VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Algoritmy vyhledávání v textu

Pattern Matching Algorithms

2017 Dušan Choleva

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Dušan Choleva**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Algoritmy vyhledávání v textu
Pattern Matching Algorithms

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je implementovat knihovnu pro vyhledávání v textech. Algoritmy implementované v této práci budou založeny na konečných automatech a budou umožňovat nejen přesné vyhledávání, ale i vyhledávání s chybou (Hammingova či Levenshteinova vzdálenost). Práce bude obsahovat následující:
1. Popis současného stavu poznání, rozbor algoritmů a jim odpovídajících konečných automatů.
2. Možnosti implementace konečných automatů.
3. Implementace knihovny.
4. Experimenty s implementovanými algoritmy, výsledky a rozbor experimentů.

Seznam doporučené odborné literatury:

[1] Holub, J.: Bit Parallelism - NFA Simulation. Proceedings of the 6th Conference on Implementation and Application of Automata, University of Pretoria, Pretoria, South Africa, July 2001, LNCS 2494, Springer-Verlag, pp. 149-160, 2002.
[2] Holub, J.: Simulation of Nondeterministic Finite Automata in Pattern Matching. Ph.D. thesis, Czech Technical University in Prague, Prague, p. 118, February 2000.
[3] Holub, J.: Reduced Nondeterministic Finite Automata for Approximate String Matching. Proceedings of the Prague Stringology Club Workshop '96, Czech Technical University in Prague, Prague, pp. 19-27, August 1996.
[4] Holub, J.: Simulation of NFA in Approximate String and Sequence Matching. Proceedings of the Prague Stringology Club Workshop '97, Czech Technical University in Prague, Prague, pp. 39-46, July 1997.

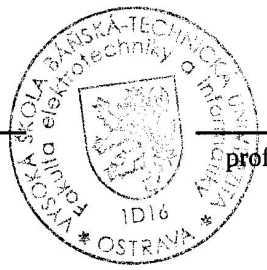
Dále podle pokynů vedoucího.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Mgr. Jiří Dvorský, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 20. dubna 2017 .....

Chtěl bych velmi poděkovat vedoucímu mé bakalářské práce, panu doc. Mgr. Jiřímu Dvorskému, Ph.D. za poskytnutí mnoha užitečných rad, trpělivost a čas strávený našimi konzultacemi.

Abstrakt

Cílem této bakalářské práce je implementace knihovny pro vyhledávání v textech. Knihovna bude umožňovat vyhledávání uživatelem určeného vzoru s určitým počtem chyb v textu založené na deterministických a nedeterministických konečných automatech. Pro přibližné porovnávání vzorů bude uživateli umožněn výběr mezi Hammingovou a Levenshteinovou vzdáleností. V první části se práce zabývá rozborem teorie týkající se využití konečných automatů pro vyhledávání v textu pomocí přibližného porovnávání vzorů. Druhá část se zabývá implementací knihoven. Třetí část se zabývá experimenty s naimplementovanými knihovnami. Závěr shrnuje výhody a nevýhody tohoto přístupu k vyhledávání v textech.

Klíčová slova: deterministický konečný automat, nedeterministický konečný automat, Hammingova vzdálenost, Levenshteinova vzdálenost, přibližné porovnávání vzorů

Abstract

The aim of this bachelor thesis is implementation of library for approximate pattern matching. This library will allow seeking of user specified pattern with specified number of maximum mistakes in text based on deterministic and nondeterministic finite automata. User will be able to choose between Hamming distance and Levenshtein distance. The first part describes use of finite automata for approximate pattern matching. The second part describes implementation of libraries. The third part focuses on experiments with implemented libraries. The conclusion of this thesis summarizes advantages and disadvantages of this approach to approximate pattern matching.

Key Words: deterministic finite automaton, nondeterministic finite automaton, Hamming distance, Levenshtein distance, approximate pattern matching

Obsah

Seznam použitých zkratek a symbolů

DKA	–	Deterministický konečný automat
NKA	–	Nedeterministický konečný automat

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Konečné automaty se v informatice dají využít k celé škále různorodých úkolů jako jsou slovní analýza či parsování při kompilaci. Tato práce se zaměřuje na využití konečných automatů k vyhledávání v textu pomocí přibližného porovnávání vzorů (approximate pattern matching). Pro porovnávání vzorů existuje řada algoritmů a v této práci se zaměříme na porovnávání vzorů pomocí Hammingovy vzdálenosti a Levenshteinovy vzdálenosti. Tyto algoritmy se dají řešit pomocí nedeterministických konečných automatů, jejichž nedeterminističnost se však může projevit jako problém, a proto prozkoumáme i možnosti převodu nedeterministického konečného automatu na deterministický. Ukážeme si však také možnosti simulace nedeterministických konečných automatů, včetně bitového paralelismu.

Součásti této práce je implementace knihovny konečných automatů, implementace knihovny generátorů nedeterministických konečných automatů pro vyhledávání v textu pomocí Hammingovy či Levenshteinovy vzdálenosti, implementace knihovny bitového paralelismu a důkladné experimenty s těmito knihovnami. Každý experiment obsahuje podrobný rozbor výsledků. Na závěr této práce si shrneme poznatky a určíme výhody a nevýhody tohoto přístupu k vyhledávání v textu, jeho možnosti využití v praxi a případná omezení.

2 Základní definice

V této kapitole definujeme základní pojmy, které jsou použity v této bakalářské práci.

Definice 1 *Abeceda Σ je neprázdná konečná množina symbolů (znaků).*

Definice 2 *Doplněk symbolu a v Σ, kde $a \in \Sigma$, je množina $\Sigma \backslash \{a\}$ a je označena $\overline{a}$.*

Definice 3 *Slovo v Σ je libovolná konečná posloupnost symbolů z Σ.*

Definice 4 *Množina všech slov v Σ je označena Σ^*.*

Definice 5 *Slovo a je podslovem slova b, jestliže platí $b = uav$, kde $a,b,u,v \in \Sigma^*$.*

Definice 6 *Délka slova u je počet symbolů ve slově $u \in \Sigma^*$ a je označena $|u|$.*

Definice 7 *Prázdné slovo je slovo o délce 0 a je označeno ϵ.*

Definice 8 *Pro jakoukoli množinu A platí, že množina všech podmnožin množiny A, se nazývá potenční množina množiny A a je označená $\mathcal{P}(A)$.*

2.1 Pozměňující operace

Definice 9 *Pozměňující operace substituce je taková operace, která změní slovo uav na slovo ubv, kde $u,v \in \Sigma^*$, $a,b \in \Sigma$, $a \neq b$ (symbol je nahrazen jiným symbolem).*

Definice 10 *Pozměňující operace vložení je taková operace, která změní slovo uv na slovo uav, kde $u,v \in \Sigma^*$, $a \in \Sigma$ (symbol je vložen do slova).*

Definice 11 *Pozměňující operace odstranění je taková operace, která změní slovo uav na slovo uv, kde $u,v \in \Sigma^*$, $a \in \Sigma$ (symbol je odstraněn ze slova).*

2.2 Vzdálenosti

Definice 12 *Hammingova vzdálenost $D_H(u,v)$ mezi dvěma slovy $u,v \in \Sigma^*$, $|u| = |v|$ je minimální počet pozměňujících operací substituce nutných k přeměně u na v.*

Definice 13 *Levenshteinova vzdálenost $D_L(u,v)$ mezi dvěma slovy $u,v \in \Sigma^*$, $|u| = |v|$ je minimální počet pozměňujících operací substituce, vložení a odstranění nutných k přeměně u na v.*

2.3 Porovnávání slov

Definice 14 *Nechť $U = \{u_1, u_2, ..., u_m\}$ a $V = \{v_1, v_2, ..., v_l\}$, $U \neq V$, jsou dva výskyty vzoru $P = p_1p_2...p_m$ v textu $T = t_1t_2...t_n$ (tj., $\forall j, 1 \leq j < m, 1 \leq u_j < u_{j+1} \leq n, 1 \leq v_j < v_{j+1} \leq na \forall j, 1 \leq j \leq m, p_j = t_{u_j}, p_j = t_{v_j}$). Výskyty U a V vzoru P se překrývají, jestliže platí $v_1 \leq u_1 \leq v_m$ nebo $u_1 \leq v_1 \leq u_m$, v opačném případě se výskyty nepřekrývají.*

Definice 15 *Přibližné porovnávání slov je definováno jako hledání výskytu vzoru $P = p_1p_2...p_m$ v textu $T = t_1t_2...t_n$ s nejvýše k povolenými chybami. Počet povolených chyb v nalezeném podslově je určen Hammingovou nebo Levenshteinovou vzdáleností. Přibližné porovnávání slov pomocí Hammingovy vzdálenosti se nazývá přibližné porovnávání slov s k neshodami a přibližné porovnávání slov pomocí Levenshteinovy vzdálenosti se nazývá přibližné porovnávání slov s k rozdíly.*

Definice 16 *Triviální výskyt s j chybami nastane připojením symbolu za konec předchozího výskytu s j - 1 chybami.*

Pozn.: Připojením symbolu dosáhneme pouze navýšení počtu chyb, což je nežádoucí, a proto tato práce triviální výskyt nebere v potaz.

Definice 17 *Přesné porovnávání slov je přibližné porovnávání slov, kde k = 0.*

2.4 Konečné automaty

Definice 18 *Nedeterministický konečný automat (NKA) je pětice $(Q, \Sigma, \delta, q_0, F)$, kde Q je neprázdná konečná množina stavů, Σ je abeceda, δ je přechodová funkce $Q \times (\Sigma \cup \{\epsilon\}) \mapsto \mathcal{P}(Q)$, $q_0 \in Q$ je počáteční stav, a $F \subseteq Q$ je množina přijímajících stavů. [2]*

Pozn.: Existují definice, které nám definují NKA s množinou počátečních stavů $Q_0 \subseteq Q$, namísto jednoho počátečního stavu q_0. Takovéto NKA však v této práci nebudeme uvažovat, protože NKA, které si ukážeme, mají vždy pouze jeden počáteční stav.

Definice 19 *Nechť $M = (Q, \Sigma, \delta, q_0, F)$ je NKA a $P \subseteq Q$. ϵ-uzávěr(P) je množina $\{ q' \mid q' \in \delta(q, \epsilon), q \in P\} \cup P$.*

Definice 20 *Deterministický konečný automat (DKA) je pětice $(Q, \Sigma, \delta, q_0, F)$, kde Q je neprázdná konečná množina stavů, Σ je abeceda, δ je přechodová funkce $Q \times \Sigma \mapsto \mathcal{P}(Q)$, $q_0 \in Q$ je počáteční stav, a $F \subseteq Q$ je množina přijímajících stavů. [2]*

Definice 21 *Konečný automat (KA) je NKA nebo DKA.*

Definice 22 *Jazyk přijímaný konečným automatem $M = (Q, \Sigma, \delta, q_0, F)$ je množina $L(M) = \{u \in \Sigma^* \mid (q_0, u) \vdash_M^* (q_f, \epsilon), q_f \in F\}$.*

Definice 23 *Dva konečné automaty jsou ekvivalentní, jestliže platí $L(M) = L(M')$.*

Definice 24 *Nepřístupný stav konečného automatu* $M = (Q, \Sigma, \delta, q_0, F)$ *je stav* q *takový, pro který neexistuje slovo* $u \in \Sigma^*$ *takové, pro které platí* $(q_0, u) \vdash_M^* (q, \epsilon)$.

Definice 25 *Aktivní stav konečného automatu* $M = (Q, \Sigma, \delta, q_0, F)$ *po přečtení slova* $u \in \Sigma^*$ *je každý stav* q, *pro který platí* $(q_0, u) \vdash_M^* (q, \epsilon)$.

Definice 26 *Hloubka stavu* q *v konečném automatu* $M = (Q, \Sigma, \delta, q_0, F)$, $q \in Q$, *je minimální počet tahů nutných k přechodu z počátečního stavu* q_0 *do tohoto stavu* q *bez použití* ϵ*-přechodů.*

Definice 27 *Úroveň stavu* q *v NKA* $M = (Q, \Sigma, \delta, q_0, F)$, $q \in Q$, *je minimální počet chyb dosažitelný ze stavu* q.

3 Konečné automaty v porovnávání vzorů

V této kapitole si rozebereme *NKA* pro přesné a přibližné porovnávání vzorů. Nejprve si však přiblížíme možnosti použití deterministických a nedeterministických automatů pro porovnávání vzorů. Také se podíváme na převod *NKA* na ekvivalentní *DKA*.

3.1 Nedeterministické konečné automaty

V této práci budeme konstruovat *NKA* (viz definice 18) tak, aby dokázaly nalézt první výskyt vzoru P v textu T. Tyto *NKA* však mohou také pokračovat v prohledávání textu T a najít všechny ostatní překrývající se výskyty (viz definice 14) vzoru P. V každém kroku automatu kontrolujeme, zda se se nacházíme v některém z přijímajících stavů. Proto tyto automaty mohou určit:

- zda se vzor P nachází v textu T,
- první výskyt vzoru P v textu T,
- počet všech výskytů vzoru P v textu T a
- všechny výskyty vzoru P v textu T.

Je také možné sestrojit takové *NKA*, které po přečtení vstupního textu T pouze určí, zda se vzor P v textu nalézá, či nikoli. Tyto *NKA* mohou být sestrojeny umístěním smyček do všech přijímajících stavů. Tyto smyčky obsahují všechny symboly abecedy Σ a zajistí, že po dosažení přijímajícího stavu zůstane tento stav aktivní po celou dobu běhu *NKA*. Po přečtení vstupního textu se zkontroluje, zda je některý z konečných stavů aktivní a určí, zda automat vzor nalezl či nikoli, v čemž také spočívá nevýhoda těchto automatů. Tato kontrola může být provedena až po přečtení celého vstupního textu, což může být v případech nalezení vzoru na začátku dlouhých textů velmi nevýhodné.

NKA, kterými se budeme zabývat a které budeme konstruovat, hledají všechny překrývající se výskyty vzoru P v textu T. Například, máme vzor $P = aa$ a vstupní text $T = aabaaa$. *NKA* nám určí, že se vzor P v textu T nalézá třikrát a to na pozicích $a\underline{a}baaa$, $aaba\underline{a}a$ a $aabaa\underline{a}$.

3.2 Deterministické konečné automaty

Pro vyhledávání v textech nemohou být *NKA* použity přímo díky jejich nedeterminismu, který zajišťuje, že se z jednoho stavu automatu můžeme jedním symbolem dostat rovnou do několika stavů. Také může nastat situace, kdy pro daný symbol neexistuje přechod. V tom případě se nedostaneme do žádného stavu. Máme však dvě možnosti, jak *NKA* využít:

- Simulovat běh *NKA* a
- Převést *NKA* na ekvivalentní *DKA* (viz definice 20).

Simulacemi běhu *NKA* se budeme zabývat v další kapitole. Nyní si blíže přiblížíme možnost převodu *NKA* na ekvivalentní *DKA*. Tento převod se provádí standardní podmnožinovou konstrukcí, která je popsána v algoritmu 1.

Algoritmus 1: Převod *NKA* na ekvivalentní *DKA* [1]

Vstup : *NKA* $M = (Q, \Sigma, \delta, q_0, F)$
Výstup : *NKA* M' přijímající jazyk L(M)
Metoda: $M' = (Q', \Sigma, \delta', q_0', F')$ kde Q', δ', q_0', a F' jsou sestrojeny následovně:
$Q' \leftarrow \mathcal{P}(\mathrm{Q})$
$\delta'(A, a) \leftarrow \bigcup_{q \in A} \epsilon\text{-}uzávěr(\delta(q, a)), \forall a \in \Sigma, A \in Q'$
$q_0' \leftarrow \epsilon\text{-}uzávěr(\{q_0\})$
$F' \leftarrow \{A \mid A \in Q', A \cap F \neq \varnothing\}$

Tento algoritmus sestrojí *DKA* s $2^{|Q|}$ stavy, což se však u *NKA* s vyšším počtem stavů projeví jako velmi neefektivní řešení. Nejenže počet stavů ekvivalentního *DKA* roste exponenciálně, ale také je mnoho těchto stavů nepřístupných (viz definice 24). Z tohoto důvodů používáme upravenou verzi podmnožinové konstrukce, která do výsledného ekvivalentního *DKA* nevkládá nepřístupné stavy. Tato konstrukce je popsána v algoritmu 2.

