Podstawy Programowania 1

Algorytmy dla tablic jednowymiarowych

Arkadiusz Chrobot

Katedra Systemów Informatycznych

20 listopada 2024

Plan

Wstęp

Na tym wykładzie zostanie przedstawionych kilka podstawowych algorytmów dla tablic jednowymiarowych. W szczególności omówione zostanie wyszukiwanie danych oraz sortowanie (porządkowanie). Należą one do najczęściej wykonywanych operacji przez systemy komputerowe. Są to na tyle istotne zagadnienia, że D. E. Knuth poświęcił im cały trzeci tom książki pt. „Sztuka programowania”. Obie operacje dotyczą wielu typów struktur danych, ale na bieżącym wykładzie będą opisane w kontekście tej jednej, wymienionej na początku.

Wyszukiwanie wartości minimalnej

Algorytm

W niektórych zagadnieniach należy znaleźć wartość minimalną w nieuporządkowanej tablicy. Algorytm jej wyszukiwania jest stosunkowo prosty:

1. Zapamiętaj w osobnej zmiennej wartość pierwszego elementu tablicy, jako wartość minimalną.
2. Przeglądaj kolejne elementy tablicy; jeśli któryś z nich ma wartość mniejszą od tej w zmiennej, to ją w niej umieść; teraz to będzie wartość najmniejsza.
3. Jeśli odwiedzone zostały wszystkie elementy w tablicy, to wartość najmniejsza znajduje się w zmiennej.

Wyszukiwanie wartości minimalnej

Implementacja

Poniższa funkcja implementuje ten algorytm dla tablicy o liczbie elementów określonej wartością stałej `NUMBER_OF_ELEMENTS`:

```
int find_min(int *array)
{
    int min;
    unsigned int i;
    min=array[0];
    for(i=1; i<NUMBER_OF_ELEMENTS; i++)
        if(min>array[i])
            min=array[i];
    return min;
}
```

Wyszukiwanie wartości maksymalnej

Implementacja

Algorytm wyszukiwania wartości maksymalnej w tablicy jest bardzo podobny do wyszukiwania wartości minimalnej - wystarczy tylko zmienić kilka wyrazów. Kod funkcji go implementującej różni się od kodu funkcji szukającej wartości minimalnej nazwą funkcji, zmiennej oraz znakiem w warunku instrukcji warunkowej:

```
int find_max(int *array)
{
    int max;
    unsigned int i;
    max=array[0];
    for(i=1; i<NUMBER_OF_ELEMENTS; i++)
        if(max<array[i])
            max=array[i];
    return max;
}
```

Wyszukiwanie ekstremów

Łatwo zauważyć, że w przypadku, gdy potrzebujemy zarówno wartości minimalnej, jak i maksymalnej, a nie tylko jednej z nich, korzystniej będzie szukać ich obu przeglądając tablicę tylko raz, a oba wyniki zwracać przez parametry funkcji:

```
void find_exterme_values(int array[], int *min, int *max)
{
    unsigned int i;
    *max = *min = array[0];
    for(i=1; i<NUMBER_OF_ELEMENTS; i++) {
        if(*min>array[i])
            *min=array[i];
        if(*max<array[i])
            *max=array[i];
    }
}
```

Wyszukiwanie liniowe

Algorytm

Często spotykanym problemem jest lokalizacja wartości w nieuporządkowanej tablicy. Jest wiele odmian tego problemu. Nas będzie interesowała ta, w której należy podać indeks pierwszego elementu od początku tablicy, który zawiera szukaną wartość. Algorytm rozwiązania tego problemu jest prosty: należy przeglądać kolejne elementy tablicy, aż do napotkania takiego, który zawiera szukaną wartość. Wówczas należy zwrócić wartość jego indeksu. Alternatywnie, jeśli żaden element tablicy nie zawiera takiej wartości, to należy zwrócić określoną wartość, która ten fakt zasygnalizuje np. `-1`.

