Mr Nina Lipljin, znanstveni asistent
Fakultet organizacije i informatike
V a r a ž d i n

UDK 381.32
Stručni rad

I Z R A D A I P R O C E S O P T I M A L I Z A C I J E R J E Š E N J A J E D N O G P R O G R A M S K O G P R I M J E R A

U okviru ovog rada autorica je prezentirala postupak izrade i proces optimalizacije rješenja jednog programskog primjera.

U prvoj fazi izrade programskog rješenja polazi se od općenitog pristupa odvijanju automatizirane obrade podataka prema detaljnom raščlanjivanju zadanog problema, što je rezultiralo prvim (detaljnim) rješenjem programskog primjera.

Druga faza razradjuje postupak optimalizacije, čiji je krajnji rezultat drugo, optimalno rješenje. Ono je kvalitetnije, sažetije i kraće. Analizom broja naredbi i kvalitativnih osobina predloženog optimalnog rješenja dokazano je da je proces optimalizacije proveden s razlogom te da je ostvareno dosta vrlo vrijednih poboljšanja, pri čemu su očekivani rezultati obrade ostali nepromijenjeni.

1. UVOD

Postupak organizacije automatizirane obrade podataka u onom dijelu koji se odnosi na izradu programskog rješenja može se podijeliti u dva dijela:

- organizacija operacija, odnosno koraka koji su sastavni dio same obrade, te
- prilagodjavanje dobivenog rješenja konkretnom programskom jeziku koji se koristi.

Ukoliko su ova dva dijela pravilno izvedena, treći dio koji se odnosi na pisanje programskih naredbi (tzv. kodiranje)predstavlja rutinski posao i obično se ne smatra kreativnim postupkom.

Pokušat ćemo na jednom konkretnom programskom primjeru pokazati na koji način se provode prva dva, tzv. kreativna dijela postupka organizacije automatizirane obrade podataka.U skladu s

tim bit će predložena i dva rješenja spomenutog programskog primjera. Od ta dva rješenja jedno je optimalno i predstavlja krajnji, očekivani rezultat.

2. OPIS PROGRAMSKOG ZADATKA

Zadatak programa je kreiranje sekvencijalno organizirane datoteke na magnetskom disku.

Postupak obrade

Program učitava kartice. Datoteka je sortirana uzlazno prema vrijednostima šifre artikla. Od podataka sadržanih na karticama s jednakom vrijednošću šifre artikla program komponira jedan izlazni slog i ispisuje ga na magnetski disk. Količina artikla u izlaznom slogu predstavlja sumu ulaznih količina. Iznos se izračunava množenjem sume ulaznih količina i cijene artikla. U toku obrade broje se učitani slogovi i slogovi ispisani na magnetski disk. O tome se na kraju obrade daje obavijest putem poruka na konzoli. Format poruka:

```
BROJ PROČITANIH SLOGOVA: ZZZBZZ9
BROJ NAPISANIH SLOGOVA:  ZZZBZZ9
```

Pregled i opis datoteka

1. Ulazna datoteka KARTICE na čitaču kartica, sortirana uzlazno prema vrijednostima šifre artikla. Moguća je pojava većeg broja slogova s jednakom vrijednošću šifre artikla. Opis sloga:

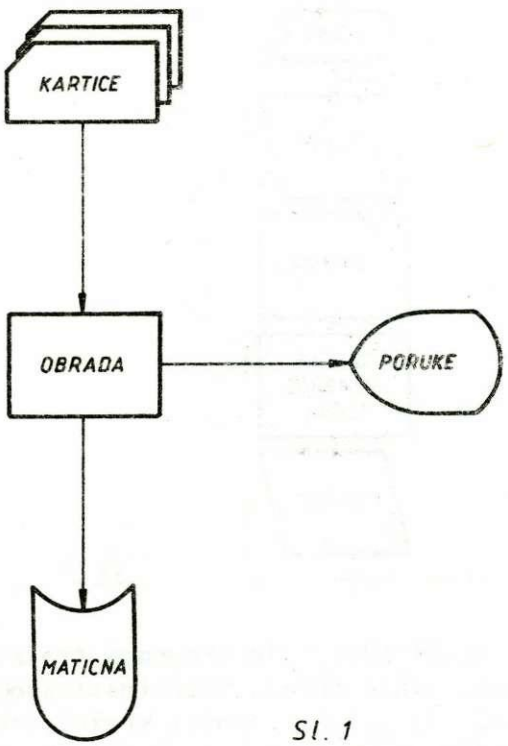
Naziv polja	Dužina	Tip	Primjedba
VK	3	N	Vrst kartice
SIFRA	5	N	Šifra artikla
NAZIV	25	AN	Naziv artikla
KOLICINA	5	N	Količina ulaza
CIJENA	5(2)	N	Cijena artikla
FILLER	53	AN	Neiskorišteni dio kartice

2. Izlazna datoteka MATICNA na magnetskom disku, sekvencijalne organizacije i pristupa, standardnih labela, limit datoteke 5000 slogova, faktor blokiranja 20 slogova. Opis sloga:

Naziv polja	Dužina	Tip	Primjedba
VS	3	N	Vrst sloga
SIF	5	N	Šifra artikla
NAZ	25	AN	Naziv artikla
SUMA-KOL	6	N	Suma ulaznih količina
CIJ	5(2)	N	Cijena artikla
IZN	11(2)	N	Iznos u dinarima

Napomena: opisi slogova prilagodjeni su tipu elektroničkog računala (BURROUGHS B 1700) koje je predvidjeno za izvodjenje programa, te programskom jeziku (COBOL) u kojem će rješenje programskog primjera biti napisano.

3. SISTEMSKI DIJAGRAM TOKA OBRADE



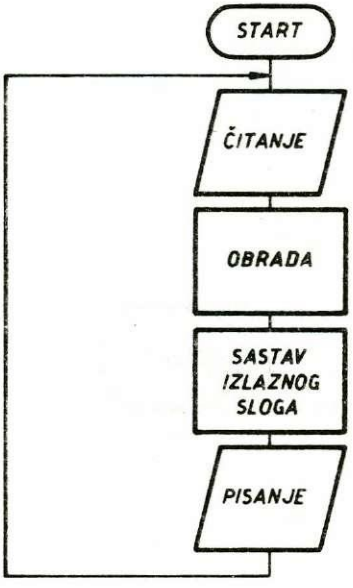
Sl. 1

4. ANALIZA PROGRAMSKOG PRIMJERA

Prilikom analize programskog primjera uvijek se polazi od općenitih pretpostavki zadanog problema koje se kasnije detaljno razradjuju. Polazne točke, odnosno općenite pretpostavke ovog primjera jesu:

- učitavanje ulaznih podataka,
- obrada ulaznih podataka,
- sastavljanje izlaznog sloga i
- ispisivanje izlaznog sloga.