Algoritmus 2: Převod *NKA* na ekvivalentní *DKA* [1]

Vstup : *NKA* $M = (Q, \Sigma, \delta, q_0, F)$
Výstup : *DKA* M' přijímající jazyk L(M)
Metoda: $M' = (Q', \Sigma, \delta', q_0', F')$ kde Q', δ', q_0', a F' jsou sestrojeny následovně:
$q_0' \leftarrow \epsilon\text{-}uzávěr(\{q_0\})$
$Q' \leftarrow \varnothing$
$S \leftarrow \{q_0'\}$ ▷ množina aktivních stavů
for each $q' \in S$ **do**
  for each $a \in \Sigma$ **do**
    $\delta'(q', a) \leftarrow \bigcup_{q \in q'} \epsilon\text{-}uzávěr(\delta(q, a))$
    if $\delta'(q', a) \notin Q'$ **and** $\delta'(q', a) \notin S$ **then**
      $S \leftarrow S \cup \delta'(q', a)$
    endif
  endfor
  if $q' \cap F \neq \varnothing$ **then**
    $F' \leftarrow F' \cup \{q'\}$
  endif
  $S \leftarrow S \backslash \{q'\}$
  $Q' \leftarrow Q' \cup \{q'\}$
endfor

DKA jsou, díky jejich determinismu, ideální pro vyhledávání v textu. Mají lineární asymptotickou složitost $\mathcal{O}(n)$, kde n je délka vstupního textu T. Dá se tedy říci, že *DKA* jsou závislé na délce vstupního textu. Jako velká nevýhoda se zatím jeví doba převodu *NKA* na *DKA*, která je závislá na vstupní abecedě Σ, počtu stavů Q a velikostí přechodové funkce *NKA*.

3.3 Porovnávání vzorů

3.3.1 Přesné porovnávání vzorů (Exact string matching)

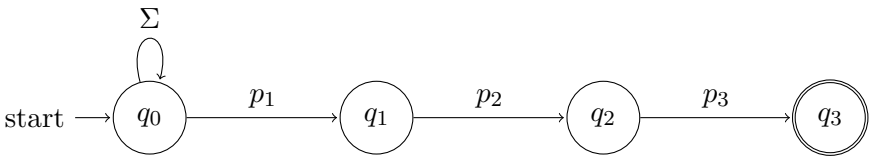
Přesné porovnávání vzorů je nejjednodušší variantou přibližného porovnávání vzorů. Slouží k vyhledávání všech překrývajících se výskytů vzoru $P = p_1p_2...p_m$ v textu $T = t_1t_2...t_n$. Proto nám stačí sestrojit jednoduché *NKA* takové, které přijímají jazyk $L(M) = \{uP \mid u \in \Sigma^*\}$. V tomto *NKA* představuje každý stav q_i, krom počátečního stavu, q_0, počet prvních i nalezených symbolů vzoru P ve vstupním textu T. Abychom mohli přeskočit symboly neodpovídající vzoru, přidáme na počáteční stav smyčku všech symbolů abecedy Σ. Pro představu je konstrukce tohoto *NKA* popsána v algoritmu 3.

Algoritmus 3: Konstrukce *NKA* pro přesné porovnávání vzorů [1]

Vstup : Vzor $P = p_1p_2...p_m$
Výstup : *NKA* M přijímající jazyk $L(M) = \{uP \mid u \in \Sigma^*\}$
Metoda: *NKA* $M = (\{q_0, q_1, ..., q_m\}, \Sigma, \delta, q_0, \{q_m\})$, kde přechodová funkce δ je sestrojena následovně:

$\delta(q_0, a) \leftarrow \{q_0\}, \forall a \in \Sigma \backslash \{p_1\}$ ▷ smyčky na počáteční stav
$\delta(q_0, p_1) \leftarrow \{q_0, q_1\}$ ▷ smyčka a dopředný přechod pro znak p_1
for $i \leftarrow 1, 2, ..., m-1$ **do**
  $\delta(q_i, p_{i+1}) \leftarrow \{q_{i+1}\}$ ▷ dopředné přechody
endfor
$\delta(q_m, a) \leftarrow \varnothing, \forall a \in \Sigma$ ▷ z přijímajícího stavu nevede žádný přechod

Mějme vzor P délky $m = 3$. Algoritmus 3 nám sestrojí čtyřstavový *NKA*, který je znázorněn na obrázku 1.



Obrázek 1: *NKA* pro přesné porovnávání vzoru pro vzor P délky 4

V této práci se dále přesným porovnáváním vzorů zabývat nebudeme, protože se jedná o triviální případ přibližného porovnávání vzorů s chybou 0 (viz definice 17). Poslouží nám však jako základ pro *NKA*, které již budou umět porovnávat vzory s chybou. Tyto *NKA* si představíme v následujících podkapitolách.

3.3.2 Přibližné porovnávání vzorů pomocí Hammingovy vzdálenosti

U přibližného porovnávání vzorů pomocí Hammingovy vzdálenosti pro vzor $P = p_1p_2...p_m$ musíme vzít v úvahu nejvýše k povolené chyby (viz definice 15). Proto používáme $k+1$ kopií *NKA* pro přesné porovnávání vzorů $(M_0, M_1...M_k)$, kde automat M_0 slouží pro případ žádné chyby,

automat M_1 slouží pro případ jedné chyby,..., automat M_k slouží pro případ k chyb. Tyto k+1 automaty jsou propojeny přechody představující pozměňující operaci *substituce* (viz definice 9). Každý takový přechod je označen symbolem $\overline{p}_{j+1}$, který představuje doplněk symbolu p_{j+1} (viz definice 2) vzoru P a vede ze stavu q_j automatu M_i do stavu q_{j+1} automatu M_{i+1}, $0 \leq i < k$, $0 \leq j < m$. Tahy těmito přechody se tedy zvyšuje nejmenší počet chyb a hloubka automatu. Počátečním stavem tohoto automatu je počáteční stav automatu M_0. Nepřístupné stavy výsledného automatu jsou odstraněny. Konstrukce tohoto automatu je popsána v algoritmu 4.

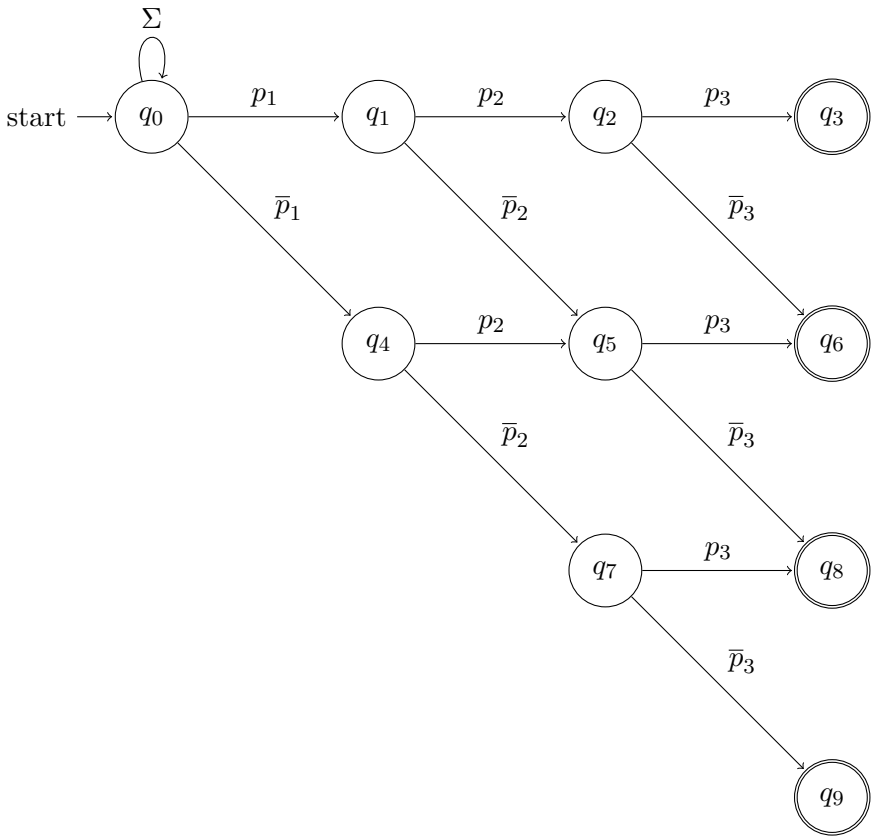
Algoritmus 4: Konstrukce *NKA* pro přibližné porovnávání vzorů pomocí Hammingovy vzdálenosti (verze $\overline{p}$) [1]

Vstup : Vzor $P = p_1p_2...p_m$, nejvyšší počet povolených chyb k, $k < m$
Výstup : *NKA* M přijímající jazyk $L(M) = \{uv \mid u, v \in \Sigma^*, D_H(P, v) \leq k\}$
Metoda: *NKA* $M = (Q, \Sigma, \delta, q_0, F)$, kde Q, δ a F jsou sestrojeny následovně:

```
Q ← {q_0, q_1, ..., q_{|Q|-1}}, |Q| = (k + 1)(m + 1 − k/2)
l ← 0                                              ▷ úroveň stavu
r ← 1                                              ▷ hloubka stavu
for i ← 0, 1, ..., |Q| − 1 do
  if r > m then
    δ(q_i, a) ← ∅, ∀a ∈ Σ                          ▷ z přijímajícího stavu nevede žádný přechod
    F ← F ∪ {q_i}
    l ← l + 1                                      ▷ posun na další úroveň
    r ← l + 1                                      ▷ přesun na první stav další úrovně
  else
    δ(q_i, p_r) ← {q_{i+1}}                        ▷ odpovídající přechod
    if l < k then
      s ← i + m + 1 − l                            ▷ číslo pravého nižšího sousedního stavu
      δ(q_i, a) ← {q_s}, ∀a ∈ Σ \{p_r}             ▷ substituční přechod
    endif
    r ← r + 1                                      ▷ posun v hloubce úrovně l
  endif
endfor
δ(q_0, a) ← δ(q_0, a) ∪ {q_0}, ∀a ∈ Σ              ▷ smyčky na počáteční stav
```

Mějme vzor P délky $m = 3$ a chybou $k = 3$. Algoritmus 4 nám sestrojí *NKA*, který je znázorněn na obrázku 2.

Další možností, jak reprezentovat pozměňující operace *substituce* v *NKA* je označení přechodů symbolem Σ, namísto symbolu $\overline{p}_{j+1}$. Tyto *NKA* (Σ verze) se konstruují podobně jako $\overline{p}$ verze, a to spojením k+1 kopií *NKA* nebo pomocí algoritmu 5, který je upravenou verzí algoritmu 4. Rozdíl spočívá v existenci přechodu označeného symbolem p_{j+1} vedoucího ze stavu q_j automatu M_i do stavu q_{j+1} automatu M_{i+1}, $0 \leq i < k$, $0 \leq j < m$. Jelikož pak existuje přechod ze stavu q_j automatu M_i do stavu q_{j+1} automatu M_{i+1} pro každý symbol abecedy Σ, označujeme tyto přechody souhrnně symbolem Σ.



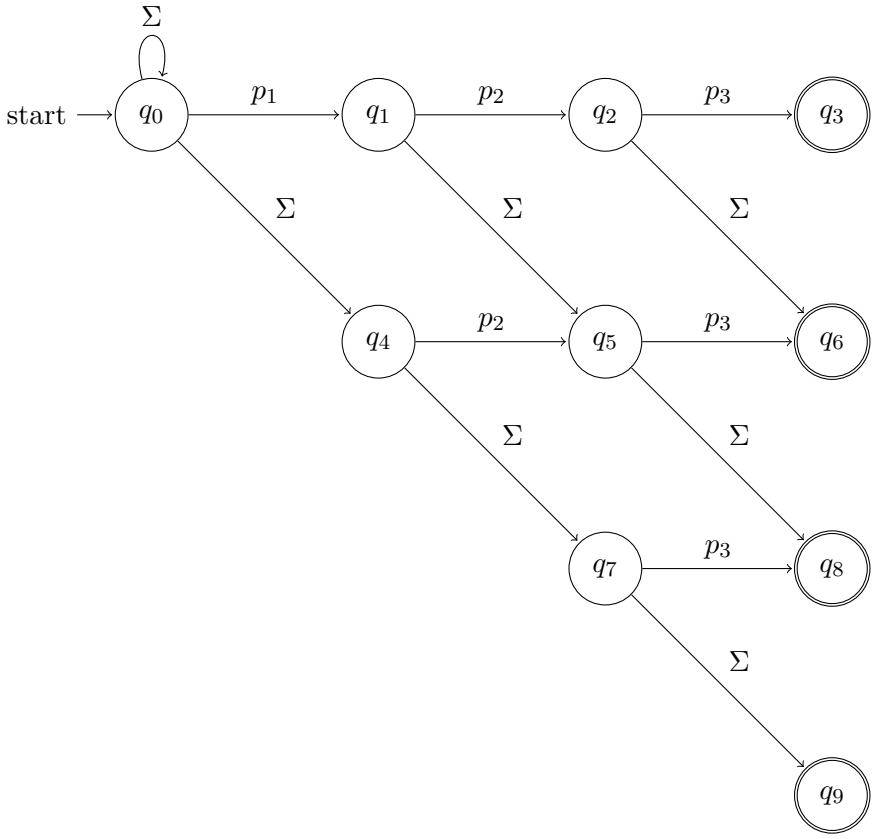
Obrázek 2: *NKA* pro přibližné porovnávání vzoru pomocí Hammingovy vzdálenosti pro vzor P délky 3 s chybou 3 (verze $\overline{p}$)

Algoritmus 5: Konstrukce *NKA* pro přibližné porovnávání vzorů pomocí Hammingovy vzdálenosti (verze Σ) [1]

Vstup : Vzor $P = p_1 p_2 ... p_m$, nejvyšší počet povolených chyb k, $k < m$
Výstup : *NKA* M přijímající jazyk $L(M) = \{uv \mid u, v \in \Sigma^*, D_H(P, v) \leq k\}$
Metoda: *NKA* $M = (Q, \Sigma, \delta, q_0, F)$, kde Q, δ a F jsou sestrojeny následovně:

```
Q ← {q_0, q_1, ..., q_{|Q|-1}}, |Q| = (k + 1)(m + 1 − k/2)
l ← 0                                          ▷ úroveň stavu
r ← 1                                          ▷ hloubka stavu
for i ← 0, 1, ..., |Q| − 1 do
  if r > m then
    δ(q_i, a) ← ∅, ∀a ∈ Σ                      ▷ z přijímajícího stavu nevede žádný přechod
    F ← F ∪ {q_i}
    l ← l + 1                                  ▷ posun na další úroveň
    r ← l + 1                                  ▷ přesun na první stav další úrovně
  else
    if l < k then
      s ← i + m + 1 − l                        ▷ číslo pravého nižšího sousedního stavu
      δ(q_i, p_r) ← {q_{i+1}, q_s}             ▷ odpovídající a substituční přechody
      δ(q_i, a) ← {q_s}, ∀a ∈ Σ \{p_r}         ▷ substituční přechod
    else
      δ(q_i, p_r) ← {q_{i+1}}                  ▷ odpovídající přechod na úrovni k
    endif
    r ← r + 1                                  ▷ posun v hloubce úrovně l
  endif
endfor
δ(q_0, a) ← δ(q_0, a) ∪ {q_0}, ∀a ∈ Σ          ▷ smyčky na počáteční stav
```

Mějme vzor P délky $m = 3$ a chybou $k = 3$. Algoritmus 5 nám sestrojí *NKA*, který je znázorněn na obrázku 3.



Obrázek 3: *NKA* pro přibližné porovnávání vzoru pomocí Hammingovy vzdálenosti pro vzor P délky 3 s chybou 3 (verze Σ)

3.3.3 Přibližné porovnávání vzorů pomocí Levenshteinovy vzdálenosti

U přibližného porovnávání vzorů pomocí Levenshteinovy vzdálenosti pro vzor $P = p_1p_2...p_m$ můžeme, stejně jako u přibližného porovnávání vzorů pomocí Hammingovy vzdálenosti, použít $k+1$ kopií *NFA* pro přesné porovnávání vzorů $(M_0, M_1...M_k)$ a spojit je pozměňující operací *substituce*. Levenshteinova vzdálenost však k porovnávání používá, krom operace *substituce*, také operace *vložení* (viz definice 10) a *odstranění* (viz definice 11). Náš výsledný *NFA* tedy musí obsahovat tyto operace.

Přechody představující pozměňující operaci *vložení* jsou označeny symbolem $\overline{p}_{j+1}$ a vedou ze stavu q_j automatu M_i do stavu q_j automatu M_{i+1}, $0 \leq i < k$, $0 < j < m$. Tahy těmito přechody se zvyšuje nejmenší počet chyb, hloubka automatu se však nemění.

Přechody představující pozměňující operaci *odstranění* jsou označeny symbolem ϵ a vedou ze stavu q_j automatu M_i do stavu q_{j+1} automatu M_{i+1}, $0 \leq i < k$, $0 < j < m$. Tahy těmito

přechody se zvyšuje nejmenší počet chyb a hloubka automatu. Na rozdíl od pozměňujících operací *substituce* a *vložení* však nedojde k přečtení symbolu ze vstupního textu T.

Počátečním stavem tohoto automatu je počáteční stav automatu M_0. Nepřístupné stavy výsledného automatu jsou odstraněny. Konstrukce tohoto automatu je popsána v algoritmu 6.

