

Algorytmy sortowania zewnętrznego

1 Wstęp

Bardzo często przy rozwiązywaniu praktycznych problemów spotykana jest konieczność posortowania danych. W przypadku niewielkiej ich ilości, która całkowicie mieści się w pamięci operacyjnej zalecane jest użycie znanych algorytmów sortowania wewnętrznego, takich jak np. quicksort. Jednak w przypadku ogromnej ilości danych lub niewielkiej dostępnej pamięci, algorytmy sortowania wewnętrznego nie dają się łatwo zastosować lub stają się powolne.

Problem ten doprowadził do powstania oddzielnej klasy algorytmów sortujących — algorytmów sortowania zewnętrznego. W algorytmach tych zakłada się dostępność niewielkiego, pomocniczego obszaru pamięci. Dodatkowo dostępna jest duża ilość miejsca na zewnętrznym, bardzo powolnym, nośniku pamięci — najczęściej dysku twardym lub taśmie magnetycznej.

2 Zewnętrzne sortowanie przez scalanie

Algorytm ten stanowi modyfikację sortowania przez scalanie. Przed przystąpieniem do jego omówienia, konieczne jest przyjęcie następujących założeń:

- Plik składa się z N danych do posortowania (np. liczb całkowitych)
- Jednorazowo w pamięci RAM mieści się M danych.
- Rozpatruje się przypadki, gdy $N \gg M$. Jeśli $N \leq M$, zalecane jest użycie algorytmu sortowania wewnętrznego (np. quicksort).

Algorytm ten, dla sortowania rosnąco, można przedstawić następująco:

1. Przyjmij, że plik wejściowy składa się z $\lceil \frac{N}{M} \rceil$ fragmentów (z których każdy mieści się w dostępnej pamięci).
2. (Iteracyjnie) każdy fragment wczytaj do pamięci i posortuj go dowolną metodą sortowania wewnętrznego.
3. $x=2$
4. Tak długo, aż nie zostanie tylko jeden fragment wykonuj:
 - (a) Scalaj ze sobą po dwa fragmenty. Spowoduje to powstanie $\lceil \frac{N}{xM} \rceil$ fragmentów. Każdy taki fragment jest już uporządkowany rosnąco.
 - (b) $x=x*2$

2.1 Przykład

Niech dane do posortowania mają postać:

1	5	6	2	3	7	1	2	9	3	4	1	5	4	4	2
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Oznacza to, że $N = 16$ liczb do posortowania. Ponadto, niech w pamięci na raz mieszczą się co najwyżej $M = 4$ liczby.

Na plik wejściowy, na podstawie (1) kroku algorytmu można spojrzeć, jak na zbiór $\lceil \frac{16}{4} \rceil = 4$ fragmentów. Fragmenty o numerach nieparzystych przepisuje się do 1. pliku, parzystych — do drugiego.

1	5	6	2	3	7	1	2	9	3	4	1	5	4	4	2
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Zgodnie z (2) dla każdego bloku oddzielnie dokonuje się wczytania do pamięci i posortowania, otrzymując:

1	2	5	6	1	2	3	7	1	3	4	9	2	4	4	5
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Ponieważ jest więcej, niż jeden blok (krok 4), więc należy scalać po dwa bloki:

1	2	5	6	+	1	2	3	7	=	1	1	2	2	3	5	6	7
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

oraz

1	3	4	9	+	2	4	4	5	=	1	2	3	4	4	4	5	9
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Wynik scalenia umieszcza się w pliku źródłowym otrzymując:

1	1	2	2	3	5	6	7	1	2	3	4	4	4	5	9
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Ponieważ jest więcej, niż jeden blok (krok 4), więc należy podzielić plik na bloki, przepisując każdy nieparzysty blok do 1. pliku a każdy parzysty — a następnie scalić te pliki. Spowoduje to powstanie następującego pliku:

1	1	1	2	2	2	3	3	4	4	4	5	5	6	7	9
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Ponieważ pozostał tylko jeden blok (krok 4), więc sortowanie kończy się

2.2 Warianty zewnętrznego sortowania przez scalanie

Wadą opisanego sortowania jest konieczność wykonywania dużej liczby przebiegów przez plik. Z tego powodu często proponuje się scalanie ze sobą co k części (w opisanym w poprzednim punkcie algorytmie oczywiście $k = 2$). Powoduje to spadek liczby przejść przez plik z $\log_2 \lceil \frac{N}{M} \rceil$ do $\log_k \lceil \frac{N}{M} \rceil$

2.3 Liczba plików używanych w algorytmie

Opisywany algorytm wymaga co najmniej trzech plików. Powoduje to konieczność wielokrotnego przemieszczania wskaźnika pliku („skakania” po pliku). Powoduje to znaczący spadek wydajności algorytmu w przypadku praktycznym.

