Wykład 5_3
Sortowanie zewnętrzne - c.d.

3. Algorytm sortowania za pomocą łączenia polifazowego
4. Algorytm ograniczania liczby serii za pomocą kopcowego rozdzielania serii początkowych
5. Podsumowanie

Definicja:
Algorytmami sortowania zewnętrznego nazywamy takie algorytmy, które sortują dane umieszczone w pamięci zewnętrznej.

Założenia:

1. Pamięć wewnętrzna ma ograniczone rozmiary

2. Pamięć zewnętrzna ma rozmiary „nieograniczone”

3. Czas dostępu do danych w pamięci wewnętrznej jest niezależny od położenia danych (np. dostęp indeksowany w tablicach)

4. Czas dostępu do danej w pamięci wewnętrznej jest dużo mniejszy od czasu dostępu do danej w pamięci zewnętrznej, stąd jest pomijany w szacowaniu wydajności algorytmów zewnętrznych

5. Bezpośrednio po zapisie lub odczycie danej w pamięci zewnętrznej dostęp do niej jest niesekwencyjny - zarówno do odczytu i zapisu

6. Czas dostępu do danych w pamięci zewnętrznej jest zależny od położenia- zaleca się sekwencyjne czytanie i zapis danych, gdyż koszt dostępu niesekwencyjnego jest dużo wyższy od sekwencyjnego

1.Algorytm sortowania za pomocą łączenia polifazowego

Przykład 1: Sortowanie polifazowe z trzema plikami

Serie w plikach			Idealne serie		Numer poziomu scalania - *l*	Liczba serii
f1	*f2*	*f3*	f_1	f_2		
13	8	0	13	8	6	21
5	0	8	8	5	5	13
0	5	3	5	3	4	8
3	2	0	3	2	3	5
1	0	2	2	1	2	3
0	1	1	1	1	1	2
1	0	0	1	0	0	1

$f_1^0 = 1$, $f_2^0 = 0$, $f_2^{l+1} = f_1^l$, $f_1^{l+1} = f_1^l + f_2^l$ dla $l > 0$

Jeśli $f_1^l = f_l$, to mamy wzory rekurencyjne definiujące ciąg Fibonacciego rzędu 1:

$f_{i+1} = f_i + f_{i-1}$ dla i ≥ 1 oraz $f_1 = 1, f_0 = 0$

Wniosek: Początkowe liczby serii na dwóch plikach muszą być dwoma kolejnymi elementami ciągu Fibonacciego rzędu 1, natomiast trzeci plik służy do łączenia serii na kolejnym poziomie.

Przykład 2: Sortowanie polifazowe z sześcioma plikami

Serie idealne					Serie w plikach						Numer poziomu	Liczba
f_1	f_2	f_3	f_4	f_5	*f1*	*f2*	*f3*	*f4*	*f5*	*f6*	scalania - *l*	serii
16	15	14	12	8	16	15	14	12	8	0	5	65
8	8	7	6	4	8	7	6	4	0	8	4	33
4	4	4	3	2	4	3	2	0	4	4	3	17
2	2	2	2	1	2	1	0	2	2	2	2	9
1	1	1	1	1	1	0	1	1	1	1	1	5
1	0	0	0	0	0	1	0	0	0	0	0	1

$$f_5^{l+1} = f_1^l$$

$$f_4^{l+1} = f_1^l + f_5^l = f_1^1 + f_1^{l-1}$$

$$f_3^{l+1} = f_1^l + f_4^l = f_1^1 + f_1^{l-1} + f_1^{l-2}$$

$$f_2^{l+1} = f_1^l + f_3^l = f_1^1 + f_1^{l-1} + f_1^{l-2} + f_1^{l-3}$$

$$f_1^{l+1} = f_1^l + f_2^l = f_1^1 + f_1^{l-1} + f_1^{l-2} + f_1^{l-3} + f_1^{l-4}$$

Stąd, jeśli

$$f_i = f_1^i$$

$$f_{i+1} = f_i + f_{i-1} + f_{i-2} + f_{i-3} + f_{i-4} \text{ dla i} \geq 4$$

$$f_4 = 1$$

$$f_i = 0 \quad \text{dla} \quad i \prec 4$$

Są to wzory rekurencyjne definiujące liczby Fibonacciego rzędu *p=4*

$$f_{i+1}^p = f_i^p + f_{i-1}^p + \ldots + f_{i-p}^p \text{ dla } i \geq p \text{ oraz}$$

$$f_p^p = 1$$

$$f_i^p = 0 \text{ dla } \quad 0 \leq i \leq p$$

Wniosek:

Początkowe liczby serii dla idealnego sortowania polifazowego z *n* plikami są sumami *n-1, n-2,...,1* kolejnych liczb Fibonacciego rzędu *n-2*. Oznacza to, że liczba serii jest sumą *n-1* takich sum Fibonacciego.

l \ n	3	4	5	6	7	8
1	2	3	4	5	6	7
2	3	5	7	9	11	13
3	5	9	13	17	21	25
4	8	17	25	33	41	49
5	13	31	49	65	81	97
6	21	57	94	129	161	193
7	34	105	181	253	321	385
8	55	193	349	497	636	769
9	89	355	673	977	1261	1531
10	144	653	1297	1721	2501	3049
11	233	1201	2500	3777	4961	6073
12	377	2209	4819	7425	9841	12097
13	610	4063	9289	14597	19521	24097
14	987	7473	17905	28697	38721	48001
15	1597	13745	34513	56417	76806	95617
16	2584	25281	66526	110913	152351	190465
17	4181	46499	128233	218049	302201	379399
18	6765	85525	247177	428673	599441	755749
19	10946	157305	476449	842749	1189041	1505425
20	17711	289329	918385	1656801	2358561	2998753

Tabela 6.1. Liczby serii dające się idealnie rozdzielić - zgodne z liczbami Fibonacciego rzędów: 1(3 pliki), 2(4 pliki), 3(5 plików), 4(6 plików), 5(7 plików), 6(8 plików).

Uwaga: W rzeczywistych plikach początkowo liczba serii nie zawsze jest idealną sumą. Wówczas należy zasymulować istnienie hipotetycznych pustych serii w ten sposób, aby suma serii rzeczywistych i hipotetycznych była sumą idealną. Podczas rozdzielania początkowego serii należy odpowiednio równomiernie „rozdzielać" serie fikcyjne oraz rzeczywiste na *n - 1* plików.

Liczba serii fikcyjnych jest zdefiniowana następująco:

$$d_i = f_i^l - f_i^{l-1} \quad \text{dla } i = 1,..,n-1 \text{ (numer pliku)}$$

Liczbę fikcyjnych serii należy wyznaczać w momencie ustalania kolejnego poziomu rozdzielania serii

Przykład 3: Zawartość pliku złożonego z 25 elementów - 12 serii

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
-1	-4	0	5	7	4	-4	8	-1	5	9	2	7	4	7	9	-5	-2	-5	-6	-2	-8	5	2	5

1.Po podziale 12 serii na 5 plików uzyskano 10 serii, idealnie rozłożonych na 1 i 2 poziomie, natomiast na poziome 3 uzupełnionych seriami fikcyjnymi

Poziom 1: a = (1, 1, 1, 1, 1, 0), d = (1, 1, 1, 1, 1, 0) → d = (0, 0, 0, 0, 0, 0)

Plik 1	1 seria	indeksy	0	d[1] = 1 → d[1] = 0				
		wartości	-1					
Plik 2	1 seria	indeksy	0	1	2	3	d[2] = 1 → d[2] = 0	
		wartości	-4	0	5	7		
Plik 3	1 seria	indeksy	0	d[3] = 1 → d[3] = 0				
		wartości	4					
Plik 4	1 seria	indeksy	0	1	d[4] = 1 → d[4] = 0			
		wartości	-4	8				
Plik5	1 seria	indeksy	0	1	2	d[5] = 1 → d[5] = 0		
		wartości	-1	5	9			

Poziom 2: a = (2, 2, 2, 2, 1, 0), d = (1, 1, 1, 1, 0, 0) → d = (0, 0, 0, 0, 0, 0)

Plik 1	2 serie	indeksy	1	2	3	4	5	d[1] = 1 → d[1] = 0
		wartości	*2*	*7*	4	7	9	
Plik 2	2 serie	indeksy	4	5	d[2] = 1 → d[2] = 0			
		wartości	-5	-2				
Plik 3	2 serie	indeksy	1	d[3] = 1 → d[3] = 0				
		wartości	-5					
Plik 4	2 serie	indeksy	2	3	d[4] = 1 → d[4] = 0			
		wartości	-6	-2				

Poziom 3: a = (4, 4, 4, 3, 2, 0), d = (2, 2, 2, 1, 1, 0) → d = (1, 2, 2, 1, 1, 0)

