V Y U Ž I T Í S T R O M O V Ý C H S T R U K T U R P Ř I Z P R A C O V Á N Í H R O M A D N Ý C H D A T

Ing. Fryderyk TOMASZEK, Karel VRBENSKÝ, prom. mat.

ABSTRAKT

V referátu pojednáváme o rozšíření množiny instrukcí jazyka PL1 o prostředky pro vkládání, vyhledávání a rušení položek v poli se stromovou organizací prvků. Uvádíme také několik příkladů praktického použití těchto struktur.

1. Úvod

Při řešení úloh z oblasti hromadného zpracování dat se často vyskytuje požadavek získávat informace z tabulek, nebo je v tabulkách měnit, nebo doplňovat.
Pro řešení tohoto problému na úrovni souborů používáme číselníky a soubory s přímým přístupem. Ne vždy informace jsou tak rozsáhlé, že vyžadují uložení do souboru. Někdy můžeme vystačit i s polem ve vnitřní paměti počítače. Když k tomu připojíme skutečnost, že použití virtuální paměti podstatně snižuje problémy s místem pro uložení dat uvnitř počítače, můžeme se potom více soustředit na zkracování času zpracování. Podmínkou ovšem je, abychom s daty uloženými ve vnitřní paměti stroje uměli efektivně manipulovat. Pod tímto pojmem budeme rozumět následující tři činnosti:

- vkládání
- rušení
- vyhledávání.

Z tohoto hlediska dosáhneme poměrně dobrých výsledků budou-li data v paměti vytvářet tzv. binární vyvážený strom. Vlastnosti této datové struktury jsou uvedeny v referátu doc. Honzíka. My se zaměříme na její praktické využití při zpracování dat.

2. Procedury pro manipulaci s daty ve vnitřní paměti počítače

Vytvořili jsme tři procedury:

- RS1 pro rušení
- RS2 pro vkládání
- RS3 pro vyhledávání.

Všechny tři byly realizovány v programovacím jazyce ASSEMBLER. Žádná z uvedených procedur neobsahuje data, která by ji spojovala s určitou stromovou strukturou aplikačního programu. Tu musí definovat programátor. Data budeme ukládat do jednorozměrného pole

s maximální četností N. Jedna z položek stanoví klíč /1/, podle kterého je budeme vyhledávat. Další /2/ slouží k označení zrušeného záznamu. To znamená, že rušení záznamu se neprovádí jeho fyzickým odstraněním ze stromu. Následující položka /3/ obsahuje informaci o vyváženosti stromu. Jednotlivé prvky pole jsou do něho vkládány ve stejném pořadí v jakém přicházejí ze vstupu. V paměti vytvářejí jenom uzly stromové struktury. Každý uzel musí obsahovat kromě svého klíče, také informaci o tom, kde se nachází jeho levý /4/ a pravý /5/ následník. Touto informací je index příslušného prvku v poli. Je-li hodnota následníka 0 znamená to, že prvek na této cestě nemá následníka.
Výše popsané položky mohou naplňovat a měnit pouze obslužné procedury RS1, RS2, RS3. Programátor může pokračovat v definování dalších položek /6,7/, ale o jejich obsah se již musí starat sám.
Příklad popisu pole v jazyce PL1 je následující:

```
DCL    1 AR_STROM /0:300/    STATIC,
          2 AKLIC CHAR/5/,                /1/
          2 AZRUS CHAR/1/,                /2/
          2 ABAL BIN FIXED/15/,           /3/
          2 ALLINK BIN FIXED/15/,         /4/
          2 ARLINK BIN FIXED/15/,         /5/
          2 ADATA,
            3 ADAT1 CHAR/8/,              /6/
            3 ADAT2 FIXED/7/;             /7/
```

Dále musíme ještě definovat společnou pracovní oblast, pomocí které bude programátor komunikovat s obslužnými procedurami.
Do první položky /1/ v této oblasti vloží hodnotu klíče prvku, který chce v poli vyhledat nebo zrušit nebo do pole zavést. Další dvě položky naplňují obslužné procedury. Do první /2/ vkládají stavový kód výsledku své činnosti. Např. v případě duplicity při zavádění je zde hodnota S4. Do druhé položky /3/ vloží procedury hodnotu indexu prvku pole, který byl například vyhledán. Vloží-li zde hodnotu 0, znamená to, že činnost byla neúspěšná. Bližší specifikace chyby je ve stavovém kódu /viz také tab.1/. Například při pokusu o duplicitní vložení prvku do pole bude hodnota indexu 0 a stavový kód S4.

