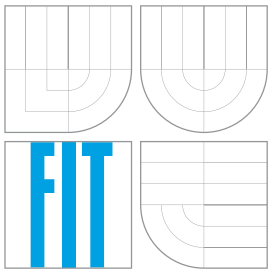
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

VYHLEDÁVÁNÍ TRIPLEXŮ V DNA SEKVENCÍCH

TRIPLEX DETECTION IN DNA SEQUENCES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MICHAL MUCHA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MARTÍNEK TOMÁŠ, Ph.D.

BRNO 2011

Abstrakt

Tento dokument popisuje návrh algoritmu pro vyhledávaní triplexů v DNA sekvencích. Zabívá se rozborem problematiky vyhledávaí a podmínkami utvoření těchto struktur. Dále popisuje implementaci navrhnutého algoritmu v jazyce Java a jeho následné testování.

Abstract

This document describes design of algorithm used for finding triplexes in DNA sequenses. Search and conditions of formations of such structures are also discussed. It is also describing implementation of designed algorithm in Java programming language and its following testing.

Klíčová slova

DNA, triplex, prohledávaná sekvence

Keywords

DNA, triplex, search sequence

Citace

Michal Mucha: Vyhledávání triplexů v DNA sekvencích, bakalářská práce, Brno, FIT VUT v Brně, 2011

Vyhledávání triplexů v DNA sekvencích

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáše Martínka, Ph.D.

.....
Michal Mucha
18. května 2011

Poděkování

Tímto bych chtěl poděkovat panu Ing. Tomášovi Martínkovi, Ph.D. za jeho odborné a pedagogické vedení při vypracovávaní této práce a za poskytnutí studijních materiálů a odkazů na literaturu, z které jsem čerpal.

Obsah

Kapitola 1

Úvod

Cieľom práce je implementovať už existujúci, alebo navrhnúť nový algoritmus pre vyhľadávanie intramolekulárnych triplexov v molekulách DNA, vo zvolenom programovacom jazyku. Program bude umožňovať užívateľovi zadať prehľadávaný úsek DNA, dĺžku prehľadávanej sekvencie, typ triplexu a ďalšie parametre vyhľadávania.

DNA je často považovaná za návod, alebo kód, pomocou ktorého je možné zostrojiť iné komponenty buniek (napr. bielkoviny a RNA molekuly). Každý bunkový organizmus má uloženú genetickú informáciu v tejto molekule.

Kanonická štruktúra DNA je dvojzávitnicová špirála, avšak v niektorých prípadoch môže vzniknúť tretie vlákno, hovoríme vtedy o triplexoch. Tak ako zmena zdrojového kódu programu môže spôsobiť zmeny v programe samotnom, tak aj tieto zmeny štruktúry môžu ovplivňovať niektoré biologické deje a z toho dôvodu je potrebné mať nástroj, ktorý dokáže tieto štruktúry identifikovať. Keďže DNA obsahuje veľké množstvo informácií (ľudský genóm obsahuje približne 3,3 miliárd bázových párov[3]), je potrebné aby algoritmus, použitý na vyhľadávanie triplexov bol časovo a pamäťovo čo najefektívnejší.

Druhá kapitola sa zaoberá všeobecným rozborom problematiky DNA. Popisuje ďalej problematiku vzniku nekanonických štruktúr a taktiež obsahuje zoznam databáz, ktoré obsahujú úseky DNA. Tretia kapitola je venovaná rozboru problematiky vyhľadávania triplexov, pre konkrétne prípady, ktoré môžu vzniknúť. Táto kapitola obsahuje aj popis komplementárnych prvkov, kotré sa nachádzajú v triplexových štruktúrach. Štvrtá kapitola je venovaná návrhu riešenia a zdôvodnenia použitých metód. V piatej kapitole je popísaná implementácia zvoleného algoritmu, zvolený programovací jazyk a použité dátové štruktúry. V šiestej kapitole je popísané testovanie a zhodnotenie výsledkov testov. Siedma kapitola je venovaná rozboru možných rozšírení aplikácie v budúcnosti.

Kapitola 2

Rozbor problematiky

Táto kapitola obsahuje základný popis molekuly DNA a jej vlastností. Ďalej sú tu popísané možné zmeny základnej štruktúry molekuly. Problematika triplexov je popísaná na konci kapitoly.

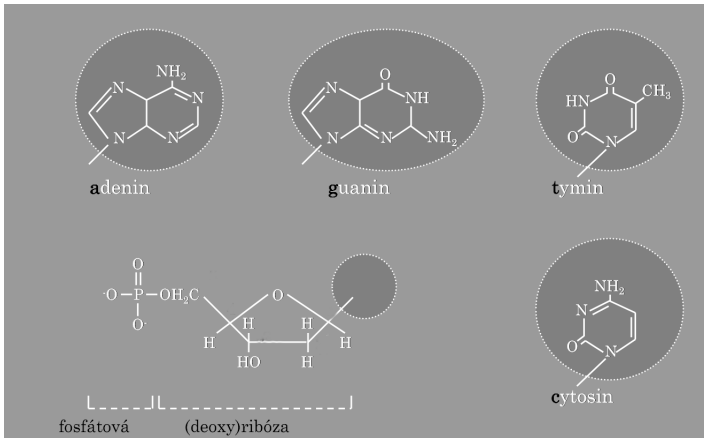
2.1 Štruktúra DNA

Deoxyribonukleová kyselina(DNA) je biopolymér zložený zo sekvencie štyroch rôznych nukleotidov vo forme dvojzávitnicovej špirály. [9] Informácie uchované v DNA umožňujú vytvorenie organizmu so špecifickými vlastnosťami. Tieto vlastnosti sú dané štruktúrou samotnej molekuly. Vďaka informáciam v DNA je každý organizmus jedinečný. Nukleotidy obsahnuté v DNA sú zložené z týchto častí (viď obr.2.1):

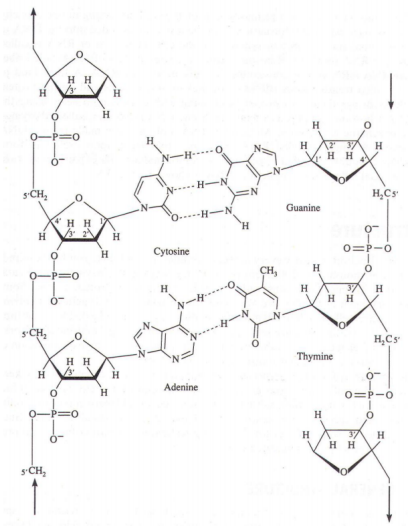
- Deoxyribóza(monosacharid odvodený od sacharidu ribóza)
- Fosfátová skupina
- Nukleová báza

Nukleotidy delíme podľa bázy na puríny a pyrimidíny. Nukleotidy s purínovou bázou obsiahnuté v DNA sú Adenín(A) a Guanín(G). Nukleotidy s pyrimidínovou sú Tymín(T) a Cytozín(C). Na jednom vlákne sú nukleotidy prepojené fosfodiesterovou väzbou, ktorá vzniká prepojením fosfátovej skupiny jedného nukleotidu s deoxyribózovým cukrom druhého[3]. Esterové väzby zahŕňajú v sebe atóm kyslíka, pričom fosfodiesterová vazva má dve takéto väzby na oboch stranách fosforu. Dlhé reťazce nukleotidov sú nazývané polynukleotidmi. Dlhé polynukleotidy potom tvoria chromozómy. Medzi dvojicou nukleotidov na dvoch vláknach DNA vznikajú vodíkové väzby, nazývané Watson-Crick-ové väzby podľa ich objaviteľov James D. Watson a Francis Crick. Túto dvojicu potom nazývame *bázový pár*. Podľa kanonického, Watson- Crick formovania tvorí adenín bázový pár s tymínom a guanín s cytozínom. Medzi guanínom a cytozínom vznikajú tri vodíkové väzby a medzi tymínom a adenínom vznikjú dve väzby(znázornené na obrázku 2.2)

Na základe tejto vlastnosti vieme určiť komplementaritu vláken molekuly a tým pádom každé vlákno obsahuje kompletnú genetickú informáciu o danom organizme. Nukleotidy sú radené za sebou a spojenie medzi nimi je zaručené fosfátovými zvyškami. Primárna štruktúra je daná poradím nukleotidov[3].



Obrázek 2.1: Štruktúra nukleotidov [3]



Obrázek 2.2: Párovanie báz DNA [3]

2.2 Zápis DNA

DNA sa dá zapísať ako poradie nukleotidov, alebo rad písmen, označujúcih tieto nukleotidy. Napriek tomu, že vlákna DNA sú komplementárne niesú rovnako orientované, sú antiparalelné($5'$ koniec jedného vlákna odpovená $3'$ koncu druhého vlákna) – ide o reverzný komplement.

