

# Bioinformatika 1

## Sufiksno stablo

---

**Mirjana Domazet-Lošo**  
**FER, 2020./2021.**



Creative Commons Imenovanje-Nekomercijalno-Dijeli pod istim uvjetima 3.0

# Uvod (1)

- poravnavanje – najčešći postupak u bioinformatici
- BLAST: usporedba upitnog slijeda s velikim skupom sljedova (bazom podataka)
- je li BLAST prikladan alat za poravnanje dva genoma (npr. duljine  $10^6$ bp ili  $10^9$ bp)?
- poravnanje dugačkih (blisko srodnih) genoma:
  - MUMmer (Marçais et al. 2018, Kurtz *et al.*, 2004)
  - MegaBLAST (Morgulis *et al.*, 2008)

# Uvod (2)

## Motivacija

- želimo indeksirati velike količina teksta, npr. skup nukleotidnih ili aminokiselinskih sljedova

## Pitanje

Postoji li struktura podataka koja može pronaći podniz  $P$  u tekstu  $S$  u linearnom vremenu ovisno o duljini podniza, a neovisno o duljini teksta koji se pretražuje?

# Osnovni pojmovi

- neka je  $S$  niz znakova duljine  $n$ , gdje su znakovi elementi abecede  $\Sigma$
- broj znakova abecede  $\Sigma$  je  $\sigma$
- znak na  $i$ -tom mjestu u nizu  $S$  je  $S[i]$  ili  $S_i$ ,  $1 \leq i \leq n$
- **podniz** od  $S$  je  $S[i, j]$ ,  $1 \leq i, j \leq n$
- **prefiks** od  $S$  je podniz  $S[1, j]$ ,  $1 \leq j \leq n$
- **sufiks** od  $S$  je podniz  $S[i, n]$  ili kraće  $s_i$ ,  $1 \leq i \leq n$

# Problem traženja podniza u nizu (1)

Neka su zadani niz  $S$  i podniz  $P$

(još se koriste nazivi:  $S$  – tekst;  $P$  – uzorak)

## Problem

- kako pronaći sva pojavljivanja podniza  $P$  u nizu  $S$   
(eng. *string matching problem*)?
- što ako želimo više puta ponoviti traženje  $P$  u  $S$ ?

# Problem traženja niza u podnizu (2)

- pretprocesirati S:
  - tekst/niz S, koji je (približno) vremenski konstantan
  - pretražuje se različite uzorke/regularne izraze (eng. *pattern*; *regular expression*)
- podatkovna struktura koja to omogućuje:  
sufiksno stablo

# *Trie* ili prefiksno stablo (1)

- Prefiksno stablo (eng. *prefix tree*; *trie*)
  - uređena podatkovna struktura u obliku stabla, gdje su ključevi podataka (obično) znakovni nizovi (E. Fredkin, 1960.)
  - npr. pohrana rječnika ili popisa ključnih riječi
  - izvorni naziv: ***trie*** (E. Fredkin, 1960.)
    - dolazi od *retrieval*

# *Trie* ili prefiksno stablo (2)

## **Povijesna crtica**

(<http://xlinux.nist.gov/dads//HTML/trie.html>)

"As defined by me, nearly 50 years ago, it is properly pronounced "tree" as in the word "retrieval".

At least that was my intent when I gave it the name "Trie". The idea behind the name was to combine reference to both the structure (a tree structure) and a major purpose (data storage and retrieval)."

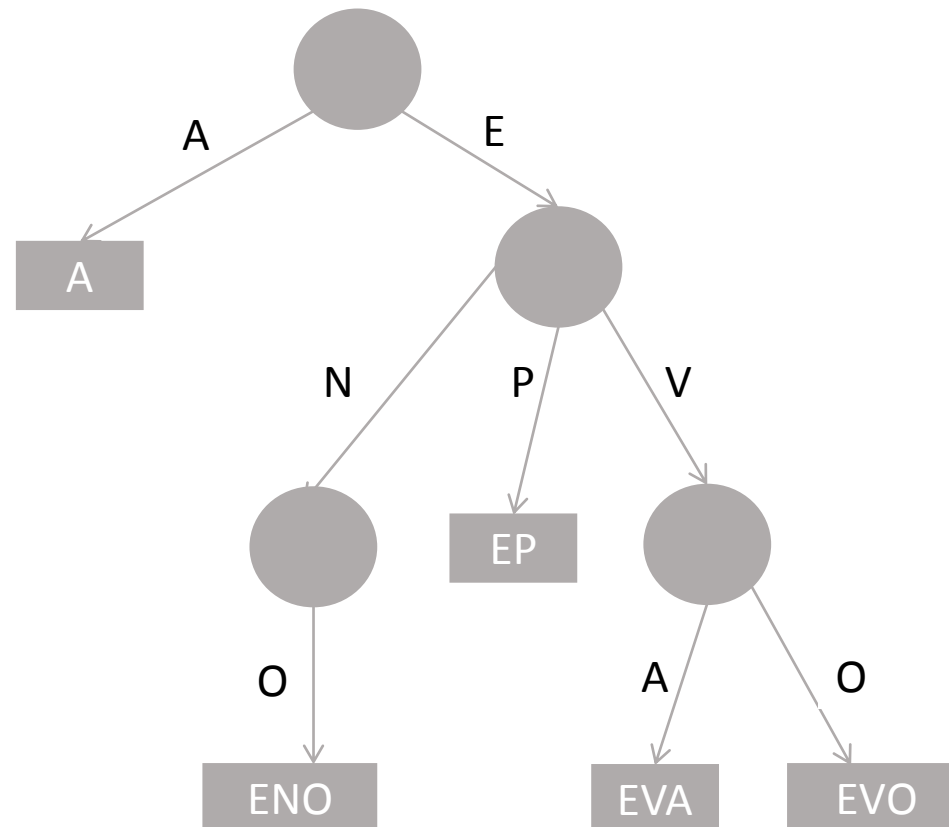
(E. Fredkin, 2008)



# Prefiksno stablo

- podatkovna struktura za pohranu znakovnih nizova tako da znakovi na putu od korijena do nekog čvoru stabla predstavljaju *prefiks* jednog ili više znakovnih nizova nad kojima je stablo izgrađeno
- prazan niz se pohranjuje u korijenu stabla
- jedan znak se pohranjuje na jednoj razini stabla
- kompaktno prefiksno stablo (eng. *compact prefix tree*)
  - prostorna učinkovitost: svaki čvor-roditelj, koji bi imao samo jedno dijete, automatski je spojen s čvorom-djetetom

Prefiksno stablo za riječi: "A", "EVO", "EVA", "EP", "ENO"



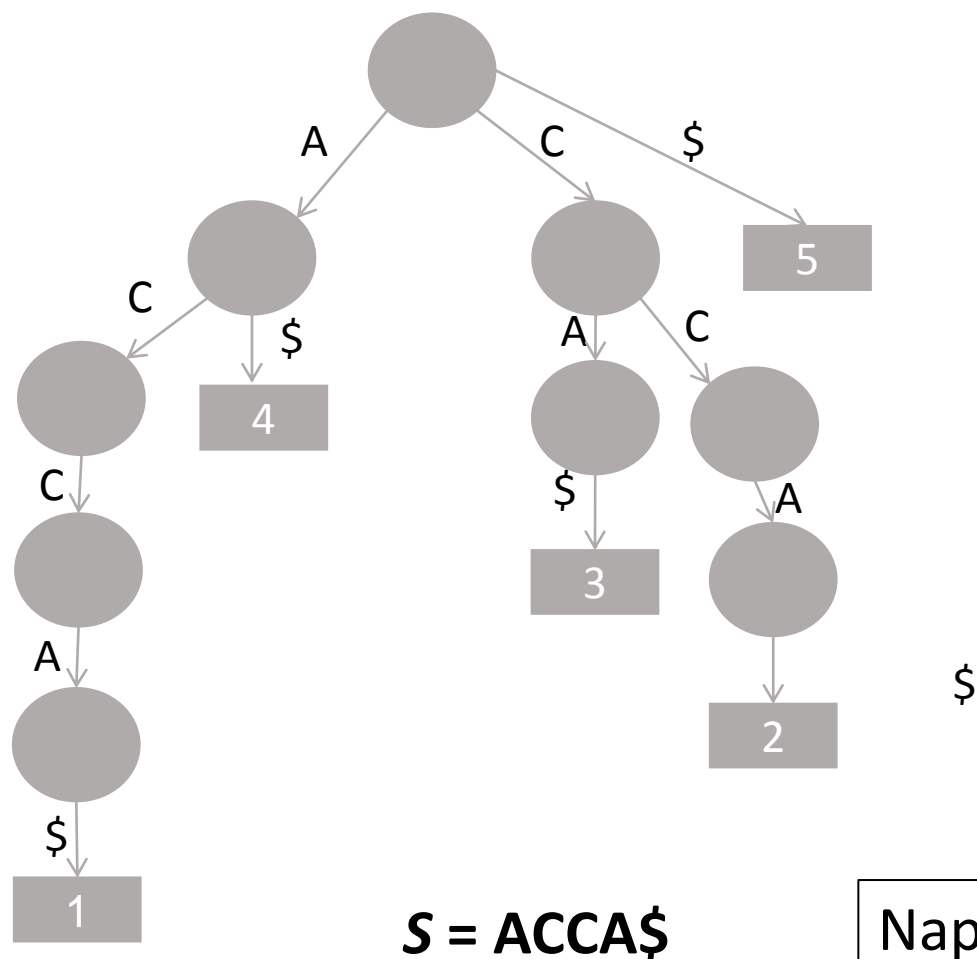
# Sufiksno stablo (1)