Wyszukiwanie liniowe

Implementacja

Poniższa funkcja implementuje przedstawiony algorytm:

```
int find_value_index(int array[], int value)
{
    unsigned int i;
    for(i=0; i<NUMBER_OF_ELEMENTS; i++) {
        if(array[i]==value)
            return i;
    }
    return -1;
}
```

Sortowanie

Rodzaje sortowania

Powstało wiele algorytmów sortowania, które różnią się wieloma cechami. Sortowanie wewnętrzne przeprowadzane jest w całości w pamięci operacyjnej komputera. Sortowanie zewnętrzne oznacza porządkowanie danych umieszczonych w pamięci masowej. Sortowanie stabilne zachowuje początkowy względny porządek jednakowych wartości. Np. jeśli w tablicy są dwie liczby 8, które oznaczymy jako 8' i 8", to po posortowaniu tej struktury danych 8' nadal będzie przed 8". W przypadku sortowania niestabilnego, 8" znajdzie się przed 8'. Jeżeli sortowana jest tablica liczb, to taka cecha ma niewielkie znaczenie, ale dla tablic o elementach innych typów 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.

Sortowanie

Przykładowy algorytm

Sortowanie zostanie przedstawione na tym wykładzie na przykładzie jednego algorytmu sortującego tablicę jednowymiarową. Wykonuje on sortowanie w miejscu i wewnętrzne, ale niestabilne, choć można go zmodyfikować, tak aby sortował stabilnie. Efektywność tego algorytmu nie jest duża. Używając pojęć z dziedziny *złożoności obliczeniowej*, czas jego wykonania można opisać jako proporcjonalny do kwadratu liczby elementów tablicy, którą sortuje. Jego niewątpliwą zaletą jest to, że jest stosunkowo prosty. Zostanie on przedstawiony w wersji, która sortuje tablicę rosnąco/niemalejąco, ale zmiana kierunku sortowania nie jest skomplikowana.

Sortowanie przez wybór

Opis algorytmu

Algorytm sortowania przez wybór (ang. *selection sort*) został 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.

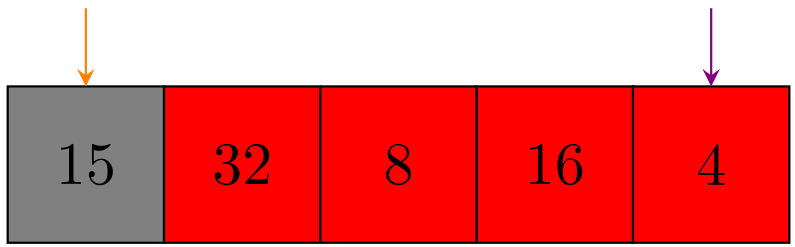
Sortowanie przez wybór

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.

