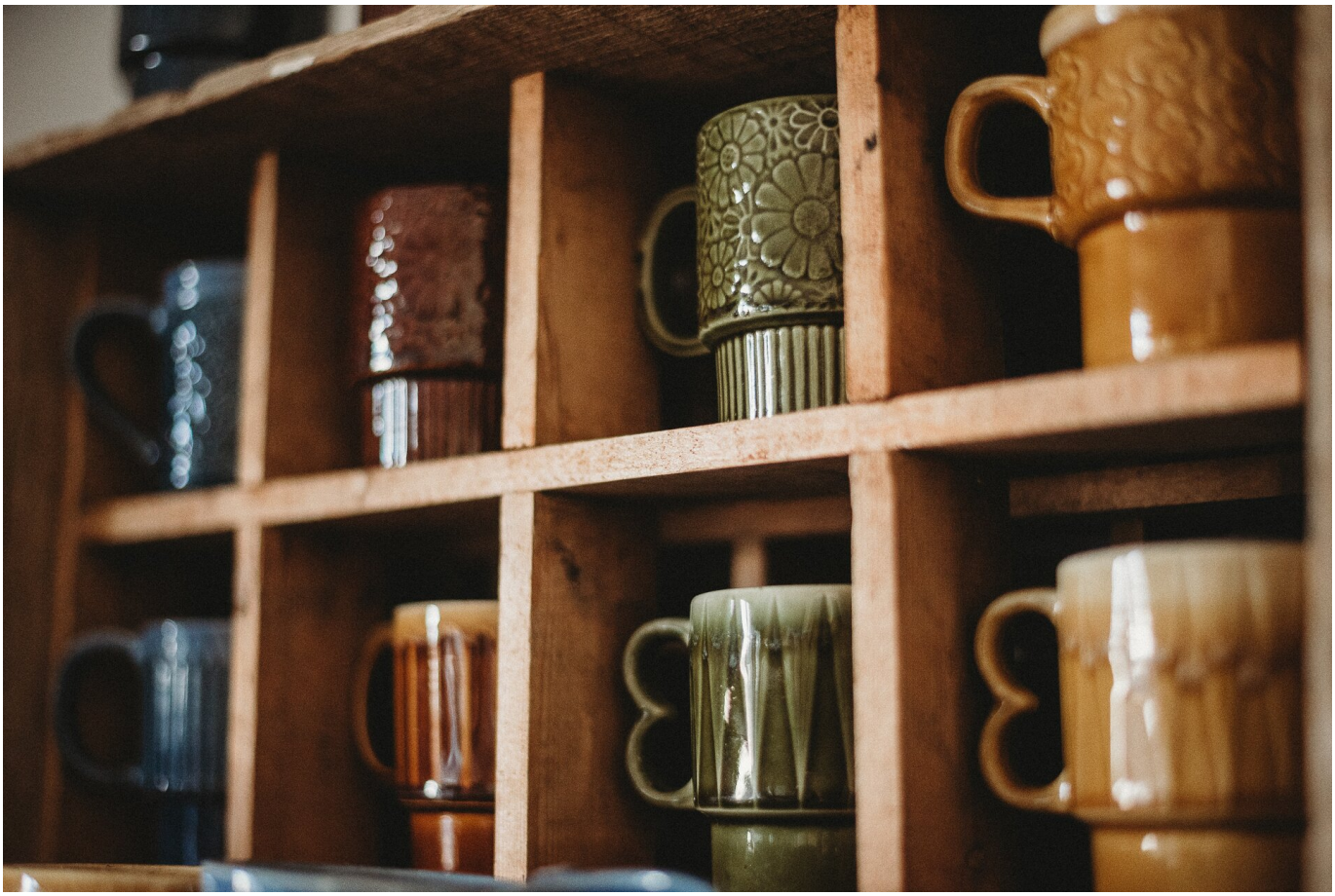
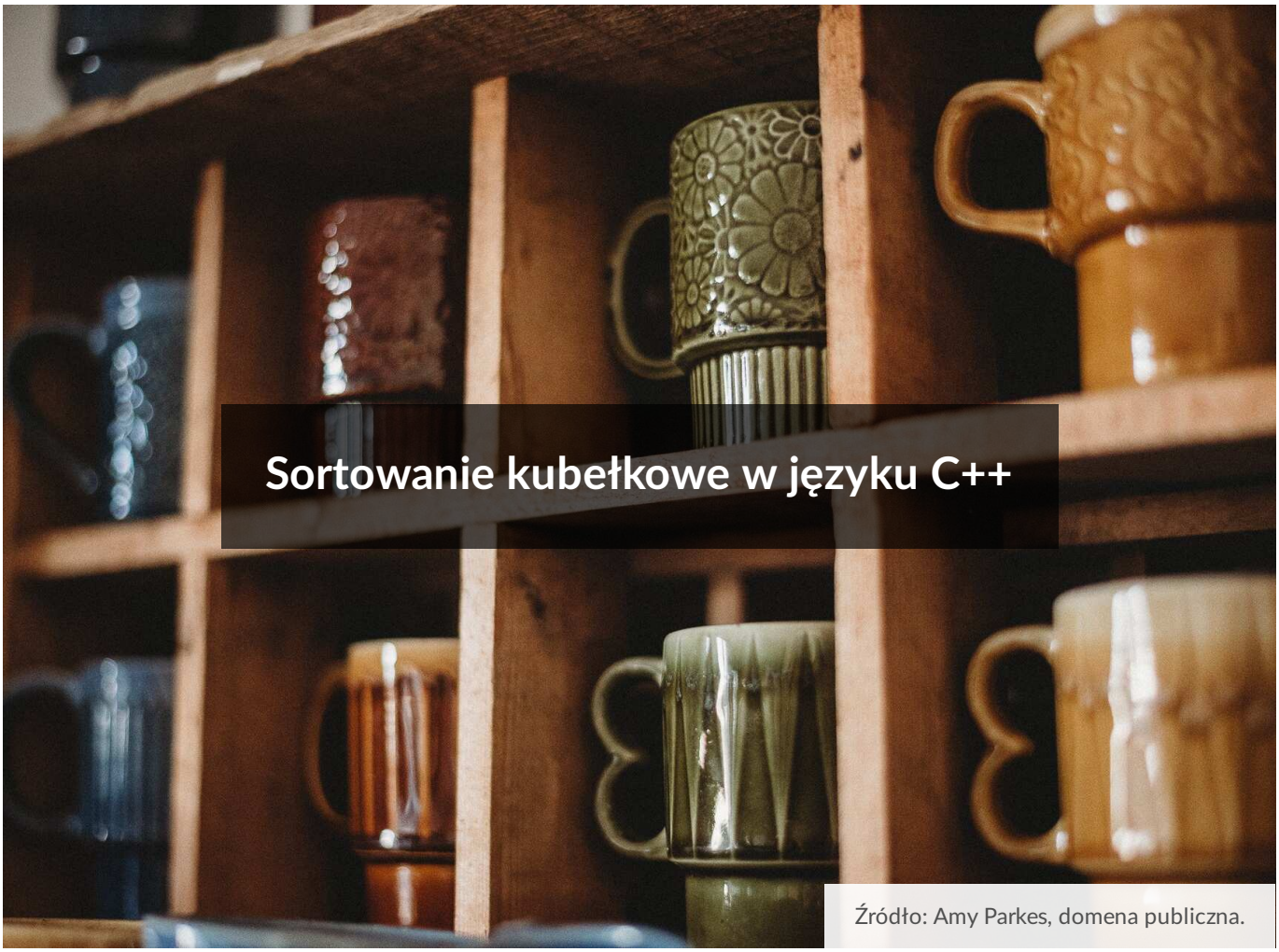


