Sprawozdanie

Zadanie 3: Wyszukiwanie kliki w grafie stworzonym z podciągów sekwencji nukleotydowych z wprowadzonymi motywami

I. Opis algorytmu

a) Wczytanie instancji

Do wczytania instancji, czyli zarówno pliku *.fasta*, jak i *.qual*, zawierających kolejno losowo wybrane sekwencje nukleotydowe wraz z wprowadzonymi motywami w pliku *.fasta* oraz ocenami wiarygodności tych nukleotydów w pliku *.qual*, służą dwie funkcje, do których przekazywane są odpowiednie wektory będące buforami pomocniczymi. Do wczytania pojedynczej sekwencji potrzebne jest jednorazowe wywołanie funkcji, która wypełnia bufor pomocniczy odczytanymi z pliku nukleotydami, a zmiennej typu *string*, która do funkcji przekazana jest poprzez referencję, przypisywany jest identyfikator sekwencji. Aby ułatwić wprowadzanie sekwencji do wektora, w pliku *.fasta* na końcu linii z sekwencją dodałam znak “<”. W ten sposób funkcja przepisuje nukleotydy z pliku aż do napotkania znaku “<” lub do momentu, w którym plik się skończy. Efektem jednorazowego wywołania funkcji jest wypełnienie jednego wektora sekwencją oraz zapisanie identyfikatora sekwencji w odpowiedniej zmiennej.
Przy wczytywaniu pliku *.qual* wywoływana zostaje kolejna funkcja, w której również wykorzystany został wektor. Tak samo jak w poprzedniej funkcji jednorazowe jej wywołanie zapewnia wypełnienie wektora wartościami wiarygodności dla poszczególnych nukleotydów jednej sekwencji. Przy wczytywaniu kolejnych liczb oznaczających oceny wiarygodności wykorzystany został rozmiar wektora przechowującego odpowiednią sekwencję nukleotydową. W ten sposób nie było już konieczności wprowadzania żadnych zmian w pliku *.qual*, które ułatwiłyby wczytywanie ocen jakości.

b) Usuwanie nukleotydów poniżej pewnego progu wiarygodności

W tym etapie wykorzystywana jest kolejna funkcja, która dla wprowadzonego wektora z sekwencją oraz odpowiadającemu tej sekwencji wektora przechowującego oceny wiarygodności nukleotydów, wyszukuje w pętli *for* wszystkie liczby mniejsze od progu wprowadzonego przez użytkownika. Jako że indeks każdego nukleotydu w wektorze z sekwencją pokrywa się z indeksem oceny jakości tego nukleotydu w wektorze z ocenami, to indeksy te wykorzystane są w usuwaniu nukleotydów będących poniżej progu ustalonego przez użytkownika. Przy każdej iteracji pętli sprawdzane są kolejne oceny wiarygodności i gdy zostanie znaleziona liczba poniżej danego progu, to indeks obecnie sprawdzanej pozycji w wektorze z ocenami wykorzystywany jest do przypisania wartości “0” na tej samej pozycji w wektorze z sekwencjami. Po sprawdzeniu wszystkich ocen wykonywana jest kolejna pętla, która ma za zadanie wypełnienie dwóch pozostałych wektorów przekazywanych do funkcji. W każdej iteracji pętli sprawdzany jest nukleotyd z sekwencji z już usuniętymi pozycjami. Gdy pod obecnie sprawdzanym indeksem w wektorze znajduje się znak inny niż “0”, to do jednego wektora (odpowiadającemu już “ostatecznej” sekwencji) zapisywany jest nukleotyd, a do drugiego (odpowiadającemu oryginalnym pozycjom nieusuniętych nukleotydów w sekwencji) zapisywany jest numer tego indeksu zwiększony o jeden. W ten sposób po wywołaniu funkcji otrzymano dwa wektory, które będą wykorzystywane w kolejnych krokach. Wektory z początkową sekwencją oraz ocenami jej wiarygodności zostają wyczyszczone w celu zwolnienia pamięci, gdyż nie będą już więcej potrzebne.

c) Utworzenie podciągów będących wierzchołkami w grafie

Do tworzenia oraz przechowywania podciągów, będących wierzchołkami w grafie, wraz z informacjami dotyczącymi pozycji nukleotydu w sekwencji początkowej, od którego dany podciąg się zaczyna, oraz identyfikatora sekwencji, z której ten podciąg pochodzi wykorzystywana jest klasa *Vertice* oraz zawarty w niej konstruktor, a dokładniej jego lista inicjalizacyjna. Aby ułatwić operacje na wierzchołkach, które będą wykonywane w kolejnych krokach algorytmu, oraz by nie wykorzystywać w nich informacji o sekwencjach nukleotydowych, do klasy dodana została również zmienna oznaczająca numer danego wierzchołka. W ten sposób każdy podciąg utworzony przez program będzie obiektem klasy *Vertice* o unikalnym indeksie.
Do utworzenia konkretnych podciągów wykorzystywana jest odpowiednia

