Návrh a prototypová implementace databáze pro snadnější práci se strukturami nukleových kyselin

Bc. Ondřej Čečák

Fakulta elektrotechnická
České vysoké učení technické v Praze

10. června 2011

Obsah

Stručné zadání práce

Zadané části práce:

1. návrh a implementace datových struktur potřebných pro databázi struktur nukleových kyselin
2. importy dat z veřejné databáze PDB obohacených analýzou programem 3DNA
3. návrh a prototypová implementace webové aplikace pro příjemné dotazování nad daty
4. zobrazení výsledků s použitím grafické aplikace JMol

Motivace – cíl a účel

- aktuální téma, studium struktury pro pochopení funkce a interakce biomakromolekul v organizmech
- přímo nezpracovaná oblast, v současné době neexistuje podobný nástroj pro rychlé zpracování dotazů nad požadovanými daty
- cíl je zpracovat analyzovaná data struktur pro snadné použití – vyhledávání
- aplikace má sloužit zejména výzkumníkům a studentům VŠCHT v Praze

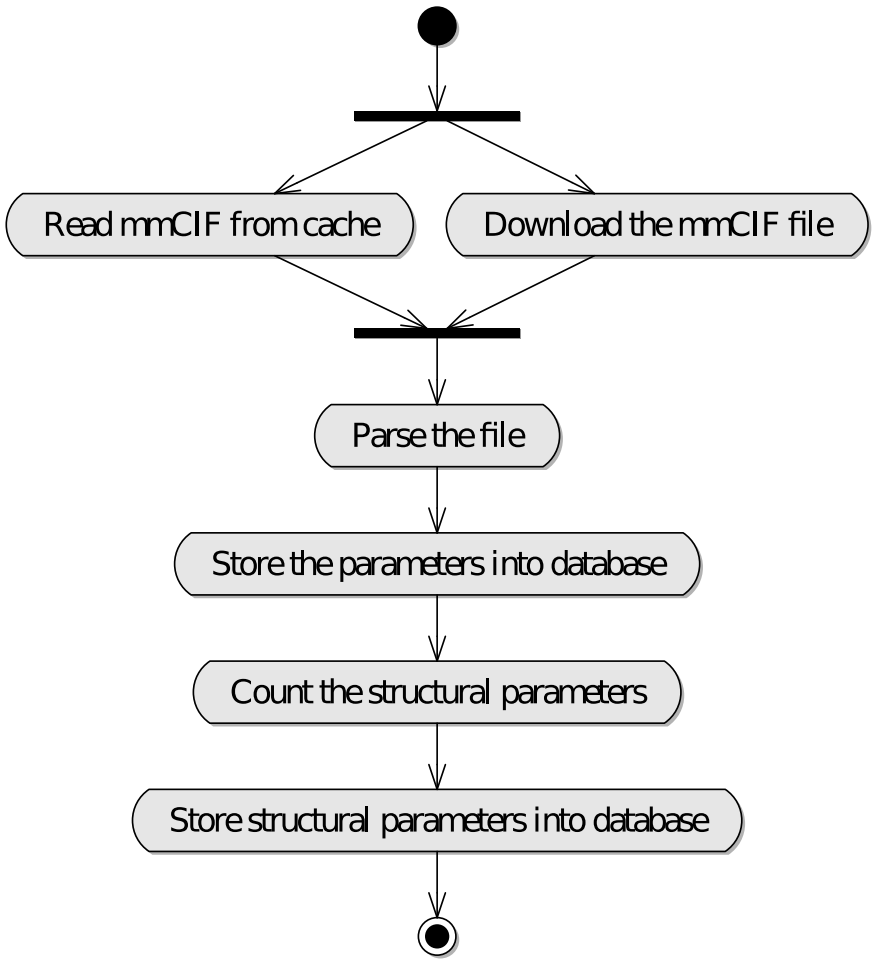
Návrh

Návrh a následně pracovní postup:

1. nastudování problematiky, rešerše existujících řešení
2. import dat z veřejné databáze PDB
3. analýza struktur programem 3DNA
4. databáze struktur a souvisejících parametrů v PostgreSQL
5. funkční prototyp aplikace pro dotazování
6. vizualizace v grafickém appletu JMol
7. testování importu