Plik 1	3 serie	indeksy	6	7	d[1] = 2 → d[1] = 1
		wartości	-8	5	
Plik 2	2 serie	indeksy	6	7	d[2] = 2 → d[2] = 1 → d[2] = 2
		wartości	*2*	*5*	

Podsumowanie (3 poziom) : a = (4, 4, 4, 3, 2, 0), d = (1, 2, 2, 1, 1, 0)

Plik 1	3 serie	wartości	-1	2	7	4	7	9	-8	5
Plik 2	2 serie	wartości	-4	0	5	7	-5	-2	2	5
Plik 3	2 serie	wartości	4	-5						
Plik 4	2 serie	wartości	-4	8	-6	-2				
Plik 5	1 seria	wartości	-1	5	9					

2.Łączenie 10 serii rozłożonych na 5 plikach - od trzeciego poziomu rozłożenia

2.1. łączenie plików 1, 4, 5 na pliku 6-tym; a= (4,4,4,3,2,0), poziom 3
→ d= (1,2,2,1,1,0), z=a[5]=2, k=0
→ d= (0,1,1,0,0,1), z=1, k=0
→ d=(0,0,0,0,0,1), z=1, k=3 - trzy pliki (*k*) po jednej serii (*z*)

Plik 1	3 serie	indeksy	0	1	2	3	4	5	6	7
		wartości	-1	2	7	4	7	9	-8	5
Plik 4	2 serie	indeksy	0	1	2	3				
		wartości	-4	8	-6	-2				
Plik 5 koniec	1 seria	indeksy	0	1	2					
		wartości	-1	5	9					
Plik 6		indeksy	0	1	2	3	4	5	6	7
		wartości	-4	-1	-1	2	5	7	8	9

2.2. łączenie plików 1, 2, 3, 4 na pliku 5-tym; a= (2,2,2,2,1,0), poziom 2
→ d= (1,0,0,0,0,0), z=a[5]=1, k=0
→ d= (1,0,0,0,0,0), z=1, k=0
→ d= (0,0,0,0,0,0), z=1, k=4 - cztery pliki (*k*) po jednej serii (*z*)

Plik 1	2 seria	indeksy	3	4	5	6	7					
		wartości	4	7	9	-8	5					
Plik 2	2 serie	indeksy	0	1	2	3	4	5	6	7		
		wartości	-4	0	5	7	-5	-2	2	5		
Plik 3	2 serie	indeksy	0	1								
		wartości	4	-5								
Plik 4 koniec	1 seria	indeksy	2	3								
		wartości	-6	-2								
Plik 5		indeksy	0	1	2	3	4	5	6	7	8	9
		wartości	-6	-4	-2	0	4	4	5	7	7	9

2.3. łączenie plików 5, 6, 1, 2, 3 na pliku 4-tym; a = (1, 1, 1, 1, 1, 0), poziom 1 $\rightarrow$ d = (0, 0, 0, 0, 0, 0), z=a[5]=1, k=5 - pięć plików (*k*) po jednej serii (*z*)

Plik 5	1 seria	indeksy	0	1	2	3	4	5	6	7	8	9
		wartości	-6	-4	-2	0	4	4	5	7	7	9
Plik 6	1 seria	indeksy	0	1	2	3	4	5	6	7		
		wartości	-4	-1	-1	2	5	7	8	9		
Plik 1	1 seria	indeksy	6	7								
		wartości	-8	5								
Plik 2	1 seria	indeksy	4	5	6	7						
		wartości	-5	-2	2	5						
Plik 3	1 seria	indeksy	1									
		wartości	-5									

Plik 4 - 1 seria - a = (1, 0, 0, 0, 0, 0), d=(0,0,0,0,0,0), poziom 0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	15	16	17	18	19
-8	-6	-5	-5	-4	-4	-2	-2	-1	-1	0	2	2	4	4	5	5	5	5	7	7	7	8	9	9

Uwaga: Plik, na którym następuje łączenie serii, jest ostatnim plikiem, natomiast staje się plikiem pierwszym w kolejnym kroku łączenia.