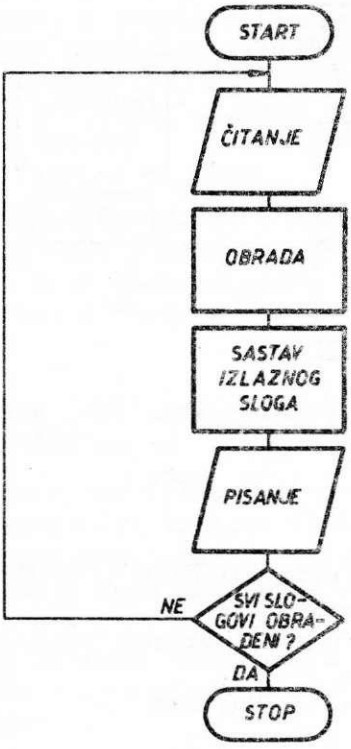
Navedeni elementi sačinjavaju zajedničke parametre obrade svakog sloga. Njihovim objedinjavanjem stvaraju se uvjeti za odvijanje kružnog toka programa (sl.2), što je osnovna pretpostavka automatizirane obrade podataka:



Sl. 2

Ovako promatrano (sl.2) odvijanje programa trajalo bi vremenski beskonačno dugo. Svaka obrada mora imati svoj početak i svoj kraj, što znači da jednom započet kružni tok odvijanja obrade mora u odredjenom momentu biti prekinut. Logičan slijed razmišljanja dovodi do potrebe determiniranja trenutka prekida odvijanja neke obrade, tj. do odredjivanja njezinog kraja. To je faza u kojoj je ustanovljeno da su svi podaci

obradjeni, pa bi daljnje odvijanje programa bilo ne samo besmisleno već i pogrešno. Grafički prikaz odvijanja obrade,gdje kružni tok više nije beskonačno iteriran (sl.3), predstavlja ujedno i početak sastavljanja detaljnog dijagrama toka obrade za programski primjer koji je predmet analize:

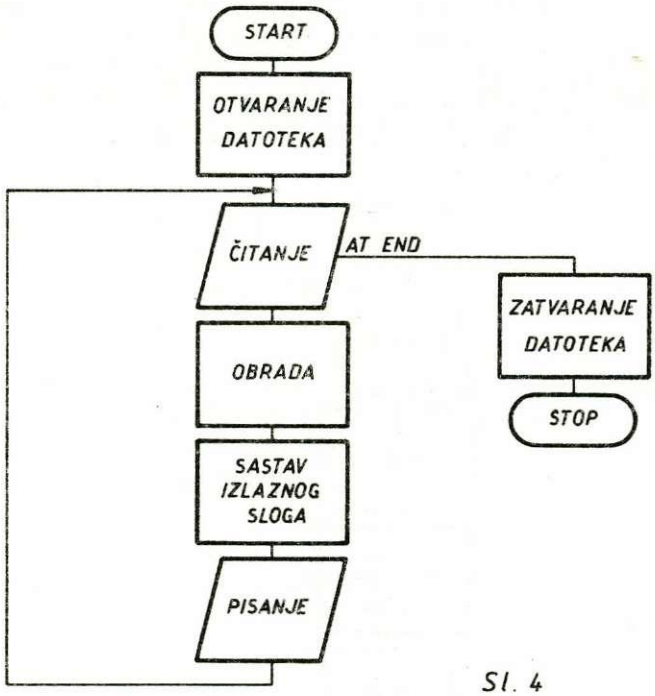


Sl. 3

Program za koji se sastavlja detaljni dijagram toka obrade bit će napisan u programskom jeziku COBOL, pa je crtež, odnosno grafički prikaz potrebno tome prilagoditi (sl.4).

Predložen dijagram toka obrade predstavlja polaznu točku za rješavanje zadanog problema.

U slijedećoj fazi utvrdjuju se ostali programski elementi, a to su:
- otvaranje i zatvaranje datoteka,
- čitanje slogova ulazne datoteke KARTICE,
- brojanje pročitanih slogova,



Sl. 4

- obrada koju sačinjava zbrajanje količina ulaznih slogova iste šifre i izračunavanje iznosa,
- usporedjivanje ulaznih šifri zbog potrebe grananja programa,
- sastavljanje izlaznog sloga,
- pisanje izlaznog sloga datoteke MATICNA,
- brojanje napisanih slogova i
- na kraju obrade, ispisivanje poruka putem konzole.

Ovako nanizani elementi programa ne mogu svojim redoslijedom jamčiti pravilan tok odvijanja obrade. Redoslijed elemenata programa mora biti logički povezan, na temelju njihove logičke uvjetovanosti.

Korisno je elemente programa promatrati s navedenog aspekta logičke uvjetovanosti u odnosu na mogućnost pravilnog odvijanja obrade. Na taj način analiziramo ih kao:

- skupinu elemenata koji mogu bitno utjecati na pravilan tok odvijanja obrade i

- skupinu elemenata o kojima treba voditi računa, ali koji ne mogu bitno utjecati na pravilan tok odvijanja obrade.

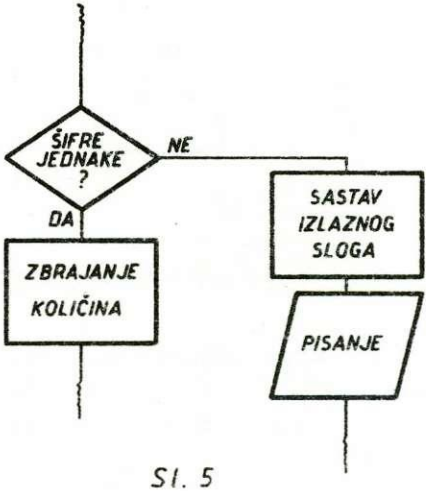
U svakom slučaju, ključ za dobro logičko rješenje programskog zadatka predstavlja skupina bitnih elemenata.

