

Obrada teksta

Slajdovi sa predavanja¹

© Goodrich, Tamassia, Goldwasser

Katedra za informatiku, Fakultet tehničkih nauka, Univerzitet u Novom Sadu

2022.

¹Po uzoru na materijale sa: <https://github.com/mbranko/asp-slajdovi>

String

- **string** je niz karaktera
- primeri stringova:
 - Python program
 - HTML dokument
 - DNK sekvenca
 - digitalna slika
- **alfabet** Σ je skup mogućih karaktera za familiju stringova
- primeri alfabeta:
 - ASCII
 - Unicode
 - $\{0, 1\}$
 - $\{A, C, G, T\}$

String

- neka je P string dužine m
 - **podstring** $P[i..j]$ od P je podsekvencu od P koja sadrži karaktere sa rangom između i i j
 - **prefiks** od P je podstring tipa $P[0..i]$
 - **sufiks** od P je podstring tipa $P[i..m-1]$
- za date stringove T (tekst) i P (šablon, *pattern*) *pattern matching* problem je pronalaženje podstringa od T koji je jednak P
- primene:
 - editori teksta
 - mašine za pretragu (*search engines*)
 - bioinformatika

Nalaženje podstringa grubom silom

- nalaženje **grubom silom** (*brute force*) poredi šablon P sa tekstom T za svaki mogući položaj P u odnosu na T sve dok se
 - ne pronađe poklapanje
 - ne testiraju sve pozicije
- gruba sila radi u $O(nm)$ vremenu
- primer najgoreg slučaja:
 - $T = aaa \dots ah$
 - $P = aaah$
 - može da se pojavi u slikama i DNK sekvencama
 - retko u tekstovima

Nalaženje podstringa grubom silom

BruteForceMatch(T, P)

Input: tekst T dužine n i šablon P dužine m

Output: indeks početka podstringa u T jednakog P ili -1 ako nije pronađen

for $i \leftarrow 0$ **to** $n - m$ **do**

$j \leftarrow 0$

{testiramo položaj i }

while $j < m \wedge T[i + j] = P[j]$ **do**

$j \leftarrow j + 1$

if $j = m$ **then**

return i

{poklapanje na i }

else

break

return -1

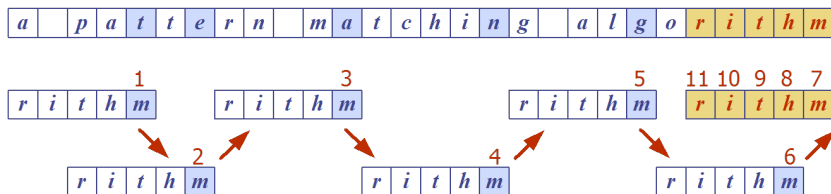
{nije pronađen}

Gruba sila u Pythonu

```
def find_brute_force(T, P):  
    """Funkcija vraća najmanji indeks teksta T  
    od kojeg počinje podstring P (u suprotnom -1)."""  
    n, m = len(T), len(P) # odredi dužine oba stringa  
    for i in range(n-m+1): # pokušaj za svaki potencijalni početni indeks u T  
        k = 0 # indeks u šablonu P  
        while k < m and T[i + k] == P[k]: # k-ti karakter P se podudara  
            k += 1  
        if k == m: # ako smo došli do kraja šablona,  
            return i # podstring T[i:i+m] odgovara P  
    return -1 # nije pronađeno nijedno poklapanje počevši od i
```

Boyer-Moore

- Boyer-Moore algoritam se zasniva na dva principa:
 - ogledalo: poredi P sa podsekvencom u T idući unazad
 - skok: ako se razlika otkrije u $T[i] = c$



Boyer-Moore: funkcija poslednjeg pojavljivanja

- **Boyer-Moore** algoritam formira **last occurence** funkciju L koja mapira alfabet Σ na cele brojeve gde je $L(c)$ definisano kao
 - najveći indeks i takav da $P[i] = c$
 - -1 ako takvog indeksa nema
- primer: $\Sigma = \{a, b, c, d\}$ $P = abacab$

c	a	b	c	d
$L(c)$	4	5	3	-1

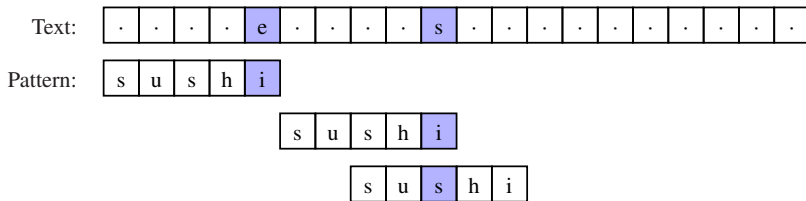
- može se predstaviti kao niz indeksiran numeričkim kodovima karaktera
- može se izračunati za $O(m + s)$ vreme gde je m dužina P a s je veličina Σ