Algorytm sortowania polifazowego - poziom konceptualny

(1) *Dopóki nie wyczerpiesz pliku źródłowego lub rozłożysz serie na n-1 plikach wyjściowych zgodnie z wymaganiami 1-go poziomu, wykonuj, co następuje:*

(1.1) *Wyznacz na 1-ym poziomie kolejny aktywny plik wyjściowy zgodnie z planowaną liczbą serii idealnych czyli liczb Fibonacciego rzędu n-2*

(1.2) *Weź z pliku źródłowego po 1 serii i umieść na wyznaczonym pliku wyjściowym;*

(2) *Dopóki nie wyczerpiesz pliku źródłowego, wykonuj, co następuje:*

(2.1) *Wyznacz na wybranym poziomie aktywny plik wyjściowy zgodnie z planowaną liczbą serii idealnych czyli liczb Fibonacciego rzędu n-2. Jeśli osiągnięto idealną liczbę serii w tym pliku, przejdź do kroku* ***(2.3)***

(2.2) *Umieść na wyznaczonym pliku wyjściowym kolejną serię. Jeśli należy ona do serii umieszczonej jako ostatnia w pliku, umieść w nim następną serię i przejdź do kroku* ***(2.1)*** *lub zwiększ liczbę fikcyjnych serii tego pliku, jeżeli został wyczerpany plik źródłowy i przejdź do kroku* ***(3)****.*

(2.3) *Wybierz wyższy poziom rozkładania serii zgodnie z ciągiem Fibonacciego rzędu n-2, wyznacz idealną oraz fikcyjną liczbę serii dla każdego z n-1 plików wyjściowych;*

(3) *Ustaw n - 1 plików wejściowych;*

(4) *Dopóki nie połączysz elementów z wszystkich serii w jedną serię, czyli nie osiągniesz poziomu 0, wykonuj:*

(4.1) *Ustaw n - ty plik wyjściowy i wyznacz najmniejszą liczbę serii na danym poziomie {idealna liczba serii w pliku n-1};*

(4.2) *Dopóki nie wyczerpiesz wyznaczonej idealnej liczby serii, które należy połączyć na danym poziomie, wykonuj:*

(4.2.1) *wyznacz aktywne pliki wejściowe nie posiadające serii fikcyjnych;*

(4.2.2) *jeśli istnieją serie fikcyjne dla wszystkich plików wejściowych, to połącz po 1 serii z każdego pliku w jedną serię fikcyjną na pliku wyjściowym, w przeciwnym przypadku wykonaj:*

(4.2.3) *połącz po 1 serii fikcyjnej z każdego z nieaktywnych plików wejściowych, jeśli istnieją, oraz dopóki nie wyczerpiesz jednej serii rzeczywistej na każdym z aktywnych plików wejściowych, wykonuj:*

(4.2.2.1) *weź po jednej serii o tym samym numerze z każdego aktywnego pliku wejściowego;*

(4.2.2.2) *połącz te serie w jedną i umieść ją w pliku wyjściowym;*

(4.2.2.3) *eliminuj pliki z wyczerpaną serią lub wyczerpane;*

(4.2.4) *zmniejsz liczbę serii idealnych łączonych na danym poziomie;*

(4.3) *Zamień pliki, plik wyjściowy staje się plikiem wejściowym o numerze1;*

(4.4) *Zmniejsz poziom oraz wyznacz serie idealne oraz fikcyjne*

Algorytm sortowania polifazowego - poziom projektowy

n - liczba plikow; k - liczba aktywnych plikow wejściowych;
z - liczba serii łączonych na danym poziomie; poziom - liczba poziomów
i-numery na mapie plików wejściowych; j-numery plików wyjściowych
plik - file of obiekt; f0: plik - plik źródłowy; bufor: obiekt - bufor pliku
pliki=array[nrpliku] of plik; f: pliki- tablica plików wejściowych i wyjściowych
t, ta: array[nrpliku] of nrpliku-mapy numerów plikow wejściowych i wyjściowych
a, d: array[nrpliku] of integer, gdzie: a[j] - idealna liczba serii na pliku j
d[j] - liczba fikcyjnych serii na pliku j
ostatni: array[nrpliku] of integer, ostatni[j]-klucz ostatniego elementu pliku j