Procedura	INDEX	STAVKOD	Poznámka
RS2 Vložení	N	00	Vložení záznamu provedeno úspěšně.
	0	S4	Duplicita klíče záznamu - vložení neprovedeno. P obsahuje index, na kterém byl nalezen duplicitní klíč.
	0	S6	Pokus o překročení horní meze pole - vložení neprovedeno
RS1 Zrušení	N	S8	Rušený záznam je již označený za zrušený /AZRUS = # /
	N	00	Zrušení provedeno úspěšně
	0	S2	Záznam nenalezen - zrušení neprovedeno
	0	S0	Stromová struktura nebyla vytvořena - zrušení neprovedeno
RS3 Vyhledání	N	00	Vyhledání záznamu provedeno úspěšně
	0	S8	Vyhledaný záznam je označený za zrušený /AZRUS = # /. P obsahuje index, na kterém byl nalezen zrušený záznam.
	0	S2	Záznam nenalezen
	0	S0	Nebyla vytvořena stromová struktura - záznam nenaleten

$N \in \langle 1, \text{horní mez pole AR\_STROM} \rangle$

Tab. 1

I když je tato situace považována za chybovou, může být v některých případech využita. Např. při potřebě součtovat v rámci stejného klíče. Potom ale musíme znát index prvku pole, ve kterém je stejný klíč jako zaváděný. Tato informace se nachází v jedné /P/ ze sedmi pracovních položek, které jsou také definovány v oblasti. Další pracovní položka /L/, která může být pro programátora užitečná, obsahuje aktuální počet prvků v poli. Je-li tato položka 0, obslužné procedury považují pole za prázdné. K vymazání obsahu pole proto stačí vynulovat počet prvků v poli a další ukládání se bude provádět od jeho začátku. Příklad popisu pracovní oblasti v jazyce PL1 je následující:

```
DCL      1 WOBLAST          STATIC,
           2 INKA           CHAR/5/,        /1/
           2 STAVKOD        CHAR/2/,        /2/
           2 INDEX          BIN FIXED/15/,  /3/
           2 /L,P,Q,R,S,T,A/  BIN FIXED/15/; pracovní položky
```

