INFORMATICA 1/1983

ALGORITMI ZA ISKANJE REZERVIRANIH BESED

M. GAMS

UDK: 519.688

INSTITUT „JOŽEF STEFAN“, JAMOVA 39, LJUBLJANA

V članku so opisani algoritmi za iskanje rezerviranih besed. Osnovna algoritma sta binarno iskanje in iskanje z razpršenimi tabelami. V članku so opisane izboljšave teh dveh osnovnih algoritmov. Podana je tudi njihova časovna in prostorska analiza.

ALGORITHMS FOR SEARCHING FOR RESERVED WORDS. This article describes algorithms for searching for reserved words. Basic algorithms are binary search and hash search. This article describes basic algorithms with their improvements and provides time and memory requirements analysis.

1. Uvod

V računalniških programih pogosto srečamo rezervirane besede. To so besede, ki imajo posebeh, rezerviran pomen. Take besede srečamo v prevajalnikih, urejevalnikih, operacijskih sistemih itd. Rezervirane besede so del teksta, npr. programa v programskem jeziku, zato morajo računalniški programi vsebovati postopke za iskanje rezerviranih besed. Osnovna algoritma za iskanje rezerviranih besed sta binarno iskanje in iskanje z razpršenimi tabelami. Z izboljšavami teh dveh algoritmov je mogoče doseči precejšne časovne prihranke.

2. Opis algoritmov

Najprej bomo opisali osnovna algoritma in nato njune izboljšave. Algoritme bomo opisovali v Pascalu. Kjer pa bomo prekoračili izrazne možnosti tega jezika, se to posebej označeno. V tem poglavju bomo govorili tudi o kompleksnosti algoritmov. S tem pojmom bomo označili povprečno število primerjanj neznane besede z rezerviranimi.

2.1. Binarno iskanje

Opis algoritma:

```
lower := 1; higher := bottom;
found := false;
while (lower<=higher) and not found do
begin
  middle := (lower + higher) div 2;
  if words[middle] > unknownWord
    then higher := middle
    else if words[middle] < unknownWord
      then lower := middle
      else found := true
end;
```

```
                WORDS
              ------------
lower =>      |AND       |
              |ARRAY     |
              |...       |
              |...       |
              |...       |
middle =>     |...       |
              |...       |
              |...       |
              |...       |
              |...       |
higher =>     |WITH      |
              ------------
```

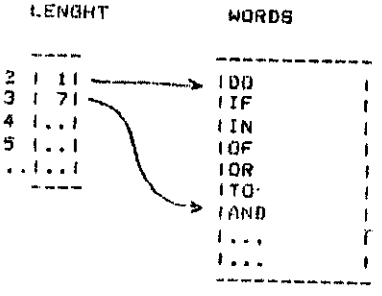
Opombe:

- na večini obstoječih Pascalovih prevajalnikov operatorja "<" in ">" nista implementirana za primerjanje sestavljenih tipov,
- rezervirane besede v tabeli "WORDS" morajo biti urejene po abecednem redu,
- spremenljivki "lower" in "higher" morata biti pred začetkom izvajanja binarnega iskanja prirejeni na vrh, oziroma dno tabele.

Kompleksnost algoritma za binarno iskanje je reda velikosti log(n), kjer je n število rezerviranih besed. Ker je rezerviranih besed običajno nekaj deset, je log(n) < 5 ali 6.

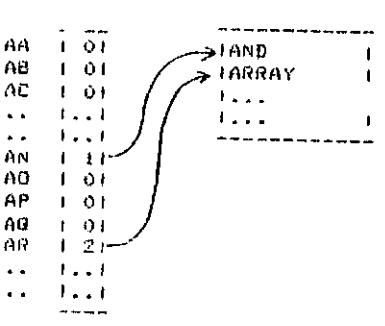
Najbolj običajen trik za pohitritev iskanja je grupiranje rezerviranih besed v skupine. Eden primerov za grupiranje v skupine je po dolžini besed, tako da so znotraj iste skupine besede iste dolžine, urejene po abecednem redu. Algoritem za binarno iskanje je potrebno samo malenkostno spremeniti. Binarno iskanje vršimo v tabeli "WORDS" tako, da

Pred izvajanjem osnovnega algoritma priredimo spremenljivkama "lower" in "higher" vrednosti iz vnaprej generirane tabele, ki vsebuje indekse začetka in konca skupin besed iste dolžine. Na primer, če je "l" dolžina neznane besede, bo "lower := lenght[l]" in "higher := lenght[l+1] - 1". Primer za jezik Pascal je shematično prikazan malo naprej.



