosti. Prva je v tem, da podatkovno strukturo poimenujemo po eni od osnovnih množic podatkov. Druga, ki zbega matematika, pa je, da za elemente množic pogosto uporabimo isto ime kot za množico samo. Tako smo že srečali dvojiško drevo, ki je lahko ime podatkovne strukture, množica vseh dvojiških dreves ali pa kar podatek tipa dvojiško drevo, kot je bilo v našem primeru. Povzemimo. Podatkovno strukturo sestavljajo štiri komponente:

1. Osnovna zaloga vrednosti strukture, torej množica podatkov, katere ime struktura privzame
2. Zaloge vrednosti preostalih podatkov, ki jih struktura združuje
3. Operacije (funkcije), ki so v podatkovni strukturi definirane
4. Aksiomi, ki opisujejo lastnosti operacij

V opisu podatkovne strukture ni besede o tem, kako operacije v resnici tudi izpeljemo. Definicija torej pove le, KAJ je struktura, ne pa, KAKO jo predstaviti. Pravzaprav že pri preprostem podatku, kot je celo število, med uporabo upoštevamo le, kaj lahko z njim počnemo, skoraj nič pa njegove predstavitve. Kakorkoli ga je že v računalniku predstavil proizvajalec računalnika, na primer v dvojiškem pozicijskem zapisu v predznačeni notaciji, komplementu do 2 ali komplementu do 1, je moral poskrbeti, da njegova predstavitev res zadošča aksiomom celih števil.

Poglejmo si primer podatkovne strukture, *vrsto s prednostjo*. Vrsta je podatkovna struktura, iz katere jemljemo podatke v istem vrstnem redu, kot so vanjo prišli. Pri vrsti s prednostjo so podatki linearno urejeni in iz vrste brišemo vedno največji podatek. Za začetek jo opišimo formalno, podobno, kot smo formalno zapisali algoritem.

```
structure vrsta (podatek, Boolean);
{Struktura vrsta s prednostjo nad poljubno urejeno zalogo
  vrednosti podatek.}
begin
  declare
    {Pripravi prazno vrsto s prednostjo.}
    pripravi  : ∅ → vrsta;
    {Vstavi podatek v vrsto s prednostjo in vrne novo vrsto.}
    vstavi    : (podatek, vrsta) → vrsta;
    {Vrne največji podatek v vrsti s prednostjo.}
    največji  : vrsta → podatek;
    {Izbriše največji podatek v vrsti s prednostjo.}
    odstrani  : vrsta → vrsta;
    {Je vrsta s prednostjo prazna?}
    prazna    : vrsta → Boolean;
  where
    prazna (pripravi) : = true;
    prazna (vstavi (p, v)) : = false;
    največji (pripravi) : = napaka;
    največji (vstavi (p, v)) : =
      if prazna (v) then p
      else max (p, največji (v));
    odstrani (pripravi) : = napaka;
    odstrani (vstavi (p, v)) : =
      if prazna (v) then pripravi
      else
        if p = največji (v) then v
        else vstavi (p, odstrani (v));
end.
```

Podatkovna struktura 1. Vrsta s prednostjo

Razberimo komponente podatkovne strukture. S to strukturo opišemo vrsto (s prednostjo) in združimo množice *vrsta, podatek ter Boolean.* V strukturi je definiranih petero operacij: *pripravi, vstavi, največji, odstrani* in prazna. Zadnje tri vežejo v **where** ... **end** delu strukture našteti aksiomi. Funkcija *max* je tu operacija nad množico *podatek,* ki nam za poljubna dva podatka vrne večjega.

Aksiomi nam popolnoma neodvisno od tega, kako vrsto s prednostjo res izvedemo, definirajo, kako delujejo posamezne operacije. Prepričajmo se z zgledom. Za podatek vzemimo prve štiri elemente zaporedja, ki smo ga v tabeli 1 uredili v kopico. Zgrajena vrsta s prednostjo je

$$v = vstavi\ (23,\ vstavi\ (8,\ vstavi\ (16,\ vstavi\ (10,\ pripravi)))).$$

Poiščimo največji podatek. Po drugem paru pravil bo to takole.

