Sortowanie szybkie w języku C++

Źródło: Kelvyn Ornettte Sol Marte, domena publiczna.

Wiesz już, że algorytmy różnią się od siebie złożonością czasową. Algorytm sortowania szybkiego, który zaprezentowany został w e-materiale Sortowanie szybkie, osiąga złożoność $O\left(n\ \log_n\right)$. Jak zaimplementować go w programie? Omówimy to zagadnienie na przykładzie języka C++.

Implementację sortowania szybkiego w innych językach programowania znajdziesz w e-materiałach:

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

Więcej zadań? Przejdź do: Sortowanie szybkie – zadania maturalne.

> **Twoje cele**

- Zaimplementujesz w języku C++ algorytm sortowania szybkiego, tzw. *quick sort*.
- Wykorzystasz zdobytą wiedzę do zrozumienia i praktycznego zastosowania algorytmu sortowania szybkiego.
- Prześledzisz i utrwalisz informacje dotyczące algorytmu sortowania szybkiego.

Film samouczek

Wiesz już, na czym polega sortowanie szybkie oraz w jakich sytuacjach jest przydatne. Zapoznajmy się z implementacją tego algorytmu w języku C++.

> **Ważne!**
>
> Algorytm *quick sort* korzysta z metody „dziel i zwyciężaj". Oznacza to, że jest algorytmem rekurencyjnym. To wydajny algorytm (obecnie jeden z najszybszych), ale jego wadą jest to, że obciąża pamięć komputera.

> **Polecenie 1**
> Napisz program, który posortuje niemalejąco wyniki ankiety dotyczącej równości w życiu społecznym, wykorzystując sortowanie szybkie (*quick sort*).
>
> CBOS, luty 2000, cbos.pl [dostęp 25.05.2021 r.].
>
> **Specyfikacja problemu:**
>
> *Dane:*
>
> - `liczbaElementow` - liczba naturalna; liczba elementów tablicy `dane`
> - `dane` - tablica liczb całkowitych o liczbie elementów równej `liczbaElementow`
>
> *Wynik:*
>
> - `dane` – tablica liczb całkowitych posortowana w kolejności niemalejącej

> **Polecenie 2**
> Porównaj swoje rozwiązanie z filmem.

Trwa wczytywanie danych ..

Film dostępny pod adresem /preview/resource/RICC3E2AHyHL3

Film nawiązujący do treści materiału: Sortowanie tablicy metodą quick sort.

Plik o rozmiarze 1.36 KB w języku polskim

Polecenie 3
Napisz program sortujący nierosnąco wzrost członków pewnej rodziny, podany w centymetrach.

Specyfikacja problemu:

Dane:

- `wzrosty[]` – tablica liczb całkowitych

Wynik:

- `wzrosty` – tablica liczb całkowitych posortowana w kolejności nierosnącej

Przeczytaj

Algorytm sortowania szybkiego – implementacja w języku C++

Aby skonstruować algorytm sortowania szybkiego, możemy napisać dwie funkcje, w tym jedną wywoływaną rekurencyjnie. Wykonujemy w tym celu następujące kroki:

1. Tworzymy nagłówek funkcji, w którym zapisujemy typ zwracanej wartości (w tym przypadku funkcja nic nie zwraca, zatem podajemy typ `void`), nazwę funkcji `quicksort`, oraz w okrągłych nawiasach podajemy argumenty, jakie funkcja przyjmuje.
2. Zapisujemy warunek sprawdzany w funkcji. Sprawdzamy, czy `IndeksPoczatkowy` jest mniejszy od `IndeksKoncowy`. Gdyby tak nie było, rozmiar tablicy byłby ujemny bądź równy 0, zatem nie byłoby możliwe operowanie na nieistniejącej tablicy.
3. W przypadku, gdy tablica istnieje, deklarujemy zmienną `a` równą zwracanej wartości przez funkcję `PodzielTablice()`. Następnie rekurencyjnie wywołujemy funkcję `quicksort` dla dwóch oddzielnych części tablicy: pierwszy raz dla części od początku tablicy do wartości zmiennej `a` pomniejszonej o 1, zaś drugi dla części tablicy od wartości zmiennej `a` powiększonej o 1 do końca tablicy.