- neka je zadan niz  $S$  duljine  $n$ , gdje su znakovi niza iz abaceđe  $\Sigma$
- na kraj niza  $S$  dodaje se znak za kraj niza koji ne postoji u  $\Sigma$  (obično se koristi \$)
- za svaki sufiks: točno jedan list u  $T$ ,  
tj. niti jedan sufiks neće biti prefiks nekog drugog

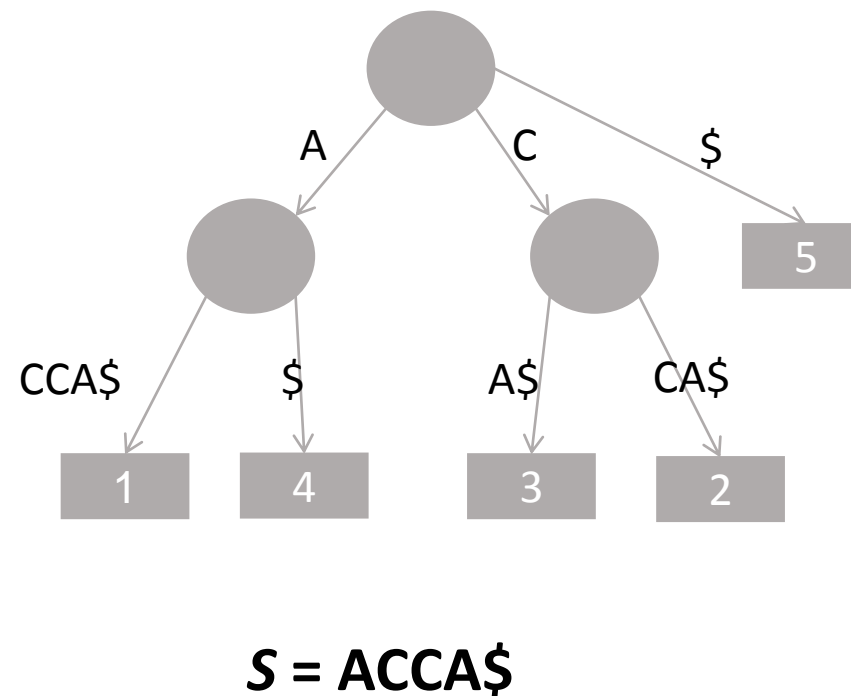
# Sufiksno stablo (2)

- **sufiksno stablo  $T$**  sadrži sve sufikse niza  $S$  tako da je svaki sufiks pridružen jednom listu
- *suffix **trie***: svaka grana stabla sadrži samo jedan znak
- *suffix **tree***: svaka grana stabla može sadržavati jedan ili više znakova
  - sažimanje unutarnjih čvorova koji su u *suffix trie* imali samo jedno dijete

## Suffix trie



## Sufiksno stablo (suffix tree)



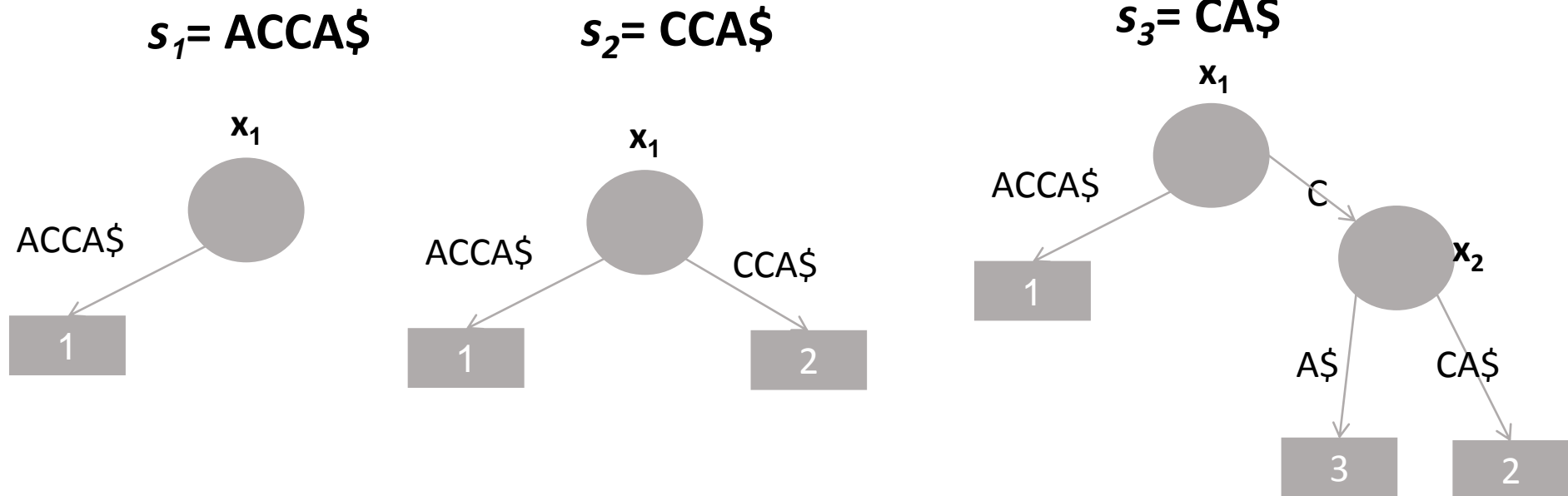
Napomena: \$ je abecedno veći od svih znakova iz  $\Sigma$

# Sufiksno stablo (3)

- **listovi** (eng. *leaf; terminal node*) stabla  $T$  su numerirani 1 do  $n$ 
  - u listovima sufiksnog stabla su pohranjeni indeksi (početne pozicije) sufiksa u nizu  $S$
- **korijen stabla** ima onoliko čvorova djece koliko je znakova abecede + jedan čvor za \$
- svaki **unutarnji čvor** (eng. *inner node; branch node*) ima  $\geq 2$  djece
- svaki **brid/grana** (eng. *edge; branch*) stabla je označena jednim podnizom niza  $S$
- **znakovna dubina** (eng. *string depth*) čvora  $x$ :  
duljina svih znakova na putu od korijena do  $x$
- **dubina čvora** (eng. *node-depth*)  $x$ :  
broj čvorova na putu od korijena do čvora  $x$

# Izgradnja sufiksnog stabla u vremenu $O(n^2)$ (1)

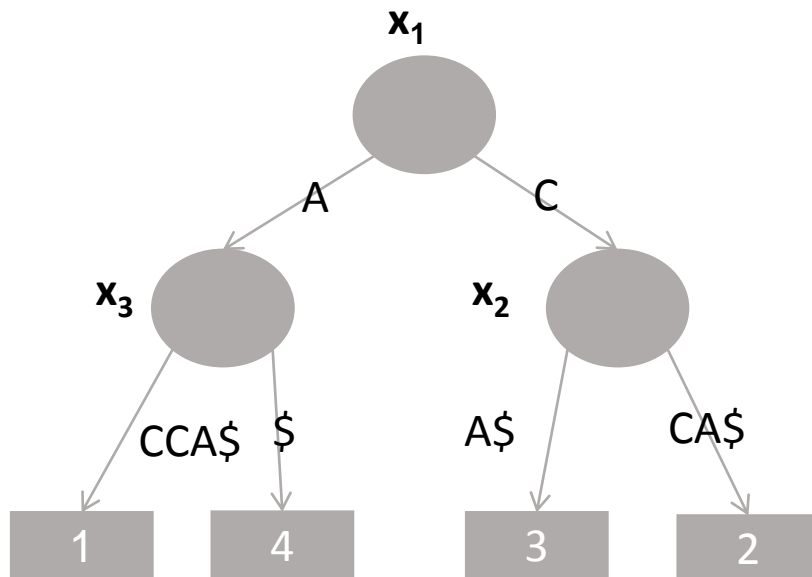
$S = \text{ACCA}\$, n = |S|$



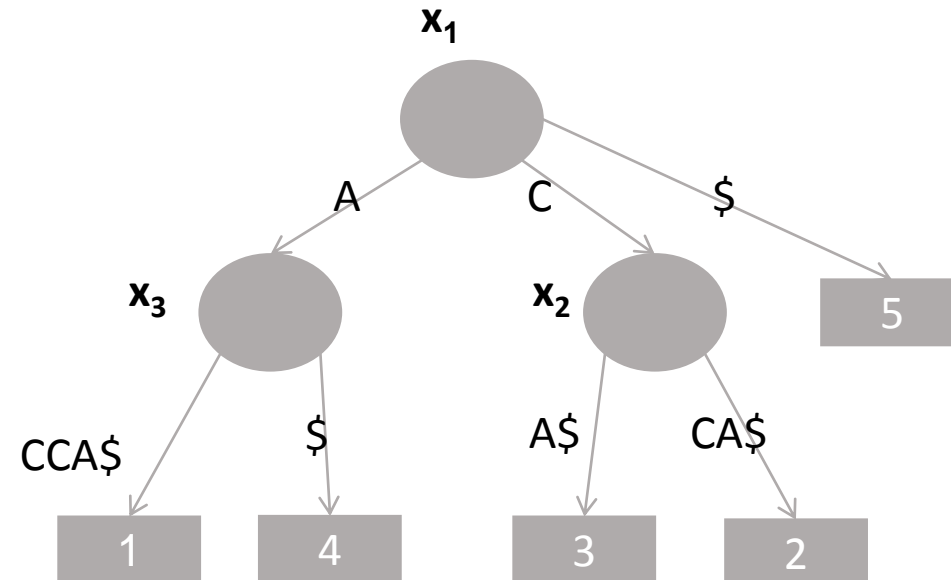
Napomena: \$ je abecedno veći od svih znakova iz  $\Sigma$

