POUŽITÍ SYSTÉMU D PRO ŘEŠENÍ ÚLOH ZPRACOVÁNÍ HROMADNÝCH DAT

Miroslav Zvoníček, prom.mat.,
Výpočtové středisko, 1.BZKG k.p. Brno

1. Úvod

Současné programovací jazyky neposkytují pro oblast zpracování hromadných dat /ZHD/ dostatečně silné datové typy, dovolující realizovat složitější datové vazby v operační paměti počítače.

Cílem článku je ukázat možnosti použití datového typu síť binárních vyvážených stromů /BVS/ a seznamů /dále jen síť/ pro řešení úloh ZHD. V odstavci 2 jsou uvedeny základní vlastnosti datového typu síť spolu s prostředky systému D, který tento datový typ implementuje jako rozšíření systému PL/1. V odstavci 3 jsou uvedeny vlastnosti základních vazebních schémat, vhodných pro řešení jednotlivých typů úloh ZHD, spolu s technologií jejich použití v systému D. V závěru jsou zhodnoceny dosavadní zkušenosti s používáním systému D.

2. Základní vlastnosti systému D

Základní koncepce systému D vychází z následujících vlastností :

- podpora úloh ZHD
- podobnost vazebních vlastností datového typu síť s vazbami v databázových systémech síťového typu
- náhrada stávajících algoritmů ZHD vyhledávacími algoritmy a z toho vyplývající výrazné omezení používání třídicího programu.

Jazyk D je realizován jako rozšíření jazyka PL/1, takže programátor má současně k dispozici jazykové prostředky systému D a PL/1. Nový datový typ síť umožňuje efektivní práci se seznamy a BVS, které mohou být zapojeny do sítě vztahů typu set /vlastník-> člen/.

Výsledný datový typ síť je tedy multigraf jednotlivých typů vět. Jsou dovoleny věty pevné i proměnné délky. Výskyty určitého

typu věty jsou organizovány do setů, přičemž výskyt setu může být:

- jednosměrný nebo obousměrný seznam
- BVS s povolenými nebo zakázanými duplicitami klíčů, počet klíčů BVS není omezen, pro každý klíč lze volit vzestupné nebo sestupné řazení.

Síť je vytvářena ve virtuální paměti počítače.

Organizace dat do setů a vlastnosti BVS poskytují silné a optimální nástroje pro řazení, vyhledávání, vytváření čítačů a sumátorů, řízení totálových činností.

Základní příkazy jazyka D provádějí činnosti :

- příkaz # DCL umožňuje deklarovat síť, jednotlivé typy vět sítě, vazby /sety/ mezi nimi, čítače, sumátory a tabulky intervalů. Počet deklarovaných sítí v programu není omezen
- vytváření vět a jejich vazeb příkazy # INSERT, # CONNECT
- vyhledávání vět v síti:

skalární operace /nastavující jednu větu/ pomocí příkazů:

- # PREPARE počáteční nastavení věty výskytu setu
- # NEXT nastavení další věty pro BVS v pořadí určeném hodnotami klíčů
- # PRIOR nastavení předcházející věty
- # CROSS přechod do jiného setu pro nastavenou větu
- # OWNER nastavení vlastníka výskytu setu.

množinové operace /nastavující množinu vět/ příkazem SELECT cesta;, který tvoří hlavičku iterativní skupiny /příp. vícestupňové/, nastavující jednotlivé věty množiny, "cesta" určuje cestu v síti s případnými výběrovými podmínkami na jednotlivé uzly cesty

- rušení vět a vazeb příkazem # DELETE
existuje dvojí režim operace DELETE:
  - pasivace věty /bez reorganizace BVS/
  - fyzický výmaz věty /s reorganizací BVS/

volba režimu je součástí deklarace sítě

- rušení sítě /uvolnění paměti a stanovení počátečních stavů/ příkazem # DELETE síť;
- modifikace vět a vazeb příkazem # MODIFY
- přesun věty pod jiného vlastníka příkazem # SHIFT
- automatické vytváření čítačů a sumátorů různých typů a potřebných úrovní

Příkaz # SELECT cesta; provádí:

- vyhledání vět daných vlastností ve výskytu setu :
  - rychlé vyhledání věty /vět/ daných klíčů pomocí vlastností BVS
  - plošný průchod celým výskytem setu /směrem vpřed nebo vzad/ s uplatněním výběrových podmínek na jednotlivé věty; při plošném průchodu BVS směrem vpřed jsou věty nastavovány v pořadí určeném hodnotami klíčů, pro směr vzad v pořadí opačném
  - plošný průchod od nastavené věty /vpřed nebo vzad/ s uplatněním výběrových podmínek
- přechod na výskyt setu nižší úrovně
- přechod do jiného setu pro nastavenou větu
- přechod na vlastníka
- přechod na uschovanou větu pomocí STORAGE proměnných, které umožňují uschovat adresu current věty.