funkcja, do której przekazywany jest m.in. wektor. Jego zadaniem będzie przechowywać wskaźniki na obiekty, które powstaną po wyszukaniu każdego podciągu w danej sekwencji. By wyszukać podciągi najpierw zostaje określone, do którego indeksu wektora ma wykonywać się pętla *for* (jest to wartość rozmiaru całego wektora z sekwencją, od której została odjęta długość podciągu). Następnie w tej pętli dodaję kolejne zmienne: *j*, która na początku przyjmuje wartość obecnie iterowanego indeksu wektora, oraz *count*, która zostanie wykorzystana jako licznik. W kolejnej pętli do zmiennej pomocniczej typu *string* dodawane są kolejne nukleotydy, które znajdują się pod indeksem *j* w wektorze z sekwencją. Z każdym wykonaniem tej pętli zmienne *j* oraz *count* zwiększają się o jeden. Pętla wykonuje się do momentu, w którym *count* osiągnie wartość równą długości podciągu określonej przez użytkownika. Gdy to się stanie, pobierana jest pozycja nukleotydu, od którego zaczął się podciąg, a następnie zmienne przechowujące informacje, które są niezbędne do utworzenia obiektu klasy *Vertice* za pomocą listy inicjalizacyjnej. Po utworzeniu obiektu wskaźnik na niego wskazujący zostaje umieszczony w specjalnym buforze. Zmienna *count* przyjmuje wartość 0 i następuje kolejna iteracja pętli *for*. W ten sposób na podstawie podanej do funkcji sekwencji zostaną utworzone wszystkie podciągi o konkretnej długości.
Zarówno poprzednie, jak i ta funkcja działa tylko na jednej sekwencji, dlatego potrzebne jest wywołanie jej tyle razy, ile sekwencji chcemy wykorzystać do stworzenia podciągów.

d) Połączenie odpowiednich wierzchołków krawędziami

Biorąc pod uwagę sposób, w jaki w moim algorytmie wyszukiwana jest klika, zdecydowałam się na reprezentację grafu w postaci listy sąsiedztwa przechowywanej w wektorze wektorów. W tym i w następnym kroku wykorzystywane będą unikalne indeksy, które zostały przypisane wierzchołkom (podciągom) podczas ich tworzenia. Do tworzenia krawędzi wykorzystywana jest pętla, która iteruje po każdym wskaźniku obecnym w wektorze wskaźników odpowiadającym konkretnej sekwencji, i która ostatecznie po każdej iteracji utworzy listę sąsiedztwa dla danego wierzchołka. W pętli najpierw do wektora pomocniczego dodany zostaje unikalny indeks wierzchołka, dla którego tworzona jest lista. Następnie wywoływana jest funkcja, do której przekazywane są m.in. indeks obecnie iterowanego wskaźnika oraz dwa wektory ze wskaźnikami na obiekty: jeden wektor będący referencją do obecnie iterowanego wektora wskaźników, a drugi będący referencją do wektora wskaźników należącego do innej sekwencji. W funkcji za pomocą pętli oraz wskaźników klasy *Vertice*

porównywane są nazwy poszczególnych obiektów. W przypadku, gdy porównywane nazwy są takie same, do pomocniczego bufora dodany zostaje indeks porównywanego wierzchołka z innej sekwencji. Gdy zostaną porównane wszystkie podciągi utworzone z drugiej sekwencji, funkcja kończy działanie. W pętli funkcja wywoływana jest cztery raz tak, by utworzyć pełną listę wierzchołków, z którymi połączony jest obecnie sprawdzany wierzchołek. Na koniec każdej iteracji bufor pomocniczy dodawany jest do wektora wektorów reprezentującego graf. Przed kolejną iteracją bufor jest czyszczony. Pętle dla pozostałych sekwencji zostały utworzone w sposób analogiczny. Następnie sprawdzane jest, czy graf zawiera jakąkolwiek listę sąsiedztwa o rozmiarze większym niż cztery. Jest to bowiem warunek konieczny do wyszukania (i w ogóle istnienia) kliki, które zostało opisane w kolejnym kroku.

e) Wyszukanie w grafie w sposób heurystyczny kliki

Reprezentację kliki w swoim grafie potraktowałam w sposób następujący: klikę tworzyć będzie taki podzbiór pięciu wierzchołków, który znajduje się w każdej kolumnie w wektorze wektorów *Graph* reprezentującej listę sąsiedztwa każdego wierzchołka znajdującego się w takim podzbiorze. Aby lepiej wyjaśnić to podejście do reprezentacji kliki posłużę się konkretnym przykładem.
Podzbiór wierzchołków składa się z następujących elementów: *{1, 2, 3, 4, 5}*. By taki podzbiór można było potraktować jako klikę należy sprawdzić w grafie kolumnę odpowiadającą kolejno wierzchołkowi o indeksie *1, 2, 3, 4* oraz *5*. Jeżeli w przypadku każdego wierzchołka w jego kolumnie znajdzie się taki sam podzbiór (przy czym bierze się pod uwagę całą kolumnę, a zatem zarówno indeks wierzchołka, jak i jego listę następników), to dany podzbiór tworzy klikę.
Przy opracowywaniu takiego podejścia i ogółem całej heurystyki posłużyłam się zasadą tworzenia krawędzi, która została zawarta w zadaniu, gdyż wiadomo, że dany wierzchołek w swojej liście sąsiedztwa nie będzie posiadał wierzchołka, który odpowiada podciągowi pochodzącemu z tej samej sekwencji. Zatem wszystkie wierzchołki będące w tej liście sąsiedztwa to wierzchołki pochodzące z innych sekwencji niż wierzchołek, do którego listy sąsiedztwa należą.