# Izgradnja sufiksnog stabla u vremenu $O(n^2)$ (2)

$s_4 = A\$$



$s_5 = \$$



Napomena: \$ je abecedno veći od svih znakova iz  $\Sigma$



# Izgradnja sufiksnog stabla u $O(n)$ vremenu

Sufiksno stablo može biti izgrađeno u vremenu  $O(n)$  za niz duljine  $n$  čiji su znakovi iz abecede konstantne duljine:

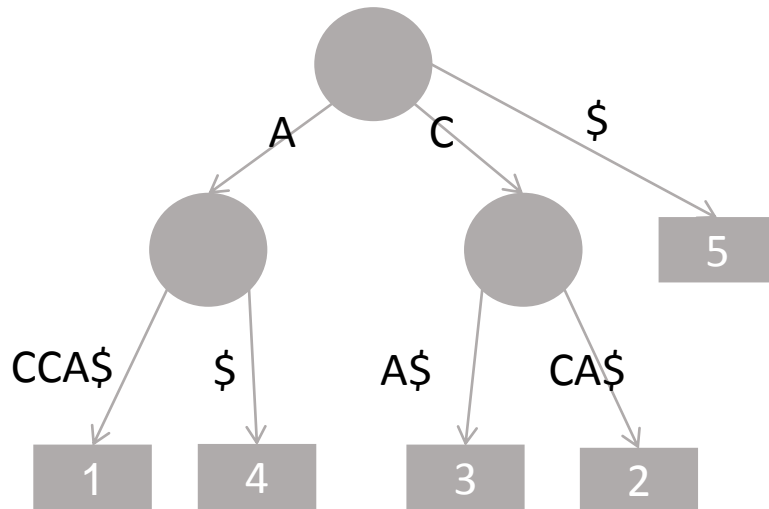
- Weiner, 1973 ("*Algorithm of the Year 1973*", Knuth)
- McCreight, 1976
- Ukkonen, 1995 → *on-line* algoritam za konstrukciju sufiksnog stabla u  $O(n)$  vremenu
  - postupna izgradnja stabla (znakovi se dodaju jedan po jedan kako dolaze – slijeva nadesno)
  - ideja se temelji na izgradnji implicitnog sufiksnog stabla i korištenju sufiksni veza (eng. *suffix links*)

# Implicitno sufiksno stablo

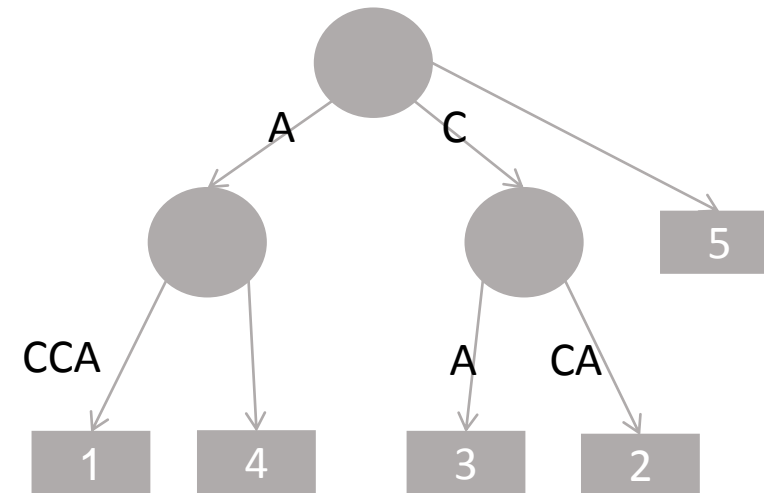
- Kako iz sufiksnog stabla izgraditi implicitno sufiksno stablo?
  1. Obrisati oznake za kraj niza \$.
  2. Ukloniti grane koje sadrže samo prazan niz.
  3. Sve unutarnje čvorove, koji više nemaju 2 djeteta, maknuti iz stabla (sažimanje unutarnjih čvorova koji su imali samo jedno dijete)

# Implicitno sufiksno stablo – primjer (1)

**$S = \text{ACCA}\$$**



1. Obrisati oznake za kraj niza \$.

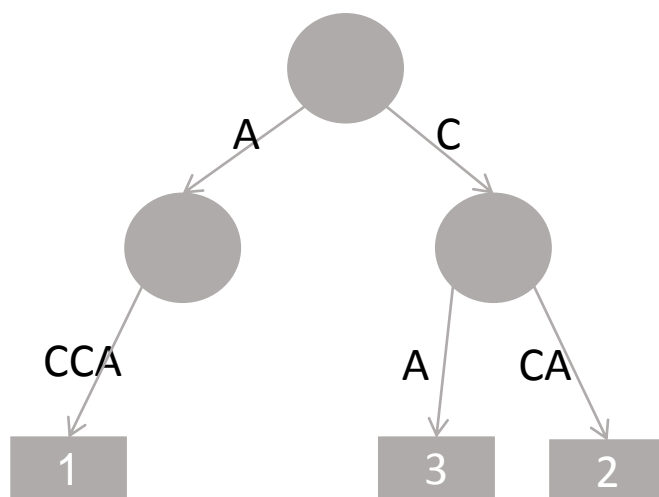


Napomena: \$ je abecedno veći od svih znakova iz  $\Sigma$

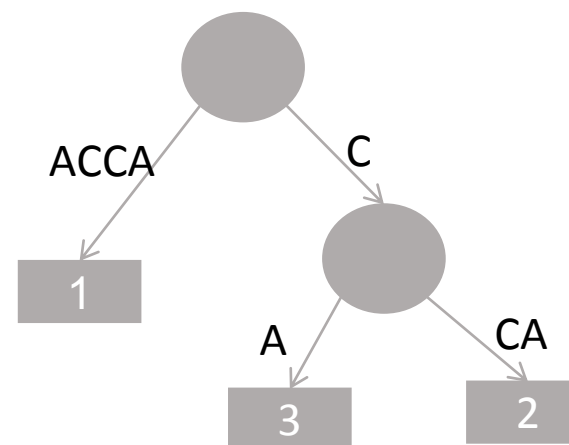
# Implicitno sufiksno stablo – primjer (2)

**$S = \text{ACCA}\$$**

2. Ukloniti grane koje sadrže samo prazan niz.



3. Sve unutarnje čvorove, koji više nemaju 2 djeteta, maknuti iz stabla.



# Sufiksne veze

## Sufiksna veza:

pokazivač od čvora  $v$  do čvora  $s(v)$ , gdje je

**1.  $xa$**  - oznaka puta do  $v$

$x$  je znak,  $a$  podniz ( $a$  može biti prazan niz  $\varepsilon$ )

**2.  $a$**  - oznaka puta do  $s(v)$

$a$  podniz (može biti prazan niz  $\varepsilon$ )

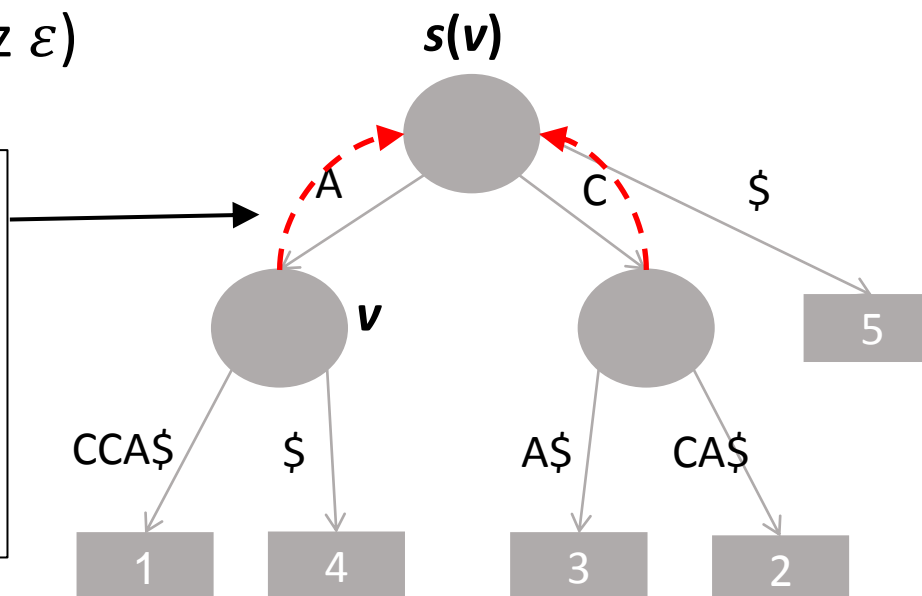
### Primjer:

1. Oznaka puta do  $v$ :

$xa = A$  ( $x = A$ ,  $a = \varepsilon$ )

