Zofia Dusińska
Katarzyna Krawczyk

Sprawozdanie 3

Translacja sekwencji aminokwasowej do sekwencji nukleotydowej

Opis programu:

Głównym celem programu jest uzyskanie sekwencji aminokwasowej z wprowadzonej przez użytkownika sekwencji nukleotydowej. Wprowadzenie sekwencji nukleotydów rozpoczynających się od kodonu START i kończących się kodonem STOP jest kluczowe. Brak tych kodonów lub ich nieprawidłowe umiejscowienie (zmiana miejsca kodonu START z kodonem STOP) spowoduje wyświetlenie komunikatu o błędzie. Należy również zauważyć, że sekwencja musi być podzielna przez trzy, co oznacza, że musi składać się z kompletnych kodonów. W przypadku nieprawidłowej sekwencji, np. z powodu mutacji lub błędów użytkownika, program poinformuje, że sekwencja jest nieprawidłowa.

Założenia:

- Akceptowane są tylko zasady azotowe (a, c, u, g);
- Sekwencje odczytywane są w kierunku kodon START do STOP;
- Jeśli jakieś kodony występują przed kodonem START lub po kodonie STOP, Zostaną one pominięte podczas translacji (chyba, że znajdą się między inną parą START - STOP);
- Sekwencje w plikach nie mogą zawierać kodonu STOP na początku ani w jej środku;
- Akceptowane są wyłącznie sekwencje złożone z trójek nukleotydów (kodonów);
- W sekwencji wprowadzanej przez użytkownika kodony nie mogą być oddzielone spacjami.

Opis ideowy opracowanego schematu rozwiązania:

Ideą naszego projektu jest stworzenie programu, który pozwoli użytkownikowi na przekształcenie sekwencji nukleotydów na sekwencję aminokwasów. Pamiętając, że kod genetyczny jest zdegenerowany - kilka różnych kodonów koduje ten sam aminokwas. Ma on wymagać od użytkownika jedynie wprowadzenia sekwencji nukleotydowej zgodnie z zasadami kodu genetycznego (rozpoczynać się od kodonu

START i kończyć się kodonem STOP, w przypadku sekwencji wprowadzanej za pomocą pliku .fasta kodony START i STOP nie są wskazane). Zgodnie z założeniami projektu mamy użyć interfejsu graficznego, dlatego ważne jest, aby był on czytelny i zrozumiały dla użytkownika. W razie wpisania błędnej sekwencji użytkownik zostanie o tym poinformowany w oknie wynikowym o nazwie: **Translation Result**.

Opis przebiegu translacji:

Translacja to proces, w którym informacja genetyczna zawarta w sekwencji nukleotydów mRNA jest używana do syntezowania łańcucha polipeptydowego (polipeptydu) składającego się z aminokwasów. Translacja zachodzi na rybosomach, strukturach komórkowych, które składają się z rRNA (rybosomalnego RNA) i białek rybosomalnych. Kod genetyczny zawarty w sekwencji nukleotydów mRNA jest odczytywany przez tRNA (transportujący RNA), które dostarczają odpowiednie aminokwasy do rybosomu, gdzie następuje synteza białkowa.

W przypadku RNA występują cztery zasady azotowe: adenina (A), uracyl (U), cytozyna (C) i guanina (G). Rozpoczęcie translacji związane jest z rozpoznaniem kodonu startowego (sekwencja trzech nukleotydów - najczęstszym jest AUG) przez rybosomy. W trakcie tego procesu mała podjednostka rybosomu łączy się z mRNA, a tRNA związane z metioniną łączy się z kodonem startowym. Następnie duża podjednostka rybosomu dołącza, tworząc kompleks inicjacyjny gotowy do syntezy białka. Cały proces jest dokładnie regulowany i zaczyna się od rozpoznania kodonu startowego jako punktu inicjacji translacji. Po rozpoczęciu translacji rybosom przemieszcza się wzdłuż mRNA, odczytując kolejne kodony i łącząc odpowiednie aminokwasy dostarczane przez tRNA, co prowadzi do powstania łańcucha polipeptydowego białka.
Przykładowe kodony:

- Kodon UUU koduje aminokwas Phe (fenyloalanina)
- Kodon GAA koduje aminokwas Glu (kwas glutaminowy)
- Kodon CCU koduje aminokwas Pro (prolina)
- Kodon AGG koduje aminokwas Arg (arginina)