Biorąc pod uwagę tę własność funkcja, która wyszukuje klikę, szuka takiego podzbioru w każdej kolejnej kolumnie wektora wektorów *Graph*, i jest umieszczona w pętli *for*, z której pobiera iterowaną zmienną. Zmienna ta służy jako odniesienie do numeru indeksu odpowiadającemu obecnie

sprawdzanej kolumnie w *Graph*, przy czym sprawdzane są tylko kolumny, które mają przynajmniej pięć elementów. Gdy taki warunek jest spełniony, to do wektora pomocniczego *buffer* dodawany jest numer wierzchołka *V*, dla którego jest obecnie sprawdzana lista sąsiedztwa, a do *bufferInx* zostaje dodany indeks, pod którym ten wierzchołek znajduje się w *Graph*. Następnie z listy sąsiedztwa *V* sprawdzane są po kolei wierzchołki oraz ich listy sąsiedztwa. W pierwszej pętli *for* pobierany jest numer kolejnego wierzchołka *V1* z listy sąsiedztwa *V*, a w zagnieżdżonej w niej drugiej pętli *for* przechodzi się po odpowiednich komórkach w *Graph*, szukając numeru wierzchołka *V1*. Jeżeli odpowiadająca mu kolumna ma więcej niż cztery elementy, to cała ta kolumna zostaje porównywana z kolumną odpowiadającą wierzchołkowi *V*. Przy każdym takim samym znalezionym elemencie zmienna *count* rosła o jeden. Po porównaniu całych kolumn sprawdzana jest wartość zmiennej *count* - jeżeli wynosi ona przynajmniej 5, to do wektora *buffer* zostaje dodany numer *V1*, a do wektora *bufferInx* zostaje dodany indeks *V1* odpowiadający indeksowi tego wierzchołka w *Graph*. Następnie zmienna *count* ulega wyzerowaniu i ma miejsce kolejna iteracja. Po sprawdzeniu każdej kolumny w *Graph* dla każdego wierzchołka z listy sąsiedztwa *V* następuje sprawdzenie rozmiaru zmiennej *buffer* - gdy jest on równy co najmniej pięć, to wykonywany jest kolejny krok, a gdy nie, to funkcja kończy działanie.
W wektorze *buffer* może znajdować się pięć wierzchołków tworzących klikę, dlatego kolejnym krokiem będzie porównanie całej jego zawartości z listami sąsiedztw wierzchołków, które znajdują się w tym wektorze. Przy wykorzystaniu pętli *for* każdy element *buffer* porównywany jest z każdym elementem znajdującym się w odpowiedniej kolumnie w *Graph.* W przypadku, gdy porównywane elementy będą sobie równe, zmienna *count* rośnie. Po sprawdzeniu całej kolumny odpowiadającej konkretnemu wierzchołkowi *V1* następuje sprawdzenie, czy zmienna count równa się rozmiarowi całego *buffer* - oznacza to bowiem, że wierzchołek *V1* jest jedynym wierzchołkiem w *buffer*, który reprezentuje daną sekwencję. Zatem każdy taki wierzchołek dodawany jest do wektora *clique*, ponieważ jeżeli w *buffer* istnieje podzbiór pięciu wierzchołków tworzących klikę, to wierzchołek *V1* na pewno do takiego podzbioru należy. W przypadku jednak, gdy wartość *count* jest mniejsza od rozmiaru *buffer*, to do kolejnego wektora pomocniczego *buffer2* dodawany jest numer tego wierzchołka, a do *buffer2Inx* dodawany jest jego indeks z *Graph*. *Buffer2* reprezentuje wierzchołki, które nie pochodzą z różnych sekwencji, jednak niekoniecznie wszystkie wierzchołki tam dodane będą pochodzić z jednej sekwencji. Zależy to od rozmiaru, jaki będzie miał wektor *clique* po wykonaniu już

wszystkich pętli *for* i porównaniu każdej kolumny z wektorem *buffer*. Aby mieć jak największa pewność, że alogrytm znajdzie jakąkolwiek klikę rozważyłam pięć następujących przypadków:

1) Wektor *clique* zawiera pięć elementów

W takim przypadku nie istnieje konieczność rozważania żadnego z elementów *buffer2*, ponieważ taki wektor jest pusty. Dzieje się tak dlatego, że w poprzednim kroku algorytm sprawdził, że zawartość *buffer* tworzy pięć wierzchołków pochodzących z różnych sekwencji i nie dodał żadnego elementu do *buffer2.* Klika zostaje odnaleziona, a funkcja kończy swoje działanie.