Po wykonaniu tych kroków uzyskujemy kod, który prezentuje się następująco:

```
1 void quicksort(int tab[],int IndeksPoczatkowy, int IndeksKoncowy)
2 {
3
4     if (IndeksPoczatkowy < IndeksKoncowy)
5     {
6         int a = PodzielTablice(tab, IndeksPoczatkowy, IndeksKoncc
7
8         quicksort(tab, IndeksPoczatkowy, a - 1);
9         quicksort(tab, a + 1, IndeksKoncowy);
```

```
    }
}
```

Następnie tworzymy funkcję `PodzielTablice()`, która będzie odpowiedzialna za posortowanie tablicy w taki sposób, aby liczba `pivot` była na środku, liczby od niej mniejsze po jej lewej stronie, a większe – po prawej. W tym celu musimy wykonać następujące kroki:

1. Tworzymy nagłówek funkcji, w którym zapisujemy typ zwracanej zmiennej (`int`), nazwę funkcji, a następnie w nawiasach okrągłych jej argumenty:

`int tab[]` – tablica, na której będziemy wykonywać operacje
`int IndeksPoczatkowy, int IndeksKoncowy`

```
int PodzielTablice(int tab[], int IndeksPoczatkowy, int IndeksKon
{ }
```

2. Deklarujemy zmienną `int pivot` równą liczbie zapisanej pod indeksem końcowym w tablicy `tab[]`.

```
int PodzielTablice(int tab[], int IndeksPoczatkowy, int IndeksKon
{
    int pivot = tab[IndeksKoncowy];
}
```

3. Deklarujemy zmienną `int IndeksMniejszegoElementu` równą `IndeksPoczatkowy - 1`.

```
int PodzielTablice(int tab[], int IndeksPoczatkowy, int IndeksKon
{
    int pivot = tab[IndeksKoncowy];

    int IndeksMniejszegoElementu = IndeksPoczatkowy - 1;
}
```

4. Tworzymy pętlę `for` wykonującą się o jeden raz mniej, niż wynosi różnica indeksów: końcowy minus początkowy.

```
int PodzielTablice(int tab[], int IndeksPoczatkowy, int IndeksKon
{
    int pivot = tab[IndeksKoncowy];

    int IndeksMniejszegoElementu = IndeksPoczatkowy - 1;

    for (int j = IndeksPoczatkowy; j < IndeksKoncowy; j++)
    {

    }
}
```

5. W pętli sprawdzamy warunek, czy aktualnie badana liczba jest mniejsza od liczby `pivot`.

```
int PodzielTablice(int tab[], int IndeksPoczatkowy, int IndeksKon
{
    int pivot = tab[IndeksKoncowy];

    int IndeksMniejszegoElementu = IndeksPoczatkowy - 1;

    for (int j = IndeksPoczatkowy; j < IndeksKoncowy; j++)
    {
        if (tab[j] < pivot)
        {

        }
    }
}
```

6. Jeśli tak jest, zwiększamy `IndeksMniejszegoElementu` o jeden oraz zamieniamy miejscami liczby pod indeksami o wartościach `IndeksuMniejszegoElementu` oraz obecnie sprawdzanego elementu, czyli `j`.

```
int PodzielTablice(int tab[], int IndeksPoczatkowy, int IndeksKon
{
    int pivot = tab[IndeksKoncowy];

    int IndeksMniejszegoElementu = IndeksPoczatkowy - 1;

    for (int j = IndeksPoczatkowy; j < IndeksKoncowy; j++)
    {
        if (tab[j] < pivot)
        {
            IndeksMniejszegoElementu++;
            swap(tab[IndeksMniejszegoElementu], tab[j]);
        }
    }
}
```

7. Po wyjściu z pętli zamieniamy miejscami elementy o indeksach `IndeksMniejszegoElementu + 1` oraz `IndeksKoncowy` (czyli liczbę `pivot`).

```
int PodzielTablice(int tab[], int IndeksPoczatkowy, int IndeksKon
{
    int pivot = tab[IndeksKoncowy];

    int IndeksMniejszegoElementu = IndeksPoczatkowy - 1;

    for (int j = IndeksPoczatkowy; j < IndeksKoncowy; j++)
    {
        if (tab[j] < pivot)
        {
            IndeksMniejszegoElementu++;
            swap(tab[IndeksMniejszegoElementu], tab[j]);
```

```
        }
    }
    swap(tab[IndeksMniejszegoElementu + 1], tab[IndeksKoncowy]);
}
```

> **Ważne!**
>
> Zamiast zastosowania funkcji `swap()`, która zamienia wartości dwóch obiektów będących argumentami, można to zrealizować z wykorzystaniem zmiennej pomocniczej. Warto o tym pamiętać, ponieważ często na egzaminie maturalnym korzystanie z takich funkcji jest zakazane.
>
> Spróbuj wprowadzić w kodzie poprawkę tak, by funkcja była zbędna.

