Sortowanie kubełkowe

Źródło: Amy Parkes, domena publiczna.

Algorytm sortowania kubełkowego (ang. *bucket sort*) to algorytm porządkujący dane w czasie liniowym. Dane, które mają zostać posortowane, muszą spełniać pewne kryteria. Po pierwsze, powinny być równo rozłożone. Po drugie, musimy znać ich zakres. W przypadku tego algorytmu sortowanie odbywa się w miejscu.

Algorytmy, w których sortowanie odbywa się w miejscu, do posortowania zbioru danych potrzebują stałej ilości pamięci komputera, niezależnej od rozmiaru danych wejściowych. W ich przypadku wszelkie potrzebne do otrzymania wyniku obliczenia są wykonywane w pamięci, w której znajdują się dane.

Nazwa tego sortowania łączy się z jego działaniem. Zakres danych do sortowania dzielony jest na określoną liczbę kubełków. Następnie dane przypisywane są do odpowiednich kubełków i sortowane w ich ramach.

Implementację sortowania kubełkowego przedstawiamy w e-materiałach:

- Sortowanie kubełkowe w języku C++,
- Sortowanie kubełkowe w języku Java,
- Sortowanie kubełkowe w języku Python.

Więcej zadań? Sortowanie kubełkowe – zadania maturalne

Twoje cele

- Prześledzisz kolejne kroki algorytmu sortowania kubełkowego.
- Wyjaśnisz, w których sytuacjach warto zastosować sortowanie kubełkowe, a kiedy jest to rozwiązanie nieefektywne.
- Scharakteryzujesz złożoność czasową algorytmu sortowania kubełkowego.
- Zastosujesz sortowanie kubełkowe do porządkowania wybranych zbiorów danych.

Przeczytaj

Sortowanie kubełkowe należy do algorytmów, które porządkują dane w czasie liniowym. Sortowane dane powinny być liczbami całkowitymi, może być ich wiele, jednak ich zakres nie powinien być zbyt duży.

Zasada działania algorytmu – liczby całkowite

Jak sama nazwa wskazuje, w algorytmie posługujemy się kubełkami. Pracę rozpoczynamy od ustalenia, ilu ich potrzebujemy. W tym celu wyszukujemy w sortowanej tablicy największą oraz najmniejszą wartość. Oznaczamy je jako x_{min} oraz x_{max}. Wartości te definiują liczbę kubełków k. Potrzebujemy osobnego kubełka dla każdej liczby całkowitej w przedziale $[x_{min};\ x_{max}]$, co opisane jest wzorem:

$$k = x_{\max} - x_{\min} + 1$$

Atrybutem każdego kubełka jest indeks odpowiadający konkretnej liczbie z sortowanej tablicy. Rozpoczynamy od pierwszego kubełka, nadając mu indeks x_{min}, a kończymy na ostatnim, któremu przydzielimy indeks x_{max}.

Przyjmujemy, że każdy kubełek początkowo przechowuje liczbę 0, jednak docelowo wartość ta ma przedstawiać liczbę wystąpień danego składnika sortowanej tablicy. Po kolei odczytujemy liczbę powtórzeń poszczególnych elementów i aktualizujemy liczbę elementów umieszczonych w kubełkach.

Ostatnim krokiem jest zapełnianie tablicy wynikowej. Wpisujemy indeksy kolejnych kubełków do tablicy tyle razy, ile wynosi wartość danego kubełka. Rozpoczynamy od indeksu x_{min}, a kończymy na x_{max}. W rezultacie tej operacji otrzymujemy posortowaną tablicę.

> **Przykład 1**
>
> Używając algorytmu sortowania kubełkowego, posortujemy niemalejąco zbiór $\{1,\ 0,\ 1,\ 7,\ 5,\ 4,\ 4\}$. Wartości x_{min} oraz x_{max} prezentują się następująco:
>
> $$x_{\min} = 0$$
>
> $$x_{\max} = 7$$
>
> Obliczamy, ilu kubełków potrzebujemy.

$$k = 7 - 0 + 1 = 8$$

Jako oznaczenie kubełków przyjmiemy a_x. Wartość każdego z nich początkowo równa jest 0.

$$a_0 = 0, \quad a_1 = 0, \quad a_2 = 0, \quad a_3 = 0, \quad a_4 = 0, \quad a_5 = 0, \quad a_6 = 0, \quad a_7 = 0$$

Ustalamy liczbę wystąpień kolejnych składników zbioru.

