Bioinformatyka 2

Bioinformatyka RNA

zadania

2022/2023

Maks. 20pkt -> 5pkt

|  |  |
| --- | --- |
| **Format Vienna** | |
| > Linia nagłówka  AUCGAGAAAUCGAAC  .((((....)))).. |  |
| >  AUCGAGCCAUGGCUGAAAUCGAAC  .(((((.(...)))(...))).). |  |
| Zadanie 1 (1pkt) Jaka będzie struktura dla:  1234567890123456789012346  .((..(((...)))..((..)))). |  |
| Zadanie 2 (1pkt) Jaka będzie struktura dla:  UUGGUUCGCCAAAAGGC  ((.....[[[))..]]] |  |

|  |  |
| --- | --- |
| **Obliczanie liczby struktur** | |
| **Przykład** Liczba struktur RNA - skojarzenie doskonałe  Dla danego łańcucha RNA *S* który zawiera tą samą liczbę A i U oraz C i G podać możliwą liczbę kombinacji wiązania par.  – liczba par adenin/uracyli  – liczba par guanin/cytozyn  Liczba wszystkich struktur (skojarzenia doskonałe)   |  |  |  | | --- | --- | --- | |  |  |  |   **Rozwiązanie**  np. dla *A1G2C3U4A5G6U7C8A9U10* | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  |  |  |  | | --- | --- | --- | --- | | permutacja | A1 | = | U4 | | A5 | = | U7 | | A9 | = | U10 | |  | |  |  |  | | --- | --- | --- | | G2 | = | C3 | | G6 | = | C8 | | |
| Zadanie 3 (2pkt) Liczba struktur RNA - skojarzenie największe  Dla danego łańcucha RNA *S* który zawiera parujące się zasady w różnych proporcjach podać możliwą liczbę kombinacji wiązania par.  np. dla   |  |  |  | | --- | --- | --- | | CCGG  2 | CCGGG  6 | CCGGAAUUU  12 |   CCCAAAGGG | Czy rzeczywista liczba struktur będzie większa czy mniejsza od przewidzianej w zad. 3. Odpowiedź uzasadnij  Ilość wszystkich możliwych struktur CCCAAAGGG wynosi 6.  Nie będzie tyle struktur ze względu na zawadę steryczną. Gdy powstaje spinka to pętla może mieć najmniej 3-4 nukleotydy. Pętle uniemożliwiają niektóre wiązania, ponieważ zdarza się że nukleotydy które mogłyby się ze sobą wiązać są od siebie zbyt oddalone w przestrzeni. |

|  |  |
| --- | --- |
| **Algorytm Nussinova**  algorytm wyszukujący strukturę o maksymalnej liczbie sparowanych zasad | |
| Zadanie 4 (3pkt)  wykonać predykcje dla sekwencji:  AUUAACCGCGGUUAAUCGCG  używając implementacji w Python:   1. [**Nussinov**](http://rna.informatik.uni-freiburg.de/Teaching/index.jsp?toolName=Nussinov)   oraz udoskonalonym programem   1. [**RNAfold**](http://rna.tbi.univie.ac.at/cgi-bin/RNAWebSuite/RNAfold.cgi) (ustawienia domyślne)   W obu przypadkach: min. loop length = 4  Wstawić uzyskane struktury w formacie Vienna. Porównać te struktury i uzasadnić dlaczego się od siebie różnią. | Algorytm Nussinov:  <https://github.com/cgoliver/Nussinov/blob/master/nussinov.py>   1. Algorytm Nussinov   ((((((....))))))....     1. RNAfold   ......((((......))))    Struktury się od siebie różnią, ponieważ algorytm Nussinov maksymalizuje liczbę dopasowań między komplementarnymi zasadami, RNAfold natomiast wyznacza strukturę poprzez wybór tych o minimalnej energii swobodnej (MFE) |
| Zadanie 5 (3pkt) Zmodyfikuj implementacje algorytmu Nussinov (Python) tak, aby:   * uwzględniał pary wooble G-U; * zamiast maksymalizacji liczby par realizowana była maksymalizacja punktacji przypisanej różnym parom. Np. score(‘G’,’C’)=3, score(‘A’,’U’)=2, score(‘G’,’U’) = 1.   Modyfikacje do pewnego momentu można testować na portalu <http://rna.informatik.uni-freiburg.de/Teaching/index.jsp?toolName=Nussinov> | |

