Rad sa stingovima

Algoritmi

Definicija string-a

- ***String*** je niz (sekvenca) karaktera (iz datog skupa karaktera)
  - Skup karaktera čine: slova (mala i velika), cifre, znaci interpunkcija, matematički i neki drugi simboli.
- Primeri:
  - „Danas je lep dan.“
  - „Izmeneni napon je 11.76 kV“
  - „AABAABBBABABAABABBBAABB“
  - „GTACCGTCA“ – primer iz biologije gde se DNK lanac sastoji od osnovnih molekula: A - *adenine*, C - *cytosine*, G - *guanine* i T - *thymine*.

Podstring

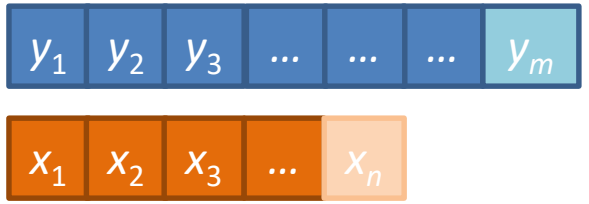
- **Podstring** (***substring***) – je deo stringa koji sadrži uzastopne karaktere.
  - Npr. string ABCDAADDCBAA ima podstringove
    - A, BCDA, CBAA, ABCDAA, itd, ali
    - AAA, ACADBA nisu podstringovi.
- Svaki podstring je ujedno i podsekvenca ali obrnuto ne važi.

Algoritmi rada sa stringovima

- Ovde se spominju samo neki od zanimljivih algoritama
  - Najduža zajednička podsekvenca
  - Transformacija jednog stringa u drugi
  - Podudaranje stringova (*string matching*)

Najduža zajednička podsekvenca

- Engl. *longest common subsequence* LCS – za data dva stringa pronalazi najduži *subsequence* koji se nalazi u oba stringa.
  - Npr. stringovi: CATCGA i GTACCGTCA imaju dva LCS: CTCA i TCGA.
  - Algoritam koristi dinamičko programiranje (ideja je da je *subsequence* podstringa ujedno i *subsequence* originalnog stringa)
    - Ako je $x_n = y_m$, onda je rešenje: LCS stringova $X_{1..n-1}$ i $Y_{1..m-1}$ sa dodatim x_n
    - Inače, ili se odbacuje x_n ili y_m (probaju se obe opcije).



  - Algoritam ima dve faze:
    - Izgrađuje pomoćnu tabelu dimenzije $(n+1)(m+1)$ gde su n i m dužine datih stringova – složenost faze je $\Theta(nm)$
    - Pomoću tabele nalazi LCS – složenost je $O(n+m)$

Korak	LCS
Potproblemi	$L_{i,j}\ , i = 1..n, j = 1..m$
Broj potproblema	$\Theta(nm)$
Probati	Uporediti $x_i = y_j$
Broj izbora	$\Theta(1)$
Odnos potproblema	$L(i,j) = \begin{cases} L(i-1,j-1)+1, & x_i = y_j \\ \max\{L(i-1,j), L(i,j-1)\}, & x_i \neq y_j \end{cases}$ $L(i,0) = L(0,j) = 0$
Vreme potproblema	$\Theta(1)$
Algoritam	Izračunati $L(i,j)$ for $i = 1, \dots, n$ for $j = 1, \dots, m$
Ukupno vreme	$\Theta(nm)$
Originalan problem	$L(n,m)$

