Podstawy Programowania 1

Sortowanie tablic jednowymiarowych

Arkadiusz Chrobot

Katedra Systemów Informatycznych

30 listopada 2020

Plan

1. Sortowanie
2. Sortowanie przez wybór
3. Sortowanie przez wstawianie
4. Sortowanie bąbelkowe
5. Wyszukiwanie binarne
6. Wartość minimalna i maksymalna w posortowanej tablicy
7. Zakończenie

Sortowanie

Operacje sortowania (porządkowania) i wyszukiwania danych są jednymi z najczęściej wykonywanych przez komputery. Prof. Knuth poświęcił tym zagadnieniom cały trzeci tom swojej książki pt. „Sztuka programowania”, który po raz pierwszy zostały wydany na początku lat siedemdziesiątych. Można się z niego dowiedzieć, że gdyby wszystkie komputery zostały zatrzymane, to większość z nich wykonywałaby właśnie te operacje. Sortowanie może dotyczyć wielu struktur danych, ale na tym wykładzie będzie nas interesowało sortowanie tablic jednowymiarowych. Wartości w strukturach danych mogą być porządkowane malejąco lub rosnąco, jeśli się one nie powtarzają lub nierosnąco bądź niemalejąco w przeciwnym przypadku.

Sortowanie

Rodzaje sortowania

Powstało wiele algorytmów sortowania, które różnią się wieloma cechami. Sortowanie wewnętrzne przeprowadzane jest w całości w pamięci operacyjnej komputera[1]. Sortowanie zewnętrzne oznacza porządkowanie danych umieszczonych w pamięci masowej. Sortowanie stabilne zachowuje początkowy względny porządek jednakowych wartości, np. jeśli mamy w tablicy dwie liczby 10, które oznaczymy jako 10' i 10", to po posortowaniu tej struktury danych 10' nadal będzie przed 10". Jeśli sortowanie jest niestabilne, to 10" znajdzie się przed 10'. W przypadku liczb taka cecha ma niewielkie znaczenie, ale w przypadku innych typów danych może być bardzo pomocna. Algorytmy sortujące w miejscu (łac. *in situ*) nie zwiększają zapotrzebowania na miejsce w pamięci operacyjnej w trakcie wykonywania ich kolejnych kroków. Dzięki temu to zapotrzebowanie daje się przewidzieć przed ich rozpoczęciem. W dalszej części wykładu skupimy się na sortowaniu liczb naturalnych, ale inne dane (np. znaki) też mogą być porządkowane przez komputer.

[1] Dla zainteresowanych: pomija się tu kwestię *pamięci wirtualnej*.

Sortowanie

Przykładowe algorytmy

Sortowanie zostanie omówione na tym wykładzie na przykładzie trzech algorytmów sortujących tablicę jednowymiarową. Wszystkie te algorytmy stosują sortowanie w miejscu i wewnętrzne. Dwa oferują sortowanie stabilne, a trzeci (sortowanie przez wybór) można zmodyfikować tak, aby też tak sortował. Ich efektywność dla tablic o małej liczbie elementów jest również porównywalna. Wszystkie zostaną omówione w wersji, która sortuje tablicę rosnąco/niemalejąco, ale zostaną podane informacje, jak przekształcić je, aby tablica była porządkowana odwrotnie.

Sortowanie przez wybór

Opis algorytmu

Algorytm sortowania przez wybór (ang. *selection sort*) jest stosunkowo prosty. Został on opracowany na bazie algorytmu wyszukiwania minimum w tablicy nieposortowanej (nieuporządkowanej). Jego działanie opiera się na spostrzeżeniu, że w tablicy posortowanej najmniejsza wartość powinna się znaleźć w pierwszym elemencie. Zatem należy znaleźć element o takiej wartości w całej tablicy i jeśli nie jest on pierwszym jej elementem, to jego wartość należy zamienić miejscami z wartością elementu pierwszego. To postępowanie należy powtórzyć wyłączając pierwszy element i porządkując drugi. Te powtórzenia powinny trwać tak długo, aż zostaną uporządkowane wartości dwóch ostatnich elementów w tablicy.