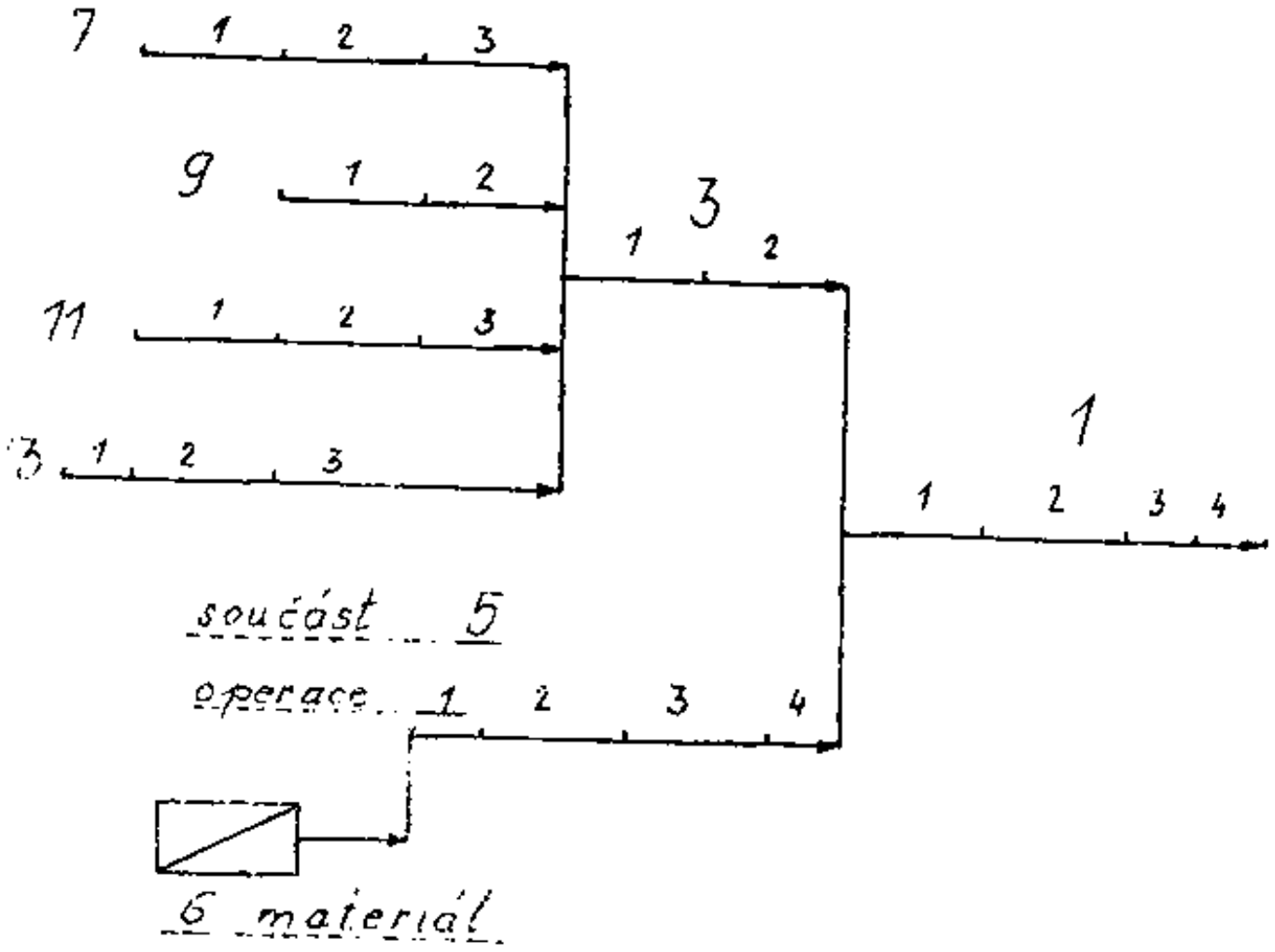
Chceme-li vložit do pole prvek, naplníme klíč v pracovní oblasti a vyvoláme proceduru RS2. V PL1 to budou instrukce:
INKA = klíč_prvku; CALL RS2 /AR_STROM, W/ kde W je adresa pracovní oblasti. Výsledek činnosti RS2 zjistíme pomocí položky INDEX případně STAVKOD. Stejný postup bude při vyhledávání /RS3/ a rušení /RS1/.

3. Realizace procedur

Manipulace se stromovými strukturami je programově řešena v jazyce ASSEMBLER. Spojení na vyšší programovací jazyk je provedeno standartním způsobem přes parametrický list. Modul je k dispozici v současné době ve dvou verzích, pro programovací jazyk PL1 a PL1 - optimalizační.
Modul je tvořen jednou řídící sekcí, která má tři vstupní body, každý pro plnění jedné základní funkce. Přeložený proveditelný modul je uložen v AUTOMATIC CALL LIBRARY programu LINKAGE-EDITOR programovacích jazyků PL1 a PL1-opt. a je opatřen kromě základního jména třemi ALIAS jmény RS1, RS2 a RS3 shodnými s jednotlivými ENTRY. Praktickým důsledkem je, že aplikační programátor využije popisované funkce stromových struktur prostým zavoláním odpovídajícího vstupního bodu v nosném programu. Celková velikost assemblerovského modulu je 2.418 bytů.

