**Scalanie zbiorów uporządkowanych**

Podstawową operacją algorytmu jest scalanie dwóch zbiorów uporządkowanych w jeden zbiór również uporządkowany. Operację scalania realizujemy wykorzystując pomocniczy zbiór, w którym będziemy tymczasowo odkładać scalane elementy dwóch zbiorów. Ogólna zasada jest następująca:

|  |
| --- |
| 1. Przygotuj pusty zbiór tymczasowy
2. Dopóki żaden ze scalanych zbiorów nie jest pusty, porównuj ze sobą pierwsze elementy każdego z nich i w zbiorze tymczasowym umieszczaj mniejszy z elementów usuwając go jednocześnie ze scalanego zbioru.
3. W zbiorze tymczasowym umieść zawartość tego scalanego zbioru, który zawiera jeszcze elementy.
4. Zawartość zbioru tymczasowego przepisz do zbioru wynikowego i zakończ algorytm.
 |

**Przykład:**

Połączmy za pomocą opisanego algorytmu dwa uporządkowane zbiory: {1 3 6 7 9 } z { 2 3 4 6 8 }

|  |  |  |
| --- | --- | --- |
| **Scalanezbiory** | **Zbiórtymczasowy** | **Opis wykonywanych działań** |
| **[1]** 3  6  7  9  **2**  3  4  6  8  |   | Porównujemy ze sobą najmniejsze elementy scalanych zbiorów. Ponieważ zbiory te są już uporządkowane, to najmniejszymi elementami będą zawsze ich pierwsze elementy. |
|     3  6  7  9  2  3  4  6  8  | **[1]** | W zbiorze tymczasowym umieszczamy mniejszy element, w tym przypadku będzie to liczba 1. Jednocześnie element ten zostaje usunięty z pierwszego zbioru |
|     **3**  6  7  9 **[2]** 3  4  6  8  |  **1**  | Porównujemy kolejne dwa elementy i mniejszy umieszczamy w zbiorze tymczasowym. |
|    **[3]** 6  7  9    **3** 4  6  8  |  **1[2]** | Następne porównanie i w zbiorze tymczasowym umieszczamy liczbę 3. Ponieważ są to elementy równe, to nie ma znaczenia, z którego zbioru weźmiemy element 3. |
|      **6** 7  9    **[3]** 4  6  8  |  **1 2[3**] | Teraz do zbioru tymczasowego trafi drugie 3. |
|      **6** 7  9      **[4]** 6  8  | **1 2 3[3**] | W zbiorze tymczasowym umieszczamy mniejszy z porównywanych elementów, czyli liczbę 4. |
|      **[6]** 7  9        **6** 8  | **1 2 3 3[4**] | Porównywane elementy są równe, zatem w zbiorze tymczasowym umieszczamy dowolny z nich. |
|        **7** 9        **[6]** 8  |  **1 2 3 3 4[6**] | Teraz drugą liczbę 6. |
|        **[7]** 9          **8** |  **1 2 3 3 4 6[6**] | W zbiorze tymczasowym umieszczamy liczbę 7 |
|          **9**         **[8]** |  **1 2 3 3 4 6 6[7**] | Teraz 8 |
|          **[9]** | **1 2 3 3 4 6 6 7[8]** | Drugi zbiór jest pusty. Od tego momentu już nie porównujemy, lecz wprowadzamy do zbioru tymczasowego wszystkie pozostałe elementy pierwszego zbioru, w tym przypadku będzie to liczba 9. |
|  | **1 2 3 3 4 6 6 7 8[9]** | Koniec scalania. Zbiór tymczasowy zawiera wszystkie elementy scalanych zbiorów i jest uporządkowany. Możemy w dalszej kolejności przepisać jego zawartość do zbioru docelowego. |

Z podanego przykładu możemy wyciągnąć wniosek, iż operacja scalania dwóch uporządkowanych zbiorów jest dosyć prosta. Diabeł jak zwykle tkwi w szczegółach.

**Algorytm scalania dwóch zbiorów**

Przed przystąpieniem do wyjaśniania sposobu**łączenia dwóch zbiorów uporządkowanych** w jeden zbiór również uporządkowany musimy zastanowić się nad sposobem **reprezentacji danych**. Przyjmijmy, iż elementy zbioru będą przechowywane w jednej tablicy, którą oznaczymy literką *d*. Każdy element w tej tablicy będzie posiadał swój numer, czyli **indeks** z zakresu od 1 do *n*.

Kolejnym zagadnieniem jest sposób **reprezentacji scalanych zbiorów**. W przypadku algorytmu sortowania przez scalanie zawsze będą to dwie przyległe połówki zbioru, który został przez ten algorytm podzielony. Co więcej, wynik scalenia ma być umieszczony z powrotem w tym samym zbiorze.