Import dat

- jazyk Python
- exaktní analýza mmCIF dat externí knihovnou mmLib
- výpočet strukturních dat externím programem 3DNA 2.0
- uložení získaných dat do databáze

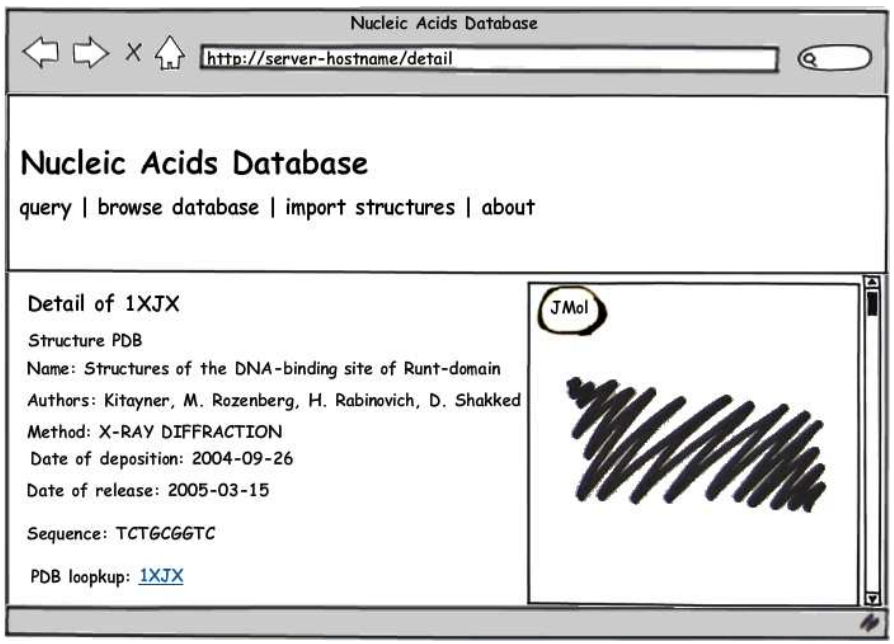


PostgreSQL databáze

- „mělká“ struktura – vazba přes identifikátor biomolekuly, data organizovaná v tabulkách podle typu dat
- B-tree indexy, vyhledávání se složitostí $O(\log n)$
- uživatelské datové typy po úvodní rešerši zamítnuty
- PostgreSQL sekvence a transakce – podpora pro paralelní zpracování dat
- nenašel jsem podobnou práci, nejblíže je návrh zpracování PDB dat „PDB-SQL“ pro ukládání a vyhledávání alpha uhlíkových pozic do MySQL