|  |  |  |
| --- | --- | --- |
| Metoda szacowania energii swobodnej (*Free Energy Methods*) | | |
| * pary G-C są bardziej stabilne niż A-U, a te bardziej stabilne niż G-U; * oddziaływanie warstwowe – *stacking interactions* - Pary zasad znajdujące się obok siebie stabilizują się wzajemnie przez oddziaływanie przyciągające między ich pierścieniami aromatycznymi; * pętle i wybrzuszenia destabilizują odpowiednie pary zasad.   ODPOWIEDŹ NA KOŃCU PLIKU!! | | A unified view of polymer, dumbbell, and oligonucleotide DNA  nearest-neighbor thermodynamics | PNAS |
| Zadanie 6 (4pkt) Obliczyć energię znanych nam już struktur:    External loop : 0.40  Interior loop ( 1, 16) AU; ( 2, 15) UA: -1.10  Interior loop ( 2, 15) UA; ( 3, 14) UA: -.90  Interior loop ( 3, 14) UA; ( 4, 13) AU: -1.30  Interior loop ( 4, 13) AU; ( 5, 12) AU: -.90  Interior loop ( 5, 12) AU; ( 6, 11) CG: -2.20  Hairpin loop ( 6, 11) CG : 3.60  AUUAACCGCGGUUAAUCGCG  ((((((....)))))).... ( -2.40)  **-2.40 kcal/mol** | External loop : -.30  Interior loop ( 7, 20) CG; ( 8, 19) GC: -2.40  Interior loop ( 8, 19) GC; ( 9, 18) CG: -3.40  Interior loop ( 9, 18) CG; ( 10, 17) GC: -2.40  Hairpin loop ( 10, 17) GC : 3.90  AUUAACCGCGGUUAAUCGCG  ......((((......)))) (-4.6)    **-4.60 kcal/mol–** bardziej stabilna | |

|  |
| --- |
| Zadanie 7 (3pkt) Wykorzystując doświadczenie zdobyte w zadaniu 6 zaproponuj sekwencje dla podanej struktury w formacie Vienna, która uzyskuje możliwie najniższą energię (weryfikacja poprzez RNAfold):  .(((((....))))).(((...))).  CGCGGCGCCUGCCGUCGCUUUGAGCC -> Wynik z RNAfold, -8.30  --------------------------------------------------------    CGCGGCGCCUGCCGUCGCUUUGAGCC  .((((((.....))))))........ -8.60 |