Sortowanie kubełkowe w języku C++



Źródło: Amy Parkes, domena publiczna.

Sortowanie kubełkowe jest jednym z najbardziej intuicyjnych sortowań, często używamy go w życiu codziennym. Polega ono na wkładaniu elementów zbioru do odpowiednich kubełków (które można porównać do szufladek lub pudełek), przy założeniu, że w poszczególnych pojemnikach znajdują się elementy o tej samej wartości. W tym e-materiale dowiesz się, jak zaimplementować algorytm sortowania kubełkowego w języku programowania C++.

Więcej informacji o sortowaniu kubełkowym znajdziesz w e-materiale Sortowanie kubełkowe.

Implementację sortowania kubełkowego w innych językach przedstawiamy w e-materiałach:

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

Więcej zadań? Zajrzyj do e-materiału Sortowanie kubełkowe – zadania maturalne.

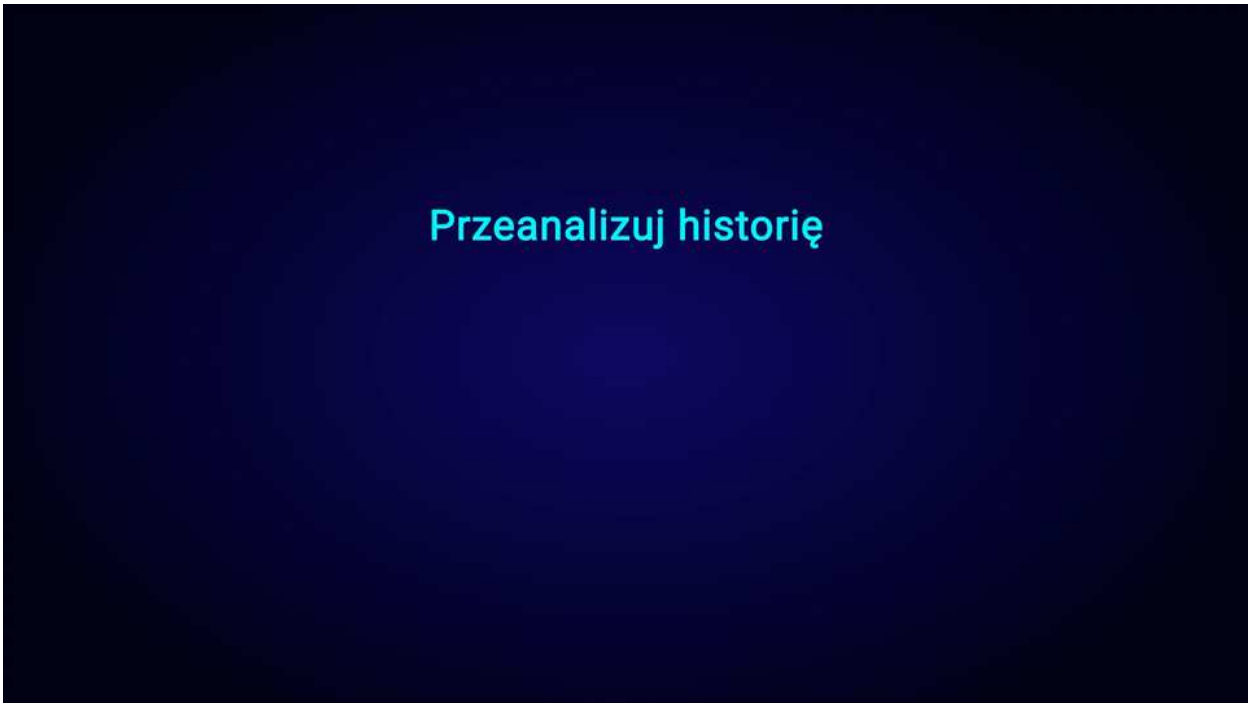
> **Twoje cele**
>
> - Zaimplementujesz algorytm sortowania kubełkowego w języku C++.