2. Oznaka puta do  $s(v)$ :

$a = \varepsilon$

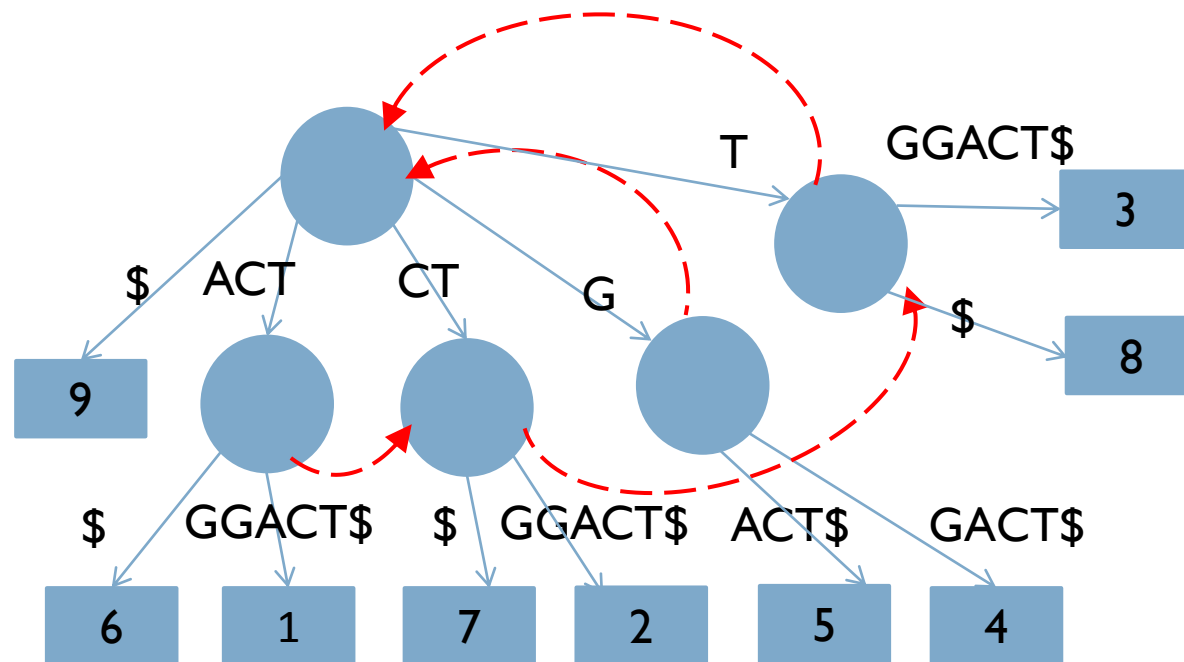


# Sufiksne veze - primjer

Za zadani niz  $S = \text{ACTGGACT}\$$  potrebno je nacrtati sufiksno stablo.

Pretpostavite da je znak  $\$$  leksikografski manji od ostalih znakova niza  $S$ .

1	2	3	4	5	6	7	8	9
A	C	T	G	G	A	C	T	\$



# Ukkonenov algoritam – ideja (1)

- grade se implicitna sufiksna stabla  $T_i$  za svaki **prefiks**  $S[1, i]$ ,  
 $i = 1, \dots, n = |S|$ 
  - prvo se gradi  $T_1$  za prefiks  $S[1, 1]$  (tj. dodaje se sufiks  $S[1, 1]$ )
  - zatim se, proširenjem  $T_1$ , gradi  $T_2$  tako da se dodaju sufiksi prefiksa  $S[1, 2]$ :  
 $S[1, 2]$  i  $S[2, 2]$
  - postupak se ponavlja za  $T_3, \dots, T_n$

## Općenito:

- $i$ -ti korak: gradi se  $T_i$  za sufikse podniza  $S[1, i]$  (ukupno  $i$  sufiksa)
- $(i + 1)$  korak: gradi se  $T_{i+1}$  tako da se  $T_i$  proširuje za sufikse podniza  $S[1, i + 1]$

# Ukkonenov algoritam – ideja (2)

- u  $(i + 1)$ -koraku dodaju se sufiksi kroz  $i + 1$  proširenja (eng. *extension*):
  - za svako  **$j$ -to proširenje** ( $1 \leq j \leq i+1$ ) sufiks se dodaje:
    - produljivanjem grane (brida) do lista dodavanjem novog znaka na oznaku postojeće grane (pravilo 1), ili
    - dodavanjem nove grane (pravilo 2)
  - ako sufiks već postoji u stablu, ne obavlja se ništa (pravilo 3)
- posljednje izgrađeno implicitno sufiksno stablo (za zadnji znak u nizu  $S$ , tj. za  $\$$ ) je pravo sufiksno stablo



# Ukkonenov algoritam – naivna izvedba

Izgradi  $T_1$

**za**  $i = 1$  **do**  $n - 1$  **radi**

/\* korak  $i + 1$ : izgradnja  $T_{i+1}$  iz  $T_i$  \*/

**za**  $j = 1$  **do**  $i + 1$  **radi** /\*  $j$ -to proširenje \*/

/\* osigurati da  $S[j, i + 1]$  bude u stablu korištenjem pravila 1, 2 ili 3 \*/

U trenutnom stablu pronaći kraj puta koji počinje u korijenu, a označen je s  $S[j, i]$ .

Ako je potrebno, dodati na kraj puta znak  $S[i + 1]$  .

**kraj**

**kraj**

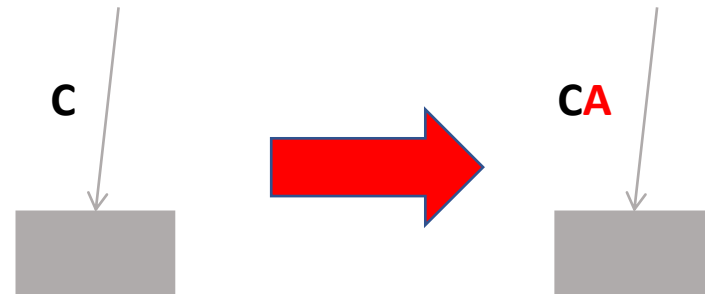
# Ukkonenov algoritam: pravilo 1

- Implicitno proširenje

- ako put  $S[j, i]$  završava kao list, onda se dodaje  $S[i + 1]$  na kraj oznake grane koja vodi do lista
- *jednom list, uvijek list*, tj. grana koja završava u listu, uvijek će završavati u listu

- Primjer:

$$S[j, i] = C \rightarrow S[j, i + 1] = CA$$



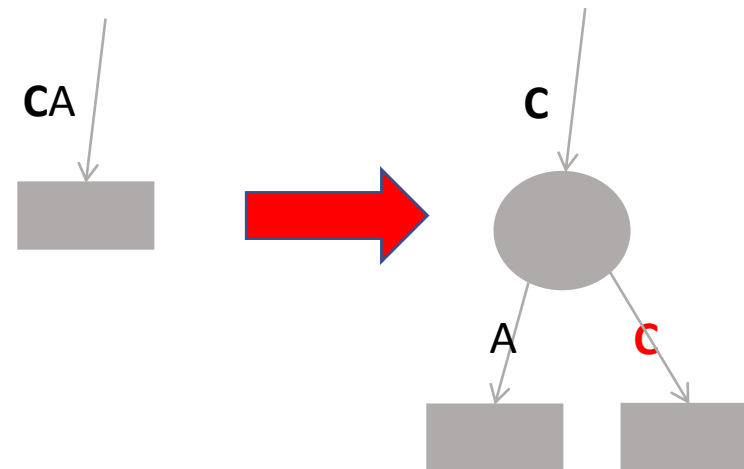
# Ukkonenov algoritam: pravilo 2

- EksPLICITNO dodavanje novog lista:  
ako postoji put  $S[j, i]$  i iz njega se ne nastavlja niti jedan put koji počinje s  $S[i + 1]$ , onda treba dodati novu granu koja vodi do lista, a koja će biti označena s  $S[i + 1]$
- pri tome, ako  $S[j, i]$  završava unutar grane, onda treba dodati i novi unutarnji čvor

- Primjer:

Želimo dodati  $S[j, i + 1] = \mathbf{CC}$ :

$$S[j, i] = \mathbf{C} \rightarrow S[j, i + 1] = \mathbf{CC}$$



# Ukkonenov algoritam: pravilo 3

- ako u stablu postoje putevi koji se nastavljaju na  $S[j, i]$ , a jedan od njih počinje s  $S[i + 1]$ ,  
onda ne treba učiniti ništa, tj.  $S[j, i + 1]$  već postoji u stablu

# Primjer: Ukkonenov alg. – naivna izvedba

**$S = \text{ACCA}\$$**

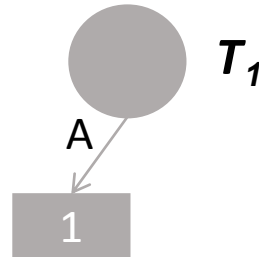
Pravilo 1: produljivanje

oznake grane do lista

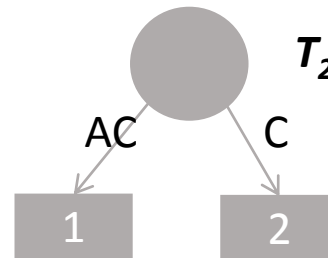
Pravilo 2: dodavanje nove grane

Pravilo 3: ako sufiks već postoji u stablu, ništa se ne obavlja

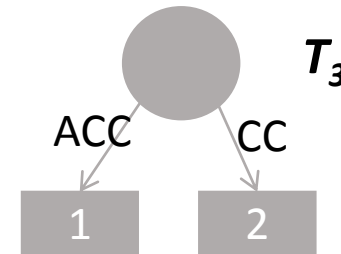
$S[1, 1] = A$



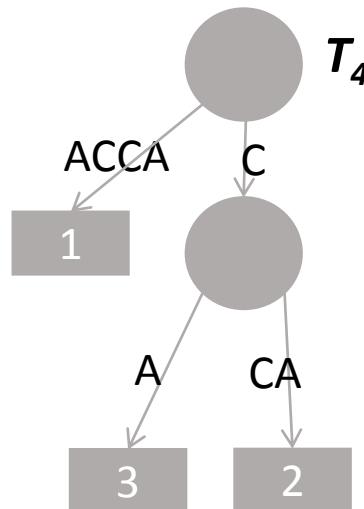
$S[1, 2] = AC$



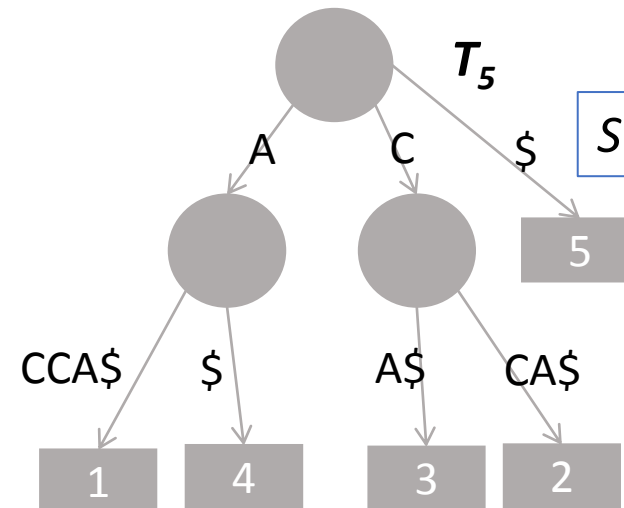
$S[1, 3] = ACC$



$S[1, 4] = \text{ACCA}$



$S[1, 5] = \text{ACCA}\$$



# Ukkonenov algoritam – vremenska složenost

- ukupno se gradi  $n$  stabala
- izgradnja stabla  $T_{i+1}$  ima  $(i + 1)$  koraka (zbog dodavanja  $i + 1$  sufiksa)
- u  $(i + 1)$  koraku dodaje se  $i + 1$  sufiksa:
  - dodavanje  $j$ -tog sufiksa, gdje je  $1 \leq j \leq i + 1$ :  
→ pronaći put  $S[j, i]$  na koji se dodaje  $S[i + 1]$  (ako već ne postoji u stablu)
  - složenost:  $O(i + 1 - j)$
- ukupna složenost naivne izvedbe:  $O(n^3)$
- složenost se smanjuje na  $O(n)$  korištenjem sufiksni veza i još nekih poboljšanja