2) Wektor *clique* zawiera cztery elementy

Oznacza to, że w *buffer2* istnieją tylko wierzchołki pochodzące z tej samej sekwencji. Zatem do wektora *clique* dodawany jest pierwszy element z *buffer2*, a klika zostaje znaleziona. Funkcja kończy swoje działanie.

3) Wektor *clique* zawiera trzy elementy

Oznacza to, że w *buffer2* znajdują się wierzchołki pochodzące z dwóch różnych sekwencji. Aby oddzielić, który wierzchołek pochodzi z której sekwencji najpierw do *clique* dodawany jest pierwszy element *V1* z *buffer2*, a następnie na podobnej zasadzie, co w poprzednim kroku sprawdzane są lista sąsiedztwa *V1* z *buffer2* z listami sąsiedztwa kolejnych elementów z tego wektora. Jeżeli *V1* nie znajduje się na liście sąsiedztwa obecnie sprawdzanego wierzchołka *V2*, to znaczy, że *V1* i *V2* pochodzą z tej samej sekwencji i wierzchołek *V2* nie może zostać dodany do *clique*. W przypadku, gdy wierzchołek *V3* posiada wierzchołek *V1* w swojej liście sąsiedztwa, to oznacza to, że *V1* i *V3* pochodzą z różnych sekwencji, dlatego *V3* zostaje dodany do *clique.* Wtedy klika zostaje znaleziona, a funkcja kończy swoje działanie.

4) Wektor *clique* zawiera dwa elementy

Działanie w takim przypadku jest zbliżone do działania w przypadku, gdy wektor *clique* zawiera trzy elementy, jednak dodawany jest jeszcze jeden wektor pomocniczy *buffer3*, do którego dodawane są wszystkie wierzchołki, które zawierają w swoich listach sąsiedztwa pierwszy element *V1* z *buffer2*. Po wypełnieniu *buffer3* wszystkie kroki z przypadku wyżej powtarzane są w sposób analogiczny. Klika zostaje znaleziona, gdy do wektora *clique* zostanie dodany taki wierzchołek *V4*, który będzie miał w swojej liście sąsiedztwa pierwszy element *V3* z *buffer3*.

5) Wektor *clique* zawiera mniej niż dwa elementy

Z takiego wektora nie uda się utworzyć kliki, więc funkcja wypisuje odpowiedni komunikat i kończy swoje działanie.

Gdy zostanie znaleziona jakakolwiek klika, funkcja kończy działanie, a pętla, w której została umieszczona, zostaje przerwana. W przypadku, gdy klika nie zostanie w ogóle znaleziona, pętla również kończy działanie i zostaje wypisany odpowiedni komunikat.

f) Wypisanie rezultatu

Jeżeli została znaleziona jakakolwiek klika, program wypisze o niej odpowiednie informacje: podciąg, a także identyfikator sekwencji oraz pozycje nukleotydów, od których zaczyna się znaleziony podciąg w konkretnej sekwencji. Informacje te wyszukiwane są wśród list wskaźników za pomocą indeksów wierzchołków znajdujących się w wektorze reprezentującym znalezioną klikę, oraz wskaźników klasy *Vertice*.

II. Oszacowanie złożoności algorytmu

Przyjmując, że m odpowiada największej długości sekwencji, a n jest długością podciągów sekwencji oraz biorąc pod uwagę fakt, że w swoim algorytmie korzystałam głównie z pętli, nierzadko w sobie zagnieżdżonych, to przy operacjach na całych sekwencjach największa złożoność może wynieść m^3 (ze względu na trzy zagnieżdżone w sobie pętle). W przypadku operacji na podciągach złożoność ta będzie większa, głównie ze względu na proces znajdowania kliki. Nadal jednak będzie to mała złożoność, ponieważ oszacowałam ją na n^4 (ze względu na to, że największa liczba zagnieżdżonych w sobie pętli wynosi cztery). Z tego więc powodu oszacowałam złożoność tego algorytmu na $O(n^4)$. Jest to zatem złożoność wielomianowa.

III. Wybrane instancje oraz wyniki testów

Instancja 1:

plik *.fasta*

```
>DOJHLOP02G54IK      length=108      xy=2823_0202      region=2
run=R_2005_09_08_15_35_38_
TGAGTGATCTTACAGAATTTAATCCAAAACGGCTGTTAGATACCATCATCTTAACTGCTAGA
AACAGGCCTTTCTGACCTAATTGGCGGCCAATCACTCTCCAAAAGG<
>DOJHLOP01B6G9Y      length=98      xy=0776_3172      region=1
run=R_2005_09_08_15_35_38_
TTCAGGTAAAAGTTGATCTAATCCTGGGACACCACTCCAAAGGAAACAAGGCAATACAGCTA
TTGAGGACATCTGTTAGATACCATCCTTAGAAACCG<
>DOJHLOP02HM4YB      length=66      xy=3016_3393      region=2
run=R_2005_09_08_15_35_38_
CAGTAGCGCCAACAACAGCTACTTTTGATTTACTCAAATCAATTCCTCTGTTAGATGGCATT
TGCT<
>DOJHLOP01E2OD5      length=124      xy=1963_2667      region=1
run=R_2005_09_08_15_35_38_
TGTATTCCAGGTCCTGAAAACTTCTCACCCAAGTTCTGTTAGATACCGTCATCAATCATTAA
ACAAGGTCTTTTAATTTTAGATGGGTCAATTTCTAAGGTTTTAGGCATGCTTGCAACCCAAA
<
>DOJHLOP01DNKEA      length=128      xy=1381_1904      region=1
run=R_2005_09_08_15_35_38_
ATAACGTTGGAAAAGATGGATTTCATCGGCAAGGCTTCTGAAAGGCACGGGTTTTCGGCTGT
TGGTTTGAAATCAAATATTCACTGTTAGATACCACGAACTATTTTGGTTTACTTTTTTTTTT
ATGG
```

