

Obsah

I	Teoretická část	4
1	Obecně o predikci textu	4
1.1	Definice pojmu prediktivní text	4
1.2	Automatické opravy a predikce textu	4
1.3	Úskalí jazyků s rozvinutou flexí	4
2	Klíčové vlastnosti systému pro predikci textu	5
3	Smysl automatického doplňování pro uživatele	6
3.1	Predikce jako způsob zjednodušení zápisu	6
3.2	Predikce jako inspirace	7
4	Jak predikce textu funguje	8
4.1	Slovníkový model	8
4.1.1	Frekvenční slovníky	9
4.2	Statistický model	9
5	Měření účinnosti zadávání textu	10
5.1	KSPC	10
5.1.1	Primitivní výpočet KSPC	10
5.1.2	Jazykový model	11
5.1.3	Obecný výpočet KSPC	11
5.1.4	Obecné dělení metod zadávání dle KSPC	11
5.2	WPM	12
5.3	Vyhodnocování chyb	12
6	Rozšířené metody zadávání textu	12
6.1	Klávesnice ITU	12
6.1.1	Multitap	13
6.1.2	Slovníková disambiguace	13
6.1.3	Prefixová disambiguace	15
6.1.4	LetterWise	15
6.1.5	WordWise	15
6.2	QWERTY klávesnice na chytrých telefonech	16
6.2.1	Barevné zvýrazňování kláves	16
6.2.2	Predikce celých slov	17
6.2.3	Swype	18
6.2.4	Predikce na fyzických QWERTY klávesnicích	18
II	Praktická část	20

7	Analýza	20
7.1	Backend	20
7.1.1	Forma komunikace	21
7.2	Frontend	21
8	Návrh	22
8.1	JavaScript	22
8.2	AJAX	22
8.3	jQuery	23
8.4	HTML	23
8.5	CSS	24
9	Implementace	24
9.1	Uživatelské rozhraní	24
9.2	Zpracování textu v editoru	25
9.3	Zobrazení predikcí	26
9.3.1	Alternativní predikce	27
9.4	Vkládání predikcí	28
10	Závěr	28

Úvod

Různé implementace predikce textu se objevují v mobilních telefonech a elektronických osobních asistentech již přes dvě dekády, přičemž jejich hlavním úkolem je usnadnit zadávání textu na klávesnicích mobilních zařízení. V poslední době se s rozmachem chytrých telefonů objevily systémy predikce textu (a automatické opravy slov, která s ní souvisí) i pro jednoznačné klávesnice typu QWERTY, a to z důvodu zvýšení komfortu psaní. Tyto systémy mají ve většině dvě nevýhody. První z nich je závislost na konkrétním operačním systému, konkrétním zařízení (především v případě starších telefonů) či na konkrétní aplikaci, která zajišťuje zobrazení klávesnice na dotykovém displeji. Takové systémy tudíž nejsou použitelné například pro psaní na stolním počítači na klasické QWERTY klávesnici. Druhou nevýhodou je pak fakt, že systémy většinou musí spoléhat pouze na lokálně uložená jazyková data, která tak musí být relativně kompaktní co do datové velikosti a nemusí tedy obsahovat potřebné množství dat, aby uživatel nebyl omezován nedokonalostmi jazykového modelu.

Protipólem velkého množství aplikací pro predikci textu pro mobilní zařízení je pak téměř prázdná množina takových aplikací pro klasické počítače se standardní klávesnicí. Textové procesory typu LibreOffice Writer¹ mívají vestavěné primitivní způsoby doplňování, které ale čerpají data pouze z textu, který je v danou chvíli v editovaném dokumentu. Textové editory typu Atom² mívají podobnou funkcionalitu, navíc například při editaci programového kódu nabízí relativně komplexní doplňování, ale pouze podle předem připravených šablon.

Cílem této práce je tedy vytvořit implementaci systému predikce textu, která bude čerpat data z velkého korpusu textů, a jednak bude jednak nezávislá na platformě. Vzhledem k tomu, že korpusová data není možné uchovávat na straně klienta, bylo logickým krokem zvolit model klienta a serveru, který bude zajišťovat generování predikcí. Tento systém na serveru již v době tvorby této práce existoval, což umožnilo soustředit se především na vývoj uživatelského prostředí.

