FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Implementace vyhledávání v řetězcích pomocí kompaktního suffixového automatu
Student:	Josef Erik Sedláček
Vedoucí:	Ing. Jan Trávníček
Studijní program:	Informatika
Studijní obor:	Teoretická informatika
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Nastudujte návrh a implementaci algoritmu vyhledávání v řetězcích pomocí kompaktního suffixového automatu.
Navrhněte a implementujte datovou strukturu pro reprezentaci kompaktního suffixového automatu.
Implementujte algoritmus konstrukce a algoritmus dotazování se do kompaktního suffixového automatu.
Zvolte vhodné množiny řetězců pro experimenty. Proveďte měření rychlosti běhu implementovaného algoritmu a porovnejte ji s již implementovanými algoritmy vyhledávání v řetězcích z Knihovny algoritmů vyvíjené na Katedře teoretické informatiky.

Seznam odborné literatury

Dodá vedoucí práce.

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

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

V Praze dne 30. prosince 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Implementace vyhledávání v řetězcích pomocí kompaktního suffixového automatu

Josef Erik Sedláček

Katedra Teoretické Informatiky
Vedoucí práce: Ing. Jan Trávníček

9. května 2018

Poděkování

Chtěl bych velmi poděkovat svému vedoucímu Ing. Janu Trávníčkovi, že mi vždy poradil a pomohl, a že to se mnou nevzdal, přestože jsem tak neschopný člověk.

Dále bych chtěl poděkovat svým spolužákům, kteří mi pomohli projít těmi třemi roky studia nebo mi poradili se samotnou bakalářskou prací byť třeba jen s tím, jak se přeloží zdrojový kód LaTeXu. Jmenovitě děkuji Matyáši Křišťanovi, Miroslavu Sochorovi, Janu Uhlíkovi, Nikole Nguyenové, Michalu Cvachovi, Honzovi Vuovi, Martinu Bobkovi a Tungu Vuovi.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 9. května 2018 ……

Odkaz na tuto práci

Sedláček, Josef Erik. *Implementace vyhledávání v řetězcích pomocí kompaktního suffixového automatu*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Práce se zabývá problémem rozhodnout, jestli pro zadaná slova w a u platí, že u je podslovem w a případně vrátit množinu pozic, kde se u nachází. Konkrétně se zabývá přímou konstrukcí kompaktního suffixového automatu, který dokáže problém rozhodnout s lineární časovou i paměťovou složitostí vzhledem k délce slova w. Výsledkem je implementace algoritmu pro konstrukci automatu v jazyce C++ do Algoritmové knihovny ALIB.

Klíčová slova Hledání v řetězcích, Kompaktní suffixový automat, Suffixový strom, Datové struktury, Trie, Řetězce

Abstract

The thesis is concerned with the problem of deciding, whether it is true, given input words w and u, that u is a substring of w, and eventually outputting a set of positions where u is found in w. Specifically, the content of this thesis deals with a direct construction of the compact suffix automaton, which can decide the problem in linear time and space in relation to the length of the word w. The result is an implementation in C++ of an algorithm for constructing the automaton as a part the Algorithms Library ALIB.

Keywords Pattern matching on strings, Compact directed acyclic word graph, Suffix tree, Data structures, Trie, Strings

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V problémech vyhledávání v textu se zjišťuje, jestli pro slova w a u platí, že u je podslovem w. Často je slovo w dotazováno opakovaně, takže je výhodné sestavit datovou strukturu nad zadaným slovem. Suffixový strom je vhodnou strukturou pro rozhodnutí takového problému, a proto byl podrobně studován a upravován. Další takovouto strukturou je suffixový automat.

Praktické využití je například v genetice, kde je možné DNA reprezentovat jako slova nad abecedou s množinou znaků {a, c, g, t}. Databáze jsou stále větší a hledání v nich zabírá více času i místa a je tedy důležité mít efektivní datovou strukturu, která dokáže problém řešit rychle s co nejmenším množstvím vyžadované paměti. Komprese je typicky v takovémto případě nepoužitelná, protože se s ní ztrácí přímý přístup k podřetězcům. Na rozdíl od toho si kompaktní suffixový automat udržuje přímý přístup k podřetězcům a zároveň minimalizuje potřebné množství paměti. Takovýto automat lze získat kompakcí suffixového automatu, tedy odstraněním všech stavů s výstupním stupněm 1 a spojením jejich přechodů. Ekvivalentním postupem je minimalizace suffixového stromu. I když se asymptoticky oproti suffixovému stromu paměťová složitost nezmění, prakticky se ušetří až 2/3 stavů a polovina přechodů [1].

Tato práce se zabývá přímou konstrukcí kompaktního suffixového automatu, která, oproti nepřímé, šetří pamětí už při samotné konstrukci.

Cíl práce

Cílem této bakalářské práce je nastudování několika algoritmů pro přímou konstrukci suffixového automatu. Následně je jeden z algoritmů naimplementován v jazyce C++ do algoritmové knihovny ALIB a testování jeho efektivity.

Struktura práce

Kapitola 1 pokrývá základní definice a značení, které jsou potřeba k pochopení dalších částí práce a samotného algoritmu. V kapitole 2 je nejprve zmíněno několik jednoduchých úvah, jak jednoduše kompaktní suffixový automat zkonstruovat a následně je popsán samotný algoritmus pro jeho přímou konstrukci. V kapitole 4 popisuje konkrétní implementaci zmíněného algoritmu v algoritmové knihovně. Nakonec kapitola 5 porovnává implementace různých indexových struktur a za touto kapitolou následuje závěr.

Kapitola 1

Základní definice

Kapitola shrnuje základní pojmy z teorie automatů. Formální definice převzaty z [2] a [3]

1.1 Slova

Abeceda značená Σ je konečná množina znaků.

Slovo (též řetězec) je konečná posloupnost znaků. Σ^* značí množinu všech slov nad Σ. Prázdné slovo se značí ε. Množina slov bez prázdného slova je znázorněna Σ^+. Formálně tedy $\Sigma^+ = \Sigma^* - \{\varepsilon\}$.

Zřetězení: Nechť x a y jsou dvě slova z Σ^*. Připojení slova y za slovo x se nazývá *zřetězení* a značí se zapsáním slov za sebe: xy.

Operace zřetězení

- je asociativní - $\forall x, y, z \in \Sigma^* : (xy)z = x(yz)$,
- není komutativní - $\exists x, y \in \Sigma^* : xy \neq yx$,
- ε se pro ni chová jako neutrální prvek - $\forall x \in \Sigma^* : \varepsilon x = x\varepsilon = x$.

Délka slova: Nechť w je nějaké slovo, pak $|w|$ značí délku w. Platí $|\varepsilon| = 0$.

Předpona, přípona, podslovo: Nechť $x, y, z \in \Sigma^*$ a $w = xyz$. Pak se x nazývá předponou, y podslovem (též faktorem) a z příponou slova w. Množina všech předpon w bude značena $P(w)$, faktorů $F(w)$ a přípon $S(w)$.

i-tý znak slova w se značí $w[i]$ pro $1 \leq i \leq |w|$.

Podslovo určitého slova w začínající na i-tém a končící na j-tém znaku se označuje $w[i : j]$ pro $1 \leq i \leq j \leq |w|$. V případě, že $j < i$, pak $w[i : j] = \varepsilon$.

Množina slov: Pro nějakou množinu M slov $w_1, w_2, ..., w_\ell \in \Sigma^*$ platí, že $|M|$ značí počet slov v dané množině a $\|M\|$ značí součet délek všech jejích slov. Množina předpon, faktorů a přípon všech slov v M se značí $P(M), F(M)$ a $S(M)$. Libovolná podmnožina slov $P \subseteq \Sigma^*$ se též nazývá *formální jazyk*.

Pravý a levý kontext: Nechť $S \subseteq \Sigma^*$ a $x \in \Sigma^*$. $Sx^{-1} = \{u | ux \in S\}$ a $x^{-1}S = \{u | xu \in S\}$.

Toto lze chápat, jako že v operaci zřetězení existuje nějaký inverzní prvek w^{-1} ke každému slovu w takový, že $ww^{-1} = \varepsilon$.

1.2 Konečný automat

Končený automat je výpočetní model skládající se ze stavů a přechodů. Automat sestrojený pro množinu slov S poté dokáže přijmout slovo w a rozhodnout, jestli dané slovo w je v množině S.

1.2.1 Deterministický konečný automat

Deterministický konečný automat (DKA) je pětice $(Q, \Sigma, \delta, q_0, F)$

- Q je konečná množina stavů automatu.
- Σ je konečná vstupní abeceda.
- δ je zobrazení z $Q \times \Sigma \to Q$.
- $q_0 \in Q$ je počáteční stav.
- $F \subseteq Q$ je množina koncových stavů.

Konfigurace automatu: Nechť $M = (Q, \Sigma, \delta, q_0, F)$ je konečný automat. Dvojici $(q, w) \in Q \times \Sigma^*$ nazveme konfigurací konečného automatu M.

Přechod automatu: Nechť $M = (Q, \Sigma, \delta, q_0, F)$ je deterministický konečný automat a nechť $\vdash_M$ je relace nad $Q \times \Sigma^*$ taková, že $(q, aw) \vdash (p, w)$ právě tehdy, když $\delta(q, a) = p$, kde $q, p \in Q$, $a \in \Sigma$ a $w \in \Sigma^*$. Prvek $\vdash_M$ se nazývá přechodem v automatu M. $\vdash_M^*$ tranzitivní a reflexivní uzávěr relace $\vdash_M$.

1.2.2 Nedeterministický konečný automat

Nedeterministický konečný automat (NKA) je pětice $(Q, \Sigma, \delta, q_0, F)$

- Q je konečná množina stavů automatu.
- Σ je konečná vstupní abeceda.
- δ je zobrazení z $Q \times \Sigma \to 2^Q$, kde 2^Q značí potenční množinu Q

- $q_0 \in Q$ je počáteční stav.
- $F \subseteq Q$ je množina koncových stavů.

Konfigurace nedeterministického konečného automatu odpovídá konfiguraci deterministického.

Přechod automatu: Nechť $M = (Q, \Sigma, \delta, q_0, F)$ je deterministický konečný automat a nechť $\vdash_M$ je relace nad $Q \times \Sigma^*$ taková, že $(q, aw) \vdash (p, w)$ právě tehdy, když $p \in \delta(q, a)$, kde $q, p \in Q$, $a \in \Sigma$ a $w \in \Sigma^*$. Prvek $\vdash_M$ se nazývá přechodem v automatu M.

Přijetí slova: Nechť $M = (Q, \Sigma, \delta, q_0, F)$ je konečný automat (NKA nebo DKA) a $w \in \Sigma^*$ je nějaké slovo. Automat přijímá w právě tehdy, když $\exists (q_0, w) \vdash_M^* (q, \varepsilon)$ pro nějaké $q \in F$.

1.2.3 Kompaktní konečný automat

Zásadní rozdíl oproti DKA a NKA je, že přechod může být na více znaků. Kompaktní končený automat vznikne odstraněním všech stavů, ze kterých vychází právě jeden přechod, kromě stavů koncových a počátečního. Dojde také k nahrazení všech přechodů vedoucích do odstraněných stavů více znakovými přechody.

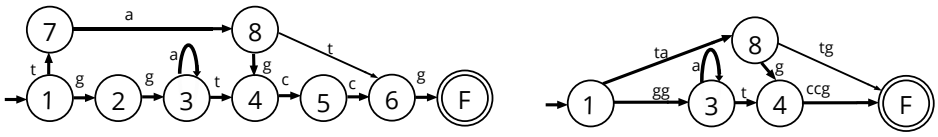
Kompaktní konečný automat (deterministický i nedeterministický) je pětice $(Q, \Sigma, \delta, q_0, F)$

- Q je konečná množina stavů automatu.
- Σ je konečná vstupní abeceda.
- δ je zobrazení z $Q \times \Sigma^+ \to Q$ (respektive 2^Q).
- $q_0 \in Q$ je počáteční stav.
- $F \subseteq Q$ je množina koncových stavů.