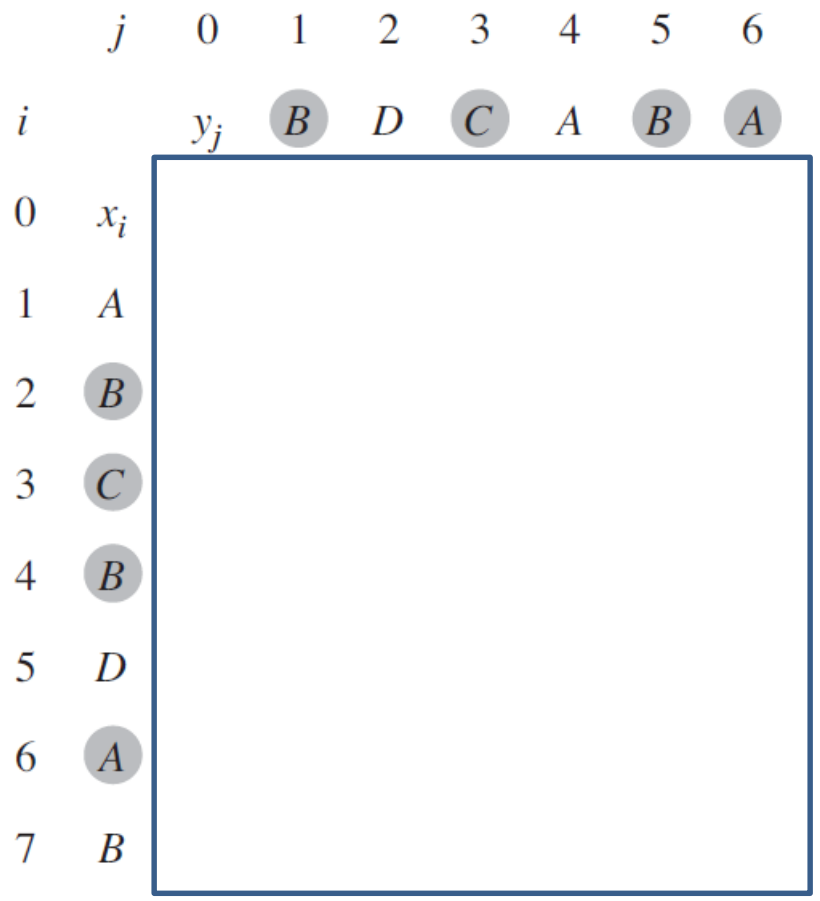
Rešenje

- Ulaz: stringovi X i Y

```
LCS_DUŽINA(X, Y)
1  n=X.length, m=Y.length
2  L[0..n,0..m], b[1..n,1..m]     // tabele
3  for i=1 to n
4     L[i,0] = 0
5  for j=1 to m
6     L[0,j] = 0
7  for i=1 to n
8     for j=1 to m
9        if X[i] == Y[j]
10          L[i,j] = L[i-1,j-1] + 1,      b[i,j]='↖'
11       elseif L[i-1,j] ≥ L[i,j-1]
12          L[i,j] = L[i-1,j],            b[i,j]='↑'
13       else
14          L[i,j] = L[i,j-1],            b[i,j]='←'
15 return L, b
```

Primer

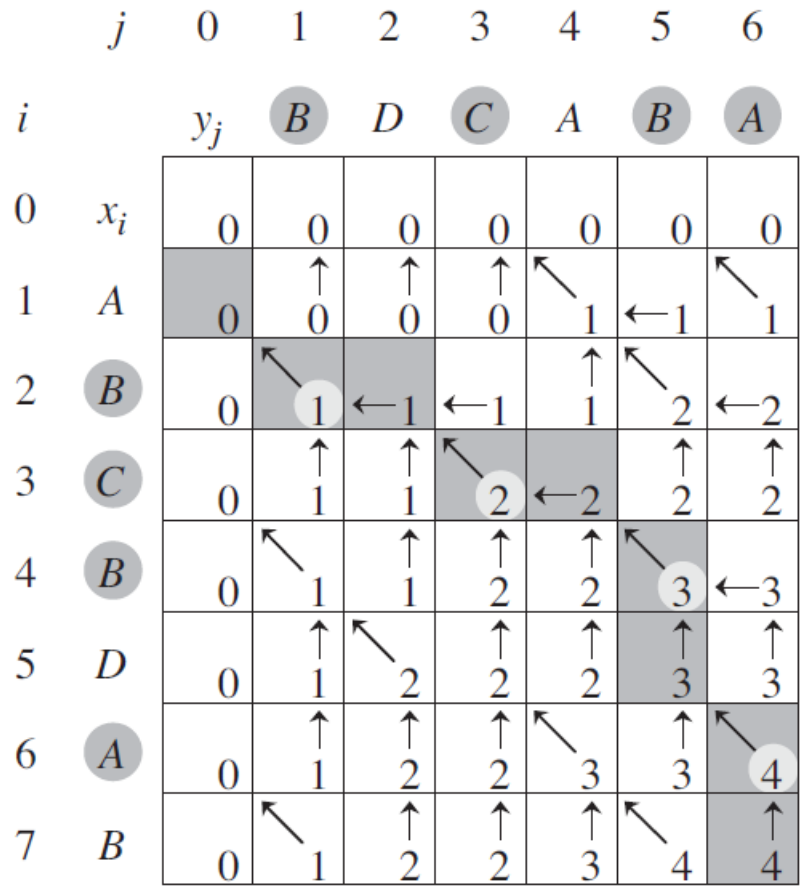
- $X = \text{"ABCBDAB"}, Y = \text{"BDCABA"}$




```
LCS_DUŽINA(X, Y)
1   n=X.length, m=Y.length
2   L[0..n,0..m], b[1..n,1..m]// tabele
3   for i=1 to n
4      L[i,0] = 0
5   for j=1 to m
6      L[0,j] = 0
7   for i=1 to n
8      for j=1 to m
9         if X[i] == Y[j]
10           L[i,j] = L[i-1,j-1] + 1, b[i,j]='↖'
11        elseif L[i-1,j] ≥ L[i,j-1]
12           L[i,j] = L[i-1,j],       b[i,j]='↑'
13        else
14           L[i,j] = L[i,j-1],       b[i,j]='←'
15  return L, b
```