- Przeanalizujesz program realizujący w języku C++ algorytm sortowania kubełkowego liczb należących do zakresu podanego przez użytkownika.
- Rozwiążesz zadania wymagające znajomości algorytmu sortowania kubełkowego.

Film samouczek

Polecenie 1

Zapoznaj się z animacją przedstawiającą sortowanie kubełkowe.



Film dostępny pod adresem /preview/resource/Rcz1Xt1KeUHUU

Film mówiący o sortowaniu kubełkowym.

Polecenie 2

Prześledź działanie algorytmu w praktyce. W tym celu przygotuj zestaw przedmiotów do posortowania. Mogą to być żetony, korki, kapsle czy inne małe przedmioty, mające charakterystyczną cechę, według której można je posortować.

Następnie ułóż je w losowej kolejności w jednej linii. Powstanie w ten sposób zbiór, który posortujesz.

Wszystkie ułożone w linii przedmioty pogrupuj w zbiory. Elementy można zidentyfikować poprzez ich charakterystyczną cechę. W przykładzie przedstawionym w animacji był to rodzaj przyboru szkolnego. W twoim przypadku może być to kolor, rozmiar czy inne dowolne kryterium.

Następnie wybierz zbiór i zacznij układać jego elementy, jeden po drugim, w linii. Gdy dany zbiór się skończy, wybierz następny i powtarzaj proces. Przedłużaj układaną linię aż do momentu, gdy skończą się wszystkie zbiory.

Polecenie 3

Napisz program sortujący w kolejności niemalejącej dane, za pomocą metody kubełkowej. Przetestuj jego działanie dla wyników ankiety, dotyczącej opinii o traktacie lizbońskim, przeprowadzonej w lipcu 2008 roku, po referendum w Irlandii.

Ankietowanym zadano pytanie: Jaka jest pana/pani opinia o traktacie lizbońskim po referendum w Irlandii?

Wyniki ankiety:

Raczej dobra – 18 %

Zdecydowanie dobra – 3 %

Trudno jednoznacznie powiedzieć – 29%

Zdecydowanie zła – 21%

Raczej zła – 29%

Specyfikacja problemu:

Dane:

- `n` – liczba naturalna; liczba elementów tablicy `dane`
- `dane` – `n`-elementowa tablica zawierająca liczby naturalne z przedziału [0, 100]; tablica liczb do posortowania
- `wartoscMin` – liczba naturalna z przedziału [0, 100]; najmniejsza z wartości do posortowania odczytana z tablicy `dane`
- `wartoscMax` – liczba naturalna z przedziału [0, 100]; największa z wartości do posortowania odczytana z tablicy `dane`

Wynik:

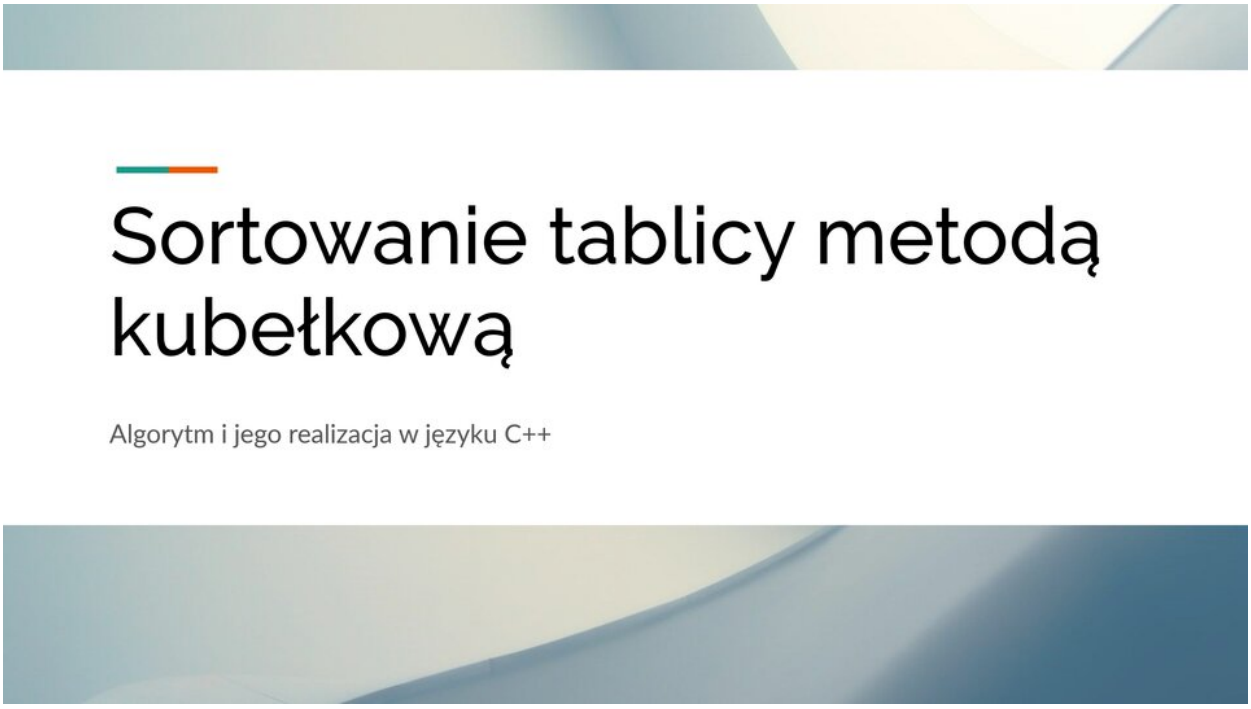
- `dane` – posortowana niemalejąco tablica liczb naturalnych z przedziału [0, 100]

```
#include <iostream>

using namespace std;

int main () {

    // Tutaj dodaj kod. Żeby coś wypisać, użyj polecenia: cout

return 0;

}
```

```