$$največji\ (vstavi\ (23,\ vstavi\ (8,\ vstavi\ (16,\ vstavi\ (10,\ pripravi))))) =$$
$$max\ (23,\ največji\ (vstavi\ (8,\ vstavi\ (16,\ vstavi\ (10,\ pripravi))))) =$$
$$max\ (23,\ max\ (8,\ največji\ (vstavi\ (16,\ vstavi\ (10,\ pripravi))))) =$$
$$max\ (23,\ max\ (8,\ max\ (16,\ največji\ (vstavi\ (10,\ pripravi))))) =$$
$$max\ (23,\ max\ (8,\ max\ (16,\ 10))) =$$
$$max\ (23,\ max\ (8,\ 16)) = max\ (23,\ 16) = 23$$

Aksiomatična definicija podatkovne strukture pove le, KAJ struktura je, nič pa ne pove o njeni predstavitvi. No, če želimo strukturo uporabiti, moramo izpeljati njeno predstavitev. To pomeni, da moramo pri sestavi programa vse podatke in operacije prvotne strukture predstaviti s podatki in operacijami, ki jih pozna izbrani programski jezik. Ob tem se nam sámo po sebi ponudi vprašanje, ali je bilo vredno vložiti toliko truda v formalni opis strukture, saj jo bomo tako ali tako tudi izpeljali. Toda prav z ločitvijo seznama operacij in njihove izpeljave pa storimo podoben pomemben korak, kot smo ga v razvoju algoritma (slika 3). V prvem koraku iskanja predstavitve podatkov je najvažneje, da izluščimo nabor operacij in njihovih lastnosti, razmišljanje o njihovi izpeljavi pa odložimo za pozneje. Na drugem koraku že vemo, kakšna mora biti podatkovna struktura. Tu se odločamo, običajno med več možnostmi, o konkretni predstavitvi strukture. Pri enih predstavitvah izpeljemo učinkoviteje ene operacije, pri drugih druge. Pričakovana pogostnost posameznih operacij v konkretnem problemu bo vsekakor pomemben faktor pri izbiri predstavitve.

Vrsto s prednostjo lahko predstavimo na mnogo načinov. Odločamo se med kopico, *2—3 drevesom* dveh oblik, *najbolj levim drevesom* itd. Katero predstavitev bomo izbrali, je seveda odvisno od narave problema, ki ga rešujemo. Če se odločimo za kopico, operacijo *vstavi* že imamo (algoritem 2). Sestavimo še operacije *pripravi, prazna, največji* in *odstrani* (algoritmi 2—6).

```
procedure pripravi (k , n);
{Pripravi prazno vrstno s prednostjo, kopico k.}
begin
   zagotovi prostor za kopico k;
   {Prostor zagotovimo v različnih programskih jezikih kaj različno. Običajno
   se moramo vnaprej omejiti na največjo možno kopico in nato pri vstavljanju preverjati, če je še kaj prostora na razpolago.}
   n : = 0;
end.
```

Algoritem 3. Pripravi prazno vrsto s prednostjo

```
function prazna (k , n);
{Pove, ali je vrsta s prednostjo prazna ali ne.}
begin
   if n = 0 then prazna : = true
   else prazna : = false;
end.
```

Algoritem 4. Je vrsta s prednostjo prazna?

```
function največji (k, n);
{Vrne največji podatek vrste s prednostjo.}
begin
   if n = 0 then napaka
   else največji : = k[1]
end.
```

Algoritem 5. Največji podatek v vrsti s prednostjo

```
procedure odstrani (k, n);
{Odstrani iz vrste s prednostjo največji podatek in vrne
   v k[1 : n : = n — 1] za ta podatek skrčeno vrsto
   s prednostjo.}
```

```
begin
  if n = 0 then napaka
  else
    begin
      {Vrsta s prednostjo ni prazna.}
      i : = 1;
      j : = 2 * i;
      e : = k[n];
      n : = n — 1;
      {Na tem mestu smo največji podatek že izločili in skrčili kopico. Mesto
      i je prazno. Spustimo ga v globino, tako da vanj sodi podatek e.}
      while j ≤ n do
      begin
        if (j < n) and (k[j] < k[j + 1])
          then j : = j + 1;
        {j kaže na večjega sina i.}
        if e ≥ k[j] then exit (while)
        else
          begin
            k[i] : = k[j];
            i : = j;  j : = 2 * j
          end;
      end
      k[i] : = e
    end
end.
```