Pro zajištění co nejvyšší kompatibility s různými systémy byla zvolena forma zásuvného modulu do webového HTML editoru CKEditor³. Použitými technologiemi tedy byl především JavaScript pro programovou logiku, AJAX pro komunikaci se serverem a HTML a CSS pro uživatelské rozhraní.

¹<https://www.libreoffice.org/discover/writer/>

²<https://atom.io/>

³<http://ckeditor.com/>

Část I

Teoretická část

1 Obecně o predikci textu

1.1 Definice pojmu prediktivní text

Prediktivní text, neboli automatické doplňování textu (angl. autocomplete) je funkce v aplikaci, která na základě uživatelského vstupu (většinou prvních několika znaků slova) doplňuje zbytek části textu (většinou slova). V ideálním případě funkce jednoznačně rozpozná, co chtěl uživatel napsat, a nabídne mu právě to doplnění, jež uživatel chce. V praxi obvykle uživatel dostává na výběr z několika možností, které systém vyhodnotil jako nejpravděpodobnější doplnění vstupu. Systémy se liší způsobem zužování výběru možností a jejich řazením, stejně tak jako vizuálním podáním nabídky.

Všechny systémy automatického doplňování textu fungují nejlépe v implementacích na omezených doménách, tzn. například doplňování e-mailových adres v e-mailových klientech, klíčových slov určitého programovacího jazyka v programátorských editorech či doplňování ve formulářových polích s předem vymezeným očekávaným vstupem.

1.2 Automatické opravy a predikce textu

Automatické opravy a predikce textu jsou dva velmi úzce svázané pojmy. Tím prvním se myslí opravení textu poté, co jej uživatel chybně zadal, tím druhým pak nabídnutí uživateli text předtím, než jej zadal. Na pomezí těchto dvou technik se ještě vyskytuje třetí, a to disambiguace vstupu z nejednoznačné klávesnice. Všechny tyto tři techniky jsou úzce spjaty proto, že vyžadují nějakým způsobem odhadnout, co uživatel chtěl, nebo bude chtít napsat.

Ačkoliv je praktická část této práce zaměřena na predikci textu a opravami se nezabývá, zdálo se vhodné popsat i vývoj a některé způsoby řešení problematiky disambiguace vstupu a automatických oprav, protože řeší podobné problémy.

1.3 Úskalí jazyků s rozvinutou flexí

Systémy pro predikci textu určené k tvorbě volného textu jsou obvykle, jak bude zřejmé z dalšího popisu jednotlivých aplikací, založeny na více či méně rozsáhlém lokálně uloženém korpusu slov či slovních spojení, ze kterého vybírají možná doplnění. Tyto systémy zpravidla umožňují uživateli přidávat manuálně slova, která napsal a nejsou v základním korpusu, což slouží jako základní uživatelské přizpůsobení. Lokální korpus má ovšem tu nevýhodu, že jeho velikost je značně limitována možnostmi zařízení, tj. jeho pamětovou a výpočetní kapacitou. To je problém především pro jazyky s rozvinutou flexí, jako je čeština, které potřebují na rozdíl analytických jazyků,

např. angličtiny, poněkud větší množství tvarů jednotlivých slov.

Rozdílnost angličtiny a češtiny v ohledu množství slovních tvarů připadajících na jedno lemma lze porovnat například na dvou velkých korpusech, enTenTen13 a czTenTen12. Jde o rozsáhlé korpusy vytvořené z textů získaných na Internetu, které byly vyčištěny a zbaveny duplikátů nástrojem Onion⁴. Anglický korpus enTenTen13 obsahuje 39 452 714 slov a 37 366 565 lemmat (základních slovních tvarů) (Sketch Engine, 2015a). Naproti tomu český korpus czTenTen12 obsahuje slov 18 725 879 a lemmat pouze 13 976 481 (Sketch Engine, 2015b). V češtině tedy podle výše uvedených dat připadá na jedno lemma 1,339 slovního tvaru, naproti tomu v angličtině na jedno lemma připadá pouze 1,052 slovního tvaru. Je tedy vidět, že slovník pro češtinu by v ideálním případě měl být minimálně 1,27× větší než slovník o stejném množství základních tvarů pro angličtinu.

doplnit to info z majky o teoretickém poměru?

