Ing. Karel Johanovský
SPŠ-JIA

VYHLEDÁVÁNÍ PRVKŮ V POLI

Popisy, principy, realizace a složitosti

Problém

- Vstup: Máte zadáno pole celých čísel, nebo jakýchkoliv jiných prvků, které lze nějak rozumně porovnávat. Dále máte zadán prvek, který se může vyskytovat v tomto poli.

- Úkol: Máte najít algoritmus, který prohledá toto pole a ohlásí, zda se v něm hledaný prvek vyskytuje.

Nyní trochu podrobněji…

- Máme zadáno pole
  - Tzn. homogenní datovou strukturu, u které máme přístup k libovolné její položce v konstantním čase na základě indexu.
- Prvky se dají porovnávat
  - Tzn. pro dva libovolné prvky můžeme říci, zda jsou si rovny, nebo nikoliv.
- Zajímá nás pouze to, zda prvek v poli je nebo ne
  - Zjištění pozice, nebo počtu výskytů apod. se dosáhne „drobnou“ úpravou základních alg.

Co nás zajímá?

- U vyhledávacích algoritmů nás zajímá, podobně jako u jiných, jejich složitost. Čili jak roste výpočetní čas (doba trvání algoritmu), když zvyšuji požadavky (rozsah vstupních dat).
- Složitost vyhledávání se zapisuje jako funkce, která nám říká, kolik musíme udělat elementárních operací nad zadanými daty, abychom mohli uživateli s jistotou říci, zda se hledaný prvek v poli vyskytuje nebo ne.
- U vyhledávání bývá zvykem za tuto operaci považovat porovnání dvou prvků, čili jinými slovy řečeno, složitost nám zde říká kolik musíme udělat porovnání, abychom nalezli hledaný prvek.

Naivní vyhledávání

Naivní vyhledávání

- Základní myšlenka:
  - Mám pole prvků a u pole znám jeho velikost. Projdu tedy toto pole pomocí cyklu se známým počtem opakování a u každého prvku se zeptám, zda není roven tomu hledanému.
- Výhody
  - Nejsou potřeba žádné předchozí úpravy pole
- Nevýhody
  - Hrubá síla

Naivní vyhledávání - JAVA

```
public static boolean Naive(int[] array, int search)
{
  boolean find = false;
  for(int i = 0; i < array.length; i++)
  {
    if(array[i] == search) find = true;
  }
  return find;
}
```

Proměnná find představuje stav hledání. Na začátku předpokládáme, že hledaný prvek v poli není a že tedy nebude nalezen (false), poté postupně procházíme všechny prvky pole a hledáme, jakmile nalezneme hledaný prvek, tak find překlopíme na (true).