|  |  |  |
| --- | --- | --- |
| Predykcja struktury w oparciu o prawdopodobieństwa parowania zasad | | |
| Prawdopodobieństwo uzyskania struktury :  , gdzie  suma statystyczna (*partition function*)  Obliczanie sumy częściowej () jest proporcjonalne do liczby struktur a te są funkcja eksponencjalną długości sekwencji  – stała Boltzmana  – temperatura w Kelwinach np. |  | |
| Zadanie 8 (3pkt) Dla podanej sekwencji: >tRNA-Cys-GCA-9-1-fragment GGGGGUAUAGCUCAGGGGUAGAGCAUUUGACUGCAGAUCAAGAGGUCCCUGGUUCAAAUCCAGGUGCCCCCU  uzyskano podzbiór struktur: |  | |
| struktura free energy in kcal/mol  (((((((..((((.......))))...........((((....))))(((((.......)))))))))))). -28.10  (((((((..((((.......))))....((((.(.......).))))(((((.......)))))))))))). -27.90  ((((((((.((((.......))))(((((((((..((((....))))..)))).)))))....)))))))). -27.80  ((((((((.((((.......))))(((((((((..((((....))))..))).))))))....)))))))). -27.80  (((((((..((((.......))))....((((...........))))(((((.......)))))))))))). -27.60  (((((((..((((.......))))....(((..(.......)..)))(((((.......)))))))))))). -27.50  ((((((((.((((.......)))).((((((((..((((....))))..)))).)))).....)))))))). -27.20  ((((((((.((((.......)))).((((((((..((((....))))..))).))))).....)))))))). -27.20  ((((((((.((((.......))))...........((((....)))).((((.......)))))))))))). -27.20  ((((((...((((.......))))...........((((....))))(((((.......))))).)))))). -27.20  (((((((...(((...(((...(((......)))..)))..)))...(((((.......)))))))))))). -27.10  ((((((((.((((.......))))((((((((...((((....))))...))).)))))....)))))))). -27.00  ((((((((.((((.......))))((((((((...((((....))))...)).))))))....)))))))). -27.00  ((((((((.((((.......))))....((((.(.......).)))).((((.......)))))))))))). -27.00  (((((((..((((.......)))).((((((....).))))).....(((((.......)))))))))))). -27.00  (((((((..((((.......))))...........(((......)))(((((.......)))))))))))). -27.00  ((((((...((((.......))))....((((.(.......).))))(((((.......))))).)))))). -27.00  ((((((((.((((.......))))(((((((((..(((......)))..)))).)))))....)))))))). -26.70  ((((((((.((((.......))))(((((((((..(((......)))..))).))))))....)))))))). -26.70  ((((((((.((((.......))))....((((...........)))).((((.......)))))))))))). -26.70  (((((((..((((.......)))).(((((.......))))).....(((((.......)))))))))))). -26.70  ((((((...((((.......))))....((((...........))))(((((.......))))).)))))). -26.70  Policzyć prawdopodobieństwo każdej struktury. Stworzyć dwa wykresy zależności *P(E)*. Jeden dla unikalnych wartości energii. Drugi sumując wartości prawdopodobieństwa dla struktur o takiej samej energii. Kod do wygenerowania wykresu pierwszego poniżej.  **import** numpy **as** np  **import** matplotlib**.**pyplot **as** plt  k **=** 0.001985875 #kcal/mol\*K  T **=** 273**+**37 #Kelvin  en **=** **[-**28.10**,-**27.90**,-**27.80**,-**27.80**,-**27.60**,-**27.50**,-**27.20**,-**27.20**,-**27.20**,-**27.20**,-**27.10**,-**27.00**,-**27.00**,-**27.00**,-**27.00**,-**27.00**,-**27.00**,-**26.70**,-**26.70**,-**26.70**,-**26.70**,-**26.70**]**  x **=** sorted**(**list**(**set**(**en**)))**  Z **=** 0  y **=** **[]**  y2 **=** **[]**  **print(**len**(**list**(**x**)))**  d **=** dict**(**zip**(**x**,[**en**.**count**(**el**)** **for** el **in** x**]))**  **print(**d**)**  **for** i **in** en**:**  y**.**append**(**np**.**exp**(-**i**/(**k**\***T**)))**  **for** i **in** x**:**  y2**.**append**(**d**[**i**]\***np**.**exp**(-**i**/(**k**\***T**)))**  Z **=** sum**(**y**)**  y **=** np**.**array**(**y**)/**Z  y2 **=** np**.**array**(**y2**)/**Z  plt**.**plot**(**en**,**y**)**  plt**.**show**()** |  | |
| P każdej struktury: [0.15586128, 0.11262811, 0.19148336, 0.06918463, 0.05881172, 0.014450628, 0.03071008, 0.15663414, 0.08018035]  Kolejno dla: [-28.10,-27.90,-27.80,-27.60,-27.50,-27.20,-27.10,-27.00,-26.70] | |  |

**import** sys

**import** numpy **as** np

**import** pandas **as** pd

min\_loop\_length **=** 4 *#0*

**def** score(pair):

scoring **=** {("A", "U"): 2,("U", "A"): 2,("G", "C"): 3,("C", "G"): 3,("U", "G"): 1,("G", "U"): 1}