Zgodnie z powyższym program do sekwencji nukleotydowych przyporządkowuje odpowiednie nazwy aminokwasów, pamiętając, że każda sekwencja nukleotydowa zakończona jest jednym z trzech kodonów STOP: UAA, UAG lub UGA, której program nie wyświetla w sekwencji aminokwasowej. Jeżeli użytkownik wprowadzi kilka poprawnych sekwencji (rozpoczynających się kodonem START i zakończonych kodonem STOP), program wydrukuje wszystkie sekwencje we wprowadzonej kolejności. Będą one ponumerowane oraz podpisane tak, aby użytkownik nie miał problemu z rozpoznaniem wprowadzonej sekwencji. W przypadku kiedy między sekwencjami wystąpią kodony (bez kodonu START i STOP), nie zostaną one zsekwencjonowane, natomiast zostaną zignorowane przez program.

Demonstracja użycia programu:

Dla ułatwienia odczytania danej sekwencji, kodony zostały oddzielone spacją, w programie prosimy o wpisywanie sekwencji bez spacji.

Użytkownik może wprowadzić:

- Jedną sekwencję rozpoczynającą się od AUG, czyli metioniny, po której będą znajdowały się kodony ACU CCU UGU UAA. Program wydrukuje sekwencję aminokwasów w kolejności: metionina, tyrozyna, prolina, cysteina. Musimy pamiętać, że kodon STOP nie zostanie wydrukowany w przeciwieństwie do kodonu START- metioniny.
- AUG GCU UUU UAG AUG AGA GGU AGU UGA jest to sekwencja prowadząca do powstania dwóch oddzielnych białek jednego zbudowanego z sekwencji aminokwasów: metionina, alanina i fenyloalanina oraz drugiego: metioniny, arginina, glicyna i seryna.
- AUG GCU AUU UAA AUG, czyli metioninę (kodon START), alaninę, izoleucynę (kodon STOP nie zostaje wydrukowany), a drugi kodon START nie zostanie odczytany, gdyż nowa sekwencja nie jest zakończona kodonem STOP.
- AUG AUG UGA zostaną wydrukowane dwie metioniny, ale w przypadku gdy na końcu zabraknie kodonu STOP (np. UGA), wydrukowany zostanie komunikat o nieprawidłowej sekwencji.
- AUG GCU UGA UGG zostanie wydrukowana sekwencja Met-Ala, program zignoruje drugi kodon STOP.
- AUU UAU UGU UAA w kolejności: izoleucyna, tyrozyna, cysteina oraz kodon STOP. Brakuje metioniny czyli kodonu START, więc nie powstanie sekwencja aminokwasowa, program wydrukuje błąd.

Przepływ danych w programie:

KLASA GuiInterface

Na samym początku program importuje moduł customtkinter jako ctk, klasę odpowiadającą za translację z pliku gui_project_translation_class.py, klasę odpowiedzialną za utworzenie okna wynikowego z pliku gui_project_second_gui.py oraz klasę SaveLoadClear z pliku gui_project_slc.py odpowiedzialną za zapisywanie, wczytywanie i czyszczenie sekwencji. Pierwszym elementem uruchamianym w naszym głównym pliku jest warunek if __name__ == "__main__". Sprawdza on, czy skrypt jest uruchamiany bezpośrednio, a nie importowany jako moduł w innym skrypcie. Spełnienie tego warunku oznacza, że skrypt jest uruchamiany jako główny program. Następnie tworzona jest instancja klasy GuiInterface(), która reprezentuje interfejs użytkownika. Za pomocą metody mainloop(), inicjalizowany jest główny cykl zdarzeń GUI, umożliwiający interakcję użytkownika z interfejsem oraz obsługę zdarzeń i akcji.