Za realne primere se kompleksnost algoritma približno dvakrat zmanjša in je med 2 in 3.

Drug način grupiranja je po prvi ali po prvih dveh črkah. V prvem primeru je tabela za določanje indeksov v tabeli WORDS razdeljena na 26 in v drugem na 676 delov. Dodatno potrošimo 26 ali 676 celic spomina, iskanje neznane besede pa ima kompleksnost med 2 in 1. Možne so še razne kombinacije, npr. grupiranje po dolžini in po prvi črki itd.



2.2. Iskanje z razpršenimi tabelami

Osnovni algoritem iskanja z razpršenimi tabelami temelji na funkciji "h", ki preslika v našem primeru besede v indeks tabele. Kompleksnost takih algoritmov je običajno kar konstanta. Kompleksnost je 1 kadar je funkcija h injektivna (h(w1) <> h(w2)). Običajno imajo razpršitvene funkcije lastnosti, ki so v povprečju blizu injektivnosti, tako da se možno približajo kompleksnosti 1. Druga pomembna lastnost razpršitvenih funkcij je to, da običajno razpršijo podatke v precej večjo tabelo, oziroma da je v tabeli precej praznih mest. Odtod tudi ime "razpršene tabele".

Kadar je razpršitvena funkcija injektivna, navadno rečemo, da je "popolna" (perfect). Kadar je razpršitvena funkcija surjektivna, rečemo, da je "minimalna". Minimalna razpršitvena funkcija ima v našem primeru tabelo "WORDS" polno zasedeno.

V zadnjih letih je nekaj odkritij povečalo zanimanje za minimalne ali skoraj minimalne popolne razpršitvene funkcije [1,2,3]. V [1] je opisana minimalna razpršitvena funkcija, implementirana na rezerviranih besedah jezika Pascal. Razpršitvena funkcija je oblike

```
h(Word) = value[Word[1]] + lenght +
          value[Word[lenght]]
```

"lenght" je dolžina neznane besede "Word"
"Word[i]" je i-ta črka besede "Word"
"value[letter]" je izbrana vrednost črke "letter"

Obitno funkcija ne more biti popolna, kadar obstajata dve rezervirani besedi enake dolžine, ki imata paroma enaki prvi in zadnji črki. V [2] je pokazano, da obstaja še več drugačnih protiprimerov, torej postopek iskanja razpršitvenih funkcij na zgornji način ne jamči rešitve. V [3] je omenjena metoda, ki nima te slabosti, vendar je tako zgenerirana funkcija časovno zahtevnejša.

Problem iskanja razpršitvene funkcije je problem prirejanja vrednosti posameznim črkam. Denimo, da imamo samo 20 različnih začetnih in končnih črk rezerviranih besed, ter da vsaki lahko priredimo največ 20 različnih vrednosti. Kompleksnost preprostega algoritma, ki preračuna vse variante, je 20**20 (dvajset na dvajset), torej praktično nezvedljivo. Napišimo sa:

```
repeat
  if overlap then letter := pred(letter)
    else begin letter := succ(letter);
               value[letter]:= -1
    end;
  repeat
    value[letter] := value[letter] + 1;
    checkOverlapping(overlap,value,words)
  until not overlap or
        (value[letter])>maxValue)
until not overlap and (letter = 'z')
```

