VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

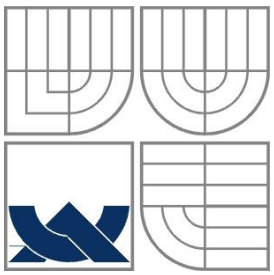
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

VYHLEDÁVÁNÍ KVADRUPLEXŮ V DNA SEKVENCÍCH

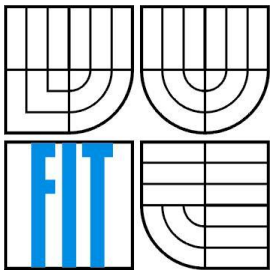
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. MARTIN SEČKA
AUTHOR

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

VYHLEDÁVÁNÍ KVADRUPLEXŮ V DNA SEKVENCÍCH

QUADRUPLEX DETECTION IN DNA SEQUENCES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN SEČKA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. TOMÁŠ MARTÍNEK, Ph.D.

BRNO 2012

Abstrakt

Tato diplomová práce se zabývá vyhledáváním a vizualizací a filtrací sekvencí potenciálně tvořících kvadruplexy v sekvencích DNA. Ve spolupráci s Biofyzikálním ústavem AV ČR byla za tímto účelem vytvořena webová aplikace, která využívá nový algoritmus pro hledání všech možných umístění kvadruplexů v rámci zadané sekvence. Návrh a implementace tohoto algoritmu jsou také součástí této práce.

Abstract

This master's thesis focuses on search for sequences potentially forming quadruplexes in DNA sequences, their visualization and filtration. For this purpose a web application was created in cooperation with the Institute of biophysics of Academy of science of Czech republic. This application uses a newly created algorithm able to find all possible formations of quadruplex within given input sequence. Design and implementation of this algorithm are also part of this thesis.

Klíčová slova

DNA, sekundární struktura, kvadruplex, webová aplikace, algoritmus

Keywords

DNA, secondary structure, quadruplex, web application, algorithm

Citace

Sečka Martin: Vyhledávání kvadruplexů v DNA sekvencích, diplomová práce, Brno, FIT VUT v Brně, 2012

Vyhledávání kvadruplexů v DNA sekvencích

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Tomáše Martínka, PH.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Martin Sečka
22.5.2012

Poděkování

Rád bych poděkoval Ing. Tomáši Martínkovi, Ph.D. za četné rady, trpělivost a čas, které mi věnoval při tvorbě této práce.

Obsah

Obsah 1
1 Úvod 2
2 Molekulární biologie 3
2.1 Informační makromolekuly 3
2.1.1 Párování bází 3
2.1.2 Sekundární struktura DNA 4
2.2 Centrální dogma molekulární biologie 5
3 Kvadruplexy 8
3.1 Vyhledávání kvadruplexů v genomu 10
Quadparser 10
QuadPredict 11
QFinder a AllQFinder 11
4 Návrh a implementace 12
4.1 Webová aplikace 12
4.1.1 Implementační technologie 12
4.1.2 Specifikace 12
4.1.3 Implementace 14
4.2 Hledání kvadruplexových sekvencí 25
4.2.1 Specifikace 25
4.2.2 Algoritmus programu pro hloubkovou analýzu 27
5 Experimenty 35
6 Závěr 36
Literatura 37

1 Úvod

G-kvadruplexy jsou specifické sekundární struktury DNA, jejichž význam v biologických procesech začíná v posledních letech stále častěji vyplouvat na povrch. Výzkum kvadruplexů nachází uplatnění nejenom v oblasti genetiky a medicíny, ale i v nanotechnologiích. Výpočetní analýza DNA sekvencí na přítomnost kvadruplexů, popř. algoritmický odhad jejich stability, je důležitým předpokladem pro umožnění jejich studia v laboratořích.

Tato diplomová práce se zabývá tvorbou webové aplikace umožňující analýzu DNA sekvencí na přítomnost nekanonických kvadruplexových struktur. Aplikace provádí vyhledávání kvadruplexů na sekvencích celých chromozomů a výsledky hledání je pak schopná vizualizovat a filtrovat tak, aby bylo snadné najít v nich relevantní informace. Kromě samotného vyhledání kvadruplexových sekvencí je aplikace schopná tyto sekvence anotovat informacemi o pozici vůči genům a provést shlukovou analýzu těchto sekvencí. Nedílnou součástí práce je také nově vytvořený algoritmus schopný vyhledat na zadané vstupní sekvenci všechna možná umístění potenciálních kvadruplexových sekvencí.

Práce vznikala za spolupráce s Biofyzikálním ústavem Akademie věd ČR, jejíž zaměstnanci poskytli potřebnou zpětnou vazbu a připomínky k fungování a rozhraní vytvořené aplikace.

První část práce obsahuje stručný úvod do molekulární biologie v takové míře, aby byly alespoň zhruba vysvětleny v dalších kapitolách často používané pojmy. To zahrnuje popis informačních makromolekul a také stručné vysvětlení centrálního dogmatu molekulární biologie. Další kapitola se zabývá konkrétněji problematikou G-kvadruplexů, především popisem jejich biologického významu a pak také existujícími nástroji pro jejich vyhledávání a analýzu.

Obsahem kapitoly čtvrté je návrh a implementace výsledné aplikace. Kapitola popisuje obecnou specifikaci vytvořených aplikací a následně obsáhne způsob implementace uživatelského rozhraní, stejně jako popis nejdůležitějších použitých algoritmů včetně nově vytvořeného algoritmu pro vyhledávání kvadruplexů.

Kapitola pět obsahuje výsledky dosažené pomocí vytvořené aplikace na lidském genomu.

2 Molekulární biologie

Molekulární biologie studuje vztah struktury a interakcí biologických makromolekul (biomakromolekul) k funkcím a vlastnostem živých soustav. Jinými slovy zkoumá vztah mezi fyzikální a chemickou úrovní živých soustav, které představuje struktura a interakce biologických makromolekul a biologickou úrovní, kterou představují funkce a vlastnosti živých soustav [1].

Cílem této kapitoly není popsat všechny základní struktury a procesy spadající do oblasti molekulární biologie, ale pouze stručně osvětlit pojmy, které jsou v této práci dále používány nebo s ní blíže souvisí.

2.1 Informační makromolekuly

Biologické makromolekuly, mezi nimiž dochází v živých soustavách k přenosu genetické informace jsou informační makromolekuly. Jsou to molekuly polymerního charakteru, tzn. že jsou složeny z mnoha kovalentně spojených malých molekul označovaných jako **monomery** [1].

Nukleové kyseliny jsou makromolekulární látky tvořené polynukleotidovými řetězci. Podle druhu polynukleotidového řetězce se rozlišují dva typy nukleových kyselin [1]:

a) **Kyselina ribonukleová** (RNA), tvořená jedním nebo dvěma komplementárními polyribonukleotidovými řetězci, jejímiž monomery jsou ribonukleotidy kyselina uridylová, cytidilová, adenylová a guanylová [1].
b) **Kyselina deoxyribonukleová** (DNA) sestávající z jednoho nebo dvou komplementárních polydeoxyribonukleotidových řetězců, jejímiž monomery jsou deoxyribonukleotidy kyselina deoxytymidylová, deoxycytidylová, deoxyadenylová a deoxyguanylová [1].

Nukleotid je sloučenina nukleozidu s kyselinou fosforečnou. Molekula nukleotidu sestává z pětiuhlíkového sacharidu neboli pentózy (ribózy u RNA a deoxyribózy u DNA), kyseliny trihydrogenfosforečné a purinové nebo pyrimidinové báze. Purinové báze jsou adenin (A) a guanin (G), pyrimidinové báze jsou cytosin (C), tymin (T), popř. uracil (U) v případě RNA [1].

Pořadí nukleotidů v DNA a RNA se označuje jako **primární struktura** DNA nebo RNA a obsahuje informaci, podle níž se v buňce vytváří primární struktura proteinu. Úsek polynukleotidového řetězce, který se vyznačuje určitým pořadím nukleotidů se označuje jako **nukleotidová sekvence** [1].

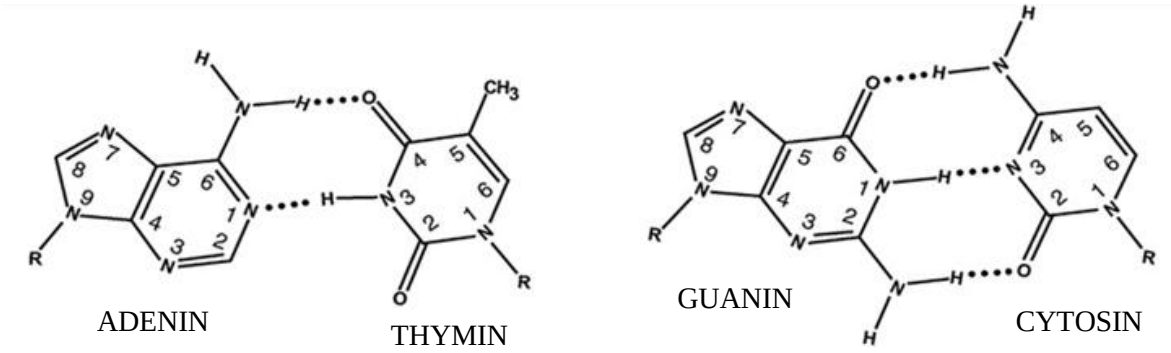
V polynukleotidovém řetězci jsou nukleotidy navzájem spojeny 3', 5'-fosfodiesterovou vazbou. V důsledku toho je jeden konec řetězce označován jako **3'-konec** (končí -OH skupinou) a druhý jako **5'-konec** (končí fosfátovou skupinou) [1,8].

2.1.1 Párování bází

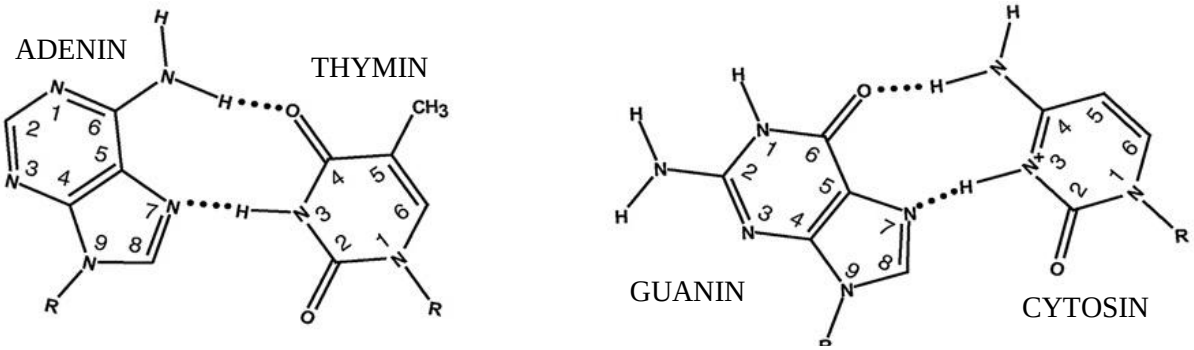
Párováním bází se rozumí spojení dvou bází vodíkovými vazbami. Párováním bází mezi dvěma DNA řetězci vzniká dvouřetězcová DNA neboli duplex, mezi třemi DNA-řetězci třířetězcová DNA neboli triplex a mezi čtyřmi DNA-řetězci čtyřřetězcová DNA neboli kvadruplex [1].

Základním párováním bází je **Watson-Crickovo párování**. Uplatňuje se obecně ve dvouřetězcových DNA, během tvorby RNA a v dvouřetězcových RNA. Podle tohoto pravidla se prostřednictvím vodíkových vazeb adenin páruje s thyminem a guanin s cytosinem. Při interakcích RNA s DNA se adenin páruje s uracilem. Nukleotidové sekvence, které se spojují Watsonovým-

Crickovým způsobem se označují jako **komplementární**, což má zásadní význam pro kopírování DNA [1]. Alternativním způsobem párování bází je **Hogsteenovo párování**.



Obrázek 1: **Watson-Crickovo párovámí bází.** *Guanin se páruje cytosinem pomocí tří vodíkových vazeb, adenin se páruje s thyminem pomocí dvou vodíkových vazeb. Převzato z [13].*

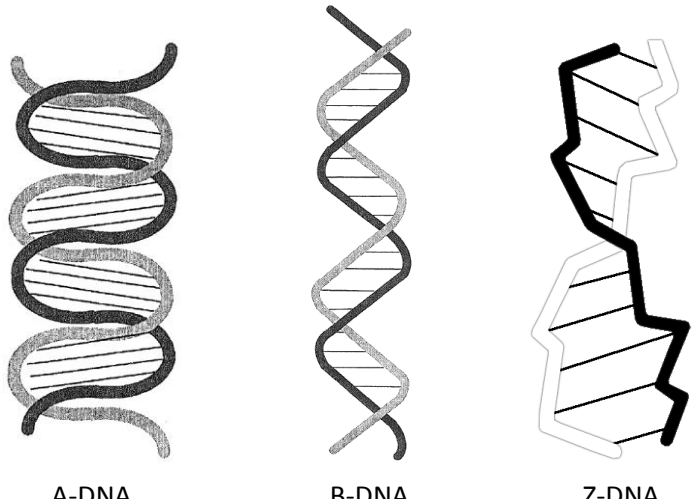


Obrázek 2: **Hoogsteenovo párovámí bází.** *V obou zobrazených případech se báze párují pomocí dvou vodíkových vazeb. Převzato z [13].*

2.1.2 Sekundární struktura DNA

Nejčastější podoba sekundární struktury DNA je tzv. **dvoušroubovice** sestávající ze dvou polydeoxyribonukleotidových řetězců šroubovitě ovíjejících společnou osu neboli osu dvoušroubovice. Oba řetězce jsou navzájem komplementární, tj. jejich nukleotidové sekvence vyhovují pravidlu o párování bází a také **antiparalelní**, tj. liší se směrem fosfodiesterové vazby. Toto se označuje jako **antiparalelizmus**, kterým se rozumí orientace komplementárních polynukleotidových řetězců ve směru 3' --> 5' na jednom řetězci a 5' --> 3' na řetězci druhém [1].

Tato struktura, označovaná jako B-DNA, však zdaleka není jedinou možnou formou. Mezi alternativní struktury patří pravotočivá dvoušroubovice A-DNA a levotočivá Z-DNA, křížová forma DNA, třívláknové triplexy a pak také čtyřvláknové kvadruplexy, kterými se tato práce hlouběji zabývá [1,9].



Obrázek 3: **Různé konformace DNA.** *Převzato z [1]*.

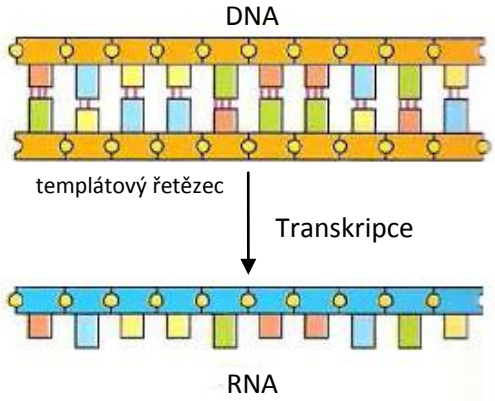
2.2 Centrální dogma molekulární biologie

Centrální dogma molekulární biologie popisuje směr a způsob přenosu informace mezi biopolymery. Základními procesy probíhajícími s jistými obměnami v buňkách všech organismů jsou transkripce, translace a replikace.

Transkripce (přepis) je proces, kterým buňka realizuje své geny. Tímto procesem vzniká veškerá RNA v buňce. Podle jednoho genu může vzniknout mnoho kopií RNA a jedna RNA může dá vzniknout několika identickým molekulám proteinu. Protože v buňce bývá obsažena jen jedna kopie genu na haploidní genom, umožňuje tato amplifikace nasyntetizovat požadované množství proteinu mnohem rychleji, než kdyby DNA byla sama templátem pro syntézu proteinů [8].