Algoritmus 6: Konstrukce *NKA* pro přibližné porovnávání vzorů pomocí Levenshteinovy vzdálenosti (verze $\overline{p}$) [1]

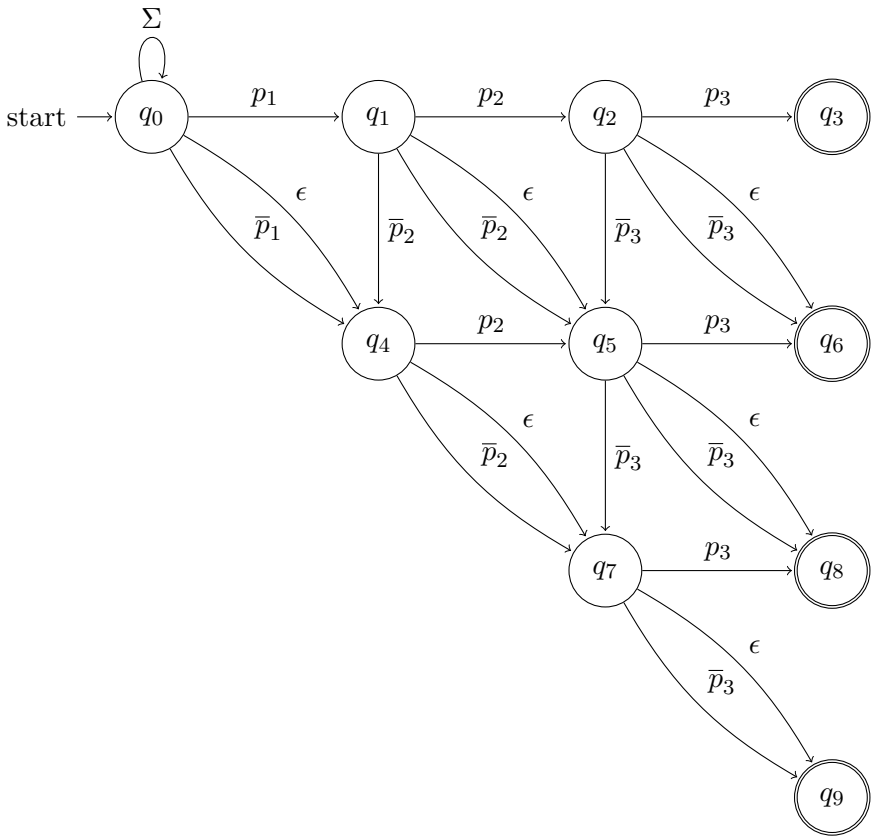
Vstup : Vzor $P = p_1p_2...p_m$, nejvyšší počet povolených chyb k, $k < m$
Výstup : *NKA* M přijímající jazyk $L(M) = \{uv \mid u, v \in \Sigma^*,\ D_L(P, v) \leq k\}$
Metoda: *NKA* $M = (Q, \Sigma, \delta, q_0, F)$, kde Q, δ a F jsou sestrojeny následovně:

$Q \leftarrow \{q_0, q_1, ..., q_{|Q|-1}\}$, $|Q| = (k+1)(m+1-\frac{k}{2})$
$l \leftarrow k$ ▷ úroveň stavu
$r \leftarrow m$ ▷ hloubka stavu
for $i \leftarrow |Q|-1, |Q|-2, ..., 0$ **do**
  if $r = m$ **then**
    $\delta(q_i, a) \leftarrow \varnothing, \forall a \in \Sigma$ ▷ z přijímajícího stavu nevede žádný přechod
    $F \leftarrow F \cup \{q_i\}$
  else
    $\delta(q_i, p_{r+1}) \leftarrow \{q_{i+1}\}$ ▷ odpovídající přechod
    if $l < k$ **then**
      $s \leftarrow i + m + 1 - l$ ▷ číslo pravého nižšího sousedního stavu
      $\delta(q_i, a) \leftarrow \{q_s\}, \forall a \in \Sigma \setminus \{p_{r+1}\}$ ▷ substituční přechod
      if $r > l$ **then**
        $\delta(q_i, a) \leftarrow \delta(q_i, a) \cup \{q_{s-1}\}, \forall a \in \Sigma \setminus \{p_{r+1}\}$ ▷ vkládající přechod
      endif
      $\delta(q_i, a) \leftarrow \delta(q_i, a) \cup \delta(q_s, a), \forall a \in \Sigma$ ▷ odstraňující přechod
    else
      $\delta(q_i, a) \leftarrow \varnothing, \forall a \in \Sigma \setminus \{p_{r+1}\}$
    endif
  endif
  $r \leftarrow r - 1$ ▷ zpětný posun v hloubce úrovně l
  if $r < l$ **then**
    $r \leftarrow m$ ▷ přesun na přijímací stav předchozí úrovně
    $l \leftarrow l - 1$ ▷ posun na předchozí úroveň
  endif
endfor
$\delta(q_0, a) \leftarrow \delta(q_0, a) \cup \{q_0\}, \forall a \in \Sigma$ ▷ smyčky na počáteční stav

---

Mějme vzor P délky $m = 3$ a chybou $k = 3$. Algoritmus 6 nám sestrojí *NKA*, který je znázorněn na obrázku 4.

Stejně jako u přibližného porovnávání vzorů pomocí Hammingovy vzdálenosti, můžeme i v těchto *NKA* reprezentovat pozměňující operaci *substituce*, a nově také pozměňující operaci *vložení*, označením přechodů symbolem Σ, namísto symbolu $\overline{p}_{j+1}$. Tyto *NKA* (Σ verze) se konstruují podobně jako $\overline{p}$ verze, a to spojením k+1 kopií *NKA* nebo pomocí algoritmu 7, který je upravenou verzí algoritmu 6. Rozdílem jsou přechody pozměňující operace *substituce* a *vložení*



Obrázek 4: *NKA* pro přibližné porovnávání vzoru pomocí Levenshteinovy vzdálenosti pro vzor P délky 3 s chybou 3 (verze $\overline{p}$)

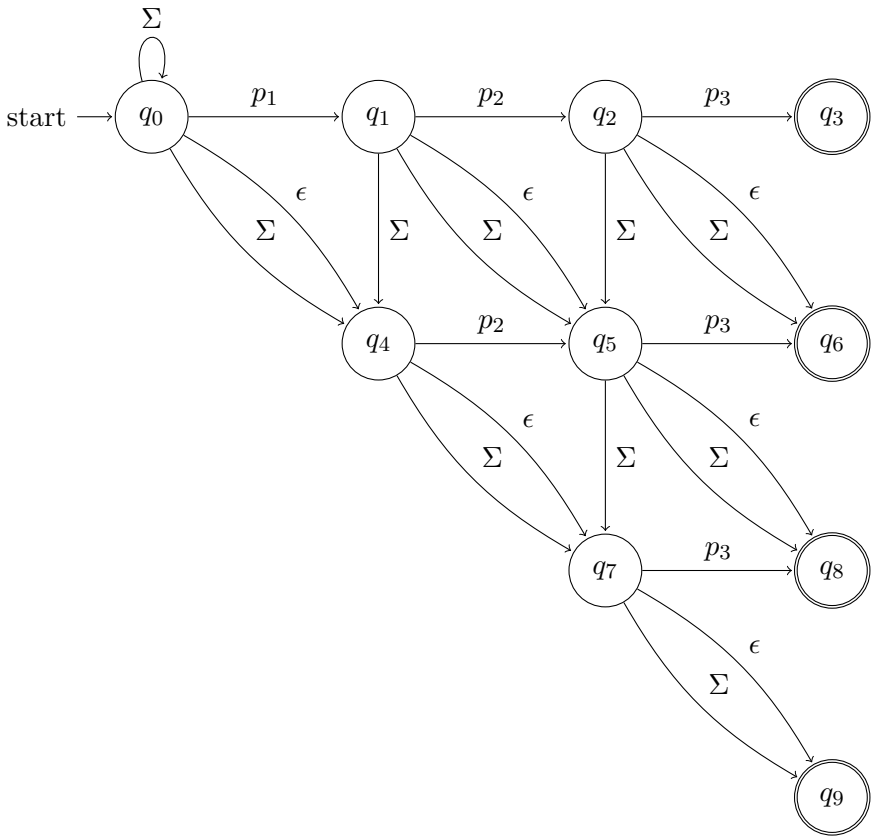
doplněné o přechod pro symbol p_{j+1}. Jelikož pak existuje přechod pro všechny symboly abecedy Σ, označujeme tyto přechody souhrnně symbolem Σ.

Mějme vzor P délky $m = 3$ a chybou $k = 3$. Algoritmus 7 nám sestrojí *NKA*, který je znázorněn na obrázku 5.

Algoritmus 7: Konstrukce *NKA* pro přibližné porovnávání vzorů pomocí Levenshteinovy vzdálenosti (verze Σ) [1]

Vstup : Vzor $P = p_1 p_2 ... p_m$, nejvyšší počet povolených chyb k, $k < m$
Výstup : *NKA* M přijímající jazyk $L(M) = \{uv \mid u, v \in \Sigma^*,\ D_L(P, v) \leq k\}$
Metoda: *NKA* $M = (Q, \Sigma, \delta, q_0, F)$, kde Q, δ a F jsou sestrojeny následovně:

$Q \leftarrow \{q_0, q_1, ..., q_{|Q|-1}\}$, $|Q| = (k+1)(m+1-\frac{k}{2})$
$l \leftarrow k$ ▷ úroveň stavu
$r \leftarrow m$ ▷ hloubka stavu
for $i \leftarrow |Q|-1, |Q|-2, ..., 0$ **do**
  if $r = m$ **then**
    $\delta(q_i, a) \leftarrow \varnothing, \forall a \in \Sigma$ ▷ z přijímajícího stavu nevede žádný přechod
    $F \leftarrow F \cup \{q_i\}$
  else
    if $l < k$ **then**
      $s \leftarrow i + m + 1 - l$ ▷ číslo pravého nižšího sousedního stavu
      $\delta(q_i, p_{r+1}) \leftarrow \{q_{i+1}, q_s\}$ ▷ odpovídající a substituční přechody
      $\delta(q_i, a) \leftarrow \{q_s\}, \forall a \in \Sigma \setminus \{p_{r+1}\}$ ▷ substituční přechod
      if $r > l$ **then**
        $\delta(q_i, a) \leftarrow \delta(q_i, a) \cup \{q_{s-1}\}, \forall a \in \Sigma$ ▷ vkládající přechod
      endif
      $\delta(q_i, a) \leftarrow \delta(q_i, a) \cup \delta(q_s, a), \forall a \in \Sigma$ ▷ odstraňující přechod
    else
      $\delta(q_i, p_{r+1}) \leftarrow \{q_{i+1}\}$ ▷ odpovídající přechod na úrovni k
      $\delta(q_i, a) \leftarrow \varnothing, \forall a \in \Sigma \setminus \{p_{r+1}\}$
    endif
  endif
  $r \leftarrow r - 1$ ▷ zpětný posun v hloubce úrovně l
  if $r < l$ **then**
    $r \leftarrow m$ ▷ přesun na přijímající stav předchozí úrovně
    $l \leftarrow l - 1$ ▷ posun na předchozí úroveň
  endif
endfor
$\delta(q_0, a) \leftarrow \delta(q_0, a) \cup \{q_0\}, \forall a \in \Sigma$ ▷ smyčky na počáteční stav



Obrázek 5: *NKA* pro přibližné porovnávání vzoru pomocí Levenshteinovy vzdálenosti pro vzor P délky 3 s chybou 3 (verze Σ)

4 Možnosti simulace NKA

4.1 Základní simulace běhu NKA

Simulovat běh *NKA* pomocí této metody je nutné tehdy, pokud doba převodu *NKA* na ekvivalentní *DKA* přesahuje námi tolerovanou mez. Algoritmus pro simulaci běhu *NKA* je popsán v algoritmu 8. Tento algoritmus je podobný algoritmu 2 pro převod *NKA* na ekvivalentní *DKA*. V každém kroku simulace *NKA* je vytvořena nová množina aktivních stavů S, která je vytvořena tahy všemi dostupnými přechody ze stavů předchozí množiny aktivních stavů. Prozkoumáme tak všechny možné cesty. Pokud je některý ze stavů množiny aktivních stavů obsažen v množině přijímajících stavů F, byl vzor nalezen. Simulace končí po přečtení celého vstupního textu nebo tehdy, pokud je množina aktivních stavů prázdná.

Algoritmus 8: Simulace běhu *NKA* [1]

Vstup : *NKA* $M = (Q, \Sigma, \delta, q_0, F)$, vstupní text $T = t_1 t_2 ... t_n$
Výstup : Výstup simulace *NKA*
Metoda: Je použita množina aktivních stavů S

```
S ← ϵ-uzávěr({q_0})
i ← 1
while i ≤ n and S ≠ ∅ do
  S ← ⋃_{q∈S} ϵ-uzávěr(δ(q, t_i))
  if q' ∩ F ≠ ∅ then
    write(Informace spojené s každým přijímajícím stavem z S ∩ F)
  endif
  i ← i + 1
endwhile
```

4.2 Bitový paralelismus

Bitový paralelismus je metoda simulace běhu *NKA*, která využívá bitové vektory. Nad těmito celými bitovými vektory provádí bitové operace (např. *OR*, *AND*, *ADD*). Představiteli bitového paralelismus jsou algoritmy Shift-Or, Shift-And, Shift-Add. V této práci se zaměříme na použití algoritmu Shift-Or.

4.2.1 Shift-Or algoritmus

Shift-Or algoritmus používá matice R^l, $0 \leq l \leq k$ velikosti $m \times (n+1)$ a maskovací matici D o velikosti $m \times |\Sigma|$. Každý prvek $r^l_{j,i}$, $0 < j \leq m$, $0 \leq i \leq n$, obsahuje 0, pokud je vzdálenost mezi vzorem $p_1 ... p_j$ a koncem slova v textu T na pozici i menší nebo rovno l. V opačném případě prvek

obsahuje 1. Každý prvek $d_{j,x}$, $0 < j \leq m$, $x \in \Sigma$, obsahuje 0, pokud $p_j = x$. V opačném případě prvek obsahuje 1. Matice jsou implementovány jako tabulky bitových vektorů následovně [1]:

$$R_i^l = \begin{bmatrix} r_{1,i}^l \\ r_{2,i}^l \\ . \\ . \\ . \\ r_{m,i}^l \end{bmatrix} \text{ and } D[x] = \begin{bmatrix} d_{1,x} \\ d_{2,x} \\ . \\ . \\ . \\ d_{m,x} \end{bmatrix}, 0 \leq i \leq n, 0 \leq l \leq k, x \in \Sigma \tag{1}$$

4.2.2 Přesné porovnávání vzorů

U přesného porovnávání vzorů jsou bitové vektory R_i^0, $0 \leq i \leq n$, počítány následovně [1]:

$$\begin{aligned} r_{j,0}^0 &\leftarrow 1, & 0 < j \leq m \\ R_i^0 &\leftarrow \mathbf{shl}(R_{i-1}^0) \text{ OR } D[t_i], & 0 < i \leq n \end{aligned} \tag{2}$$

Operace **shl**() ve formuli 2 představuje bitovou operaci left shift, která vkládá 0 na začátek vektoru. Operace OR představuje bitovou operaci or. Term $\mathbf{shl}(R_{i-1}^0)$ OR $D[t_i]$ představuje shodu, která probíhá tak, že pozice i v textu T je inkrementována, pozice ve vzoru P je inkrementována operací **shl**() a pozice odpovídající vstupnímu symbolu t_i jsou vybrány termem OR $D[t_i]$. Vzor P je nalezen na pozici $t_{i-m+1}...t_i$, pokud $r_{m,i}^0 = 0$, $0 < i \leq n$.

Příklad maskovací matice D pro vzor $P = aab$ je znázorněn v tabulce 1. Příklad matice R^0 pro přesné porovnávání vzorů pro vzor $P = aab$ a text $T = aabbab$ je znázorněn v tabulce 2.

Tabulka 1: Matice D pro vzor $P = aab$

D	a	b	c
a	0	1	1
a	0	1	1
b	1	0	1

Tabulka 2: Matice R^0 pro přesné porovnávání vzorů pro vzor $P = aab$ a text $T = aabbab$

R^0	-	a	a	b	b	a	b
a	1	**0**	**0**	1	1	**0**	1
a	1	1	**0**	1	1	1	1
b	1	1	1	**0**	1	1	1

Tento Shift-Or algoritmus, popsaný formulí 2, simuluje běh *NKA* pro přesné porovnávání vzorů sestrojeného algoritmem 3. [1]

4.2.3 Přibližné porovnávání vzorů pomocí Hammingovy vzdálenosti

U přibližného porovnávání vzorů pomocí Hammingovy vzdálenosti jsou vektory R_i^l, $0 \leq l \leq k$, $0 \leq i \leq n$, počítány následovně [1]:

$$\begin{aligned} r_{j,0}^l &\leftarrow 1, && 0 < j \leq m,\ 0 \leq l \leq k \\ R_i^0 &\leftarrow \mathbf{shl}(R_{i-1}^0)\ \text{OR}\ D[t_i], && 0 < i \leq n \\ R_i^l &\leftarrow (\mathbf{shl}(R_{i-1}^l)\ \text{OR}\ D[t_i])\ \text{AND}\ \mathbf{shl}(R_{i-1}^{l-1}), && 0 < i \leq n,\ 0 < l \leq k \end{aligned} \tag{3}$$

Operace AND ve formuli 3 představuje bitovou operaci and. Term $\mathbf{shl}(R_{i-1}^0)$ OR $D[t_i]$ představuje shodu a term $\mathbf{shl}(R_{i-1}^{l-1})$ představuje pozměňující operaci *substituce*, která probíhá tak, že pozice i v textu T je inkrementována, pozice ve vzoru P je inkrementována a vzdálenost l je inkrementována. Vzor P je nalezen s nanejvýš k chybami na pozici $t_{i-m-1}...t_i$, pokud $r_{m,i}^k = 0$, $0 < i \leq n$. Nejvyšší počet chyb nalezeného vzoru je $D_H(P, t_{i-m-1}...t_i) = l$, kde l je nejmenší počet takový, že $r_{m,i}^l = 0$.

