Bioinformatyka 2 - kurs mały

Wprowadzenie i sprawy organizacyjne. Podstawy bioinformatyki sekwencji.

Krzysztof Murzyn

Zakład Biofizyki Obliczeniowej i Bioinformatyki
Wydział Biochemii, Biofizyki i Biotechnologii
Uniwersytet Jagielloński

Plan wykładu

1. **Podstawowe informacje**
   - Terminy zajęć
   - Zawartość kursu
   - Cele kursu
   - Praktyczne wskazówki
2. **Podstawy bioinformatyki sekwencji**
   - Porównywanie sekwencji
   - Algorytmy heurystyczne
3. **Wprowadzenie do programowania w Pythonie**
   - Typy danych
   - Konstrukcje syntaktyczne

Kontakt

dr hab. Krzysztof Murzyn
Zakład Biofizyki Obliczeniowej i Bioinformatyki
WBBiB UJ, pok. B028 (wtorek 11:00-11:45), Teams rozmowa / czat
tel. (12) 664-63-79
email: krzysztof.murzyn@uj.edu.pl
http://bioinfo.mol.uj.edu.pl/modmol/People/KrzysztofMurzyn

Zajęcia i warunki zaliczenia

- 5 wykładów (łącznie 10h), czwartek 10:30-12, sala 1.01.13, stacjonarny test pojedynczego wyboru $(+2/-1/0)$ z pytaniami otwartymi i zamkniętymi
- 5 ćwiczeń (po 3h, 135 min), stacjonarny test praktyczny (zestaw zadań do samodzielnego rozwiązania) - łącznie 20h, wszystkie ćwiczenia stacjonarnie
- ćwiczenia rozpoczynają się w tygodniu po pierwszym wykładzie, koordynatorem ćwiczeń (harmonogram, oceny, etc) jest mgr Adrian Kania

- zaliczenie kursu obejmuje:
  - zaliczenie ćwiczeń
    wykonanie ćwiczeń ($5 \cdot 6$ pkt), wynik testu praktycznego (120 min, ok. 5 zadań, 70 pkt); łącznie: 100 pkt (ZAL (50+)/NZAL)
  - zaliczenie wykładu – ocena do średniej, warunek wstępny: zaliczenie ćwiczeń, wynik punktowy zaliczenia ćwiczeń (maks. 100 pkt) powiększony o wynik testu pojedynczego wyboru zawierającego pytania z zagadnień poruszanych na wykładach i ćwiczeniach (teoria, maks. 100 pkt); łącznie: 200 pkt

Procentowa skala ocen	
bdb	≥ 90 %
+db	$[\,80, 90\,)$ %
db	$[\,70, 80\,)$ %
+dst	$[\,60, 70\,)$ %
dst	$[\,50, 60\,)$ %
ndst	< 50 %

Wykłady

1. Wprowadzenie i sprawy organizacyjne. Podstawy bioinformatyki sekwencji. Wprowadzenie do programowania w Pythonie. Pozyskiwanie i przetwarzanie danych biologicznych. [Murzyn/Kania]
2. Metody przewidywania i walidacji struktury przestrzennej białek. Metaserwery predykcyjne. [Murzyn]
3. Zaawansowane protokoły obliczeniowe w modelowaniu molekularnym biocząsteczek. [Murzyn]
4. Techniki nauczania maszynowego w zastosowaniu do analizy danych z mikromacierzy. [Kania]
5. Techniki przetwarzania i analizy danych z sekwencjonowania nowej generacji. Potoki analityczne na serwerze Galaxy. [Majta]

Prezentacje z wykładów (PDF) będą na bieżąco udostępniane w materiałach kursowych w dedykowanym zespole Teams.

Ćwiczenia

1. Pozyskiwanie i przetwarzanie danych biologicznych z wykorzystaniem technik programowania w Pythonie. [Kania]
2. Metody przewidywania i walidacji struktury przestrzennej białek. Metaserwer predykcyjne. [Gucwa]
3. Zaawansowane protokoły obliczeniowe w modelowaniu molekularnym biocząsteczek. [Gucwa]
4. Techniki nauczania maszynowego w zastosowaniu do analizy danych z mikromacierzy. [Gucwa]
5. Techniki przetwarzania i analizy danych z sekwencjonowania nowej generacji. Serwer Galaxy. [Majta]

Materiały do ćwiczeń udostępniane przez ćwiczeniowca.

Wiedza i umiejętności

Wiedza

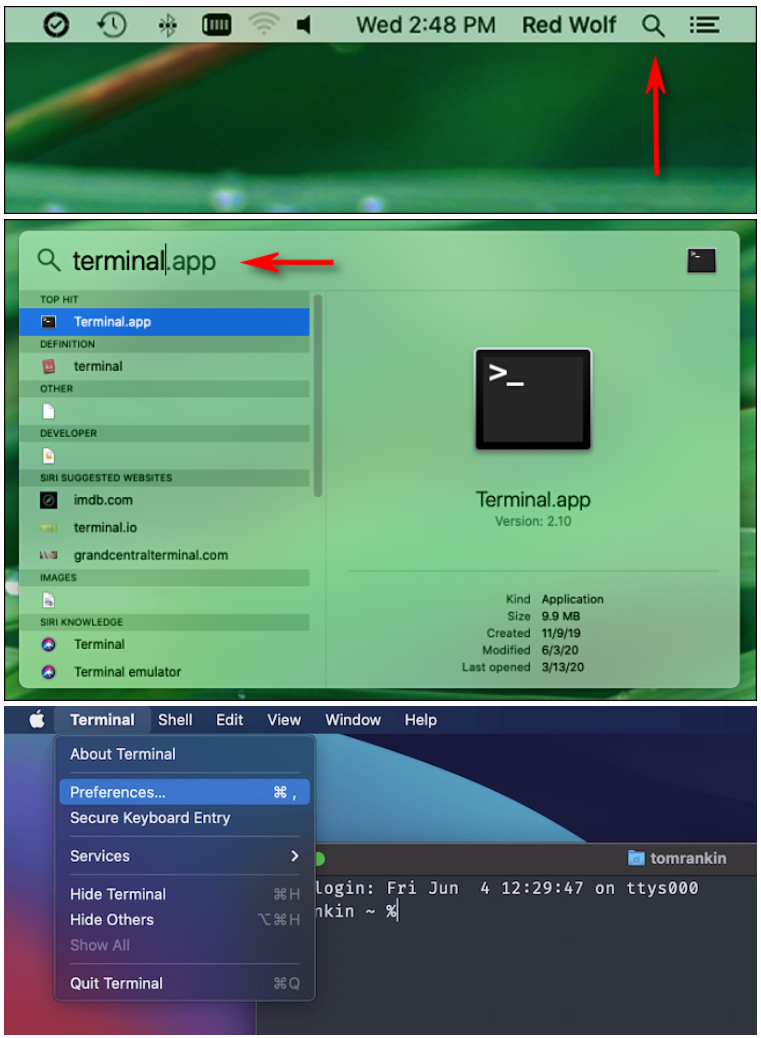
- wybrane zagadnienia bioinformatyki sekwencji oraz bioinformatyki strukturalnej
- pozyskiwanie i przetwarzanie danych biologicznych
- przewidywanie i walidacja struktury przestrzennej białek (modelowanie porównawcze, metaserwer predykcyjny iTasser)
- wybrane protokoły obliczeniowe w modelowaniu molekularnym
- techniki nauczania maszynowego w zastosowaniach bioinformatycznych
- techniki przetwarzania i analizy danych z sekwencjonowania nowej generacji (serwer Galaxy)