Príklad:

1. $5'$ – GTATCC – $3'$
2. $3'$ – CATAGG – $5'$

Smer vláken DNA označujeme podľa orientácie deoxyribózy, podľa konvencie sa vlákna zapisujú v smere $5' \rightarrow 3'$[3].

2.3 Databázy DNA

Pre uchovávanie sekvencií existuje pomerne veľké množstvo databáz. Tieto databázy budú hlavným zdrojom dát používaných pri testovaní výlsedného programu. V tejto práci uvádzam prehľad tých najzákladnejších :

- *Swis-Prot protein knowledge-base* – databáza poskytujúca sekvencie proteínov a ich anotácie
- *Nucleeotide Sequence Base* – European Bioinformatics Institute
- *The EMBL Nucleotide Sequence Database* – Primárny európsky zdroj sekvencií DNA a RNA
- *DNA Bank of Japan*
- *GenBank* – voľne prístupná databáza sekvencií
- *UCSC Genome browser*

2.4 Nekanonická štruktúra

Primárna štruktúra DNA môže byť pozmenená na základe lokálnej sekvencie, teploty, pH, a topológie. Po prvý krát bola nekanonická štruktúra objavená v roku 1962, kedy R. Sinskeimer zistil, že DNA nadobúda uzavretú cirkulárnu formu u niektorého druhu vírusu.[8] Medzi nekanonické štruktúry patrí rozvinutá DNA, hairpiny, kvadruplexy, triplexy a iné. V tejto práci sa budem zaoberať práve triplexami. Ostatné prípady nekanonickej DNA uvádzam v stručnom prehľade na obrázku2.3.

2.5 Triplexy

Triplexy boli objavené v roku 1957 vedcami G. Felsenfeld, David R. Davies a Alexander Rich[1] . Tento objav ummožnil vzniku nového prístupu k sekvenčne špecifickému rozpoznávaniu DNA a používajú sa na označovanie mutácií špecifických miest pre zvolené gény.[7]

2.5.1 Štruktúra triplexov

Triplexy sú tvorené nukleotidmi na vláknach molekuly DNA, medzi ktorými sú Watson-Crick-ové väzby(duplexy) a nukleotidom z tretieho vlákna, ktoré je prepojené Hoogsteenovými, alebo reverznými Hogsteenovými väzbami[2]. *intermolekulárne* triplexy vznikajú medzi triplexotvornýni oligonukleotidmi inej molekuly (triplex forming oligonucleotides TFO) a bázovým párom na cieľovej sekvencii DNA. Ďalším prípadom sú *intramolekulárne* triplexy, ktoré sú tvorené v homopurín-homopyridínových úsekoch DNA, keď je zatočenie molekuly dostatočne veľké[2]. Cieľom práce je zostrojiť program pre vyhľadávanie práve intramolekulárnych triplexov. intermolekulárnymi triplexmi sa ďalej zaoberať v tejto práci nebudem. Intramolekulárne triplexy sú potom rozdelené na *vnútro-vláknové* (intrastrand), kde Hogsteenové alebo reverzné Hogsteenové väzby vznikajú pri zahnutí vlákna na seba a *medzi-vláknové* (interstrand), kde tieto väzby vznikajú pri zahnutí jedného z vláken na druhé.

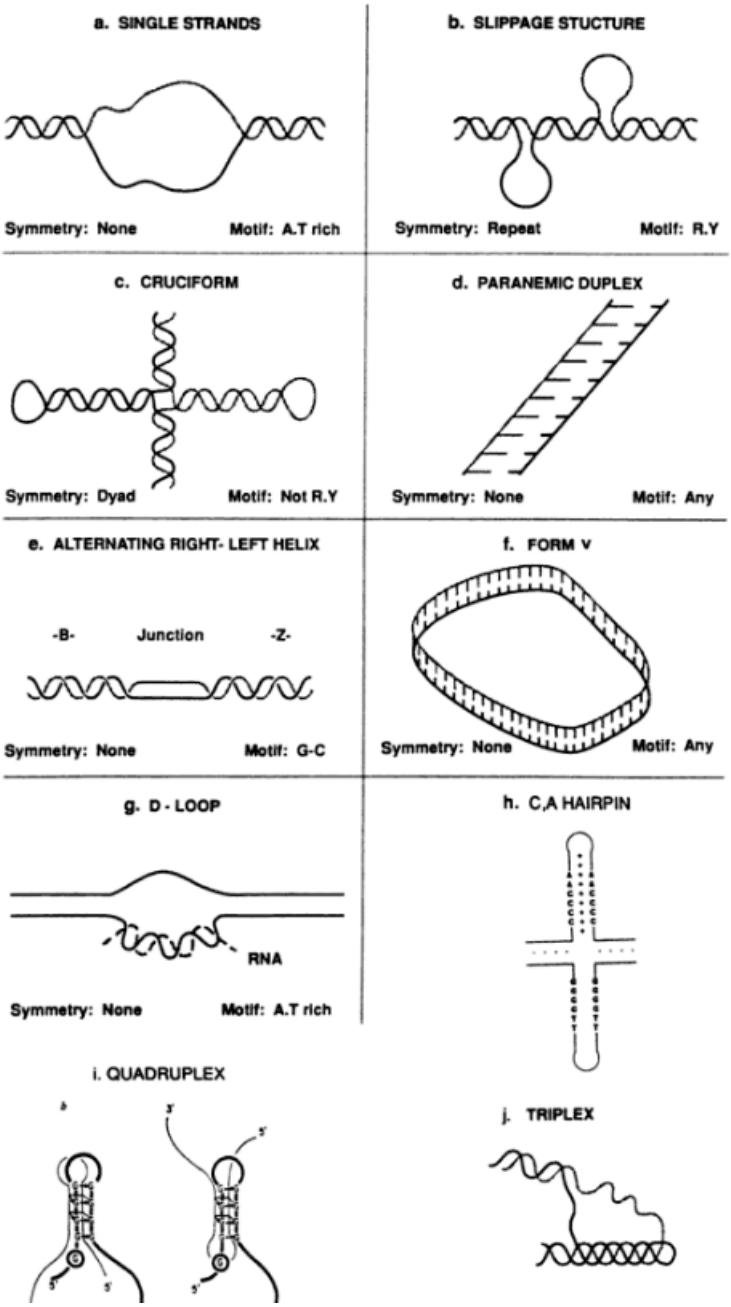
Podľa orientácie vláken sa dajú triplexy rozdeliť na paralélne alebo antiparalélne. Paralelizmus je určený podľa orientácie prostredného vlákna vzhľadom na tretie vlákno triplexu. Ak je prostredné vlákno orientované v smere napríklad $5' \rightarrow 3'$ a tretie vlákno v smere $3' \rightarrow 5'$ ide o antiparalélny triplex. Pri zhodnej orientácii ide o paralélny triplex.

2.5.2 Typy triplexov

Pri formovaní triplexov je dôležitý fakt, že puríny môžu utvoriť 2 vodíkové mostíky s tretím vláknom. Preto triplexy vznikajú na úsekoch, kde na jednom vlákne je homopurínová sekvencia a na komplementárnom vlákne tým pádom homopyridínová sekvencia[8]. Hovoríme potom o *PyPu* úseku. Pokiaľ sa takéto úseky v molekule DNA nenachádzajú vznik triplexu nie je možný.

Na duplex sa potom môže naviazať aj pyrimidín aj purín podľa toho rozlišujeme *PyPu*Py* a *PyPu*Pu* triplexy. Pre *PyPu*Py* môžu potom vznikať trojice CG*C a TA*T a pre *PyPu*Pu* trojice CG*G TA*A[2].

Pri zápise triplexov je duplex oddelený symbolom * od tretieho purínu alebo pyrimidínu. Samotné triplexy potom zapisujeme tak, že na prvé miesto uvádzame pyrimidínové vlákno, na druhé miesto purínové vlákno a na tretie miesto zapisujeme purínové alebo pyrimidínové tretie vlákno triplexu.



Obrázek 2.3: Prehľad nekanonických foriem DNA (zdroj:[8])

Kapitola 3

Problematika vyhľadávania

Táto kapitola je venovaná rozboru problematiky vyhľadávania. V prvej časti je popísané prirovnanie problému k palindrómu. Ďalej popisuje množiny vyhľadávaných prvkov a rozdelenie triplexov na kategórie podľa ktorých bude vyhľadávanie prebiehať.