Naivní vyhledávání - C

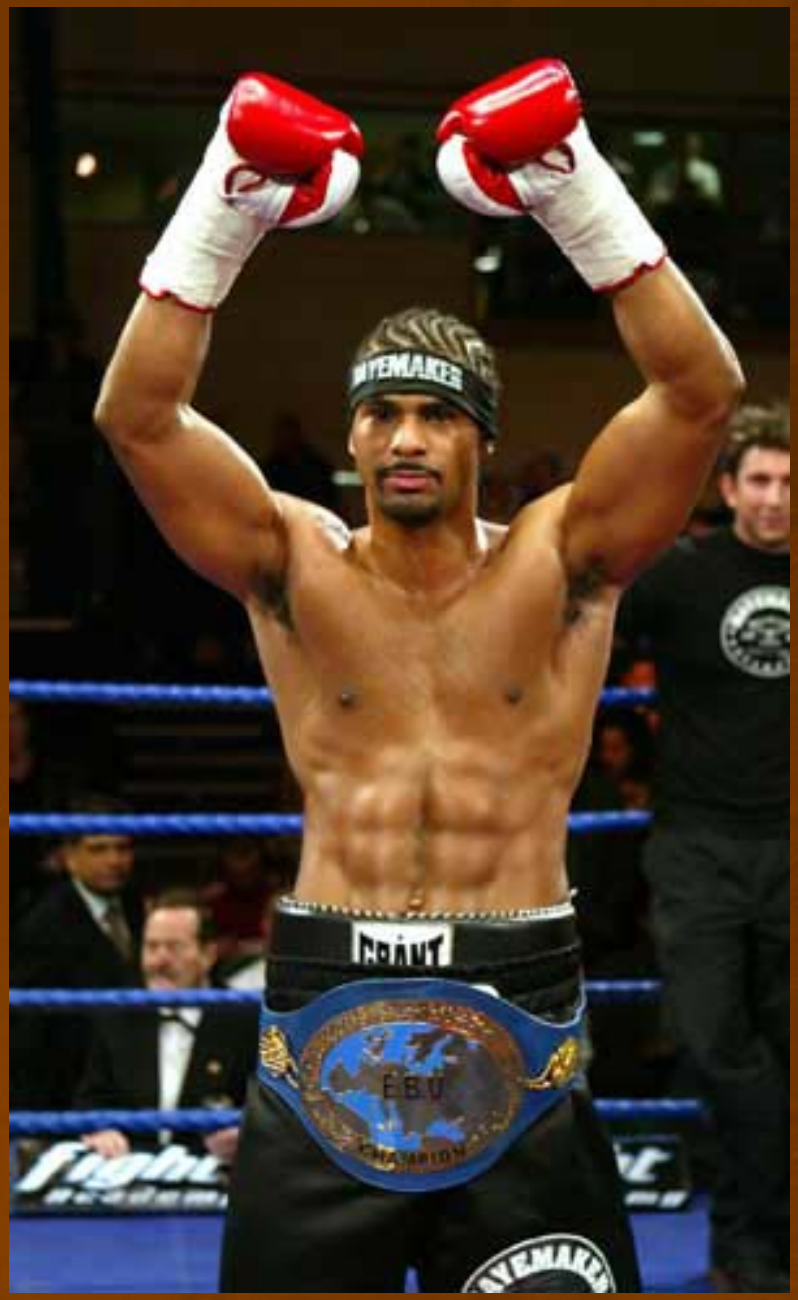
```
int Naive(int* array, int search)
{
  int find = 0;
  int i = 0;
    for(i = 0; i < N; i++)
    {
      if(array[i] == search) find = 1;
    }
    return find;
}
```

Proměnná find představuje stav hledání. Na začátku předpokládáme, že hledaný prvek v poli není a že tedy nebude nalezen (0), poté postupně procházíme všechny prvky pole a hledáme, jakmile nalezneme hledaný prvek, tak find překlopíme na (1).

Naivní vyhledávání

- Složitost
  - Celkem uděláme N porovnání, kde N je počet prvků v poli. Bohužel jich uděláme vždy N.
  - Cyklus se známým počtem opakování je konstruován, tak aby udělal pevný počet cyklů.
  - A tudíž i když hledaný prvek nalezneme na prvním místě v poli, tak i přesto zbytečně prohledáme zbytek pole i když nám pouze stačí informace „NALEZENO“.

$$Min \approx Max \approx Avg \approx N$$



Vyhledávání bez zarážky

Vyhledávání bez zarážky

- Základní myšlenka
  - Nebudu vždy prohledávat celé pole, ale skončím jakmile jsem nalezl prvek. K tomuto použiji cyklus s neznámým počtem průchodů.
- Výhody
  - Nejsou potřeba, žádné předchozí úpravy pole.
- Nevýhody
  - Dva testy v každém kroku.

Vyhledávání bez zarážky - JAVA

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

I zde proměnná find představuje stav nalezenosti hledaného prvku. Jak je vidět v podmínce, hledání probíhá dokud platí, že prvek nebyl nalezen a současně, že jsme stále v platných položkách pole.

Vyhledávání bez zarážky - C

```
int withoutStop(int* array, int search)
{
  int find = 0;
    int i = 0;
    while((find == 0) && (i < N))
    {
      if(array[i] == search) find = 1;
      i++;
    }
    return find;
}
```

I zde proměnná find představuje stav nalezenosti hledaného prvku. Jak je vidět v podmínce, hledání probíhá dokud platí, že prvek nebyl nalezen a současně, že jsme stále v platných položkách pole.

