Sortowanie pozycyjne liczb w języku C++

Źródło: Jan Antonin Kolar, domena publiczna.

Algorytm sortowania pozycyjnego (w języku angielskim nazywany *radix sort*) jest jednym z podstawowych sposobów porządkowania danych. Szczególnie dobrze sprawdza się przy sortowaniu dużego zbioru liczb należących do niewielkiego zakresu. W e-materiale Sortowanie pozycyjne liczb przedstawiliśmy najważniejsze informacje dotyczące zagadnienia. W tym e-materiale zajmiemy się implementacją tego algorytmu w języku C++.

Implementację sortowania pozycyjnego liczb w innych językach przedstawiamy w e-materiałach:

- Sortowanie pozycyjne liczb w języku Java,
- Sortowanie pozycyjne liczb w języku Python.

Więcej zadań? Sortowanie pozycyjne liczb – zadania maturalne.

Twoje cele

- Utrwalisz wiedzę dotyczącą sortowania liczb za pomocą algorytmu sortowania pozycyjnego.
- Przeanalizujesz implementację algorytmu sortowania pozycyjnego liczb, wykonaną w języku C++.
- Wykonasz kilka ćwiczeń z programowania w języku C++, w tematyce dotyczącej algorytmu sortowania pozycyjnego.

Film samouczek

Sortowanie pozycyjne liczb

W tej sekcji zaimplementujemy algorytm sortowania pozycyjnego liczb wykorzystujący sortowanie przez zliczanie. Algorytm sortowania przez zliczanie jest algorytmem stabilnym, co oznacza, że po posortowaniu elementy o równej wartości będą ułożone w takiej samej kolejności, w jakiej były w nieposortowanym zbiorze. Jest to warunek niezbędny w sortowaniu pozycyjnym.

Problem 1

W lipcu 2008 roku CBOS przeprowadziło ankietę. Treść brzmiała następująco:

Niedawno Irlandczycy odrzucili w referendum traktat lizboński. Czy pana/pani zdaniem prezydent Polski powinien w tej sytuacji ratyfikować traktat lizboński, czy też nie?

Wyniki ankiety:

- 34% odpowiedziało: „Trudno powiedzieć”,
- 29% odpowiedziało: „Raczej tak”,
- 23% odpowiedziało: „Zdecydowanie tak”,
- 10% odpowiedziało: „Raczej nie”,
- 4% odpowiedziało: „Zdecydowanie nie”.

Napisz program sortujący niemalejąco wyniki ankiety, dotyczącej opinii o traktacie lizbońskim, wykorzystując sortowanie pozycyjne liczb (*radix sort*). W implementacji użyj stabilnego algorytmu sortowania przez zliczanie (*counting sort*).

Specyfikacja:

Dane:

- `dane[]` – jednowymiarowa tablica liczb naturalnych; dane do posortowania
- `liczbaElementow` – liczba naturalna; długość tablicy `dane`
- `liczbaCyfr` – liczba naturalna dodatnia; liczba cyfr tworzących największą liczbę z tablicy `dane`

Wynik:

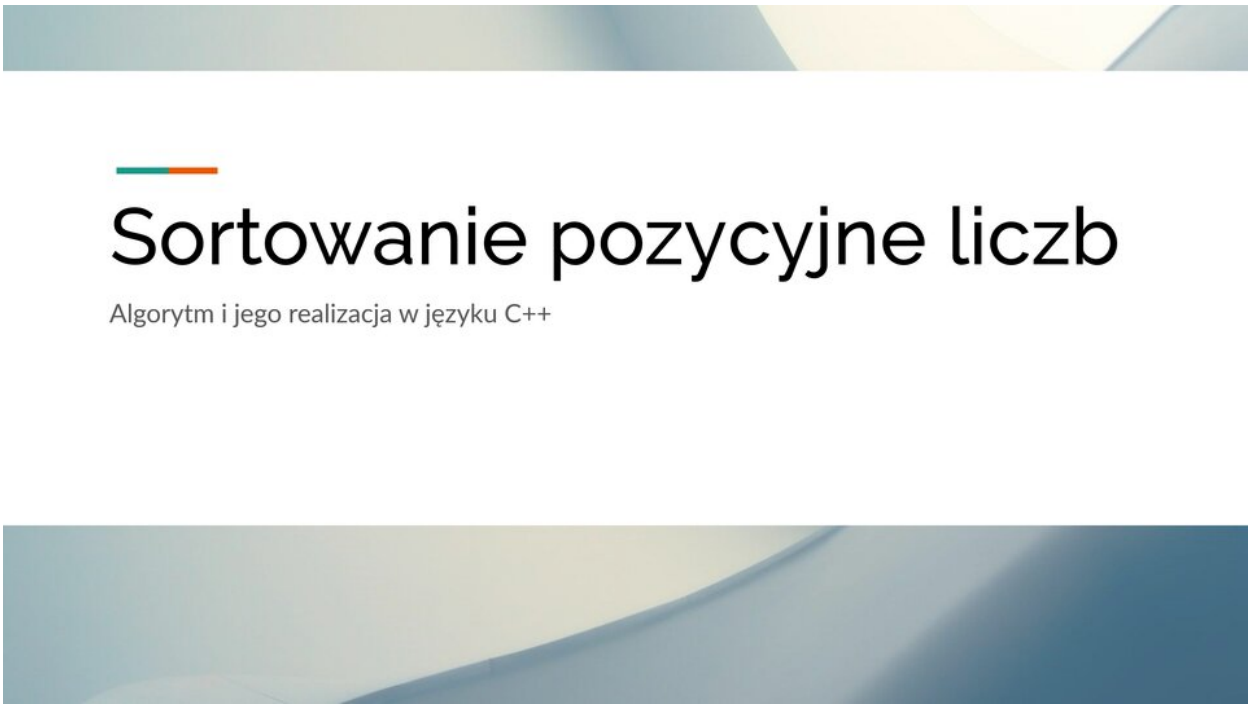
- `dane` – posortowana niemalejąco tablica liczb naturalnych

```
#include <iostream>
using namespace std;

int dane[] = {29, 23, 10, 4, 34};
int liczbaElementow = 5;
int liczbaCyfr = 2;

int main () {

    // Tutaj dodaj kod.
}
```

```
1
```

Polecenie 1

Porównaj swoje rozwiązanie z przedstawionym w filmie.



Film dostępny pod adresem /preview/resource/R11oyFi2YbPVj

Lekcja poświęcona sortowaniu pozycyjnemu liczb w języku C++.

Kod programu zaprezentowanego w filmie:

Plik o rozmiarze 1.22 KB w języku polskim

> **Polecenie 2**
>
> Dodaj do swojego programu komentarze tak, żeby był zrozumiały dla osoby, która nie potrafi programować.

Przeczytaj

Sortowanie pozycyjne

Sortowanie pozycyjne liczb jest metodą sortowania opierającą się na porządkowaniu elementów ze względu na cyfry umieszczone na tych samych pozycjach.

W tej sekcji zapoznamy się z implementacją algorytmu sortowania pozycyjnego liczb, wykorzystującego sortowanie bąbelkowe. Ten algorytm, podobnie jak algorytm sortowania przez zliczanie, cechuje stabilność. Sortowanie bąbelkowe swoje działanie opiera na dwóch zagnieżdżonych pętlach. Jeżeli elementy nie zostały umieszczone w odpowiedniej kolejności, zamienia je miejscami.

Implementacja w języku C++

Zanim przystąpimy do implementacji, zapoznajmy się ze specyfikacją.

Specyfikacja:

Dane:

- `dane[]` – jednowymiarowa tablica liczb naturalnych; dane do posortowania
- `liczbaElementow` – liczba naturalna; długość tablicy dane
- `liczbaCyfr` – liczba naturalna dodatnia; liczba cyfr tworzących największą liczbę z tablicy dane

Wynik:

- `dane` – posortowana niemalejąco tablica liczb naturalnych

Mamy do posortowania następujący zbiór: **{562, 3, 11, 25, 303, 4, 500}**. Zaczynamy od stworzenia tablicy o nazwie `dane`, w której umieszczamy elementy zbioru. Ponadto tworzymy dwie zmienne: `liczbaElementow`, przechowującą liczbę elementów zbioru, a także `liczbaCyfr`, której przypiszemy liczbę cyfr, z jakiej składa się największa liczba w zbiorze.

```
#include <iostream>
#include <cmath>
using namespace std;

int dane[] = {562, 3, 11, 25, 303, 4, 500};
int liczbaElementow = 7;
```

```
int liczbaCyfr = 3;
```