Liczba 0 występuje jeden raz, zatem kubełek a_0 przyjmuje wartość 1.

Liczba 1 występuje dwa razy, zatem kubełek a_1 przyjmuje wartość 2.

Liczba 2 nie występuje ani razu, zatem kubełek a_2 przyjmuje wartość 0 itd.

W konsekwencji otrzymujemy następujące kubełki:

$$a_0 = 1, \quad a_1 = 2, \quad a_2 = 0, \quad a_3 = 0, \quad a_4 = 2, \quad a_5 = 1, \quad a_6 = 0, \quad a_7 = 1$$

Możemy teraz posortować liczby na podstawie wartości kubełków, rozpoczynając od a_0. Wartości z kubełków oznaczają, ile razy ma wystąpić liczba podana jako indeks kubełka. Wpisujemy do wynikowej tablicy odpowiednio jedną liczbę 0, dwie liczby 1 itd. Dopiero z niej wypisujemy posortowany zbiór:

$$\{0, 1, 1, 4, 4, 5, 7\}$$

Przykład 2

Wróćmy do kwestii dużej różnicy między maksymalną i minimalną wartością sortowanego zbioru. Załóżmy, że do posortowania mamy następujący zbiór: $\{1,\ 0,\ 1,\ 7,\ 5,\ 4,\ 104\}$. Wartości x_{min} oraz x_{max} prezentują się następująco:

$$x_{\min} = 0$$

$$x_{\max} = 104$$

Ustalamy, ilu kubełków potrzebujemy:

$$k = 104 - 0 + 1 = 105$$

Pomimo że mamy do posortowania tyle samo elementów co w przykładzie 1, znacząco zwiększa się liczba kubełków, które należy utworzyć. Co powoduje, że zajmujemy więcej zasobów obliczeniowych komputera.

> **Ważne!**
>
> Nie zawsze w danym kubełku przechowujemy jedną liczbę. Istnieje możliwość stworzenia kubełków zawierających elementy tablicy z wybranych przedziałów (wszystkie przedziały muszą być tej samej długości). Wówczas wewnątrz każdego z kubełków należy wykonać osobno proces sortowania. Można do tego użyć dowolnego algorytmu sortującego. W ostatnim kroku wpisujemy po kolei posortowane wartości z kubełków do tablicy.

Pseudokod

Zapiszmy algorytm sortowania kubełkowego dla liczb całkowitych za pomocą pseudokodu.

Specyfikacja problemu:

Dane:

- `n` – liczba elementów w tablicy `tab` do posortowania
- `tab[0..n-1]` – tablica liczb całkowitych zawierająca dane do posortowania

Wynik:

- `tab[0..n-1]` – tablica liczb całkowitych posortowana niemalejąco

Ustalmy teraz liczbę kubełków potrzebnych do wykonania algorytmu. Skorzystamy z przywoływanego już wzoru, a wynik zapiszemy w zmiennej `k`. Wymagane do tego będzie wskazanie wartości minimalnej oraz maksymalnej, które znajdują się w tablicy `tab`.

```
xmin ← tab[0]
xmax ← tab[0]

dla i = 1, 2, ..., n - 1 wykonuj:
    jeżeli tab[i] < xmin:
        xmin ← tab[i]
    jeżeli tab[i] > xmax
        xmax ← tab[i]

k ← xmax - xmin + 1
```

Następnie tworzymy kubełki i przypisujemy im domyślną wartość 0.

```
kubelki[0..k - 1]
```

```
dla i = 0, 1, ..., k - 1 wykonuj:
    kubelki[i] ← 0
```

