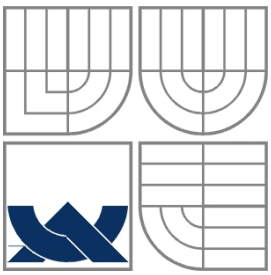
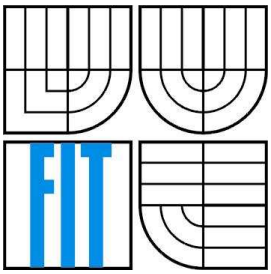


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

TECHNIKY PRO POROVNÁVÁNÍ BIOLOGICKÝCH SEKVENCÍ

TECHNIQUES FOR COMPARING BIOLOGICAL SEQUENCES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Roman Sladký
AUTHOR

VEDOUCÍ PRÁCE — Ing. Ivana Rudolfová
SUPERVISOR

BRNO 2008

Abstrakt

V práci se seznámíme se výstavbou a funkcí základních biologických jednotek DNA, RNA a proteinů. Data, která poskytují, se uchovávají v biologických databázích, které jsou celosvětově propojeny pro lepší komunikaci a dostupnost veškerých informací při vědeckém bádání. Tajemství života je skryto v genech. Geny jsou kódovány sekvencemi nukleotidů a dávají vznik proteinům, které jsou tvořeny sekvencemi aminokyselin. Nejrozšířenějšími technikami pro porovnávání sekvencí jsou algoritmy FASTA a BLAST. V práci je popsán program PSProt, který je založen na bázi zmíněných algoritmů. Slouží k porovnávání sekvencí proteinů. Porovnávaný protein se ale nejdříve pomyslně syntetizuje ze zadaného oligonukleotidu DNA, který kóduje potenciální protein. Nejpodobnější proteiny jsou pak vyhledány heuristikou hitbodů a poté algoritmem semiglobálního zarovnání upraveno jejich výsledné skóre rozhodující pro vyhledávání.

Klíčová slova

Molekulární biologie, DNA, RNA, nukleotid, aminokyselina, protein, databáze, EBI, ENBL-Bank, Swiss-Prot, TrEMBL, PDB, EnSembl, Needlemann-Wunsch, matice PAM a BLOSUM, BLAST, FASTA, PSProt, Arabidopsis thaliana, hitbody, PSProt

Abstract

This work presents the building up of basic biological units DNA, RNA and proteins as well as their function. Provided data are kept in biological databases which are connected worldwide to supply preferable communication along with all kinds of available information to be used in the scientific research. The secret of alive is hidden in genes coded in sequences of nucleotides. Genes enable the creation of proteins which are made of sequences of amino-acids. The wide-spread methods of comparing these sequences are FASTA and BLAST algorithms. Their base is used for the PSProt program which is described in this work. PSProt program is the tool for comparing the sequences of proteins. First it is necessary to synthesise the protein from the DNA oligonucleotide because it codes the surveyed protein. The most similar proteins are searched out by heuristic of hitpoints, then their final score that is essential for aligning is modified by semiglobal alignment algorithm.

Keywords

Molecular Biology, DNA, RNA, nucleotide, amino-acid, protein, database, EBI, ENBL-Bank, Swiss-Prot, TrEMBL, PDB, EnSembl, Needlemann-Wunsch, PAM and BLOSUM matrix, BLAST, FASTA, PSProt, Arabidopsis thaliana, hitpoints, PSProt

TECHNIKY PRO POROVNÁVÁNÍ BIOLOGICKÝCH SEKVENCÍ

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Ivany Rudolfové.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Bc. Roman Sladký
19.5.2008

Poděkování

Chtěl bych poděkovat především Ing. Ivaně Rudolfové, která se mě ujala a vedla mě k cíli.

Obsah

1 Úvod

Biologické sekvence, čili sekvence aminokyselin či nukleotidů, DNA, RNA, 3D struktura proteinů a další informace o nosičích genetické informace. To vše se skrývá pod pojmem ***bioinformatika***. Relativně mladou vědeckou oblast zajímají především souvislosti mezi genetickými kódy, vlastnosti a závislosti chování živých organismů. Snaha organizovat experimentálně či výpočty získaná data vedla ke vzniku prvních databází zaměřených na biologická data. Nejrůznější biologické informace (sekvence nukleotidů v DNA či v RNA, sekvence aminokyselin kódovaných bázovými nukleotidy adeninem, guaninem, cytosinem a uracilem, ...) jsou uschovávány a efektivně organizovány v databázích biologických dat, které jsou dále děleny podle specifikace na databáze sekvencí nukleotidů (např. EMBL-Bank, DDBJ), databáze sekvencí proteinů (např. Swiss-Prot, TrEMBL), databáze struktur proteinů (např. PDB, MSD), genomové databáze (např. Ensembl), databáze obsahující informace o expresi genů (např. ArrayExpress), atd.

Data jsou v těchto „databankách“ uložena přehledně tak, jak vyžaduje jejich specifikace, připravena ke snadnému použití pro biologické výpočty a práce. Databáze jsou většinou veřejně přístupná, „neškodná data“, jejichž další postup třetím osobám nemůže jakýmkoli způsobem ohrozit např. probíhající výzkum.

Biologickými sekvencemi jsou např. i sekvence aminokyselin tvořící proteiny. Proteiny se významně podílejí na činnosti a funkci každého živého organismu, od nejjednodušších bakterií až po organismy nejsložitější, např. člověka. Ať už je řeč o podpoře pohybových funkcí, posilování pojivové tkáně, katalýze chemických reakcí v těle organismů, kontrole trávení, metabolismu, reprodukce, řízení imunitního systému, ... Proteiny řídí téměř vše, proto je zajímavé se zabývat jejich strukturou, složením, trojrozměrným tvarem, do kterého se protein sbalí, aby mohl vykonávat právě onu biologickou funkci, atd. Proteiny jsou vytvářeny složitou cestou z DNA, přes RNA, procesy zvanými translace a transkripce.

Cílem této diplomové práce bylo seznámit se s technikami pro porovnávání biologických sekvencí a na základě získaných vlastností navrhnout vlastní metodu pro porovnávání sekvencí. V rámci diplomové práce byl navržen a implementován program PSProt, který slouží k nalezení nejpodobnějších proteinů ke všem proteinům, které se mohou syntetizovat ze vstupního sekvence nukleotidů DNA.

První kapitola práce, nepočítáme-li úvod, je věnována seznámením se se „stavebními prvky“ sekvencí. Těmito stavebními prvky jsou nukleotidy v DNA, resp. RNA a aminokyseliny v proteinech. Nastíníme si jak vypadá a co znamená struktura nukleotidů v DNA, vznik RNA, jakožto katalyzátoru pro převod DNA na protein, důležitou část každého organismu.

V kapitole „Databáze biologických sekvencí“ jsou stručně popsány nejznámější světové databáze biologických dat, jejich spolupráce a podpora.

Následující kapitola je věnováno tomu, co je informatice nejbližší, a sice porovnání sekvencí. Sekvence aminokyselin či nukleotidů kóduje genetickou informaci, v těchto sekvencích jsou skryty geny a ty určují mnoho parametrů živého organismu. Sledováním podobností a paralel vývoje sekvencí jsou vědci fascinování od samotného objevu genů.

Nejznámějšími technikami pro porovnání biologických sekvencí jsou nástroje BLAST a FASTA. S těmito technikami se blíže seznámíme a jednu z těchto metod blíže rozebereme v další fázi diplomové práce, naimplementujeme ji a pokusíme se dojít k novým poznatkům v oblasti hledání algoritmů pro porovnání biologických sekvencí.

V páté kapitole se seznámíme s programem PSProt vyvinutým pro nalezení proteinu z oligonukleotidu DNA (kratší sekvence nukleotidů, zpravidla 10-1000 bází) a jeho porovnáním s proteinovou databází. Výsledkem je seznam nejpodobnějších proteinů s parametry podobnosti s každým potenciálním proteinem nalezeným ve vstupní sekvenci DNA. Program využívá nový způsob kombinovaného porovnávání, a sice heuristiku hitbodů a modifikace algoritmu Needleman-Wunsch, semiglobálního zarovnání, pro nalezení nejpodobnějšího proteinu. Implementace tohoto programu je popsána v kapitole 6.

Testy a experimenty ověřující funkčnost programu jsou předmět předposledních dvou kapitol.

V závěru této práce shrneme všechny prerekvizity potřebné k vývinu programu, vzešlé výsledky a nová zjištění.

2 Bioinformatika a molekulární biologie

Bioinformatika je v užším slova smyslu chápána jako soubor mechanismů pro kódování a sdílení informace v živých systémech, organismech. V širším slova smyslu je to pak veškerá matematická práce s těmito daty, ať už biomatematická analýza chování živých organismů, studie procesů uvnitř systému, či jiné operace.

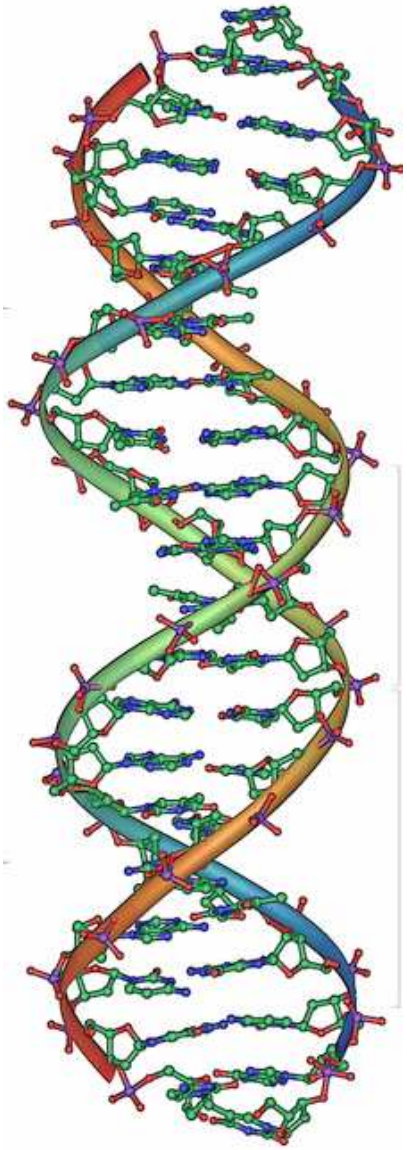
Každý živý tvor je seskupení neživých stavebních jednotek, molekul. Aby celý tento systém vytvářel živý organismus, musí mít nastavená nějaká pravidla, podle kterých se bude chovat. Soubor těchto pravidel (informací) pro regulaci vnitřního složení, růstu, reprodukci, udržování chodu organismu, které si buňky mezi sebou předávají, se nazývá ***genom*** (souboru genů nacházejících se v DNA). Pro zajímavost lidský genom obsahuje přibližně 30 000 genů a tvoří jej asi 3 000 000 000 bází [1].

Informace (data) jsou uloženy na molekulární úrovni. Jsou reprezentována sekvencemi nukleotidů v DNA či RNA, sekvencemi aminokyselin tvořící proteiny. Druh a pořadí návaznosti jednotlivých aminokyselin jsou kódovány v molekule mRNA, která po vycestování z jádra slouží jako vzor k tvorbě velkého množství proteinů, tj. bílkovin, které organismus v daný okamžik potřebuje. Proces rozdělení dvoušroubovice DNA, přeměny její části na RNA (proces transkripce), případný přenos vzniklé molekuly RNA mimo jádro a následná syntéza proteinu z templátového řetězce RNA (proces translace) je stejný pro všechny živé organismy na Zemi a je nazýván ***ústředním dogmatem molekulární biologie***.

Předpokladem práce s bioinformatikou je znalost molekulární biologie, hlavně pak jejich základních pojmů jako jsou ***DNA***, ***RNA***, ***protein***.

2.1 DNA

Každý živý organismus na planetě Zemi je určen a řízen informacemi, které se nacházejí v jednom z nejpozoruhodnějším objektu, a sice ve dvoušroubovici DNA (deoxyribonukleová kyselina). DNA byla izolována již v roce 1869. Do sebe šroubovitě zapletené řetězce nukleotidů se vyskytují v jádře buněk všech organismů (s výjimkou prokaryot). DNA je tvořena částmi, které řídí tvorbu proteinů (geny) a jinými nekódujícími sekvencemi nukleotidů. To, že geny jsou součástí či jakýmsi produktem DNA, vědci přiznali v roce 1952.



Obrázek 1 – DNA [13]

DNA je primárním nositelem informace. Dvouvláknová pravotočivá šroubovice držící pohromadě díky vodíkovým vazbám. Každé vlákno je tvořeno sekvencí nukleotidů, které se nepravidelně opakují a svým pořadím tak určují jedinečnou primární strukturu a tudíž genetickou informaci, která je pak přenášena. V celé DNA se vyskytují celkem pouze ***4 různé nukleotidy*** – adenin (A), thymin (T), cytosin (C) a guanin (G). Každý nukleotid se skládá z fosfátové skupiny, cukru a dusíkaté báze. Rozdíl je právě v dusíkatých bázích.

Obě vlákna DNA drží pohromadě díky vazbám nepravidelně poskládaných nukleotidů. Tyto vazby jsou schopny vytvořit pouze dva a dva nukleotidy spolu a udržet tak vlákna u sebe. Toto se nazývá principem komplementarity. Adenin se váže jen a pouze s thyminem, protože oba mají dva volné vodíky, které spolu vytvoří vodíkovou vazbu, cytosin a guanin se vážou třemi vodíkovými vazbami. Vedle počtu vodíkových vazeb je další důvodem výběru dvojic prostorová skladnost molekuly. Jelikož je pořadí nukleotidů na jednom vlákně šroubovice odvoditelné z protilehlého (vlákna jsou antiparalelní), můžeme hovořit o redundantní informaci, kterou šroubovice DNA nese. Polynukleotidový řetězec je ohraničen konci, které jsou označeny jako 5´konec a 3´konec, přičemž většina operací jako např. čtení řetězce pro tvorbu RNA probíhá ve směru od 5´konce. [3]

Replikace DNA probíhá jednoduše uvolněním vodíkových vazeb na jednom místě šroubovice a navázáním komplementárních nukleotidů a tím se z jedné DNA vytvoří dvě dceřiné molekuly DNA.

Jestliže si buňky žádají nějaký protein, příslušný gen (tj. sekvence několika příslušných nukleotidů) se nejprve zkopíruje na řetězec RNA, který pak slouží jako templát pro tvorbu samotného proteinu, neboli sekvenci aminokyselin.

2.2 RNA

Cesta informace od DNA až do výsledného potřebného proteinu vede „trnitou cestou“ přes RNA. Jak již bylo řečeno výše, jedná se o centrální dogma celé molekulární biologie, proto se na RNA podíváme blíže. Molekula RNA nenese de facto genetickou informaci, pouze ji přenáší při tvorbě proteinů a pak sama zaniká.

Prvním krokem celé syntézy proteinu je přepis genu do molekuly RNA v jádře, neboli ***transkripce***. Podobně jako DNA je i polynukleotidový řetězec RNA spojen fosfodiesterovými vazbami, avšak již se zde nevyskytuje thymin, ale je nahrazen uracilem (U). Uracil má podobné vazebné schopnosti jako thymin, především stejný počet vodíkových můstků, proto se také páruje s adeninem. Dalším, pro nás již méně významným rozdílem, je jiná cukerná složka nukleotidů. Zatímco v DNA to byla deoxyribosa, v RNA je to ribosa. Zásadním rozdílem je ale to, že RNA se již nevyskytuje v antiparalelní dvoušroubovici, ale tvoří lineární řetězec, který se ale sbalí do různých tvarů. 3D struktura řetězců RNA je rovněž rozhodující pro její funkci.

Přepis začíná rozvolněním dvoušroubovice DNA, podobně jako tomu je při její replikaci, avšak tentokráte je ihned po navázání ribonukleotidu (nukleotidu v RNA) zničena vodíková vazba nově vznikajícího RNA řetězce s templátovým vláknem DNA a naopak je obnovena původní vodíková vazba s druhým vláknem DNA. Tímto způsobem se vlákno RNA ihned vytěsňuje. V porovnání s celkovou délkou DNA je délka RNA zanedbatelná. Například délka lidského DNA je přibližně 250 miliónů párů bází, délka RNA je maximálně několik tisíc ribonukleotidů.

Enzymy, které se pohybují po DNA a rozvolňují ji pro syntézu RNA, se nazývají ***RNA-polymerázy***, a dvoušroubovici rozvíjí ve směru od 5´konce ke 3´konci. Jelikož k uvolnění vlákna RNA dochází téměř okamžitě, je možno nasyntetizovat během jedné transkripce větší množství molekul RNA, a tím tak téměř geometricky zvýšit produkci výsledných proteinů. RNA-polymeráza katalyzující transkripci je méně dokonalá než DNA-polymeráza, rozvíjí dvoušroubovici DNA pro replikaci, a to minimálně v absenci korektorské schopnosti. DNA-polymeráza totiž po sobě ještě výsledné spojení nukleotidů zkontroluje, avšak RNA-polymeráza tuto schopnost nemá. Transkripce proto nemusí být dokonalá, na rozdíl od replikace DNA totiž vzniklá RNA neslouží pro trvalé uchování genetické informace, po přepisu na protein je zničena. Chybovost syntézy RNA je $1 : 10^4$.

Začátek genu (úseku DNA) rozpoznává RNA-polymeráza snadněji u jednodušších organismů, tedy bakterií. U eukaryotních buněk je to složitější. Jakmile se rozezná na DNA ***promotor*** (přesně daná sekvence nukleotidů, která znamená začátek genu), začne syntéza RNA. Konec genu ukazuje sekvence nukleotidů označená jako ***terminátor***. U bakteriálních buněk pak dále ***proteosyntéza*** (syntéza proteinu z DNA) probíhá ještě před dokončení syntézy samotné RNA přímo v cytoplazmě, protože tam se také nachází jak bakteriální DNA, tak ribosomy, které proteosyntézu katalyzují.

U eukaryotických buněk, tedy i lidských, se ihned syntéza proteinu ještě nekoná. Tolik potřebné ribosomy se totiž nacházejí v cytoplazmě, ***primární transkript*** (tj. holý řetězec RNA vzniklý transkripcí) je uzavřen v jádře. Před transportem z jádra do cytoplazmy jsou na primárním transkriptu provedeny posttranskripční úpravy (oba konce primárního transkriptu jsou zmodifikovány- přidání čepičky na 5´-konec a 3´-konec je polyadenylován), které molekulu stabilizují.

Poslední úpravou jaderné RNA před jejím vycestováním do cytoplazmy je vystřižení intronů. Pro vědce v průběhu 80.let 20.století bylo otázkou, proč je RNA v cytoplazmě o tolik kratší, než jaderná RNA? Zjistilo se, že geny jsou v DNA nepravidelně rozděleny na kódující sekvence nukleotidů (exony) a nekódující (introny). A tudíž i transkribovaná RNA. Bylo zjištěno, že introny jsou ještě před transportem do cytoplazmy z již upravené primárního transkriptu vystřiženy. Výsledkem je sestřih RNA, tedy ***funkční mRNA***, která je transportována z jádra do cytoplazmy. Sestřihem RNA se zde nebudeme dopodrobna zaobírat.[4]

2.2.1 mRNA, tRNA a rRNA

Známe celkem tři typy RNA.

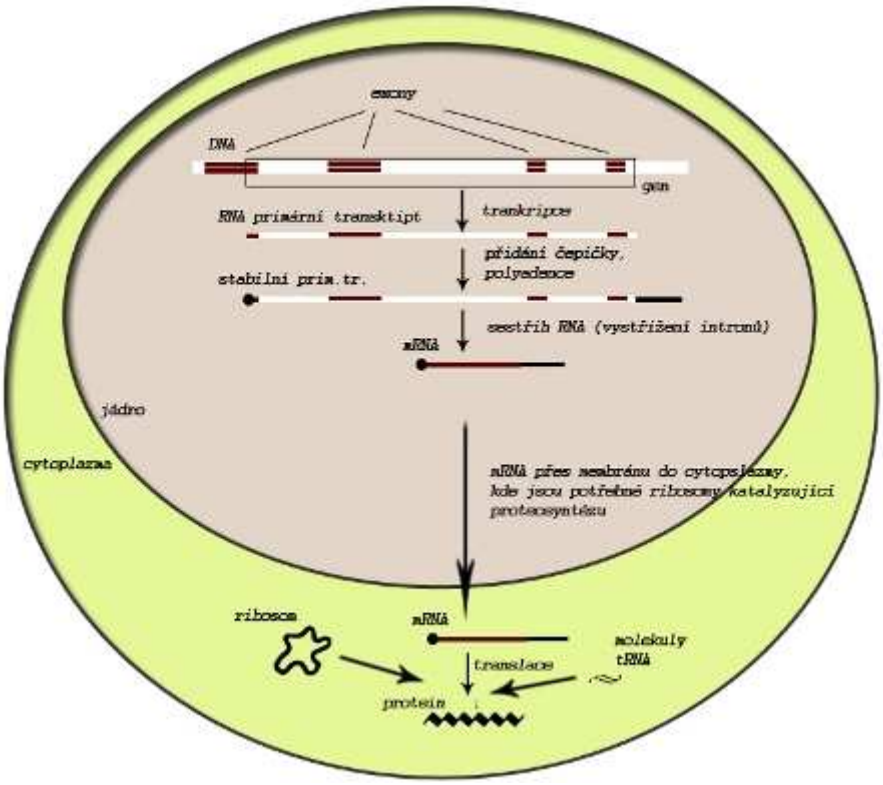
Informační neboli mediátorová RNA (***mRNA***, messenger RNA) je syntetizována jako samotný templát (vzor) pro translaci (syntézu proteinu). Má funkci kódování. Neinformační RNA molekuly mají funkci - hrají důležitou roli při překladu mRNA na protein.

rRNA (ribosomální RNA) tvoří jádro ribosomů v cytoplazmě, na kterých probíhá proteosyntéza.

tRNA (transferová RNA) je adaptorem, který při proteosyntéze vybírá správné aminokyseliny pohybující se v cytoplazmě. Obsahují přibližně kolem 80 nukleotidů. Každá tRNA slouží k přenosu pouze jedné konkrétní aminokyseliny. Jelikož se v přírodě vyskytuje pouze 21 různých aminokyselin, v každé buňce musí být 21 různých tRNA. Vytvořením párů bází mezi různými oblastmi téže molekuly se tRNA umí sbalit do tvaru připomínající čtyřlístek (celkem 4 oblasti se na sebe naváží). tRNA obsahují část nukleotidů zvanou antikodon, kterým rozpoznají kodon na m NA, na volný 3´-konec se naváže aminokyselina příslušná pro každou tRNA. Jelikož se první dva nukleotidy kodonu přesně párují s prvními dvěma nukleotidy antikodonu, jen vazba třetí dvojice není tak pevná, dochází k tzv. ***kolísavému párování bází***, což je jev, kdy více kodonů, lišících se pouze ve třetím nukleotidu, může znamenat stejnou aminokyselinu. Ve skutečnosti existuje pro mnoho aminokyselin více tRNA s různými kodony, a naopak i některé antikodony tRNA se mohou párovat s více než jedním kodonem v mRNA. [4]

2.3 Vznik proteinu, proteosyntéza

Informace obsažená v mRNA, která přišla do cytoplazmy, bude přeložena translací do řetězce aminokyselin, tedy do proteinu. ***Translace***, neboli překlad, je o mnoho složitější než je přepis úseku DNA do RNA. Jestliže transkripci je možno si představit jako prosté opsání čtyř písmen A, C, G, T na komplementární (a výměnu U za T), pak v translaci je mechanismus syntézy jiný, srovnatelný s překladem slov do jiných jazyků. Z jazyka nukleotidů do jazyka aminokyselin.



Obrázek 2 – Tvorba proteinu

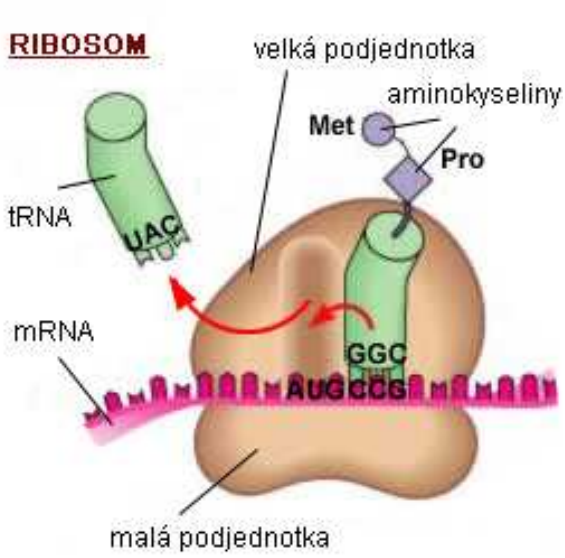
Výsledný protein je sekvencí 20 různých aminokyselin. Každá aminokyselina je kódována trojicí nukleotidů po sobě jdoucích v mRNA. Pravidla pro kódování jsou obecně známá jako ***genetický kód***. Sestavení tohoto „slovníku“ se podařilo světovým vědcům v roce 1966. Každá trojice nukleotidů by tedy měla kódovat jednu aminokyselinu. Tak by ale mělo vzniknout celkem 4^3, tedy 64 aminokyselin. V přírodě se vyskytuje aminiokyselin pouze 21, tudíž některé různé trojice nukleotidů (***kodony***) mohou kódovat stejnou aminokyselinu.