3.1 Palindróm s medzerou

Keďže pri vzniku triplexov vznikajú v reťazcoch DNA zrkadlovo sa opakujúce sa reťazce nukleotidov,je problematika vyhľadávania intramolekulárnych triplexov podobná problematike vyhľadávania palindrómov s medzerou. Definícia palindrómu podľa [4]:

Definice 3.1.1. Nech $A = a_1a_2 \dots a_{m-1}a_m$ je reťazec dĺžky m, $B = b_1b_2 \dots b_{g-1}b_g$ je reťazec dĺžky g pre ktorý platí, $b_1 \neq b_g$, potom

- Palindróm s medzerou je reťazec v tvare

$$P_g = ABA^R = a_1 \dots a_{m-1}a_m b_1 b_2 \dots b_{g-1}b_g a_m a_{m-1} \dots a_2 a_1$$

- Medzerou nazývame nesymetrickú časť $B = b_1b_2 \dots b_{m-1}b_m$, dĺžky $|B| \geq 2$

V našom prípade medzera predstavuje ohyb vlákna, ktoré tvorí triplex. Tento ohyb potom nieje vo výslednej sekvencii zahrnutý. Pre utvorenie triplexov je minilálna dĺžka ohybu jeden nukleotid, v tomto prípade by sa dal problém definovať ako nepárny palindróm avšak je to len jeden z možných prípadov a preto beriem do úvahy palindrómy s medzerou dĺžky väčšou alebo rovnou jednej[4].

Dôvodom prečo nemôžme definovať problém presne ako palindróm s medzerou sú množiny komplementárnych prvkov. Pre PyPuPy trojicu sú to TA*T a CG*C a pre PyPuPu TA*A a CG*G[5]. Taktiež sa nemusí jednať o symboly rovnakého vlákna (v niektorých prípadoch je potrebné porovnávať nukleotidy prvého s nukleotidmi druhého vlákna). Preto pre rôzne druhy triplexov vznikajú kombinácie znakov, ktoré nevyhovujú pravidlám palindrómov. Tieto pravidlá sa však dajú definovať dodatočne. Navyše, triplexy nemusia byť dokonalé a môžu obsahovať chyby ako zámena(dôjde k zmene znaku oproti očakávanej hodnote), vloženie(na určité miesto sekvencie je vložený znak, prípadne sekvencia), alebo vynechanie (očakávaný znak, prípadne sekvencia je vynechaná).

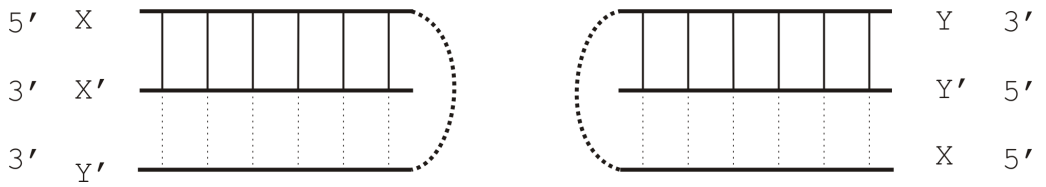
3.2 Komplemetárne prvky

Triplexy môžu vzniknúť rovnako z prvého aj druhého vlákna molekuly. Pre vyhľadávanie je dôležitá aj orientácia vlákna ktoré tentno triplex vytvára, vznikajú potom paralélne a anti-paralélne triplexy(paralelizmus je určený podľa orientácie prostredného vlákna vzhľadom na tretie vlákno triplexu). Pri anti-paralélnych triplexoch sú tvorené reverzné Hogsteenové väzby medzi nukleotidmi vlákna molekuly a nukleotidmi vzniknutého tretieho vlákna. Na základe týchto znalostí dokážeme rozlíšiť osem možností, ktorými môžu byť tieto štruktúry utovorené(viď obrázky 3.1, 3.2, 3.3, 3.4).

Pri podrobnejšom skúmaní týchto možností je možné určiť hľadané trojice nukleotidov špecifické pre jednotlivé typy. Taktiež dokážeme určiť, ktoré vlákna nás pri vyhľadávaní budú zaujímať, keďže na vstupe predpokladáme prvé vlákno molekuly. Všetky prípady majú rovnakú množinu komplementárnych prvkov, avšak na vstupe je len jedno vlákno molekuly a preto potrebujeme poznať, ktorý prípad budeme vyhľadávať.

Na zálklade nižšie uvedených možností utvorenia triplexov bude zostavený návrh algoritmu a následne bude tento návrh implementovaný. Pre vyhľadávanie sú dôležité prvky prvého vlákna a vlákna tretieho. Druhé vlákno si totiž dokážeme dopočítať na základe komplementarity vláken DNA (viď kapitola 2.1). V obrázkoch 3.1, 3.2, 3.3 a 3.4 sú vlákna reprezentované hrubými, horizontálnymi čiarami, prerušovaný úsek je ohyb vlákna netvoriaci Hogsteenové alebo reverzné Hogsteenové väzby. Tenké vertikálne čiary reprezentujú Watson-Crickové väzby a prerušované znázorňujú Hogsteenové alebo reverzné Hogsteenové väzby. Na obrázkoch znaky X a Y reprezentujú dva nukleotidy rovnakého vlákna, ktoré budú súčasťou triplexu. Apostrof označuje komplementárny nukleotid. Obrázky podľa [6]

1. Tretie vlákno vzniká zahnutím prvého vlákna na druhé, vzniká paralélny triplex. Porovnávame nuklotity prvého vlákna.



Obrázek 3.1: Paralélne triplexy tvorené prvým vláknom

Pre tento prípad môžu vznikať štruktúry:

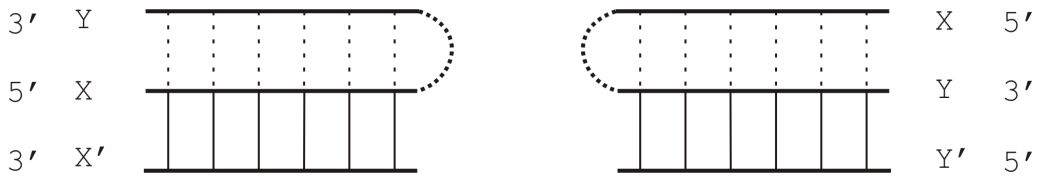
1.vlákno	Py	**T**	**C**	Py	**T**	**C**
2.vlákno	Pu	A	G	Pu	A	G
3.vlákno	Py	**T**	**C**	Pu	**A**	**G**

Vyhľadávané dvojice sú v tomto prípade

- PyPuPu = {TA, CG}[1]
- PyPuPy = {TT, CC}

[1]Na prvé miesto z dvojice uvádzam nukleotid prvého, na druhé miesto nukleotid tretieho vlákna

2. Tretie vlákno vzniká zahnutím prvého vlákna na seba, vzniká antiparalélny triplex. Porovnávame nukleotidy prvého vlákna. .



Obrázek 3.2: Antiparalélne triplexy tvorené prvým vláknom

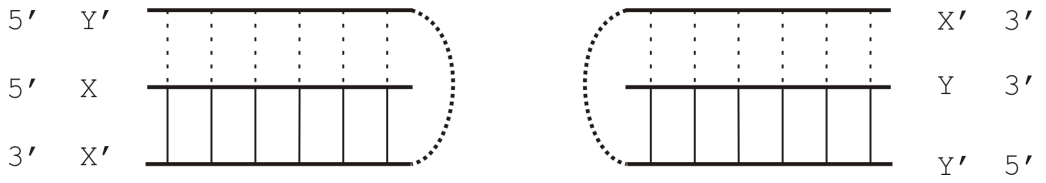
Pre tento prípad môžu vznikať štruktúry:

3.vlákno	Py	**T**	**C**	Py	**A**	**G**
1.vlákno	Pu	**A**	**G**	Pu	**A**	**G**
2.vlákno	Py	T	C	Pu	T	C

Vyhľadávané dvojice sú v tomto prípade

- PyPuPu = {AA, GG}
- PyPuPy = {AT, GC}

3. Tretie vlákno vzniká zahnutím druhého vlákna na prvé, vzniká paralélny triplex. Porovnávame nukleotidy prvého vlákna s nuklotidmi druhého vlákna.



Obrázek 3.3: Paralélne triplexy tvorené druhým vláknom

Pre tento prípad môžu vznikať štruktúry:

3.vlákno	Py	**T**	**C**	Py	**A**	**G**
1.vlákno	Pu	**A**	**G**	Pu	**A**	**G**
2.vlákno	Pu	T	C	Pu	T	C

Vyhľadávané dvojice sú v tomto prípade

- PyPuPu = {AA, GG}
- PyPuPy = {AT, GC}

4. Tretie vlákno vzniká zahnutím druhého vlákna na seba, vzniká antiparalélny triplex. Porovnávame nukleotidy prvého vlákna s nukleotidmi druhého vlákna.