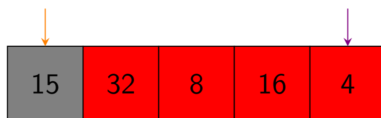
Sortowanie przez wybór

Symulacja

Następny slajd pokazuje wizualizację sortowania tablicy o pięciu elementach zawierającej liczby naturalne przy pomocy algorytmu sortowania przez wybór. Pomarańczowa strzałka pokazuje na element, którego wartość jest bieżąco porządkowana, a fioletowa, na element, który ma najmniejszą wartość, spośród tych, które należą do fragmentu tablicy zawierającego porządkowany element i wszystkie elementy leżące na prawo od niego. Element porządkowany jest dodatkowo zaznaczony na szaro. Element oznaczony na zielono, to ten, którego wartość została już uporządkowana, a czerwony, to ten, którego wartość musi jeszcze być uporządkowana. W wizualizacji pominięto kroki wyszukiwania elementu o najmniejszej wartości. Są one podobne do tych w algorytmie znajdowania wartości minimalnej w nieuporządkowanej tablicy.

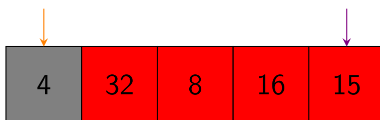
Sortowanie przez wybór

Symulacja



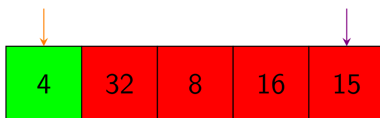
Sortowanie przez wybór

Symulacja



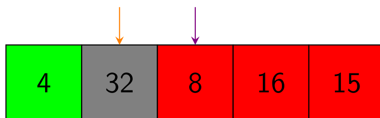
Sortowanie przez wybór

Symulacja



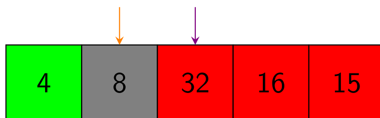
Sortowanie przez wybór

Symulacja



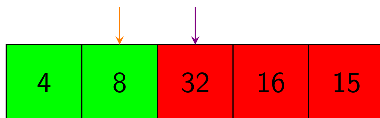
Sortowanie przez wybór

Symulacja



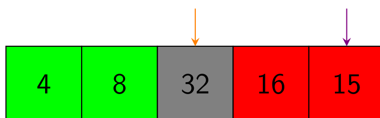
Sortowanie przez wybór

Symulacja



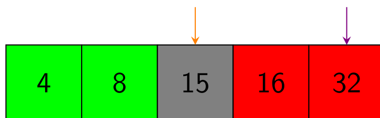
Sortowanie przez wybór

Symulacja



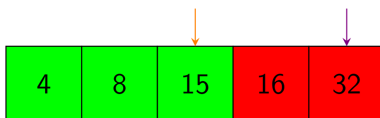
Sortowanie przez wybór

Symulacja



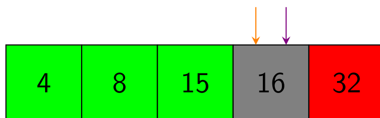
Sortowanie przez wybór

Symulacja



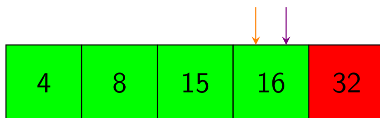
Sortowanie przez wybór

Symulacja



Sortowanie przez wybór

Symulacja



Sortowanie przez wybór

Symulacja

4	8	15	16	32

Sortowanie przez wybór

Implementacja

```
void selection_sort(int array[])
{
    int i,j;

    for(i=0; i<NUMBER_OF_ELEMENTS-1; i++) {
        int min = i;
        for(j=i+1; j<NUMBER_OF_ELEMENTS; j++)
            if(array[min]>array[j])
                min = j;
        if(min!=i)
            swap(&array[min],&array[i]);
    }
}
```