{ kopiuj serie początkowe}
(1) *dopóki not Eof(f0) or (j < n -1), wykonuj co następuje:*
(1.1) *wybierz_plik; kopiuj_serię;*
(1.2) *ostatni[j]= f0↑.klucz;*
{kopiuj resztę serii początkowych:}
(2) *dopóki not Eof(f0) wykonuj, co następuje:*
(2.1) *wybierz_plik;*
(2.2) *jeśli ostatni[j] ≤f0↑.klucz, to:* {skończ jedną serię i rozpocznij nową}
(2.2.1) *kopiuj_serię;*
(2.2.2) *jeśli Eof(f0), to d[j]=d[j]+1,w przeciwnym wypadku kopiuj_serię;*
(2.3) *w przeciwnym wypadku kopiuj_serię;*
(3) *i← 1; dopóki i<=n-1 wykonuj, co następuje: Reset(f[t[i]]); i ← i+1;*
{łącz serie z plików o numerach t[1]..t[n-1] na t[n]}
(4) *dopóki poziom > 0 wykonuj, co następuje:*
(4.1) *z ← a[n-1]; {liczba łączonych serii}; d[n]← 0; Rewrite(f[t[n]]);*
(4.2) *dopóki z > 0 wykonuj, co następuje:* {łącz po 1 serii}
{określ liczbę aktywnych plików wejściowych i połącz serie fikcyjne}
(4.2.1) *k ← 0; i ← 1; powtarzaj n - 1 razy, co następuje:*
(4.2.1.1) *jeśli d[i] > 0, to d[i]← d[i]-1,*
(4.2.1.2) *w przeciwnym wypadku: k ← k + 1; ta[k] ← t[i];*
(4.2.2) *jeśli k = 0, to d[n] ← d[n]+1;*
(4.2.3) *w przeciwnym wypadku połącz jedną serię rzeczywistą z każdym z plików ta[1]..ta[k];*
(4.2.4) *z ←z -1;*
(4.3) *Reset(f[t[n]]);* {ustaw plik wyjściowy do kolejnego łączenia}
(4.4) *zamień kolejne pliki w mapie t; oblicz a oraz d dla następnego poziomu;*
{ustaw plik wyjściowy na kolejnym poziomie, połączone dane znajdują się na t[1]}
(4.5)*Rewrite(f[tn]]); poziom ←poziom -1;*

```
        {sortowanie polifazowe z n plikami - główne procedury}
const n= 6;                                        {liczba plików}
      nazwa= 'seriapl';
type obiekt= record
               klucz: integer;
             end;
   nrpliku= 1..n;
   plik= file of obiekt;
   pliki= array[nrpliku] of plik;
   serie_plikow= array[nrpliku] of longint;
   mapa_plikow= array[nrpliku] of nrpliku;
   bufory= array[nrpliku] of obiekt;

procedure Wydruk(var f: plik; n: integer);
  {.....}
procedure Pokaz_plik(nazwa: string);
  {.....}
procedure Plikowe_sortowanie_polifazowe(nazwa: string);
  procedure Podziel_serie(var a, d: serie_plikow; var f: pliki; var f0: plik;
                                  var poziom: integer);
      procedure Wybierz_plik( var a, d: serie_plikow; var poziom: integer;
                                  var j: nrpliku);
       var i: nrpliku; z: longint;
         begin
           if d[j] < d[j+1] then inc(j)
          else
            begin
              if d[j]=0 then
               begin
                inc(poziom);
                z:= a[1];
                for i:= 1 to n-1 do
                 begin
                  d[i]:= z+a[i+1]-a[i];
                  a[i]:= z+a[i+1];
                 end;
               end;
              j:= 1;
            end;
          dec(d[j]);
         end;                                  {koniec procedury Wybierz_plik}
```

```
procedure Kopiuj_serie(var f0, f: plik; var buf: obiekt; var pisz: boolean;
                                  var ostatni: obiekt);
 var buf1: obiekt; koniec_serii: boolean;
  procedure Kopiuj(var f0, f: plik; var koniec_serii, pisz: boolean;
                              var buf2: obiekt);
    begin
       if not Eof(f0) and (FilePos(f0)=0) then     begin
         Read(f0, buf1); Write(f, buf1);
        end
       else     begin
         Write(f, buf2); buf1:= buf2;
        end;
       if Eof(f0) then     begin
         koniec_serii:= True; pisz:= false;
        end
       else     begin
         Read(f0, buf2);  koniec_serii:= buf1.klucz > buf2.klucz;
         pisz:= true;
        end;
    end;                                          {koniec procedury Kopiuj}
begin
  repeat
   Kopiuj(f0, f, koniec_serii, pisz, buf);
  until koniec_serii;
  ostatni:= buf1
end;                                          {koniec procedury Kopiuj_serie}

procedure Podziel_serie_poczatkowe(var a, d: serie_plikow;
                var f0: plik; var f: pliki;  var poziom: integer;  var j: nrpliku;
