FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Implementace automatových algoritmů na hledání rozšířených pokrytí
Student:	Rajmund Hubert Hruška
Vedoucí:	Ing. Ondřej Guth, Ph.D.
Studijní program:	Informatika
Studijní obor:	Teoretická informatika
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Na základě dohody s vedoucím práce nastudujte algoritmy pro vyhledávání přibližných rozšířených pokrytí v textových řetězcích [1] a tyto algoritmy implementujte jako součást knihovny algoritmů FIT [2]. Implementaci v rámci knihovny vhodným způsobem otestujte.

Seznam odborné literatury

[1] Guth, Ondřej. On approximate enhanced covers under Hamming distance. Discrete Applied Mathematics. 2019. ISSN 0166218X. DOI 10.1016/j.dam.2019.01.015
[2] Algorithms Library Toolkit [software]. Dostupné z: https://gitlab.fit.cvut.cz/algorithms-library-toolkit.

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 8. ledna 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalárska práca

Implementácia automatových algoritmov na hľadanie rozšírených pokrytí

Rajmund Hubert Hruška

Katedra teoretické informatiky
Vedúci práce: Ing. Ondřej Guth, Ph.D.

4. júna 2020

Poďakovanie

Ďakujem vedúcemu tejto bakalárskej práce, ktorým je Ing. Ondřej Guth, Ph.D., za vedenie práce a pomoc s problémami, na ktoré som v priebehu narazil. Ďalej ďakujem svojej rodine za podporu pri štúdiu.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov. V súlade s ustanovením § 46 odst. 6 tohoto zákona týmto udeľujem bezvýhradné oprávnenie (licenciu) k užívaniu tejto mojej práce, a to vrátane všetkých počítačových programov ktoré sú jej súčasťou alebo prílohou a tiež všetkej ich dokumentácie (ďalej len „Dielo“), a to všetkým osobám, ktoré si prajú Dielo užívať.

Tieto osoby sú oprávnené Dielo používať akýmkoľvek spôsobom, ktorý nezhoršuje hodnotu Diela, a za akýmkoľvek účelom (vrátane komerčného využitia). Toto oprávnenie je časovo, územne a množstevne neobmedzené. Každá osoba, ktorá využije vyššie uvedenú licenciu, sa však zaväzuje priradiť každému dielu, ktoré vznikne (čo i len čiastočne) na základe Diela, úpravou Diela, spojením Diela s iným dielom, zaradením Diela do diela súborného či zpracovaním Diela (vrátane prekladu), licenciu aspoň vo vyššie uvedenom rozsahu a zároveň sa zaväzuje sprístupniť zdrojový kód takého diela aspoň zrovnateľným spôsobom a v zrovnateľnom rozsahu ako je zprístupnený zdrojový kód Diela.

V Prahe 4. júna 2020 . . . . . .

Odkaz na túto prácu

Hruška, Rajmund Hubert. *Implementácia automatových algoritmov na hľadanie rozšírených pokrytí.* Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Táto práca sa zaoberá popisom automatových algoritmov na hľadanie približných rozšírených pokrytí a uvoľnených približných rozšírených pokrytí a ich implementáciou do knižnice algoritmov ALT. Maximálny počet chýb, ktorým je vyjadrená približnosť, je počítaný Hammingovou vzdialenosťou. Základom algoritmov je deterministický sufixový konečný automat, ktorý vzniká z nedeterministického sufixového konečného automatu podmnožinovou metódou.

Kľúčové slová približné rozšírené pokrytie, knižnica algoritmov (ALT), Hammingova vzdialenosť, sufixový konečný automat, podmnožinová metóda

Abstract

The aim of this thesis is a description of automata algorithms for computing approximate enhanced covers and relaxed approximate enhanced covers and implementation of these algorithms to Algorithms Library Toolkit (ALT). Maximum number of errors, which is used as a metric of approximation, is computed under Hamming distance. The main idea behind these algorithms is deterministic suffix finite automaton constructed from nondeterministic suffix finite automaton by subset construction.

Keywords approximate enhanced cover, Algorithms Library Toolkit (ALT), Hamming distance, suffix finite automaton, subset construction

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Jednou z popredných disciplín teoretickej informatiky je teória automatov a gramatík. Štúdium textových reťazcov, súčasť tejto disciplíny, nachádza uplatnenie v mnohých vedeckých odboroch ako je napríklad bioinformatika, ale tiež v mnohých odborných procesoch ako napríklad kompresia dát.

Medzi skúmané problémy v rámci oboru patrí hľadanie pravidelností v textových reťazcoch. Príkladom takýchto pravidelnosti sú približné rozšírené pokrytie a uvoľnené približné rozšírené pokrytie. Tieto pravidelnosti zaviedol *Ondřej Guth* v [1, 2], kde zároveň predstavil algoritmy na ich výpočet. Táto práca je zameraná na implementáciu spomínaných algoritmov do knižnice algoritmov [3]. Práca sa nezaoberá implementáciou algoritmu na hľadanie presných (t. j. nie približných) rozšírených pokrytí – autorovi práce nie je známy automatový algoritmus, ktorý by riešil problém výpočtu presných rozšírených pokrytí.

Výsledok práce je prospešný pre študentov FIT ČVUT v Prahe, predovšetkým pre tých, ktorí študujú predmety zaoberajúce sa vyhľadávaním v texte s použitím automatových algoritmov (napríklad MI-AVY – Automaty ve vyhledávání). Prostredníctvom knižnice algoritmov [3], si môžu problémy experimentálne vyskúšať a pozrieť sa na zdrojový kód, kde je zdokumentované riešenie.

Ciele práce

Hlavným cieľom práce je popis automatových algoritmov na hľadanie približných rozšírených pokrytí a uvoľnených približných rozšírených pokrytí a ich následná implementácia do knižnice algoritmov [3].

Ďalším, nemenej dôležitým cieľom, je dôkladné testovanie. Správnosť implementácie daných algoritmov sa overí pomocou jednotkových testov, ktoré sú súčasťou implementácie, a pomocou testovacieho skriptu.

Štruktúra práce

Táto práca je štrukturovaná následovne. V kapitole 1 sú uvedené definície základných pojmov. Kapitola 2 sa zaoberá dôkladným popisom zvolených algoritmov od popisu myšlienky, cez pseudokód až po dôkaz časovej a pamäťovej zložitosti. V nasledujúcej kapitole je popísaná úprava predstavených algoritmov, tak aby boli dodržané štandardné programátorské praktiky a aby bolo vyhovené implementačným detailom programovacieho jazyka, v ktorom je knižnica algoritmov napísaná, a pritom bola zachovaná správnosť a časová zložitosť použitých algoritmov. Taktiež je predstavená knižnica algoritmov. V kapitole 4 sú popísané spôsoby testovania kódu. V závere je zhodnotenie splnenia cieľov práce a sú uvedené možné návrhy na rozšírenie tejto práce.

Kapitola 1

Základné definície

V tejto kapitole sú zavedené definície všetkých pojmov, ktoré sa používajú v nasledujúcich kapitolách. Taktiež sú predstavené niektoré základné algoritmy používané pri práci s automatmi.

Všetky definície a algoritmy, ktoré sú uvedené v tejto kapitole, sú prevzaté z [1, 4, 5].

1.1 Reťazce

Definícia 1.1. *Abeceda* je konečná množina symbolov, značí sa symbolom A.

Definícia 1.2. *Reťazec* x nad abecedou A, je konečná postupnosť symbolov z abecedy A. Množina všetkých reťazcov nad abecedou A sa značí ako A^*, t. j. $x \in A^*$.

Definícia 1.3. *Dĺžka* reťazca x je počet symbolov, ktorými je tvorený, a značí sa ako $|x|$. *i-ty* symbol reťazca x sa označuje ako $x[i]$.

Definícia 1.4. Prázdna postupnosť symbolov z abecedy A sa nazýva *prázdny reťazec* a značí sa symbolom ϵ. Dĺžka prázdneho reťazca je 0, teda $|\epsilon| = 0$.

Príklad 1.1. Štvrtý symbol reťazca $x = abaca$ nad abecedou $A = \{a, b, c\}$ je c, teda $x[4] = c$ a dĺžka reťazca je $|x| = 5$.

Definícia 1.5. *Zreťazením* reťazcov $x, y \in A^*$, t. j. pripojením reťazca y za reťazec x, vznikne reťazec xy.

Definícia 1.6. Nech $p, s, u, x \in A^*$, kde $x = pus$, potom p je *prefix reťazca* x, s je *sufix reťazca* x a u je *faktor reťazca* x (niekedy tiež nazývaný podreťazec). Reťazec p je *vlastný prefix* ak navyše platí $p \neq x$. Faktor reťazca x, ktorý začína na i-tom symbole a končí na j-tom symbole, sa označuje ako $x[i..j]$.

Príklad 1.2. Nech $x = abacacab$. Reťazec $abac$ je vlastný prefix reťazca x, reťazec cab je sufix reťazca x a reťazec $caca$ je faktor reťazca x.

Definícia 1.7. Editačná operácia *zámena* (tiež známa ako *substitúcia*) v reťazci $x \in A^*$ je nahradenie nejakého symbolu $x[i], 1 \leqslant i \leqslant |x|$ iným symbolom z A.

Definícia 1.8. Nech $x, y \in A^*$, $|x| = |y|$. *Hammingova vzdialenosť* reťazcov x, y (označovaná ako $H(x, y)$) je minimálny počet operácií zámeny potrebný k zmene x na y.

Príklad 1.3. Nech $x = bacba$ a $y = aabba$. Hammingova vzdialenosť reťazcov x, y je 2, t. j. $H(x, y) = 2$.

Poznámka 1.1. Existujú aj iné vzdialenostné funkcie, pre potreby práce je však použitá iba Hammingova vzdialenosť. Približnosť bude preto ďalej definovaná iba pre Hammingovu vzdialenosť.

Definícia 1.9. Nech $p, s, u, v, w \in A^*$ a $k \geqslant 0$. Reťazec v je *k-približný faktor* reťazca w, ak je možné zapísať w ako pus a $H(u, v) \leqslant k$.

Definícia 1.10. Reťazec $u \in A^*$ *sa vyskytuje* v reťazci $w \in A^*$, ak u je faktor reťazca w.

Definícia 1.11. Faktor u reťazca w, $u, w \in A^*$, *sa vyskytuje na pozícii i* (nazývaná tiež pozícia konca) reťazca w, ak pre každé $j \in \{1, \ldots, |u|\}$ platí, že $u[j] = w[i - |u| + j]$.

Príklad 1.4. Reťazec cca sa vyskytuje v reťazci $abccabab$ na pozícii 5.

Definícia 1.12. Pozícia l reťazca $w \in A^*$ *leží v nejakom výskyte* reťazca $u \in A^*$ v w, ak sa u vyskytuje na pozícii i v w a $i - |u| < l \leqslant i$.