Skupiny # SELECT je dovoleno do sebe libovolně vřazovat /obecně jakékoliv skupiny jazyka D/. Implicitní vřazení je určeno jednotlivými částmi /uzly/ cesty. Na různých úrovních skupin # SELECT je možno pracovat s týmž setem, což systém řeší pomocí instancí current,vět. Je možno se dovolávat určené instance. Pokud "cesta" neurčuje souvislou cestu v síti začínající vstupním bodem sítě, systémově se doplní na souvislou /pokud je doplnění jednoznačné/.

Existují dva typy current vět: current věta setu a current věta typu věty.

Programátor nemá přímý přístup na nastavenou větu, z důvodů nebezpečí narušení systémové části věty, obsahující směrníky a příznaky. Uživatelská část věty se při jejím nastavení přesouvá do tzv. obrazové věty, přístupné programátorovi.

Výběrová podmínka jazyka D je konjukcí elementárních podmínek, které jsou :

- logický výraz PL/1
- nalezení větyv v tabulce a její zpřístupnění.
  Tvar podmínky : [NO JIN (tab klíč=výraz [,-J...)

  Tabulkou může být výskyt libovolného setu, reprezentovaný BVS se zakázanými duplicitními hodnotami klíčů. Můžeme tedy pracovat s množinou tabulek v síti vztahů.
  Vyhledávání v tabulce může být provedeno dle diskrétních hodnot nebo dle příslušnosti do intervalů
- ekvivalence výskytu dvou setů dle počtu prvků, hodnot klíčů pořadím si odpovídajících prvků, příp. dalších položek nebo výrazů nad nimi
- podmínky na existenci current vět, zda určují týž prvek, zapojení výskytu věty v určitém setu.

Výběrové podmínky lze aplikovat pomocí příkazu # WHEN, který tvoří hlavičku neiterativní skupiny, příkazu # DO WHEN, tvořícího hlavičku iterativní skupiny a příkazu # SELECT.

BVS jsou implementovány jako AVL stromy, v průběhu práce jsou neustále udržovány ve vyváženém stavu, takže nemůže dojít k jejich degeneraci. Pro vstupní body sítě /pro set neexistuje vlastnický typ věty/ je možno navíc zvolit hešovací mechanismus, takže výskyt setu je představován množinou BVS, z které je vybíráno hešovací funkcí dle klíče nejvyšší priority.

Systém D podporuje práci se sekvenčními množinami dat /soubory/ a systémem IDMS, jako zdrojů pro naplnění sítě.

Vzhledem k vysoké rychlosti zpracování, kterou systém D svými prostředky umožňuje, je možno využívat i větších objemů virtuální paměti počítače a čerpat z výhod virtuálního aparátu, který realizuje nejefektivnější přístup k diskové paměti.

Vlastnosti systému D dovolují výrazně zredukovat počet třídění, který byl pro realizaci úlohy dříve potřebný. Vyhledávací algoritmy a s nimi související algoritmy vytváření vazeb, čítačů a sumátorů, jsou dostatečně rychlé i při plnění sítě nesetříděnými množinami dat.

Překlad konstruktů jazyka D do jazyka PL/1 je realisován jednoprůchodovým předkompilátorem. Algoritmy jednotlivých příkazů jsou generovány in-line, není použito interpretačních prostředků, takže výsledný program odpovídá na míru požadavkům programátora.

Existuje možnost volby ladicího režimu, při kterém se generují příkazy pro tisk výsledků plněných příkazů jazyka D. Ladicí tisky je možno staticky /laděné úseky/ i dynamicky řídit.

Potřeba přesného pohybu v síti a vlastnosti jazyka D svláště příkazu #SELECT, si vynutily změny pojmu set :

- vlastník nepatří do výskytu setu
- pohyb v určitém setu nastavuje pouze current větu tohoto setu a current větu příslušného typu věty.