1. Klasą naszego głównego okna w programie jest klasa GuiInterface, która dziedziczy od klasy CTk z modułu customtkinter. Dzięki temu nie trzeba tworzyć obiektu okna (np. window = CTk()), gdyż konstruktor naszej klasy wywołuje konstruktora klasy nadrzędnej, co automatycznie inicjalizuje obiekt okna przy tworzeniu instancji klasy GuiInterface. W konstruktorze tej klasy poprzez zastosowanie “self” najpierw ustawiane są podstawowe właściwości okna przy wykorzystaniu odpowiednich metod z modułu customtkinter. Minimalna wielkość okna poprzez metodę self.minsize(ustala szerokość na 900 pikseli i wysokość okna na 500 pikseli). Tytuł jak i ikona okna zostają ustawione przez metodę self.title(“**Translation App** ”) oraz self.iconbitmap, która ustawia ikonę okna na “icon.ico”. Wykorzystałyśmy również metodę self.set_appearance_mode(“dark”), co pozwoliło nam na uzyskanie wyglądu z ciemnym motywem.
2. Następnie utworzone zostają atrybuty instancji toplevel_window o wartości początkowej None oraz self.translation_class również z wartością None, które będą wykorzystane później w programie.
3. Wszystkie obiekty interfejsu graficznego użytkownika (GUI), np. przyciski, etykiety, czy pola tekstowe, zostały przypisane jako potomek okna głównego, co oznacza, że jest utworzony wewnątrz tego okna (master=self).
4. Jako pierwszy widget został stworzony obiekt etykiety z parametrami odpowiadającymi za: ustawienie tekstu wyświetlanego na etykiecie ("Enter sequence below”), ustawienie czcionki oraz dobranie koloru tekstu. Następnie podane są parametry rozmieszczenia etykiety w oknie z ustawionymi marginesami (padx/pady). Kolejnym stworzonym obiektem jest pole tekstowe, które służy jako miejsce do wprowadzenia tekstu przez użytkownika (sekwencji nukleotydowej) oraz jest ustawiana jego wielkość i rozmieszczenie w oknie GUI, a także czcionka tekstu. W przypadku rozmieszczenia zastosowałyśmy również opcje konfiguracyjne jako argumenty do metody rozmieszczenia (.pack()), które sprawiają, że dany obiekt rozszerza się, aby wypełnić dostępne miejsce wewnątrz okna oraz rozwija w obu kierunkach (oś pozioma i pionowa).
5. Następnie zostaje utworzony atrybut instancji self.slc, który przechowuje instancję klasy SaveLoadClear, przekazując jej argument, będący polem tekstowym. W konstruktorze tej klasy zostaje utworzony obiekt będący polem tekstowym oraz pusta lista wykorzystana w metodzie wczytywania plików. Obiekt konstruktora klasy GuiInterface (self.slc) zostanie wykorzystany w przyciskach **Load** oraz **Clear**.
6. Później zostaje utworzonych 6 przycisków:
    - ➔ do translacji (przycisk **Translation**),
    - ➔ do załadowania pliku przez użytkownika (przycisk **Load**),
    - ➔ do utworzenia losowej sekwencji (przycisk **Random**),
    - ➔ do sprawdzenia powtarzających się sekwencji aminokwasowych (przycisk **Check** w klasie CheckClass - widoczny w oknie **Aminoacid checking**),

- ➔ do wyczyszczenia pola sekwencji (przycisk **Clear**),
- ➔ przycisk do zapisywania (przycisk **Save** w klasie ToplevelWindow - widoczny w oknie **Translation Result**),

7. Wygląd wszystkich przycisków został zaprojektowany tak samo, jednakże w przypadku przycisku do translacji zmieniono jego szerokość i ustawiono nad pozostałymi. W przypadku pozostałych przycisków są one układane od lewej strony (pod przyciskiem Translation) z ustawionymi takimi samymi marginesami. Dodano również opcję ich rozszerzania, kiedy zmieniane są rozmiary okna. Ponadto dodano zmianę koloru po najechaniu myszką na dany przycisk oraz po jego kliknięciu. Jedynymi zauważalnymi różnicami w nich to wyświetlana nazwa oraz wykorzystane komendy.