U promatranom programskom primjeru ključni problem nalazi se u pravilnom izboru mjesta i načina usporedjivanja ulaznih šifri. Čitanje ulaznih podataka, njihov transfer i smještaj u onaj prostor centralne memorije, koji je unaprijed osiguran (rezerviran) opisom u DATA DIVISION, predstavlja prvi korak rješavanja ključnog problema.

Polazeći od činjenice da se u ulaznoj datoteci može pojaviti veći broj slogova s jednakom vrijednošću šifre artikla, od kojih se stvara samo jedan izlazni slog, moguće je zaključiti da je ustanovljavanje momenta promjene šifre važno iz više razloga:

- tako dugo dok pročitani slogovi imaju istu šifru, kumuliraju se ulazne količine, a
- u momentu promjene šifre (ali prije slijedećeg čitanja) potrebno je sastaviti izlazni slog kombinirajući vrijednosti prethodnih slogova iste šifre (VK, SIFRA, NAZIV i CIJENA) sa sadržajem polja koja nose sumu količina i iznos.

Evo grafičkog prikaza ovog mjesta programa (sl.5) koji je potrebno uklopiti u cjelinu tako da se ne poremeti kružni tok odvijanja obrade:

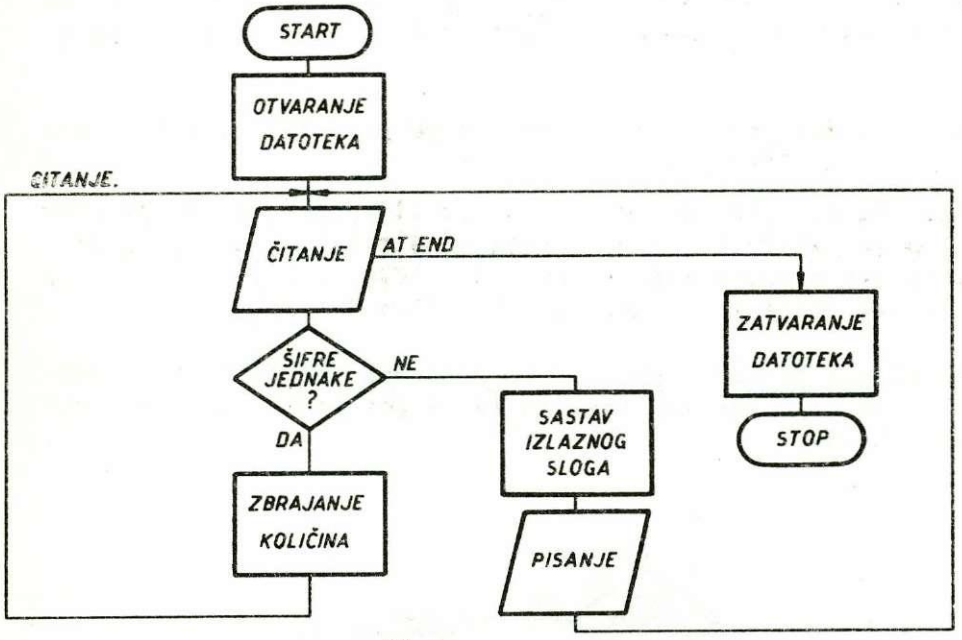


Sl. 5

Prilikom učestalih čitanja slogova ulazne datoteke novopročitani slogovi prekrivaju (na izvjestan način brišu) podatke prethodno pročitanog sloga. Nasuprot tome, usporedba mora sadržavati dva elementa čijim ispitivanjem postižemo rezultate usporedjivanja. U predloženom fragmentu detaljnog dijagrama (sl.5) usporedba JEDNAKE ŠIFRE ima značenje:

> "Usporedi upravo pročitanu šifru s onom koja je pročitana neposredno prije nje!"

Postavlja se suštinsko pitanje: kako izvršiti usporedjivanje dviju šifara kad je poznato da je novopročitana prekrila prethodnu? Na ovom stupnju analize programskog primjera dijagram toka obrade izgledao bi ovako (sl.6):



Sl. 6

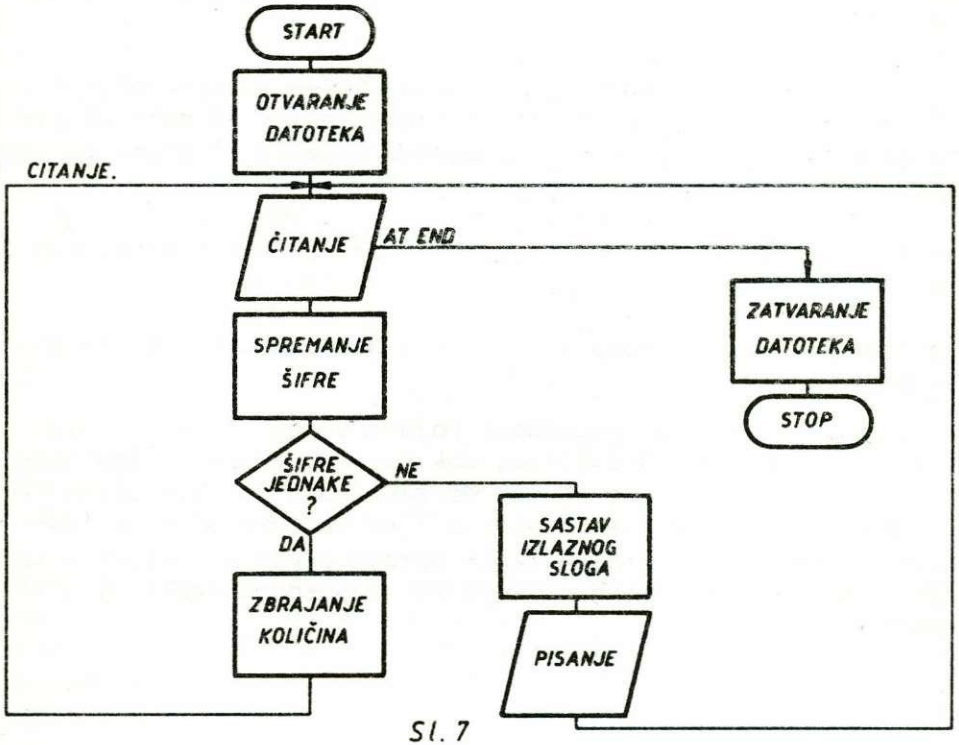
Predloženo rješenje (sl.6) nije u skladu s problemom koji se pokušava riješiti:

- rezultati obrade neće nikada odgovarati stvarnom stanju,
- ovako postavljena usporedba utječe na usmjeravanje programa uvijek u pravcu DA jer dovodi do usporedjivanja šifre sa samom sobom,

- uslijed toga nikad neće doći do sastavljanja i pisanja izlaznog sloga, količine će se zbrajati bez obzira na stvarnu promjenu šifre, a
- rezultati obrade bit će pogrešni (suma svih ulaznih količina nalazit će se u memoriji, a izlazna datoteka, koja je trebala biti kreirana programom, neće se pojaviti).

