Opisany w tym rozdziale algorytm sortowania kubełkowego pozwala sortować zbiory liczb całkowitych - najlepiej o dużej ilości elementów, lecz o małym zakresie wartości. Zasada działania jest następująca:

1. Określamy zakres wartości, jakie mogą przyjmować elementy sortowanego zbioru. Niech *wmin* oznacza najmniejszą wartość, a *wmax* niech oznacza wartość największą.
2. Dla każdej możliwej wartości przygotowujemy kubełek-licznik, który będzie zliczał ilość wystąpień tej wartości w sortowanym zbiorze. Liczba liczników jest równa (*wmax* - *wmin* + 1). Każdy licznik jest początkowo ustawiony na wartość zero.
3. Przeglądamy kolejno elementy zbioru od pierwszego do ostatniego. Dla każdego elementu zbioru zwiększamy o jeden zawartość licznika o numerze równym wartości elementu. Na przykład, jeśli kolejny element zbioru ma wartość 3, to zwiększamy licznik o numerze 3. W efekcie po przeglądnięciu wszystkich elementów zbioru liczniki będą zawierały ilość wystąpień każdej z możliwych wartości. Jeśli dany licznik zawiera 0, to wartość równa numerowi licznika w zbiorze nie występuje. Inaczej wartość ta występuje tyle razy, ile wynosi zawartość jej licznika.
4. Przeglądamy kolejne liczniki zapisując do zbioru wynikowego ich numery tyle razy, ile wynosi ich zawartość. Zbiór wyjściowy będzie posortowany.

**Przykład**:

Dla przykładu posortujemy opisaną wyżej metodą zbiór { 2 6 4 3 8 7 2 5 7 9 3 5 2 6 }.

Najpierw określamy zakres wartości elementów (w tym celu możemy na przykład wyszukać w zbiorze element najmniejszy i największy). U nas zakres wynosi:

|  |
| --- |
| *wmin* = 2,  *wmax* = 9 |

Potrzebujemy zatem:

|  |
| --- |
| *wmax* - *wmin* + 1 = 9 - 2 + 1 = 8 liczników. |

Liczniki ponumerujemy zgodnie z wartościami, które będą zliczały:

|  |
| --- |
| [2] [3] [4] [5] [6] [7] [8] [9] |

Na początku sortowania wszystkie liczniki mają stan zero:

|  |
| --- |
| [2:0] [3:0] [4:0] [5:0] [6:0] [7:0] [8:0] [9:0] |

Teraz przeglądamy kolejne elementy zbioru zliczając ich wystąpienia w odpowiednich licznikach:

|  |
| --- |
| { 2 6 4 3 8 7 2 5 7 9 3 5 2 6 } [2:3] [3:2] [4:1] [5:2] [6:2] [7:2] [8:1] [9:1] |

Zapis [2:3] oznacza, iż licznik numer 2 zawiera liczbę 3, a to z kolei oznacza, iż liczba 2 pojawiła się w zbiorze 3 razy. Przeglądamy kolejne liczniki począwszy od licznika o najmniejszym numerze (w przypadku sortowania malejącego przeglądanie rozpoczynamy od licznika o największym numerze) i zapisujemy do zbioru wynikowego tyle razy numer licznika, ile wynosi jego zawartość:

|  |
| --- |
| [2:3] [3:2] [4:1] [5:2] [6:2] [7:2] [8:1] [9:1] { 2 2 2 3 3 4 5 5 6 6 7 7 8 9 } |

**Uwaga, pułapka:**

Algorytm wymaga ponumerowania liczników kolejno od *wmin* do *wmax*. Niektóre języki programowania (np. C++) nie pozwalają numerować dowolnie elementów tablic - zwykle numeracja rozpoczyna się od liczby 0. W takich przypadkach musimy dokonać przeliczenia indeksu licznika, na zliczaną przezeń wartość i na odwrót. Wzory są na szczęście bardzo proste:

**Mamy wartość w i chcemy znaleźć indeks licznika, który ją zlicza:**

https://eduinf.waw.pl/inf/alg/003_sort/images/0020.p6.gif

**Mamy indeks licznika i chcemy znaleźć zliczaną wartość w:**

https://eduinf.waw.pl/inf/alg/003_sort/images/0020.p7.gif

## Opis algorytmu

### Specyfikacja problemu

#### Dane wejściowe

|  |  |
| --- | --- |
| d[ ] | - sortowany zbiór liczb całkowitych. Indeksy elementów rozpoczynają się od 1. |
| *n* | - liczba elementów w zbiorze, *n* ∈ N |
| *w*min | - minimalna wartość elementów zbioru,  *w*min ∈ C |
| *w*max | - maksymalna wartość elementów zbioru,  *w*max ∈ C |