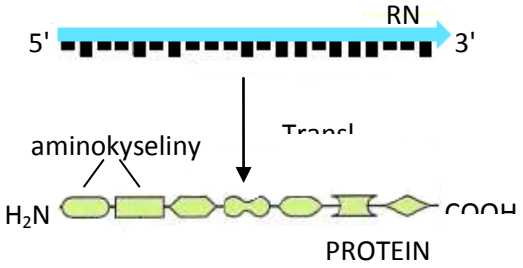
Transkripce začíná rozvolněním krátkého úseku dvoušroubovice DNA, jeden z řetězců pak slouží jako templát pro syntézu RNA. Sekvence RNA je určena komplementárním párováním bází, tzn. že RNA je komplementární k templátovému řetězci DNA. Enzymy, které přepisují DNA do RNA, se nazývají **RNA-polymerázy**. Před začátkem transkripce musí RNA-polymeráza přesně rozpoznat začátek genu a navázat se na toto místo. U bakterií má RNA-polymeráza snahu slabě se vázat na molekulu DNA a potom se po ní rychle pohybovat skluzem. Jakmile RNA-polymeráza rozpozná v DNA sekvenci tzv. **promotoru**, která obsahuje informaci o začátku transkripce, pevně se naváže na tento úsek. Za rozpoznání promotorové sekvence v bakteriální DNA je primárně zodpovědná podjednotka RNA-polymerázy nazývaná sigma-faktor. U eukaryotických buněk je situace složitější a pro navázání RNA-polymerázy je potřeba více transkripčních faktorů [8].

Většina genů v buňce kóduje aminokyselinovou sekvenci proteinů a molekuly RNA vzniklé transkripcí těchto genů jsou společně nazývány mediátorová RNA (mRNA). Konečným produktem jiných genů je samotná RNA, jako např. transferová RNA (tRNA), která připojuje aminokyseliny do rostoucího aminokyselinového řetězce při translaci [8].



Obrázek 4: **Transkripcí vzniká RNA**, *která je komplementární k jednomu z řetězců DNA. Převzato z [8].*

Překlad informace z RNA na strukturu proteinu je proces nazývaný **translace.** Pravidla, kterými se řídí přenos z mRNA do aminokyselinové sekvence jsou nazývána jako **genetický kód.** Sekvence nukleotidů RNA je postupně čtena ne jako jednotlivé báze, ale jako jejich trojice (triplety). Protože RNA je lineární polymer složený ze čtyř bází, lze spojením tří nukleotidů vytvořit 64 kombinací. V proteinech se však obvykle vyskytuje 21 aminokyselin, proto jsou některé skupiny tripletů překládány jako stejná aminokyselina. Každá skupina tří nukleotidů RNA se nazývá **kodon** a určuje jednu aminokyselinu [8].

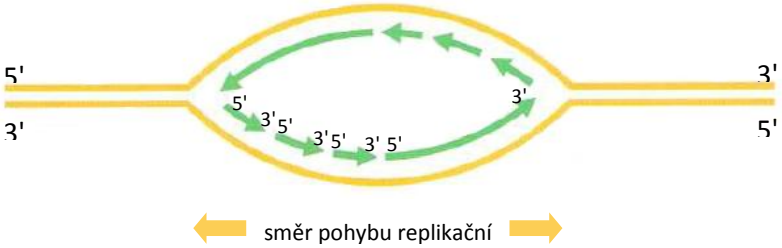


Obrázek 5: **Translací vzniká z RNA protein.** *Převzato z [8].*

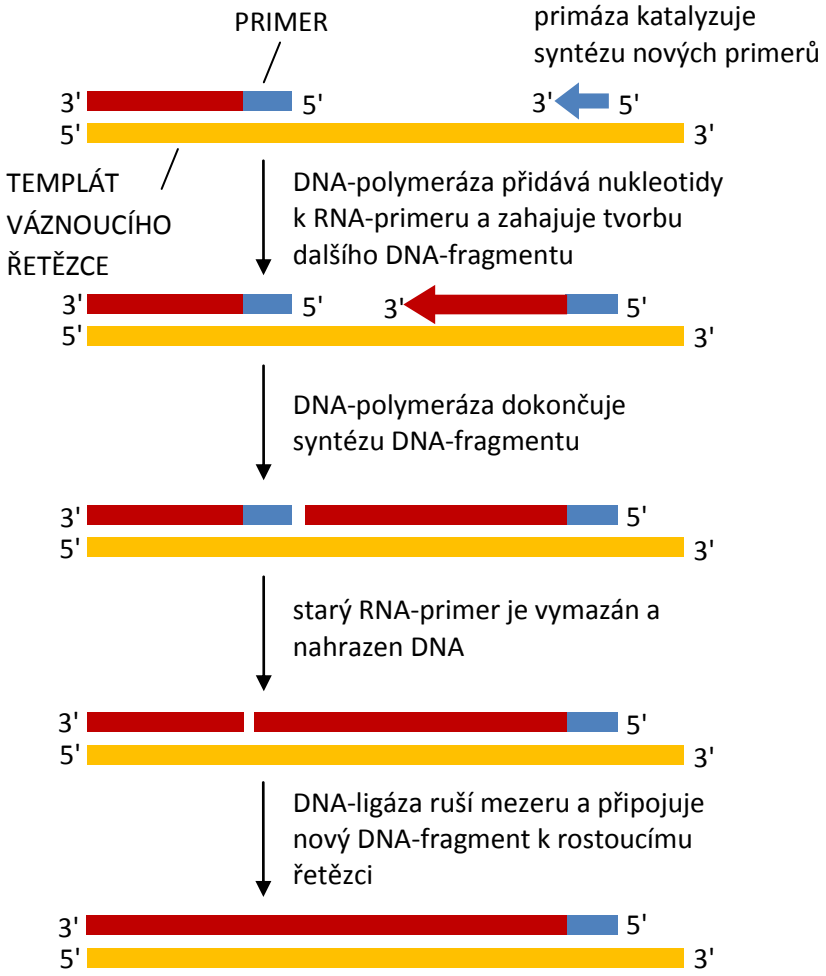
Replikace DNA dává vznik dvěma novým dvoušroubovicím, které pocházejí z dvoušroubovice mateřské. Obě jsou kromě vzácných chyb jejími věrnými kopiemi. Oba řetězce při tomto procesu slouží jako templáty pro vznik nového vlákna, každá dceřiná dvoušroubovice je proto tvořena jedním původním a jedním nově vzniklým vláknem. Místům, kde je struktura dvoušroubovice DNA nejdříve narušena, se říká **replikační počátky** a jsou určeny speciální nukleotidovou sekvencí. Pro začátky replikace jsou typické útvary ve tvaru Y, které se nazývají **replikační vidličky**. V nich jsou navázány proteiny replikačního aparátu, které se pohybují ve směru replikace a rozvíjejí dvoušroubovicovou strukturu za současné syntézy nového řetězce. V jednom replikačním počátku se vytvoří dvě vidličky, které se pohybují směrem od sebe, a proto je replikace bakteriálního i eukaryotního chromosomu nazývána *obousměrná* [8].

DNA-polymeráza je schopná katalyzovat růst řetězce DNA pouze ve směru 5'-->3'. V opačném směru roste DNA *diskontinuálně*, což znamená, že jsou syntetizovány krátké úseky DNA (**Okazakiho fragmenty**), které jsou následně spojovány v kontinuální řetězec. Řetězec tvořený kontinuálně se nazývá **vedoucí řetězec**, druhé vlákno se nazývá **váznoucí řetězec.** DNA-polymeráza může nový nukleotid připojit pouze ke spárovaným nukleotidům a není schopná sama začít replikaci na jednom vlákně. Proto existuje enzym *primáza* schopný syntetizovat zcela nové vlákno podle jednořetězcové DNA. Primázou vytvořené úseky jsou dlouhé přibližně 10 bp a poskytují svůj 3'-konec jako začátek pro DNA-polymerázu. Tyto úseky se nazývají **primery**. Při syntéze vedoucího řetězce je potřeba pouze jeden primer, syntéza opožďujícího se řetězce je však diskontinuální a

vyžaduje neustálou tvorbu primerů. Pro vytvoření souvislého vlákna DNA je ještě z opožďujícího se řetězce je třeba odstranit primery (provádí enzym *nukleáza*), nahradit je DNA (*opravná DNA-polymeráza*) a nakonec spojit všechny úseky dohromady (*ligáza*) [8].



Obrázek 6: **Směr pohybu replikačních vidliček.** *Krátké šipky směřující proti směru replikace znázorňují diskontinuálně syntetizované Okazakiho fragmenty.Převzato z [8].*

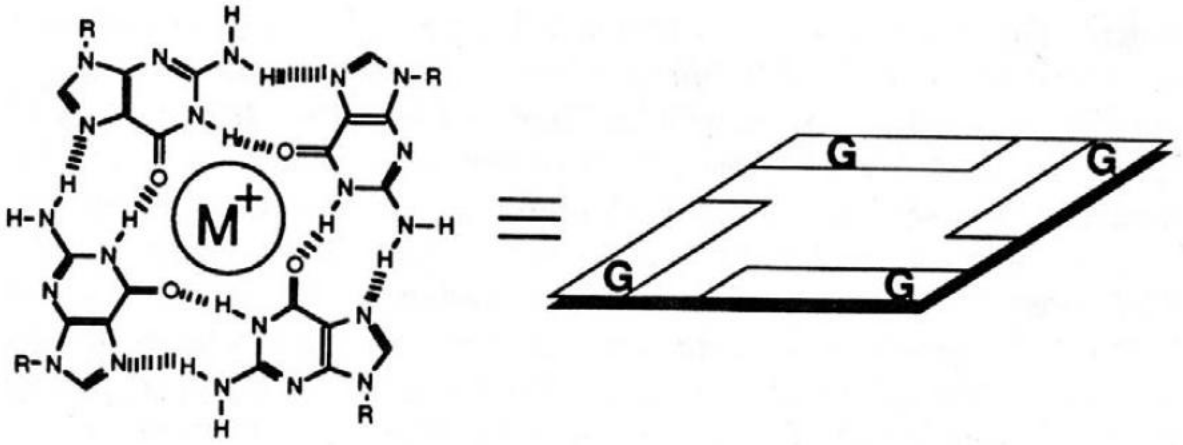


Obrázek 7: **Syntéza fragmentů opožďujícího se řetězce.**

3 Kvadruplexy

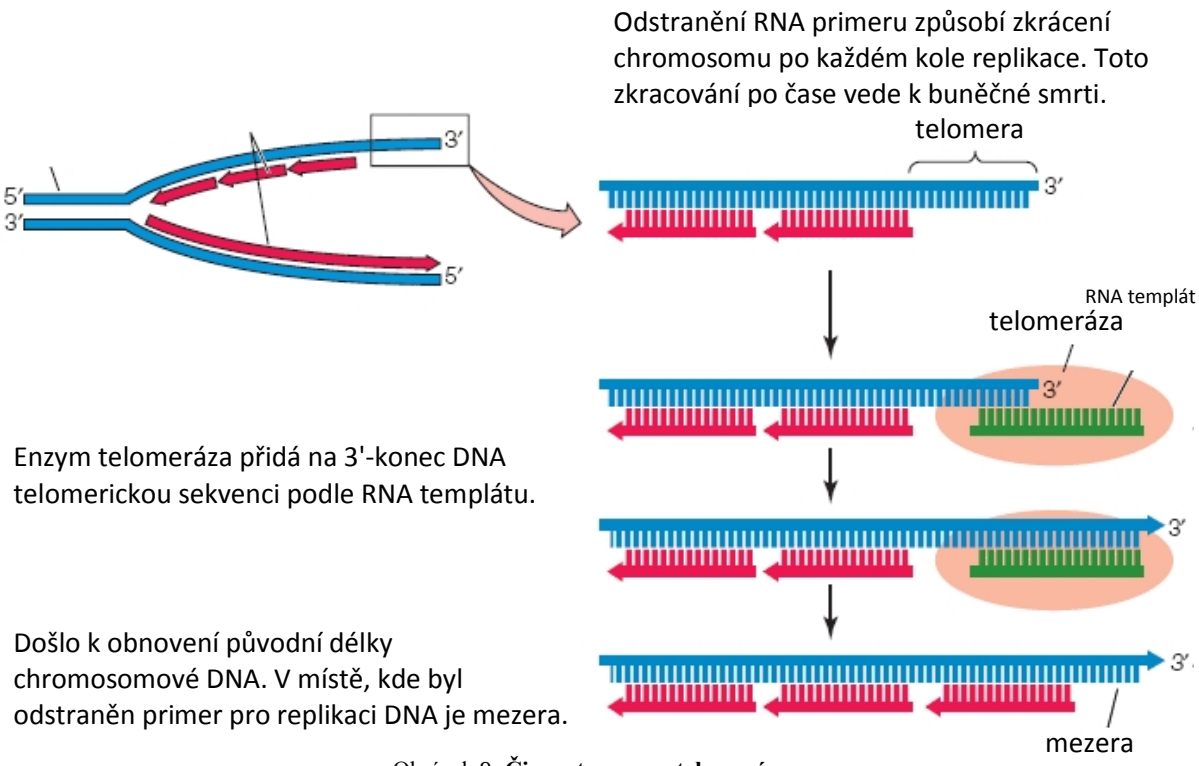
G-kvadruplexy jsou DNA a RNA struktury vyššího řádu tvořené z nukleotidových sekvencí bohatých na guanin. Jejich základním strukturním motivem jsou čtveřice Hoogsteenovými vodíkovými vazbami spojených guaninových bází nazývané **G-tetrády**. Tyto tetrády se skládají na sebe a tvoří čtyřvláknovou šroubovicovou strukturu podobně jako Watson-Crickovy bázové páry v dvouvláknové DNA [2].

Kvadruplexy mohou být tvořeny jednou (**intramolekulární**), dvěma nebo čtyřmi (**intermolekulární**) samostatnými vlákny DNA nebo RNA a mohou vykazovat vysokou rozmanitost topologií způsobenou částečně velkým množstvím kombinací různě směřujících vláken, stejně jako velikostí smyček a délkami celých kvadruplexových sekvencí. Obecně je lze definovat jako strukturu tvořenou jádrem z minimálně dvou tetrád, jež jsou pospolu drženy **smyčkami** vznikajícími ze sekvencí jiných nukleotidů, které se obvykle do tvorby tetrád nezapojují [2].



Obrázek 8: **G-tetráda** *je cyklické uspořádání čtyř guaninových bází navzájem spojených vždy dvěma vodíkovými vazbami.Kapsa vzniklá uvnitř tetrády je schopná pojmout monovalentní kationt [9].*

Těmto strukturám nebyla přikládána důležitost, dokud nebylo zjištěno, že krátké na guanin bohaté sekvence na koncích telomerických DNA v eukaryotických chromozomech jsou schopné se spolu spojovat a utvářet tak čtyřvláknové struktury mající alespoň dvě těsně sousedící G-tetrády. **Telomery** jsou oblasti DNA na koncích chromozomů obsahující tandemové repetice nesestávající pouze z guaninových bází. Enzymy provádějící replikaci DNA při ní nejsou schopny dojít až na úplný konec chromozomu. Existence telomerické DNA umožňuje zkrácení chromozomu o tyto repetice, které nenesou žádnou informaci, a zabraňují poškozování genů při replikaci. V nepřítomnosti telomer by v důsledku zkrácení nedocházelo k replikování celé užitečné informace. Telomerická DNA tedy také není replikována celá a o její obnovování se stará enzym telomeráza. Zvýšená aktivita telomerázy je spojována s přibližně 85 % případů rakoviny a bylo ukázáno, že telomerické kvadruplexy tuto aktivitu snižují [2,3,4]. Činnost telomerázy je ilustrována na obrázku 9.