# Poboljšanja algoritma

- Pravila za produljenja sufiksa:

- (1) "jednom list, uvijek list" (eng. *once a leaf, always a leaf*)
- (2) u  $(i + 1)$ -koraku: eksplicitno proširenje samo za  $j \geq j_{pret} + 1$
- (3) uvjet prekida  $(i + 1)$ -koraka (eng. "*show-stopper*")

- Kod eksplicitnog dodavanja sufiksa koriste se:

- (1) sufiksne veze
- (2) trik *preskoči i prebroji* (eng. *skip and count*) - za dodavanje sufiksa  $S[j, i]$  iz sufiksa  $S[j, i - 1]$ , koji je već u stablu

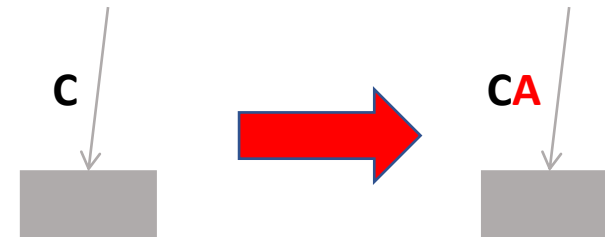
# Ukkonenov algoritam: pravilo 1 (1)

- Implicitno proširenje

- ako put  $S[j, i]$  završava kao list, onda se dodaje  $S[i + 1]$  na kraj oznake grane koja vodi do lista
- *jednom list, uvijek list*, tj. grana koja završava u listu, uvijek će završavati u listu

- Primjer:

$$S[j, i] = C \rightarrow S[j, i + 1] = C\textcolor{red}{A}$$

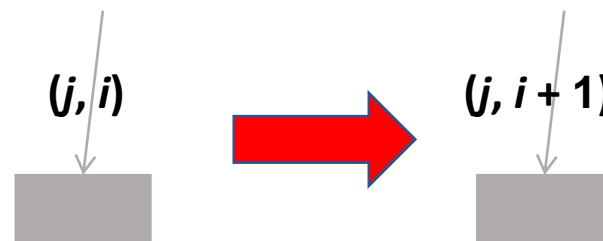




# Ukkonenov algoritam: pravilo 1 (2)

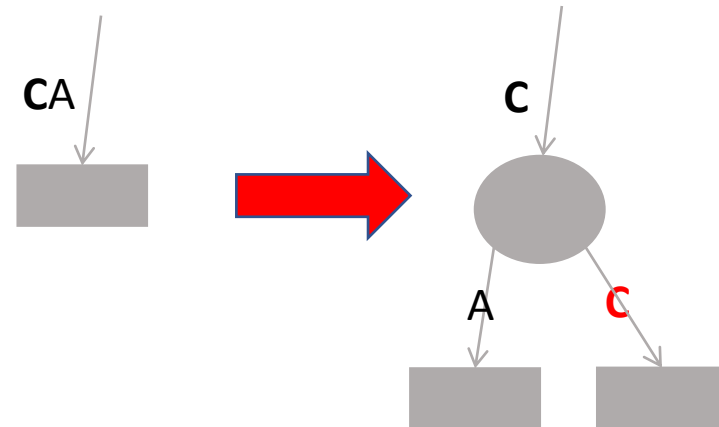
UNAPRJEĐENJE → implicitno proširenje

- grane koje vode do listova, označavaju se, umjesto znakovima, indeksima  $p$  i  $k$ , tj. *početkom* i *krajem* podniza  $S[p, k]$  koji je oznaka te grane (eng. *edge-label compression*)
- Posljedica:
  - za listove se u  $(i + 1)$ -koraku dodaje novi znak na oznaku grane koja vodi do lista tako da se indeks  $k = i$  promijeni u  $k = i + 1$
  - obavlja se u  $O(1)$  vremenu



# Ukkonenov algoritam: pravilo 2 (1)

- EksPLICITNO dodavanje novog čvora:  
ako postoji put  $S[j, i]$  i iz njega se ne nastavlja niti jedan put koji počinje s  $S[i + 1]$ ,  
onda treba dodati novu granu koja vodi do lista,  
a koja će biti označena s  $S[i + 1]$
- pri tome, ako  $S[j, i]$  završava unutar grane, onda treba dodati i novi unutarnji čvor
- Primjer:  
 $S[j, i] = \mathbf{C} \rightarrow S[j, i + 1] = \mathbf{CC}$



# Ukkonenov algoritam: pravilo 2 (2)

## UNAPRJEĐENJE:

- u  $(i + 1)$ -koraku: neka indeks  $j_{pret}$  označava list koji je posljednji eksplicitno dodan u  $i$ -tom koraku
- kandidati za eksplicitno proširenje u  $(i + 1)$ -koraku su:  $j \geq j_{pret} + 1$
- s obzirom da Ukkonenov algoritam ukupno ima  $n$  koraka,  $j$  je ograničen s  $n$   
→ obavlja se  $n$  eksplicitnih proširenja ukupno kroz sve korake!

# Ukkonenov algoritam: pravilo 3 (1)

- ako u stablu postoje putevi koji se nastavljaju na  $S[j, i]$ ,  
a jedan od njih počinje s  $S[i + 1]$ ,  
onda ne treba učiniti ništa, tj.  $S[j, i + 1]$  već postoji u stablu
- Posljedica:  
ako je  $S[j, i + 1]$  u stablu,  
onda su također i  $S[j + 1, i + 1], \dots, S[i + 1, i + 1]$   
→ Pravilo 3 nam omogućuje završetak  $(i + 1)$ -koraka  
(eng. "*show-stopper*")

# Ukkonenov algoritam: pravilo 3 (2)

## UNAPRJEĐENJE:

$(i + 1)$ -korak završavamo kada  $j > i + 1$

ili

za prvi  $j$  za koji vrijedi pravilo 3

# Dubina čvora $s(v)$ - lema

## Lema:

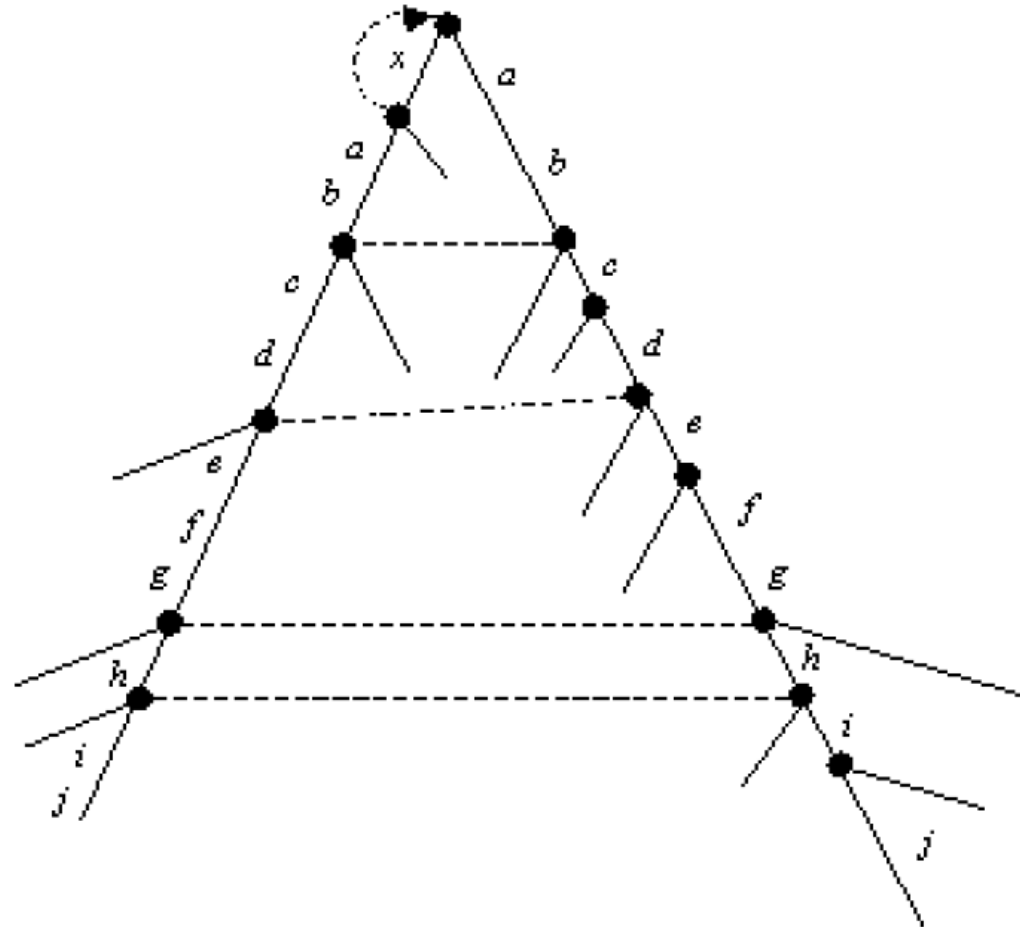
Neka je  $(v, s(v))$  sufiksna veza od čvora  $v$  prema čvoru  $s(v)$ .