Evidentno je da ovako postavljena usporedba nije našla pravo mjesto u programskom rješenju. Analiza predloženog dijagrama toka obrade (sl.6) može pokazati da je njegov osnovni nedostatak u tome što se šifra uspoređuje sa samom sobom, umjesto da se novopročitana šifra usporedi s onom koja je pročitana neposredno prije nje.

Izlaz iz ove naoko zamršene situacije nalazi se u PRIVREMENOM ČUVANJU PRETHODNE ŠIFRE. To znači da pročitane šifre treba, prijenosom na neko drugo mjesto, privremeno sačuvati kako bi se mogla izvršiti usporedba prethodne šifre s novopročitanom. Tada će usporedjivanje, a samim tim i grananje programa, biti realno. Program bi se na ovom stupnju analize i izrade rješenja mogao odvijati ovako (sl.7):



Sl. 7

Čini se na prvi pogled kao da je problem riješen. Nažalost, već sasvim površna analiza otkriva kako nije učinjeno ništa drugo osim što je nepotrebno proširen crtež (sl.7), a time i program. Sada pitanje vezano za usporedbu doduše glasi kako treba:

"Da li je pročitana šifra jednaka spremljenoj?"

Pitanje je postavljeno pravilno, no, kako se "spremanje šifre" ne nalazi na dobrom mjestu, ovo se pitanje suštinski ne razlikuje od prethodnog, osim u jednom:

- ranije smo vršili usporedbu vrijednosti jednog polja s vrijednošću tog istog polja, a
- sada usporedjujemo vrijednost jednog polja s vrijednošću drugog.

Medjutim, kako se vrijednost prvog polja nipošto ne može razlikovati od vrijednosti drugog (prijenos vrijednosti iz jednog u drugo polje vrši se odmah nakon čitanja, a neposredno prije same usporedbe), smisao, a time i rezultat usporedjivanja, ostaje isti - pogrešan.

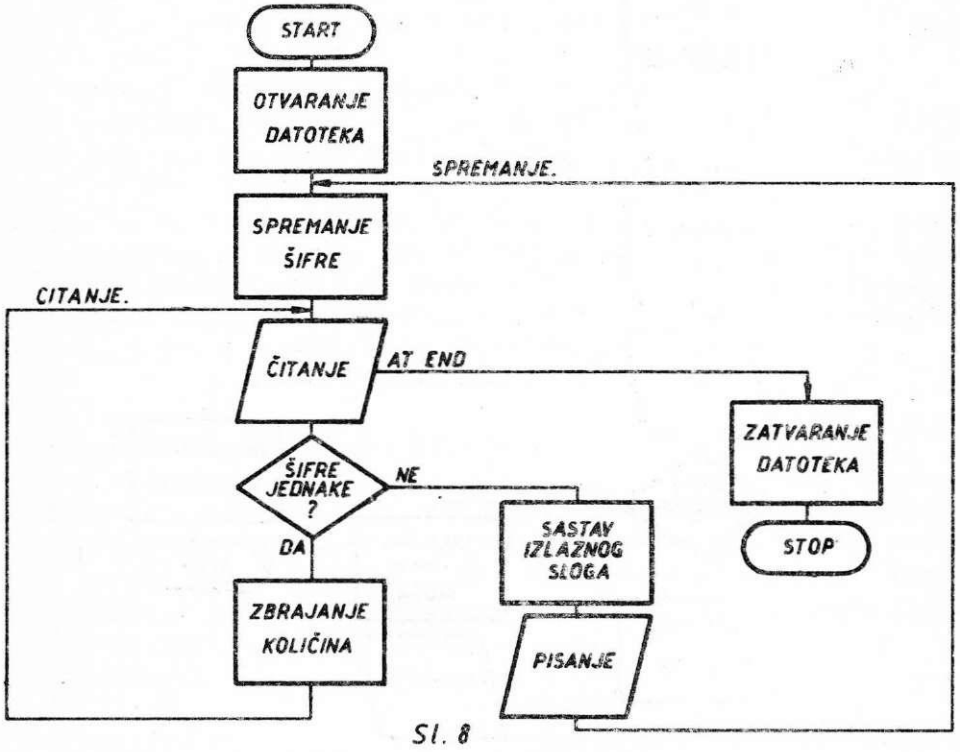
Spoznaja o potrebi postojanja posebnog polja za spremanje šifre i nefunkcionalnost predloženog rješenja (sl.7) može ukazivati samo na jedno - mjesto u programu koje je odabrano za spremanje šifre ne odgovara svojoj namjeni.

Analizirajući bit problema moguće je doći do novih konstatacija:

- spremanje, tj. privremeno čuvanje svake pročitane šifre nije neophodno,
- činjenica da postoji mogućnost pojave većeg broja slogova s jednakom vrijednošću šifre, dok posebno značenje ima samo moment promjene šifre, dovodi do zaključka da nije bitno spremanje svih pročitanih šifara (jer su neke od njih jednake, prema tome, za njih će se obrada odvijati uvijek u istom pravcu), već samo onih koje se medjusobno razlikuju, te nadalje

- odluka da se odvojeno promatra spremanje šifre od samog postupka čitanja otvara mogućnost da se te dvije operacije razdvoje vremenski i prostorno.