Obrázek 3.4: Antiparalélne triplexy tvorené druhý vláknom

Pre tento prípad môžu vznikať štruktúry:

1.vlákno	Py	**T**	**G**	Py	**T**	**C**
2.vlákno	Pu	A	C	Pu	A	G
3.vlákno	Pu	**T**	**G**	Pu	**A**	**G**

Vyhľadávané dvojice sú v tomto prípade

- PyPuPu = {TA, CG}
- PyPuPy = {TT, CC}

Môžeme si všimnúť, že množny komplementárnych prvkov pre prvý a štvrtý prípad sú zhodné. Tak isto sú zhodné množiny prvkov druhého a tretieho prípadu. Avšak triplexy sú v týchto prípadoch tvorené nukleotidmi rôznych vláken, preto je potrebné rozdeliť to na tieto typy.

Pre každý prípad potom existujú dve možné orientácie triplexu, podľa toho, či je zahnutý $5'$, alebo $3'$ koniec vlákna. Pre týchto osem prípadov bude vytvorené vyhľadávanie.

Kapitola 4

Návrh riešenia

V tejto kapitole sú popísané návrhy riešenia a ich rozbor. V úvode rozoberá vstupné parametre. Pokračuje rozborom dvoch možných návrhov. Návrh s využitím dynamického programovania bol nakoniec zvolený pre implementáciu

Pre vyhľadávanie bude použitá upravená verzia niektorého z algoritmov pre vyhľadávanie palindrómov (úprava vzhľadom na obmedzenia, ktoré boli popísané v kapitole 3.2). Užívateľ na vstup programu zadá prehľadávanú sekvenciu a parametre vyhľadávania:

- Typ triplexu
- Maximálnu dĺžku ohybu
- Minimálnu dĺžku sekvencie triplexu
- Maximálnu dĺžku sekvencie triplexu
- Percento chýb prípustné pre sekvenciu

4.1 Dvoj-úrovňové vyhľadávanie

Triplexy sa môžu vyskytovať na pomerne krátkych(vzhľadom na celkovú dĺžku sekvencie) úsekoch sekvencie DNA. Kvôli tomu bude vyhľadávanie rozdelené na dve úrovne, pričom prvá, rýchla kontrola označí potenciálne úseky a následne podrobná kontrola zistí, či sa na danom úseku vyskytuje hľadaná štruktúra.

4.1.1 Prvá úroveň

Keďže jednou z podmienok pre tvorbu intramolekulárnych triplexov je výskyt homopurín-homopyridínových sekvencií v molekule DNA, je možné vykonať rýchlu kontrolu, ktorá vyhľadá tieto sekvencie a uloží ich pozíciu. Výsledok tejto metódy bude potom vstupom pre podrobnejšiu kontrolu.

Minimálnu dĺžku a počet chýb v sekvencii, ktorá bude akceptovaná touto metódou určí užívateľ. Táto kontrola dokáže priamo označiť, že sa v danej sekvencii triplex so zadanými parametrami nevyskytuje a tým pádom nebude potrebné vykonávať ďalšiu kontrolu.

4.1.2 Druhá úroveň

Pokiaľ prvá úroveň nevylúči výskyt triplexu je potrebné pre označené úseky prvej úrovne vykonať podrobnejšiu kontrolu. V tejto úrovni je možné použiť zložitejší a časovo náročnejší algoritmus na prehľadávanie sekvencie, keďže sa už nejedná o celú sekvenciu DNA ale, len o jej úseky.

Výstupom tejto úrovne bude potom umiestnenie nájdených štruktúr, prípadne správa o nenájdení štruktúr so zadanými parametrami.

4.2 Dynamické programovanie

Dvoj-úrovňové vyhľadávanie by bolo však pameťovo a časovo náročné, preto som sa rozhodol zvoliť iný prístup k problematike a tým je práve dynamické programovanie.

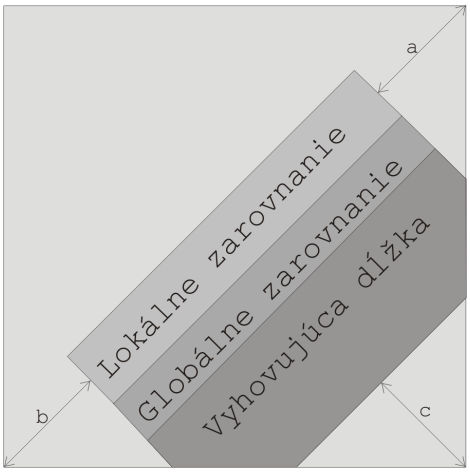
Pre každý typ triplexu je osobitne (v samostatnom behu) zostavená skórovacia matica, pričom riadky reprezentujú sekvenciu vlákna a stĺpce potom reprezentujú reverznú podobu tejto sekvencie. Táto matica je potom vyplnená smerom z dola na hor po hlavnej a vedľajšej antidiagonále, pričom hlavná antidiagonála reprezentuje prostredný bod štruktúr s nepárnym počtom nukleotidov a vedľajšia antidiagonála štruktúr s párnym počtom. Skóre je potom vypočítané pohybom po diagonálach, pričom počiatočný bod tohoto pohybu je práve uvedený bod na hlavnej a vedľajšej antidiagonále. Výsledná hodnota pre danú bunku v matici je určená porovnaním znakov na indexoch(x určuje znak sekvencie, y reverznej podoby viď. obr.4.1). Podľa skórovacích tabuliek na základe typu triplexu bude určené, či hodnota bude penalizovaná, alebo inkrementovaná.

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

Obrázek 4.1: Prehľadávanie skórovacej matice. Uvedené hodnoty sú ilustračné

Tento prístup predpokladá, že začiatok prehľadávania je v tomto prípade prostriedok triplexu, tým pádom prehľadávanie stále začína v polovici ohybu, ktorý netvorí Hogsteenové väzby. Pre tento ohyb je potrebné implementovať iný prístup hodnotenia ako pre zvyšnú časť triplexu. Pre hodnotenie ohybu bude použité lokálne zarovnanie sekvencie a tým pádom výsledné skóre nikdy neklesne pod nulu. Maximálna veľkosť ohybu je určená užívateľom. Pokiaľ bude tento limit prekročený, algoritmus predpokladá pozíciu mimo ohybu.

Zvyšok sekvencie je potom prehľadávaný globálnym zarovnaním viď.4.2. Pokiaľ skóre v tomto úseku prekročí určitú hranicu, ktorá je prepočítaná na základe aktuálnej dĺžky



Obrázek 4.2: Prehľadávanie priestoru v matici.

sekvencie a percentuálneho zastúpenia chýb, je táto sekvencia považovaná za potenciálny triplex, až kým hodnota skóre neklesne pod túto hranicu. Pokiaľ skóre klesne pod hranicu prípustnosti je posledný vyhovuvujúci výskyt uložený.

Pokiaľ je užívateľom špecifikovaná hranica minilálnej dĺžky tak nieje potrebné prehľadávať celý priestor matice. Tento priestor je obnedzený len na tie body antidiagonály u ktorých je možné týchto hodnôt dosiahnuť a zároveň pokiaľ hodnota skóre vyhovuje podmienkam avšak nedosahuje minimálnej dĺžky nieje považovaná za potenciálny výskyt. Na obrázku 4.2 je tento priestor obmedzený vzdialenosťami a a b, ktoré sú vypočítané na základe minimálnej dĺžky triplexu pred prehľadávaním priestoru. Vzdialenosť c je zvyšná, neprehľadávaná časť, ktorá závisí od celkovej dĺžky sekvencie a užívateľom obmedzenej maximálnej dĺžky triplexu. Pokiaľ aktuálny počet porovnaných nukleotidov presiahne danú hodnotu, je uložená posledná hodnota nadobudajúca dostatočné skóre. Ak nieje maximálna hodnota špecifikovaná je prehľadávaný celý priestor pod antidiagonálami, čo spôsobí porovnávanie celej vstupnej sekvencie a z pohľadu algoritmu umožňuje nález triplexu dĺžky vstupu.

4.2.1 Odhad zložitosti

Pre implementáciu skórovacej matice by bolo potrebné priestor veľkosti n^2. Matica takýchto rozmerov by bola pre reálne dáta prakticky nepoužiteľná. Avšak pre z tento priestor nieje potrebné uchovávať v pamäti a je možné výpočty prevádzať len na základe posledných dvoch antidiagonál. Preto pamäťová náročnosť je $O(2n)$.