Přechod automatu: Nechť $M = (Q, \Sigma, \delta, q_0, F)$ je kompaktní konečný automat a nechť $\vdash_M$ je relace nad $Q \times \Sigma^*$ taková, že $(q, aw) \vdash (p, w)$ právě tehdy, když $\delta(q, a) = p$ (respektive $p \in \delta(q, a)$), kde $q, p \in Q$, $a \in \Sigma^+$ a $w \in \Sigma^*$. Prvek $\vdash_M$ se nazývá přechodem v automatu M.

Z definice je vidět, že automat se liší pouze v přechodové funkci. Jak již bylo zmíněno přechod nemusí být pouze na jeden znak, ale na libovolné neprázdné slovo. Definice konfigurace a přijetí slova jsou totožné s těmi uvedenými výše.

Příklad kompaktního automatu na obrázku 1.1



Obrázek 1.1: Příklad deterministického automatu a jeho kompaktní ekvivalent

1.3 Grafy

Definice z teorie grafů jsou převzaty z [4] a [5].

Neorientovaný graf je dvojice (V, E), kde:

- V je neprázdná konečná množina vrcholů,
- E je množina hran.

Hrana je dvouprvková podmnožina V.

Orientovaný graf je dvojice (V, E), kde:

- V je neprázdná konečná množina vrcholů,
- E je množina orientovaných hran.

Orientovaná hrana $(u, v) \in E$ je uspořádaná dvojice vrcholů $u, v \in V$.
Vrchol u je předchůdce v a v je následník u. Platí $E \subseteq V \times V$.

Stupeň vrcholu v v neorientovaném grafu G značený $deg_G(v)$ je počet hran obsahující vrchol v.

Výstupní stupeň vrcholu v v orientovaném grafu G značený $deg_G^-(v)$ je počet orientovaných hran vycházejících z vrcholu v.

Vstupní stupeň vrcholu v v orientovaném grafu G značený $deg_G^+(v)$ je počet orientovaných hran končících ve vrcholu v.

Zdroj je vrchol v orientovaném grafu, který má $deg_G^+ = 0$.

Stok je vrchol v orientovaném grafu, který má $deg_G^- = 0$.

Cesta je neorientovaný graf $G(V, E)$, kde $V = \{1, \ldots, n\}$ a $E = \{\{i, i+1\} | i \in \{i, \ldots, n-1\}\}$.

Souvislý graf se nazývá neorientovaný graf $G = (V, E)$ právě tehdy, když pro každé dva vrcholy $v, u \in V$ existuje cesta mezi v a u. Jinak je G *nesouvislý.*

Slabě souvislý graf se nazývá orientovaný graf $G = (V, E)$ právě tehdy, když pro každé dva vrcholy $v, u \in V$ existuje neorientovaná cesta mezi v a u.

Silně souvislý je se nazývá orientovaný graf $G = (V, E)$ právě tehdy, když pro každé dva vrcholy $v, u \in V$ existuje cesta z v do u a naopak.

Kružnice je neorientovaný graf $G = (V, E)$, kde $V = \{1, \ldots, n\}$ a $E = \{\{i, i+1\} | i \in \{i, \ldots, n-1\}\} \cup \{\{n, 1\}\}$.

Orientovaná kružnice je graf $G = (V, E)$, kde $V = \{1, \ldots, n\}$ a $E = \{(i, i+1) | i \in \{i, \ldots, n-1\}\} \cup \{(n, 1)\}$.

Kružnice v grafu G je právě tehdy, když nějaký jeho podgraf je izomorfní s kružnicí.

Strom je souvislý graf, ve kterém se nenachází kružnice. Orientovaný graf G je strom, pokud se v něm nenachází kružnice po odstranění orientací všech jeho hran.

Acyklický je orientovaný graf, ve kterém neexistuje orientovaná kružnice.

1.4 Indexové datové struktury

V této části jsou formálně definované čtyři indexové datové struktury. Konkrétně Suffixová trie, strom, automat a kompaktní automat. Tato práce je zaměřená hlavně na kompaktní suffixový automat, ale jak je vidět na obrázku 1.2 všechny vychází z nejjednodušší trie. Z obrázku je zřejmé, že suffixový strom je pouze kompaktní verze trie a suffixový automat lze získat jejím minimalizováním. Dále je vidět, že po provedení obou operací je výsledkem kompaktní suffixový automat. Před nadefinováním indexových struktur je však nejprve třeba uvést ještě několik definic o vztazích mezi řetězci.

Následující definice jsou převzaty z [3].

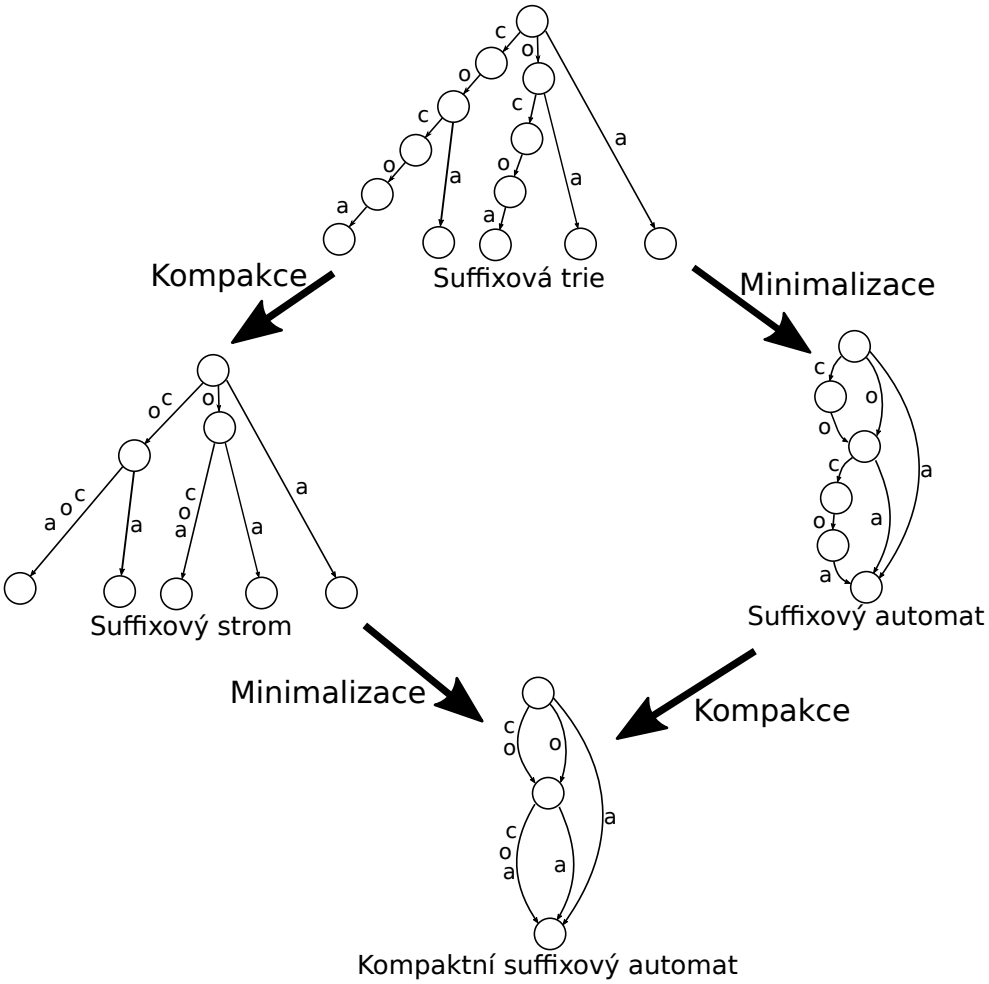
1.4.1 Ekvivalence nad řetězci

Definice 1 *Nechť $w, x, y \in \Sigma^*$. O x a y se řekne, že jsou v ekvivalenci:*

- $x \equiv_w^L y \Leftrightarrow P(w)x^{-1} = P(w)y^{-1}$,
- $x \equiv_w^R y \Leftrightarrow x^{-1}S(w) = y^{-1}S(w)$.

Třídy této ekvivalence se značí $[x]_w^L$, respektive $[x]_w^R$.

Řečeno slovy. Nechť pro nějaké x, y platí $y \equiv_w^L x$. Ekvivalence platí právě tehdy, když se rovnají množiny M a N, kde M vznikne následujícím způsobem: Vezme se množina $P(w)$ tedy množina všech předpon slova w a z ní se vyberou



Obrázek 1.2: Vztah mezi suffixovou trií, stromem, automatem a kompaktním automatem pro slovo *cocoa* [3]

pouze ta slova, která končí na x. Pro každé takové slovo $u = vx$ se do množiny M přidá slovo v. Totožně se vytvoří množina N, kde místo x bude y.

Je vhodné poukázat na to, že ε je vždy ve své vlastní třídě a všechna slova, která nejsou podslovem původního, tvoří také vlastní třídu.

Příklad 1 *Nechť $w = coco$, $x = co$ a $y = c$. Pak $P(w) = \{coco, coc, co, c, \varepsilon\}$. Z $P(w)$ se nyní vyberou všechna slova končící na x a vznikne množina $\{coco, co\}$. Slova končící na y jsou $\{coc, c\}$. Po odstranění x respektive y z konců příslušných množin, vznikne v obou případech množina $\{co, \varepsilon\}$ a platí tedy $x \equiv_w^L y$.*

Tvrzení 1 (Blumer aj. [6]) *Nechť $w \in \Sigma^*$ a $x, y \in F(w)$. Pokud $x \equiv_w^L y$, pak je x předponou y nebo naopak.*

Obdobně pokud $x \equiv_w^R y$, *pak* x *je příponou* y *nebo naopak.*

Definice 2 *Nechť* $x \in F(w)$. $\xrightarrow[x]{w}$ *značí nejdelší řetězec třídy* $[x]_w^L$ *a bude se nazývat* reprezentantem *dané třídy.*

Obdobně $\xleftarrow[x]{w}$ *je reprezentant* $[x]_w^R$.

Tvrzení 2 (Inenaga [7]) *Nechť* $w \in \Sigma^*$. *Pro libovolný řetězec* $x \in F(w)$ *existují unikátní řetězce* $\alpha, \beta \in \Sigma^*$ *takové, že* $\xrightarrow[x]{w} = x\alpha$ *a* $\xleftarrow[x]{w} = \beta x$

Příklad 2 *Nechť* $w = coco$, *pak* $\xrightarrow[\varepsilon]{w} = \varepsilon$, $\xrightarrow[c]{w} = \xrightarrow[co]{w} = co$, $\xrightarrow[coc]{w} = \xrightarrow[coco]{w} = coco$, $\xrightarrow[o]{w} = o$, $\xrightarrow[oc]{w} = \xrightarrow[oco]{w} = oco$.

Příklad 3 *Nechť* $w = coco$, *pak* $\xleftarrow[\varepsilon]{w} = \varepsilon$, $\xleftarrow[c]{w} = c$, $\xleftarrow[o]{w} = \xleftarrow[co]{w} = co$, $\xleftarrow[oc]{w} = \xleftarrow[coc]{=} coc$, $\xleftarrow[oco]{w} = \xleftarrow[coco]{w} = coco$.

Definice 3 *Nechť* $x \in F(w)$. $\xleftrightarrow[x]{w}$ *je slovo* $\beta x \alpha$, *kde* $\alpha, \beta \in \Sigma^*$ *a platí* $\xrightarrow[x]{w} = x\alpha$, $\xleftarrow[x]{w} = \beta x$.

Definice 4 *Nechť* $x, y \in \Sigma^*$. $x \equiv_w y$, *pokud:*

- $x, y \in F(w)$ *a* $\xleftrightarrow[x]{w} = \xleftrightarrow[y]{w}$ *nebo*
- $x, y \notin F(w)$.

  Třídy této ekvivalence jsou značeny $[x]_w$ *a* $\xleftrightarrow[x]{w}$ *je reprezentant třídy. Reprezentant je nejdelší řetězec ve třídě.*

Příklad 4 *Nechť* $w = coco$, *pak* $\xleftrightarrow[\varepsilon]{w} = \varepsilon$, $\xleftrightarrow[c]{w} = \xleftrightarrow[co]{w} = \xleftrightarrow[o]{w} = co$, $\xleftrightarrow[coc]{w} = \xleftrightarrow[coco]{w} = \xleftrightarrow[oc]{w} = \xleftrightarrow[oco]{w} = coco$.