Příklad matic R^l pro přibližné porovnávání vzorů pomocí Hammingovy vzdálenosti pro vzor $P = aab$ s chybou $k = 2$ a text $T = aabbab$ je znázorněn v tabulce 3.

Tabulka 3: Matice R^l pro přibližné porovnávání vzorů pomocí Hammingovy vzdálenosti pro vzor $P = aab$ s chybou $k = 2$ a text $T = aabbab$

R^0	-	a	a	b	b	a	b
a	1	**0**	**0**	1	1	**0**	1
a	1	1	**0**	1	1	1	1
b	1	1	1	**0**	1	1	1
R^1	-	a	a	b	b	a	b
a	1	**0**	**0**	**0**	**0**	**0**	**0**
a	1	1	**0**	**0**	1	**0**	**0**
b	1	1	1	**0**	**0**	1	**0**
R^2	-	a	a	b	b	a	b
a	1	**0**	**0**	**0**	**0**	**0**	**0**
a	1	1	**0**	**0**	**0**	**0**	**0**
b	1	1	1	**0**	**0**	1	**0**

Tento Shift-Or algoritmus, popsaný formulí 3, simuluje běh *NKA* pro přibližné porovnávání vzorů pomocí Hammingovy vzdálenosti sestrojeného algoritmem 5. Jedná se tedy o verzi Σ.[1] Shift-Or algoritmem pro simulaci běhu *NKA* pro přibližné porovnávání vzorů pomocí Hammingovy vzdálenosti sestrojeného algoritmem 4, tedy verzí $\overline{p}$, se v rámci této práce nebudeme zabývat.

4.2.4 Přibližné porovnávání vzorů pomocí Levenshteinovy vzdálenosti

U přibližného porovnávání vzorů pomocí Levenshteinovy vzdálenosti jsou vektory R_i^l, $0 \leq l \leq k$, $0 \leq i \leq n$, počítány následovně [1]:

$$\begin{aligned} r_{j,0}^l &\leftarrow 0, && 0 < j \leq l,\ 0 < l \leq k \\ r_{j,0}^l &\leftarrow 1, && l < j \leq m,\ 0 \leq l \leq k \\ R_i^0 &\leftarrow \mathbf{shl}(R_{i-1}^0) \text{ OR } D[t_i], && 0 < i \leq n \\ R_i^l &\leftarrow (\mathbf{shl}(R_{i-1}^l) \text{ OR } D[t_i]) \text{ AND} \\ \mathbf{shl}(R_{i-1}^{l-1} &\text{ AND } R_i^{l-1}) \text{ AND } (R_{i-1}^{l-1} \text{ OR } V), && 0 < i \leq n,\ 0 < l \leq k \end{aligned} \tag{4}$$

Formule 4 používá pomocný vektor V. Tento vektor slouží k tomu, aby nevznikl přechod pozměňující operace *vložení* vedoucí do přijímajících stavů, který je nežádoucí. Vektor V je implementován následovně [1]:

$$V = \begin{bmatrix} v_1 \\ v_2 \\ . \\ . \\ . \\ v_m \end{bmatrix}, \text{ kde } v_m = 1 \text{ a } v_j = 0,\ \forall j,\ 1 \leq j < m \tag{5}$$

Term $\mathbf{shl}(R_{i-1}^l)$ OR $D[t_i]$ ve formuli 4 představuje shodu, term $\mathbf{shl}(R_{i-1}^{l-1})$ představuje pozměňující operaci *substituce*, term R_i^{l-1} představuje pozměňující operaci *odstranění*, která probíhá tak, že pozice i v textu T není inkrementována, pozice ve vzoru P je inkrementována a vzdálenost l je inkrementována. Term R_{i-1}^{l-1} představuje pozměňující operaci *vložení*, která probíhá tak, že pozice i v textu T je inkrementována, pozice ve vzoru P není inkrementovaná a vzdálenost l je inkrementována. Term OR V zajišťuje, že nevznikne přechod pozměňující operace *vložení* vedoucí do přijímajících stavů. Vzor P je nalezen s nanejvýš k rozdíly na pozici i, pokud $r_{m,i}^k = 0$, $0 < i \leq n$. Nejvyšší počet rozdílů nalezeného vzoru je l, kde l je nejmenší počet takový, že $r_{m,i}^l = 0$.

Příklad matic R^l pro přibližné porovnávání vzorů pomocí Levenshteinovy vzdálenosti pro vzor $P = aab$ s chybou $k = 2$ a text $T = aabbab$ je znázorněn v tabulce 4.

Tento Shift-Or algoritmus, popsaný formulí 4, simuluje běh *NKA* pro přibližné porovnávání vzorů pomocí Levenshteinovy vzdálenosti sestrojeného algoritmem 7. Jedná se tedy o verzi Σ.[1] Shift-Or algoritmem pro simulaci běhu *NKA* pro přibližné porovnávání vzorů pomocí Levenshteinovy vzdálenosti sestrojeného algoritmem 6, tedy verzí $\overline{p}$, se v rámci této práce nebudeme zabývat.

Tabulka 4: Matice R^l pro přibližné porovnávání vzorů pomocí Levenshteinovy vzdálenosti pro vzor $P = aab$ s chybou $k = 2$ a text $T = aabbab$

R^0	-	a	a	b	b	a	b
a	1	**0**	**0**	1	1	**0**	1
a	1	1	**0**	1	1	1	1
b	1	1	1	**0**	1	1	1
R^1	-	a	a	b	b	a	b
a	**0**	**0**	**0**	**0**	**0**	**0**	**0**
a	1	**0**	**0**	**0**	1	**0**	**0**
b	1	1	**0**	**0**	**0**	1	**0**
R^2	-	a	a	b	b	a	b
a	**0**	**0**	**0**	**0**	**0**	**0**	**0**
a	**0**	**0**	**0**	**0**	**0**	**0**	**0**
b	1	**0**	**0**	**0**	**0**	**0**	**0**

5 Implementace knihovny konečných automatů

Deterministické a nedeterministické konečné automaty jsou základem této práce. Proto začneme s implementací knihovny konečných automatů. Knihovna bude, stejně jako ostatní knihovny, implementována v jazyce C#, což je jazyk spadající pod framework .NET. Všechny použité kolekce patří do jmenného prostoru System.Collections.Generic. *DKA* a *NKA* sdílí mnoho společným prvků, kterými jsou:

- neprázdná konečná množina stavů Q,
- abeceda Σ,
- počáteční stav q_0 a
- množina přijímajících stavů F.

Z tohoto důvodu vytvoříme abstraktní třídu konečných automatů, která obsahuje tyto prvky. Jednotlivé stavy budou označeny číselně, proto volíme datový typ int. Pro symboly volíme datový typ char. Množinu stavů Q implementujeme pomocí kolekce SortedSet[3], která bude automaticky stavy řadit. Také dokáže rychle zjistit, zda se stav v kolekci nachází či nikoli. Abecedu Σ implementujeme také pomocí kolekce SortedSet. Pro počáteční stav q_0 volíme datový typ int. Množinu přijímajících stavů F implementujeme také pomocí kolekce SortedSet.

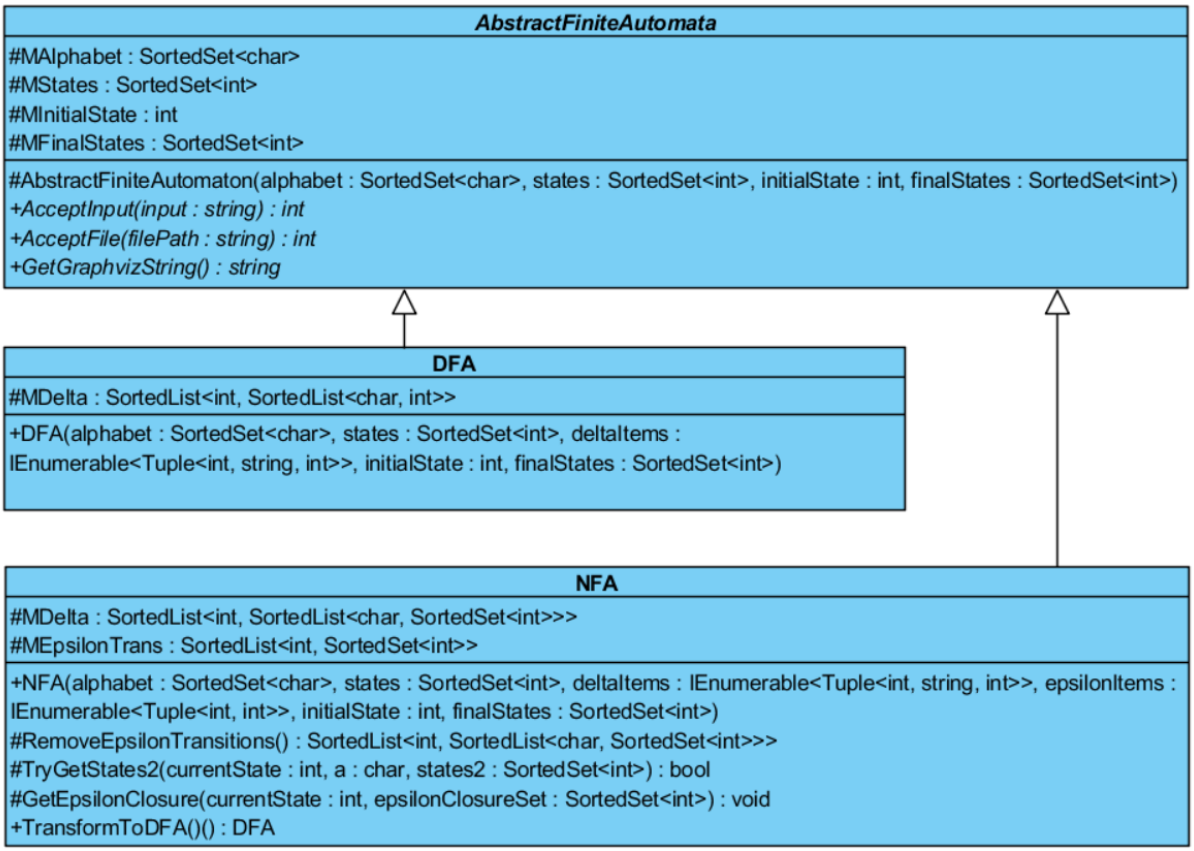
Prvky jsou do konečných automatů ukládány pomocí konstruktoru. Hlavním rozdílem mezi *DKA* a *NKA* je přechodová funkce δ, kterou *DKA* a *NKA* implementují samostatně. Abstraktní třída deklaruje metody pro přijímání vstupního textu, přijímání celého souboru a metodu pro získání DOT řetězce pro generátor grafů Graphviz. Knihovna konečných automatů je znázorněna pomocí třídního diagramu na obrázku 6. Nyní se podíváme na implementaci jednotlivých konečných automatů.

5.1 Deterministické konečné automaty

Determinismus *DKA* zajišťuje, že pro každý stav *DKA* platí, že existuje právě jeden přechod do jiného stavu *DKA* pro každý symbol abecedy Σ. Proto musíme implementovat přechodovou funkci tak, abychom mohli pro každý stav jednoduše určit, do jakého stavu se dostaneme daným symbolem abecedy Σ. Jako vhodné řešení se jeví použití řazených seznamů SortedList[4] (viz výpis 1). Takto implementovaná přechodová funkce umožňuje snadno a rychle získat stav, do kterého se přes daný symbol abecedy Σ dostaneme z aktivního stavu. Kolekce SortedList totiž obsahuje metodu TryGetValue, která dokáže rychle zjistit, zda se daný klíč v kolekci nalézá či nikoli a případně, pokud je klíč nalezen, vrátit s ním asociovanou hodnotu.

```
SortedList<int, SortedList<char, int>> MDelta;
```

Výpis 1: Implementace přechodové funkce *DKA*



Obrázek 6: Třídní diagram knihovny konečných automatů

5.1.1 Metoda pro přijímání vstupních textů (AcceptInput)

Tato metoda prochází vstupní text od začátku do konce. Na začátku metoda nastaví aktivní stav na počáteční stav *DKA*. Pro každý symbol vstupního textu pak metoda zjišťuje stav, do kterého se daným symbolem dostaneme. Metoda aktualizuje aktivní stav na právě nalezený stav. Pokud je nalezený stav obsažen v množině přijímajících stavů MFinalStates, inkrementuje metoda počet shod. Po přečtení vstupního textu metoda vrací počet nalezených shod (viz výpis 2).

```
public override int AcceptInput(string input)
{
   int matches = 0;
   int currentState = MInitialState;
   foreach (var ch in input)
   {
      SortedList<char, int> destStates;
      if (MDelta.TryGetValue(currentState, out destStates))
      {
         int state2;
         if (destStates.TryGetValue(ch, out state2))
         {
            if (MFinalStates.Contains(state2))
            {
               matches++;
            }
            currentState = state2;
         }
      }
   }
   return matches;
}
```

Výpis 2: Implementace metody pro přijímání vstupních textů *DKA*

5.1.2 Metoda pro přijímání souborů (AcceptFile)

K důkladnému otestování tohoto přístupu k vyhledávání v textu budeme muset provést experimenty i na extrémně dlouhých textech. Tyto texty je lepší načítat rovnou ze souboru pomocí bufferu, který nám šetří operační paměť počítače. K tomuto slouží tato metoda, která je upravenou verzí metody pro přijímání vstupních textů. Namísto vstupního textu však tato metoda jako parametr přijímá cestu k souboru. Vstupní text je načítán do bloku pevné délky. Délka bloku

byla zvolena 1024 znaků, čili 2kB. Tyto bloky pak prochází stejným způsobem jako metoda pro přijímání vstupních textů.

5.2 Nedeterministické konečné automaty

Nedeterminismus *NKA* zajišťuje, že se z jednoho stavu *NKA* můžeme jedním symbolem dostat rovnou do několika stavů. Také může nastat situace, kdy pro daný symbol neexistuje přechod. V tom případě se nedostaneme do žádného stavu. Proto musíme implementovat přechodovou funkci tak, abychom mohli pro každý stav jednoduše určit, do jakých stavů se dostaneme daným symbolem abecedy Σ. Stejně jako u *DKA*, se jako vhodné řešení jeví použití řazených seznamů SortedList. Pro množinu stavů však použijeme kolekci SortedSet (viz výpis 3)

```
SortedList<int, SortedList<char, SortedSet<int>>> MDelta;
```

Výpis 3: Implementace přechodové funkce *NKA*

NKA však krom běžných přechodů obsahuje také ϵ přechody, které nejsou označeny žádným symbolem. Tyto přechody budeme implementovat pomocí kolekce SortedList. Pro množinu stavů opět použijeme kolekci SortedSet (viz výpis 4)

```
SortedList<int, SortedSet<int>> MEpsilonTrans;
```

Výpis 4: Implementace přechodové funkce *NKA* (ϵ-přechody)

Jak jsme si již uvedli v kapitole 3.2, *NKA* nemohou být pro vyhledávání v textu použity přímo díky jejich nedeterminismu. Možnost převodu *NKA* na ekvivalentní *DKA* jsme si již ukázali, stejně jako jsme si ukázali i možnost základní simulace běhu *NKA* v kapitole 4.1.

5.2.1 Metoda pro odstranění ϵ přechodů (RemoveEpsilonTransitions)

Tato metoda vytvoří pro *NKA* novou přechodovou funkci odstraněním všech ϵ přechodů. Metoda pro každý stav množiny stavů Q vytvoří ϵ-*uzávěr* (viz definice 19). Následně pro každý symbol abecedy Σ zjistí, do kterých stavů se z ϵ-*uzávěru* daného stavu můžeme daným symbolem dostat. Tyto informace jsou poté uloženy do nové přechodové funkce. Po iteraci všemi stavy množiny stavů Q metoda vrací novou přechodovou funkci.

5.2.2 Metoda pro přijímání vstupních textů (AcceptInput)

Tato metoda je založena na algoritmu 8. Na rozdíl od tohoto algoritmu však metoda odstraní všechny ϵ přechody najednou na začátku pomocí metody pro odstranění ϵ přechodů. Nutnost provádět ϵ-*uzávěr* stavů tedy odpadá. Metoda počítá počet nalezených shod. Pokud je některý ze stavů množiny aktivních stavů obsažen v množině přijímajících stavů MFinalStates, inkrementuje metoda počet shod. Další shody na stejné pozici metoda ignoruje. Po přečtení vstupního textu metoda vrací počet nalezených shod (viz výpis 5).