V důsledku toho jsou pohyby v různých setech i nad týmž typem věty navzájem nezávislé, sety lze souběžně procházet.

Při provádění příkazů jazyka D mohou vzniknout chybové stavy, způsobené nekorektností údajů potřebných pro jejich provedení /existence current vět, zapojení věty v určitém setu/. Vznik chybových stavů je značně závislý na tom, zda příkazy jsou prováděny uvnitř skupiny #SELECT, nebo mimo ni. Uvnitř skupiny /skupin/ #SELECT je vznik chybových stavů minimální, neboť existují current věty setů a typů vět určených cestami v aktivních příkazech #SELECT /vlastnost skupiny #SELECT/ a mnohé chybové situace lze testovat již během kompilace. Mimo skupiny #SELECT programátor někdy musí testovat jisté situace, pro tyto účely slouží poslední uvedený typ podmínek jazyka D. Pokud vznikne chybová situace, je hlášena chyba /je možno zvolit detailní hlášení typu a místa chyby/ a ukončí se provádění programu.

Následující příklad ukazuje průchod výskytem setu S pomocí skalárních operací a pomocí příkazu #SELECT :

```
a) #PREPARE S FORWARD;          /* nastavení prvního prvku   */
     #DO WHEN(CRS S);           /* test ex. current věty setu*/
      /* zpracování věty */
      #NEXT S;                  /* další věta                */
     #END;                      /*konec cyklu zpracování věty*/
```

```
b/ #SELECT 3;
      /* zpracování větve */
   #END;                          /* konec skupiny #SELECT */
```

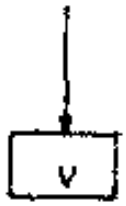
3. Základní vazební schémata a jejich použití

Jak bylo v úvodu řečeno, je cílem tohoto odstavce vysvětlit vlastnosti, popis a použití vybraných jednoduchých schémat sítě. Současně bude snaha ukázat, že tato schémata vystačí k řešení značného objemu úloh ZHD. Pro složitější úlohy je možno aplikovat ten postup, že výsledné schéma sítě vytvoříme ze základních schémat dle požadavků dané úlohy, případně použijeme více sítí.

I když datový typ síť nevypadá na první pohled jednoduše, mají ukázky jednotlivých použití rovněž k tomu, aby demonstrovaly stručnost a jednoduchost výrazových prostředků jazyka D.

3.1 xxx

Nejjednodušším schématem je vazba, ve které existuje pouze jeden typ věty zapojený do jediného setu /vstupní bod sítě s jediným výskytem/ nad tímto typem věty - viz obr. 1



obr. 1

Pokud je set reprezentován seznamem, je schéma použitelné pro implementaci fronty nebo zásobníku. Práci se zásobníkem ukazuje následující příklad.

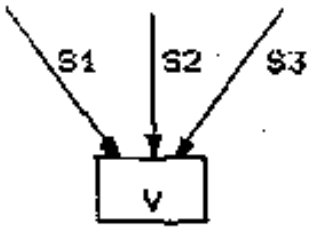
```
#DCL NT NET MEMBER V(/* popis věty */)
     LINK SET(STACK)TO V;
/* zápis do stacku */          #INSERT STACK;
/* výběr ze stacku */          #PREPARE STACK LAST;
                               #WHEN(CRS STACK);
                                /* věta ze stacku nastavena */
 /* výmaz ze stacku */          #DELETE STACK;
                              [#ELSE; /* prázdný stack */]
                               #END;
```

Zajímavější použití poskytuje set representovaný BVS. Pak je schéma vhodné pro řazení nebo implementaci tabulky. Následující příklad ukazuje řazení souboru I1 dle klíčů K1, K2:

```
#DCL NT NET MEMBER V(2 K1 FIXED(9),2 K2 CHAR(10)
                        /* další položky */)
    LINK TO V TREE(K1,K2)DUPL
  SI1 SDS FILE I1 MEMBER V;  /* deklarace souboru v jaz. D#/
#FLOW SI1;  #INSERT V;  #END;
#SELECT V; WRITE FILE(O1)FROM(V); #END;
```

Set nad typem věty V nemá v doplňku LINK explicitně určené jméno, takže se mu přiřadí jméno dle jména věty. Set musí být deklarován s povolením duplicit /DUPL/ a bez hešovacího mechanismu /soubor by byl setříděn pouze po částech/. Sekvenční množina dat SI1 je reprezentována souborem I1. Jiný možný způsob reprezentace je virtuální, realizovaný pomocí procedury, předávající jednotlivé věty množiny. Iterativní skupina #FLOW realizuje tok vět ze souboru I1, které jsou operací INSERT ukládány do BVS. V rámci skupiny #SELECT je proveden výstup seřazené množiny dat do výstupního souboru.