Umiejętności

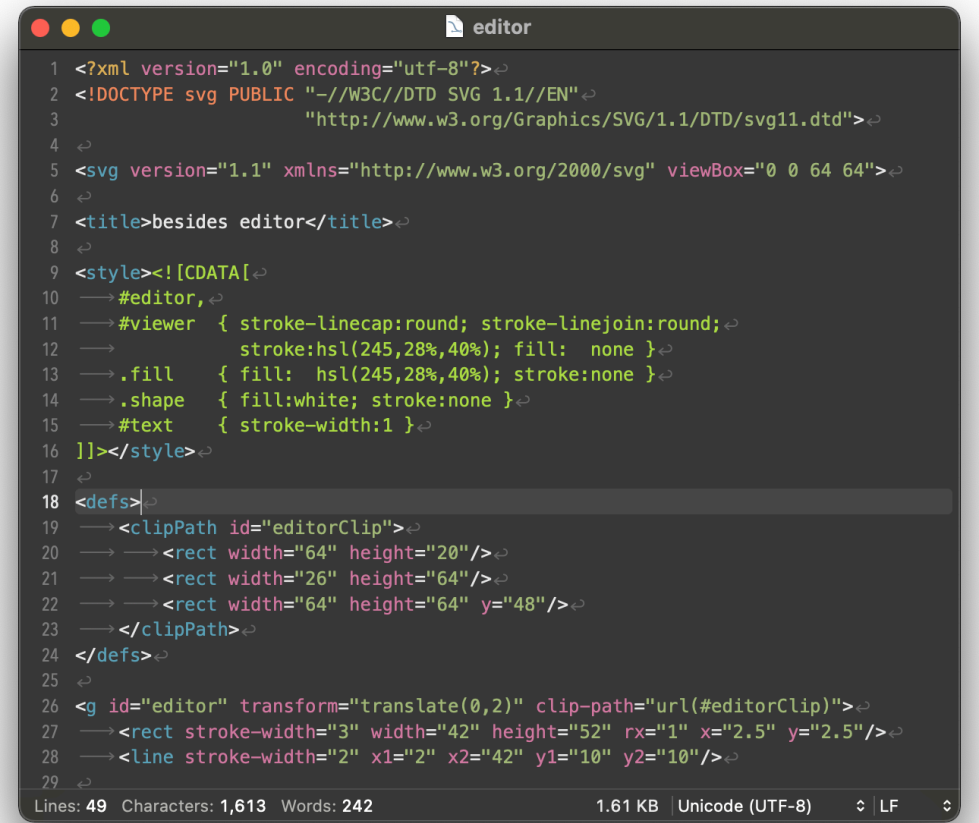
- uruchamianie i modyfikowanie prostych programów w języku Python v3
- obsługa specjalistycznego oprogramowania bioinformatycznego: Jupyter, Modeller, Gromacs
- korzystanie ze specjalistycznych serwisów internetowych, m.in. NCBI Entrez, iTasser, Galaxy

macOS

Terminal tekstowy



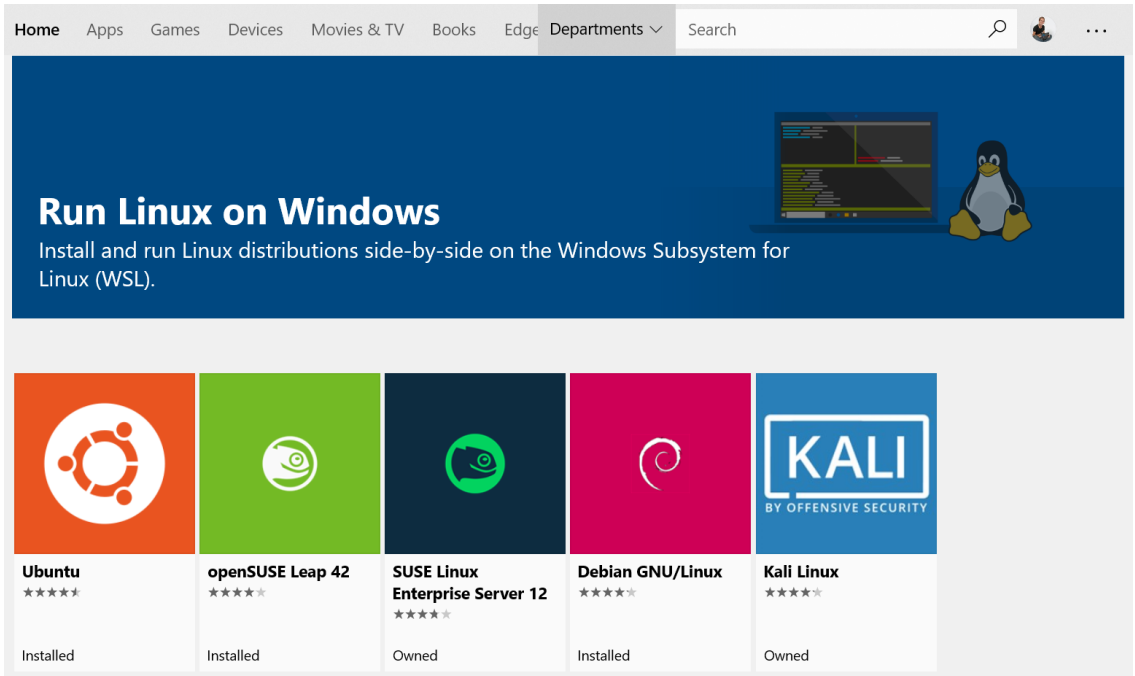
- Edytor tekstu (np. CotEditor, coteditor.com)



- Zmiana domyślnego programu powłoki (zsh) na coś innego (np. bash, csh): chsh -s /bin/bash.

Windows 10+

- Windows Subsystem for Linux,
  instalacja: https://docs.microsoft.com/en-us/windows/wsl/install,
  dobre praktyki: https://docs.microsoft.com/en-us/windows/wsl/setup/environment



- edytor tekstu (VSCode (dla chcących programować), SublimeText4 (lekki i minimalistyczny))

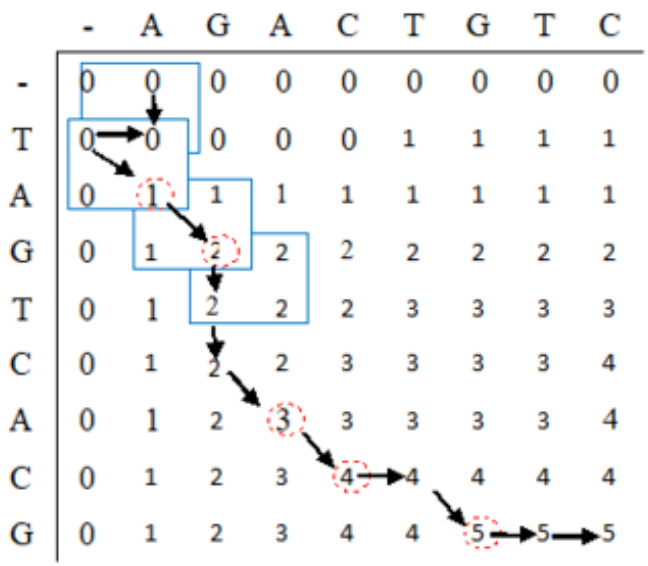
MacOSX/Win10+/Linux: CONDA

- Conda otwartoźródłowy systemem zarządzania pakietami oprogramowania dostępny na conda.io dla systemów Win10+, Linux i MacOSX napisany w języku Python
- Conda umożliwia tworzenie izolowanych środowisk programistycznych (dowolny język programowania) oraz instalowanie/aktualizowanie/usuwanie pakietów oprogramowania z uwzględnieniem zależności między nimi
- dystrybucja pakietów realizowana jest w kanałach (ang. channels), w których zgrupowano (zazw. tematycznie) różnorodne oprogramowanie
- minimalistyczną wersją systemu Conda jest Miniconda (przy instalacji wybiera się domyślną wersję interpretera Pythona, np. Python v3.11); wersje interpretera można później łatwo zmienić (upgrade/downgrade)
- wyszukiwanie pakietów oprogramowania można zrealizować z linii poleceń (np. conda search pymol) lub on-line na stronie anaconda.org (Anaconda to Conda poszerzona o zestaw pakietów Pythona (głównie z zakresu Danetyki (Data Science), wzgl. Inżynierii i Analizy Danych)
- użytkownicy Windows: warto zainstalować WindowsTerminal (Microsoft Store, free)

```
$ conda --version
$ conda create --name wbbib-python   # no i np. niestety nie ta wersja pythona, którą chcemy
$ conda activate wbbib-python
$ python --version
$ conda search python                # sprawdźmy, jakie wersje pythona dostępne
$ conda install python=3.8.3         # w bieżącym środowisku instalujemy to co potrzebujemy
$ python --version
$ conda deactivate wbbib-python
```

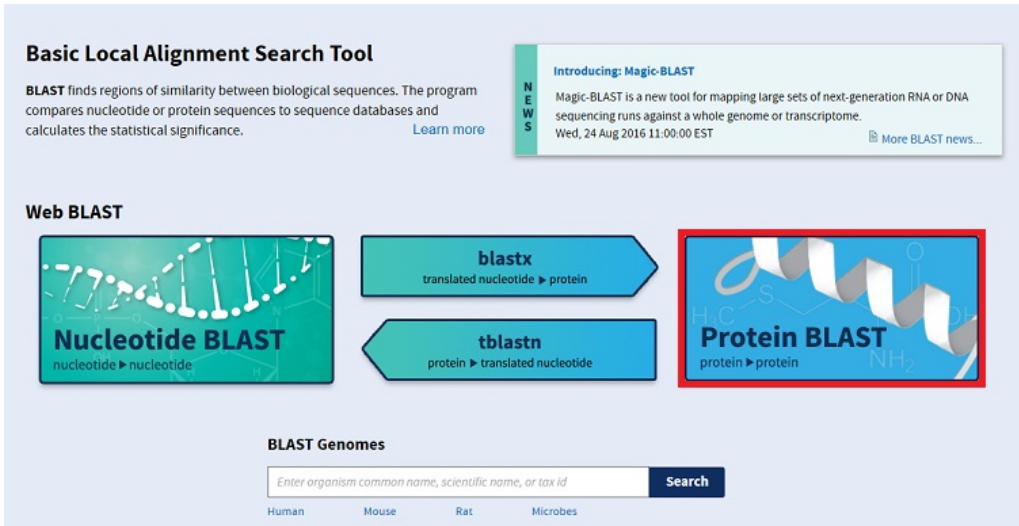
Podstawy bioinformatyki sekwencji

Porównywanie sekwencji



	-	A	G	A	C	T	G	T	C
-	0	0	0	0	0	0	0	0	0
T	0	0	0	0	0	1	1	1	1
A	0	1	1	1	1	1	1	1	1
G	0	1	2	2	2	2	2	2	2
T	0	1	2	2	2	3	3	3	3
C	0	1	2	2	3	3	3	3	4
A	0	1	2	3	3	3	3	3	4
C	0	1	2	3	4	4	4	4	4
G	0	1	2	3	4	4	5	5	5