plik *.qual*

```
>DOJHLOP02G54IK      length=108      xy=2823_0202      region=2
run=R_2005_09_08_15_35_38_
32 32 25 31 30 32 27 32 32 30 26 27 31 32 28 31 27 28 27 14 31
27 30 31 26 25 25 22 13 31 31 27 32 25 26 28 31 31 32 26 30 27
29 30 29 30 32 32 31 31 31 27 31 27 31 29 29 32 31 19 27 28 27
18 29 27 31 26 31 27 27 27 18 32 32 26 32 28 21 28 31 27 27 21
31 27 32 21 12 23 14 29 27 27 30 32 32 27 28 22 27 21 26 25 22
11 28 22
>DOJHLOP01B6G9Y      length=98      xy=0776_3172      region=1
run=R_2005_09_08_15_35_38_
31 28 30 32 30 26 30 26 25 22 11 31 30 27 32 32 31 28 29 31 28
30 31 28 26 27 27 13 22 31 29 30 28 29 31 31 31 28 29 28 18 28
26 26 25 9 30 31 28 27 20 31 31 27 29 27 31 32 23 30 29 26 28
22 29 16 20 10 26 11 32 30 32 26 25 30 30 29 25 31 32 27 29 25
29 28 30 15 31 27 10 26 23 22 2 30 27 27
```

>DOJHLOP02HM4YB length=66 xy=3016_3393 region=2 run=R_2005_09_08_15_35_38_

32 30 32 32 29 32 32 32 31 27 31 27 30 31 27 32 25 32 32 28 31 31 25 25 22 12 30 32 29 28 17 32 31 32 32 28 28 16 31 29 31 27 31 27 31 27 29 26 30 29 27 24 29 25 27 30 26 32 32 29 27 22 13 31 30 31

>DOJHLOP01E2OD5 length=124 xy=1963_2667 region=1 run=R_2005_09_08_15_35_38_

32 31 32 31 31 28 31 28 28 31 28 31 31 28 28 30 24 24 19 3 31 31 28 32 31 29 30 28 27 14 31 28 29 22 22 25 31 27 32 29 29 31 32 25 31 31 26 31 26 27 31 29 32 31 26 32 32 27 29 24 27 26 13 30 25 19 25 19 24 30 25 25 22 11 30 25 25 25 21 9 32 31 27 29 29 28 18 32 30 30 25 29 28 18 31 17 31 28 26 24 24 23 21 12 31 31 27 29 31 31 31 31 29 24 32 17 28 26 28 27 15 29 28 18

>DOJHLOP01DNKEA length=128 xy=1381_1904 region=1 run=R_2005_09_08_15_35_38_

32 31 30 27 31 32 31 28 31 27 25 24 20 5 29 24 31 29 24 24 29 28 16 28 31 29 28 30 26 32 23 15 29 24 31 30 27 32 28 31 23 22 3 30 25 31 24 29 24 23 6 25 25 20 6 31 27 20 31 25 29 28 26 28 23 24 23 7 27 26 26 11 32 31 28 27 14 18 31 30 25 31 28 26 31 31 30 27 29 30 27 31 26 29 29 25 31 32 30 26 31 11 31 24 24 19 5 30 26 29 28 18 18 29 7 7 6 6 6 5 4 3 2 1 31 29 30 27

Wyniki testów na tej instancji:

1)

```
Podciag: CATC

ID sekwencji: DOJHLOP02G54IK    Pozycja: 44
ID sekwencji: DOJHLOP01B6G9Y    Pozycja: 52
ID sekwencji: DOJHLOP02HM4YB    Pozycja: 40
ID sekwencji: DOJHLOP01E2OD5    Pozycja: 53
ID sekwencji: DOJHLOP01DNKEA    Pozycja: 24
```

Próg wiarygodności: 28
Długość podciągu: 4

2)

```
Podciag: CTGTTAGAT

ID sekwencji: DOJHLOP02G54IK    Pozycja: 33
ID sekwencji: DOJHLOP01B6G9Y    Pozycja: 73
ID sekwencji: DOJHLOP02HM4YB    Pozycja: 48
ID sekwencji: DOJHLOP01E2OD5    Pozycja: 36
ID sekwencji: DOJHLOP01DNKEA    Pozycja: 84
```