Primer

- $X = \text{"ABCBDAB"}, Y = \text{"BDCABA"}$



i		j	0	1	2	3	4	5	6
				y_j	B	D	C	A	B	A
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B		0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C		0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B		0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D		0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A		0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B		0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

```
LCS_ISPIS(b,X,i,j)
1  if i==0 or j==0
2     return
3  if b[i,j]=='↖'
4     LCS_ISPIS(b,X,i-1,j-1)
5     Print X[i]
6  elseif b[i,j]='↑'
7     LCS_ISPIS(b,X,i-1,j)
8  else
9     LCS_ISPIS(b,X,i,j-1)
```

- Rešenje = "BCBA" dužine 4

Transformacija jednog stringa u drugi

- Opis: potrebno je transformisati string X u string Z upotrebom minimalnog broja operacija.
  - X čine karakteri $x_i, i = 1,2, \dots n$
  - Z čine karakteri $z_j, j = 1,2, \dots m$
- Operacije:
  - *Copy* - kopiranje karaktera: $z_j = x_i, i = i + 1, j = j + 1$
  - *Replace* - zamena karaktera: $z_j = x_i = a, i = i + 1, j = j + 1$
  - *Delete* - brisanje karaktera: $i = i + 1$ („preskoči“ karakter u X)
  - *Insert* - ubacivanje karaktera: $z_j = a,\ j = j + 1$ (X se ne dira).
- Problem: za date cene operacija naći „najjeftiniju“ transformaciju.
- Npr. ATGATCGGCAT postaje CAATGTGAATC ...
- Ovaj algoritam ne radimo, nego posmatramo samo primer.

Primer transformacije stringa

Operation	X	Z
initial strings	ATGATCGGCAT	
delete A	ATGATCGGCAT	
replace T by C	ATGATCGGCAT	C
replace G by A	ATGATCGGCAT	CA
copy A	ATGATCGGCAT	CAA
copy T	ATGATCGGCAT	CAAT
replace C by G	ATGATCGGCAT	CAATG
replace G by T	ATGATCGGCAT	CAATGT
copy G	ATGATCGGCAT	CAATGTG
replace C by A	ATGATCGGCAT	CAATGTGA
copy A	ATGATCGGCAT	CAATGTGAA
copy T	ATGATCGGCAT	CAATGTGAAT
insert C	ATGATCGGCAT	CAATGTGAATC

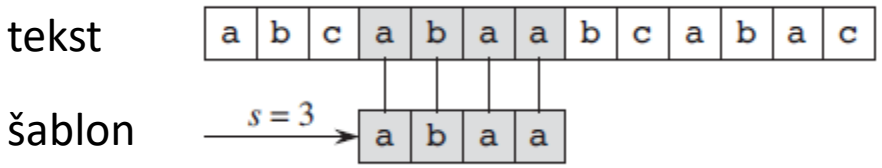
- Cena ove transformacije je $5c_k + 5c_z + c_b + c_u$, gde su c_k cena kopiranja, c_z cena zamene, c_b cena brisanja i c_u cena ubacivanja.
- Problem: za date c_k, c_z, c_b, c_u naći „najjeftiniju“ transformaciju.

Podudaranje stringova (*string matching*)

- Opis: pronalaći tekst šablona (*pattern-a*) u datom tekstu.
- Imamo dva stringa: string teksta T (dužine n) i string šablona P (dužine m, $m \leq n$), i želimo da pronađeno sve pojave P u T.