PRZYCISK TRANSLATION:

1. Po naciśnięciu przycisku Translation zostaje uruchomiona metoda open_window, której przekazywany jest argument “result”. W zależności czy zmienna toplevel_window jest pusta lub okno “poboczne” (toplevel) nie istnieje oraz czy argument (which_widnow) jest argumentem “result”, tworzy nowe okno poprzez przypisanie klasy ToplevelWindow do powyższej zmiennej.
2. Dana klasa (ToplevelWindow) dziedziczy od klasy CTkToplevel z modułu customtkinter (tak samo jak w przypadku konstruktora w klasie GuiInterface - opisany punkt 1 w akapicie KLASA GuiInterface). W danej klasie w konstruktorze zostają ustawione właściwości okna, np. wielkość okna z zablokowaną możliwością zmiany jego szerokości/wysokości, ustawienie ikony okna w przypadku systemu windows i tytuł okna oraz atrybuty: etykieta, pole tekstowe (z zablokowaną możliwością wpisywania), przycisk zapisywania. Wygląd danych atrybutów został zaprojektowany bardzo podobnie jak w oknie głównym. Również wykorzystano zaimportowaną klasę SaveLoadClear, którą przypisano do zmiennej self.slc, zastosowanej w przycisku Save (opisane w akapicie przycisk Save).
3. W klasie głównego okna po utworzeniu pobocznego okna wykorzystujemy metodę grab_set (z customtkintera) do zablokowania głównego okna, gdy poboczne jest aktywne. Na końcu uruchamiana jest metoda show_result, której przekazywanym argumentem jest metoda get_text.
4. W metodzie get_text zmiennej translation_class zostaje przypisana klasa TranslationClass, której przekazywany jest argument - sekwencja nukleotydowa wpisana przez użytkownika w pole tekstowe. W konstruktorze tej klasy TranslationClass zostają utworzone obiekty: słownik aminokwasów, zbiór zasad azotowych, przekazana sekwencja z pola tekstowego, lista kodonów z metody generate_codon_list oraz zmienna (result_check) z przypisaną wartością None (wykorzystana później w kodzie). Metoda generate_codon_list zwraca listę, która została utworzona przez

wykorzystanie list comprehension. Generuje ona listę kodonów, dzieląc sekwencję na trójki - zaczyna od pierwszego znaku, aż do końca sekwencji, jeśli przekazana sekwencja nie jest pusta.

5. Po utworzeniu wszystkich obiektów w klasie TranslationClass, wywołana zostaje metoda translation na instancji self.translation_class (w klasie GuiInterface), którą metoda get_text zwraca.
    - **a)** Na początku sprawdzane jest czy metody check_sequence i check_codons są prawdziwe. Jeśli tak to możliwe jest przejście do następnej części kodu. W przeciwnym wypadku zwracany jest wynik o błędnej sekwencji.
    - **b)** Metoda check_sequence zwraca wartość logiczną True/False poprzez sprawdzenie, czy litery w sekwencji należą do zbioru zasad azotowych oraz czy długość sekwencji jest podzielna przez trzy bez reszty (oznacza to, że sekwencja wprowadzona składa się z kodonów). Jeżeli któreś z tych założeń nie jest spełnione funkcja all zwraca False, a tym samym w oknie GUI pojawia się komunikat o nieprawidłowościach w podanej sekwencji.
    - **c)** Metoda check_codons zwraca wartość logiczną True/False poprzez sprawdzenie, czy podana sekwencja zaczyna się kodonem START i czy jest zakończona jednym z trzech kodonów stop: "UAA", "UAG", "UGA". Jeżeli tak nie jest, to funkcja all zwraca wynik False, a tym samym użytkownikowi wyświetli się komunikat o nieprawidłowościach w sekwencji.
    - **d)** W przypadku kiedy obie powyższe metody zwróciły wynik True, zostają utworzone 3 zmienne: result jako pusta lista, current_sequence jako pusty string oraz codon_start_found z wartością logiczną False.
    - **e)** Następnie zachodzi iteracja danych przez pętlę for, po czym sprawdzane jest czy kodon z listy kodonów jest równy kodonowi START (AUG). Jeśli tak to zmiennej codon_start_found przypisuje się wartość logiczną True. Dodatkowo do zmiennej current_sequence dodawana jest wartość klucza dla kodonu START ze słownika aminokwasów, czyli wartość Met.
    - **f)** Jeśli pierwszy warunek nie jest spełniony, to dane przechodzą do drugiego, który sprawdza, czy wartość klucza dla danego kodonu ze słownika aminokwasów jest równa “Stp” (sprawdzanie czy dany kodon jest kodonem STOP) oraz czy kodon START został odnaleziony (wartość logiczna True zmiennej codon_start_found). Jeśli oba warunki są spełnione, to do listy result zostaje dodana cała utworzona sekwencja aminokwasów (current_sequence) bez ostatniego elementu, którym jest myślnik. Tej zmiennej (current_sequence) również przypisuje się ponownie pusty string, co będzie wykorzystane do szukania nowych sekwencji białek oraz zmiennej codon_start_found przypisuje się wartość logiczną False.