Obrázek 9: **Činnost enzymu telomerázy.**

Mimo tuto funkci bylo ukázáno, že sekvence s potenciálem pro tvorbu intramolekulárních kvadruplexů jsou silně zastoupeny v promotorových oblastech různých organismů a že mají spojitost s kontrolou genové exprese [7]. Dále bylo ukázáno, že utvoření G-kvadruplexu v 5'UTR[1] může vést ke snížení translační aktivity a byly také nalezeny proteiny interagující výhradně s kvadruplexy [10].

Sekvence tvořící potenciální jednomolekulární G-kvadruplexy lze popsat pomocí vzorce:

$$G_mX_nG_mX_oG_mX_pG_m \qquad (1)$$

kde *m* je počet po sobě jdoucích guaninových bází v každé krátké sekvenci označené jako G, které se obvykle přímo účastní interakcí v G-tetrádách. X_n, X_o a X_p může být jakákoliv kombinace bází včetně guaninu, které tvoří smyčky. Z této notace rovněž vyplývá, že sekvence G mohou být různých délek a v případě, že je jedna delší než druhá, budou se některé z guaninových bází nacházet v oblasti smyček [2].

Dvoumolekulární kvadruplexy jsou téměř vždy tvořeny spojením dvou identických sekvencí popsaných vzorcem

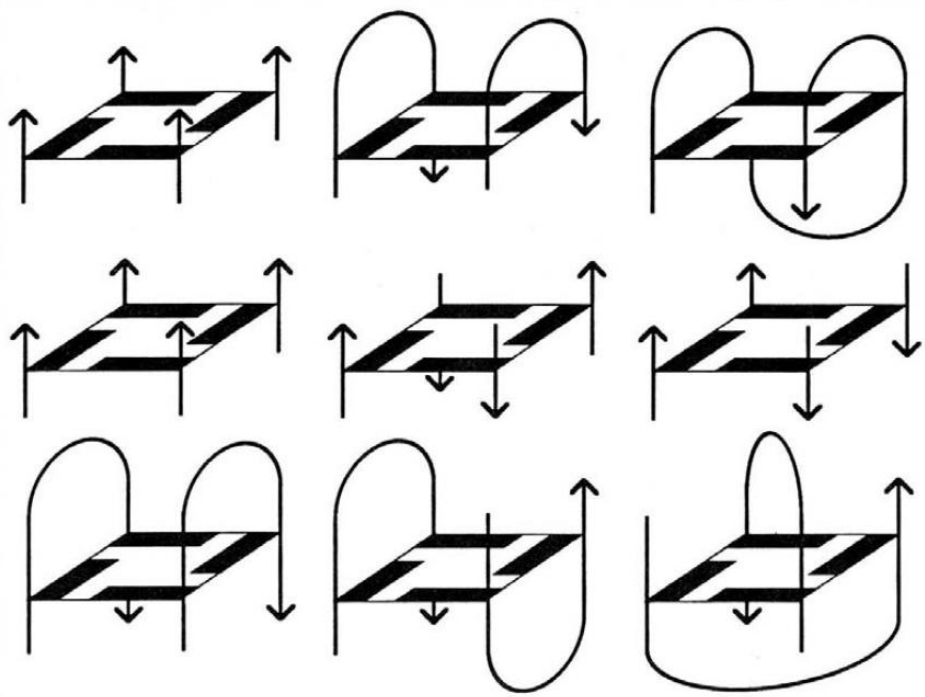
$$X_nG_mX_oG_mX_p \qquad (2)$$

kde *n* a *p* mohou ale nemusí být nulové. Čtyřvláknové kvadruplexy se mohou utvořit spojením ze čtyř vláken odpovídajících vzorcům

$$X_nG_mX_o \qquad (3)$$

$$G_mX_nG_m \qquad (4)$$

[1] UTR; z anglického *untranslated region* - dvě sekvence obklopující z obou stran kódující sekvenci mRNA.



Obrázek 10: **Polymorfismus tetrád G-kvadruplexů.** *Tyto struktury mohou být tvořeny jedním, dvěma nebo čtyřmi řetězci DNA (horní řada). Relativní uspořádání sousedních řetězců může být paralelní, střídavě paralelní nebo sousedně paralelní (prostřední řada). Spojovací smyčky mohou být uspořádány vedle sebe, proti sobě nebo diagonálně (třetí řada) [9].*

3.1 Vyhledávání kvadruplexů v genomu

Aby bylo možné identifikovat a prozkoumat nové sekvence tvořící kvadruplex, je užitečné identifikovat tyto sekvence rychlým a jednoduchým způsobem, založeným výhradně na pořadí bází v sekvenci. Pro potvrzení kvadruplexové struktury výsledků těchto metod je pak možné využít biofyzikální metody [4].

Nejpodstatnějším omezením metody předpovědi založené na vyhledávání podle pořadí bází je její neschopnost předpovědět termodynamickou stabilitu struktury, což je pro G-kvadruplexy klíčová vlastnost. Aby mohl kvadruplex zastávat biologicky významnou roli, je nutné, aby byl dostatečně stabilní za tělní teploty [5].

Quadparser

Za účelem rychlé identifikace kvadruplexových sekvencí byl vyvinut algoritmus *Quadparser*[2] schopný identifikovat potenciální kvadruplexové sekvence v datech ve formátu FASTA. Pro potřeby tohoto algoritmu bylo odvozeno jednoduché pravidlo, které sekvence tvořící kvadruplex popisuje. Do úvahy byly při tvorbě tohoto pravidla vzaty čtyři aspekty: stechiometrie vláken; počet tetrád v kvadruplexovém jádře, přítomnost mutací či mezer a délka a složení smyček. Toto pravidlo sice předvídá sekvence, které by mohly tvořit kvadruplex, nevylučuje však, že některé nebo všechny motivy budou vytvářet jiné struktury [4].

Protože za fyziologických podmínek je koncentrace DNA velmi nízká, není vznik vícevláknových kvadruplexů příliš pravděpodobná. Analýza je z tohoto důvodu zaměřena na sekvence formující jednomolekulární kvadruplexové struktury [4].

V principu mohou G-kvadruplexy být tvořeny libovolným množstvím navrstvených G-tetrád. Obecně lze říct, že stabilita roste společně s navyšujícím se počtem tetrád. Samostatné tetrády byly

[2] dostupné online na adrese http://www.quadruplex.org/?view=quadparser_web

objeveny pouze v silně koncentrovaných guaninových roztocích a není pravděpodobné, že by měli fyziologický význam. Protože dvoutetrádové struktury jsou obecně mnohem méně stabilní, byly v původním algoritmu uvažovány pouze sekvence schopné vytvořit tři a více tetrád [4].

Intramolekulární kvadruplex musí obsahovat tři smyčky, které spojují tetrády dohromady a hrají významnou roli pro stabilitu kvadruplexu. Bylo zjištěno, že smyčky o délce 1 až 7 bází jsou schopny tvořit kvadruplexy, stabilita se však snižuje s narůstající délkou. Není vyloučeno, že se kvadruplexové struktury mohou utvářet i se smyčkami o délce 8 bází a více, kvůli snižující se stabilitě byla jejich délka ve vyhledávacím pravidle omezena shora na 7 bází [4].

Na základě těchto úvah bylo vytvořeno následující pravidlo / vzorec popisující sekvence, které za přibližně fyziologických podmínek utvářejí kvadruplexy [4]:

$$G_{3+}N_{1-7}G_{3+}N_{1-7}G_{3+}N_{1-7}G_{3+} \quad (5)$$

QuadPredict

QuadPredict[3] poskytuje webovou službu, která za pomoci algoritmu využívajícího Bayesovské učení předpovídá stabilitu nezměřeného G-kvadruplexu, přesněji řečeno teplotu tání G-kvadruplexů podle jejich sekvence a koncentrace kationtů K^+, Na^+, Mg^{2+} a NH_4^+. Mimo samotné předpovědi program uvádí také míru nejistoty, tj. předpokládanou odchylku předpovědi ve stupních Celsia. Tato míra je počítána na základě podobnosti analyzované sekvence s trénovacími daty algoritmu. Pro trénování algoritmu byla využita sada již změřených sekvencí, přičemž tato sada je nadále rozšiřována novými daty [6,10].

QFinder a AllQFinder

Tyto dva algoritmy byly vytvořeny v rámci bakalářské práce Jiřího Němce v roce 2010 na Fakultě informačních technologií Vysokého učení technického v Brně. Jejich návrh vychází z dříve existujících nástrojů, program proto obsahuje podobná nastavení a na výstupu generuje podobnou sadu informací. Rozdělení na dva programy bylo zvoleno z důvodu vyšší efektivity [11].

Program QFinder provádí rychlou a efektivní analýzu vstupní sekvence a na výstup vypisuje jednotlivé sekvence (a informace o nich, které zároveň slouží jako vstup pro druhý program), které odpovídají pravidlu specifikovanému parametry programu. Program AllQFinder pak slouží k hloubkové analýze jediné sekvence a zjištění všech možných potenciálních kvadruplexových sekvencí a jejich hodnocení [11].

[3] dostupné online na adrese http://www.quadruplex.org/?view=tmPredict

4 Návrh a implementace

Cílem této práce je zdokonalení algoritmu pro hloubkovou analýzu potenciálních kvadruplexových sekvencí, který je implementován aplikací AllQFinder. To znamená především použití jiné metody pro procházení prostoru možných kvadruplexů v zadaných sekvencích a s tím související celkové zrychlení hledání. Druhým cílem je vytvoření webové aplikace schopné získané výsledky přehledně zobrazit a umožňující množinu výsledků omezit zadáním kritérií, jako je poloha v rámci genomu nebo vzdálenost od ostatních výsledků. Pro pokročilejší vizualizaci a snadný přístup k dalším informacím o nalezených sekvencích bude aplikace umožňovat zobrazit libovolný nález pomocí externího nástroje k tomuto určeného, konkrétně UCSC Genome Browseru.

Sekvence zobrazované touto aplikací ovšem nemusí být omezeny pouze na výsledky vyhledávání kvadruplexů, ale stejným způsobem mohou být procházeny a filtrovány i jiné typy sekvencí.

4.1 Webová aplikace

4.1.1 Implementační technologie

Důvodů, proč je pro prezentaci výsledků hledání kvadruplexů zvolena webová aplikace je několik:

- přístupnost - aplikace je ihned dostupná z jakéhokoliv PC schopného spustit moderní webový prohlížeč (tj. nové verze Firefoxu, Google Chrome, Opery nebo IE alespoň ve verzi 8). Mobilní verze aplikace vytvořena nebude.
- centralizace - veškerá data (s výjimkou některých uživatelských nastavení) jsou uložena v databázi na serveru a jsou tak přístupná odkudkoliv, kde je přístupná i samotná aplikace
- multiplatformnost - webové prohlížeče jsou k dispozici prakticky ve všech operačních systémech s grafickým uživatelským rozhraním
- zkušenost autora práce s těmito technologiemi

Posledně jmenovaný důvod je zároveň i hlavním faktorem ovlivňujícím konkrétní volbu implementačních jazyků. Serverová část aplikace bude implementována v jazyce PHP, s výjimkou částí aplikace majících na starost samotnou analýzu kvadruplexových sekvencí, které jsou implementovány v jazycích C a C++. Na straně klienta bude kromě standardních webových technologií jako XHTML, CSS a Javascript použit javascriptový framework jQuery a některé jeho doplňky. Pro ukládání persistentních dat bude použita databáze MySQL.

Pro analýzu přítomnosti potenciálních kvadruplexových sekvencí budou použity aplikace QFinder s upraveným formátem výstupů a nově vytvořená aplikace zastupující aplikaci AllQFinder při hloubkové analýze těchto výstupů. Důvodem použití programu QFinder je jednak dostupnost zdrojových kódů této aplikace a také fakt, že tato aplikace vznikla na stejné fakultě jako tato práce a nyní dojde aktivního využití.

4.1.2 Specifikace

Výsledná aplikace bude schopná v sekvencích odpovídajících chromozomům různých organismů vyhledat sekvence s potenciálem tvořit kvadruplexové struktury. Kromě těchto sekvencí bude

schopná vyhledávat i jiné sekvence, které nejsou přímo předmětem této práce a proto zde nebudou konkrétněji zmiňovány.

Nalezené sekvence budou ukládány do databáze, aby odpadla nutnost opakovaně prohledávat sekvence chromozomů, bylo možné na výsledcích provádět dodatečné analýzy a také výsledky rychle filtrovat dle uživatelem zadaných parametrů. Zároveň uložení sekvencí v databázi umožňuje pro jednotlivé chromozomy počítat statistické údaje.

Dodatečnými analýzami se konkrétně rozumí vyhledávání shluků neboli skupin sekvencí nacházejících se relativně blízko u sebe, dále anotace jednotlivých výsledků informací o pozici vůči genům daného genomu a u kvadruplexových sekvencí jejich ohodnocení.

Struktura webové aplikace

Aplikace bude rozdělena na prohlížeč výsledků (veřejná část) a administraci (neveřejná část). Veřejná část umožní procházet veškerá data nalezená analýzami sekvencí genomu, zatímco administrační část bude sloužit pro správu těchto sekvencí a správu provedených analýz a jejich výsledků.

Administrace

Administrační část aplikace bude přístupná pouze po přihlášení na účet s administrátorským oprávněním. Obsahovat bude tři základní sekce čítající správu souborů genomu, správu provedených hledání a nastavení aplikace. Práce s daty bude probíhat pomocí HTML formulářů a seznamů zobrazujících databázové záznamy.

Správa souborů se bude nadále dělit na několik podsekcí umožňujících kategorizaci jednotlivých souborů obsahujících sekvence chromozomů podle taxonomie zdrojových organismů a verze sestavení genomu (anglicky assembly). Základní jednotkou kategorizace bude po vzoru Genome Browseru klad (tj. vývojová větev, např. savci), do nějž budou zařazovány genomy. Genomy již představují konkrétní organismy, příkladem tedy může být člověk. Každý genom pak může obsahovat jednu i více verzí sekvencí sestavených a uvolněných v různou dobu. Konečně každé sestavení genomu obsahuje soubory představující daný organismus.

Vzhledem k velikosti FASTA souborů se sekvencí chromozomů (okolo 150 MB na soubor) nebude umožněno tyto soubory nahrávat na server přímo přes webové rozhraní a protokol HTTP. Namísto toho bude nutné nahrát je do určeného adresáře pomocí protokolu SFTP a v administraci umožnit takto nahrané soubory zvolit. U každého souboru pak bude (pro účely filtrace) nutné uvést velikost tohoto souboru v bázích a označení příslušného chromozomu (tyto informace se mohou sice objevovat ve FASTA hlavičce, nicméně její obsah není nijak standardizován a tak se nelze na přítomnost těchto údajů spoléhat). Proto, pokud o souboru nebude existovat záznam s metadaty, nebude soubor aplikací brán v potaz.

Nejdůležitější částí správy souborů bude seznam záznamů o FASTA souborech se sekvencemi chromozomů. Nad v něm zvolenými soubory (jedním nebo více najednou) bude možné provést vyhledání kvadruplexových či jiných sekvencí. Zároveň zde bude možnost zvolené záznamy o souborech smazat.

Správa a analýzy nalezených sekvencí bude podávat přehled o provedených hledáních, předně zde však bude umožněno provést nad množinou výsledků hledání další analýzy, tj. vyhledání shluků sekvencí, anotaci a ohodnocení kvadruplexových sekvencí.

Nastavení bude umožňovat změnu parametrů aplikace. Konkrétně se jedná o nastavení maximální vzdálenosti mezi sekvencemi shluku, maximální velikosti promotoru a vzorce pro vyhledávání kvadruplexů.

Prohlížeč výsledků

Výsledky budou zobrazitelné vždy pouze v rámci jediného chromozomu jednoho sestavení, čemuž budou funkčně přizpůsobeny i prostředky pro jejich filtraci. Výběr zobrazeného chromozomu bude probíhat skrze formulář, který nabídne výběr kladu, genomu, sestavení genomu a konečně souboru chromozomu. Tím bude zajištěna přehlednost i v případě, že databáze bude obsahovat data většího množství organismů s několika různými sestaveními jejich genomu.