Próg wiarygodności: 24
Długość podciągu: 9

3)

```
Podciag: TAGAT

ID sekwencji: DOJHLOP02G54IK    Pozycja: 37
ID sekwencji: DOJHLOP01B6G9Y    Pozycja: 77
ID sekwencji: DOJHLOP02HM4YB    Pozycja: 51
ID sekwencji: DOJHLOP01E2OD5    Pozycja: 40
ID sekwencji: DOJHLOP01DNKEA    Pozycja: 88
```

Próg wiarygodności: 25
Długość podciągu: 5

Instancja 2
plik *.fasta*

```
>DOJHLOP01DWJWJ      length=100      xy=1483_3377      region=1
run=R_2005_09_08_15_35_38_
TCGATTCTATGGAGGAGTAGCCTGCGACTGGCGGAAGCAGCATCAGCAATTAAAAAATTACT
GGACCTGATCTTATGAAGTTAGGATTGTTGACGAGGTA<
>DOJHLOP01ES3NP      length=96      xy=1854_2355      region=1
run=R_2005_09_08_15_35_38_
TTGGTATGATAATTCTCGAATTCAAAGTAGCCTGCGACTTAAAAACAAGCAAACATTTCTCT
TTTAGGATTTTCCATCTCAGCCAATTCCATCATG<
>DOJHLOP01ELHRI      length=99      xy=1767_3740      region=1
run=R_2005_09_08_15_35_38_
AGAGGCTTTCAAAGTAAGTAGCCTGCAGACTAGTCTCCTAATTCAATGCAATCTTGATTGAA
TCTTTTGCAATATCCGCATATCACGTTTATATACGTG<
>DOJHLOP01BV6VM      length=110      xy=0659_2368      region=1
run=R_2005_09_08_15_35_38_
AGCTGTATTGCCTTGTTTCCTTTGGAGTGGTGTCCCAGGATTAGATCAACTTTTACCTGAAA
AAGAGGTTGAGGAAGTAGCCTGCGACTGGTTAGCGGAAACAAGTCTTA<
>DOJHLOP01DM14Z      length=76      xy=1375_2817      region=1
run=R_2005_09_08_15_35_38_
CTCATGGTGGGCAAATTGGTCTGCTAGTAGCCTGCGTCAGGCAGTTGAGAATTCAATGTCCT
TCAATAGCTGTATT
```

plik *.qual*

```
>DOJHLOP01DWJWJ      length=100      xy=1483_3377      region=1
run=R_2005_09_08_15_35_38_
32 31 29 26 30 27 32 32 31 30 28 26 26 28 27 25 26 31 25 32 31
30 25 25 29 25 31 28 28 31 31 27 29 26 22 14 32 31 25 29 31 31
29 32 28 31 31 29 23 24 22 19 19 18 16 10 3 31 28 32 29 32 31
28 23 30 28 31 31 26 24 31 31 28 30 24 27 24 17 29 28 23 31 20
11 16 31 28 31 21 12 30 29 31 26 31 29 24 31 17
```

```
>DOJHLOP01ES3NP       length=96       xy=1854_2355       region=1
run=R_2005_09_08_15_35_38_
31 28 31 28 32 32 32 32 32 32 30 27 31 28 31 31 31 31 29 24 31
28 31 31 27 31 30 26 32 32 32 30 27 31 29 31 28 32 25 31 30 23
23 21 12 30 31 28 31 32 28 27 15 32 32 28 28 19 32 22 26 25 25
22 11 32 30 25 29 25 25 22 11 31 28 31 30 31 32 31 28 32 31 28
29 27 27 25 30 28 32 32 32 32 31 30
>DOJHLOP01ELHRI       length=99       xy=1767_3740       region=1
run=R_2005_09_08_15_35_38_
30 32 29 31 27 30 24 23 7 32 29 28 16 24 30 27 27 20 27 31 27
30 29 31 32 29 28 32 31 28 25 31 31 27 31 31 28 22 18 31 27 28
23 32 31 28 26 29 31 19 8 24 32 29 23 31 21 24 17 27 20 10 16
29 26 25 22 10 32 28 31 26 29 28 26 31 28 31 26 23 32 28 28 32
29 28 32 28 28 16 17 30 32 24 31 30 24 30 28
>DOJHLOP01BV6VM       length=110       xy=0659_2368       region=1
run=R_2005_09_08_15_35_38_
32 29 28 31 32 32 31 31 27 32 29 24 31 27 31 29 28 17 30 26 28
27 15 31 28 30 31 32 31 27 31 31 31 29 28 18 32 31 28 28 31 27
32 31 29 30 29 31 28 30 25 25 22 12 28 30 25 24 31 22 22 20 13
2 29 23 31 27 30 26 28 29 27 25 32 29 28 26 25 32 31 28 31 31
28 29 31 29 25 26 32 31 28 30 26 32 30 26 26 25 11 28 26 19 31
32 28 31 27 32
>DOJHLOP01DM14Z       length=76       xy=1375_2817       region=1
run=R_2005_09_08_15_35_38_
32 32 29 32 32 31 27 31 28 28 15 31 29 28 16 31 28 31 27 30 31
29 32 31 27 31 27 30 31 28 32 31 27 28 28 26 31 31 27 31 28 30
24 31 31 28 29 31 30 30 25 31 27 29 30 25 32 32 28 31 26 30 27
28 29 24 29 31 31 28 30 32 27 32 30 26
```

