Přihrádkové třídění řetězců

Aleš Novák, ADS I., 30.6.2011

Ke třídění množiny prvků, které nabývají hodnot převoditelných na přirozená čísla v omezeném, „rozumném“ rozsahu, můžeme použít přihrádkové třídění – bucketsort. Bucketsort prochází vstupní množinu a spočítá pro každou podmnožinu prvků stejné hodnoty „přihrádku“ ve výstupním, již setříděném poli.

Máme-li N prvků o rozsahu $0 \ldots R$ ve vstupním poli „vstup“, algoritmus v pseudo-C kódu vypadá takto:

```
1: vstup [1..N], vysledek [1..N], prihradky [0..R+1]
2: prihradky = [0,0,0..0]
3: for i in 1..N:
4:     prihradky [vstup [i] + 1] ++

5: prihradky [0] = 0
6: for r in 1..R:
7:     prihradky [r] += prihradky [r-1]

8: for i in 1..N:
9:     vysledek [prihradky [vstup [i]]++] = vstup [i]
```

Vidíme, že používáme ještě jedno pole velikosti N pro výstup a pole četností prvků dané hodnoty o velikosti R. Podle limitů cyklů je vidět časová složitost $O(N + R)$.

Cyklus na řádcích 3–4 počítá četnost jednotlivých hodnot prvků. Cyklus na řádcích 5–7 četnost hodnoty prvku nahrazuje sumou četností prvků s menší hodnotou, tím se z údaje „kolikrát se prvek s bezprostředně nižší hodnotou vyskytuje ve vstupním poli“ stane údaj „kde začíná přihrádka prvků s touto hodnotou ve výstupním poli“.

V posledním cyklu procházíme vstupní pole zepředu a přihrádky plníme také zepředu, takže vzájemné pořadí prvků stejné hodnoty se nemění – třídění je stabilní.

Poznámka: někde (např. ve zbytku tohoto textu a v ukázkovém C programu) je použita podoba algoritmu, která místo začátků počítá konce přihrádek a plní je pak odzadu. Na složitost algoritmu to nemá vliv.

Třídění řetězců

Pokud máme třídit množinu *řetězců*, složených z L prvků – znaků, jejichž hodnoty mají „rozumný“ rozsah, bychom mohli využít toho, že bucketsort je stabilní, a setřídit řetězce nejprve podle nejméně významného, tj. L-tého znaku, pak podle $(L-1)$-tého až nakonec podle 1. znaku, který je nejvýznamnější. Každý znak by se prošel právě jednou, časová složitost je tedy jako při provedení L samostatných bucketsortů - $O(L \cdot (N + R))$.

Pokud ale řetězce nabývají délek $1 \ldots L$, je situace mírně komplikovanější. Pokud bychom všechny řetězce zarovnali „mezerami“ na délku L a použili algoritmus z předchozího odstavce, může být časová složitost až $O(N^2)$ (pokud budememe mít $N - 1$ řetězců délky 1 a jeden délky N). Tj. většinu času strávíme počítáním četnosti mezer.

Jak to udělat s lineární časovou složitostí? Budeme opět porovnávat od nejméně významného, L-tého znaku - ovšem jen u těch řetězců, které mají alespoň L znaků. Pro ten účel nejprve vstupní množinu rozdělíme bucketsortem na podmnožiny P_i, kde množina P_i obsahuje znaky délky i, budeme postupovat od největšího i k nejmenšímu a postupně vytvářet výstupní pole. Po kroku pro přihrádku P_i bude dočasná výstupní množina obsahovat všechny řetězce délky minimálně i setříděné podle $i, i+1, i+2, ..., L$ znaku. Postup v pseudo-pseudo kódu bude takovýto:

```
 1: vstup [1..N], vysledek [1..N], prihradky [0..R],
    vstup_dle_delek [1..N], prihradky_delek [1..L],
    vysledek_docasny [1..N]

 2: prihradky_delek = [0,0,0..0]

 3: for i in 1..N:
 4:     prihradky_delek [delka (vstup[i]) ++

 5: for k in 2..L:
 6:     prihradky_delek [k] += prihradky_delek [k-1]

 7: for i in N..1:
 8:     vstup_dle_delek [-- prihradky_delek [vstup [i]]] = vstup [i]

 9: vysledek = []

10: k = L
11: while k > 0:
12:     vysledek = concat (
          vstup_dle_delek [prihradky_delek[k-1]..prihradky_delek [k]],
          vysledek)

13:     prihradky = [0,0,0..0]

14:     for i in 1..delka (vysledek):
15:         prihradky [vysledek [i][k]] ++

16:     for r in 1..R:
17:         prihradky [r] += prihradky [r-1]

18:     for i in N..1:
19:         vysledek_docasny [-- prihradky [vysledek [i]]] = vysledek [i]

20:     vysledek = vysledek_docasny
21:     k --
```

Vidíme, že třídění provádíme L-krát, každý znak v každém řetězci se prochází právě jednou, časová složitost je $O(RL + \sum_i^N delka(vstup[i]))$, tedy je lineární s celkovým počtem znaků všech řetězců. Paměťová náročnost je $N + R + L$, takže je třeba zaručit, že také maximální délka řetězce bude „rozumná“.

