Mgr. Ján Guniš, gunis@gjar-po.sk, Gymnázium J. A. Raymana, Prešov

Triedenie sekvenčných súborov

Priame zlučovanie, verzia 1

I keď sme si ukázali výhody a rýchlosť vnútorného triedenia, nie vždy je táto možnosť použiteľná. Ide predovšetkým o prípady, keď množstvo dát, ktoré treba utriediť, je také veľké, že sa nezmestí do operačnej pamäte. Vtedy sme nútení pracovať s dátami, ktoré sú uložené na periférnych pamäťových zariadeniach, napr. páskach alebo diskoch. Obmedzenie, ktoré nám táto reprezentácia prináša je pomerne silné. V každom okamihu máme priamy prístup len k jednému prvku súboru. Takže pri triedení dát v súboroch sme nútení používať iné techniky, než ako tomu bolo pri triedeniach vnútorných.

Jednou z najpoužívanejších a najdôležitejších techník je triedenie zlučovaním. Zlučovanie znamená, spájanie dvoch (alebo viacerých) utriedených postupností do jednej usporiadanej postupnosti prostredníctvom opakovaného výberu s pomedzi momentálne prístupných prvkov.

Zlučovanie je podstatne jednoduchšie ako triedenie a používa sa ako pomocná operácie pre zložitejší proces triedenia sekvenčných súborov.

Jedným z najjednoduchších algoritmov vonkajšieho triedenia zlučovaním je priame zlučovanie, ktoré si môžeme popísať nasledovne:

1. súbor A sa rozdelí na dve polovice, ktoré označíme B a C
2. spájaním prvkov do usporiadaných dvojíc (ak je počet prvkov párny) sa postupnosti B a C zlúčia do novej postupnosti A
3. na postupnosť A sa opätovne aplikujú kroky 1. a 2., čím získame postupnosť usporiadaných štvoríc (ak je ich počet deliteľný 4)
4. kroky 1. ž 3. sa opakujú, takže z usporiadaných štvoríc, dostaneme osmice (ak je počet deliteľný 8) atď., pričom v rámci každého kroku sa dĺžka zlučovaných postupností zdvojnásobuje. Algoritmus končí, keď je usporiadaná celá postupnosť.

Aplikácia uvedeného algoritmu na postupnosti `A=[10 2 3 7 8 9 4 1 5 6]` vyzerá nasledovne:

1. rozdelenie
```
B=[10 3 8 4 5]
C=[2 7 9 1 6]
```
2. zlúčenie B a C do novej postupnosti A (získavame usporiadané dvojice)
```
A=[2 10 3 7 8 9 1 4 5 6]
```
3. opakovanie krokov 1. a 2.

rozdelenie (po dvojiciach):
```
B=[2 10 8 9 5 6]
C=[3 7 1 4]
```
zlúčenie (do usporiadaných štvoríc):
```
A=[2 3 7 10 1 4 8 9 5 6]
```

4. opakovanie krokov 1. –3.

rozdelenie (po štvoriciach): `B=[2 3 7 10 5 6]`
`C=[1 4 8 9]`
zlúčenie (do usporiadaných osmíc): `A=[1 2 3 4 7 8 9 10 5 6]`

rozdelenie (po osmiciach): `B=[1 2 3 4 7 8 9 10]`
`C=[5 6]`
zlúčenie (do usporiadaných šesťnástic) `A=[1 2 3 4 5 6 7 8 9 10]`

Pozorný čitateľ si iste všimol, že pokiaľ počet prvkov v súbore nie je mocninou čísla 2, prichádza k istej asymetrii pri rozdeľovaní prvkov z C do A, B (počet usporiadaných n-tíc môže byť rôzny, maximálne však o jednu). Takisto pri zlučovaní A a B do C, môže nastať situácia, že posledná zlúčená usporiadaná n-tica nebude úplná (teda nebude mať n prvkov). To ale nejako zvlášť neovplyvní úspešnosť nášho algoritmu.