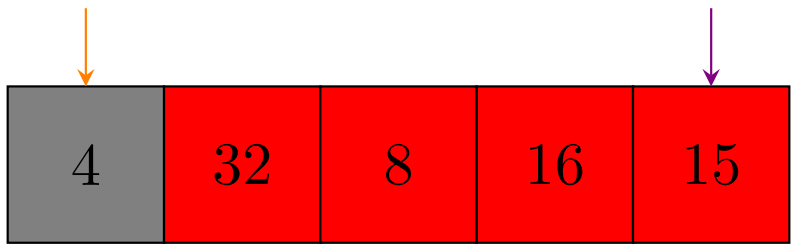
Sortowanie przez wybór

Symulacja



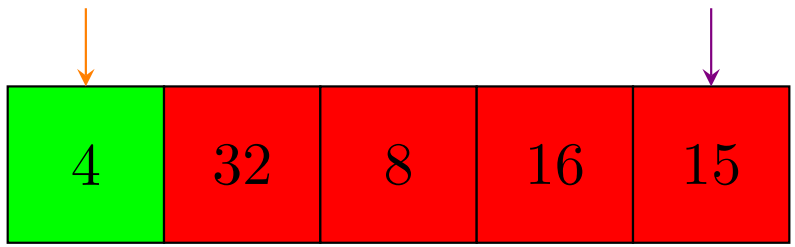
Sortowanie przez wybór

Symulacja



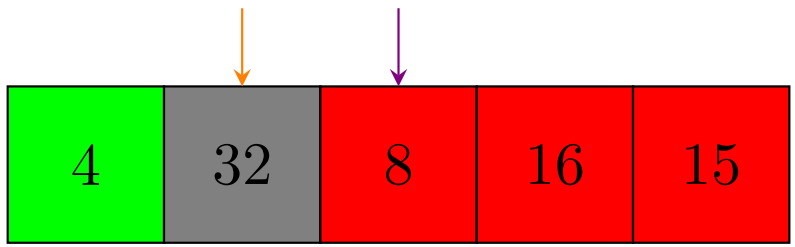
Sortowanie przez wybór

Symulacja



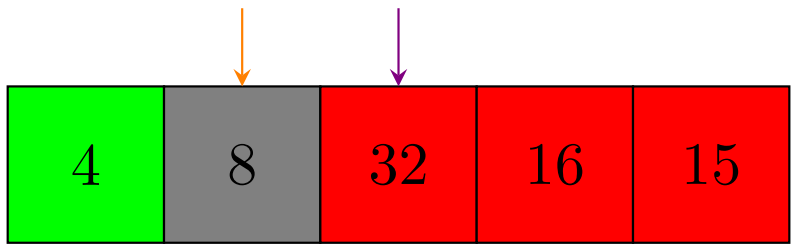
Sortowanie przez wybór

Symulacja



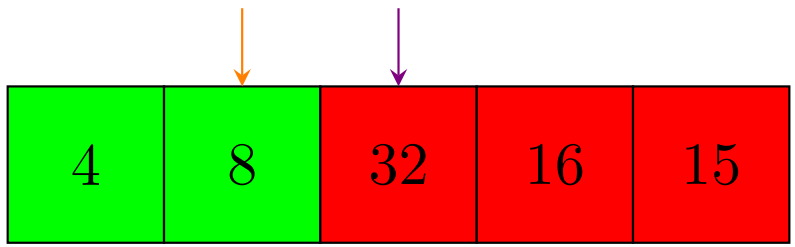
Sortowanie przez wybór

Symulacja



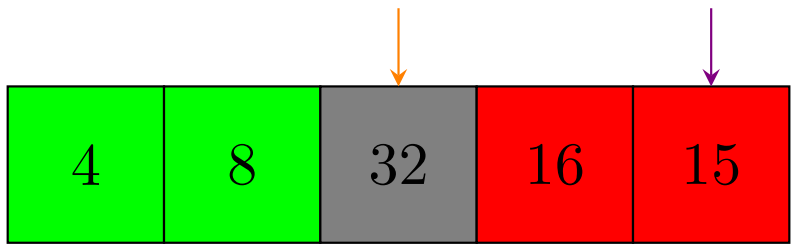
Sortowanie przez wybór

Symulacja



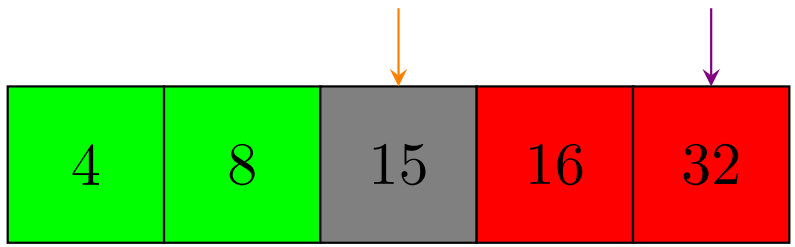
Sortowanie przez wybór

Symulacja



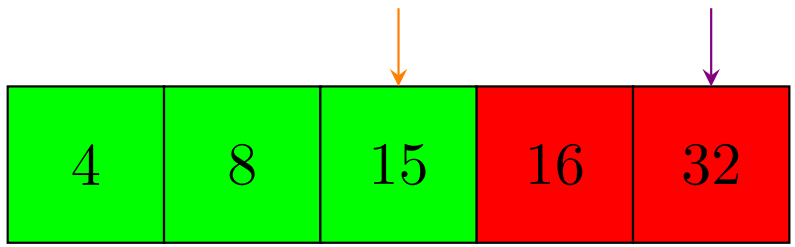
Sortowanie przez wybór

Symulacja



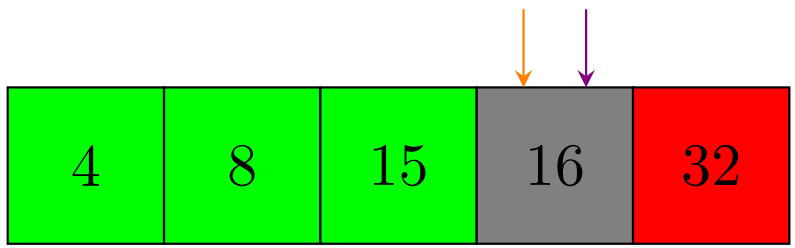
Sortowanie przez wybór