Boyer-Moore

- **bad character shift**: preskoči sigurno neuspešna poređenja
- 0: ako P ne sadrži c : pomeri P tako da se poklope $P[0]$ i $T[i + 1]$
- 1: ako P sadrži c i poslednja pojava c je levo od pozicije i : pomeri P tako da se $T[i]$ poklopi sa poslednjom pojavom c u P
- 2: ako P sadrži c i poslednja pojava c je desno od pozicije i : pomeri P za jedno mesto

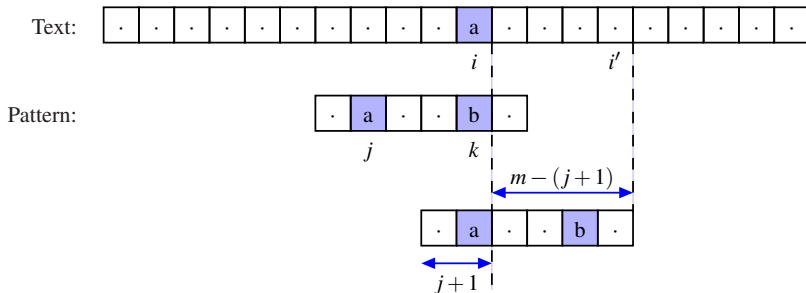
Boyer-Moore skok, slučaj 0

- slučaj 0 – ako P ne sadrži c :
- pomeri P tako da se poklope $P[0]$ i $T[i + 1]$



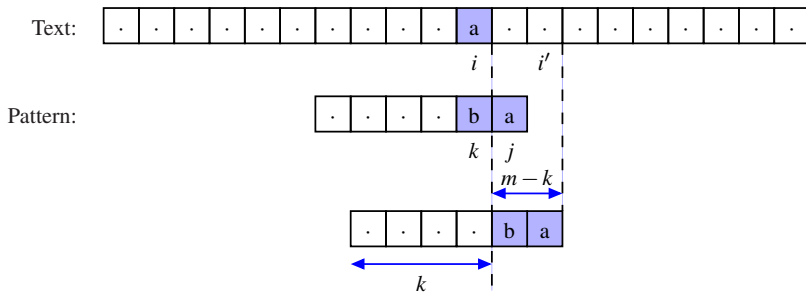
Boyer-Moore skok, slučaj 1

- slučaj 1 – ako P sadrži c i poslednja pojava c je levo od pozicije i :
- pomeri P tako da se $T[i]$ poklopi sa poslednjom pojavom c u P



Boyer-Moore skok, slučaj 2

- slučaj 2 – ako P sadrži c i poslednja pojava c je desno od pozicije i :
- rešenje A: pomeri P za jedno mesto
- rešenje B: pomeri P tako da se $T[i]$ poklopi sa sledećom pojavom c u P – last occurrence funkcija više nije dovoljna!



Boyer-Moore algoritam

BoyerMooreMatch(T, P, Σ)