4. Praktické příklady

Většina aplikací stromových struktur při zpracování dat v NMIG je v subsystému strojírenské výroby. Následující příklady budou právě z této oblasti. Nemůžeme zde popisovat všechny funkce subsystému. Předepsaný rozsah článku to nedovoluje a nebylo by to ani účelné. Zaměříme se pouze na jednu, a to plánování kusové strojírenské výroby. Přitom popíšeme jenom určité problémy, pro řešení kterých jsme použili stromové struktury. Zároveň u některých uvedeme možný způsob řešení bez jejich použití. Aby uvedené příklady nepůsobily příliš vytrženě z kontextu, v krátkosti se zmíníme o základních souborech subsystému. Jedná se o kmenový soubor zakázek a kmenový soubor pracovišť. Oba jsou sekvenční. Kmenový soubor zakázek odráží strukturu výrobků. Předpokládejme, že máme zakázku, jejíž strukturu je znázorněna na obr. 1.



obr. 1

Informace o ní jsou uloženy do čtyř typů záznamů:

typ Z - zakázka
S - součást
M - materiál
O - operace

Jednotlivé záznamy jsou uspořádány podle klíče, který tvoří číslo zakázky, číslo součásti a číslo operace, a který je uveden v každém záznamu. Jelikož na třetí úrovni v klíči může být kromě čísla operace také číslo materiálu, zvyšujeme číslo operace o 500, čímž docílíme toho, že operace budou zařazeny až za materiál. Uspořádání záznamu zakázky z obr. 1 je následující:

Typ záznamu	č. zakázky	č. součásti	č. operace
Z	31942204	0	0
S	31942204	1	0
O	31942204	1	501
O		1	502
O		1	503
O		1	504
S		3	0
O		3	501
O		3	502
S		5	0
M		5	6
O		5	501
O		5	502
.			
.			

Kmenový soubor pracovišť obsahuje 180 záznamů a každý se týká jednoho pracoviště. Jako první příklad uvedeme použití stromových struktur při aktualizaci tohoto souboru.

Příklad 1.

Popis problému: v souboru existují dva druhy pracovišť, jednoduché a skupinové. V záznamu skupinového pracoviště jsou uvedena čísla ostatních pracovišť skupiny /maximálně 5/. V průběhu aktualizace musíme prověřit úplnost a správnost jednotlivých skupin, viz obr. 2. Znázorňuje seskupení tří pracovišť. V záznamu

s číslem pracoviště 30801 prověříme, zda-li pracoviště uvedená v položkách "ostatní pracoviště skupiny" jsou zavedena do souboru jako skupinové a zda-li obsahují číslo pracoviště 30801. Totéž prověříme u ostatních pracovišť skupiny.

číslo pracoviště		ostatní pracoviště skupiny		
30801	.....	30720	30415	
:	:			
30720	.....	30801	30415	
30415	.....	30801	30720	

obr. 2

Řešení: vstupní kmenový soubor pracovišť zavedeme do pole se stromovou strukturou. Zpracujeme veškeré aktualizační transakce a potom prověříme vazby mezi skupinovými pracovištěmi tak, že sekvenčně procházíme záznamy v poli a u skupinového pracoviště použitím procedury RSJ zjistíme správnost vazeb. V závěru zapíšeme obsah pole na výstupní kmenový soubor, přičemž vynecháme všechny prvky pole označené jako zrušené. U tohoto způsobu aktualizace zpravidla odpadá potřeba třídit transakční data.
Řešení stejného problému klasickým zpracováním sekvenčního souboru by bylo dost obtížné a vyžadovalo by další programy. Jednodušší by bylo použít soubor s přímým přístupem. Vedlo by to k nárůstu času zpracování v důsledku velkého počtu I/O operací.

Příklad 2.