Symulacja



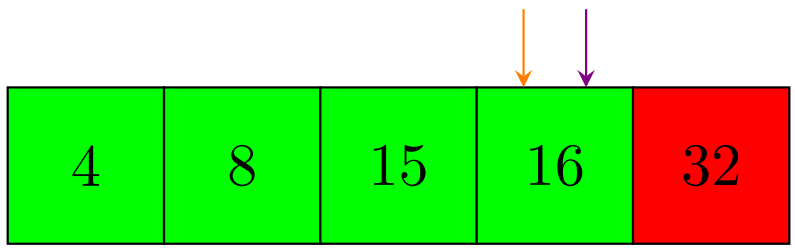
Sortowanie przez wybór

Symulacja



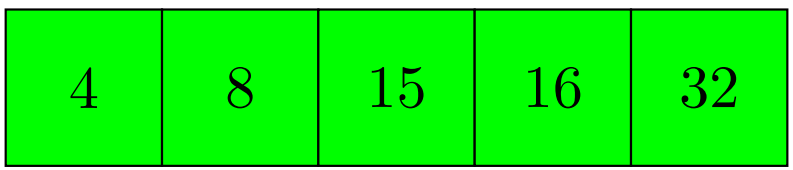
Sortowanie przez wybór

Symulacja



Sortowanie przez wybór

Symulacja



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. Jeśli zmienimy znak nierówności w instrukcji warunkowej, to funkcja `selection_sort()` będzie sortowała malejąco/nierosnąco. Aby podnieść jej efektywność można ją zmodyfikować tak, aby porządkowała 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.

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.

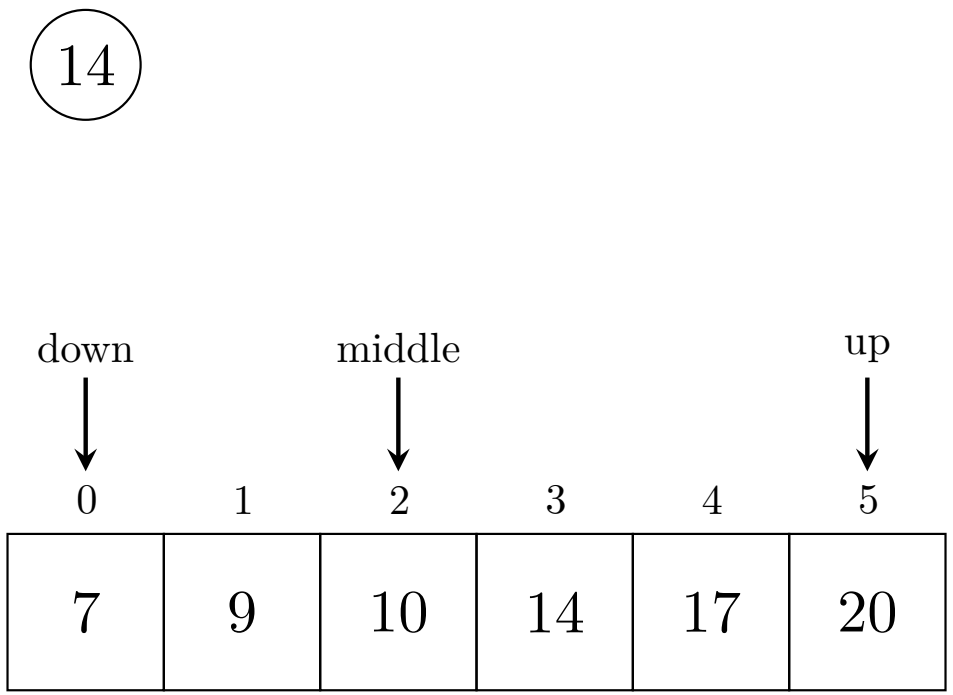
Wyszukiwanie binarnego

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.