Możliwe jest również zapisywanie każdego bloku w oddzielnym pliku. Daje to możliwość wyłącznie sekwencyjnego odczytu danych z pliku. Strategia ta również umożliwia zastosowanie wielu dysków twardych (na każdy blok po jednym) co znacząco przyspiesza działanie programu w rzeczywistym systemie. Metoda ta jednak znajduje zastosowanie bardzo rzadko z uwagi na konieczność użycia ogromnej liczby dysków. Problem ten doprowadził do wynalezienia modyfikacji opisanego algorytmu.

3 Zrównoważone scalanie trzykierunkowe

Algorytm ten stanowi rozwinięcie zewnętrznego sortowania przez scalanie. W metodzie tej zakłada się, że:

- Dysk podzielony jest na N danych do posortowania (np. liczb całkowitych)
- Jednorazowo w pamięci RAM mieści się M danych.
- Rozpatruje się przypadki, gdy $N \gg M$. Jeśli $M \leq N$, zalecane jest użycie algorytmu sortowania wewnętrznego (np. quicksorta).
- Dostępnych jest $2p$ urządzeń zewnętrznych (np. dysków twardych), z których można korzystać podczas sortowania.

W początkowym przejściu algorytmu dokonuje się podziału danych na bloki o długości m znaków. Każdy blok przed umieszczeniem w urządzeniu wyjściowym, podlega operacji sortowania (przy pomocy dowolnej metody wewnętrznej). Blok pierwszy umieszcza się na pierwszym urządzeniu, drugi — na drugim i tak

dalej. W momencie, gdy liczba bloków przekroczy p , kolejne bloki dopisywane są cyklicznie począwszy od urządzenia 1 aż do urządzenia p . Innymi słowy, dopisywanie bloków do urządzeń zachodzi cyklicznie modulo $p+1$. Po operacji podziału następuje konieczność scalenia p bloków na raz. Wynik każdego takiego scalenia jest umieszczany cyklicznie na urządzeniach wyjściowych począwszy od $p+1$, aż do $2p$. Jeśli pozostały niescalone bloki, dokonuje się powtórnej operacji scalenia. Rolę urządzeń wyjściowych pełnią w nim dyski od 1 do p . Takie cykliczne operacje scalania przy pomocy dysków od 1 do p oraz od $p+1$ do $2p$ wykonuje się aż do scalenia wszystkich bloków, czyli posortowania wszystkich danych. Sortowanie to nazywa się sortowaniem zrównoważonym z uwagi na równomierne wykorzystywanie wszystkich urządzeń zewnętrznych w procesie scalania. Metoda sortowania nie różni się od opisanej wcześniej; zmienia się jedynie sposób wykorzystywania dysków.

3.1 Przykład

Niech $2p = 4$. Oznacza to, że dostępne są 4 dyski twarde. Dane do posortowania mają postać:

1	5	6	2	3	7	1	2	9	3	4	1	5	4	4	2
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Oznacza to, że $N = 16$ liczb do posortowania. Ponadto, niech w pamięci na raz mieszczą się co najwyżej $M = 4$ liczby.

Na plik wejściowy, na podstawie (1) kroku algorytmu można spojrzeć, jak na zbiór $\lceil \frac{16}{4} \rceil = 4$ fragmentów. **Przed zapisaniem każdego bloku danych są one sortowane w pamięci** (jest to możliwe, gdyż każdy blok z założenia daje się zapisać w dostępnej pamięci). Na pierwszym dysku umieszcza się pierwszy blok, na drugim dysku — drugi. Czynność ta wykonywana jest cyklicznie, więc trzeci blok umieszczany jest na pierwszym dysku, zaś czwarty — na drugim.