Potřebujeme-li seřadit množinu dat dle několika kritérií, použijeme schéma /3 řazení/:



obr. 2

Popis setů bude například:

```
LINK SET(S1)TO V TREE(K1)
LINK SET(S2)TO V TREE(K2,K3)
LINK SET(S3)TO V TREE(K4)
```

Budeme-li požadovat řazení dle klíčů K2, K3, budeme pracovat se setem S2.

```
#SELECT S2; /* zpracování věty */ #END;
```

Značné množství úloh ZHD /výběrové řízení, vyceňování/ vyžaduje práci s tabulkami /číselníky/. Ukažme zřízení toku vět z množiny SI1, pro každou větu se provede vyhledání v tabulce současně s nastavením vyhledané věty. Pro tabulku je vhodné zvolit hešovací mechanismus .

```
#DCL NT NET MEMBER TAB(2 TK1 FIXED(5),2 TK2 CHAR(20)
                        /* další položky */)
    LINK TO TAB TREE(TK1,TK2)HASH
  SI2 SDS FILE I1 MEMBER TAB       /*deklarace tab. souboru*/
  SI1 SDS FILE I2 MEMBER V(2 X1 FIXED(5),2 X2 CHAR(20)
                           /* další položky */);
#FLOW SI2; #INSERT TAB; #END;      /* vytvoření tabulky */
#FLOW SI1;
  #WHEN(IN TAB(TK1=X1,TK2=X2));
    /* k dispozici věta TAB a V */
 [#ELSE;
    /* věta tabulky nebyla nalezena */]
  #END;
#END;
```

Tabulek je možno využít k součtování. Programátor může deklarovat sumátory typu IN, což jsou proměnné tabulkové věty, ke kterým bude přičítán určený výraz, jestliže bude nalezena věta daných klíčů v tabulce /přičtení budou provedena k sumátorům v nalezené větě/. Následující příklad ukazuje zjednodušené řešení úlohy typu vyceňování. Máme soubor ceník, obsahující pro každý materiál cenu za jednotku a soubor obrat, jehož věta obsahuje číslo materiálu a přijaté nebo vydané /záporné/ množství. Máme určit celkové obraty pro jednotlivé materiály.

```
#DCL NT NET MEMBER CENIK(2 CMAT FIXED(11),2 JCENA FIXED(6,2)
                        ,2 SCENA FIXED(9,2))
    LINK TO CENIK TREE(CMAT)
    SUM IN CENIK(SCENA FROM MNOZSTVI*JCENA)
  FCEN SDS FILE I1 MEMBER CENIK
  OBRATY SDS FILE I2 MEMBER OBRAT(2 CMAT FIXED(11),
                                  2 MNOZSTVI FIXED(3));
#FLOW FCEN; #INSERT CENIK; #END;
#FLOW OBRATY;
  #WHEN(IN CENIK(CMAT=OBRAT.CMAT)); #END;
#END;
#SELECT CENIK;
  PUT SKIP EDIT(CENIK.CMAT,SCENA)(F(11),F(13,2));
#END;
```

Deklarace setu CENIK neobsahuje klíčové slovo HASH, určující hešovací mechanismus, neboť ceník potřebujeme získat seřazený pro závěrečný výstup výsledků.

Tabulky mohou být vícedrovňové, na libovolné úrovni může být určeno vyhledání dle příslušnosti k intervalu /viz dále/. Plnění tabulky může být prováděno ze souboru, pomocí procedury předávající jednotlivé věty, nebo může být hodnota tabulkových vět určena přímo v programu :

```
DO TK=5,27,132,93,48,56,418;   INSERT TAB; END;
```

Důležité je, že množina dat, kterou je tabulka plněna, nemusí být seřazena. Nové věty lze dát do tabulkového souboru libovolně , změnu věty lze provést zařazením věty se změněným obsahem před měněnou větu /stará věta je duplicitní a do tabulky se již nezařadí/. Je možno využít zřetězení souboru změn s původním souborem.