Definícia 1.13. Reťazec $v \in A^*$ *sa k-približne vyskytuje na pozícii i* (nazývaná tiež k-približná pozícia konca) reťazca w, ak existuje faktor u reťazca w, ktorý sa vyskytuje na pozícii i v reťazci w a $H(u, v) \leqslant k$.

Definícia 1.14. Pozícia l reťazca $w \in A^*$ *leží v nejakom k-približnom výskyte* reťazca $v \in A^*$ v w, ak sa v k-približne vyskytuje na pozícii i v w a $i - |v| < l \leqslant i$.

Definícia 1.15. Hranica reťazca $x \in A^*$ je súčasne vlastný prefix a sufix reťazca x.

Definícia 1.16. Reťazec $w \in A^*$ je *k-približná hranica* reťazca $x \in A^*$, ak $H(x[1..|w|], w) \leqslant k$ a $H(x[|x| - |w| + 1..|x|], w) \leqslant k$.

Definícia 1.17. Hranica u reťazca y je *rozšírené pokrytie* reťazca y, ak počet pozícií v y, ktoré ležia v nejakom výskyte u v y, je maximálny medzi množinou všetkých hraníc reťazca y.

Definícia 1.18. Reťazec $w \in A^*$ je *k-približné rozšírené pokrytie* reťazca $x \in A^*$, ak w je hranica x a počet pozícií v x, ktoré ležia v nejakom k-približnom výskyte w v x, je maximálny medzi množinou všetkých hraníc reťazca x.

Definícia 1.19. Reťazec $w \in A^*$ je *uvoľnené k-približné rozšírené pokrytie* reťazca $x \in A^*$, ak w je faktor reťazca x a zároveň w je k-približná hranica x a počet pozícií v x, ktoré ležia v nejakom k-približnom výskyte w v x, je maximálny medzi množinou všetkých k-približných hraníc reťazca x.

Príklad 1.5. Nech $w = abacababacaba$ a $k = 1$. Reťazec aba je 1-približné rozšírené pokrytie a reťazec aca je uvoľnené 1-približné rozšírené pokrytie reťazca w.

Poznámka 1.2. V texte je častokrát uvedený termín približné rozšírené pokrytie. Myslí sa tým k-približné rozšírené pokrytie a obdobne to platí aj pre uvoľnené k-približné rozšírené pokrytie.

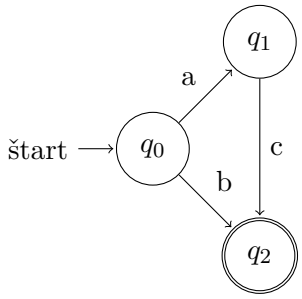
1.2 Konečné automaty

Definícia 1.20. *Deterministický konečný automat* (DKA) je usporiadaná pätica $M = (Q, A, \delta, q_0, F)$, kde

- Q je neprázdna konečná množina stavov
- A je neprázdna konečná vstupná abeceda
- $\delta : Q \times A \to Q$ je prechodová funkcia
- $q_0 \in Q$ je počiatočný stav
- $F \subseteq Q$ je množina koncových stavov

Konečné automaty sa častokrát znázorňujú formou orientovaných grafov, kde množina vrcholov je rovná množine stavov a hrana (q_i, q_j), ohodnotená symbolom a, patrí do množiny hrán práve vtedy, keď $\delta(q_i, a) = q_j$. Počiatočný stav je označený šípkou s nápisom štart a koncové stavy sú označené dvojitým orámovaním. Príklad konečného automatu je uvedený na obrázku 1.1.

Definícia 1.21. *Rozšírená prechodová funkcia deterministického konečného automatu* M, sa značí ako δ^* a je definovaná pre $q \in Q$, $a \in A$, $u \in A^*$ induktívne: $\delta^*(q, \epsilon) = q, \delta^*(q, ua) = \delta(\delta^*(q, u), a)$.



Obr. 1.1: Príklad konečného automatu $M = (Q, A, \delta, q_0, F)$, kde množina stavov $Q = \{q_0, q_1, q_2\}$, $A = \{a, b, c\}$, $F = \{q_2\}$ a $\delta(q_0, a) = q_1$, $\delta(q_0, b) = q_2$, $\delta(q_1, c) = q_2$.

Definícia 1.22. Stav $q_1 \in Q$ je *predchodca* stavu $q_2 \in Q$, ak $\exists a \in A$ také, že $\delta(q_1, a) = q_2$.

Definícia 1.23. Automat $M = (Q, A, \delta, q_0, F)$ je typu *„trie“*, keď každý stav $q \in Q \setminus \{q_0\}$ má práve jedného predchodcu.

Definícia 1.24. *Ľavý faktor* stavu $q \in Q$ automatu typu trie, je reťazec $w \in A^*$, taký, že $\delta^*(q_0, w) = q$.

Definícia 1.25. *Nedeterministický konečný automat* (NKA) je usporiadaná pätica $M = (Q, A, \delta, q_0, F)$, kde

- Q je neprázdna konečná množina stavov
- A je neprázdna konečná vstupná abeceda
- $\delta : Q \times A \to 2^Q$ je prechodová funkcia[1]
- $q_0 \in Q$ je počiatočný stav
- $F \subseteq Q$ je množina koncových stavov

Definícia 1.26. *Rozšírená prechodová funkcia nedeterministického konečného automatu* M_N, sa značí ako δ^* a je definovaná pre $q_1, q_2 \in Q$, $a \in A$, $u \in A^*$ induktívne: $\delta^*(q_1, \epsilon) = \{q_1\}, \delta^*(q_1, ua) = \bigcup_{q_2 \in \delta^*(q_1, u)} \delta(q_2, a)$.

Každý konečný automat je buď deterministický alebo nedeterministický, pričom nedeterministický automat sa dá previesť na deterministický algoritmom 1.2.

[1] 2^Q značí množinu všetkých podmnožín Q

Definícia 1.27. Reťazec w *je akceptovaný* automatom $M = (Q, A, \delta, q_0, F)$ práve vtedy, keď $\delta^*(q_0, w) \in F$. Jazyk $L(M) = \{w \mid w \in A^*, \delta^*(q_0, w) \in F\}$ je jazyk akceptovaný automatom M. Automat M *akceptuje množinu reťazcov* $B \subseteq L(M)$, práve vtedy, keď každý reťazec $u \in B$ je akceptovaný automatom M.

Definícia 1.28. Konečné automaty M_1 a M_2 sú *ekvivalentné* práve vtedy, keď $L(M_1) = L(M_2)$.

Definícia 1.29. *Sufixový automat* pre reťazec $x \in A^*$ je konečný automat, ktorý akceptuje množinu všetkých sufixov reťazca x. Nech S je množina všetkých k-približných sufixov reťazca u definovaná následovne: $v \in S$, ak pre sufix s reťazca x, taký, že $|s| = |v|$, platí $H(s, v) \leqslant k$. *k-približný sufixový automat* pre reťazec x je konečný automat, ktorý akceptuje množinu k-približných sufixov S reťazca x.

Podľa [1] je možné zostrojiť nedeterministický k-približný sufixový automat pre reťazec w nasledujúcim spôsobom. Najprv sa vytvorí $|w| + 1$ stavov (jeden pre každý symbol a jeden počiatočný stav). Tieto stavy sú označené q_i^0 pre $0 < i \leqslant |w|$ a počiatočný stav je označený ako q_0^0. Stav q_{i-1}^0 prechádza na symbol $w[i]$ do stavu q_i^0. Vrstva týchto stavov a prechodov sa kopíruje z predošlej $(j-1)$-vrstvy pre každé $1 \leqslant j \leqslant k$, pričom sa vynechá stav s najnižším i. To znamená, že sa vytvorí vrstva so stavmi $q_1^1 \dots q_{|w|}^1$, potom vrstva $q_2^2 \dots q_{|w|}^2$, a tak ďalej, až sa nakoniec vytvorí vrstva $q_k^k \dots q_{|w|}^k$. Pre každý stav q_{i-1}^{j-1} sa pridá prechod do stavu q_i^j pre každý symbol $a \in A \setminus \{w[i]\}$. Nakoniec sa pridajú prechody z počiatočného stavu do stavov q_i^0 pre symbol $w[i]$, pre všetky $i > 0$, a z počiatočného stavu do stavov q_i^1 pre všetky symboly z $A \setminus \{w[i]\}$. Pseudokód tohto postupu je uvedený ako algoritmus 1.1.

Algoritmus 1.1 Konštrukcia nedeterministického k-približného sufixového automatu pre reťazec w

Vstup: vstupný reťazec $w \in A^*$, maximálna Hammingova vzdialenosť k
Výstup: nedeterministický k-približný sufixový automat $M_N = (Q, A, \delta, q_0, F)$ pre reťazec w

$Q \leftarrow \{q_i^j \mid 0 \leqslant i \leqslant |w| \wedge 0 \leqslant j \leqslant k \wedge j \leqslant i\}$
$F \leftarrow \{q_{|w|}^j \mid 0 \leqslant j \leqslant k\}$
$\delta(q_{i-1}^j, w[i]) \leftarrow \{q_i^j\}$ pre všetky i, j, také, že $1 \leqslant i \leqslant |w|$, $0 \leqslant j \leqslant k$
$\delta(q_{i-1}^{j-1}, a) \leftarrow \{q_i^j\}$ pre všetky a, i, j, také, že $1 \leqslant i \leqslant |w|$, $1 \leqslant j \leqslant k$, $a \in A \setminus \{w[i]\}$
$\delta(q_0, w[i]) = \{q_i^0\}$ pre všetky i také, že $2 \leqslant i \leqslant |w|$
return (Q, A, δ, q_0^0, F)

Definícia 1.30. Nech $M = (Q, A, \delta, q_0, F)$ je k-približný deterministický sufixový automat pre reťazec x. *Chrbtica* automatu M je automat $M_B = (Q_B, A, \delta_B, q_0, F_B)$ taký, že pre všetky $0 < i \leqslant |x|$ platí $Q_B = \{q_i \,|\, q_i \in Q\} \cup \{q_0\}$, $\delta_B(q_{i-1}, x[i]) = q_i$ a $F_B = \{q \,|\, q \in Q_B \cap F\}$.

Poznámka 1.3. Definícia 1.30 je oproti definícii v [1] pozmenená. Podľa pôvodnej definície by stav $q_0 \notin Q_B$, čo nedáva zmysel.