5.2.3 Metoda pro přijímání souborů (AcceptFile)

K důkladnému otestování tohoto přístupu k vyhledávání v textu budeme muset provést experimenty i na extrémně dlouhých textech. Tyto texty je lepší načítat rovnou ze souboru pomocí bufferu, který nám šetří operační paměť počítače. K tomuto slouží tato metoda, která je upravenou verzí metody pro přijímání vstupních textů. Namísto vstupního textu však tato metoda jako parametr přijímá cestu k souboru. Vstupní text je načítán do bloku pevné délky. Délka bloku byla zvolena 1024 znaků, čili 2kB. Tyto bloky pak prochází stejným způsobem jako metoda pro přijímání vstupních textů.

5.2.4 Metoda pro převod NKA na DKA (TransformToDFA)

Tato metoda je založena na algoritmu 2, tedy podmnožinové konstrukce. Na rozdíl od tohoto algoritmu však metoda odstraní všechny ϵ přechody najednou na začátku pomocí metody pro odstranění ϵ přechodů. Nutnost provádět ϵ-*uzávěr* stavů tedy odpadá. K ukládání všech unikátních podmnožin, tedy nové množiny stavů konstruovaného *DKA*, metoda používá kolekci Dictionary, která má jako klíč kolekci EquatableSortedSet. Tato kolekce je implementována v této práci, a umožňuje nám porovnávat (Equals) a hashovat (GetHashCode) kolekci SortedSet na základě jejího obsahu, nikoli reference. Hodnotou klíčů je pak číselné označení těchto podmnožin. Po převodu metoda vrací nově vytvořený *DKA*.

5.3 Serializace konečných automatů

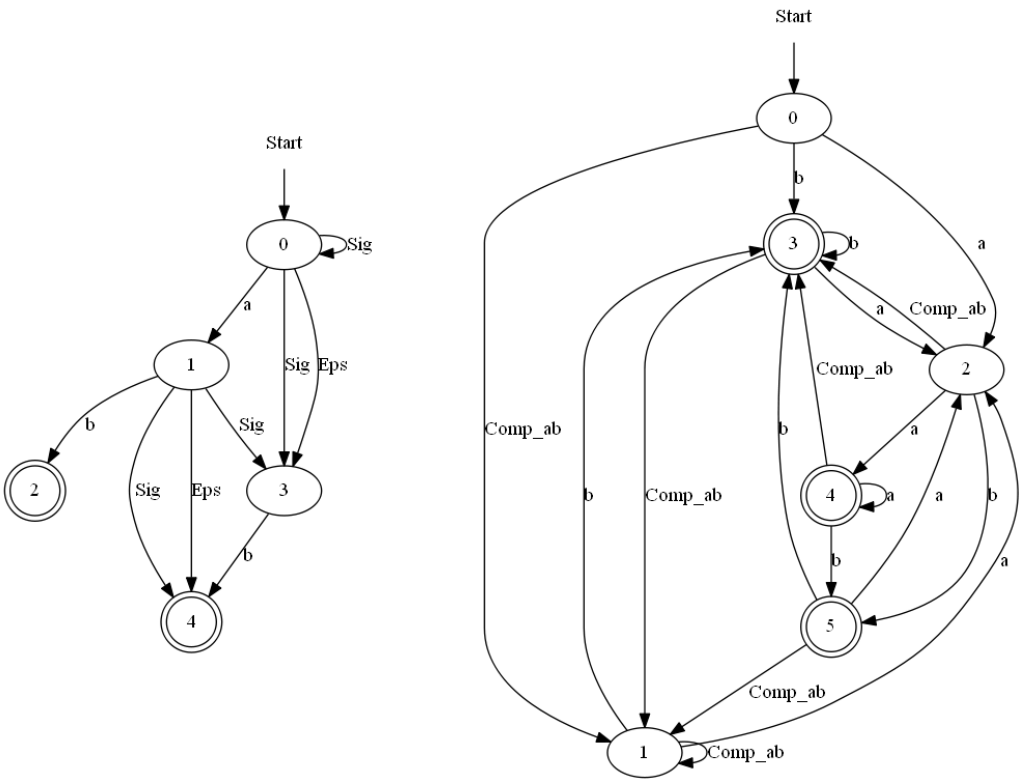
Protože některé automaty mohou být generovány velmi dlouhou dobu, je možné je ukládat pomocí binární serializace. Serializaci provádí třída AutomataDAO pomocí statických metod Save a Load. Metoda Save má dva parametry: konečný automat určený k uložení a cestu k souboru. Metoda Load má pouze jeden parametr a to cestu k uloženému souboru. Oproti metodě Save však vrací načtený konečný automat.

5.4 Graphviz - vizualizace konečných automatů

Pro vizualizaci *DKA* a *NKA* je použit Graphviz. Graphviz je open source grafový vizualizační software. Jako vstup požaduje DOT řetězec, který musíme nejprve vygenerovat metodami GetGraphvizString. Tuto metodu implementují oba konečné automaty. Nevýhodou Graphviz je nemožnost použití zvláštních znaků, jako jsou například písmena řecké abecedy. Proto je znak Σ označen jako Sig a znak ϵ označen jako Eps. Doplněk množiny je označen jako Comp. Ukázka výstupu pro Levenshteinovu vzdálenost je na obrázku 7.

```
public override int AcceptInput(string input)
{
   int matches = 0;
   SortedList<int, SortedList<char, SortedSet<int>>> noEpsDelta =
       RemoveEpsilonTransitions();
   SortedSet<int> notProcessedStateSet = new SortedSet<int> { MInitialState};
   for (int i = 0; i < input.Length && notProcessedStateSet.Count > 0; i++)
   {
      bool isFound = false;
      SortedSet<int> nextStateSet = new SortedSet<int>();
      foreach (var notProcessedState in notProcessedStateSet)
      {
         SortedList<char, SortedSet<int>> destStates;
         if (noEpsDelta.TryGetValue(notProcessedState, out destStates))
         {
            SortedSet<int> states2;
            if (destStates.TryGetValue(input[i], out states2))
            {
               foreach (var state2 in states2)
               {
                  if (MFinalStates.Contains(state2) && !isFound)
                  {
                     matches++;
                     isFound = true;
                  }
                  nextStateSet.Add(state2);
               }
            }
         }
      }
      notProcessedStateSet = nextStateSet;
   }
   return matches;
}
```

Výpis 5: Implementace metody pro přijímání vstupních textů *NKA*



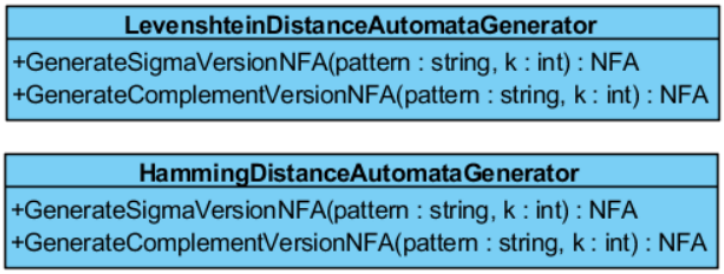
(a) NKA, P = ab, k = 1

(b) DKA, P = ab, k = 1

Obrázek 7: Ukázka výstupu Graphviz pro Levenshteinovu vzdálenost

6 Implementace knihovny generátorů NKA

S naimplementovanými konečnými automaty můžeme začít implementovat knihovnu generátorů *NKA* pro vyhledávání vzorů pomocí Hammingovy a Levenshteinovy vzdálenosti. Knihovna generátorů *NKA* je znázorněna pomocí třídního diagramu na obrázku 8.



Obrázek 8: Třídní diagram knihovny generátorů NKA

6.1 NKA pro porovnávání vzorů pomocí Hammingovy vzdálenosti

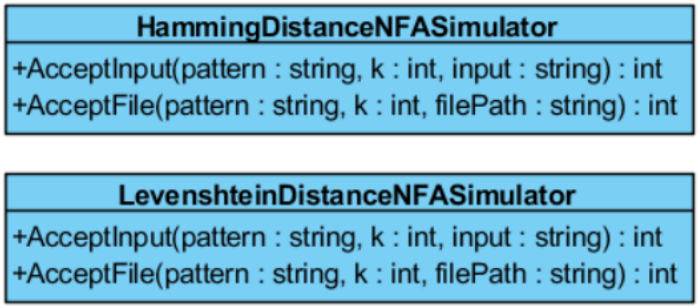
Tato třída obsahuje dvě metody pro generování *NKA* pro přibližné porovnávání vzorů pomocí Hammingovy vzdálenosti. První metodou je metoda GenerateComplementVersionNFA pro generování verze $\overline{p}$. Algoritmus generování je založen na algoritmu 4. Metoda vrací vygenerovaný *NKA*. Druhou metodou je metoda GenerateSigmaVersionNFA pro generování verze Σ. Algoritmus generování je založen na algoritmu 5. Metoda vrací vygenerovaný *NKA*.

6.2 NKA pro porovnávání vzorů pomocí Levenshteinovy vzdálenosti

Tato třída obsahuje dvě metody pro generování *NKA* pro přibližné porovnávání vzorů pomocí Levenshteinovy vzdálenosti. První metodou je metoda GenerateComplementVersionNFA pro generování verze $\overline{p}$. Algoritmus generování je založen na algoritmu 6. Metoda vrací vygenerovaný *NKA*. Druhou metodou je metoda GenerateSigmaVersionNFA pro generování verze Σ. Algoritmus generování je založen na algoritmu 7. Metoda vrací vygenerovaný *NKA*.

7 Implementace knihovny bitového paralelismu

Nyní se podíváme na implementaci knihovny simulátorů běhu *NKA* pomocí metody bitového paralelismu. Knihovna bitového paralelismu je znázorněna pomocí třídního diagramu na obrázku 9.



Obrázek 9: Třídní diagram knihovny bitového paralelismu

7.1 Simulátor NKA pro porovnávání vzorů pomocí Hammingovy vzdálenosti

Tato třída obsahuje dvě metody pro simulování běhu *NKA* pro přibližné porovnávání vzorů pomocí Hammingovy vzdálenosti. Obě tyto metody simulují *NKA* verze Σ. První metodou je metoda AcceptInput pro přijímání vstupních textů. Druhou metodou je metoda AcceptFile pro přijímání celých souborů. Obě tyto metody jsou založeny na formuli 3. Vektory jsou implementovány pomocí datového typu ulong, jehož velikost 64b, nám umožňuje používat vzory o délce až 64. Maskovací matice D je, z důvodu rychlosti, implementována jako pole. Matice R^l jsou implementovány jako dvourozměrné pole, kde počet řádků je $k+1$ a počet sloupců je $n+1$. Po přečtení vstupního textu vrací obě metody počet nalezených shod.

7.2 Simulátor NKA pro porovnávání vzorů pomocí Levenshteinovy vzdálenosti

Tato třída obsahuje dvě metody pro simulování běhu *NKA* pro přibližné porovnávání vzorů pomocí Hammingovy vzdálenosti. Obě tyto metody simulují *NKA* verze Σ. První metodou je metoda AcceptInput pro přijímání vstupních textů. Druhou metodou je metoda AcceptFile pro přijímání celých souborů. Obě tyto metody jsou založeny na formuli 4. Vektory jsou implementovány pomocí datového typu ulong, jehož velikost 64b, nám umožňuje používat vzory o délce až 64. Maskovací matice D je, z důvodu rychlosti, implementována jako pole. Matice R^l jsou implementovány jako dvourozměrné pole, kde počet řádků je $k+1$ a počet sloupců je $n+1$. Po přečtení vstupního textu vrací obě metody počet nalezených shod.

8 Experimenty s porovnáváním vzorů

Experimenty byly prováděny s konečnými automaty verze Σ.

8.1 Alenka v říši divů

Název souboru: alice29.txt
Znaky (včetně mezer): 152089
Zdroj: `http://corpus.canterbury.ac.nz/resources/cantrbry.zip`
Popis: V tomto experimentu testujeme, zda automat nalezne výskyt dlouhého vzoru, pokud jej budeme, s lineárně stoupajícím počtem chyb k, pozměňovat.

8.1.1 Hammingova vzdálenost, DKA

Závěr: Vzor byl nalezen v pěti testech. V posledním testu, ve kterém jsme snížili délku vzoru, však nalezen nebyl, jelikož Hammingova vzdálenost neumí porovnávat slova různých délek.

Tabulka 5: Alenka v říši divů, Hammingova vzdálenost, DKA

Vzor	|Σ|	k	|Q|	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
Oh, you foolish Alice!	256	0	23	5888	20	19.94	18.27	19.17	403	1
Oh, you foalish Alice!	256	1	88	22528	20	29.59	19.37	21.40	450	1
Oh, you foallsh Alice!	256	2	331	84736	20	24.05	22.71	23.15	487	1
Oh, you foallsh Alice?	256	3	1262	323072	20	27.54	25.02	25.90	544	1
Hh, you foallsh Alice?	256	4	4916	1258496	20	30.73	28.09	29.41	618	1
Hh, yo foallsh Alice?	256	5	21045	5387520	20	41.11	33.12	36.37	764	0

8.1.2 Hammingova vzdálenost, NKA

Závěr: Vzor byl nalezen v pěti testech. V posledním testu, ve kterém jsme snížili délku vzoru, však nalezen nebyl, jelikož Hammingova vzdálenost neumí porovnávat slova různých délek.

Tabulka 6: Alenka v říši divů, Hammingova vzdálenost, NKA

Vzor	|Σ|	k	|Q|	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
Oh, you foolish Alice!	256	0	23	278	20	69,42	57,18	58,90	1238	1
Oh, you foalish Alice!	256	1	45	5931	20	114.01	106.18	108.34	2278	1
Oh, you foallsh Alice!	256	2	66	11327	20	173.48	158.73	162.68	3421	1
Oh, you foallsh Alice?	256	3	86	16466	20	260.43	239.23	244.04	5132	1
Hh, you foallsh Alice?	256	4	105	21348	20	343.15	312.54	325.20	6839	1
Hh, yo foallsh Alice?	256	5	117	24687	20	465.18	429.52	446.30	9386	0

8.1.3 Levenshteinova vzdálenost, DKA

Závěr: Vzor byl nalezen ve všech šesti testech. V testech s chybou 3 a výše byly nalezeny 2 shody na dvou různých pozicích, protože Levenshteinova vzdálenost provedla také pozměňující operaci *odstranění*. Tato operace odstranila poslední symbol vzoru. V posledním testu, ve kterém jsme snížili délku vzoru, byl nalezen, protože Levenshteinova vzdálenost umí porovnávat slova různých délek.

Tabulka 7: Alenka v říši divů, Levenshteinova vzdálenost, DKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
Oh, you foolish Alice!	256	0	23	5888	20	19.84	18.57	19.09	401	1
Oh, you foalish Alice!	256	1	143	36608	20	21.25	19.94	20.49	431	1
Oh, you foallsh Alice!	256	2	955	244480	20	27.32	24.69	25.78	542	1
Oh, you foallsh Alice?	256	3	5833	1493248	20	29.69	28.75	29.11	612	2
Hh, you foallsh Alice?	256	4	35351	9049856	20	46.03	34.82	39.08	821	2
Hh, yo foallsh Alice?	256	5	178270	45637120	20	48.24	40.81	43.70	919	2

8.1.4 Levenshteinova vzdálenost, NKA

Závěr: Vzor byl nalezen ve všech šesti testech. V testech s chybou 3 a výše byly nalezeny 2 shody na dvou různých pozicích, protože Levenshteinova vzdálenost provedla také pozměňující operaci *odstranění*. Tato operace odstranila poslední symbol vzoru. V posledním testu, ve kterém jsme snížili délku vzoru, byl nalezen, protože Levenshteinova vzdálenost umí porovnávat slova různých délek.

Tabulka 8: Alenka v říši divů, Levenshteinova vzdálenost, NKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
Oh, you foolish Alice!	265	0	23	278	20	74.22	55.01	57.97	1219	1
Oh, you foalish Alice!	256	1	45	11307	20	117.99	106.60	109.18	2296	1
Oh, you foallsh Alice!	256	2	66	21823	20	199.17	180.82	185.86	3909	1
Oh, you foallsh Alice?	256	3	86	31826	20	311.70	282.32	293.17	6166	2
Hh, you foallsh Alice?	256	4	105	41316	20	609.74	478.49	518.53	10906	2
Hh, yo foallsh Alice?	265	5	117	47727	20	1212.30	718.00	791.90	16656	2

8.2 Pí I

Název souboru: pi.txt
Znaky (včetně mezer): 1000000
Zdroj: `http://corpus.canterbury.ac.nz/resources/misc.zip`
Popis: V tomto experimentu testujeme, zda automat nalezne dva stejně dlouhé vzory délky $m = 8$ s různým počtem chyb v prvním milionu čísel konstanty π. Dvěma vzory jsou náhodné datum a prvních 8 čísel konstanty π.

8.2.1 Hammingova vzdálenost, DKA

Závěr: S chybou $k = 0$ se nám nepodařilo datum nalézt. Prvních 8 čísel konstanty π bylo s chybou $k = 0$ nalezeno právě jednou. S postupným navyšováním chyb se zvyšoval počet nalezených shod obou vzorů.