3.2 SETS

Jako další schéma uvažujme hierarchické vazby, přičemž každý typ věty je zapojen pouze v jediném setu - viz obr. 3.

obr. 3

Pokud jsou všechny sety THEk, postupně s klíči K1, K2,...,Kk, bez duplicit, pak vyhledání určité věty provedeme:

```
#SELECT V1(K1=X1)
       V2(K2=X2)
         .
       Vk(Kk=Xk);
  /* zpracování vět V1,V2,...,Vk */
#END;
```

Jsou-li na některé úrovni povoleny duplicity, nebo je příslušný set reprezentován seznamem, je vhodné pro výběr použít doplňku WHERE (podmínky pro plošný průchod). Předpokládejme na úrovni k seznam a máme najít první větu, pro kterou platí 10<=Xk<=50 .

```
#SELECT V1(K1=X1)
       V2(K2=X2)
         .
       Vk ONLY WHERE(Xk>=10,Xk<=50);
  /* zpracování vět */
#END;
```

Pokud neuvedeme ONLY, budou postupně nastaveny všechny věty, vyhovující podmínce WHERE.

Pro 11b. úroveň lze volit plošný průchod celým výskytem setu. Požadujeme-li nastavit všechny vazbami určené kombinace výskytů vět V1, V2, ..., Vk, provedeme :

```
#SELECT V1 V2 ... Vk;
   /* věty nastaveny */
#END;
```

Zápis lze zkrátit: SELECT Vk; /* věty nastaveny */ #END;

Určuje-li hierarchické schéma víceúrovňovou tabulku, musí být každý set representován BVS /pro úroveň i>1 je množina výskytů vět representována množinou BVS/. Předpokládejme, že potřebujeme pracovat s dvouúrovňovou tabulkou. Na první úrovni bude hledáno dle diskrétních hodnot K, na druhé dle příslušnosti k intervalu <B,B1>. Popis a použití demonstruje následující příklad. Všimněme si transformace tabulkového souboru do dvou úrovní.

```
#DCL NT NET MEMBER T1(2 A FIXED(5))
                MEMBER T2(2 B FIXED(7),2 B1 FIXED(7))
     LINK TO T1 TREE(A)HASH
          TO T2 TREE(B)
     RTAB T2I SET T2(B:B1)
   FT SDS FILE I1 MEMBER T(2 A FIXED(5),2 B FIXED(7),
                           2 B1 FIXED(7));
#FLOW FT;                          /* plnění tabulky */
 T1=T,BY NAME; T2=T,BY NAME;
 #INSERT T1; #INSERT T2 VIA T1;
#END;   /* hledání v tabulce T; */
        #WHEN(IN T1(A=X),IN T2I(B=Y));
          /* nastavena věta T1, T2 */
        #ELSE; /* hledání neúspěšné */
        #END;
```

Doplněk RTAB určuje alias jméno T2I setu T2, jehož použití v podmínce IN určuje hledání dle intervalu. Klíč B setu T2 se považuje za dolní mez a je mu přiřazena horní mez B1. Sety T1, T2 nesmí mít povoleny duplicity. Vzhledem k tomu budou odstraněny duplicitní hodnoty klíče A: v paměti bude věta s určitou hodnotou A pouze jednou a na ní budou navěšeny/v BVS/ příslušné věty určující intervaly. První podmínka IN hledá vlastníka pro hledání pomocí druhé podmínky IN.

Zkusme nyní transformovat sekvenční množinu dat, kterou potřebujeme zpracovat, do hierarchického tvaru, obdobně jak bylo ukázáno pro dvouúrovňovou tabulku. Zvolme klíče K1, K2.

Schéma pak bude mít 3 úrovně:

V 1 TREE s klíčem K1

V 2 TREE s klíčem K2

V 3 obsahující zbývající části věty, jako seznam

Tato operaci můžeme rovněž charakterizovat jako oddělení klíčů od věty. Výsledné zobrazení ve virtuální paměti počítače nám poskytne následující výhody :

- odstranění redundance hodnot klíčů
- lze získat seřazenou množinu původních vět
- snadné programování totálových činností
  /činnosti na začátku a konci skupin se stejnými hodnotami klíčů/
- vytváření sumátorů lib.úrovní.