Algorytmy heurystyczne



Zapis sekwencji: format FASTA

```
> gi|730028 | sp | P40692 | MLH1_HUMAN DNA MISMATCH REPAIR PROTEIN MLH1
MSFVAGVIRRLDETVVNRIAAGEVIQRPANAIKEMIENCLDAKSTSIQVIVKEGGLKLIQ
IQDNGTGIRKEDLDTTSKLQSFEDLASISTYGFRGEALASISHVAHVTITTKTADGKCAY
RASYSDGKLKAPPKPCAGNQGTQITVEDTRRKALKNPSEEYGKILEVVGRYSVHNAGISF
SVKKQGETVADVRTLPNASTVDNIRSIFGNAVSRELIEIGCEKMNGYISNANYSVKKCIF
LLFINHRLVESTSLRKAIETVYAAYLPKNTHPFLYLSLEISPQNVDVNVHPTKHEV
```

gi|730028 numer identyfikacyjny sekwencji

sp kod bazy danych sekwencji (np. gb, emb, pdb)

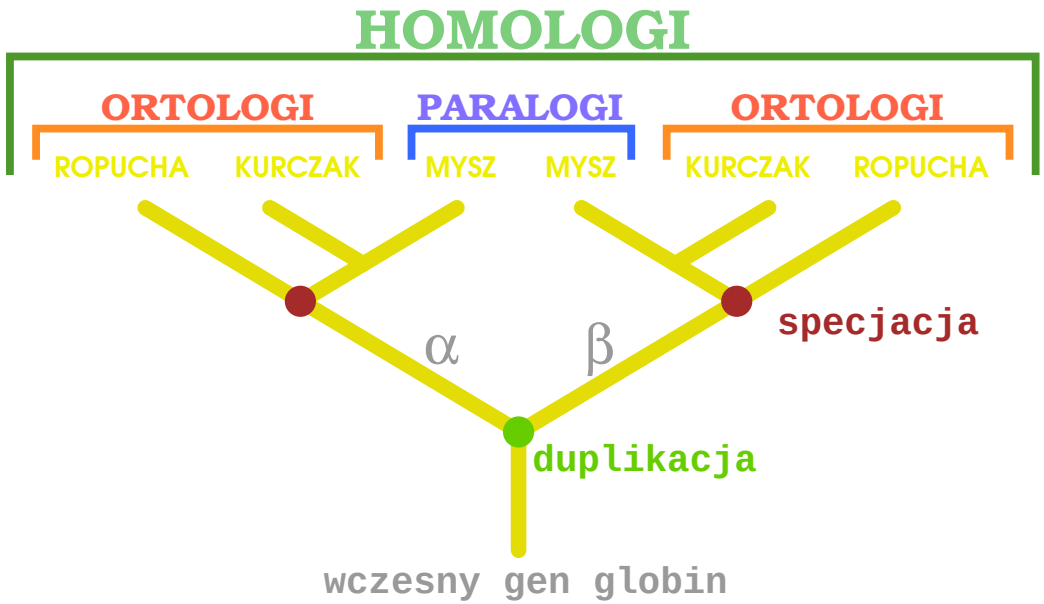
P40692 kod dostępowy, alfanumeryczny identyfikator rekordu bazy danych (zalecane używanie w cytowaniach)

MLH1_HUMAN nazwa rekordu: alfanumeryczny kod (max. 10 znaków) identyfikujący sekwencję; nie uwzględniany w wielu formatach, stąd nie zaleca się stosować go jako identyfikatora

DNA MISMATCH.. zwięzły opis sekwencji

Homologia, konwergencja czy przypadek

- podobieństwo sekwencji może wynikać z (1) pełnienia przez nie podobnych funkcji w podobnym środowisku (białka błonowe), (2) wspólnego pochodzenia ewolucyjnego, lub (3) przypadku
- homologia: opis relacji między genami / białkami (sekwencjami) wskazujący na ich wspólne pochodzenie ewolucyjne (własność binarna, przechodność homologii)
- sekwencje homologiczne łączy zależność ortologii jeśli ich ostatni przodek istniał w momencie specjacji
- jeśli ostatni wspólny przodek istniał w chwili duplikacji genu w genomie przodka, wtedy obie sekwencje łączy zależność paralogii



---

białko zaangażowane w poreplikacyjną korektę DNA

```
MSH2_HUMAN    626 EK-GQGRIILKASRHACVEVQDEIAFIPNDVYFEKDKQMFHIITGPNMGGKSTYIRQTGV 684
MSH2_YEAST    644 PMDSERRTHLISSRHPVLEMQDDISFISNDVTLESGKGDFLIITGPNMGGKSTYIRQVGV 703
MUTS_ECOLI    574 DK---PGIRITEGRHPVVEQVLNEPFIANPLNLSPQRR-MLIITGPNMGGKSTYMRQTAL 629
                          :  .**. :*   : .**.* : :.  :  : **************:**..:
```

Złudne podobieństwo sekwencji

HBA_HUMAN łańcuch α globin, CZŁOWIEK
HBB_HUMAN łańcuch β globin, CZŁOWIEK
LGB2_LUPLU leghemoglobina, ŁUBIN
GTA1_CAVPO S-transferaza glutationu, ŚWINKA MORSKA

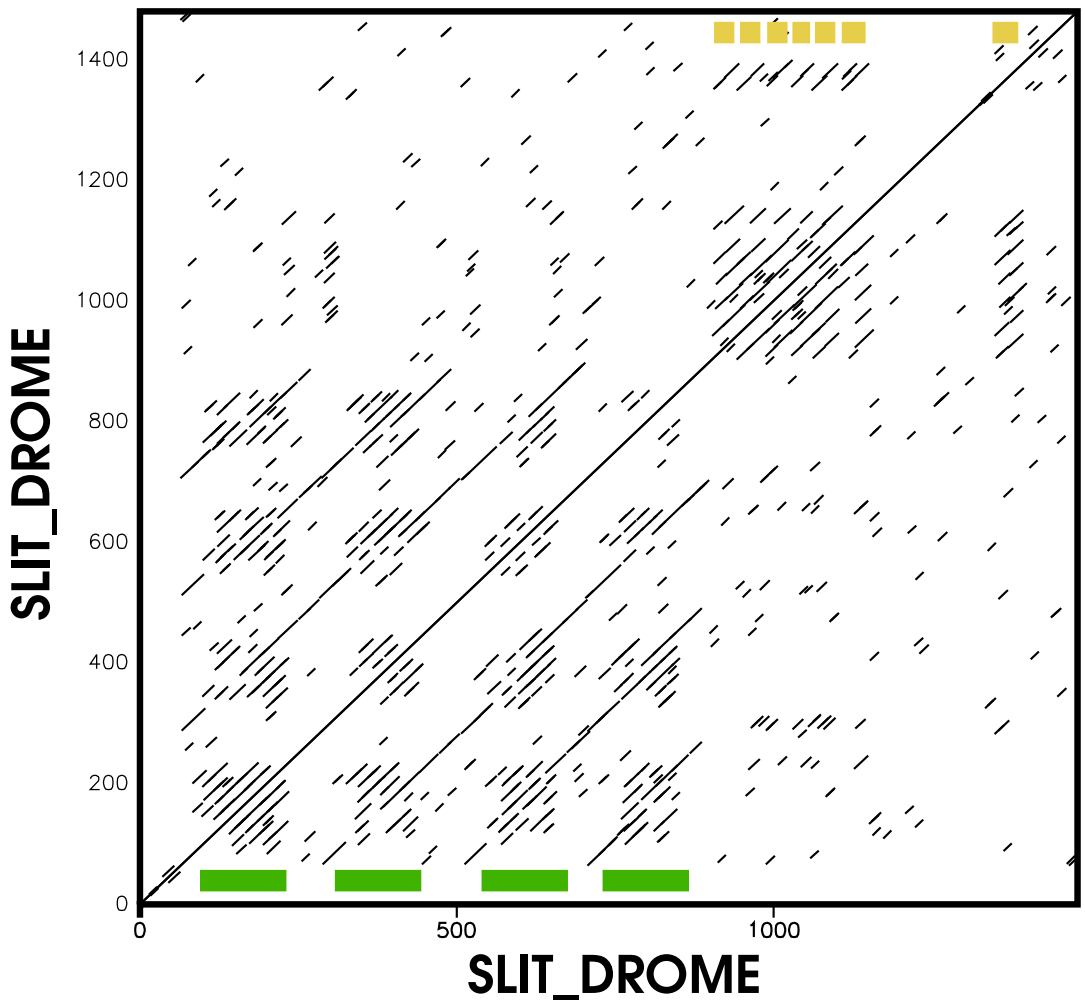
```
 HBA_HUMAN   HGSAQVKGHGKKVADALTNAVAHVDDMPNAL          | 38.7%
              |: :|| ||||| | :: :||:|::               | 64.5%
 HBB_HUMAN   MGNPKVKAHGKKVLGAFSDGLAHLDNLKGTF          |

 HBA_HUMAN   HGSAQVKGHGKKVADALTNA-----VAHVDDMPNAL     | 22.5%
               : ::: |  ||   :  |     |  |      |     | 38.7%
LGB2_LUPLU   GNNPELQAHAGKVFKLVYEAAIQLQVTGVVVTDATL     |

 HBA_HUMAN   HGSAQVKGHGKKVAD-ALTNAVAHVDDMPNAL         | 29.0%
             ||   : |:    ||  ||  :  |::   :|        | 48.4%
GTA1_CAVPO   HGQGYLVGNKLSKADILLTELLYMVEEFDASL         |
```