Następnie implementujemy funkcję typu `void`, która jest odpowiedzialna za sortowanie liczb na danej pozycji za pomocą algorytmu sortowania bąbelkowego. Funkcja przyjmuje parametr równy numerowi aktualnej pozycji.

```
#include <iostream>
#include <cmath>
using namespace std;

int dane[] = {562, 3, 11, 25, 303, 4, 500};
int liczbaElementow = 7;
int liczbaCyfr = 3;

void sortowanieBabelkowe(int pozycjaCyfry) {
}
```

Algorytm sortowania bąbelkowego (ang. *bubble sort*) bazuje na utworzeniu zagnieżdżonej pętli `for`, w której sprawdzane jest, czy element poprzedni jest większy od następnego, to znaczy, czy ułożone są one w kolejności malejącej. Jeżeli jest to prawda, zamieniamy elementy ze sobą, ponieważ naszym celem jest posortowanie tablicy w porządku niemalejącym.

W algorytmie sortowania pozycyjnego sortujemy liczby na pojedynczych pozycjach. Tworzymy zatem dodatkowo zmienną `wspolczynnikCyfry`, która jest potrzebna do odczytania cyfry na odpowiedniej pozycji liczby ze zbioru. Nie będziemy sortować całych liczb, lecz tylko cyfry na konkretnej pozycji. Umożliwi nam to operacja polegająca na podzieleniu liczby przez `wspolczynnikCyfry`, a następnie obliczeniu reszty z dzielenia jej przez 10.

```
#include <iostream>
#include <cmath>
using namespace std;

int dane[] = {562, 3, 11, 25, 303, 4, 500};
int liczbaElementow = 7;
int liczbaCyfr = 3;

void sortowanieBabelkowe(int pozycjaCyfry) {

```

```
    int wspolczynnikCyfry = pow(10, pozycjaCyfry);

    for(int i = liczbaElementow - 1; i >=0; i--) {
        for(int j = 0; j < i; j++) {
            if(((dane[j] / wspolczynnikCyfry) % 10) > ((dane[j+1]
                int temp = dane[j];
                dane[j] = dane[j+1];
                dane[j+1] = temp;
            }
        }
    }
}
```

Możemy zoptymalizować kod, wprowadzając flagę, dzięki której zredukujemy liczbę niepotrzebnych iteracji pętli. Wystarczy, że dodamy zmienną logiczną `bool` o nazwie `czyZmiana`, która domyślnie będzie przyjmować wartość `false` – taką ustawiamy wartość początkową flagi.

Jeżeli chociaż w jednej iteracji pętli wewnętrznej zostaną zamienione ze sobą dwa elementy tablicy, ustawiamy wartość flagi na `true`. W przypadku, gdy pętla wewnętrzna zakończy swoje działanie, a w jednym całym przejściu nie zostaną zamienione ze sobą żadne elementy, wartość flagi nie zmieni się, tym samym kończąc sortowanie.

```
#include <iostream>
#include <cmath>
using namespace std;

int dane[] = {562, 3, 11, 25, 303, 4, 500};
int liczbaElementow = 7;
int liczbaCyfr = 3;

void sortowanieBabelkowe(int pozycjaCyfry) {

    int wspolczynnikCyfry = pow(10, pozycjaCyfry);
    bool czyZmiana = false;

    for(int i = liczbaElementow - 1; i >=0; i--) {
        for(int j = 0; j < i; j++) {
            if(((dane[j] / wspolczynnikCyfry) % 10) > ((dane[j+1]
                int temp = dane[j];
                dane[j] = dane[j+1];
```

```
                dane[j+1] = temp;

                czyZmiana = true;
            }
        }

        if(!czyZmiana) {
            break;
        }
    }
}
```

Następnie w głównej funkcji programu używamy sortowania bąbelkowego w sortowaniu pozycyjnym zbioru liczb. Tworzymy pętlę `for`, która będzie iterować po pozycjach liczb – od pozycji 0 do pozycji równej wartości zapisanej w zmiennej `liczbaCyfr`. Dla każdej pozycji zostanie wywołana funkcja `sortowanieBabelkowe` z argumentem równym pozycji sortowanej cyfry.

```
#include <iostream>
#include <cmath>
using namespace std;

int dane[] = {562, 3, 11, 25, 303, 4, 500};
int liczbaElementow = 7;
int liczbaCyfr = 3;

void sortowanieBabelkowe(int pozycjaCyfry) {

    int wspolczynnikCyfry = pow(10, pozycjaCyfry);

    for(int i = liczbaElementow - 1; i >=0; i--) {
        for(int j = 0; j < i; j++) {
            if(((dane[j] / wspolczynnikCyfry) % 10) > ((dane[j+1]
                int temp = dane[j];
                dane[j] = dane[j+1];
                dane[j+1] = temp;
            }
        }
    }
}
int main() {
```

```
24
25      for(int numerCyfry = 0; numerCyfry < liczbaCyfr; numerCyfry++
26          sortowanieBabelkowe(numerCyfry);
27      }
28
29      return 0;
30  }
```