Sortowanie przez wybór

Komentarz do implementacji

Licznik zewnętrznej pętli `for` wyznacza kolejny element tablicy, którego wartość powinna być uporządkowana (odpowiednik pomarańczowej strzałki z wizualizacji). Wewnętrzna pętla `for` realizuje algorytm wyszukiwania najmniejszej wartości we fragmencie tablicy zawierającym elementy, które są położone na prawo od elementu porządkowanego. Proszę zwrócić uwagę, że tym razem nie tyle interesuje nas sama wartość, co jej położenie, a więc indeks elementu, który ją zawiera (odpowiednik fioletowej strzałki). Jeśli po zakończeniu pętli wewnętrznej okaże się, że zmienna `min` wskazuje na inny element niż `i` (strzałki się nie pokrywają), to wartości elementów indeksowanych przez te zmienne są zamieniane miejscami. Implementacja funkcji `swap()` została podana na poprzednim wykładzie. Aby ten algorytm sortował stabilnie wystarczy ostrą nierówność w pierwszej instrukcji warunkowej zamienić na nieostrą. Jeśli zmienimy jej znak, to funkcja będzie sortowała malejąco/nierosnąco. Aby podnieść efektywność tego algorytmu można zmodyfikować go tak, aby porządkował tablicę „z obu końców" szukając jednocześnie wartości minimalnej i maksymalnej.

Funkcja swap()

Inna implementacja (ciekawostka)

```
void swap(int *first, int *second)
{
    if(first!=second) {
        *first ^= *second;
        *second ^= *first;
        *first ^= *second;
    }
}
```

Przedstawiona funkcja dokonuje wymiany wartości dwóch zmiennych przekazanych jej przez wskaźniki, ale bez użycia dodatkowej zmiennej. Jest to możliwe dzięki wykorzystaniu odwracalności działania operatora ^ (xor). Ta metoda nie działa jednak, gdy jest stosowana tylko dla jednej zmiennej. Co więcej, w takim przypadku spowoduje ona wyzerowanie tej zmiennej. Dlatego w instrukcji warunkowej funkcja sprawdza, czy parametry wskaźnikowe zawierają różne adresy. Jeśli byłby to ten sam adres, to nie zostaną wykonane żadne operacje.

Funkcja `swap()`

Inna implementacja (ciekawostka) - kontynuacja

Zaletą takiej implementacji funkcji `swap()` jest mniejsze zużycie pamięci, ale w porównaniu z „klasyczną” implementacją jest ona wolniejsza i nie daje się wprost zastosować do innych typów zmiennych (np. `double`). Istnieją też inne warianty tego rozwiązania, stosujące inne operatory, z też podobnymi ograniczeniami.

Sortowanie przez wstawianie

Opis algorytmu

Innym rozwiązaniem problemu sortowania tablic jednowymiarowych jest algorytm sortowania przez wstawianie (ang. *insertion sort*). Działanie tego algorytmu przypomina zachowanie amatorów gier karcianych, którzy wstawiają do swojej tali nowo pobraną kartę, przesuwające te, które już trzymają w ręku. Algorytm ten rozpoczyna porządkowanie tablicy od jej drugiego elementu, porównując jego wartość z tą, która jest w pierwszym elemencie. Jeśli ta ostatnia jest większa, to przesuwa ją (kopiuje w prawo), a w jej miejsce wstawia tę z drugiego elementu. Podobnie postępuje dla wartości z trzeciego elementu. Porównuje ją z wartościami elementów leżących na lewo od niego. Tak długo, jak są one większe przesuwa je w prawo. Jeśli napotka element zawierający wartość mniejszą lub równą od tej, którą porządkuje, to wstawia tę ostatnią przed tym elementem. Czynności te powtarza dla wartości pozostałych elementów tablicy tak długo, aż wszystkie uporządkuje.

Sortowanie przez wstawianie

Symulacja

Kolejny slajd zawiera wizualizację porządkowania tablicy jednowymiarowej przez algorytm sortowania przez wstawianie. Na szaro zaznaczony jest element, z którego pobierana jest porządkowana wartość. Jest ona przesuwana nad elementami z których wartościami jest porównywana. Proszę zauważyć, że zamiast typowych wymian wartości elementów, jak w algorytmie sortowania przez wybór jest tu stosowane kopiowanie w prawo wartości większych od tej, która jest porządkowana. Takie operacje można nazwać „półwymianami”. Podobnie jak w poprzedniej symulacji, elementy oznaczone na zielono są uporządkowane, a te zaznaczone na czerwono wymagają jeszcze uporządkowania.