Powtórzenia domen w białku SLIT

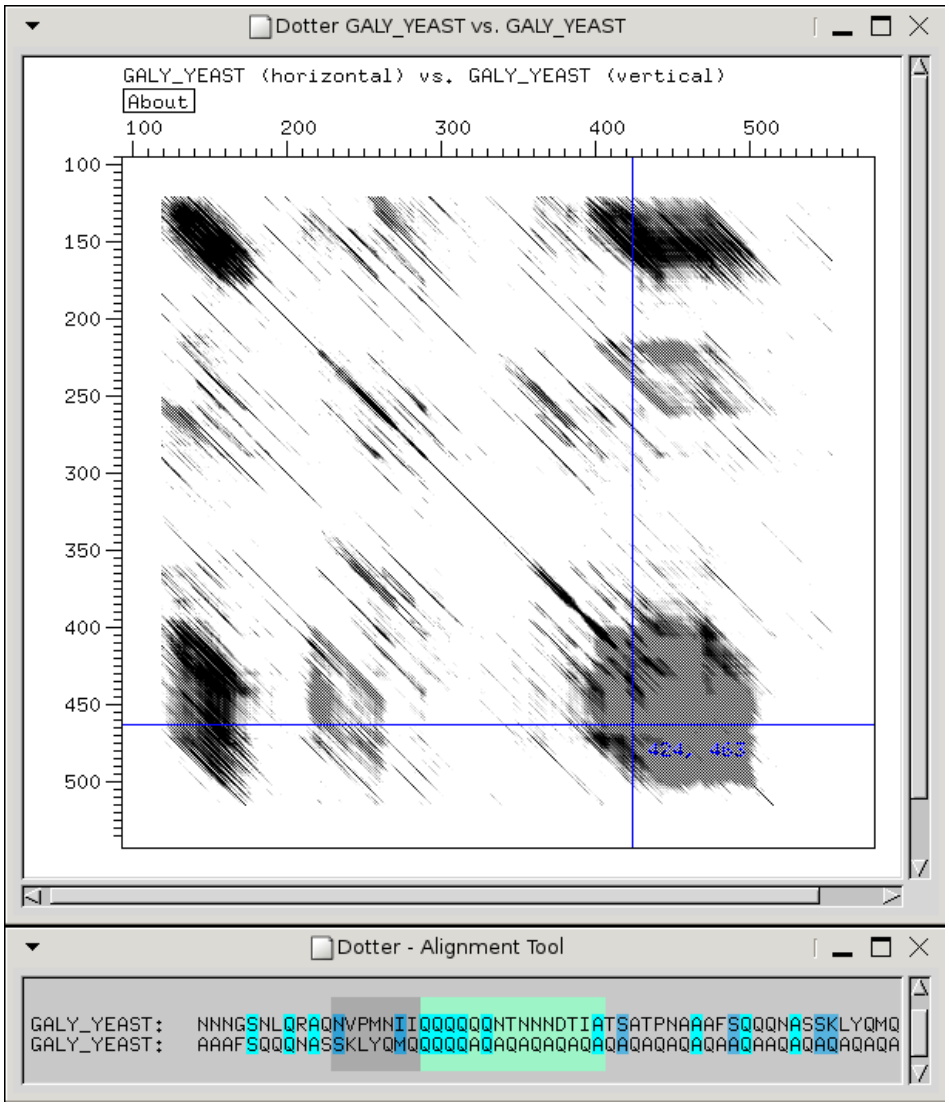
białko SLIT
(*Drosophila melanogaster*):

24LRR (*Leucine-Rich-Repeats*),
7 EGF

dotmatcher
`W13:T20 blosum62`



Identyfikacja LCR



- sekwencje o niskiej złożoności składu (LCR, ang. *Low Complexity Regions*) to odcinki, które cechuje nietypowy skład aminokwasowy/nukleotydowy
- identyfikację odcinków LCR często prowadzi się w oparciu o tzw. parametr złożoności składu K, który można wyrazić na wiele sposobów, np. dla:

$$K = \frac{1}{L} \log_N \frac{L!}{\prod_i n_i!}$$

parametr K będzie przyjmował wartości między 0.0 (niska złożoność składu) a 1.0 (wysoka złożoność składu)

Dopasowanie lokalne

CEL: identyfikacja krótkich fragmentów sekwencji o możliwie największym podobieństwie

ZASTOSOWANIA: domeny białkowe
egzony w genomowym DNA

REALIZACJA:

- zmodyfikowany algorytm dopasowania częściowego
- przy obliczaniu $\mathcal{F}$, za każdym razem, gdy $\mathcal{F}(i, j) < 0 \Rightarrow \mathcal{F}(i, j) = 0$, co oznacza rozpoczynanie nowego lokalnego dopasowania
- w kompletnej macierzy $\mathcal{F}$: $\forall_{i,j}\, \mathcal{F}(i, j) \geq 0$
- rekonstrukcja dopasowania (*backtracing*) zaczyna się od największej wartości $\mathcal{F}$ a kończy w pozycji, gdzie $\mathcal{F}(i, j) = 0$

Dopasowanie lokalne: algorytm

Programowanie dynamiczne i *backtracking*

$$\mathcal{F}(i,j) = \max \begin{cases} \mathcal{F}(i-1, j-1) \pm 1 \\ \mathcal{F}(i-1, j) - 2 \\ \mathcal{F}(i, j-1) - 2 \\ 0 \end{cases}$$

		T	G	A	T	A	G	G	A	C
	0	0	0	0	0	0	0	0	0	0
T	0	1	0	0	1	0	0	0	0	0
A	0	0	0	1	0	2	0	0	1	0
G	0	0	1	0	0	0	3	1	0	0
C	0	0	0	0	0	0	1	2	0	1

```
T G A T A G G A C
- - - T A G - - -
```

Macierz PAM250

Cys	12																
Ser	0	2															
Thr	-2	1	3														
Pro	-3	1	0	6													
Ala	-2	1	1	1	2												
Gly	-3	1	0	-1	1	5											
Asn	-4	1	0	-1	0	0	2										
Asp	-5	0	0	-1	0	1	2	4									
Glu	-5	0	0	-1	0	0	1	3	4								
Gln	-5	-1	-1	0	0	-1	1	2	2	4							
His	-3	-1	-1	0	-1	-2	2	1	1	3	6						
Arg	-4	0	-1	0	-2	-3	0	-1	-1	1	2	6					
Lys	-5	0	0	-1	-1	-2	1	0	0	1	0	3	5				
Met	-5	-2	-1	-2	-1	-3	-2	-3	-2	-1	-2	0	0	6			
Ile	-2	-1	0	-2	-1	-3	-2	-2	-2	-2	-2	-2	-2	2	5		
Leu	-6	-3	-2	-3	-2	-4	-3	-4	-3	-2	-2	-3	-3	4	2	6	
Val	-2	-1	0	-1	0	-1	-2	-2	-2	-2	-2	-2	-2	2	4	2	4
Phe	-4	-3	-3	-5	-4	-5	-4	-6	-5	-5	-2	-4	-5	0	1	2	-1
Tyr	0	-3	-3	-5	-3	-5	-2	-4	-4	-4	0	-4	-4	-2	-1	-1	-2
Trp	-8	-2	-5	-6	-6	-7	-4	-7	-7	-5	-3	2	-3	-4	-5	-2	-6