Na koniec wyświetlamy posortowaną tablicę. Tak wygląda skończony kod programu sortującego zbiór liczb algorytmem sortowania pozycyjnego:

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4
5  int dane[] = {562, 3, 11, 25, 303, 4, 500};
6  int liczbaElementow = 7;
7  int liczbaCyfr = 3;
8
9  void sortowanieBabelkowe(int pozycjaCyfry) {
10
11     int wspolczynnikCyfry = pow(10, pozycjaCyfry);
12
13     for(int i = liczbaElementow - 1; i >=0; i--) {
14         for(int j = 0; j < i; j++) {
15             if(((dane[j] / wspolczynnikCyfry) % 10) > ((dane[j+1]
16                 int temp = dane[j];
17                 dane[j] = dane[j+1];
18                 dane[j+1] = temp;
19             }
20         }
21     }
22 }
23 int main() {
24
25     for(int numerCyfry = 0; numerCyfry < liczbaCyfr; numerCyfry++
26         sortowanieBabelkowe(numerCyfry);
27     }
28     for(int i = 0; i < liczbaElementow; i++){
29         cout << dane[i] << " ";
30     }
```

```
31
32      return 0;
33  }
```

Algorytm sortowania bąbelkowego nie jest wykorzystywany w praktyce przy implementacji sortowania pozycyjnego z powodu jego kwadratowej złożoności czasowej. Program, który przeanalizowaliśmy, jest przykładowy, został przedstawiony w celu poznania możliwości sortowania pozycyjnego. Częściej wykorzystuje się inne algorytmy, np. sortowanie kubełkowe, czy sortowanie przez zliczanie, którego główną zaletą jest liniowa złożoność czasowa $O(n + k)$, gdzie n to liczebność sortowanego zbioru, a k to rozpiętość danych. W przypadku sortowania przez zliczanie użytego w algorytmie sortowania pozycyjnego rozpiętość danych nie jest duża (sortujemy liczby dziesiętne, więc na każdej pozycji sortujemy liczby z zakresu 0–9).

Słownik

sortowanie bąbelkowe

polega na porównywaniu ze sobą dwóch sąsiadujących elementów w kolejności od lewej do prawej i zamianie ich ze sobą, jeżeli znajdują się w nieprawidłowej kolejności

sortowanie kubełkowe

polega na umieszczaniu elementów sortowanej tablicy do kubełków, które zawierają liczby z określonego przedziału; każdy kubełek zostaje posortowany oddzielnie; w ostatnim kroku tworzona jest tablica końcowa, do której zapisywane są dane z kolejnych kubełków

sortowanie przez zliczanie

polega na sprawdzeniu, ile wystąpień kluczy mniejszych od danego występuje w sortowanej tablicy; algorytm przystosowany jest do sortowania zbioru liczb, które mogą posiadać skrajnie różne, niepowtarzające się wartości

stabilny algorytm sortowania