Stránka s výsledky bude sestávat ze tří záložek - samotných výsledků hledání, výsledků shlukové analýzy a celkových statistik zvoleného chromozomu.

Záložkou zobrazenou ve výchozím stavu po zvolení chromozomu bude **seznam výsledků** nalezených na daném chromozomu. Konkrétní nalezená sekvence bude zobrazena včetně krátkého okolí na obou koncích nálezu. Vzhledem k tomu, že sekvence potenciálně obsahující kvadruplexy mohou svojí délkou dosahovat často několika stovek bází, budou tyto sekvence ve výpisu zkráceny a jejich úplnou podobu bude možné zobrazit dodatečně najetím kurzorem na požadovaný výsledek.

O každém výsledku budou k dispozici informace o jeho poloze v rámci chromozomu (pozice začátku, pozice konce), vláknu, na kterém se výsledek nachází, jeho délce, počtu G skupin (nebo C skupin v závislosti na vlákně DNA) a anotaci. Podle všech těchto informací s výjimkou anotace bude umožněno zobrazené výsledky řadit jak vzestupně, tak sestupně. Anotace bude zobrazena jako seznam genů, do nichž nalezená sekvence alespoň částečně zasahuje, u každého tohoto genu navíc budou připojeny informace o části zasaženého genu, případně index exonu (tj. číslo exonu od začátku transkripce počínaje nulou) nebo vzdálenost od začátku transkripce v případě, že se nalezená sekvence nachází v promotoru genu.

Velmi podobným způsobem bude proveden i prohlížeč **výsledků shlukové analýzy**. Kromě začátku, konce a délky shluku bude zobrazena i informace o počtu sekvencí ve shluku a průměrné vzdálenosti mezi sousedními sekvencemi shluku. Každý takovýto shluk pak bude možno otevřít a zobrazit si i informace o příslušejících sekvencích, které budou podány způsobem prakticky identickým (kromě filtrace a řazení) se seznamem výsledků.

Záložka **statistiky** obsáhne několik souhrnných údajů o zobrazovaném chromozomu a provedených analýzách. Těmito údaji jsou celkový počet nalezených sekvencí, počet sekvencí ležících mimo geny, počet sekvencí v exonech, počet sekvencí v intronech a počet sekvencí v promotorech.

4.1.3 Implementace

Framework

Aplikace na straně serveru využívá vlastní jednoduchý framework obstarávající nejběžnější součásti webových stránek, jako je autentizace a autorizace uživatelů, práce s databází (pouze MySQL), uživatelská nastavení nebo nastavení stránek, vkládání CSS a javascriptových souborů do hlavičky stránek, tvorbu různých jazykových verzí, zpracování asynchronních požadavků a další.

Dále díky použití konvencí pro umístění a jména PHP tříd, souborů a adresářů usnadňuje programátorovi načítání tříd a zároveň načítá pouze ty třídy, které vytvářená aplikace skutečně využívá. Obsahuje také třídy pro usnadnění tvorby uživatelského rozhraní, tzn. především třídy pro tvorbu administračních částí webů (anglicky backend) a třídy generující HTML formuláře (včetně prvků dodaných knihovnou jQueryUI) a automatizující jejich propojení s databází.

Tato kapitola nemá být komplexním popisem či dokonce dokumentací tohoto frameworku, dává si za cíl pouze stručně popsat jeho strukturu a přístup k tvorbě webových stránek do takové

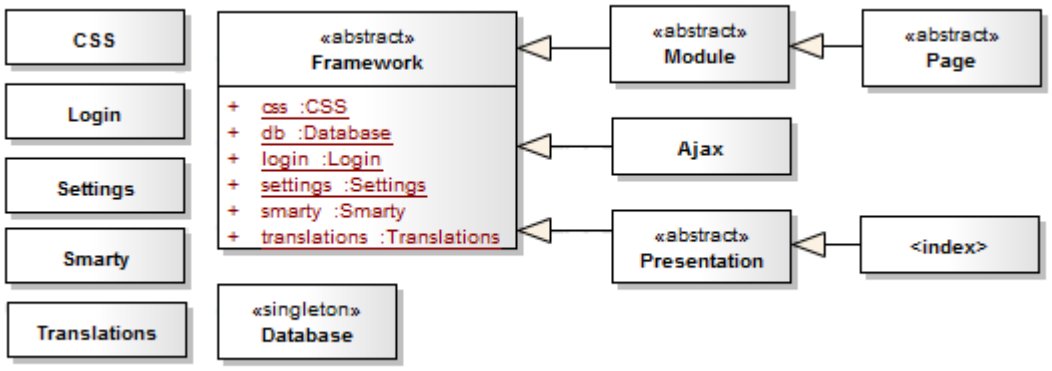
míry, aby byly tyto informace relevantní pro implementaci webové aplikace, která je výstupem této práce, a zároveň bylo možné se na ně odkazovat v dalším textu.

Základní struktura

Základem celého frameworku je třída *Framework*, která je předkem téměř všech ostatních tříd, které jsou buď přímo součástí frameworku nebo projektu na něm vytvářeného. Jejím účelem je poskytnout přístup k jiným třídám a inicializovat ve formě statických proměnných základní objekty sloužící pro přístup k databázi (třída *Database*), zprávu CSS souborů (*CSS)*, autentizaci a autorizaci uživatelů (*Login*) a překlady (*Translations*).

Kvůli oddělení logické a prezentační části aplikace je ve frameworku zahrnut šablonovací systém Smarty 3[4], jehož instance je rovněž přístupná všem potomkům třídy *Framework*. Nicméně každý objekt má svou vlastní instanci, do které jsou vždy přiřazeny základní proměnné a konstanty frameworku, jako je konfigurace, informace o přihlášeném uživateli, aktuální jazyková verze apod. Důvodem pro více než jednu instanci Smarty je potřeba oddělení jmenných prostorů šablon.

Výchozím bodem projektu je třída dědící od třídy *Presentation*, která na základě cesty v URL rozhodne, jaká stránka bude zpracována a zobrazena. Tato třída nese název projektu a nachází se v souboru *index.php* nebo v jiném v závislosti na nastavení serveru či specifikaci projektu. Úkolem třídy *Presentation* je zavolat metodu *GetHTML()* zpracovávané stránky a výsledný HTML kód přiřadit do Smarty šablony, která obsahuje kód pro kostru (layout) stránky. V případě webové aplikace tvořené v této práci je to šablona *presentation.tpl* v adresáři *templates* v kořenovém adresáři projektu.



Obrázek 11: **Diagram hierarchie tříd** použitého frameworku.

Typy tříd a objektů

Framework rozlišuje tři typy tříd na základě toho, od jaké základní třídy dědí. Těmito typy jsou stránky (odvozené od třídy *Page*), moduly (odvozené od třídy *Module*) a klasické třídy (u nichž dědění z jiné třídy není podmínkou).

Moduly představují většinou samostatně fungující části stránek (např. přepínač jazyků nebo seznam), které jsou do stránky vkládány pouze při splnění nějaké podmínky nebo jsou použity na různých místech webu a je proto žádoucí, aby jejich implementace byla vyčleněna do samostatného souboru a třídy. Není však vyloučena vzájemná závislost mezi moduly a jeden modul může využívat modulu jiného.

Moduly jsou načítány metodou *Framework::LoadModule()*, která se pokusí vyhledat třídu modulu v předem daných umístěních. Těmi jsou adresáře *modules* a *framework/modules* (vyhledávání v adresářích probíhá v tomto pořadí, aby bylo pro potřeby konkrétního projektu možné

[4] http://www.smarty.net/

znovu implementovat modul, který již je obsažen v modulech frameworku). Soubor modulu musí být pojmenován podle adresáře, v němž se nachází a mít předponu *module.* Např. pokud je modul uložen v adresáři *modules/example-module*, musí jméno souboru být *module.example-module.php.* Název třídy modulu se řídí rovněž podle názvu adresáře, nicméně je zapsán notací camelCase a začíná se předponou *Module*, např. *ModuleExampleModule.*

Stránky naproti tomu představují obsah unikátní v rámci projektu. Zjednodušeně lze říct, že jedné stránce odpovídá jedna cesta v URL (část URL bezprostředně za doménou a číslem portu). Jsou načítány, umístěny a pojmenovány analogicky k modulům. Uloženy jsou v adresářích *pages* a *framework/pages* a jejich soubory mají předponu *page* (předponu názvu třídy tvoří *Page*). Načítání probíhá pomocí metod *Framework::LoadPage().*

Klasické třídy jsou knihovny fungující samostatně a zastávají spíše funkce jako je manipulace se soubory a souborovým systémem nežli generování HMTL kódu. Jsou v adresářích *classes* a *framework/classes* s předponou *class* (název třídy předchází předpona *Class*). Pro načtení třídy slouží metoda *Framework::LoadClass().*

Stránky a moduly by měly implementovat alespoň metodu *GetHTML*, která vrací (X)HTML kód stránky nebo modulu. Pro vytvoření tohoto kódu lze využít Smarty, jehož instance je vytvořena v konstruktoru každé stránky či modulu. Metoda *FetchTemplate* automaticky zpracuje šablonu uloženou v adresáři *_templates* ve stejném adresáři jako je umístěna stránka nebo modul a vrátí vygenerovaný kód. Pokud metodě není předán parametr, je zpracována šablona se stejným jménem, jako je název adresáře modulu nebo stránky, a příponou *.tpl*, např. pro modul v adresáři *example-module* je to soubor *example-module.tpl*.

Asynchronní požadavky

Asynchronní požadavky je možné zpracovávat pomocí třídy *Ajax* v souboru *framework/class.ajax.php*, která přijímá dva různé parametry předávané HTTP metodou GET a to parametry *page* a *module*. Pokud jsou přítomny oba tyto parametry, má parametr *module* prioritu a parametr *page* je ignorován. Na základě parametru je vytvořen objekt stránky nebo modulu a je zavolána jeho metoda *GetAJAX()*, která v závislosti na své implementaci může vracet HTML kód, prostý text, XML, JSON nebo také nic. Pokud metoda *GetAJAX()* přijímá parametry, je nutné je také specifikovat a předat metodou GET jako pole *arguments* (položky tohoto pole jsou pak ve stejném pořadí jako v poli předány volání metody *GetAJAX()*). Vrácená hodnota je vytištěna na výstup. V kořenovém adresáři aplikace je umístěn soubor *ajax.php*, který do sebe zahrnuje soubor *framework/class.ajax.php* a umožňuje tak zkrátit adresu souboru, kterému jsou asynchronní požadavky předávány. Např. pokud by byl požadavek předán na URL

```
/ajax.php?module=example-module&arguments[]=prvni&arguments[]=2
```

byla by výstupem tohoto požadavku hodnota vrácená voláním metody *GetAjax("prvni", 2)* objektu třídy *ModuleExampleModule* uložené v souboru *modules/example-module/module.example-module.php*.

Autentizace a autorizace

Přihlašování, odhlašování a kontrolu práv přihlášeného uživatele má na starost třída *Login*. Data uživatelů jsou ukládána do několika tabulek databáze, z nichž základ tvoří tabulka *users*. Každý uživatel má přiřazenou roli (atribut *role_id* odkazující do tabulky *roles*), každá role má přiřazena určitá práva (tabulka *rights* a vazebná tabulka *roles_rights*). Pokud má uživatel přiřazenu roli s identifikátorem 0, jedná se o superuživatele, který má přístup do jakékoliv části aplikace a je schopen měnit práva všech ostatních uživatelů. Takovému uživateli říkejme *vlastník aplikace*. Protože webová

aplikace vytvářená v této práci využívá autentizace a autorizace pouze pro přístup do administrační části, je potřeba pouze jediný uživatel, který je vlastníkem aplikace.

Formuláře

HTML formuláře jsou vytvářeny převážně definicí přímo v jazyce PHP za pomoci třídy *Forms* a všech ostatních tříd přítomných v podadresáři *framework/forms*. HTML kód každého prvku formuláře je generován jinou třídou v adresáři *framework/forms/elements*. Důvodem, proč formuláře nejsou zapisovány přímo v jazyce HTML je častá nutnost ukládání a načítání jejich vstupů do nebo z databáze. Tento na chyby náchylný proces třída *Forms* téměř automatizuje. Druhou výhodou je možnost snadno definovat pokročilé prvky formuláře využívající pro svou činnost javascript. Takové prvky pak lze znovu používat v jakémkoliv dalším formuláři bez nutnosti psát jejich kód znovu.

Třída *Forms* je využita nejenom v administrační části aplikace, ale také ve formulářích tvořících uživatelské rozhraní filtrů v prohlížeči výsledků.

Jazykové verze

Textové řetězce každé stránky nebo modulu je možné načítat ze samostatného souboru nebo z databáze. To dovoluje vytvářet stránky v různých jazykových mutacích. Tyto soubory jsou uloženy v podadresáři *_translations* každé stránky nebo modulu a mají formát konfiguračních .ini souborů. O jejich načtení se stará metoda *Module::LoadTranslation()*, která načte řetězce do objektu *t*, jehož každá vlastnost (property) obsahuje jeden řetězec. Soubory překladů jsou pojmenovány stejně jako příslušný modul nebo stránka, navíc však v názvu obsahují i tečkou oddělený mezinárodní kód jazyka, jehož překlad obsahují. Anglický překlad modulu *ExampleModule* tak bude uložen v souboru *example-module.eng.ini*. Příklad dvou překladů lze vidět na obrázku 12.

```
;cze (komentar)                ;eng
string_1 = "Řetězec 1"         string_1 = "String 1"
string_2 = "Řetězec 2"         string_2 = "String 2"
```

Obrázek 12: **Příklad dvou souborů překladů** obsahujících český a anglický překlad dvou textů.

Třída *Module* implementuje metodu *Module:t()*, která umožňuje přístup do objektu *t* a přijímá dva parametry, kde první je identifikátor řetězce (odpovídá přesně názvu vlastnosti objektu) a druhý je výchozí překlad, který je použit v případě, že vlastnost není v objektu nalezena. Také díky ní lze jednoduše pomocí regulárního výrazu naleznout všechna místa v kódu, kde je překlad vkládán a soubor s překladem automaticky vygenerovat a použít jako základ pro překlad do dalších jazyků. Funkce se stejným účelem pojmenovaná rovněž *t* je přiřazena i do Smarty.

Záložky

Mechanismus záložek je implementován v jQueryUI Tabs[5], nicméně jejich chování je mírně upraveno / rozšířeno pro vyšší uživatelský komfort. Obsah záložek je načítán dynamicky pomocí asynchronních požadavků s výjimkou případu, kdy je načítána celá stránka, tj. při prvním načtení nebo po obnovení stránky. V takovém případě je aktuální záložka načtena přímo se zbytkem stránky, aby nebylo nutné čekat na vyřízení dvou HTTP požadavků, což s sebou zároveň nese také nechtěný efekt "probliknutí" stránky, kdy je dynamicky načtený obsah vkládán do prakticky prázdného dokumentu. Toto platí i pro víceúrovňové záložky použité v administraci.

Informace o poslední zvolené záložce je uchovávána pomocí souboru cookie, do nějž je uložen index zvolené záložky. Pomocí zásuvného modulu jQuery History je zajištěno očekávané chování prohlížeče při přechodu vpřed a zpět v historii prohlížení, tzn. že přepnutí záložky vytvoří záznam v

[5] http://jqueryui.com/demos/tabs/

historii prohlížeče a přechod zpět a vpřed pak způsobí přechod na předcházející nebo následující záložku.

Pokud má některá stránka obsahovat záložky, je nutné, aby namísto od třídy *Page* dědila od třídy *PageTabs*, která implementuje všechny výše zmíněné úpravy chování. Tato třída dále zavádí konvenci pro umístění stránek implementujících obsah záložek do adresáře *_tabs* umístěného ve stejném adresáři jako je třída stránky se záložkami.