Algoritem 6. Odstrani podatek iz vrste s prednostjo

Vsaka predstavitev podatkovne strukture mora zadoščati prvotnim aksiomom. Preverimo za vajo, da operacija *največji* res zadošča aksiomom vrste s prednostjo. Kratek pogled na algoritme nas prepriča, da je vedno $n \geq 0$ in n strogo pozitiven, če kopica ni prazna. Torej $n = 0$ označuje prazno vrsto in klic funkcije *največji (pripravi)* res vrne *napako*. Naj bo sedaj $n > 0$. V kopici je vrednost očeta večja od vrednosti sinov. Po dogovoru o izpeljavi kopice v tabeli je odtod

$$k[i] \geq max\,(k[2 * i],\ k[2 * i + 1]$$
za vse smiselne i

Torej velja

$$k[1] \geq k[i], \text{ za vse smiselne } i$$

in je $k[1]$ največji podatek. Naša funkcija največji zadošča tudi drugemu aksiomu, kar smo hoteli pokazati. Dokaz pravilnosti procedure *odstrani* prepustimo bravcu v pouk in zabavo.

LITERATURA

[1] A. V. Aho, J. E. Hopcroft, J. D. Ullman, *The Design and Analysis of Computer Algorithms,* Reading, Mass., Addinson-Wesley, 1974.

[2] E. Horowitz, S. Sahni, *Fundamentals of Computer Algorithms,* Maryland, Computer Science Press, Inc., 1978.

[3] E.Horowitz, S. Sahni, *Fundametals of Data Structures,* Maryland, Computer Science Press, Inc., 1977.

[4] J. Kozak, *Podatkovne strukture in algoritmi,* v pripravi.

ŠTUDIJ MATEMATIKE NA MOSKOVSKI DRŽAVNI UNIVERZI

V Sovjetski zvezi se po srednji šoli, ki jo končajo že s 17 leti, okoli 5 milijonov dijakov vpiše na visoke šole; teh je 860. Med njimi je 65 univerz, ki vzgajajo raziskovalce, 200 pedagoških inštitutov, ki dajejo učitelje za srednje šole, 100 kmetijskih inštitutov, iz katerih prihajajo agronomi, veterinarji, zootehniki in mehaniki, potem so še medicinski inštituti, ki pripravljajo zdravstvene delavce, največ pa je politehničnih inštitutov, ki dajejo inženirje raznih strok.

Med sedemnajstimi fakultetami Moskovske državne univerze (MGU) sta tudi dve matematični: Fakulteta za mehaniko in matematiko (MEH-MAT) in Fakulteta za numerično matematiko in kibernetiko (VMK). Slednja je stara šele okoli 15 let. Kot samostojna enota je bila osnovana zaradi vse večjega pomena uporabne matematike in računalništva. MEH-MAT ima mnogo daljšo tradicijo, saj je bila med prvimi fakultetami MGU in je lani slavila že svojo 225-letnico. Sedaj ima MEH-MAT 18 odsekov (kateder), ki so razdeljeni na dve skupini. V matematični skupini so odseki: matematična analiza, višja algebra, višja geometrija in topologija, diferencialna geometrija, verjetnostni račun, splošni problemi upravljanja, teorija števil, diferencialne enačbe, teorija funkcij in funkcionalna analiza, matematična logika in matematična statistika. V mehanski skupini pa so: teoretična mehanika, aeromehanika in dinamika plinov, teorija prožnosti, hidromehanika, uporabna mehanika, dinamika plinov in valovanj in teorija plastičnosti.

Matematični odseki sprejmejo letno okoli 250 študentov, mehanski pa okoli 150. Kandidati za sprejem opravljajo pisna izpita iz matematike in ruskega jezika in ustna iz matematike in fizike. Izpiti so dokaj zahtevni, tako da jim je kos le vsak peti kandidat. Za tuje študente je rezerviranih 3 % mest in jih fakulteta sprejme po priporočilu ministrstva za višje šolstvo brez dodatnih preizkušenj. Velika selekcija pri vpisu ima za posledico, da osip v nadaljnjem študiju ne presega 2 %.