Kako će se u nekim slučajevima (jednake šifre) petlja ponavljati samo od operacije čitanja, a u nekim (promjena šifre) još i od operacije spremanja šifre, u dijagramu toka obrade potrebno je operaciju spremanja staviti ispred čitanja, da bi kružni tok odvijanja programa bio logičan i neopterećen prekomjernim detaljima (sl.8):

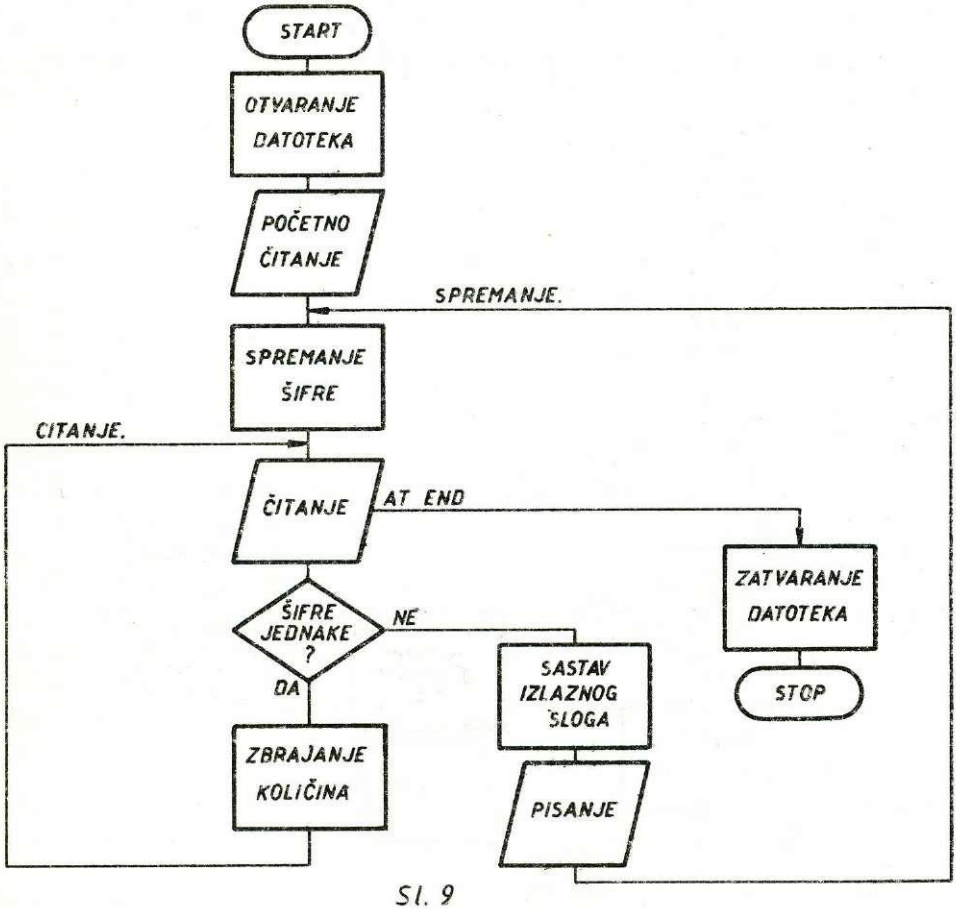


Sl. 8

Ovakvom rješenju (sl.8), koje je već vrlo blizu dobrog, može se staviti samo jedan prigovor:

- polazeći od dogovorene pretpostavke čitanja dijagrama toka obrade (odozgo prema dolje) postavlja se pitanje: što će se pojaviti na samom početku programa u polju predvidjenom za spremanje šifre kad se pouzdano zna da još nije došlo do izvršenja ni jedne naredbe čitanja?!

Odgovor na ovo pitanje vrlo je jednostavan: da bi se šifra mogla prenijeti u predvidjeno polje, treba je najprije pročitati. Realizacija ovog razmišljanja nikako ne predvidja ponovno premještanje čitanja ispred operacije spremanja šifre. Izlaz pronalazimo u postavljanju posebne operacije početnog čitanja koje neće biti sastavni dio kružnog odvijanja obrade (sl.9):



Sl. 9

PREDLOŽENI DIJAGRAM TOKA OBRADE PREDSTAVLJA DOBRO RJEŠENJE ZA DANOG PROBLEMA. Primjetno je da se prilikom njegova sastavljanja (sl.9) nije vodilo računa o detaljima (npr. brojanje slogova, poruke na konzoli i sl.), već samo o onoj skupini elemenata koja bitno utječe na pravilan tok odvijanja programa.

Nakon izrade dobrog rješenja detalje je vrlo lako ugraditi u program. U predloženom rješenju neke formulacije predstavljaju rješenje s općeg stajališta, tako da se to rješenje može dodavanjem detalja promijeniti, ali ne bitno.

Tako,na primjer, operacija "spremanje šifre" pretpostavlja privremeno čuvanje i ostalih elemenata sloga, koji bi takodjer bili prekriveni (izbrisani) novim čitanjem, a ti su elementi potrebni prilikom sastavljanja izlaznog sloga. Ako se programsko rješenje promatra iz tog ugla, neminovno slijedi zaključak da će se "sastav izlaznog sloga" izvoditi korištenjem "spremljenih", a ne upravo pročitanih elemenata. Elementi sloga spremaju se poslije čitanja, što dovodi do toga da će naredba čitanja utvrditi kraj ulazne datoteke prije nego što bude sastavljen i napisan posljednji slog. Za taj slog potrebno je predvidjeti posebnu obradu u okviru djelovanja klauzule AT END.