Tada vrijedi da je **dubina čvora** (eng. *node-depth*)  $v$  najviše za jedan veća od dubine čvora  $s(v)$ , tj.

$$ND(v) \leq ND(s(v)) + 1$$

Dubina čvora  $s(v)$  može biti:

- manja za jedan od dubine čvora  $v$
- jednaka dubini čvora  $v$
- veća od dubine čvora  $v$



Za svaki čvor  $v$  s oznakom puta  $x\alpha$  ( $\alpha$  je niz), pripadajući čvor  $s(v)$  ima oznaku puta  $\alpha$ .

Npr. čvor s oznakom puta  $xab$  ima dubinu čvora 2, dok je dubina čvora s oznakom puta  $ab$  jedan. Čvor s oznakom puta  $abcdefg$  ima dubinu četiri, itd.

# Sufiksne veze - lema

Ako je u  $j$ -tom proširenju  $(i + 1)$ -koraka dodan novi unutarnji čvor  $v$ , čija je oznaka puta  $x\alpha$ , onda

- (1) oznaka puta  $\alpha$  već postoji u stablu, ili
- (2) će novi čvor, s oznakom puta  $\alpha$ , biti kreiran u  $(j + 1)$ -proširenju  $(i + 1)$ -koraka (Pravilo 2), ili
- (3)  $\alpha = \varepsilon$  i  $s(v)$  je korijen stabla

## Posljedica:

- svaki unutarnji čvor  $v$  će imati sufiksnu vezu, koja polazi iz  $v$ , do kraja sljedećeg proširenja, tj. u  $(j + 1)$ -koraku



# Ukkonenov alg.: dodavanje sufiksa korištenjem sufiksni veza (1)

Proširenje  $j \geq 2$  u koraku  $i + 1$  (Gusfield, 1997, Ch. 6)

1. Pronaći prvi čvor  $v$  iznad ili točno na  $S[j - 1, i]$  tako da iz  $v$  polazi sufiksna veza ili je  $v$  korijen stabla.

Ovaj korak zahtijeva "hodanje prema gore" po najviše jednom bridu.

Neka  $\gamma$  predstavlja znakovni niz između  $v$  i  $S[j - 1, i]$  ( $\gamma$  može biti  $\epsilon$ ).

2. Ako  $v$  nije korijen stabla, onda se korištenjem sufiksne veze od  $v$  do  $s(v)$  pomaknuti do čvora  $s(v)$  te se zatim spustiti po putu korištenjem niza  $\gamma$ .

Ako je  $s(v)$  korijen stabla, onda slijediti put  $S[j, i]$  iz korijena (kao u naivnom algoritmu).

3. Korištenjem pravila o proširenju, osigurati da niz  $S[j, i]S[i + 1]$  bude u stablu.

4. Ako je u proširenju  $j - 1$  dodan novi čvor  $w$  s oznakom puta  $x\alpha$ , onda niz  $\alpha$  mora završavati u  $s(w)$  i treba dodati sufiksnu vezu  $(w, s(w))$  iz  $w$  u  $s(w)$ .

# Ukkonenov alg.: dodavanje sufiksa korištenjem sufiksni veza (2)

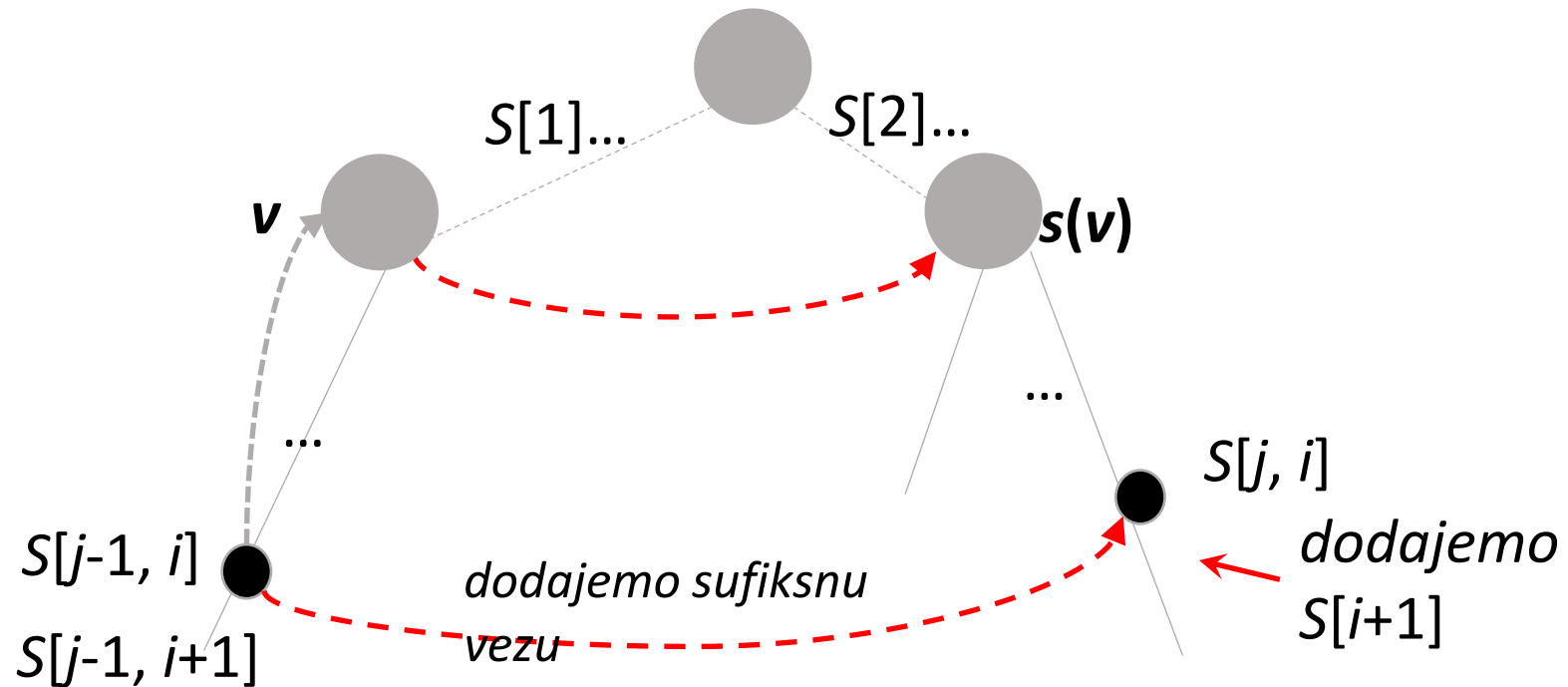
- prvo proširenje u  $(i + 1)$ -koraku:
  - nađemo kraj puta čija je oznaka  $S[1, i]$  u stablu i onda ga proširujemo za  $S[i + 1]$  tako da dobijemo  $S[1, i + 1]$
- za drugo proširenje u  $(i + 1)$ -koraku:
  - umjesto da krećemo od korijena tražiti  $S[2, i + 1]$ , tražimo čvor  $v$  ispod kojeg završava put označen s  $S[1, i]$  (jedino ako nema  $v$ , onda krećemo od korijena)
  - kada smo pronašli  $v$ , onda slijedimo sufiksnu vezu  $(v, s(v))$ , gdje je  $s(v)$  čvor ispod kojeg završava  $S[2, i]$ 
    - traženje nastavljamo od  $s(v)$  i onda dodamo  $S[i + 1]$  iza  $S[2, i]$
    - preskočili smo znakove na putu od korijena do  $s(v)$ !

# Ukkonenov alg.: dodavanje sufiksa korištenjem sufiksni veza (3)

Brzo traženje (eng. *fastscan*):