                var pisz: boolean; var ostatni: bufory; var buf: obiekt);
var i: nrpliku;
  begin                    {początkowa liczba serii idealnych i fikcyjnych}
     for i:= 1 to n-1 do
        begin
          a[i]:= 1; d[i]:= 1;
        end;
     poziom:= 1; j:= 1; a[n]:= 0; d[n]:= 0;
     repeat
        Wybierz_plik(a, d, poziom, j);
        Kopiuj_serie(f0, f[j], buf, pisz, ostatni[j]);
     until (Eof(f0) and not pisz) or (j=n-1);
    end;                          {koniec procedury Podziel_serie_poczatkowe}
```

```
  procedure Uzupelnij_serie_poczatkowe(var a, d:serie_plikow;
                    var f0: plik; var f: pliki; var poziom: integer;  var j: nrpliku;
                    var pisz: boolean; var ostatni: bufory; var buf: obiekt);
     begin
      while not Eof(f0) or pisz do
       begin
        Wybierz_plik(a, d, poziom, j);
        if ostatni[j].klucz <= buf.klucz then
         begin                                          {dalszy ciąg starej serii}
           Kopiuj_serie(f0, f[j], buf, pisz, ostatni[j]);        {oraz nowa seria}
           if (Eof(f0) and not pisz) then inc(d[j])
           else Kopiuj_serie(f0, f[j], buf, pisz, ostatni[j]);
         end                                                    {nowa seria}
        else Kopiuj_serie(f0, f[j], buf, pisz, ostatni[j]);
       end;
      end;                          {koniec procedury Uzupelnij_serie_poczatkowe}

var  j: nrpliku;                                        {wybrany plik}
     ostatni: bufory;{ostatnio zapisane elementy w bieżących seriach  plików czyli
                              ostatni[j]=klucz ostatniego obiektu w pliku j}
     buf: obiekt;                                       {pomocniczy bufor pliku}
     pisz: boolean;                          {zmienna określająca wyczerpanie serii}
begin
   Podziel_serie_poczatkowe(a, d, f0, f, poziom, j, pisz, ostatni, buf);
   Uzupelnij_serie_poczatkowe(a, d, f0, f, poziom, j, pisz, ostatni, buf);
end;                                                    {koniec procedury Podziel_serie}
```

```
procedure Polifazowe_laczenie_serii(var a, d: serie_plikow;
                                    var poziom: integer; var f: pliki; var t: mapa_plikow);
  procedure Lacz_po_jednej_serii(var dane, dane1: bufory; var k: integer;
                                 var ta, t: mapa_plikow; var f: pliki);
    var min, x: integer;i,  mx: nrpliku;
        koniec_pliku, koniec_serii: boolean;
        buf, buf0: obiekt;                          {pomocnicze bufory plików}
    begin
     repeat
      i:= 1; mx:= 1;
      min:= dane[ta[1]].klucz;
      while i < k do
       begin
        inc(i)  x:= dane[ta[i]].klucz;
        if x < min then
         begin
          min:= x; mx:= i;
         end;
       end;  {ta[mx] zawiera obiekt najmniejszy (znajduje się on w elemencie
              tablicy dane[ta[mx]]), umieść go na t[n]}
       koniec_pliku:= Eof(f[ta[mx]]);
       koniec_serii:= False;
       buf:= dane[ta[mx]];
       Write(f[t[n]], buf);
       if not koniec_pliku then
        begin
         Read(f[ta[mx]], dane[ta[mx]]);
         koniec_serii:= buf.klucz > dane[ta[mx]].klucz;
        end;
       if koniec_serii then
        begin       {pomiń ten plik - zapamiętaj w dane1 element z nowej serii}
         dane1[ta[mx]]:= dane[ta[mx]];
                    {oraz zlikwiduj luki w tab. ta oraz zmniejsz liczbę plikow k}
         ta[mx]:= ta[k]; dec(k);
        end;
       if koniec_pliku then
        begin          {pomiń ten plik f[ta[mx]] - zlikwiduj luki w tab. ta oraz
                        zmniejsz liczbę plikow k}
         ta[mx]:= ta[k]; dec(k);
        end;
      until k=0;
    end;                                  {koniec procedury Lacz_po_jednej_serii}
```

```
procedure Lacz_serie_na_aktywnym_pliku(var dane, dane1: bufory;
      var t: mapa_plikow; var f: pliki; var d: serie_plikow;  var z: longint);
 var i: nrpliku;