Každá operácia, pri ktorej sa manipuluje naraz z celou množinou údajov sa nazýva fáza. Najmenší podproces, ktorého opakovaním sa realizuje triedenie sa nazýva prechod, alebo krok. Usporiadanú postupnosť budeme nazývať beh. My sme potrebovali štyri prechody, z ktorých každý pozostával z fáz rozdelenia postupnosti a zlúčenia.

Na utriedenie postupnosti sme potrebovali 3 súbory (pásky). Takýto spôsob triedenia nazývame **priame 2 fázové 3 páskové zlučovanie**.

Uvedený algoritmus zapísaný v jazyku Turbo Pascal by mohol vyzerať nasledovne:

```
 1: program priame_2fazove_3paskove_zlucovanie;
 2: uses crt;
 3: const pocet = 100;                                    {pocet zaznamov v subore}
 4: type zaznam = char;                                      {typ zaznamu v subore}
 5:      paska = file of zaznam;                           {definovanie typu suboru}
 6: var subor : paska;                                        {prepojenie na subor}
 7:     x : zaznam;                                 {premenna pre citanie zo suboru}
 8:
 9: procedure generuj (var a : paska);                  {generovanie prvkov suboru}
10: var x_a : zaznam;
11:     i : longint;
12: begin
13:  rewrite(a);randomize;
14:  for i := 1 to pocet  do begin                   {generovanie prvkov suboru a}
15:                           x_a := chr(random(26)+97);
16:                           write(a, x_a);
17:                          end;
18: end;
19:
20: procedure utried (var a : paska);
21: var b,c : paska;
22:     x_a, x_b, x_c : zaznam;
23:     pocet_zaznamov, poc_b, poc_c, dlzka : longint;    {pocitadla pri zapisoch}
24:     cit_b, cit_c : boolean;                      {z ktoreho suboru sa ma citat}
25:
26:  procedure rozdel;
27:  var poc : longint;
28:  begin
29:   reset(a);rewrite(b);rewrite(c);
30:   repeat
```

```
31:    poc := 0;
32:    while not(eof(a)) and (poc < dlzka) do begin
33:                                            read(a,x_a);
34:                                            write(b,x_a);
35:                                            inc(poc);
36:                                           end;
37:    poc := 0;
38:    while not(eof(a)) and (poc < dlzka) do begin
39:                                            read(a,x_a);
40:                                            write(c,x_a);
41:                                            inc(poc);
42:                                           end;
43:   until eof(a);
44:  end;
45:
46:  procedure zluc;
47:  begin
48:   reset(b); reset(c); rewrite(a);
49:   poc_b := 0; poc_c := 0;     {pocet zaznamov precitanych z jednotlivych behov}
50:   repeat                                                        {zlucovanie behov}
51:    cit_b := true; cit_c := true;         {z ktoreho suboru sa bude prvok citat}
52:                           {pokial sa aspon z jedneho suboru neprecital cely beh}
53:    while (poc_b < dlzka) and (poc_c < dlzka) do
54:          begin
55:           if cit_b then read(b, x_b);
56:           if cit_c then read(c, x_c);
57:           if x_b < x_c
58:              then begin
59:                    write(a, x_b);
60:                    inc(poc_b);
61:                    cit_b := true; cit_c := false;
62:                    {ak z B cely beh, alebo koniec B}
63:                    if (poc_b = dlzka) or (eof(b))
64:                       then begin                       {dokopiruj zvysok behu z C}
65:                             write(a,x_c); inc(poc_c);
66:                             while (poc_c<dlzka) and not(eof(c)) do
67:                                    begin
68:                                     read(c,x_c); write(a,x_c); inc(poc_c);
69:                                    end
70:                             end
71:                  end
72:              else begin
73:                    write(a, x_c);
74:                    inc(poc_c);
75:                    cit_c := true; cit_b := false;
76:                    {ak z C cely beh, alebo koniec C}
77:                    if (poc_c = dlzka) or (eof(c))
78:                       then begin                       {dokopiruj zvysok behu z B}
79:                             write(a,x_b); inc(poc_b);
80:                             while (poc_b<dlzka) and not(eof(b)) do
81:                                   begin
82:                                    read(b,x_b); write(a,x_b); inc(poc_b);
83:                                   end
84:                             end
85:                  end;
86:          end;
87:    poc_b := 0; poc_c := 0;
88:   until eof(b) or eof(c);     {ak sa z jedneho precitalo vsetko, tak sa konci}
89:  end;
90:
91: begin
92:  assign(b,'b.dat'); rewrite(b);                  {vytvorenie pomocnych suborov}
93:  assign(c,'c.dat'); rewrite(c);
94:  pocet_zaznamov := filesize(a);                        {pocet poloiek v subore}
95:  dlzka := 1;                                           {dlzka aktualnych behov}
96:  while dlzka < pocet_zaznamov do   {triedenie <=> dlzka behu < pocet zaznamov}
97:        begin
98:         rozdel;
```

```
 99:          zluc;
100:          while not(eof(b)) do begin read(b, x_b); write(a, x_b); end;
101:          while not(eof(c)) do begin read(c, x_c); write(a, x_c); end;
102:          dlzka := dlzka * 2;
103:         end;
104:  erase(b); erase(c);
105: end;
106:
107: begin
108:  clrscr;
109:  assign(subor,'a.dat');
110:  generuj(subor);
111:  utried(subor);
112:  reset(subor);
113:  while not(eof(subor)) do begin read(subor, x); write(x,' ');end;
114:  close(subor);
115:  readln;
116: end.
```