Antikodon tRNA rozpozná kodon mRNA na základě již známé komplementarity C-G, A-U. Pro přesnou a rychlou translaci je ale zapotřebí ***ribosomu***. Tento komplex se pohybuje podél mRNA, zachytává molekuly tRNA a spojuje aminokyseliny na jejich 3´-konci ve výsledný aminokyselinový řetězec. Skládá se z mnoha proteinů a několika druhů rRNA.

Ribosom má velkou a malou podjednotku. Malá podjednotka, vytvořená především prostřednictvím rRNA, zodpovídá za správné spárování kodonu mRNA a antikodonu tRNA, velká

podjednotka, kterou tvoří především proteiny, zodpovídá na navázání aminokyselin a tvorbu samotného nového proteinu. Ribosom se pohybuje podél mRNA (opět směrem od 5´-konce), překládá trojice nukleotidů přes tRNA na aminokyseliny a spojuje je v protein. Po úspěšné syntéze proteinu se ribosom rozpadne na podjednotky.

Místo, ve kterém začne proteosyntéza, je kritickým místem celé syntézy proteinu. Rozhoduje o čtecím rámci trojice nukleotidů. Změnou by se mohla znehodnotit celá proteosyntéza a vnikl by tak úplně jiný protein, tvořený úplně jinou sekvencí aminokyselin, které byly pospojovány na základě správných nalezení kodonů antikodony tRNA, ovšem se špatným začátkem celého řetězce. Toto ošetřuje tzv. ***inicializační tRNA***, což je jedna z mnoha tRNA v cytoplazmě buňky. Inicializační tRNA má na 3´-konci methionin (u bakterií formylmethionin), který je později z proteinu odstraněn. Methionin se pak váže ještě na několik tRNA, ale inicializační je jen jedna.

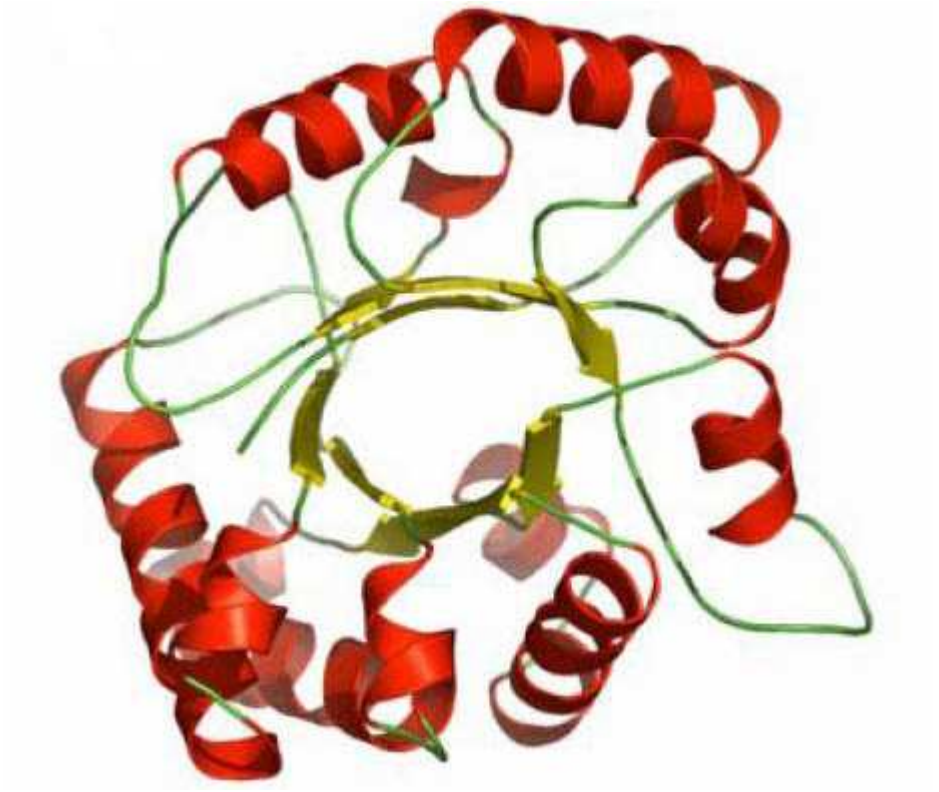


Obrázek 3 – Ribosom [14, převzato s úpravami]

Vlastní proteosyntéza začíná tak, že se inicializační tRNA naváže na malou podjednotku, za asistence několika inicalizačních faktorů. Po navázání malá podjednotka „nasedne“ na 5´-konec mRNA (s čepičkou) a pohybuje se po mRNA směrem ke 3´-konci až do chvíle, kdy inicializační tRNA nachází ***startovací kodon*** AUG. Pak se inicializační faktory uvolní a dovolí tak připojení velké podjednotce. Začíná ***elogance*** (prodlužování), což znamená, že se vážou další molekuly tRNA, rozpoznávající kodony za kodonem AUG, rozpoznaným inicializační tRNA, a vzniká protein. Až do nalezení terminačního kodonu (***stop-kodonu***). Tím je jeden z kodonů UAA, UAG nebo UGA. Tyto kodony se vážou na tRNA, které nemají na sobě navázanou žádnou aminokyselinu, proto proteosynéza skončí. (pozn.: kodon UGA slouží pro připojení aminokyseliny selenocysteinu, ale pouze když se kodon UGA vyskytuje v sousedství určitých nukleotidů). Sekvence mezi start-kodonem a stop-kodonem se nazývá ***ORF (Open Reading Frame)***. [4]

Konec proteosyntézy pak doprovázejí terminační faktory, přes chemickou vazbu vody se pak celý řetězec aminokyselin uvolní do cytoplazmy, ribosom se rozpadne na opět na velkou a malou podjednotku a obě tyto části jsou pak připraveny se navázat na inicializační faktor pro další proteosyntézu.

2.4 Proteiny



Obrázek 4 – Protein, ilustrace [13]

Proteiny zastávají v organismu obrovskou řadu funkcí. Vznikají při proteosyntéze přepisem z mRNA. Skládají z jednoho či více polypeptidických řetězců, což jsou lineární polymery aminokyselinových zbytků. Některé aminokyseliny si tělo neumí vyrobit, musí se přijímat potravou.

Z obrovského množství mnohdy zásadních funkcí mají proteiny např. stavební funkce, podílí se na tvorbě extracelulárního matrixu, tedy hmoty, v níž jsou umístěny buňky. Těmito proteiny jsou například kolagen či elastin. Na tvorbě organismu se rovněž podílí např tubulin či aktin.

Jednou z nejdůležitějších funkcí je rovně katalýza chemických reakcí, přesněji tvorba a tvorba a rozpad kovalentních vazeb molekul (znám je např. pepsin, který odbourává v žaludku proteiny z potravy). Takovým proteinům se říká enzymy.

Transportní funkce zastává například protein známý jako hemoglobin, který přenáší po buňkách kyslík). Dalšími funkcemi proteinů jsou například pohybové funkce (myosin v kosterních svalech umožňuje pohyb, ve svalech), zásobní funkce (železo se ukládá v játrech tak, že se váže na protein feritin), signální funkce (přenáší informační signály z buňky do buňky- např. insulin). Proteiny fungují i jako receptory vnějších vlivů, detekují v buňkách fyzikální a chemické signály a předávají je ke zpracování buňce (např. rhodopsin v oční sítnici zachycuje světlo).

Proteiny rovněž regulují samotnou genovou expresi.

2.4.1 Struktury proteinu

Proteinové vlákno se po vytvoření sbalí do 3D tvaru, který je vedle pořadí aminokyselin a délky celého řetězce rozhodující pro určení funkce proteinu v organismu.

Každá ***aminokyselina*** se skládá z alfa uhlíku, aminoskupiny (skupina obsahující dusík), karboxylové skupiny, jednoho vodíku a postranního řetězce R, kterým se jednotlivé aminokyseliny liší. Postranní řetězec zásadně ovlivňuje chování a funkci celého proteinu. Rozlišujeme aminokyseliny hydrofobní, polární a s nábojem. To určuje postranní řetězec chemickými vlastnostmi atomů, které ho tvoří.

Nepolární, hydrofobní aminokyseliny obsahují v postranním řetězci jen uhlík a vodík. Tyto dva prvky mezi sebou vytváří kovalentní vazby, které nejsou elektricky nabité, což znamená, že tyto aminokyseliny jsou nepolární, bez možnosti vytvořit vodíkové vazby, pokud jsou ve vodném prostředí. Jsou tedy z tohoto hlediska pro aminokyselinu, potažmo celý protein nepodstatné a neužitečné, čili celá proteinová struktura se pokouší, při snaze maximalizovat možnost vazby s okolním vodným prostředím při sbalení do 3D struktury, schovat tyto hydrofobní aminokyseliny dovnitř struktury.

Polární aminokyseliny obsahují v postranním řetězci kyslík nebo dusík. Tyto prvky jsou elektricky nabité, schopny vytvářet vodíkové vazby s molekulou vody. Struktura se tedy snaží je dostat na povrch. Aminokyseliny s nábojem mají ještě větší možnost interakce s vodou, nalézáme je tudíž rovněž na povrchu 3D struktury.[13]

Obrázek 5 – struktura α-helix [13]

3 Databáze biologických dat

V posledních letech dochází k velkému rozmachu oblasti vědy zvané molekulární biologie, je získáváno obrovské množství dat, a ta se musí nějakým způsobem účelně a přehledně uschovávat pro pozdější použití a sekundární výpočty. Proto vznikla v roce 1980 v Heidelbergu v Německu EMBL Nucleotide Sequence Data Library (nyní známé jako EMBL-Bank) jako první světová databanka sekvencí nukleotidů. Původní záměr byl vystavět celosvětovou počítačovou databázi. V roce 1992 byla vytvořena oficiální instituce ***EBI (European Bioinformatics Institut)*** jako nezisková organizace pro výzkum a rozvoj v oblasti bioinformatiky. Spravuje databáze biologických dat, tzn. informací o nukleových kyselinách, sekvencích proteinů a strukturách proteinů. Tato data jsou přístupná veřejnosti, většinou zabývající se výzkumem v oblasti molekulární biologie. EBI databáze sama vytváří a zprostředkovává data v použitelné formě třetím osobám.

V roce 1995 se sídlo definitivně přesunulo do Wellcome Trust Genome Campus ve Velké Británii.

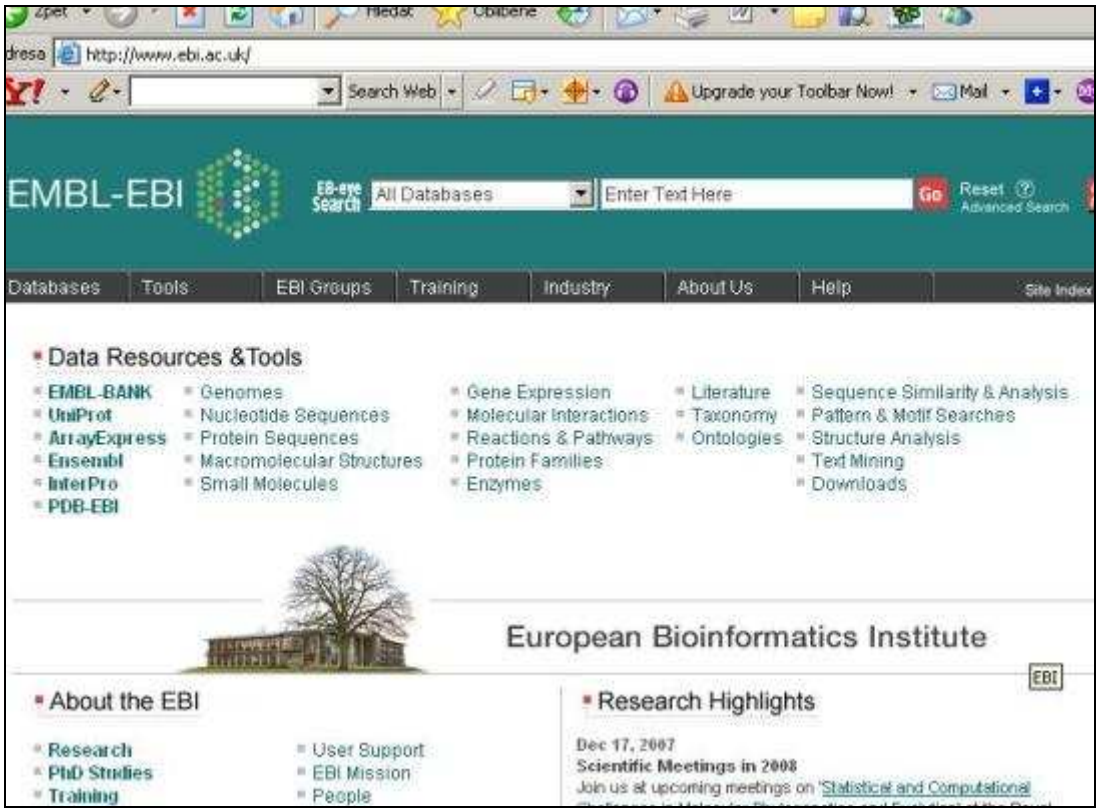
Dnes již můžeme všechny databáze rozdělit na primární a sekundární. Primární databáze obsahují experimentálně získaná data, sekundární databáze obsahují data již vypočítané, analýzou získané z dat primárních databází.

Sekundárními databázemi se zabývat nebudeme.

3.1 Primární databáze

3.1.1 Databáze sekvencí nukleotidů

Sekvence nukleotidů se nacházejí v řetězcích DNA i RNA. Hlavní evropská databáze těchto sekvencí, obsahující nejen samotné sekvence, ale i informace o nich a odkazy na další údaje je ***EMBL-Bank***. Dalšími organizace jsou GenBank v USA (vznik v roce 1982), DDBJ (DNA Database of Japan) v Japonsku, databáze dbEST či GSDB. Všechny tři organizace mezi sebou denně komunikují, předávají si nové informace a poznatky, aby měli všichni přístup k nejaktuálnějším informacím. Denně přibývají nová data, nové spojitosti a detaily sekvencí. Mají i podobnou strukturu, liší se v detailech. GenBank například nepoužívá k označení typu informace dvoupísmenný kód na začátku řádku, ale celé slovo, používá jiný vyhledávací systém (Entrez), atd.



Obrázek 6 –European Bioinformatics Institut (EBI) [5]

3.1.1.1 Práce s databází EMBL-Bank

Vyhledávání v databází sekvencí nukleotidů je veřejnosti usnadněno zpřístupněním dotazovacího nástroje ***SRS (Sequence Retrieval System)*** pro získávání dat z databáze na oficiálních www stránkách EBI http://www.ebi.ac.uk/ přes Databases -> Database Browsing nebo přímo na adrese http://srs.ebi.ac.uk/.

Další možností zisku dat z databáze je připojení na ftp server EBI. Navíc jsou v pravidelných tříměsíčních intervalech vydávány na DVD nosičích aktualizované databáze EBI.

Do databáze je možno vložit novou sekvenci nukleotidů či rozšířit informace o již vložené sekvenci. Ve vědeckých článcích se pak autoři odkazují na identifikační číslo sekvence v databázi, dokonce musí do databáze sekvenci a veškeré informace o ní vložit dříve, než vyjde samotná publikace s ní spojená. Sekvence je tedy do databáze přidána ještě před publikováním prvního článku, zmiňujícího tuto sekvenci, vědci zabývající se danou problematikou pak mají potřebné informace ihned po ruce. Pro možnost vložení byla naimplementováno rozhraní WebIn na oficiálních www stránkách. Součástí vložení jakékoli informace je samozřejmě nejen holá sekvence či strohá data, ale i další povinné údaje o autorovi, odkazy na literaturu zahrnující další informace k dané sekvenci a další podrobnosti.

3.1.1.2 Struktura dat v EMBL-Bank

Data jsou v databázi členěna do několika zásadních tříd a divizí. Každý záznam v databázi obsahuje informace právě o jedné sekvenci nukleotidů.

Veřejně přístupné záznamy spadají do třídy standard. Pro větší přehlednost se celá databáze dělí do několika divizí, označujících se třípísmennou kombinací. Do jedné divize spadají většinou taxonomicky příbuzné sekvence jako např. viry (označení VRL), houby (FUN), nebo z jiného společného hlediska, např. geny (GSS). Divize EST (expressed sequence tags) zahrnuje jednoduché sekvence nukleotidů, většinou nepodstatné a neobsahující žádné další hodnotné informace. O tom, do jaké divize sekvence spadá, informuje identifikační řádek.

Samotný záznam je pak konstruován co nejsrozumitelněji pro lidskou i počítačovou řeč. Každá informace má svůj dvojznakový identifikátor umístěný na začátku řádku, který nám říká, jaký typ informace o sekvenci můžeme na řádku očekávat.

Na zvoleném příkladu vidíme nejdůležitější řádky. ID- základní identifikátor sekvence včetně třídy, divize, id čísla v databázi, atd. XX- prostý volný řádek pro větší přehlednost. AC- řádek pro příbuzných či podobných sekvencí, na které se můžeme přes odkaz dostat. Je vždy minimálně jeden, protože obsahuje sám sebe. DT- datum vložení a poslední modifikace záznamu. DE- popis sekvence (za anglického „description“). KW- výstižná slova (z anglického „key words“), slouží pro lepší orientaci při vyhledávání a jiných referenčních operacích. OS- taxonomické označení organismu, ze kterého byla sekvence získána. OC- třída organismů, ze kterých byla sekvence získána. RN- referenční číslo v záznamu. RP- vymezuje hranice sekvence, která se přesně objevila v publikaci, která na sekvenci odkazuje. RA- autor publikace, která na sekvenci odkazuje. RT- název publikace, která na sekvenci odkazuje. RL- typ publikace a její název či odkaz na ni. CC- poznámkové řádek. FH- nadpis poznámky, vždy se pojí s řádky FT. FT- poznámka k sekvenci, detaily jednotlivých úseků. SQ- sekvence nukleotidů. //- indikuje konec záznamu. Existují i další řádky, které nejsou v popisu této jednoduché sekvence (divize EST) uvedeny.

Každý řádek musí mít přesný formát v závislosti na jeho typu.

Příklad 1: Záznam v databázi EMBL-Bank.

```
ID   DC499669; SV 1; linear; mRNA; EST; INV; 731 BP.
XX
AC   DC499669;
XX
DT   18-DEC-2007 (Rel. 94, Created)
DT   18-DEC-2007 (Rel. 94, Last updated, Version 1)
XX
DE   Monosiga ovata cDNA, clone: MOC-071P07, 5'end.
XX
KW   5'-end sequence (5'-EST); expressed sequence tag.
XX
OS   Monosiga ovata
OC   Eukaryota; Choanoflagellida; Codonosigidae; Monosiga.
XX
RN   [1]
RP   1-731
RA   Fujiyama A., Toyoda A., Kuroki Y., Suzuki Y., Sugano S., Sakaki Y.;
```

```
RT   ;
RL   Submitted (18-JUL-2007) to the EMBL/GenBank/DDBJ databases.
RL   Contact:Atsushi Toyoda The Institute of Physical and Chemical
RL   Research(RIKEN), Genomic Sciences Center(GSC); 1-7-22
RL   Suehiro-chou,Tsurumi-ku, Yokohama, Kanagawa 230-0045, Japan URL
RL   :http://stt.gsc.riken.jp
XX
RN   [2]
RA   Fujiyama A., Toyoda A., Suzuki Y., Sugano S., Kuroki Y., Sakaki Y.;
RT   "Evolution of choanoflagellate genes";
RL   Unpublished.
XX
CC   Vector: pME18S-FL3;
CC   Site_1: DraIII;
CC   Site_2: DraIII;
CC   1st strand cDNA was primed with an oligo (dT) primer [GCGGCT
CC   GAAGACGGCCTATGTGGCCTTTTTTTTTTTTTTTT];double-stranded cDNA w
CC   as ligated to a DraIII adaptor [GGCCUACUGG], digested and di
CC   rectionally cloned into distinct DraIII sites of the pME18S-
CC   FL3. Library was size selected for 1.0 kb, with a average in
CC   sert size of ~1.2kb. Library constructed by Yutaka Suzuki (U
CC   niversity of Tokyo, Institute of Medical Science).Custom pri
CC   mersrecommended for sequencing: 5' end primer 5'-GGATGTTGCCT
CC   TTACTTCTA-3' and 3' end primer 5'-CGACCTGCAGCTCGAGCACA-3'.
XX
FH   Key             Location/Qualifiers
FH
FT   source          1..731
FT                   /organism="Monosiga ovata"
FT                   /strain="2A4"
FT                   /mol_type="mRNA"
FT                   /clone_lib="Full length cDNA Library, Monosiga ovata, MOC."
FT                   /clone="MOC-071P07"
FT                   /tissue_type="whole cell"
FT                   /db_xref="taxon:81526"
XX
SQ   Sequence 731 BP; 161 A; 257 C; 181 G; 132 T; 0 other;
     tgaaggctgt cgctgccctc ctcgttgcgc ttgcggcttg cgcccttgcc gagccctact        60
     tccaggagga cttcactggc gactgggcca gcaactgggt gcagtccaag gccaagtctg       120
     actacggcaa gttcgttggc tccactggca agttcttcgg tgatgccgag atctccaagg       180
     gcatcaagac cagcgaggat gccaagaact acgccctgtc tgcctagttc gctcccttct       240
     ccaacgaggg caagaagctc atcatccagt tcactgtcaa gcacgagcag gacatcgact       300
     gcggtggcgg ctacgtcaag ctgttccctt ccaccatcga ccagaaggac atgcacggtg       360
     gcgagggcga gacccctтac aacatcatgt tcggccccga catctgcggt cccggccacc       420
     gcaaggtcca cgtgatcttc gcgtacaagg gcaccaacca ccagaccaag aagaccattg       480
     cctgcaagtc cgacaccctc actcaccttt acaccctcat tgtcaacccc gacaacacct       540
     acgaggttcg cattgacgat gccaaggtcg agtctggctc cctgtacgag gacttcgatt       600
     tcctcccccc caagctcatc aacgaccccg ctcagtccaa gcccaaggac tgggtcgacc       660
     agaagaccat cgccgacccc gacgacaaga agcccgctga ctgggacgtc cccgagacca       720
     ttgccgaccc c                                                           731
//
```

Veškeré poznámky a jazyk jsou vedeny v angličtině, co nejvíce srozumitelné širší veřejnosti, maximálně s využitím odborných termínů molekulární biologie.

3.1.2 Databáze sekvencí proteinů

Organizace EBI má pod sebou i hlavní databázi proteinových sekvencí ***UniProtKB*** (UniProt Knowledgebase). Data o sekvencích proteinů jsou organizována podobně jako data v EMBL-Bank kvůli přehlednosti a vzájemné korespondenci databází. Databázi UnitProtKB tvoří dvě poddatabáze – Swiss-Prot a TrEMBL.

Data v databázi Swiss-Prot jsou složena záznamy o sekvencích proteinů, které se již objevily v některé publikaci a je na ně odkazováno, zatímco TrEMBL obsahuje počítačovou analýzou vypracované záznamy, které se ještě nikde v publikacích neobjevily, nikdo je zatím nepopsal a nezabýval se jimi. Jakmile se takto stane a sekvenci někdo zdokumentuje a popíše, záznam se přesune do databáze Swiss-Prot.[9]

Data jsou přístupna podobně jako u databáze EMBL-Bank na webových stránkách EBI opět pomocí nástroje SRS, na ftp serveru nebo prostřednictvím pravidelně vydávaného nosiče s aktuálním stavem databáze. Vkládat data lze jen do databáze Swiss-Prot, data v TrEMBL jsou generována.

