Seminár z algoritmizácie a programovania 1

Martin Bobák
Ústav informatiky
Slovenská akadémia vied

Obsah prednášky

1. **Algoritmy prehľadávania**
2. **Zhrnutie**

Spätná väzba:
https://forms.gle/iKbuLdF6xDtNSEDP8

Predpoklady

- budeme pracovať s poľom celých čísiel
  - ak by sme v poli mali uložený iný typ údajov, mení sa len spôsob porovnávania prvkov
- pole môže obsahovať tú istú hodnotu viackrát
  - zaujíma nás index prvého výskytu

Vyhľadávanie v neusporiadanom poli

Sekvenčné vyhľadávanie

Od začiatku poľa postupne zväčšuje index pokiaľ nepríde na hodnotu, ktorá je väčšia alebo rovná alebo pokiaľ nepríde na koniec poľa

- k prvkom poľa pristupujeme priamo pomocou indexu (poradie prvku v poli)

Časová zložitosť: **O(n)**

Sekvenčné vyhľadávanie

```
int sekvencne(int pole[], int n, int x)
{
   int i=0;

   while(i < n){
      if(pole[i] == x)
         return i;
      i++;
   }

   return -1;
}
```

Sekvenčné vyhľadávanie so zarážkou

Najprv umiestni na koniec vstupného poľa hľadaný prvok, potom prehľadávame pole od jeho začiatku až po výskyt hľadaného prvku.

Ušetrili sme jedno porovnanie v každej iterácii

Časová zložitosť: **O(n)**

Sekvenčné vyhľadávanie so zarážkou

```
int sekvencne_zarazka(int pole[], int n, int x)
{
   pole[n]=x;
   int i=0;

   while(pole[i] != x)
      i++;
   if (i < n)
      return i;
   else
      return -1;
}
```

Vyhľadávanie extrému

Hľadáme najmenšiu, alebo najväčšiu hodnotu v poli (algoritmus je možné rozšíriť tak, aby hľadal oba extrémy naraz).

Postupne prechádzame prvky poľa a aktualizujeme hodnotu lokálneho extrému.

Časová zložitosť: **O(n)**

Vyhľadávanie extrému

```
int sekvencne_max(int pole[], int n)
{
    int max = pole[0];
    int i=1;

    while(i < n){
        if(pole[i] > max)
            max = pole[i];
        i++;
    }
    return max;
}
```

Vyhľadávanie k-tej hodnoty

- nájdeme minimum, odstránime minimum z poľa a následne hľadáme minimum v upravenom poli. Toto opakujeme k-krát.
  - neefektívne
- vyberiem jeden prvok. Pole podľa neho čiastočne usporiadam (vľavo sú menšie prvky a v pravo sú väčšie prvky).
  - podobne ako Quick sort

Časová zložitosť: **$O(n^2)$**, v priemere **$O(n)$**

Vyhľadávanie v usporiadanom poli

Sekvenčné vyhľadávanie (s podmienkou)

Od začiatku poľa postupne zväčšuje index pokiaľ nepríde na hodnotu, ktorá je väčšia alebo rovná alebo pokiaľ nepríde na koniec poľa

- pole je usporiadané – ak narazíme na väčší prvok, hľadaný prvok sa určite v poli nenachádza

Časová zložitosť: **O(n)**

Sekvenčné vyhľadávanie (s podmienkou)