```

Polecenie 4

Porównaj swoje rozwiązanie z przedstawionym w filmie.



Film dostępny pod adresem /preview/resource/RqIff499iVOh9

Film nawiązujący do treści materiału: Sortowanie tablicy metodą kubełkową.

Plik o rozmiarze 709.00 B w języku polskim

Polecenie 5

Dodaj do swojego programu komentarze tak, żeby był zrozumiały dla osoby, która nie potrafi programować.

Przeczytaj

Sortowanie kubełkowe

Algorytm sortowania kubełkowego polega na zliczeniu wystąpień danych wartości w tablicy wejściowej, a następnie na wygenerowaniu tablicy wyjściowej, zawierającej dane w odpowiednich kubełkach, posortowane zgodnie z ich liczbą.

Sortowanie liczb całkowitych

Algorytm pozwala sortować liczby całkowite.

Specyfikacja problemu:

Dane:

- `n` – liczba naturalna; liczba elementów tablicy `tabWejsciowa`
- `tabWejsciowa` – n-elementowa tablica, przechowująca liczby całkowite

Wynik:

- `tabPosortowana` – posortowana niemalejąco tablica liczb całkowitych

W algorytmie pod indeksem 0 w `tabZliczajaca` znajduje się kubełek przeznaczony dla liczb o najmniejszej wartości ze zbioru wejściowego. Natomiast ostatni element tablicy zliczającej staje się kubełkiem do zliczania wartości maksymalnej.

Zaczynamy od znalezienia wartości minimalnej i maksymalnej wśród sortowanych liczb. Wykorzystamy poznany już algorytm, który został zaprezentowany w e-materiale Sortowanie kubełkowe.

```
// znajdowanie wartości minimalnej i maksymalnej
int minLiczba = tabWejsciowa[0];
int maxLiczba = tabWejsciowa[0];

for(int i = 1; i < n; i++)
    {
    if(tabWejsciowa[i] < minLiczba)
        minLiczba = tabWejsciowa[i];
    else if(tabWejsciowa[i] > maxLiczba)
        maxLiczba = tabWejsciowa[i];
    }
```

Kolejnym krokiem jest utworzenie tablicy zliczeń. Rozmiar tablicy zliczającej zapiszemy w zmiennej `k`. Wyznaczamy go jako różnicę wartości maksymalnej i minimalnej powiększonej o 1: `k = maxLiczba - minLiczba + 1`. Następnie zerujemy wszystkie jej elementy.

```
1    // przygotowanie tablicy zliczającej
2    int k = maxLiczba - minLiczba + 1;
3    int tabZliczajaca[k] = {0};
```

> **Przykład 1**
>
> Na podstawie kilku zbiorów możemy sprawdzić, czy podany wzór na rozmiar tablicy jest poprawny.
>
> **Dla zbioru:**
>
> - {0, 5, 4} – zbiór z liczbami nieujemnymi
> - `max`: 5, `min`: 0
> - wymagane kubełki: {0, 1, 2, 3, 4, 5}
>
> obliczymy zgodnie z podanym wcześniej wzorem rozmiar tablicy:
>
> $$5 - 0 + 1 = 6$$
>
> Rzeczywiście liczba wymaganych kubełków zgadza się z otrzymaną długością tablicy.
>
> **Dla zbioru:**
>
> - {-3, -1} – zbiór z liczbami ujemnymi
> - `max`: -1, `min`: -3
> - wymagane kubełki: {-3, -2, -1}
>
> obliczamy zgodnie z wcześniej podanym wzorem długość tablicy:
>
> $$-1 - (-3) + 1 = 3$$
>
> Ponownie otrzymaliśmy poprawną długość tablicy, która zgadza się z liczbą wymaganych kubełków.
>
> **Dla zbioru:**
>
> - {0, -1, 2} – zbiór z liczbami dodatnimi i ujemnymi
> - `max`: 2, `min`: -1
> - wymagane kubełki: {-1, 0, 1, 2}

obliczamy zgodnie z podanym wcześniej wzorem długość tablicy:

$$2 - (-1) + 1 = 4$$

Po raz kolejny liczba wymaganych kubełków zgadza się z otrzymaną długością tablicy.