$L \leftarrow \text{lastOccurence}(P, \Sigma)$

$i \leftarrow m - 1$

{indeks u T }

$j \leftarrow m - 1$

{indeks u P }

repeat

if $T[i] = P[j]$ **then**

if $j = 0$ **then**

return i

{poklapanje na i }

else

$i \leftarrow i - 1$

$j \leftarrow j - 1$

else

$l \leftarrow L[T[i]]$

{indeks poslednjeg pojavljivanja}

$i \leftarrow i + m - \min(j, 1 + l)$

{dva slučaja za skok}

$j \leftarrow m - 1$

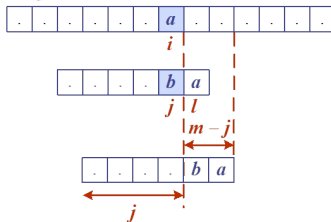
until $i > n - 1$

return -1

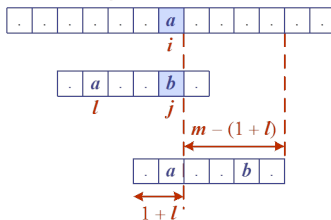
{nije pronađen}

Boyer-Moore

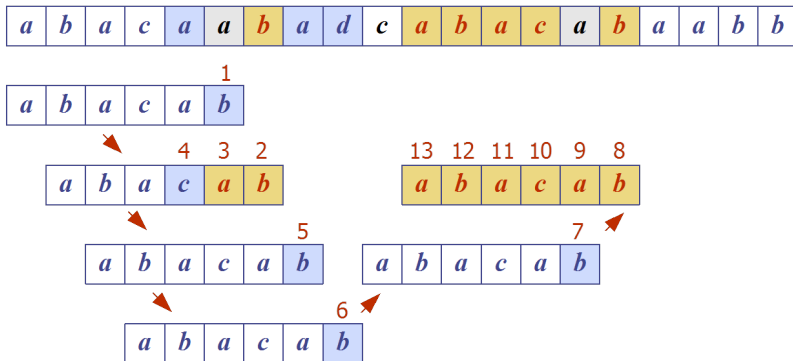
slučaj 1: $j \leq 1 + l$



slučaj 2: $1 + l \leq j$



Boyer-Moore: primer

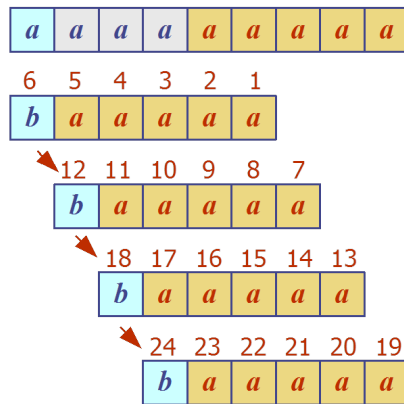


Boyer-Moore u Pythonu

```
def find_boyer_moore(T, P):
    """Funkcija vraća najmanji indeks teksta T
    od kojeg počinje podstring P (u suprotnom -1)."""
    n, m = len(T), len(P)          #odredi dužine oba stringa
    if m == 0:
        return 0                    # trivijalni slučaj za prazan string
    last = {}                        # napravi 'last' rečnik
    for k in range(m):
        last[ P[k] ] = k            # poslednja pojava karaktera pregaziće prethodnu vrednost
    # poravnaj kraj šablona i tekst na indeksu m-1
    i = m-1                          # indeks teksta T
    k = m-1                          # indekst šablona P
    while i < n:
        if T[i] == P[k]:            # ako se podudaraju karakteri
            if k == 0:
                return i            # šablon počinje na i-tom indeksu teksta
            else:
                i -= 1               # ispitaaj prethodni karakter
                k -= 1              # i teksta T i šablona P
        else:
            j = last.get(T[i], -1)  # last(T[i]) vraća -1 ako se ne nalazi u rečniku
            i += m - min(k, j + 1)  # dva slučaja za skok
            k = m - 1               # restart na kraju šablona
    return -1                       # nije pronađeno nijedno poklapanje počevši od i
```


Boyer-Moore: analiza

- Boyer-Moore je $O(nm + s)$
- primer najgoreg slučaja:
 - $T = aaa \dots a$
 - $P = baaa$
- najgori slučaj nije verovatan u tekstovima
- znatno brži od grube sile za tekstove na prirodnom jeziku

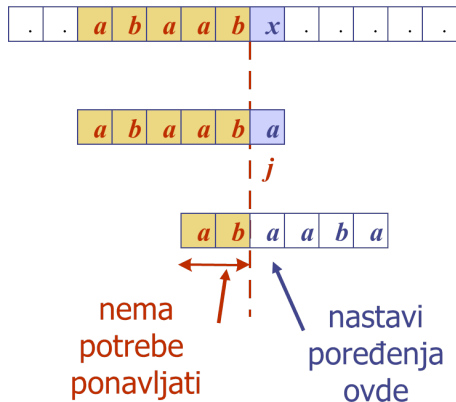


Boyer-Moore: analiza

- najgori slučaj za BM je $O(nm)$, isto kao i gruba sila
- za realne tekstove malo verovatan
- postoji i drugo pravilo za skok, **good suffix shift**, koje se zasniva na ideji koju koristi KMP algoritam (sledeći)

Knuth-Morris-Pratt

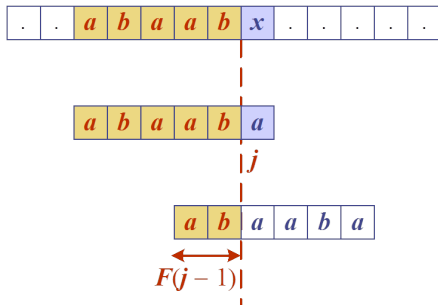
- **Knuth-Morris-Pratt** poredi tekst sa šablonom **slevo u desno** ali pomera šablon pametnije od grube sile
- kada se nađe razlika, koliko **najviše** možemo pomeriti šablon da izbegnemo suvišna poređenja?
- odgovor: najveći prefiks $P[0..j]$ koji je sufiks $P[1..j]$