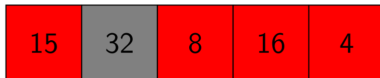
Sortowanie przez wstawianie

Symulacja

15	32	8	16	4

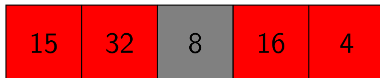
Sortowanie przez wstawianie

Symulacja



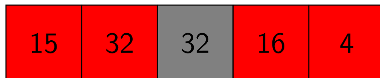
Sortowanie przez wstawianie

Symulacja



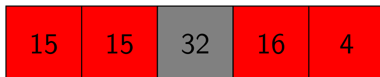
Sortowanie przez wstawianie

Symulacja



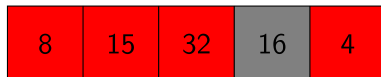
Sortowanie przez wstawianie

Symulacja



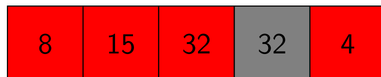
Sortowanie przez wstawianie

Symulacja



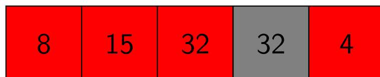
Sortowanie przez wstawianie

Symulacja



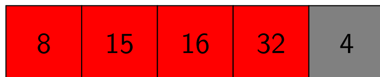
Sortowanie przez wstawianie

Symulacja



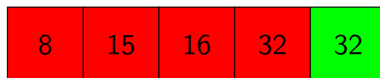
Sortowanie przez wstawianie

Symulacja



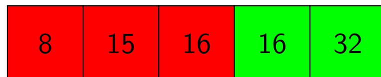
Sortowanie przez wstawianie

Symulacja



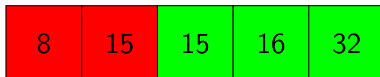
Sortowanie przez wstawianie

Symulacja



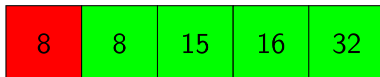
Sortowanie przez wstawianie

Symulacja



Sortowanie przez wstawianie

Symulacja



Sortowanie przez wstawianie

Symulacja

4	8	15	16	32

Sortowanie przez wstawianie

Implementacja

```
void insertion_sort(int array[])
{
    int i;
    for(i=1;i<NUMBER_OF_ELEMENTS;i++) {
        int key = array[i];
        int j = i-1;
        while(j>=0&&array[j]>key) {
            array[j+1]=array[j];
            j--;
        }
        array[j+1]=key;
    }
}
```

Sortowanie przez wstawianie

Komentarz do implementacji

Implementacja tego algorytmu jest bardziej skomplikowana, niż implementacja sortowania przez wybór. Licznik pętli `for` wyznacza element, którego wartość ma zostać uporządkowana. W pętli `while` ta wartość porównywana jest z wartościami elementów leżących na lewo od niej w tablicy. Jeśli są one większe, to funkcja przesuwa je w prawo o jeden element. Dopiero wtedy gdy zostanie napotkana wartość mniejsza lub równa tej porządkowanej, to ta ostatnia jest wstawiana przez elementem zawierającym taką wartość. W przypadku gdy pętla wewnętrzna zatrzyma się „poza" tablicą, to wartość ze zmiennej `key` jest kopiowana do pierwszego elementu tablicy. Aby ta funkcja sortowała malejąco/nierosnąco wystarczy zmienić znak w wyrażeniu `array[j]>key`.

Sortowanie przez wstawianie

Porównanie z sortowaniem przez wybór

Można zaobserwować, że sortowanie przez wstawianie jest pod pewnymi względami przeciwieństwem sortowania przez wybór:

- sortowanie przez wybór dopasowuje wartość do elementu, a sortowanie przez wstawianie element do wartości,
- sortowanie przez wybór przegląda tablicę „w prawo”, a sortowanie przez wstawianie „w lewo”.

Sortowanie bąbelkowe

Opis algorytmu

Istnieje kilka wersji algorytmu sortowania bąbelkowego (ang. *bubble sort*). W jednej z nich algorytm ten przegląda tablicę od końca, po jednym elemencie i porównuje wartości bieżącego elementu z jego sąsiadem z lewej strony. Jeśli nie znajdują się one w oczekiwanym porządku, to algorytm zamienia je miejscami. Tablica jest przeglądana w ten sposób wielokrotnie, aż do całkowitego jej uporządkowania. Po każdym powtórzeniu przeglądania, co najmniej jeden element z lewej strony tablicy zyskuje prawidłową wartość i przy kolejnej iteracji jest już pomijany.