Tvrzení 3 (Inenaga aj.[3]) *Pro jakékoliv slovo* $x \in F(w)$ *platí* $\xleftrightarrow[x]{w} = (\xrightarrow[\xleftarrow[x]{w}]{w}) = (\xleftarrow[\xrightarrow[x]{w}]{w})$.

Nyní už je možné formálně nadefinovat již zmíněné indexové struktury. Struktury jsou definovány jako orientované, acyklické grafy se slovy přiřazeným k jejich hranám. Tedy (V, E), kde $E \subseteq V \times \Sigma^+ \times V$, kde druhá složka hrany reprezentuje k ní přiřazené slovo. Ke každé struktuře se kvůli časové efektivitě při konstrukci zavedou takzvané *suffix linky.*

1.4.2 Suffixová trie

Definice 5 *Suffixová trie je orientovaný strom (V,E), pro který platí:*

- $V = \{x | x \in F(w)\}$,
- $E = \{(x, a, xa) | x, xa \in F(w), a \in \Sigma\}$.

Suffix link $F = \{(ax, x) | x, ax \in F(w), a \in \Sigma\}$.

Neformálně řečeno každý vrchol v trii reprezentuje nějaké podslovo původního slova, pro které byla trie sestavena. Hrana vede vždy mezi z vrcholu u do v na jeden znak a právě tehdy, když podslovo, které reprezentuje v, vznikne zřetězením podslova reprezentující u se znakem a.

Suffix link vede z vrcholu, který reprezentuje nějaké podslovo x, do vrcholu, který reprezentuje podslovo x bez prvního znaku.

1.4.3 Suffixový strom

Definice 6 *Suffixový strom je orientovaný strom (V,E), pro který platí:*

- $V = \{\xrightarrow[x]{w} | x \in F(w)\}$,
- $E = \{(\xrightarrow[x]{w}, a\beta, \xrightarrow[xa]{w}) | x, xa \in F(w), a \in \Sigma, \beta \in \Sigma^*, \xrightarrow[xa]{w} = xa\beta, \xrightarrow[x]{w} \neq \xrightarrow[xa]{w}\}$.

Suffix link $F = \{(\xrightarrow[ax]{w}, \xrightarrow[x]{w}) | x, ax \in F(w), a \in \Sigma, \xrightarrow[ax]{w} = a \xrightarrow[x]{w}\}$.

Vrcholy suffixového stromu jsou podmnožinou vrcholů suffixové trie. To znamená, že pro některé podslova neexistuje vrchol a jsou na hraně. Takovým podslovům se říká, že jsou reprezentována *implicitním* vrcholem. V opačném případě existuje pro dané podslovo *explicitní* vrchol.

Někdo si můžeš povšimnout, že takto nadefinovaný strom může mít vrchol s výstupním stupněm jedna. Například pro slovo *coco*. Pravdou je, že strom jistě nebude mít vrchol s výstupním stupněm jedna pouze v případě, že se poslední znak nevyskytuje nikde jinde v řetězci. Této vlastnosti se bude následně využívat. Pro samotnou konstrukci stromu se bude používat jeho upravená verze, ve které se takové vrcholy nevyskytují nikdy.

Tvrzení 4 (McCreight [8]) *Nechť $w \in \Sigma^*$ a $|w| > 1$. Pak pro suffixový strom (V, E) postavený nad w platí:*

- $|V| \leq 2|w| - 1$,
- $|E| \leq 2|w| - 2$.

1.4.4 Suffixový automat

Definice 7 *Suffixový automat je graf (V,E), pro který platí:*

- $V = \{[x]_w^R | x \in F(w)\}$,
- $E = \{([x]_w^R, a, [xa]_w^R) | x, xa \in F(w), a \in \Sigma\}$.

Suffix link $F = \{([ax]_w^R, [x]_w^R) | x, ax \in F(w), a \in \Sigma, , [ax]_w^R \neq [x]_w^R\}$.

Vrchol $[\varepsilon]_w^R$ je nazýván zdrojem a vrchol s výstupním stupněm rovnému nule stokem.

Tvrzení 5 (Blumer aj. [9]) *Nechť $w \in \Sigma^*$ a $|w| > 1$. Pak pro suffixový automat (V, E) postavený nad w platí:*

- $|V| \leq 2|w| - 1$,
- $|E| \leq 3|w| - 3$.

1.4.5 Kompaktní suffixový automat

Definice 8 *Kompaktní suffixový automat je graf (V,E), pro který platí:*

- $V = \{[\frac{w}{x}\rightarrow]_w^R | x \in F(w)\}$,
- $E = \{([\frac{w}{x}\rightarrow]_w^R, a\beta, [\frac{w}{xa}\rightarrow]_w^R) | x, xa \in F(w), a \in \Sigma, \beta \in \Sigma^*, \frac{w}{xa}\rightarrow = xa\beta, \frac{w}{x}\rightarrow \neq \frac{w}{xa}\rightarrow\}$.

Suffix link
$F = \{([\frac{w}{ax}\rightarrow]_w^R, [\frac{w}{x}\rightarrow]_w^R) | x, ax \in F(w), a \in \Sigma, \frac{w}{ax}\rightarrow = a\ \frac{w}{x}\rightarrow, [\frac{w}{ax}\rightarrow]_w^R \neq [\frac{w}{x}\rightarrow]_w^R\}$.

Vrchol $[\frac{w}{\varepsilon}\rightarrow]_w^R$ je nazýván zdrojem a vrchol s výstupním stupněm rovnému nule stokem. Délka vrcholu $[\frac{w}{x}\rightarrow]_w^R$ se definuje jako $|\ \frac{w}{x}\leftrightarrow\ |$.

Stejně jako u suffixového stromu je možné si i zde všimnout, že se v automatu podle této definice může vyskytovat vrchol s výstupním stupněm jedna. Stejně jako u stromu se však tento problém vyřeší tím, že se na konec vstupního slova přidá speciální znak, který se nevyskytuje jinde v celém slově. Tato vlastnost bude více rozebrána v části 2.3.1.

Tvrzení 6 (Blummer aj. [6]) *Nechť $w \in \Sigma^*$ a $|w| > 1$. Pak pro kompaktní suffixový automat (V, E) postavený nad w platí:*

- $|V| \leq |w| + 1$,
- $|E| \leq 2|w| - 2$.

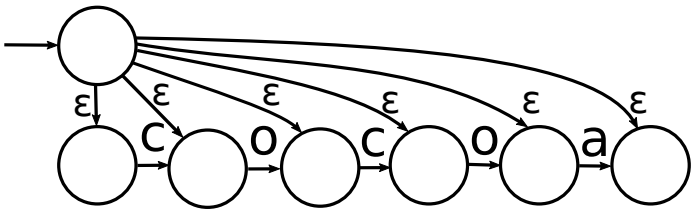
Kapitola 2

Algoritmy pro konstrukci indexových struktur

V této kapitole jsou nejprve zmíněny možné postupy, kterými lze kompaktní suffixový automat zkonstruovat. Následuje popis algoritmu pro přímou konstrukci suffixového stromu popsaný v [10], jehož úpravou vzejde algoritmus pro přímou konstrukci kompaktního suffixového automatu. Definice a algoritmy jsou přebrány z článku [3], jelikož to je článek, podle kterého byl algoritmus implementován.

2.1 Existující řešení

Zřejmě nejjednodušší postupem pro sestavení kompaktního suffixového automatu je navržení nedeterministického automatu s epsilon přechody a jeho následná determinizace, minimalizace a kompakce. Návrh automat je velmi jednoduchý. Pro řetězec délky n je potřeba $n + 1$ stavů, kde z i-tého stavu povede přechod do i+prvního na i-tý znak v řetězci. Následně se přidá počáteční stav, ze kterého povedou ε přechody do každého stavu. Příklad takového nedeterministického suffixového automatu je na obrázku 2.1.



Obrázek 2.1: Příklad nedeterministického suffixového automatu pro slovo *cocoa*

Již zmíněným postupem je konstrukce trie a její následná minimalizace a kompakce, jak bylo znázorněno na obrázku 1.2. Nejpřímočařejším postupem

pro vytvoření suffixové trie je vložení všech přípon zadaného slova do trie. Například tedy pro slovo *coco* by se postupně do trie vkládala slova *o, co, oco, coco*. Obě předchozí řešení ale nejsou příliš efektivní.

První popsaný algoritmus pro konstrukci kompaktního suffixového automatu byl popsán v [9]. Tento algoritmus nejprve zkonstruuje suffixový automat a následně odstraní všechny stavy s jedním výstupním přechodem. Toto řešení ovšem stále mělo nevýhodu v paměťové složitosti právě kvůli potřebě konstruovat nekompaktní verzi automatu. První řešení pro přímou konstrukci bylo popsáno v [11]. Tento algoritmus ovšem nebyl on-line, tedy bylo potřeba dopředu znát celý řetězec, pro který má být automat sestaven a nemohl být zkonstruován proudově.

V [3] se právě tento problém vyřešil a byl popsán první on-line algoritmus pro přímou konstrukci kompaktního suffixového automatu. Z tohoto důvodu byl právě tento algoritmus zvolen k implementaci v této práci. Algoritmus přímo vychází z Ukkonenova algoritmu pro přímou konstrukci suffixového stromu [10].

2.2 On-line konstrukce suffixové trie

Tato část je uvedena, aby bylo jednodušší pochopit, jak funguje ukkonenova přímá konstrukce suffixového stromu. Trie je velmi jednoduchá struktura, a proto nyní bude následovat stručnější popis její on-line konstrukce popsané v [10]. Konstrukce stromu je pak v základu podobná konstrukci trie.

Pro libovolné podslovo $x \in F(w)$ značí $suf(x)$ vrchol, který odpovídá suffix linku daného podslova x. Kvůli zjednodušení algoritmu je zaveden vrchol $\perp$, pro který platí, že $suf(\varepsilon) = \perp$. Dále nechť existují hrany $(\perp, a, \varepsilon)$ pro všechna $a \in \Sigma$. Není třeba definovat $suf(\perp)$, protože není nikdy použit. Výhoda zavedení tohoto vrcholu je, že při konstrukci není potřeba rozlišovat mezi prázdnou a neprázdnou příponou.

Definice 9 *Pro libovolné slovo $w \in \Sigma^*$ a znak $a \in \Sigma$ nejdelší opakovaná přípona slova wa značená $LRS(wa)$ je nejdelší prvek množiny $F(w) \cap S(wa)$.*

Neformálně $LRS(wa)$ značí nejdelší příponu slova wa takovou, že je tuto příponu možné nalézt někde ve slově po odstranění posledního znaku. Zde je odstraněný znak značený a.

Nechť $w \in \Sigma^*, a \in \Sigma$ a $|w| = n$. Dále nechť $u_1, \ldots, u_n, u_{n+1}, u_{n+2}$ je uspořádaná posloupnost všech přípon slova wa v pořadí od nejdelší po nejkratší. Tedy $u_1 = wa$ a $u_{n+2} = \varepsilon$. Všechny přípony se rozdělí do dvou skupin následovně:

1. $u_1, \ldots, u_\ell$,
2. $u_{\ell+1}, \ldots, u_{n+2}$, kde $u_{\ell+1} = LRS(wa)$.

Nyní ke zmíněné konstrukci suffixové trie. Protože je konstrukce on-line, postupuje od prvního znaku po poslední a postupně přidává současný znak do trie sestavené pro všechny předchozí znaky. Je tedy možné dívat se na konstrukci „induktivně“, kde základní krok je triviální vytvoření trie s pomocným vrcholem $\perp$ a vrcholem reprezentující prázdné slovo ε. Nyní se uvažuje, že už je hotová trie pro nějaké slovo w a v současném kroku je potřeba přidat znak a. Z definice $LRS(wa)$ plyne, že všechny přípony ve druhé skupině už v trii jsou a tudíž není třeba je přidávat.