Przeglądamy wartości z tablicy wejściowej i zapisujemy ich wystąpienia w tablicy zliczeń. Indeks licznika dla danej wartości wyliczamy jako różnicę tej wartości i wartości minimalnej.

```
for(int i = 0; i < n; i++)
{
    int wartosc = tabWejsciowa[i];
    tabZliczajaca[wartosc - minLiczba]++;
}
```

Przykład 2

Zbiór posiada elementy: {-5, 0, 2}, sprawdźmy, pod jakim indeksem będą one zliczane.

- `minLiczba` = -5
- `maxLiczba` = 2
- `k = 2 - (-5) + 1 = 8`
- `tabZliczajaca[k] = {0, 0, 0, 0, 0, 0, 0, 0}`

Indeks dla wartości -5 obliczamy w następujący sposób: -5 - (-5) = 0. Najmniejsza wartość będzie zliczana w komórce o indeksie 0.

Indeks dla wartości 2 wynosi: 2 - (-5) = 7. Największa wartość będzie zliczana w komórce o indeksie 7.

Następnie do tablicy wyjściowej dopisujemy odpowiednią liczbę wystąpień zliczonych wartości. Do tablicy zapisujemy indeks powiększony o wartość `minLiczba`. Jest to operacja odwrotna do wcześniejszej, kiedy indeks w tablicy zliczeń wyliczaliśmy jako pomniejszenie zliczanej wartości o wartość `minLiczba`.

```
    int tabPosortowana[n];
    int j = 0;
    for(int i = 0; i < k; i++)
    {
        while(tabZliczajaca[i] > 0)
        {
            tabPosortowana[j] = i + minLiczba;
```

```
            tabZliczajaca[i]--;
            j++;
        }
    }
```

> **Już wiesz**
>
> Ze względu na numerowanie komórek tablicy zliczającej od 0:
>
> - indeks wartości zliczanej ustalamy, korzystając ze wzoru: `indeks = wartość - wartość_minimalna`,
> - zliczaną wartość wyznaczamy zgodnie ze wzorem: `wartość = indeks + wartość_minimalna`.

Cały kod prezentuje się następująco:

```
#include <iostream>
#include <climits>

using namespace std;

int main()
{
    int tabWejsciowa[] = {3, 2, 3, 4, -4, 2, 3, 2, 0, -5};
    int n = 10;

    // znajdowanie wartości minimalnej i maksymalnej
    int minLiczba = tabWejsciowa[0];
    int maxLiczba = tabWejsciowa[0];

    for(int i = 1; i < n; i++)
        {
        if(tabWejsciowa[i] < minLiczba)
            minLiczba = tabWejsciowa[i];
        else if(tabWejsciowa[i] > maxLiczba)
            maxLiczba = tabWejsciowa[i];
        }

    // przygotowanie tablicy zliczającej
    int k = maxLiczba - minLiczba + 1;
    int tabZliczajaca[k] = {0};

```

```
27      // zliczanie wartości
28      for(int i = 0; i < n; i++)
29      {
30          int wartosc = tabWejsciowa[i];
31          tabZliczajaca[wartosc - minLiczba]++;
32      }
33
34      // porządkowanie
35      int tabPosortowana[n];
36      int j = 0;
37      for(int i = 0; i < k; i++)
38      {
39          while(tabZliczajaca[i] > 0)
40          {
41              tabPosortowana[j] = i + minLiczba;
42              tabZliczajaca[i]--;
43              j++;
44          }
45      }
46
47      // wypisanie uporządkowanych wartości
48      for(int i = 0; i<n; i++)
49      {
50          cout << tabPosortowana[i] << " ";
51      }
52
53  return 0;
54  }
```

Wynikiem algorytmu jest posortowana tablica: `{-5 -4 0 2 2 2 3 3 3 4}`.

> **Ważne!**
>
> Korzystamy z `maxLiczba - minLiczba + 1` komórek pamięci.
>
> Algorytm ma więc złożoność pamięciową O(`max - min + 1`).

Sortowanie liczb rzeczywistych

Specyfikacja problemu:

Dane:

- `n` – liczba naturalna; liczba elementów tablicy `tabWejsciowa`
- `tabWejsciowa` – n-elementowa tablica, przechowująca liczby rzeczywiste

Wynik:

- `tabWejsciowa` – posortowana niemalejąco tablica liczb rzeczywistych

Liczby rzeczywiste również mogą być porządkowane z użyciem algorytmu sortowania kubełkowego. Będzie się on jednak różnił od dotychczas przedstawionego.

W przypadku algorytmu dla liczb całkowitych każda liczba ma własny osobny kubełek, który zawiera licznik jej wystąpień. Natomiast podczas sortowania liczb rzeczywistych kubełki są przedziałami, do których trafiają liczby należące do danego przedziału. Przedziały mogą być wyznaczane dowolnie, jednak wymaga to odpowiedniego dostosowania algorytmu.

W implementacji wykorzystamy taką samą liczbę kubełków (przedziałów), jak w wersji algorytmu dla liczb całkowitych. Wartości, które trafią do danego kubełka, będą się mieścić w przedziałach `[x, x + 1)` dla liczb nieujemnych i `(x - 1, x]` dla liczb ujemnych. Np. liczba `3.7` trafi do kubełka reprezentującego przedział `[3, 4)`, a liczba `-4.2` do przedziału `(-5, -4]`.