Zliczamy, ile razy dana wartość występuje w wejściowej tablicy `tab`. Ustalamy, w którym kubełku należy zapisać informację o liczbie wystąpień danej liczby. Odejmiemy od każdej rozpatrywanej liczby wartość zmiennej `xmin`, co spowoduje zapisanie liczby wystąpień najmniejszej liczby w tablicy w kubełku o indeksie 0, drugiej najmniejszej do kubełka o indeksie 1 itd. Dla każdej liczby dokonujemy inkrementacji wartości znajdującej się w przypisanym do niej kubełku.

```
dla i = 0, 1, ..., n - 1 wykonuj:
    kubelki[tab[i] - xmin] ← kubelki[tab[i] - xmin] + 1
```

Wypełniamy tablicę posortowanymi liczbami. Odczytujemy, ile razy wystąpił dany element, a następnie tyle razy wypisujemy go do wynikowej tablicy.

```
indeks_tab ← 0
indeks_kubelka ← 0

dopóki indeks_tab < n wykonuj:
    jeżeli kubelki[indeks_kubelka] != 0:
        # kubełek niepusty, zapisz zliczoną wartość do tablicy wy
        tab[indeks_tab] = xmin + indeks_kubelka

        # zmniejsz o 1 liczbę wystąpień wartości
        kubelki[indeks_kubelka] ← kubelki[indeks_kubelka] - 1

        # przejdź do następnej komórki
        indeks_tab ← indeks_tab + 1

    w przeciwnym razie:
        # kubełek pusty, przejdź do następnego kubełka
        indeks_kubelka ← indeks_kubelka + 1
```

Implementacja algorytmu sortowania kubełkowego zapisana za pomocą pseudokodu prezentuje się następująco:

```
xmin ← tab[0]
xmax ← tab[0]
```

```
3
4  dla i = 1, 2, ..., n - 1 wykonuj:
5      jeżeli tab[i] < xmin:
6          xmin ← tab[i]
7      jeżeli tab[i] > xmax
8          xmax ← tab[i]
9
10 k ← xmax - xmin + 1
11 kubelki[0..k - 1]
12 dla i = 0, 1, ..., k - 1 wykonuj:
13     kubelki[i] ← 0
14
15 dla i = 0, 1, ..., n - 1 wykonuj:
16     kubelki[tab[i] - xmin] ← kubelki[tab[i] - xmin] + 1
17
18 indeks_tab ← 0
19 indeks_kubelka ← 0
20
21 dopóki indeks_tab < n wykonuj:
22     jeżeli kubelki[indeks_kubelka] != 0:
23         # kubełek niepusty, zapisz zliczoną wartość do tablicy wy
24         tab[indeks_tab] = xmin + indeks_kubelka
25
26         # zmniejsz o 1 liczbę wystąpień wartości
27         kubelki[indeks_kubelka] ← kubelki[indeks_kubelka] - 1
28
29         # przejdź do następnej komórki
30         indeks_tab ← indeks_tab + 1
31
32     w przeciwnym razie:
33         # kubełek pusty, przejdź do następnego kubełka
34         indeks_kubelka ← indeks_kubelka + 1
```

Złożoność czasowa algorytmu

Ważną cechą każdego algorytmu jest jego złożoność czasowa, która pozwala porównywać między sobą różne metody obliczeniowe. Aby ją oszacować, nie mierzy się czasu wykonywania algorytmu, który zależny jest od implementacji i sprzętu, lecz wyróżnia się jakąś operację dominującą. Liczba wykonanych operacji dominujących zależy od rozmiaru wejścia, można więc powiedzieć, że złożoność czasowa to pewna funkcja danych

wejściowych. Rząd wielkości tej funkcji wyrażającej liczbę operacji dominujących zapisuje się za pomocą notacji dużego O.

W przypadku przedstawionego wcześniej algorytmu sortowania kubełkowego rozmiarem wejścia będzie liczba elementów w tablicy wejściowej, którą oznaczymy jako n, oraz liczba kubełków – k.