Vyhledávání bez zarážky

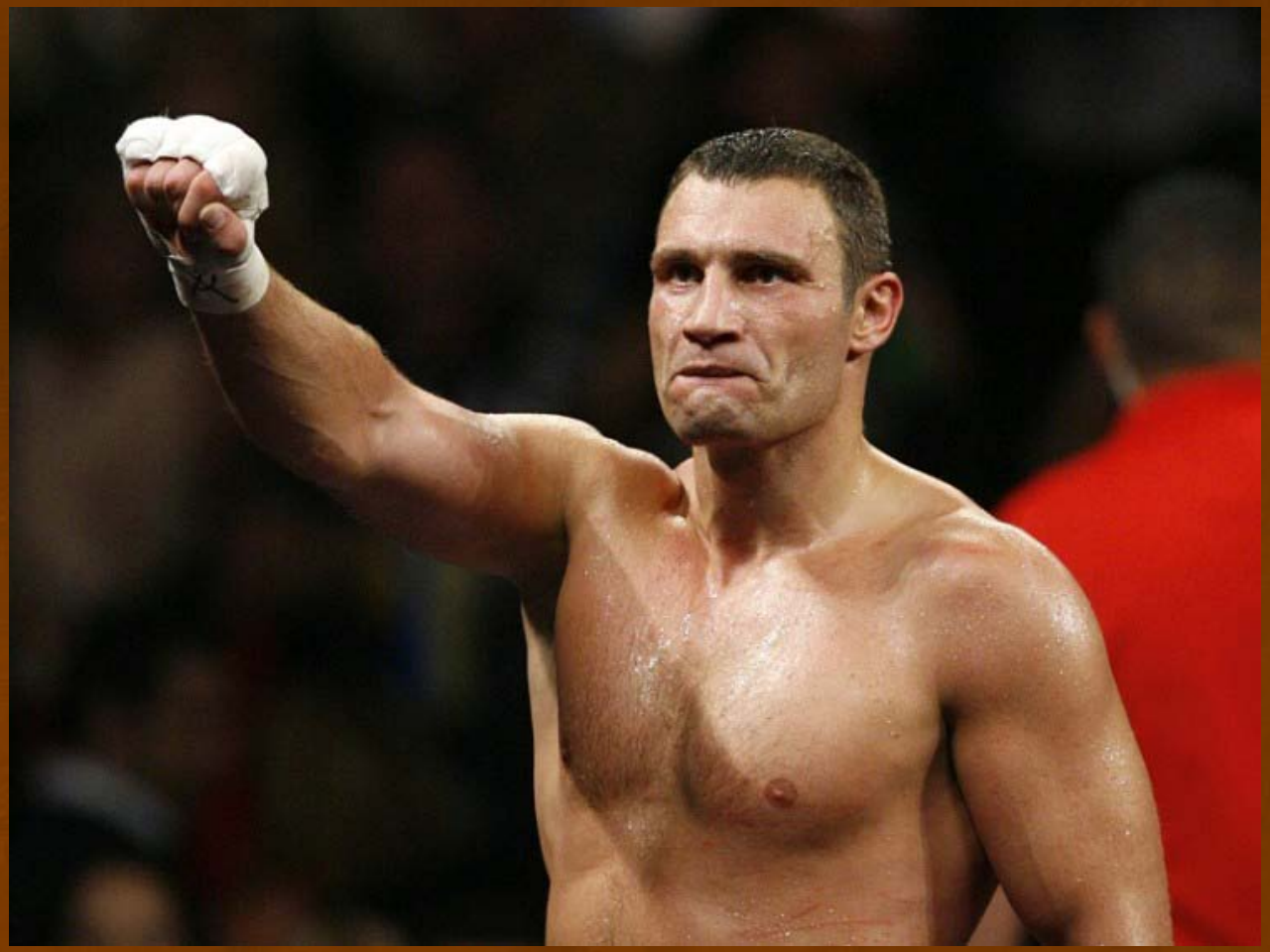
- Složitost
  - Jak již bylo zmíněno v každém kroku je nutné udělat 2 testy – jeden na konec pole a druhý na to, zda již nebyl prvek nalezen.
  - Jakmile jedna z těchto podmínek není splněna vyhledávání končí.

$$Min \approx 2$$

$$Max \approx 2N$$

$$Avg \approx \frac{2N+2}{2} \approx N+1$$

Vyhledávání se zarážkou



Vyhledávání se zarážkou

- Základní myšlenka
  - Protože ve chvíli kdy spouštím vyhledávání, znám hledaný prvek, přidám na konec pole ještě jeden prvek, jehož hodnota je rovna hledanému. Poté prohledávám, ale již kontroluji pouze to, zda jsem nalezl hledaný prvek – protože ho vždy najdu.
- Výhody
  - Jeden test v každém kroku.
- Nevýhody
  - Nutná úprava pole – přidání hledaného prvku na konec.