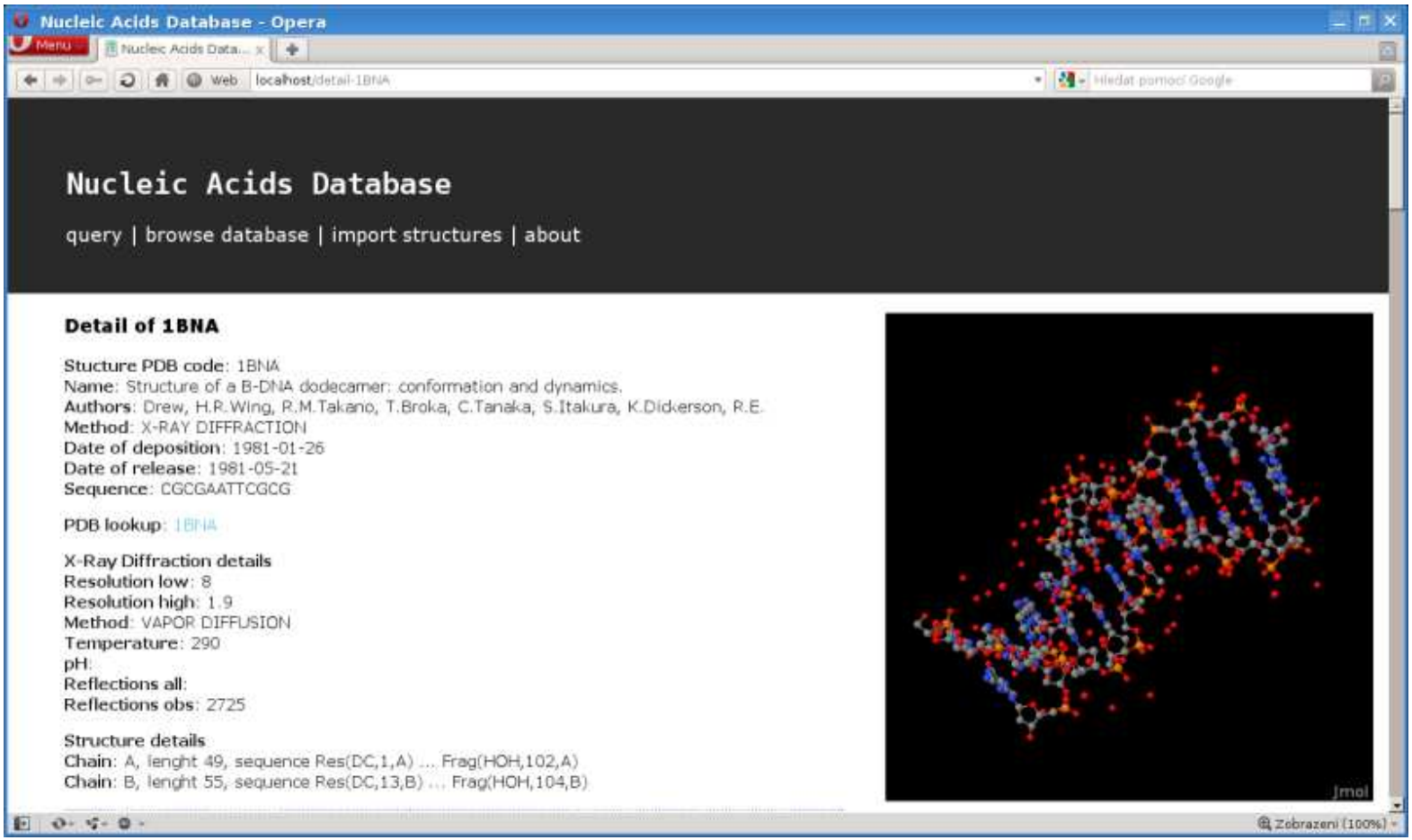
Funkční prototyp webové aplikace

- jazyk PHP
- Java Applet JMol
- jazyk JavaScript
- uložení získaných dat do databáze



Realizace

- komponenty implementovány dle návrhu



Testování

- import všech struktur obsahující RNA nebo DNA
- 5316 souborů mmCIF, cca 7 GB organizovaného textu, nejstarší z roku 1978
- 3 pracovní procesy na CPU Intel Quad 9400 2,66 GHz cca 13 hodin čistého procesorového času
- kontrolní skript v BASHi sledující výstup importního skriptu
- 95,96 % dat importováno, zbytek označen jako chyba analýzy mmView nebo 3DNA

Testování komponent aplikace

- 5.101 struktur, 4.303.860 základních jednotek, 36.071.202 atomů
- 12.600.746 strukturních parametrů
- cca 1,9 GB čistých textových dat v SQL dumpu
- dotazy na strukturu průměrně do 6 sekund
- výsledek zobrazen se zvýrazněnými detaily včetně prezentace výsledné části struktury v JMol
- testování použitelnosti na poučených uživatelích PC v pořádku

Zhodnocení

- zadání splněno, určené části databáze pro snadnější práci se strukturami nukleových kysel jsem plně vypracoval pod licencí GNU/GPL a otestoval importem všech známých struktur s DNA a RNA
- vlastní diplomová dokumentuje v angličtině úvodní rešerši, návrh a detaily implementace
- závěr – komplexní data jako strukturální parametry makrobiomolekul mohou být pomocí relační databáze vyhledávány velmi rychle
- cíl splněn, aplikace pomůže pracovníkům a studentům VŠCHT v Praze při výzkumu a studiu
- další možné pokračování – vylepšení webové aplikace, knihovny mmLib nebo programu pro analýzu 3DNA

Diskuze: Oponentský posudek I.