Nechť $v_1, \ldots, v_{n+1}$ je posloupnost přípon slova w od nejdelší po nejkratší. Tedy $v_1 = w, v_{n+1} = \varepsilon$. Je zřejmé, že $v_k a = u_k$ pro libovolné $1 \leq k \leq n+1$. Algoritmus pro konstrukci trie začne přidáním nejdelší přípony u_1, tedy právě celého slova wa. Vytvoří se nový vrchol, do kterého povede hrana označená znakem a z vrcholu $v_1 = w$. Tento vrchol tedy vznikl zřetězením $wa = v_1 a = u_1$. Následně se přesune na vrchol v_2, což je možné v konstantním čase za použití suffix linků neboť $suf(v_1) = v_2$. Je vytvořen vrchol u_2 a hrana (v_2, a, u_2) a nastaví se $suf(u_1) = u_2$. Takto se pokračuje až dokud se nedojde k vrcholu $u_{\ell+1}$. Tímto postupem se suffixová trie zkonstruuje on-line.

2.3 Ukkonenův algoritmus pro přímou konstrukci suffixového stromu

V této části bude popsán Ukkonenovův algoritmus pro on-line přímou konstrukci suffixového stromu poprvé popsán v [10]. Tento algoritmus ovšem tvoří strom, který vychází z definice 6, je ale mírně upravený.

2.3.1 Upravený suffixový strom

Nechť pro nějaké slovo w značí $Strom(w)$ upravený suffixový strom sestavený nad slovem w. Tento strom vznikne ze stromu popsaného v definici 6 odebráním všech vrcholů s výstupním stupněm jedna. Hrany vedoucí do takového vrcholu se přesměrují do následníka odstraněného vrcholu. Za popisek přesměrovaných hran se zřetězí popisek výstupní hrany odstraněného vrcholu. Výstupní hrana se odstraní úplně. Pro formální definici je nejprve potřeba zavést několik dalších značení.

- Nechť X_w je relace nad Σ^* definovaná následovně:

  $X_w = \{(x, xa) | x \in F(w), a \in \Sigma$ takové, že existuje jediné a, pro které platí $xa \in F(w)\}$.

- Nechť $\equiv_w'^L$ je ekvivalenční uzávěr relace X_w. Tedy nejmenší nadmnožina X_w taková, že je symetrická, reflexivní a tranzitivní.

- Nechť $\overset{w}{\underset{x}{\Rightarrow}}$ značí nejdelší slovo ve třídě ekvivalence, do které patří x v ekvivalenci $\equiv_w'^L$.

Příklad 5 *Nechť* $w = coco$. *Pak* $\overset{w}{\underset{\varepsilon}{\Rightarrow}} = \varepsilon, \overset{w}{\underset{c}{\Rightarrow}} = \overset{w}{\underset{co}{\Rightarrow}} = \overset{w}{\underset{coc}{\Rightarrow}} = \overset{w}{\underset{coco}{\Rightarrow}} = coco, \overset{w}{\underset{o}{\Rightarrow}} = \overset{w}{\underset{oc}{\Rightarrow}} = \overset{w}{\underset{oco}{\Rightarrow}} = oco$

Tvrzení 7 (Inenaga [7]) *Pro libovolné slovo* $w \in \Sigma^*$ *platí, že libovolná třída ekvivalence nad* $\equiv_w'^L$ *je sjednocením jedné nebo více tříd nad* $\equiv_w^L$.

Tvrzení 8 (Inenaga [7]) *Nechť* $x \in \Sigma^*$, *pak* $\overset{w}{\underset{x}{\rightarrow}}$ *je předponou* $\overset{w}{\underset{x}{\Rightarrow}}$. *Navíc pokud* $\overset{w}{\underset{x}{\rightarrow}} \neq \overset{w}{\underset{x}{\Rightarrow}}$, *pak* $\overset{w}{\underset{x}{\rightarrow}} \in S(w)$.

Tvrzení 9 (Inenaga [7]) *Pokud pro množinu* $S(w) - \{\varepsilon\}$ *platí, že žádný prvek není předponou jiného, pak* $\overset{w}{\underset{x}{\rightarrow}} = \overset{w}{\underset{x}{\Rightarrow}}$ *pro libovolné slovo* $x \in F(w)$.

Poslední tvrzení 9 mimo jiné říká, že pokud se poslední znak nenachází jinde ve slově, pak $\overset{w}{\underset{x}{\rightarrow}} = \overset{w}{\underset{x}{\Rightarrow}}$. Tohoto se využívá jak při konstrukci suffixového stromu, tak kompaktního suffixového automatu. Konkrétně se na konec slova, pro který je strom stavěn, připojí speciální znak. Tento znak bude značen \$.

Nyní už je možné formálně nadefinovat zmíněný upravený suffixový strom.

Definice 10 *Strom*(w) *je strom* (V, E), *pro který platí*

- $V = \{\overset{w}{\underset{x}{\Rightarrow}} | x \in F(w)\}$,
- $E = \{(\overset{w}{\underset{x}{\Rightarrow}}, a\beta, \overset{w}{\underset{xa}{\Rightarrow}}) |, x, xa \in F(w), a \in \Sigma, \beta \in \Sigma^*, \overset{w}{\underset{xa}{\Rightarrow}} = xa\beta, \overset{w}{\underset{x}{\Rightarrow}} \neq \overset{w}{\underset{xa}{\Rightarrow}}\}$.

Suffix link
$F = \{(\overset{w}{\underset{ax}{\Rightarrow}}, \overset{w}{\underset{x}{\Rightarrow}}) | x, ax \in F(w), a \in \Sigma, \overset{w}{\underset{ax}{\Rightarrow}} = a \overset{w}{\underset{x}{\Rightarrow}}\}$

Protože se v této definici oproti definici 6 změnila jen $\overset{w}{\rightarrow}$ na $\overset{w}{\Rightarrow}$, platí z tvrzení 9, že pokud se poslední znak nenachází jinde ve slově, pak se tento upravený strom rovná suffixovému stromu definovanému v 6.

Hrany, které vedou do listu budou nazývány *otevřené*.

2.3.2 Ukkonenův algoritmus

Aby měl algoritmus prostorovou složitost lineární, jsou popisky hran reprezentovány jako dvojice celých čísel (k, p), kde k je počáteční pozice podslova a p je koncová pozice podslova. Konkrétně tedy nechť $(\overset{w}{\underset{x}{\Rightarrow}}, \alpha, \overset{w}{\underset{x\alpha}{\Rightarrow}})$ je hrana v $Strom(w)$ sestavený pro nějaké slovo w. Pak taková hrana bude reprezentována jako $(\overset{w}{\underset{x}{\Rightarrow}}, (k, p), \overset{w}{\underset{x\alpha}{\Rightarrow}})$, kde platí $w[k : p] = \alpha$.

Implicitní vrchol, který reprezentuje nějaké $y \in F(w)$, bude reprezentován dvojicí $(\overset{w}{\underset{x}{\Rightarrow}}, \alpha)$, kde $\overset{w}{\underset{x}{\Rightarrow}}$ je nějaký explicitní vrchol, $\alpha \in F(w)$ a platí $y = \overset{w}{\underset{x}{\Rightarrow}} \alpha$. Takovýchto párů samozřejmě může být více a je možné ním reprezentovat i explicitní vrcholy, zde však budou uvažovány jen páry pro vrcholy, které mají co nejmenší délku slova α. Obdobně jako popisek hrany je v algoritmu

i popisek těchto párů reprezentován dvojicí (k, p) takovou, že $w[k:p]$ se rovná popisku daného páru.

Stejně jako tomu bylo u algoritmu pro konstrukci trie, zde je také možné rozdělit přípony podle nejdelší opakované přípony. Nechť tedy je hotový $Strom(w)$ pro nějaké slovo $w \in \Sigma^*$. Nyní se má přidat další znak $a \in \Sigma$. Stejně jako u trie není potřeba vkládat přípony z druhé skupiny tedy o velikosti $LRS(wa)$ a kratší. Přípony slova wa z první skupiny se však dají rozdělit na podskupinu podle $LRS(w)$ na:

1.a $u_1, \ldots, u_k$,

1.b $u_{k+1}, \ldots, u_\ell$, kde $u_{k+1} = LRS(w)a$.

„*Přípony ze skupiny 1.a jsou ty, které jsou reprezentovány listy ve stromě* $Strom(w)$*, z čehož plyne, že* $\forall i\ 1 \leq i \leq k$ *platí* $\overset{w}{\underset{v_i}{\Rightarrow}} = v_i$ *a* $\overset{wa}{\underset{u_i}{\Longrightarrow}} = u_i$“ [3]. ($u_i$ jsou přípony wa, v_i jsou přípony w.) Protože platí $v_i a = u_i$ Vyplývá z předchozí věty, že list ve stromě $Strom(w)$ je také listem v $Strom(wa)$, což je velmi důležitý fakt. Díky tomuto je možné jakoukoliv hranu $(\overset{w}{\underset{x}{\Rightarrow}}, \alpha, \overset{w}{\underset{x\alpha}{\Longrightarrow}})$ vedoucí do listu reprezentovat jako (p, ∞), kde $w[p : |w|] = \alpha$. Důsledkem je, že není třeba se starat o přípony ze skupiny 1.a, protože hrana vždy povede až do konce slova, aniž by ji bylo třeba stále prodlužovat.

Pro přípony ze skupiny 1.b je postup náročnější. Nechť je k dispozici $Strom(w)$, do kterého se přidává znak a a v současnou chvíli se konkrétně vkládá přípona u_j. Podobně jako u trie je potřeba najít vrchol, který odpovídá příponě v_j a vytvořit novou hranu z tohoto vrcholu. Takovýto vrchol se bude nazývat *aktivní bod*. Oproti trii je ale hlavní rozdíl v tom, že takový vrchol může být implicitní. Nejprve však bude popsána ta jednodušší možnost.

Pokud je aktivní bod explicitní vrchol v_j, pak $\overset{w}{\underset{v_j}{\Rightarrow}} = \overset{wa}{\underset{v_j}{\Longrightarrow}} = v_j$. V takovém případě není postup nijak náročný. Vytvoří se nová hrana $(\overset{wa}{\underset{v_j}{\Longrightarrow}}, a, \overset{wa}{\underset{v_j a}{\Longrightarrow}})$. (Pro připomenutí $v_j a = u_j$.) Takováto hrana ale bude otevřená, takže její popisek je reprezentován dvojicí $(n+1, \infty)$, kde $|wa| = n+1$ a $wa[n+1] = a$. (Indexuje se od jedničky.) Následně se aktivní bod přesune na $suf(\overset{w}{\underset{v_j}{\Rightarrow}}) = v_{j+1}$, aby se mohla vložit další přípona u_{j+1}.

Pokud je aktivní bod implicitní vrchol, znamená to, že $\overset{w}{\underset{v_j}{\Rightarrow}} \neq v_j$, ale $\overset{wa}{\underset{v_j}{\Longrightarrow}} = v_j$. Slovy řečeno, že se vrchol po přidání a stane explicitním. Nechť $(\overset{w}{\underset{x}{\Rightarrow}}, \alpha)$ je pár odpovídající tomuto implicitnímu vrcholu, který je v současnou chvíli aktivním bodem. Jinak řečeno $\overset{w}{\underset{x}{\Rightarrow}} \alpha = v_j$. Hrana, na které je požadovaný implicitní vrchol, je $(\overset{w}{\underset{x}{\Rightarrow}}, \alpha\beta, \overset{w}{\underset{x\alpha\beta}{\Longrightarrow}})$, pro nějaké neprázdné $\beta \in \Sigma^*$. Tuto hranu je třeba rozdělit na dvě v místě, kde se nachází aktivní bod. V tomto místě bude vytvořen nový vrchol. Formálněji tedy bude hrana $(\overset{w}{\underset{x}{\Rightarrow}}, \alpha\beta, \overset{w}{\underset{x\alpha\beta}{\Longrightarrow}})$ nahrazena hranami $(\overset{wa}{\underset{x}{\Longrightarrow}}, \alpha, \overset{wa}{\underset{x\alpha}{\Longrightarrow}})$ a $(\overset{wa}{\underset{x\alpha}{\Longrightarrow}}, \beta, \overset{wa}{\underset{x\alpha\beta}{\Longrightarrow}})$. $\overset{wa}{\underset{x\alpha}{\Longrightarrow}}$ je zmíněný nový vrchol, který odpovídá v_j.