Przeanalizujmy program, który wyszukuje wartość minimalną i maksymalną, przygotowuje tablicę zliczającą, zapisuje kolejne wartości w odpowiednich kubełkach i drukuje ich zawartość.

```
#include <iostream>
#include <climits>
#include <vector>

using namespace std;

void wyswietlTablice(vector<float>* tab, int k) {
    for (int i = 0; i < k; i++)
    {
        if (!tab[i].empty())
        {
            cout << "Kubełek numer " << i <<" : ";
            for (int j = 0; j < (int)tab[i].size(); j++)
            {
                cout << tab[i].at(j) << "; ";
            }
            cout << endl;
        }
```

```
    }
    cout << endl;
}

int main()
{
    float tabWejsciowa[] = { 3.5, -4.2, 3.2, 4, -4, 2, 3.9, 2, 0,
    int n = 10;

    int minLiczba = tabWejsciowa[0];
    int maxLiczba = tabWejsciowa[0];

    for(int i = 1; i < n; i++)
        {
        if(tabWejsciowa[i] < minLiczba)
            minLiczba = tabWejsciowa[i];
        else if(tabWejsciowa[i] > maxLiczba)
            maxLiczba = tabWejsciowa[i];
        }

    const int k = maxLiczba - minLiczba + 1;
    vector<float> *tabZliczajaca = new vector<float>[k];

    // zliczanie wartości
    for (int i = 0; i < n; i++)
    {
        float wartosc = tabWejsciowa[i];
        int indeksKubelka = (int)wartosc - (int)minLiczba;
        tabZliczajaca[indeksKubelka].push_back(wartosc);
    }

    // wypisanie tabZliczajaca
    wyswietlTablice(tabZliczajaca, k);

    delete[] tabZliczajaca;
    return 0;
}
```

Elementami tablicy zliczającej są tablice dynamiczne typu `vector`. Używamy takich tablic jako kubełków, ponieważ nie wiemy, ile wartości do nich trafi. Tablice tego typu

umożliwiają zastosowanie wielu użytecznych metod ułatwiających dodawanie i przeglądanie elementów.

Zwróćmy uwagę na kilka fragmentów kodu:

- `vector` – utworzenie tablicy `tabZliczajaca`, której elementami jest `k` tablic pozwalających przechowywać wartości typu `float`,
- `if (!tablicaWektorow[i].empty())` – metoda `empty()` zwraca prawdę, jeżeli tablica jest pusta, dzięki temu sprawdzamy, czy kubełek zawiera jakieś liczby,
- `tablicaWektorow[i].at(j)` – z kubełka o indeksie `i` przy użyciu metody `at()` odczytujemy element o indeksie `j`,
- `delete[] tabZliczajaca;` – usuwamy tablicę z pamięci.

W pętli umieszczającej liczby w odpowiednich kubełkach:

```
    for (int i = 0; i < n; i++)
    {
        float wartosc = tabWejsciowa[i];
        int indeksKubelka = (int)wartosc - (int)minLiczba;
        tabZliczajaca[indeksKubelka].push_back(wartosc);
    }
```

Odczytane z tablicy wejściowej wartości oraz wartość minimalną konwertujemy na liczby całkowite, co powoduje ich ewentualne zaokrąglenie. Np. dla wartości `3.2` i `3.9` po konwersji otrzymamy `3`, dla wartości `-2.2` i `-2.9` otrzymamy `-2`.

Następnie indeks kubełka, do którego trafi liczba, wyznaczamy podobnie jak w przypadku algorytmu dla liczb całkowitych:
`indeksKubelka = (int)wartosc - (int)minLiczba`.

Do posortowania zawartości poszczególnych kubełków użyjemy sortowania bąbelkowego. Elementy kubełków zostaną posortowane w porządku nierosnącym za pomocą funkcji `sortowanieBabelkowe()`.

```
void sortowanieBabelkowe(vector<float>& tab)
{
    for (int j = tab.size() - 1; j > 0; j--)
        for (int i = 0; i < j; i++) {
            if (tab[i] < tab[i + 1]) {
                float tmp = tab[i];
                tab[i] = tab[i + 1];
                tab[i + 1] = tmp;
```

```
            }
        }
}
```

Wywołanie funkcji sortującej umieścimy w pętli:

```
    for (int i = 0; i < k; i++)
    {
        if (!tabZliczajaca[i].empty())
        {
            sortowanieBabelkowe(tabZliczajaca[i]);
        }
    }
```

Kolejnym krokiem jest przepisanie elementów do tablicy wejściowej. Po kolei odczytujemy kubełki z tablicy zliczającej i przepisujemy zawarte w nich liczby do tablicy wyjściowej, zaczynając od ostatniej, czyli najmniejszej:

```
    int j = 0;
    for (int i = 0; i < k; i++)
    {
        while (!tabZliczajaca[i].empty())
        {
            tabWejsciowa[j] = tabZliczajaca[i].back();
            tabZliczajaca[i].pop_back();
            j++;
        }
    }
```

Metoda `back()` pozwala na odczytanie ostatniego elementu tablicy typu `vector`, a metoda `pop_back()` usuwa ostatni element takiej tablicy.