Operacje dominujące w przedstawionym algorytmie:

1. Podczas wyszukiwania elementu minimalnego i maksymalnego operacją dominującą jest porównanie, trzeba wykonać $2 \cdot n - 2$ operacji.
2. Podczas przygotowywania kubełków operacją dominującą jest przypisanie, trzeba wykonać k takich operacji.
3. Podobnie zliczenie wystąpień również wymaga n operacji przypisania.
4. Przepisanie liczb do tablicy wynikowej wymaga sprawdzenia (operacja porównania) k kubełków i wykonania n przypisań, daje to: $n + k$ operacji.

Funkcję wyrażającą liczbę operacji dominujących możemy więc zapisać jako $f(n) = 4 \cdot n + 2 \cdot k - 2$. Ponieważ przy zapisie oszacowania pomijamy stałe, ostatecznie mamy do czynienia ze złożonością liniową $O(n + k)$.

Jeżeli elementów będzie dużo, ale o małym zakresie wartości (n znacznie większe niż k), to algorytm będzie działał w czasie liniowym ze względu na liczbę elementów w tablicy `tab`: $O(n)$.

Natomiast w przypadku odwrotnym, czyli gdy będzie do posortowania mało elementów, ale z dużego zakresu wartości, algorytm będzie działał w czasie liniowym ze względu na zakres wartości: $O(k) = O(x_{max} - x_{min} + 1)$.

Zasada działania algorytmu – liczby rzeczywiste

> **Polecenie 1**
>
> Zapoznaj się z algorytmem sortowania kubełkowego liczb rzeczywistych z przedziału $< 0;\ 1)$. Przeanalizuj kolejne jego kroki, a następnie zastanów się nad różnicami pomiędzy sortowaniem liczb całkowitych i zmiennoprzecinkowych.

Mamy do posortowania zbiór liczb rzeczywistych z przedziału $< 0;\ 1)$. Prezentuje

1

się on następująco:
$\{0,38;\ 0,04;\ 0,31;\ 0,24;\ 0,65;\ 0,71;\ 0,20;\ 0,82\ \ 0,33\}$
.

2

Tak jak w przypadku liczb całkowitych, przygotowujemy odpowiednią liczbę kubełków. Tworzymy ich 10 i nadajemy indeksy od 0 do 9. Kubełki nie będą jednak przechowywały liczby wystąpień danej wartości w zbiorze, a same elementy tego zbioru.

3

Przeanalizujmy sposób przydzielania elementów do odpowiednich kubełków. Tak jak w przypadku liczb całkowitych kluczową rolę odgrywają tu indeksy, jednak w tym wypadku proces jest bardziej złożony. Mnożymy razy 10 liczby ze zbioru. W ten sposób otrzymujemy wartości z przedziału $< 0;\ 10)$. W kolejnym kroku porównujemy indeksy kubełków z częścią całkowitą danej liczby (najlepiej poprzez wykonanie na porównywanej liczbie operacji podłogi, czyli wyliczenia największej liczby całkowitej nie większej od zadanej liczby). Jeżeli obie wartości są równe, element trafia do rozpatrywanego kubełka.

4

$0.38 \cdot 10 = 3.8$ $0.31 \cdot 10 = 3.1$ $0.33 \cdot 10 = 3.3$

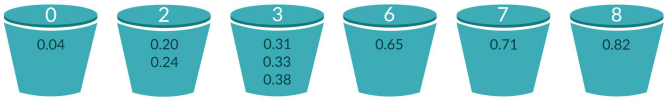
$[0.38] = 3$ $[0.31] = 3$ $[0.33] = 3$

Wykonujemy operację z poprzedniego kroku dla każdego elementu. Na rysunku zaprezentowano wartości umieszczane w kubełku o indeksie 3.

5

Gdy wypełnimy już wszystkie kubełki, pozostaje kwestia sortowania. Zanim uporządkujemy cały zbiór, sortujemy wartości w pojedynczych kubełkach. Możemy to zrobić, używając dowolnej metody, w tym również sortowania kubełkowego. W omawianym wypadku zajmujemy się jedynie liczbami z kubełków o indeksach 2 i 3. Gdy uporządkujemy wszystkie wartości, możemy przejść do ostatniego kroku.