Algoritmus 1.2 Transformácia NKA na DKA použitím podmožinovej konštrukcie

---

Vstup: nedeterministický konečný automat $M = (Q, A, \delta, q_0, F)$
Výstup: deterministický konečný automat $M' = (Q', A, \delta', q_0', F')$ taký, že $L(M) = L(M')$

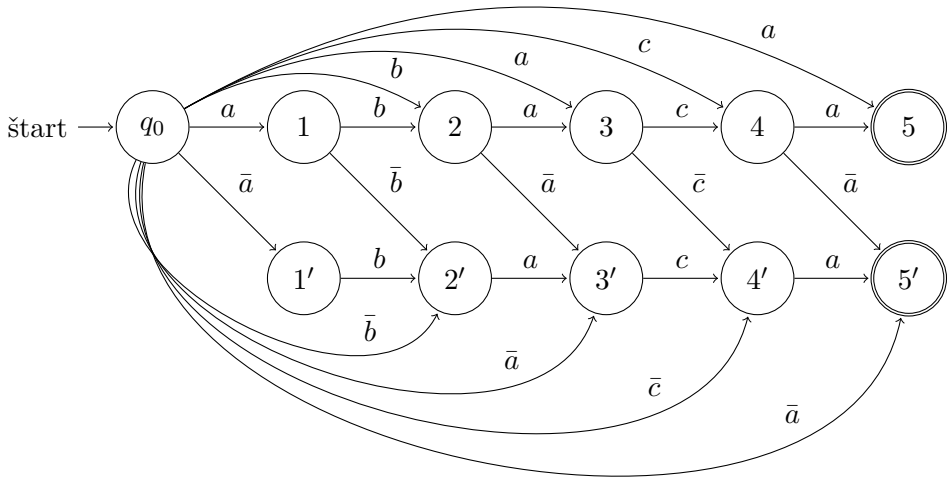
$Q' \leftarrow \{\{q_0\}\}$
$q_0' \leftarrow \{q_0\}$ // každý stav DKA je podmnožina stavov NKA
q_0' je neoznačený
while $\exists$ neoznačený stav $q' \in Q'$ **do**
  $\delta'(q', a) \leftarrow \bigcup \delta(p, a)$ pre $p \in q'$ a $\forall a \in A$
  $Q' \leftarrow Q' \cup \delta'(q', a)$ pre $\forall a \in A$
  $q' \in Q'$ je označený
end while
$F' \leftarrow \{q' \,|\, q' \in Q', q' \cap F \neq \emptyset\}$
return $(Q', A, \delta', q_0', F')$

Definícia 1.31. *d-podmnožina* (deterministická podmnožina) stavu q deterministického konečného automatu je usporiadaná množina prvkov (usporiadanie je definované ďalej), označovaná ako $d(q)$. Každý prvok korešponduje so stavom nedeterministického konečného automatu. Prvok e je dvojica čísel: *hĺbka*(e) a *úroveň*(e) (v anglickom jazyku *depth(e)* a *level(e)*), kde *hĺbka*(e) reprezentuje pozíciu konca a *úroveň*(e) reprezentuje Hammingovu vzdialenosť od nejakého faktora u. Prvky d-podmnožiny sú usporiadané vzostupne podľa *hĺbky*.

V diagramoch konečných automatov je hĺbka prvku d-podmnožiny označená prirodzeným číslom a úroveň je daná počtom jednoduchých úvodzoviek v hornom indexe.

1.3 Asymptomatická zložitosť

Definícia 1.32. Nech je daná funkcia $g(n)$, potom $\mathcal{O}(g(n))$ je nekonečná množina funkcií $\{f(n) \,|\, ((\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geqslant n_0)) : f(n) \leqslant c \cdot g(n)\}$.



Obr. 1.2: Nedeterministický 1-približný sufixový automat pre reťazec $abaca \in A^*$ ($\bar{a}$ značí doplnok, t. j. $\bar{a} = A \setminus \{a\}$). Hĺbka stavu je označená číslom a úroveň stavu je označená v pravom hornom indexe.

Kapitola 2

Popis existujúcich algoritmov

Táto kapitola sa zaoberá popisom algoritmov na hľadanie približných rozšírených pokrytí a uvoľnených približných rozšírených pokrytí, tak ako sú uvedené v [1].

Najprv je predstavená myšlienka algoritmu, potom pseudokód a následne je uvedený dôkaz časovej a pamäťovej zložitosti.

2.1 Približné rozšírené pokrytie

Myšlienka algoritmu uvedeného v [1] je pomerne jednoduchá. Vstupný reťazec je prechádzaný znak po znaku od začiatku a pre každú pozíciu sa zisťuje, či je to pozícia konca nejakej hranice vstupného reťazca. V prípade, že je to hranica, tak sa spočíta počet pozícií, ktoré ležia vo všetkých k-približných výskytoch tejto hranice vo vstupnom reťazci. Ak je tento počet väčší alebo rovný doteraz nájdenému najväčšiemu počtu, tak sa príslušne aktualizuje množina k-približných rozšírených pokrytí.

Ako základná indexovacia štruktúra je zvolená chrbtica deterministického k-približného sufixového automatu pre vstupný reťazec x, ktorý vznikol z ekvivalentného nedeterministického k-približného sufixového automatu podmnožinovou konštrukciou podobnou algoritmu 1.2. Z definície 1.30 vyplýva, že chrbtica je typu trie. Pre každý stav q_i tak platí, že ľavý faktor stavu q_i je prefix dĺžky i.

Ak d-podmnožina stavu q chrbtice obsahuje prvok e taký, že $\mathit{hĺbka}(e) = |e|$, tak q je koncový stav. Ak navyše platí, že $\mathit{úroveň}(e) = 0$, tak stav q zodpovedá nejakej hranici.

Poznámka 2.1. Aby stav q skutočne zodpovedal hranici, tak musí platiť, že prvý prvok d-podmnožiny má úroveň 0, ale to pre chrbticu platí vždy.

Podľa [1], pre každý stav q chrbtice deterministického k-približného sufixového automatu platí, že jeho d-podmnožina je rovná množine pozícií koncov

k-približných výskytov ľavého faktora. Vďaka tomu je možné spočítať počet písmen reťazca x, ktoré sú pokryté ľavým faktorom stavu q.

Nech p je ľavý faktor stavu q. Existujú tri prípady pre každé dva susedné pozície $i, j, i < j$ k-približných výskytov v d-podmnožine stavu q:

- $(j - i < |p|)$ pokrytých je $j - i$ písmen
- $(j - i = |p|)$ pokrytých je $|p|$ písmen
- $(j - i > |p|)$ pokrytých je $|p|$ písmen

Pseudokód tohto výpočtu je uvedený ako funkcia 1.

Funkcia 1 distEnhCov

Vstup: stav q deterministického k-približného sufixového automatu pre reťazec x

Výstup: počet písmen reťazca x, ktoré sú pokryté hranicou prislúchajúcou stavu q

```
r ← 0
e_f je prvý prvok d-podmnožiny
m ← hĺbka(e_f)
E je d-podmnožina stavu q
for i ∈ 2..|E| do
  if hĺbka(E[i]) − hĺbka(E[i − 1]) < m then
    r ← r+ hĺbka(E[i]) − hĺbka(E[i − 1])
  else
    r ← r + m
  end if
end for
return r
```

Poznámka 2.2. Vo funkcii 1, tak ako je uvedená v [1] je chyba. Popis chyby a návrh opravy je prezentovaný v kapitole 3.

V [1] sa píše, že nie je nutné udržiavať si celú chrbticu v pamäti. Po spracovaní stavu q_i sa spracováva stav q_{i+1} a stav q_i možno zahodiť. Je preto možné použiť prehľadávanie do hĺbky. Na to sú potrebné dve funkcie – jedna na konštrukciu prvého stavu a druhá, ktorá zo stavu q_i vypočíta nasledujúci stav q_{i+1}.

Prvý stav chrbtice sa počíta následovne. Vytvorí sa stav q_1 s prázdnou d-podmnožinou. Znak na každej pozícii $i, 1 \leqslant i \leqslant |x|$ vstupného reťazca x sa porovná s prvým znakom reťazca x. Ak sa znak na pozícii i rovná znaku na prvej pozícii, tak sa do d-podmnožiny vloží prvok s hĺbkou i a úrovňou 0. V opačnom prípade, ak je $k > 0$, tak sa do d-podmnožiny pridá prvok s hĺbkou i a úrovňou 1. Pseudokód tohto výpočtu je uvedený ako funkcia 2.

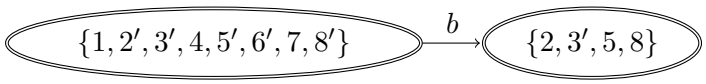
Funkcia 2 constrFirstState
Vstup: vstupný reťazec x, maximálna Hammingova vzdialenosť k
Výstup: prvý stav q_1 chrbtice automatu pre x

```
q_1 je nový stav
for i ∈ 1..|x| do
  e je prvok d-podmnožiny, taký že hĺbka(e) ← i
  if x[1] = x[i] then
    úroveň(e) ← 0
    pridaj e do d-podmnožiny stavu q_1
  else if k > 0 then
    úroveň(e) ← 1
    pridaj e do d-podmnožiny stavu q_1
  end if
end for
return q_1
```

Príklad prvého stavu chrbtice, tak ako ho vytvorí funkcia 3, je zobrazený na obrázku 2.1.

Obr. 2.1: Prvý stav chrbtice 1-približného deterministického sufixového automatu pre reťazec $x = abbabcab$.

Ďalší stav q_n sa zo stavu q_p počíta podobne ako prvý stav. Vytvorí sa stav q_n s prázdnou d-podmnožinou. Z definície d-podmnožiny, je každý prvok stavom ekvivalentného nedeterministického automatu. Pre každý prvok $e \in q_p$ sa teda spočíta jeho nasledujúci stav a ten sa uloží do d-podmnožiny stavu q_n. V prípade, že by prechod z prvku e v nedeterministickom automate nebol definovaný, tak sa tento prvok nepoužije (typicky, ak hĺbka$(e) = |x|$ alebo úroveň$(e) = k$). Počet prvkov d-podmnožiny stavu q_n je preto menší alebo rovný počtu prvkov d-podmnožiny stavu q_p. Príklad konštrukcie nasledujúceho stavu je zobrazený na obrázku 2.2. Pseudokód je uvedený ako funkcia 3.



Obr. 2.2: Nasledujúci stav chrbtice 1-približného deterministického sufixového automatu pre reťazec $x = abbabcab$.

Poznámka 2.3. Vo funkcii 3 tak, ako je uvedená v [1], nie je medzi vstupnými parametrami Hammingova vzdialenosť k. Zjavne ale ide o chybu, preto je

Funkcia 3 constrNextState
Vstup: vstupný reťazec x, Hammingova vzdialenosť k, stav q_p
Výstup: stav q_n ďalší stav chrbtice automatu pre x

```
qn je nový stav
for ep ∈ qp do
  if hĺbka(ep) < |x| then
    en je prvok d-podmnožiny, taký že hĺbka(en) ← hĺbka(ep) + 1
    if x[i] = x[hĺbka(en)] then
      úroveň(en) ← úroveň(ep)
      pridaj en do d-podmnožiny stavu qn
    else if úroveň(ep) < k then
      úroveň(en) ← úroveň(ep) + 1
      pridaj en do d-podmnožiny stavu q1
    end if
  end if
end for
return qn
```

v tomto texte daný parameter pridaný.