**Przykład:**

Prześledźmy prosty przykład. Mamy posortować zbiór o postaci: { 6 5 4 1 3 7 9 2 }

|  |  |
| --- | --- |
| **Sortowany zbiór** | **Opis wykonywanych operacji** |
| d[1] | d[2] | d[3] | d[4] | d[5] | d[6] | d[7] | d[8] |
| 6 | 5 | 4 | 1 | 3 | 7 | 9 | 2 | Zbiór wyjściowy. |
| 6 | 5 |  4 | 1 | 3 | 7 | 9 | 2 | Pierwszy podział. |
| 6 | 5 | 4 | 1 | 3 | 7 | 9 | 2 | Drugi podział |
| 6 | 5 | 4 | 1 | 3 | 7 | 9 | 2 | Trzeci podział. |
| 5 | 6 | 1 | 4 | 3 | 7 | 2 | 9 | Pierwsze scalanie. |
| 1 | 4 | 5 | 6 | 2 | 3 | 7 | 9 | Drugie scalanie. |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 9 | Trzecie scalanie. Koniec. |

Ponieważ w opisywanym tutaj algorytmie sortującym scalane podzbiory są przyległymi do siebie częściami innego zbioru, zatem logiczne będzie użycie do ich definicji indeksów wybranych elementów tych podzbiorów:

|  |  |
| --- | --- |
| *ip* | - indeks pierwszego elementu w młodszym podzbiorze |
| *is* | - indeks pierwszego elementu w starszym podzbiorze |
| *ik* | - indeks ostatniego elementu w starszym podzbiorze |

Przez podzbiór młodszy rozumiemy podzbiór zawierający elementy o indeksach mniejszych niż indeksy elementów w podzbiorze starszym.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| pozostała część zbioru | *ip* | *...* | *is* | *...* | *ik* | pozostała część zbioru |
| młodszy podzbiór | starszy podzbiór |

Indeks końcowego elementu młodszej połówki zbioru z łatwością wyliczamy - będzie on o 1 mniejszy od indeksu pierwszego elementu starszej połówki.

**Przykład:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Po pierwszym podziale prezentowanego powyżej zbioru otrzymujemy następujące wartości indeksów:

|  |  |
| --- | --- |
| Młodsza połówka | Starsza połówka |
| *ip =*1 | *is =*5 |
| *ik =*8 |

 | Po kolejnym podziale połówek otrzymujemy 4 ćwiartki dwuelementowe. Wartości indeksów będą następujące:

|  |  |
| --- | --- |
| Młodsza połówka | Starsza połówka |
| Młodszaćwiartka | Starszaćwiartka | Młodszaćwiartka | Starszaćwiartka |
| *ip =*1 | *is =*3 | *ip =*5 | *is = 7* |
| *ik =*4 | *ik =*8 |

 |

**Specyfikacja algorytmu scalania**

**Scalaj(***ip, is, ik***)**

**Dane wejściowe**

|  |  |
| --- | --- |
| d[ ] | - scalany zbiór |
| *i*p | - indeks pierwszego elementu w młodszym podzbiorze,  *i*p  N |
| *i*s | - indeks pierwszego elementu w starszym podzbiorze,  *i*s  N |
| *i*k | - indeks ostatniego elementu w starszym podzbiorze,  *i*k  N |

**Dane wyjściowe**

|  |  |
| --- | --- |
| d[ ] | - scalony zbiór |

**Zmienne pomocnicze**

|  |  |
| --- | --- |
| p[ ] | - zbiór pomocniczy, który zawiera tyle samo elementów, co zbiór d[ ]. |
| *i*1 | - indeks elementów w młodszej połówce zbioru d[ ],  *i*1  N |
| *i*2 | - indeks elementów w starszej połówce zbioru d[ ],  *i*2  N |
| *i* | - indeks elementów w zbiorze pomocniczym p[ ],  *i*  N |

**Lista kroków algorytmu scalania**

|  |  |
| --- | --- |
| K01: | *i*1 ← *i*p;   *i*2 ← *i*s;   *i* ← *i*p |
| K02: | **Dla** *i* = *i*p, *i*p + 1, ..., *i*k: **wykonuj**    **jeśli** (*i*1 = *i*s) ∨ (*i*2 ≤ *i*k  i d[*i*1] > d[*i*2]), **to**        p[*i*] ← d[*i*2];   *i*2 ← *i*2 + 1    **inaczej**        p[*i*] ← d[*i*1];   *i*1 ← *i*1 + 1 |
| K03: | **Dla** *i* = *i*p, *i*p + 1,...,*i*k: d[*i*] ← p[*i*] |
| K04: | **Zakończ** |

**Schemat blokowy algorytmu scalania**



Operacja scalania dwóch podzbiorów wymaga dodatkowej pamięci o rozmiarze równym sumie rozmiarów scalanych podzbiorów. Dla prostoty na potrzeby naszego algorytmu zarezerwujemy tablicę *p* o rozmiarze równym rozmiarowi zbioru *d*[ ]. W tablicy *p* algorytm będzie tworzył zbiór tymczasowy, który po zakończeniu scalania zostanie przepisany do zbioru *d*[ ] w miejsce dwóch scalanych podzbiorów.

