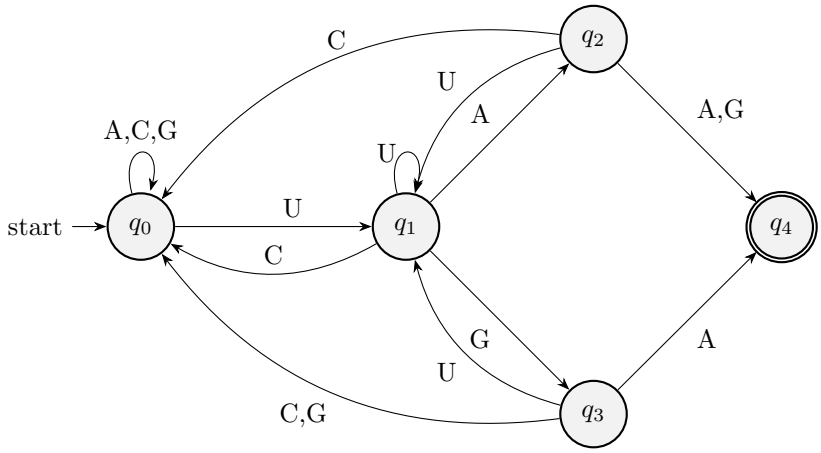
# Algoritmi v bioinformatiki - 1. Domača naloga

Jan Panjan

March 30, 2025

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

   (a) **Grafično:**




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

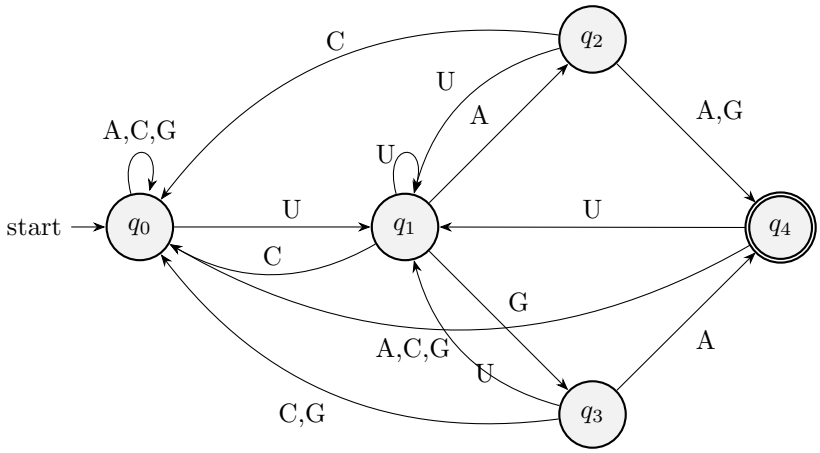
   - $\Sigma = \{A, U, C, G\}$
   - $Q = \{q_1, q_2, q_3, q_4\}$
   - $q_0 = q_0$
   - $F = \{q_4\}$
   - stanja so pisana samo s številko in pot ki pripelje do končnega stanja je označena z rdečo barvo, da je bolj berljivo.

| $\delta$ | 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** $[\Sigma, Q, q_0, F, \delta]$**:**

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

| $\delta$ | 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 deluje na osnovi $\delta-$tabele, tabela pa je lahko izgrajena neposredno iz grafičnega zapisa našege končenga avtomata oziroma preko formalnega zapisa peterke. Predpostavljam torej, da je tabela za naš končni avtomat že izgrajena. Iskanje vzorca v besedilu AUAUAAUGCUUGA poteka tako, da premaknemo končni avtomat v primerno stanje v tabeli glede na prebrani znak

$$\delta[q, c] = q' \tag{1}$$

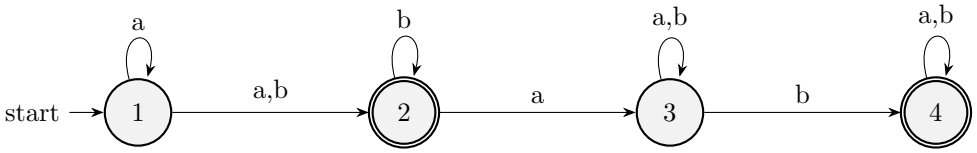
kjer je $q$ trenutno stanje, $c$ znak, ki ga preberemo iz besedila in $q'$ novo stanje.

Dani vzorec mRNA vsebuje **dva** stop kodona: AUAUAAUGCUUGA

3. *Konstruirajte deterministično končni avtomat za naslednji nedeterministično 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.