Osnovni algoritem lahko pohitrimo z raznimi triki. Namesto zaporednega izbiranja črk začnemo z najbolj pogostimi. Nadaljne izboljšave niso objavljene v literaturi, čeprav so v praksi najbrž uporabljane. Najbolj učinkovito pohitritev imenujem "superbacktrack". Kadar se dve rezervirani besedi prekrivata tako, da jo ne moremo ločiti s spreminjanjem vrednosti tekoče črke, moramo spremeniti vrednost črke, ki nastopa v prirejanju in smo ji zadnji določili vrednost. Poglejmo si primer:

```
value[letter]
---------
e |  0
r |  1
t |  4
f |  ?   <== tekoča črka

w1 = for
w2 = file

h(w1)= value[f] + 3 + value[r] = value[f] + 4
h(w2)= value[f] + 4 + value[e] = value[f] + 4
```

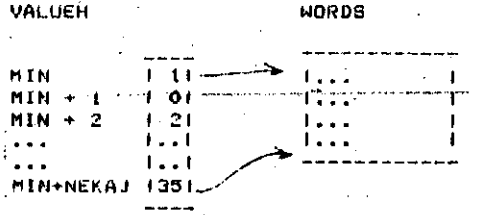
"Superbacktrack" v takem primeru poveča vrednost črke "r" za 1.

Avtor je opazil še en koristen recept. Ko zasledujemo izvajanje algoritma, opazimo, da so nekatera zaporedja črk zelo neprijetna. Takrat program spreminja vrednosti črk korak za korakom brez "superbacktracka". Takrat je najbolje prekiniti izvajanje in spremeniti vrstni red prirejanja vrednosti črkam.

Tako je za generiranje minimalne popolne razpršitvene funkcije za Pascalove rezervirane besede avtorjev program potrošil le nekaj sekund. Poglejmo si primer, ko vnaprej izberemo najmanjšo možno vrednost razpršitvene funkcije 3:

	VALUE	WORDS
1	(E) 0	
2	(P) 0	
3	(D) 0	END
4	(T) 1	ELSE
5	(N) 3	TYPE
6	(O) 8	PACKED
7	(R) 6	NOT
8	(F) 11	THEN
9	(L) 11	PROCEDURE
10	(A) 15	DO
11	(C) 19	TO
12	(I) 19	RECORD
13	(M) 23	REPEAT
14	(V) 25	DOWNTO
15	(W) 14	FILE
16	(B) 21	OR
17	(G) 19	NIL
18	(H) 15	AND
19	(S) 31	WHILE
20	(U) 20	FOR
21	(Y) 17	OF
22	(J) 0	FUNCTION
23	(K) 0	CASE
24	(Q) 0	IN
25	(X) 0	CONST
26	(Z) 0	MOD
27	------	LABEL
28		DIV
29		BEGIN
30		PROGRAM
31		GOTO
32		IF
33		WITH
34		VAR
35		SET
36		UNTIL
37		ARRAY

Za Pascalove rezervirane besede znamo enostavno in hitro določiti minimalno popolno razpršitveno funkcijo. Kaj pa če imamo v rezerviranih besedah tudi besede kot "odr" in "car" ali "fi" in "if"? Očitno moramo izbrati drugačno obliko razpršitvene funkcije. Nato poskusimo generirati tabelo "value". Ker nam postopek ne jamči rešitve, prav lahko poskušamo zaman. V takih primerih je smiselno najprej iskati popolno razpršitveno funkcijo, ki ni minimalna. Tabela "WORDS" je zdaj dimenzije N + NEKAJ, kjer je N število rezerviranih besed. Ko najdemo eno rešitev, lahko zmanjšujemo "NEKAJ", dokler še najdemo rešitev. Da ne bi trošili preveč spomina, si pomagamo še z naslednjim trikom:



Tako potrošimo še N + NEKAJ celic spomina.

3. Podrobnejša časovna in prostorska analiza algoritmov

Do sedaj smo govorili le o povprečnem številu potrebnih primerjanj neznane besede z rezerviranimi. To smo označili s "kompleksnostjo algoritma". Sledi podrobnejša analiza algoritmov. Za eno operacijo bomo šteli eno primerjanje, eno prirejanje, eno računsko operacijo (+,*..), en dostop polja itd. Te operacije časovno niso enakovredne, zato so ocene okvirne in odvisne od računalnika. Privzeli bomo še, da en znak zavzema eno besedo spomina, da je rezervirana beseda sestavljena iz 10 znakov in da je vseh rezerviranih besed 32.