- Primer: tekst je GTAACAGTAAACG, a šablon je AAC ...

- Rešenje: GT**<u>AAC</u>**AGTA**<u>AAC</u>**G



Neki algoritmi podudaranja stringova

- **Algoritam grube sile** (*brute force*) je „naivan“ algoritam gde se vrši poređenje šablona sa podstringom koji počinje na poziciji k, slovo po slovo.
- **Rabin-Karp algoritam** poredi heš vrednost šablona i heš vrednosti podstringova (iz teksta) dužine šablona. Za računanje heš vrednosti podstringova koristi se *rolling* heš.
- **Konačni automat** tokom poređenja sa šablonom prelazi is stanja u stanje
- **KMP (*Knuth-Morris-Pratt*) algoritam** – pogodan kada šablon ima podstringove koji se ponavljaju, i tako da tokom poređenja šablon „pomera“ iza podstringa koji se ponavlja
- **…**

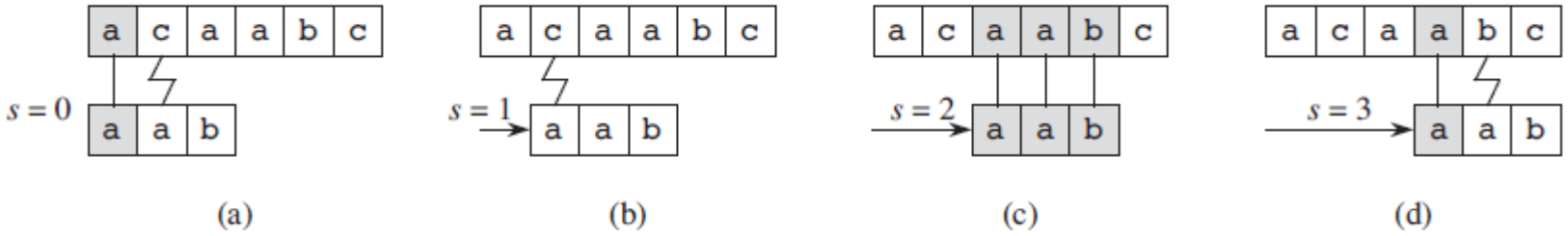
Složenosti algoritama podudaranja stringova

Algoritam	Pretprocesiranje	Poređenje	Zauzeće memorije
Algoritam grube sile	-	$\Theta(nm)$	-
Rabin-Karp algoritam	$\Theta(m)$	$\Omega(n + m)$, $O((n-m)m)$	$O(1)$
Konačni automat	$O(m\|\Sigma\|)$	$O(n)$	$O(m\|\Sigma\|)$
KMP (*Knuth-Morris-Pratt*) algoritam	$\Theta(m)$	$\Theta(n)$	$\Theta(m)$

Algoritam grube sile

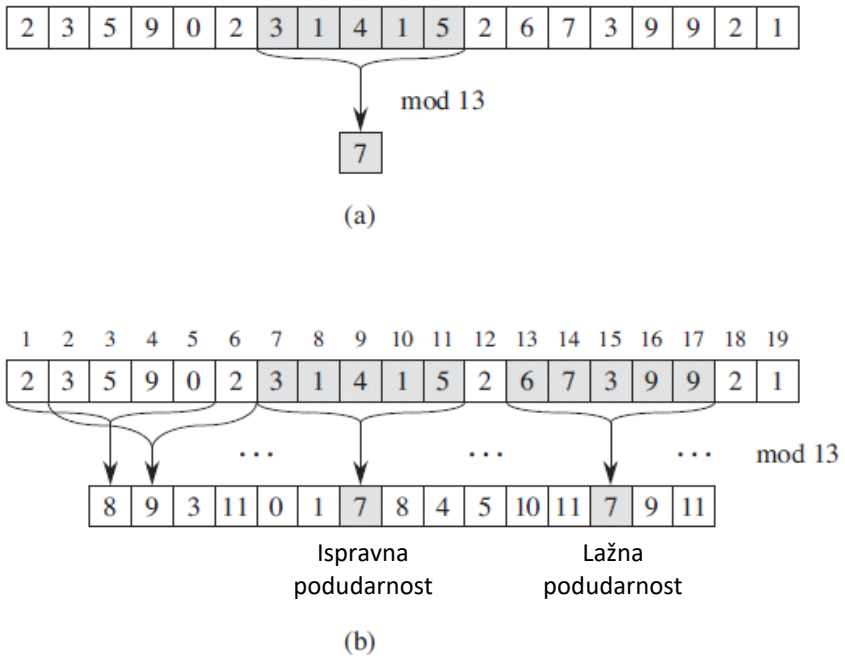
- poredi šablon sa podstringom teksta, gde se taj podstring pomera za po jedno slovo
- „naivan“ - poređenje ne koristi informacije iz prethodnih poređenja

```
PODUDARANJE-STRINGOVA-GRUBA-SILA(T,P)
1  n = T.length
2  m = P.length
3  for s = 0 to n-m
4     for j = 1 to m
5        if T[s+j] != P[j]
6           break
7        elseif j == m
8           nađen na indeksu s+1
```