**Nedeterminističen končni avtomat:**



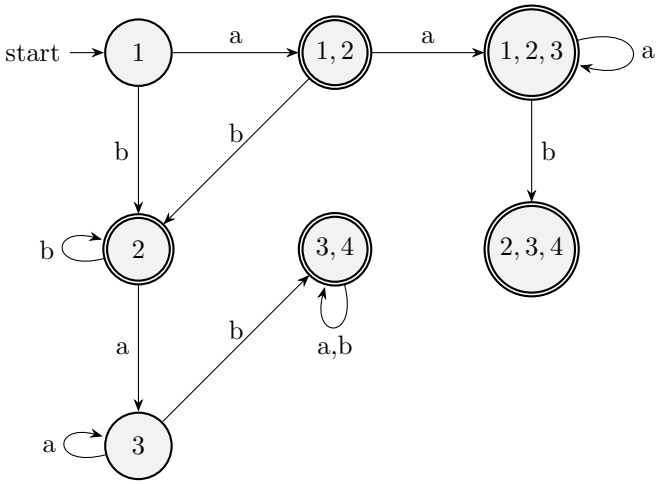

**$\delta-$tabela:**

| $\delta$ | 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.

**Deterministično 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* $\pi$. *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.

   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
```

   S funkcijo $\pi$ 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 |

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 | $\epsilon$ | 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 | $\epsilon$ | $\epsilon$ | - | $r.add(i-q+1)$, $i = m : STOP$ |

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.

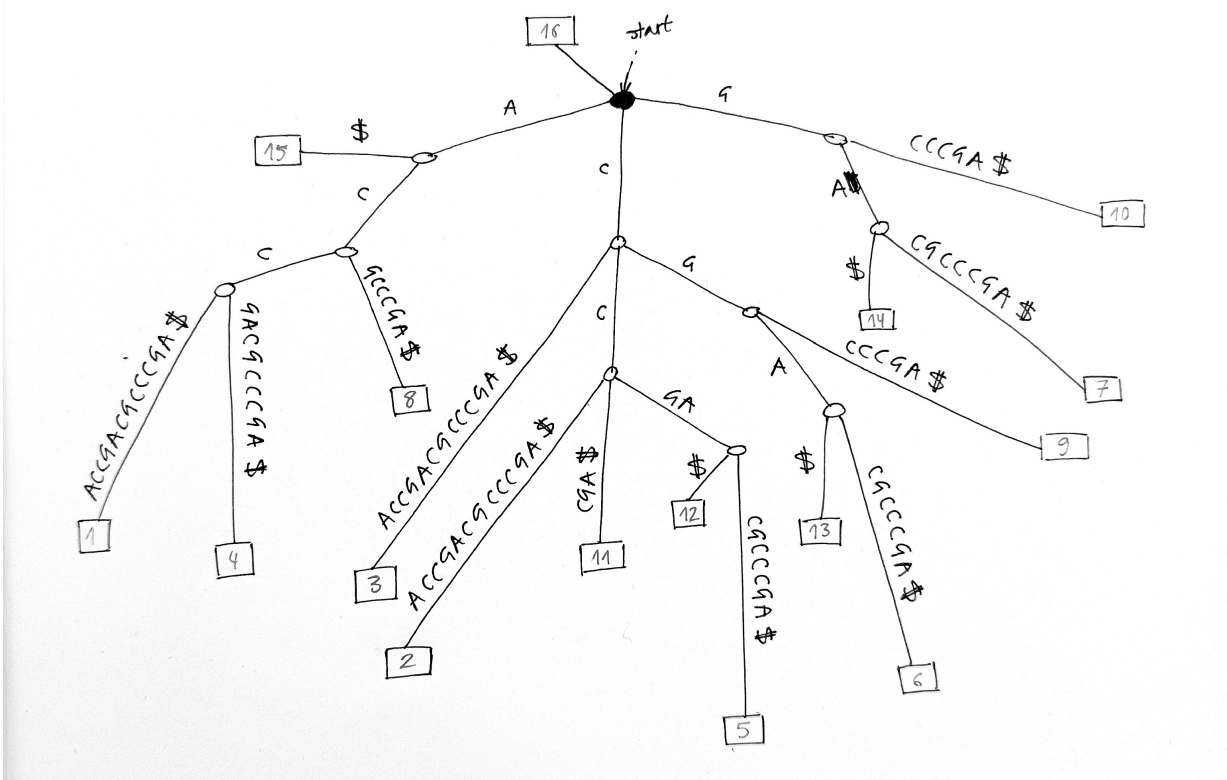
Vse pripone našega besedila:

| | |
|---|---|
| 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. Zgostil sem ga, da je bolj pregleden:



**Kje in kolikokrat se v besedilu pojavi aminokislina prolin:** prolin kodira več kodonov in sicer CCU, CCC, CCA ter CCG. Da najdemo vse njegove pojavitve v besedilu se spustimo po izgrajenem drevesu. Začnemo v korenu ter se pomikamo v pravo poddrevo glede na prebrani znak. Ko smo v notranjem vozlišču, kjer se vzorec konča, se spustimo do vseh listov v trenutnem poddrevesu, tako da najdemo vse začetne indekse kjer se pojavi vzorec v besedilu.

i. CCU se v besedilu **ne pojavi**

ii. CCC ima začetek na **11.** indeksu

iii. CCA na **2.** indeksu

iv. CCG na **5.** in **12.** indeksu

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

**Priponsko polje** izgradimo iz indeksov, ki smo jih dali v liste priponskega drevesa, tako da pripone uredimo po abecednem vrstnem redu.

| indeks v polju (m) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| indeks v besedilu (i) | 16 | 15 | 1 | 4 | 8 | 3 | 2 | 11 | 12 | 5 | 13 | 6 | 9 | 14 | 7 | 10 |

**Kje in kolikokrat se v besedilu pojavi aminokislina prolin:** ujemanja najdemo z bisekcijo. Vsako iteracijo izberemo srednji indeks v polju (oziroma levi, če je velikost polja soda) in primerjamo njegovo pripono z vzorcem. Če je vzorec manjši glede na abecedni vrstni red, se pomaknemo v levo polovico polja in odstranimo srednji indeks ter desno polovico (obratno, če je vzorec večji). Če pride do ujemanja z vzorcem, se pomaknemo v levo polovico polja in nadaljujemo z bisekcijo. Iskanje se konča, ko zmanjka indeksov v polju.

**CCU:**

| m | i | primerjava | ujemanje | akcija |