Na koniec wypisujemy tablicę wejściową:

```
    for (int i = 0; i < n; i++)
    {
        cout << tabWejsciowa[i] << "  ";
    }
```

Cały kod prezentuje się następująco:

```
#include <iostream>
#include <climits>
#include <vector>

using namespace std;

void sortowanieBabelkowe(vector<float>& tab)
{
    for (int j = tab.size() - 1; j > 0; j--)
        for (int i = 0; i < j; i++) {
            if (tab[i] < tab[i+1]) {
                float tmp = tab[i];
                tab[i] = tab[i+1];
                tab[i+1] = tmp;
            }
        }
}

void wyswietlTablice(vector<float>* tab, int k) {
    for (int i = 0; i < k; i++)
    {
        if (!tab[i].empty())
        {
            cout << "Kubełek numer " << i <<" : ";
            for (int j = 0; j < (int)tab[i].size(); j++)
            {
                cout << tab[i].at(j) << "; ";
            }
            cout << endl;
        }
    }
    cout << endl;
}

int main()
{
    float tabWejsciowa[] = { 3.5, -4.2, 3.2, 4, -4, 2, 3.9, 2, 0,
    int n = 10;

```

```
    // znajdowanie wartości minimalnej i maksymalnej
    int minLiczba = tabWejsciowa[0];
    int maxLiczba = tabWejsciowa[0];

    for(int i = 1; i < n; i++)
        {
        if(tabWejsciowa[i] < minLiczba)
            minLiczba = tabWejsciowa[i];
        else if(tabWejsciowa[i] > maxLiczba)
            maxLiczba = tabWejsciowa[i];
        }

    const int k = maxLiczba - minLiczba + 1;
    vector<float> *tabZliczajaca = new vector<float>[k];

    // zliczanie wartości
    for (int i = 0; i < n; i++)
    {
        float wartosc = tabWejsciowa[i];
        int indeksKubelka = (int)wartosc - (int)minLiczba;
        tabZliczajaca[indeksKubelka].push_back(wartosc);
    }

    // wypisanie tabZliczajaca
    wyswietlTablice(tabZliczajaca, k);

    // sortowanie kubełków
    for (int i = 0; i < k; i++)
    {
        if (!tabZliczajaca[i].empty())
        {
            sortowanieBabelkowe(tabZliczajaca[i]);
        }
    }

    // wypisanie tabZliczajaca
    wyswietlTablice(tabZliczajaca, k);

    // przepisanie elementów do tablicy wejściowej
    int j = 0;
    for (int i = 0; i < k; i++)
    {
```

```
82          while (!tabZliczajaca[i].empty())
83          {
84              tabWejsciowa[j] = tabZliczajaca[i].back(); //odczytan
85              tabZliczajaca[i].pop_back(); //usunięcie ostatniego e
86              j++;
87          }
88      }
89
90      for (int i = 0; i < n; i++)
91      {
92          cout << tabWejsciowa[i] << "  ";
93      }
94
95      delete[] tabZliczajaca;
96  return 0;
97  }
```

W programie umieściliśmy dodatkowe wywołanie funkcji wypisującej kubełki (tablicę zliczającą), aby pokazać wynik działania funkcji `sortowanieBabelkowe()`.

Po uruchomieniu programu powinniśmy zobaczyć następujące komunikaty:

```
Kubełek numer 0 : -5.5;
Kubełek numer 1 : -4.2; -4;
Kubełek numer 5 : 0;
Kubełek numer 7 : 2; 2;
Kubełek numer 8 : 3.5; 3.2; 3.9;
Kubełek numer 9 : 4;

Kubełek numer 0 : -5.5;
Kubełek numer 1 : -4; -4.2;
Kubełek numer 5 : 0;
Kubełek numer 7 : 2; 2;
Kubełek numer 8 : 3.9; 3.5; 3.2;
Kubełek numer 9 : 4;

-5.5  -4.2  -4  0  2  2  3.2  3.5  3.9  4
```

Słownik

> **indeks tablicy**

numer, dzięki któremu możemy odnieść się do konkretnego elementu w tej tablicy

kubełek

komórka w pamięci, obiekt, czy innego rodzaju struktura, identyfikowana jednoznacznie poprzez unikalną cechę sortowanego zbioru (w przypadku tablicy liczb każdy kubełek identyfikowany byłby unikalną liczbą), w których zliczamy liczbę wystąpień elementów, zawierających daną cechę

typ całkowitoliczbowy

typ prymitywny, w którym nie ma wartości po przecinku; może przechowywać jedynie wartości całkowitoliczbowe, w tym wartości ujemne

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Napisz program, za pomocą którego w tablicy liczb całkowitych znajdziesz najmniejszą i największą liczbę, a następnie wypiszesz obie liczby.

Działanie programu przetestuj dla tablicy
`liczby = {3, 1, 0, 10, 6, 8, 4, 6, 2, 3, 0, 6, 6, 10, 6}` oraz `n = 15`.

Specyfikacja problemu:

Dane:

- n – liczba naturalna; liczba elementów tablicy `liczby`
- `liczby` – n-elementowa tablica liczb naturalnych

Wynik:

- najmniejsza i największa wartość liczbowa z tablicy `liczby`

Twoje zadania

1. Dla podanych domyślnie danych wejściowych wypisz najmniejszą liczbę z tablicy.
2. Następnie wypisz największą liczbę z tablicy.

Ćwiczenie 2

Napisz program, który dla zadanej tablicy zliczy wystąpienia każdego z elementów do odpowiedniego kubełka. Wypisz zawartość każdego kubełka w następującej postaci: `indeks kubełka:zawartość kubełka`, np. `0:2`.

Działanie programu przetestuj dla następującej tablicy `liczby = {3, 1, 0, 10, 6, 8, 4, 6, 2, 3, 0, 6, 6, 10, 6}` oraz `n = 15`.