3.1.2.1 Swiss-Prot

Byla založena v roce 1986 ve Švýcarsku, spjata se SIB (Swiss Institute of Bioinformatics), je součástí UniProtKB. Data v ní jsou organizovány podobně jako data v databázi sekvencí proteinů EMBL-Bank, tedy každý řádek v záznamu začíná dvojznakovým identifikátorem řádku, který předurčí jeho formát. Záznamy popisují detailně proteinové sekvence, tedy sekvence aminokyselin je tvořící, odkazy na literaturu, která se sekvencí zabývá, detaily o organismu, v němž byla sekvence objevena, funkci proteinu v organismu, struktury proteinu, bližší popis některých částí sekvence (tyto informace mohou mít přínos pro detailní zkoumání). Záznamy jsou samozřejmě co nejpodrobnější, obsahují i např. informace o postiženích či změnách organismu při špatné činnosti proteinu v důsledku např. mutace proteinu, úplné absence, atd. Záznamy popisující stejnou sekvenci jsou slučovány v jeden záznam a informace o nich jsou spojeny. Pokud dochází k ostrému střetu názorů na protein, jsou u spojeného záznamu uvedeny oba názory. V záznamu se nachází odkazy přímo do databází struktur proteinů a databází nukleotidů na záznam pojící se s danou proteinovou sekvencí. Toto propojení všech databází usnadňuje vědcům zabývající se komplexně molekulární biologií práci při hledání spojitostí, podobností, atd.

Struktura záznamu v poddatabázi je velmi podobná struktuře dat v databázi EMBL-Bank. Od TrEMBL se liší zařazením do třídy standard (záznamy poddatabáze TrEMBL je třídy preliminary). Některé řádky jsou pozměněny v závislosti na tom, kolik je jich přidáno (např. řádek označující se GN, který popisuje gen, ze kterého byl protein vyextrahován, jeho ORF, atd.)

3.1.2.2 TrEMBL

Rovněž databáze TrEMBL obsahuje záznamy popisující proteinové sekvence, avšak detailní informace o těchto počítačovou analýzou vytvořených sekvencích ještě nebyly nikde publikovány nebo obsahuje záznamy popsané nedostatečně na to, aby byly přidány do databáze Swiss-Prot. Podklady pro vytvoření těchto sekvencí se nacházejí v některé databázi nukleotidů, např. EMBL-Bank..

Společně s holou sekvencí aminokyselin, čili potenciálního proteinu, jsou do záznamu databáze TrEMBL vložena surová data získaná z databází nukleotidů a rovněž z databáze Swiss-Prot z podobných proteinových sekvencí. Tato data jsou v momentě dostatečného ručního upravení a popsání v publikaci či literatuře přesunuta do databáze Swiss-Prot. Redundance dat je ošetřena podobně jako v databázi Swiss-Prot sloučením do jednoho záznamu.

3.1.2.3 Další databáze sekvence proteinů

Další databází sekvencí proteinů je ***PIR*** (Protein Information Resource) sídlící na univerzitě v Georgetownu, založená v roce 1984. V roce 2002 společně se SIB a EBI vytvořili na objednávku NIH (Národního institutu zdraví) UniProt.

Jinými organizacemi jsou např. německá ***MIPS*** (Munich Information center for Protein Sequences) nebo ***NRL-3D***. [6]

3.1.3 Databáze struktur proteinů

Největší databází struktur proteinů je ***PDB*** (Protein Data Bank) vyvinuta v Brookhaven National Laboratory a udržována v rámci RSCB (Research Collaboratory for Structural Bioinformatics).

Struktura jako taková se určuje většinou pomocí rentgenové krystalografické analýzy (cca z 80%), nukleární magnetické resonance (cca z 18%) nebo modelováním. Struktura proteinu je v záznamech těchto databází popsána pomocí souřadnic všech atomů v osách x, y, z.

Nalézají se zde informace o tom, jakou metodou byla ta struktura zjištěna, s jakou přesností byla určena. Je popsána sekundární struktura proteinu, jeho aktivní místa, odkazy na ostatní databáze. Část záznamu tedy popisuje lokální struktury, část popisuje vazby (disulfické vazby, které se v proteinu váží). Databáze se opět pokouší svým objemem pokrýt všechna pro pozdější výzkum použitelná smysluplná data. Dalšími informacemi, které se k záznamu (proteinu) váží, jsou např. informace o organismu, z něhož byl protein získán. Data jsou organizována v deseti sekcích, každé označené klíčovým slovem, které předurčuje formát řádku záznamu.

Databáze struktur proteinů je opět propojena s ostatními databázemi sekvencí proteinů a nukleotidů.

3.1.4 Genomové databáze

Sekvenací DNA získáme jednotlivé geny a tudíž celý genom (soubor všech genů organismu). Ve snaze tyto geny nashromáždit a popsat tak pomocí nich organismus vznikly genomové databáze jako např. ***EnSembl***. V těchto databázích najedeme všechny geny (dosud zjištěné a zdokumentované)

daného živočišného druhu včetně jejich polohy na chromozomech. Lidský genom je se sekvenován od roku 1975 a bylo zjištěno, že člověk má přibližně 30 až 40 000 genů. [7]

Ensembl je spojením mezi EMBL a Wellcome Trust Sanger Institute s cílem vytvořit systém udržující popis všech genomů velkých eukaryotických organismů.

3.1.5 Databáze zahrnující informace o expresi genů

Úroveň exprese genů se určuje speciálními mikročipy- destičkami se sondami na povrchu, na které se váže jednovláknová mRNA. Sondy tvoří řetězec nukleotidů charakteristický pouze pro jeden konkrétní gen. Tato sekvence musí být a dostatečně dlouhá a zároveň taková, aby se nesbalila do 3D struktury a nevytvořila vazby mezi svými nukleotidy. Tím by znemožnila reakci mRNA. Na mikročipy se nanášejí vzorky obarvených tkání, ty mRNA se naváží na příslušné sondy a ve skenerech se pak přečte výsledná barevná matice sond. Barva přitom určuje množství navázané mRNA. Tento test se často používá pro porovnání genové exprese zdravé a nemocné tkáně.

Výsledky těchto testů se pak uchovávají v databázích. Záznamy v těchto databázích pak zahrnují velmi velké matice reálných čísel.

Jednou z těchto databází je ***ArrayExpress***, kterou rovněž zaštiťuje EBI. Momentálně je v ní uloženo přes 3200 experimentů.

3.2 Sekundární databáze

Obsahují výsledky analýzy dat z primárních databází. Jsou sestaveny pomocí mnohačetného porovnávání homologních sekvencí pro zachycení konzervovaných oblastí a tudíž jsou data rozřazována do rodin. Všechny informace jsou reprezentovány většinou nějakou abstraktní formou, jako například regulárním výrazem, bloky, chemickými strukturami, apod.

Z databáze UniProt je na principu regulárních výrazů získaných z proteinových sekvencí vystavěna databáze ***Prosite***, na principu Markovových modelů pak sekundární databáze ***Pfam***. Kombinacemi záznamů ze sekundárních databází Prosite a Prints pak můžeme vystavět záznamy v databázi ***BLOCKS***. Všechny tyto databáze obsahující informace o rodinách proteinů.

4 Porovnávání sekvencí

Z evoluce vyplývá, že všechny dnes žijící organismy se vyvinuly z jednoho druhu. S touto revoluční teorií přišel v šedesátých letech 19.století Charles Darwin a vědci ji přijali. Logicky podobné organismy mohou mít stejné nebo podobné geny, které se mohly ze společného předka vyvinout. Sekvence se mohou v běhu času změnit a mít pak i jinou funkci. Ke změnám dochází několika způsoby, a sice mutací (záměnou), insercí (vložením nového) nebo delecí (vymazáním). Porovnávání proteinových sekvencí či sekvencí nukleotidů se snaží najít odpověď nejen na tyto otázky.

Při porovnávání sekvencí hledáme optimální (nejlepší) zarovnání (*simple alignment*), čili nejvyšší počet shodných symbolů na stejných pozicích v porovnávaných řetězcích. Symboly zastupují nukleotidy, pokud porovnáváme sekvence nukleotidů DNA nebo RNA, aminokyseliny jsou to při porovnávání proteinových sekvencí. Porovnávat lze i celé skupiny sekvencí, ale to je pro momentální počítačovou výpočetní sílu příliš náročná záležitost, proto se při porovnávání skupiny sekvencí (*multiple alignment*) využívá nejrůznější heuristiky.

4.1 Porovnávání dvou sekvencí

Nejjednodušší porovnávání dvou sekvencí se liší uvažováním inserce nebo delece. To se v praxi projeví vložením mezery. Inserce nebo delece se v přírodě vyskytuje mnohem vzácněji než mutace, proto se někdy uvažuje pouze mutace a sekvence se tak porovnávají ***bez vkládání mezer***.

4.1.1 Porovnávání sekvencí bez vkládání mezer

Pak se problém porovnání redukuje pouze na problém zarovnání, tedy výběr začátku v delší sekvenci pro postupné zarovnání. Velkým měřítkem pro úspěch porovnání je výběr ohodnocení shody a neshody symbolů. Celkové ohodnocení porovnání dvou sekvencí se počítá jako suma všech shod plus suma všech neshod. Ohodnocení neshody v tomto případě mělo být záporné nebo minimálně rovno nule. Jelikož tato ohodnocení mohou být subjektivní a mylná, v praxi se využívají skórovací matice pro ohodnocení všech dvojic prvků nukleotidů a aminokyselin.

Příklad 2: porovnání sekvencí AABCA a ABA- výběr zarovnání, ohodnocení pro shodu: +1, ohodnocení pro neshodu: 0.

```
AABCA                AABCA                AABCA
ABA  ...skore: 1     ABA ...skore: 2      ABA ...skore: 1
```

V tomto případě se jasně jeví jako nejlepší varianta 2, ale v praxi toto platit nemusí. Záleží totiž jak moc výsledné porovnání ovlivní chyba (záměna C a A), přičemž toto je ovlivněno především chemickými a fyzikálními vlastnostmi obou prvků.

4.1.2 Porovnávání sekvencí s vkládáním mezer

Porovnávání dvou sekvencí s vkládáním mezer zahrnuje i možnosti inserce a delece do jedné ze sekvencí. Tady kromě ohodnocení shody a neshody do konečného výsledku musíme započítat i pokutu za vložení mezery. K nalezení nejvhodnějšího zarovnání se používají matice a jeden z výpočetních algoritmů.

Nejjednodušší algoritmus vyplňuje pozici matice určené oběma sekvencemi booleovskou hodnotou true (1) pro shodu a false (0) pro neshodu. Výsledné diagonální shluky jedniček (shody) pak slouží k nalezení výsledné cesty. Tento algoritmus je nevhodný jednak z podobného důvodu jako v porovnávání sekvencí bez vkládání mezer, tedy konstantním ohodnocením za neshodu a jednak také tím, že shluky „jedniček“ často komplikují nalezení jednoho nejoptimálnějšího řešení. Navíc pro delší sekvence je počítačový výpočet velmi náročný.

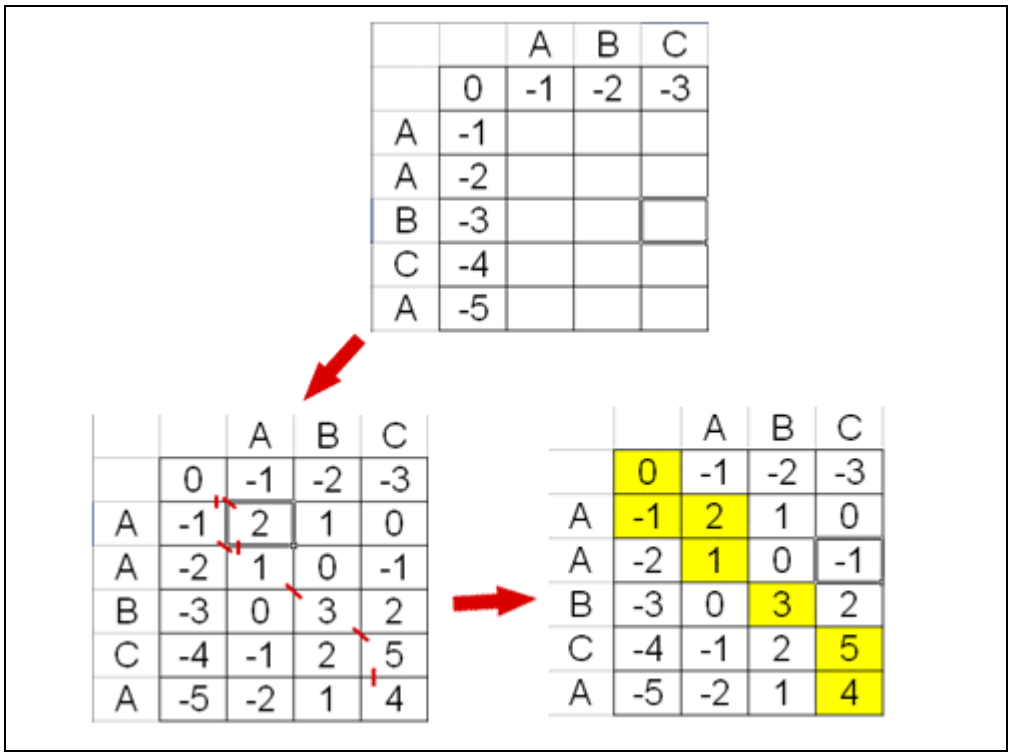
4.1.2.1 Algoritmus Needlemann-Wunsch

Nejvyužívanějším algoritmem pro nalezení nejoptimálnějšího zarovnání v porovnávání dvou sekvencí s vkládáním mezer je algoritmus ***Needlemann-Wunsch***. Tento algoritmus využívá dynamického programování, tedy přístupu řešení problému s rozdělením na podproblémy, konkrétně nalezení optimálního zarovnání každé pozice. Celkem třemi způsoby můžeme porovnávat jednotlivé pozice. Vložením mezery do jedné sekvence a porovnáním, vložením mezery do druhé sekvence a porovnáním, nebo prostým porovnáním bez vložení mezery. Vložení mezery se samozřejmě penalizuje. Hledáme maximální hodnotu pro shodu, čili nejoptimálnější zarovnání pro pozici.[12]

Popis algoritmu Needlemann-Wunsch: matice mající v obou záhlavích tabulky částečného skóre po jednoznakovém odsazení porovnávané sekvence se inicializuje v souřadnicích (1,1) hodnotou nula a dále po zbytek řádku a sloupce se přičítá postupně ohodnocení za vložení mezery (většinou -1). Inicializovanou tabulku pak vyplňujeme tak, že do každého dalšího pole se dostaneme třemi možnými způsoby popsanými výše, přičemž vybereme maximální ze tří možných ohodnocení.

Při vertikálním či horizontálním pohybu přičítáme k předchozí buňce pokutu za vložení mezery, při diagonálním pohybu přičítáme hodnotu za shodu/neshodu podle skórovacích tabulek PAM nebo BLOSUM. Po vyplnění celé tabulky se v pravém dolním rohu nachází hodnota skóre optimálního zarovnání obou sekvencí. Zpětným hledáním nejlepší cesty do levého horního rohu najdeme nejoptimálnější zarovnání obou sekvencí. Těchto zarovnání může být v některých případech i víc.[12]

Příklad 3: Předpokládejme ohodnocení shody pro A, B i C hodnotou 2 (toto je imaginární příklad, ve skutečnosti tyto hodnoty určují skórovací tabulky PAM a BLOSUM). Najdeme optimální zarovnání pro sekvence AABCA a ABC algoritmem Needlemann-Wunsch. Nejdříve inicializujeme tabulku částečného skóre, vyplníme ji a poté najdeme zpětnou cestu optimální zarovnání.

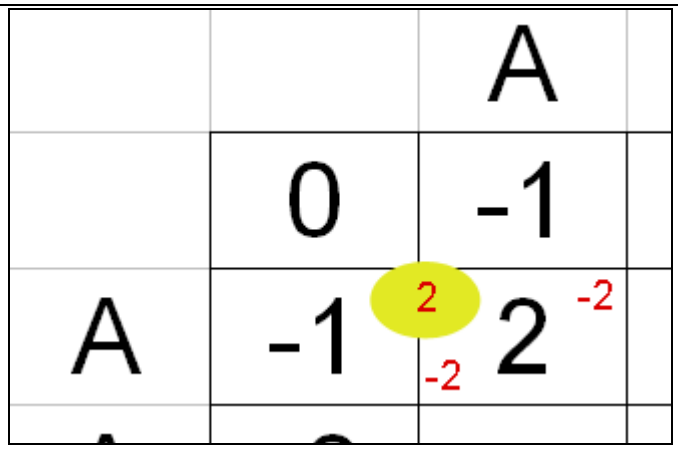


Obrázek 8 – Algoritmus Needlemann-Wunsch, tabulka částečného skóre

Zjistíme, že optimální zarovnání s vloženými mezerami jsou dvě:

AABCA AABCA

A_BC_ _ABC_



Obrázek 9 – Výběr optimálního zarovnání pozice při odhodnocení pro shodu = 2

4.1.3 Matice PAM a BLOSUM

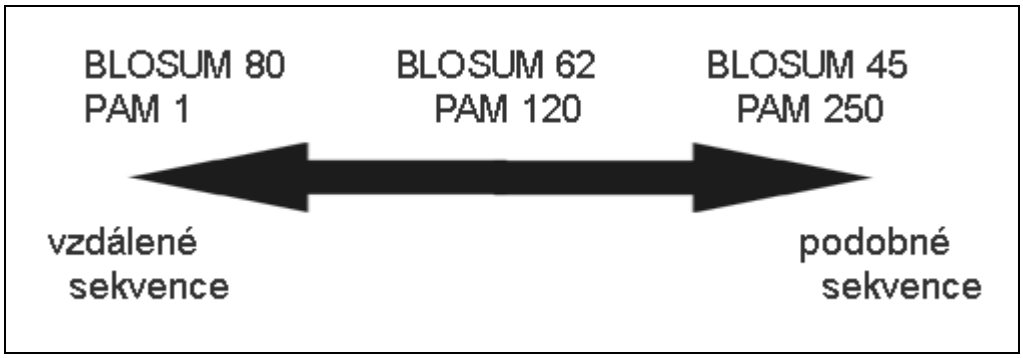
Skórovací matice pro nukleotidy je matice jednodušší se zohledněním pouze záměny podobných nukleotidů uracilu a thyminu (uracil jej nahrazuje v RNA) s adeninem oproti záměně adeninu např. s cytosinem. Pro aminokyseliny jsou to matice PAM a BLOSUM.

Obě rodiny matic se využívají pro zjištění skóre pro záměnu dvou prvků.

Rodina matic PAM (Point Accepted Mutation) je založena na globálních závislostech blízkých proteinů. Např. matice PAM1 je vypočítána z porovnání sekvencí s ne víc než 1% odchylkou.

Rodina matic BLOSUM (BLOcks Substitution Matrix) je založena na lokálních závislostech, na vypozorovaných srovnáních. Matice BLOSUM 62 je hojně používána, je implicitní, např. v nástroji BLAST 2.0 (bude o ní další zmínka v kapitole 6.1.3).

BLOSUM matice s vyššími čísly a PAM matice s nižšími čísly jsou tvořeny pro skóre záměny v podobných sekvencích, naopak BLOSUM matice s nižšími čísly a PAM matice s vyššími čísly jsou tvořeny pro vzdálené sekvence.[16]



Obrázek 9 – Matice PAM a BLOSUM

4.2 Porovnávání skupiny sekvencí

Teoreticky by bylo možné použít metodu Needlemann-Wunsch s použitím vícedimensionální tabulky částečného skóre, ovšem se zvětšujícím se počtem porovnávaných sekvencí by dimense tabulky rostla a velmi rychle se zvětšovala výpočetní složitost algoritmu, a již záhy by se tento algoritmus stal naprosto nepoužitelný. Z tohoto důvodu vznikly jiné algoritmy pro porovnávání skupiny sekvencí jako např. Clustal

4.2.1 Clustal

Metoda Clustal využívá evolučního stromu, který vytvoří z porovnávaných sekvencí imaginární evoluční strom, na jehož základně pak zarovná dvojice sekvencí, které jsou podobné. V listech stromu se nacházejí jednotlivé organismy, v uzlech jejich předchůdce, abstraktní organismus, ze kterého vzešly. Metoda využívá dynamického programování (rozložení problému na podproblémy, které se postupně řeší). Nenalezne sice úplně optimální řešení, ale i přes menší složitost nalezne téměř optimální řešení.

4.3 Nástroje pro porovnávání sekvencí

Pokud chceme porovnat sekvenci s kompletní databází, postupné procházení po dvojicích či skupinách by bylo opět velmi náročné a prakticky nemožné. Využívá se tedy nejrůznějších heuristik a indexů.. V současnosti se pro porovnávání dotazované sekvence s velkými databázemi používají algoritmy členící se do dvou rodin- algoritmy Fasta a Blast.

4.3.1 FASTA

Algoritmus pro porovnávání dotazované sekvence se sekvencemi v databázi Fasta byl dokončen v roce 1985.

Jeho princip je následující: každá podsekvence délky *k* se porovná s podsekvencemi každé sekvence v databázi. Pro každé takové porovnání se vytváří výsledek ve tvaru 2D-matice se zaznamenanými shodami, které jsou reprezentovány tečkami. Několik po sobě jdoucích shod v diagonále znamená podobnou či shodnou oblast. Vybere 10 oblastí s největším počtem identit a pomocí skórovacích tabulek je ohodnotí. Ostatními oblastmi se pak Fasta již nezabývá.

Parametr *k* (délka podsekvence) má zásadní vliv na nalezení shodných oblastí. Pro malé *k* je výpočet přesnější, avšak výpočet je náročnější a tudíž i delší. pro větší *k* je zase výpočet rychlejší, ale s jistou tolerancí chyb. Je třeba nalézt kompromis mezi přesností a rychlostí.

Algoritmus pokračuje výběrem jednoho regionu s největším skóre, označovaného jako *init1*, který pak snaží spojit s ostatními vybranými regiony (které mají skóre nad určitým prahem). Nejlepší pospojovaná kombinace sekvencí je pak označovaná jako *initn*. Initn skore pak slouží jako selektor sekvencí pro další krok. V tom se provede zarovnání s dotazovanou sekvencí- výsledné skore je *opt skore* a pomocí něj pak sekvence seřadí. [11]

4.3.2 BLAST

Rodina algoritmů Blast (Basic Local Alignment Search Tool) vznikla v roce 1990, kdy byl vyvinut základní algoritmus Blast1. Blast1 ale nepočítal s vkládáním mezer, byl zato velmi rychlý. V letech 1996-1997 nezávisle na sobě vznikly dvě nadstavby nástroje Blast1, NCBI-Blast2 a WU-Blast2, které již pracovaly s vkládáním mezer. NCBI-Blast2 byla vyvinuta organizací National Center for Biotechnology Information, WU-Blast2 na Washington University. Narozdíl od algoritmů Fasta, algoritmy Blast již při porovnávání neuvažují čistě o identitě prvků, ale uvažují také o ***podobnosti*** sekvencí.

Algoritmus pro ***Blast1***: dotazovaná sekvence se rozdělí na podsekvence délky *w* a porovná se všemi podsekvencemi délky *w* z databáze. Jelikož se hledají i podobné prvky, ke každé podsekvenci vygeneruje seznam slov, pro které je skóre větší než práh *T*. Pak se hledají identity mezi každým takovýmto slovem ze seznamu a slovy stejné délky ze sekvencí databáze. Pro rychlejší výpočet jsou seznamy slov délky *w* pro práh *T* předpočítané. Úspěch, tedy nalezení identity mezi slovem ze seznamu slov podsekvence a podsekvencí sekvence z databáze, se nazývá hit. Při nalezení hitu se algoritmus Blast pokouší rozšířit řetězec na obě strany a snaží se najít nejdelší podobnou podsekvenci (prodloužit délku w na možné maximum podle podobnosti slov, dokud skóre neklesne pod určitý práh). Zde se ještě stále hledá zarovnávání se bez vkládání mezer. Každá podobná dvojice podsekvecí se nazývá High scoring Segment Pair – HSP, ta s nejlepší skórem pak Maximum Segment Pair – MSP. Blast1 tedy najde nejpodobnější podsekvenci v porovnání bez vkládání mezer a seřadí je. [11]

4.3.2.1 NCBI-Blast2

Algoritmy Blast2 mají stejný základ jako Blast1, ale počítají již i s možností inserce či delece, tedy s vložením mezer.