Pre prehľadávanie celého trojuholníka pod antidiagonálami je časová náročnosť $n^2/2$ krokov. Avšak v reálnych prípadoch nieje potrebné prehľadávať celý tento priestor. Užívateľ má možnosť obmedziť maximálnu dĺžku triplexu, čo značne zníži časovú náročnosť. Na základe tohoto obmedzenia je potom náročnosť $O(2kn)$ pričom k popisuje maximálnu dĺžku akceptovaného triplexu.

Kapitola 5

Implementácia

V tejto kapitole budem rozoberať popis implementácie zvoleného algoritmu, objektový návrh programu a použité dátové štruktúry. Ďalej táto kapitola obsahuje prehľad nastaviteľných parametrov programu.

Pre implementáciu programu som sa rozhodol zvoliť hore spomínaný návrh, ktorý využíva dynamické programovanie (viď. 4.2).

5.1 Progrmaovací jayzk

Pre implementáciu by bolo možné použiť viacero rôznych prístupov spojených s programovaním, napríklad objektovo, alebo procedurálne. Rozhodol som sa využiť objektový prístup, ktorý umožňuje vyššiu úroveň abstrakcie a zároveň možnosť jednoduchého prístupu k ďalším rozšíreniam programu v budúcnosti.

Kvôli platformovej nezávislosti výsledného programu a vysokej úrovni abstrakcie som ako programovací jazyk zvolil jazyk *Java 6.0*.

5.2 Objektový návrh

Nasledujúce rozdelenie programu do balíkov som zvolil kvôli lepšej možnosti nasledujúcich rozšírení a oddeleniu dátových typov od algoritmov. Triedy programu sú rozdelené do troch balíkov :

- `Data`
- `Search`
- `Triplex`

Baík `Data` obsahuje dátove štruktúry potrebné pre použitie vyhľadavania. Trieda `DnaData.java` implementuje dátové štruktúry potrebné pre prácu s nukleotidmi. V triede `TextMiner.java` je implementované načítavanie dát zo súboru. Výsledné dáta sú uložené v objekte typu `DnaData`. Pri načítavaní je priamo dopočítané aj komplementárne vlákno k vláknu zadnému. Trieda `ScoreValues.java` obsahuje informácie o potenciálnom výskyte triplexu a metódu na prevedenie týchto hodnôt na zobraziteľný reťazec znakov.

Balík `Search` obsahuje jedinú triedu `DeepSearch.java`. Táto trieda obsahuje implementáciu samotného vyhľadávacieho algoritmu podľa návrhu v kapitole 4.2. Podrobnejší popis implementácie je v sekcii 5.5.

V balíku `Triplex` sa nachádza jediná trieda `Main.java`. táto trieda má nastarosti základne funkcie programu ako spracovanie parametrov, výpisy nálezov, volanie metód a vytváranie objektov.

5.3 Vstup programu

Predpokladaným vstupným súborom je súbor, ktorý obsahuje textové dáta, ktoré reprezentujú nukleotidy obsiahnuté v prvom vlákne molekuly DNA. Predpokladaná orientácia vlákna je v smere $5' \rightarrow 3'$. Pre opačnú orientáciu vlákna nebude funkcia programu korektná. Nesprávnu orientáciu nieje možné overiť za behu programu, preto je táto kontrola na užívateľovi.

Načítavanie dát má na starosti trieda `TextMiner.java` balíka `Data`, ktorá berie do úvahy jedine znaky reprezentujúce tieto nukleotidy, všetky ostatné znaky sú ignorované. Na načítanie dát je využitý vysoko-úrovňový prístup triedy `java.io.BufferedReader`. Súbor je potom prechádzaný po znakoch a jednotlivé znaky sú ukladané do dátovej štruktúry, ktorá je popísaná v nasledujúcej sekcii (viď 5.4).

Po prekódovaní a uložení daného nukleotidu je automaticky dopočítaný nukleotid komplementárneho vlákna, ktorý je taktiež uložený.

5.4 Reprezentácia DNA

DNA je možné reprezentovať viacerími spôsobmi, napríklad ako postupnosť znakov, stromové štruktúry, pole celých čísel, prípadne bitové polia. Pri objektovom návrhu riešenia som sa sa rozhodol implementovať DNA ako samostatný objekt, ktorý obsahuje zadané vlákno a vlákno k nemu komplementárne. Tento objekt je reprezentovaný triedou `DnaData.java` v balíku `Data`. Pre zvolený algoritmus bolo najvýhodnejšie použiť pole celých čísel, keďže pre hodnotenie sú použité tabuľky a celé čísla je možné použiť na ich indexáciu. V DNA sú obsiahnuté len štyri nukleotidy, preto je použitý najmenší celočíselný typ jazyka Java, `java.Lang.Byte`. Nukleotidy sú potom reprezentované ako je uvedené v tabuľke 5.1. Nukleotidy

Nukleotid	Hodnota
A	0
G	1
T	2
C	3

Tabulka 5.1: Reprezentácia nukleotidov

sú uložené v kontajneri `java.util.ArrayList`, ktorý umožňuje dynamicky pridávať nukleotidy počas behu programu a tým pádom je veľkosť vstupnej sekvencie obmedzená jedine nastavením Java Virtual Machine a veľkosťou operačnej pamäte počítača, na ktorom je program spúšťaný.

5.5 Vyhľadávanie

Pre vyhľadávanie je implementovaný samostatný objekt, do ktorého je v konštruktore priradená sekvencia DNA, ktorá bude prehľadávaná. A metóda, ktorá implementuje samotné vyhľadávanie. Vyhľadávanie je možné spustiť s rôznymi parametrami. Pre tento jeden objekt

je potom možné vyhľadávať rôzne typy triplexov, bez potreby vytvárania ďalších objektov. Pri priďeľovaní skóre je predpokladaný jeden typ chyby a síce zámena znaku, pokiaľ triplex obsahuje iné druhy chýb (vynechanie, vloženie), budú tieto chyby hodnotené ako zámena znaku. Pri porovnávaní znakov dochádza k zvyšovaniu alebo znižovaniu skóre, podľa toho, či kombinácia daných nukleotidov môže utvoriť triplex.

5.5.1 Parametre vyhľadávania

Pre korektnú funkčnosť algoritmu je potrebné určiť niektoré základné parametre vyhľadávania. Tieto parametre sú zadávané pri každom volaní metódy. Zoznam parametrov :

- Typ triplexu
- Minimálna dĺžka ohybu
- Maximálna dĺžka ohybu
- Minimálna dĺžka triplexu
- Maximálna dĺžka triplexu
- Percentuálne zastúpenie chýb prípustné pre triplex

5.5.2 Tabuľky hodnôt

Trieda obsahuje osem štvorcových matíc rozmerov 4x4, pričom osi X a Y reprezentujú kombinácie nukleotidov. Matice sú určené pre každý typ triplexu osobitne a sú pevne dané. Pri spustení vyhľadávania je potom vybraná matica odpovedajúca danému typu.

5.5.3 Pohyb

Pre implementáciu nieje potrebné uchovávať celý pamäťový priestor skórovacej matice spomínanej v kapitole 4.2. Na vyhodnotenie je dostačujúce udržiavať informáciu o posledných dvoch antidiagonálach, čím sa značne zníži pamäťová náročnosť. Pohyb po antidiagonále je implementovaný pomocou hlavného cyklu, ktroý je obmedzený dĺžkou sekvencie a vstupnými parametrami. V tomto cykle sa nachádzajú dva vnorené cykly, ktoré vykonávajú porovnávania a zapisovanie skóre, pričom jeden prebieha pre reťazce párnej dĺžky a druhý pre reťazce nepárnej dĺžky.

5.5.4 Výstup

Výsledky vyhľadávania sú návratovou hodnotou metódy `searchM( parametre)`, ktorá je typu `java.util.ArrayList<ScoreValues>`. Pre výpis reťazca nálezov je možné potom použiť cyklus, v ktorom je volaná prekrytá metóda `toString`.

5.6 Beh programu

Keďže ide o konzolovú aplikáciu všetky parametre vyhľadávania je potrebné zadať ako argumenty príkazového riadku. Na spracovanie týchto argumentov je použitá knižnica `JArgs` od autora Steve Purcell[1], ktorá je priložená k programu a potrebná na jeho chod. Táto knižniica

[1] Viac informácií o knižnici na stránke http://jargs.sourceforge.net/

umožňuje spracovanie argumentov v štýle GNU (getopt). Nasleduje kontrola týchto argumentov. Pri absencii esenciálnych parametrov pre funkčnosť algoritmu je vyhlásená chyba a program je ukončený. Pokiaľ je argument určujúci percentuálne zastúpenie chýb väčší ako 50, je vypísaná varovná hláška, následne je užívateľ žiadaný o potvrdenie, či chce vo vyhľadávaní napriek tomu pokračovať.