Vyhledávání se zarážkou - JAVA

```
public static boolean withStop(int[] array, int search)
{
  boolean find = false;
  int i = 0;
  int[] newarray = new int[array.length+1];
  for(i = 0; i < array.length; i++) newarray[i] = array[i];
  newarray[array.length] = search;
  i = 0;
  while(find == false)
  {
    if(newarray[i] == search) find = true;
    i++;
  }
  if (i == newarray.length) return false;
  else return true;
}
```

V první části je nutné vytvořit nové pole o velikosti toho původního + 1 prvek.
Poté to staré pole do nového zkopírovat a na konec umístit hledaný prvek.
Na závěr musí být ještě jedna kontrola, kde jsme hledaný prvek našli – pokud to bylo na zarážce, pak v původním poli nebyl.

Vyhledávání se zarážkou - C

```
int withStop(int* array, int search)
{
  int find = 0;
  int i = 0;
  int newarray[N+1];
  for(i = 0; i < N; i++) newarray[i] = array[i];
  newarray[N] = search;
  i = 0;
  while(find == 0)
  {
    if(newarray[i] == search) find = 1;
    i++;
  }
  if (i == N+1) return 0;
  else return 1;
}
```

V první části je nutné vytvořit nové pole o velikosti toho původního + 1 prvek.
Poté to staré pole do nového zkopírovat a na konec umístit hledaný prvek.
Na závěr musí být ještě jedna kontrola, kde jsme hledaný prvek našli – pokud to bylo na zarážce, pak v původním poli nebyl.

Vyhledávání se zarážkou

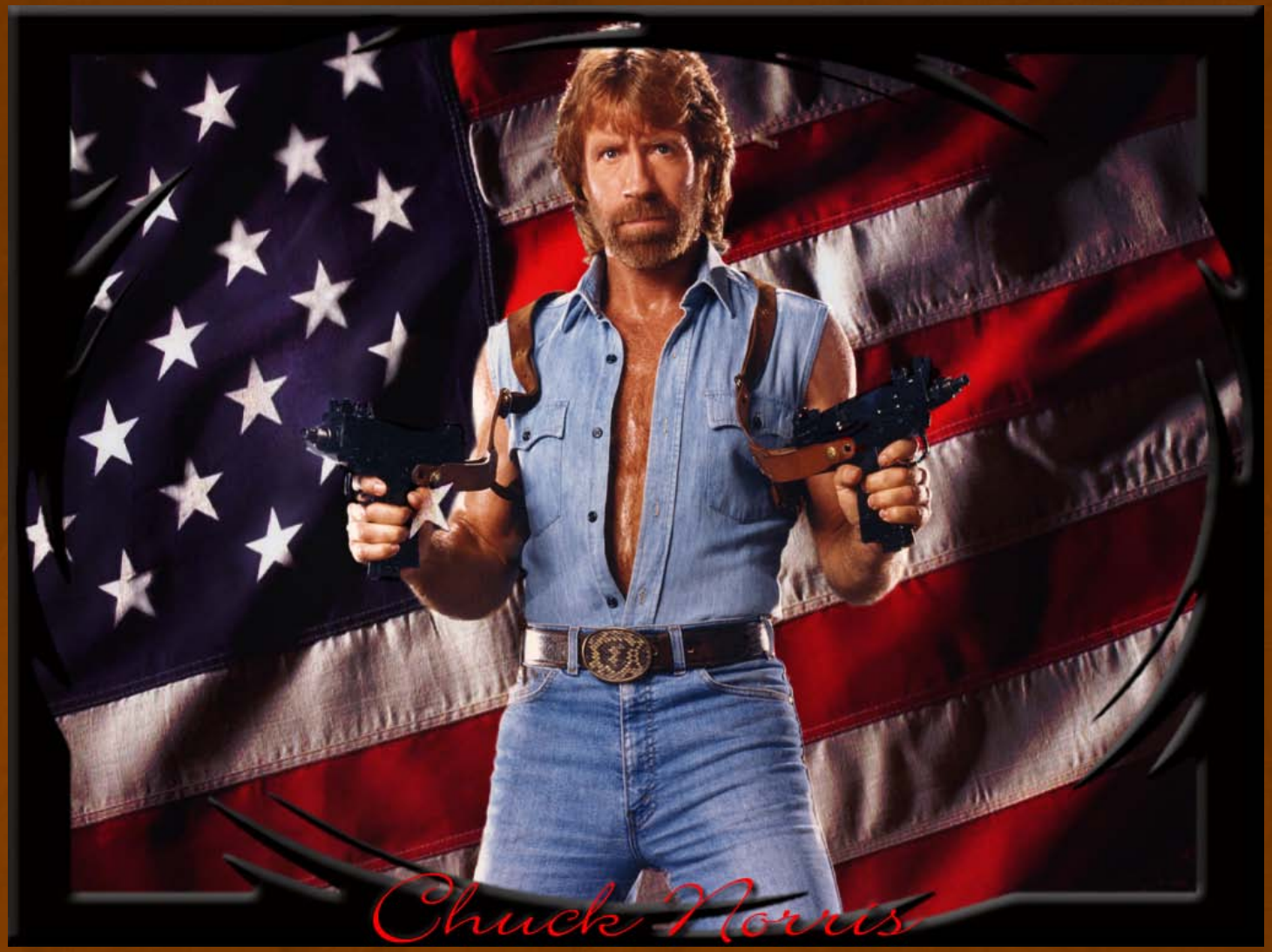
- Složitost
  - Oproti předchozímu, nám zde stačí testovat pouze to, zda jsme nalezli hledaný prvek.
  - Čili vždy ho najdeme, otázkou je pouze kde. Proto musí po samotném hledání následovat ještě jedna podmínka, která nám řekne kde byl prvek nalezen.