#### Dane wyjściowe

|  |  |
| --- | --- |
| d[ ] | - posortowany zbiór liczb całkowitych. |

#### Zmienne pomocnicze

|  |  |
| --- | --- |
| *l*w[ ] | - tablica liczników wartości o indeksach od *w*min do *w*max. Każdy licznik przechowuje liczbę całkowitą |
| *i*, *j* | - zmienne licznikowe pętli, *i*,*j* ∈ C |

### Lista kroków

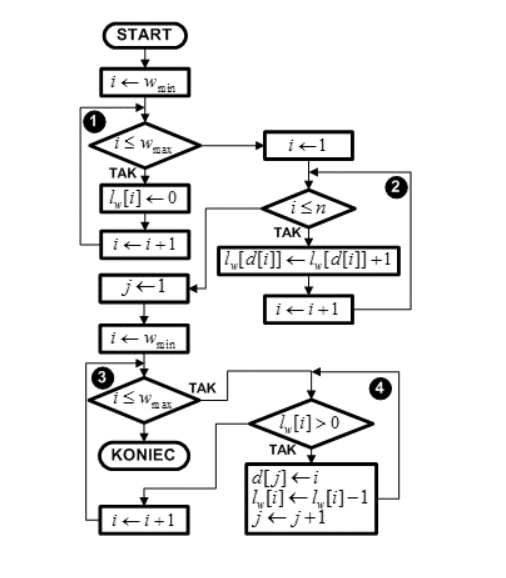
|  |  |
| --- | --- |
| K01: | **Dla** *i* = *w*min , *w*min + 1,...,*w*max:     **wykonuj:** *l*w[*i*] ← 0 |
| K02: | **Dla** *i* = 1,2,...,*n*:     **wykonuj:** *l*w[ d[*i*] ] ← *l*w[ d[*i*] ] + 1 |
| K03: | *j* ← 1 |
| K04: | **Dla** *i* = *w*min , *w*min + 1,...,*w*max:     **wykonuj krok** K05 |
| K05: | **Dopóki** *l*w[*i*] > 0: **wykonuj:**         d*[j*] ← *i*         *l*w[*i*] ← *l*w[*i*] - 1        *j* ← *j* + 1 |
| K06: | **Zakończ** |

Algorytm ma klasę czasowej złożoności obliczeniowej *O*(*m* + *n*)*,* gdzie *m* oznacza ilość możliwych wartości, które mogą przyjmować elementy zbioru, a *n* to ilość sortowanych elementów. Jeśli *m* jest małe w porównaniu z *n* (sortujemy dużo elementów o małym zakresie wartości), to na czas sortowania będzie miała wpływ głównie ilość elementów *n* i klasa złożoności uprości się do postaci *O*(*n*). Dzieje się tak dlatego, iż przy równomiernym rozkładzie dużej ilości elementów o małym zakresie wartości liczniki będą równomiernie zapełnione (stan każdego licznika będzie dążył do [n/m]). Zatem algorytm wykona:

1. *m* operacji zerowania liczników - czas pomijalnie mały i przy dużym *n* nie wpływa istotnie na klasę algorytmu.
2. *n* operacji zwiększania liczników
3. *n* operacji przesłania numerów liczników do zbioru wynikowego - ilość pustych liczników będzie dążyła do zera.

W sytuacji odwrotnej, gdy sortujemy mało elementów o dużym zakresie wartości klasa złożoności zredukuje się z kolei do *O*(*m*) - spróbuj uzasadnić ten wynik samemu na podstawie analizy działania poszczególnych fragmentów algorytmu.

### Schemat blokowy



Algorytm realizujemy w czterech pętlach.

Pętla nr 1 zeruje kolejne liczniki *lw*[ ].

W pętli nr 2 przeglądamy kolejne elementy zbioru od pierwszego do ostatniego. Dla każdego elementu zwiększamy licznik o numerze równym wartości elementu. Po zakończeniu tej pętli liczniki zawierają liczbę wystąpień poszczególnych wartości elementów w sortowanym zbiorze.

Zmienna *j* służy do umieszczania w zbiorze wyjściowym kolejnych elementów. Umieszczanie rozpoczniemy od początku zbioru, dlatego zmienne ta przyjmuje wartość 1.

Pętla nr 3 przegląda kolejne liczniki. Jeśli zawartość danego licznika jest większa od zera, to pętla nr 4 umieści w zbiorze wyjściowym odpowiednią ilość numerów licznika, które odpowiadają zliczonym wartościom elementów ze zbioru wejściowego.

Po zakończeniu pętli nr 3 elementy w zbiorze wyjściowym są posortowane. Kończymy algorytm.