Nastavení

Framework implementuje prostředky pro ukládání nastavení jednotlivých uživatelů vytvořených projektů nebo nastavení týkajících se přímo chování daného projektu. Obě tyto varianty jsou implementovány shodně a to ve třídě *Settings* v souboru *framework/class.settings.php*. Tato třída obsahuje pouhé tři metody *get(), set()* a *remove()*. Tyto metody na základě předaných parametrů s identifikátorem nastavení a uživatele načtou, uloží nebo odstraní záznam o nastavení z databáze. Tyto záznamy jsou uchovávány v tabulce *users_settings*. Pokud je jako identifikátor uživatele použita nula, je toto nastavení bráno jako nastavení aplikace.

Použité moduly a třídy

Zde jsou stručně popsány nejdůležitější moduly a třídy použité při implementaci webové aplikace. Popis se opět omezuje pouze na vlastnosti, které byly při implementaci využity.

List - pomocí tohoto modulu je implementován prakticky celý prohlížeč výsledků analýz. Modul jako parametr přijímá SQL dotaz, pomocí nějž jsou z databáze načítána data, která jsou modulem zobrazena ve formě seznamu se stránkováním a řazením. Tyto dvě operace načítají nová data asynchronně, aby nedocházelo k neustálému znovunačítání celého dokumentu. Kromě prostého zobrazení dat ve formě tabulky lze definovat Smarty šablonu představující jeden řádek seznamu a data tak zobrazovat v podstatě libovolným způsobem.

Seznamu lze rovněž definovat tlačítka provádějící akce nad zvoleným řádkem nebo řádky. Předdefinována jsou tlačítka pro mazání a odkazování do formuláře umožňujícího daný řádek seznamu upravit.

Progressbar - Tento modul umožňuje zobrazit průběh dlouhotrvajících operací pomocí periodického zasílání asynchronních dotazů na server, kde je informace o průběhu uložena v souboru. O aktualizaci informací v tomto souboru se musí postarat metoda implementující danou dlouhotrvající operaci.

CSV Parser - tato třída parsuje CSV soubor zadaný ve formě textového řetězce do formy objektu. Po každém získání parametrem definovaného počtu řádku předá vytvořené objekty callback metodě (nebo metodám), která se stará o jejich další zpracování. Toto dávkové zpracování je zavedeno z důvodu paměťového limitu, který je definován pro jeden proces zpracovávající PHP skript. Bez zavedení tohoto způsobu rozsáhlejší CSV soubory, jako je například výstup programu QFinder při analýze lidského chromozomu 1, tento limit překračovaly, což vedlo k fatální chybě a předčasnému ukončení běhu skriptu.

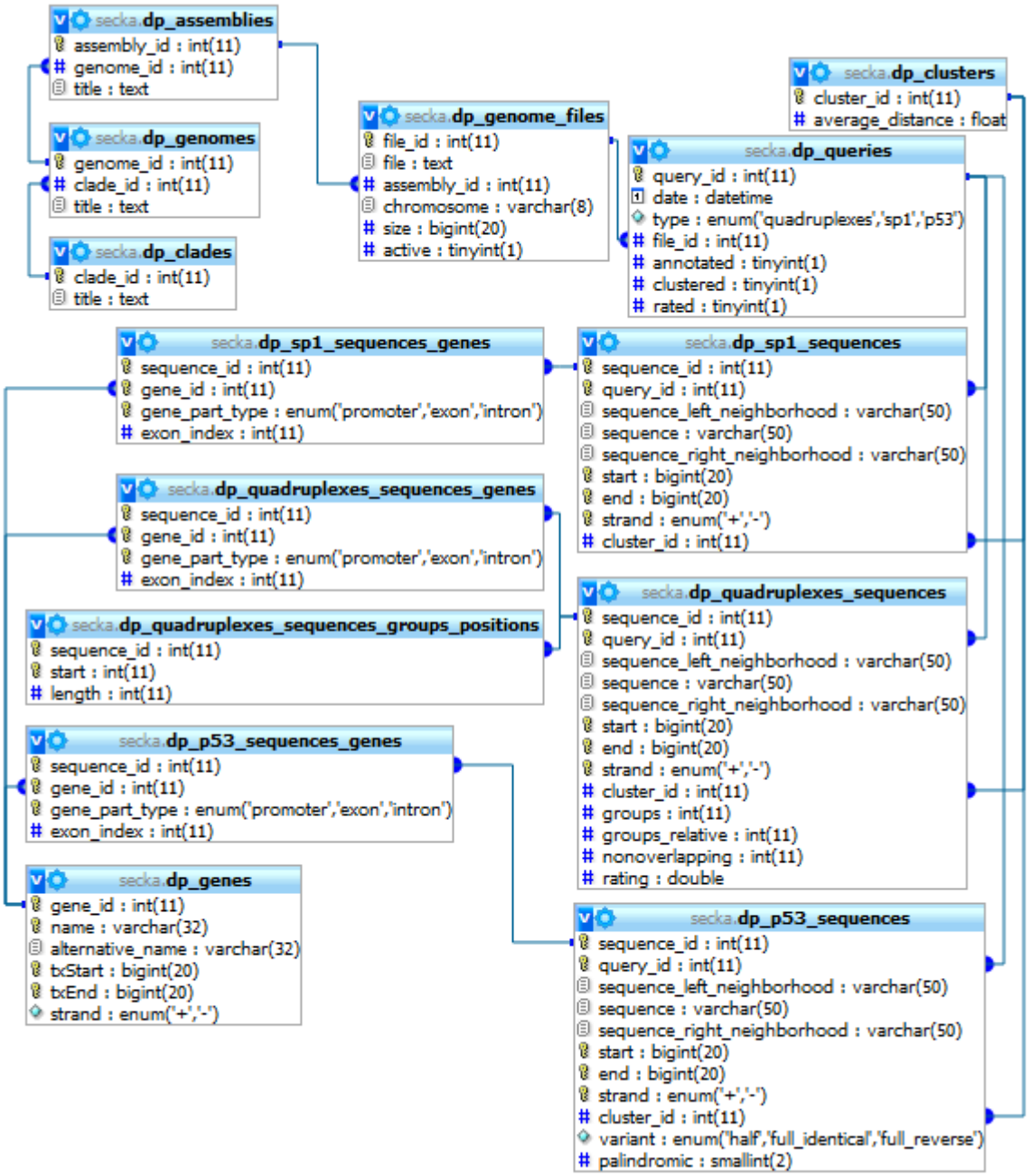
User settings - umožňuje definovat formulář pro úpravu nastavení uživatele či celé aplikace. K ukládání a načítání nastavení využívá prostředky frameworku implementované v třídě *Settings.*

Databáze

Webová aplikace používá pro ukládání perzistentních dat databázovou technologii MySQL. Částečné schéma použité databáze (pouze tabulky s daty o genomech a analýzách) lze vidět na obrázku 13 (názvy tabulek budou v dalším textu uvedeny bez předpony *dp_*).

Kategorizace souborů genomů je modelována tabulkami *clades*, *genomes*, *assemblies* a *genome_files*. Záznamy o analýzách provedených nad jednotlivými soubory jsou uloženy v tabulce

queries. Sloupec *type* udává typ analýzy provedené nad souborem *file_id* a společně tyto dva atributy tvoří unikátní index, což znamená, že nad každým souborem lze provést pouze jednu analýzu od každého typu. Sloupce *annotated*, *clustered* a *rated* obsahují hodnoty typu boolean (0 nebo 1) a obsahují informaci o provedení dodatečných analýz nad sekvencemi nalezenými příslušnou analýzou sekvence chromozomu.



Obrázek 13: **Částečné schéma databáze** webové aplikace. Zobrazené vazby mezi tabulkami představují integritní omezení.

Samotné výsledky analýz jsou rozprostřeny mezi tabulky, jejichž název končí *sequences*, *sequences_genes*, a tabulku *quadruplexes_groups_positions*. První jmenované obsahují především informace o pozicích každé sekvence nalezené během analýzy a informaci o shluku, do kterého sekvence náleží. V závislosti na délce nálezu mohou obsahovat i samotnou nalezenou sekvenci a její okolí. Tabulky končící *sequences_genes* slouží jednak jako vazební tabulky mezi záznamy o sekvencích a záznamy o genech, do kterých spadají, ale zároveň obsahují i další informace získané

anotací nalezených sekvencí - typ části genu, do které sekvence náleží (*gene_part_type*) a případný index exonu (*exon_index*). V tabulce *quadruplexes_groups_positions* jsou obsaženy pozice G skupin v kvadruplexech, které jsou součástí výstupu programu QFinder.

Důvodem pro rozdělení tabulky pro sekvence a jejich anotaci na více téměř identických tabulek byl jednak fakt, že počet záznamů v nich se pohybuje v řádech statisíců a rozdělení tak značně urychluje práci s nimi, a také skutečnost, že tabulka s výsledky vyhledávání kvadruplexových sekvencí obsahuje atributy, které u ostatních analýz nejsou potřebné.
Data v tabulce *clusters* jsou záznamy o nalezených shlucích. U každého z nich je ve sloupci *average_distance* vypočítána průměrná vzdálenost mezi sousedními sekvencemi ve shluku. V tabulce *genes* se nalézají data o genech, ve kterých leží některá z nalezených sekvencích. Tato data jsou kopírována z tabulek získaných z databáze aplikace UCSC Genome Browser, kde jsou k dispozici stažitelné soubory tabulek její databáze[6].

Práce s databází

Protože různé třídy operují s databází podobným nebo stejným způsobem, bylo vhodné vyčlenit tyto operace do samostatných modulů tak, aby každý modul pracoval s jednou databázovou tabulkou. Toho samozřejmě vzhledem ke vztahové provázanosti mezi tabulkami nelze dosáhnout stoprocentně a toto rozdělení je pouze přibližné. Konkrétně se jedná o moduly, jejichž název končí slovem *Worker* a to *ModuleGenesWorker*, *ModuleGenomeBrowserWorker*, *ModuleGenomeFilesWorker*, *ModuleQuadruplexesWorker*, *ModuleQueriesWorker* a *ModuleSequencesWorker*. Poslední jmenovaný modul provádí operace nad všemi tabulkami databáze obsahujícími výsledky nějakého hledání, což jsou všechny tabulky, jejichž název končí *sequences*.

Uživatelské rozhraní

Aplikace v základu sestává ze dvou stránek. Těmi jsou **administrace** a **prohlížeč výsledků**. Obě tyto části jsou následně členěny pomocí záložek obsahujících další stránky. To umožňuje pracovat s aplikací pohodlněji, než kdyby při změně zobrazené sekce pokaždé docházelo k znovunačtení celého dokumentu a tím i odsunutí zobrazované oblasti stránky na začátek dokumentu.

Vytvořeny jsou dvě funkčně shodné jazykové verze - česká a anglická. Další text se bude odvolávat výhradně na českou verzi textových řetězců aplikace.

Administrace

Administrace je záložkami členěna na sekci Soubory genomu, Analýzy nalezených sekvencí a Nastavení.

Soubory genomu - seznam záznamů o souborech se sekvencemi chromozomů je implementován ve třídě *PageGenomeFilesList* (pokud nebude uvedeno jinak, jsou popisované metody obsaženy v této třídě), jejíž umístění relativně k adresáři se stránkami aplikace je *admin/_tabs/genomes/_tabs/genome-files-list*. Seznam má definována tři tlačítka pro provádění analýz nad soubory genomu, které se nachází v souborovém systému. Těmito analýzami jsou jednak vyhledání kvadruplexových sekvencí a navíc také hledání SP1 a P53 vazebných sekvencí. Každá z těchto analýz může být provedena nad jediným souborem nebo dávkově nad více soubory. Započetí analýzy zobrazí ukazatel postupu této analýzy, aby bylo, vzhledem k tomu, že analýza může trvat i několik minut, patrné, v jakém stádiu se analýza nachází. O jeho aktualizaci se stará metoda *setSearchProgress()*.

Asynchronní požadavek na vyhledání kvadruplexů je postupně předán metodě *searchForQuadruplexes()*, která voláním PHP funkce *shell_exec()* spouští aplikaci QFinder. Ta daný

[6] http://hgdownload.cse.ucsc.edu/downloads.html

soubor analyzuje a výsledky této analýzy jsou vráceny ve formátu CSV. Ten je zpracován třídou *CSV Parser* a takto získaná data jsou předána metodě *ModuleQuadruplexWorker::insertCSVLine()*, která data ukládá do databáze.

Čtyři záložky editorů slouží pro zadávání, úpravu a mazání metainformací o souborech genomů. Všechny formuláře v nich jsou generovány pomocí třídy *Forms*. **Editor kladů** pracuje se záznamy tabulky *clades*, **editor genomů** se záznamy tabulky *genomes*, **editor sestavení** se záznamy tabulky *assemblies* a **editor souborů genomu** pracuje se záznamy tabulky *genome_files* a umožňuje procházet a volit soubory genomu uložené v souborovém systému.

Analýzy nalezených sekvencí - tato sekce obsahuje záložku pro každý že tří typů vyhledatelných sekvencí. Všechny záložky jsou implementovány jedinou třídou *PageSequencesList*. Seznam na každé záložce obsahuje všechna provedená hledání daného typu a tři tlačítka umožňující spouštění dodatečných analýz nad jedním nebo více zvolenými hledáními dávkově.

Nastavení - tato část administrace je implementována třídou stránky *PageSettings* a všechna nastavení, která umožňuje měnit jsou načítána a ukládána zpět do databáze s využitím modulu frameworku *ModuleUserSettings*.

Prohlížeč výsledků

Protože aplikace dokáže vyhledat více typů sekvencí, je součástí prohlížeče přepínání mezi těmito typy sekvencí. Prohlížení všech typů sekvencí je však implementováno stejnými třídami.

Formulář pro výběr chromozomu, jehož výsledky mají být zobrazeny, je implementován za pomoci třídy *Forms* ve stránce *PageResults*. Na základě zvoleného typu sekvence a chromozomu formulář může směřovat na různá URL, jejichž formát je

```
... /typ-sekvence/id-klad-genom-sestavení-chromozom/
```

kde však pouze *typ sekvence* a *id* mají vliv na zobrazený obsah (zbylé údaje jsou v URL obsaženy pro její lepší čitelnost). *Id* je celočíselný identifikátor databázového záznamu o daném souboru chromozomu.

Zbytek prohlížeče je tvořen třemi záložkami této stránky pro prohlížení nalezených sekvencí, shluků a statistik týkajících se zvoleného chromozomu a typu sekvencí.

Nalezené sekvence - obsah záložky je implementován ve třídě stránky *PageSequences*. Vzhled a funkce seznamu s nalezenými sekvencemi jsou v případě zobrazení kvadruplexových sekvencí (výstupů programu QFinder) mírně odlišné než v případě zbylých dvou typů.

Seznam u kvadruplexů oproti jiným typům navíc obsahuje informaci o počtu G skupin, podle které je možné sekvence i filtrovat. Dále kvadruplexy jako jediné obsahují sekvence natolik dlouhé, že není možné je přehledně zobrazit celé. Sekvence delší než 30 znaků jsou proto zkráceny a zobrazení celé sekvence je možné po najetí kurzorem na ni. HTML kód s úplným řetězcem sekvence je generován a načten asynchronně modulem *ModuleSequenceDetail*, který sekvenci čte přímo ze souboru genomu v souborovém systému a to pomocí třídy *ClassGenomeFilesReader*. O samotné zobrazení HTML kódu se sekvencí se stará doplněk JQuery *qTip*[7].

