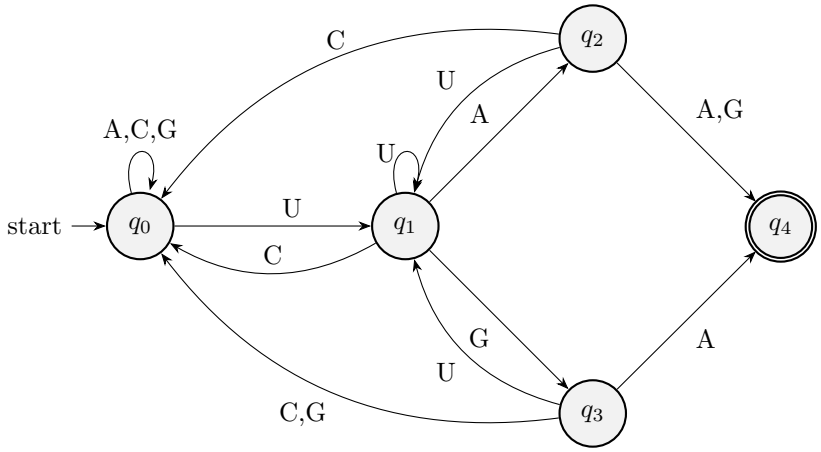
Algoritmi v bioinformatiki - 1. Domača naloga

Jan Panjan

March 26, 2025

1. *Konstruirajte deterministični končni avtomat, ki v mRNK materialu prepozna zaključne kodone.*

   (a) Grafično:



   (b) S formalnim opisom peterike $[\Sigma, Q, q_0, F, \delta]$:

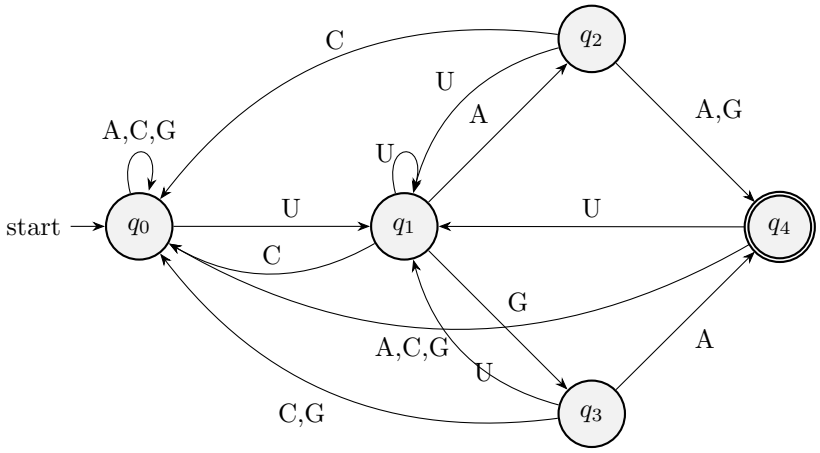
   - $\Sigma = \{A, U, C, G\}$
   - $Q = \{q_1, q_2, q_3, q_4\}$
   - $q_0 = q_0$
   - $F = \{q_4\}$
   - stanja so pisana samo s številko in pot ki pripelje do končnega stanja je označena z rdečo barvo, da je bolj berljivo.

δ	A	U	C	G
0	0	1	0	0
1	2	1	0	3
2	4	1	0	4
3	4	1	0	0
4	/	/	/	/

2. *Kako se rešitev 1. naloge spremeni, če želimo s pomočjo končnega avtomata poiskati vse pojavitve zaključnih kodonov? Zapišite algoritem in ponazorite njegovo delovanje na delu mRNK AUAUAAUGCUUGA. Koliko zaključnih kodonov vsebuje dani mRNK?*

   Njegovo končno stanje se spremeni, tako da ponovno začne iskati vzorec, ko pride enkrat do končnega stanja. To je vidno grafično kot povezava od q_4 do q_0 in spremenjene vrednosti v zadnji vrstici $\delta-$tabele.

   (a) Grafično:



   (b) S formalnim opisom peterike $[\Sigma, Q, q_0, F, \delta]$:

   - $\Sigma = \{A, U, C, G\}$
   - $Q = \{q_1, q_2, q_3, q_4\}$
   - $q_0 = q_0$
   - $F = \{q_4\}$
   -

δ	A	U	C	G
0	0	1	0	0
1	2	1	0	3
2	4	1	0	4
3	4	1	0	0
4	0	1	0	0

   Algoritem za iskanje STOP kodonov v mRNA vzorcu: algoritem za izgradnjo $\delta-$tabele smo podali na vajah in ga ne bom ponovno napisal. Predpostavljam torej, da je tabela za naš končni avtomat že izgrajena. Iskanje vzorca v besedilu AUAUAAUGCUUGA poteka tako:

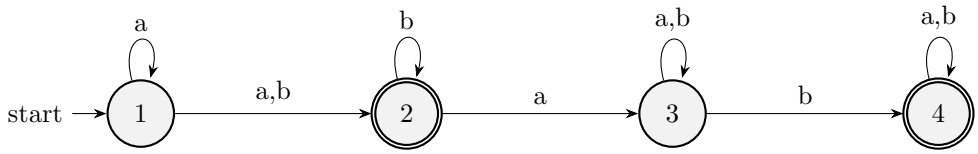
   kaj je mišljeno kot algoritem tu..? (poglej zapiske od vaj/predavanj, mislim da je podano)

Dani vzorec mRNA vsebuje dva stop kodona: AUAUAAUGCUUGA

3. *Konstruirajte deterministični končni avtomat za naslednji nedeterministični končni avtomat.*

   Deterministični končni avtomat iz nedeterminističnega izgradimo s pomočjo $\delta-$tabele tako da zapišemo vse neprazne podmnožice stanj. Za stanje npr. $1,2$ naredimo unijo sledečih stanj za a in b, t.j $1,2,3$ in 2., t.j $1,2,3$ in 2. Da se izognemo pisanju nepotrebnih stanj, naredimo tabelo samo za dosegljiva stanja.

   Nedeterministični končni avtomat:



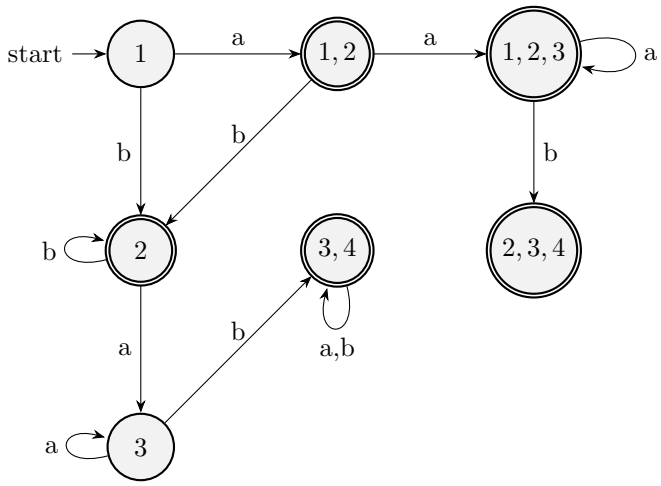
   $\delta-$tabela:

δ	a	b
1	1,2	2
2	3	2
3	3	3,4
4	4	4
1,2	1,2,3	2
3,4	3,4	3,4
1,2,3	1,2,3	2,3,4
2,3,4	3,4	2,3,4

   Končna stanja sta 2 in 4, zato bo imel novi končni avtomat za končna stanja vsa stanja, ki vsebujejo tako 2 ali 4.

   Zgodi se, da stanje 4 odpade, saj nobeno stanje ne vodi vanj.

   Deterministični končni avtomat:



4. *Poleg postopka z uporabo končnih avtomatov, poznamo tudi druge načine, kako odgovoriti na vprašanje "Kje in kolikokrat se vzorec p pojavi v besedilu t"? Naj bo naše besedilo* $t = ACCACCGACGCCCGA$.

   (a) *Za vzorec* $p = CCGA$*, ponazorite delovanje algoritma KMP tako, da poiščete vzorec p v besedilu t in opišite, kako nam pri tem pomaga funkcija* π*. Kolikokrat in kje se vzorec p pojavi v besedilu t?*

   KMP ali Knuth-Moriss-Prath-ov algoritem deluje na principu najdaljše predpone vzorca, ki je tudi pripona v dotedaj preiskanem besedilu. Z uporabo $\pi-$funkcije se algoritem izogne nepotrebnim primerjavam znakov v besedilu, za katere že vemo, da se ujemajo na začetku vzorca.

   S funkcijo π najdemo najdaljšo predpono vzorca, ki je tudi njegova najdaljša pripona. Za naš vzorec izgleda tako:

p	C	C	G	A
$\pi(p)$	0	1	0	0

   Psevdokoda za algoritem KMP izgleda tako:

```
def KMP(t, p):
 n = |t|
 m = |p|
 pi = Construct(p) // izgradi zgornji pi-seznam
 r = []             // seznam ki hrani začetne indekse
       //         kjer se pojavi ujemanje
 q = 0

 for i = 1 to n - 1:
  while q > 0 and p[q + 1] is not t[i]:
    q = pi
```

```
 if p[q + 1] == t[i]:
q += 1

 if q == m:
   r = r.add(i - q + 1)

return r
```