Známější NCBI-Blast2 byl publikován v roce 1997. Omezením pro hledání podobností podsekvencí s vkládáním mezer je to, že v diagonále v maximální vzdálenosti *A* musí ležet další hit. Toto omezení s sebou přináší menší přesnost, ovšem tato nevýhoda může být částečně potlačena zvolením menšího prahu *T* v první fázi výpočtu. Po vybrání pouze některých hitů se tedy provádí zarovnání s vkládáním mezer.[5]

4.3.2.2 WU-Blast2

Vytvořen Warrenem Gishem z Washingtonské Univerzity v St.Louis v roce 1996, ale nebyl nikdy pořádně zdokumentován. Stále se vyvíjí. Již teď je to však mocný nástroj s mnoha možnosti spojující části NCBI-Blast2 a Fasta. Je jednodušší na použití než NCBI-Blast2.[5]

5 Program PSProt

Na základě získaných informací byl v rámci diplomové práce vyvinut program, který slouží pro porovnávání modifikovaných dat sekvence s proteinovou databází PSProt (Porovnání S Proteiny).

Diplomový projekt PSProt slouží ve zkratce pro zjištění, zda vstupní sekvence, odpovídající řetězci nukleotidů DNA, obsahuje takovou sekvenci, která se pomocí translace přepíše na odpovídající RNA a transkripcí vygeneruje určitý protein a následně zjistí, který protein z databáze je tomuto potencionálnímu proteinu nejpodobnější.

Je to tedy nástroj pro porovnávání sekvencí proteinů. Byl vyvinut na základně znalostí algoritmů FASTA a BLAST. Bližší je ale BLASTu, který se zabývá porovnáváním i podobných sekvencí, ve své verzi BLAST 2.0 i s vkládáním mezer. Zatímco FASTA porovnává postupně všechny podsekvence (a neuvažuje přitom o podobnosti, hledá pouze „čisté shody“), výsledky zaznamenává do 2D tabulky a následně hledá diagonální shody a konečné spojení, PSProt pro každou sekvenci a protein používá jinou heuristiku. Touto heuristikou je součet hitbodů (vysvětleno níže). PSProt uvažuje rovněž o možnosti v přírodě poměrně častého jevu, a sice mutace. V PSProtu je povolena pouze jedna mutace aminokyseliny při hledání hitu a jeho prodlužování. To, zda uvažovat o mutaci či ne a hledat podobné sekvence pouze na základě jejich čisté podobnosti, lze v programu přepnout příslušným parametrem. Zajímavější a jistě i vhodnější je ale uvažovat o mutaci a tedy vyhledávat i podobnosti sekvencí. Tímto se tedy se přibližujeme algoritmu BLAST.

Stejně jako BLAST nalézá PSProt pro každé podsekvence porovnávaných sekvencí shody (hity), které se snaží následně prodloužit na obě strany až do chvíle, než celková hodnota chyby klesne pod určitý práh podobnosti. Tento práh je v PSProtu určen jedinou povolenou mutací v hitu. Zatímco BLAST má předpočítaný jakýsi slovník pro každé podsekvencí (dále jen „oken“), PSProt takovýto slovník neobsahuje a mutaci hledá přímo na místě porovnání a potenciální shody, čili hitu. Absence slovníku je dána tím, že nepředpokládáme porovnávání příliš velkých oken a s tím spojenou větší časovou náročnost při zjišťování mutace okna. Pokud by velikost okna dosahovala řádově desítek prvků (aminokyselin), pokus o mutaci tohoto okna za účelem nalezení hitu by byl již velmi časově náročný a pak by se předdefinovaný slovník mutací oken vyplatil. Nicméně PSProt tento slovník neobsahuje, vzhledem k předpokládané menší velikosti okna je výhodnější realizovat mutaci prvek po prvku.

V nadstavbě BLASTu (ve verzích BLAST2) se porovnávají sekvence i s vkládáním mezer. PSProt tuto nadstavbu zajišťuje až po výběru nejlepších proteinů na základně heuristiky hitbodů a následně při zarovnání a výpočtu výsledného skóre. Pokuta za vložení mezery je standardně -1, avšak dá se v nastavení programu lehce změnit. Výsledné skóre zarovnání poopraví pořadí nejpodobnějších sekvencí, tedy nejlepších několik vybraných na základě hitbodů. Počet vybraných proteinů lze také nastavit.

5.1 Myšlenka programu PSProt

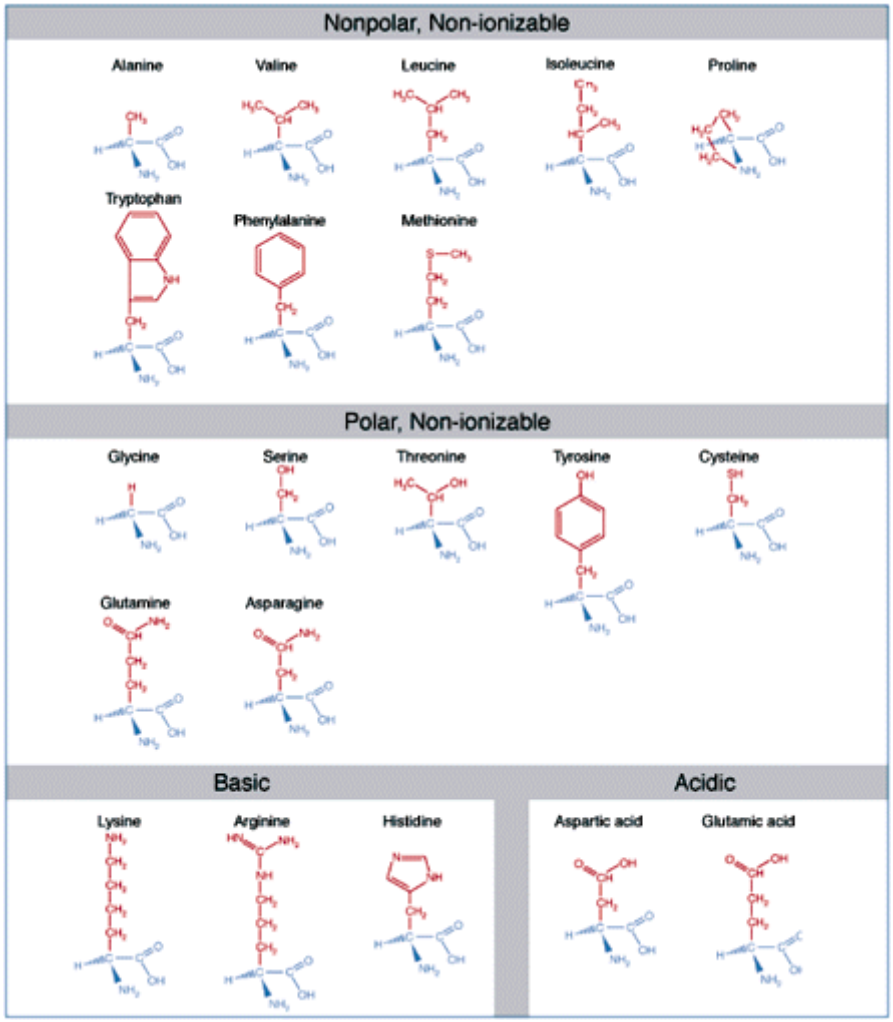
5.1.1 Počáteční zpracování vstupní sekvence

Celý algoritmus se skládá z několika kroků. Ve vstupní sekvenci (řetězci DNA) jsou nalezeny všechny sekvence, z nichž může vzniknout jeden celý protein. Každý takový potenciální protein je pak porovnáván se všem proteiny z databáze, načež je nalezeno několik nejlepších výsledků, čili několik nejpodobnějších proteinů.

Transkripce probíhá v souladu s pravidly přepisu DNA na RNA v jádře. Nezabýváme se tudíž sestřihem primárního transkriptu, avšak bereme v úvahu to, že ne celý vstupní řetězec může později posloužit jako RNA-templát pro syntézu proteinu. Proto je nutné nalézt ve vstupní sekvenci takovou její část, ze které je později generován protein. Tu ohraničuje start kodon a jeden z trojice stop kodonů. Po nalezení všech možných povolených (dostatečně dlouhých) sekvencí následuje překlad, neboli translace sekvence RNA na potenciální protein.

5.1.2 Heuristika hitbodů

Každý takový protein je pak podroben porovnání s každým prvkem databáze. Jedná se o porovnávání více sekvencí, ne více podsekvencí dané sekvence. Systém je podobný BLASTu. Program prochází postupně každé okno v řetězci potenciálního proteinu a porovná jej s každým stejně velkým oknem každého jednoho proteinu v databázi. Je-li nalezena shoda, je pro tuto dvojici navýšen počet hitbodů. Poté se program snaží hit prodloužit na obě strany. Je-li přitom povolena mutace, nehledá se pouze totální shoda, ale i podobnost oken, tj. mutace jedné aminokyseliny. Ovšem ze stejné skupiny. Aminokyseliny se vyskytují ve čtyřech skupinách (viz Tabulka 11). Jsou buď nepolární, polární, nebo s elektrickým nábojem (kladným a záporným). Mutace aminokyseliny je povolena pouze v rámci stejné skupiny.



Obrázek 11 – skupiny aminokyselin [17]

Pokud není nalezeno stejné okno, program zkouší okno povoleným způsobem zmutovat. Pokud se mu to podaří, rovněž se pokouší prodloužit mutovaný hit na obě strany, ovšem již bez možnosti mutace. V hitu je povoleno mutovat pouze jednou.

Čistý hit (naprosto stejné okno ve vstupním proteinu a v proteinu z databáze) je ohodnocen větším počtem hitbodů, než mutovaný hit. A jinak je také ohodnoceno prodloužení hitů. V programu je použito nastavení čistý hit: 5 hitbodů, mutovaný hit: 2 hitbody, prodloužení hitu: 3 hitbody. Prodloužení hitu lze považovat za důležité, proto je ohodnoceno lépe než nalezení mutovaného hitu. Toto lze ovšem v programu měnit dle požadavků..

Takto jsou ohodnoceny všechny dvojice potenciální protein – protein z databáze. Nyní již není větší problém vybrat nejlepších několik (v programu lze nastavit přesný počet) podobných pro každý nalezený potenciální protein. Vybrané pak projdou následným zarovnáním a bude vypočteno celkové skóre zarovnání.

5.1.3 Semiglobální zarovnání

Pro zarovnání vybraných sekvencí a výpočet jejich výsledného skóre, které v konečné fázi rozhodne o nejpodobnější, existuje několik modifikací metody Needleman-Wunsch. Tato metoda není používána při porovnávání skupin sekvencí, protože je příliš časově náročná, což samozřejmě platí také pro program PSProt. Každý potenciální protein by musel projít tímto zarovnáním se všemi databázovými proteiny a to by znamenalo velkou časovou latenci výpočtu. Avšak v programu je za pomocí heuristiky hitbodů vybráno dané množství nejlepších sekvencí, které se budou zarovnávat. Proto lze algoritmus Needleman-Wunsch nyní použít.

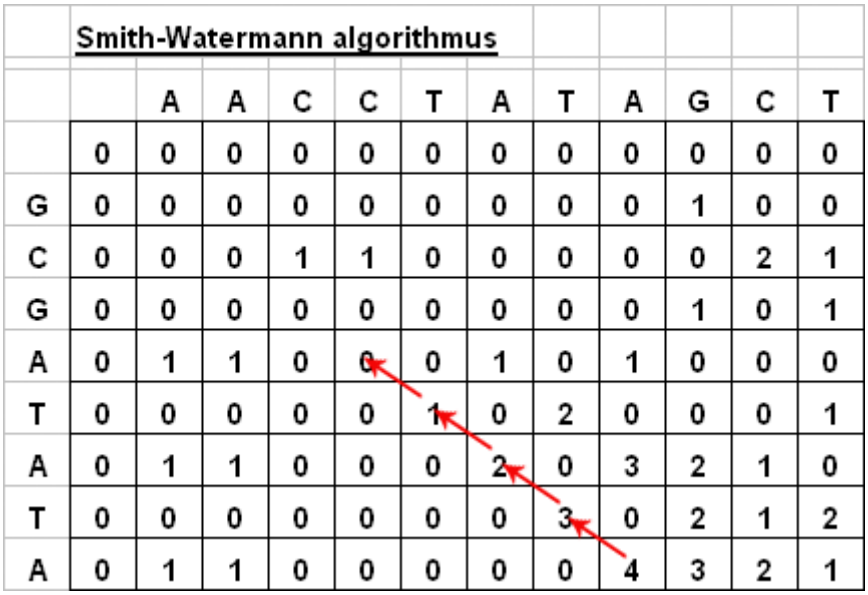
Předpokládejme zadanou vstupní sekvenci DNA (oligonukleotid) a její následný potenciální protein nesrovnatelně kratší než je skutečný protein v databázi. Zarovnání pomocí ryzího algoritumu Needleman-Wunsch by znamenalo nalezení každého jednoho prvku podsekvencí (aminokyseliny) a jejích zarovnání pod sebe. Při nesrovnatelné délce obou sekvencí by pak zarovnání vypadalo následovně:

```
K__R_T_Q_I___E_____________________________________H_______
KKKRRTKQKIIEIEIEKKKKKKKKKKAALSKKKKDENRNRMKIENHHHHHHH
```

Ve většině takových to případů, kdy lze předpokládat různé délky sekvencí, je ale lepší nalézt skupinu aminokyselin pohromadě, abychom mohli detekovat alespoň podobnou podsekvenci.

Pro nalezení nejlepší podsekvence existuje algoritmus jedna z mnoha modifikací algoritmu Needleman-Wunsch, a sice algoritmus **Smith-Watermann**. Ten najde nejlepší lokální zarovnání obou porovnávaných sekvencí, tedy nejlepší podsekvence. Po počáteční inicializaci prvního a posledního sloupce na 0 tento algoritmus přičítá jedničku při diagonálním shodě, při neshodě však stejnou hodnotu odečte. Nepovoluje pokles hodnoty jakéhokoli prvku matice do negativních hodnot. V takovém případě přiřadí poli nulu. Po výpočtu nalezne maximální prvek matice a jeho zpětnou diagonální cestou až k hodnotě 0 určí nejlepší lokální zarovnání, nejdelší shodnou podsekvenci [2].

Zbytek řetězce však víceméně ignoruje, a proto bylo od původní myšlenky zařadit tento algoritmus do výpočtu upuštěno. Přece jenom uvažujeme o hledání stejné nebo co možno nejpodobnější sekvence, ne pouze o nalezení menší sekvence jako podsekvence delší z uvažovaných.



Obrázek 12 – Smith-Waterman algoritmus

Např. při zarovnání daných sekvencí na obrázku 12 algoritmus Smith-Waterman nalezne nejpodobnější podsekvence „TATA“.

Vhodnějším kandidátem pro zarovnání programem PSProt porovnávaných sekvencí je tzv. **semiglobální zarovnání**. Jeho modifikace oproti algoritmu Needleman-Wunsch spočívá v několika málo rozdílech, konkrétně rušení pokut za vložení mezery pro některé sloupce a řádky. První sloupec a první řádek matice je opět inicializován na 0. Poslední sloupec matice a poslední řádek jsou zproštěny pokuty za vložení mezery při výpočtu hodnoty matice pro vertikální pohyb v posledním sloupci a horizontální pohyb v posledním řádku matice. Při prostém porovnávání bez mutace by postup vypadal následovně:

Obrázek 13 – Semiglobální zarovnání

Program PSProt ale provádí toto zarovnání na základně podobnosti, tj. využívá jednu ze skórovacích matic PAM a BLOSUM, které určují míru podobnosti aminokyselin, konkrétně matici BLOSUM 62. Pak by matice zarovnání vypadala takto:

Obrázek 14 – Semiglobální zarovnání se skórovací maticí

Výsledné zarovnání by tedy vypadalo takto:

RKKEEHLE

KKEEHL

6 Implementace programu PSProt

V této kapitole se seznámíme s detaily implementace programu PSProt. Program vyl vyvinut jako konzolová aplikace se vstupními parametry určujícími běh programu. Programován byl ve vývojovém prostředí Dev-C++ verze 4.9.9.2 s kompilátorem MinGW vyvinutým autory v roce 1991. Kompilován byl pod operačním systémem Windows XP.

6.1 Prerekvizity: Vstupní data

Program PSProt bude používat několik povinných vstupních souborů, ve kterých jsou uloženy informace nepostradatelné k jeho správnému běhu. Největší z nich je databázový soubor `databaze.txt` (část v příloze 1), který obsahuje implicitní databázi. V souboru `blosum62.txt`. je uložena skórovací matice pro aminokyseliny. Soubor `geneticky_kod.txt` obsahuje pravidla pro transkripci nukleotidů na aminokyseliny. Vstupní sekvence se nachází v souboru `sekvence.txt`. Nacházíme zde ještě soubor `readme.txt`, který, jak už jméno souboru napovídá, slouží k zobrazení základních informací o programu. Má stejnou funkci jako spuštění programu s parametrem –h.

6.1.1 Databáze

Databázový soubor využívaný programem PSProt musí být v předepsaném tvaru. Každý jeden prvek databáze, tedy protein, odpovídá jednomu řádku souboru. Formát, kterým je protein popsán, je podobný formátu FASTA, kterým je popsán protein v proteinové databázi UniProtKB/Swiss-Prot či Trembl:

```
>sw|P15711|104K_THEPA 104 kDa microneme/rhoptry antigen (p104).
MKFLILLFNILCLFPVLAADNHGVGPQGASGVDPITFDINSNQTGPAFLTAVEMAGVKYL
QVQHGSNVNIHRLVEGNVVIWENASTPLYTGAIVTNNDGPYMAYVEVLGDPNLQFFIKSG
DAWVTLSEHEYLAKLQEIRQAVHIESVFSLNMAFQLENNKYEVETHAKNGANMVTFIPRN
GHICKMVYHKNVRIYKATGNDTVTSVVGFFRGLRLLLINVFSIDDNGMMSNRYFQHVDDK
YVPISQKNYETGIVKLKDYKHAYHPVDLDIKDIDYTMFHLADATYHEPCFKIIPNTGFCI
TKLFDGDQVLYESFNPLIHCINEVHIYDRNNGSIICLHLNYSPPSYKAYLVLKDTGWEAT
THPLLEEKIEELQDQRACELDVNFISDKDLYVAALTNADLNYTMVTPRPHRDVIRVSDGS
EVLWYYEGLDNFLVCAWIYVSDGVASLVHLRIKDRIPANNDIYVLKGDLYWTRITKIQFT
```

Formát popisu proteinu v databázi PSProt bude následující:

```
ID proteinu | NAZEV a popis proteinu | RETEZEC aminokyselin
```

Protein je tedy charakterizován identifikátorem, jménem (a popisem) a samotným řetězcem. Všechny tři tyto instance musí být odděleny separátorem '|'.

Databáze se načítá ze souboru **„databaze.txt“**, avšak uživatel si může zvolit jiný databázový soubor, pokud má takovýto k dispozici.

Program PSProt má k dipozici databázi tvořenou proteiny syntetizované z rostliny **Arabidopsis thaliana** (Huseníček rolní).

Tato květina se používá již dlouhou dobu jako modelová rostlina pro genetické pokusy. Je to efemerní (krátkoživotní) [10] druh, proto se vystřídá v krátkém čase velký počet generací, a má jeden z nejmenších genomů v rostlinné říši. Další zajímavou vlastností je schopnost samoopylení (autogamie), lze do ní jednoduše vpravovat jiné geny [20].

Databáze těchto proteinů je doplněna o několik imaginárních proteinů z prostých testovacích důvodů.

Obrázek 15 – Huseníček rolní [18]

6.1.2 Přepisovací pravidla – genetický kód

Genetický kód je souhrn pravidel, podle kterých se genetická informace již transformovaná z DNA do RNA přepisuje na sekvenci aminokyselin- protein (viz Tabulka 1).

		Druhá báze			
		U	**C**	**A**	**G**
První báze	**U**	UUU (Phe/F)Fenylalanin	UCU (Ser/S)Serin	UAU (Tyr/Y)Tyrosin	UGU (Cys/C)Cystein
		UUC (Phe/F)Fenylalanin	UCC (Ser/S)Serin	UAC (Tyr/Y)Tyrosin	UGC (Cys/C)Cystein
		UUA (Leu/L)Leucin	UCA (Ser/S)Serin	UAA Ochre (*STOP*)	UGA Opal (*STOP*)
		UUG (Leu/L)Leucin,	UCG (Ser/S)Serin	UAG Amber (*STOP*)	UGG (Trp/W)Tryptofan
	C	CUU (Leu/L)Leucin	CCU (Pro/P)Prolin	CAU (His/H)Histidin	CGU (Arg/R)Arginin
		CUC (Leu/L)Leucin	CCC (Pro/P)Prolin	CAC (His/H)Histidin	CGC (Arg/R)Arginin
		CUA (Leu/L)Leucin	CCA (Pro/P)Prolin	CAA (Gln/Q)Glutamin	CGA (Arg/R)Arginin
		CUG (Leu/L)Leucin	CCG (Pro/P)Prolin	CAG (Gln/Q)Glutamin	CGG (Arg/R)Arginin
	A	AUU (Ile/I)Isoleucin,	ACU (Thr/T)Threonin	AAU (Asn/N)Asparagin	AGU (Ser/S)Serin
		AUC (Ile/I)Isoleucin	ACC (Thr/T)Threonin	AAC (Asn/N)Asparagin	AGC (Ser/S)Serin
		AUA (Ile/I)Isoleucin	ACA (Thr/T)Threonin	AAA (Lys/K)Lysin	AGA (Arg/R)Arginin
		AUG (Met/M)Methionin, (START)	ACG (Thr/T)Threonin	AAG (Lys/K)Lysin	AGG (Arg/R)Arginin
	G	GUU (Val/V)Valin	GCU (Ala/A)Alanin	GAU (Asp/D)Aspartát	GGU (Gly/G)Glycin
		GUC (Val/V)Valin	GCC (Ala/A)Alanin	GAC (Asp/D)Aspartát	GGC (Gly/G)Glycin
		GUA (Val/V)Valin	GCA (Ala/A)Alanin	GAA (Glu/E)Kyselina glutamová	GGA (Gly/G)Glycin
		GUG (Val/V)Valin	GCG (Ala/A)Alanin	GAG (Glu/E)Kyselina glutamová	GGG (Gly/G)Glycin

Tabulka 1 – Tabulka genetického kódu [8]

Tato pravidla se nalézají v souboru **„geneticky_kod.txt“** ve formátu AAAX, kde AAA představuje kodon (trojici nukleotidů) a X aminokyselinu, se kterou se kodon váže. Každé pravidlo se nachází na jednom řádku.

6.1.3 Skórovací matice BLOSUM 62

Program PSProt uvažuje stejně jako BLAST při porovnávání sekvencí o podobnosti, proto využívá k zarovnání skórovací matici, která zohledňuje rozdíl mezi nahrazením jednotlivých aminokyselin. Tato náhrada za prostou konstantní penalizaci za neshodu pomáhá najít lepší zarovnání, pravděpodobnější totožnost proteinu a s tím spojené i podobnější vlastnosti obou proteinů.

PSProt využívá skórovací matici BLOSUM 62, kterou využívá ve svém algoritmu i nástroj BLAST. Je to „zlatá střední cesta“. Nepředpokládáme, že program bude striktně porovnávat sekvence, u kterých se počítá s výhradně lokálními nebo globálními závislostmi proteinů.

V tabulce níže (Tabulka 2) jsou uvedeny prvky u sebe v přibližné podobnosti. Všimněme si, že např. prvky skupiny zásaditých aminokyselin (H Histidin, R Arginin, K Lysin) jsou přímo vedle sebe a skóre vzájemných porovnání není tak špatné jako s jinými prvky. Stejně tak prvky skupiny kyselých aminokyselin (D Aspartát, E kyselina glutamová) jsou ve vzájemné konfrontaci ohodnoceny skórem 2, čili dokonce kladným. V celé matici čísla udávající skóre dvou aminokyselin okolo diagonály nejsou tak malá jako ve vzdálenějších částech skórovací matice.

To je vlastnost všech skórovacích matic, nejen BLOSUM 62. Jednotlivé matice se liší ohodnocením každého pole.