U každé kvadruplexové sekvence je možné volitelně provést a zobrazit hloubkovou analýzu, která zobrazí veškerá možná umístění kvadruplexů v rámci nalezené sekvence. Po kliknutí na ikonku + přítomnou u každé zobrazené kvadruplexové sekvence je metodě *getDeepAnalysis()* třídy *PageSequences* předán asynchronní požadavek s identifikátorem sekvence a jejím typem. Tato metoda spouští pomocí PHP funkce *shell_exec()* program pro hloubkovou analýzu s parametry danými aktuálním nastavením aplikace. Stejně jako při rychlé analýze programem QFinder je i výsledek hloubkové analýzy nejdříve zpracován třídou *ClassCSVParser*. Nicméně výsledky hloubkové analýzy nejsou ukládány do databáze, ale pouze seřazeny sestupně podle jejich

[7] http://craigsworks.com/projects/qtip/

ohodnocení a zobrazeny. Protože u některých sekvencí z rychlé analýzy je možné najít až stovky umístění, je jejich počet při zobrazení omezen na nejlépe hodnocených dvacet.

Shluky - záložka zobrazující shluky je implementována ve třídě stránky *PageClusters*. Sekvence náležící do shluku tentokrát nejsou načítány asynchronně po kliknutí na ikonku +, ale jsou v dokumentu přítomny již ve chvíli, kdy je načten seznam shluků.

Statistiky - třída stránky implementující tuto záložku má název *PageStatistics*. Všechny zobrazené statistiky jsou počtem sekvencí určitých vlastností na daném chromozomu. Jako takové jsou všechny získány pomocí databázové agregační funkce COUNT.

Filtrace

Filtry seznamů s výsledky jsou implementovány pomocí modulu frameworku *ModuleFilter*, který je funkčně spojen s modulem *ModuleList* implementujícím samotné seznamy. Formuláře filtrů jsou generovány třídou *Forms*, nicméně výsledné formuláře oproti formulářům v administraci nijak nespolupracuje s databází. Namísto toho je obsah filtračního formuláře ukládán do COOKIE, aby bylo jeho nastavení k dispozici jak při obnovení celého dokumentu, tak i při změně řazení nebo stránky filtrovaného seznamu, kdy je na server zasílán asynchronní požadavek.

Většina položek filtru upravuje klauzuli HAVING SQL dotazů, který získává data zobrazovaná v seznamech. Tato klauzule narozdíl od klauzule WHERE umožňuje odkazovat se na data pomocí aliasů vytvořených v klauzuli SELECT.

Výjimku tvoří filtr maximální délky položky. Protože nalezené sekvence a geny, ve kterých se mohou nalézat, existují ve vztahu 1:m, musí být data o anotaci vybírána poddotazem. Tzn. že nastavení filtru se musí projevit přímo v tomto poddotazu, nikoliv v klauzuli HAVING hlavního dotazu. Zároveň to znamená, že tento filtr, pokud je použit samostatně bez filtrace podle anotace, neovlivňuje počet výsledků hledání, ale pouze data jejich anotace.

Propojení s aplikací Genome Browser

Pro zobrazení nalezené sekvence v aplikaci UCSC Genome Browser postačuje vytvořit jednoduchý hypertextový odkaz předávající HTTP metodou GET parametry určující zobrazenou část genomu a tracky. Formát URL tohoto odkazu je

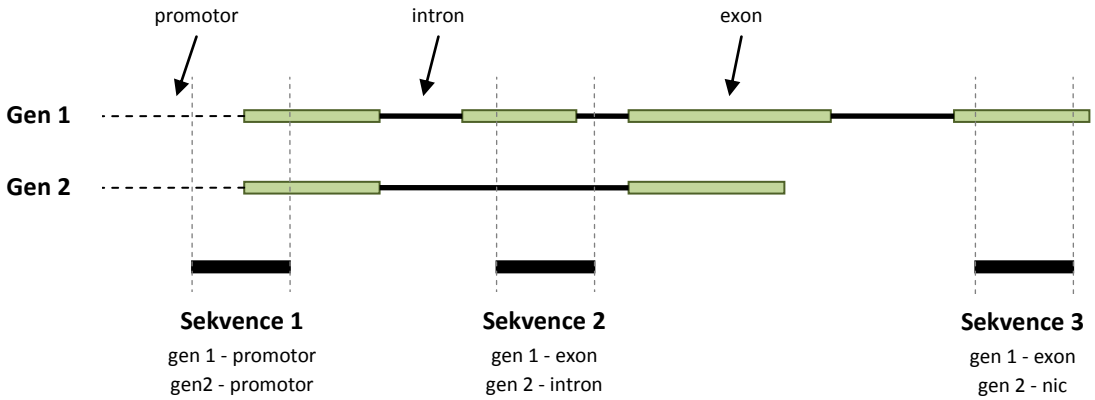
```
http://genome.ucsc.edu/cgi-bin/hgTracks?position=chromozom:od-
do&refGene=pack&knownGene=pack&mrna=pack&cons46way=full
```

přičemž *chromozom* je název chromozomu (u člověka chr1 až chr22, chrX a chrY) a parametry *od* a *do* určují rozmezí zobrazovaných dat na udaném chromozomu. Zbylé parametry udávají stopy, které má Genome Browser zobrazit. Kromě těchto stop mohou však být po načtení aplikace zobrazeny i jiné, protože Genome Browser si nastavení stop pamatuje a pokud není způsob jejich zobrazení udán v URL, zůstává nezměněn od posledního zobrazení.

Odkaz na vyhledané sekvenci zobrazí rozsah celé sekvence plus jejího krátkého okolí, tj. 10bp na obou koncích sekvence. Naproti tomu odkazy na genech nalezených anotací zobrazují přesně rozsah daného genu a stejně tak i odkazy na výsledcích hloubkové analýzy zobrazují pouze část sekvence, na které se nalezený kvadruplex formuje.

Anotace

Informace o poloze sekvence vůči genům daného chromozomu je pro posouzení významu této sekvence klíčová. Aplikace je proto schopná na základě informací o pozicích a struktuře známých genů anotovat nalezené sekvence informací nejen o umístění vůči genům, ale i v rámci genů. O každé nalezené sekvenci tak je známo, nachází-li se zcela mimo gen, v promotoru genu nebo v jeho intronu či exonu.



Obrázek 14: **Anotace nalezených sekvencí**.

Za promotorovou oblast je považován úsek chromozomu bezprostředně předcházející začátek transkripce genu. Délka tohoto úseku je při analýze nastavena na deset tisíc bází, nicméně tento údaj je možné nastavením filtru snížit až na nulu.

Pokud anotovaná sekvence zasahuje do více než jedné části genu, pak je anotována pouze jedna z těchto částí a to prioritně v pořadí promotor, exon, intron.

Anotace je implementována v modulu *SequencesWorker* v metodě *AnnotateQuerySequences()*. Tato metoda přijímá jako parametr identifikátor hledání (hodnotu atributu *queryId* z databázové tabulky *queries*) a anotuje všechny sekvence nalezené v tomto hledání. Způsob anotace je ukázán v algoritmu 1.

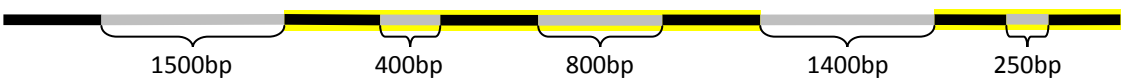
```
// pro každý výsledek daného hledání získaný z databáze
sequences = getSequencesFromDB(queryId);
foreach (sequences as sequence) {
   // vybere z databáze takové geny (tj. název, začátek a konec transkripce a
   // začátky a konce exonů), pro které platí, že začátek nebo konec sekvence
   // výsledku hledání leží v oblasti mezi jejich začátkem a koncem transkripce.
   // Navíc vypočte podle maximální délky promotoru a vlákna genu pozice začátku a
   // konce promotorů těchto genů.
   genes = getGenesFromDB(sequence.start, sequence.end);
   // pro každý nalezený gen
   foreach (genes as gene) {
      // pokud začátek nebo konec sekvence výsledku leží v oblasti promotoru
      if (
         sequence.start within gene.promoterStart, gene.promoterEnd ||
         sequence.end within gene.promoterStart, gene.promoterEnd
      ) {
         // uloží do databáze informaci, že sekvence leží v promotoru genu
         setSequenceInPromoter(sequence);
      }
      // jestli začátek nebo konec sekvence výsledku leží v oblasti některého exonu
      else if (
         sequence.start within gene.exonStarts, gene.exonEnds ||
         sequence.end within gene.exonStarts, gene.exonEnds
      ) {
         // uloží do databáze informaci, že sekvence leží v exonu genu a jeho index
         setSequenceInExon(sequence, exonIndex);
      }
      else {
         // ulož do databáze informaci, že sekvence leží v intronu genu
         setSequenceInIntron(sequence);
      }
   }
}
```

Algoritmus 1: **Anotace výsledků hledání** informací o umístění sekvence vůči genům.

Pozice začátku a konce promotoru genu závisí na nastavené maximální délce promotoru a vláknu dvoušroubovice, na kterém se daný gen nachází. Pokud je gen na kladném vláknu, pak je promotor umístěn před pozicí začátku transkripce genu. V opačném případě, kdy je gen umístěn na záporném vlákně se promotor nachází za koncem transkripce tohoto genu.

Shluková analýza

Shluková analýza znamená nalezení skupin sekvencí, které spolu potenciálně mohou funkčně souviset. Za shluk je považována množina sekvencí, ve které žádné dvě sousední sekvence nejsou od sebe vzdáleny více než daný počet bází (tento počet je možné nastavit v administraci aplikace).



Obrázek 15: **Tvorba shluků** při maximální vzdálenosti sousedních sekvencí 1000 bází. Černé úseky představují nalezené sekvence, žlutě podbarvené části jsou shluky. Nejlevější sekvence není součástí žádného shluku.

Stejně jako v případě anotace je i shluková analýza implementována v modulu *SequencesWorker*, tentokrát v metodě *searchForQueryClusters()* přijímající jako parametr opět identifikátor hledání. Průběh hledání shluků a přiřazení sekvencí do shluků je popsán algoritmem 2.

```
// pro každý výsledek daného hledání získaný z databáze
sequences = getSequencesFromDB(queryId);
foreach (sequences as sequence) {
   // získej z databáze identifikátory všech sekvencí, jejichž začátek je roven
   // nebo leží za začátkem sekvence zkoumaného výsledku a zároveň není od konce
   // zkoumané sekvence dále než nastavený počet bází
   ids = getIdsOfSequencesInVicinity(sequence);

   // pokud byly získány alespoň 2 identifikátory (tj. alespoň jeden, který
   // neidentifikuje samotnou zkoumanou sekvenci)
   if (count(ids) >= 2) {
      // pokud zkoumaná sekvence již patří do některého shluku
      if (inCluster(sequence)) {
         // zařaď všechny sekvence nalezené v okolí do tohoto shluku
         foreach (ids as id) {
            setCluster(id, getCluster(sequence));
         }
      }
      else {
         // vytvoř nový shluk
         clusterId = createCluster();

         // zařaď všechny sekvence nalezené v okolí do tohoto shluku
         foreach (ids as id) {
            setCluster(id, clusterId);
         }
      }
   }
}
```

Algoritmus 2: **Shluková analýza nalezených sekvencí.**

4.2 Hledání kvadruplexových sekvencí

4.2.1 Specifikace

Vyhledávání kvadruplexů bude zčásti využívat algoritmů vytvořených v rámci bakalářské práce Jiřího Němce v roce 2010 [11]. Zůstane tedy zachováno rozdělení analýzy na dvě fáze. Pro první fázi rychlého vyhledání sekvencí, které mohou obsahovat kvadruplexy bude použit přímo program QFinder, kterému bude pouze upraven formát výstupu. Program AllQFinder použit nebude, protože jeho algoritmus využívá pro vyhledání možných umístění kvadruplexu ve vstupní sekvenci hrubé síly a zbytečně generuje celou množinu konfigurací kvadruplexového vzorce (1) a testuje postupně všechna možná zarovnání všech jeho prvků se vstupní sekvencí. Součástí této práce je proto i navržení efektivnějšího algoritmu schopného nalézt všechna možná umístění kvadruplexů rychleji.

Změny v programu QFinder

Díky prezentaci výsledných informací v podobě webové stránky odpadá nutnost mít informace ve výstupech snadno čitelné člověkem. Naopak účelná je snadná strojová zpracovatelnost, aby data mohla být jednoduše uložena do databáze. Z těchto dvou důvodů byl pro výstup zvolen formát CSV, původní formát je však v programu zachován a CSV výstup musí být zapnut pomocí přepínače *-c*. Vstupem programu zůstane soubor ve formátu FASTA.

Každý řádek výstupního CSV souboru představuje jeden úsek některé vstupní sekvence (pro účely vytvářené webové aplikace je tato sekvence vždy jediná a to sekvence zahrnující celý chromozom organismu), obsahující jednu nebo více kvadruplexových sekvencí. Formát řádku tohoto souboru vypadá následovně:

```
header;base;position;length;g_groups;g_groups_relative;nonoverl
apping;groups_positions;sequence_left_neighborhood;sequence;seq
uence_right_neighborhood
```

kde *header* je přesnou kopií FASTA hlavičky příslušné vstupní sekvence, *base* je báze tvořící tetrády (G nebo C v případě komplementárního řetězce), *position* je pozice začátku nalezené sekvence v analyzované sekvenci, *length* je délka nalezené sekvence, *g_groups* je počet G skupin v nalezené sekvenci, *g_ groups_relative* je počet G skupin v nalezené sekvenci se započtením G skupin, které se mohou vyskytovat v jedné nepřetržité sekvenci guaninů (např. skupina GGGGGGGG obsahuje dvě takové skupiny), *nonoverlapping* je počet nepřekrývajících se kvadruplexů v nalezené sekvenci (je možné si je představit jako počet dílů, na které se dá výsledná sekvence rozdělit, aby tvořily nezávislé kvadruplexy [11]), *g_positions* je pozice a délka všech G skupin v nalezené sekvenci, *sequence_left_neighborhood* a *sequence_right_neighborhood* je krátký úsek vstupní sekvence předcházející a následující za nalezenou kvadruplexovou sekvenci a konečně *sequence* je samotná nalezená sekvence.

Rozhraní programu pro hloubkovou analýzu

Stejně jako v případě programu QFinder se bude jednat o konzolovou aplikaci. Kromě vstupní sekvence program pro svou práci bude potřebovat informace o parametrech, které byly původně předány programu QFinder. Těmi jsou minimální a maximální délka smyčky, minimální a maximální délka sekvencí tvořících tetrády a báze tvořící tetrády (nejobvykleji G nebo C v závislosti na vlákně sekvence).

Význam	**Krátký přepínač**	**Dlouhý přepínač**	**Výchozí hodnota**	**Povinný**	**Vyžaduje argument**
Minimální délka G-skupiny	-m	-l1	1	Ne	Ano
Maximální délka G-skupiny	-n	-l2	7	Ne	Ano
Minimální délka smyčky	-o	-g1	3	Ne	Ano
Maximální délka smyčky	-p	-g2	5	Ne	Ano
Vstupní sekvence	-s	---	---	Ano	Ano
Báze G-skupiny	-b	---	G	Ne	Ano
Vypsat pouze nejlepší hodnocení	-r	---	---	Ne	Ne
Nápověda	-h	---	---	Ne	Ne

Tabulka 1: **Parametry příkazové řádky** programu pro hloubkovou analýzu potenciálních kvadruplexových sekvencí a jejich výchozí hodnoty.

Pro spuštění algoritmu bude vyžadováno pouze zadání sekvence. Při vynechání jednoho či více ze zbývajících pěti jmenovaných parametrů použije program výchozí hodnoty, které jsou odvozeny ze vzorce 5. Podrobněji všechny možné přepínače a jejich význam popisuje tabulka 1.
Výstup programu bude opět ve formátu CSV a každý řádek výstupu bude mít formát

```
tetradeCount;rating;groupStarts;sequence
```

kde *tetradeCount* je počet tetrád, které nalezený kvadruplex vytváří, *rating* je hodnocení kvadruplexu, *groupStarts* jsou indexy začátků skupin bází tvořících tetrády oddělené symbolem | a *sequence* je vstupní sekvence, ve které je struktura kvadruplexu vyznačena oddělením smyček a skupin bází

tvořících tetrády symbolem |. Druhým možným výstupem je vypsání pouze nejvyššího ohodnocení ze všech nalezených kvadruplexů. Tento mód se zapíná pomocí přepínače *-r* a bude využit webovou aplikací při hodnocení výstupních sekvencí programu QFinder.