Uvedené možnosti ukazuje následující příklad, včetně naznačení výstupu součtových a položkových údajů. Kumulace je prováděna pro položky P1, P2 do obou úrovní V1, V2.

```
#DCL S11 SDS FILE 11 MEMBER V(2 K1 FIXED(5),2 K2 FIXED(7),
                               2 P1 FIXED(6),2 P2 FIXED(6))
  NT NET MEMBER V1(2 K1 FIXED(5),
                   2 SP1 FIXED(11),2 SP2 FIXED(11))
         MEMBER V2(2 K2 FIXED(7),
                   2 SP1 FIXED(11),2 SP2 FIXED(11))
         MEMBER V3(2 P1 FIXED(6),2 P2 FIXED(6))
  LINK ID V1 TREE(K1) TO V2 TREE(K2) TO V3
  SUM INSERT V2(ALL SP1 FROM V.P1, ALL SP2 FROM V.P2);
#FLOW S11;          /* tranformace do hier. zobrazení */
  V1=V;BY NAME; V2=V;BY NAME; V3=V;BY NAME;
  #INSERT V1; #INSERT V2 VIA V1; #INSERT V3 VIA V2;
#END;
```

```
#SELECT V1;
  /* začátek skupiny pro K1 */
  #SELECT V2;
    /* začátek skupiny pro K2 */
    #SELECT V3;
      /* detailní činnosti, k dispozici všechny položky */
    #END;
    /* konec skupiny pro K2 */
  #END;
    /* konec skupiny pro K1 */
#END;
```

Sety pro úrovně klíčů je nutno deklarovat jako TREE bez duplicit, pro nejnižší úroveň volíme seznam /není potřebná, pokud nepožadujeme položkovou sestavu/. Při práci s tabulkami byly vysvětleny sumátory IN provádějící součtování pro věty vybrané pomocí tabulky. Obecné sumátory jsou typu INSERT, neboť jsou systémově zajišťovány v rámci operace INSERT /CONNECT/. Jsou rovněž založeny na vyhledávání, takže nevyžadují seřazenost množiny dat. Mají několik možných tvarů, použitý tvar má následující význam:

výraz za FROM je sumován do sumátoru určeného před FROM. Požadavek sumátoru určité úrovně je určen uvedením jeho jména /určeného v doplňku SUM/ v popisu věty této úrovně. Výraz za FROM musí být definován před prováděním operace INSERT nebo CONNECT pro set uvedený v doplňku SUM. Sumátory jsou kompletně vytvořeny po skončení plnění setu určeného v doplňku SUM /po ukončení skupiny #FLOW SI1/.

Existují další jednoduchá schémata sítě se zajímavým použitím, ale již bychom překročili omezený rozsah tohoto příspěvku.

4. Závěr

Systém D je v používání asi 3/4 roku. Přes tuto krátkou dobu lze říci, že dovoluje jednoduchý a přehledný zápis i složitých algoritmů ZHD a nutí programátora ke strukturalisaci programu. Úlohy je nutno dělit na mnohem menší počet částí, ať už jde o kroky

úlohy nebo procedury jednotlivé kroky realizující, hlavně z těch důvodů, že v jedném programu lze provést několik spracovatelských cyklů, aniž by ztratil na přehlednosti, a není potřebné užívat třídicího programu. Většinou není potřebné používat pracovních souborů. Jsou sníženy nároky na čas procesoru, služby operačního systému, vnější paměť a kapacitu přenosových kanálů.

Nezanedbatelná je úspora programátorské i analytické práce.

Namátkově byla provedena srovnání časových nároků vybraných úloh s použitím systému D a bez použití systému D.
Uveďme několik výsledků:

1/ Součtování 2 položek do 4 úrovní, 27000 vět
/původní řešení: třídění, součtování pomocí skupinových změn/
monoprovoz: program v D 2x menší nároky na čas procesoru
6x menší čas trvání JOBu
multiprovoz: 13x menší čas trvání JOBu

2/ Řádková a sloupcová kumulace, 15000 vět
4x menší nároky na čas procesoru než úloha realizovaná CULPRITEM, 7x menší čas průchodu JOBu.

3/ Vyhledávání v tabulce /4000 vět/ pro jednotlivé věty souboru /s použitím hešování/
6x menší čas procesoru než u programu vypracovaného v PL/1 s použitím půlení intervalu.