V [1] sa píše, že keď má d-podmnožina stavu q iba jeden prvok, tak nie je nutné spracovávať ďalšie stavy. V takom prípade ľavý faktor q nie je hranica a ani ľavý faktor žiadneho z nasledujúcich stavov nie je hranica.

Aktualizácia množiny nájdených približných rozšírených pokrytí prebieha následovne. Pre daný stav q sa spočíta, koľko pokrýva písmen (pomocou funkcie 1), nech je to h_n. Ak je to rovnaký počet, ako doteraz maximálny počet h, tak sa pridá hranica reprezentovaná stavom q do množiny nájdených približných rozšírených pokrytí C. Ak je ale počet pokrytých písmen h_n väčší ako doteraz maximálny počet h, tak sa h aktualizuje na túto novú hodnotu h_n, všetky prvky množiny C sa zahodia a pridá sa tam hranica reprezentovaná stavom q. Pseudokód tohto výpočtu je uvedený ako funkcia 4.

Použitím všetkých popísaných funkcií vznikne algoritmus 2.1 na počítanie všetkých k-približných rozšírených pokrytí.

Príklad 2.1. Tento príklad vychádza z príkladu uvedeného v [1].

Nech je daný vstupný reťazec $x = abacaccababa$ a Hammingova vzdialenosť $k = 1$. Prvý stav q_1 chrbtice deterministického 1-približného sufixového automatu je $\{1, 2', 3, 4', 5, 6', 7', 8, 9', 10, 11', 12\}$. Ide o stav s hĺbkou 1 a Hammingova vzdialenosť je 1, preto tento stav nie je dôležitý. Následne sa vytvorí nasledujúci stav q_2, ktorého d-podmnožina je $\{2, 4', 6', 9, 11\}$. Posledný prvok má hĺbku 11, takže stav nereprezentuje hranicu reťazca x. d-podmnožina ďalšieho stavu je $\{3, 5', 10, 12\}$. Hĺbka posledného prvku je 12 a jeho úroveň je 0, takže stav reprezentuje hranicu. Počet pokrytých písmen je 10, a tak sa

Funkcia 4 updateEnhCov

Vstup: stav q, množina nájdených pokrytí C, počet pokrytých písmen h
Výstup: aktualizovaná množina C a počet pokrytých písmen h

```
h_n ← distEnhCov(q)
u ← ľavý faktor stavu q
if h_n > h then
  h ← h_n
  C ← ∅
end if
if h_n = h then
  C ← C ∪ {u}
end if
return (C, h)
```

aktualizuje množina nájdených pokrytí. Nasledujúce dva stavy nie sú koncové a stav, ktorý nasleduje po nich obsahuje iba jeden prvok vo svojej d-podmnožine, takže sa výpočet zastaví.

Algoritmus 2.1 Počítanie všetkých k-približných rozšírených pokrytí

Vstup: reťazec x, maximálna Hammingova vzdialenosť k
Výstup: množina všetkých k-približných rozšírených pokrytí

```
C ← ∅
h ← 0
q_1 ← constrFirstState(x, k)
p ← 1
q_p ← q_1
for i ∈ 2..|x| do
  q_n ← constrNextState(x, k, q_p)
  p ← p + 1
  odstráň q_p z pamäte
  if počet prvkov d−podmnožiny stavu q_n je menší ako 2 then
    break
  end if
  e_f je posledný prvok d-podmnožiny stavu q_n
  if hĺbka(e_f) = x and úroveň(e_f) = 0 and p > k then
    (C, h) ← updateEnhCov(q_n, C, h)
  end if
end for
return C
```

2.1.1 Analýza časovej a pamäťovej zložitosti

Nasledujúce dôkazy vychádzajú z [1].

Počas výpočtu je potrebné si v pamäti udržiavať vstupný reťazec x, množinu výsledkov C, dĺžku p aktuálneho prefixu, stavy a ich d-podmnožiny. Každý stav je charakterizovaný svojou d-podmnožinou, ktorá je v pamäti uložená ako pole. Prvky d-podmnožiny sú dvojice čísel, každé číslo v algoritme 2.1 je v rozsahu 0 až $\max(|x|, k)$.

Cyklus vo funkcii 2 má $|x|$ iterácií. V každej iterácii sa pridá najviac jeden prvok do d-podmnožiny. Preto stav q_1 vo svojej d-podmnožine obsahuje najviac $2|x|$ čísel.

Počas konštrukcie nasledujúceho stavu (funkcia 3), sa z každého prvku d-podmnožiny stavu q_p vytvorí nový prvok d-podmnožiny stavu q_n. Tento nový prvok sa do d-podmnožiny nepridá iba v dvoch prípadoch – v prípade, že prekročí maximálnu úroveň k alebo v prípade, že jeho hĺbka presiahne dĺžku reťazca x. Z toho vyplýva, že počet prvkov v d-podmnožine nerastie, vo väčšine prípadov dokonca klesá.

Pre identifikáciu nájdeného približného rozšíreného pokrytia stačí poznať dĺžku prefixu. Pri nájdení daného pokrytia sa tak do množiny C uloží dĺžka aktuálneho prefixu p. Množina C tak obsahuje najviac $|x|$ čísel.

V algoritme 2.1 sú v každom kroku v pamäti uložené najviac dva stavy, a teda d-podmnožiny obsahujú spolu najviac $4|x|$ čísel.

Z vyššie uvedeného vyplýva, že pamäťová zložitosť algoritmu 2.1 je $\mathcal{O}(n)$, kde $n = |x|$.

Funkcia 2 obsahuje cyklus, ktorý má $|x|$ iterácii. V každej iterácii sa pridá prvok do d-podmnožiny, pričom toto pridávanie trvá konštantne dlho, pretože d-podmnožina je uložená ako pole.

Cykly vo funkciách 1 a 3 trvajú $\mathcal{O}(|q_p|)$, teda $\mathcal{O}(|x|)$, pričom všetky príkazy v týchto cykloch trvajú konštantne dlho.

Aktualizácia množiny nájdených pokrytí (funkcia 4) trvá, pri použití optimálnej implementácie množiny, logaritmický čas od počtu prvkov v množine. Predtým sa ale zavolá funkcia 1, takže časová zložitosť funkcie 4 je $\mathcal{O}(|x|)$

Hlavný cyklus v algoritme 2.1 má najviac $|x| - 1$ iterácií. V každej iterácii sa zavolá funkcia 3 a v niektorých prípadoch funkcia 4. Celý cyklus má tak v najhoršom prípade kvadratickú zložitosť a preto časová zložitosť algoritmu 2.1 je $\mathcal{O}(n^2)$.

2.2 Uvoľnené približné rozšírené pokrytie

V [1] sa odkazuje na algoritmus 3, pseudokód tohto algoritmu sa ale v článku nenachádza. V tejto časti je preto uvedený slovný popis algoritmu a pseudokód funkcie, ktorá sa v tomto algoritme používa, tak ako je uvedené v [1]. Samotný pseudokód algoritmu 3, o ktorom sa v článku píše, je dielom autora tejto práce a je uvedený ako algoritmus 2.2.

Myšlienka algoritmu na hľadanie uvoľnených k-približných rozšírených pokrytí je podobná tej na hľadanie k-približných rozšírených pokrytí. Prechádza sa stavmi deterministického k-približného sufixového automatu a pre každý stav sa zisťuje, či jeho ľavý faktor je kandidátom na uvoľnené k-približné rozšírené pokrytie.

Podľa [1], chrbtica deterministického k-približného sufixového automatu na počítanie uvoľnených k-približných rozšírených pokrytí nestačí. Dôvodom je fakt, že podľa definície uvoľneného rozšíreného pokrytia (1.19), faktor nemusí byť presný prefix a sufix. Môže sa tak stať, že uvoľnené k-približné pokrytie bude prijaté stavom, ktorý nie je súčasťou chrbtice.

Vytvorí sa nedeterministický k-približný sufixový automat M_N podľa algoritmu 1.1. Následne sa postupne, stav po stave, vytvára ekvivalentný deterministický k-približný automat M algoritmom podobným podmnožinovej konštrukcii (algoritmus 1.2). Výsledkom tohto algoritmu je podľa [1] automat typu trie.

Každý novovzniknutý stav q automatu M je skontrolovaný, či môže byť kandidátom na uvoľnené približné rozšírené pokrytie. Kandidátom je v prípade, že je to koncový stav, reprezentuje faktor reťazca x, jeho d-podmnožina obsahuje aspoň dva prvky a ľavý faktor stavu q je približný prefix. Aby ľavý faktor stavu q bol približný prefix, tak musí platiť, že hĺbka prvého prvku d-podmnožiny stavu q je zhodná s hĺbkou stavu q. Hĺbka stavu q je určená počtom prechodov z počiatočného stavu q_0 do stavu q.

Tak, ako pri výpočte približných rozšírených pokrytí, tak aj v prípade výpočtu uvoľnených rozšírených pokrytí nemá zmysel počítať následníka stavu s iba jedným prvkom v d-podmnožine. Okrem toho, sú v [4] uvedené ďalšie dve podmienky, kedy už nemá zmysel počítať následníka nejakého stavu. Nech M je k-približný deterministický sufixový automat pre reťazec w, q_1 a q_2 sú stavy automatu M také, že q_1 je predchodca q_2, u_1 je ľavý faktor stavu q_1 a u_2 je ľavý faktor stavu q_2. Ak u_1 nie je k-približný prefix reťazca w, tak ani u_2 nemôže byť k-približný prefix reťazca w. Ak u_1 nie je faktor reťazca w, tak ani u_2 nemôže byť faktor reťazca w. V oboch prípadoch nemá zmysel počítať následníka stavu q_1.

Podľa [1] je každý faktor u, ktorý je kandidátom na uvoľnené približné rozšírené pokrytie, reprezentovaný ako dvojica čísel $(\text{hĺbka}(e_x), |u|)$, kde e_x je prvok d-podmnožiny s úrovňou 0. Hĺbka prvku e_x tak určuje pozíciu konca reťazca u.

Hlavnou časťou algoritmu na výpočet všetkých k-približných uvoľnených pokrytí je funkcia 5. Táto funkcia simuluje konštrukciu deterministického k-približného sufixového automatu prehľadávaním do hĺbky, pričom každý prechádzaný stav je skontrolovaný, či je možným kandidátom. Keď už stav nie je potrebný, tak sa odstráni z pamäte.