Hodnocení kvadruplexu bude spočteno na základě počtu tetrád a průměrného rozdílu délek smyček podle vzorce

$$\text{hodnocení} = \text{počet tetrád} / (\text{průměrný rozdíl délek smyček} + 1) \qquad (6)$$

4.2.2 Algoritmus programu pro hloubkovou analýzu

Algoritmus je založen na hledání cest o vrcholové délce 4 v acyklických orientovaných grafech vytvořených na základě vstupní sekvence a vstupních parametrů kvadruplexového vzorce. Každý z vrcholů této cesty představuje jednu ze čtyř skupin, jejichž báze tvoří tetrády kvadruplexu.

Pro účely dalšího popisu nazývejme **multiskupinou** posloupnost bází tvořících tetrády (G nebo C) takové délky, že pro daný počet tetrád výsledného kvadruplexu může tato posloupnost být rozdělena na dvě a více menších podsekvencí o délce rovné tomuto počtu tetrád a oddělených takovým počtem bází, aby byl větší nebo roven minimální velikosti mezery. Zmíněné menší podsekvence multiskupin nazývejme **podskupiny.** Množinu posloupnosti, které podmínky pro multiskupinu nesplňují nazývejme **skupinami** a její podmnožinu posloupnosti o délce přesně rovné požadovanému počtu tetrád zvěme **minimálními skupinami.**

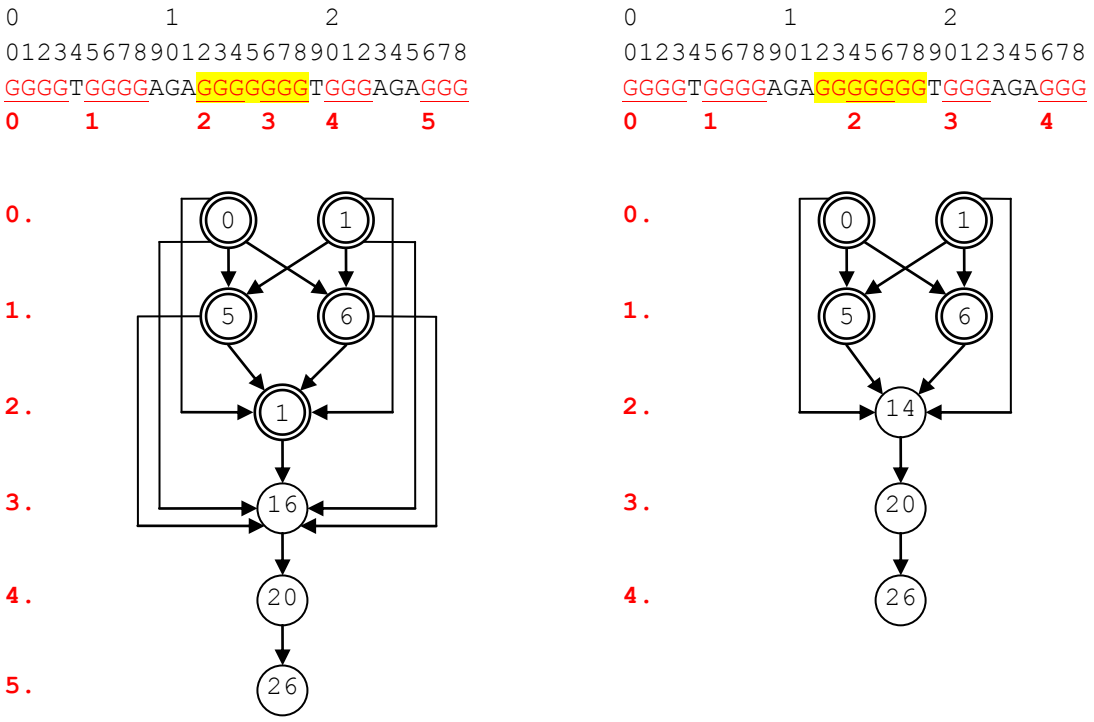
Počet tetrád = 3
GGGGGGG – multiskupina
GGGGG – skupina
GGG - minimální skupina

Počet tetrád = 5
GGGGGGG - skupina
GGGGG - minimální skupina
GGG - netvoří skupinu

Obrázek 16: **Ukázka označení posloupností bází** tvořících tetrády kvadruplexu pro dva různé počty tetrád. Podtržené části skupin ilustrují možné umístění podsekvence bází tvořících tetrády ve výsledném kvadruplexu. Zobrazená multiskupina obsahuje dvě podskupiny.

Pro jednoduchost zatím uvažujme sekvenci s jedinou multiskupinou. Vrcholy prohledávaného grafu jsou tvořeny pouze skupinami a minimálními skupinami. Multiskupiny jsou před vytvořením grafu převedeny na skupiny a graf je pak vytvořen pro každé možné umístění podskupin v rámci multiskupiny neboli **konfiguraci multiskupiny.**

Graf pro každou konfiguraci multiskupiny je sestaven po úrovních tak, že každá úroveň obsahuje vrcholy představující veškerá možná umístění podsekvencí jedné skupiny o délce odpovídající počtu tetrád. Těmto umístěním říkejme **konfigurace skupiny.** Hrany grafu pak mohou existovat pouze mezi vrcholy dvou úrovní, nikdy uvnitř úrovně. Hrany jsou vytvořeny spojením všech vrcholů každé úrovně se všemi vrcholy následující úrovně a každé další úrovně, za kterou následují alespoň ještě dvě úrovně.



Obrázek 17: **Ukázka grafů** pro vyhledání kvadruplexů se třemi tetrádami a dvě různé konfigurace žlutě podbarvené multiskupiny (tato multiskupina jich má celkem 6 a graf musí být sestrojen pro každou z nich). Čísla v uzlech odpovídají indexům znaků v řetězci sekvence. Podtržené části sekvence představují skupiny dané konfigurace. Algoritmus graf prochází z každého uzlu, ze kterého je možné vytvořit cestu vrcholové délky 4. V grafech jsou tyto uzly vyznačeny dvojitou čarou.

Ve všech grafech vytvořených pro každou možnou konfiguraci multiskupin v analyzované sekvenci jsou následně vyhledány všechny cesty o vrcholové délce 4. Výchozími uzly pro tyto cesty jsou všechny uzly grafu s výjimkou uzlů posledních tří úrovní.

Ve složitějším případě, kdy prohledávaná sekvence obsahuje více než jednu multiskupinu, je nutné vytvořit a prohledat graf pro každý prvek kartézského součinu množin konfigurací všech multiskupin. Prvku výsledné množiny říkejme **konfigurace vstupní sekvence**.

Obrázek 18: **Vstupní sekvence s více multiskupinami.** Graf musí být vytvořen pro každou kombinaci konfigurace první a druhé multiskupiny (neboli konfiguraci vstupní sekvence) - celkem 36 možností.

Základ algoritmu

Program implementující algoritmus pro hloubkovou analýzu výstupní sekvence programu QFinder je napsán v jazyce C++. Základní kostra programu, je ukázána v algoritmu 3.

```
// pro každý možný počet tetrád n, jehož rozsah je dán vstupními (1)
for (n = minGroup; n <= maxGroup; ++n) {
   // zjisti které skupiny obsahují alespoň n bází (2)
   relevantGroups = getRelevantGroups(groups, n);

   // zjisti, které z těchto skupin jsou multiskupinami
   // a zároveň vygeneruj všechny její konfigurace (3)
   multiGroups = getMultigroups(relevantGroups);

   // zjisti maximalní počet skupin v konfiguraci sekvence
   // a pokud existuje dostatečný počet skupin, aby tvořily kvadruplex (4)
   if (getMaxGroupsCount() >= 4) {
      // dokud existuje další konfigurace sekvence (vektor skupin) (5)
      while (sequenceConfig = getNextSequenceConfig()) {
         // pro každou skupinu v konfiguraci
         for (i = 0; i < sequenceConfig.size(); ++i) {
            // vypočti všechny konfigurace skupiny (6)
            sequenceConfig[i].configs = getGroupConfigs(sequenceConfig[i]);
            // vytvoř seznam skupin, které mohou následovat po této (6)
            sequenceConfig[i].next = getPossibleNextGroups(sequenceConfig);
         }

         // pro každou skupinu z této konfigurace vstupní sekvence, která může
         // tvořit začátek výsledného kvadruplexu
         lastStartGroupIndex = sequenceConfig.size() - 4;
         for (startGroup = 0; startGroup <= lastStartGroupIndex; ++startGroup) {
            // nalezni a vypiš všechny takové kvadruplexy (7)
            findQuadruplexes(sequenceConfig, startGroup);
         }
      }
   }
   else exit();
}
```

Algoritmus 3: **Hloubková analýza kvadruplexových sekvencí**, které jsou výstupem programu QFinder.

Jednotlivé body algoritmu rozebereme podrobněji:

ad 1) V závislosti na vstupních parametrech udávajících minimální a maximální počet tetrád tvořících hledané kvadruplexy je hledání provedeno několikrát, jednou pro každý povolený počet tetrád.

ad 2) Protože jednotlivé skupiny ve vstupní sekvenci mohou mít různou délku (zdola omezenou minimálním počtem tetrád), jsou z těchto skupin vyfiltrovány ty, které mají délku menší než je aktuální *n*.

ad 3) Algoritmus identifikuje ve zbylých skupinách multiskupiny pro aktuální *n* a vygeneruje a uloží si všechny jejich konfigurace (viz dále).

ad 4) Pokud součet maximálních počtů podskupin všech multiskupin je alespoň 4, existuje šance, že mohou tvořit kvadruplexy a algoritmus pokračuje. V opačném případě algoritmus skončí, protože pro žádný vyšší počet tetrád než je aktuální *n* 4 či více skupin rovněž po filtraci nezůstanou.

ad 5) Postupně jsou kombinováním konfigurací multiskupin generovány konfigurace vstupní sekvence, nad nimiž probíhá zbylá část algoritmu.

ad 6) Každé skupině aktuální konfigurace jsou vypočtena všechna možná umístění podřetězce o délce *n*. Každá tato skupina odpovídá jedné úrovni grafu na obrázku 17 a každé umístění je

jedním vrcholem tohoto grafu v dané úrovni. Následně je u každé skupiny vytvořen seznam všech možných skupin, jež mohou obsahovat další sekvenci tetrády tvořících bází ve výsledném kvadruplexu. Tento seznam reprezentuje hrany zmiňovaného grafu.

ad 7) Rekurzivním procházením vytvořeného grafu počínaje od všech skupin aktuální konfigurace jsou vypsány všechny cesty v grafu o vrcholové délce 4, které tvoří hledané kvadruplexy.

Použité datové typy

Program využívá tři datové typy definující struktury, které reprezentují skupinu, multiskupinu a nalezené kvadruplexy.

Datový typ skupiny má název *group_t* a obsahuje atributy *offset*, který obsahuje pozici prvního znaku skupiny v rámci vstupní sekvence, *length* obsahující počet bází ve skupině, *positions*, do nějž jsou ukládány pozice konfigurací skupiny generované v bodě 6 algoritmu 3 a *next*, který slouží pro uložení indexů možných následníků skupiny, které jsou zjištěny v tomtéž bodě algoritmu 33.

Datový typ multiskupiny je pojmenován *multigroup_t*. Tato struktura je využívána jako rozšíření struktury *group_t*, což znamená, že pro každou proměnnou typu *multigroup_t* v programu vždy existuje struktura typu *group_t*. Atribut *isMultigroup* drží informaci o tom, zda je příslušná struktura *group_t* multiskupinou či nikoliv. Dalšími atributy, které jsou relevantní pouze v případě, že *isMultigroup* má hodnotu *true*, jsou atribut *configs*, který je vektorem vektorů pozic podskupin multiskupiny, atribut *currentConfig*, který obsahuje index ukazující na vektor podskupin ve vektoru *configs* a atributy *previous* a *next*, které obsahují index předchozí a následující struktury *multigroup_t* s atributem *isMultigroup* rovným *true* v rámci vektoru struktur *multigroup_t*. Poslední čtyři jmenované atributy jsou využívány při generování konfigurací multiskupin a konfigurací vstupní sekvence.

Název datového typu struktury pro uložení nalezených kvadruplexů je *quadruplex_t* a tato struktura obsahuje počet tetrád v atributu *tetradeCount*, vektor pozic sekvencí tetrády tvořících bází v atributu *groupStarts* a délky jednotlivých smyček ve vektoru *loopLengths*.

Rozlišení multiskupin a generování jejich konfigurací

Multiskupinou je každá skupina taková, že maximální počet podskupin, které může ve svých konfiguracích obsáhnout je větší než jedna. Způsob generování všech konfigurací multiskupiny lze vidět na algoritmu 4.

Maximální počet podskupin *PP* je vypočítán jako

$$PP = \lfloor (DS + MDS)/(n + MDS) \rfloor \qquad (7)$$

kde *DS* je délka zkoumané skupiny, *MDS* je minimální délka smyčky a *n* je počet tetrád.

```
   GGGGGGGGGGGGGG        GGGGGGGGGGGGGG        GGGGGGGGGGGGGG
 1. GGG-GGG-GGG---     9. GGG---GGG--GGG    17. --GGG-GGG-GGG-
 2. GGG-GGG--GGG--    10. GGG----GGG-GGG    18. --GGG-GGG--GGG
 3. GGG-GGG---GGG-    11. -GGG-GGG-GGG--    19. --GGG--GGG-GGG
 4. GGG-GGG----GGG    12. -GGG-GGG--GGG-    20. ---GGG-GGG-GGG
 5. GGG--GGG-GGG--    13. -GGG-GGG---GGG
 6. GGG--GGG--GGG-    14. -GGG--GGG-GGG-
 7. GGG--GGG---GGG    15. -GGG--GGG--GGG
 8. GGG---GGG-GGG-    16. -GGG---GGG-GGG
```

Obrázek 19: **Ukázka postupného generování konfigurací multiskupiny** o délce 14 pro 3 podskupiny a 3 tetrády.

```
// pro každý možný počet podskupin od jedné do maximálního počtu podskupin
for (subgroupsCount = 1; subgroupsCount < PP; ++ subgroupsCount) {
   // vygeneruj počáteční konfiguraci, ve které jsou podskupiny na svých
   // nejlevějších možných pozicích, a prohlas ji za nově nalezenou
   currentConfig = getInitialConfig();
   found = true;

   // dokud jsou nacházeny nové konfigurace
   while (found) {
      // ulož nalezenou konfiguraci
      multigroup.configs.push_back(currentConfig);

      // pokus se naleznout další konfiguraci úpravou předchozí
      found = false;
      // pro každou podskupinu k předchozí konfigurace od nejpravější
      // po nejlevější
      for (k = subgroupsCount - 1; k >= 0; --k) {
         if (k == subgroupsCount - 1) { // pokud jde o nejpravejsi skupinu
            if (isMovableToRight()) { // pokud ji lze posunout o pozici doprava
               currentConfig[k]++; //posune skupinu o pozici doprava
               found = true; // byla nalezena nová konfigurace
               break; // v tomto cyklu jiz nic neposunuje
            }
         }
         else { //nejde o nejpravější skupinu
            if (isMovableToRight()) { // pokud ji lze posunout o pozici doprava
               currentConfig[k]++; // posune skupinu o pozici doprava

               // posune vsechny pravejsi skupiny tesne za ni
               moveGroupsOnRight();
               found = true;
               break; // v tomto cyklu jiz nic neposunuje
            }
         }
      }
   }
}
```