9																	
7	10																
0	0	17															

- macierz punktacji podstawień aminokwasowych $\mathcal{S}$ jest symetryczna
- średnia ocena podstawień w obrębie wydzielonych grup reszt aminokwasowych jest wyraźnie większa od zera, co wskazuje, że tego typu wymiany są częste w sekwencjach homologicznych
- w obrębie grupy, reszty aminokwasowe wykazują podobieństwo własności fizykochemicznych

$$\mathcal{S}_{ij} = 10 \log_{10} \frac{p(i,j|H)}{p(i,j|R)}$$

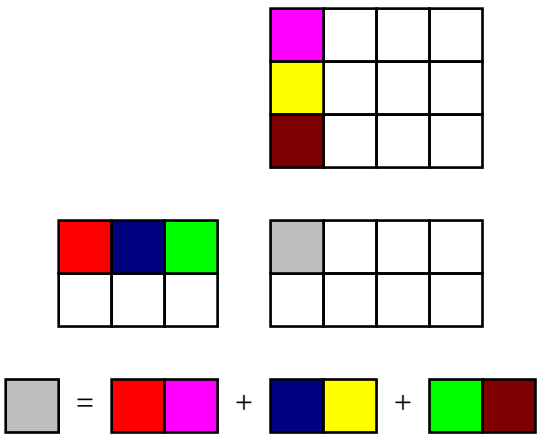
częstość podstawień w sekw. homologicznych w porównaniu z sekw. niespokrewnionymi

$\mathcal{S}_{ij}$	$\frac{p(i,j\|H)}{p(i,j\|R)} = 10^{\mathcal{S}_{ij}/10}$	
-7	0.2	mniejsza
+3	2.0	większa
0	1.0	taka sama

PAM

jako seria macierzy podstawień aminokwasowych (MPA) albo jednostka czasu ewolucji

Poszczególne macierze PAM zostały zoptymalizowane w celu możliwie najbardziej biologicznie poprawnego porównywania sekwencji o zróżnicowanej odległości ewolucyjnej



odległ. ewol. w PAM	ID [%]
30	75
80	50
120	38
250	20

- kolejne mnożenia PAM1 przez siebie:

  Zmiany w długim czasie są ekstrapolowane na podstawie zmian w krótkim czasie co prowadzi jednak do propagacji i kumulowania błędów wynikających z ograniczeń przyjętego modelu podstawień aminokwasowych

  Np. przy porównywaniu blisko spokrewnionych sekwencji obserwowane podstawienia aminokwasowe wynikają głównie (80%) ze zmian pojedynczego nukleotydu, podczas gdy dla dłuższych czasów dominujące stają się zmiany dwu- i trzy nukleotydowe

Względna entropia MPA

- wielkość entropii względnej H MPA oznacza średnią ocenę pojedynczej pozycji dopasowania (nie mylić ze średną ważoną wartością MPA, która powinna być liczbą ujemną)

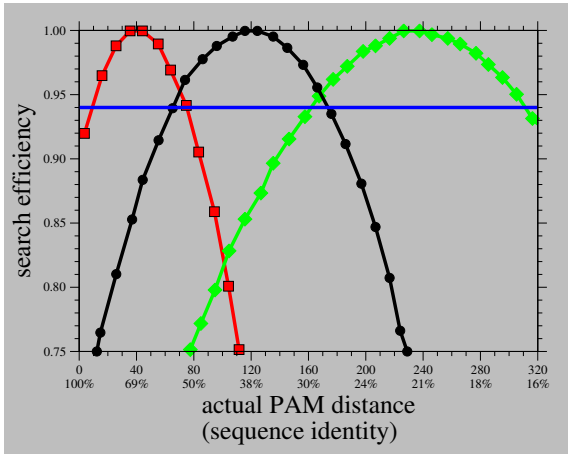
$$H = \sum_{i,j} p(i,j|H)\, s_{ij} = \sum_{i,j} p(i,j|H) \log_2 \frac{p(i,j|H)}{p(i,j|R)}$$

- znajomość średniej ilości informacji na pojedynczej pozycji dopasowania (H), umożliwia wyznaczenie najmniejszej długości MSP ($\min|\mathrm{MSP}|$) koniecznej do uznania za istotne statystycznie wyników wyszukiwania przeprowadzonego z wykorzystaniem określonej MPA

PAM distance	H [bit]	$\min\|\mathrm{MSP}\|$ (30 bit)
0	4.17	8
20	2.95	11
40	2.26	14
60	1.79	17
80	1.44	21
100	1.18	26
120	0.98	31
140	0.82	37
160	0.70	43
180	0.60	51
200	0.51	59
220	0.45	68
240	0.39	78
260	0.34	89
280	0.30	100
300	0.27	113
320	0.24	127
340	0.21	141

- dla dużych wartości H, już względnie krótkie dopasowania (MSP) mogą być uznane za istotne statystycznie

PAM: wpływ wyboru MPA na wynik dopasowania



macierz PAM	zakres długości
40	9÷21
120	19÷50
240	47÷123

Tabela

Średni wynik (w bitach) na pozycję dopasowania wyznaczanego z podaną macierzą PAM dla sekwencji o zadanej odległości ewolucyjnej

PAM matrix used	Actual PAM distance D of segments							
	40	80	120	160	200	240	280	320
40	2.26	1.31	0.62	0.10	-0.30	-0.61	-0.86	-1.06
80	2.14	1.44	0.92	0.53	0.23	-0.02	-0.21	-0.37
120	1.93	1.39	0.98	0.67	0.42	0.22	0.06	-0.07
160	1.71	1.28	0.95	0.70	0.50	0.33	0.20	0.09
200	1.51	1.16	0.90	0.68	0.51	0.38	0.26	0.17
240	1.32	1.05	0.82	0.65	0.51	0.39	0.29	0.21
280	1.17	0.94	0.75	0.60	0.48	0.38	0.30	0.23
320	1.03	0.84	0.68	0.56	0.46	0.37	0.30	0.24

- przeszukiwanie jest efektywne jeśli różnica między bieżącą oceną jego wyników a oceną uzyskaną z wykorzystaniem optymalnie dobranej macierzy jest mniejsza od 2 bitów (4-krotna różnica w istotności statystycznej, $32/34 \approx 0.94$)
- maksymalna ocena jeśli $D = M$; dla każdej macierzy M, im mniejszy rzeczywisty dystans ewolucyjny D tym większa ocena dopasowania

Szukanie homologów: problem klasyfikacji

czułość (ang. *sensitivity*) – parametr (C) określający zdolność odszukania wszystkich sekwencji homologicznych

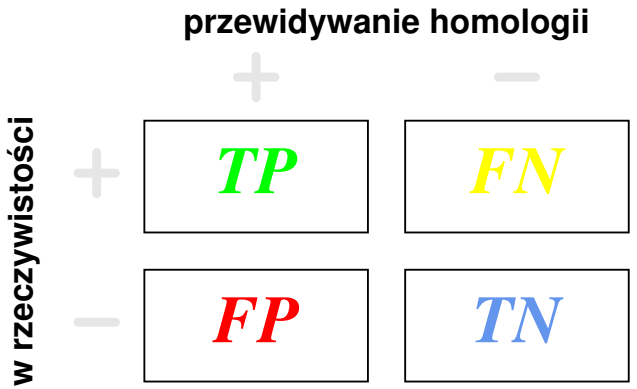
$$C = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

- wynik przeszukiwania jest zwykle mniejszy od oczekiwanego, ponieważ nie zawiera odległych homologów o marginalnym podobieństwie sekwencji
- sekwencje homologiczne, które nie zostały odszukane określane są jako wynik fałszywie ujemny (FN, ang. *false negatives*)

selektywność (swoistość, ang. *selectivity*) – parametr (S) określający zdolność odszukania wyłącznie sekwencji homologicznych

$$S = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

- wynik przeszukiwania może zawierać dodatkowe sekwencje niesłusznie uznane za homologi, tzw. wynik fałszywie dodatni (FP, ang. *false positives*)



TP : poprawnie zidentyfikowane homologii

FN : niewyszukane homologii

FP : sekwencje niehomologiczne błędnie opisane jako homologii

TN : rzeczywiste sekwencje niehomologiczne

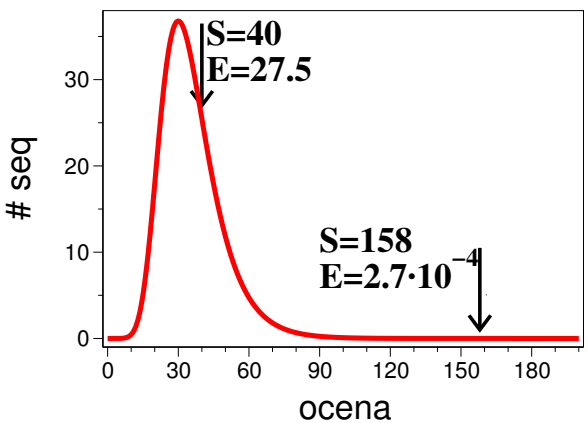
Wyniki przeszukiwania bazy danych sekwencji