Následující kroky jsou už podobné případu, kdy je k dispozici explicitní vrchol. Tedy je vytvořená nová hrana $(\overset{wa}{\underset{x\alpha}{\Longrightarrow}}, a, \overset{wa}{\underset{x\alpha a}{\Longrightarrow}})$. Jedná se o otevřenou hranu, tudíž dvojice reprezentující popisek bude opět $(n+1, \infty)$.

Nastává ovšem problém ve chvíli, kdy se má přejít na další příponu. Suffix link právě vytvořeného vrcholu $\overset{wa}{\underset{x\alpha}{\Longrightarrow}}$ ještě není spočten a $suf(\overset{wa}{\underset{x\alpha}{\Longrightarrow}})$ a není v současnou chvíli definován. Suffix link nového vrcholu se ale dá najít následovně. Vrchol $\overset{wa}{\underset{x}{\Longrightarrow}}$, který je rodičem, nově vzniklého vrcholu $\overset{wa}{\underset{x\alpha}{\Longrightarrow}}$ má určitě $suf(\overset{wa}{\underset{x}{\Longrightarrow}})$ definovaný a tento vrchol je explicitní. Platí $suf(\overset{wa}{\underset{x}{\Longrightarrow}})\alpha = v_{j+1}$. Tohoto se využije tak, že se z vrcholu, který odpovídá $suf(\overset{wa}{\underset{x}{\Longrightarrow}})$, projde na hranu, která má ve svém popisku α. Tím se dojde do místa, kde buď je explicitní vrchol, který odpovídá v_{j+1}, nebo se zde nachází implicitní vrchol, který se v následujícím kroku stane explicitní. Jinak řečeno platí následující $suf(\overset{wa}{\underset{x\alpha}{\Longrightarrow}}) = suf(\overset{wa}{\underset{x}{\Longrightarrow}})\alpha$, kde je vrchol $suf(\overset{wa}{\underset{x}{\Longrightarrow}})\alpha = v_{j+1}$, který po přidání následující přípony bude určitě explicitní.

Důležité je také připomenout vrchol značený $\perp$. Jak bylo zmíněno jedná se o pomocný vrchol, který usnadňuje algoritmus v tom, že není třeba zvlášť řešit případ prázdné přípony ε. Z tohoto vrcholu vede hrana na každý znak abecedy do vrcholu ε. Tohoto se v algoritmu docílí následovně. Algoritmus pracuje s upraveným slovem w, kterému se na jeho záporné pozice $w[-1], \ldots, w[-|\Sigma|]$ přidají postupně všechny znaky abecedy. Pro jistotu nechť je ještě jednou řečeno, že hrany jsou značeny dvojicí celých čísel. Z vrcholu $\perp$ vedou následující hrany: $(\perp, (-k, -k), \varepsilon)$, pro všechna $-|\Sigma| \leq k \leq -1$.

Nyní konečně následuje pseudokód tohoto algoritmu 1. Před ním je uvedeno několik komentářů.

Řádek 1 Na konci w je speciální znak \$. Na záporných pozicích w jsou vyskládané všechny znaky abecedy.

Řádek 5 s je vrchol a k a i jsou celá čísla.

Řádek 10 Funkce vrací pár, kde první prvek je vrchol a druhý číslo. Tento pár je dříve zmíněná reprezentace vrchol, jak bylo řečeno ve druhém odstavci této kapitoly.

Řádek 11 c je znak a *předchozíR* je ukazatel na vrchol.

Řádek 21 Funkce vrací *pravda*, pokud zadaný vrchol je $LRS(wa)$.

Řádek 27 Funkce přijme pár (s, α), který reprezentuje nějaký vrchol a minimalizuje α. Vrací pár (vrchol,číslo).

Řádek 33 Proměnné k', p', s' jsou přepsány.

Řádek 35 Funkce se volá v případě, že aktivní bod je implicitní vrchol. Hranu je potřeba rozdělit a vytvořit nový vrchol, jak již bylo řečeno v popisu algoritmu. Funkce vrací nově vytvořený vrchol.

Na závěr této části věta o složitosti tohoto algoritmu.

Tvrzení 10 (Ukkonen [10]) *Nechť Σ je konečná abeceda. Pro libovolné slovo $w \in \Sigma^*$ je možné zkonstruovat Strom(w) on-line v čase $\mathcal{O}(|w|)$ a s využitím $\mathcal{O}(|w|)$ prostoru.*

Algoritmus 1 Ukkonenův on-line algoritmus pro přípomou konstrukci suffixového stromu [10]

1: **Algoritmus** pro konstrukci Strom(w)
2: vytvoř vrcholy *kořen* a $\perp$;
3: **pro každé** $1 \leq j \leq |\Sigma|$: vytvoř hranu $(\perp, (-j, -j)$,*kořen*);
4: $suf(kořen) \leftarrow \perp$;
5: $(s, k) \leftarrow (kořen, 1)$; $i \leftarrow 0$;
6: **opakuj**
7: $\quad i \leftarrow i + 1$;
8: $\quad (s, k) \leftarrow$ Aktualizuj$(s, (k, i))$;
9: **dokud** $w[i] \neq \$$;

10: **funkce** Aktualizuj$(s, (k, p))$
11: $\quad c \leftarrow w[p]$; *předchozíR* se nechá neinicializováno;
12: $\quad$ **dokud neplatí** Zkontroluj LRS(s,(k,p-1),c) **opakuj**
13: $\quad\quad$ **pokud** $(k \leq p - 1)$ **pak** $r \leftarrow$ Rozděl hranu(s,(k,p-1));
14: $\quad\quad$ **jinak** $r \leftarrow s$; ▷ Vrchol je explicitní.
15: $\quad\quad$ vytvoř vrchol r'; vytvoř hranu $(r, (p, \infty), r')$;
16: $\quad\quad$ **pokud** je *předchozíR* inicializováno **pak** *suf(předchozíR)* $\leftarrow r$;
17: $\quad\quad$ *předchozíR* $\leftarrow r$;
18: $\quad\quad$ $(s, k) \leftarrow$ Minimalizuj pár$(suf(s), (k, p - 1))$;
19: $\quad$ **pokud** *přechozíR* je inicializováno **pak** *suf(předchozíR)* $\leftarrow s$;
20: $\quad$ **vrať** Minimalizuj pár$(s, (k, p))$;

21: **funkce** Zkontroluj LRS$(s, (k, p))$
22: $\quad$ **pokud** $k \leq p$ **pak**
23: $\quad\quad$ nechť $(s, (k', p'), s')$ je hrana vedoucí z s začínající na $w[k]$;
24: $\quad\quad$ **vrať** $(c = w[k' + p - k + 1])$;
25: $\quad$ **jinak**
26: $\quad\quad$ **vrať** pravda v případě, že z vrcholu s vede hrana začínající na c;

27: **funkce** Minimalizuj pár$(s,(k,p))$
28: **pokud** $k > p$ **pak vrať** (s,k); ▷ Explicitní vrchol
29: nechť $(s,(k',p'),s')$ je hrana vedoucí z s začínající na $w[k]$;
30: **dokud** $(p'-k') \leq (p-k)$ **opakuj**
31: $k \leftarrow k+p'-k'+1$; $s \leftarrow s'$;
32: **pokud** $k \leq p$ **pak**
33: nechť $(s,(k',p'),s')$ je nová hrana vedoucí z s začínající na $w[k]$;
34: **vrať** (s,k);

35: **funkce** Rozděl hranu$(s,(k,p))$
36: vytvoř vrchol r;
37: nechť $(s,(k',p'),s')$ je hrana vedoucí z s začínající na $w[k]$;
38: nahraď hranu za $(s,(k',k'+p-k),r)$ a $(r,(k'+p-k+1,p'),s')$;
39: **vrať** r;

2.4 On-line konstrukce kompaktního suffixového automatu

V této části se nejprve nadefinuje upravený kompaktní suffixový automat podobně, jako tomu bylo u suffixového stromu. Pak budou popsány některé důležité myšlenky, které jsou potřeba pro samotný algoritmus, jehož popis bude následovat. Algoritmus byl popsán v [3]. Celá část je zakončená pseudokódem on-line algoritmu pro přímou konstrukci kompaktního suffixového automatu.

2.4.1 Upravený kompaktní suffixový automat

Upravený kompaktní suffixový automat sestavený pro nějaké slovo w bude z důvodu přehlednosti značen $Automat(w)$ obdobně, jako tomu bylo u upraveného suffixového stromu (definice 10).

Definice 11 *$Automat(w)$ je orientovaný acyklický graf (V,E), pro který platí*

- $V = \{[\underset{x}{\overset{w}{\Rightarrow}}]^R_w | x \in F(w)\}$,
- $E = \{([\underset{x}{\overset{w}{\Rightarrow}}]^R_w, a\beta, [\underset{xa}{\overset{w}{\Rightarrow}}]^R_w) | x, xa \in F(w), a \in \Sigma, \beta \in \Sigma^*, \underset{xa}{\overset{w}{\Rightarrow}} = xa\beta, \underset{x}{\overset{w}{\Rightarrow}} \neq \underset{xa}{\overset{w}{\Rightarrow}}\}$.

Suffix link
$F = \{([\underset{ax}{\overset{w}{\Rightarrow}}]^R_w, [\underset{x}{\overset{w}{\Rightarrow}}]^R_w) | x, ax \in F(w), a \in \Sigma, \underset{ax}{\overset{w}{\Rightarrow}} = a \underset{x}{\overset{w}{\Rightarrow}}, [\underset{x}{\overset{w}{\Rightarrow}}]^R_w \neq [\underset{ax}{\overset{w}{\Rightarrow}}]^R_w\}$

Vrchol $[\underset{\varepsilon}{\overset{w}{\Rightarrow}}]^R_w$ je zdroj a vrchol s výstupním stupněm nula je stok, který je vždy jen jeden a to $[\underset{w}{\overset{w}{\Rightarrow}}]^R_w$.

Délka vrcholu $[\underset{x}{\overset{w}{\Rightarrow}}]^R_w$ je definována jako $|(\underset{x}{\overset{\overset{w}{\leftarrow}}{\Rightarrow}})|$.

Definice 11 je stejná jako definice 8, jen je operace $\xrightarrow{w}$ nahrazená za $\overset{w}{\Rightarrow}$. Díky tomu z tvrzení 9 plyne, že pokud se poslední znak nenachází jinde ve slově w, pro které je automat konstruován, pak se $Automat(w)$ rovná kompaktnímu suffixovému automatu z definice 8.

2.4.2 Algoritmus pro přímou konstrukci kompaktního suffixového automatu

Algoritmus vychází z Ukkonenova algoritmu a mnohé kroky jsou proto velmi podobné. V celé této části nechť $w \in \Sigma^*, |w| = n, a \in \Sigma$. Jak už bylo několikrát řečeno dříve, přípony slova w se dělí na tři skupiny podle nejdelší opakované přípony. Nechť $u_1, \ldots, u_{n+2}$ jsou všechny přípony slova wa seřazené od nejdelší wa po nejkratší ε. Nechť $v_1, \ldots, v_{n+1}$ jsou všechny přípony slova w. Platí, že $v_i a = u_i$, kde $1 \leq i \leq n+1$. Nechť $u_{\ell+1} = LRS(wa)$ a $v_{k+1} = LRS(w)$.

Pro představu bude nejjednodušší vypsat rozdíly v konstrukci kompaktního automatu oproti stromu. První rozdíl je, že všechny přípony z první skupiny jsou ve stejné třídě ekvivalence nad $\equiv^R_{wa}$. Formálněji řečeno $[\overset{wa}{\underset{u_1}{\Longrightarrow}}]^R_{wa} = [\overset{wa}{\underset{u_2}{\Longrightarrow}}]^R_{wa} = \cdots = [\overset{wa}{\underset{u_\ell}{\Longrightarrow}}]^R_{wa}$. Tento vrchol odpovídá stoku, který byl zmíněn u definice 11, což znamená, že všechny otevřené hrany vedou do jediného vrcholu. Navíc platí $\overset{w}{\underset{v_i}{\Rightarrow}} = v_i$ a $\overset{wa}{\underset{u_i}{\Longrightarrow}} = u_i$, kde $1 \leq i \leq k$. Díky těmto vlastnostem je možné reprezentovat otevřené hrany $(s, (p, |w|), t)$, kde t je stok a $|w|$ je aktuální délka slova, která se s přidáním každého dalšího znaku inkrementuje. Implementováno to může být pomocí ukazatele na globální proměnnou. Toto je potřeba z důvodu, aby prodloužení všech otevřených hran po přidání nového znaku bylo konstantní, co se časové složitosti týče.