V hlavnom programe sa volá procedúra generuj, ktorá generuje obsah súboru A. Počet záznamov je *const pocet = 100* a záznam je typu *char*. Char sa javí ako celkom vhodný, pretože počas ladenia programu si môžeme priebežne kontrolovať obsah súborov. Zmenou definície typu *zaznam* a úpravou procedúry *generuj* je možné pracovať s ľubovoľným typovým súborom. Následne sa volá procedúra *utried*, ktorá sa postará o samotné utriedenia záznamov v súbore.

Procedúra *utried*, obsahuje lokálne procedúry *rozdel* (rozdelí súbor A do B a C), *zluc* (zlúči B a C do A). Telo procedúry striedavo volá *rozdel* a *zluc* a to dovtedy, kým *dlzka* < *pocet_zaznamov* (dĺžka behu je menšia ako počet záznamov v súbore).

Možné vylepšenie, priame zlučovanie, verzia 2

Vzhľadom na to, že fáza rozdeľovania nespôsobuje preusporiadanie prvkov, nezaraďujeme ju do triedenia. V tomto zmysle je vlastne neproduktívnou, aj keď predstavuje polovicu všetkých kopírovacích operácií. Pokúsme sa ju teda odstrániť a spojiť fázu zlučovania a rozdeľovania do jednej fázy. Namiesto zlučovanie behov z dvoch súborov do jedného ich budeme zlučovať a rovnomerne distribuovať do dvoch súborov, ktoré budú predstavovať vstup do ďalšej fázy. Na rozdiel od prvej dvojfázovej metódy je táto jednofázová podstatne výhodnejšia, aj keď je potrebný štvrtý súbor. Táto metóda sa nazýva **priame jednofázové 4 páskové zlučovanie**, alebo aj **vyvážené zlučovanie**.