Rabin-Karp algoritam

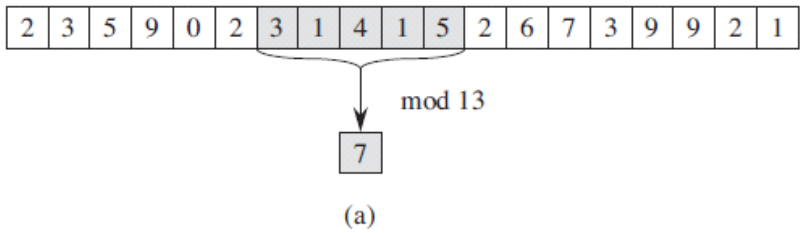
- Poredi heš vrednost šablona i heš vrednosti podstringova (iz teksta) dužine šablona
  - Poklapanje može biti „lažno“



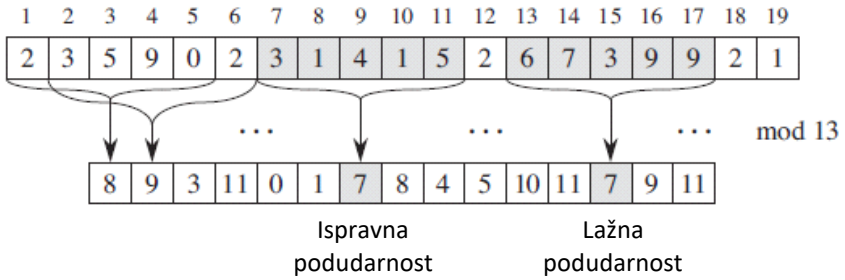
Rabin-Karp algoritam

- Poredi heš vrednost šablona i heš vrednosti podstringova (iz teksta) dužine šablona
  - Poklapanje može biti „lažno“
- Koristi se *rolling* heš

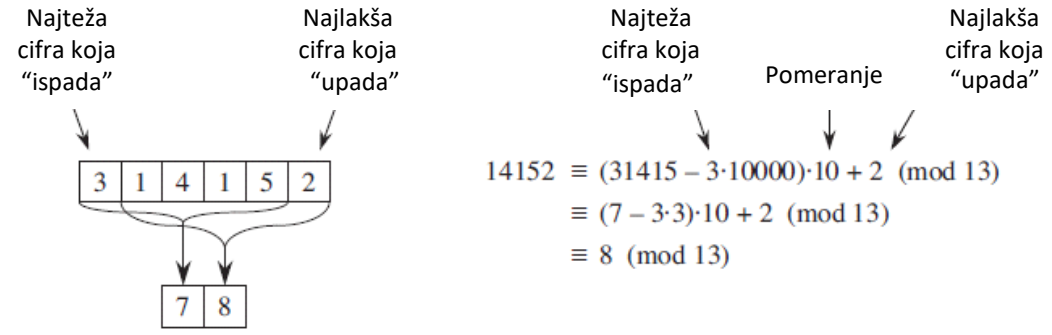
- Primer prikazuje princip upotrebom decimalnih cifara



(a)



(b)



(c)