nakon što smo dodali  $S[j - 1, i + 1]$ , želimo dodati  **$S[j, i + 1]$**

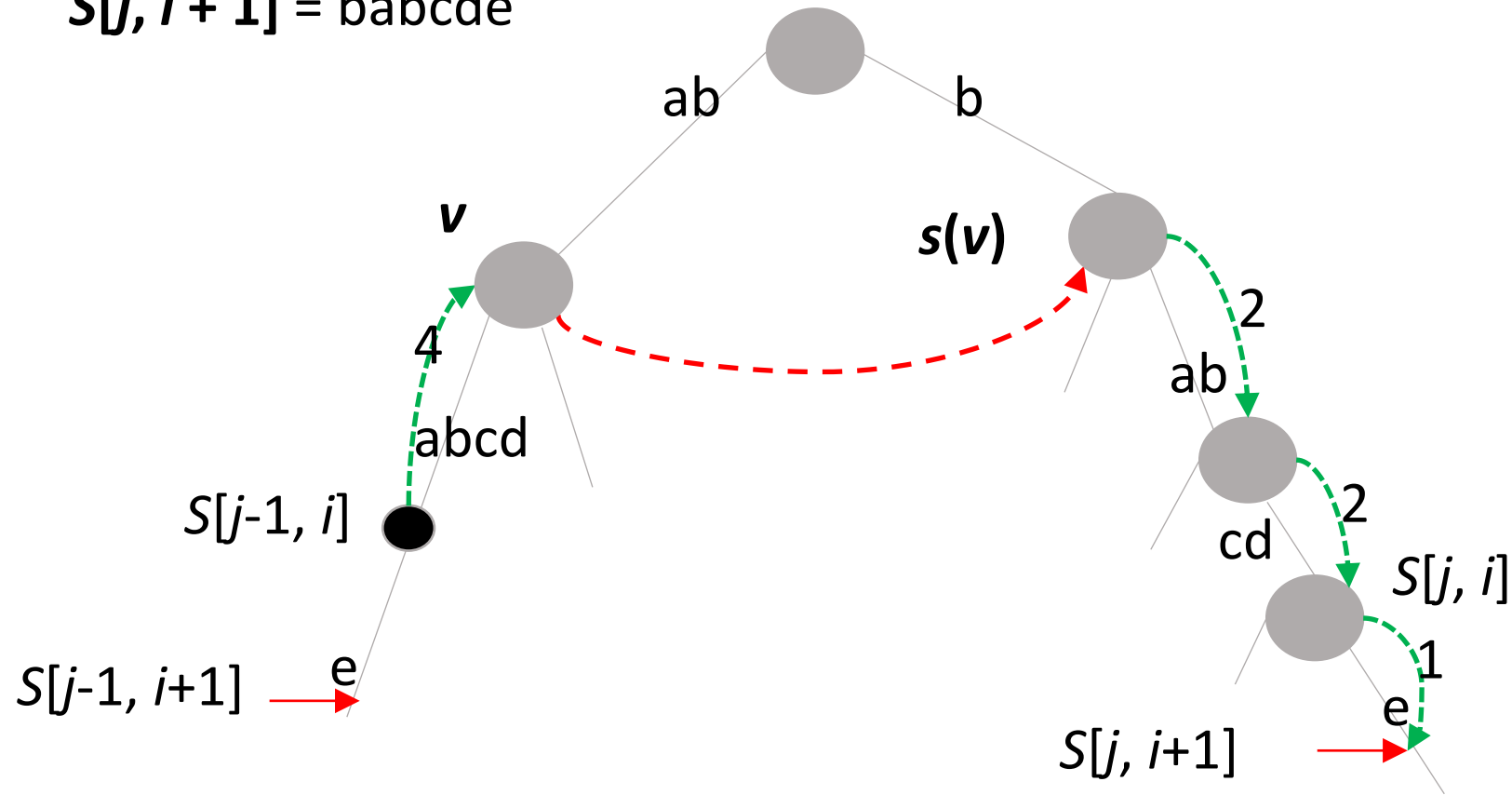
- pomicanje do čvora  $v$  ispod kojeg je  $S[j - 1, i]$  :  $O(1)$
- pomicanje od  $v$  do  $s(v)$  ispod kojeg je  $S[j, i]$  :  $O(1)$



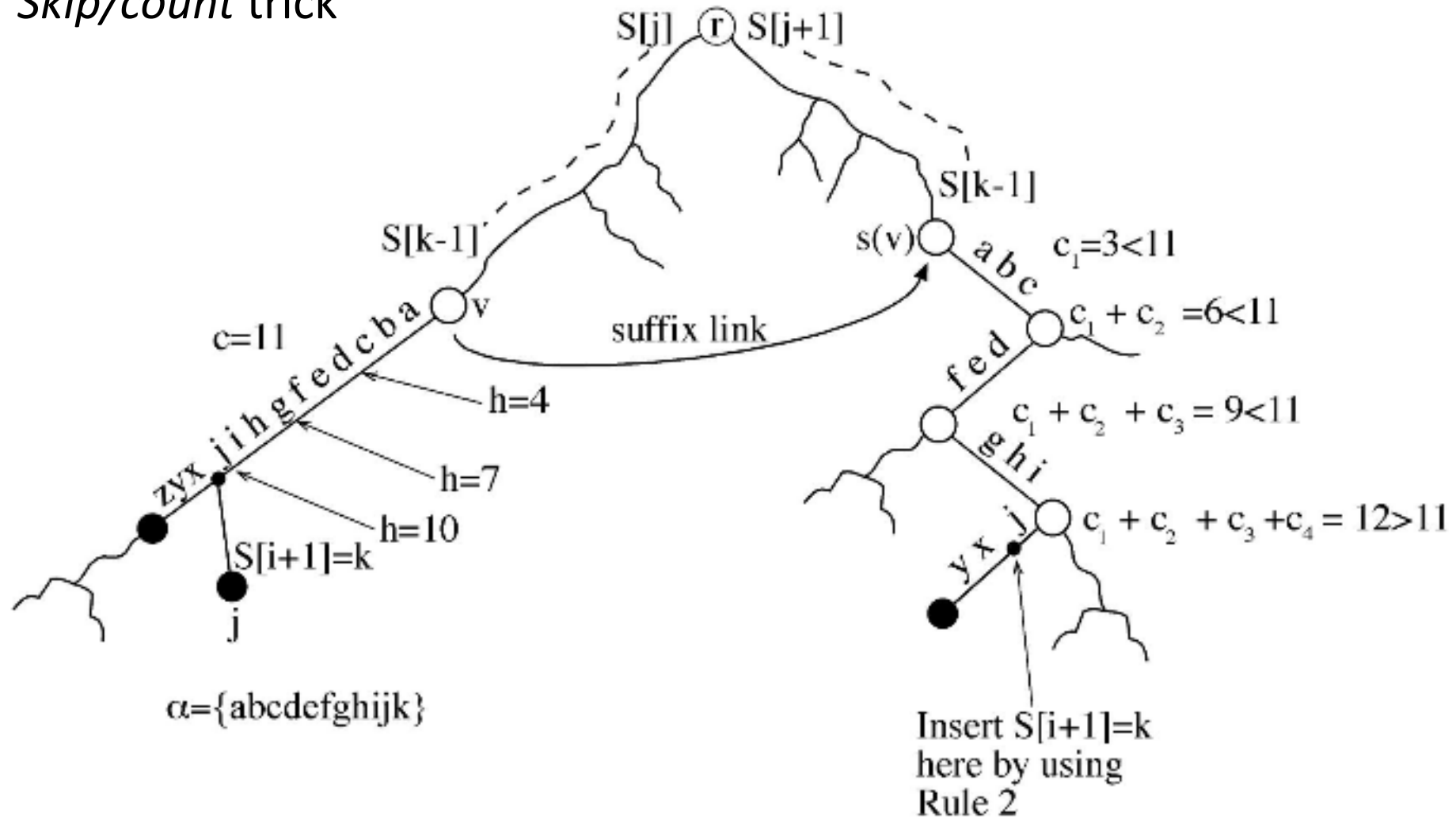
# Ukkonenov alg.: dodavanje sufiksa korištenjem sufiksnihi veza (4)

$S[j - 1, i + 1] = \text{ababcde}$

$S[j, i + 1] = \text{babcdde}$



## Skip/count trick



<http://www.cs.duke.edu/courses/fall14/compsci260/resources/suffix.trees.in.detail.pdf>

f

# Ukkonenov alg.: dodavanje sufiksa korištenjem sufiksni veza (5)

- *skip/count*: umjesto uspoređivanja znakova, "skakanje" od čvora do čvora prema dolje
- brzo traženje (eng. *fastscan*) od  $s(v)$  prema dolje:
  - pronaći odgovarajuću granu koja ide iz čvora i ako je broj znakova na grani zajedno s prethodno uspoređivanim znakovima  $< |\gamma|$ , onda se pomiče na čvor-dijete
  - dovoljno je usporediti samo prvi znak na grani
- kada se zbroji broj "hodanja" po čvorovima prema dolje u stablu (eng. *down-walk*), ukupan broj po svim koracima je  $O(n)$  za niz duljine  $n$  (niti jedan čvor u stablu nema dubinu  $> n$ )

# Ukkonenov algoritam u vremenu $O(n)$

Izgradi  $T_1$

$j_{\text{pret}} = 1$

**za**  $i = 1$  **do**  $n - 1$  **radi** /\* ukupno:  $n$  eksplicitnih proširenja,  $|S| = n$  \*/

/\* korak  $i + 1$ : izgradnja  $T_{i+1}$  iz  $T_i$  \*/

Obavi implicitna proširenja, tj.  $k = i + 1$  /\* Pravilo 1 za  $j \leq j_{\text{pret}}$  \*/

**za**  $j = j_{\text{pret}} + 1$  **do**  $i + 1$  **radi** /\*  $j$ -to proširenje \*/

/\* osiguravamo da je  $S[j, i + 1]$  u stablu: Pravilo 2 ili 3 \*/

U trenutnom stablu pronađi kraj puta koji počinje u korijenu, a označen je s  $S[j, i]$ .