6



{ 0.04, 0.20, 0.24, 0.31, 0.33, 0.38, 0.65, 0.71, 0.82 }

Wypisujemy wartości z kubełków do zbioru. Rozpoczynamy od kubełka z indeksem 0, a kończymy na tym z indeksem 9. Wartości odczytujemy w takiej kolejności, w jakiej zostały posortowane w kubełku. W rezultacie otrzymujemy gotowy, uporządkowany zbiór.

Pseudokod

Zapiszmy algorytm sortowania kubełkowego dla liczb rzeczywistych za pomocą pseudokodu.

Specyfikacja problemu:

Dane:

- `n` – liczba elementów w tablicy `tab` do posortowania
- `tab[0..n-1]` – tablica liczb rzeczywistych z przedziału $\langle 0; 1)$ zawierająca dane do posortowania

Wynik:

- `tab[0..n-1]` – tablica liczb rzeczywistych posortowana niemalejąco

Przyjmujemy także, że kubełków jest 10. Każdy z nich będzie listą (tablicą z możliwością dowolnego jej rozszerzania) i zawierał będzie liczby z przedziału $\langle x; x + 0{,}1)$ dla $x = 0; 0{,}1; \ldots; 0{,}9$. Tworzymy zatem te kubełki.

```
kubelki[0..9]
dla i = 0, 1, ..., 9 wykonuj:
    kubelki[i] ← nowa, pusta lista
```

Następnie każdą liczbę z sortowanej tablicy `tab` umieszczamy w odpowiadającym jej kubełku.

```
dla i = 0, 1, ..., n - 1 wykonuj:
    indeks ← tab[i] * 10
    zaokrąglij w dół zawartość zmiennej indeks
    dodaj tab[i] do kubełka kubelki[indeks]
```

Jako algorytm pomocniczy implementujemy sortowanie bąbelkowe.

```
funkcja sortowanieBabelkowe(tab)
    n ← długość(tab)
    dla i = 0, 1, ..., n - 1 wykonuj:
        dla j = 1, 2, ..., n - 1 - i wykonuj:
            jeżeli tab[j - 1] > tab[j]:
                zamień tab[j - 1] z tab[j]
```

Wykorzystując sortowanie bąbelkowe, sortujemy każdy kubełek osobno.

```
dla i = 0, 1, ..., 9 wykonuj:
    sortowanieBabelkowe(kubelki[i])
```

Na koniec przepisujemy liczby z kolejnych kubełków do tablicy wynikowej.

```
j ← 0
dla i = 0, 1, ..., 9 wykonuj:
    dla k = 0, 1, ..., długość(kubelki[i]) - 1 wykonuj:
        tab[j] ← kubelki[i][k]
        j ← j + 1
```

Cały algorytm zapisany za pomocą pseudokodu prezentuje się następująco:

```
funkcja sortowanieBabelkowe(tab)
    n ← długość(tab)
    dla i = 0, 1, ..., n - 1 wykonuj:
        dla j = 1, 2, ..., n - 1 - i wykonuj:
            jeżeli tab[j - 1] > tab[j]:
                zamień tab[j - 1] z tab[j]

kubelki[0..9]
dla i = 0, 1, ..., 9 wykonuj:
    kubelki[i] ← nowa, pusta lista

dla i = 0, 1, ..., n - 1 wykonuj:
    indeks ← tab[i] * 10
    zaokrąglij w dół zawartość zmiennej indeks
```

```
    dodaj tab[i] do kubełka kubelki[indeks]

dla i = 0, 1, ..., 9 wykonuj:
    sortowanieBabelkowe(kubelki[i])

j ← 0
dla i = 0, 1, ..., 9 wykonuj:
    dla k = 0, 1, ..., długość(kubelki[i]) - 1 wykonuj:
        tab[j] ← kubelki[i][k]
        j ← j + 1
```