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W
C	**9**	-1	-1	-3	0	-3	-3	-3	-4	-3	-3	-3	-3	-1	-1	-1	-1	-2	-2	-2
S	-1	**4**	1	-1	1	0	1	0	0	0	-1	-1	0	-1	-2	-2	-2	-2	-2	-3
T	-1	1	**4**	1	-1	1	0	1	0	0	0	-1	0	-1	-2	-2	-2	-2	-2	-3
P	-3	-1	1	**7**	-1	-2	-1	-1	-1	-1	-2	-2	-1	-2	-3	-3	-2	-4	-3	-4
A	0	1	-1	-1	**4**	0	-1	-2	-1	-1	-2	-1	-1	-1	-1	-1	-2	-2	-2	-3
G	-3	0	1	-2	0	**6**	-2	-1	-2	-2	-2	-2	-2	-3	-4	-4	0	-3	-3	-2
N	-3	1	0	-2	-2	0	**6**	1	0	0	-1	0	0	-2	-3	-3	-3	-3	-2	-4
D	-3	0	1	-1	-2	-1	1	**6**	2	0	-1	-2	-1	-3	-3	-4	-3	-3	-3	-4
E	-4	0	0	-1	-1	-2	0	2	**5**	2	0	0	1	-2	-3	-3	-3	-3	-2	-3
Q	-3	0	0	-1	-1	-2	0	0	2	**5**	0	1	1	0	-3	-2	-2	-3	-1	-2
H	-3	-1	0	-2	-2	-2	1	1	0	0	**8**	0	-1	-2	-3	-3	-2	-1	2	-2
R	-3	-1	-1	-2	-1	-2	0	-2	0	1	0	**5**	2	-1	-3	-2	-3	-3	-2	-3
K	-3	0	0	-1	-1	-2	0	-1	1	1	-1	2	**5**	-1	-3	-2	-3	-3	-2	-3
M	-1	-1	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	**5**	1	2	-2	0	-1	-1
I	-1	-2	-2	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	**4**	2	1	0	-1	-3
L	-1	-2	-2	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	**4**	3	0	-1	-2
V	-1	-2	-2	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	1	**4**	-1	-1	-3
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	**6**	3	1
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	2	-2	-2	-1	-1	-1	-1	3	**7**	2
W	-2	-3	-3	-4	-3	-2	-4	-4	-3	-2	-2	-3	-3	-1	-3	-2	-3	1	2	**11**

Tabulka 2 – Skórovací matice BLOSUM 62 [15]

6.1.4 Vstupní sekvence

Vstupní sekvenci programu lze načíst ze souboru sekvence.txt, nebo zadat přímo jako parametr programu. Pohodlnější je ovšem vzhledem k předpokládané větší délce sekvence DNA načítat tuto sekvenci ze souboru.

Formát souboru sekvence.txt je prostý, obsahuje pouze sekvenci nukleotidů A (adenin), T (thymin), C (cytosin) a G (guanin). Tuto sekvenci bychom mohli zapsat **regulárním výrazem (A+T+C+G)***. Z této sekvence se načte pouze programem definovaný maximální počet nukleotidů.

6.2 Parametry programu

Program má všechny své volitelné parametry implicitně nastaveny, ale dají se měnit dle požadavků uživatele při spuštění programu. Program PSProt má tyto nastavitelné parametry:

1, databáze ([-d jmeno_databazoveho_souboru])

Implicitně je nastavený databázový soubor databaze.txt, ve kterém se nachází proteiny syntetizované z rostliny Arabidopsis thaliana (Huseníček rolní). Pro vlastní databázový soubor platí stejné požadavky na formát, jako v databaze.txt.

2, vlastní vstupní sekvence ([-r vstupni_sekvence])

Implicitně je vstupní sekvence načítána ze souboru sekvence.txt, může však být zadána jako parametr. (viz. 6.1.4)

3, velikost okna pro porovnávání ([-o velikost_okna])

Velikost okna výrazně ovlivňuje prvotní výpočet nejpodobnějších proteinových sekvencí před zarovnáváním.

4, povolení mutace ([-m 0/1])

Pokud nastavíme mutaci (parametr do hodnoty 1), při nelézání hitu se povoluje jednu aminokyselinu z okna pro porovnávání (podsekvence) zmutovat tak, abychom dostali hit, popřípadě jej mohli prodloužit. Mutace je povolena ovšem pouze pro aminokyseliny ze stejné aminokyselinové skupiny. Mutace je implicitně v programu povolena.

5, počet nejpodobnějších proteinových sekvencí ([-t pocet])

Parametr nastavuje počet nejlepších proteinů z databáze, které se po prvotním porovnávání na základě hitbodů budou zarovnávat a přepočítávat jejich výsledné skóre. Tento parametr by logicky neměl být větší než celkový počet proteinů v databázi.

6, minimální délka pro porovnávání ([-x delka])

Parametr slouží pro selekci nalezených potenciálních proteinů ze sekvence RNA. Nemá smysl porovnávat a zarovnávat s databází protein o velikosti tří aminokyselin. Připomeňme, že do proteinu se nezapočítává aminokyselina vzniklá ze start kodonu. Uvnitř proteinu však lze kódovat další start kodon jako methyonin.

7, ohodnocení hitů ([-a hitbody_cisty_hit] [-b hitbody_prodlouzeni_hitu] [-c hitbody_mutovany_hit])

Každé nalezený hit (totožné okno, čili podsekvence porovnávaných proteinů) je ohodnocen, rovněž jeho prodloužení a mutovaný hit. To, jakou váhu má mít jaký hit, lze nastavit v programu nebo jako parametr.

8, pokuta za vložení mezery ([-g pokuta])

Pokuta za vložení mezery je jedním z rozdílových faktorů pro různé modifikace algoritmu pro porovnávání a zarovnání dvou sekvencí Needleman-Wunsch. A rovněž

je pokuta za vložení mezery jedním z určujících faktorů celkového skóre. Většinou je tato pokuta ohodnocena stržením jedničky, čili implicitně je tento parametr nastaven na 1.

9, nápověda ([-h])

Nápověda k programu.

6.3 Knihovny

Standardní funkce jsou v programu PSProt vybírány ze čtyř připojených knihoven.

Ze standardní knihovny <stdlib.h> jsou v programu použity funkce atoi( ); a itoa( ); pro řetězce na číslo a zpátky a funkce pro práci s pamětí mallloc( ); realloc ( ); free( );

Knihovna <stdio.h> obsahuje funkce pro vstupní a výstupní operace, <string.h> obsahuje funkce pro práci s celými řetězci v programu jsou použity např. strcpy( ); strcmp ( ); atd.

Knihovnu <time.h> využijeme při zjišťování aktuálního času výpočtu pro zaznamenání do výstupního souboru.

6.4 Konstanty a globální proměnné

Na začátku kódu jsou definovány symbolické konstanty, které slouží především pro ohraničení načítání dat.

```
#define ROZMER 100
```

Udává, po kolika načtených prvcích databáze se struktura databáze bude realokovat se zvětšeným paměťovým prostorem.

```
#define NOVE 100
```

Udává, pro kolik nových prvků databáze se zvětší paměťový prostor

```
#define MAX_PRVKU_DB 20000
```

Maximální rozměr databáze, maximální možný počet načtených proteinů z databázového souboru.

```
#define SEQMAX 10
```

Podobný jako ROZMER u databáze. Paměťový prostor pro potenciální proteiny z načtené sekvence se alokuje vždy po SEQMAX prvcích. Nepředpokládá se velké množství nalezených potenciálních proteinů, proto by mělo být číslo řádově o několik dimenzí menší než ROZMER.

`#define MAXDELKA 300`

Udává maximální délku načítané sekvence DNA v nukleotidech a zároveň maximální délku řetězce aminokyselin proteinu načítaného z databázového souboru.

`#define TOPMAX 100`

Udává maximální počet top porovnávaných proteinů na základě hitbodů. Skutečný počet udává proměnná TOP. TOPMAX slouží pouze jako konstanta pro počítání s vícerozměrnými poli.

Globálními proměnnými jsou nastavitelné parametry programu:

```
int povoleni_mutace = 1;
int OKNO = 4;
int TOP = 20;
int MIN_DELKA_POROVNAVANI = 3*9;
int CISTY_HIT = 5;
int PRODL_HIT = 3;
int MUTOVANY_HIT = 2;
int PENALTY_GAP = 1;
```

Rovněž také soubory `FILE *f`, se kterým jsou asociovány v průběhu programu soubory dle potřeby a logovací soubor `FILE *flog`, do kterého je zaznamenáván průběh výpočtu.

6.5 Funkce

6.5.1 Main (int argc, char *argv[])

Řídící funkce programu, definuje hlavní pole pro porovnávání a volá postupně příhodné funkce.

Po zpracování argumentů z příkazové řádky, které mohou změnit implicitní hodnoty pro porovnávání a vyhledávání, se načtou hlavní vstupní data (sekvence a databáze). Následně proběhne translace, čili překlad DNA na sekvenci RNA. Z výsledné sekvence nukleotidů reprezentující RNA se pak pomocí funkce `trim()` získají všechny povolené (minimální požadované délky) ořezané sekvence. A ty jsou pak pomocí `trankripce()` přeloženy na potenciální protein – dynamické pole `*proteiny`. Rozměr tohoto pole bude třikrát menší než rozměr pole pro ořezané sekvence. Toto vyplývá z transkripce (3 nukleotidy kódují jednu aminokyselinu).

Nyní už známe počet potenciálních proteinů i počet všech proteinů v databázi a můžeme přikročit k prvnímu porovnání, pomocí heuristiky hitbodů. Funkce `main()` zavolá jednu z hlavních

funkcí `nalezeni_hitu()`. Pole `hity[][]` bude po výpočtu obsahovat hitbody každých porovnávaných proteinů. Posléze vybereme [TOP] nejlepších (funkce `vyber()`) a jejich indexy naplníme sestupně pole `top[][]`.

Tyto vybrané sekvence pak projdou semiglobálním zarovnáním a výpočtem výsledného skóre zarovnání. Funkke zavolá druhou ze stěžejích výpočetních funkcí `zarovnani()`, která provede druhý výpočet a zapíše jej do pole `skore[][]`. Výsledky každého zarovnání se všemi detaily jsou vypsány do speciálních souborů.

Nakonec je vypsán výsledný soubor results/Results.txt, ve kterém jsou shrnuty všechny výsledky programu (funkce `vypis_hlavni_soubor()`).

6.5.2 Ziskani_sekvence (STRING seq)

Funkce `ziskani_sekvence()` slouží pro načtení vstupní sekvence DNA ze souboru sekvence.txt do pole `seq`. Načte maximálně MAXDELKA znaků první řádky souboru, přičemž každý znak odpovídá jednomu nukleotidu. Zbytek řádky ignoruje.

Provede konverzi z malých písmen na velká a zkontroluje, zda formát odpovídá struktuře DNA (pouze nukleotidy A (adenin), T (thymin), C (cystosin), G (guanin). Při nesprávném formátu sekvence DNA nahlásí chybu a skončí.

6.5.3 Nacteni_databaze (STRING **ID, STRING **NAZEV, STRING **RETEZ)

Funkce `nacteni_databaze()` zpracuje databázový soubor (cesta určena implicitně v adresáři programu databaze.txt nebo parametrem –d).

Soubor načítá řádku po řádce a funkcí `strtok()` z knihovny string.h separuje řádku pomocí oddělovače ‘|’ do tří řetězců, které znamenají identifikátor ID, název NAZEV a řetězec aminokyselin RETEZ jednoho proteinu. Délka každého ze tří řetězců je omezena parametrem maximální délky. To se týká především délky proteinu. Pokud není databáze v požadovaném tříprvkovém formátu, program PSProt zahlásí chybu, vrátí nulový počet proteinů a skončí.

Po načtení sekvence aminokyselin se provádí validace vstupu. Všechna malá písmena v řetězci se převedou na velká a kontroluje se, zda formát sekvence odpovídá struktuře proteinu, tzn., jestli obsahuje jen a pouze značky pro aminokyseliny. Prakticky jde o celou abecedu (včetně univerzálně používaného symbolu X nahrazující libovolnou aminokyselinu) kromě písmen B, J, O, U a Z. Pokud se do proteinu „vloudí“ neexistující aminokyselina, program celý tento „protein“ ignoruje.

PSProt za běhu zpracovávání databáze počítá počet načtených proteinů a pokud překročí předpokládaný počet, všechna pole se realokují na zvětšený paměťový prostor funkcí `pamet()`. Průběh realokace paměti se zaznamenává do logovacího souboru.

6.5.4 Pamet (STRING **POLE, int nova_velikost)

V případě překročení předpokládaného počtu proteinů v databázi, pro který primárně alokujeme dostatek paměti, slouží tato funkce k realokaci daného dynamického pole do většího paměťového prostoru.

PSProt funkci využívá při vytváření dynamických polí struktury databáze a při načítání potenciálních proteinů ze sekvence RNA. Nemá smysl hned na začátku alokovat obrovský paměťový prostor pro velkou databázi, když můžeme databázi načítat z vlastního databázového souboru a ten může mít libovolný (malý) počet proteinů. Stejně tak nemůžeme předem vědět, jakou sekvenci DNA bude program zpracovávat a kolik proteinů může úsek DNA syntetizovat.

6.5.5 Translace (STRING seq)

Funkce přeloží sekvenci DNA v souladu s principem komplementarity na RNA řetězec. Thymin je nahrazen uracylem.

6.5.6 Trim (STRING seq, STRING **trim_seq)

Správně načtená sekvence a přeložená sekvence RNA se pak podrobí několika úpravám. Funkce `trim()` najde v sekvenci potenciální proteiny minimální požadované délky (princip nalezení viz. Tabulka 3). Všechny tyto nalezené proteiny uloží do pole ořezaných sekvencí. Vrátí celkový počet nalezených proteinů.

Pole ořezaných sekvencí `trim_seq` je rovněž polem dynamickým a PSProt operuje s pamětí stejně, jako u načítání databázových proteinů. Čili nejprve alokuje menší paměť pro předpokládaný počet získaných ořezaných sekvencí a je-li jich více, realokuje celé pole v paměti na větší paměťový prostor.

Nalezení start a stop kodonů je zaznamenáváno do logovacího souboru.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
původní sekvence																								
A	**A**	**A**	**U**	**G**	**C**	**A**	**G**	**G**	**A**	**U**	**G**	**C**	**U**	**G**	**A**	**C**	**G**	**A**	**C**	**U**	**G**	**A**	**A**	**\0**
testují se tři nukleotidy																								
A	**A**	**A**	**U**	**G**	**C**	**A**	**G**	**G**	**A**	**U**	**G**	**C**	**U**	**G**	**A**	**C**	**G**	**A**	**C**	**U**	**G**	**A**	**A**	**\0**
Nalezen start kodon na pozici 2																								
A	**A**	**A**	**U**	**G**	**C**	**A**	**G**	**G**	**A**	**U**	**G**	**C**	**U**	**G**	**A**	**C**	**G**	**A**	**C**	**U**	**G**	**A**	**A**	**\0**
pohybujeme se po třech polích, hledá se stop kodon																								
A	**A**	**A**	**U**	**G**	**C**	**A**	**G**	**G**	**A**	**U**	**G**	**C**	**U**	**G**	**A**	**C**	**G**	**A**	**C**	**U**	**G**	**A**	**A**	**\0**
nalezen stop kodon na pozici 20																								
A	**A**	**A**	**U**	**G**	**C**	**A**	**G**	**G**	**A**	**U**	**G**	**C**	**U**	**G**	**A**	**C**	**G**	**A**	**C**	**U**	**G**	**A**	**A**	**\0**
pokud je délka potenciálního proteinu větší nebo rovna minimální požadované délce, byl nalezen první potenciální protein																								
A	**A**	**A**	**U**	**G**	**C**	**A**	**G**	**G**	**A**	**U**	**G**	**C**	**U**	**G**	**A**	**C**	**G**	**A**	**C**	**U**	**G**	**A**	**A**	**\0**
hledání pokračuje další možnou počáteční pozicí																								
A	**A**	**A**	**U**	**G**	**C**	**A**	**G**	**G**	**A**	**U**	**G**	**C**	**U**	**G**	**A**	**C**	**G**	**A**	**C**	**U**	**G**	**A**	**A**	**\0**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
nalezen start kodon, ale i když jsme na poslední pozici, nebyl nalezen stop kodon																								
A	**A**	**A**	**U**	**G**	**C**	**A**	**G**	**G**	**A**	**U**	**G**	**C**	**U**	**G**	**A**	**C**	**G**	**A**	**C**	**U**	**G**	**A**	**A**	**\0**
hledání pokračuje další možnou počáteční pozicí																								
A	**A**	**A**	**U**	**G**	**C**	**A**	**G**	**G**	**A**	**U**	**G**	**C**	**U**	**G**	**A**	**C**	**G**	**A**	**C**	**U**	**G**	**A**	**A**	**\0**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
konec prohledávání (nepodaří se testovat celou trojici nukleotidů, resp. 3.nukleotid neexistuje)																								
A	**A**	**A**	**U**	**G**	**C**	**A**	**G**	**G**	**A**	**U**	**G**	**C**	**U**	**G**	**A**	**C**	**G**	**A**	**C**	**U**	**G**	**A**	**A**	**\0**

Tabulka 3 – Nalezení potenciálního proteinu

6.5.7 Transkripce (STRING trim_seq, STRING_3 **proteiny, int pocet_seq)

Funkce převede všechny ořezané sekvence na potenciální proteiny (sekvenci aminokyselin). Těmito sekvencemi je naplněno dynamické pole proteiny, které bylo předtím alokováno, jelikož již je znám počet potenciálních proteinů.

6.5.8 Nalezeni_hitu (STRING_3 **proteiny, STRING **RETEZ, int *hits, int pocet_seq, int pocet_proteinu_db)

Stěžejní funkce heuristiky hitbodů.

Všechny potenciální proteiny se porovnají s databázovými proteiny a míra jejich shody je zaznamenána do tabulky hitů. Ve dvou cyklech se porovnávají postupně všechny podsekvence

aminokyselin obou proteinů, ohodnocení shody je zadáno parametry CISTY_HIT, PRODL_HIT a MUTOVANY HIT. Systém porovnávání podsekvencí je přiblížen v tabulce 4.

(1) Porovnání oken		
T K K R R T S G …	HITBODY:	0
S K K R R T T E …		
(2) Neshoda, posunutí okna v 2. proteinu		
T K K R R T S G …	HITBODY:	0
S K K R R T T E …		
(3) … okno po projetí celého databázového proteinu, porovnává se další okno…		
T K K R R T S G …	HITBODY:	0
S K K R R T T E …		
(4) Nalezení hitu		
T K K R R T S G …	HITBODY:	5
S K K R R T T E …	(přičtení bodů za čistý hit +5)	
(5) Pokus o prodloužení nalevo, neshoda		
T K K R R T S G …	HITBODY:	5
S K K R R T T E …		
(6) Pokus o prodloužení napravo, hit!		
T K K R R T S G …	HITBODY:	8
S K K R R T T E …	(přičtení bodů za prodloužený hit +3)	
(7) Pokus o další prodloužení napravo, neshoda		
T K K R R T S G …	HITBODY:	8
S K K R R T T E …		
(8) Pokračování posunutím okna v databázovém proteinu, proces se opakuje		
T K K R R T S G …	HITBODY:	8
S K K R R T T E …		

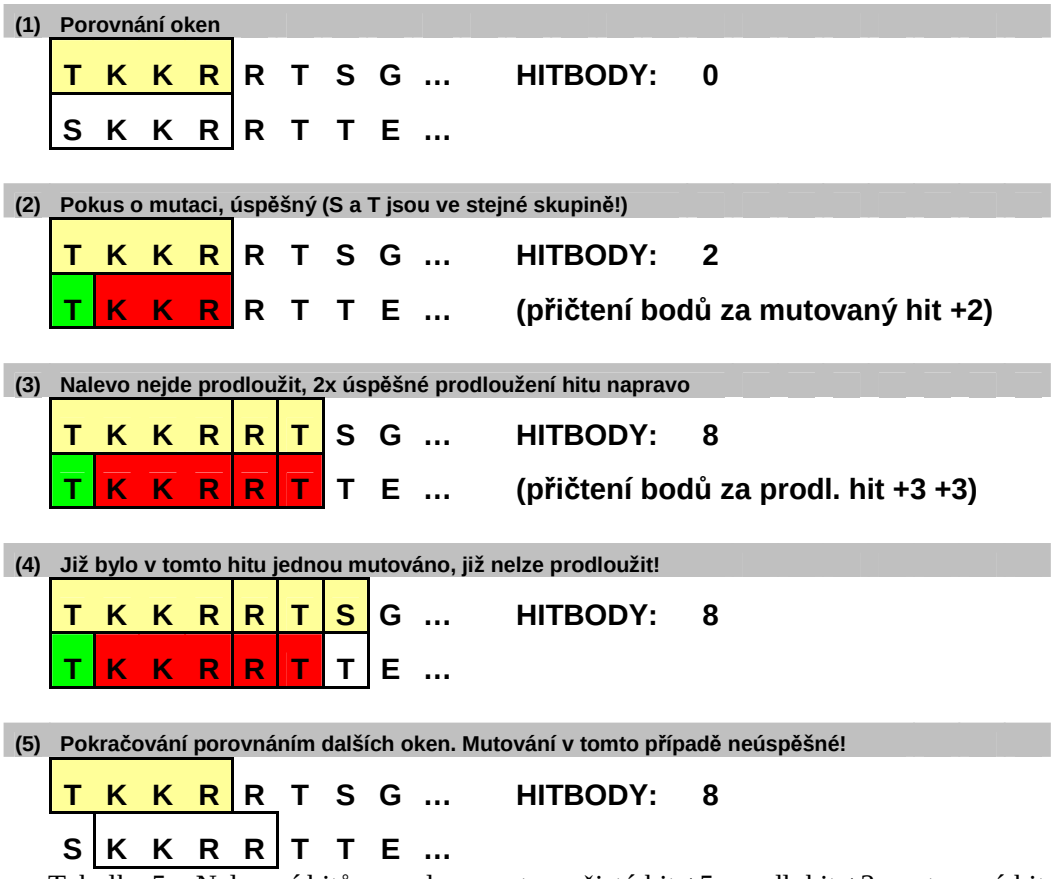
Tabulka 4 – Nalezení hitů, bez mutace, čistý hit: +5, prodloužený hit +3

Velikost porovnávaných oken je v tomto případě (4). Je nalezen hit na pozici 1 v potenciálním proteinu a na pozici 1 v databázovém proteinu. Po nalezení tohoto hitu se PSProt snaží tento hit prodloužit na obě strany, je- li to možné. Po úspěšném prodloužení jsou přičteny další hitbody.

Při porovnávání v dalším průběhu nastane situace, že bude nalezen hit na 2. pozici v potenciálním proteinu a 2. pozici v databázovém proteinu. Po úspěšném prodloužení doleva se dostaneme do stejné situace jako v tabulce na řádku (6). Tato redundance výpočtu ale vůbec

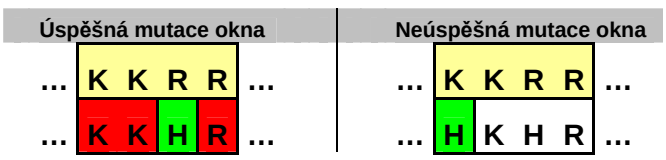
nevadí, jelikož se pouze znovu započítají hitbody za prodloužení hitu. Delší hit má tedy o to větší váhu než několik menších hitů dále od sebe. To je ovšem jen správně a tento jev je ve vyhledávání PSProtem pozitivem.

V tabulce 4 je načrtnuto porovnávání bez povolení mutace. Pokud je mutace povolena, dochází k změnám při výpočtu hitbodů. V zásadě je systém výpočtu stejný, při jednotlivých porovnání ovšem při neúspěchu dochází k pokusu o mutaci. Buď již porovnávaného okna, nebo při čistém hitu v prodlužování okna (viz. Tabulka 5).



Tabulka 5 – Nalezení hitů, povolena mutace, čistý hit +5, prodl. hit +3, mutovaný hit +2

Mutaci okna lze provést samozřejmě i v jiné než první pozici uvnitř okna. Vždy ale jen jednu aminokyselinu a to ze stejné skupiny.



Tabulka 6 – Mutace okna

6.5.9 Vyber (int *pole, int max1, int max_pole, int *vystup_pole)

Funkce `vyber()`, mapování indexů nejlepších prvků, je volaná dvakrát a slouží k prostému vybrání nejlepších prvků ze vstupního pole. Poprvé je volána pro výběr TOP nejlepších podobných proteinů z databáze po heuristice hitbodů, podruhé pro přehodnocení pořadí po výsledném skóre každého zarovnání.