algorytm sortowania, w którym elementy o równej wartości po posortowaniu będą ułożone w takiej samej kolejności, w jakiej były w nieposortowanym zbiorze

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Napisz program sortujący pozycyjnie tak, aby cyfry na kolejnych pozycjach były sortowane metodą sortowania przez wstawianie. Zbiór ma być posortowany nierosnąco.

Przetestuj działanie programu dla zbioru **{5000, 23, 567, 34909, 2, 98, 1010, 8, 90}**.

Specyfikacja:

Dane:

- `dane[]` – jednowymiarowa tablica liczb całkowitych; liczby do posortowania
- `liczbaElementow` – liczba naturalna; liczba elementów umieszczonych w tablicy `dane`
- `liczbaCyfr` – liczba naturalna dodatnia; liczba cyfr tworzących największą liczbę z tablicy `dane`

Wynik:

- `dane` – posortowana nierosnąco tablica liczb całkowitych

Twoje zadania

1. Program wyświetla posortowany nierosnąco zbiór liczb {5000, 23, 567, 34909, 2, 98, 1010, 8, 90}.

Ćwiczenie 2

Zawodnicy dwóch drużyn rywalizowali w pewnej konkurencji, której celem było uzyskanie jak najmniejszej liczby punktów. W każdym etapie można było zdobyć liczbę punktów będącą naturalną potęgą liczby 10. Okazało się, że zawodnicy drużyny A uzyskali parzyste miejsca w rywalizacji, a zawodnicy drużyny B zdobyli miejsca nieparzyste. Ile punktów drużyna A powinna stracić, aby doszło do remisu w klasyfikacji drużynowej? Napisz program wyświetlający różnicę punktową między drużyną A i B. Zauważ, że liczby w zbiorze składają się wyłącznie z cyfr 0 i 1. Cyfry na kolejnych pozycjach posortuj, używając algorytmu sortowania przez zliczanie.

Przetestuj działanie programu dla następujących wyników **{1000, 110010, 100001, 11, 10, 101, 1010, 0, 1100110, 1}**.

Specyfikacja:

Dane:

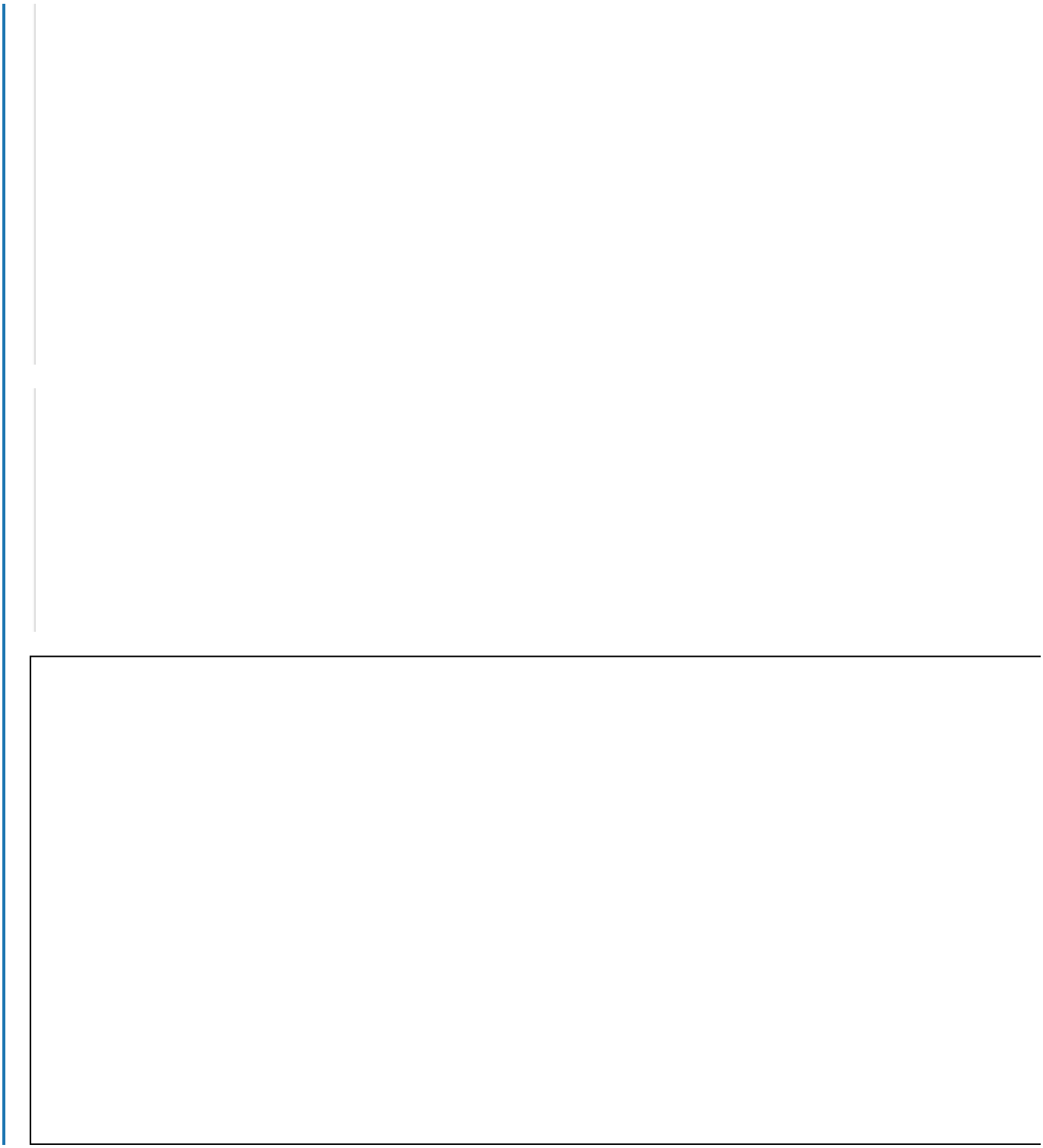
- `dane[ ]` – jednowymiarowa tablica liczb całkowitych; liczby do posortowania
- `liczbaElementow` – liczba naturalna; liczba elementów umieszczonych w tablicy `dane`
- `liczbaCyfr` – liczba naturalna dodatnia; liczba cyfr tworzących największą liczbę z tablicy `dane`

Wynik:

- `roznica` – liczba całkowita

Twoje zadania

1. Program wyświetla różnicę punktów między drużynami. Zawodnicy uzyskali następujące wyniki: 1000, 110010, 100001, 11, 10, 101, 1010, 0, 1100110, 1.



Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Sortowanie pozycyjne liczb w języku C++

Grupa docelowa:

Szkoła ponadpodstawowa, 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:

3) objaśnia, a także porównuje podstawowe metody i techniki algorytmiczne oraz struktury danych, wykorzystując przy tym przykłady problemów i algorytmów, w szczególności:

a) wyszukiwanie elementów liniowe i przez połowienie (do znajdowania elementów w zbiorze, sortowania przez wstawianie, przybliżonego rozwiązywania równań, sprawdzania przynależności punktu do wielokąta wypukłego),

i) struktury dynamiczne: stos, kolejka, lista (do realizacji algorytmu: ONP, symulacji problemu Flawiusza, sortowania leksykograficznego),

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):

- Utrwalisz wiedzę dotyczącą sortowania liczb za pomocą algorytmu sortowania pozycyjnego.
- Przeanalizujesz implementację algorytmu sortowania pozycyjnego liczb, wykonaną w języku C++.
- Wykonasz kilka ćwiczeń z programowania w języku C++, w tematyce dotyczącej algorytmu sortowania pozycyjnego.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- burza mózgów;
- 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 pozycyjne liczb w języku C++". Uczniowie mają zapoznać się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. Metodą burzy mózgów uczniowie przypominają sobie najważniejsze informacje dotyczące sortowania pozycyjnego liczb.
2. Nauczyciel wyświetla temat i cele zajęć zawarte w sekcji „Wprowadzenie". Następnie wspólnie z uczniami ustala kryteria sukcesu.
3. **Rozpoznanie wiedzy uczniów.** Nauczyciel prosi wybranego ucznia lub uczniów o przedstawienie sytuacji problemowej związanej z tematem 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".
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Film samouczek". Uczniowie wspólnie zapoznają się z treścią polecenia 1. Pracując w parach, piszą program, a następnie porównują swoje rozwiązania z przedstawionym w filmie.
3. **Ćwiczenie umiejętności.** Uczniowie indywidualnie wykonują ćwiczenie nr 1 z sekcji „Sprawdź się". Nauczyciel sprawdza poprawność wykonanych zadań, omawiając je wraz z uczniami.

Faza podsumowująca:

1. Nauczyciel ponownie wyświetla na tablicy temat lekcji zawarty w sekcji „Wprowadzenie" i inicjuje krótką rozmowę na temat zrealizowanych celów (czego uczniowie się nauczyli).
2. Na koniec zajęć z programowania w C++ nauczyciel prosi uczniów o rozwinięcie zdania: „Na dzisiejszych zajęciach nauczyłam/łem się jak...".

Praca domowa:

1. Uczniowie wykonują ćwiczenie 2 z sekcji „Sprawdź się”.

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka C++.
- Oficjalna dokumentacja techniczna dla kompilatora GCC/G++ 4.5 (lub nowszej wersji).
- 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 „Przeczytaj” można wykorzystać jako materiał do lekcji powtórkowej.