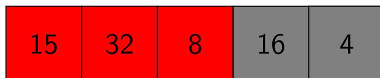
Sortowanie bąbelkowe

Symulacja

Kolejny slajd przedstawia symulację sortowania tablicy jednowymiarowej przy pomocy algorytmu bąbelkowego. Szary kolor oznacza parę elementów, których wartości są bieżąco porównywane. Czerwonym kolorem zaznaczone są elementy, które jeszcze będą sprawdzane, a zielonym te, które zostały już uporządkowane.

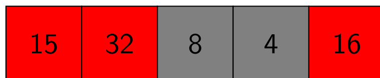
Sortowanie bąbelkowe

Symulacja



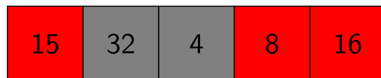
Sortowanie bąbelkowe

Symulacja



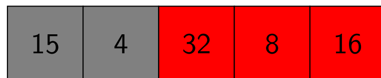
Sortowanie bąbelkowe

Symulacja



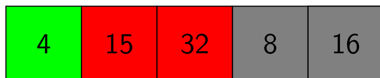
Sortowanie bąbelkowe

Symulacja



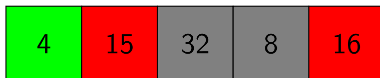
Sortowanie bąbelkowe

Symulacja



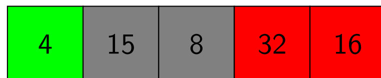
Sortowanie bąbelkowe

Symulacja



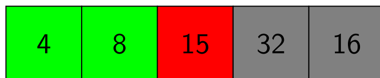
Sortowanie bąbelkowe

Symulacja



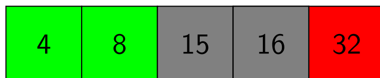
Sortowanie bąbelkowe

Symulacja



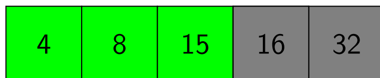
Sortowanie bąbelkowe

Symulacja



Sortowanie bąbelkowe

Symulacja



Sortowanie bąbelkowe

Symulacja

4	8	15	16	32

Sortowanie bąbelkowe

Implementacja

```
void bubble_sort(int array[])
{
    int i,j;
    for(i=0; i<NUMBER_OF_ELEMENTS; i++)
        for(j=NUMBER_OF_ELEMENTS-1; j>i; j--)
            if(array[j-1]>array[j])
                swap(&array[j-1],&array[j]);
}
```

Sortowanie bąbelkowe

Komentarz do implementacji

Wewnętrzna pętla `for` realizuje przeglądanie tablicy od końca i porównywanie par sąsiadujących z sobą elementów. Zakres tego przeglądania w kolejnych iteracjach ogranicza licznik pętli zewnętrznej. Aby posortować tablicę w odwrotnym porządku wystarczy zmienić znak nierówności w instrukcji warunkowej.

Sortowanie - porównanie algorytmów

Wszystkie przedstawione algorytmy mają zbliżone cechy, poza stabilnością, która w przypadku algorytmu sortowania przez wybór nie jest domyślnie zapewniona, ale może być łatwo zrealizowana. To co je różni, to wydajność. Choć z teorii złożoności obliczeniowej, która zajmuje się efektywnością algorytmów, wynika, że czas działania wszystkich tych algorytmów jest proporcjonalny do kwadratu liczby elementów tablicy[2], to występują między nimi znaczące różnice indywidualne. Algorytm sortowania bąbelkowego wypada w tym zestawieniu najgorzej. Nie jest to najgorszy możliwy algorytm sortowania, bo takim jest *BogoSort*, który polega na losowym przestawianiu wartości elementów i sprawdzaniu, czy są w odpowiedniej kolejności. Niemniej sortowanie bąbelkowe uważane jest za antyprzykład sortowania, z uwagi na dużą liczbę kosztownych wymian wartości elementów, jaką przeprowadza.

[2] Dokładniejsza analiza złożoności czasowej algorytmów sortowania, której wartość może zależeć od konfiguracji początkowej wartości elementów tablicy, będzie prowadzona na zajęciach z Algorytmów i Struktur Danych.