Słownik

> **kubełek**
>
> podprzedział wyodrębniony z sortowanego zbioru danych
> **sortowanie**
>
> czynność polegająca na uporządkowaniu danych w zbiorze względem danego kryterium
> **złożoność czasowa**
>
> ilość czasu potrzebnego do wykonania zadania, wyrażona jako funkcja ilości danych
> **złożoność obliczeniowa**
>
> ilość zasobów komputerowych potrzebnych do wykonania zadania

Schemat interaktywny

> **Polecenie 1**
>
> Utwórz schemat interaktywny, realizujący zliczanie liczby elementów w kubełkach dla nieposortowanego zbioru {2, 4, 2, 0, 1, 2}.

Polecenie 2

Zapisz algorytm realizujący zliczanie liczby elementów w kubełkach dla nieposortowanego zbioru {2, 4, 2, 0, 1, 2}, wykorzystując wybrany język programowania.

```
1
```

```
1
```

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Zaznacz wszystkie poprawne odpowiedzi. Wskaż, co mogą przechowywać kubełki używane w sortowaniu kubełkowym.

- [ ] Gotową posortowaną tablicę.
- [ ] Dwie wartości z tablicy przekazane do porównania.
- [ ] Informację o liczbie wystąpień danej wartości w tablicy.
- [ ] Elementy sortowanej tablicy.

Ćwiczenie 2

Dokończ zdanie.
Do posortowania zawartości kubełka możemy pomocniczo użyć...

- ( ) sortowania kubełkowego lub innego algorytmu sortującego.
- ( ) dowolnego algorytmu sortującego innego niż sortowanie kubełkowe.
- ( ) wyłącznie sortowania kubełkowego.

Ćwiczenie 3

Zaznacz poprawną odpowiedź. Wskaż, ilu potrzeba kubełków do posortowania zbioru {0, 4, 2, 1, 3, 3, 4}, wykorzystując opisaną w e-materiale metodę dla liczb całkowitych.

- ◯ 7
- ◯ 8
- ◯ 4
- ◯ 5

Ćwiczenie 4

Połącz wybrane kubełki (numer podany obok kubełka to jego indeks) z odpowiadającymi im wartościami, zakładając że sortować będziemy zbiór {0, 2, 4, 1, 2, 2, 1, 4, 1, 2}.

Kubełek 3	1
Kubełek 1	4
Kubełek 0	2
Kubełek 2	0
Kubełek 4	3

Ćwiczenie 5

Połącz wybrane kubełki (numer podany obok kubełka to jego indeks) z odpowiadającymi im wartościami, zakładając że sortować będziemy zbiór {0.10, 0.21, 0.01, 0.21, 0.28, 0.23, 0.36, 0.06, 0.18, 0.22, 0.15, 0.19}.

Kubełek 0	2
Kubełek 1	4
Kubełek 2	5
Kubełek 3	1

Ćwiczenie 6

Zdanie prawdziwe zaznacz na zielono, a fałszywe na czerwono.

	prawda	fałsz
Sortowanie kubełkowe można wykorzystać zarówno do zbiorów liczb całkowitych, jak i zmiennoprzecinkowych.		
Sortowanie bardzo dużych zbiorów z użyciem sortowania kubełkowego jest niemożliwe.		
Złożoność obliczeniowa sortowania kubełkowego zawsze wyliczana jest z tego samego wzoru.		
Kubełki zawsze przechowują tylko jeden element.		

Ćwiczenie 7

Wskaż, jaka będzie największa liczba w sortowanym zbiorze, wiedząc, że używamy metody dla liczb całkowitych, najmniejszą wartością w zbiorze jest 3, a liczba kubełków wynosi 14.

- ◯ 15
- ◯ 16
- ◯ 14
- ◯ 17

Ćwiczenie 8

Uzupełnij tekst.