6.5.10 Zarovnej (STRING seq, STRING *trim_seq, int *hity, STRING *ID, STRING *NAZEV, STRING *RETEZ, STRING_3 *proteiny, int *top, int pocet_seq, int skore[])

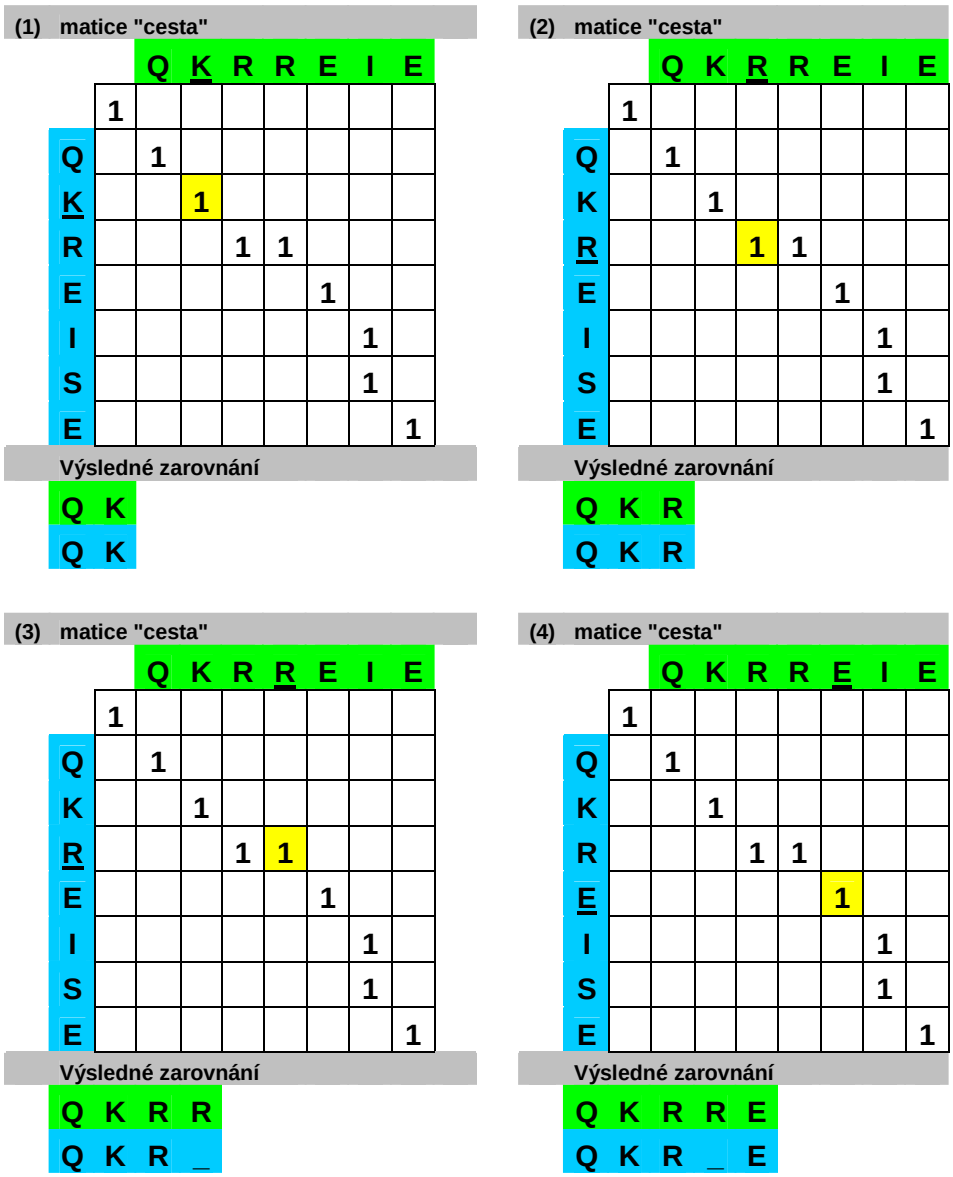
Druhá nejdůležitější funkce PSProtu. Realizuje algoritmus semiglobálního zarovnání, de facto lehce modifikovaný Needleman-Wunsch.

Zarovnání se neprovádí pro všechny proteiny z databáze, ale pouze pro několik vybraných, nejlepších na základně heuristiky hitbodů. Pro zaznamenávání hodnot slouží dvojrozměrná pole MATICE – záznam skóre, pomMATICE – záznam indexu pole při zpětné hledání cesty, booleovská dvojrozměrná matice cesta – výsledná cesta pro následující textové zarovnání porovnávaných proteinů do pole zarovnani[].

Inicializace prvních řádků tabulky je zřejmá. Pro matici MATICE první sloupec a řádek na 0 (semiglobální zarovnání), pomMATICE pak logicky na předchozí buňku ve sloupci či řádku. Řádek po řádku se pak počítá obsahu pole MATICE jako maximum ze tří nabídnutých možností (horizontální pohyb + pokuta, vertikální pohyb + pokuta, diagonální pohyb + hodnota načtené skórovací matice BLOSUM 62). Do pomocné pomMATICE se pak zaznamená index pole s největším z porovnávaných prvků, přičemž se dává přednost diagonálnímu pohybu. Důsledkem takovéto vyhodnocení je co nejmenší počet vkládaných mezer a přímější zarovnání sekvencí. Kontroluje se přitom, zda již nejsme na posledním řádku či v posledním sloupci matic, což by mělo za důsledek při pohybu v tomto řádku/sloupci zrušení penalizace za vložení mezery.
Výsledným skórem po zarovnání je pak pravý dolní roh MATICE. Od toho se také odvíjí cesta při zpětném hledání zarovnání. V matici pomMATICE se v pravém dolním rohu nachází adresa pole v matici MATICE, odkud jsme se na poslední pole dostali. Takto postupujeme pomMATICÍ až k levému hornímu rohu, přičemž v každém poli zaznamenáme v poli cesta 1.

Obecně můžeme najít více nejlepších zarovnání se stejným (nejlepším) výsledným skóre, ale PSProt zaznamenává pouze jednu cestu ve snaze najít co nejpřímější zarovnání.

Nakonec se prochází maticí cesta „po jedničkách“ a podle toho se zarovnávají proteiny do výsledné podoby, dle tabulky 7.



Tabulka 7 – Zarovnávání dle matice cesty

Pokud je nalezena jednička v diagonálním směru, zapíší se na pozici příslušné aminokyseliny do obou proteinů a jsou inkrementovány pozice aminokyseliny obou proteinů. Pokud je nalezena jednička ve vertikálním směru, zapíše se v jednom proteinu mezera a ve druhém aminokyselina, přičemž se pozice u proteinu, do kterého byla vložena mezera, neinkrementuje.

Výsledky každého zarovnání včetně všech průběžných tabulek a výpočtů se zaznamenávají do výstupního souboru porovnání daných proteinů, průběžné pomocné výpočty do logovacího souboru.

6.6 Výstup

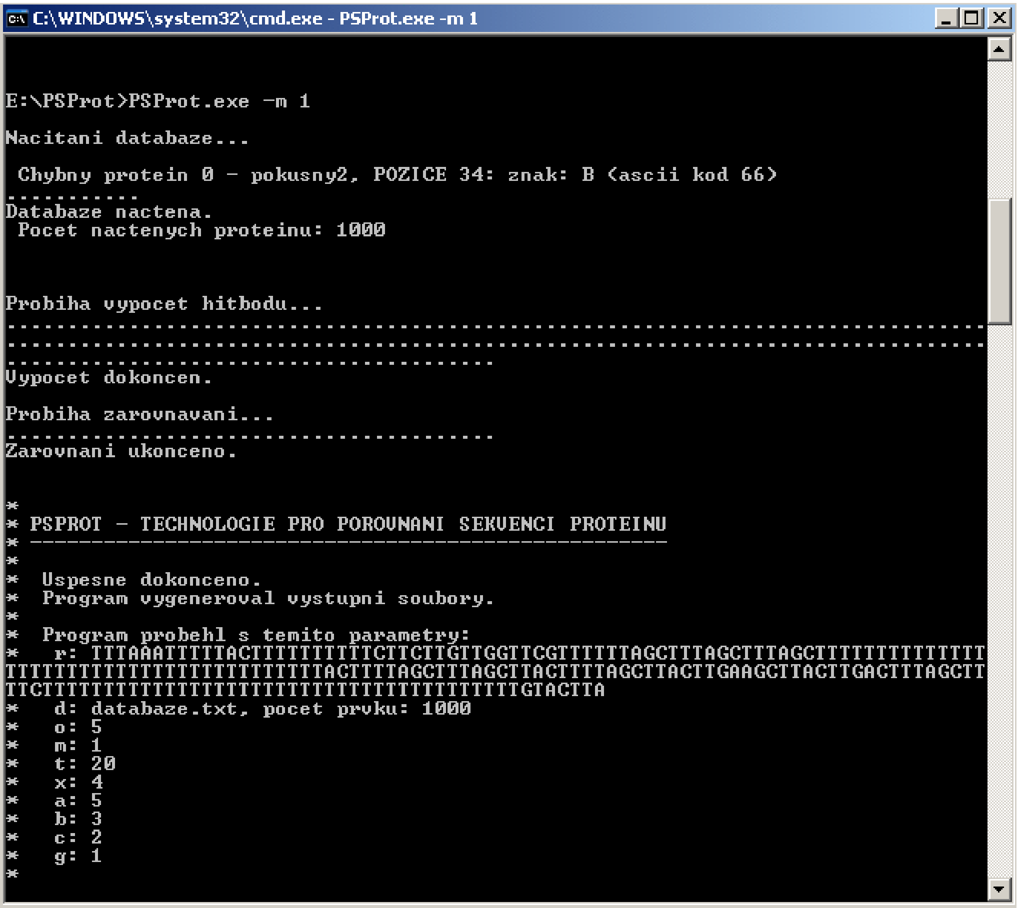
PSProt generuje výstupový adresář results/. Výstup je realizován funkcemi `vypis_soubor()` a `vypis_hlavni_soubor()`.

Hlavní výstupní soubor **Results.txt** obsahuje nejdůležitější výstupní informace celého programu v přehledné formě (viz. Příloha 2). Je zde zobrazena vstupní sekvence DNA, z kterého souboru jsou získány proteiny pro porovnávání (a jejich načtený počet), parametry, s jakými výpočet a porovnání proběhl. A konečně pro každý potenciální protein (a jeho původní templát RNA) seznam nejpodobnějších nalezených proteinů z databáze a jejich porovnávací hitbody a skóre. Připojeno je i jméno souboru, ve kterém se nachází detaily porovnávání. Toto jméno se skládá z čísla potenciálního proteinu a pořadí v nejlepších nalezených proteinů k potenciálnímu.

V detailních souborech jsou pak informace doplněny o výsledné zarovnání obou proteinů a výpočetní matice (matice skore, matice pomMATICE, matice cesty). Viz. příloha 3.

Do logovacího souboru results/log.txt jsou během celého běhu výpočtu zaznamenávány detaily výpočtu, které by byly v hlavních výstupních souborech nicneříkající, avšak slouží pro kontrolu výpočtu (viz. příloha 4).

Kromě toho informuje PSProt o průběhu výpočtu přímo ve standardním výstupu (viz. Obrázek 15).



Obrázek 15 – Výstup

6.7 Řešené problémy při implementaci

Při implementaci jsem řešil několik problémů spojených ať už s programovacím jazykem C, realizováním algoritmů, či desinterpretací biologických faktů.

- Realizace mutace
- Prodlužování hitu za hranice sekvence
- Neexistující aminokyselina X
- Rozměry matic pro zarovnání
- Příliš dlouhý řádek ve vstupním databázovém souboru
- Způsob ukončení načtené sekvence aminokyselin z databáze
- Malá databáze v porovnání s požadovaným počtem nejpodobnějších proteinů

6.7.1 Realizace mutace

Při porovnávání oken a nenalezení hitu se algoritmus snaží o mutaci okna. To znamená pokus o změnu prvního prvku skupiny tak, aby se shodoval s prvkem na stejné pozici druhého okna, a přitom byl ze stejné skupiny aminokyselin. Nejprve jsem tedy určil pomocí pomocné funkce `zjisti_skupinu()` příslušnou skupinu, kam prvek patří, a postupně porovnával se všemi prvky skupiny kromě prvku samotného. Pokud jsem nalezl shodu (možná mutace), bylo porovnáno zmutované okno s druhým. Pokud se okna rovnala, nalezl jsem hit. Pokud ovšem ne, musel jsem pokračovat v cyklu pro další prvek okna (druhý). Teprve až jsem takto prošel všechny prvky okna a přitom nenalezl mutovaný hit, mohl jsem prohlásit mutaci za neúspěšnou. Řešením je použití proměnných KONEC a MUTOVÁNO a smyčky while za podmínky (! KONEC). Při úspěchu některého z těchto porovnávaných zmutovaných oken jsem nastavil proměnnou MUTOVANO na true, KONEC na false. Při neúspěchu zůstane KONEC na true a porovnávání těchto oken skončí celkovým neúspěchem. K inkrementaci hit se dostane buď rovností oken nebo přes nastavení proměnné MUTOVANO na true při úspěšné mutaci okna.

Úryvek algoritmu mutace v příloze 5.

6.7.2 Prodlužování hitu za hranice sekvence

Chyba odhalena v průběhu po prozkoumání logovacího souboru. Při nalezení hitu na pokraji jedné či druhé sekvence se PSProt snaží porovnat prvky vedle hitu. Např. byly-li oba hity na konci pole, pak se porovnávaly i prvky ‘\0’ a hit byl chybně prodloužen. Dodatečně pak byly do kódu doplněny podmínky na pozici v prodlužovaného hitu v jednom či druhém proteinu.

6.7.3 Neexistující aminokyselina X

V původní skórovací matici blosum62 [16] bylo pouze 20 aminokyselin, avšak v některých proteinech se objevovala aminokyselina X. Při výpočtech matice částečného skóre se pak projevila chyba při vypočítávání diagonální možnosti přechodu, připočítání neinicializovaného místa v paměti (nebylo nalezeno ohodnocení pro dvojici [X, druhá aminokyselina]). Toto číslo bylo největší ze všech a vedlo tak k deformaci všech matic (skóre, pomocné matice i matice cesty). Do matice BLOSUM (soubor blosum62.txt) byl tedy přidán řádek X podle jiné verze této skórovací matice [15], která již s ní počítá. V některých publikacích se totiž písmenem X označuje i neznámá nebo nespecifikovaná aminokyselina [19].

6.7.4 Rozměry matic pro zarovnání

Matice pro výpočet skóre jsou co do obou rozměrů vždy o 1 delší než je délka obou proteinů, jejich aminokyseliny tvoří záhlaví. A to díky vložení počátečního nulového sloupce a řádku. Chyba nezapočítání tohoto inicializačního řádku a sloupce se pak projevila předčasným chybovým ukončením běhu programu již při výpočtu matice skóre nebo chybami při tvorbě zarovnání, neboť poslední řádek a sloupce matice byly ignorovány. Při výpočtu totiž mimo jiné přistupujeme k matici cesty jako k jednorozměrnému poli a počítáme s délkou matice. Chybná délka řádku matice (zmenšená o 1) se tudíž projevovala s každým dalším řádkem výrazněji.

6.7.5 Příliš dlouhý řádek ve vstupním databázovém souboru

Dlouhý řádek databáze funkce `fgets()` dostatečně správně nezpracuje, jelikož přesahuje délku řádku. Resp. zpracuje maximální počet znaků a zbytek řádky zpracuje jako nový řádek, čili další protein. To ovšem způsobí chybu formátu databáze, protože nově vzniklý řádek s největší pravděpodobností neobsahuje dva oddělovače. Proto musíme při načítání příliš dlouhého řádku po načtení třetí instance proteinu přečíst zbytek řádky až do znaku ‘\n’ ručně. Jednoduchým cyklem

```
while (getc(f) != '\n')
      ;             // prazdna smycka
```

6.7.6 Způsob ukončení načtené sekvence aminokyselin

Každý protein v je v databázovém souboru uložen na jednom řádku. Řádek se načítá stylem načtení celého řádku (`fgets()`) s daným počtem znaků. Pokud je řádek kratší a ve správném formátu, do třetího hlavního proteinového pole RETEZ se správně načte zbytek řádku po druhém oddělovači '|'. Funkce `fgets()` ovšem načte celý řádek i s ukončovacím znakem '\n', avšak je-li načítaný řádek delší, při kopírování do pole RETEZ funkcí `strncpy()` načteme pouze znaky (aminokyseliny), jelikož v tomto případě řádek není ukončen '\n'. Proto pro ukončení vždy zjistíme délku řetězce RETEZ (buď podmínkou na ukončovací znak '\n' nebo je počet stanoven na MAXDELKA) a ukončíme jej standardním ukončovacím znakem řetězce '\0'.

6.7.7 Malá databáze v porovnání s požadovaným počtem nejpodobnějších proteinů

Jestliže požadujeme nejpodobnějších 10 proteinů z databáze a databáze obsahuje pouze 8 proteinů, program by skončil chybou, protože by již na 9. nejlepší pozici neměl protein pro hodnocení. Tato chyba byla eliminována zarážkou počtu požadovaných proteinů maximálně na počet proteinů v databázi. Tato chyba by se v praxi pravděpodobně asi nevyskytla, jelikož databáze s proteiny by měly obsahovat nesrovnatelně více proteinů než bude požadovaný počet nejpodobnějších nalezených.

7 Vstupní testy

Vstupní test 1	**Vstup databáze**
Problém	Vstupní soubor neexistuje
Popis	Chybné zadání vstupního databázového souboru.
Výstup	`Soubor neexistujici_db.txt se nepodarilo otevrit!` `Databaze je prazdna!`
	Nevytvořen žádný výstupní soubor.

Vstupní test 2	**Vstup databáze**
Problém	Prázdný soubor
Popis	Databázový soubor je prázdný.

Výstup	`Nacitani databaze...`
	`Databaze nactena.`
	` Pocet nactenych proteinu: 0`
	`Databaze je prazdna!`

	Vytvořen pouze soubor log.txt s obsahem: `Pocet nactenych proteinu z databaze: 0`

Vstupní test 3	**Vstup databáze**
Problém	Data v nesprávném formátu
Popis	Databázový soubor neodpovídá formátu, libovolný soubor, na kterém nejsou minimálně dva oddělovače na řádce.

Výstup	`Nacitani databaze...`
	`Databazovy soubor neni v pozadovanem formatu!`
	`Databaze je prazdna!`

	Vytvořen pouze prázdný soubor log.txt.

Vstupní test 4	**Vstup databáze**
Problém	Data v nesprávném formátu
Popis	Na řádce jsou více než dva oddělovače. Případná chyba formátu proteinu na jednom řádku.
Výstup	Výpočet proběhne v pořádku, pokud jsou první tři oddělené prvky na řádku ve správném formátu, zbytek řádky se ignoruje. Pokud data nejsou v požadovaném formátu, program daný celý řádek ignoruje. `Chybny protein prot1 - chybny, POZICE 4: znak: Z (ascii kod 90)`
	Výpočet proběhne v pořádku se všemi výstupními soubory.

Vstupní test 5	**Vstup sekvence**
Problém	Data v nesprávném formátu nebo soubor neexistuje.
Popis	Řádka v souboru sekvence.txt či parametr -r obsahuje jiné znaky než znaky pro nukleotidy A, C, G, T (velká i malá písmena) nebo vstupní soubor neexistuje a zároveň nebyla zadána vstupní sekvence z příkazové řádky.

Výstup	`CHYBA POZICE 9: znak: X (ascii kod 88)`
	`Chyba pri nacitani sekvence!`
	`Sekvence neni ve spravnem formatu!`

	Nevytvořen žádný výstupní soubor. Zobrazí se informace o chybě.

Vstupní test 6	**Vstup sekvence**
Problém	Nenalezen potenciální protein
Popis	Sekvence nukleotidů RNA neobsahuje žádný start kodon nebo žádný stop kodon nebo nejsou v takovém postavení, aby mezi sebou uzavírali dostatečně dlouhý později syntetizovaný potenciální protein.

Výstup	`Nebyl nalezen ani jeden potenialni protein!`
	`log.txt:`
	`Nalezen start kodon na pozici 0`
	`Pocet vzniklych potencialnich proteinu ze vstupni sekvence: 0`

	Výpočet proběhne v pořádku, je vytvořen soubor Results.txt a log.txt s nulovými výsledky.

Vstupní test 7	**Výpočet**
Problém	Chyba v požadavcích
Popis	Je větší počet zadán požadovaných nejpodobnějších proteinů než obsahuje celá databáze.
Výstup	
	Výpočet proběhne v pořádku, pouze počet požadovaných nejpodobnějších proteinů se automaticky sníží na celkový počet řádně načtených proteinů z databáze.

8 Experimenty

Výsledky programu PSProt závisí na zadaných vstupních hodnotách a parametrech. Pro experimenty byla zvolena sekvence nukleotidů, ze které se může syntetizovat potencionální protein: `KKKRRTTKQKIEIEIEKKKKKKKKKKKKNENRNRMKIE`. Do databáze byly vloženy imaginární proteiny upravené tak, aby ověřovaly přesnou funkci programu a funkčnost se dala ověřit na vypočítaných výsledcích:

```
ImaginareFITvlozenoI|temer presny protein|
KKKRRTTKQKIIEIEIEKKKKKKKKKKKKNENRNRMKIE
ImaginareFITworst|insertovany presny protein|
KKKRRTTKQKIIEIEIEKKKKKKKKKAALSKKKKDENRNRMKIE
ImaginareFIT|presny protein|
KKKRRTTKQKIEIEIEKKKKKKKKKKKKNENRNRMKIE
```

ImaginareFIT je totožný protein jako porovnávaný, ImaginareFITvlozenoI je téměř rovněž identický, ovšem je do něj vložena aminokyselina isoleucin (I). Protein ImaginareFITworst vyjadřuje nejhorší ze tří uměle pro experimenty vytvořených proteinů, je do něj vložena ještě podsekvence několika aminokyselin navíc.

PSProt by měl tedy vyhodnotit jako nejpodobnější protein přesný protein ImaginareFIT, jako druhý nejpodobnější pak ImaginareFITvlozenoI a třetí v pořadí by se měl objevit ImaginareFITworst. Pokud se tedy jinde v databázi nenachází podobnější protein.

Výsledek spuštěný nad databází s výše uvedenými vstupy a parametry -o 4 -m 1 -t 20 -x 4 -a 5 -b 3 -c 2 -g 1:

```
*  Nejpodobnejsi proteiny
*  -- 1, ImaginareFIT, presny protein,  5126 hitbodu, skore: 187
*  -- 2, ImaginareFITvlozenoI, temer presny protein, 3666 hitbodu, skore: 186
*  -- 3, ImaginareFITworst, insertovany presny protein, 1884 hitbodu, skore: 176
*  -- 4, At4g11385.1, hypothetical protein,  4163 hitbodu, skore: 102
...
```

Program tedy správně určil již na základně heuristiky hitbodů jako nejpodobnější protein ImaginareFIT, přesný protein s 5126 hitbody, i další pořadí určil správně. Hitbody jsou ve větším rozpětí než skóre zarovnání. Po vložení aminokyseliny I totiž dojde k porušení velkého („kompletního proteinového“) hitu. Na skóre se vložení projeví vložením mezery a tedy penalizací za toto vložení. Tato pokuta je nastavena na 1 (parametr –g 1) a proto má ImaginareFITvlozenoI pouze o 1 horší skóre než ImaginareFIT.