Pseudokod Rabin-Karp algoritma

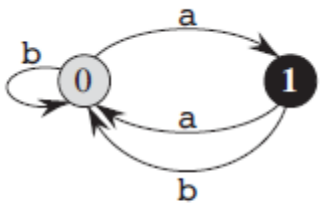
```
PODUDARANJE-STRINGOVA-RABIN-KARP(T,P)
1  d = ..., q = ...                    // d – zapis cifre, q - moduo
2  n = T.length
3  m = P.length
4  h = d^(m-1) mod q                   // težina prve cifre
5  hP = 0                              // heš vrednost šablona
6  hT = 0                              // heš vrednost podstringa
7  for i = 1 to m                      // priprema
8     hP = (d*hP + P[i]) mod q
9     hT = (d*hT + T[i]) mod q
10  for s = 0 to n-m                   // poređenja za smicanja s
11    if hP == hT                      // kandidat za podudaranje?
12      if P[1..m] == T[s+1..s+m]           // provera
13         nađen na indeksu s+1
14    if s < n-m
15      hT = d*(hT - T[s+1]*h) + T[s+m+1] mod q
```

Konačni automat

- Konačni automat sadrži:
  - Skup stanja Q
  - q_0 početno stanje
  - $A \subseteq Q$ podskup različitih dozvoljenih stanja
  - Σ konačan skup ulaznih vrednosti (ulazni alfabet)
  - δ funkciju za opis prelaza stanja $\delta: Q \times \Sigma \rightarrow Q$

state	input a	input b
0	1	0
1	0	0

(a)



(b)

Primer konačnog automata za poređenje stringova

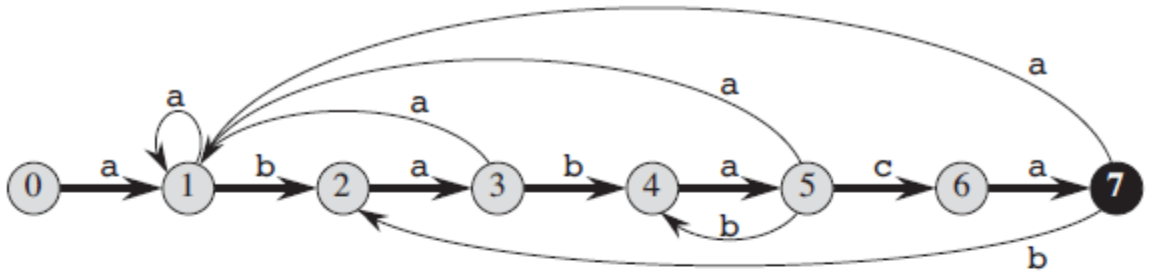
- Šablon je: **ababaca**
- Stanje odgovara broju slova koja se podudaraju
- Dolazak u stanje 7 odgovara podudaranju šablona

- Primer teksta:

  abababacaba

Primer konačnog automata za poređenje stringova

- Šablon je: **ababaca**
- Stanje odgovara broju slova koja se podudaraju
- Dolazak u stanje 7 odgovara podudaranju šablona



(a)

	input			
state	a	b	c	P
0	1	0	0	a
1	1	2	0	b
2	3	0	0	a
3	1	4	0	b
4	5	0	0	a
5	1	4	6	c
6	7	0	0	a
7	1	2	0	

(b)

i	—	1	2	3	4	5	6	7	8	9	10	11
$T[i]$	—	a	b	a	b	a	b	a	c	a	b	a
state $\phi(T_i)$	0	1	2	3	4	5	4	5	6	7	2	3

(c)

Pseudokod podudaranja stringova Konačnim automatom

```
PODUDARANJE-STRINGOVA-KONAČNI-AUTOMAT(T,P)
1  n = T.length
2  q = 0                // stanje
3  for i = 1 to n
4     q = δ(q, T[i])
5     if q == m
6        nađen na indeksu i-m+1
```