W sytuacji, gdy pomiędzy największą i najmniejszą liczbą w sortowanym zbiorze jest spora różnica oraz zawiera on mało elementów, wówczas sortowanie kubełkowe [ ]. Liczba kubełków [ ], dlatego [ ].

zajęty zostanie spory obszar pamięci | jest efektywnym rozwiązaniem | zależy tylko od liczby elementów w zbiorze | nie jest efektywnym rozwiązaniem | pamięć nie zostanie mocno obciążona | będzie mała | będzie duża

Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Sortowanie kubełkowe

Grupa docelowa:

Liceum ogólnokształcące i technikum, liceum ogólnokształcące, technikum, zakres rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

I. Rozumienie, analizowanie i rozwiązywanie problemów na bazie logicznego i abstrakcyjnego myślenia, myślenia algorytmicznego i sposobów reprezentowania informacji.

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

Treści nauczania – wymagania szczegółowe

I. Rozumienie, analizowanie i rozwiązywanie problemów.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

3) objaśnia dobrany algorytm, uzasadnia poprawność rozwiązania na wybranych przykładach danych i ocenia jego efektywność;

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

3) sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

I + II. Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

1) zapisuje za pomocą listy kroków, schematu blokowego lub pseudokodu, i implementuje w wybranym języku programowania, algorytmy poznane na wcześniejszych etapach oraz algorytmy:

d) jednoczesnego wyszukiwania elementu najmniejszego i największego,

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Prześledzisz kolejne kroki algorytmu sortowania kubełkowego.
- Wyjaśnisz, w których sytuacjach warto zastosować sortowanie kubełkowe, a kiedy jest to rozwiązanie nieefektywne.
- Scharakteryzujesz złożoność czasową algorytmu sortowania kubełkowego.
- Zastosujesz sortowanie kubełkowe do porządkowania wybranych zbiorów danych.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- ćwiczenia praktyczne.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda.

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Sortowanie kubełkowe". Nauczyciel prosi uczniów o zapoznanie się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. Prowadzący wyświetla na tablicy interaktywnej zawartość sekcji „Wprowadzenie" i omawia cele do osiągnięcia w trakcie lekcji.
2. **Rozpoznanie wiedzy uczniów.** Uczniowie tworzą pytania dotyczące tematu zajęć, na które odpowiedzą w trakcie lekcji.

Faza realizacyjna:

1. **Praca z tekstem.** Nauczyciel ocenia, na podstawie informacji na platformie, stan przygotowania uczniów do zajęć. Jeżeli jest ono niewystarczające, prosi wybraną osobę o przedstawienie najważniejszych informacji z sekcji „Przeczytaj". W kolejnym kroku uczniowie analizują omówione w sekcji przykłady i implementują przedstawione za pomocą pseudokodu algorytmy w wybranych językach programowania.
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Schemat interaktywny". Wybrana osoba czyta treść polecenia 1. Uczniowie indywidualnie opracowują algorytmy. W kolejnym kroku nauczyciel sprawdza poprawność wykonania zadania. Następnie wykonują polecenie 2 z tej sekcji.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują indywidualnie ćwiczenie nr 1, a następnie porównują swoje odpowiedzi z kolegą lub koleżanką.
4. W kolejnym etapie uczniowie dobierają się w pary i wykonują ćwiczenia nr 2-6. Następnie konsultują swoje rozwiązania z inną parą uczniów i ustalają jedną wersję odpowiedzi.

Faza podsumowująca:

1. Nauczyciel zadaje pytania podsumowujące, np.
   – jakie są zasady działania sortowania kubełkowego?
   – jakie są jego wady i zalety?
   – jak porządkować liczby z użyciem algorytmu sortowania kubełkowego?
   – w których sytuacjach warto oraz nie warto zastosować sortowanie kubełkowe?
   – co oznacza pojęcie złożoność obliczeniowa?
2. Nauczyciel wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W kontekście ich realizacji podsumowuje przebieg zajęć, a także wskazuje mocne i słabe strony pracy uczniów.

Praca domowa:

1. Uczniowie wykonują ćwiczenia 7-8 z sekcji „Sprawdź się".

Wskazówki metodyczne:

- Uczniowie mogą wykorzystać treści w sekcjach: „Przeczytaj”, „Schemat interaktywny”, „Sprawdź się” jako materiał do lekcji powtórkowej.