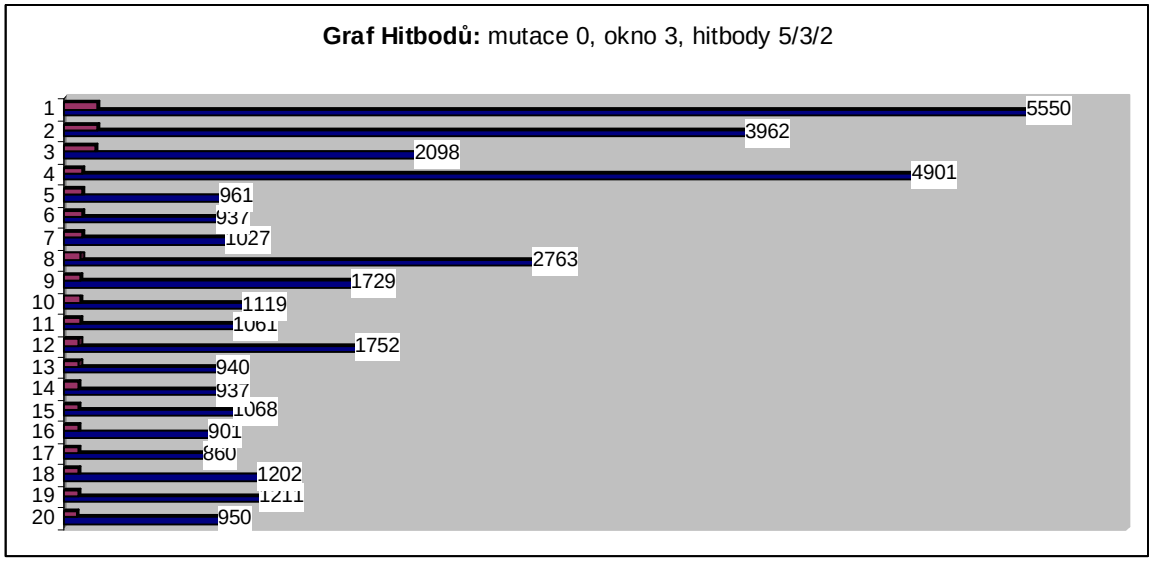
Jako čtvrtý nejpodobnější byl nalezen At4g11385.1, hypothetical protein, se značně menším skóre, ovšem velkým počtem hitbodů. To je způsobeno tím, že obsahuje velkou sekvenci aminokyseliny K (kyselina glutamová), stejně jako v porovnávaném potenciálním proteinu. Bude proto vybrán do nejpodobnějších proteinů, ale při zarovnání dojde k velkým ztrátám za vložené

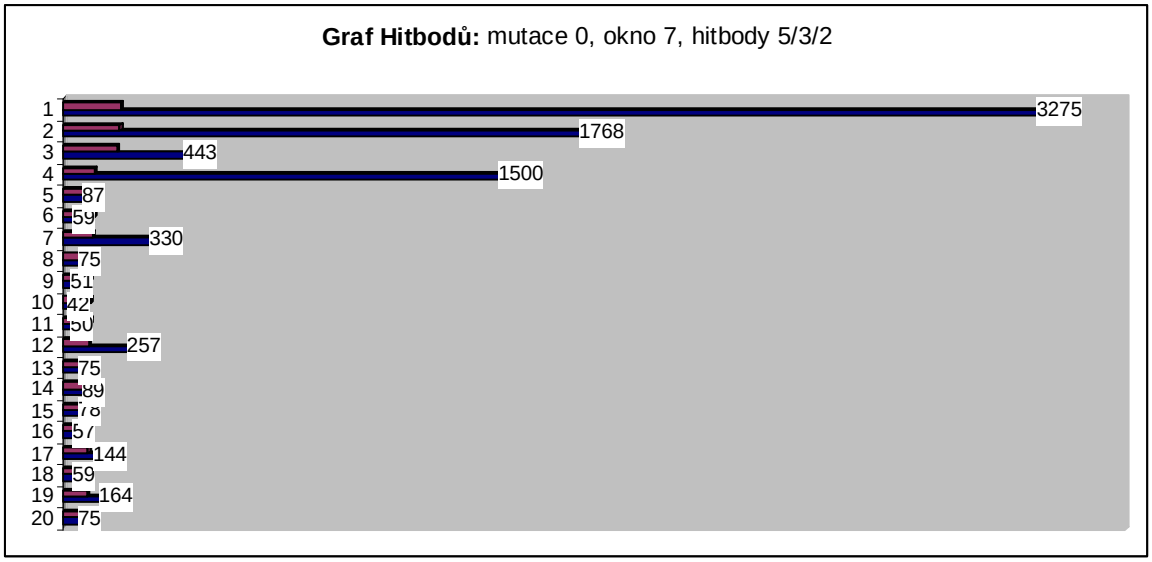
mezery. Výsledné zarovnání tohoto proteinu je semiglobálním zarovnáním vměstnáno do co nejmenšího prostoru v okolí největší stejné podsekvence. Jelikož je ale Atlg01810.1 hypothetical protein poměrně o hodně delší, dochází ve výpočtu skóre za vložení mezer k velké penalizaci, proto je skóre menší než by heuristika hitbodů předpovídala.

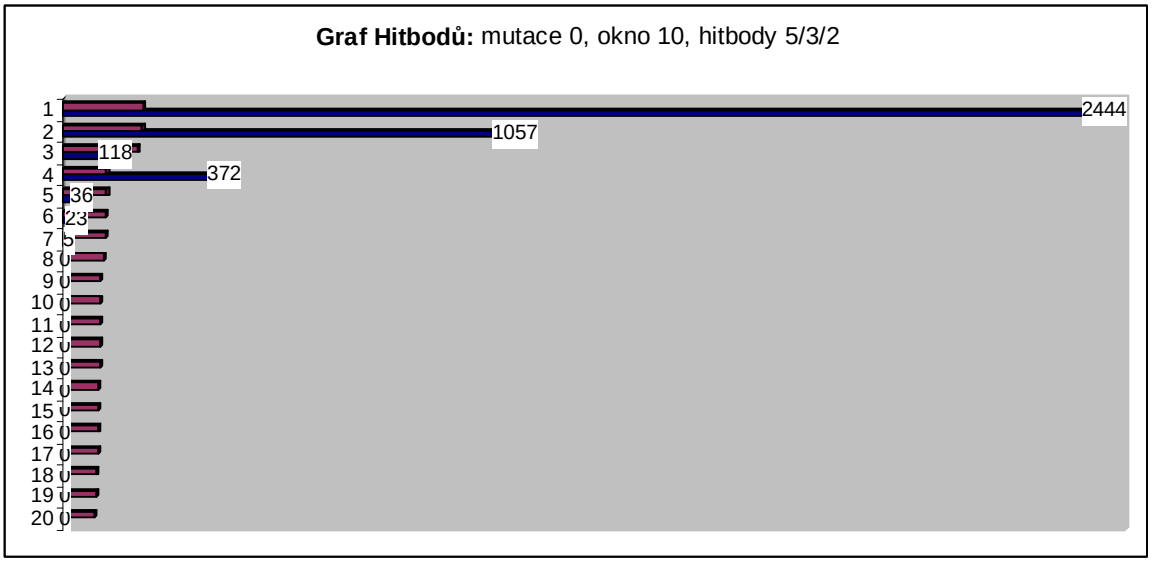
Správnost výpočtu byla tedy tímto ověřena.

8.1 Výpočet bez mutace

Výpočet hitbodů a následná regulace semiglobálním zarovnáním spolu souvisí. Hitbody jsou hrubým výpočtem a selektují nejpodobnější proteiny, které projdou zarovnáním. V následujících grafech je vybráno 20 top nejpodobnějších nalezených proteinů z databáze, seřazených podle výsledného skóre zarovnání (oranžový sloupec). Hodnoty modrého sloupce v grafu znamenají počet předvypočtených hitbodů. Cílem výpočtu bylo zjistit vztah mezi pořadí na základě heuristiky hitbodů a skóre po semiglobálním zarovnání při různé velikosti okna.







Obrázky 16 – 18: Grafy hitbodů při různých délkách porovnávacího okna bez mutace

V grafech na obrázcích 16-18 vidíme, že se zvětšujícím se oknem pro porovnání se selektují jen proteiny, které v sobě obsahují minimálně stejně dlouhé podsekvence, aby dostaly nějaké hitbody. Ve všech případech (okno 3, 5, 10) je na prvních čtyřech místech stejná nejpodobnější čtveřice. Ostatní porovnávané proteiny znamenají víceméně náhodné hity, a proto jsou po úpravě semiglobálního zarovnání seřazeny jinak, ovšem to není tolik podstatné, jako to, že nejpodobnější proteiny se drží v popředí. Po zvětšení okna nad 15 již ztrácíme téměř všechny ostatní porovnávané proteiny a zůstanou pouze ImaginareFIT, přesný protein a ImaginareFITvlozenoI, u ostatních bylo zjištěno 0 hitbodů a proto je další pořadí již nevýznamné.

První čtyři místa na všech grafech obsadily předpokládané proteiny v pořadí ImaginareFIT, ImaginareFITvlozenoI, ImaginareFITworst, At4g11385.1. Další pořadí není podstatné.

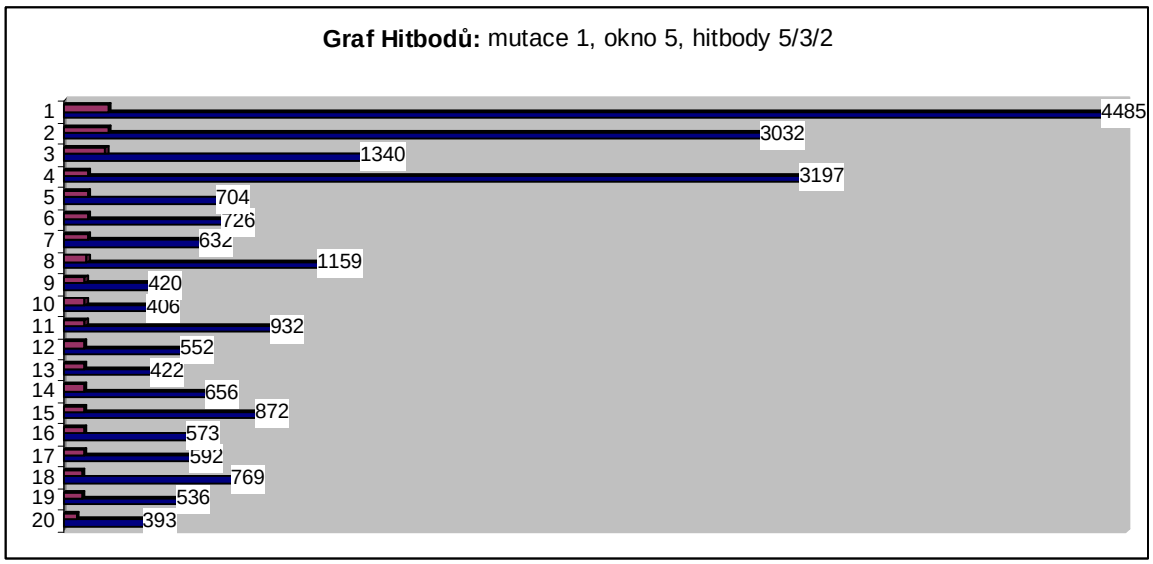
Samotný výpočet hitbodů trval pro 20 000 databázi proteinů a jeden nalezený potenciální protein u malých oken přibližně 11 vteřin, u okna o velikosti 10 trvá něco pod 6 vteřin. Při neuvažované mutaci tedy dostáváme výsledek velmi rychle pro jakékoli okna.

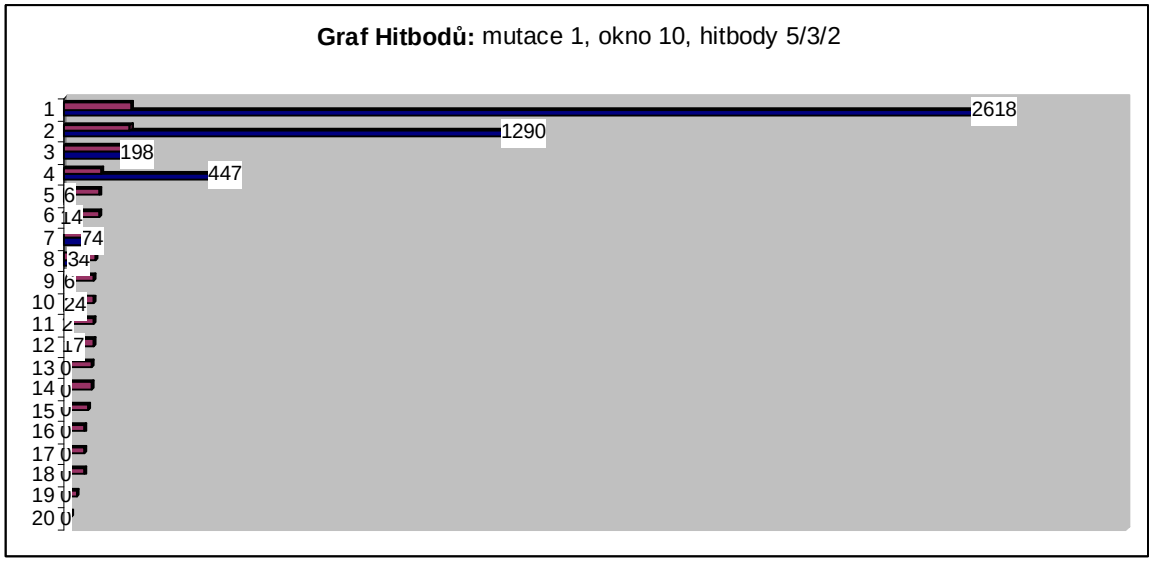
8.2 Výpočet s mutací

Pro výpočet s mutací získáváme u všech porovnání logicky větší počet hitbodů, přičemž opravdu podobné proteiny se drží na předních pozicích co do podobnosti s porovnávaným. Pro velikost okna 10 získáváme o málo více proteinů s nenulovými hitbody, ale toto je dáno opět pouze náhodnými a ještě navíc zmutovanými hity. Čtvrtý protein, který je nepoměrně delší než trojice imaginárních, má další hitbody za náhodné mutace podél celého proteinu. Nutno podotknout, že tyto nedostatky jednak opraví semiglobální zarovnání, a navíc se pravděpodobně tyto délkové rozdíly v praxi alespoň částečně smažou.

Pokud počítáme s mutací jedné aminokyseliny v hitu, výpočet se značně zpomalí. A to ještě s rostoucí délkou okna, neboť pro větší okno musíme zkoušet mutaci vícekrát, pokud není nalezena. Konkrétní naměřené hodnoty se pro jednotlivá okna měnily. Pro velikost okna 3 trval výpočet 3 minuty a 11 vteřin, pro velikost okna 5 to již bylo 3 minuty a 42 vteřin a pro desetiprvkové okno doba výpočtu překročila 4 minuty.

S větší velikostí okna získáváme přesnější výsledky, avšak platíme za to větší časovou náročností výpočtu.

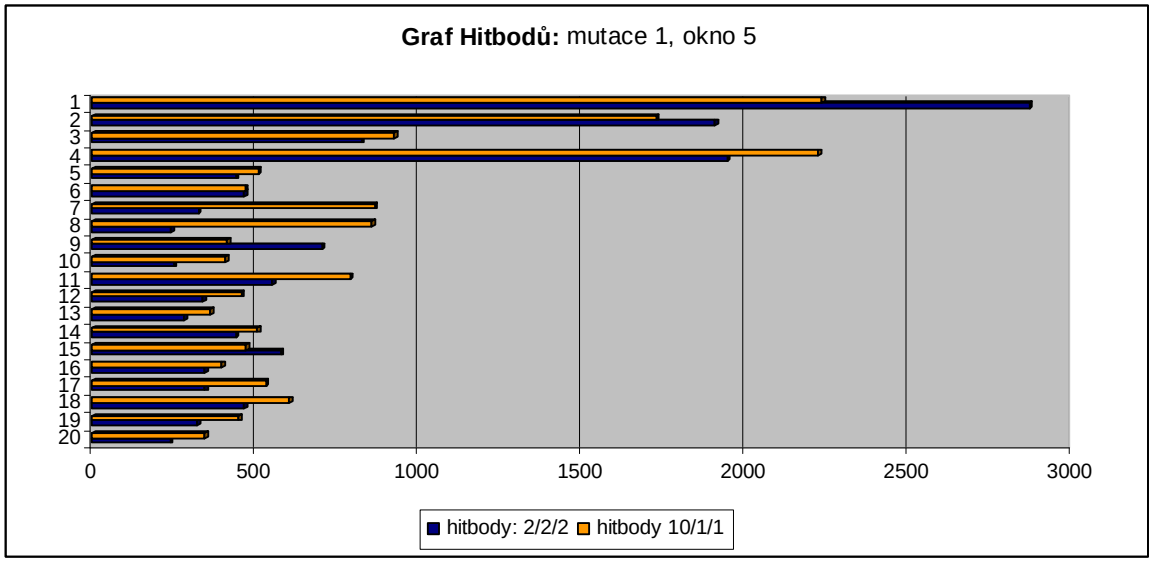




Obrázky 19 – 20: Grafy hitbodů při různých délkách porovnávacího okna s mutací

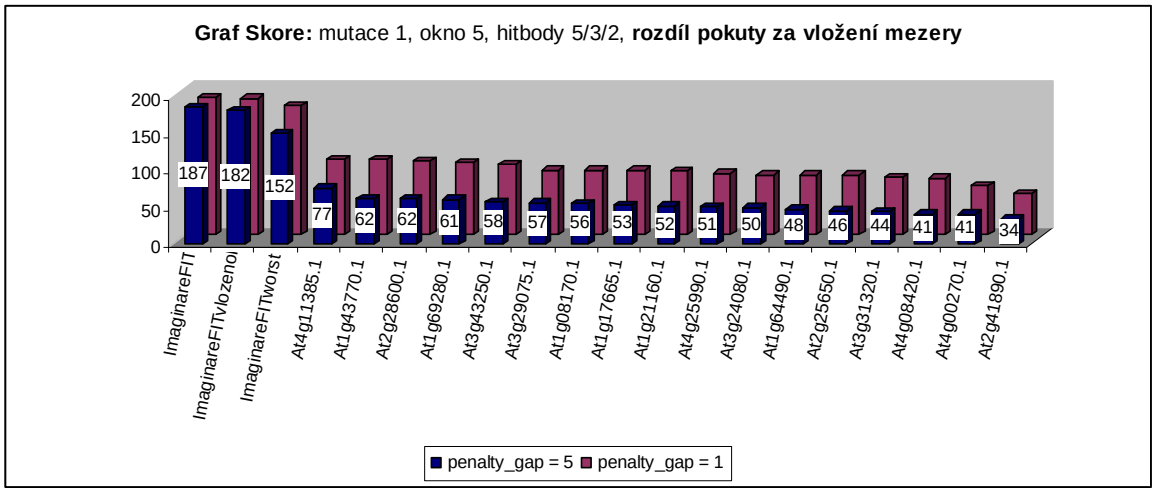
8.3 Ohodnocením hitů a pokuta za vložení mezery

Pokud smažeme rozdíly mezi jednotlivými hity, nebo nastavíme přemrštěnou hodnotu např. čistému hitu, dostáváme horší výsledky než při vyvážené hodnotě 5/3/2. Výsledky hitbodů jsou zkreslenější. Čtvrtý protein má dokonce při ohodnocení hitů 10/1/1 téměř stejnou hodnotu jako první ImaginareFIT. To je dáno tím, že jsou vícekrát porovnávány tytéž sekvence a to hraje větší roli. Čistý hit přebije prodlužování či mutaci.



Obrázek 21 – Grafy hitbodů porovnávaných proteinů při různém ohodnocení

Při zvětšení pokuty za vložení mezery dostaneme pro vzdálené proteiny menší hodnoty, pro přesný protein se hodnota nezmění vůbec, pro podobné proteiny méně. Příliš velká pokuta za vložení se ale také nedoporučuje, může to smazat veškeré nabyté skóre. Na grafu jsou vidět sloupce skóre pro top 20 proteinů s pokutou za vložení mezery 5 a 1.



Obrázek 22 – Graf skóre při různých pokutách za vložení mezery

9 Závěr

Bioinformatika v sobě spojuje biologii a informatiku. Úvod práce obsahuje shrnutí všeho důležitého v biologické části, tedy vše, co potřebuji vědět o prvcích sekvencí, které budu později porovnávat. Nukleotidy adenin, cytosin, guanin, thymin a uracil v DNA, resp. RNA, jejich určení sledu aminokyselin tvořící proteiny. Tato data jsou pak uložena v databázích. Seznámil jsem se stručně i s organizací dat v nich. Porovnávat obecně biologické sekvence pomocí počítače je při obrovském množství již zjištěných dat pro biology nutnost. I pro mě bylo z hlediska poznání složení organismu vlastního druhu velmi zajímavé se tímto problémem částečně zabývat.

Ve druhé části práce je popsán v jazyce C implementovaný program PSProt, který čerpá inspiraci ze dvou nejznámějších algoritmů pro porovnávání sekvencí FASTA a BLAST, přičemž je podobnější rozšířenějšímu BLASTu. Podobně jako tyto programy slouží PSProt k nalezení nejpodobnějších proteinů k proteinu, který může být vytvořen ze zadané sekvence DNA. V programu PSProt je použit vlastní algoritmus využívající dva selektivní algoritmy k výběru a pozdější úpravě pořadí nejpodobnějších proteinů z databáze. Jsou jimi heuristika hitbodů a varianta algoritmu Needleman-Wunsch- semiglobální zarovnání- aplikované na vybrané nejpodobnější proteiny. Jako porovnávací databáze byla použita skutečná databáze proteinů rostliny Arabidopsis thaliana (Huseníček rolní), používající se ke genetickým pokusům pro své specifické vlastnosti, jako je např. abnormálně rychlý růst a střídání počtu generací či malý genom [20].

Program pracuje ve třech fázích. Nejprve zpracuje načtenou sekvenci DNA, najde v ní gen kódující možný protein, tzn. nalezne příslušnou podsekvenci ohraničenou start kodonem a stop kodonem, přeloží ji na RNA a posléze pomocí tabulky genetického kódu na potenciální protein. Zadaná sekvence DNA může přitom obsahovat více genů, které syntetizují protein. Každý takový se nejprve na základě heuristiky hitbodů porovná se všemi proteiny z databáze, kterou má k dispozici. Algoritmus porovnává postupně všechny podsekvence („okna“), uvažuje podobnost aminokyselin každých dvou proteinů a mutaci jedné aminokyseliny v rámci jednoho nalezeného hitu (shodné podsekvence), přičemž může dojít k několika nevyhnutelným negativním jevům, jako je například křížení hitů (nalezených shod) nebo duplikaci výběru hitů při prodlužování. Tyto jevy ale pouze podporují vybranou podobnost a budou neutralizovány při pozdějším semiglobálním zarovnání nejpodobnějších proteinů. Semiglobální zarovnání reprezentuje výsledný výpočet skóre podobnosti pouze pro daný počet na základě heuristiky hitbodů vybraných nejpodobnějších proteinů. Pro každý potenciální protein, který lze syntetizovat z úseku zadané sekvence DNA, je tedy výstupem seznam nejpodobnějších proteinů z databáze, jejich zarovnání s hledaným proteinem včetně vložení mezer, jejich parametry podobnosti (hitbody, skóre). Pro výpočet skóre byla použita skórovací matice BLOSUM 62, kterou při vyhledávání nejpodobnějších proteinů či sekvencí používá také BLAST.

Za větší přesnost při výběru platíme větší časovou náročností výpočtu. Nejdůležitějšími parametry toto ovlivňující je povolení v přírodě poměrně častého jevu, mutace, při prvotním vyhledávání pomocí heuristiky hitbodů a rovněž také velikost porovnávaných podsekvencí („oken“). Mutace výpočet algoritmu výrazně zpomalí.

Výsledná práce by se mohla dále rozvíjet. Na vstup totiž zadáváme oligonukleotid (10-1000 nukleotidů dlouhá sekvence [21]) DNA kódujícími řetězec RNA, který se následně přepíše na protein. Gen v DNA se ale skládá nejen ze sekvencí kódujících pozdější protein, tzv. exonů, ale také z dalších (zpravidla delších) částí, které budou v procesu transkripce vystřiženy, tzv. intronů. Rozhraní mezi exony a introny je ale pro tuto práci především z biologického hlediska příliš detailní a proto tento sestřih PSProt neuvažuje. Mechanismus rozpoznání toho, kde začíná a kde končí v genu exon a vystřižení ostatních intronů by mohlo být tudíž jednou z nadstaveb této práce.