Čeprav je učni program matematike v vseh sovjetskih splošnih srednjih šolah enoten, vendarle obstajajo določene razlike v znanju učencev. Zato je MEH-MAT razvila vrsto dejavnosti, da bi omogočila vsem, ki so dovolj sposobni in želijo študirati matematiko, približno enake začetne pogoje. S tem namenom deluje pri fakulteti matematična šola, imenovana MALI MEH-MAT, v kateri lahko mladi Moskovčani ob sobotah poglabljajo svoje matematično znanje. Poskrbljeno je tudi za učence iz bolj oddaljenih krajev, saj Moskovčani predstavljajo le okoli tretjino vseh študentov na MGU. Matematični časopis Kvant, ki izhaja mesečno in sodi med najboljše periodične izdaje te vrste na svetu, objavlja vsako leto »sprejemne izpite« za tako imenovano Zvezno dopisno matematično šolo (VZMŠ). Tistim, ki uspešno rešijo predpisane naloge, prične nato MEH-MAT redno pošiljati brošure, v katerih so podrobneje obravnavana določena poglavja iz elementarne matematike in nekatere osnove višje matematike (reševanje enačb in neenačb, funkcije in

---

To je drugi del referata, ki ga je imel avtor na občnem zboru Društva v Cerknem. Prvi del je bil objavljen v prejšnji številki Obzornika.

njihovi grafi, limite itd.). Dopisni učenci VZMŠ rešujejo naloge iz teh brošur in jih pošiljajo na MEH-MAT, kjer jih pregledujejo in ocenjujejo študenti višjih letnikov in asistenti. Dopisna šola traja tri leta in tisti, ki jo uspešno končajo, prejmejo diplomo, ki daje olajšave pri vpisu na fakulteto. Najboljši med njimi prejmejo tudi povabila na enomesečni pripravljalni tečaj na sami MEH-MAT neposredno pred pričetkom sprejemnih izpitov.

Po reviji Kvant in VZMŠ se je pričelo tudi moje prvo srečanje z MEH-MAT MGU. Kot dijak idrijske gimnazije sem zasledil omenjeni natečaj in ker mi je uspelo naloge rešiti, sem jih sklenil poslati na navedeni naslov, ne da bi resnično verjel v odgovor. Pol leta kasneje sem dobil obvestilo, da sem uspešno opravil sprejemni izpit in pričelo se je redno dopisovanje, ki se je končalo s tem, da sem poleg diplome spomladi 1974. leta prejel tudi vabilo, da se udeležim poletnega pripravljalnega tečaja.

Obisk na MEH-MAT je napravil name velik vtis. Pozanimal sem se za možnosti vpisa tujih študentov in vložil prošnjo na našem Republiškem sekretariatu za prosveto. Mojo odločitev je podprl tudi štipenditor Inštitut Jožef Stefan.

Priznati moram, da je bilo prvo leto študija zame dokaj naporno. Predavanja in vaje so trajala po 6 do 8 ur dnevno. Proste dneve smo imeli le ob nedeljah, ob izpitih pa so bile zasedene tudi te. V dveh izpitnih rokih (v januarju in juniju) je bilo treba opraviti 9 izpitov, med samim šolskim letom pa prav toliko kolokvijev. V 34 tednih, kolikor jih obsegata oba semestra, smo v prvem letniku imeli 272 ur analize, 218 ur algebre in geometrije, 126 ur analitične geometrije, 64 ur matematične logike, 120 ur marksizma, 72 ur tujega jezika in 136 ur telesne vzgoje. O vsem tem in o rokih za kontrolne naloge, kolokvije in izpite smo bili podrobno obveščeni v posebni brošuri na začetku leta. Ta je vsebovala tudi nadrobne učne načrte. Tak tempo študija je mogoče vzdržati predvsem zaradi dobre organizacije bivanja, prehrane in dela na MEH-MAT. Nemoskovčani, ki so sprejeti na MGU, imajo zagotovljeno sobo v študentskem naselju, ki premore preko 10 000 mest. Domovi, upravni prostori, predavalnice in športno rekreacijski kompleks univerze tvorijo enotno študentsko mesto v malem, v katerem se novinci brez težav hitro vključijo v kolektivno delo. Univerzitetna knjižnica, ki nosi ime Maksima Gorkega, ima vso potrebno znanstveno in leposlovno literaturo; študentje jo lahko vzamejo s seboj ali pa predelujejo v udobnih čitalnicah. Redna predavanja dopolnjuje množica popoldanskih akademskih seminarjev, ki jih vodijo svetovno znani matematiki in na katerih se diskusije o aktualnih vprašanjih teorije in prakse večkrat zavlečejo pozno v noč.