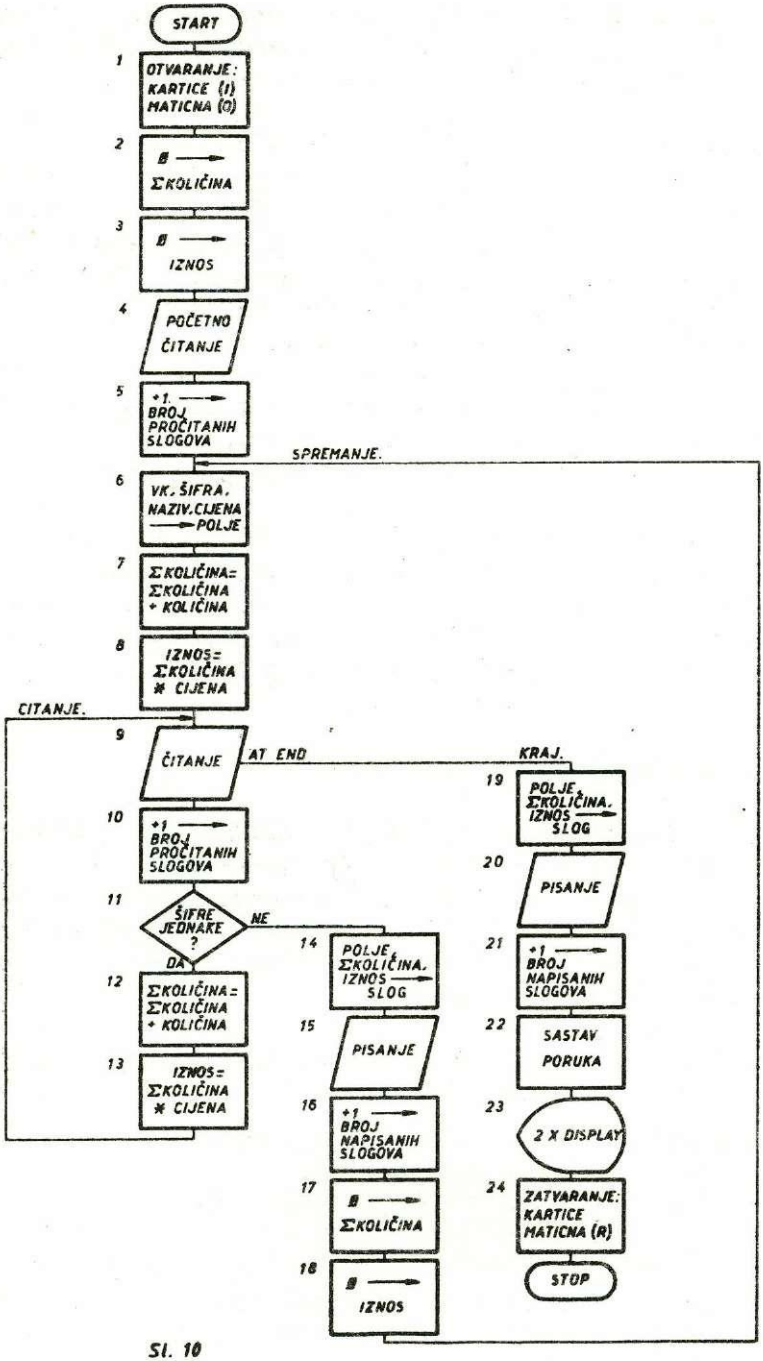
Nakon što je na ovaj način izvršena detaljna analiza programskog zadatka, moguće je izvršiti finalizaciju detaljnog dijagrama toka obrade, gdje su razradjene sve komponente programa. Predloženi dijagram toka obrade (sl.10) predstavlja prvo, detaljno rješenje promatranog programskog primjera.

5. IZNALAŽENJE OPTIMALNOG RJEŠENJA

Posljednju i završnu fazu rješavanja zadanog programskog zadatka predstavlja prilagodjavanje mogućnostima odredjenog programskog jezika, odnosno iznalaženje optimalnog rješenja. U konkretnom slučaju riječ je o analizi i konačnom dotjerivanju prvog, detaljnog rješenja (sl.10),što bi trebalo rezultirati optimalnim rješenjem. Postupak obuhvaća:

- prilagodjavanje detaljnog dijagrama toka obrade pravilima i mogućnostima programskog jezika,
- sintezu elemenata koji se ponavljaju nekoliko puta, ako je takav postupak moguć i opravdan, te
- izostavljanje suvišnih elemenata čija se realizacija može postići adekvatnim opisom podataka ili na neki drugi način.

Analizom predložene, detaljne varijante dijagrama toka obrade (sl.10)moguće je pronaći njezin optimalni,a time i konačni oblik.


START
1
OTVARANJE: KARTICE (I) MATICNA (O)
2
0 → ΣKOLIČINA
3
0 → IZNOS
4
POČETNO ČITANJE
5
+1 → BROJ PROČITANIH SLOGOVA
SPREMANJE.
6
VK. ŠIFRA, NAZIV. CIJENA → POLJE
7
ΣKOLIČINA= ΣKOLIČINA + KOLIČINA
8
IZNOS= ΣKOLIČINA * CIJENA
CITANJE.
9
ČITANJE
AT END
KRAJ.
10
+1 → BROJ PROČITANIH SLOGOVA
11
ŠIFRE JEDNAKE ?
NE
DA
12
ΣKOLIČINA= ΣKOLIČINA + KOLIČINA
13
IZNOS= ΣKOLIČINA * CIJENA
14
POLJE, ΣKOLIČINA, IZNOS → SLOG
15
PISANJE
16
+1 → BROJ NAPISANIH SLOGOVA
17
0 → ΣKOLIČINA
18
0 → IZNOS
19
POLJE, ΣKOLIČINA, IZNOS → SLOG
20
PISANJE
21
+1 → BROJ NAPISANIH SLOGOVA
22
SASTAV PORUKA
23
2 X DISPLAY
24
ZATVARANJE: KARTICE MATICNA (R)
STOP

Sl. 10

U smislu prilagodjavanja rješenja programskom jeziku COBOL može se mnogo toga učiniti. Posebno polje za spremanje podataka uopće neće biti potrebno jer izlazno područje sloga može vršiti ujedno i funkciju polja za privremeno čuvanje (spremanje)podataka ulaznih slogova s jednakom vrijednošću šifre. Dio izlaznog područja sloga, koji se odnosi na količinu, posebno će se tretirati. Količina, koja kao numeričko polje sudjeluje u operacijama zbrajanja, zahtijeva takodjer adekvatan postupak.

U predloženom detaljnom rješenju (sl.10) neki se lementi pojavljuju dva puta:

a) unošenje nula u polje SUMA KOLIČINA (2,17),
b) unošenje nula u polje IZNOS (3, 18),
c) brojanje pročitanih slogova (5, 10),
d) brojanje napisanih slogova (16, 21),
e) sastavljanje izlaznog sloga (14, 19),
f) pisanje (15, 20),
g) čitanje (4, 9),
h) izračunavanje vrijednosti polja SUMA KOLIČINA (7, 12) i
i) izračunavanje vrijednosti polja IZNOS (8, 13).

Neke od ovih pojava mogu se izbjeći bilo njihovim zanemarivanjem, bilo spretnijim sastavljanjem dijagrama toka obrade.Medjutim, neka su ponavljanja nužna jer se njihovo odvijanje dogadja u raznim dijelovima programa i u različito vrijeme, a bez njih program bi bio manjkav.