- lista sekwencji uszeregowanych według malejącego podobieństwa do kwerendy wyznaczonego w oparciu o przyjęty system punktacji
- dla każdej z wyszukanych sekwencji podawany jest jej identyfikator, fragment opisu, długość sekwencji, punktowa ocena podobieństwa z kwerendą, ta sama ocena wyrażona w jednostkach bezwzględnych (bitach), wartość parametru E (ang. *E-value*)
- wartość $E < 0.01$ sugeruje istnienie homologii między sekwencją kwerendy a wyszukaną sekwencją bazodanową

```
PRIO_BOVIN P10279 bos taurus (bovine). major prion ( 264) 1430 266.0 5.6e-71
PRIO_CHICK P27177 gallus gallus (chicken). major p ( 273)  438  88.5 1.7e-17
K1CI_HUMAN P35527 homo sapiens (human). keratin, t ( 622)  206  47.3 9.3e-05
RB56_HUMAN Q92804 homo sapiens (human). tata-bindi ( 592)  194  45.1  0.0004
....
ROA1_MACMU Q28521 macaca mulatta (rhesus macaque). ( 319)  157  38.2   0.026
LEG3_CANFA P38486 canis familiaris (dog). galectin ( 295)  156  38.0   0.027
K1CJ_MOUSE P02535 mus musculus (mouse). keratin, t ( 569)  159  38.8    0.03
K1CM_MOUSE P08730 mus musculus (mouse). keratin, t ( 437)  157  38.4   0.032
K1CJ_BOVIN P06394 bos taurus (bovine). keratin, ty ( 526)  158  38.6   0.032

GRP8_ARATH Q03251 arabidopsis thaliana (mouse-ear  ( 169)  151  36.9   0.035
EGG2_SCHJA P19469 schistosoma japonicum (blood flu ( 207)  152  37.1   0.035
....
EGG1_SCHMA P06649 schistosoma mansoni (blood fluke ( 173)  127  32.6    0.69
SANT_PLAFV P09593 plasmodium falciparum (isolate v ( 375)  131  33.6    0.72
```

Strategia przeszukiwania baz danych sekwencji

- bardzo szybkie wyznaczenie przybliżonego dopasowania lokalnego z każdą z sekwencji w bazie danych
- jeśli ocena takiego przybliżonego dopasowania jest odpowiednio wysoka, konstruowane są kolejno coraz bardziej dokładne dopasowania (im dokładniejszy algorytm wyznaczania dopasowania tym większa jego złożoność obliczeniowa, dlatego algorytm programowania dynamicznego uruchamiany jest wyłącznie zwykle dla kilkudziesięciu sekwencji najbardziej podobnych do sekwencji kwerendy)
- statystyczna ocena istotności wyników przeszukiwania bazy danych:

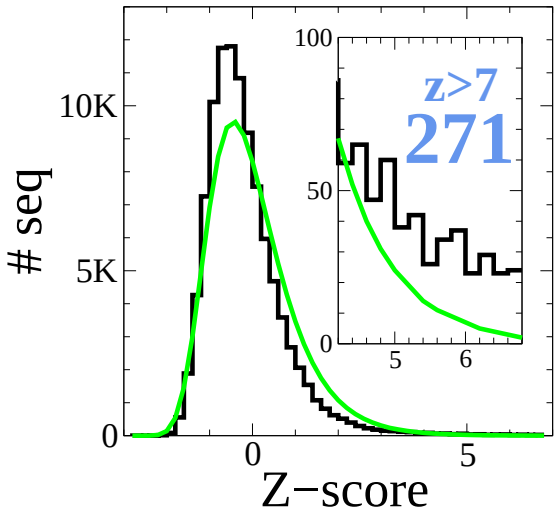


  - jaka jest średnia liczba sekwencji, których podobieństwo jest przypadkowe (analogia do relacji między szumem a sygnałem)
  - jak bardzo wyznaczona ocena podobieństwa między kwerendą a odszukaną sekwencją bazodanową różni się od średniej oceny porównań z innymi sekwencjami

FastA W.R. Pearson & D.J. Lipman (1988) Proc. Natl. Acad. Sci. USA 85:2444–2448

BLAST S.F. Altschul, T.L. Madden, A.A. Schäffer, J. Zhang, Z. Zhang, W. Miller, D.J. Lipman (1997) Nucleic Acids Res. 25:3389–3402

Wpływ składu aminokwasowego na oceny istotności statystycznej



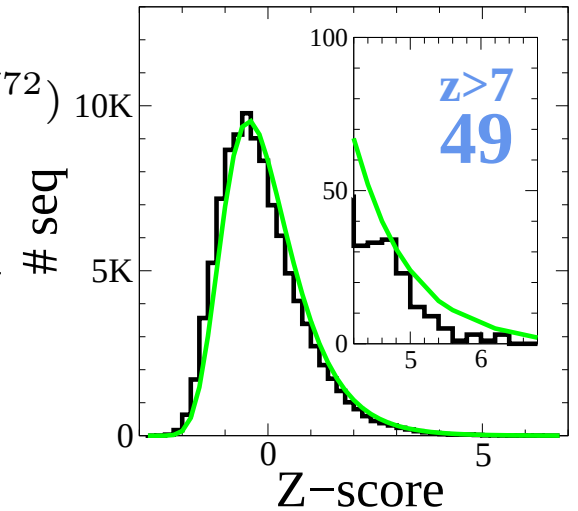
$$P(z \geq \mathcal{Z}) = 1 - \exp(-e^{-1.282\mathcal{Z} - 0.5772})$$

$$E(z \geq \mathcal{Z}) = DP(z \geq \mathcal{Z})$$

$$P(z \geq 7) < 7.11 \cdot 10^{-5}$$

$$E(101529) < 7.22$$

rzeczywistych homologów: 43



Założenia modelu statystycznego, przyjęte przy wyznaczaniu wartości *E* (ang. *E-value* ↔ *Error per query*, *false-positive rate*):

- oceny podobieństwa sekwencji tworzą rozkład wartości ekstremalnej
- losowo wygenerowane sekwencje mają takie same własności jak rzeczywiste sekwencje niespokrewnione

W przypadku białek o nietypowym składzie aminokwasowym, ostatnie założenie nie jest spełnione.

PRIO_ATEPA P51446

MANLGYWMLVLFVATWSDLGLCKKRPKPGG
WNTGGSRYPGQGSPGGNRYPPQGGGWGQPH
GGGWGQPHGGGWGQPHGGGWGQPHGGGWGQ
AGGTHNQWNKPSKPKTNMKHMAGAAAAGAV
VGGLGGYMLGSAMSRPLIHFGNDYEDRYYR
ENMYRYPNQVYYRPVDQYNNQNNFVHDCVN
ITIKQHTVTTTTKGENLTETDVKMMERVVE
QMCITQYERESQAYYQRGSSMVLFSSPPVI
LLISFLIFLIVG

Standaryzacja ocen lokalnego podobieństwa

- standardową jednostką podobieństwa pary sekwencji jest BIT (1 `bit` odpowiada ilości informacji koniecznej do rozróżnienia między dwiema możliwościami, np. dzięki odpowiedziom TAK/NIE na odpowiednio sformułowane pytania; dla dokonania wyboru spośród N możliwości potrzeba $\log_2 N$ `bitów` informacji):

$$E(s \geq S_{\mathrm{pkt}}) = \mathcal{K} m n \, \mathrm{e}^{-\lambda S_{\mathrm{pkt}}}$$

$$S_{\mathrm{bit}} = \frac{\lambda S_{\mathrm{pkt}} - \ln \mathcal{K}}{\ln 2}$$

$$E(s \geq S_{\mathrm{bit}}) = \frac{mn}{2^{S_{\mathrm{bit}}}}$$

- wzrost oceny o 1 `bit` powoduje dwukrotny wzrost istotności statystycznej (tj. E staje się dwukrotnie mniejsze)

Maskowanie sekwencji

- wiele sekwencji aminokwasowych i nukleotydowych zawiera powtarzające się odcinki sekwencji (powtórzenia tandemowe w DNA, struktury skręconych helis: *coiled-coils*, sekwencje rozproszone (ALU), etc.) lub fragmenty o niskiej złożoności składu
- jeśli sekwencja kwerendy zawiera taki fragment, za statystycznie istotne wyniki przeszukiwania bazy sekwencyjnej mogą być uznane dopasowanie z niespokrewnionymi ewolucyjnie sekwencjami

problem	opis	rozwiązanie
powtórzenia	tandemowe, np. CACACA: VNTR (*Variable Numer of Tandem Repeats*, STR (*Short Tandem Repeats*), sekwencje rozproszone: SINE (*Short Interspersed Repetitive Element*, np. ALU), LINE, etc.	DUST, RepeatMasker, XNU
niska złożoność	fragmenty sekwencji złożone z jendego lub niewielkiej liczby typów reszt (*Low Complexity Regions*: LCR, np. *leucine-rich/proline-rich* regions), odcinki poly-A w DNA	DUST, SEG
wektory	fragmenty wektorów klonujących są często na końcach, rzadziej w środku sekwencji bazowych/kwerendy	CrossMatch, VecScreen