Po spracovaní argumentov nasleduje načítanie dát zo súboru. Program predpokladá súbor odsahujúci jedine textové dáta zložené z množiny znakov T, A, G, C, t, a, g, c. Všetky ostatné znaky v súbore budú ignorované. Pri načítavaní je autamaticky dopočítané komplementárne vlákno.

Z načítaných dát je vytvorený objekt vhodný na prehľadávanie. Následne na to je spustené vyhľadávanie so zadanými parametrami. Výsledky sú potom vypísané na štandardný výstup.

V prípade akejkoľvek chyby (nesprávne parametre, nedostatok pamäte a pod.) je program ukončený a na štandardný chybový výstup je vypísaná správa o chybe.

5.7 Metriky

V tabuľke 5.2 sú popísané metriky výsledného programu.

Počet balíkov	3
Počet tried	5
Celkový počet riadkov	604
Objem zdrojových kódov	19.4kB
Objem binárnych súborov	13.9kB
Objem knižníc	11.9kB
Objem spustiteľného súboru	15.1kB

Tabulka 5.2: Metriky programu

Kapitola 6

Testovanie

Táto kapitola popisuje postupy použité pri testovaní a ladení programu. V úvodnej časti sú tieto postupy vymenované a v nasledujúcich častiach sú podrobnejšie popísané a zhodnotené výsledky.

Pri testovaní programu som použil viacero typov zdrojových dát. V tejto kapitole sú popísané postupy a výsledky testovania programu. Majoritná časť testovania prebiehala na platforme `Windows 7 64-bit` a `Java SE Rutine Enviroment 1.6.0_18`. Procesor `Intel Core 2 Duo T9400 2.53gHz` a pamäť RAM typu DDR2 o veľkosti 4096MB.

6.1 Typy testovania

Testovanie prebiehalo v troch fázach.

1. Testovanie základnej funkčnosti
2. Záťažové testovanie
3. Testovanie na reálnych dátach

V nasledujúcej časti budú tieto testovania rozobrané podrobnejšie.

6.2 Základná funkčnosť

Základná funkčnosť programu bola testovaná na špeciálnych dátach. Reálne dáta by pre tieto účely boli príliš neprehľadné. Pre toto testovanie som vytvoril osobitnú hodnotiacu tabuľku a metódu pre vypísanie celej skórovacej matice.

Dáta na vstupe programu tvorili palindróm s medzerou, pričom ich dĺžka nepresahovala viac ako 20 znakov. Toto obmedzenie bolo zavedené kvôli overeniu správnosti vizuálne a pomocou JUnit testov, ktoré porovnávali predpokladané výsledky s reálnymi výsledkami. Výpis skórovacej matice slúžil na vizuálne porovnanie hodnotenia prehľadávania. Samotná hodnotiaca tabuľka bola nastavená tak, aby týmto pravidlám vyhovovala. Po doladení funkčnosti programu vo finálnej verzii bola táto tabuľka odstránená, aby nevznikali možné problémy pri volaní vyhľadávania nereálnych štruktúr. Tak isto bola odstránená metóda pre zobrazovanie skórovacej matice, ktorá je pamäťovo a časovo náročná pre väčšie objemy.

Podobným spôsobom som testoval základnú funkčnosť na reálnych dátach, pričom som na určené miesto vložil palindróm s medzerou a predpokladom výstupu bola spomínaná pozícia v súbore. Pri tomto spôsobe už nebolo možné vypisovať celú skórovaciu maticu. Pri

reťazcoch dlhších ako 80 znakov by bol tento výpis nečitateľný v konzolovom okne, navyše vzniknutá pamäťová náročnosť bola kvadratická.

6.3 Záťažové testovanie

Pri tomto testovaní som sa sústredil na výkonnosť algoritmu a merania dĺžky vyhľadávania. Keďže algoritmus sa neukončuje po prvom výskyte potenciálneho triplexu, ale vyhľadáva všetky možné výskyty, jeho časová náročnosť bude pre rovnako dlhé rôzne sekvencie konštantná. Pokiaľ uvažujeme rovnaké parametre vyhľadávania.

Faktory, ktoré najviac ovplivňujú časovú náročnosť sú minimálna a maximálna dĺžka triplexu a celková dĺžka prehľadávanej sekvencie, pretože ovplivňujú prehľadávaný priestor. Samotná dĺžka behu programu je taktiež ovplivnená všetkými výpismi, ktoré zaberajú v porovnaní s ostatnými operáciami mnohonásobne viac času. Po ukončení vyhľadávania je počet výpisov v programe obmedzený na výpisy potenciálnych výskytov.

Pri tomto testovaní som taktiež overoval maximálnu prijateľnú dĺžku vstupnej sekvencie. V provom rade je dôležité mať dostatok prístupnej operačnej pamäti. Maximálna prijateľná dĺžka je tiež obmedzená nastaveniami Java Virtual Machine. Pri dlhších sekvenciách môže byť hlásená chyba nedostatku pamäti. Tento limit je možné zmeniť priradením väčšieho množstva pamäte v nastaveniach JVM. Pre používanie tohoto programu pre reálne dáta môže byť nevyhnutné tieto zmeny v nastaveniach vykonať.

V tabuľke 6.1 parametre pri ktorých boli vedené testovania programu. Parametre typ triplexu a percentuálne zastúpenie chýb nemajú vpliv na dĺžku prehľadávanej sekvencie, preto v tabuľke niesú uvedené. Na obrázku 6.1 je uvedený príklad výsledkov testu pre

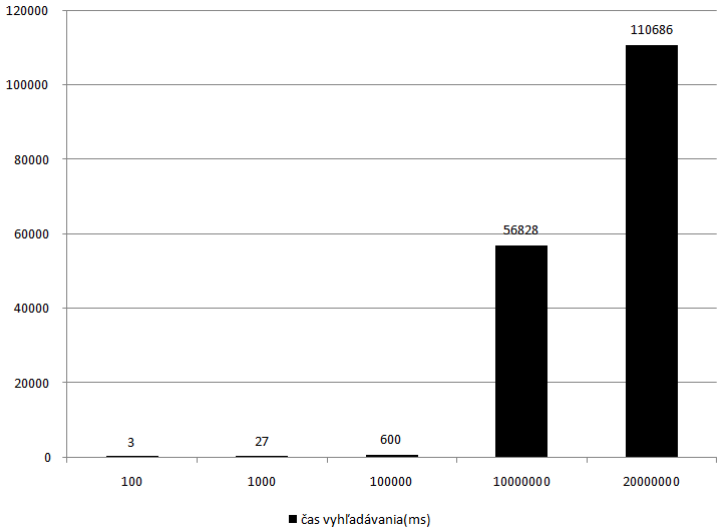
Min. dĺžka t.	Max. dĺžka t.	Min. dĺžka ohybu	Max. dĺžka ohybu
10	100	1	10
10	100	10	20
50	1000	10	20
50	100	10	20
100	1000	1	10

Tabulka 6.1: Prehľad parametrov použitých pri testovaní

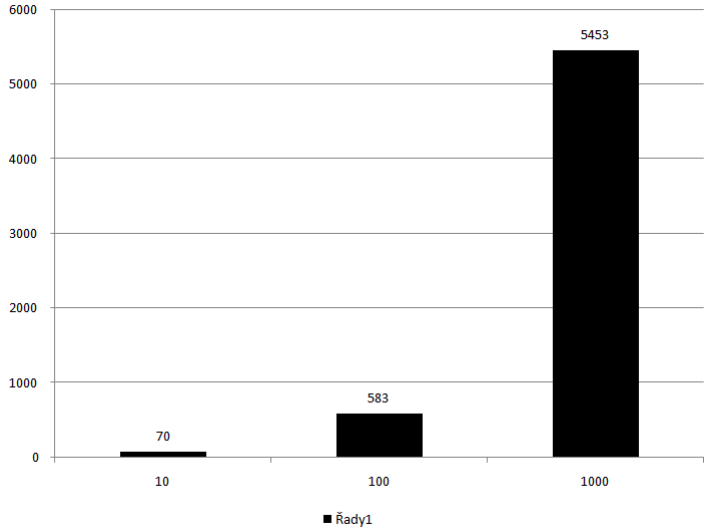
nastavenia popísané v prvom riadku tabuľky 6.1. Grafy pre ostatné hodnoty sú uložené v prílohe vzhľadom na ich veľkosť. Druhý obrázok obsahuje závislosť obmedzenia maximálnej dĺžky sekvencie. Pre tento obrázok je použitý vstupný súbor o veľkosti 100000 nukleotidov. Stĺpce vyjadrujú rozdiel v časoch pri nastaveniach (viď obr. 6.2). Dĺžka vyhľadávania pomerne rapídne stúpa pri zmene nastavení maximálnej dĺžky, nakoľko sa tím zväčšuje prehľadávaný priestor.