Otázka 1

Čím je způsobeno, že u struktur např. 3kuy, 2vju, 1kx3 a dalších se zobrazují neúplné seznamy nukleotidů a případně žádné jejich další parametry, přestože program 3DNA generuje čisté a bezchybné výstupní soubory?

Všechny tyto struktury (a několik dalších) mají záporné indexy jednotlivých residuí, se kterým se nástroj pro import nevyrovná správně kvůli příliš širokému regulárnímu výrazu (znaménko u celého čísla tak bylo oříznuto).

Diskuze: Oponentský posudek I. – demonstrace

Původní výraz:

```
conversionArray... =
 int(re.sub(r'?\D', "", numberBuilder...
```

Opravená verze:

```
conversionArray... =
 int(re.sub(r'[^-0-9]*', "", numberBuilder...
```

Diskuze: Oponentský posudek II.

Otázka 2

Zajímaly by mě bližší důvody, čím je způsobena značná prodleva při dotazu „browse database“ z webového rozhraní.

Stránka „browse database“ provádí výpočet statistik databáze, 14× zavolá dotaz `SELECT COUNT(*)` pro všechny relevantní tabulky a vypočítá výsledek.
V našem případě databázový stroj zavolá logický ekvivalent scan převedený na fyzické operace exekutoru, Seq Scan. Operace Seq Scan nemá žádnou podřízenou operaci – data získává voláním storage manageru, který má na starosti plnění a správu obsahu datové cache. Když není konkrétní databáze nějakou dobu využívána, cache na výsledek Seq Scan se vyprázdní a data se tedy musí znovu získat fyzickým procházením dat na disku, které je časově náročné.

Diskuze: Oponentský posudek II. – demonstrace

Prováděcí plán dotazu typu `SELECT COUNT(*)` s daty v cache:

```
EXPLAIN ANALYZE SELECT COUNT(*) FROM structureParameter;
QUERY PLAN
-----------------------------------------------------------


Aggregate (cost=6646.39..6646.40 rows=1 width=0)
 (actual time=46.124..46.124 rows=1 loops=1)
 -> Seq Scan on structureparameter
  (cost=0.00..5919.51 rows=290751 width=0)
  (actual time=0.007..26.099 rows=235766 loops=1)
Total runtime: 46.157 ms
```

Pokud by se v reálném použití ukázalo, že tento typ dotazu zpomaluje práci, bude nejlepší řešení data vhodně předpočítat pro snadné získání, například pomocí databázového TRIGGERu.

Diskuze: Oponentský posudek III.

Otázka 3

Jak je databáze zabezpečena z hlediska využití kombinace HTML formulářových polí, skriptovacího jazyka PHP a přístupu do databáze? Nehrozí neoprávněný přístup a spuštění SQL příkazů typu `DROP DATABASE name;` skrze formulářová pole webového rozhraní?

- nasazení není předpokládáno v „nepřátelském“ prostředí (běžný uživatel může importem dat by-design smazat celou databázi), bezpečnost je zajištěna na úrovni práv uživatele databáze
- HTML vstupy jsou přesto ošetřeny před útoky typu SQL injection na úrovní PHP kódu

Diskuze: Oponentský posudek III. – demonstrace

PHP kód:

```
$where = "";
$where .= $_POST["detailSelect"] . " ";
$where .= "='" . $_POST["detail"] . "'";
$query = "SELECT biomoleculeId FROM biomoleculeData
  WHERE $where;";
$result = pg_exec($link, $query);
```

Výsledný SQL dotaz s ošetřeným útokem typu SQL injection:

```
SELECT biomoleculeId
  FROM biomoleculeData
  WHERE name ='\'; DROP TABLE biomolecule; --';
```

Diskuze: Oponentský posudek IV.

Připomínka

Za nevhodný považuji název projektu „Nucleic Acids Database“, protože téměř koliduje s názvem celosvětově uznávaného zdroje strukturních dat nukleových kyselin „Nucleic Acid Database“ (`http://ndbserver.rutgers.edu/`).

Název aplikace byl myšlen zkratkou NADB. S panem oponentem souhlasím, po obhajobě práce bude projekt přejmenován na NASQYT – „Nucleic Acids Structure QuerY Tool“. Děkuji za připomínku.