$$Min \approx 2$$

$$Max \approx N + 1$$

$$Avg \approx \frac{N + 3}{2}$$

Binární vyhledávání



Binární vyhledávání

- Základní myšlenka
  - V seřazeném poli se podívám na prostřední prvek, pokud je to ten který hledám – končím. Pokud ne, tak zjistím zda prvek který hledám je menší nebo větší jak ten který byl uprostřed – dále postup opakuji už jen v příslušné polovině pole.
- Výhody
  - Nejrychlejší způsob vyhledávání. V každém kroku odpadne polovina z prohledávané části pole.
- Nevýhody
  - Musíme mít pole seřazené podle velikosti.

Binární vyhledávání - JAVA

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

Zde vytvoříme dva indexy do pole, levý který ukazuje na první prvek pole a pravý který ukazuje na poslední.
Z těchto poté počítáme prostřední prvek a tento porovnáme s hledaným prvkem.
Pokud se nerovnají, pak buď levý index posunu doprava, nebo pravý doleva a to opakuju dokud se nepřekříží.

Binární vyhledávání - C

```
int binary(int* array, int search)
{
  int l = 0;
  int r = N - 1;
  do
  {
    int middle = (l+r)/2;
    if (array[middle] == search) return 1;
    else if (array[middle] < search) l = middle+1;
    else r = middle - 1;
  }
  while(l <= r);
  return 0;
}
```

Zde vytvoříme dva indexy do pole, levý který ukazuje na první prvek pole a pravý který ukazuje na poslední.
Z těchto poté počítáme prostřední prvek a tento porovnáme s hledaným prvkem.
Pokud se nerovnají, pak buď levý index posunu doprava, nebo pravý doleva a to opakuju dokud se nepřekříží.

Binární vyhledávání

- Složitost
  - Vyhledávání je velmi rychlé.
  - V každém kroku, pokud není prvek nalezen, odpadne polovina prvků z prohledávané části pole.

$$Min \approx 1$$

$$Max \approx \log_2(N)$$

$$Avg \approx \frac{\log_2(2N)}{2}$$

DOTAZY K VĚCI?

POKUD NE,
TAK DĚKUJI ZA POZORNOST