Ako už bolo uvedené na začiatku tejto časti, v [1] sa odkazuje na neuvedený algoritmus 3, ktorý počíta všetky k-približné uvoľnené rozšírené pokry-

Funkcia 5 processState

Vstup: stav q_i čiastočne skonštruovaného $M = (Q, A, \delta, q_0, F)$, maximálna Hammingova vzdialenosť k, nedeterministický k-približný sufixový automat $M_N = (Q_N, A, \delta_N, q_{0_N}, F)$, reťazec u

C je globálna premenná, množina uvoľnených k-približných rozšírených pokrytí

h je globálna premenná, maximálny počet pokrytých pozícií

for $a \in A$ **do**
  q je nový stav
  e_x je nedefinovaný prvok
  hĺbka(q) $\leftarrow$ hĺbka(q_i) + 1
  for $e_j \in d(q_i)$ **do**
    for $e_l \in \delta_N(e_j, a)$ **do**
      pridaj e_l do d-podmnožiny stavu q
      if $e_l \in F_N$ **then**
        $F \leftarrow F \cup \{q\}$
      end if
      if úroveň(e_l) = 0 **then**
        $e_x \leftarrow e_l$
      end if
    end for
  end for
  $e_1 \in d(q)$ je prvý prvok d-podmnožiny
  if $|d(q)| > 1$ **and** hĺbka(e_1) = hĺbka(q) **then**
    if e_x je definovaný **then**
      $u \leftarrow ua$ // u je ľavý faktor stavu q
      $Q \leftarrow Q \cup \{q\}$
      $\delta(q_i, a) \leftarrow q$
      if $q \in F$ **and** hĺbka(q) $> k$ **then**
        $(C, h) \leftarrow$ updateEnhCov(q, C, h)
      end if
      processState(q, k, M_N, u)
    end if
  end if
  odstráň stav q z pamäte
  odstráň posledný symbol z u
end for

tia. Pseudokód algoritmu 2.2 je teda dielom autora tejto práce. Pri návrhu sa vychádza z popisu uvedeného v [1].

Príklad 2.2. Tento príklad vychádza z príkladu uvedeného v [1].

Nech je daný vstupný reťazec $x = abacaccababa$ a Hammingova vzdialenosť

$k = 1$. Najprv sa vytvorí nedeterministický 1-približný sufixový automat M_N. Po niekoľkých výpočtoch funkcie 5 sú pre ľavý faktor $u = aba$ v pamäti uložené stavy $\{1, 2', 3, 4', 5, 6', 7', 8, 9', 10, 11', 12\}$, $\{2, 4', 6', 9, 11\}$ a $\{3, 5', 10, 12\}$. Počas konštrukcie posledného zo spomínaných stavov sa zistí, že tento stav je možným kandidátom. Počet pokrytých písmen týmto stavom je 10, a preto sa do množiny uvoľnených približných rozšírených pokrytí uloží ľavý faktor u. Následne sú spracované stavy $\{4, 6', 11'\}$ a $\{5, 12'\}$. Stav $\{5, 12'\}$ pokrýva 10 pozícií, takže sa do množiny uvoľnených približných rozšírených pokrytí pridá jeho ľavý faktor. Počas ďalšieho spracovania sa pre ľavý faktor $u = aca$ vytvoria stavy $\{2', 4, 6, 7', 9', 11'\}$ a $\{3', 5, 7', 8', 10', 12'\}$. Stav $\{3', 5, 7', 8', 10', 12'\}$ pokrýva 12 pozícií, čo je viac ako bolo doterajšie maximum, a preto sa množina uvoľnených približných rozšírených pokrytí vyprázdni a pridá sa tam nájdený ľavý faktor.

Algoritmus 2.2 Počítanie všetkých uvoľnených k-približných rozšírených pokrytí

Vstup: reťazec $x \in A^*$, maximálna Hammingova vzdialenosť k
Výstup: množina všetkých uvoľnených k-približných rozšírených pokrytí

```
C ← ∅
h ← 0
M_N je nedeterministický k-približný sufixový automat pre reťazec x
for a ∈ A do
    q_a je stav determistického k-približného sufixového automatu M =
    (Q, A, δ, q_0, F) taký, že δ(q_0, a) = q_a
    u ← a
    processState(q_a, k, M_N, u)
    odstráň q_a z pamäte
end for
return C
```

2.2.1 Analýza časovej a pamäťovej zložitosti

Nasledujúci dôkaz pochádza z [1].

Nech je daný vstupný reťazec x nad abecedou A, Hammingova vzdialenosť k a nech dĺžka vstupného reťazca x je n, teda $n = |x|$.

V pamäti sú uložené: číslo h, množina C obsahujúca dvojice čísel, stavy a prechody automatu M_N a d-podmnožiny automatu M. Všetky objekty majú rovnakú štruktúru a obmedzenia, ako bolo uvedené v kapitole 2.1.1.

Podľa [4] má nedeterministický sufixový automat M_N celkom $(n+1)(k+1) - \frac{k^2+k}{2}$ stavov a $|A|(n(k+1) - 1 + \frac{k-k^2}{2}) + n - k + 1$ prechodov. V [1] sa ale píše, že na konštrukciu deterministického automatu M sú potrebné iba stavy a prechody z počiatočného stavu a ostatné stavy automatu M_N sa dajú vypočítať v čase $\mathcal{O}(1)$. Celkovo tak M_N potrebuje $\mathcal{O}(n(k + |A|))$ priestoru.

Vo funkcii 5 sa vytvárajú nasledovníci stavu q deterministického automatu M, pričom v pamäti je vždy uložený najviac jeden nasledovník. Každý taký nasledovník obsahuje d-podmnožinu prvkov, ktoré sú nasledovníkmi prvkov d-podmnožiny stavu q. Z toho vyplýva, že hĺbky prvkov d-podmnožiny nasledovníka rastú. Hĺbka prvku korešponduje s pozíciou vo vstupnom reťazci, a preto nemôže byť väčšia ako dĺžka vstupného reťazca. Preto je v pamäti uložených najviac n stavov. Každý stav deterministického sufixového automatu obsahuje najviac n prvkov vo svojej d-podmnožine.

Pamäťová zložitosť algoritmu 2.2 je $\mathcal{O}(n^2)$.

Podľa [1] je možné zostrojiť nedeterministický automat M_N v čase $\mathcal{O}(n(k+|A|))$. Maximálny počet stavov deterministického automatu M, vytvorených rekurzívnym volaním funkcie 5 je $\mathcal{O}((k+|A|)n^2)$. To je rovné súčtu počtu faktorov reťazca x a počtu následníkov, ktoré už nereprezentujú faktor a nie sú ďalej spracovávané.

Každá d-podmnožina má najviac n prvkov. Nájdenie nasledovníka prvku je vďaka prechodovej funkcii δ_N spočítateľné v konštantnom čase. d-podmnožinu je teda možné zostrojiť v lineárnom čase.

Ako už bolo uvedené v 2.1.1, funkcia 1 má lineárnu zložitosť $\mathcal{O}(n)$. Ľavý faktor daného stavu je uložený ako dvojica čísel a nie ako celý reťazec, preto má funkcia 4 taktiež lineárnu časovú zložitosť.

Podľa [1] je tak časová zložitosť algoritmu 2.2 rovná počtu skonštruovaných stavov vynásobeného maximálnou veľkosťou d-podmnožiny, teda $\mathcal{O}(n^3 \cdot (k+|A|))$.

Kapitola 3

Implementácia

Táto kapitola sa zaoberá samotnou implementáciou popísaných algoritmov do knižnice algoritmov [3].

Najprv je predstavená knižnica algoritmov ALT. Následne je popísaná úprava algoritmov, ktorej hlavným cieľom je odstránenie duplicity kódu. Ďalej sú predstavené triedy a štruktúry, v ktorých sú algoritmy implementované.

3.1 ALT

Základy knižnice algoritmov položil *Martin Žák* vo svojej bakalárskej práci [6]. Skratka názvu knižnice vychádza z prvých písmen jej anglického názvu – `Algorithms Library Toolkit`. ALT je zoskupením rôznych knižníc a dvoch spustiteľných programov, `aql2` a `agui2`, ktoré sprístupňujú algoritmy z týchto knižníc.

Knižnica algoritmov ALT je napísaná v jazyku `C++`, a preto je implementácia tiež zrealizovaná v tomto jazyku.

Pred použitím knižnice je potrebné stiahnuť zdrojové kódy z [3], kde je zároveň uvedený postup inštalácie.

Knižnica algoritmov má licenciu GNU GPL, a preto sú zdrojové kódy implementácie algoritmov na hľadanie približných rozšírených pokrytí a uvoľnených približných rozšírených pokrytí licencované pod rovnakou licenciou [7].

3.2 Úprava algoritmov

V predchádzajúcej kapitole je uvedené, že vo funkcii 1 je chyba. Tá spočíva v tom, že sa pri počítaní počtu pokrytých písmen neberie do úvahy prvý prvok d-podmnožiny. Chybu a očakávaný výsledok ilustruje príklad 3.1. Úprava je jednoduchá – stačí na konci výpočtu k výsledku pripočítať hĺbku prvého prvku d-podmnožiny. Nejedná sa o komplikovanú úpravu, a preto pseudokód upravenej funkcie uvedený nie je.

Príklad 3.1. Pre stav q taký, že jeho d-podmnožina je $\{3, 7, 8, 13\}$, by funkcia 1 vrátila výsledok 7, ale správny výsledok je 10.

V algoritme 2.2 je skonštruovaný nedeterministický k-približný sufixový automat pre vstupný reťazec x. Vo funkcii 5 sa ale používa iba časť automatu, a to množina koncových stavov a prechodová funkcia.

Zistiť, či je daný stav q deterministického sufixového automatu koncový, sa dá jednoducho porovnaním hĺbky posledného prvku d-podmnožiny stavu q s dĺžkou vstupného reťazca. Stav q je koncový práve vtedy, keď sú obe hodnoty zhodné.

Simuláciu prechodovej funkcie je možné dosiahnuť jednoduchou úpravou funkcie 3. Pre počítanie uvoľnených k-približných rozšírených pokrytí tak nie je nutné skonštruovať nedeterministický k-približný sufixový automat.

Z definície 1.19 je zrejmé, že pre $k = 0$ a ľubovoľný reťazec x je množina približných rozšírených pokrytí zhodná s množinou uvoľnených približných rozšírených pokrytí. V [1] sa nepredpokladá $k = 0$, a preto pôvodné algoritmy nevrátia vždy správny výsledok, čo je možné vidieť na príklade 3.2. Správne výsledky i pre $k = 0$ je možné dosiahnuť otestovaním prvého stavu, pre ktorý musí platiť, že reprezentuje nejakú hranicu reťazca x.

Príklad 3.2. Pre vstupný reťazec $x = aba$ a Hammingovu vzdialenosť $k = 0$ vrátia algoritmy 2.2 a 2.1 prázdnu množinu. Pre takýto vstup je však približným rozšíreným pokrytím, a takisto aj uvoľneným približným rozšíreným pokrytím, reťazec a.

Zistiť, či stav q deterministického sufixového automatu je hranica, je možné sériou porovnaní. Musí platiť, že hĺbka prvého prvku d-podmnožiny je rovná hĺbke stavu q, hĺbka posledného prvku d-podmnožiny je rovná dĺžke vstupného reťazca a úrovne oboch prvkov musia byť rovné nule. Vzhľadom na to, že nemá cenu brať do úvahy približných rozšírených pokrytí hranicu, ktorej dĺžka je menšia ako k, tak musí tiež platiť, že hĺbka stavu q je väčšia ako k.