KMP: funkcija neuspeha

- KMP analizira šablon da pronađe njegove prefikse unutar samog šablona
- funkcija neuspeha** $F(j)$ je veličina najvećeg prefiksa $P[0..j]$ takvog da je ujedno i sufiks $P[1..j]$
- ako nema poklapanja za $P[j] \neq T[i]$ pomeramo $j \leftarrow F(j-1)$

j	0	1	2	3	4	5
$P[j]$	a	b	a	a	b	a
$F(j)$	0	0	1	1	2	3



Knuth-Morris-Pratt algoritam

- funkcija neuspeha se može prikazati nizom koji se izračuna za $O(m)$
- u svakoj iteraciji petlje, ili
 - i se poveća za 1, ili
 - pomeraj $i - j$ se poveća za najmanje 1 (primeti da $F(j-1) < j$)
- \Rightarrow nema više od $2n$ iteracija u petlji
- \Rightarrow KMP je $O(m + n)$

```

KMPMatch( $T, P$ )
 $F \leftarrow$  failureFunction( $P$ )
 $i \leftarrow 0$ 
 $j \leftarrow 0$ 
while  $i < n$  do
  if  $T[i] = P[j]$  then
    if  $j = m - 1$  then
      return  $i - j$ 
    else
       $i \leftarrow i + 1$ 
       $j \leftarrow j + 1$ 
  else
    if  $j > 0$  then
       $j \leftarrow F[j - 1]$ 
    else
       $i \leftarrow i + 1$ 
return  $-1$ 

```

{poklapanje}

{nije pronađen}

KMP: izračunavanje funkcije neuspeha

- funkcija neuspeha se može prikazati nizom koji se izračuna za $O(m)$
- slično kao i sam KMP algoritam
- u svakoj iteraciji petlje, ili
 - i se poveća za 1, ili
 - pomeraj $i - j$ se poveća za najmanje 1 (primeti da $F(j-1) < j$)
- \Rightarrow nema više od $2n$ iteracija u petlji

failureFunction(P)

$F[0] \leftarrow 0$

$i \leftarrow 1$

$j \leftarrow 0$

while $i < m$ **do**

if $P[i] = P[j]$ **then**

$F[i] \leftarrow j + 1$ {poklapa se $j + 1$ znakova}

$i \leftarrow i + 1$

$j \leftarrow j + 1$

else if $j > 0$ **then**

$j \leftarrow F[j - 1]$ {koristi F da pomeriš P }

else

$F[i] \leftarrow 0$ {nema poklapanja}

$i \leftarrow i + 1$

Knuth-Morris-Pratt: primer

a b a c a a b a c c a b a c a b a a b b

1 2 3 4 5 6
a b a c a b

7
a b a c a b

8 9 10 11 12
a b a c a b

j	0	1	2	3	4	5
$P[j]$	a	b	a	c	a	b
$F(j)$	0	0	1	0	1	2

13
a b a c a b

14 15 16 17 18 19
a b a c a b

Knuth-Morris-Pratt u Pythonu ₁

```
def find_kmp(T, P):
    """Funkcija vraća najmanji indeks teksta T
    od kojeg počinje podstring P (u suprotnom -1)."""
    n, m = len(T), len(P)          # odredi dužine oba stringa
    if m == 0:
        return 0                    # trivijalni slučaj za prazan string
    fail = compute_kmp_fail(P)      # oslanjamo se na pomoćnu funkciju za pravljenje tabele
    j = 0                            # početni indeks teksta T
    k = 0                            # početni indeks šablona P
    while j < n:
        if T[j] == P[k]:            # P[0:1+k] se poklapa do sada
            if k == m - 1:          # poklapanje je gotovo
                return j - m + 1
            j += 1                  # napreduj na sledeći karakter
            k += 1
        elif k > 0:
            k = fail[k-1]           # ponovo iskoristi sufiks P[0:k]
        else:
            j += 1
    return -1                        # nije pronađeno nijedno poklapanje počevši od i
```


Knuth-Morris-Pratt u Pythonu 2

```
def compute_kmp_fail(P):
    """Pomoćna funkcija koja računa i vraća KMP 'fail' listu."""
    m = len(P)
    fail = [0] * m      # na početku, pretpostavka je da nema poklapanja
    j = 1
    k = 0
    while j < m:         # izračunaj f(j) tokom ovog prolaza, ako nije 0
        if P[j] == P[k]: # k + 1 karakter se poklapa do sad
            fail[j] = k + 1
            j += 1
            k += 1
        elif k > 0:      # k prati šablon poklapanja
            k = fail[k-1]
        else:            # nije pronađen pogodak počevši od indeksa j
            j += 1
    return fail
```