Tabulka 9: Pí I, Hammingova vzdálenost, DKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
01091994	256	0	9	2304	20	118	114	115	2428	0
01091994	256	1	39	9984	20	127	125	126	2644	2
01091994	256	2	161	41216	20	164	151	156	3275	23
31415926	256	0	9	2304	20	130	119	123	2590	1
31415926	265	1	39	9984	20	151	129	136	2855	1
31415926	256	2	161	41216	20	171	1508	154	3234	23

8.2.2 Hammingova vzdálenost, NKA

Závěr: S chybou $k = 0$ se nám nepodařilo datum nalézt. Prvních 8 čísel konstanty π bylo s chybou $k = 0$ nalezeno právě jednou. S postupným navyšováním chyb se zvyšoval počet nalezených shod obou vzorů.

Tabulka 10: Pí I, Hammingova vzdálenost, NKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
01091994	256	0	9	264	20	430	353	365	7673	0
01091994	256	1	17	2319	20	744	696	706	14838	2
01091994	256	2	24	4117	20	1194	1101	1127	23699	23
31415926	256	0	9	264	20	411	369	379	7979	1
31415926	256	1	17	2319	20	944	732	762	16032	1
31415926	256	2	24	4117	20	1263	1162	1198	25194	23

8.2.3 Levenshteinova vzdálenost, DKA

Závěr: S chybou $k = 0$ se nám nepodařilo datum nalézt. Prvních 8 čísel konstanty π bylo s chybou $k = 0$ nalezeno právě jednou. S postupným navyšováním chyb se zvyšoval počet nalezených shod obou vzorů. Rozdíl oproti Hammingově vzdálenosti nastal až u testu s chybou 3, kde Levenshteinova vzdálenost našla více shod.

Tabulka 11: Pí I, Levenshteinova vzdálenost, DKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
01091994	256	0	9	2304	20	116	113	114	2396	0
01091994	256	1	45	11520	20	129	123	125	2621	2
01091994	256	2	178	45568	20	246	181	208	4377	74
31415926	256	0	9	2304	20	133	119	122	2557	1
31415926	256	1	45	11520	20	143	134	137	2882	2
31415926	256	2	178	45568	20	183	163	168	3541	93

8.2.4 Levenshteinova vzdálenost, NKA

Závěr: S chybou $k = 0$ se nám nepodařilo datum nalézt. Prvních 8 čísel konstanty π bylo s chybou $k = 0$ nalezeno právě jednou. S postupným navyšováním chyb se zvyšoval počet nalezených shod obou vzorů. Rozdíl oproti Hammingově vzdálenosti nastal až u testu s chybou 3, kde Levenshteinova vzdálenost našla více shod.

Tabulka 12: Pí I, Levenshteinova vzdálenost, NKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
01091994	265	0	9	264	20	376	352	356	7477	0
01091994	256	1	17	4111	20	915	733	778	16367	2
01091994	256	2	24	7445	20	1833	1253	1577	33166	74
31415926	256	0	9	264	20	422	387	397	8347	1
31415926	256	1	17	4111	20	824	754	784	16481	2
31415926	265	2	24	7445	20	1699	1319	1416	29791	93

8.3 Pí II

Název souboru: pi.txt
Znaky (včetně mezer): 1000000
Zdroj: `http://corpus.canterbury.ac.nz/resources/misc.zip`
Popis: V tomto experimentu testujeme, zda má velikosti abecedy Σ, při konstantní chybě k a stejné vzoru, vliv na dobu průchodu textu.

8.3.1 Hammingova vzdálenost, DKA

Závěr: Doba průchodu textem se nezvýšila. Velikost abecedy Σ tedy nemá vliv na rychlost deterministického automatu.

Tabulka 13: Pí II, Hammingova vzdálenost, DKA

Vzor	|Σ|	k	|Q|	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
0123456789	10	2	121	1210	20	144	140	142	2977	1
0123456789	20	2	121	2420	20	143	140	141	2976	1
0123456789	30	2	121	3630	20	162	144	146	3068	1
0123456789	40	2	121	4840	20	175	144	147	3098	1
0123456789	50	2	121	6050	20	148	145	146	3071	1
0123456789	60	2	121	7260	20	141	139	140	2952	1

8.3.2 Hammingova vzdálenost, NKA

Závěr: Doba průchodu textem se nezvýšila. Velikost abecedy Σ tedy nemá vliv na rychlost nedeterministického automatu.

Tabulka 14: Pí II, Hammingova vzdálenost, NKA

Vzor	|Σ|	k	|Q|	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
0123456789	10	2	30	227	20	1252	1084	1116	23467	1
0123456789	20	2	30	427	20	1134	1081	1109	23333	1
0123456789	30	2	30	627	20	1161	1101	1136	23894	1
0123456789	40	2	30	827	20	1217	1123	1183	24888	1
0123456789	50	2	30	1027	20	1250	1107	1180	24813	1
0123456789	60	2	30	1227	20	1153	1103	1113	23409	1

8.3.3 Levenshteinova vzdálenost, DKA

Závěr: Doba průchodu textem se nezvýšila. Velikost abecedy Σ tedy nemá vliv na rychlost deterministického automatu.

Tabulka 15: Pí II, Levenshteinova vzdálenost, DKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
0123456789	10	2	305	3050	20	163	151	153	3209	1
0123456789	20	2	305	6100	20	157	151	152	3202	1
0123456789	30	2	305	9150	20	168	154	157	3299	1
0123456789	40	2	305	12200	20	169	152	157	3306	1
0123456789	50	2	305	15250	20	158	155	156	3287	1
0123456789	60	2	305	18300	20	162	159	160	3372	1

8.3.4 Levenshteinova vzdálenost, NKA

Závěr: Doba průchodu textem se nezvýšila. Velikost abecedy Σ tedy nemá vliv na rychlost nedeterministického automatu.

Tabulka 16: Pí II, Levenshteinova vzdálenost, NKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
0123456789	10	2	30	397	20	1443	1265	1307	27497	1
0123456789	20	2	30	767	20	1348	1249	1289	27107	1
0123456789	30	2	30	1137	20	1388	1258	1327	27903	1
0123456789	40	2	30	1507	20	1391	1284	1310	27562	1
0123456789	50	2	30	1877	20	1306	1269	1282	26954	1
0123456789	60	2	30	2247	20	1408	1273	1319	27750	1

8.4 Bible

Název souboru: bible.txt
Znaky (včetně mezer): 4047392
Zdroj: `http://corpus.canterbury.ac.nz/resources/large.zip`
Popis: V tomto experimentu testujeme, zda má nejvyšší počet chyb k, při konstantní velikosti abecedy Σ a stejném vzoru, vliv na dobu průchodu textu.

8.4.1 Hammingova vzdálenost, DKA

Závěr: Doba průchodu textem se, společně s lineárně stoupajícím počtem chyb, velmi mírně zvyšovala. Jedná se však o zanedbatelnou změnu.

Tabulka 17: Bible, Hammingova vzdálenost, DKA

Vzor	$\lvert\Sigma\rvert$	k	$\lvert Q\rvert$	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
Satan	256	0	6	1536	20	529	475	488	9766	56
Satan	256	1	20	5120	20	645	494	537	10762	57
Satan	256	2	58	14848	20	740	577	630	12612	2651
Satan	256	3	123	31488	20	730	633	654	13749	116236
Satan	265	4	179	45824	20	657	635	639	13442	898178
Satan	256	5	218	55808	20	730	647	659	13866	4047388

8.4.2 Hammingova vzdálenost, NKA

Závěr: Doba průchodu textem se, společně s lineárně stoupajícím počtem chyb, lineárně zvyšovala.

Tabulka 18: Bible, Hammingova vzdálenost, NKA

Vzor	$\lvert\Sigma\rvert$	k	$\lvert Q\rvert$	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
Satan	256	0	6	261	20	1714	1419	1483	29699	56
Satan	256	1	11	1545	20	3284	2655	2971	59509	57
Satan	256	2	15	2572	20	5029	4137	4307	86283	2651
Satan	256	3	18	3342	20	7434	5968	6578	138360	116236
Satan	256	4	20	3855	20	10701	7618	7970	167628	898178
Satan	256	5	21	4111	20	10377	9128	9384	197363	4047388

8.4.3 Levenshteinova vzdálenost, DKA

Závěr: Doba průchodu textem se, společně s lineárně stoupajícím počtem chyb, velmi mírně zvyšovala. Jedná se však o zanedbatelnou změnu. Oproti Hammingově vzdálenosti nalezla Levenshteinova vzdálenost v posledním testu 4 další shody. Tyto shody byly nalezeny na konci prohledávaného textu a byly nalezeny, protože Levenshteinova vzdálenost umí porovnávat slova různých velikostí.

Tabulka 19: Bible, Levenshteinova vzdálenost, DKA

Vzor	$\|\Sigma\|$	**k**	$\|Q\|$	**Přechodů**	**Iterací**	**Max[ms]**	**Min[ms]**	**Avg[ms]**	**Sum[ms]**	**Shod**
Satan	256	0	6	1536	20	654	481	540	10812	56
Satan	256	1	24	6144	20	598	543	554	11097	135
Satan	256	2	53	13568	20	622	589	597	11955	3665
Satan	256	3	69	17664	20	770	623	675	14192	205801
Satan	256	4	73	18688	20	1050	621	708	14897	1857052
Satan	256	5	73	18688	20	697	605	642	13504	4047392

8.4.4 Levenshteinova vzdálenost, NKA

Závěr: Doba průchodu textem se, společně s lineárně stoupajícím počtem chyb, lineárně zvyšovala. Oproti Hammingově vzdálenosti nalezla Levenshteinova vzdálenost v posledním testu 4 další shody. Tyto shody byly nalezeny na konci prohledávaného textu a byly nalezeny, protože Levenshteinova vzdálenost umí porovnávat slova různých velikostí.

Tabulka 20: Bible, Levenshteinova vzdálenost, NKA

Vzor	$\|\Sigma\|$	**k**	$\|Q\|$	**Přechodů**	**Iterací**	**Max[ms]**	**Min[ms]**	**Avg[ms]**	**Sum[ms]**	**Shod**
satan	265	0	6	261	20	1926	1449	1650	33046	56
satan	256	1	11	2569	20	3867	2692	2966	59417	135
satan	256	2	15	4364	20	4707	4568	4610	92336	3665
satan	256	3	18	5646	20	9926	7455	7898	166124	205801
satan	256	4	20	6415	20	13898	10946	11974	251850	1857052
satan	265	5	21	6671	20	15506	13714	14086	296276	4047392

8.5 E.coli

Název souboru: E.coli
Znaky (včetně mezer): 4638690
Zdroj: `http://corpus.canterbury.ac.nz/resources/large.zip`
Popis: V tomto experimentu testujeme, zda má délka vzoru, při konstantní chybě k a velikosti abecedy Σ, vliv na dobu průchodu textu.

8.5.1 Hammingova vzdálenost, DKA

Závěr: Doba průchodu textem se, společně s lineárně stoupající délkou vzoru, mírně zvyšovala. Jedná se však o zanedbatelnou změnu.

Tabulka 21: E.coli, Hammingova vzdálenost, DKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
agct	256	1	14	3584	20	736	627	678	14262	239622
agcttt	256	1	22	5632	20	763	664	708	14881	24440
agcttttc	256	1	31	7936	20	776	657	703	14780	2486
agcttttcat	256	1	41	10496	20	814	637	715	15040	179
agcttttcattc	265	1	51	13056	20	826	632	741	15593	9
agcttttcattctg	256	1	61	15616	20	811	664	732	15389	1

8.5.2 Hammingova vzdálenost, NKA

Závěr: Doba průchodu textem se, společně s lineárně stoupající délkou vzoru, mírně zvyšovala.

Tabulka 22: E.coli, Hammingova vzdálenost, NKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
agct	256	1	9	1287	20	5644	5039	5270	110850	239622
agcttt	256	1	13	1803	20	6856	4973	5304	111559	24440
agcttttc	256	1	17	2319	20	6028	5220	5462	114881	2486
agcttttcat	256	1	21	2835	20	8301	5113	5547	116664	179
agcttttcattc	256	1	25	3351	20	5614	5146	5380	113163	9
agcttttcattctg	256	1	29	3867	20	6832	5188	5510	115882	1

8.5.3 Levenshteinova vzdálenost, DKA

Závěr: Doba průchodu textem se, společně s lineárně stoupající délkou vzoru, mírně zvyšovala. Jedná se však o zanedbatelnou změnu.

Tabulka 23: E.coli, Levenshteinova vzdálenost, DKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
agct	256	1	16	4096	20	779	595	690	14509	423656
agcttt	256	1	27	6912	20	844	674	736	15487	39605
agcttttc	256	1	40	10240	20	856	694	756	15895	4055
agcttttcat	256	1	55	14080	20	861	668	738	15513	324
agcttttcattc	256	1	72	18432	20	888	713	793	16670	23
agcttttcattctg	256	1	85	21760	20	886	655	772	16231	2

8.5.4 Levenshteinova vzdálenost, NKA

Závěr: Doba průchodu textem se, společně s lineárně stoupající délkou vzoru, mírně zvyšovala.

Tabulka 24: E.coli, Levenshteinova vzdálenost, NKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
agct	265	1	9	2055	20	8640	5262	5643	118679	423656
agcttt	256	1	13	3083	20	7880	5456	6386	134321	39605
agcttttc	256	1	17	4111	20	6389	5607	5840	122837	4055
agcttttcat	256	1	21	5139	20	6168	5622	5829	122609	324
agcttttcattc	256	1	25	6167	20	6524	5613	6110	128503	23
agcttttcattctg	265	1	29	7195	20	7687	5648	6023	126675	2

8.6 LA Times

Název souboru: latimes.txt
Znaky (včetně mezer): 498360166
Popis: V tomto experimentu měříme dobu průchodu velmi dlouhým textem s různými vzory a chybami k. Vzhledem k tomu, že očekávané dlouhou dobu průchodu textem, provedeme pouze 2 iterace textem u každého testu.

8.6.1 Hammingova vzdálenost, DKA

Závěr: Doba průchodu textem je, dle očekávání, dlouhá. Vzhledem k době průchodu *NKA* se jedná o několikanásobné zlepšení.

Tabulka 25: LA Times, Hammingova vzdálenost, DKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[s]	Min[s]	Avg[s]	Sum[s]	Shod
<DOC>	256	0	6	1536	2	67.6	66.5	67.1	134.3	131896
Chernobyl	256	0	10	2560	2	59.5	59.4	59.4	119.1	218
<HEADLINE>	256	0	11	2816	2	68.3	66.6	67.5	135.1	127561
George Bush	256	1	44	11264	2	75.1	69.8	72.4	145.1	983
Bill Clinton	265	1	50	12800	2	71.0	67.5	69.3	138.8	16
Watergate scandal	256	1	76	19456	2	74.6	73.9	74.3	148.7	43

8.6.2 Hammingova vzdálenost, NKA

Závěr: Doba průchodu textem je, dle očekávání, velmi dlouhá. Důvodem je hlavně nedeterminismu *NKA*. Základní metoda simulace *NKA* se zde projevila jako nevhodná pro velmi dlouhé texty.

Tabulka 26: LA Times, Hammingova vzdálenost, NKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[s]	Min[s]	Avg[s]	Sum[s]	Shod
<DOC>	256	0	6	261	2	187.5	187.4	187.5	375.5	131896
Chernobyl	256	0	10	265	2	173.9	168.7	171.3	343.0	218
<HEADLINE>	256	0	11	266	2	195.1	191.1	193.1	386.8	127561
George Bush	256	1	23	3093	2	363.0	336.3	349.7	700.4	983
Bill Clinton	256	1	25	3351	2	351.4	344.9	348.1	697.4	16
Watergate scandal	256	1	35	4641	2	394.0	338.0	366.0	733.1	43

8.6.3 Levenshteinova vzdálenost, DKA

Závěr: Doba průchodu textem je, dle očekávání, dlouhá. Vzhledem k době průchodu *NKA* se jedná o několikanásobné zlepšení.

Tabulka 27: LA Times, Levenshteinova vzdálenost, DKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[s]	Min[s]	Avg[s]	Sum[s]	Shod
<DOC>	256	0	6	1536	2	62.6	61.9	62.3	124.7	131896
Chernobyl	256	0	10	2560	2	61.0	60.9	61.0	122.1	218
<HEADLINE>	256	0	11	2816	2	66.3	64.8	65.5	131.3	127561
George Bush	256	1	65	16640	2	68.1	66.0	67.0	134.3	2020
Bill Clinton	256	1	72	18432	2	64.0	63.8	63.9	128.0	32
Watergate scandal	256	1	111	28416	2	75.9	71.5	73.7	147.6	94

8.6.4 Levenshteinova vzdálenost, NKA

Závěr: Doba průchodu textem je, dle očekávání, velmi dlouhá. Důvodem je hlavně nedeterminismu *NKA*. Základní metoda simulace *NKA* se zde projevila jako nevhodná pro velmi dlouhé texty.