Algoritmus 4: **Generování konfigurací multiskupiny.**

Generování konfigurací vstupní sekvence

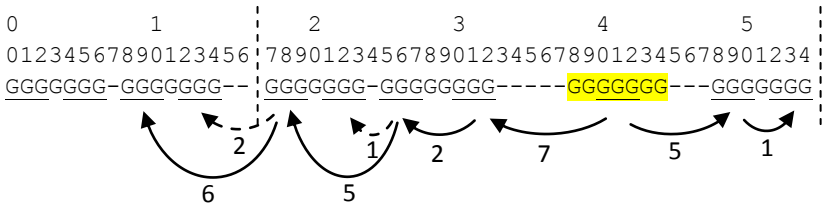
Při generování konfigurací vstupní sekvence se uplatňuje podobný princip jako při generování konfigurací multiskupin. Namísto kombinování pozic podskupin jsou však konfigurace vstupní sekvence vytvářeny postupným procházením prvků kartézského součinu množin konfigurací multiskupin. Nad každým takovýmto prvkem je pak vytvořen graf popisovaný v kapitole 4.2.2. Informace o multiskupinách jsou v programu uloženy ve vektoru struktur typu *multigroup_t*. Naposled použitá konfigurace vstupní sekvence je tedy reprezentována všemi atributy *currentConfig* v těchto strukturách. První konfigurace vstupní sekvence se skládá z prvních konfigurací multiskupin. Způsob generování dalších konfigurace z konfigurace předchozí popisuje algoritmus 5.

```
// od nejpravější skupiny
currentMultigroup = rightmostMultigroup;
do {
   // pokud má skupina další konfiguraci
   if (multigroups[currentMultigroup].configs.size() - 1 >
      multigroups[currentMultigroup].currentConfig) {
      // posune multiskupinu do následující konfigurace
      multigroups[currentMultigroup].currentConfig++;

      // nastav všem pravějším multiskupinám jejich první konfiguraci
      while (true) {
         // posune se o skupinu vpravo
         currentMultigroup = multigroups[currentMultigroup].next;
         if (currentMultigroup != -1) { // pokud skupina vpravo existuje
            // nastaví její první konfiguraci
            multigroups[currentMultigroup].currentConfig = 0;
         }
         else {
            found = true;
            break; // byla nalezena nová konfigurace
         }
      }
   }
   else {
      // pokud existuje nějaká multiskupina více vlevo
      if (multigroups[currentMultigroup].previous != -1) {
         // pokusí se ji posunout v dalším cyklu
         currentMultigroup = multigroups[currentMultigroup].previous;
      }
      else {
         found = false;
         break; // nebyla nalezena další konfigurace
      }
   }
} while (true); // dokud není nalezena nová konfigurace
```

Algoritmus 5: Generování konfigurací vstupní sekvence.

Po naleznutí nové konfigurace jsou do vektoru datových struktur *group_t* uloženy všechny skupiny a podskupiny multiskupin. Při generování nové konfigurace vstupní sekvence je vždy změněna konfigurace pouze části multiskupin. Aby zbytečně nedocházelo k opakovanému prohledávání nezměněné části konfigurace a nalézání již nalezených kvadruplexů, je zmíněný vektor ořezán pouze na skupiny, u nichž existuje šance, že utvoří kvadruplex společně s některou podskupinou z konfigurace nejlevější multiskupiny, jejíž konfigurace byla alespoň jednou změněna (obrázek 20). Takto upravený vektor je již vhodnou reprezentací zmiňovaného grafu, do níž jsou ještě doplněny všechny možné konfigurace skupin a seznam skupin, které mohou po každé skupině následovat (viz bod 6 algoritmu 3).



Obrázek 20: **Metoda ořezání vektoru skupin reprezentujícího konfiguraci sekvence.** Žlutě vyznačená multiskupina je nejlevější multiskupina, jejíž konfigurace byla alespoň jednou změněna. Podtržením jsou vyznačeny aktuální konfigurace multiskupin. Šipky ukazují skupiny, do kterých se dá z jiné skupiny dostat, aniž by byla porušena podmínka maximální velikosti smyčky (zde 7). V úvahu jsou přitom dále brány jen skoky do nejzazších skupiny (čárkovaná šipka ukazuje skok, který není nejzazší). Vektor je pak ořezán tak, aby na každé straně od podtržené žluté podskupiny byly maximálně 3 skoky.

Procházení grafu

Procházení grafem je implementováno jako rekurzivní funkce s maximálním zanořením do čtvrté úrovně, ve které je nalezen a vypsán výsledný kvadruplex (v každé úrovni je kvadruplexu ukládanému do struktury typu *quadruplex_t* přidána jedna skupina bází tvořících tetrády). Tato funkce je volána pro každou skupinu, která může tvořit začátek kvadruplexu. Princip této funkce popisuje algoritmus 6.

```
// pro každou možnou konfiguraci skupiny
for (i = 0; i < groups[groupsIndex].positions.size(); ++i) {
   // pokud přidáním této konfigurace není porušena podmínka maximální
   // délky smyčky
   if (loopLength <= maxLoop) {
      // rozšíří kvadruplex o tuto konfiguraci
      q.groupStarts.push_back(groups[groupsIndex].positions[i]);

      // pokud se nachází v nižší než čtvrté úrovni zanoření
      if (level < 3) {
         // pro každou možnou následující G skupinu
         for (int j = 0; j < groups[groupsIndex].next.size(); ++j) {
            //vola rekurzivne tuto funkci
            crawlGraph(groups, groups[groupsIndex].next[j], level + 1, q);
         }
      }
      else { // pokud se nachází ve čtvrté úrovni zanoření
         // vytiskne kvadruplex
         printQuadruplex(q);
      }
   }
}
```

Algoritmus 6: **Rekurzivní algoritmus procházení grafem** a hledání výsledných kvadruplexů.

Časová a prostorová složitost

Časovou složitost algoritmu nelze určit přesně, protože je silně ovlivněna tvarem vstupní sekvence, tedy počtem a délkou skupin a multiskupin a také jejich pořadím. Lze však říct do jaké míry ji ovlivní jednotlivé parametry vstupní sekvence.

Pro standardní rozsah 3 až 5 tetrád je nutné provést 3 iterace. Každá iterace sice obsahuje rozdílné počty skupin a multiskupin, ale pro jednoduchost zacházejme s jednotlivými iteracemi tak, jako by byly rovnocenné. Pokud by tomu tak nebylo, bylo by nutné uvažovat následující vzorce pro každou iteraci zvlášť.

Algoritmus prochází všechny možné konfigurace vstupní sekvence. Uvažujme, opět pro jednoduchost, případ, kdy vstupní sekvence obsahuje pouze multiskupiny stejné délky. Počet konfigurací vstupní sekvence *SCC* se pak vypočítá jako

$$SCC = MCC^{MC} \tag{8}$$

kde *MCC* je počet konfigurací jedné multiskupiny a *MC* je počet multiskupin. *SCC* tedy roste s počtem multiskupin exponenciálně.

Při procházení grafu je nutné projít také všechny konfigurace skupin. Mějme sekvenci, ve které nejsou žádné multiskupiny a uvažujme u všech skupin stejnou délku *GL*. Maximální počet konfigurací jedné skupiny *GCC* je

$$GCC = GL - TC + 1 \tag{9}$$

kde *TC* je počet tetrád ve výsledném algoritmu. Při počtu skupin *GC* je počet cest *PC* o vrcholové délce 4 v jednom grafu

$$PC = (GC - 3)\, GCC^4 \tag{10}$$

Obecně lze tedy říci, že časová náročnost závisí především na počtu multiskupin a pak také na jejich délkách, protože s délkou multiskupiny roste počet jejích konfigurací.

Prostorová složitost algoritmu je rovněž odvozena od těchto parametrů. Závislost na počtu multiskupin ovšem není v tomto případě exponenciální, protože v jednom okamžiku je vygenerována pouze jedna konfigurace vstupní sekvence a po jejím zpracování je zahozena. Paměť využitá pro uložení multiskupin a jejich konfigurací závisí na lineárně na jejich počtu. Stejná závislost platí i pro uložení skupin. Procházení grafů využívá při volání rekurzivní funkce implicitní zásobník, nicméně protože je zanoření funkce omezeno do čtvrté úrovně, je paměťová složitost této funkce konstantní a lze ji zanedbat.

5 Experimenty

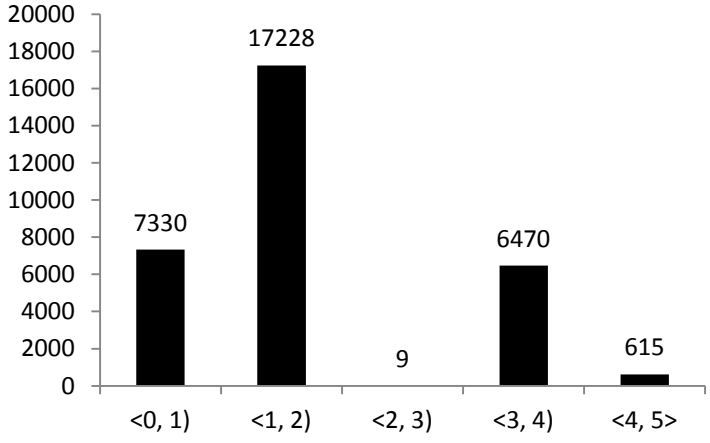
Aplikace byla testována na sekvencích lidského genomu, konkrétně na sestavení *hg19* z února roku 2009. Statistické údaje o výsledcích hledání je možné vyčíst přímo z webové aplikace, zde proto jsou uvedeny pouze údaje z hledání kvadruplexových sekvencí programem QFinder a následné anotace pro první chromozom:

- **Celkový počet nalezených sekvencí:** 31652
- **Počet sekvencí nezasahujících do genů:** 11224
- **Počet sekvencí v exonech:** 1853
- **Počet sekvencí v intronech:** 15338
- **Počet sekvencí v promotorech:** 5883
- **Počet shluků:** 5129

Tabulka 2 a histogram na obrázku 21 ukazují rozložení hodnocení kvadruplexových sekvencí.

Interval	<0,0>	<0,1>	<1,1>	<1,2>	<2,2>	<2,3>	<3,3>	<3,4>	<4,4>	<4,5>	<5,5>
Počet sekvencí	200	12708	5378	17228	0	6479	6470	7018	548	615	67

Tabulka 2: **Rozložení hodnocení** kvadruplexových sekvencí.



Obrázek 21: **Histogram rozložení hodnocení** kvadruplexových sekvencí.

6 Závěr

Tato práce seznámila čtenáře se základy molekulární biologie a způsoby uložení informace v DNA sekvencích do takové míry, aby byl schopen pochopit účel a význam vytvořené aplikace. Následně byla podrobněji rozebrána problematika G-kvadruplexů a jejich vyhledávání, včetně popisu existujících aplikací a algoritmů řešících tento problém. Pro finální webovou aplikaci byly zvoleno jak použití algoritmu vytvořeného dříve na Fakultě informačních technologií VUT v Brně, tak vytvoření algoritmu nového, z nichž každý odvádí jinou část práce a navzájem se doplňují.

Webová aplikace je schopna detekované výsledky jak vizualizovat, tak i filtrovat podle více parametrů. Kromě zobrazení výsledků přímo ve vytvořené aplikaci je schopná nalezený výsledek předat externí aplikaci UCSC Genome Browser, která dokáže výsledek vizualizovat značně pokročilejším způsobem.

Kromě kvadruplexových sekvencí aplikace vyhledává i jiné typy sekvencí. Mezi těmito typy sekvencí však aplikace není schopná naleznout souvislosti. Do budoucna by tak aplikace mohla být rozšířena o schopnost analyzovat vztahy mezi různými množinami sekvencí.

Literatura

[1] ROSYPAL, Stanislav. *Úvod do molekulární biologie*: *Díl první. (Informační makromolekuly-Molekulární biologie prokaryot)*. 3. vyd. Brno: Stanislav Rosypal, 1999, 300 s. ISBN 80-902-5620-1

[2] BURGE, S., G. N. PARKINSON, P. HAZEL, A. K. TODD a S. NEIDLE. Quadruplex DNA: sequence, topology and structure. *Nucleic Acids Research* [online]. **34**(19), 5402-5415 [cit. 2011-12-27]. DOI: 10.1093/nar/gkl655. Dostupné z: http://www.nar.oxfordjournals.org/cgi/doi/10.1093/nar/gkl655

[3] Telomere. *Wikipedia*: *The Free Encyclopedia* [online]. 2011-12-10 [cit. 2011-12-28]. Dostupné z: http://en.wikipedia.org/wiki/Telomere

[4] HUPPERT, J. L. Prevalence of quadruplexes in the human genome. *Nucleic Acids Research* [online]. **33**(9), 2908-2916 [cit. 2011-12-28]. DOI: 10.1093/nar/gki609. Dostupné z: http://www.nar.oupjournals.org/cgi/doi/10.1093/nar/gki609

[5] STEGLE, O., L. PAYET, J.-L. MERGNY, D. J. C. MACKAY a J. L. HUPPERT. Predicting and understanding the stability of G-quadruplexes. *Bioinformatics* [online]. **25**(12), i374-i1382 [cit. 2011-12-30]. DOI: 10.1093/bioinformatics/btp210 . Dostupné z: http://bioinformatics.oxfordjournals.org/cgi/doi/10.1093/bioinformatics/btp210

[6] Quadruplex.org. TODD, Alan. UNIVERSITY OF LONDON, SCHOOL OF PHARMACY. *Quadruplex.org* [online]. [cit. 2011-12-30]. Dostupné z: http://www.quadruplex.org

[7] JOHNSON, Jay E., Jasmine S. SMITH, Marina L. KOZAK a F. Brad JOHNSON In vivo veritas: Using yeast to probe the biological functions of G-quadruplexes. *Biochimie* [online]. **90**(8), 1250-1263 [cit. 2011-12-30]. DOI: 10.1016/j.biochi.2008.02.013. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0300908408000473

[8] ALBERTS, Bruce a ALBERTS. *Základy buněčné biologie*: *Úvod do molekulární biologie buňky*. 2. vyd. Překlad Arnošt Kotyk, Bohumil Bouzek, Pavel Hozák. Ústí nad Labem: Espero Publishing, 1998, 630 s. ISBN 80-902-9060-4.

[9] MIRKIN, S.M. Discovery of alternative DNA structures: a heroic decade (1979-1989). [online]. 8 [cit. 2012-01-04]. Dostupné z: http://ase.tufts.edu/biology/labs/mirkin/documents/2008FBS.pdf

[10] WONG, Han Min, Oliver STEGLE, Simon RODGERS a Julian Leon HUPPERT. A Toolbox for Predicting G-Quadruplex Formation and Stability. *Journal of Nucleic Acids* [online]. **2010**, 1-6 [cit. 2012-01-04]. DOI: 10.4061/2010/564946. Dostupné z: http://www.hindawi.com/journals/jna/2010/564946/

[11] Němec Jiří: Vyhledávání specifických struktur v DNA sekvencích, bakalářská práce, Brno, FIT VUT v Brně, 2010

[12] RAIBER, E.-A., R. KRANASTER, E. LAM, M. NIKAN a S. BALASUBRAMANIAN. A non-canonical DNA structure is a binding motif for the transcription factor SP1 in vitro. *Nucleic Acids Research* [online]. - [cit. 2012-01-06]. DOI: 10.1093/nar/gkr882. Dostupné z: http://www.nar.oxfordjournals.org/cgi/doi/10.1093/nar/gkr882

[13] JOHNSON, R. E. Biochemical evidence for the requirement of Hoogsteen base pairing for replication by human DNA polymerase . Proceedings of the National Academy of Sciences [online]. 102(30), 10466-10471 [cit. 2012-01-09]. DOI: 10.1073/pnas.0503859102. Dostupné z: http://www.pnas.org/cgi/doi/10.1073/pnas.0503859102

[14] O'CONNOR, Clare. Telomeres of Human Chromosomes. *Nature Education* [online]. [cit. 2012-01-10]. Dostupné z: http://www.nature.com/scitable/topicpage/telomeres-of-human-chromosomes-21041