|---|---|---|---|---|
| 7 | 11 | $CCC\cdots < CCU$ | ne | desno |
| 11 | 6 | $CGA\cdots > CCU$ | ne | levo |
| 9 | 5 | $CCA\cdots < CCU$ | ne | desno |
| 10 | 13 | $CGA\cdots > CCU$ | ne | levo = end |

Ni ujemanj.

**CCC:**

| m | i | primerjava | ujemanje | akcija |
|---|---|---|---|---|
| 7 | 11 | $CCC\cdots = CCC$ | ja | levo |
| 3 | 4 | $ACC\cdots < CCC$ | ne | desno |
| 5 | 3 | $CAC\cdots < CCC$ | ne | desno |
| 6 | 2 | $CCA\cdots < CCC$ | ne | desno = end |

Ujemanje najdemo na 11. indeksu.

**CCA:**

| m | i | primerjava | ujemanje | akcija |
|---|---|---|---|---|
| 7 | 11 | $CCC\cdots > CCA$ | ne | levo |
| 3 | 4 | $ACC\cdots < CCA$ | ne | desno |
| 5 | 3 | $CAC\cdots < CCA$ | ne | desno |
| 6 | 2 | $CCA\cdots = CCA$ | ja | desno = end |

Ujemanje najdemo na 2. indeksu.

**CCG:**

| m | i | primerjava | ujemanje | akcija |
|---|---|---|---|---|
| 7 | 11 | $CCC\cdots < CCG$ | ne | desno |
| 11 | 6 | $CGA\cdots > CCG$ | ne | levo |
| 9 | 5 | $CCG\cdots = CCG$ | ja | levo |
| 8 | 12 | $CCG\cdots = CCG$ | ja | levo = end |

Ujemanji najdemo na 5. in 12. indeksu.

(d) *Besedilo t želimo zakodirati. Kateri način kodiranja nam bo dal najkrajši zapis:*

- *uporaba fiksne dolžine kod,*
- *uporaba Huffmanovega algoritma,*
- *uporaba Burrows-Wheeler transformacije, algoritma MTF in Huffmanovega algoritma (v tem vrstnem redu).*

*Odgovor ustrezno utemeljite.*

**Fiksna dolžina kod:** naša abeceda ima 4 znake (A, C, G, U) oziroma 5, če dodamo še \$, ki ga potrebujemo za označevanje konca besedila. Vsak znak zakodiramo z enakim številom bitov. V tem primeru bi bila 2-bita premalo ($2^2 = 4$), zato potrebujemo vsaj 3-bite ($2^3 = 8$).

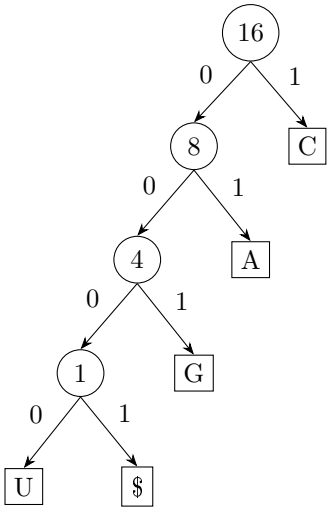
| znak | koda |
|---|---|
| A | 000 |
| C | 001 |
| G | 010 |
| U | 011 |
| \$ | 100 |

Vsak znak iz abecede je predstavljen z 8-biti (1-bajt). Dolžina našega besedila je 16 znakov, kar pomeni, da potrebujemo 16 bajtov oziroma 128 bitov za normalen zapis besedila. Če ga s pomočjo kodiranja z 3-biti zakodiramo, dobimo $16 \cdot 3 =$ **48-bitov**, kar je 80 bitov prihranka!

**Huffmanov algoritem:** Huffmanov algoritem deluje na principu frekvenc znakov. Je tako imenovano entropijsko kodiranje. S pomočjo frekvenc znakom dodelimo dolžine kod, ki so obratno sorazmerne z njihovimi frekvencami (višja frekvenca implicira krajšo kodo).

| znak | frekvenca ($t_i$) |