Konštrukcia prvého stavu je pre oba algoritmy podobná. V algoritme na hľadanie približných rozšírených pokrytí sa porovnáva prvý symbol reťazca so všetkými symbolmi reťazca. V algoritme na hľadanie uvoľnených približných rozšírených pokrytí sa porovnáva symbol abecedy so všetkými symbolmi reťazca. Rozšírením funkcie pre výpočet prvého stavu je tak možné použitie v oboch algoritmoch. Pseudokód upravenej funkcie je uvedený ako funkcia 6.

Podobne, ako konštrukciu prvého stavu, je možné i konštrukciu nasledujúceho stavu rozšíriť tak, aby sa dala použiť pre výpočet uvoľnených približných rozšírených pokrytí. Pseudokód je uvedený ako funkcia 7.

Použitím upravených funkcií vznikne upravený algoritmus 3.1 na počítanie všetkých k-približných rozšírených pokrytí.

Popísanou úpravou funkcií na vytváranie stavov je možné zjednodušiť rekurzívnu funkciu používanú algoritmom na počítanie uvoľnených približných rozšírených pokrytí. Pseudokód je uvedený ako funkcia 8.

Funkcia 6 constrFirstState2

Vstup: vstupný reťazec x, maximálna Hammingova vzdialenosť k, symbol a
Výstup: prvý stav q_1 deterministického k-približného sufixového automatu pre reťazec x

```
q_1 je nový stav
for i ∈ 1..|x| do
  e je prvok d-podmnožiny, taký že hĺbka(e) ← i
  if a = x[i] then
    úroveň(e) ← 0
    pridaj e do d-podmnožiny stavu q_1
  else if k > 0 then
    úroveň(e) ← 1
    pridaj e do d-podmnožiny stavu q_1
  end if
end for
return q_1
```

Funkcia 7 constrNextState2

Vstup: vstupný reťazec x, Hammingova vzdialenosť k, stav q_p, symbol a
Výstup: nasledujúci stav q_n deterministického k-približného sufixového automatu pre x

```
q_n je nový stav
for e_p ∈ q_p do
  if hĺbka(e_p) < |x| then
    e_n je prvok d-podmnožiny, taký že hĺbka(e_n) ← hĺbka(e_p) + 1
    if a = x[hĺbka(e_n)] then
      úroveň(e_n) ← úroveň(e_p)
      pridaj e_n do d-podmnožiny stavu q_n
    else if úroveň(e_p) < k then
      úroveň(e_n) ← úroveň(e_p) + 1
      pridaj e_n do d-podmnožiny stavu q_1
    end if
  end if
end for
return q_n
```

Finálna úprava sa týka samotného algoritmu na počítanie uvoľnených približných rozšírených pokrytí. Pred zavolaním rekurzívnej funkcie 8 sa stav vytvorený funkciou 6 skontroluje, či reprezentuje hranicu. Výsledný algoritmus je uvedený ako 3.2.

Vo všetkých prípadoch sa jedná o úpravy, ktoré je možné vyhodnotiť v konštantnom čase. Časová a pamäťová zložitosť sú tak po úprave nezmenené.

Algoritmus 3.1 Upravený algoritmus na počítanie všetkých k-približných rozšírených pokrytí

Vstup: reťazec x, maximálna Hammingova vzdialenosť k
Výstup: množina všetkých k-približných rozšírených pokrytí

```
C ← ∅
h ← 0
q_1 ← constrFirstState2(x, k, x[1])
if q_1 reprezentuje hranicu then
  (C, h) ← updateEnhCov(q_1, C, h)
end if
p ← 1
q_p ← q_1
for i ∈ 2..|x| do
  q_n ← constrNextState2(x, k, q_p, x[i])
  p ← p + 1
  odstráň q_p z pamäte
  if počet prvkov d-podmnožiny stavu q_n je menší ako 2 then
    break
  end if
  e_f je posledný prvok d-podmnožiny stavu q_n
  if hĺbka(e_f) = |x| and úroveň(e_f) = 0 and p > k then
    (C, h) ← updateEnhCov(q_n, C, h)
  end if
end for
return C
```

3.3 Implementácia do ALT

Od d-podmnožiny stavu sa vyžaduje prístup k jej prvkom v konštantnom čase. Knižnica ALT v súčasnej dobe neponúka štruktúru deterministického k-približného sufixového automatu pre Hammingovu vzdialenosť, ktorý je možné konštruovať postupne a zároveň je možné pristúpiť k d-podmnožine ľubovoľného stavu. Z toho dôvodu boli vytvorené štruktúry `Element` a `State`.

Štruktúra `Element` reprezentuje prvok d-podmnožiny. Obsahuje preto dve členské premenné, a to `depth` pre hĺbku a `level` pre úroveň.

Stav deterministického k-približného sufixového automatu je reprezentovaný štruktúrou `State`. Štruktúra obsahuje premennú `depth` pre hĺbku stavu, dvojicu čísel nazývanú `lfactor` pre ľavý faktor a pole objektov `elements` typu `Element`, ktoré predstavuje d-podmnožinu daného stavu. V každom stave sa udržuje invariant, že prvky poľa `elements` sú zoradené podľa ich hĺbky vzostupne.

Spoločné funkcie, ktoré využívajú oba algoritmy, sú definované v abstraktnej triede `ApproximateEnhancedCoversCommon`. V tejto triede sú taktiež uve-

Funkcia 8 processState2

Vstup: vstupný reťazec x, maximálna Hammingova vzdialenosť k, stav q_i čiastočne skonštruovaného $M = (Q, A, \delta, q_0, F)$, množina uvoľnených približných rozšírených pokrytí C, maximálny počet pokrytých pozícií h

```
for a ∈ A do
  q ← constrNextState2(x, q_i, k, a)
  e_x je nedefinovaný prvok
  úroveň(q) ← úroveň(q_i) + 1
  E je d-podmnožina stavu q
  for e ∈ E do
    if úroveň(e) = 0 then
      e_x ← e
    end if
  end for
  e_1 ∈ E je prvý prvok d-podmnožiny stavu q
  if |E| > 1 and hĺbka(e_1) = hĺbka(q) and e_x je definovaný then
    e_l ∈ E je posledný prvok d-podmnožiny stavu q
    if hĺbka(e_l) = |x| and úroveň(q) > k then
      (C, h) ← updateEnhCov(q, C, h)
    end if
    processState2(x, k, q, C, h)
  end if
  odstráň stav q z pamäte
end for
```

dené definície štruktúr `Element` a `State`. Všetky funkcie a štruktúry majú atribút `protected`, sú teda prístupné iba pre triedy, ktoré sú potomkami triedy `ApproximateEnhancedCoversCommon`.

Algoritmus na počítanie približných rozšírených pokrytí je implementovaný v triede `ApproximateEnhancedCoversComputation`, ktorá dedí z triedy `ApproximateEnhancedCoversCommon`.

Algoritmus na počítanie uvoľnených približných rozšírených pokrytí je implementovaný v triede `RelaxedApproximateEnhancedCoversComputation`, ktorá taktiež dedí z triedy `ApproximateEnhancedCoversCommon`. V tejto triede je navyše napísaná definícia funkcie 8, ktorú využíva exkluzívne algoritmus 3.2.

Obe triedy implementujú metódu `compute`, ktorá na vstupe očakáva reťazec x a Hammingovu vzdialenosť k, a vráti príslušnú množinu približných rozšírených pokrytí, resp. množinu uvoľnených približných rozšírených pokrytí. Metóda `compute` je tak jedinou verejnou metódou.

Pre prístup k algoritmom z terminálu prostredníctvom programu `aql2`, je nutné algoritmy zaregistrovať. Registrácia je spravená prostredníctvom triedy `registration::AbstractRegister`.

Všetky funkcie, v ktorých sa pracuje so vstupným reťazcom, obsahujú šab-

Algoritmus 3.2 Upravený algoritmus na počítanie všetkých uvoľnených k-približných rozšírených pokrytí

Vstup: reťazec $x \in A^*$, maximálna Hammingova vzdialenosť k
Výstup: množina všetkých uvoľnených k-približných rozšírených pokrytí

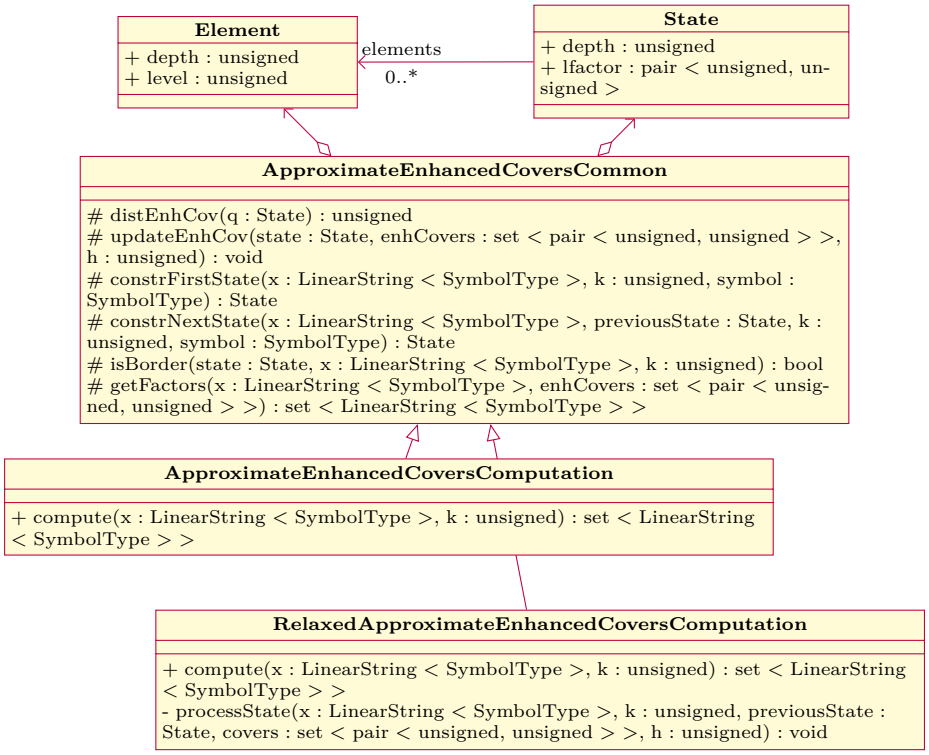
```
C ← ∅
h ← 0
for a ∈ A do
  q_a ← constrFirstState2(x, k, a)
  if q_a reprezentuje hranicu then
    (C, h) ← updateEnhCov(q_a, C, h)
  end if
  processState2(x, k, q_a)
  odstráň q_a z pamäte
end for
return C
```

lónu `SymbolType`. Vďaka tomu je možné počítať približné rozšírené pokrytia a uvoľnené približné rozšírené pokrytia i pre reťazce, ktoré sú tvorené z iných symbolov, než zo symbolov typu `char`. Pre správnu funkciu je ale nutné, aby bolo možné symboly vzájomne porovnávať.