Potek algoritma je opisan z naslednjo tabelo:

q	i	$p[q+1]$	$t[i]$	match	action
0	1	C	A	No	incr i
0	2	C	C	Yes	incr q,i
1	3	C	C	Yes	incr q,i
2	4	G	A	No	$q = \pi[2] = 1$
1	4	C	A	No	$q = \pi[1] = 0$
0	4	C	A	No	incr i
0	5	C	C	Yes	incr q,i
1	6	C	C	Yes	incr q,i
2	7	G	G	Yes	incr q,i
3	8	A	A	Yes	incr q,i
4	9	ϵ	C	No	$r.add(i-q+1)$, $q = \pi[4]$
0	9	C	C	Yes	incr q,i
1	10	C	G	No	$q = \pi[1]$
0	10	C	G	No	incr i
0	11	C	C	Yes	incr q,i
1	12	C	C	Yes	incr q,i
2	13	G	C	No	$q = \pi[2]$
1	13	C	C	Yes	incr q,i
2	14	G	G	Yes	incr q,i
3	15	A	A	Yes	incr q,i
4	16	ϵ	ϵ	-	$r.add(i-q+1)$, $i = m : STOP$

Vzorec se pojavi dvakrat v našem besedilu. Algoritem najde ujemanja na indeksih 5 in 12.

(b) *Zgradite priponsko drevo za besedilo t. Opišite, kako s pomočjo priponskega drevesa učinkovito odgovorimo na vprašanje "Kje in kolikokrat se v besedilu t pojavi aminokislina prolin?"*

Priponsko drevo izgradimo tako, da najprej najdemo vse pripone našega besedila t, nato pa jih (po abecednem vrstnem redu) dodajamo v drevo z začetkom v korenu. Pripone, ki imajo enako pripono bodo sledile istemu poddrevesu, a ga bodo nato razcepile na več podvej. V listih drevesa hranimo začetni indeks pripone.

Vse pripone našega besedila:

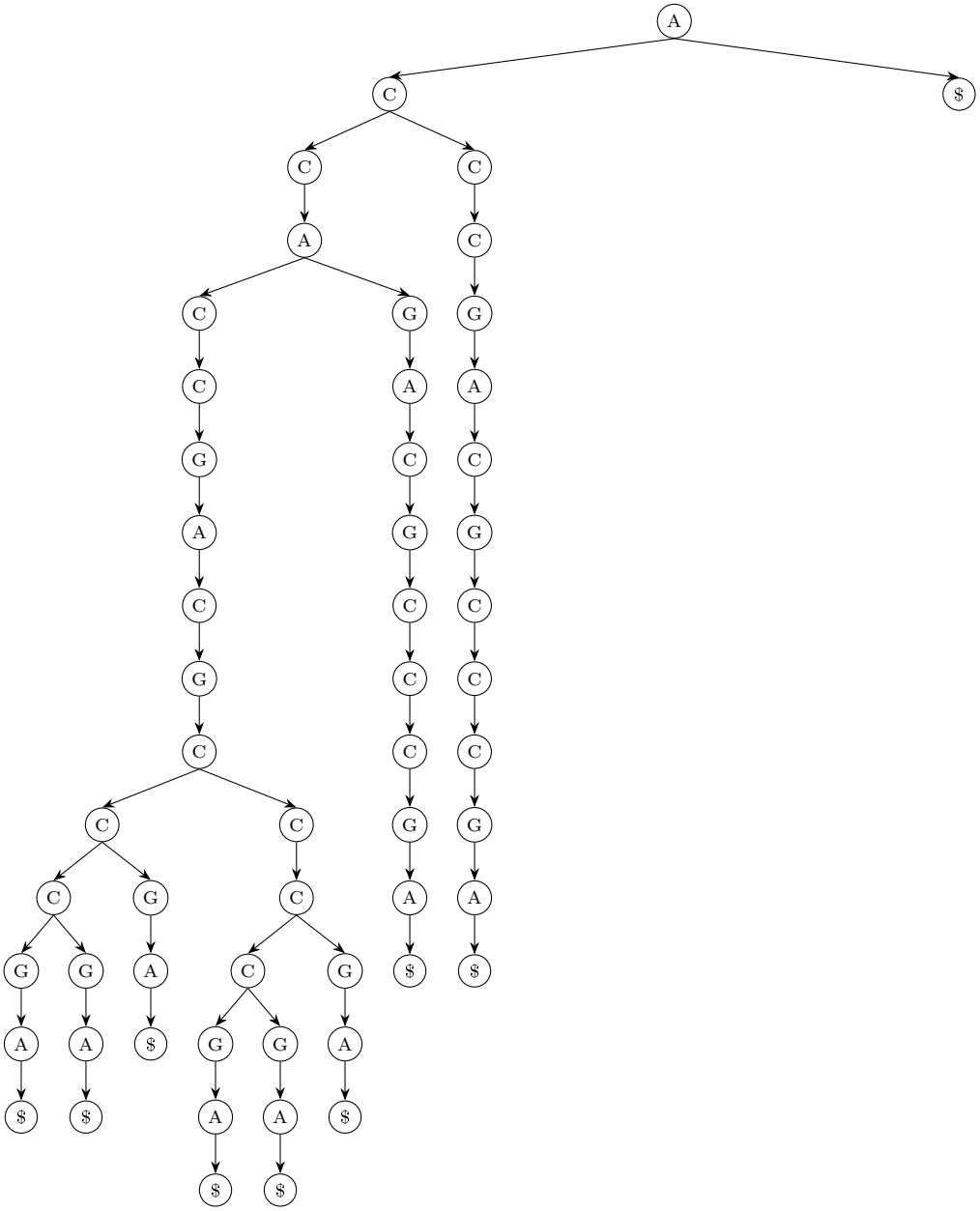
1. ACCACCGACGCCCGA$
2. CCACCGACGCCCGA$
3. CACCGACGCCCGA$
4. ACCGACGCCCGA$
5. CCGACGCCCGA$
6. CGACGCCCGA$
7. GACGCCCGA$
8. ACGCCCGA$
9. CGCCCGA$
10. GCCCGA$
11. CCCGA$
12. CCGA$
13. CGA$
14. GA$
15. A$
16. $