|---|---|
| A | 4 |
| C | 8 |
| G | 3 |
| U | 0 |
| \$ | 1 |

Iz frekvenc izgradimo drevo, tako da združujemo znake z najnižjimi frekvencami v eno vozlišče. Na koncu dobimo drevo, ki ga uporabimo za kodiranje znakov (v vozliščih je zapisana vsota frekvenc znakov).




Kode dobimo tako, da se iz korena sprehodimo do listov, kjer se nahaja pripadajoči znak. Na poti do lista dodamo 0, če gremo levo in 1, če gremo desno.

| znak | frekvenca ($t_i$) | koda |
|---|---|---|
| A | 4 | 01 |
| C | 8 | 1 |
| G | 3 | 001 |
| U | 0 | 0000 |
| \$ | 1 | 0001 |

Iz teh kod zdaj sestavimo zakodirano besedilo:

$$\text{HE(t)} = 01110111001011001111001010001$$

Dolžino besedila najhitreje izračunamo preko frekenc znakov in dolžin kod:

$$\begin{aligned} & t_A \cdot |01| + t_C \cdot |1| + t_G \cdot |001| + t_U \cdot |0000| + t_\$ \cdot |0001| \\ &= 4 \cdot 2 + 8 \cdot 1 + 3 \cdot 3 + 0 \cdot 4 + 1 \cdot 4 \\ &= 8 + 8 + 9 + 0 + 4 \\ &= 29 \end{aligned}$$

Zakodirano besedilo ima torej 29-bitov, kar je boljše kot 48-bitov pri fiksni dolžini kod.

**Burrows-Wheeler transformacija:** Burrows-Wheeler transformacija je algoritem, ki uporabimo za predprocesiranje besedila, da je nadaljne kodiranje bolj učinkovito. Deluje na principu t.i. *rotacij besedila*. Najprej izgradimo vse rotacije besedila, nato pa jih uredimo po abecednem vrstnem redu. Na koncu odčitamo zadnje znake vseh rotacij, da dobimo transformirano besedilo.

| i | rotacije | i | urejeno | zadnji znak |
|---|---|---|---|---|
| 1. | ACCACCGACGCCCGA\$ | 16. | \$... | A |
| 2. | CCACCGACGCCCGA\$A | 15. | A\$... | G |
| 3. | CACCACGCCCGA\$AC | 1. | ACCA... | \$ |
| 4. | ACCGACGCCCGAA\$ACC | 4. | ACCG... | C |
| 5. | CCGACGCCCGAA\$ACCA | 8. | ACG... | G |
| 6. | CGACGCCCGAA\$ACCAC | 3. | CA... | C |
| 7. | GACGCCCGA\$ACCACC | 2. | CCA... | A |
| 8. | ACGCCCGA\$ACCACCG | 11. | CCCG... | G |
| 9. | CGCCCGA\$ACCACCGA | 12. | CCGA\$... | C |
| 10. | GCCCGA\$ACCACCGAC | 5. | CCGAC... | A |
| 11. | CCCGA\$ACCACCGACG | 13. | CGA\$... | C |
| 12. | CCGA\$ACCACCGACGC | 6. | CGAC... | C |
| 13. | CGA\$ACCACCGACGCC | 9. | CGC... | A |
| 14. | GA\$ACCACCGACGCCC | 14. | GA\$... | C |
| 15. | A\$ACCACCGACGCCCG | 7. | GAC... | C |
| 16. | \$ACCACCGACGCCCGA | 10. | GC... | C |

$$BWT(t) = AG\$CGCAGCACCACCC$$