Študij je dvostopenjski. Prvi dve leti študenti spoznavajo osnove moderne matematike, nato pa nastopi specializacija in si vsak izbere tisto področje, na katerem namerava kasneje delati.

Za primer osnovnih predmetov navajam snov iz višje algebre, katere obdelavi sta namenjena dva semestra: sistemi linearnih enačb; R^n, linearna neodvisnost, baza; matrike, linearne mnogoterosti; determinante; binarne operacije, grupe, morfizmi; kolobarji in obsegi; C, kolobar polinomov; ničle polinoma, simetrični polinomi; osnovni teorem algebre; ortogonalne in unitarne grupe; delovanje grupe na množici; enostavne grupe; teoremi Sylowa; končne Abelove grupe; linearne upodobitve grup, karakter upodobitve; nerazstavljive upodobitve, tenzorski produkt upodobitev.

V drugi etapi študija, ki traja tri leta, je glavna vloga prepuščena individualnemu delu študenta in njegovega mentorja, ki je priznan strokovnjak na izbranem področju. Mentor zasleduje premike v reševanju danih problemov v literaturi, svetuje svojemu varovancu dodatno literaturo, mu postavlja konkretne še nerešene naloge in pomaga pri publikacijah. Na ta način večina študentov v zadnjih treh letih uspe prodreti v osrčje sodobne matematične znanosti in se usposobiti za nadaljnje samostojno raziskovalno-ustvarjalno delo. Naj s primerom razporeda ur v četrtem letniku potrdim tako usmeritev.

V četrtem letniku je 32 tednov predavanj, skupno 756 ur, ki so tako porazdeljene:

— numerične metode in programiranje (128 ur)
— optimalno upravljanje (72 ur)
— teoretična mehanika (72 ur)
— kvantna mehanika (128 ur)
— specialni tečaji (4 semestri) (128 ur)
— zgodovina matematike (28 ur)
— marksizem (100 ur)
— tuji jeziki (36 ur)
— telesna vzgoja (64 ur)

Študij vključuje še prakso, ki traja en semester in jo lahko študent opravlja v kakem raziskovalnem inštitutu, delovni organizaciji ali kar na univerzi. Na koncu čakajo šudente poleg zagovora diplomske naloge še zaključni izpiti iz matematike, tujega jezika in filozofije. Študent, ki je v vsem času študija dobil 75 % maksimalnih ocen ter pri zaključnih izpitih, pri praksi in pri zagovoru diplomske naloge oceno odlično, prejme ob zaključku posebno »rdečo« diplomo. Ta mu daje prednost pri zaposlovanju, pa tudi odpira vrata na 3. stopnjo, v »aspiranturo«. Seveda ga kljub temu ob vpisu še čakajo zahtevni izpiti.

Med študijem na tretji stopnji morajo študenti opraviti še 7 izpitov »kandidatskih minimov«, objaviti v tisku dva članka na temo svoje disertacije in sodelovati na fakulteti z določenim pedagoškim delom. Njihova glavna obveznost pa je zahtevno raziskovalno delo pod vodstvom mentorja, kar naredi podiplomski študij vsa tri leta dokaj intenziven. Študij se konča z zagovorom disertacijskega projekta pred 17-člansko komisijo, ki jo sestavljajo profesorji različnih visokih šol.

Na koncu moram povedati, da je name napravilo močan vtis izredno sodelovanje tako med študenti samimi, kot med študenti in profesorji, kar je nedvomno ustvarilo ozračje, v katerem smo radi delali.

Dušan Pagon

LITERATURA

Spravočnik dlja postupajuščih v Moskovskij universitet, Moskva, Izdatelstvo Moskovskogo universiteta, 1983.

DOMAČE VESTI

SEMINAR UMETNA INTELIGENCA

CAS (Center for Advanced Studies) iz Beograda in ETAN/SVI prirejata mednarodni poletni seminar z naslovom Umetna inteligenca od 27. avgusta do 1. septembra v Dubrovniku (hotel Palace).