Wyniki testów na tej instancji:

1)

```
Podciag: AGTAGCCTG

ID sekwencji: DOJHLOP01DWJWJ    Pozycja: 16
ID sekwencji: DOJHLOP01ES3NP    Pozycja: 26
ID sekwencji: DOJHLOP01ELHRI    Pozycja: 17
ID sekwencji: DOJHLOP01BV6VM    Pozycja: 76
ID sekwencji: DOJHLOP01DM14Z    Pozycja: 26
```

Próg wiarygodności: 20

Długość podciągu: 9

2)

```
Podciag: GCCTGC

ID sekwencji: DOJHLOP01DWJWJ    Pozycja: 64
ID sekwencji: DOJHLOP01ES3NP    Pozycja: 30
ID sekwencji: DOJHLOP01ELHRI    Pozycja: 21
ID sekwencji: DOJHLOP01BV6VM    Pozycja: 80
ID sekwencji: DOJHLOP01DM14Z    Pozycja: 30
```

Próg wiarygodności: 27
Długość podciągu: 6

3)

```
Podciag: TAGCC

ID sekwencji: DOJHLOP01DWJWJ    Pozycja: 18
ID sekwencji: DOJHLOP01ES3NP    Pozycja: 28
ID sekwencji: DOJHLOP01ELHRI    Pozycja: 19
ID sekwencji: DOJHLOP01BV6VM    Pozycja: 78
ID sekwencji: DOJHLOP01DM14Z    Pozycja: 28
```

Próg wiarygodności: 25
Długość podciągu: 5

Instancja 3
plik *.fasta*

```
>DOJHLOP02GK3KF      length=100      xy=2583_2237      region=2
run=R_2005_09_08_15_35_38_
TGGGGATAGGCGTCGCAGACGAGTCTATATTGTTTGAACATAGTGTTTACACAGTTGCAAGC
CCTGAAGCTTGTGCTTCGATTCTATGGAGGGATGCTGG<
>DOJHLOP01ANFAH      length=102      xy=0149_2711      region=1
run=R_2005_09_08_15_35_38_
ATTACATACAGATGTTTTGTGATGATTGGAGTCTATAATGTGGAGACAGGAACGGAACTGAT
GATCAAGCTCTAATAGGGAGGGTTAGCTCGAATAGGGTAG<
>DOJHLOP01CPWLL      length=122      xy=0998_0183      region=1
run=R_2005_09_08_15_35_38_
CAGGTCCTGAAACTTCTCACCCAAGTTTTTAGGGTATCCACCATCAATCATTAAACAAGGTC
TTTTAATTTTAGATGGGTCAATTTCTAAGGAGTCTATGTTGCTTGCAACCCAAATAACAA<
>DOJHLOP01BGUHQ      length=120      xy=0484_3276      region=1
run=R_2005_09_08_15_35_38_
CCCAAGCTGTATGAGTATTACCAGTGGTAAACCTTTGCCAATCGAGTCTATATTGACTAACT
TGTTGATTTTGGAGCAAAATTTAAAAATTCTACAAAAATTTATTGATTGTAAAGCCCA<
>DOJHLOP02JNU60      length=109      xy=3845_1050      region=2
run=R_2005_09_08_15_35_38_
GGTTAAATTGTTGGATGCAAAAGCTTGCATAGATCTATTAGAAATTTGTCAAGAGTCTATCT
TGTACAGAATTTAATCCAAAAACGGTAGTTCAGTTACCACATCTTAA
```