Triedenie postupnosti `A=[10 2 3 7 8 9 4 1 5 6]` bude teraz vyzerať nasledovne:

rozdeľ A do C a D: `C=[10 3 8 4 5]`
`D=[2 7 9 1 6]`

zlúč C a D a rovnomerne ich rozdeľ do A a B: `A=[2 10 8 9 5 6]`
vytvárame dvojice `B=[3 7 1 4]`

zlúč A a B a rovnomerne ich rozdeľ do C a D: `C=[2 3 7 10 5 6]`
vytvárame štvorice `D=[1 4 8 9]`

zlúč C a D a rovnomerne ich rozdeľ do A a B: `A=[1 2 3 4 7 8 9 10]`
vytvárame osmice `B=[5 6]`

zlúč A a B a rovnomerne ich rozdeľ do C a D: `C=[1 2 3 4 5 6 7 8 9 10]`
vytvárame šesťnástice `D=[]`

Dĺžka kompletného behu je 16, počet záznamov v súbore je 10, algoritmus končí.

Ideálny prípad je, ak počet záznamov v súbore je mocninou 2. V takom prípade vieme distribuovať behy rovnomerne a pri zlučovaní nevznikajú žiadne nekompletné behy. Ak ale táto podmienka nie je splnená, nemá to žiaden vplyv na úspešnosť triedenia. Uvedený algoritmus zapísaný v jazyku Turbo Pascal by mohol vyzerať nasledovne:

```
 1: program priame_1fazove_4paskove_zlucovanie;
 2: uses crt;
 3: const pocet = 10;                          {pocet generovanych zaznamov v subore}
 4: type zaznam = char;                                  {typ zaznamu v subore}
 5:      paska = file of zaznam;                       {definovanie typu suboru}
 6: var subor : paska;                                    {prepojenie na subor}
 7:     x : zaznam;                                {premenna pre citanie zo suboru}
 8:
 9: procedure generuj (var a : paska);                {generovanie prvkov suboru}
10: var x_a : zaznam;
11:     i : longint;
12: begin {generuj}
13:  rewrite(a);randomize;
14:  for i := 1 to pocet  do begin                  {generovanie prvkov suboru a}
15:                           x_a := chr(random(26)+97);
16:                           write(a, x_a);
17:                          end;
18: end;
19:
20: procedure utried (var a : paska);
21: var c,b,d : paska;                                          {pomocne subory}
22:     x_a, x_b : zaznam;                              {hodnota precitana z A a B}
23:     pocet_zaznamov, poc_a, poc_b, dlzka : longint;    {pocitadla pri zapisoch}
24:     cit_a, cit_b : boolean;                        {z ktoreho suboru sa ma citat}
25:     poc : longint;                {pocet zaznamov prekopirovnych do C alebo  D}
26:     do_1 : boolean;          {prepinac, podla ktoreho sa kop. do A/C alebo B/D}
27:     abcd : boolean;                    {zlucujeme A a B do C a D, alebo naopak}
28:
29:  procedure rozdel;                                          {rozdeli A do C a D}
30:  begin {rozdel}
31:   reset(a);rewrite(c);rewrite(d);
32:    while not(eof(a)) do begin
33:                          read(a,x_a);
34:                          write(c,x_a);
35:                          if not(eof(a)) then begin
36:                                               read(a,x_a);
37:                                               write(d,x_a);
38:                                              end;
39:                          end;
40:  end;
41:
42:  procedure zluc(var a,b,c,d : paska);{zlucuje behy z A a B striedavo do C a D}
43:
44:   procedure zapis(hod : zaznam);       {podla prepinaca zapisuje do C alebo D}
45:   begin{zapis}
46:    if do_1 then write(c, hod)
47:            else write(d, hod);
```

```
48:    inc(poc);
49:    if poc = 2*dlzka then begin
50:                            poc := 0;
51:                            do_1 := not(do_1);
52:                           end;
53:   end;
54:
55:   begin{zluc}
56:    reset(a); reset(b);rewrite(c);rewrite(d);
57:    poc_a := 0; poc_b := 0;    {pocet zaznamov precitanych z jednotlivych behov}
58:    poc := 0;                              {pocet zaznamov zapisanych v zlucovani}
59:    do_1 := true;                                        {zaciname zapisovat do C}
60:    repeat                                                      {zlucovanie behov}
61:     cit_a := true; cit_b := true;        {z ktoreho suboru sa bude prvok citat}
62:                           {pokial sa aspon z jedneho suboru neprecital cely beh}
63:     while (poc_a < dlzka) and (poc_b < dlzka) do
64:           begin
65:            if cit_a then read(a, x_a);
66:            if cit_b then read(b, x_b);