Raziskovanje umetne inteligence se je zadnje čase močno razmahnilo. Vzpodbudila sta ga na eni strani uspešna uporaba nekaterih metod umetne inteligence in na drugi razvoj z njo povezanih računalnikov pete generacije. Na seminarju bodo predavali priznani strokovnjaki s tega področja: P. H. Winston in T. Lozano-Perez z massachusettskega tehnološkega inštituta, R. Michalski z univerze zvezne države Illinois, D. Michie z edinburške univerze, E. Haikova s praške univerze in I. Bratko z ljubljanske univerze. Poročali bodo o sedanjem stanju po znanstveni in tehnični plati, o razvoju in posledicah.

Domači udeleženci morajo prispevati za kotizacijo 15 000 dinarjev na žiro račun 60803-678-12115, tujci pa 250 dolarjev na: Beobanka — Beograd — Yugoslavia, 60811-620-16-151-25733-421-01062, Yugoslav Society for ETAN (z oznako: For Seminar Artificial Intelligence).

Organizacijski odbor vabi na seminar. Prijave sprejema Center for Advances studies, Slobodan Janković, Kneza Miloša 9/IV, p. p. 56, 11001 Beograd. Pojasnila lahko dobite tudi pri Tanji Majaron na Institutu J. Stefan.

Tanja Majaron

NOVI ČLANI DMFA SRS V LETU 1983

1322. Bandelj Elido
1323. Blatnik Tone
1324. Burja Vladislava
1325. Butina Marta
1326. Cehner Dragica
1327. Cugelj Marjeta
1328. Dremelj Janka
1329. Fonda Magda
1330. Forca Muha Eda
1331. Golob Jožica
1332. Gregorc Dragica
1333. Gril Lidija
1334. Herljević Andjelko
1335. Kakhel Emil
1336. Jurinčič Irena
1337. Kelšin Dragutin
1338. Klanjšek Marta
1339. Kramberger Miran
1340. Koprivec Franc
1341. Majcenovič Marija
1342. Marinček Miloš in Marko
1343. Mehonić Ahmet
1344. Močnik Zvonka
1345. Kreft Vanja
1346. Munih Marko in Sonja
1347. Omerzel Doris
1348. Peitler Ivanka
1349. Pfajfar Barbara
1350. Pupis Vesna
1351. Primožič Ana
1352. Rosa Irena
1353. Šmigoc Marija
1354. Tkalčič Diomira
1355. Zver Marija

Po nekaj letih je ostalo število naročnikov in članov društva nespremenjeno. 34 članov je prenehalo prejemati društveno revijo zaradi smrti, odpovedi ali pa zaradi nerednega plačevanja naročnine in članarine. Slednjim smo ustavili pošiljanje Obzornika za matematiko in fiziko zato, ker so dolžni plačati naročnino že za dve leti. O njihovi izključitvi iz članstva društva je razpravljal UO DMFA SRS. Prav toliko pa je novih članov, večinoma zunaj ljubljanskega območja. Člani družine lahko prejemajo le en izvod revije in plačujejo le eno članarino. Danes šteje naše društvo že 876 članov. V zadnjem letu se je na Obzornik naročilo tudi 18 novih študentov. V posebni akciji smo povabili v društvo tudi študente matematike in fizike na pedagoških akademijah v Ljubljani in Mariboru.

Ciril Velkovrh

PODIPLOMSKI SEMINAR ZA UČITELJE MATEMATIKE

VTO Matematika in mehanika pripravlja v šolskem letu 1984/85 podiplomski seminar za učitelje matematike. Seminar bo kasneje prerasel v podiplomski študij. Ob februarskem seminarju DMFA smo z anketo ugotovili, da bi bilo mogoče izvesti seminar ob petkih popoldne.

V šolskem letu 1984/85 bo predaval profesor I. Vidav, Izbrana poglavja iz teorije mere. Profesor N. Prijatelj bo vodil seminar funkcionalne analize. To pomeni skupaj 4 ure vsak petek popoldne.

Z Zavodom SR Slovenije za šolstvo smo se dogovorili za pomoč pri obveščanju. Obvestila in obrazec za vpis bomo poslali v maju vsem srednjim šolam v Sloveniji. Seveda vabimo tudi profesorje matematike, ki učijo na osnovnih šolah.

Prijave pričakujemo do 15. septembra 1984.

Peter Petek

VPRAŠALNIK

Ime: .....

Priimek: .....

Naslov (doma in na delovnem mestu)

.....

.....

Delovna organizacija: .....

Delovno mesto: .....