Správnost algoritmu nahlédneme indukcí - tvrdíme, že po i-tém kroku je dočasná výstupní množina setříděná podle znaku $(L - i + 1)$ až L. Po prvním kroku to je triviálně vidět, protože dočasná výstupní množina obsahuje pouze řetězce z P_L, setříděné podle L. znaku. Při i-tém kroku se před dočasný výstupní seznam připojí množina P_{L-i+1} a dohromady se třídí podle $(L - i + 1)$-

tého znaku. Pokud dva řetězce mají hodnotu znaku stejnou, zachovává se jejich vzájemné pořadí (bucketsort je stabilní), takže se udržuje jejich slovníkové uspořádání z přechozího kroku. Pokud je některý z nich z množiny P_{L-i+1}, je správně umístěn více vepředu, protože ve slovníkovém uspořádání řetězec ABC předchází řetězci $ABCD$.

Přikládám ukázkové řešení v jazyce C. Použil jsem dvě pole velikosti N, která prohazuji, jedno vždy obsahuje dočasně setříděnou množinu řetězců délky $k+1$, do druhého pole jde setříděné zřetězení množiny P_k a množiny z prvního pole. Je třeba zřetězovat je v tomto pořadí (protože řetězec ABC má ve výstupu předcházet řetězci $ABCD$).

Ukázkový postup algoritmu

Nakonec uvádím ukázku třídění řetězců: `abc`, `andalusan`, `b`, `cadmium`, `calcium`, `dac`, `kleptoman`, `zetor`, `zita`. Každá dvojice tabulek představuje jeden krok algoritmu. Levá tabulka obsahuje vstupní množiny, dvojitá čára odděluje množinu P_{L+1-i} pro i-tý krok nahoře, dočasný výsledek dole. Pravá tabulka obsahuje dočasný výsledek po konci kroku. Podbarvené jsou znaky, podle kterých se setřiďuje.

a	n	d	a	l	u	s	a	n
k	l	e	p	t	o	m	a	n

a	n	d	a	l	u	s	a	n
k	l	e	p	t	o	m	a	n

a	n	d	a	l	u	s	a	n
k	l	e	p	t	o	m	a	n

a	n	d	a	l	u	s	a	n
k	l	e	p	t	o	m	a	n

c	a	l	c	i	u	m		
c	a	d	m	i	u	m		
a	n	d	a	l	u	s	a	n
k	l	e	p	t	o	m	a	n

c	a	l	c	i	u	m		
c	a	d	m	i	u	m		
k	l	e	p	t	o	m	a	n
a	n	d	a	l	u	s	a	n

c	a	l	c	i	u	m		
c	a	d	m	i	u	m		
k	l	e	p	t	o	m	a	n
a	n	d	a	l	u	s	a	n

k	l	e	p	t	o	m	a	n
c	a	l	c	i	u	m		
c	a	d	m	i	u	m		
a	n	d	a	l	u	s	a	n

z	e	t	o	r				
k	l	e	p	t	o	m	a	n
c	a	l	c	i	u	m		
c	a	d	m	i	u	m		
a	n	d	a	l	u	s	a	n

c	a	l	c	i	u	m		
c	a	d	m	i	u	m		
a	n	d	a	l	u	s	a	n
z	e	t	o	r				
k	l	e	p	t	o	m	a	n

z	i	t	a					
c	a	l	c	i	u	m		
c	a	d	m	i	u	m		
a	n	d	a	l	u	s	a	n
z	e	t	o	r				
k	l	e	p	t	o	m	a	n

z	i	t	a					
a	n	d	a	l	u	s	a	n
c	a	l	c	i	u	m		
c	a	d	m	i	u	m		
z	e	t	o	r				
k	l	e	p	t	o	m	a	n

d	a	c						
a	b	c						

z	i	t	a					
a	n	d	a	l	u	s	a	n
c	a	l	c	i	u	m		
c	a	d	m	i	u	m		
z	e	t	o	r				
k	l	e	p	t	o	m	a	n

d	a	c						
a	b	c						
a	n	d	a	l	u	s	a	n
c	a	d	m	i	u	m		
k	l	e	p	t	o	m	a	n
c	a	l	c	i	u	m		
z	i	t	a					
z	e	t	o	r				

d	a	c						
c	a	d	m	i	u	m		
c	a	l	c	i	u	m		
a	b	c						
z	e	t	o	r				
z	i	t	a					
k	l	e	p	t	o	m	a	n
a	n	d	a	l	u	s	a	n

b								
d	a	c						
c	a	d	m	i	u	m		
c	a	l	c	i	u	m		
a	b	c						
z	e	t	o	r				
z	i	t	a					
k	l	e	p	t	o	m	a	n
a	n	d	a	l	u	s	a	n

a	b	c						
a	n	d	a	l	u	s	a	n
b								
c	a	d	m	i	u	m		
c	a	l	c	i	u	m		
d	a	c						
k	l	e	p	t	o	m	a	n
z	e	t	o	r				
z	i	t	a					

Reference

[1] Zápisy z ADS I. - Martin Mareš `http://mj.ucw.cz/vyuka/1011/ads1/`

[2] Přihrádkové třídění - Tomáš Slavíček `http://www.vbnet.cz/`