67:            if x_a < x_b
68:               then begin
69:                     zapis(x_a);
70:                     inc(poc_a);
71:                     cit_a := true; cit_b := false;
72:                     {ak z A cely beh, alebo koniec A}
73:                     if (poc_a = dlzka) or (eof(a))
74:                        then begin                        {dokopiruj zvysok behu z B}
75:                              zapis(x_b);inc(poc_b);
76:                              while (poc_b<dlzka) and not(eof(b)) do
77:                                     begin
78:                                      read(b,x_b);zapis(x_b);inc(poc_b);
79:                                     end
80:                             end
81:                  end
82:               else begin
83:                     zapis(x_b);
84:                     inc(poc_b);
85:                     cit_b := true; cit_a := false;
86:                     {ak z B cely beh, alebo koniec B}
87:                     if (poc_b = dlzka) or (eof(b))
88:                        then begin                        {dokopiruj zvysok behu z A}
89:                              zapis(x_a);inc(poc_a);
90:                              while (poc_a<dlzka) and not(eof(a)) do
91:                                    begin
92:                                     read(a,x_a);zapis(x_a);inc(poc_a);
93:                                    end
94:                             end
95:                  end;
96:           end;
97:     poc_a := 0; poc_b := 0;
98:    until eof(a) or eof(b);     {ak sa z jedneho precitalo vsetko, tak sa konci}
99:   end;
100:
101:  begin{utried}
102:   assign(b,'b.dat');rewrite(b);                    {vytvorenie pomocnych suborov}
103:   assign(c,'c.dat');rewrite(c);
104:   assign(d,'d.dat');rewrite(d);
105:   pocet_zaznamov := filesize(a);                          {pocet zaznamov v subore}
106:   dlzka := 1;                                              {dlzka aktualnych behov}
107:   rozdel;                                  {rozdelenie vstupneho suboru A do B a C}
108:   abcd := true;             {smer prveho prerozdelenia, pred vykonom sa zneguje}
109:   while dlzka < pocet_zaznamov do  {triedenie <=> dlzka behu < pocet poloziek}
110:        begin
111:         abcd := not(abcd);
112:         if abcd then begin
113:                       zluc(a,b,c,d);
114:                       while not(eof(a)) do begin       {dokopiruje sa zvysok A}
115:                                              read(a, x);
```

```
116:                                              if do_1 then write(c, x)
117:                                                      else write(d, x);
118:                                             end;
119:                        end
120:                    else begin
121:                          zluc(c,d,a,b);                    {dokopiruje sa zvysok C}
122:                          while not(eof(c)) do begin
123:                                              read(c, x);
124:                                              if do_1 then write(a, x)
125:                                                      else write(b, x);
126:                                             end;
127:                        end;
128:          dlzka := dlzka * 2;                       {dlzka behov sa zdvojnasobila}
129:         end;
130:    if abcd then begin                          {triedenie sa skoncilo, upratuje sa}
131:                  close(a);erase(a); {pomocne sub. sa mazu a vysl. sa premenuje}
132:                  close(b);erase(b);
133:                  close(d);erase(d);
134:                  close(c);rename(c,'a.dat');
135:                  assign(a,'a.dat');
136:                 end
137:            else begin
138:                  close(b);erase(b);
139:                  close(c);erase(c);
140:                  close(d);erase(d);
141:                 end;
142:   end;
143:
144: begin
145:   clrscr;
146:   assign(subor,'a.dat');
147:   generuj(subor);
148:   utried(subor);
149:   reset(subor);
150:   while not(eof(subor)) do begin read(subor, x); write(x,' ');end;
151:   close(subor);
152:   readln;
153: end.
```

Popíšme si teraz činnosť programu.