      k: integer;                    {k - liczba plikow zawierających łączone serie}
      ta: mapa_plikow;    {mapa plików umożliwiająca łączenie rzeczywistych
                           serii z plików bez serii fikcyjnych (d[i]=0)}
   begin
     repeat
        k:= 0;
        for i:= 1 to n-1 do
           if d[i] > 0 then dec(d[i])                    {31cz serie fikcyjne}
           else
            begin
             inc(k); ta[k]:= t[i];                {wyznacz aktywne pliki wejściowe}
            end;
          if k=0 then inc(d[n])        {łącz po 1 serii fikcyjnej z każdego pliku}
          else                        {lub łącz jedną rzeczywistą serią z t[1]..t[k]}
            Lacz_po_jednej_serii(dane, dane1, k, ta, t, f);
          dec(z);
     until z= 0;
     Reset(f[t[n]]);                    {i podaj pierwszy element do łączenia}
     if not Eof(f[t[n]]) then  Read(f[t[n]], dane1[t[n]]);
   end;                        {koniec procedury Lacz_serie_na_altywnym_pliku}

procedure Uporzadkuj_liczby_serii_i_numery_plikow(
                                  var a, d: serie_plikow; var t: mapa_plikow);
 var i,  tn: nrpliku;
      dn, z: longint;
  begin
     {zamień kolejno pliki przesuwając je o 1 w gore oraz oblicz liczby nowych
     serii idealnych w tablicy a oraz fikcyjnych w tablicy d}
     tn:= t[n]; dn:= d[n]; z:= a[n-1];
     for i:= n downto 2 do
       begin
        t[i]:= t[i-1];
        d[i]:= d[i-1];
        a[i]:= a[i-1]-z;
       end;
     t[1]:= tn; d[1]:= dn; a[1]:= z;          {połączone serie są teraz na t[1]}
  end;          {koniec procedury Uporzadkuj_liczby_serii_i_numery_plikow}
```

```
{początek procedury Polifazowe_laczenie_serri}
var i: nrpliku;          z: longint;
    dane, dane1: bufory;                    {tablice buforów łączonych plików}
  begin          {[31]cz z t[1]..t[n-1] na t[n]. Dane do łączenie w tablicach dane i
                                            dane1-czyli pierwsze elementy z kolejnych plików}
    for i:= 1 to n-1 do Reset(f[i]);
    for i:= 1 to n do t[i]:= i;
    Assign(f[t[n]],char(48+n)+'serpl');          {wybierz plik początkowy}
    for i:= 1 to n-1 do          {wczytaj pierwsze elementy serii z n-1 plikow}
     if not Eof(f[t[i]]) then Read(f[t[i]], dane[i]);
    dane1:= dane;
    repeat
       z:= a[n-1];                          {liczba serii do połączenia}
       d[n]:= 0;   Rewrite(f[t[n]]);                    {plik wyjściowy}
                          {łącz serie z plikow wg f[ta[i]] na aktywnym pliku f[tn]]}
       Lacz_serie_na_aktywnym_pliku(dane,dane1, t, f, d, z);
                                          {przygotuj nowe dane do kolejnego łączenia serii}
       Uporzadkuj_liczby_serii_i_numery_plikow(a, d, t);
       dane:= dane1;   {odtwórz pierwsze elementy serii na kolejnym poziomie }
       dec(poziom);
     until poziom=0;
  end;                                  {koniec procedury Polifazowe_laczenie_serii}
{początek procedury Plikowe_laczenie_polifazowe}
 var  i: nrpliku;   poziom: integer;
      a, d: serie_plikow; {a[j]=idealna liczba serii w pliku j}
                          {d[i]=liczba fikcyjnych serii w pliku j}
      t: mapa_plikow;     {mapa numerów plików pomocniczych}
      f: pliki;           {tablica zmiennych plikowych}
      f0: plik;           {f0 jest plikiem wejściowym z liczbami pseudolosowymi}
 begin
  Assign(f0, nazwa); Reset(f0);          {ustaw plik źródłowy do czytania}
  for i:= 1 to n-1 do                    {ustaw pliki pomocnicze do pisania}
   begin
    Assign(f[i], char(48+i)+'serpl'); Rewrite(f[i]);
   end;
  Podziel_serie(a, d, f, f0, poziom);{rozłóż serie początkowe na plikach roboczych}
  Polifazowe_laczenie_serii(a, d, poziom, f, t);          {połącz serie polifazowo}
                     {usuń pliki pomocnicze i zamień plik f[t[1]] na plik posortowany}
  for i:= 1 to n do  Close(f[t[i]]);
  Close(f0); Erase(f0);
  Rename (f[t[1]], nazwa);
 end;    {koniec procedury Plikowe_sortowanie _polifazowe}
```