plik *.qual*

```
>DOJHLOP02GK3KF      length=100      xy=2583_2237      region=2
run=R_2005_09_08_15_35_38_
31 24 24 19 6 29 30 32 31 27 28 32 32 32 32 31 32 31 29 30 28
31 27 31 27 27 29 31 27 27 32 32 27 26 12 30 22 13 25 29 31 31
28 26 27 26 25 10 29 30 27 29 31 32 31 27 32 31 31 27 30 29 28
18 31 31 30 24 31 32 29 26 32 31 29 32 31 27 32 32 31 30 27 30
32 29 25 26 24 27 29 28 17 22 31 27 31 31 29 26
>DOJHLOP01ANFAH      length=102      xy=0149_2711      region=1
run=R_2005_09_08_15_35_38_
31 30 26 27 31 31 29 26 32 31 31 31 30 32 24 24 22 12 31 25 29
27 32 32 32 31 28 28 27 27 30 27 31 31 28 28 27 27 31 31 31 29
26 29 29 31 31 30 24 23 31 28 30 29 27 31 27 32 32 27 32 31 26
32 29 32 31 28 32 24 27 31 21 29 27 32 31 23 22 2 30 28 27 19
30 28 22 32 27 31 32 24 29 27 17 23 23 22 5 31 25 32
>DOJHLOP01CPWLL      length=122      xy=0998_0183      region=1
run=R_2005_09_08_15_35_38_
29 29 22 14 31 25 18 28 31 27 27 19 32 27 20 28 24 23 27 27 26
12 29 27 31 22 22 21 15 5 31 23 22 5 31 21 24 31 26 31 28 22
28 26 31 30 26 22 30 21 31 27 25 25 18 29 31 28 27 21 25 29 26
25 22 10 31 28 26 25 22 10 32 31 26 27 24 23 7 24 29 29 23 24
23 7 29 32 31 28 27 27 26 25 28 26 31 29 25 31 31 29 31 32 25
19 32 29 30 28 27 26 12 28 27 19 26 29 26 31 28 26
>DOJHLOP01BGUHQ      length=120      xy=0484_3276      region=1
run=R_2005_09_08_15_35_38_
28 28 16 31 26 30 31 16 29 32 24 31 31 30 32 30 32 27 21 31 26
20 29 32 28 25 18 16 27 26 12 28 23 29 28 16 31 28 23 29 24 23
31 30 26 29 31 28 31 28 32 29 32 28 27 25 31 32 29 24 30 30 27
30 29 24 31 26 25 25 22 11 28 26 32 31 31 25 25 21 9 23 22 2
22 22 21 15 6 24 23 18 31 29 30 21 21 20 16 9 25 24 9 31 29 26
31 32 21 12 22 27 27 27 19 32 23 22 3 22
>DOJHLOP02JNU60      length=109      xy=3845_1050      region=2
run=R_2005_09_08_15_35_38_
27 14 29 23 28 27 15 31 27 32 31 26 31 27 26 32 32 31 24 24 20
6 28 31 30 26 31 31 32 29 32 28 23 30 29 32 28 30 25 27 30 25
24 7 28 28 16 31 30 31 30 24 31 28 32 32 26 29 30 32 31 29 32
29 26 27 20 31 31 31 27 27 27 13 31 26 18 29 26 22 22 20 14 3
23 25 23 24 22 32 31 27 29 30 32 31 27 30 30 27 29 12 25 29 32
29 24 31 27
```

Wyniki testów na tej instancji:
1)

```
Podciag: GAGTCTAT

ID sekwencji: DOJHLOP02GK3KF    Pozycja: 21
ID sekwencji: DOJHLOP01ANFAH    Pozycja: 29
ID sekwencji: DOJHLOP01CPWLL    Pozycja: 92
ID sekwencji: DOJHLOP01BGUHQ    Pozycja: 44
ID sekwencji: DOJHLOP02JNU60    Pozycja: 53
```

Próg wiarygodności: 25
Długość podciągu: 8

2)

```
Podciag: ATTG

ID sekwencji: DOJHLOP02GK3KF    Pozycja: 29
ID sekwencji: DOJHLOP01ANFAH    Pozycja: 25
ID sekwencji: DOJHLOP01CPWLL    Pozycja: 2
ID sekwencji: DOJHLOP01BGUHQ    Pozycja: 52
ID sekwencji: DOJHLOP02JNU60    Pozycja: 6
```

Próg wiarygodności: 27
Długość podciągu: 4

3)

```
Podciag: GAGTC

ID sekwencji: DOJHLOP02GK3KF    Pozycja: 21
ID sekwencji: DOJHLOP01ANFAH    Pozycja: 29
ID sekwencji: DOJHLOP01CPWLL    Pozycja: 92
ID sekwencji: DOJHLOP01BGUHQ    Pozycja: 44
ID sekwencji: DOJHLOP02JNU60    Pozycja: 53
```

Próg wiarygodności: 17
Długość podciągu: 5

IV. Wnioski

Celem zadania było m.in. sprawdzenie zachowania się programu przy różnych parametrach wpisywanych na początku jego działania. Od momentu zapoznania się z zadaniem przypuszczałam, że wyniki działania na tej samej instancji mogą się różnić w zależności od parametrów, które się wprowadzi, jednak nie myślałam, że zmiany te mogą być aż tak drastyczne. Oznacza to zatem, że wiarygodność nukleotydów pełni bardzo istotną rolę w odszukiwaniu kliki, ponieważ odfiltrowanie nawet jednego nukleotydu z

podciągu, którego wierzchołki wcześniej tworzyły klikę, może tej samej kliki nie znaleźć albo nie znaleźć żadnej kliki w ogóle. Przy testowaniu programu nawet na instancjach z już wprowadzonymi motywami parametry musiałam dobierać dość ostrożnie, by znaleźć jakąkolwiek klikę. Jednak w przypadku, gdy ta klika została już odnaleziona, prawie zawsze zawierała w sobie fragment wprowadzonego motywu. Z kolei praktycznie w ogóle nie zdarzyło się, żeby algorytm znalazł taką klikę, która nie zawierałaby się we wprowadzonym motywie. Można zatem dojść do wniosku, że parametry progu wiarygodności oraz długości podciągu mają kluczowe znaczenie w wyszukiwaniu podciągu, który utworzyłby klikę i jakiekolwiek najdrobniejsze zmiany w tych parametrach mogą albo zupełnie zmienić wynik działania programu, albo w ogóle nie dać żadnego rozwiązania.