Sortowanie - porównanie algorytmów

Kontynuacja

Algorytmu sortowania bąbelkowego nie należy stosować w praktyce. Niektórzy informatycy proponują go w ogóle nie nauczać. Różnice w efektywności dwóch pozostałych algorytmów nie są wielkie, jednak częściej szybciej działa sortowanie przez wstawianie niż sortowanie przez wybór. Zależy to jednak od zastosowania. Jeśli częściej wykonywane są porównania elementów, niż zamiana ich wartości, to efektywniej działa sortowanie przez wybór. W przeciwnym przypadku krócej tablicę sortuje algorytm sortowania przez wstawianie.

Wyszukiwanie binarne

Opis algorytmu

Okazuje się, że jeśli tablica jest uporządkowana (rosnąco/niemalejąco), to można zastosować dla niej algorytm wyszukiwania wartości znacznie szybszy niż w przypadku tablic nieposortowanych. Tym algorytmem jest algorytm wyszukiwania binarnego, nazywany także algorytmem wyszukiwania połówkowego. Jego działanie składa się z kilku kroków. Pierwszym jest wyznaczenie elementu środkowego tablicy. Jeśli tablica ma parzystą liczbę elementów, to zostaje nim element położony na lewo od jej środka. Wartość tego elementu jest porównywana z poszukiwaną wartością. Jeśli są one sobie równie, to zwracany jest indeks tego elementu i algorytm kończy działanie. Jeśli natomiast wartość elementu środkowego jest większa od wartości szukanej, to znaczy, że jeśli ta ostatnia w ogóle występuje w tablicy, to będzie znajdowała się we fragmencie tablicy położonym na lewo od elementu środkowego. Jeśli wynik porównania będzie odwrotny, to znaczy, że szukana wartość może znajdować się we fragmencie położonym na prawo do środka. Algorytm powtarza to działanie dla wyznaczonego fragmentu tablicy.

Wyszukiwanie binarne

Opis algorytmu - kontynuacja

Algorytm wyszukiwania binarnego tak długo powtarza opisane na poprzednim slajdzie czynności, aż znajdzie szukaną wartość, jeśli występuje ona w tablicy, lub gdy wyznaczony fragment tablicy będzie tak mały, że nie będzie zawierał żadnych jej elementów. Ten ostatni przypadek występuje wtedy, gdy w tablicy nie ma szukanej wartości. To, że algorytm się zawsze zatrzyma można wywnioskować po tym, że po każdym powtórzeniu jego działania fragment tablicy, który trzeba przeszukać staje się krótszy o około połowę. Jest to też przyczyną tego, że ten algorytm jest szybszy od algorytmu wyszukiwania liniowego. W przypadku wyszukiwania binarnego w każdym kroku liczba elementów, które trzeba sprawdzić redukuje się o około połowę, a w przypadku wyszukiwania liniowego tylko o jeden. Choć opis algorytmu wyszukiwania binarnego jest stosunkowo prosty, to jego szczegóły nastręczają problemów w implementacji. Według Jona Bentley'a algorytm ten był znany już w 1946 roku, ale jego poprawna realizacja pojawiła się dopiero w 1962.

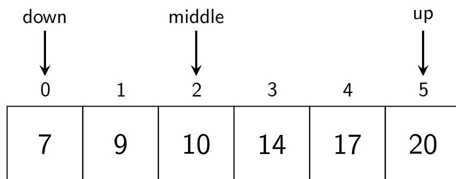
Wyszukiwanie binarnego

Symulacja

Kolejny slajd zawiera symulację działania algorytmu wyszukiwania połówkowego dla posortowanej tablicy liczby naturalnych o sześciu elementach. Strzałki oznaczone jako `down` i `up` wyznaczają odpowiednio element początkowy i element końcowy fragmentu tablicy, który ma być w danej iteracji zbadany. Początkowo ten fragment obejmuje całą tablicę, ale wraz z wykonywaniem kolejnych powtórzeń przez algorytm będzie się zmniejszał. Strzałka `middle` wskazuje na środkowy element przeszukiwanego obszaru. Szukana wartość została umieszczona po lewej stronie slajdu w okręgu.