V [1] sa výsledná množina nájdených približných rozšírených pokrytí, resp. uvoľnených približných rozšírených pokrytí vráti ako množina čísel, resp. množina dvojíc čísel. Implementácia v knižnici algoritmov ale vráti výsledok ako množinu reťazcov. Všetkých faktorov je celkom $\mathcal{O}(n^2)$, ale k-približných faktorov je ešte viac. To má za následok zhoršenie pamäťovej zložitosti oboch algoritmov.

Všetky triedy sú uložené v mennom priestore `stringology::cover`, čím je zabránené možným kolíziám.

Popísané štruktúry a triedy sú zobrazené na obrázku 3.1, kde sú taktiež vyznačené dôležité vzťahy medzi nimi.



Obr. 3.1: Diagram tried použitých na implementáciu algoritmov na hľadanie približných rozšírených pokrytí a uvoľnených približných rozšírených pokrytí.

Kapitola 4

Testovanie

V tejto kapitole je popísané, aké spôsoby boli použité na otestovanie správnosti implementovaných algoritmov. Možných vstupov je nekonečne veľa, a preto žiadne reálne testovanie nedokáže zaručiť správnosť riešenia, ale môže pomôcť odhaliť chyby.

Kapitola je rozdelená na tri časti, podľa použitých spôsobov testovania.

4.1 Jednotkové testy

Základným spôsobom testovania sú jednotkové testy. Tieto testy sú súčasťou implementácie v knižnici algoritmov [3].

Testy približných rozšírených pokrytí a uvoľnených približných rozšírených pokrytí sú v zásade rovnaké – líšia sa iba vstupom a výstupom. Medzi testované vstupy patria: jednoduchý reťazec s jedným (uvoľneným) približným rozšíreným pokrytím, reťazec z príkladu v [1], prázdny reťazec, reťazec s Hammingovou vzdialenosťou 0 (korešponduje s hľadaním presných rozšírených pokrytí), a reťazec s rôznymi hodnotami Hammingovej vzdialenosti. Konkrétne vstupy sú uvedené v tabuľke 4.1 pre približné rozšírené pokrytia, resp. v tabuľke 4.2 pre uvoľnené približné rozšírené pokrytia.

Jednotkové testy je možné spustiť z adresára `build` príkazom `make test`.

4.2 Náhodné dáta

Test náhodnými dátami sa skladá z dvoch častí – naivného referenčného riešenia a testovacieho skriptu.

Naivné riešenie je napísané v jazyku `C++` a má dva povinné parametre a jeden voliteľný parameter, ktoré sú programu predané z príkazovej riadky. Prvým parametrom je vstupný reťazec, ďalším je maximálna Hammingova vzdialenosť. Nepovinný parameter musí mať tvar buď `-r` alebo `--relaxed` a značí výber počítania uvoľnených k-približných rozšírených pokrytí.

Číslo testu	Vstupný reťazec	Hammingova vzdialenosť	Očakávaný výstup
1	abacaccababa	1	{aba}
2	abababaccababa	2	{aba, ababa}
3	ϵ	0 1 15 -1	$\emptyset$
4	x ab aa aba aaa	0	$\emptyset$ $\emptyset$ {a} {a} {a, aa}
5	adesgvsade adesgvsade abcd	3 2 1	$\emptyset$ {ade} $\emptyset$

Tabuľka 4.1: Prehľad testovaných vstupov pre približné rozšírené pokrytia.

Číslo testu	Vstupný reťazec	Hammingova vzdialenosť	Očakávaný výstup
1	abcabc	1	{abc}
2	abacaccababa	2	{cca, aba, aca, acc, abaca, ababa}
3	ϵ	0 1 15 -1	$\emptyset$
4	x ab aa aba aaa	0	$\emptyset$ $\emptyset$ {a} {a} {a, aa}
5	abbbabab	4 3	{bbaba, bbbaba, babab, bbbab, abbbab, abbba, bbabab} {abbb, bbba, bbab, bbbab, abab, bbabab, abbbab}

Tabuľka 4.2: Prehľad testovaných vstupov pre uvoľnené približné rozšírené pokrytia.

Testovací skript je napísaný v jazyku `Bash` a je ovládaný dvomi parametrami – maximálnou dĺžkou reťazca `MAX_LEN` a počtom iterácii `MAX_ITER`. Pre každú dĺžku $l \in \{1, \ldots, \texttt{MAX\_LEN}\}$ prebieha `MAX_ITER` iterácii. V každej iterácii sa vygeneruje náhodná veľkosť abecedy a následne sa vygeneruje náhodný reťazec dĺžky l spolu s náhodnou Hammingovou vzdialenosťou. Tento náhodne vygenerovaný reťazec a náhodná Hammingova vzdialenosť sú použité ako vstup pre testované riešenie v `aql2` a naivné riešenie. Výstupy z oboch programov sú porovnané a v prípade, že sú rozdielne, tak testovací skript skončí s chybou. Maximálna dĺžka reťazca tak označuje zároveň počet testov.

Pre preklad naivného riešenia je v skripte použitý prekladač `g++`. Jednoduchou úpravou skriptu je ale možné nastaviť preklad iným prekladačom (prípadne preklad iného zdrojového súboru).

Testovací skript je možné spustiť bez parametrov, ale okrem toho ponúka tri prepínače:

- `-h` zobrazí krátku nápovedu
- `-i NUM` nastaví počet iterácii na `NUM`
- `-l NUM` nastaví maximálnu dĺžku reťazca na `NUM`

Predvolená hodnota pre `MAX_LEN` je 100 a pre `MAX_ITER` je 10.

Spustiť testovanie je možné príkazom `./test.sh` z adresára, v ktorom sa testovací skript nachádza. Pred prvým spustením je nutné nastaviť právo na spustenie príkazom `chmod u+x test.sh`. Skript očakáva, že je nainštalovaný príkaz `aql2`.

Svojou podstatou je pri takmer každom spustení testovacieho skriptu vygenerovaná iná sada testov než pri predošlých spusteniach. Efektívnosť testovania ale nie je optimálna – už pri malých hodnotách počtu testov a počtu iterácii trvá testovanie pomerne dlho.

Poznámka 4.1. Na počítači s procesorom `Intel Pentium N3530` a operačným systémom `Lubuntu 19.04`, bol skript spustený päťkrát s predvolenými parametrami a priemerná doba trvania bola 7 minút 34 sekúnd.

4.3 Experimentálne výsledky

Táto podkapitola sa zaoberá experimentálnymi testami, s cieľom porovnať výsledky s [1]. Ako vstupný reťazec bol použitý rovnaký vstup, a to chromozóm kvasinky pivnej [8]. Konkrétne bolo použitých prvých n písmen z daného chromozómu.

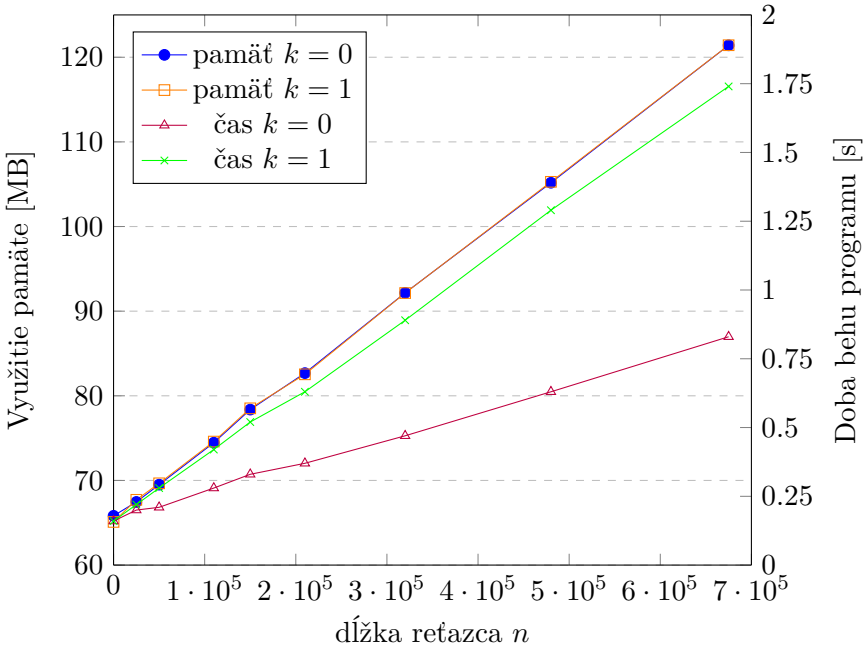
Testovanie bolo spustené na počítači s procesorom `Intel Pentium N3530` a operačným systémom `Lubuntu 19.04`. Maximálne množstvo použitej pamäte a celkový čas v režime `user` boli namerané pomocou programu `GNU Time` dostupného v [9].

Namerané výsledky sú zaznamenané v nasledujúcich grafoch. V každom grafe je na ľavej osi uvedené maximálne množstvo použitej pamäte v megabajtoch a na pravej osi je celková doba behu programu v sekundách.

4.3.1 Približné rozšírené pokrytie

Graf závislosti času a pamäte od dĺžky vstupného reťazca pri fixnej hodnote Hammingovej vzdialenosti k je na obrázku 4.1. Výsledky sú porovnateľné s tými, ktoré sú uvedené v [1].

Krivka, ktorá znázorňuje využitie pamäte pre $k = 0$, splýva s krivkou, ktorá znázorňuje využitie pamäte pre $k = 1$.



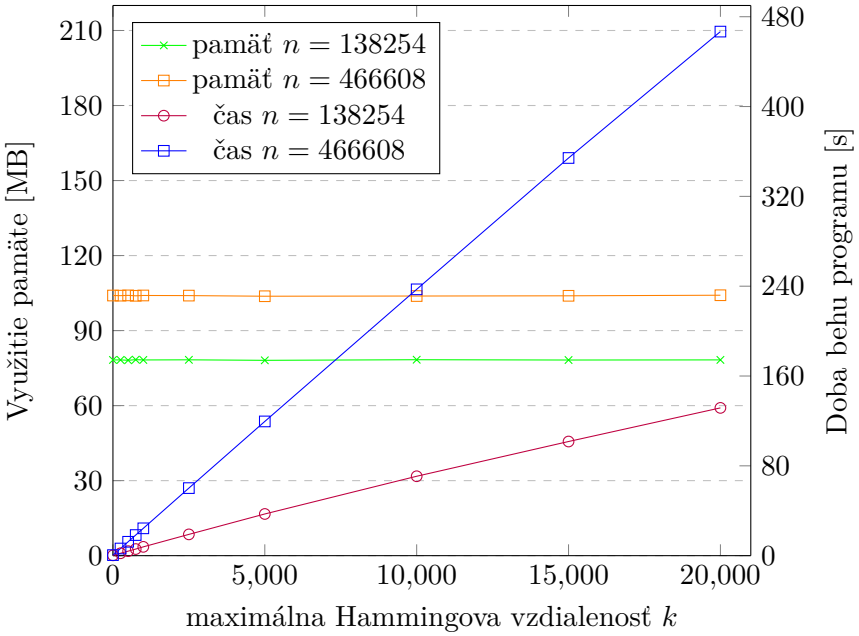
Obr. 4.1: Graf závislosti času a pamäte od dĺžky vstupného reťazca pri fixnej maximálnej Hammingovej vzdialenosti počas počítania približných rozšírených pokrytí.

