

Vyhledávání

Vyhledávání je další velmi důležitou a často se vyskytující úlohou. Při ní máme zadanou nějakou množinu (multimnožinu) prvků a cílem je nalézt mezi nimi takový prvek, který má danou hodnotu vyhledávacího klíče, anebo případně zjistit, že takový prvek mezi nimi není.

Máme n prvků

$$a_0, a_1, \dots, a_{n-1}$$

a dále máme danou hodnotu vyhledávacího klíče, označme ji x . Hledáme prvek a_i , pro který platí

$$a_i.key = x \quad .$$

Poznámka: V následujících popisovaných algoritmech vyhledávání je pro zjednodušení jejich popisu v operacích srovnání vynechán zápis vyhledávacího klíče – například místo srovnání $a_i.key = x$ nebo $a[i].key = x$ budeme psát $a_i = x$.

Metody vyhledávání

- Vyhledávání v lineárních datových strukturách
- Vyhledávací stromy
- Číslíkové vyhledávání
- Hašovací tabulky

Vyhledávání v lineárních datových strukturách

Sekvenční vyhledávání

Mezi nejjednodušší případy vyhledávání patří vyhledávání v lineární datové struktuře, tj. v poli nebo v seznamu. Přepokládáme přitom, že prvky jsou v ní uloženy v libovolném pořadí (nesetříděné). Není zde jiný způsob, než prvky postupně procházet (zpravidla od začátku směrem ke konci) a každý srovnat s hledanou hodnotou. Počet srovnání se přitom pohybuje od 1, jestliže hledaný prvek je hned první, po n , jestliže hledaný prvek je až poslední anebo hledaný prvek mezi prohledávanými prvky není obsažen (nebyl nalezen). Tedy

$$\text{průměrný počet srovnání (je-li prvek nalezen)} = \frac{1+n}{2} .$$

$$\text{maximální počet srovnání} = n .$$

Sekvenční vyhledávání v lineární datové struktuře má časovou složitost $\Theta(n)$.

Pseudokód:

```
SequentialSearch(A, n, x)
```

```
    i ← 0
```

```
    while i < n
```

```
        if A[i] = x
```

```
            return i
```

```
        i ← i + 1
```

```
    return n
```

Je zřejmé, že v cyklu se kromě srovnání, zda prvek je roven hledané hodnotě, rovněž vždy dělá srovnání, zda už nejsme na konci pole (podmínka $i < n$). To lze odstranit a vyhledávání urychlit tak, že pole s prvky vytvoříme o jeden prvek větší a jako poslední prvek v poli před každým prohledáváním dáme hledanou hodnotu (užívá se pro ni označení zarážka). Pak cyklus hledání vždy nalezne hledaný prvek a bude jen zapotřebí zjistit, zda tento prvek byl nalezen ještě před zarážkou.

Pseudokód:

SequentialSearchSentinel(A, n, x)

A[n] ← x

i ← 0

while true

if A[i] = x

return i

i ← i + 1

Srovnání obou metod

Vygenerováno 100 000 náhodných čísel, hledáno 20 000 náhodně vybraných čísel z nich:

Sekvenční hledání	Čas v sekundách
Bez zářázky	2.89
Se zářázkou	2.08

Binární vyhledávání v setříděném poli

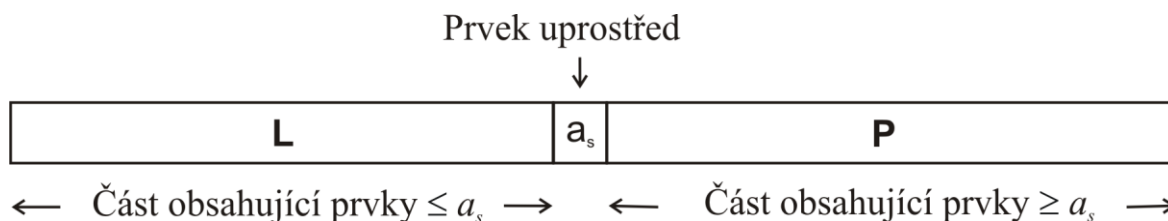
V případě pole je pro vyhledávání mnohem příznivější případ, když prvky jsou v něm uspořádány (seřazeny) dle velikosti vyhledávacího klíče. Tj. platí pro ně

$$a_0 \leq a_1 \leq \dots \leq a_{n-2} \leq a_{n-1} \quad .$$

Zde se dá použít algoritmus binárního vyhledávání, často také nazývaný vyhledávání půlením intervalu.

Popis algoritmu

Vezmeme prvek, který je v poli uprostřed (je-li počet prvků sudý, jsou uprostřed dva prvky - zde vezmeme jeden z nich, při implementaci metody to zpravidla bývá ten levý), označme jeho index s .