Analiza ponovljenih elemenata detaljnog rješenja (sl.10) pokazuje:

a) unošenje nula u polje SUMA KOLIČINA može se pojednostaviti prikladnijim smještajem naredbe unutar obrade SPREMANJE,no taj se proces mora izvršiti prije početka obrade CITANJE,

b) unošenje nula u polje IZNOS može se zanemariti pod pretpostavkom da se to polje tretira kao mjesto rezultata, tako da se prilikom izračunavanja iznosa koristi klauzula GIVING,

c) brojanje pročitanih slogova moguće je smjestiti unutar obrade CITANJE, ali prije same naredbe čitanja,čime se postiže efekt pribrajanja i prvog pročitanog sloga,

d) brojanje napisanih slogova neophodno je izvršavati dva puta, jednom unutar petlje, a drugi puta u okviru obrade KRAJ, kako bi se pribrojio i posljednji slog,

e) sastavljanje izlaznog sloga može se zanemariti ukoliko se prihvati varijanta da područje izlaznog sloga predstavlja ujedno i polje za spremanje i privremeno čuvanje podataka,

f) pisanje izlaznog sloga mora se izvršiti dva puta iz već navedenih razloga (jednom u petlji, a drugi puta unutar obrade KRAJ, kako bi se napisao i posljednji slog),

g) čitanje nije moguće pojednostaviti jer jedno od njih predstavlja početno čitanje, a drugo čitanje u petlji (ovako smještene naredbe čitanja čine okosnicu optimalnog rješenja),

h) izračunavanje vrijednosti polja SUMA KOLIČINA može se simplificirati ako se ova naredba smjesti unutar obrade ČITANJE, ali prije same naredbe čitanja, kako bi se pribrojila i količina prvog pročitanog sloga,

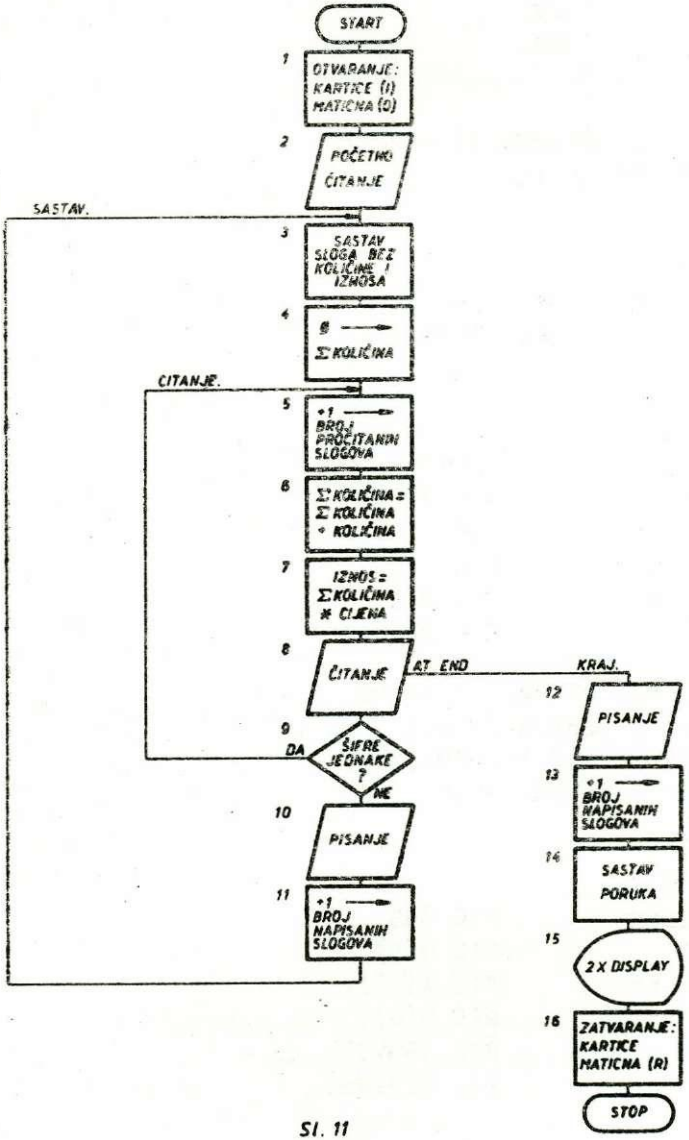
i) izračunavanje vrijednosti polja IZNOS nije potrebno izvršavati na dva mjesta ukoliko se ta aritmetička operacija smjesti neposredno iza izračunavanja vrijednosti polja SUMA KOLIČINA.

Iz svega ovdje navedenog proizlazi da je postojeće, detaljno rješenje (sl.10) moguće u mnogome pojednostaviti, a time i poboljšati. Elementi čije ponavljanje se ne može izbjeći jesu naredbe čitanja i pisanja te brojanje napisanih slogova. Ponavljanje bi se moglo eliminirati samo korištenjem naredbe PERFORM, ali takvo rješenje ne bi dovelo do bitnih kvalitativnih promjena. Veličina potprograma od samo dvije naredbe (MOVE i WRITE) ne umanjuje broj ukupno napisanih naredbi jer bi tada trebalo dopisati dvije naredbe PERFORM kojih sada nema. Naprotiv, upotreba naredbe PERFORM s potprogramima produžila bi vrijeme trajanja obrade.

Optimalno rješenje zadanog primjera (sl.11), koje neminovno proizlazi, rezultat je analize i izmjena detaljnog rješenja koje su provedene na osnovi te analize.