Druhý rozdíl je, že někdy se některé vrcholy spojí v jeden. Nechť aktivní bod je implicitní vrchol, který reprezentuje příponu v_j. Pak může existovat přípona v_h taková, že $h \neq j$ a $v_h \equiv^R_{wa} v_j$. V takovém případě se tyto vrcholy spojí v jeden explicitní vrchol $[\overset{wa}{\underset{v_j}{\Longrightarrow}}]^R_{wa}$. Test na spojování vrcholů bude prováděn na základě tvrzení 12.

Posledním výrazným rozdílem je, že některé vrcholy, které byly spojené, je potřeba rozdělit. Nechť $x, y \in F(w) : \overset{w}{\underset{x}{\Rightarrow}} = x \wedge \overset{w}{\underset{y}{\Rightarrow}} = y$ a $x \equiv^R_w y$, což znamená, že obě podslova jsou reprezentována stejným explicitním vrcholem $[x]^R_w = [y]^R_w$. Může ovšem nastat, že $x \not\equiv^R_{wa} y$, tedy po přidání nového znaku je potřeba vrcholy rozdělit na dva $[x]^R_{wa}$ a $[y]^R_{wa}$. Nechť $|x| > |y|$, pak takovýto případ může nastat jen, pokud $x \notin S(wa) \wedge y \in S(wa)$. Podmínky na rozdělení vrcholů jsou v tvrzení 13.

2.4.2.1 Spojování implicitních vrcholů

Jak již bylo řečeno, ve struktuře $Automat(w)$ se může stát, že při konverzi na $Automat(wa)$ se některé implicitní vrcholy spojí v jeden explicitní. Rozhod-

nutí, jestli se vrcholy spojí, je děláno na základě testování ekvivalence $\equiv^R_{wa}$ daných vrcholů.

Tvrzení 11 (Inenaga aj. [3]) *Nechť $w \in \Sigma^*$ a $x \in F(w)$. Platí, že x je podslovo $\overset{w}{\underset{x}{\leftrightarrow}}$ a nachází se v něm právě jednou.*

Tvrzení 12 (Inenaga aj. [3]) *Nechť $w \in \Sigma^*$. Pro libovolná dvě slova $x, y \in F(w) \wedge y \in S(x)$ platí $x \equiv^R_w y \Leftrightarrow [\overset{w}{\underset{x}{\rightarrow}}]^R_w = [\overset{w}{\underset{x}{\rightarrow}}]^R_w$.*

Pro libovolné slovo $x \in F(w)$ je $[\overset{w}{\underset{x}{\rightarrow}}]^R_w$ nejbližší explicitní potomek v kompaktním suffixovém automatu z definice 8 a v něm se tedy ekvivalence dvou slov rozhodne pomocí tvrzení 12.

Problém ale je, že $\overset{w}{\underset{x}{\rightarrow}}$ nemusí být explicitní v upraveném kompaktním automatu $Automat(w)$. To znamená, že při testování ekvivalence by se testoval vrchol $[\overset{w}{\underset{x}{\Rightarrow}}]^R_w$ místo $[\overset{w}{\underset{x}{\rightarrow}}]^R_w$. „*Takový případ ale nemůže nastat, protože se přípony zpracovávají v pořadí od nejdelší po nejkratší*“ [3].

2.4.2.2 Dělení explicitních vrcholů

Může se stát, že vrchol, který je v $Automat(w)$ explicitní, je třeba rozdělit na dva různé vrcholy po přidání znaku a.

Tvrzení 13 (Blummer aj. [9]) *Nechť $w \in \Sigma^*, a \in \Sigma$. Nechť $z = LRS(wa)$. Pro libovolné slovo $x \in F(w)$ takové, že $x = \overset{w}{\underset{x}{\leftarrow}}$ platí:*

$$[x]^R_w = [x]^R_{wa} \cup [z]^R_{wa}, \text{ pokud } z \in [x]^R_w \wedge x \neq z,$$

$$= [x]^R_{wa} \text{ jinak.}$$

Z tvrzení 13 plyne, že je potřeba řešit pouze vrcholy $[x]^R_w$, kde $z \in [x]^R_w \wedge z = LRS(wa)$. To znamená, že maximálně jeden vrchol bude rozdělen při přidání dalšího znaku. Pokud $z \neq \overset{w}{\underset{x}{\leftarrow}}$, pak dojde k rozdělení na dva vrcholy $[x]^R_{wa}$ a $[z]^R_{wa}$. V případě, že $z = \overset{w}{\underset{x}{\leftarrow}}$, se vrchol nedělí. Rozhodnutí o této rovnosti se provádí na základě délek vrcholů $\overset{w}{\underset{x}{\leftarrow}}$ a z.

Nechť $y \in F(w)$ je slovo, pro které platí $\overset{w}{\underset{y}{\leftarrow}}\, a = z$. Pak platí následující:

- $z = \overset{w}{\underset{x}{\leftarrow}} \Leftrightarrow$ *délka*$([y]^R_w) + |a| =$ *délka*$([x]^R_w)$,
- $z \neq \overset{w}{\underset{x}{\leftarrow}} \Leftrightarrow$ *délka*$([y]^R_w) + |a| <$ *délka*$([x]^R_w)$.

Pro pomocný vrchol $\perp$ se jeho délka nadefinuje jako -1, díky čemuž není třeba zvlášť ošetřovat případ, kdy $z = \varepsilon$.

Testování, zda má být vrchol rozdělen, bude prováděno na základě tvrzení 13 i v upraveném kompaktním automatu $Automat(w)$. Protože dělené můžou být jen explicitní vrcholy, je potřeba kontrolovat jen případy, kdy $z = \overset{wa}{\underset{z}{\Longrightarrow}}$, kde $z = LRS(wa)$.

Tvrzení 14 (Inenaga aj. [3]) *Nechť $w \in \Sigma^*$ a $z = LRS(w)$. Pokud $z = \overset{w}{\underset{z}{\Rightarrow}}$, pak pro libovolné $x \in F(w)$ platí $\overset{w}{\underset{x}{\Rightarrow}} = \overset{w}{\underset{x}{\rightarrow}}$.*

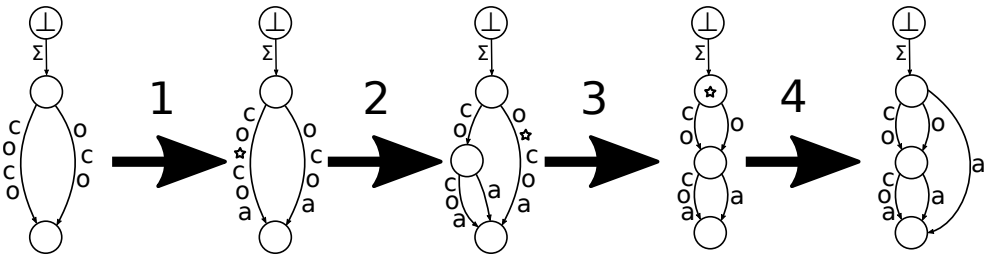
Toto tvrzení zaručuje, že reprezentant $[\overset{w}{\underset{x}{\Rightarrow}}]^R_w$ se rovná $\overset{w}{\underset{x}{\leftrightarrow}}$, pokud jsou podmínky uvedené v tvrzení splněny. Dělení vrcholů tedy bude probíhat následovně: Pokud $z \neq \overset{w}{\underset{x}{\leftrightarrow}}$, pak je vrchol rozdělen na dva $[x]^R_{wa}$ a $[z]^R_{wa}$. Pokud $z = \overset{w}{\underset{x}{\leftrightarrow}}$, pak se vrchol $[x]^R_w$ nedělí. Rozhodnutí o této rovnosti se provádí na základě délek vrcholů $\overset{w}{\underset{x}{\leftrightarrow}}$ a z

Nechť $\alpha \in F(w)$ a $y \in F(w)$ je slovo, pro které platí $\overset{w}{\underset{y}{\leftrightarrow}}\ \alpha = z$. Pak platí následující:

- $z = \overset{w}{\underset{x}{\leftrightarrow}} \Leftrightarrow$ *délka*$([y]^R_w) + |\alpha| =$ *délka*$([x]^R_w)$,
- $z \neq \overset{w}{\underset{x}{\leftrightarrow}} \Leftrightarrow$ *délka*$([y]^R_w) + |\alpha| <$ *délka*$([x]^R_w)$.

Na obrázku 2.2 je znázorněno přidání nového znaku do automatu. Konkrétně se jedná o přidání znaku a do automatu, který je v současné chvíli sestaven pro slovo *coco*. Vrchol se znakem $\perp$ odpovídá pomocnému vrcholu.

V prvním kroku se prodlouží otevřené hrany. Hvězdička znázorňuje aktivní bod, ze kterého je třeba přidat novou příponu. Ve druhém kroku byla přidána nová přípona *coa*, čímž se implicitní vrchol stal explicitním. Ve třetím kroku je přidána připona *oa* a na tomto kroku je vidět spojení dvou vrcholů. V posledním kroku je přidána poslední neprázdná přípona *a*.



Obrázek 2.2: Převod struktury $Automat(coco)$ na $Automat(cocoa)$ [3]

2.4.2.3 Pseudokód

Nyní následuje pseudokód tohoto algoritmus. Funkce *Zkontroluj LRS* a *Minimalizuj pár* jsou identické s těmi v Ukkonenově algoritmu. Přesto zde budou pro přehlednost uvedeny znovu. Před samotným pseudokódem jsou uvedeny komentáře.

Řádek 1 Na konci w je speciální znak \$. Na záporných pozicích w jsou vyskládané všechny znaky abecedy.

Řádek 4 Proměnné e je v algoritmus globální proměnná. To znamená, že pokaždé, když se e změní, tak se na všech místech, kam se e přiřadilo, také změní hodnota.

Řádek 5 Jak bylo řečeno u komentáře k řádku 4. Přiřazení e do $délka(stok)$ není přiřazení hodnotově. Je potřeba zařídit, aby se při změně e změnila i hodnota $délka(stok)$.

Řádek 11 Funkce *Aktualizuj* se stará o přidání nového znaku do automatu.

Řádek 12 c je znak a *předchozíR* je ukazatel na vrchol.

Řádek 15 Při prvním průchodu není s' inicializováno. V takovém případě porovnání vždy vrací nepravda.

Řádek 24 Opět se přiřazuje e. Je tedy potřeba zařídit, aby se délka popisku hrany měnila spolu s přidáváním dalších znaků.

Řádek 44 Funkce je stejná, jako v Ukkonenově algoritmu jen s přidaným řádkem 48, kde se navíc počítá délka vrcholu.

Řádek 50 Funkce vrací nejbližší explicitní vrchol. Pokud přijatý vrchol je explicitní, pak vrátí přijatý vrchol.

Řádek 54 Funkce se stará o přesměrování hran do existujících vrcholů. Tato funkce zajišťuje spojování vrcholů, jak bylo řečeno v části 2.4.2.1.

Řádek 57 Funkce se stará o dělení explicitních vrcholů, jak bylo pospáno v části 2.4.2.2.