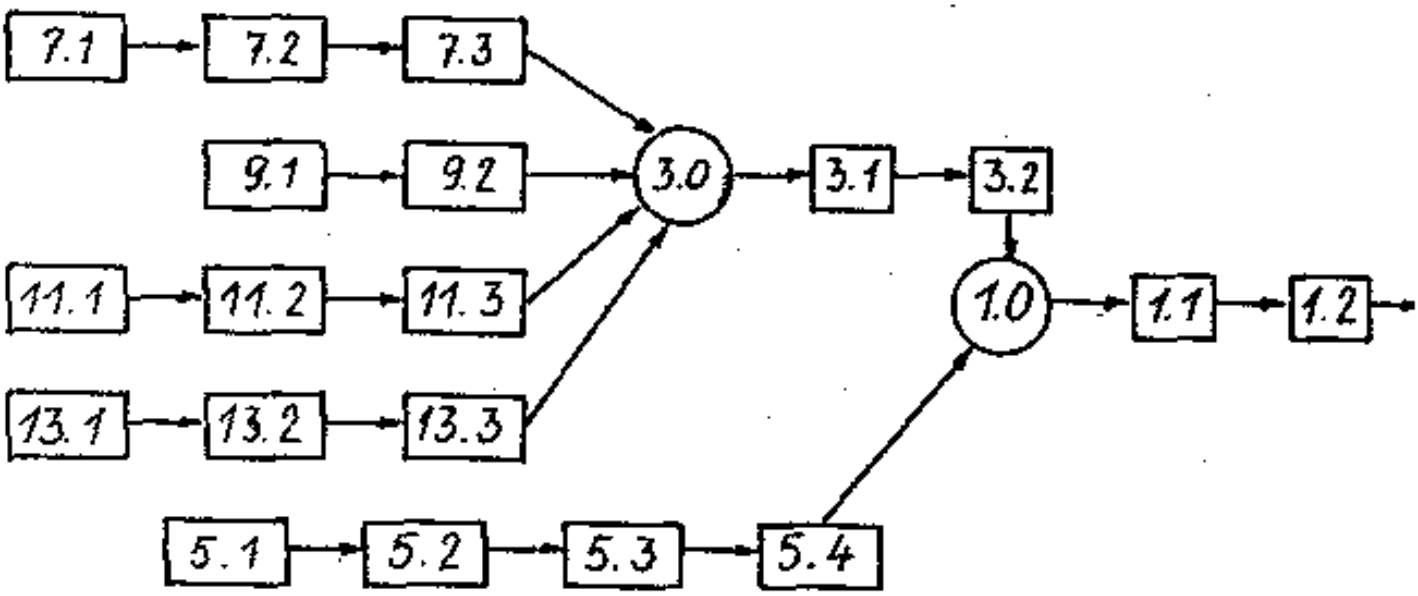
Popis problému: každá zakázka představuje výrobu určitého náhradního dílu, který má své číslo. Jednotlivé součásti, ze kterých se zakázka skládá obsahují kromě jiných údajů také číslo mikrosnímku. Stejná součást může být použita k výrobě různých náhradních dílů, tzn., že se může objevit v rámci různých zakázek. Je požadováno vytvořit soubor, ve kterém klíčem záznamu bude číslo mikrosnímku součásti a dále bude zde uvedeno maximálně pět čísel náhradních dílů, které jako poslední obsahovaly danou součást. Výběr se má provést z celého archívu ukončených zakázek /asi 350 tisíc záznamů/.

Řešení: klasickým způsobem bychom pravděpodobně postupovali tak, že v prvním kroku vytvoříme pracovní soubor, jehož záznamy budou obsahovat číslo mikrosnímku, číslo náhradního dílu a časový údaj o termínu výroby součástí. Ve druhém kroku tento soubor setřídíme podl čísla mikrosnímku. Ve třetím kroku budeme vytvářet požadovaný soubor, přičemž u stejného čísla mikrosnímku doplníme číslo náhradního dílu v závislosti na časovém údaji.
S použitím stromové struktury jsme tento problém vyřešili v jednom kroku. Definovali jsme tabulku s četností 30 000. Klíčem pro přístup k datům v ní, bylo číslo mikrosnímku. Datové oblast obsahovala pole s četností 5 pro uchování čísel náhradních dílů a pracovala jako front se vstupem u 1. četnosti. Postupně jsme četli archívní soubor a z každé součásti jsme číslo mikrosnímku vkládali do tabulky a číslo náhradního dílu do příslušné datové oblasti. Při duplicitách mikrosnímků se provedly jen posuny v datové oblasti. Jelikož uspořádání zakázek na archívním souboru bylo v souladu s termíny ukončení jejich výroby, měli jsme v tabulce čísla náhradních dílů, které jako poslední použily příslušný mikrosnímek. Na závěr jsme prvky pole setřídili v paměti podle čísla mikrosnímku a přesunuli na vnější soubor. Pro třídění jsme z praktických důvodů použili algoritmus "haldy" /heap sort/ místo algoritmu pro uspořádaný výstup prvků ze stromu.
Program při zpracování potřeboval 1,564 Mbytů virtuální paměti. Čas centrální jednotky byl 3 min 56 sek. Celkový počet vstupu do tabulky byl 44 131 a počet prvků v tabulce 21 404. Zpracování bylo provedeno na IBM 370/145.