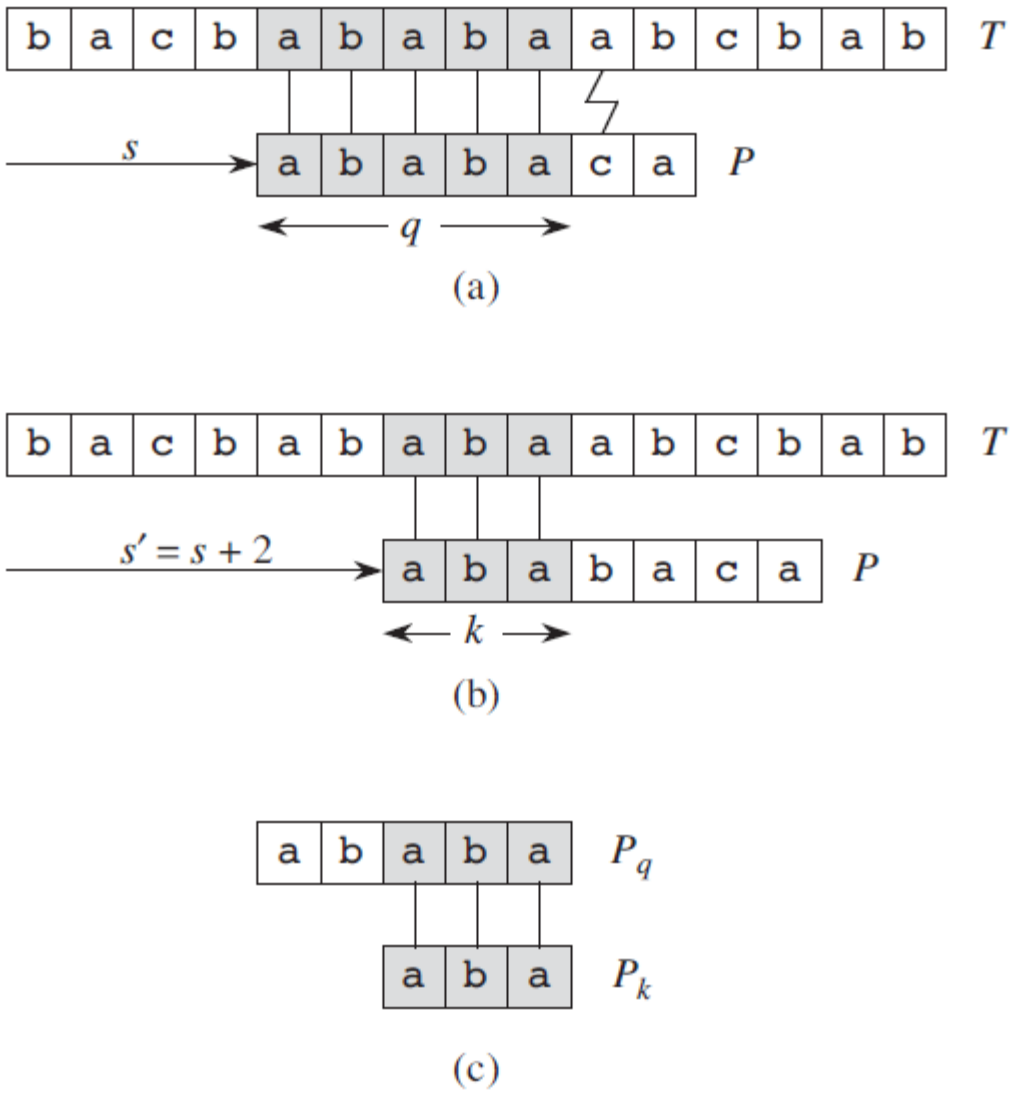
KMP algoritam

- Algoritam izbegava računanja funkcije prelaza stanja δ tako što koristi niz $\pi[1..m]$ koji se popuni u amortizovanom vremenu $\Theta(m)$
  - Funkcija prelaza stanja u konačnom automatu za stanja $q = 0,1, \dots m$ i neki karakter $a \in \Sigma$ određuje novo stanje $\delta(q, a)$ koje predstavlja broj podudarnih karaktera iz teksta i šablona
  - KMP računa π nezavisno od a gde se određuje pomeraj s u odnosu na početak šablona, tako što se zna da su karakteri iz šablona do tog pomeraja podudarni sa tekstom.

- Ukupna složenost: priprema $\Theta(m)$ + pretraga $\Theta(n)$ = $\Theta(n + m)$

KMP primer (1)

- Primer poređenja sa tekstom sa pomerajem s
- Neslaganje karaktera šablona „preskače“ ponovno poređenje početnih k karaktera šablona