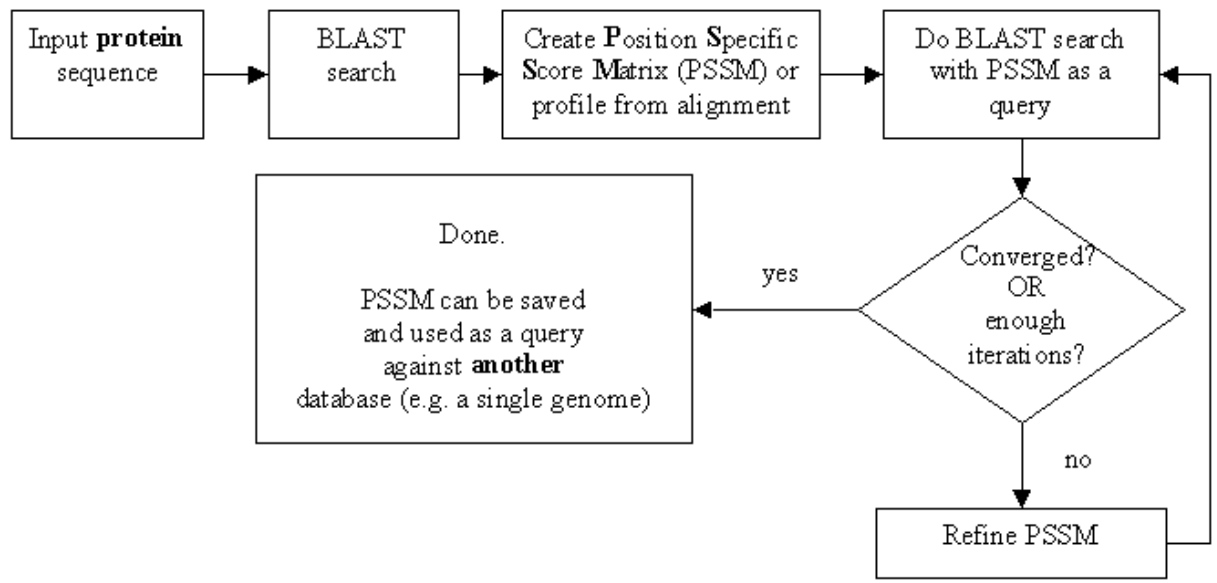
Macierz podstawień vs. profile podstawień

- przeszukiwanie bazy, w trakcie którego podobieństwo sekwencji białkowych oceniane jest na podstawie macierzy podstawień, nader często nie pozwala na popawne zidentyfikowanie odległych homologów wykazujących marginalne podobieństwo do sekwencji kwerendy
- znacznie bardziej czułe jest wyszukiwanie oparte o tzw. profile podstawień (PSSM, *Position Specific Scoring Matrix*), w których ocena podobieństwa/różnicy pary reszt aminokwasowych w porównywanych sekwencjach zależy nie tylko od rodzaju podstawienia (tj. np. $F \leftrightarrow W$) ale również od pozycji w sekwencji.
  Stąd mutacje (podstawienia/przerwy) w obrębie konserwatywnych rejonach centrów aktywnych białek cechuje zdecydowanie większy koszt niż zmiana tego samego typu mająca miejsce w funkcjonalno-strukturalnie neutralnym fragmencie sekwencji.

		A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
1	Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1
2	L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1
3	P	-1	-2	-2	-2	-3	-2	-1	-2	-2	-3	-3	-1	-3	-4	8	-1	-1	-4	-3	-3
4	S	1	-1	0	-1	-1	0	0	-1	-1	-3	-3	0	-2	-3	-1	5	1	-3	-2	-2
5	C	-1	-4	-3	-4	9	-3	-4	-3	-3	-2	-2	-3	-2	-3	-3	-1	-1	-3	-3	-1
6	T	0	-1	0	-1	-1	-1	-1	-1	-2	-2	-3	-1	-2	-3	-1	4	3	-3	-2	-2
7	Y	-2	-3	-3	-4	-3	-2	-3	-4	1	-1	-1	-3	-1	5	-4	-2	-2	1	7	-2
8	Y	-1	-1	-1	-1	-2	0	-1	-2	6	-2	-1	-1	-1	1	-1	-1	-1	0	5	-2
9	V	-1	-2	-2	-2	-1	-2	-2	-2	-2	1	2	-2	0	-1	-2	-2	-1	-2	-1	4
10	S	-1	-1	-1	-1	-3	3	3	-2	-1	-2	1	0	-1	-2	-2	2	-1	-3	-2	-2

PSI-BLAST

- metoda *Position Specific Iterated Blast* pozwala na iteracyjne przeszukiwanie bazy sekwencyjnej z wykorzystaniem dynamicznie tworzonego profilu podstawień:
  - w pierwszej iteracji, podobieństwo sekwencji oceniane jest w taki sam sposób jak w metodzie BLAST (np. BLOSUM62, $\gamma(k) = 10 + k$)
  - w oparciu o wyszukane sekwencje, których ocena podobieństwa charakteryzuje się współczynnikiem istotności statystycznej mniejszym od zadanej wartości progowej (zwykle $E < 0.01$), tworzony jest profil sekwencyjny używany w kolejnej iteracji



© Olga Zhaxybayeva

Obiekty

- Python jest językiem programowania obiektowego, co w praktyce oznacza, że w Pythonie **wszystko jest obiektem**
- obiekt jest abstrakcyjnym pojęciem odwołującym się do elementarnego składnika programu; utworzenie obiektu prowadzi do pojawienia się jego instancji, czyli wydzielonego w pamięci komputera obszaru zajmowanego przez dane tekstowe lub binarne oraz kod wykonywalny funkcji (tzw. metod), które na nich działają
- obiektem w języku Python są zarówno zmienne typu tekstowego (string), zmiennoprzecinkowego (float), całkowitego (int) jak i funkcje, klasy (przepis definiujący obiekt) i moduły (oddzielne pliki tekstowe zawierające dane tekstowe i kody źródłowe funkcji lub klas)
- nazwa obiektu (`[a-zA-Z_][a-zA-Z0-9_]*\.py`) nie może być na liście słów zastrzeżonych (and, print, import,..)
- elementy składowe danego obiektu to jego atrybuty; standardowa funkcja dir( ) wyświetla listę atrybutów danego obiektu
- poszczególne atrybuty obiektu są dostępne dzięki tzw. notacji z kropką, tj. `ob_name.att_name`

Operatory

przypisania tożsamość obiektu (tzw. instancje) definiuje jego nazwa (identyfikator) za pomocą operatora przypisania = (ang. *assignment*)

```
>>> tekst = 'Witaj'                                    # przypisanie zwykłe
>>> bruenet, ile_znakow = 'Witaj J-23', len(tekst)     # jednoczesne
>>> j23 = bruener = 'Stawka większa niż życie'         # wielokrotne
>>> bruenet.swapcase()
'wITAJ j-23'
```

zawierania in (np. `print('A' in 'Abracadabra')`)

tożsamości is (np. `A is A`, `A is B`)

matematyczne dodawanie (`3+(1+2j).real`), odejmowanie (`4-2`), mnożenie (`2*'ha'`), dzielenie (`10.2/3.8`), reszta z dzielenia (`10%3`), potęgowania (`2**8`)

logiczne not, or (lub), and, albo

Typy sekwencyjne

lista (ang. *list*) pozwala na przechowywanie uporządkowanej sekwencji obiektów dowolnego typu. Listę definiuje się poprzez przypisanie wybranemu identyfikatorowi sekwencji obiektów ujętych w nawiasy kwadratowe i rozdzielonych przecinkami.

```
>>> a = [1,'ala',14,'aga',\
...      10.2,'abra']
>>> a
[1, 'ala', 14, 'aga', 10.2, 'abra']
>>> a.sort()
>>> a
[1, 10.2, 14, 'abra', 'aga', 'ala']
>>> a.append([2,3.14]); print(a)
[1, 10.2, 14, 'abra', 'aga', 'ala', [2, 3.14]]
>>> a[3] = a[3].upper(); a[2:5]
[14, 'ABRA', 'aga']
>>> b = a[-5:-1]; b.reverse(); b
[[2, 3.14], 'ala', 'aga', 'ABRA', 14]
```

Typy sekwencyjne

krotka (ang. *tuple*) niemodyfikowalna lista (stała kolejność obiektów, wartości zmiennych). Składnia: `(a, b, ...)`

```
>>> xyz = (1, 0.5, 'trzy')
>>> xyz[-2:]
(0.5, 'trzy')
>>> xyz[-1]
'trzy'
>>> xyz[-1] = 3
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object doesn't support item assignment
```

Szeregi arytmetyczne i pętle

range jest niemutowalną sekwencją liczb całkowitych, często wykorzystywaną w pętlach, składnia:
range([start,] stop[, step])