Příklad 3.

Popis problému: kapacitní vytěžování pracovišť. Seznam operací u dané součástí předatavuje technologický postup při její výrobě. Každá operace obsahuje číslo pracoviště, na kterém bude vykonána, časovou normu pro její trvání a termín, ve kterém má být zahájena. Naším úkolem je vytvořit seznam operací pro jednotlivá pracoviště. Počet operací pro pracoviště je omezen výší jeho kapacitního fondu v daném termínu.

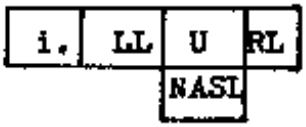
Řešení: výše uvedený popis vystihuje pouze podstatu problému. Ve skutečnosti je situace složitější, proto také způsob řešení jenom naznačíme a nebudeme zacházet do podrobností.
V paměti počítače budeme definovat tři pole se stromovou strukturou. První /A/ bude obsahovat všechna pracoviště. Klíčem je číslo pracoviště. Takto budeme mít kdykoliv k dispozici jeho kapacitní fondy. Do druhého pole /B/ budeme vkládat uzly zakázky a to tak, aby vytvářely její strukturu /viz obr. 1/. Uzel zakázky dostaneme zřetězením čísla součásti s číslem operace. Na obr. 3 je fragment zakázky z obr. 1 vyjádřen pomocí uzlů.