Parametrami wejściowymi do algorytmu są indeksy *ip*, *is* oraz *ik*, które jednoznacznie definiują położenie dwóch podzbiorów do scalenia w obrębie tablicy *d*[ ]. Elementy tych podzbiorów będą indeksowane za pomocą zmiennych *i*1 (młodszy podzbiór od pozycji *ip*do *is -*1) oraz *i*2 (starszy podzbiór od pozycji *is* do *ik*). Na początku algorytmu przypisujemy tym zmiennym indeksy pierwszych elementów w każdym podzbiorze.

Zmienna *i* będzie zawierała indeksy elementów wstawianych do tablicy *p*[ ]. Dla ułatwienia indeksy te przebiegają wartości od *ip* do *ik*, co odpowiada obszarowi tablicy *d*[ ] zajętemu przez dwa scalane podzbiory. Na początku do zmiennej *i* wprowadzamy indeks pierwszego elementu w tym obszarze, czyli *ip*.

Wewnątrz pętli sprawdzamy, czy indeksy *i*1 i *i*2 wskazują elementy podzbiorów. Jeśli któryś z nich wyszedł poza dopuszczalny zakres, to dany podzbiór jest wyczerpany - w takim przypadku do tablicy *p* przepisujemy elementy drugiego podzbioru.

Jeśli żaden z podzbiorów nie jest wyczerpany, porównujemy kolejne elementy z tych podzbiorów wg indeksów *i*1 i *i*2. Do tablicy *p*[ ]zapisujemy zawsze mniejszy z porównywanych elementów. Zapewnia to uporządkowanie elementów w tworzonym zbiorze wynikowym. Po zapisie elementu w tablicy *p*[ ], odpowiedni indeks *i*1 lub *i*2 jest zwiększany o 1. Zwiększany jest również indeks *i*, aby kolejny zapisywany element w tablicy *p*[ ] trafił na następne wolne miejsce. Pętla jest kontynuowana aż do zapełnienia w tablicy *p*[ ] obszaru o indeksach od *ip* do *ik*.

Wtedy przechodzimy do końcowej pętli, która przepisuje ten obszar z tablicy *p*[ ] do tablicy wynikowej *d*[ ]. Scalane zbiory zostają zapisane zbiorem wynikowym, który jest posortowany rosnąco.

**Specyfikacja algorytmu sortującego**

**Sortuj\_przez\_scalanie(***ip, ik***)**

**Dane wejściowe**

|  |  |
| --- | --- |
| d[ ] | - sortowany zbiór |
| *i*p | - indeks pierwszego elementu w młodszym podzbiorze,  *i*p  N |
| *i*k | - indeks ostatniego elementu w starszym podzbiorze,  *i*k  N |

**Dane wyjściowe**

|  |  |
| --- | --- |
| d[ ] | - posortowany zbiór |

**Zmienne pomocnicze**

|  |  |
| --- | --- |
| *i*s | - indeks pierwszego elementu w starszym podzbiorze,  *i*s  N |

**Lista kroków algorytmu sortującego**

|  |  |
| --- | --- |
| K01: | *i*s ←  (*i*p + *i*k + 1) **div** 2 |
| K02: | **Jeśli** *i*s - *i*p > 1, **to** **Sortuj\_przez\_scalanie**(*i*p, *i*s - 1) |
| K03: | **Jeśli** *i*k - *i*s > 0, **to** **Sortuj\_przez\_scalanie**(*i*s, *i*k) |
| K04: | **Scalaj**(*i*p, *i*s, *i*k) |
| K05: | **Zakończ** |

**Schemat blokowy algorytmu sortującego**



Algorytm sortowania przez scalanie jest algorytmem rekurencyjnym. Wywołuje się go z zadanymi wartościami indeksów *ip* oraz *ik*. Przy pierwszym wywołaniu indeksy te powinny objąć cały zbiór *d*, zatem *ip =*1, a *ik = n*.

Najpierw algorytm wyznacza indeks *is*, który wykorzystywany jest do podziału zbioru na dwie połówki:

- młodszą o indeksach elementów od *ip* do *is -*1
- starszą o indeksach elementów od *is* do *ik*

Następnie sprawdzamy, czy dana połówka zbioru zawiera więcej niż jeden element. Jeśli tak, to rekurencyjnie sortujemy ją tym samym algorytmem.

Po posortowaniu obu połówek zbioru scalamy je za pomocą opisanej wcześniej procedury scalania podzbiorów uporządkowanych i kończymy algorytm. Zbiór jest posortowany.

W przykładowych programach procedurę scalania umieściliśmy bezpośrednio w kodzie algorytmu sortującego, aby zaoszczędzić na wywoływaniu.