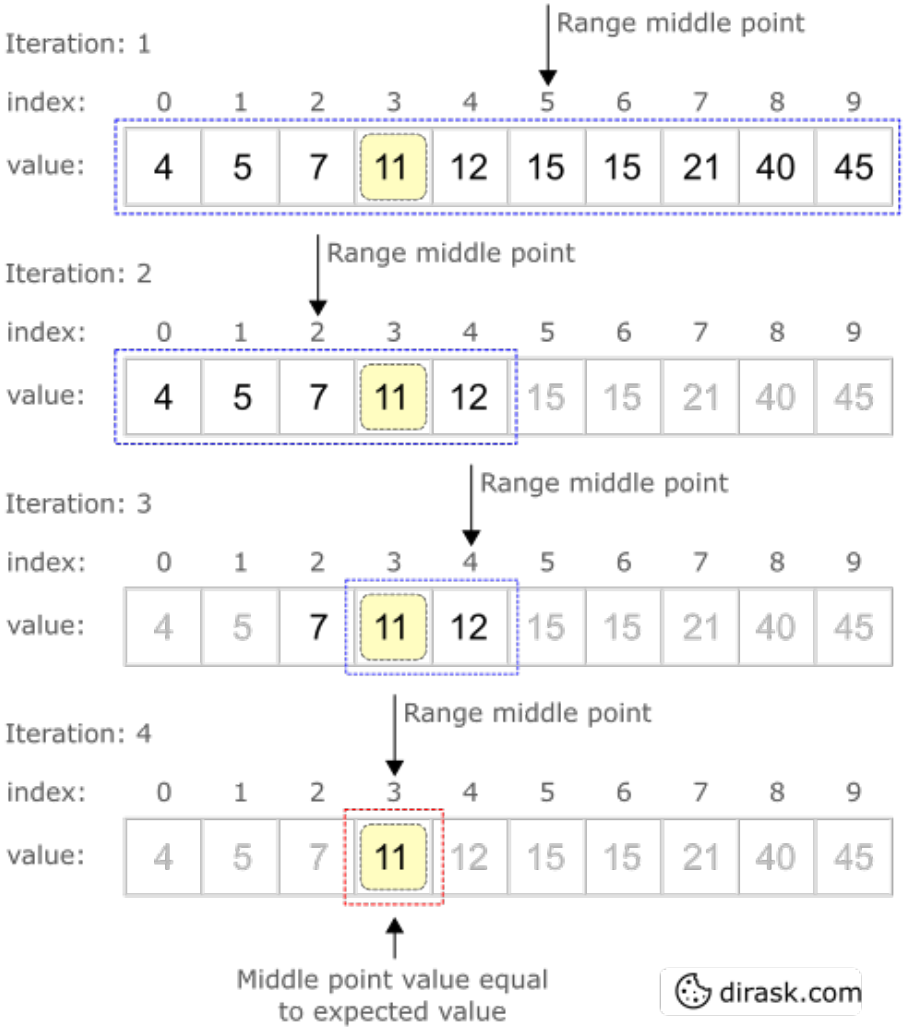
```
int sekvencne(int pole[], int n, int x)
{
   int i=0;

   while(i < n){
      if(pole[i] < x)
         i++;
      else
         if (pole[i]==x)
             return i;
         else return -1;
   }

   return -1;
}
```

Binárne vyhľadávanie

Nájdenie stredu intervalu - ak je hľadaná hodnota menšia ako hodnota stredného prvku hľadanie v ľavej polovici, inak v pravej polovici

Časová zložitosť: **O(log n)**

Binárne vyhľadávanie



https://dirask.com/posts/JavaScript-binary-search-algorithm-example-gp5lgD

Binárne vyhľadávanie

```
int binarne(int pole[], int n, int x) {
   int m, l = 0, r = n-1;
   while (l <= r) {
      m = (l + r) / 2;

      if (x == pole[m])
         return m;
      if (x < pole[m])
         r = m - 1;
      else
         l = m + 1;
   }
   if (pole[m] == x)
      return m;
   else
      return -1;}
```

Binárne vyhľadávanie v jazyku C

```
#include <stdlib.h>
```

```
qsearch(&key, array, length, sizeof(type), compFunc);
```

hľadaný prvok

```
int compFunc(const void *a, const void b*){
    return(*(int *)a - *(int *)b);}
```

záporné číslo znamená, že *a < *b

kladné číslo znamená, že *a > *b

0 znamená, že a == b

http://www.cplusplus.com/reference/cstdlib/bsearch/

Interpolačné vyhľadávanie

Podobne ako binárne vyhľadávanie, len inak vyberáme "stred".

stred = lava_hranica +

(hladany_prvok – prvok_na_lavej_hranici) * (prava_hranica – lava_hranica) /

(prvok_na_pravej_hranici – prvok_na_lavej_hranici)

Predpokladáme, že prvky sú rozdelené rovnomerne. Čím je rozdelenie prvkov nerovnomernejšie, tým je horšie časová zložitosť -> **v najhoršom prípade O(n).**

Časová zložitosť: **O(log log n)**

Interpolačné vyhľadávanie

```
int interpolacne(int pole[], int lavy_o, int pravy_o, int hladany)
{
   while (pole[lavy_o] <= hladany && hladany <= pole[pravy_o]) {
            float menovatel=(pole[pravy_o]-pole[lavy_o]);
            if (menovatel==0) {
                  if (pole[lavy_o]==hladany) return lavy_o;
                  else return -1;
            }
            int stred = lavy_o + (hladany - pole[lavy_o]) *
((pravy_o-lavy_o) / menovatel);
            if (pole[stred]==hladany) return stred;
            else if (pole[stred] < hladany) lavy_o=stred+1;
            else pravy_o=stred-1;
      }
      return -1;
}
```

Fibbonacciho vyhľadávanie

Podobne ako binárne vyhľadávanie, mierne efektívnejšie (nepoužíva delenie)

Predpokladáme, že prvky sú rozdelené rovnomerne. Čím je rozdelenie prvkov nerovnomernejšie, tým je horšie časová zložitosť -> **v najhoršom prípade O(n).**