Následně provedeme srovnání hledané x hodnoty s hodnotou středního prvku a_s :

- Nejprve srovnáme, zda je $x < a_s$:

Pokud ano, pak zřejmě hledaný prvek, pokud v poli vůbec je, musí být v části L , jež je nalevo od středního prvku a_s . Je-li část L neprázdná

(obsahuje aspoň jeden prvek), rekurzivně na ni provedeme stejný postup. Je-li už prázdná, vyhledávání neúspěšně končí. Hledaný prvek není v poli obsažen.

- Pokud neplatí $x < a_s$, uděláme další srovnání. Srovnáme, zda je $x > a_s$:
Pokud ano, musíme v dalším kroku hledání pokračovat v části P , jež je napravo od středního prvku a_s . Je-li část P neprázdná (obsahuje aspoň jeden prvek), rekurzivně na ni provedeme stejný postup. Je-li už prázdná, vyhledávání neúspěšně končí.
- Pokud není ani $x > a_s$, zbývá už jen možnost, že platí $x = a_s$, čímž vyhledávání končí, neboť prvek a_s je hledaným prvkem.

Příklad. Necht' v poli jsou čísla

1 3 5 8 12 15 21 24 32 40

A máme najít číslo $x=15$.

Uprostřed jsou dva prvky. Dle popisu algoritmu vezmeme levý z nich - číslo 12:

1 3 5 8 12 15 21 24 32 40

Protože hledaná hodnota je větší, pro další krok vezmeme část napravo. Její střední prvek je číslo 24:

15 21 24 32 40

Protože hledaná hodnota je menší, vezmeme část nalevo. Její střední prvek je číslo 15:

15 21

A střední prvek je zde roven hodnotě v proměnné x , čímž je i hledaným prvkem.

Pseudokód – rekurzivní verze:

```
BinarySearch(A, k, l, x)
    if k>l
        return -1
    s ← (k+l)/2
    if x<A[s]
        return BinarySort(A,k,s-1,x)
    if x>A[s]
        return BinarySort(A,s+1,l,x)
    return s
```

Pseudokód – nerekurzivní verze:

```
BinarySearch(A, n, x)
    k ← 0
    l ← n-1
```

```

while k ≤ 1
    s ← (k+1)/2
    if x < A[s]
        l ← s-1
    else
        if x > A[s]
            k ← s+1
        else
            return s
return -1

```

Složitost metody

Při odvození složitosti vyjdeme z délky prohledávané části pole. V prvním kroku začínáme celým polem, tedy n prvky. Pokud v něm hledaný prvek nebyl nalezen, pak do dalšího kroku vezmeme část nalevo od něho nebo napravo od něho. Délka částí je $\frac{n-1}{2}$ nebo $\frac{n}{2}$ podle toho, zda počet prvků n je lichý nebo sudý. Pro odvození vezmeme ten „horší“ případ, kdy délka části vybrané pro následující krok je $\frac{n}{2}$.

Krok	Délka prohledávané části
1.	n
2.	$\frac{n}{2}$
3.	$\frac{n}{2^2}$
4.	$\frac{n}{2^3}$
....	
k-tý	$\frac{n}{2^{k-1}}$

Zřejmě vyhledávání skončí nejpozději v kroku, kdy délka prohledávané části je už tvořena jen jedním prvkem. Tedy položíme

$$\frac{n}{2^{k-1}} = 1 \quad .$$

Úpravou

$$n = 2^{k-1} \quad .$$

A použitím funkce logaritmu

$$k = \lceil \log_2(n) + 1 \rceil \quad .$$

Maximální počet kroků logaritmicky závisí na počtu prvků v prohledávané posloupnosti. V každém kroku provádíme nejvýše dvě operace srovnání. První operací zjistíme, zda hledaný prvek je menší než střední prvek. Pokud ano, pokračujeme v hledání v části nalevo. Pokud ne, druhou operací srovnání zjistíme, zda hledaný je větší než střední prvek, čímž rozhodneme, zda pokračovat v hledání v části napravo anebo už jsme hledaný prvek našli.

Závěr: Složitost binárního vyhledávání je $\Theta(\ln(n))$. Z ní plyne, že binární vyhledávání je mnohem rychlejší než sekvenční vyhledávání, kdy prvky nejsou uspořádány dle velikosti vyhledávacího klíče (setříděny).

Srovnání obou verzí

Vygenerováno 10 000 000 náhodných čísel, hledáno 1 000 000 náhodně vybraných čísel z nich:

Binární hledání	Čas v sekundách
Rekurzivní verze	1.47
Nerekurzivní verze	0.64