5. Podsumowanie

1. Zakłada się, że w ciągu losowo rozłożonych kluczy spodziewana długość serii równa się ***2***.
2. Liczba ***k*** przebiegów w **sortowaniu zewnętrznym za pomocą łączenia naturalnego** jest równa $k = \log_2 r$, gdzie ***r*** jest liczbą serii, natomiast ***2*** oznacza liczbę plików. Całkowita liczba przesunięć elementów wynosi w najgorszym przypadku $r \lceil \log_2 (r) \rceil$,
3. W ***sortowaniu zewnętrznym za pomocą wielokierunkowego łączenia wyważonego*** korzystającego z ***N=2p*** plików, z ***r*** serii rozłożonych równomiernie na ***p*** plikach uzyskuje się r/p^k serii po ***k*** przebiegach, stąd liczba przebiegów ***k*** potrzebna do posortowania ***n*** elementów rozłożonych na ***p*** plikach jest równa: $k = \log_p(n)$. Całkowita liczba przesunięć elementów w najgorszym przypadku wynosi $n \lceil \log_p (n) \rceil$
3. ***Sortowanie polifazowe*** przy użyciu ***N*** plików działa zawsze jako łączenie ***N-1*** kierunkowe, a nie jako ***N/2*** kierunkowe. Liczba spodziewanych ***k*** przebiegów wynosi w przybliżeniu $\log_N(n)$, gdzie ***n*** jest liczbą elementów do posortowania, ***N*** zaś stopniem operacji łączenia. Daje to zmniejszenie liczby przebiegów w stosunku do wielokierunkowego łączenia wyważonego przy tej samej liczbie plików.
4. W ciągu losowo rozłożonych kluczy spodziewana długość serii równa się ***2***, natomiast po przejściu przez kopiec o rozmiarze ***m***, wynosi ***2m*** na podstawie analizy probabilistycznej. Stąd współczynnik usprawnienia wynosi ***m***.

Wniosek: Efektywne rozwiązanie sortowania dużych plików można rozwiązać przez:

- przygotowanie serii początkowych metodą **rozdzielania serii przez kopcowanie** (współczynnik zmniejszenia liczby serii w pliku jest równy rozmiarowi kopca czyli maksymalnej liczbie elementów tablicy reprezentującej kopiec);
- zastosowanie **sortowania polifazowego**.

Np. plik zawierający 54 290 055 168 serii początkowych (około 108 GB) po przejściu przez stóg o rozmiarze 32678 (liczba 2-bajtowych elementów stogu w segmencie pamięci równym 65536 bajtów) będzie zawierał 1656801 serii i w wyniku sortowania polifazowego może być posortowany w 20 częściowych przebiegach (tabela 6.1)

Wyniki pomiarów wykonanych za pomocą programu srt_plk2.exe w G:\LABOR\ZKRUK\WYK6

1. Plik nieposortowany

Rodzaj algorytmu	Liczba elementów					
	1000	3000	30000	60000	90000	200000
rozdzielanie serii przez kopcowanie	0.16	0.55	5.50	10.54	15.71	35.21
łączenie naturalne	5.93	22.02	1066.27	*	*	*
wielokierunkowe łączenie wyważone	1.32	4.23	53.27	113.75	171.15	416.99
łączenie polifazowe	1.15	4.12	51.35	105.119	163.28	384.75

2. Plik posortowany

Rodzaj algorytmu	Liczba elementów					
	1000	3000	30000	60000	90000	200000
rozdzielanie 1 serii przez kopcowanie	0.22	0.55	5.3	10.49	15.33	34.88
łączenie naturalne	0.33	0.99	9.89	*	*	*
wielokierunkowe łączenie wyważone	0.16	0.55	5.22	10.28	15.48	34.28
łączenie polifazowe	0.49	1.59	15.43	30.37	45.48	101.02

3. Plik nieposortowany z rozdzieloną liczbą serii przez kopcowanie

Rodzaj algorytmu	Liczba elementów					
	1000	3000	30000	60000	90000	200000
liczba serii idealna	36	107	1071	2142	3214	7142
liczba serii rzeczywista	35	102	1002	2002	3002	6668
rozdzielanie serii przez kopcowanie	0.16	0.55	5.50	10.66	16.2	35.49
łączenie naturalne	4.01	13.84	247.94	*	*	*
wielokierunkowe łączenie wyważone	0.88	3.13	42.45	83.16	140.55	349.05
łączenie polifazowe	0.82	2.91	39.44	82.12	126.10	306.43