g) Jednakże w przypadku gdy oba powyższe warunki nie zostały spełnione, to do current_sequence zostaje dodana wartość klucza danego kodonu ze słownika aminokwasów.

h) Na samym końcu danej metody (translation) zmiennej result_check przypisuje się wartość, która przedstawi osobne sekwencje aminokwasowe w każdej nowej linijce. Do tego wykorzystano funkcję enumerate, która iteruje po elementach listy result i zwraca indeks (zaczynając od 1) oraz wartość każdego elementu. Następnie poprzez zastosowanie list comprehension zostaje utworzona lista napisów: Sekwencja aminokwasowa (indeks): (sekwencja aminokwasowa). Ostatnią część kodu stanowi funkcja join z wykorzystaniem separatora "\n", dzięki czemu elementy najpierw zostają połączone w jedną całość, a przez separator, elementy zostaną oddzielone znakiem nowej linii. W skrócie: każda sekwencja znajdzie się w osobnej linijce.

i) Dana metoda zwraca utworzoną zmienną result_check wraz z podpisaną sekwencją nukleotydową wprowadzoną przez użytkownika.

6. Zwrócona wartość z metody get_text zostaje przekazana do metody show_result klasy ToplevelWindow. Dana metoda wstawia przekazany wynik do pola tekstowego, dzięki czemu użytkownik widzi wynik translacji danej sekwencji. Dodatkowo pole tekstowe zostaje zablokowane, przez co użytkownik nie ma możliwości ingerencji w otrzymany wynik.
7. Wróciwszy do naszego początkowego warunku (czy zmienna toplevel_window jest pusta albo okno "poboczne" nie istnieje), jeśli nie jest on prawdziwy, to na istniejące okno "poboczne" zostaje ustawiony focus.

PRZYCISK LOAD

1. Po naciśnięciu przycisku Load zostaje uruchomiona metoda load_file klasy SaveLoadClear (przypisana do zmiennej self.slc).
2. W wyniku uruchomienia danej metody najpierw zostaje wykorzystana inna metoda tej klasy, czyli clear_message, dzięki której usuwana jest zawartość pola tekstowego.
3. Poprzez zastosowanie funkcji askopenfilename() z modułu filedialog wraz z operatorem walrus (:=), możliwe jest jednoczesne przypisanie ścieżki pliku do zmiennej filename oraz sprawdzenie, czy użytkownik wybrał plik przy wykorzystaniu okna dialogowego, które umożliwia otwieranie plików.
4. Następnie otwierany jest wybrany plik w trybie do czytania, po czym do zmiennej lines przypisywana jest zawartość danego pliku w postaci listy, gdzie każdy element odpowiada linijce w pliku. W dalszej kolejności wykorzystano list comprehension, dzięki czemu zawartość lines zmniejsza się. Zmienna zawiera jedynie linijki pliku, w których nie ma znaku ">" (znak ten obecny jest w przypadku plików FASTA).
5. Następnie zachodzi iteracja po elementach listy, gdzie brane pod uwagę są jedynie elementy niebędące pustym stringiem. W dalszej kolejności

sprawdzane jest, czy w danym elemencie pierwszą trójką nie jest kodon START oraz czy ostatnią nie jest jeden z kodonów STOP. W przypadku prawdziwości danego warunku, do zmiennej sequence (pusta lista utworzona w konstruktorze klasy SaveLoadClear) zostaje dodany element (sekwencja) z dołączonym kodonem START na początku oraz STOP (“UAA”) na końcu. Jeśli sprawdzany element posiada już dane kodony, to zostaje od razu dodany do listy sequence.

6. Na samym końcu do pola tekstowego (pole tekstowe przekazane jako argument do konstruktora - opisane przy self.slc w KLASA GuiInterface pkt 5.) wklejana jest dana lista. Poprzez zastosowanie funkcji .join z separatorem “”, wszystkie elementy zostają połączone w jedną całość, co umożliwia ich przyszłą translacje.

