18. Vyhledávání

Způsob zjištění, zda je hledaný prvek přítomen v prohledávaných datech. Vyhledávání probíhá v organizované datové struktuře (pole, spojový seznam, arraylist, soubor…). Vyhledávací algoritmy typicky vrací logickou hodnotu (**true** / **false**), která indikuje, zda byl hledaný prvek nalezen. Nejčastěji prohledáváme pole.

Mezi parametry patří **složitost**. S ní související hardwarové nároky, nároky na vstupní data (struktura, datové typy, nutnost seřazení prvků).

Možnosti algoritmu jsou dány jazykem, ve kterém je programován. To ovlivňuje například datové struktury a typy, které mohou být prohledávány (neobjektové jazyky neumožní použití spojových seznamů, striktně typované jazyky neumožňují použití nehomogenní datové struktury).

Vyhledávací algoritmy

Brute Force (Naivní vyhledávání)

Algoritmus prochází celé pole, od prvního prvku až po poslední. U každého prvku kontroluje, zda není shodný s hledaným. I když najde shodu, prochází celé pole do konce.

Výhody

- Jednoduchost
- Aplikovatelný pro všechny datové typy a struktury
- Není třeba upravovat pole
- Jednoduchou úpravou lze získat další funkce

Nevýhody

- Prohledává celé pole

Složitost = N

```
public static boolean bruteForce(int[] array, int search){
        int count = 0;
        boolean result = false;
        for(int i = 0; i<array.length; i++){
            if(array[i] == search){
                result = true;
                count++; //počítání výskutů
            }
        }
        System.out.println(count);
        return result;
}
```

Vyhledávání bez zarážky

Algoritmus prochází pole, dokud nenajde hledaný prvek. Když najde shodu, pole dál neprochází.

Výhody

- Jednoduchost
- Aplikovatelný pro všechny datové typy a struktury
- Není třeba upravovat pole

Nevýhody

- Dva testy v každém kroku (zda nebyl nalezen prvek a zda není konec pole)

Složitost = 2*N

```
public static boolean withoutStop(int[] array, int search){
        boolean result = false;
        int i = 0;
        while((i < array.length) && (result == false)){
            if(array[i] == search){
                result = true;
            }
            i++;
        }
        return result;
}
```

Vyhledávání se zarážkou

Aby byla vyloučena nutnost dvou testů bez rizika, že se „sáhne“ mimo pole, přidá se na konec pole další prvek, který bude stejný jako hledaný.

Pokud se přidává další prvek jako **zarážka** je nutné pole překopírovat do nového a většího.

Poté stačí jediný test pro každý krok, protože je zaručené, že prve bude vždy nalezen a vyhledávání skončí.

Nakonec je nutné zjistit, zda byl hledaný prvek v původním poli. Pokud byl hledaný prvek nalezen na posledním indexu, znamená to, že v původním poli nebyl.

Výhody

- Jednoduchost
- Aplikovatelný pro všechny datové typy a struktury

Nevýhody

- Nutná úprava pole

Složitost = N+1

```
public static boolean withStop(int[] array, int search){
        int newArray[] = new int[array.length+1];
        System.arraycopy(array,0,newArray,0,array.length);
        newArray[array.length] = search;
        boolean result = false;
        int i = 0;
        while (result == false){
            if(newArray[i] == search){
                result = true;
            }
            i++;
        }
        if(i == newArray.length){
            return false;
        }else{
            return true;
        }
}
```

Binární vyhledávání

Nejdříve se otestuje prvek uprostřed. Je-li nalezena shoda, vyhledávání skončí.

Pokud ne, porovná se velikost a podle ní se rozhodne, ve které polovině pole se bude dále vyhledávat.

Z vybrané poloviny se vybere prostřední prvek a vyhledávání pokračuje až do doby, kdy je prvek nalezen, nebo než už pole dále nejde dělit.

Zde se vytvoří dva indexy do pole, **levý**, který ukazuje na první prvek pole, a **pravý**, který ukazuje na poslední.

Z těchto se poté počítá prostřední prvek a tento prvek se porovná s hledaným prvkem.

Pokud se nerovnají, pak se buď levý index posune doprava, nebo pravý doleva. Toto se opakuje, dokud se nepřekříží.

Výhody

- Efektivnost

Nevýhody

- Nutná úprava pole (seřazení)
- Pouze pro číselné datové typy

Složitost = $\log_2(N)$

```
public static boolean binary(int[] array, int search){
    int left = 0;
    int right = array.length - 1;
    do{
        int middle = (left + right)/2;
        if(array[middle] == search){
            return true;
        }else if(array[middle] > search){
            right = middle - 1;
        }else{
            left = middle + 1;
        }
    }while(left <= right);
    return false;
}
```