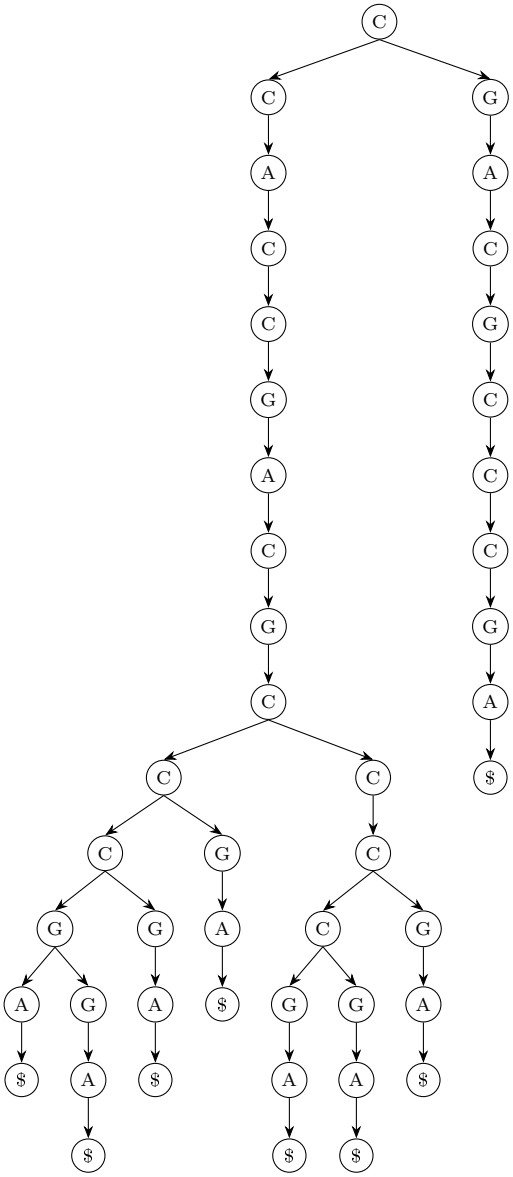
Indeksi urejeni po abedecnem vrstnem redu:

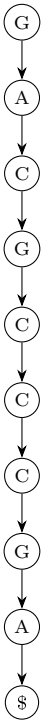
16, 15, 1, 4, 8, 3, 2, 11, 12, 5, 13, 6, 9, 14, 7, 10

Iz tega zdaj izgradimo priponsko drevo. Razdelil sem ga na 3 slike. Vsaka slika vsebuje poddrevesa, ki se začnejo s svojo črko (torej A, C in G).

(mislim, da je boljše da samo prilepiš slike na roke narisane)


A
C
$
C
C
A
C
C
G
G
A
A
C
C
G
G
C
C
G
G
C
C
C
C
C
C
C
C
C
C
C
C
C
G
C
G
G
G
A
C
G
A
A
$
G
G
A
A
$
$
$
$
A
A
$
$
$


C
C
G
A
A
C
C
C
G
G
C
A
C
C
C
G
G
C
A
C
C
$
C
G
C
G
A
C
G
A
G
G
A
G
G
A
$
$
A
$
G
G
A
$
A
A
$
$
$
$


G
A
C
G
C
C
C
G
A
$