PRZYCISK RANDOM

1. Po naciśnięciu przycisku Random wywołana zostaje metoda random_seq z klasy GuiInterface, w której zmiennej translation_class zostaje przypisana klasa TranslationClass z argumentem None. W rezultacie nie powstanie w tej klasie lista kodonów, gdyż nie będzie przez nas tym razem wykorzystywana. Następnie dana metoda zwraca wynik w postaci wklejonego tekstu do pola tekstowego, który otrzymał z metody random_sequence z klasy TranslationClass.
2. Wywołanie tej metody (random_sequence) powoduje powstanie sekwencji zaczynającej się od kodonu START (“AUG”), po czym przy wykorzystaniu modułu random i funkcji choices, losowo wybierane są kodony ze słownika aminokwasy (z wykluczeniem kodonów STOP i START) w ilości od 3 do 10, co zostało określone przez funkcję randint z tego samego modułu. Pod koniec dodawany jest jeden z kodonów STOP, który również jest wybierany w sposób losowy ze słownika aminokwasów. Pod koniec wszystkie kodony zostają połączone w całość poprzez zastosowanie .join() oraz separatora “” (pusty ciąg znaków), a uzyskany wynik (losowa sekwencja kodonów) zostaje przekazany do pola tekstowego, który jest widoczny dla użytkownika.

PRZYCISK CHECK

1. Na samym początku użytkownik powinien wpisać sekwencję nukleotydową w pole tekstowe, gdyż będzie ona wykorzystana do porównania z wpisaną (przez użytkownika) w następnym oknie (po kliknięciu przycisku Check) sekwencją aminokwasową.
2. Po naciśnięciu przycisku Check, zostaje wywołana metoda open_window z przekazanym argumentem “check” (sytuacja bardzo podobna tylko z innym argumentem - opisane w przycisku Translation pkt. 1.).
3. Po sprawdzeniu czy dany argument jest argumentem “check”, wywołana zostanie metoda get_text z klasy GuiInterface (opisana w przycisk Translation pkt. 4 i 5). Wywołujemy tę metodę, aby uzyskać wynik translacji przekazanej

do zmiennej result_check, którą przekazujemy do klasy CheckClass (self.translation_class.result_check).

4. W konstruktorze klasy CheckClass zostają ustawione właściwości okna (tytuł, ikona oraz minimalna wielkość) i atrybuty (etykieta, górne pole tekstowe, przycisk Compare, zablokowane dolne pole tekstowe - w nim będzie wyświetlany wynik). Wszystkie atrybuty zostały stworzone podobnie jak w klasie GuiInterface. Ponadto w tym konstruktorze utworzona zostaje zmienna, której przekazywana jest sekwencja aminokwasowa uzyskana po translacji (result_check).
5. Aby sprawdzić, czy wprowadzona sekwencja aminokwasowa (okno **Aminoacid checking**) znajduje się we wprowadzonej sekwencji nukleotydowej po translacji (główne okno), należy nacisnąć przycisk Compare. Zostanie to opisane w akapicie przycisk compare.
6. Po utworzeniu instancji klasy CheckClass (przypisanej do zmiennej toplevel_window), wykorzystywana jest funkcja grab_set do zablokowania okna głównego, póki “poboczne” nie zostanie zamknięte.

PRZYCISK COMPARE

1. Po naciśnięciu przycisku Compare zostaje wywołana metoda sequence_comparison z klasy CheckClass.
2. Na samym początku do zmiennej amino_acids_seq przypisywana zostaje wartość (sekwencja aminokwasowa) wprowadzona przez użytkownika w górnym polu tekstowym. Następnie pole tekstowe dolne (tzw. wynikowe) zostaje odblokowane i usunięta zostaje z niego zawartość.
3. W celu ułatwienia zrozumienia kodu sekwencja wprowadzona przez użytkownika w oknie głównym, po translacji (opisane poniżej) będzie nazywana **sekwencją aminokwasową**. Sekwencja wprowadzona w oknie z klasy CheckClass, którą będziemy sprawdzać czy występuje w sekwencji aminokwasowej - **sekwencją białkową**.
4. W dalszej kolejności przechodzimy do instrukcji warunkowej sprawdzającej czy sekwencja aminokwasowa nie jest pusta, czy długość podanej sekwencji białkowej jest większa od 0 oraz czy sekwencja białkowa występuje w sekwencji aminokwasowej. Jeśli wszystkie warunki zostały spełnione, to do pola wynikowego zostaje wklejona zarówno sekwencja aminokwasowa z głównego okna, jak i sekwencja białkowa.
    a) Utworzona zostaje zmienna start_index z wartością “1.0”, która zostanie wykorzystana w pętli while.
    b) Następnie rozpoczyna się pętla while, której warunkiem zakończenia będzie brak kolejnych wystąpień sekwencji białkowej (wprowadzonej przez użytkownika).
    c) W pętli while wyszukiwane są kolejne wystąpienia sekwencji białkowej w sekwencji aminokwasowej. Do zmiennej start_index zostaje