Hlavný program volá procedúru *generuj*, ktorá vygeneruje obsah súboru a.dat. Záznam je typu *char*, ako bolo spomenuté vyššie, počas ladenia si vieme ľahko skontrolovať obsah jednotlivých súborov. Potom sa volá procedúra *utried*, ktorá realizuje samotné triedenie. V jej tele sa vytvoria pomocné súbory (b.dat, c.dat, d.dat), vstupný súbor A sa prerozdelí do C a D. Premenná *abcd* registruje smer prerozdeľovania (A a B do C a D, alebo naopak). Nasleduje samotné triedenie. Pokiaľ *dlzka* < *pocet_zaznamov* opakuje sa nasledovné:

Volá sa procedúra *zluc*, s parametrami A, B, C, D alebo C, D, A, B, podľa smeru zlučovania. Potom sa dokopíruje prípadný zvyšok v súbore A (resp. C) do C alebo D, (resp. A alebo B). Je to určené hodnotou premennej *do_1* (kopíruje sa do prvého alebo druhého z dvojice A, B resp. C, D). Dĺžka behu sa zdvojnásobí. A celý postup sa opakuje. Samozrejme prerozdeľovanie bude opačné (zmena hodnoty *abcd*). Po ukončení sa uprace diskový priestor, zmažú sa nepotrebné pomocné súbory, výsledná množina utriedených dát (môže byť buď v a.dat alebo c.dat) sa premenuje na a.dat a premenná A sa spojí so súborom a.dat.

Analýza priameho 1 fázového 4 páskové zlučovania

Vzhľadom na to, že po každom prechode sa hodnota premennej *dlzka* zdvojnásobí, a triedenie končí ak *dlzka* >= *pocet_zaznamov* , je potrebných $\lceil \log_2(\text{pocet\_zaznamov}) \rceil$ prechodov. Celkový počet presunov prvkov je

pocet_zaznamov . $\lceil \log_2(\text{pocet\_zaznamov}) \rceil$, pretože v každom prechode sa presunie presne pocet_zaznamov prvkov. Ak teda nepočítame prvé prerozdelenie A do C a D.

Otázku počtu porovnaní môžeme zanedbať pretože triedenie sa realizuje v súbore. Náročnosť operácie presunu prevažuje nad náročnosťou operácie porovnania často aj o niekoľko rádov.

Čo sa týka pamäťovej zložitosti, je rádovo 4 . pocet_zaznamov. Ostatné premenné sú zanedbateľné (i keď ide o operačnú pamäť).

Ďalšie možné vylepšenia, ale už len teoreticky :-)

1. Pri priamom zlučovaní nezískame žiadne výhody, ak sú záznamy už čiastočne usporiadané. Vždy zlučujeme behy s pevnými dĺžkami 1, 2, 4, 8 ... Riešením je zlučovať behy s čo možno najväčšou dĺžkou. Toto triedenie sa nazýva **prirodzené zlučovanie**.
2. Zložitosť algoritmu triedenia je priamo úmerná počtu potrebných prechodov, pretože každý prechod zahŕňa kopírovanie celej množiny záznamov. Jednou z možností ako redukovať počet prechodov, je distribúcia behov na viac ako dva súbory. Toto triedenie sa nazýva **vyvážené viaccestné zlučovanie**.
3. Predstavme si triedenie zlučovaním pomocou troch súborov A, B, C. V každom prechode sa záznamy z dvoch súborov zlučujú do tretieho súboru. Len čo sa dosiahne koniec jedného z dvoch vstupných súborov, zmení sa jeho orientácia na výstupný a triediaci proces pokračuje zlučovaním behov z pôvodného cieľového súboru a zostávajúceho vstupného súboru. Tento postup môžeme aplikovať aj na viac súborov. Takéto triedenie sa nazýva **polyfázové triedenie**.

Použité zdroje:

Wirth, N.: Algoritmy a štruktúry údajov, Bratislava, ALFA, 1989

http://www.chami.com/colorizer/