Ocenimo število osnovnih operacij za primerjanje besed med seboj:

```
while (a[i]=b[i]) and (i<dimWord) do
        i := i + 1;
```

Imamo:

```
        1 prirejanje (:=)
        1 rač.op.    (+)
        1 log.op.    (and)
        2 primerjanji (=,<)
        2 dostopa polja ([i])
        +1 za kontrolni konstrukt "while"
       ---------------------------------
        8 osnovnih operacij
```

Za enaki besedi načeloma porabimo toliko primerjanj kot je dimenzija besede (maksimalna dovoljena dolžina). Za različni besedi porabimo bistveno manj primerjanj, v povprečju bomo za naše ocene vzeli 1,5.

A - neznana beseda je rezervirana
B - neznana beseda ni rezervirana

ALGORITEM	ŠTEVILO OPERACIJ A	ŠTEVILO OPERACIJ B	CELIC SPOMINA
binarno	180	125	320
+ po dolžini	140	60	330
+ po prvi črki	120	55	346
+ po prvi čr. + dolž.	80	20	580
+ po prvih dveh črkah	55	10	896
minim.pop. razpr. f.	55	12	346
popolna razpr. f.	57	14	385

Opombe:

- ocene so okvirne, ker so zelo odvisne od oblike rezerviranih besed itd.

- konkretne izvedbe algoritmov lahko malo pohitrimo z enostavnimi triki, ki so v časovni analizi upoštevani. Npr. kadar v povprečju dostikrat prirejamo enake besede, bomo primerjanje naredili takole:

```
while (unKnownWord[i] = WORDS[indeks,i]) and
      (WORDS[index,i] <> ' ') do
        i := i + 1;
```

Tako primerjanje precej pohitrimo.

- v oceni je bila privzeta taka oblika razpršitvene funkcije kot za Pascalove rezervirane besede.

- v praktični uporabi algoritmov, npr. pri prevajalnikih, odpade večji del potrošenega časa na branje, tako da so prihranki manjši.

Praktični rezultati:

Skupina študentov (M.Bradeško, B.Oremuž, M.Hrvatin, T.Kokalj) je merila razmerje med osnovnim binarnim iskanjem in popolno razpršitveno funkcijo za Pascalove rezervirane besede. Razmerje se je sukalo od 1:6 do 1:9 glede na način generiranja testnih besed od samih rezerviranih do samih nerezerviranih. To razmerje ustreza teoretično izračunanemu.

Rad bi se zahvalil tudi I. Mozetiču in N. Lavrač za konstruktivne pripombe.

Zaključek analize:

Natančna analiza algoritmov za iskanje rezerviranih besed da določeno prednost popolnim minimalnim ali skoraj minimalnim razpršitvenim funkcijam. Algoritem z razdelitvijo rezerviranih besed po prvih dveh črkah je časovno celo malenkost hitrejši, vendar troši bistveno več spominskega prostora. Popolne razpršitvene funkcije imajo le to dodatno neprijetnost, da moramo najti obliko razpršitvene funkcije inda moramo prirediti vrednosti črkam. To pa je običajno delo nekaj dni.

4. Zaključek

Praktični in teoretični rezultati dajejo prednost uporabi algoritmov z minimalnimi ali skoraj minimalnimi razpršitvenimi funkcijami za iskanje rezerviranih besed. Sama uporaba razpršitvene funkcije je 5 do 10 krat hitrejša od osnovnega binarnega iskanja. Zaradi podaljšanega branja je v praktičnih primerih prihranek precej manjši, vendar še vedno upoštevanja vreden. Zato je smiselno pohitriti obstoječo programsko opremo s popolno razpršitveno funkcijo.

Literatura:

1. R.J.Cichelli: Minimal Perfect Hash Functions made simple, CACM, Vol.23, Num.1, Januar 1980, str. 17-19.

2. G.Jaeschke, G.Osterburg: On Cichelli's Minimal Perfect Hash Function Method, CACM, Vol.23, Num.12, December 1980, str. 728-729.

3. G. Jaeschke: Reciprocal Hashing: A Method For Generating Minimal Perfect Hashing Functions, CACM, Vol.24, Num.12, December 1981, str. 829-833.