Specyfikacja problemu:

Dane:

- n – liczba naturalna; liczba elementów tablicy `liczby`
- `liczby` – n-elementowa tablica liczb naturalnych

Wynik:

- zawartość kolejnych kubełków zgodnie z następującym wzorem:
  `indeks kubełka:zawartość kubełka`

Wskazówka:

Pamiętaj o wyzerowaniu tablicy `kubelki` przed rozpoczęciem zliczania. Możesz wykorzystać następujący kod:

```
1 int kubelki[n] = {0};
```

Twoje zadania

1. Dla zadanej tablicy zlicz wystąpienia każdej z liczb, a następnie umieść ją w odpowiednim kubełku. Wypisz zawartość kubełków zgodnie z następującym wzorem: `indeks kubełka:zawartość kubełka`.

Ćwiczenie 3

Napisz program, który za pomocą algorytmu sortowania kubełkowego uporządkuje nierosnąco tablicę zawierającą liczby naturalne, a następnie wypisze posortowane elementy i liczbę wykorzystanych kubełków. Skonstruuj algorytm tak, aby wykorzystać minimalną możliwą liczbę kubełków.

Działanie programu przetestuj dla następującej tablicy
`liczby = {10, 6, 8, 4, 6, 6, 6, 10, 6}` oraz `n = 9`.

Specyfikacja problemu:

Dane:

- n – liczba naturalna; liczba elementów tablicy `liczby`
- `liczby` – n-elementowa tablica liczb naturalnych

Wynik:

- posortowana nierosnąco tablica `liczby`
- liczba wykorzystanych kubełków

Przykładowe wyjście:

```
10
10
8
6
6
6
6
6
4
Liczba kubełków: 7
```

Twoje zadania

1. Posortuj daną tablicę nierosnąco i wypisz liczbę wykorzystanych kubełków. Liczba ta musi być jak najmniejsza.

Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Sortowanie kubełkowe w języku C++

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):

- Zaimplementujesz algorytm sortowania kubełkowego w języku C++.
- Przeanalizujesz program realizujący w języku C++ algorytm sortowania kubełkowego liczb należących do zakresu podanego przez użytkownika.
- Rozwiążesz zadania wymagające znajomości algorytmu sortowania kubełkowego.

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;
- oprogramowanie dla języka C++, w tym kompilator GCC/G++ 4.5 (lub nowszej wersji) i Code::Blocks 16.01 (lub nowszej wersji), Orwell Dev-C++ 5.11 (lub nowszej wersji) lub Microsoft Visual Studio.

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Sortowanie kubełkowe w języku C++". Uczniowie mają zapoznać się z animacją w sekcji „Przeczytaj" oraz wykonać polecenie 2.

Faza wstępna:

1. Nauczyciel wyświetla temat i cele zajęć zawarte w sekcji „Wprowadzenie". Prosi uczniów, by na podstawie wiadomości zdobytych przed lekcją, zaproponowali własne kryteria sukcesu.
2. Prowadzący prosi uczniów, aby zgłaszali swoje propozycje pytań do tematu. Jedna osoba może zapisywać je na tablicy. Gdy uczniowie wyczerpią swoje pomysły, a pozostały jakieś ważne kwestie do poruszenia, nauczyciel je dopowiada.

Faza realizacyjna:

1. **Praca z multimedium.** Nauczyciel sprawdza przygotowanie uczniów do lekcji, prosząc ich o podzielenie się swoimi refleksjami po obejrzeniu animacji i wykonaniu polecenia 2 z sekcji „Film samouczek". Jeśli przygotowanie uczniów do lekcji jest niewystarczające, zapoznają się oni z animacją na nowo.
2. **Praca z multimedium.** Uczniowie w parach rozwiązują polecenie 3 z sekcji „Film samouczek". Następnie swoje rozwiązanie porównują z zaprezentowanym w filmie.
3. **Praca z tekstem.** Nauczyciel wyświetla zawartość sekcji „Przeczytaj". Uczniowie zapoznają się z jej treścią. Następnie na forum klasy uczniowie analizują omówione w niej dwie wersje algorytmu. W razie konieczności nauczyciel wyjaśnia ewentualne wątpliwości.
4. **Ćwiczenie umiejętności.** Prowadzący zapowiada uczniom, że będą rozwiązywać ćwiczenie nr 1 z sekcji „Sprawdź się". Każdy z uczniów robi to samodzielnie. Po ustalonym czasie następuje porównanie napisanych kodów podczas wspólnego omówienia rozwiązania.
5. Uczniowie w parach wykonują ćwiczenie nr 2 z sekcji „Sprawdź się". Nauczyciel sprawdza poprawność rozwiązania zadania, omawiając je wraz z uczniami.

Faza podsumowująca:

1. 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.
2. Wybrany uczeń podsumowuje zajęcia, zwracając uwagę na nabyte umiejętności, omawia ewentualne problemy podczas rozwiązania ćwiczeń z programowania w języku C++.

Praca domowa:

1. Uczniowie wykonują ćwiczenie 3 z sekcji „Sprawdź się”.
2. Uczniowie wykonują polecenie 5 z sekcji „Film samouczek”.

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka C++.
- Oficjalna dokumentacja techniczna dla oprogramowania Code::Blocks 16.01 (lub nowszej wersji), Orwell Dev-C++ 5.11 (lub nowszej wersji) lub Microsoft Visual Studio.

Wskazówki metodyczne:

- Treści w sekcji „Film samouczek” można wykorzystać jako podsumowanie i utrwalenie wiedzy uczniów.