6. OPTIMALNO RJEŠENJE

6.1 Detaljni dijagram toka obrade



Sl. 11

6.2 Programski predložak optimalnog rješenja

```
IDENTIFICATION DIVISION.
PROGRAM-ID. OBRADA.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. B-17ØØ.
OBJECT-COMPUTER. B-17ØØ.
INPUT-OUTPUT SECTION.
FILE-CONTROL
    SELECT KARTICE ASSIGN TO READER.
    SELECT MATICNA ASSIGN TO DISK.
DATA DIVISION.
FILE SECTION.

FD  KARTICE, LABEL RECORD OMITTED,
             DATA RECORD KARTICA.

Ø1  KARTICA.
    Ø2  PRIJENOS.
        Ø3 FILLER          PIC 999.
        Ø3 SIFRA           PIC 9(5).
        Ø3 FILLER          PIC X(25).
        Ø3 KOLICINA        PIC 9(5).
        Ø3 FILLER          PIC 9(5).
    Ø2  FILLER             PIC X(53).

FD  MATICNA, LABEL RECORD STANDARD,
             FILE 5ØØØ RECORDS,
             BLOCK  5Ø RECORDS,
             DATA RECORD SLOG.

Ø1  SLOG.
    Ø2  PRIHVAT.
        Ø3 FILLER          PIC 999.
        Ø3 SIF             PIC 9(5).
        Ø3 FILLER          PIC X(25).
        Ø3 SUMA-KOL        PIC 9(6).
        Ø3 CIJ             PIC 999V99.
    Ø2  IZN                PIC 9(9)V99.
```

```
WORKING-STORAGE SECTION.

77  PROCITANI              PIC 9(6) VALUE ZERO.
77  NAPISANI               PIC 9(6) VALUE ZERO.
77  PORUKA-1               PIC ZZZBZZ9.
77  PORUKA-2               PIC ZZZBZZ9.

PROCEDURE DIVISION.
POCETAK.
    OPEN INPUT KARTICE, OUTPUT MATICNA.
    READ KARTICE.

SASTAV.
    MOVE PRIJENOS TO PRIHVAT.
    MOVE ZERO TO SUMA-KOL.
CITANJE.
    ADD 1 TO PROCITANI.
    COMPUTE SUMA-KOL = SUMA-KOL + KOLICINA.
    COMPUTE IZN      = SUMA-KOL * CIJ.
    READ KARTICE AT END GO TO KRAJ.
    IF SIFRA = SIF GO TO CITANJE ELSE
                   WRITE SLOG,
                   ADD 1 TO NAPISANI,
                   GO TO SASTAV.

KRAJ.
    WRITE SLOG.
    ADD 1 TO NAPISANI.
    MOVE PROCITANI TO PORUKA-1.
    MOVE NAPISANI  TO PORUKA-2.
    DISPLAY "BROJ PROCITANIH SLOGOVA:", PORUKA-1.
    DISPLAY "BROJ NAPISANIH SLOGOVA:",  PORUKA-2.
    CLOSE KARTICE, MATICNA RELEASE.
    STOP RUN.
END-OF-JOB
```

7. ZAKLJUČAK

Polazeći od pretpostavke da se svaki posao, pa tako i programiranje, može obavljati s manje ili više uspjeha, osnovna intencija ovog rada sastoji se u tome da se analizom jednog programskog primjera dokaže kako je optimalizacija moguća i korisna i na ovom području. Koncepcijski je razradjen pristup rješavanju zadanog problema koji se sastoji u postupnom raščlanjivanju programskog zadatka, što je rezultiralo uočavanjem bitnih elemenata i postojanjem njihove logičke uvjetovanosti.

Postupak rješavanja programskog primjera kretao se u prvoj fazi od općenitog pristupa odvijanju automatizirane obrade podataka prema detaljnom raščlanjivanju zadanog problema, pri čemu su eliminirani svi oni koraci koji bi mogli dovesti do pogrešnih konstatacija i rezultata. Dobiveno je prvo, detaljno rješenje programskog primjera.

U drugoj fazi prezentiran je postupak optimalizacije prvog rješenja. Postupak se bazira na prilagodjavanju detaljnog rješenja mogućnostima programskog jezika i elektroničkog računala na kojem će se obrada odvijati, izvršena je sinteza elemenata koji se pojavljuju više puta, kao i izostavljanje suvišnih elemenata čiju je realizaciju bilo moguće postići adekvatnim opisom podataka ili na neki drugi način. Ova faza rezultirala je drugim, optimalnim rješenjem programskog primjera. Za ovo rješenje osim detaljnog dijagrama toka obrade izradjen je i predložak programskog rješenja koje je napisano u Cobolu i predvidjeno za izvodjenje na računalu Burroughs B 1700.

Razlika izmedju detaljnog i optimalnog rješenja je očita. U detaljnom rješenju prisutno je ponavljanje nekih elemenata. Od 9 slučajeva ponavljanja elemenata u optimalnom rješenju pojavljuju se samo 3. Ostali slučajevi ponavljanja izbjegnuti su, a da su pritom očekivani rezultati ostali nepromijenjeni. Na taj način broj naredbi smanjio se od 24 (detaljno rješenje) na 16 (optimalno rješenje). Ako se uzme u obzir da programski primjer pripada kategoriji primjera s relativno malim opsegom programskih zahtjeva, tada je očito da je proces optimalizacije rezultirao s dosta vrlo vrijednih poboljšanja.

L I T E R A T U R A

....., B 1700/B 1700 Systems COBOL Reference Manual, Burroughs Corporation, Detroit, Michigan, 1978.

Kliment,S., Programiranje u Cobolu, skripta, III ponovljeno izdanje, Fakultet organizacije i informatike Varaždin, Varaždin, 1976.

Lipljin,N., Grafički prikaz postupka obrade podataka kao pomoćno sredstvo u procesu obrazovanja programera, Zbornik radova IV medjunarodnog simpozija "Kompjuter na sveučilištu", str.143-150, Sveučilišni računski centar Zagreb, Zagreb, 1982.

Lipljin,N., Programski primjeri kao mogućnost metodološkog pristupa u izučavanju viših programskih jezika, magistarski rad, Sveučilište u Zagrebu, Zagreb, 1980.

Tkalac,S., Struktura i organizacija podataka, udžbenik, Fakultet organizacije i informatike Varaždin, Varaždin,1979.

Pletenac,V., Suvremena nastavna tehnologija, Centar za pedagošku izobrazbu i istraživanje u Zagrebu, Zagreb, 1979.

Zvonarević,M., Socijalna psihologija, Školska knjiga, Zagreb, 1976.

Primljeno: 1982-06-23

Lipljin N. Elaboration and the Process of Optimization of the Solution of a Programming Example

S U M M A R Y

In the first stage of the elaboration of a programming solution, the procedure follows a general data-processing approach and moves towards a detailed analysis of the problem, which results in the first (detailed) solution of the programming example.

The second stage works out a procedure for the optimization of the programming solution thus obtained. The result of this procedure appears in the form of the second, optimal,solution. It is better, more concise and shorter. To this end, apart from a detailed flowchart, a model of the programming solution is worked out, which is written in COBOL and is intended for execution on a Burroughs B 1700 computer.

By considering the number of necessary instructions and,in general, by analyzing the quantitative features of the proposed solution, it is proved that the optimization procedure is justified in view of a number of valuable improvements, while the expected processing results remain unchanged.