Ako je potrebno, dodaj na kraj puta znak  $S[i + 1]$ . /\* Pravilo 2 \*/

$j_{\text{pret}} := j$  /\* priprema za sljedeći korak \*/

**ako** je primijenjeno Pravilo 3

**onda**  $j_{\text{pret}} := j - 1$  i završi  $(i + 1)$ -korak

**kraj**

**kraj**

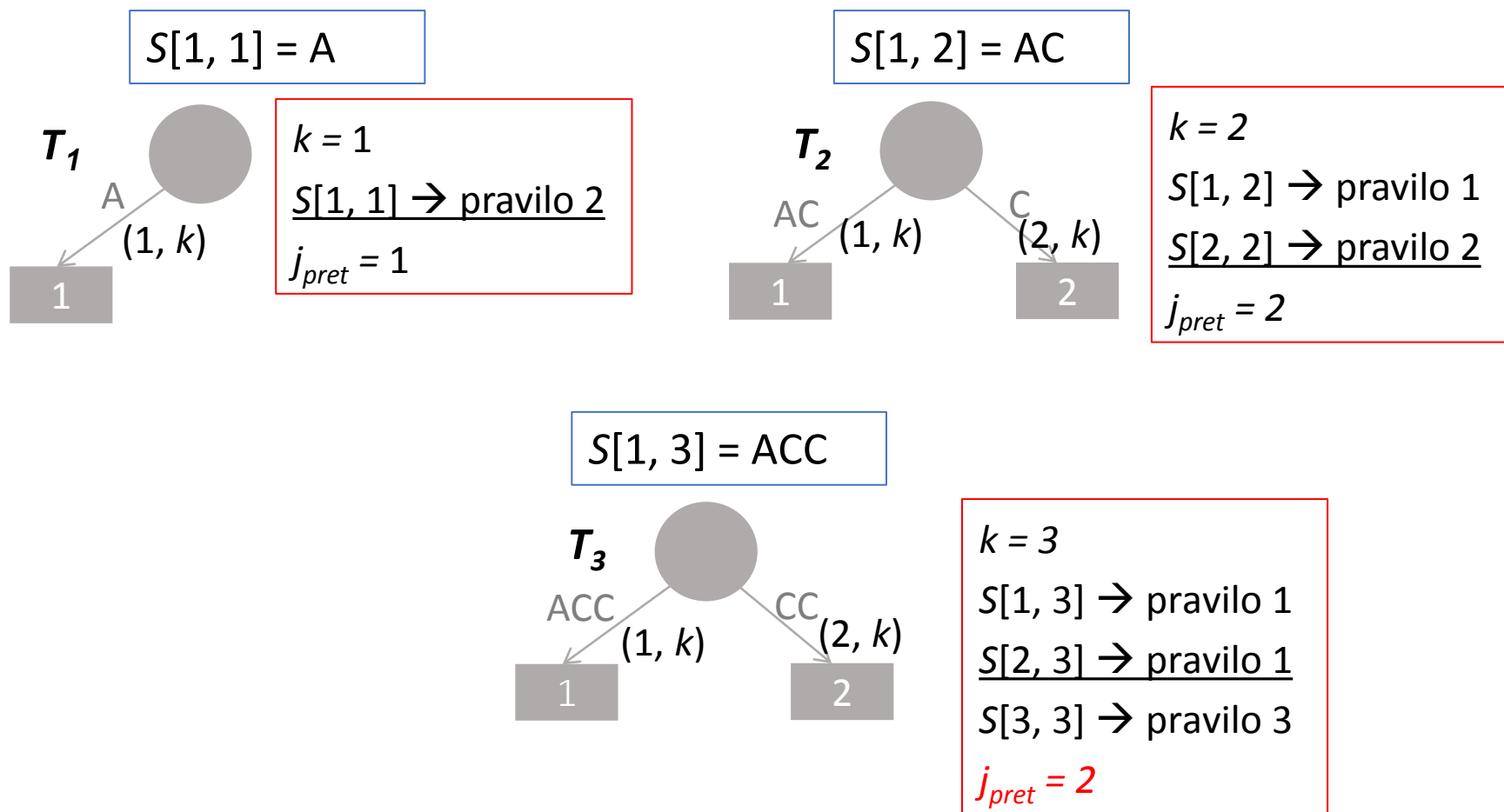
# Ukkonenov alg. u vremenu $O(n)$ - dokaz

- implicitna proširenja: u  $i$ -tom koraku:  $O(1) \rightarrow$  ukupno:  $O(n)$
- eksplicitna proširenja:
  - $j$  se nikad ne smanjuje i ostaje isti između dva susjedna koraka
  - $j$  ne može biti  $> n$
  - jedno eksplicitno proširenje:
    - 2 koraka (hodanje gore prema čvoru  $v$  + pomicanje po suf. vezi)
    - + broj koraka (= broj čvorova) na putu prema dolje od čvora  $s(v)$
  - ukupan broj hodanja po čvorovima za sve korake:  $O(n)$   
(s obzirom da  $j$  ostaje isti za dva susjedna koraka, tj. posljednji  $j$  iz  $i$ -tog koraka je prvi  $j$  u  $(i+1)$ -koraku, onda se trenutna dubina čvora ne mijenja)



# Ukkonenov algoritam $O(n)$ – primjer (1)

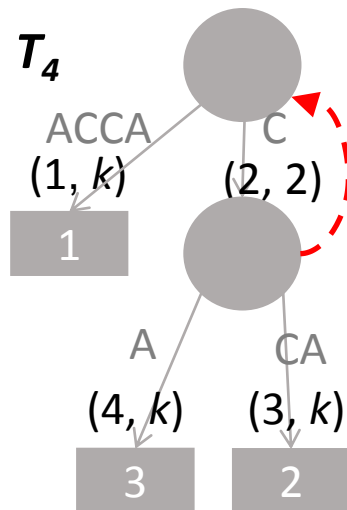
$S = \text{ACCA}\$$



# Ukkonenov algoritam $O(n)$ – primjer (2)

$S = \text{ACCA}\$$

$S[1, 4] = \text{ACCA}$



$k = 4$

$S[1, 4] \rightarrow \text{pravilo 1}$

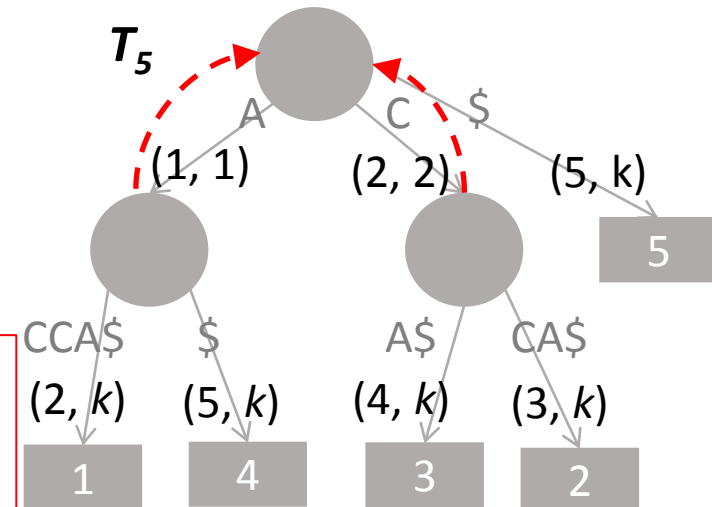
$S[2, 4] \rightarrow \text{pravilo 1}$

$S[3, 4] \rightarrow \text{pravilo 2}$

$S[4, 4] \rightarrow \text{pravilo 3}$

$j_{\text{pret}} = 3$

$S[1, 5] = \text{ACCA}\$$



$k = 5$

$S[1, 5] \rightarrow \text{pravilo 1}$

$S[2, 5] \rightarrow \text{pravilo 1}$

$S[3, 5] \rightarrow \text{pravilo 1}$

$S[4, 5] \rightarrow \text{pravilo 2}$

$S[5, 5] \rightarrow \text{pravilo 2}$

$j_{\text{pret}} = 5, \text{kraj}$

# Poopćeno sufiksno stablo

- poopćeno sufiksno stablo (eng. *generalized suffix tree*) je izgrađeno nad skupom nizova  $\{S_1, \dots, S_k\}$ 
  - stablo se izgrađuje nad konkatencijom nizova  $S_1\$_1 \dots S_k\$_k$ , gdje svaki niz  $S_i$  završava jedinstvenim znakom za kraj niza  $\$_i (\$_i \neq \$_j)$
  - sadrži sve sufikse nizova nad kojim je izgrađeno
  - izgradnja poopćenog sufiksnog stabla:  $O(|S_1| + \dots + |S_k|)$

# Sufiksna stabla - pretraživanje

- Postoji li niz  $P$ ,  $|P| = m$ , u  $S$ ?

$O(m)$

- Pronaći svih z pojavljivanja niza  $P$  u  $S$ :

$O(m + z)$

- Pronaći najdulji zajednički podniz nizova  $P_i$  i  $P_j$ :

$\Theta(|P_i| + |P_j|)$

- za svaki čvor u stablu označiti pripada li neki njegov list  $P_i$  i  $P_j$
- pronaći čvor s najvećom znakovnom dubinom, a koji pripada i  $P_i$  i  $P_j$

# Načini obilaska sufiksnog stabla

- od vrha prema dnu (eng. *top-down traversal*)
- od dna prema vrhu (eng. *bottom-up traversal*)
- korištenjem sufiksnih veza

# Implementacija sufiksni stabala - problem

- Problem:
  - memorijske najučinkovitije implementacije sufiksni stabala zahtijevaju oko 10 okteta za svaki ulazni znak, a često 15-20 okteta (Kurtz, 1999)
- Što napraviti?
  - korištenje drugih podatkovnih struktura kako bi se uskladili vremenski i memorijski zahtjevi: **sufiksna polja**
- sufiksna stabla ostaju kao konceptualni alat

# Popis literature

- Dan Gusfield, 1997. Algorithms on Strings, Trees and Sequences (Ch. 5-7)
- Ukkonen, E. 1995. On-line construction of suffix trees, *Algorithmica* 14(3): 249-260
- <http://en.wikipedia.org/wiki/Trie>
- <http://web.stanford.edu/~mjkay/gusfield.pdf>
- <http://www.cs.duke.edu/courses/fall14/compsci260/resources/suffix.trees.in.detail.pdf>
- Giegerich, R. ,Kurtz, S. 1997. From Ukkonen to McCreight and Weiner: A Unifying View of Linear-Time Suffix Tree Construction, *Algorithmica* 19(3): 331–353, doi:10.1007/PL00009177.
- Kurtz, S. 1999. Reducing the space requirements of suffix trees. *Software: Practice and Experience* 29, 1149–1171.
- Suffix Tree Construction\*, <http://stackoverflow.com/questions/9452701/ukkonens-suffix-tree-algorithm-in-plain-english>