Sem član DMFA DA / NE

Želim se vključiti v delo sekcije za uporabno matematiko in bom poravnal članarino ob prejemu prve številke OBZORNIKA
DA / NE

Še nimam revije PRESEK in se želim nanjo naročiti
DA / NE

Prijavljam se na seminar, ki bo v četrtek, 20. in petek, 21. septembra 1984, v Ljubljani, Jadranska 21, drugo nadstropje (stavba RRC).
DA / NE

Prispevek za seminar v višini 2000 din (vključuje pisne materiale, tople napitke in stroške družabnega večera dne 20. 9. 1984)

a) bom nakazal na žiro račun Društva matematikov, fizikov in astronomov SRS — Ljubljana, št. 50101-678-49168
b) bom poravnal ob prihodu na seminar.

Izpolnjen vprašalnik pošlji na naslov:

DMFA SRS, Sekcija za uporabno matematiko,
Jadranska 19, 61111 Ljubljana, p. p. 64

SEMINAR SEKCIJE ZA UPORABNO MATEMATIKO DMFA SRS

Dragi kolega, draga kolegica!

Že dolgo si želimo, da bi pri Društvu matematikov, fizikov in astronomov SR Slovenije zaživela sekcija za uporabno matematiko. Njen osnovni cilj je strokovna povezava diplomantov uporabne (prej tehniške) matematike. Radi bi tudi izrabili izkušnje uporabnih matematikov v praksi za izboljšanje sedanjih učnih programov.

Če še nisi član društva, Te prosimo, da se čimprej včlaniš in po možnosti aktivno sodeluješ pri delu te sekcije. Vsi člani DMFA prejemajo revijo OBZORNIK ZA MATEMATIKO IN FIZIKO za članarino, ki je v letu 1984 samo 450 dinarjev. Vsako leto je tudi občni zbor društva, ki ima poleg uradnega tudi družabni del.

Naša prva akcija je priprava dvodnevnega seminarja, na katerem se bomo prvič srečali in tudi dogovorili za oblike delovanja sekcije. Ob seminarju bomo izdali tudi bilten, v katerem bo povzetek vsebine seminarja.

Prilagamo vprašalnik in prosimo, da izpolnjenega vrneš, tudi če ne želiš postati član društva.

Ker se bojimo, da ne poznamo naslovov vseh diplomantov uporabne matematike, Te prosimo, da vprašalnik fotokopiraš in daš kolegu.

Pozdrav.

Sekcija za uporabno matematiko DMFA SRS

Nevenka Gorenšček

PROGRAM

Mesto seminarja: VTOZD MATEMATIKA IN MEHANIKA
61111 Ljubljana, Jadranska 19

ČETRTEK, 20. septembra 1984

9.00—10.15 Zvonimir BOHTE, Pregled metod za reševanje sistemov linearnih enačb
10.30—11.15 Jernej KOZAK, Metoda najmanjših kvadratov
11.45—12.30 Matjaž OMLADIČ, Linearna regresija
16.00—18.00 Pogovor
20.00 Družabni večer

PETEK, 21. septembra 1984

9.00—10.15 Vladimir BATAGELJ, Razvrščanje v skupine
10.30—11.15 Bojan MOHAR, Modeliranje z grafi
11.45—12.30 Tomaž PISANSKI, Kombinatorična optimizacija
16.00—18.00 Praktično delo: V. BATAGELJ, Razvrščanje v skupine

TEME POGOVORA

1. Delo uporabnih matematikov na delovnem mestu in problemi, ki jih rešujejo matematične metode
2. Teme za bodoče seminarje
   — Oblika bodočih seminarjev (referati iz prakse?)
3. Kakšno naj bo delovanje Sekcije za uporabno matematiko
   Nekaj predlogov v premislek za debato:
   — Seznanjanje z novimi področji študija uporabne matematike
   — Vloga sekcije pri sodelovanju fakultete in IMFM z gospodarstvom
   Sodelovanje z drugimi matematičnimi fakultetami
   Možnost sodelovanja z Gospodarsko zbornico oz. podobnimi ustanovami za približanje uporabne matematike praksi
   Sodelovanje pri diplomskih delih
   — Odpiranje sredinih seminarjev vsem članom sekcije (tudi zunaj Ljubljane), npr. letno poročilo o temah, ponovitve najzanimivejših tem
   — Kje in kakšne so možnosti objavljanja rezultatov dela s področja uporabe matematike v praksi