Na obrázku 4.2 je znázornený graf závislosti času a pamäte od maximálnej Hammingovej vzdialenosti pri fixnej dĺžke vstupného reťazca.

Hammingova vzdialenosť neovplyvňuje množstvo alokovanej pamäte. Čas rastie lineárne, čo je spôsobené väčším množstvom prvkov v d-podmnožine.

4.3.2 Uvoľnené približné rozšírené pokrytie

Na obrázku 4.3 je znázornený graf, kde je fixne daná dĺžka vstupného reťazca a meria sa závislosť na maximálnej Hammingovej vzdialenosti k. Využite pa-



Obr. 4.2: Graf závislosti času a pamäte od maximálnej Hammingovej vzdialenosti pri fixnej dĺžke vstupného reťazca počas počítania približných rozšírených pokrytí.

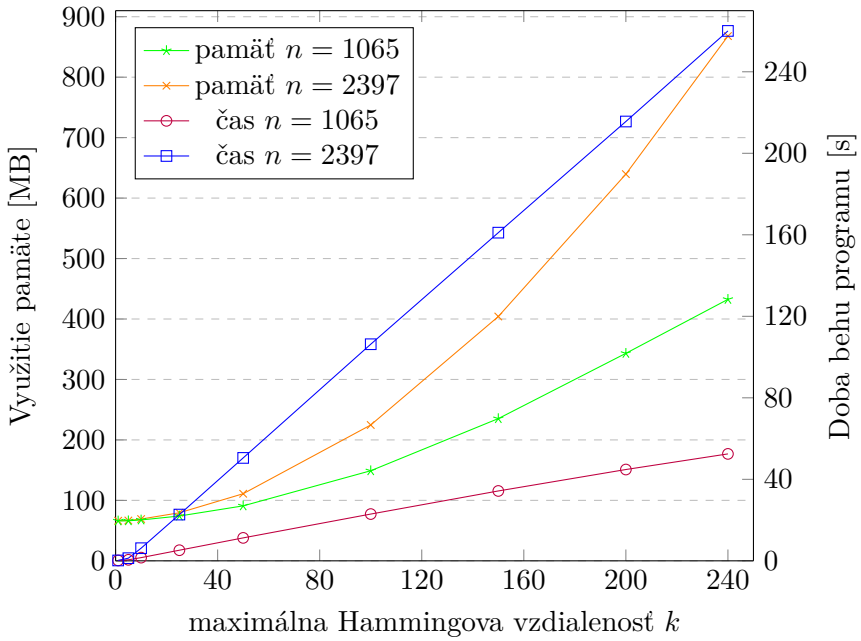
mäte sa pre rôzne k líši, napriek tomu, že pamäťová zložitosť nie je závislá na Hammingovej vzdialenosti. To je ale podľa [1] spôsobené tým, že k obmedzuje počet prvkov v d-podmnožine.

Trend krivky pre pamäť sa ale líši od trendu v [1]. Dôvod je ten, že implementácia v ALT vracia množinu nájdených uvoľnených rozšírených pokrytí ako samostatné reťazce, a nie ako dvojicu čísel.

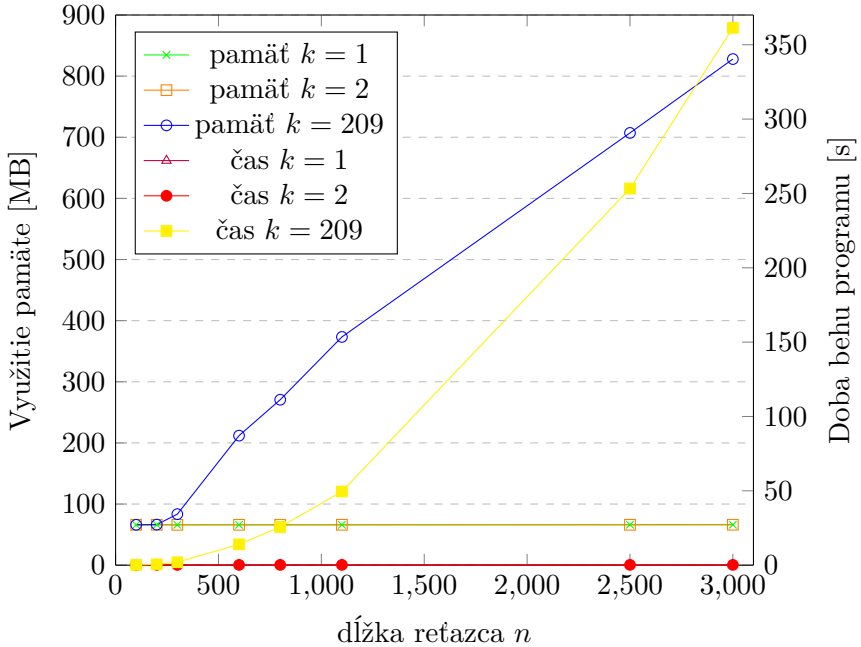
Závislosť času na maximálnej Hammingovej vzdialenosti sa pre fixný reťazec správa lineárne.

Obrázok 4.4 obsahuje graf, kde je pre zmenu zafixovaná Hammingova vzdialenosť k a líši sa počet symbolov na vstupe.

Oproti [1] je testovaných menej symbolov na vstupe. Už pri malých hodnotách dĺžky vstupu je však vidieť veľký rozdiel v pamäti, ktorý je daný rozdielnym tvarom návratovej množiny.



Obr. 4.3: Graf závislosti času a pamäte od maximálnej Hammingovej vzdialenosti pri fixnej dĺžke vstupného reťazca počas počítania uvoľnených približných rozšírených pokrytí.



Obr. 4.4: Graf závislosti času a pamäte od dĺžky vstupného reťazca pri fixnej maximálnej Hammingovej vzdialenosti počas počítania uvoľnených približných rozšírených pokrytí.

Záver

Primárnym cieľom tejto práce bol popis algoritmov uvedených v [1] a následná implementácia do knižnice algoritmov [3]. Popisu algoritmov sa venuje kapitola 2. Implementácia je rozpísaná v kapitole 3.

Ďalším cieľom bolo dôkladné testovanie – tým sa zaoberá kapitola 4. Pre potreby testovania boli použité jednotkové testy a testy na náhodných dátach.

V práci sú uvedené algoritmy, ktoré fungujú na základe podmnožinovej konštrukcie sufixových konečných automatov. Jedným z možných rozšírení tejto práce je implementácia iných než automatových algoritmov. Namiesto Hammingovej vzdialenosti, ktorá je použitá pre počítanie približnosti reťazcov, je možné použiť inú metriku (napríklad Levenshteinovu vzdialenosť). Ďalším možným rozšírením je sprístupnenie algoritmov cez grafické rozhranie. V neposlednom rade, sa ponúka úprava formy návratovej množiny tak, aby bola pamäťovo efektívnejšia.

Literatúra

[1] Guth, O.: On approximate enhanced covers under Hamming distance. *Discrete Applied Mathematics*, ročník 274, 2020: s. 67 – 80, ISSN 0166-218X, doi:https://doi.org/10.1016/j.dam.2019.01.015, [cit. 2020-05-25]. Dostupné z: `http://www.sciencedirect.com/science/article/pii/S0166218X19300320`

[2] Guth, O.: Computing All Approximate Enhanced Covers with the Hamming Distance. In *Proceedings of the Prague Stringology Conference 2016*, editace J. H. a Jan Žďárek, České vysoké učení technické v Praze, Česká republika, 2016, ISBN 978-80-01-05996-8, s. 146–157.

[3] Algorithms Library Toolkit [software]. [cit. 2020-05-20]. Dostupné z: `https://gitlab.fit.cvut.cz/algorithms-library-toolkit`

[4] Guth, O.: *Searching regularities in strings using finite automata*. Dizertační práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2014.

[5] Melichar, B.; Holub, J.; Polcar, T.: Text Searching Algorithms [online]. 2005, [cit. 2020-05-12]. Dostupné z: `http://www.stringology.org/athens/TextSearchingAlgorithms/tsa-lectures-1.pdf`

[6] Žák, M.: *Automatová knihovna – vnitřní a komunikační formát*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2014.

[7] GNU General Public License. Dostupné z: `http://www.gnu.org/licenses/gpl.html`

[8] Saccharomyces cerevisiae S288C chromosome IV, complete sequence [online]. [cit. 2020-06-02]. Dostupné z: `https://www.ncbi.nlm.nih.gov/nuccore/NC%5F001136.10`

[9] GNU Time [software]. [cit. 2020-06-02]. Dostupné z: https://www.gnu.org/software/time/

Dodatok **A**

Zoznam použitých skratiek

ALT Algorithms Library Toolkit

DKA deterministický konečný automat

NKA nedeterministický konečný automat

Dodatok B

Ukážka použitia

Program `aql2` môže byť spustený v dvoch módoch, a to buď interaktívne alebo neinteraktívne.

Pre spustenie interaktívneho vyhodnocovania, je potrebné najprv program `aql2` spustiť z príkazovej riadky. Nasledujúci kód spustí program `aql2` z terminálu a následne vypíše množinu 2-približných rozšírených pokrytí pre reťazec *abacaccababa*. Po vypísaní tejto množiny sa opustí prostredie programu `aql2`.

```
$ aql2
> print stringology::cover::ApproximateEnhancedCoversComputation
↪  "abacaccababa" 2
> exit
```

Neinteraktívne vyhodnocovanie je možné dosiahnuť prostredníctvom prepínača `-c`. Nasledujúci príkaz vypíše rovnakú množinu pre rovnaký vstup.

```
$ aql2 -c "print
↪  stringology::cover::ApproximateEnhancedCoversComputation
↪  abacaccababa 2"
```

Dodatok C

Obsah priloženého USB

- `readme.txt` ..... stručný popis obsahu USB
- `automata-library` ..... adresár so zdrojovými kódmi knižnice algoritmov
- `test`
  - `naive.cpp` ..... zdrojový kód naivného riešenia
  - `test.sh` ..... testovací skript
- `text`
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
  - `BP-hruskraj.pdf` ..... text práce vo formáte PDF