Na závěr celé části a kapitoly tvrzení, o časové složitosti algoritmu:

Tvrzení 15 (Inenaga aj. [3]) *Nechť Σ je konečná abeceda. Pro libovolné slovo $w \in \Sigma^*$ je možné zkonstruovat $Automat(w)$ on-line v čase $\mathcal{O}(|w|)$.*

Algoritmus 2 On-line algoritmus pro přímou konstrukci kompaktního suffixového automatu [3]

Algoritmus pro konstrukci Automat(w)
vytvoř vrcholy *zdroj*, *stok* a $\perp$;
pro každé $1 \leq j \leq |\Sigma|$: vytvoř hranu $(\perp, (-j, -j)$,*zdroj*);
$suf(zdroj) \leftarrow \perp$; $e \leftarrow 0$;
délka$(zdroj) \leftarrow 0$; *délka*$(\perp) \leftarrow -1$; *délka*$(stok) \leftarrow e$;
$(s, k) \leftarrow (zdroj, 1)$; $i \leftarrow 0$;
opakuj
    $i \leftarrow i + 1$; $e \leftarrow i$;
    $(s, k) \leftarrow$ Aktualizuj$(s, (k, i))$;
dokud $w[i] \neq \$$;

funkce Aktualizuj$(s, (k, p))$
    $c \leftarrow w[p]$; *předchozíR* se nechá neinicializováno;
    dokud neplatí Zkontroluj LRS(s,(k,p-1),c) **opakuj**
        pokud $(k \leq p - 1)$ **pak**
            pokud $s' =$ Vrať explicitní vrchol$(s, (k, p - 1))$ **pak**
                Přesměruj hranu$(s, (k, p - 1), r)$;
                $(s, k) \leftarrow$ Minimalizuj pár$(suf(s), (k, p - 1))$;
                Přejdi na další iteraci cyklu;
            jinak
                $s' \leftarrow$ Vrať explicitní vrchol$(s, (k, p - 1))$;
                $r \leftarrow$ Rozděl hranu$(s, (k, p - 1))$;
        jinak
            $r \leftarrow s$;
        vytvoř hranu $(r, (p, e), stok)$;
        pokud je *předchozíR* inicializováno **pak** *suf(předchozíR)* $\leftarrow r$;
        předchozíR $\leftarrow r$;
        $(s, k) \leftarrow$ Minimalizuj pár$(suf(s), (k, p - 1))$;
    pokud *přechozíR* je inicializováno **pak** *suf(předchozíR)* $\leftarrow s$;
    vrať Rozděl vrchol$(s, (k, p))$;

funkce Zkontroluj LRS$(s, (k, p))$
    pokud $k \leq p$ **pak**
        nechť $(s, (k', p'), s')$ je hrana vedoucí z s začínající na $w[k]$;
        vrať $(c = w[k' + p - k + 1])$;
    jinak
        vrať pravda v případě, že z vrcholu s vede hrana začínající na c;

36: **funkce** Minimalizuj pár$(s,(k,p))$
37: **pokud** $k > p$ **pak vrať** (s,k); ▷ Explicitní vrchol
38: nechť $(s,(k',p'),s')$ je hrana vedoucí z s začínající na $w[k]$;
39: **dokud** $(p'-k') \leq (p-k)$ **opakuj**
40: $k \leftarrow k+p'-k'+1$; $s \leftarrow s'$;
41: **pokud** $k \leq p$ **pak**
42: nechť $(s,(k',p'),s')$ je nová hrana vedoucí z s začínající na $w[k]$;
43: **vrať** (s,k);

44: **funkce** Rozděl hranu$(s,(k,p))$
45: vytvoř vrchol r;
46: nechť $(s,(k',p'),s')$ je hrana vedoucí z s začínající na $w[k]$;
47: nahraď hranu za $(s,(k',k'+p-k),r)$ a $(r,(k'+p-k+1,p'),s')$;
48: $\mathit{délka}(r) \leftarrow \mathit{délka}(s)+(p-k+1)$;
49: **vrať** r;

50: **funkce** Vrať explicitní vrchol$(s,(k,p))$
51: **pokud** $k > p$ **pak vrať** s; ▷ Přijatý vrchol byl explicitní
52: nechť $(s,(k',p'),s')$ je hrana vedoucí z s začínající na $w[k]$;
53: **vrať** s';

54: **funkce** Přesměruj hranu$(s,(k,p),r)$
55: nechť $(s,(k',p'),s')$ je hrana vedoucí z s začínající na $w[k]$;
56: nahraď tuto hranu hranou $(s,(k',k'+p-k),r)$;

57: **funkce** Rozděl vrchol$(s,(k,p))$
58: $(s',k') \leftarrow$ Minimalizuj pár$(s,(k,p))$;
59: **pokud** $k' \leq p$ **pak vrať** (s',k'); ▷ Implicitní vrchol
60: **pokud** $\mathit{délka}(s') = \mathit{délka}(s)+(p-k+1)$ **pak vrať** (s',k');
61: vytvoř vrchol r' jako kopii s' se všemi výstupními hranami;
62: $suf(r') \leftarrow suf(s')$; $suf(s') \leftarrow r'$;
63: $\mathit{délka}(r') \leftarrow \mathit{délka}(s)+(p-k+1)$;
64: **opakuj**
65: nahraď hranu vedoucí z s začínající na $w[k]$ hranou $(s,(k,p),r')$;
66: $(s,k) \leftarrow$ Minimalizuj pár$(suf(s),(k,p-1))$;
67: **dokud** $(s',k') =$ Minimalizuj pár$(s,(k,p))$;
68: **vrať** $(r',p+1)$;

Kapitola 3

Algoritmus vyhledávání v automatu

V této kapitole je popsáno, jak probíhá vyhledávání v hotovém kompaktním suffixovém automatu.

Nechť $w \in \Sigma^*$, u je podslovo slova w. Vyhledávání pracuje se strukturou $Automat(w)$ a podslovem u. Automatem se prochází, dokud se nedojde do vrcholu, který odpovídá zadanému podslovou. Procházení je prováděno tak, že se najde hrana, jejíž první znak odpovídá současnému podslovu. Jakmile se taková hrana najde, porovnají se ostatní znaky a přejde se do následujícího explicitního vrcholu. V případě, že zbytek podslova je kratší než popisek hrany, znamená to, že vrchol odpovídající zadanému podslovu je implicitní. V takovém případě se vrátí jeho nejbližší explicitní potomek a poznamená se, kolik znaků zbývalo k tomuto explicitnímu potomkovi.

Následně se z tohoto vrcholu spustí backtracking a počítá se suma délek hran, přes které je potřeba projít, než se dojde do stoku. Jakmile se dojde do stoku, uloží se současná hodnota do množiny a zkouší se další cesta.

Na výstupu je množina délek cest, které vedou z daného explicitního vrcholu do stoku. K těmto délkám se přičte poznamenaná hodnota, kolik bylo potřeba znaků, aby se došlo z implicitního vrcholu, do jeho nejbližšího explicitního potomka. Pokud se následně přičte ke každému číslu délka zadaného podslova, odpovídá každé číslo jedné pozici daného podslova v původním slově počítaném od konce.

Algoritmus 3 popisuje, jak se v automatu vyhledává. Pro připomenutí by se mělo zmínit, že řetězce se indexují od jedničky. Na řádku sedm se počítá, kolik znaků zbývalo do nejbližšího explicitního vrcholu.

Algoritmus 3 Vyhledávání v automatu

1: **Algoritmus** pro vyhledávání ve struktuře Automat(w)
2: nechť *Automat*(w) je automat na vstupu a $u \in \Sigma^*$ je hledané podslovo;
3: *současný* ← zdrojový vrchol *Automat*(w);
4: *index* ← 1
5: **dokud** *index* ≤ $|u|$ **opakuj**
6: (*současný*, *index*) ← Další vrchol(*současný*, u, *index*, w);
7: *zbývalo* ← *index* − $|u|$ − 1;
8: **vrať** Všechny cesty(*současnýVrchol*, *zbývalo*);

9: **funkce** Další vrchol($s, u, index, w$)
10: nechť $(s, (k, p), v)$ je hrana vedoucí z s začínající na $u[index]$;
11: $i \leftarrow k$;
12: **dokud** $i \leq p$ **opakuj**
13: **pokud** index > |u| **pak**
14: **vrať** $(v, index + (p - i + 1))$;
15: **pokud** $w[i] \neq u[index]$ **pak**
16: u není podslovo w;
17: $index \leftarrow index + 1$; $i \leftarrow i + 1$;
18: **vrať** $(v, index)$;

19: **funkce** Všechny cesty(v, *délka*)
20: nechť M je prázdná množina;
21: **pokud** v je stok **pak**
22: přidej číslo *délka* do M;
23: **pro všechny** hrany $(v, (k, p), r)$ vedoucí z v **opakuj**
24: $M \leftarrow M \vee$ VšechnyCesty(r, *délka* $+(p - k + 1)$);
25: **vrať** M;

Kapitola 4

Implementace

V této kapitole bude popsána implementace algoritmu pro přímou konstrukci kompaktního suffixového automatu. Při implementaci byla snaha o co největší podobnost s popsaným pseudokódem, aby po nastudování algoritmu bylo její pochopení co nejjednodušší. V první části je popsáno, jakým způsobem je automat reprezentován. Následně jsou popsány některé implementační detaily a změny oproti algoritmu 2. Na konci kapitoly je popsána implementace vyhledávání v automatu.

4.1 Reprezentace automatu

Třída automatu je šablonovaná `template<class SymbolType>`, aby bylo možné použít nějakou obecnou abecedu a slova nad ní a ne nutně datový typ `char`. Vrcholy jsou celá čísla, která se používají jako index do vektoru hran.

Popisky hran jsou i ve výsledném automatu uchovávány jako indexy pozic, odkud kam se ve slově, pro které je automat sestavený, nachází daný popisek. Je tedy potřeba stále držet původní slovo. Automat z tohoto důvodu má vektor symbolů, který je indexovaný od 1. Obsah na pozici 0 je nedefinovaný. Samotné hrany jsou z důvodu co největší podobnosti s pseudokódem reprezentovány následovně: `vector<map <pair<size_t , size_t >,unsigned>>`.

Už bylo řečeno, že index do vektoru odpovídá číslu vrcholu, kde zdroj má index 0. Na dané pozici ve vektoru se nachází všechny hrany daného vrcholu. Hrany jsou reprezentovány jako mapa, kde klíčem do ní je pár dvou celých čísel. Tento pár reprezentuje popisek hrany, který byl v algoritmu většinou značen jako (k, p). Prvkem mapy je číslo vrcholu, do kterého hrana vedla. Výsledný automat už neobsahuje pomocný vrchol $\perp$ a otevřené hrany mají pevně zadanou délku na velikost vstupního slova.

4.2 Konstrukce kompaktního suffixového automatu

Samotný algoritmus pro konstrukci je reprezentován třídou, která má jednu veřejnou statickou metodu `construct`. Metoda je šablonovaná tak, aby pracovala s obecnou abecedou, jak bylo řečeno výše. Metoda přijímá jeden parametr, kterým je slovo, pro které má být automat zkonstruován a vrací hotový automat, který byl popsán v předchozí části.

Metoda však nekonstruuje přímo automat, který se nakonec vrací. Místo toho si udržuje vlastní typ, který je definovaný soukromou třídou. Jakmile je tato interní reprezentace automatu kompletně zkonstruována, překopírují se hrany do nového automatu, který je následně navrácen. Tento interní automat si udržuje vstupní řetězec jako mapu, kde klíčem je celé číslo a prvkem SymbolType. Důvodem je nutnost použití záporných indexu, které jsou využívány u pomocného vrcholu $\perp$. Bylo zvoleno toto řešení, místo použití vektoru, ve kterém by indexy byly posunuty, protože by takový přístup znepřehlednil implementaci a převod na výsledný automat by byl náročnější. Dále jsou v interním automatu uloženy suffix linky a délky vrcholů, které ve výsledku nejsou potřebné.

Oproti algoritmu, uvedenému v této práci, se funkce pro konstrukci jmenují tak, jak byly pojmenovány v původním článku [3]. Záměna jmen je zapsána v tabulce 4.1.

Aktualizuj	-	*update*,
Zkonstroluj LRS	-	*check_end_point*,
Minimalizuj pár	-	*canonize*,
Rozděl hranu	-	*split_edge*,
Vrať explicitní vrchol	-	*extension*,
Přesměruj hranu	-	*redirect_edge*,
Rozděl vrchol	-	*separate_node*.

Tabulka 4.1: Záměna jmen v implementaci.

Významným implementačním rozdílem, oproti algoritmu je použití globální proměnné e. Jak bylo u popisu algoritmu několikrát řečeno, otevřené hrany a délka stoku pracuje s ukazatelem na proměnnou e, která se rovná délce slova, pro které je v současnou chvíli automat zkonstruován. S přidáním každého dalšího znaku se tato hodnota inkrementuje a s tím je potřeba, aby se změnili i hodnoty u otevřených hran a délky stoku. V implementaci bylo zvoleno řešení, které se poněkud vrací k Ukkonenově algoritmu pro konstrukci suffixového stromu (algoritmus 1). V tomto algoritmu je popisek otevřených hran (p, ∞).