6.4 Reálne dáta

Na zdroje reálnych dát som použil úseky molekúl DNA dostupné z databázy *UCSC Genome browser*. Pričom boli do týchto úsekov vložené sekvencie tvoriace triplexy a následne bol program spustený so zvolenými parametrami. Očakávaným výstupom programu potom boli pozície výskytu týchto úsekov. Boli použité úseky ľudskej DNA ako aj baktérie E.coli. Tieto úseky boli zvolené kvôli tomu, že už na nich boli vykonávané vyhľadávania a experimenty ohľadom triplexov.



Obrázek 6.1: Grafické zobrazenie meraní pri testoch



Obrázek 6.2: Grafické zobrazenie meraní, vzhľadom na zmenu max. dĺžky triplexu

Kapitola 7

Možné rozšírenia

Vďaka objektovému návrhu je program jednoducho rozšíriteľný. V tejto kapitole popisujem niektoré možné rozšírenia programu v budúcnosti.

7.1 Rozlišovanie chýb

Samotný vyhľadávací algoritmus neberie do úvahy všetky možné chyby triplexov a hodnotenie jednotlivých kombinácií nukleotidov je uniformné. Jedným z možných rozšírení by bolo implementovať rozlišovanie týchto chýb z odlišným hodnotením. Pri tomto rozšírení by však stúpla pamäťová náročnosť, vzhľadom k tomu, že by bolo potrebné porovnávať aj nukleotidy na diagonálach skórovacej matice, susediaci s aktuálne prehľadávanou diagonálou.

7.2 Štatistické testy

Implementáciuu štatistických testov by užívateľ mohol získať viac informácií o prehľadávanej sekvencii. Tieto testy by boli spúšťané súčasne so základným prehľadávaním. Ďalšou možnosťou by bolo spúšťať tieto testy nezávisle na hlavnej metóde prehľadávania.

7.3 Paralélne spracovanie

Súčasná implementácia programu neumožňuje paralélne spracovanie. Pre urýchlenie vyhľadávania by bolo možné algoritmus upraviť tak, aby pracoval paralélne vo viacerých vláknach pre multiprocesorové systémy. Toto rozšírenie by vyžadovalo značné zmeny algoritmu a prístupu k riešeniu problému.

7.4 Grafické užívateľské rozhranie

Ďalším možným rozšírením je implementácia grafického užívateľského rozhrania (GUI). Toto rozšírenie by zjednodušilo ovládanie programu pre užívateľa. Vzhľadom k objektovému návrhu programu by užívateľ mohol pracovať s viacerými súbormi naraz, prípadne spúšťať vyhľadávanie s rôznymi parametrami počas jedného spustenia programu. GUI by umožňovalo užívateľovi ukladať konfiguráciu programu, prípadne parametre posledných vyhľadávaní.

Rozšírenie o GUI by umožňovalo implemtáciu nástrojov pre grafické zobrazenie výsledkov, prípadne uloženie týchto obrazových dát. Spúšťanie GUI by však malo byť voliteľnou možnosťou, z dôvodu možnosti použitia programu aj na platformách, ktoré bez užívateľského rozhrania.

Kapitola 8

Záver

V tejto práci som popisoval postup návrhu a implementácie algoritmu pre vyhľadávanie triplexov v DNA sekvenciách. V úvode práce som spomenul základné znalosti o molekule DNA a o jej vlastnostiach. Taktiež som sa venoval rozboru nekanonických štruktúr, primárne triplexov. Rozborom problematiky sa mi podarilo určiť množiny komplementárnych prvkov, na základe, ktorých bolo možné navrhnúť algoritmus.

Pri prvotnom návrhu algoritmu som vychádzal zo skúseností s vyhľadávacimi algoritmami. Tento návrh sa však po podrobnejšom štúdiu problematiky javil ako neefektívny. Preto som sa rozhodol zvoliť prístup dynamického programovania.

Pre implementáciu samotného programu som zvolil programovací jazyk Java. Dôvodom tejto voľby boli predošlé skúsenosti s programovaním v tomto jazyku a v neposlednom rade jeho nezávislosť na systémovej platforme. Nevýhodou tejto voľby je, že kód v tomto jazyku je interpretovaný a preto výsledný program môže byť pomalší, ako program písaný v jazykoch prekládaných do natívneho strojového kódu.

Testovaním som overil výkonnosť algoritmu. Na základe hodnôt nameraných pri testovaní som zostavil grafy, ktoré môžu slúžiť na porovnanie výkonu s podobnými aplikáciami. Pri testovaní som taktiež zistil, že výslednú aplikáciu by bolo vhodné v budúcnosti rozšíriť o možnosť použitia grafického užívateľského rozhrania.

Literatura

[1] Felsenfeld, G.; David, R. D.; Rich, A.: FORMATION OF A THREE-STRANDED POLYNUCLEOTIDE MOLECULE. *Journal of the American Chemical Society*, ročník 79, č. 8, 1957: s. 2023–2024, doi:10.1021/ja01565a074, http://pubs.acs.org/doi/pdf/10.1021/ja01565a074.
URL http://pubs.acs.org/doi/abs/10.1021/ja01565a074

[2] Frank-Kamenetskii, M. D.; Mirkin, S. M.: Triplex DNA Structures. *Annual Review of Biochemistry*, ročník 64, č. 8, 1995: s. 65–95, doi:10.1146/annurev.bi.64.070195.000433.
URL
http://www.annualreviews.org/doi/pdf/10.1146/annurev.bi.64.070195.000433

[3] prof. Ing. Ivo Provazník Ph.D.: Úvod do medicínské informatiky. 2009.

[4] Jirkovský, V.: *Vyhledávání palindromu*. Diplomová práca, České vysoké učení technické v Praze, Fakulta elektrotechnická, 2008.

[5] Jurčo, J.: *Program pre analýzu DNA sekvencií z hľadiska výskytu triplexov*. Bakalárska práca, Masarykova Univerzita, Fakulta Informatiky, 2008.

[6] Martínek, T.; Lexa, M.; Kopeček, D.; aj.: Identification of triplex-forming sequences. 2010.

[7] Mills, M.; Arimondo, P. B.; Lacroix, L.; aj.: Energetics of strand-displacement reactions in triple helices: a spectroscopic study. *Journal of Molecular Biology*, ročník 291, č. 5, 1999: s. 1035 – 1054, ISSN 0022-2836, doi:DOI:10.1006/jmbi.1999.3014.
URL http://www.sciencedirect.com/science/article/B6WK7-45R874W-7C/2/e4c108a55784421c7b127d4f980cc53d

[8] Soyfer, V. N.; Potaman, V. N.: *Triple-helical nucleic acids*. New York: Springer, 1995, iSBN 0-387-94495-8.

[9] stránky, W.: DNA deoxyribonukleová kyselina. `http://encyklopedie.vseved.cz/DNA+deoxyribonukleová+kyselina_textu.html`, 2005 [cit. 2011-01-21].

Příloha A

Obsah CD

- `Technicka sprava` – Zložka obsahuje zdrojové texty techickej správy a súbor Makefile pre preklad do formátu pdf a samotnú technickú správu v tomto formáte.
- `triplex` – Zložka obsahuje zdrojové kódy programu a testovacie skripty
  - `build` – zložka obsauhje preložené súbory programu
  - `dist` – zložka obsahuje spustiteľný súbor triplex.jar
  - `lib` – zložka obsahuje knižnice potrebné pre chod programu
  - `nbproject` – zložka pre import projektu do preostredia NetBeans
  - `src` – zložka obsahuje zdrojové kódy programu
  - `testdata` – zložka obsahuje testovacie dáta pre program
  - `compile_windows.bat` – skript pre kommpiláciu pre windows
  - `run_test1_windows.bat` – sktipt s príkladom spustenia pre windows
  - `compile_linux.sh` – skript pre kommpiláciu pre linux
  - `run_test1_linux.sh` – sktipt s príkladom spustenia pre linux

Příloha B

Manuál

B.1 Závislosti

Pre spustenie programu je potrebné mať nainštalované prostredie Java Rutine Enviroment. Pre kompiláciu je prerekvizitou Java Development Kit. Tieto sady sú voľne dostupné na stránke http://www.java.com

B.2 Kompilácia

V koreňovej zložke programu sa nachádza súbor `build.xml`, ktorý slúži pre kompiláciu nástrojom Ant. Program je možné kompilovať príkazom `ant -compile`. V koreňovom adresási sa nachádzajú skripty `compile_linux.sh` a `compile_windows.bat`. Bez programu ant nebude tento spôsob kompilácie fungovať. Nástroj Ant je dostupný na stránke http://ant.apache.org .