To zdaj zakodiramo s pomočjo **MTF (move-to-front) algoritma**. MTF algoritem deluje tako, da se sprehodimo čez transformirano besedilo, istočasno pa urejamo in dodajamo znake v pravem vrstnem redu v pomožno tabelo. Prebran znak premaknemo na vrh tabele in odčitamo indeks iz katerega smo ga premaknili. Ostale znake premaknemo eno mesto navzdol (prva vrstica v tabeli predstavlja naše transformirano besedilo).

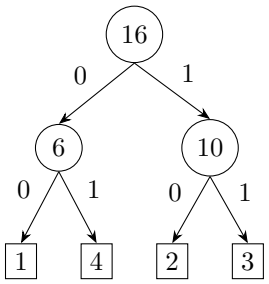
| 1 | \$ | A | G | \$ | C | G | C | A | G | C | A | C | C | A | C | C | C |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | A | \$ | A | G | \$ | C | G | C | A | G | C | A | A | C | A | A | A |
| 3 | C | C | \$ | A | G | \$ | \$ | G | C | A | G | G | G | G | G | G | G |
| 4 | G | G | C | C | A | A | A | \$ | \$ | \$ | \$ | \$ | \$ | \$ | \$ | \$ | \$ |
| 5 | U | U | U | U | U | U | U | U | U | U | U | U | U | U | U | U | U |
| | | 2 | 4 | 3 | 4 | 3 | 2 | 4 | 3 | 3 | 3 | 2 | 1 | 2 | 2 | 1 | 1 |

Zadnja vrstica je rezultat MTF algoritma: $MTF(BWT(t)) = 2434324333212211$.

To zaporedje števil nato zakodiramo s pomočjo Huffmanovega algoritma, tako kot prej:

| znak | frekvenca ($t_i$) |
|---|---|
| 1 | 3 |
| 2 | 5 |
| 3 | 5 |
| 4 | 4 |

Iz frekvenc ustvarimo drevo:




Dobimo pripadajoče kode:

| znak | frekvenca ($t_i$) | koda |
|---|---|---|
| 1 | 3 | 00 |
| 2 | 5 | 10 |
| 3 | 5 | 11 |
| 4 | 4 | 01 |

Zakodirano besedilo je torej:

$$\text{HE(MTF(BWT(t)))} = 10011101111001111111100010100000$$

Dolžino zakodiranega besedila ponovno izračunamo iz frekvenc in dolžin kod:

$$\begin{aligned} & t_1 \cdot |00| + t_2 \cdot |10| + t_3 \cdot |11| + t_4 \cdot |01| \\ &= (3 + 5 + 5 + 3) \cdot 2 \\ &= 16 \cdot 2 \\ &= 32 \end{aligned}$$

Besedilo transformirano z Burrows-Wheeler transformacijo in zakodirano z MTF ter Huffmanovim algoritmom je dolgo **32-bitov**, kar je slabše od 29-bitov pri samem Huffmanovem algoritmu.

> Mislim, da je razlog za slabši rezultat Burrows-Wheeler transformacije ta, da je naše besedilo prekratko. Burrows-Wheeler transformacija je bolj učinkovita pri daljših besedilih, ki vsebujejo veliko ponavljajočih se znakov, saj jih lahko tako uredi skupaj (na primer AA$GGGGCCCCC...). Ker BWT ne privede do boljše ureditve, MTF kodiranje ni efektivno (ta v svoj prid izkorišča dolga ponavljajoča-se zaporedja, podobno kot RLE - run-length encoding.).

**Zaključek:** Najboljše kodiranje dobimo s pomočjo Huffmanovega algoritma, ki nam da zakodirano besedilo dolžine 29-bitov.