przypisany indeks pierwszego znalezionego wystąpienia sekwencji białkowej.

d) Następnie warunek sprawdza, czy powyższa zmienna jest pusta, co oznacza brak kolejnych wystąpień sekwencji/powtórzeń. Jeśli jest spełniony, to pętla while zostaje przerwana.

e) W dalszej kolejności zostaje utworzona zmienna end_index, która jest przesunięciem od początkowego indeksu (start_index) o długość sekwencji białkowej, wraz z literą “c”, oznaczającą odniesienie do kolumny.

f) Następnie wykorzystywany jest tag “highlight” w obszarze tekstu od start_index do end_index, wskazując, że ta część tekstu zostanie wyróżniona (zmieniony zostanie kolor czcionki). Wartość start_index zostaje zaktualizowana na end_index, aby wyszukać kolejne powtórzenia sekwencji białkowej w sekwencji aminokwasowej.

g) Po wyjściu z pętli while, czyli po odnalezieniu wszystkich powtórzeń, następuje zmiana koloru czcionki wyróżnionych (powtórzonych) sekwencji białkowych w sekwencji aminokwasowej na kolor “deeppink” (oznaczona również zostaje sama sekwencja wpisana przez użytkownika). Na sam koniec pole wynikowe zostaje zablokowane w celu uniemożliwienia ingerencji użytkownika w uzyskany wynik.

5. Jeśli pierwszy warunek nie będzie spełniony (pkt. 4), sprawdzane są następujące warunki:

a) Czy sekwencja aminokwasowa nie jest pusta oraz czy długość sekwencji białkowej jest większa od 0. Jeśli dane warunki są prawdziwe, to do pola wynikowego zostanie wklejony określony komunikat. W innym przypadku - przejście do następnego warunku.

b) Czy sekwencja jest pusta - jeśli tak, to w polu wynikowym wyświetla się komunikat o zamknięciu danego okna i wpisaniu poprawnej sekwencji w głównym oknie. W innym przypadku nastąpi przejście do ostatniego warunku.

c) Jeśli żadne z powyższych warunków nie zostało spełnione, to wyświetli się komunikat, że sekwencja białkowa nie została wprowadzona przez użytkownika oraz żeby ją wpisać w powyższym polu tekstowym.

PRZYCISK CLEAR

1. Po naciśnięciu przycisku Clear zostaje wywołana metoda clear_message z klasy SaveLoadClear, co zostało opisane w przypadku przycisku Load (przycisk Load- pkt. 2).

PRZYCISK SAVE

1. Przycisk Save znajduje się w oknie wynikowym (“**Translation Results**” - TopLevelWindow). Po naciśnięciu danego przycisku, zostaje uruchomiona metoda save_file z klasy SaveLoadClear, której przekazywanym argumentem jest pole tekstowe.
2. Konstruktor klasy TopLevelWindow został opisany w akapicie Przycisk Translation pkt. 2. Z kolei konstruktor klasy SaveLoadClear został opisany w akapicie Klasa GuiInterface pkt. 5.
3. Metoda save_file wykorzystuje okno dialogowe do wyboru lokalizacji i nazwy pliku do zapisania. Ścieżka i nazwa tego pliku zostają przypisane do zmiennej filename (wykorzystanie operatora walrus :=). Jako domyślne rozszerzenie pliku, ustawione zostało na .txt, jednakże można zmienić je na .fasta (do plików FASTA).
4. W następnej kolejności dany plik jest otwierany w trybie do zapisywania plików (tryb “w”) i zapisuje zawartość z pola tekstowego.