p	Dane							
1	1	2	5	6	1	3	4	9
2	1	2	3	7	2	4	4	5
3	pusta							
4	pusta							

Następnym krokiem algorytmu jest scalenie kolejnych bloków — pierwszego bloku z pierwszego urządzenia z pierwszym z drugiego do urządzenia $p+1 = 3$. Drugi blok z pierwszego dysku jest następnie scalany z drugim blokiem drugiego dysku. Wynik scalenia jest zapisywany na $p+2 = 4$ urządzeniu. Czynności te powtarzane są cyklicznie, aż do scalenia wszystkich bloków z dysków 1 i 2:

p	Dane							
1	pusta							
2	pusta							
3	1	1	2	2	3	5	6	7
4	1	2	3	4	4	4	5	9

Następnie następuje scalenie bloków z urządzeń 3 i 4 do urządzenia 1. W ten sposób uzyskuje się dane posortowane.

4 Wielofazowe scalanie niezrównoważone

Algorytm scalania zrównoważonego charakteryzuje się koniecznością użycia dużej liczby dysków. W wielu praktycznych zastosowaniach, dąży się do zminimalizowania ich liczby przy jak najmniejszym zwiększeniu liczby przejść koniecznych do posortowania pliku. W algorytmie scalania wielofazowego korzysta się z podziału danych na części składające się z różnej liczby bloków. Po scalaniu pewnej liczby bloków, jeden z dysków pozostanie pusty. Może on zostać w następnym kroku użyty do przechowywania wyniku kolejnej operacji scalania. Eliminuje to konieczność posiadania $2p$ dysków.

Celem algorytmu jest zakończenie scalania tak, aby na dysku źródłowym (o numerze 0) pozostał jeden, posortowany blok. Zakładając, że dostępne są 3 dodatkowe napędy, bezpośrednio przed operacją scalania urządzenie 0 powinno pozostać puste, zaś na urządzeniach 1, 2, 3 powinien znajdować się po jednym bloku. W przedostatniej operacji scalania łączone są bloki z każdego z urządzeń 0, 1, 2. Postępując dalej w ten sposób możliwe jest zbudowanie tablicy rozkładu bloków. Strategia jej tworzenia jest następująca: z każdego wiersza wybiera się największą liczbę bloków b_{max} , zamienia ją na zero, a następnie dodaje do

wszystkich (poza wcześniej wybranym maksymalnym) elementów tego wiersza wartość b_{max} . Otrzymane wartości tworzą uogólniony ciąg Fibonacciego. Jeśli początkowa liczba bloków nie jest uogólnioną liczbą Fibonacciego, wprowadza się puste „bloki-atrapy”.

4.1 Przykład — wielofazowe scalanie niezerównoważone

Niech $p = 4$. Oznacza to, że dostępne są 4 dyski twarde: jeden źródłowy i 3 pomocnicze. Załóżmy, że do posortowania jest $N = 17$ bloków danych, po $M = 2$ liczby każdy. Przed przystąpieniem do sortowania konieczne jest wyznaczenie początkowej liczby bloków. Do ich wyznaczenia można użyć tablicy rozdzielającej.

4.1.1 Przygotowanie tablicy rozdzielającej

Założmy, że zostało wykonanych k operacji scalania. Z założenia, przy ostatniej (k -tej) operacji scalania, dysk 0 powinien zostać pusty (co oznaczono na rysunkach znakiem „X”), zaś na pozostałych dyskach znajdować się będą pojedyncze bloki.

	Dyski			
krok	0	1	2	3
k	X	1	1	1

Aby wyznaczyć liczbę bloków przypadających na każdy dysk w kroku $k - 1$, konieczne jest wyznaczenie maksymalnej liczby bloków w wierszu k . Wszystkie wartości są równe jeden, więc można wybrać dowolną z nich. Najkorzystniej będzie „wyzerować” dysk numer 3. Następnie należy skopiować dane z wiersza k do wiersza $k - 1$, dodając do każdego elementu wartość równą liczbie bloków wyzerowanego wiersza (w przykładzie należy dodać jeden). Na dysku 0 znajdzie się więc wartość $0 + 1$ (zero, gdyż w kroku k dysk był pusty), na dysku 1 znajdą się dwa bloki ($1 + 1 = 2$), na 2 znajdą się również dwa bloki ($1 + 1 = 2$). Rozważania te pozwalają na wypisanie następującej tablicy:

	Dyski			
krok	0	1	2	3
$k - 1$	1	2	2	X
k	X	1	1	1

Ponieważ w kroku $k - 1$ najwięcej bloków znajdowało się na dyskach 1 i 2, można wyzerować dowolny z nich. Najlepiej będzie wyzerować dysk 2. W kroku $k - 1$ na dysku 2 znajdowały się 2 bloki. Dlatego do wszystkich (poza oczywiście 2.) dysków będzie dodawana wartość 2. Ponieważ w kroku $k - 1$ na dysku 0 znajdował się 1 blok, więc w kroku $k - 2$ musiały tam być $1 + 2 = 3$ bloki. Na dysku 1 w kroku $k - 1$ znajdowały się 2 bloki, więc w kroku $k - 2$ musiały tam być $2 + 2 = 4$ bloki. Dysk 3 w kroku $k - 1$ był pusty, więc w kroku $k - 2$ musiały tam być $0 + 2 = 2$ bloki. Daje to następującą tabelę:

	Dyski			
krok	0	1	2	3
$k - 2$	3	4	X	2
$k - 1$	1	2	2	X
k	X	1	1	1

W kroku $k - 2$ najwięcej bloków znajdowało się na dysku 1. Konieczne było więc jego wyzerowanie. Dlatego do wszystkich (poza oczywiście 1.) dysków będzie dodawana wartość 4 — liczba bloków na dysku 1 w kroku $k - 2$. Na dysku 0 w kroku $k - 2$ znajdowały się 3 bloki, więc w kroku $k - 3$ musiały tam być $3 + 4 = 7$ bloków. Dysk 2 w kroku $k - 2$ był pusty, więc w kroku $k - 3$ musiały tam być $0 + 4 = 4$ bloki. Na dysku 3 w kroku $k - 2$ znajdowały się 2 bloki, więc w kroku $k - 3$ musiały tam być $2 + 4 = 6$ bloków. Pozwala to na napisanie następującej tabeli:

	Dyski			
krok	0	1	2	3
$k - 3$	7	X	4	6
$k - 2$	3	4	X	2
$k - 1$	1	2	2	X
k	X	1	1	1

W podobny sposób możliwe jest rozszerzenie tej tabeli:

	Dyski			
krok	0	1	2	3
$k - 5$	13	20	24	X
$k - 4$	X	7	11	13
$k - 3$	7	X	4	6
$k - 2$	2	4	X	2
$k - 1$	1	2	2	X
k	X	1	1	1

Tworzenie tabeli należy zakończyć w momencie, gdy suma najwyższego wiersza tabeli będzie większa lub równa ilości danych do posortowania. W przykładzie, w kroku $k - 5$ suma elementów z wiersza najwyższego wynosi $13 + 20 + 24 + 0 = 57$.

Należy zauważyć, że liczby z każdego wiersza tej tabeli to uogólnione liczby Fibonacciego.

4.1.2 Sortowanie

Z założenia do posortowania jest $N = 17$ bloków danych, po $M = 2$ liczby każdy. Wygenerowana w poprzednim punkcie tabela rozdzielająca przyjmie więc postać:

	Dyski			
krok	0	1	2	3
$k - 3$	7	X	4	6
$k - 2$	3	4	X	2
$k - 1$	1	2	2	X
k	X	1	1	1

gdyż $7 + 4 + 6 \geq 17$. Dane źródłowe powinny znajdować się na pustym dysku (w przykładzie: dysk 1). **Każdy blok przy przenoszeniu z dysku źródłowego należy posortować dowolną metodą sortowania wewnętrznego.** Blok 1. należy umieścić na dysku 0, 2. — na 2, 3. — na 3, 4. znowu na 0 itd. (oczywiście przy umieszczaniu bloków pomija się dysk źródłowy 1). Bloki należy umieszczać w ten sposób tak długo, aż łączna liczba bloków nie przekroczy wartości: 7 — dla 0 dysku, 4 — dla 2 dysku oraz 6 dla dysku 3. Ograniczenia te odczytuje się z pierwszego wiersza z tablicy rozdzielającej. Prowadzi to do następującego rozkładu bloków:

Dysk	Bloki
0	1, 4, 7, 10, 13, 15, 17
1	
2	2, 5, 8, 11
3	3, 6, 9, 12, 14, 16

W przypadku, gdyby liczba bloków była mniejsza od wymaganej, korzysta się z pustych „bloków-atrap”.

