Natjecateljsko programiranje 2016./2017.

Pretraživanja

Sadržaj

1. Složenosti algoritama - ponavljanje
2. Binarno pretraživanje
3. Ternarno pretraživanje

Složenosti algoritama - ponavljanje

- zanimljiva nam je samo *worst-case* složenost
  - koristimo *Big O* notaciju
- složenost algoritma opisuje kako broj koraka algoritma raste s veličinom ulaza (problema)
  - veličinu ulaza tipično označavamo sa n
- naprimjer, ako je broj koraka algoritma $f(n) = 4n^2 + n - 2$ tada kažemo da je njegova složenost $O(n^2)$

Složenosti - primjeri

```
int f(int n) {
  int s = 0;
  for (int i = 1; i <= n; ++i) {
    s += i;
  }
  return s;
}
```

Složenost??

$O(n)$

Složenosti - primjeri

```
int f(int n) {
  return n * (n + 1) / 2;
}
```

Složenost??

O(1)

Složenosti - primjeri

```
int f(int n) {
  int s = 0;
  for (int i = 1; i <= n; ++i) {
    for (int j = 1; j <= i; ++j) {
      s += i * j;
    }
  }
  return s;
}
```

Složenost??

$O(n^2)$

Složenosti - primjeri

```
int f(int n) {
  int s = 0;
  for (int i = 1; i <= n; ++i) {
    for (int j = 1; j <= n; j += i) {
      s += i * j;
    }
  }
  return s;
}
```

Složenost??

O(n log n)

Složenosti - primjeri

```
int f(int n) {
  int s = 0;
  while (n > 0) {
    s += n % 2;
    n /= 2;
  }
  return s;
}
```

Složenost??

O(log n)

Pretraživanja

Problem: pronaći broj u sortiranom nizu.

Primjer: pronađi 55 u nizu 1, 6, 10, 25, 34, 47, 55, 62, 87, 99

Naivno rješenje

- usporeďujemo s 55 svaki broj niza, počevši od prvoga
- Složenost??
    - *O(n)*

Brzo rješenje

1, 6, 10, 25, 34, 47, 55, 62, 87, 99

1, 6, 10, 25, 34, 47, 55, 62, 87, 99

1, 6, 10, 25, 34, 47, 55, 62, 87, 99

1, 6, 10, 25, 34, 47, 55, 62, 87, 99

Složenost??

O(log n)

Binarno pretraživanje

- temeljni problemi koji binarno pretraživanje rješava
  - u nizu oblika 000...0011...111 pronađi prvu jedinicu
  - u nizu oblika 000...0011...111 pronađi posljednju nulu
- nizovi su često implicitni, npr. funkcije nad cjelobrojnom domenom
- svi drugi problemi koji se mogu riješiti binarnim pretraživanjem se mogu reducirati na jedan od dva temeljna problema
- problem reduciramo kreiranjem predikatne funkcije koja za indeks niza vraća 0 ili 1

Primjer reduciranja problema

➔ reducirajmo problem traženja broja u sortiranom nizu na prvi temeljni problem (traženje prve jedinice)

$$p(i) = \begin{cases} 1 & \text{ako } a_i \geq 55 \\ 0 & \text{ako } a_i < 55 \end{cases}$$

i	0	1	2	3	4	5	6	7	8	9
a_i	1	6	10	25	34	47	55	62	87	99
p(i)	0	0	0	0	0	0	1	1	1	1

Primjer reduciranja problema

- reducirajmo problem traženja broja u sortiranom nizu na drugi temeljni problem (traženje posljednje nule)

$$p(i) = \begin{cases} 1 & \text{ako } a_i > 55 \\ 0 & \text{ako } a_i \leq 55 \end{cases}$$

i	0	1	2	3	4	5	6	7	8	9
a_i	1	6	10	25	34	47	55	62	87	99
$p(i)$	0	0	0	0	0	0	0	1	1	1

Kako riješiti temeljni problem?