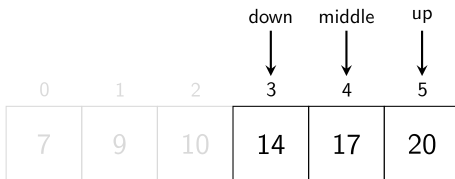
Wyszukiwanie binarne

Symulacja



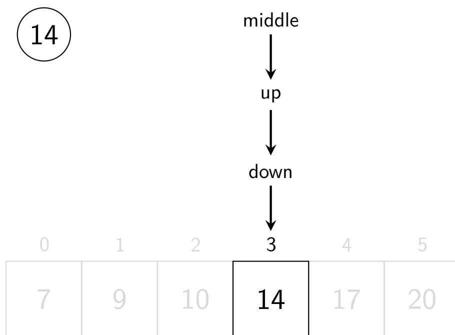
Wyszukiwanie binarne

Symulacja



Wyszukiwanie binarne

Symulacja



Wyszukiwanie binarne

Implementacja

```
int binary_search(int array[], int value)
{
    int down=0, up=NUMBER_OF_ELEMENTS-1;
    while(down<=up) {
        int middle = down+((up-down)/2);
        if(array[middle]==value)
            return middle;
        if(array[middle]<value) {
            down = middle + 1;
            continue;
        }
        if(array[middle]>value)
            up = middle - 1;
    }
    return -1;
}
```

Wyszukiwanie binarne

Komentarz do implementacji

Zmienne wyznaczające początek, koniec i środek przeszukiwanego obszaru tablicy są tak samo nazwane jak strzałki z symulacji. Pierwszym elementem zwracającym uwagę w tej implementacji jest wyrażenie wyznaczające indeks elementu środkowego przeszukiwanego fragmentu tablicy. Dlaczego nie policzyć go jako średniej arytmetycznej zmiennych `down` i `up`? Niestety takie rozwiązanie, przy bardzo dużych tablicach, zawierających ponad miliard elementów może doprowadzić do przekroczenia zakresu typu `int` przy dodawaniu tych zmiennych, co z kolei spowoduje błędne wyznaczenie takiego indeksu. Wersja tu zaprezentowana jest jedną z możliwych, które pozbawione są tej usterki. Drugi ważny element, to wykrywanie, że w tablicy nie ma szukanej wartości. Dzieje się tak, kiedy wartość zmiennej `down` będzie większa od wartości zmiennej `up` i oznacza, że przeszukiwany fragment tablicy nie ma zawiera żadnych jej elementów. W pętli `while` sprawdzany jest odwrotny warunek, czyli czy fragment zawiera choć jeden element.

Wyszukiwanie binarne

Komentarz - kontynuacja

Ostatni ciekawy element to wyłączanie sprawdzonego elementu środkowego z wyliczania granic przeszukiwanego fragmentu - skoro został już sprawdzony, to na pewno szukanej wartości w nim nie ma, a jego pozostawienie w przeszukiwanym fragmencie mogłoby spowodować błędne działanie algorytmu. Instrukcja `continue` została zastosowana, aby uniknąć sprawdzania warunku, który na pewno nie byłby spełniony. Jeśli szukana wartość nie występuje w tablicy, to funkcja zwraca wartość `-1`, natomiast jeśli szukana wartość powtarza się w tablicy, to algorytm binarny gwarantuje znalezienie jej wystąpienia, ale niekoniecznie będzie to pierwsze wystąpienie.

Wartość minimalna i maksymalna w posortowanej tablicy

Wyszukiwanie minimalnej i maksymalnej wartości w tablicy posortowanej staje się wręcz trywialnie proste. Jeśli tablica jest uporządkowana rosnąco/niemalejąco, to wartość minimalna znajduje się w jej pierwszym elemencie, a maksymalna w ostatnim. Jeśli tablica jest posortowana odwrotnie, to te wartości też umieszczone są w odwrotnej kolejności w takiej tablicy.

Podziękowania

Składam podziękowania dla dra inż. Grzegorza Łukawskiego i mgra inż. Leszka Ciopińskiego za udostępnienie materiałów, których fragmenty zostały wykorzystane w tym wykładzie.

Pytania

?

KONIEC

Dziękuję Państwu za uwagę.