Należy podkreślić, iż liczby w opisanych tabelach określają numery scalanych bloków (z których każdy zawiera M liczb), NIE zaś sortowane dane. Kolejność bloków nie jest istotna, gdyż operacja scalania wymusi właściwy porządek danych.

Następnie następuje scalanie bloków. W pierwszym kroku scalane są bloki: 1, 2 i 3. Wynik scalania zapisywany jest na pustym dysku 1:

Dysk	Bloki
0	4, 7, 10, 13, 15, 17
1	1,2,3
2	5, 8, 11
3	6, 9, 12, 14, 16

Następnie następuje scalenie bloków 4, 5 i 6. Wynik jest dopisywany na dysk 1, jako drugi blok:

Dysk	Bloki
0	7, 10, 13, 15, 17
1	1,2,3, 4,5,6
2	8, 11
3	9, 12, 14, 16

W analogiczny sposób scala się pozostałe bloki, aż do zużycia wszystkich bloków z dysku mającego najmniejszą ich ilość:

Dysk	Bloki
0	13, 15, 17
1	1,2,3, 4,5,6, 7,8,9, 10,11,12
2	
3	14, 16

Ponieważ jeden z dysków został pusty, krok 1 kończy się. Należy zwrócić uwagę na fakt, iż liczba bloków dokładnie odpowiada 2 wierszowi tabeli rozdzielającej: Na dysku 0 znajdują się 3 bloki, na dysku 1 znajdują się 4 bloki (liczy się tylko liczba bloków, ich rozmiar jest bez znaczenia), dysk 2 jest pusty, zaś dysk 3 zawiera 2 bloki.

Krok 2:

Następnie dyskiem docelowym staje się pusty dysk 2. Następuje scalenie analogiczne do kroku 1: Blok 13 z dysku 0, blok 1,2,3 z dysku 1, oraz blok 14 z dysku 3 są scalane na dysk 2:

Dysk	Bloki
0	15, 17
1	4,5,6, 7,8,9, 10,11,12
2	13, 1,2,3, 14
3	16

Analogicznie bloki 15, 4,5,6 oraz blok 16 są scalane, zaś wynik jest dołączany na dysk 2:

Dysk	Bloki
0	17
1	7,8,9, 10,11,12
2	13, 1,2,3, 14, 15, 4,5,6, 16
3	

Krok 3:

Następuje scalanie danych na docelowy dysk 3:

Dysk	Bloki
0	
1	10,11,12
2	15, 4,5,6, 16
3	17, 7,8,9, 13,1,2,3,14

Ponieważ jeden z dysków jest pusty, krok ten kończy się.

Krok 4:

Następuje scalanie danych na docelowy dysk 0:

Dysk	Bloki
0	10,11,12, 15,4,5,6,16, 17,7,8,9,13,1,2,3,14
1	
2	
3	

Ponieważ pozostał tylko jeden blok, scalanie kończy się. Scalanie odpowiednich bloków wykonuje się identycznie, jak w poprzednio omawianych algorytmach.

Należy ponownie podkreślić, iż liczby w opisanych tabelach określają numery scalanych bloków (z których każdy zawiera M liczb), NIE zaś sortowane dane. Kolejność bloków nie jest istotna, gdyż operacja scalania wymusi właściwy porządek danych.

5 Zadania do wykonania

1. Opisać działanie wszystkich zaprezentowanych algorytmów dla ciągu liter PRZYKLADSORTOWANIA.

Literatura

- [1] L.Banachowski i in. *Algorytmy i struktury danych*; WNT, 1996.
- [2] T.H.Cormen i in. *Wprowadzenie do algorytmów*, WNT, 2000
- [3] A.Drozdek *Struktury danych w języku C*, WNT, 1996
- [4] D.E.Knuth *Sztuka programowania*, WNT, 2002
- [5] R.Sedgewick *Algorytmy w C++*, Wydawnictwo RM, 1999
- [6] P.Wróblewski, *Algorytmy, struktury danych i techniki programowania*, HELION, 1996
- [7] N. Wirth, *Algorytmy + struktury danych = programy*, WNT, 2001