- w odróżnieniu od list, obiekt range zajmuje tę samę niewielką ilość pamięci operacyjnej niezależnie od rozmiaru/zakresu
- **range** można określić jako leniwą sekwencję liczb
- leniwość (wartościowanie leniwe, ang. lazy evaluation, przeciwieństwo: wartościowanie zachłanne/gorliwe (ang. eager evaluation)) w programowaniu oznacza obliczanie wyrażeń tak późno jak to możliwe

range

```
>>> range(10)
range(0,10)
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> tuple(range(1,10))
(1, 2, 3, 4, 5, 6, 7, 8, 9)
>>> list(range(3,10,2))
[3, 5, 7, 9]
>>> list(range(10,3,-2))
[10, 8, 6, 4]
>>> r = range(0,15,3)
>>> r
range(0, 15, 3)
>>> 9 in r
True
>>> 11 in r
False
>>> r[-1]
12
>>> r[1:4:2]  # (3, 9)
range(3, 12, 6)
```

Typy sekwencyjne

ciąg tekstowy (ang. *string*) sekwencja znaków ujęta między znakami apostrofu lub cudzysłowa/-ów

```
>>> cool = 'hej ho'
>>> ((cool[:3]+", ")*4).center(35)
'     hej, hej, hej, hej,      '
>>> opis = """
     hej ho, hej ho,
do pracy by sie szlo """
... ... >>> print(opis)

     hej ho, hej ho,
do pracy by sie szlo
```

Typ odwzorowujący: leksykony

dane w leksykonie (ang. *dictionary*) zorganizowane są w taki sposób, że każdemu ze zdefiniowanych słów kluczowych (ang. *keys*) odpowiada tylko jedna wartość (ang. *values*)

```
>>> kesz = {'alicja': 2150,
...         'wiktoria': 4500,
...         'terminator': 2150}
>>> kesz.keys()
['wiktoria', 'alicja', 'terminator']
>>> kesz.values()
[4500, 2150, 2150]
>>> kesz['alicja']
2150
>>> kesz[2150]
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
KeyError: 2150
```

leksykon nie może zawierać dwóch takich samych słów kluczowych i nie przechowuje informacji o kolejności swoich elementów

słowa kluczowe leksykonów mogą być typu string, int, float, tuple ale np. nie list; wartości w leksykonie mogą być dowolnego typu

```
>>> kesz['nostromo'] = 3500
>>> kesz['alicja'] = 2750
>>> del(kesz['terminator'])
>>> kesz.items()
[('wiktoria', 4500), ('alicja', 2750), ▷
                     ('nostromo', 3500)]
>>> kesz.has_key('Nostromo')
0
>>> kesz[20031017] = kesz.keys()
>>> kesz.get(20031017)
['wiktoria', 'alicja', 'nostromo']
>>> kesz[(4500,2750)] = \
...       {'naczelna':'wiktoria',
...        'szary pismak':'alicja'}
>>> x = kesz[20031017]
>>> kesz[x] = 'to sie nie uda'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: list objects are unhashable
```

W celu kontrolowania przepływu informacji w programie oraz sterowania przebiegem programu wykorzystywane są:

konstrukcje warunkowe (if..then..else), oraz

konstrukcje iteracyjne (for, while)

- iteracja (łac. *iteratio*) – czynność powtarzania (najczęściej wielokrotnego) tej samej instrukcji (albo wielu instrukcji) w pętli. Mianem iteracji określa się także wszystkie operacje wykonywane wewnątrz takiej pętli.

http://pl.wikipedia.org/wiki/Iteracja

Składnia każda z linii kodu w bloku poleceń wykonywanych w ramach struktur decyzyjnych i cyklicznych jest poprzedzona jednakową liczbą znaków odstępu lub tabulacji, stąd odpowiednie fragmenty kodu widoczne są jako bloki poleceń.

Konstrukcje warunkowe

- umożliwiają sterowanie wykonywaniem programu
- ich podstawowym elementem są wyrażenia warunkowe
- warunek jest spełniony jeśli odpowiednie wyrażenie warunkowe zwraca liczbę różną od zera lub niepusty obiekt

Składnia:

```
if ...
if ... else ...
if ... elif ...
if ... elif ... else ...
```

```
>>> name = 'Mozart'
>>> if name == 'Mozart':
...     print('Wolfgang Amadeusz',name)
... elif name == 'Bach':
...     print('Jan Sebastian',name)
... else:
...     print(name)
...
Wolfgang Amadeusz Mozart
>>>
```

Operatory porównań i logiczne

- operatory porównań:
  ==, <, >, <=, >=, !=
- koniunkcja, alternatywa, alternatywa wykluczająca, negacja

X and Y	prawda jeśli zarówno X jak i Y jest prawdą, X & Y
X or Y	prawda jeśli X lub Y jest prawdą, X | Y
operator.xor(X, Y)	prawda jeśli X albo albo Y jest prawdą, X ^ Y
not X	prawda jeśli X jest fałszem

```
>>> a, name = 1920, 'Muchomorek'
>>> if 1900 < a < 1930 and \
...        name == 'Muchomorek':
...     print('Witaj Muchomorku')
...
'Witaj Muchomorku'
>>> a = []
>>> if a:  print('Lista nie jest pusta')
...
>>> a.append(None)
>>> if a[0]:  print('Element okreslony')
...
>>> if a[0] is None:
...   print('Element jest nie okreslony')
...
Element jest nie okreslony
```

zobacz: http://docs.python.org/library/operator.html#mapping-operators-to-functions

Konstrukcje iteracyjne

Bloki poleceń w programie wykonywane wielokrotnie (np. w ramach cyklów obliczeniowych → iteracji).

- pętla `while` (dopóki warunek jest spełniony wykonuj instrukcje w bloku):
- pętla `for` (dla kolejnych elementów listy, ciągu tekstowego, tupletu, innych obiektów (tych z atrybutem `__getitem__`) wykonuj instrukcje w bloku)

```
>>> start, end, step = 10, 50, 5
>>> while start < end:
...     print(start,end='')
...     start = start + step
... else:
...     print('\nKoniec petli')
...
10 15 20 25 30 35 40 45
Koniec petli
>>> range(10,50,5)
[10, 15, 20, 25, 30, 35, 40, 45]
>>> for i in range(10,50,5):
...     print(i,end='')
... else:  print('\nKoniec petli')
...
10 15 20 25 30 35 40 45
Koniec petli
```

Sterowanie w konstrukcjach iteracyjnych

break przerywa wykonywanie instrukcji w pętli

continue rozpoczyna wykonywanie kolejnego cyklu w pętli

pass nie wykonuje żadnych działań. Jej użycie zwykle wynika z konieczności zastosowania się do reguł składniowych.

```
>>> for i in range(10,50,5):
...       if i%10: continue
...       print(i,end='')
...
10 20 30 40
>>> while 1:
...       if i<=10: break
...       print(i,end='')
...       i = i - 5
...
45 40 35 30 25 20 15
>>> while 1:
...     pass # Ctrl-C aby przerwac
...
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
KeyboardInterrupt
```

Funkcje

- w konwencji programowania strukturalnego, funkcja jest grupą (blokiem) poleceń zdefiniowaną przez programistę
- warto definiować funkcje, ponieważ:
  - zawierają implemenację procedur wykorzystywanych wielokrotnie w programie
  - przy projektowaniu struktury programu, ich użycie pozwala na podzielenie złożonego problemu na oddzielne (prostsze do realizacji) zadania
  - kod programu napisanego z użyciem funkcji jest bardzej czytelny i łatwiejszy do testowania w celu sprawdzania poprawności działania, lokalizacji i usuwania ewentualnych błędów (ang. *debugging*), poprawiania efektywności (ang. *profiling*)

- funkcję reprezentuje w programie jej nazwa (identyfikator)
- funkcja może pobierać jeden lub więcej argumentów w postaci listy ujętej w parę nawiasów okrągłych i rozdzielonych przecinkami
- lista argumentów funkcji może zawierać argumenty podane z ich domyślną wartością
- funkcja zwraca wynik dowolnego typu lub pusty obiekt (None)

```
>>> def transcribe(dna):
...     """Return DNA string as RNA string, by
...     replacing T (thymine) with U (uracil)"""
...     return dna.replace('T', 'U')
...
>>> transcribe('CGAATATACT')
'CGAAUAUACU'
>>> def baseContent(dna,base='C', norm=100):
...     b = dna.count(base)
...     percent = float(b)/len(dna)*norm
...     return percent
...
>>> myDNA = 'CGAATATACT'
>>> baseContent(myDNA)
20.0
>>> baseContent(myDNA,'A')
40.0
>>> baseContent(myDNA, norm=1)
0.2
>>> print(transcribe.__doc__)
Return DNA string as RNA string, by
   replacing T (thymine) with U (uracil)
>>> print('int:',1//5, '\tfloat:',1/5)
int: 0          float: 0.2
```

Literatura

- The Python Tutorial (https://docs.python.org/3/tutorial/)
- The Python Standard Library (https://docs.python.org/3/library/)
- The Python HOWTO (https://docs.python.org/3/howto/index.html), w szczególności: Regular Expression, Sorting, argparse, Fetch Internet Resources with urllib a dla bardziej zaawansowanych Functional Programming
- Jupyter Notebook Tutorial (https://www.dataquest.io/blog/jupyter-notebook-tutorial/)