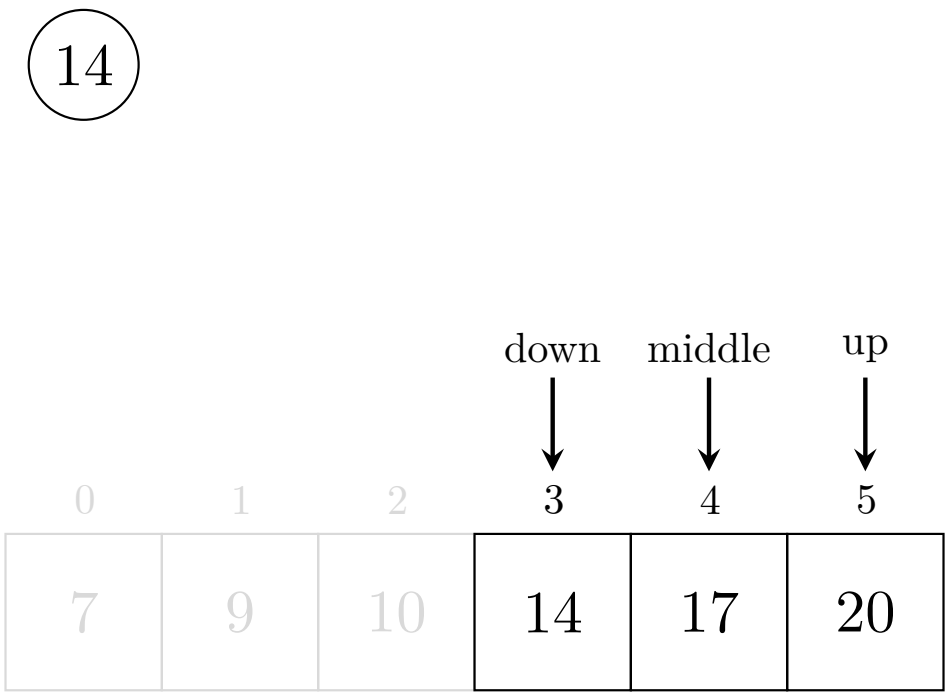
Wyszukiwanie binarne

Symulacja



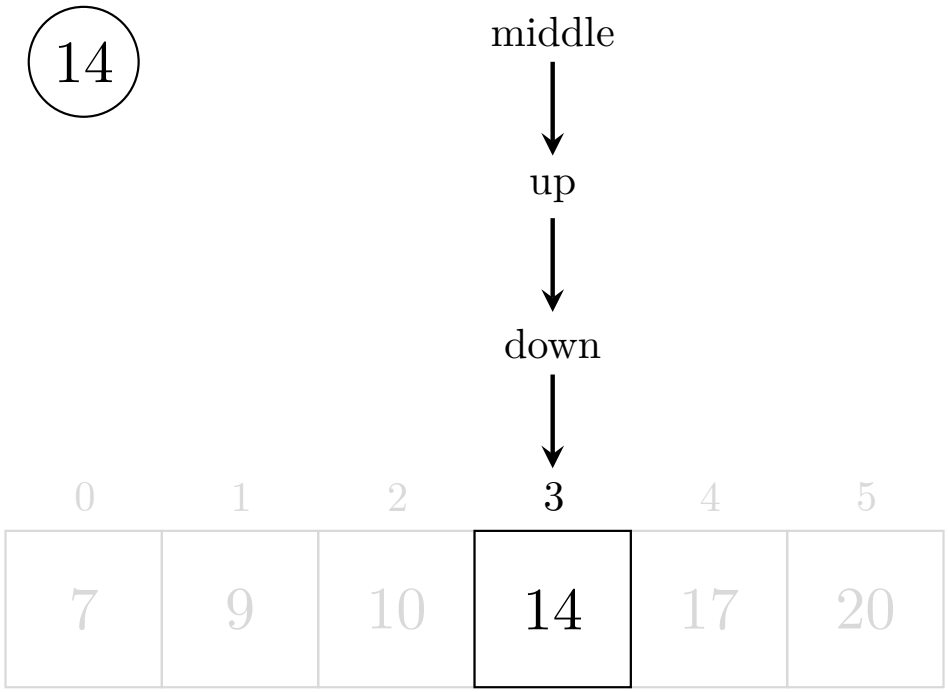
Wyszukiwanie binarne

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;
        }
        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 wykonania instrukcji `up = middle - 1;`, która powinna być uruchomiona tylko wtedy, gdy żaden z dwóch poprzedzających ją warunków nie jest 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.