- ➔ prvo definiramo početni prostor pretraživanja (npr. interval *[lo, hi]*)
- ➔ zatim *prepolavljamo* prostor pretraživanja sve dok on ne sadrži samo jedan element
- ➔ invarijanta kroz algoritam: rješenje se uvijek nalazi u trenutnom prostoru pretraživanja

Traženje prve jedinice

inicijaliziraj *lo*, *hi*

dok je *lo* < *hi* čini:

    mid = lo + (hi-lo) / 2

    ako je *p(mid) == 1* onda:

        hi = mid

    inače:

        lo = mid + 1

vrati *lo*

Traženje posljednje nule

inicijaliziraj *lo*, *hi*

dok je *lo* < *hi* čini:

    mid = lo + (hi-lo+1) / 2

    ako je *p(mid) == 1* onda:

        hi = mid - 1

    inače:

        lo = mid

vrati *lo*

Binarno pretraživanje - drugi primjer

- zadan je cijeli broj N $(1 <= N <= 10^9)$
- ispiši najmanji cijeli broj čiji je kub veći ili jednak N

Binarno pretraživanje - drugi primjer

```
lo = 1, hi = 1000
dok je lo < hi čini:
    mid = lo + (hi-lo) / 2
    ako je mid*mid*mid >= N onda:
        hi = mid
    inače:
        lo = mid + 1
vrati lo
```

Binarno pretraživanje nad realnom domenom

- budući da je prostor pretraživanja beskonačan ne možemo imati isti uvjet zaustavljanja kao kod binarnog pretraživanja nad diskretnom domenom
- dva moguća rješenja:
  - prekidanje kada interval *[lo, hi]* postane dovoljno mali
  - fiksan broj iteracija

Primjer - *sqrt(N)*

```
lo = 0, hi = 1e9
dok je hi-lo > 1e-6 čini:
    mid = lo + (hi-lo) / 2
    ako je mid*mid >= N onda:
        hi = mid
    inače:
        lo = mid
vrati hi
```

```
lo = 0, hi = 1e9
za i od 1 do 50 čini:
    mid = lo + (hi-lo) / 2
    ako je mid*mid >= N onda:
        hi = mid
    inače:
        lo = mid
vrati hi
```

Ternarno pretraživanje

- traženje minimuma ili maksimuma funkcije nad realnom domenom
- funkcija mora biti strogo rastuća pa strogo padajuća (ili obrnuto)
- glavna ideja: u svakom koraku smanjujemo prostor pretraživanja 1.5 puta
- složenost: $O(\log n)$

Ternarno pretraživanje - primjer

- ➔ pronađi minimum funkcije $f(x) = ax\^2 + bx + c$
- ➔ vrijedi $a > 0$ te $1 <= |a|,|b| <= 10^9$

Ternarno pretraživanje - primjer - rješenje

```
lo = -1e9, hi = 1e9
dok je hi-lo > 1e-6 čini:
    mid1 = lo + (hi - lo) / 3
    mid2 = hi - (hi - lo) / 3
    ako je f(mid1) < f(mid2) onda:
        hi = mid2
    inače:
        lo = mid1
vrati lo
```

Ternarno pretraživanje - nastavak

- ternarno pretraživanje na diskretnom skupu možemo raditi na sličan način, ali ga možemo i reducirati na binarno pretraživanje
- primjerice, tražimo li minimum funkcije *f* tada možemo koristiti sljedeću predikatnu funkciju:

$$p(x) = \begin{cases} 1 & \text{ako } f(x) < f(x+1) \\ 0 & \text{ako } f(x) >= f(x+1) \end{cases}$$

- minimum nalazimo traženjem prvog *x* za kojega vrijedi *p(x) = 1*

Korisni linkovi

- ➔ https://www.topcoder.com/community/data-science/data-science-tutorials/binary-search/
- ➔ http://wiki.xfer.hr/binarno_pretrazivanje/