Tabulka 28: LA Times, Levenshteinova vzdálenost, NKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[s]	Min[s]	Avg[s]	Sum[s]	Shod
<DOC>	265	0	6	261	2	183.4	179.6	181.5	363.6	131896
Chernobyl	256	0	10	265	2	171.4	171.0	171.3	343.0	218
<HEADLINE>	256	0	11	266	2	192.9	184.3	188.6	377.7	127561
George Bush	256	1	23	5653	2	323.5	322.9	323.2	647.4	2020
Bill Clinton	256	1	25	6167	2	315.2	313.5	314.4	629.7	32
Watergate scandal	265	1	35	8737	2	339.0	326.2	332.6	666.2	94

8.7 Kniha světových faktů 1992

Název souboru: world192.txt
Znaky (včetně mezer): 2473400
Zdroj: `http://corpus.canterbury.ac.nz/resources/large.zip`
Popis: V tomto experimentu měříme dobu průchodu středně dlouhým textem pro různé vzory s chybou $k = 0$.

8.7.1 Hammingova vzdálenost, DKA

Závěr: Doba průchodu textem byla velmi rychlá pro všechny vzory různých délek. Z naměřených hodnot je patrné, že délka vzoru neměla vliv na dobu průchodu textem. Z předešlých experimentů je zřejmé, že zvyšování nejvyššího počtu chyb k nemá u *DKA* vliv na dobu průchodu textem.

Tabulka 29: Kniha světových faktů 1992, Hammingova vzdálenost, DKA

Vzor	|Σ|	k	|Q|	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
Czech	256	0	6	1536	20	328.72	311.96	318.11	6690	72
republic	256	0	9	2304	20	320.47	310.70	313.61	6596	225
monarchy	256	0	9	2304	20	334.10	318.06	322.64	6786	30
Communist state	256	0	16	4096	20	356.13	320.80	332.23	6987	6
emerging democracy	265	0	19	4864	20	352.43	321.33	330.17	6944	4
dependent territory of the UK	256	0	30	7680	20	397.52	319.26	336.54	7078	43

8.7.2 Hammingova vzdálenost, NKA

Závěr: Doba průchodu textem byla rychlá pro všechny vzory různých délek. Z naměřených hodnot je patrné, že délka vzoru neměla vliv na dobu průchodu textem. Z předešlých experimentů je zřejmé, že zvyšování nejvyššího počtu chyb k u *DKA* lineárně navyšuje dobu průchodu textem.

Tabulka 30: Kniha světových faktů 1992, Hammingova vzdálenost, NKA

Vzor	|Σ|	k	|Q|	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
Czech	256	0	6	261	20	1184.53	907.18	1019.22	21437	72
republic	256	0	9	264	20	1204.16	891.65	937.84	19725	225
monarchy	256	0	9	264	20	936.14	880.63	900.78	18945	30
Communist state	256	0	16	271	20	993.98	882.65	919.99	19350	6
emerging democracy	256	0	19	274	20	1098.60	944.52	977.43	20558	4
dependent territory of the UK	256	0	30	285	20	1049.33	897.52	944.02	19855	43

8.7.3 Levenshteinova vzdálenost, DKA

Závěr: Doba průchodu textem byla velmi rychlá pro všechny vzory různých délek. Z naměřených hodnot je patrné, že délka vzoru neměla vliv na dobu průchodu textem. Z předešlých experimentů je zřejmé, že zvyšování nejvyššího počtu chyb k nemá u *DKA* vliv na dobu průchodu textem.

Tabulka 31: Kniha světových faktů 1992, Levenshteinova vzdálenost, DKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
Czech	256	0	6	1536	20	595.08	303.58	337.53	7099	72
republic	256	0	9	2304	20	347.40	313.37	324.03	6815	225
monarchy	256	0	9	2304	20	333.91	306.63	315.64	6638	30
Communist state	256	0	16	4096	20	362.35	319.93	331.74	6977	6
emerging democracy	256	0	19	4864	20	345.07	314.24	325.63	6849	4
dependent territory of the UK	256	0	30	7680	20	406.65	313.29	330.75	6956	43

8.7.4 Levenshteinova vzdálenost, NKA

Závěr: Doba průchodu textem byla rychlá pro všechny vzory různých délek. Z naměřených hodnot je patrné, že délka vzoru neměla vliv na dobu průchodu textem. Z předešlých experimentů je zřejmé, že zvyšování nejvyššího počtu chyb k u *DKA* lineárně navyšuje dobu průchodu textem.

Tabulka 32: Kniha světových faktů 1992, Levenshteinova vzdálenost, NKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
Czech	265	0	6	261	20	1695.39	857.45	1000.36	21040	72
republic	256	0	9	264	20	986.38	930.10	954.33	20072	225
monarchy	256	0	9	264	20	1113.25	862.46	895.13	18827	30
Communist state	256	0	16	271	20	1015.26	872.92	922.30	19398	6
emerging democracy	256	0	19	274	20	1025.07	924.29	951.42	20011	4
dependent territory of the UK	265	0	30	285	20	981.63	886.79	922.97	19412	43

8.8 Archiv Wikipedie

Název souboru: enwiki-20170320-pages-articles.xml
Znaky (včetně mezer): 922164187
Zdroj: `http://dumps.wikimedia.org/enwiki/20170320/enwiki-20170320-pages-articles.xml.bz2`
Popis: V tomto experimentu měříme dobu průchodu extrémně dlouhým textem. Protože má soubor velkou velikost a očekáváme velmi dlouhou dobu průchodu textem, otestujeme pouze konečné automaty pro porovnávání vzorů pomocí Hammingovy vzdálenosti. Z dosavadních experimentů však můžeme předpokládat, že by doba průchodu textu byla podobná i u Levenshteinovy vzdálenosti. Pro porovnání rychlosti si také naměříme rychlost čtení tohoto souboru metodou ReadFile (viz výpis 6)

```
public int ReadFile(string filePath)
{
   int characters = 0;
   using (StreamReader r = new StreamReader(filePath))
   {
      char[] buffer = new char[1024];
      int read;
      while ((read = r.ReadBlock(buffer, 0, buffer.Length)) > 0)
      {
         for (int i = 0; i < read; i++)
         {
            characters++;
         }
      }
   }
}
```

Výpis 6: Metoda ReadFile

8.8.1 Měření rychlosti čtení souboru

Závěr: Naměřili jsme průměrnou dobu čtení souboru pomocí metody ReadFile. Nyní můžeme otestovat dobu průchodu textem konečnými automaty pro porovnávání vzorů pomocí Hammingovy vzdálenosti a porovnat průměrné doby průchodu textu s průměrnou dobou čtení souboru pomocí metody ReadFile.

Tabulka 33: Archiv Wikipedie, Měření rychlosti čtení souboru

Iterací	**Max**[s]	**Min**[s]	**Avg**[s]
6	831	747	788

8.8.2 Hammingova vzdálenost

Závěr: U takto dlouhého textu byla doba průchodu textu velmi vysoká. *DKA*, který byl v předchozích měřeních velmi rychlý, je několikanásobně pomalejší, než čtení pomocí metody ReadFile. Rychlost *NKA* pak, vzhledem k rychlosti *DKA*, není překvapující. Základní metoda běhu *NKA* je pro takto dlouhé texty nevhodná.

Tabulka 34: Archiv Wikipedie, Hammingova vzdálenost

Automat	**Vzor**	$\|\Sigma\|$	**k**	$\|Q\|$	**Přechodů**	**Iterací**	**Max**[s]	**Min**[s]	**Avg**[s]	**Sum**[s]	**Shod**
DKA	species	256	0	8	2048	2	7985	7925	7955	7937	12108183
NKA	species	256	0	8	263	2	22273	21414	21844	43687	12108183

9 Experimenty s generováním automatů

Experimenty byly prováděny s konečnými automaty verze Σ.

9.1 Abeceda

Popis: V tomto experimentu měříme dobu generování konečných automatů pro stejný vzor a stejnou chybu k. V průběhu testování však budeme měnit velikost abecedy Σ.

9.1.1 Hammingova vzdálenost, DKA

Závěr: Doba generování *DKA* se, společně s lineárně zvyšující se velikostí abecedy Σ, zvyšovala lineárně.

Tabulka 35: Abeceda, Hammingova vzdálenost, DKA

Vzor	|Σ|	k	|Q|	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]
0123456789	10	5	3342	33420	20	505.40	484.25	490.30	10312
0123456789	20	5	3342	66840	20	870.15	760.37	805.67	16945
0123456789	30	5	3342	100260	20	1076.29	1036.98	1045.90	21998
0123456789	40	5	3342	133680	20	1451.68	1319.13	1368.94	28792
0123456789	50	5	3342	167100	20	1788.85	1593.46	1663.74	34993
0123456789	100	5	3342	334200	20	3611.28	3119.36	3260.29	68573
0123456789	150	5	3342	501300	20	4971.50	4568.98	4641.89	97632
0123456789	200	5	3342	668400	20	6321.57	6017.59	6108.47	128478

9.1.2 Hammingova vzdálenost, NKA

Závěr: Doba generování *NKA* se, společně s lineárně zvyšující se velikostí abecedy Σ, zvyšovala lineárně. Vzhledem k velmi nízké době generování však můžeme tuto změnu zanedbat.

Tabulka 36: Abeceda, Hammingova vzdálenost, NKA

Vzor	|Σ|	k	|Q|	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]
0123456789	10	5	51	455	20	3.98	0.41	1.07	22
0123456789	20	5	51	865	20	5.21	0.75	2.08	43
0123456789	30	5	51	1275	20	4.18	0.75	1.98	41
0123456789	40	5	51	1685	20	6.01	0.76	2.25	47
0123456789	50	5	51	2095	20	4.63	0.93	2.37	49
0123456789	100	5	51	4145	20	6.48	1.78	3.36	70
0123456789	150	5	51	6195	20	8.46	2.37	4.24	89
0123456789	200	5	51	8245	20	12.28	2.92	5.24	110

9.1.3 Levenshteinova vzdálenost, DKA

Závěr: Doba generování *DKA* se, společně s lineárně zvyšující se velikostí abecedy Σ, zvyšovala lineárně.

Tabulka 37: Abeceda, Levenshteinova vzdálenost, DKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]
0123456789	10	5	4099	40990	20	894.50	796.09	848.41	17844
0123456789	20	5	4099	81980	20	1363.63	1279.82	1301.04	27364
0123456789	30	5	4099	122970	20	1964.13	1786.96	1846.59	38839
0123456789	40	5	4099	163960	20	2477.96	2309.25	2363.78	49717
0123456789	50	5	4099	204950	20	3723.09	2826.78	3029.55	63719
0123456789	100	5	4099	409900	20	5409.84	5337.76	5368.12	112906
0123456789	150	5	4099	614850	20	8331.22	7881.81	7954.22	167299
0123456789	200	5	4099	819800	20	10826.77	10345.42	10506.76	220987

9.1.4 Levenshteinova vzdálenost, NKA

Závěr: Doba generování *NKA* se, společně s lineárně zvyšující se velikostí abecedy Σ, zvyšovala lineárně. Vzhledem k velmi nízké době generování však můžeme tuto změnu zanedbat.

Tabulka 38: Abeceda, Levenshteinova vzdálenost, NKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]
0123456789	10	5	51	805	20	4.57	0.40	1.15	24
0123456789	20	5	51	1565	20	4.38	0.87	2.00	42
0123456789	30	5	51	2325	20	4.77	1.17	2.43	51
0123456789	40	5	51	3085	20	5.24	1.41	2.82	59
0123456789	50	5	51	3845	20	7.33	1.58	3.12	65
0123456789	100	5	51	7645	20	10.25	2.75	4.59	96
0123456789	150	5	51	11445	20	12.27	3.49	5.67	119
0123456789	200	5	51	15245	20	17.16	4.81	8.00	168

9.2 Chyba

Popis: V tomto experimentu měříme dobu generování konečných automatů pro stejný vzor a abecedu Σ. V průběhu testování však budeme měnit chybu vzoru k.

9.2.1 Hammingova vzdálenost, DKA

Závěr: Doba generování *DKA* se, společně se zvyšující se chybou k, zvyšovala exponenciálně. Důvodem pomalé rychlosti u větších chyb je velmi vysoký počet stavů a přechodů.

Tabulka 39: Chyba, Hammingova vzdálenost, DKA

Vzor	|Σ|	k	|Q|	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]
Rosalinda	256	0	10	2560	20	8.62	4.18	5.78	121
Rosalinda	256	1	34	8704	20	31.18	24.70	27.01	568
Rosalinda	256	2	110	28160	20	111.73	99.72	105.30	2214
Rosalinda	256	3	340	87040	20	441.72	401.36	418.05	8792
Rosalinda	265	4	1004	257024	20	1702.02	1641.72	1663.35	34985
Rosalinda	256	5	2796	715776	20	6620.94	6489.99	6532.03	137387
Rosalinda	256	6	6970	1784320	20	24255.77	23745.48	23951.69	503772
Rosalinda	256	7	13988	3580928	20	67511.07	66314.99	67022.65	14

9.2.2 Hammingova vzdálenost, NKA

Závěr: Doba generování *NKA* se, společně se zvyšující se chybou k, zvyšovala lineárně. Vzhledem k velmi nízké době generování však můžeme tuto změnu zanedbat.

Tabulka 40: Chyba, Hammingova vzdálenost, NKA

Vzor	|Σ|	k	|Q|	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]
Rosalinda	256	0	10	265	20	3.94	0.26	1.05	22
Rosalinda	256	1	19	2577	20	3.67	1.11	1.88	39
Rosalinda	256	2	27	4632	20	7.37	1.66	2.72	57
Rosalinda	256	3	34	6430	20	9.46	2.60	3.96	83
Rosalinda	256	4	40	7971	20	11.67	2.71	4.53	95
Rosalinda	256	5	45	9255	20	11.05	3.01	4.49	94
Rosalinda	256	6	49	10282	20	11.76	3.33	5.08	106
Rosalinda	265	7	52	11052	20	14.69	3.56	5.64	118

9.2.3 Levenshteinova vzdálenost, DKA

Závěr: Doba generování *DKA* se, společně se zvyšující se chybou k, zvyšovala exponenciálně. Důvodem pomalé rychlosti u větších chyb je velmi vysoký počet stavů a přechodů.

Tabulka 41: Chyba, Levenshteinova vzdálenost, DKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]
Rosalinda	256	0	10	2560	20	9.92	4.21	5.90	124
Rosalinda	256	1	51	13056	20	45.51	37.19	39.65	834
Rosalinda	256	2	247	63232	20	300.57	263.48	277.92	5845
Rosalinda	256	3	797	204032	20	1405.56	1310.09	1345.64	28302
Rosalinda	256	4	1455	372480	20	3602.38	3416.06	3470.64	72997
Rosalinda	256	5	1904	487424	20	6060.76	5839.15	5944.01	125019
Rosalinda	256	6	2111	540416	20	8113.06	8047.75	8075.18	169844
Rosalinda	256	7	2171	555776	20	10179.60	9805.65	9851.37	207202

9.2.4 Levenshteinova vzdálenost, NKA

Závěr: Doba generování *NKA* se, společně se zvyšující se chybou k, zvyšovala lineárně. Vzhledem k velmi nízké době generování však můžeme tuto změnu zanedbat.

Tabulka 42: Chyba, Levenshteinova vzdálenost, NKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]
Rosalinda	265	0	10	265	20	4.33	0.16	0.99	20
Rosalinda	256	1	19	4625	20	7.54	1.51	2.48	52
Rosalinda	256	2	27	8472	20	10.83	2.95	4.60	96
Rosalinda	256	3	34	11806	20	11.07	3.50	5.06	106
Rosalinda	256	4	40	14627	20	15.99	4.21	6.74	141
Rosalinda	256	5	45	16935	20	25.21	4.82	8.34	175
Rosalinda	256	6	49	18730	20	24.05	5.31	9.26	194
Rosalinda	256	7	52	20012	20	25.78	5.73	9.94	209

9.3 Vzor

Popis: V tomto experimentu měříme dobu generování konečných automatů pro stejnou chybu k a abecedu Σ. V průběhu testování však budeme měnit délku vzoru.

9.3.1 Hammingova vzdálenost, DKA

Závěr: Doba generování *DKA* se, společně s lineárně zvyšující se délkou vzoru, zvyšovala exponenciálně.

Tabulka 43: Vzor, Hammingova vzdálenost, DKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]
xxxx	256	2	25	6400	20	31.31	25.30	28.71	603
xxxxxx	256	2	63	16128	20	86.41	79.17	82.35	1732
xxxxxxxx	256	2	129	33024	20	252.85	208.86	216.37	4550
xxxxxxxxxx	256	2	231	59136	20	514.81	442.73	459.00	9654
xxxxxxxxxxxx	265	2	377	96512	20	872.69	830.49	847.06	17816
xxxxxxxxxxxxxx	256	2	575	147200	20	1471.11	1432.61	1448.23	30460
xxxxxxxxxxxxxxxx	256	2	833	213248	20	2502.76	2339.01	2368.74	49821
xxxxxxxxxxxxxxxxxx	256	2	1159	296704	20	4342.15	3690.33	3859.32	81172

9.3.2 Hammingova vzdálenost, NKA

Závěr: Doba generování *NKA* se, společně s lineárně zvyšující se délkou vzoru, zvyšovala lineárně. Vzhledem k velmi nízké době generování však můžeme tuto změnu zanedbat.