8. Na końcu zwracamy `IndeksMniejszegoElementu + 1`

Po poprawnym wykonaniu przedstawionych kroków uzyskujemy następujący kod:

```
int PodzielTablice(int tab[], int IndeksPoczatkowy, int IndeksKon
{
    int pivot = tab[IndeksKoncowy];

    int IndeksMniejszegoElementu = IndeksPoczatkowy - 1;

    for (int j = IndeksPoczatkowy; j < IndeksKoncowy; j++)
    {
        if (tab[j] < pivot)
        {
            IndeksMniejszegoElementu++;
            swap(tab[IndeksMniejszegoElementu], tab[j]);
        }
    }
    swap(tab[IndeksMniejszegoElementu + 1], tab[IndeksKoncowy]);

    return IndeksMniejszegoElementu + 1;
```

```
}
```

Zapoznajmy się całym kodem programu, który ma posortować niemalejąco tablicę danych `tab` zawierającą `n` liczb naturalnych.

Specyfikacja problemu:

Dane:

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

Wynik:

- `tab` – `n`-elementowa tablica liczb naturalnych posortowana niemalejąco

```
#include <iostream>

using namespace std;

int PodzielTablice(int tab[], int IndeksPoczatkowy, int IndeksKon
{
    int pivot = tab[IndeksKoncowy];

    int IndeksMniejszegoElementu = IndeksPoczatkowy - 1;

    for (int j = IndeksPoczatkowy; j < IndeksKoncowy; j++)
    {
        if (tab[j] < pivot)
        {
            IndeksMniejszegoElementu++;
            swap(tab[IndeksMniejszegoElementu], tab[j]);
        }
    }
```

```
19      swap(tab[IndeksMniejszegoElementu + 1], tab[IndeksKoncowy]);
20
21      return IndeksMniejszegoElementu + 1;
22  }
23
24
25  void quicksort(int tab[],int IndeksPoczatkowy, int IndeksKoncowy)
26  {
27
28      if (IndeksPoczatkowy < IndeksKoncowy)
29      {
30          int a = PodzielTablice(tab, IndeksPoczatkowy, IndeksKoncc
31
32          quicksort(tab, IndeksPoczatkowy, a - 1);
33          quicksort(tab, a + 1, IndeksKoncowy);
34      }
35  }
36
37  int main()
38  {
39      int tab[] = {27, 5, 2023, 15, 1, 1994, 9, 12, 1987};
40      int n = sizeof(tab)/sizeof(tab[0]);
41      quicksort(tab,0,n - 1);
42
43      for (int i = 0; i < n; i++)
44          cout << tab[i] << " ";
45
46      return 0;
47  }
```

Słownik

> **pivot**
>
> element osiowy – element tablicy wybrany wedle ustalonego schematu; jest to element odpowiadający za sposób dzielenia tablicy

rekurencja

proces polegający na wywoływaniu funkcji przez siebie samą do momentu rozwiązania określonego problemu

wywołanie rekurencyjne

sytuacja, w której funkcja wywołuje samą siebie

zasada „dziel i zwyciężaj”

zasada często stosowana przez programistów przy tworzeniu algorytmów opartych o rekurencję, polegająca na tym, że jeden trudny, złożony problem dzielony jest na kilka mniejszych, prostszych do rozwiązania

złożoność czasowa

ilość czasu potrzebnego do wykonania zadania, wyrażona jako funkcja ilości danych

złożoność obliczeniowa

złożoność czasowa i złożoność pamięciowa; ilość zasobów komputerowych potrzebnych do wykonania zadania

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Napisz program, który wykorzystując algorytm sortowania szybkiego, wypisze wartość minimalną oraz maksymalną z podanej tablicy. Przetestuj działanie programu dla tablicy `tab[] = {9, 11, 0, -8, 11, 5, 20, 45, 0, 100}`.

Specyfikacja problemu:

Dane:

- n – liczba naturalna; liczba elementów tablicy `tab[]`
- `tab[]` – n-elementowa tablica liczb całkowitych

Wynik:

- minimum i maksimum dla tablicy `tab[]`; liczby całkowite

Przykładowe wyjście:

```
-8 100
```

Ćwiczenie 2

Napisz program, który przy użyciu algorytmu sortowania szybkiego uporządkuje zbiór podanych liter alfabetu łacińskiego w kolejności odwrotnej do alfabetycznej. Przetestuj działanie programu dla następującego zbioru liter
`tab[] = {a, f, e, o, b, l, q, y}`.