Literatura

[1] Wang, J. T. L., Zaki, M. J., Toivonen, H. T. T., Shasha, D.: *Data Mining in Bioinformatics*, Springer-Verlag, 2005

[2] Krane, Dan K., Raymer, Michael L.: *Fundamental Concepts of Bioinformatics*, ISBN: 0-8053-4633-3, Benjamin Cummings 2003

[3] Claverie, Jean-Michel, Notredame, Cedric: *Bioinformatics for Dummies*, ISBN: 0-7645-1696-5, Wiley Publishing, Inc., 2003

[4] Alberts, Bray, Johnson, Lewis, Raff, Roberts, Walter: *Základy buněčné biologie*, ISBN: 80-902906-0-4, Espero Publishing, 1998

[5] European Bioinformatics Institute, 2006. Dokument dostupný na URL: http://www.ebi.ac.uk/ (15.5.2008)

[6] Protein Information Resource, 2006. Dokument dostupný na URL: http://www-nbrf.georgetown.edu/ (15.5.2008)

[7] Ensembl Genome Browser, 2006. Dokument dostupný na URL: http://www.ensembl.org/ (15.5.2008)

[8] Genetický kód- Wikipedie, otevřená encyklopedie. Dokument dostupný na URL: http://cs.wikipedia.org/wiki/Genetický_kód (15.5.2008)

[9] PAČES, Jan: *Co se o sobě dovídáme z naší genetické informace*. Dokument dostupný na URL: http://www.otevrena-veda.cz/ov/users/Image/default/C1Kurzy/Biolog/7paces.pdf (15.5.2008)

[10] BioLib – efemerní druh. Dokument dostupný na URL: http://www.biolib.cz/cz/glossaryterm/dir0/id1922/ (15.5.2008)

[11] Galisson, Fredérique: *The Fasta and Blast programs*, 2000

[12] Rudolfová, Ivana: *skripta předmětu ZZN z let 2006 a 2007 na téma bioinformatika*, FIT VUT Brno

[13] Obšil, Tomáš: *Struktura proteinů a funkce enzymů*. Dokument dostupný na URL: www.otevrena-veda.cz/ov/users/Image/default/C1Kurzy/Chemie/28obsil.pdf (15.5.2008)

[14] Translation. Dokument dostupný na URL: http://www.biokurs.de/skripten/13/bs13-5.htm (15.5.2008)

[15] The BLOSUM 62 Substitution Matrix. Dokument dostupný na URL: http://www.ncbi.nlm.nih.gov/books/bv.fcgi?rid=sef.figgrp.194 (15.5.2008)

[16] NCBI Education: *Substitution Matrices*, 2000. Dokument dostupný na URL: http://www.ncbi.nlm.nih.gov/Education/BLASTinfo/Scoring2.html (15.5.2008)

[17] National Diagnostics. Dokument dostupný na URL: http://nationaldiagnostics.com/ (15.5.2008)

[18] Arabidopsis thaliana- Wikipedie, otevřená encyklopedie. Dokument dostupný na URL: http://en.wikipedia.org/wiki/Arabidopsis_thaliana (15.5.2008)

[19] Amino acid - Wikipedie, otevřená encyklopedie. Dokument dostupný na URL: http://en.wikipedia.org/wiki/Amino_acid (15.5.2008)

[20] Atlas rostlin / čeleď: Brassicaceae – rostlina Arabidopsis thaliana. Dokument dostupný na URL: http://rostliny.prirodou.cz/?rostlina=arabidopsis_thaliana (15.5.2008)

[21] Sekanina, Lukáš: *skripta předmětu BIN z let 2007 a 2008*, FIT VUT Brno

Seznam příloh

Příloha 1: Databáze pro PSProt

ImaginareFITvlozenoI|temer presny protein|KKKRRTTKQKIIEIEIEKKKKKKKKKKKKNENRNRMKIE

ImaginareFITworst|insertovany presny protein|KKKRRTTKQKIIEIEIEKKKKKKKKKAALSKKKKDENRNRMKIE

ImaginareFIT|presny protein|KKKRRTTKQKIEIEIEKKKKKKKKKKKKNENRNRMKIE

Imaginare2|druhy vymysleny protein|MIFEXKSDGKVAIITGGGKAKSIGYGIAVAYAK

At1g01010.1|NAC domain protein, putative

|MEDQVGFGFRPNDEELVGHYLRNKIEGNTSRDVEVAISEVNICSYDPWNLRFQSKYKSRDAMWYFFSRRENNKGNRQSRTTVSG
KWKLTGESVEVKDQWGFCSEGFRGKIGHKRVLVFLDGRYPDKTKSDWVIHEFHYDLLPEHQRTYVICRLEYKGDDADILSAYAI
DPTPAFVPNMTSSAGSVVNQSRQRNSGSYNTYSEYDSANHGQQFNENSNIMQQQPLQGSFNPLLEYDFANHGGQWLSDYIDLQQ
QVPYLAPYENESEMIWKHVIEENFEFLVDERTSMQQHYSDHRPKKPVSGVLPDDSSDTETGSMIFEDTSSSTDSVGSSDEPGHTRID
DIPSLNIIEPLHNYKAQEQPKQQSKEKVISSQKSECEWKMAEDSIKIPPSTNTVKQSWIVLENAQWNYLKNMIIGVLLFISVISWIILV
G

At1g01020.1|unknown protein

|MAASEHRCVGCGFRVKSLFIQYSPGNIRLMKCGNCKEVADEYIECERMIIFIDLILHRPKVYRHVLYNAINPATVNIQHLLWKLVF
AYLLLDCYRSLLLRKSDEESSFSDSPVLLSIKVRSFLFNGLN

At1g01030.1|DNA-binding protein, putative

|MDLSLAPTTTTSSDQEQDRDQELTSNIGASSSSGPSGNNNNLPMMMIPPPEKEHMFDKVVTPSDVGKLNRLVIPKQHAERYFPLDS
SNNQNGTLLNFQDRNGKMWRFRYSYWNSSQSYVMTKGWSRFVKEKKLDAGDIVSFQRGIGDESERSKLYIDWRHRPDMSLVQA
HQFGNFGFNFNFPTTSQYSNRFHPLPEYNSVPIHRGLNIGNHQRSYYNTQRQEFVGYGYGNLAGRCYYTGSPLDHRNIVGSEPLVI
DSVPVVPGRLTPVMLPPLPPPPSTAGKRLRLFGVNMECGNDYNQQEESWLVPRGEIGASSSSSSALRLNLSTDHDDDNDDGDDGD
DDQFAKKGKSSLSLNFNP

At1g01040.1|DEAD|MVMEDEPREATIKPSYWLDACEDISCDLIDDLVSEFDPSSVAVNESTDENGVINDFFGGIDHILDSIKNGGGLP
NNGVSDTNSQINEVTVTPQVIAKETVKENGLQKNGGKRDEFSKEEGDKDRKRARVCSYQSERSNLSGRGHVNNSREGDRFMNRK
RTRNWDEAGNNKKKRECNNYRRDGRDREVRGYWERDKVGSNELVYRSGTWEADHERDVKKVSGGNRECDVKAEENKSKPEE
RKEKVVEEQARRYQLDVLEQAKAKNTIAFLETGAGKTLIAILLIKSVHKDLMSQNRKMLSVFLVPKVPLVYQQAEVIRNQTCFQV
GHYCGEMGQDFWDSRRWQREFESKQVLVMTAQILLNILRHSIIRMETIDLLILDECHHAVKKHPYSLVMSEFYHTTPKDKRPAIFG
MTASPVNLKGVSSQVDCAIKIRNLETKLDSTVCTIKDRKELEKHVPMPSEIVVEYDKAATMWSLHETIKQMIAAVEEAAQASSRK
SKWQFMGARDAGAKDELRQVYGVSERTESDGAANLIHKLRAINYTLAELGQWCAYKVGQSFLSALQSDERVNFQVDVKFQESY
LSEVVSLLQCELLEGAAAEKVAAEVGKPENGNAHDEMEEGELPDDPVVSGGEHVDEVIGAAVADGKVTPKVQSLIKLLLKYQHT
ADFRAIVFVERVVAALVLPKVFAELPSLSFIRCASMIGHNNSQEMKSSQMQDTISKFRDGHVTLLVATSVAEEGLDIRQCNVVMRF
DLAKTVLAYIQSRGRARKPGSDYILMVERGNVSHAAFLRNARNSEETLRKEAIERTDLSHLKDTSRLISIDAVPGTVYKVEATGAM
VSLNSAVGLVHFYCSQLPGDRYAILRPEFSMEKHEKPGGHTEYSCRLQLPCNAPFEILEGPVCSSMRLAQQAVCLAACKKLHEMG
AFTDMLLPDKGSGQDAEKADQDDEGEPVPGTARHREFYPEGVADVLKGEWVSSGKEVCESSKLFHLYMYNVRCVDFGSSKDPF
LSEVSEFAILFGNELDAEVLSMSMDLYVARAMITKASLAFKGSLDITENQLSSLKKFHVRLMSIVLDVDVEPSTTPWDPAKAYLFV
PVTDNTSMEPIKGINWELVEKITKTTAWDNPLQRARPDVYLGTNERTLGGDRREYGFGKLRHNIVFGQKSHPTYGIRGAVASFDV
VRASGLLPVRDAFEKEVEEDLSKGKLMMADGCMVAEDLIGKIVTAAHSGKRFYVDSICYDMSAETSFPRKEGYLGPLEYNTYAD
YYKQKYGVDLNCKQQPLIKGRGVSYCKNLLSPRFEQSGESETVLDKTYYVFLPPELCVVHPLSGSLIRGAQRLPSIMRRVESMLLA
VQLKNLISYPIPTSKILEALTAASCQETFCYERAELLGDAYLKWVVSRFLFLKYPQKHEGQLTRMRQQMVSNMVLYQFALVKGLQ
SYIQADRFAPSRWSAPGVPPVFDEDTKDGGSSFFDEEQKPVSEENSDVFEDGEMEDGELEGDLSSYRVLSSKTLADVVEALIGVYY
VEGGKIAANHLMKWIGIHVEDDPDEVDGTLKNVNVPESVLKSIDFVGLERALKYEFKEKGLLVEAITHASRPSSGVSCYQRLEFVG
DAVLDHLITRHLFFTYTSLPPGRLTDLRAAAVNNENFARVAVKHKLHLYLRHGSSALEKQIREFVKEVQTESSKPGFNSFGLGDCK
APKVLGDIVESIAGAIFLDSGKDTTAAWKVFQPLLQPMVTPETLPMHPVRELQERCQQQAEGLEYKASRSGNTATVEVFIDGVQV
GVAQNPQKKMAQKLAARNALAALKEKEIAESKEKHINNGNAGEDQGENENGNKKNGHQPFTRQTLNDICLRKNWPMPSYRCV
KEGGPAHAKRFTFGVRVNTSDRGWTDECIGEPMPSVKKAKDSAAVLLLELLNKTFS

Příloha 2: výstupní soubor results.txt

```
VYSLEDKY POROVNAVANI:

* Datum vypoctu: 11.05.2008, 23:02:24
*
* Sekvence:
TTTAAATTTTTACTTTTTTTTTTCTTCTTGTTGGTTCGTTTTTTAGCTTTAGCTTTAGCTTTTTTTTTTTTTTTTTTTTTTTTT
TTTTTTTTTTTTTTTTACTTTTAGCTTTAGCTTACTTTTAGCTTACTTGAAGCTTACTTGACTTTAGCTTTTCTTTTTTTTTT
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTGTACTTA
*
* databaze v souboru: databaze.txt
* Pocet proteinu v databazi: 20000
*
* Parametry pro porovnavani:
*   velikost okna pro porovnavani: 5
*   minimalni delka proteinu pro porovnavani:  30 nukleotidu, tzn. 10 aminokyselin
*   mutace povolena
*   ohodnoceni hitu: cisty hit: 5, prodlouzeni hitu: 3, mutovany hit: 2
*   pokuta za vlozeni mezery: 5
*
* Pocet nejlepsich sekvenci pro zarovnani: 3
*
* ------------------------------------------------------------------------
*
* Proteiny nalezene ve vstupni sekvenci
*   1, KKKRRTTKQKIEIEIEKKKKKKKKKKKKNENRNRMKIE
*    -- (sekvence nukleotidu:
AAAAAAAAAAGAAGAACAACCAAGCAAAAAAUCGAAAUCGAAAUCGAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAUGAAAAUCGAAAUCGAAUGAAAAUCGAA)
*
*    -- Nejpodobnejsi proteiny
*        -- 1, ImaginareFIT, presny protein,  4485 hitbodu, skore: 187
*              Detail: soubor results/Porovnani_1_1.txt
*        -- 2, ImaginareFITvlozenoI, temer presny protein,  3032 hitbodu, skore:
182
*              Detail: soubor results/Porovnani_1_2.txt
*        -- 3, ImaginareFITworst, insertovany presny protein,  1340 hitbodu, skore:
152
*              Detail: soubor results/Porovnani_1_3.txt
```

Příloha 3: výstupní soubor porovnavani_1_3.txt

```
* VYSLEDKY POROVNAVANI - DETAILY
  ==============================

* Sekvence:
TTTAAATTTTTACTTTTTCGTTTTTTAGCTTTAGCTTTAGCTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTATTTTTTTTACTTTTAGCTT
TAGCTAACTTTTAGCTTACTTGAAGCTTACTTGACTTTAGCTTTTCTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTGTACTTA
*
* Potencionalni protein: KKQKIEIEIEKKKKKKKKKK
*    (sekvence nukleotidu: AAAAAGCAAAAAAAUCGAAAUCGAAAUCGAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA)
*
* Porovnani s proteinem z databaze:
*    ImaginareFIT, presny protein
*    QKRRIEIEKKKKKKKKKRRKKK
*    566 hitbodu
*    skore: 75
*
* Idealni zarovnani:
*    KKQKIEIEIEKKKKKKKK__KK_
*    QKRRIEIE_KKKKKKKKKRRKKK
*


MATICE skore
============

          Q   K   R   R   I   E   I   E   K   K   K   K   K   K   K   K   K   R   R   K   K   K
      0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  K   0   1   5   4   3   2   1   0   1   5   5   5   5   5   5   5   5   5   4   3   5   5   5
  K   0   1   6   7   6   5   4   3   2   6  10  10  10  10  10  10  10  10   9   8   8  10  10
  Q   0   5   5   7   8   7   7   6   5   5   9  11  11  11  11  11  11  11  11  10   9   9  11
  K   0   4  10   9   9   8   8   7   7  10  10  14  16  16  16  16  16  16  15  14  15  14  14
  I   0   3   9   8   8  13  12  12  11  10   9  13  15  15  15  15  15  15  14  13  14  13  14
  E   0   2   8   9   8  12  18  17  17  16  15  14  14  16  16  16  16  16  15  14  14  15  14
  I   0   1   7   8   7  12  17  22  21  20  19  18  17  16  15  15  15  15  14  13  13  14  14
  E   0   2   6   7   8  11  17  21  27  26  25  24  23  22  21  20  19  18  17  16  15  14  15
  I   0   1   5   6   7  12  16  21  26  25  24  23  22  21  20  19  18  17  16  15  14  13  15
  E   0   2   4   5   6  11  17  20  26  27  26  25  24  23  22  21  20  19  18  17  16  15  15
  K   0   1   7   6   7  10  16  19  25  31  32  31  30  29  28  27  26  25  24  23  22  21  20
  K   0   1   6   9   8   9  15  18  24  30  36  37  36  35  34  33  32  31  30  29  28  27  26
  K   0   1   6   8  11  10  14  17  23  29  35  41  42  41  40  39  38  37  36  35  34  33  32
  K   0   1   6   8  10   9  13  16  22  28  34  40  46  47  46  45  44  43  42  41  40  39  38
  K   0   1   6   8  10   9  12  15  21  27  33  39  45  51  52  51  50  49  48  47  46  45  44
  K   0   1   6   8  10   9  11  14  20  26  32  38  44  50  56  57  56  55  54  53  52  51  50
  K   0   1   6   8  10   9  10  13  19  25  31  37  43  49  55  61  62  61  60  59  58  57  56
  K   0   1   6   8  10   9  10  12  18  24  30  36  42  48  54  60  66  67  66  65  64  63  62
  K   0   1   6   8  10   9  10  11  17  23  29  35  41  47  53  59  65  71  70  69  70  69  68
  K   0   1   6   8  10  10  10  10  16  22  28  34  40  46  52  58  64  70  73  73  74  75  75
```

pomMATICE
=========

```
                 Q       K       R       R       I       E       I       E       K  ...
  [  0, -1][  0,  0][  0,  1][  0,  2][  0,  3][  0,  4][  0,  5][  0,  6][  0,  7][  0,  8] ...
 K[  0,  0][  0,  0][  0,  1][  1,  2][  1,  3][  1,  4][  0,  5][  1,  6][  0,  7][  0,  8] ...
 K[  1,  0][  1,  0][  1,  1][  1,  2][  1,  3][  2,  4][  2,  5][  2,  6][  2,  7][  1,  8] ...
 Q[  2,  0][  2,  0][  2,  2][  2,  2][  2,  3][  3,  4][  2,  5][  3,  6][  2,  7][  2,  9] ...
 K[  3,  0][  3,  1][  3,  1][  4,  2][  3,  3][  4,  4][  3,  5][  4,  6][  3,  7][  3,  8] ...
 I[  4,  0][  4,  1][  4,  2][  5,  2][  4,  4][  4,  4][  5,  5][  4,  6][  5,  7][  5,  8] ...
 E[  5,  0][  5,  0][  5,  2][  5,  2][  5,  3][  5,  5][  5,  5][  6,  6][  5,  7][  6,  8] ...
 I[  6,  0][  6,  1][  6,  2][  6,  3][  7,  3][  6,  4][  6,  6][  6,  6][  7,  7][  7,  8] ...
 E[  7,  0][  7,  0][  7,  2][  7,  2][  7,  3][  7,  5][  7,  5][  7,  7][  7,  7][  8,  8] ...
 I[  8,  0][  8,  1][  8,  2][  8,  3][  8,  4][  8,  4][  8,  6][  8,  6][  8,  8][  9,  8] ...
 E[  9,  0][  9,  0][  9,  2][  9,  2][  9,  3][  9,  5][  9,  5][  9,  7][  9,  7][  9,  8] ...
 K[ 10,  0][ 10,  0][ 10,  1][ 10,  2][ 10,  3][ 10,  5][ 10,  6][ 10,  7][ 10,  8][ 10,  8] ...
 K[ 11,  0][ 11,  0][ 11,  1][ 11,  2][ 11,  3][ 11,  5][ 11,  6][ 11,  7][ 11,  8][ 11,  8] ...
 K[ 12,  0][ 12,  0][ 12,  1][ 12,  2][ 12,  3][ 13,  4][ 12,  6][ 12,  7][ 12,  8][ 12,  8] ...
 K[ 13,  0][ 13,  0][ 13,  1][ 13,  2][ 13,  3][ 14,  4][ 13,  6][ 13,  7][ 13,  8][ 13,  8] ...
 K[ 14,  0][ 14,  0][ 14,  1][ 14,  2][ 14,  3][ 15,  4][ 14,  6][ 14,  7][ 14,  8][ 14,  8] ...
 K[ 15,  0][ 15,  0][ 15,  1][ 15,  2][ 15,  3][ 16,  4][ 15,  6][ 15,  7][ 15,  8][ 15,  8] ...
 K[ 16,  0][ 16,  0][ 16,  1][ 16,  2][ 16,  3][ 17,  4][ 16,  5][ 16,  7][ 16,  8][ 16,  8] ...
 K[ 17,  0][ 17,  0][ 17,  1][ 17,  2][ 17,  3][ 18,  4][ 17,  5][ 17,  7][ 17,  8][ 17,  8] ...
 K[ 18,  0][ 18,  0][ 18,  1][ 18,  2][ 18,  3][ 19,  4][ 18,  5][ 18,  7][ 18,  8][ 18,  8] ...
 K[ 19,  0][ 19,  0][ 19,  1][ 19,  2][ 19,  3][ 20,  4][ 19,  5][ 20,  6][ 19,  8][ 19,  8] ...
```

MATICE CESTY
============

```
      Q K R R I E I E K K K K K K K K R R K K K
    1
  K   1
  K     1
  Q       1
  K         1
  I           1
  E             1
  I               1
  E                 1
  I                 1
  E                   1
  K                     1
  K                       1
  K                         1
  K                           1
  K                             1
  K                               1
  K                                 1
  K                                   1 1 1
  K                                         1
  K                                           1 1
```

Příloha 4: část výstupního souboru log.txt

```
 Chybny protein 0 - pokusny2, POZICE 34: znak: B (ascii kod 66)
Pocet nactenych proteinu z databaze: 20000

Nalezen start kodon na pozici 10
Nalezen stop kodon na pozici 73

sekvence> ALOKACE NOVE PAMETI - 10

Nalezen start kodon na pozici 81
Nalezen stop kodon na pozici 99
Nalezen start kodon na pozici 109
Nalezen stop kodon na pozici 121
Nalezen start kodon na pozici 120
Nalezen stop kodon na pozici 126
Nalezen start kodon na pozici 179
Pocet vzniklych potencialnich proteinu ze vstupni sekvence: 1


1.SEKVENCE ##########

 1.DB PROTEIN ##########
   Mutace okna: KIEIE|IIEIE
  HIT KIEIE|IIEIE mutace=1
  +prodlouzeni hitu |II 1|
  +prodlouzeni hitu |EE 1|
  +prodlouzeni hitu |KK 1|
  +prodlouzeni hitu |KK 1|
  +prodlouzeni hitu |KK 1|
  +prodlouzeni hitu |KK 1|
  +prodlouzeni hitu |KK 1|
  +prodlouzeni hitu |KK 1|
  HIT IEIEI|IEIEI mutace=0
   probehla mutace K -> I! 1
  -prodlouzeni hitu |II 1|
  +prodlouzeni hitu |EE 1|
  +prodlouzeni hitu |KK 1|
...

Zarovnavani [1_1]
__\____\\\_\\\\\\\\\\\\\\\\_____________

Zarovnavani [1_2]
__\____\\\_\\\\\\\\\\\\\\\\____\_____________

Zarovnavani [1_3]
\\\\\\\\|\\\\\\\\\__\\_
```

Příloha 5: algoritmus mutace okna

```
...
for (i = 0; i < pocet_seq; i++) {
       // pro kazdy protein v databazi
       for (j = 0; j < pocet_proteinu_db; j++) {
         // pro celou sekvenci potencialniho proteinu
         k = 0;
         while ((*proteiny)[i][k+OKNO-1] != '\0') {
             strncpy(okno, (*proteiny)[i]+k, OKNO);
             okno[OKNO] = '\0'; // okno potencialniho proteinu
             // porovnava se se stejne velkymi okny vsech proteinu v databazi
             pozice_prot = 0;
             while ((*RETEZ)[j][pozice_prot+OKNO] != '\0') {
               strncpy(okno_db, (*RETEZ)[j]+pozice_prot, OKNO);
               okno_db[OKNO] = '\0';
    // ***** MUTACE POVOLENA
    if (povoleni_mutace) {
                  KONEC = 0;
                  MUTOVANO = 0;
                  while (! KONEC) {
                        if ((strcmp (okno_db, okno) == 0) || (MUTOVANO == 1)) {
                           // inkrementuj hit! (cisty hit)
                           if (! MUTOVANO) {
                              hits[i*pocet_proteinu_db+j] += CISTY_HIT;
                              } else { hits[i*pocet_proteinu_db+j] += MUTOVANY_HIT; }

                           // ######## POKUS O PRODLOUZENI HITU NA OBE STRANY
                           // NALEVO
                               kolik = 1;
                               prvek_KONEC = 0;
                               probehla_mutace = 0;
                               c = (*proteiny)[i][k-kolik];

                               while ((! prvek_KONEC) && (pozice_prot-kolik >= 0) && (k-
kolik >= 0)) {
                                     if ((*RETEZ)[j][pozice_prot-kolik] == c) {
                                              hits[i*pocet_proteinu_db+j] += PRODL_HIT;
                                              kolik++;
                                              // zmena c pro delsi prodlouzeni
                                              c = (*proteiny)[i][k-kolik];
                                              probehla_mutace = 0;
                                              }
                                     else if (! MUTOVANO) {
                                              // pokus o mutaci
                                              if (probehla_mutace) {
                                                 prvek_KONEC = 1;
                                                 }
                                                  else {
                                                       // mutace
                                                       skupina = zjisti_skupinu(c);
```

```
                                                x = 0;
                                                while (skupiny[skupina][x] != '\0') {
                                                    if ( (*RETEZ)[j][pozice_prot-
kolik] == skupiny[skupina][x] ) {

                                                        // uspesne probehla mutace
                                                        MUTOVANO = 1;
                                                        // nove c
                                                        c = skupiny[skupina][x];
                                                    }
                                                    x++;
                                                }

                                                // uspesne probehla mutace
                                                probehla_mutace = 1;
                                                }
                                        } else { prvek_KONEC = 1; }
                                }

                                // NAPRAVO
                                     ... obdobně, jen pozice se posunuje o +OKNO+kolik
                                kolik = 0;
                                // ######## KONEC PRODLUZOVANI
                            // konci pro toto OKNO v databazi, pokracuj dalsim
                            KONEC = 1;

                            }
                            // nebyl nalezen cisty hit, zkus zmutovat
                            else {

                                 // pokud nebude nalezena mutace, pro okno proteinu KONEC
                                 KONEC = 1;
                                 // pro vsechny prvky v OKNE
                                 for (y = 0; y < OKNO; y++) {
                                         // nalezeni skupiny
                                         skupina = zjisti_skupinu(okno[y]);
                                         // pro celou skupinu zkousim, zda najdu
                                         x = 0;
                                         while (skupiny[skupina][x] != '\0') {
                                             // pokud je prvek ze stejne skupiny, MUTACE
                                             if (okno_db[y] == skupiny[skupina][x]) {
                                                // do pom.promenne nakopiruj okno, mutuj
                                                strcpy(pom_okno, okno);
                                                pom_okno[y] = skupiny[skupina][x];

                                                // nove porovnani
                                                if (strcmp (okno_db, pom_okno) == 0) {
                                                   MUTOVANO = 1;
                                                   KONEC = 0;
                                                   }

                                             }
                                         x++;
                                         }
```

```
                                }
                            }
                        }

 } else { // ***** MUTACE NEPOVOLENA
        ...
         }
            // KONEC. Cyklus probiha pro dalsi podokno proteinu v db
            pozice_prot++;
            }
          // delej vse pro dalsi okno v potencialnim proteinu
          k++;
      }
    }

}
```