Tabulka 44: Vzor, Hammingova vzdálenost, NKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]
xxxx	256	2	12	2057	20	3.61	0.89	1.81	38
xxxxxx	256	2	18	3087	20	4.58	1.14	2.11	44
xxxxxxxx	256	2	24	4117	20	6.89	1.63	2.47	52
xxxxxxxxxx	256	2	30	5147	20	6.87	1.81	2.65	55
xxxxxxxxxxxx	256	2	36	6177	20	8.15	2.20	3.30	69
xxxxxxxxxxxxxx	256	2	42	7207	20	8.94	2.63	3.56	74
xxxxxxxxxxxxxxxx	256	2	48	8237	20	9.51	2.72	4.14	87
xxxxxxxxxxxxxxxxxx	265	2	54	9267	20	9.60	2.95	4.78	100

9.3.3 Levenshteinova vzdálenost, DKA

Závěr: Doba generování *DKA* se, společně s lineárně zvyšující se délkou vzoru, zvyšovala exponenciálně.

Tabulka 45: Vzor, Levenshteinova vzdálenost, DKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]
xxxx	256	2	16	4096	20	24.10	18.78	21.46	451
xxxxxx	256	2	43	11008	20	70.88	63.52	66.92	1407
xxxxxxxx	256	2	94	24064	20	182.79	165.65	173.64	3652
xxxxxxxxxx	256	2	177	45312	20	407.98	372.67	381.64	8026
xxxxxxxxxxxx	256	2	300	76800	20	763.05	736.39	748.25	15737
xxxxxxxxxxxxxx	256	2	471	120576	20	1381.34	1282.02	1310.20	27557
xxxxxxxxxxxxxxxx	256	2	698	178688	20	2339.59	2167.84	2230.34	46910
xxxxxxxxxxxxxxxxxx	256	2	989	253184	20	3704.46	3457.74	3566.26	75008

9.3.4 Levenshteinova vzdálenost, NKA

Závěr: Doba generování *NKA* se, společně s lineárně zvyšující se délkou vzoru, zvyšovala lineárně. Vzhledem k velmi nízké době generování však můžeme tuto změnu zanedbat.

Tabulka 46: Vzor, Levenshteinova vzdálenost, NKA

Vzor	$\|\Sigma\|$	k	$\|Q\|$	Přechodů	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]
xxxx	265	2	12	3337	20	4.14	1.11	2.11	44
xxxxxx	256	2	18	5391	20	6.47	1.92	2.69	56
xxxxxxxx	256	2	24	7445	20	7.84	2.41	3.48	73
xxxxxxxxxx	256	2	30	9499	20	9.86	2.92	4.26	89
xxxxxxxxxxxx	256	2	36	11553	20	10.93	3.30	4.89	102
xxxxxxxxxxxxxx	256	2	42	13607	20	13.53	3.98	6.16	129
xxxxxxxxxxxxxxxx	256	2	48	15661	20	17.06	4.67	7.45	156
xxxxxxxxxxxxxxxxxx	256	2	54	17715	20	26.08	5.30	9.49	199

10 Experimenty s bitovým paralelismem

Experimenty byly prováděny s konečnými automaty verze Σ.

10.1 Pí II

Název souboru: pi.txt
Znaky (včetně mezer): 1000000
Zdroj: `http://corpus.canterbury.ac.nz/resources/misc.zip`
Popis: V tomto experimentu testujeme, zda má velikosti abecedy Σ, při konstantní chybě k a stejné vzoru, vliv na dobu průchodu textu metody simulace běhu *NKA* bitovým paralelismem.

10.1.1 Hammingova vzdálenost, Bitový paralelismus (NKA)

Závěr: Doba průchodu textem se nezvýšila. Velikost abecedy Σ tedy nemá vliv na dobu průchodu textem.

Tabulka 47: Pí II, Hammingova vzdálenost, Bitový paralelismus (NKA)

Vzor	|Σ|	k	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
0123456789	10	2	20	56	54	55	1098	1
0123456789	20	2	20	56	54	55	1098	1
0123456789	30	2	20	60	55	56	1113	1
0123456789	40	2	20	58	55	56	1110	1
0123456789	50	2	20	57	55	56	1116	1
0123456789	60	2	20	58	55	56	1129	1

10.1.2 Levenshteinova vzdálenost, Bitový paralelismus (NKA)

Závěr: Doba průchodu textem se nezvýšila. Velikost abecedy Σ tedy nemá vliv na dobu průchodu textem.

Tabulka 48: Pí II, Levenshteinova vzdálenost, Bitový paralelismus (NKA)

Vzor	|Σ|	k	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
0123456789	10	2	20	72	70	71	1418	1
0123456789	20	2	20	72	70	71	1415	1
0123456789	30	2	20	72	70	71	1421	1
0123456789	40	2	20	73	71	72	1433	1
0123456789	50	2	20	74	71	72	1441	1
0123456789	60	2	20	79	72	74	1472	1

10.2 Bible

Název souboru: bible.txt
Znaky (včetně mezer): 4047392
Zdroj: `http://corpus.canterbury.ac.nz/resources/large.zip`
Popis: V tomto experimentu testujeme, zda má nejvyšší počet chyb k, při konstantní velikosti abecedy Σ a stejném vzoru, vliv na dobu průchodu textu metody simulace běhu *NKA* bitovým paralelismem.

10.2.1 Hammingova vzdálenost, Bitový paralelismus (NKA)

Závěr: Doba průchodu textem se, společně s lineárně stoupajícím počtem chyb, zvyšovala lineárně. Bitový paralelismus byl dvacetinásobně rychlejší, než základní metoda simulace *NKA*. Ve všech případech byl také rychlejší než *DKA*.

Tabulka 49: Bible, Hammingova vzdálenost, Bitový paralelismus (NKA)

Vzor	|Σ|	k	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
Satan	256	0	20	80	77	78	1554	56
Satan	256	1	20	153	150	152	3037	57
Satan	256	2	20	228	224	226	4521	2651
Satan	256	3	20	312	296	299	5985	116236
Satan	265	4	20	397	388	391	7838	898178
Satan	256	5	20	456	447	449	8558	4047388

10.2.2 Levenshteinova vzdálenost, Bitový paralelismus (NKA)

Závěr: Doba průchodu textem se, společně s lineárně stoupajícím počtem chyb, zvyšovala lineárně. Bitový paralelismus byl dvacetinásobně rychlejší, než základní metoda simulace *NKA*. Ve všech případech byl také rychlejší než *DKA*.

Tabulka 50: Bible, Levenshteinova vzdálenost, Bitový paralelismus (NKA)

Vzor	|Σ|	k	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
Satan	256	0	20	79	77	78	1556	56
Satan	256	1	20	185	181	183	3659	135
Satan	256	2	20	305	287	290	5804	3665
Satan	256	3	20	415	394	400	8020	205801
Satan	256	4	20	527	517	521	10442	1857052
Satan	256	5	20	617	608	612	12266	4047392

10.3 E.coli

Název souboru: E.coli
Znaky (včetně mezer): 4638690
Zdroj: `http://corpus.canterbury.ac.nz/resources/large.zip`
Popis: V tomto experimentu testujeme, zda má délka vzoru, při konstantní chybě k a velikosti abecedy Σ, vliv na dobu průchodu textu metody simulace běhu *NKA* bitovým paralelismem.

10.3.1 Hammingova vzdálenost, Bitový paralelismus (NKA)

Závěr: Doba průchodu textem se, s lineárně stoupající délkou vzoru, nezvyšovala. Délka vzoru tedy nemá vliv na dobu průchodu textem.

Tabulka 51: E.coli, Hammingova vzdálenost, Bitový paralelismus (NKA)

Vzor	|Σ|	k	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
agct	256	1	20	193	174	177	3536	239622
agcttt	256	1	20	222	173	180	3602	24440
agcttttc	256	1	20	175	171	173	3462	2486
agcttttcat	256	1	20	191	184	187	3739	179
agcttttcattc	265	1	20	197	180	183	3673	9
agcttttcattctg	256	1	20	214	176	185	3710	1

10.3.2 Levenshteinova vzdálenost, Bitový paralelismus (NKA)

Závěr: Doba průchodu textem se, s lineárně stoupající délkou vzoru, nezvyšovala. Délka vzoru tedy nemá vliv na dobu průchodu textem.

Tabulka 52: E.coli, Levenshteinova vzdálenost, Bitový paralelismus (NKA)

Vzor	|Σ|	k	Iterací	Max[ms]	Min[ms]	Avg[ms]	Sum[ms]	Shod
agct	265	1	20	235	216	219	4382	423656
agcttt	256	1	20	341	220	260	5185	39605
agcttttc	256	1	20	214	208	210	4208	4055
agcttttcat	256	1	20	290	222	246	4923	324
agcttttcattc	256	1	20	240	218	223	4466	23
agcttttcattctg	265	1	20	243	210	217	4352	2

10.4 LA Times

Název souboru: latimes.txt
Znaky (včetně mezer): 498360166
Popis: V tomto experimentu měříme dobu průchodu velmi dlouhým textem s různými vzory a chybami k pomocí metody simulace běhu *NKA* bitovým paralelismem.

10.4.1 Hammingova vzdálenost, Bitový paralelismus (NKA)

Závěr: Doba průchodu textem je, oproti základní metody simulace běhu *NKA*, velmi malá. Bitový paralelismus byl opět dvacetinásobně rychlejší.

Tabulka 53: LA Times, Hammingova vzdálenost, Bitový paralelismus (NKA)

Vzor	$\|\Sigma\|$	k	Iterací	Max[s]	Min[s]	Avg[s]	Sum[s]	Shod
<DOC>	256	0	10	9.7	9.3	9.5	94.8	131896
Chernobyl	256	0	10	9.3	9.2	9.2	92.4	218
<HEADLINE>	256	0	10	9.2	9.2	9.2	92.3	127561
George Bush	256	1	10	18.2	17.9	18.0	180.2	983
Bill Clinton	265	1	10	18.6	17.9	18.0	181.2	16
Watergate scandal	256	1	10	18.0	17.9	18.0	179.7	43

10.4.2 Levenshteinova vzdálenost, Bitový paralelismus (NKA)

Závěr: Doba průchodu textem je, oproti základní metody simulace běhu *NKA*, velmi malá. Bitový paralelismus byl opět dvacetinásobně rychlejší.

Tabulka 54: LA Times, Levenshteinova vzdálenost, Bitový paralelismus (NKA)

Vzor	$\|\Sigma\|$	k	Iterací	Max[s]	Min[s]	Avg[s]	Sum[s]	Shod
<DOC>	265	0	10	9.4	9.3	9.4	93.1	131896
Chernobyl	256	0	10	9.2	9.1	9.2	92.1	218
<HEADLINE>	256	0	10	9.3	9.2	9.2	92.4	127561
George Bush	256	1	10	22.0	21.7	21.8	218.7	2020
Bill Clinton	256	1	10	27.7	22.1	24.6	246.0	32
Watergate scandal	265	1	10	23.0	22.8	22.9	228.6	94

10.5 Archiv Wikipedie

Název souboru: enwiki-20170320-pages-articles.xml
Znaky (včetně mezer): 922164187
Zdroj: `http://dumps.wikimedia.org/enwiki/20170320/enwiki-20170320-pages-articles.xml.bz2`
Popis: V tomto experimentu měříme dobu průchodu extrémně dlouhým textem pomocí metody simulace běhu *NKA* bitovým paralelismem. Pro porovnání rychlosti si také naměříme rychlost čtení tohoto souboru metodou ReadFile (viz výpis 6)

10.5.1 Měření rychlosti čtení souboru

Závěr: Naměřili jsme průměrnou dobu čtení souboru pomocí metody ReadFile. Nyní můžeme otestovat dobu průchodu textem konečnými automaty pro porovnávání vzorů pomocí Hammingovy vzdálenosti a porovnat průměrné doby průchodu textu s průměrnou dobou čtení souboru pomocí metody ReadFile.

Tabulka 55: Archiv Wikipedie, Měření rychlosti čtení souboru

Iterací	Max[s]	Min[s]	Avg[s]
6	831	747	788

10.5.2 Měření doby průchodu metodou bitové paralelismu

Závěr: Doba průchodu textem pro oba algoritmy je, oproti základní metody simulace běhu *NKA*, velmi malá. Bitový paralelismus byl opět dvacetinásobně rychlejší. Doba průchodu byla několikanásobně menší, než u *DKA*. V porovnání s dobou čtení souboru se však jednalo o téměř dvojnásobný nárůst. I přesto tato metoda dosáhla velmi dobrých výsledků.

Tabulka 56: Archiv Wikipedie, Měření doby průchodu metodou bitové paralelismu

Algoritmus	Vzor	$\|\Sigma\|$	k	Iterací	Max[s]	Min[s]	Avg[s]	Sum[s]	Shod
Hamming	species	256	0	2	1290	1310	1300	2600	12108183
Levenshtein	species	256	0	2	1437	1309	1373	2746	12108183
Hamming	species	256	1	2	2353	2415	2384	4768	30511122
Levenshtein	species	256	1	2	2956	3168	3062	6124	32067924

11 Závěr

Cílem této bakalářské práce bylo naimplementovat a otestovat knihovnu konečných automatů a knihovnu generátorů nedeterministických konečných automatu pro vyhledávání v textu pomocí Hammingovy a Levenshteinovy vzdálenosti.

V první části jsme probrali možnosti použití konečných automatů pro vyhledávání v textu pomocí porovnávání vzorů. Ukázali jsme si konstrukci *NKA* pro přibližné porovnávání vzorů pomocí Hammingovy vzdálenosti ($\overline{p}$ verze a Σ verze) a konstrukci *NKA* pro přibližné porovnávání vzorů pomocí Levenshteinovy vzdálenosti ($\overline{p}$ verze a Σ verze). Probrali jsme možnost převodu *NKA* na ekvivalentní *DKA*. Ukázali jsme si také, jak simulovat běh *NKA* pomocí základní metody simulace *NKA* a pomocí metody bitového paralelismu.

Ve druhé části jsme si, dle algoritmů z první části, naimplementovali knihovnu konečných automatů, knihovnu generátorů *NKA* a knihovnu bitového paralelismu. Přidali jsme také možnost ukládání a načítání konečných automatů do souborů a možnost vizualizace konečných automatů pomocí generátoru grafů Graphviz.

Ve třetí části jsme důkladně otestovali implementované knihovny a provedli rozbor výsledků. Provedli jsme experimenty s porovnáváním vzorů pro vzory různých délek, abeced a chyb. Konečné automaty v těchto experimentech jsme testovali na textech krátkých délek (od 150 tisíc znaků) i dlouhých délek (po téměř 1 miliardu znaků). Také jsme provedli experimenty s generováním konečných automatů různých délek, abeced a chyb.

Z provedených experimentů jsme zjistili, že vyhledávání v textu pomocí *DKA*, jehož lineární asymptotická složitost je $\mathcal{O}(n)$, je mnohonásobně rychlejší než pomocí základní simulace běhu *NKA*, jehož doba průchodu textem je závislá hlavně na počtu chyb. Musíme však vzít v potaz dobu generování *DKA*, která je závislá na délce vzoru, velikosti abecedy a hlavně počtu chyb. Simulace běhu *NKA* pomocí bitového paralelismu však dobou průchodu přibližně dvacetinásobně předčila základní simulaci běhu *NKA*. Předčila také dobu průchodu *DKA* a velmi se blížila rychlostí čtení. Doba průchodu textem pomocí metody bitového paralelismu se navíc ukázala jako zcela nezávislá na velikosti abecedy a délce vzoru, a proto se jeví jako ideální řešení pro tento přístup k vyhledávání v textu.

Obecně tedy můžeme říci, že vyhledávání v textu pomocí porovnávání vzorů je ideální pro situace, kdy potřebujeme vyhledat krátký vzor ve velmi dlouhém textu. V praxi by se tedy dal použít kupříkladu k měření genetické vzdálenosti či ke korekci pravopisu.

Literatura

[1] HOLUB, Jan. Simulation of Nondeterministic Finite Automata in Pattern Matching. Disertační práce. České vysoké učení technické v Praze, Fakulta elektrotechnická, Praha, Únor 2000. Dostupné z: http://www.stringology.org/papers/Holub-PhD_thesis.ps.gz

[2] SAWA, Zdeněk a Petr JANČAR. Úvod do teoretické informatiky [online]. 2008. Ostrava: Vysoká škola báňská - Technická univerzita, 2008 [cit. 2017-04-02]. ISBN 9788024814964. Dostupné z: http://www.cs.vsb.cz/sawa/uti/materialy/uti.pdf

[3] SortedSet<T> Class. In: MSDN [online]. [cit. 2017-04-18]. Dostupné z: http://www.msdn.microsoft.com/en-us/library/dd412070(v=vs.110).aspx

[4] SortedList<TKey, TValue> Class. In: MSDN [online]. [cit. 2017-04-18]. Dostupné z: http://www.msdn.microsoft.com/en-us/library/ms132319(v=vs.110).aspx

[5] Dictionary<TKey, TValue> Class. In: MSDN [online]. [cit. 2017-04-18]. Dostupné z: http://www.msdn.microsoft.com/en-us/library/xfhwa508(v=vs.110).aspx

A Struktura přiloženého optického média

Struktura přiloženého optického média:

- \src - projekt s aplikací ve Visual Studiu 2015,
- \bp - PDF soubor s textem bakalářské práce.