Zdrové kódy je možné otvoriť ako projekt v programe *NetBeans IDE 6.8* a kompilovať v tomto programe pomocou položky build v kontextovom menu, alebo klávesou F11 aj bez použitia nástroja ant. Vývojové prostredie NetBeans IDE je dostupné na stránke: http://www.netbeans.org

B.3 Spustenie programu

Program je po kompilácii možné spustiť z koreňového adresára aplikácie príkazom: `java -jar dist/triplex [zoznam parametrov]`.

B.4 Parametre

- Povinné parametre, pri nezadaní program hlási chybu :

  `-f,--file` – cesta k vstupnému súboru a jeho názov

  `--minb` – minimálna dĺžka ohybu vlákna (hodnoty ¿0)

  `--maxb` – maximálna dĺžka ohybu vlákna (hodnoty ¿0)

  `-t,--type` – typ triplexu (hodnoty 1-8)

- Voliteľné parametre, pri nezadaní sú nahradené konštantnou hodnotou :

`--minln` – minimálna dĺžka triplexu. Pokiaľ chýba minimálna dĺžka je nastavená na hodnotu 1.

`--maxln` – minimálna dĺžka triplexu. Pokiaľ chýba maximálna dĺžka je dĺžka celého vstupu

`--err` – prípustná chybovosť v percentách. Pokiaľ chýba prípustná chybovosť nastavená na 0%.

`-o,--out` – cesta a názov výstupného súboru pre ukladanie dát. Pokiaľ chýba je použitý štandardný výstup.

- `-h,--help` – Parameter vypíše nápovedu k programu na štandardný výstup a ukončí program. Všetky ostatné parametre sú ignorované.

B.4.1 Príklady spustenia

```
java -jar dist/triplex -f testdata/data1k.txt -t 1 --minb 1 --maxb 10 --minln
20 -- maxln 40 -o output.txt
```

```
java -jar dist/triplex -f testdata/human.txt -t 1 --minb 1 --maxb 10 --minln
20 -- maxln 40 -err 10 -o output.txt
```

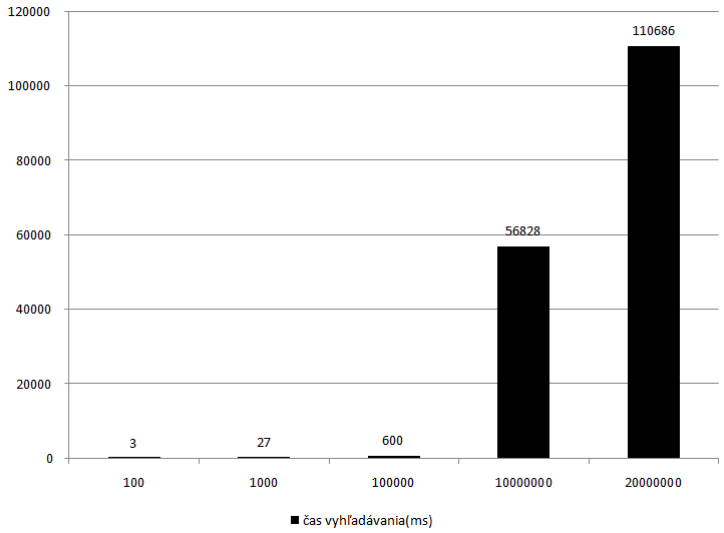
```
java -jar dist/triplex -f testdata/data100k.txt -t 1 --minb 1 --maxb 5 --minln
10 -- maxln 100 -err 15
```

Příloha C

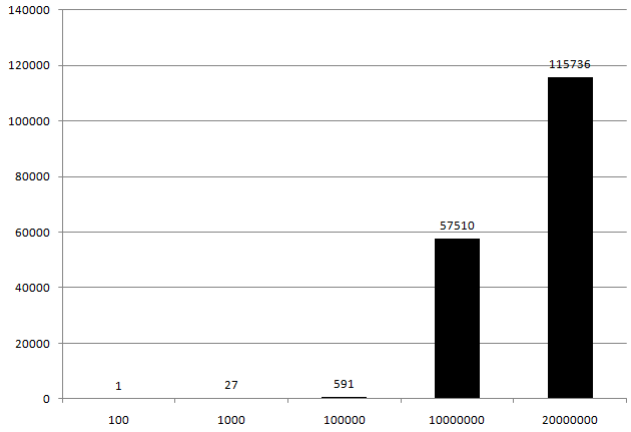
Grafická reprezentácia testov

Táto príloha obsahuje grafy reprezentujúce časy nanerané pri testovaní programu. Stĺpce grafov reprezentujú čas vyhľadávania v milisekundách. V popise grafov udávam parametre, s ktorými bolo testovanie spúšťané.

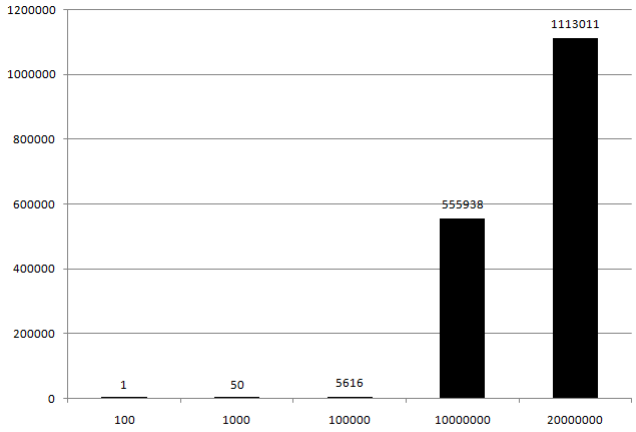
- *minln* Minimálna dĺžka triplexu
- *maxln* Maximálna dĺžka triplexu
- *minb* Minimálna dĺžka ohybu
- *maxb* Maximálna dĺžka ohybu
- *err* maximálna prípustná chybovosť



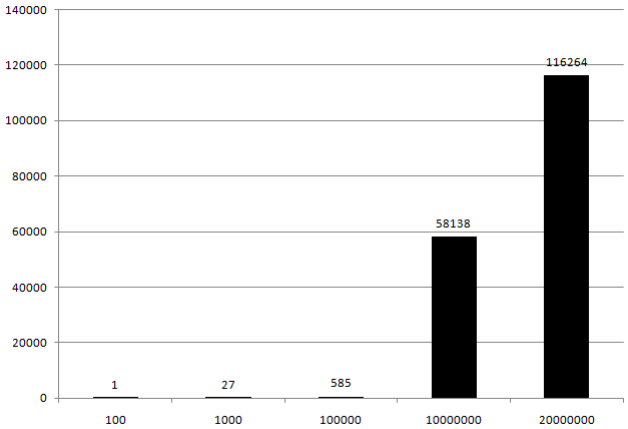
Obrázek C.1: minln = 10; maxln = 100; minb = 1; maxb = 10



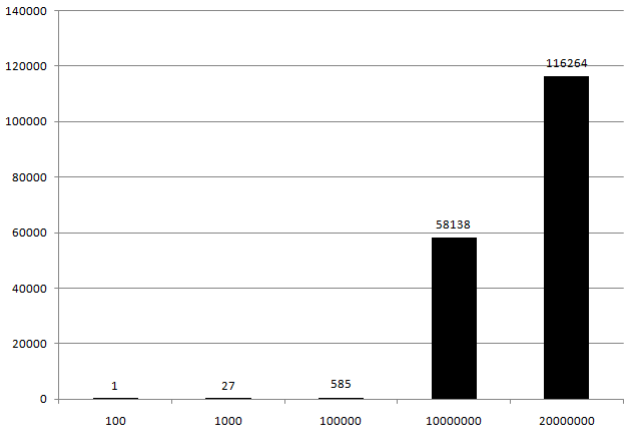
Obrázek C.2: minln = 10; maxln = 100; minb = 10; maxb = 20



Obrázek C.3: minln = 50; maxln = 1000; minb = 10; maxb = 20



Obrázek C.4: minln = 50; maxln = 100; minb = 10; maxb = 20



Obrázek C.5: minln = 100; maxln = 1000; minb = 1; maxb = 10