**if** pair **in** scoring:

**return** scoring[pair]

**def** pair\_check(tup):

**if** tup **in** [('A', 'U'), ('U', 'A'), ('C', 'G'), ('G', 'C'), ('G', 'U'), ('U', 'G')]:

**return** **True**

**return** **False**

**def** OPT(i, j, sequence):

"""

returns the score of the optimal pairing between indices i and j

"""

*#base case: no pairs allowed when i and j are less than 4 bases apart*

**if** i **>=** j**-**min\_loop\_length:

**return** 0

**else**:

*#i and j can either be paired or not be paired, if not paired then the optimal score is OPT(i,j-1)*

unpaired **=** OPT(i, j**-**1, sequence)

*#check if j can be involved in a pairing with a position t*

*# if i == 2 and j == 9:*

*# print ([(i,t-1, t+1, j-1) for t in range(i, j-min\_loop\_length)])*

pairing **=** [score((sequence[t],sequence[j])) **+** OPT(i, t**-**1, sequence) **+** OPT(t**+**1, j**-**1, sequence) **for** t **in** range(i, j**-**min\_loop\_length)\

**if** pair\_check((sequence[t], sequence[j]))]

**if** **not** pairing:

pairing **=** [0]

paired **=** max(pairing)

**return** max(unpaired, paired)

**def** traceback(i, j, structure, DP, sequence):

*#in this case we've gone through the whole sequence. Nothing to do.*

**if** j **<=** i:

**return**

*#if j is unpaired, there will be no change in score when we take it out, so we just recurse to the next index*

**elif** DP[i][j] **==** DP[i][j**-**1]:

traceback(i, j**-**1, structure, DP, sequence)

*#consider cases where j forms a pair.*

**else**:

*#try pairing j with a matching index k to its left.*

**for** k **in** [b **for** b **in** range(i, j**-**min\_loop\_length) **if** pair\_check((sequence[b], sequence[j]))]:

*#if the score at i,j is the result of adding 1 from pairing (j,k) and whatever score*

*#comes from the substructure to its left (i, k-1) and to its right (k+1, j-1)*

**if** k**-**1 **<** 0:

**if** DP[i][j] **==** DP[k**+**1][j**-**1] **+** 1:

structure**.**append((k,j))

traceback(k**+**1, j**-**1, structure, DP, sequence)

**break**

**elif** DP[i][j] **==** DP[i][k**-**1] **+** DP[k**+**1][j**-**1] **+** 1:

*#add the pair (j,k) to our list of pairs*

structure**.**append((k,j))

*#move the recursion to the two substructures formed by this pairing*

traceback(i, k**-**1, structure, DP, sequence)

traceback(k**+**1, j**-**1, structure, DP, sequence)

**break**

**def** write\_structure(sequence, structure):

dot\_bracket **=** ["." **for** \_ **in** range(len(sequence))]

**for** s **in** structure:

dot\_bracket[min(s)] **=** "("

dot\_bracket[max(s)] **=** ")"

**return** ""**.**join(dot\_bracket)

*#initialize matrix with zeros where can't have pairings*

**def** initialize(N):

*#NxN matrix that stores the scores of the optimal pairings.*

DP **=** np**.**empty((N,N))

DP[:] **=** np**.**NAN

**for** k **in** range(0, min\_loop\_length):

**for** i **in** range(N**-**k):

j **=** i **+** k

DP[i][j] **=** 0

**return** DP

**def** nussinov(sequence):

N **=** len(sequence)

DP **=** initialize(N)

structure **=** []

*#fill the DP matrix diagonally*

**for** k **in** range(min\_loop\_length, N):

**for** i **in** range(N**-**k):

j **=** i **+** k

DP[i][j] **=** OPT(i,j, sequence)

*#copy values to lower triangle to avoid null references*

**for** i **in** range(N):

**for** j **in** range(0, i):

DP[i][j] **=** DP[j][i]

traceback(0,N**-**1, structure, DP, sequence)

**return** (write\_structure(sequence, structure), structure)

**if** \_\_name\_\_ **==** "\_\_main\_\_":

print(nussinov("GGGAAGAAGAACCC"))