Po zvážení různých možností, *jako třeba vytvoření třídy, která bude reprezentovat jak číslo, tak ukazatel na e a bude vracet správnou hodnotu, s čímž by souvisela nutnost změny reprezentace hran, protože mapa nedokáže pracovat*

s měnícím se klíčem, se došlo k závěru, že nejlepší možnost je inspirovat se u Ukkonenova algoritmu a použít nekonečno. V implementaci se tedy všude, kde má být přiřazena proměnné e, přiřadí INT_MAX. S tím bohužel souvisí, že ve funkcích *Rozděl hranu, Minimalizuj pár* a *Rozděl vrchol* je třeba konstrolova, jestli se nalezená hrana (respektive délka) nerovná INT_MAX a v kladném případe nahradit hodnotu, se kterou se pracuje, hodnotou e.

Poté, co je interní automat zkonstruován, jsou nahrazeny všechny výskyty INT_MAX v otevřených hranách koncovou hodnotou e. Po této úpravě je vytvořena instance automatu, kterému se postupně předají hrany všech vrcholů, kromě pomocné $\perp$. Nakonec je tento automat vrácen.

Jak již bylo řečeno, byla snaha, aby se implementace podobala popsanému algoritmu. Je přidáno několik pomocných funkci, ty ovšem dělají jen věci, které není třeba rozebírat. Příkladem jsou funkce, které hledají hranu vrcholu, která začíná na určitý znak, což je potřeba například na řádcích 38 nebo 46 v algoritmu 2. Dalším příkladem je funkce, která vytvoří nový vrchol s kopií všech jeho hran (řádek 61).

4.3 Vyhledávání v automatu

Vyhledávání v automatu je reprezentováno třídou, která má jednu statickou metodu `query`. Metoda přijme reprezentaci automatu uvedenou v části 4.1 a nějaké podslovo slova, pro které byl automat sestaven.

Metoda vrátí `set<unsigned>`, což odpovídá množině pozic, jak bylo popsáno v kapitole 3.

Kapitola 5

Porovnaní struktur

V této kapitole jsou srovnány různé indexové struktury vzhledem k paměťové a časové efektivitě.

5.1 Paměťová složitost

V tabulce 5.1 jsou vypsány počty vrcholů a hran, které udržuje trie, strom a kompaktní automat pro různě dlouhá slova nad různě velkou abecedou.

U stromu a trie není uveden počet hran, protože ten se bez jedné rovná počtu vrcholů. U trie není ani uveden počet vrcholů pro delší slova. Slova byla náhodně generována, takže by se v nich neměly vykytovat žádné vzory. První sloupeček určuje délku slova a druhý velikost abecedy. Hodnota ve třetím, čtvrtém a pátem sloupci odpovídá počtu vrcholů počet vrcholů, z kolika se daná struktura skládala. Poslední sloupec navíc odpovídá počtu hran v kompaktním suffixovém automatu.

Trie má asymptoticky kvadratické množství vrcholů. Konkrétně je pro trii nejhorší případ, kdy se v daném slově nevyskytuje žádný znak vícekrát. V takovém případě žádné dvě přípony nebudou sdílet stejný vrchol a za každou příponu slova se přidá tolik vrcholů, jaká je délka slova. Přípon je bez prázdné n a vrcholů je v takovém případě $\frac{n(n+1)}{2} + 1$, kde $+1$ je za kořen trie.

Strom a automat mají naopak méně vrcholů s tím, čím je větší abeceda. Pro představu, proč tomu tak je, je dobré si opět představit případ, kdy se ve slově žádný znak nevyskytuje vícekrát.

Kdyby se ve slově nevyskytoval žádný znak vícekrát, bude mít $Automat(w)$ jen dva vrcholy (bez pomocného). S každým přidaným znakem by se totiž jen prodloužily existující hrany a vytvořila by se jedna nová vedoucí z ε do stoku na tento nový znak. Počet hran by se v takovém případě rovnal délce zadaného slova.

$\|w\|$	$\|\Sigma\|$	Trie	Strom	Automat	Hran
100	2	4 635	196	78	159
	4	4 904	161	55	148
	8	5 001	146	44	141
	16	5 053	131	30	128
	32	5 081	132	32	131
1 000	2	492 524	1 993	725	1 456
	4	497 371	1 616	550	1 483
	8	498 889	1 461	429	1 395
	16	499 678	1 335	331	1 326
	32	500 147	1 305	299	1 292
10 000	2	49 893 442	19 989	7 562	15 134
	4	49 956 636	16 231	5 468	14 667
	8	49 977 782	14 591	4 324	14 044
	16	49 988 279	13 757	3 663	13 566
	32	49 994 623	12 325	2 286	12 246
100 000	2		199 985	75 450	150 914
	4		162 269	54 576	146 500
	8		145 841	43 099	140 233
	16		137 338	36 130	134 901
	32		131 608	31 254	130 889
1 000 000	2		1 999 980	754 624	1 509 267
	4		1 623 597	548 233	1 469 081
	8		1 450 151	424 392	1 397 646
	16		1 346 372	333 639	1 320 702
	32		1 301 406	295 410	1 289 359

Tabulka 5.1: Srovnání množství vrcholů v různých strukturách.

5.2 Časová složitost

K měření časové efektivity byla použita funkce *clock*, která vrací počet hodinových tiků od spuštění programu. Zde uvedené hodnoty budou uváděny v těchto ticích.

Měření efektivity vyhledávání ve struktuře z časového hlediska proběhlo na náhodných řetězcích délky milion znaků. Testovány byly řetězce nad různě velkými abecedami. Konkrétně byly testovány abecedy velikosti dva, čtyři, osm, šestnáct a třicet dva.

V tabulce 5.2 jsou uvedeny počty hodinových tiků, které byly potřeba pro nalazení náhodného podslova u. Z měřených hodnot je vypsán medián. V tabulce 5.3 jsou uvedena stejná data pro kompaktní suffixový automat.

Z tabulky je vidět, že pokud je hledané podslovo delší, je vyhledávání

v náhodně vygenerovaném slově rychlejší. Důvodem je, že delší podslovo u se ve w vyskytuje méně často a při procházení automatu se dojde do stavu, který je blíže stoku. Všech existujících cest z tohoto vrcholu do stoku je poté méně.

Dále je vidět, že s větší abecedou klesá doba vyhledávání. Pro takovou naivní představu, proč tomu tak je, je možné říct, že čím větší je abeceda, tím více může mít jeden stav přechodů, čímž se šetří počet stavů, díky čemuž je rychlejší vyhledávání. U abecedy o dvou znacích nemůže mít stav více než dva přechody, protože by v takovém případě byl automat nedeterministický. Naopak ve slově, kde by se žádný znak nevyskytoval vícekrát, by automat měl pouze dva vrcholy, jak bylo popsáno výše.

Rozdíl mezi stromem a kompaktním automatem z hlediska časové efektivity není nijak závratný. Hlavní výhoda automatu je v ušetřené paměti, protože má mnhoem méně stavů, jak je také vidět z tabulky 5.1.

$\|u\|/\|\Sigma\|$	2	4	8	16	32
1	1 680 882	655 082	290 534	134 943	57 348
2	796 540	141 438	28 678	5 660	1 229
4	168 876	6 807	298	27	12
8	8 867	28	10	11	12
16	32	13	9	10	10
32	13	11	9	10	9

Tabulka 5.2: Rychlost vyhledávání ve stromě

$\|u\|/\|\Sigma\|$	2	4	8	16	32
1	1 582 860	617 691	255 690	104 227	44 490
2	716 259	129 075	24 773	4 670	1 080
4	166 026	5 467	273	20	13
8	7 007	30	9	9	13
16	30	11	8	9	10
32	12	8	9	9	9

Tabulka 5.3: Rychlost vyhledávání v automatu

Závěr

Cílem této práce byla implementace algoritmu pro přímou konstrukci kompaktního suffixového automatu do algoritmové knihovny ALIB.

Po nastudování několika článků byl pro tuto konstrukci zvolen algoritmus, který vychází z Ukkonenova algoritmu pro přímou konstrukci suffixového stromu. Algoritmus má, oproti jiným, výhodu v tom, že je on-line, což znamená, že je možné automat konstruovat proudově a není potřeba znát dopředu celé slovo, pro který bude automat stavěn.

Aby byl automat prakticky použitelný, byly naimplementovány funkce pro vyhledávání v něm. Vyhledávání přijme podslovo a vrátí množinu pozic, kde se dané podslovo nachází v původním slově, pro které byl automat zkonstruován.

Vedlejším produktem práce je implementace Ukkonenova algoritmu pro konstrukci suffixového stromu.

Implementace jsou napsané tak, aby co nejpřesněji odpovídaly zápisu popsaným autory algoritmu, díky čemuž by se mělo případnému čtenáři zjednodušit pochopení všech implementací.

Struktury byly testovány na náhodně generovaných datech a z výsledků je vyplývá, že hlavní výhodou kompaktního suffixového automatu oproti suffixovému stromu je v ušetřené paměti. V rychlosti vyhledávání je automat také rychlejší, i když ne nijak zásadně.

Literatura

[1] Crochemore, M.; Vérin, R.: Direct construction of compact directed acyclic word graphs. In *Annual Symposium on Combinatorial Pattern Matching*, Springer, Springer Berlin Heidelberg, 1997, ISBN 978-3-540-69214-0, s. 116–129.

[2] Melichar, B.: *Jazyky a překlady*. Vydavatelství ČVUT, 2003, ISBN 978-80-01-02776-9.

[3] Inenaga, S.; Hoshino, H.; Shinohara, A.; aj.: On-line construction of compact directed acyclic word graphs. *Discrete Applied Mathematics*, ročník 146, č. 2, 2005: s. 156–179.

[4] Kolář, J.: *Teoretická informatika*. 2004, ISBN 978-80-01-04331-8.

[5] Mareš, M.; Valla, T.: *Průvodce labyrintem algoritmů*. CZ. NIC, zspo, 2017.

[6] Blumer, A.; Blumer, J.; Haussler, D.; aj.: Complete inverted files for efficient text retrieval and analysis. *Journal of the ACM (JACM)*, ročník 34, č. 3, 1987: s. 578–595.

[7] Inenaga, S.: Bidirectional construction of suffix trees. *Nord. J. Comput.*, ročník 10, č. 1, 2003: str. 52.

[8] McCreight, E. M.: A space-economical suffix tree construction algorithm. *Journal of the ACM (JACM)*, ročník 23, č. 2, 1976: s. 262–272.

[9] Blumer, A.; Blumer, J.; Haussler, D.; aj.: The smallest automation recognizing the subwords of a text. *Theoretical computer science*, ročník 40, 1985: s. 31–55.

[10] Ukkonen, E.: On-line construction of suffix trees. *Algorithmica*, ročník 14, č. 3, 1995: s. 249–260.

[11] Crochemore, M.; Vérin, R.: On compact directed acyclic word graphs. In *Structures in Logic and Computer Science*, Springer, 1997, s. 192–211.

Příloha A

Seznam použitých zkratek

DKA Deterministický konečný automat

LRS Nejdelší opakovaná přípona (longest repeated suffix)

NKA Nedeterministický konečný automat

Příloha **B**

Obsah přiloženého datového úložiště

- `readme.txt` ..... stručný popis obsahu
- `Latex`
  - `BP_Sedlacek_JosefErik_2018.tex` ... zdrojový kód práce ve formátu LaTeX
  - `cvut-logo-bw.pdf` ..... logo čvut
  - `FITthesis.cls` ..... šablona práce
  - `ref.bib` ..... použitá literatura ve formátu bib
  - `VztahStruktur.pdf` .... obrázek znázorňující vztah mezi trií, stromem a automatem
  - `ObycejnyVSKompaktni.pdf` .... obrázek srovnávající konečný automat s jeho kompaktním ekvivalentem
  - `nedeterministicky.pdf` ..... obrázek s ukázkou nedeterministického suffixového automatu
  - `cocoNacocoa.pdf` ..... obrázek převodu struktury $Automat(w)$ na $Automat(wa)$
  - `csn690.bst`
- `BP_Sedlacek_JosefErik_2018.pdf` ..... text práce ve formátu PDF
- `Implementace.cpp` ..... Upravená a samostatně spustitelná verze implementace