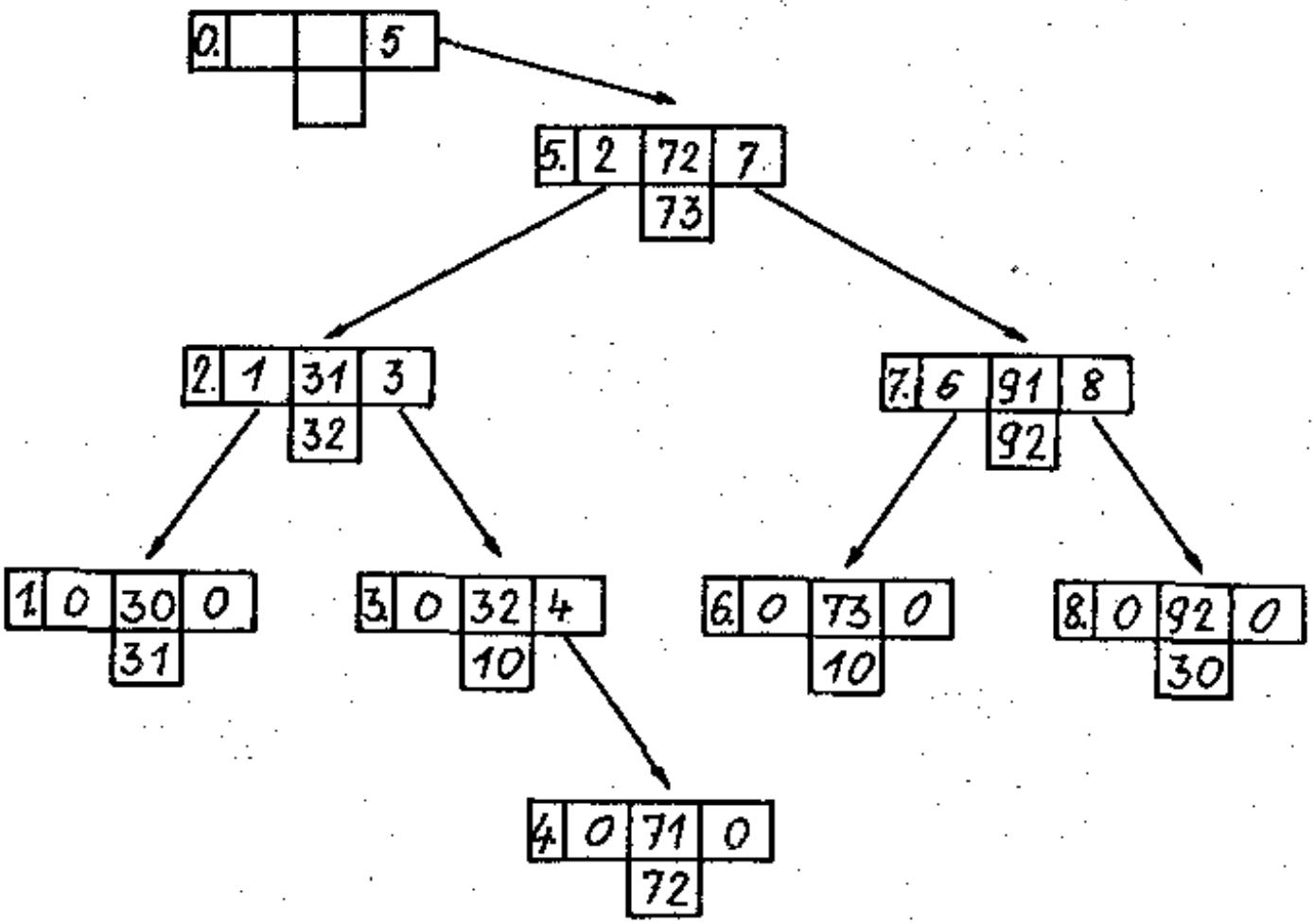
obr. 3

Uzel je také klíčem v tabulce B. V datové části je kromě jiných údajů uveden také následník daného uzlu /NASLED/. Do třetího pole /C/ vložíme uzly, které zahajují plánování jednotlivých součástí. Podle obr. 3 to budou: 7.1, 9.1, 11.1, 13.1 a 5.1. Postup při plánování je takový, že z tabulky C vezmeme první položku, například 7.1 a pomocí procedury RSJ najdeme tento uzel v tabulce B. Zaplánujeme ho a pokračujeme v plánování dalšího uzlu na této cestě. O jaký uzel se jedná, zjistíme v položce NASLED. V našem případě to bude 7.2, potom 7.3. Když narazíme na uzel vyššího celku /3.0/, ukončíme plánování na této cestě a vezmeme další prvek z tabulky C.

Po zpracování všech uzlů daného celku pokračujeme obdobným způsobem v plánování vyšších celků. Pro lepší názornost si ukážeme uložení některých uzlů v tabulce B, viz obr. 4. Uzly se ukládají do tabulky ve stejném pořadí v jakém jsou uspořádány operace v kmenovém souboru zakázek. Z titulu omezeného prostoru pro náš obrázek předpokládejme, že uzly vytváříme počínaje třetí součástí a konče 2. operací deváté součástí. Význam jednotlivých hodnot ukazuje následující schéma



i. ..... index položky v poli B
LL ..... index levého následníka
U ..... uzel zakázky
RL ..... index pravého následníka
NASL .... následující uzel ve struktuře zakázky



obr. 4

5. Závěr

Uvedené příklady byly z oblasti sekvenčně zpracovávaných souborů. Použití stromových struktur v uvedených případech přispělo zejména ke snížení počtu kroků a ke zrychlení algoritmu. Zvyšuje se také přehlednost programů. Při zpracování souborů s jinou organizací přínos stromových struktur bude nejspíše ve zrychlení zpracování v důsledku snížení počtu I/O operací na externí média.

S použitím stromových struktur je třeba uvažovat již ve fázi návrhu, kdy zpracování rozčleňujeme do jednotlivých kroků a tvoříme jejich algoritmy. Stromové struktury se nám osvědčily také při realizaci uživatelem požadovaných změn do již existujících programů. Bez nich by práce na požadavku trvala podstatně déle a ve zpracování by přibylo několik dalších kroků.

Literatura:

Knuth - Umění programování, 3. část - třídění a vyhledávání /ruský překlad/