Časová zložitosť: **O(log n)**

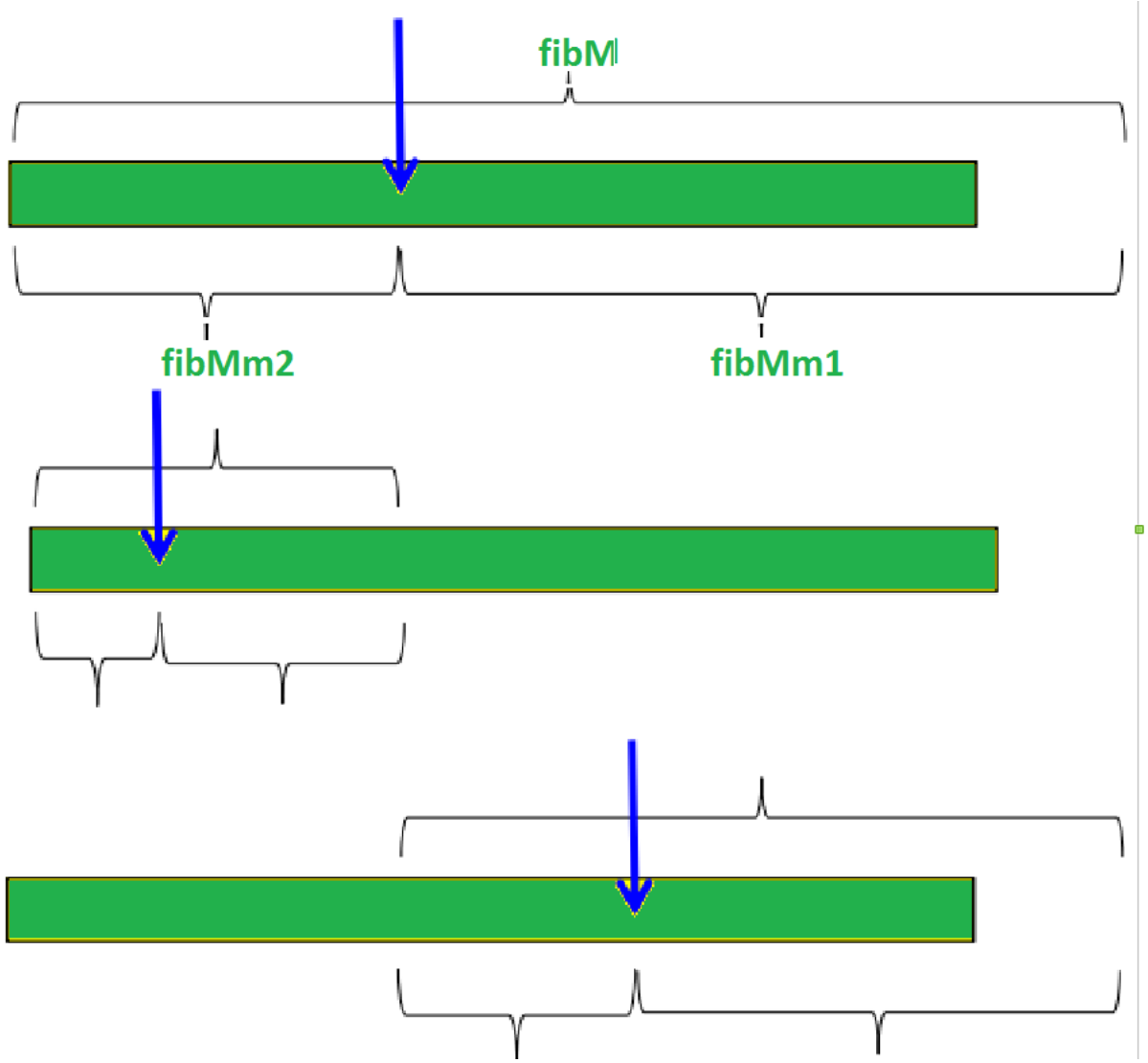
Fibbonacciho vyhľadávanie

```
int najdi_fib_index(int* pole, int velkost, int hladany){
	int fib_pole[MAX_DLZKA_FIB_RADU];
	fib_pole[0]=0;
	fib_pole[1]=1;
	int index=1;
	while(index < MAX_DLZKA_FIB_RADU && fib_pole[index] <
velkost) {
		index++;
		fib_pole[index] = fib_pole[index-2] + fib_pole[index-
1];
	}
	int i=fib_pole[index-1];
	int p=fib_pole[index-2];
	int q=fib_pole[index-3];
```

Fibbonacciho vyhľadávanie

```
        if (i==0) return -1;
        while(true){
                if (pole[i-1]==hladany) return i-1;
                else if (pole[i-1] < hladany) {
                        if (p==1) return -1;
                        else {
                                i=i+q;
                                p=p-q;
                                q=q-p;
                        }
                }
                else {
                        if (q==0) return -1;
                        else {
                                i=i-q;
                                int t=p;
                                p=q;
                                q=t-q;
}}}}
```

Fibbonacciho vyhľadávanie



Zdroj:https://www.geeksforgeeks.org/fibonacci-search/

Ďakujem vám za pozornosť!

Spätná väzba:
https://forms.gle/iKbuLdF6xDtNSEDP8