Specyfikacja problemu:

Dane:

- n – liczba naturalna; liczba elementów tablicy `tab[]`
- `tab[]` – n-elementowa tablica zawierająca małe litery alfabetu łacińskiego

Wynik:

- `tab[]` – tablica znaków posortowana w kolejności odwrotnej do kolejności alfabetycznej; elementy oddzielone są pojedynczym znakiem spacji

Przykładowe wyjście:

```
y q o l f e b a
```

Ćwiczenie 3

W ramach badania zapytano grupę respondentów o zarobki. Odpowiedzi umieszczono w tablicy. Użyj algorytmu sortowania szybkiego, aby znaleźć **medianę** zarobków w tej grupie. Program powinien wydrukować wynik na standardowe wyjście.

> **Ważne!**
>
> Mediana to wartość środkowa. Aby wyznaczyć medianę jakiegoś zbioru liczb, musimy najpierw wypisać te liczby w kolejności niemalejącej, a następnie wybrać liczbę środkową (w przypadku, gdy mamy nieparzystą liczbę liczb w zbiorze). Jeśli mamy parzystą liczbę liczb w zbiorze, to mediana jest równa średniej arytmetycznej dwóch środkowych liczb.

Przetestuj jego działanie dla tablicy
`zarobki[] = {8500.57, 6400.32, 2800.56, 3500.12, 12870.67, 3300.4`
.

Specyfikacja problemu:

Dane:

- n – liczba naturalna; liczba elementów tablicy `zarobki[]`
- `zarobki[]` – n-elementowa tablica liczb rzeczywistych

Wynik:

- mediana posortowanej tablicy `zarobki[]`; liczba rzeczywista

Przykładowe wyjście

```
6400.32
```

Dla nauczyciela

Autor: Maurycy Gast

Przedmiot: Informatyka

Temat: Sortowanie szybkie 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

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:

c) metodę dziel i zwyciężaj (jednoczesne znajdowanie minimum i maksimum, sortowanie przez scalanie i szybkie),

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 w języku C++ algorytm sortowania szybkiego, tzw. *quick sort*.
- Wykorzystasz zdobytą wiedzę do zrozumienia i praktycznego zastosowania algorytmu sortowania szybkiego.
- Prześledzisz i utrwalisz informacje dotyczące algorytmu sortowania szybkiego.

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 szybkie w języku C++”. Nauczyciel prosi uczniów o zapoznanie się z treściami w sekcji „Przeczytaj”.

Faza wstępna:

1. Nauczyciel sprawdza przygotowanie uczniów do lekcji.
2. Nauczyciel wyświetla uczniom temat, wskazuje cele zajęć oraz ustala z uczestnikami zajęć kryteria sukcesu.

3. **Rozpoznanie wiedzy uczniów.** Nauczyciel prosi wybranego ucznia lub uczniów o przedstawienie sytuacji problemowej związanej z tematem lekcji oraz w odniesieniu do poznanych języków programowania.

Faza realizacyjna:

1. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Film samouczek". Uczniowie zapoznają się z poleceniem 1. W parach opracowują program. Chętne lub wybrane osoby przedstawiają swoje programy. Nauczyciel inicjuje dyskusję na temat programów. Uczniowie porównują swoje programy z rozwiązaniem zaproponowanym w filmie.
2. **Praca z tekstem**. Uczniowie analizują i testują przedstawiony w tej sekcji algorytm. W razie potrzeby nauczyciel odpowiada na pytania.
3. **Ćwiczenie umiejętności.** Prowadzący zapowiada uczniom, że będą rozwiązywać ćwiczenie nr 1 z sekcji „Sprawdź się". Uczniowie wykonują je w parach. Po ustalonym czasie następuje porównanie napisanych kodów podczas wspólnego omówienia rozwiązań.
4. Uczniowie samodzielnie wykonują ćwiczenie nr 2 z sekcji „Sprawdź się". Chętne lub wybrane osoby przedstawiają rozwiązania i je omawiają.

Faza podsumowująca:

1. Na koniec zajęć nauczyciel raz jeszcze wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W odniesieniu do ich realizacji dokonuje szczegółowej oceny rozwiązania zastosowanego przez wybranego ucznia.
2. Nauczyciel prosi uczniów o podsumowanie zgromadzonej wiedzy w zakresie programowania w języku C++.

Praca domowa:

1. Uczniowie wykonują ćwiczenie nr 3 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:

- Film samouczek może być zaproszeniem do pracy w grupach i przygotowania prezentacji na temat wybranych klas algorytmów.