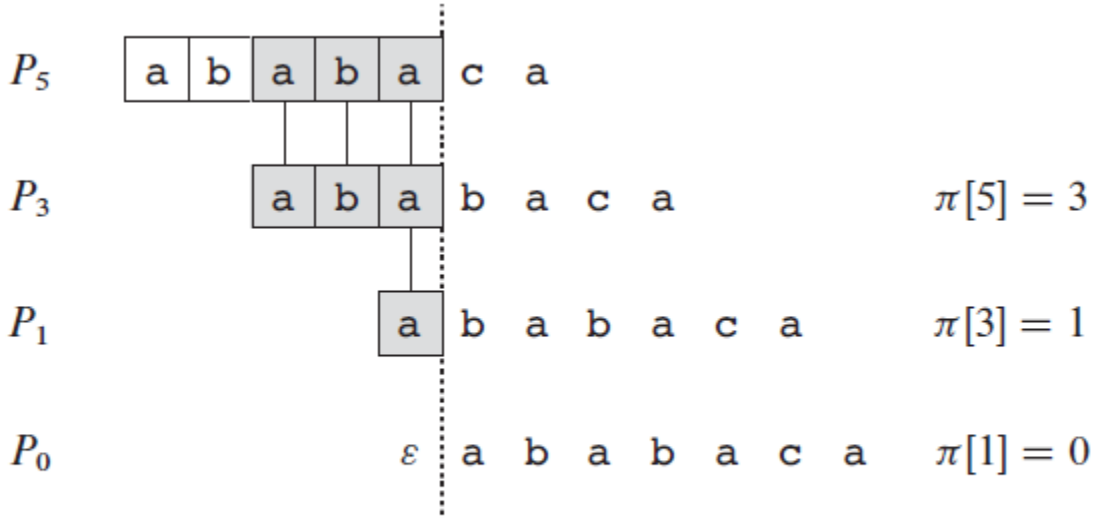


KMP primer (2)

- Određivanje π je u fazi pripreme: određuju se najduži prefikse i sufikse (π) za podstringove $P[1..]$
  - P[1..2] = “ab” – prefiks „a“ ≠ sufiks „b“, $\pi[2] = 0$
  - P[1..3] = “aba” – prefiks „a“ == sufiks „a“, a prefiks „ab“ ≠ sufiks „ba“, $\pi[3] = 1$
  - P[1..4] = “abab” – prefiks „ab“ == sufiks „ab“, a prefiks „aba“ ≠ sufiks „bab“, $\pi[4] = 2$
  - P[1..5] = “ababa” – prefiks „aba“ == sufiks „aba“, $\pi[5] = 3$
  - P[1..6] = “ababac” – prefiks „a“ ≠ sufiks „c“, $\pi[6] = 0$
  - P[1..7] = “ababaca” – prefiks „a“ == sufiks „a” , a prefiks „ab“ ≠ sufiks „ca“, $\pi[7] = 1$

i	1	2	3	4	5	6	7
$P[i]$	a	b	a	b	a	c	a
$\pi[i]$	0	0	1	2	3	0	1

(a)



(b)

KMP primer (3)

	1	2	3	4	5	6	7
P =	**A**	**B**	**A**	**B**	**A**	**C**	**A**
π =	0	0	1	2	3	0	1

- Pomera se po T (slovo po slovo) i poredi $T[i]$ sa šablonom $P[q]$, ali kada naiđe na neslaganje sa q-tim slovom šablona, postavi q tako da nastavi sa poređenjem sa sledećim slovom nakon najdužeg prefiksa-sufiksa prethodnog slova u šablonu (kada je bilo $q-1$)

```
T =   A B A B A B A E A B A B A C A D
P =   A B A B A C                       q ← π[5]+1 = 3+1 = 4
        A B A B A C                     q ← π[5]+1 = 3+1 = 4
          A B A B A C                   q ← π[3]+1 = 1+1 = 2
            A B A B A C                 q ← π[1]+1 = 0+1 = 1
              A B A B A C               q ← π[1]+1 = 0+1 = 1
                A B A B A C A           q == m
```

KMP pseudokod

```
PODUDARANJE-STRINGOVA-KMP(T,P)
1  n = T.length
2  m = P.length
3  π = ODREDI-PREFIKSE(P)
4  q = 0                              // broj jednakih karaktera
5  for i = 1 to n                     // po svim karakterima teksta
6     while q > 0 and P[q+1] ≠ T[i]
7        q = π[q]                     // sledeći karakter se ne poklapa
8     if P[q+1] == T[i]
9        q = q + 1                    // sledeći karakter šablona
10    if q == m                       // kraj poklapanja
11       nađen na indeksu i-m+1
12       q = π[q]                     // test narednog poklapanja
```

KMP pseudokod (nastavak)

```
ODREDI-PREFIKSE(P)
1  m = P.length
2  π = new [1..m]
3  π[1] = 0
4  k = 0
5  for q = 2 to m
6     while k > 0 and P[k+1] ≠ P[q]
7        k = π[k]
8     if P[k+1] == P[q]
9        k = k + 1
10    π[q] = k
11 return π
```