Tento fakt komplikuje implementaci slovníkových jazykových modelů do aplikací pro predikci českého textu, protože je nutno uchovávat větší množství slovních tvarů než pro angličtinu. Stejně tak je výrazně větší potřeba zohledňovat kontext, aby byly slovní tvary vybírány správně.

tady by se hodilo něco ocitovat, akorát že já jsem si to vymyslel.. nějaký nápad?

2 Klíčové vlastnosti systému pro predikci textu

Navzdory velmi odlišným způsobům použití systémů pro doplňování textů jsou některé jejich klíčové vlastnosti pro většinu implementací společné. Studie, které zkoumaly efektivitu těchto systémů, obecně docházejí ke podobným závěrům co se požadavků na tyto systémy týče. Ať jde o vyhledávač typu Google⁵ nebo Yahoo⁶, o našeptávač e-mailových adres v e-mailovém klientu nebo o predikci volného textu v textovém editoru, aby uživatel měl při užívání aplikace pocit, že mu doplňování textu skutečně pomáhá spíše než překází, je klíčových několik vlastností. Primární je rychlost, s níž aplikace možná doplnění uživateli zobrazuje. Nandi a Jagadish uvádí,

že by odezva neměla převyšovat 100 ms (Nandi 2007). Tento faktor je rozhodující pro to, zda si uživatel zvolí nabídnutá doplnění použít, nebo raději napíše zbytek textu sám. V případě implementace predikce textu do editoru, kde uživatel zadává volný text, jímž si je relativně jist, je tato vlastnost pro uživatelské rozhodnutí patrně klíčová. Naproti tomu ve vyhledávání lze očekávat, že uživatel bude ochoten počkat delší chvíli na zobrazení doplnění. S tímto souvisí i další vlastnost – umožnění uživateli pohodlně nabízené predikce ignorovat. (Ward et al., 2012) Rozhodne-li se totiž, že např. bude rychlejší text manuálně napsat a predikce nepoužít, je žádoucí, aby mu grafické zpracování nabídky žádným způsobem neztěžovalo takto učinit. Z tohoto důvodu je ve většině systémů nabídka předvídaných výrazů zobrazována pod textovým polem (např. automatické doplňování ve vyhledávači Google (Google, 2015)).

Zobrazení nabídky v textových editorech, kde se nejedná o jednořádkové pole s

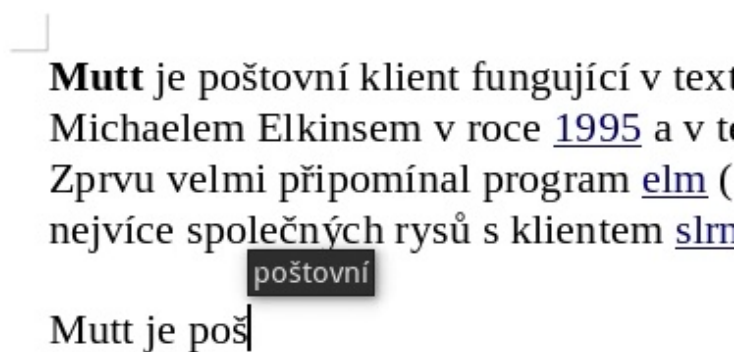
⁴<http://corpus.tools/wiki/Onion>

⁵<https://www.google.com>

⁶<https://www.yahoo.com>

očekávaným vstupem pouze několika jednotlivých slov, je předmětem několika studií, z nichž např. jedna navrhuje zobrazovat nabídku na spodní části obrazovky, aby uživatel nebyl nucen přesouvat pozornost příliš daleko od klávesnice. Lze ale polemizovat nad tím, jak je tato argumentace validní pro uživatele bez motorického nebo jiného postižení, jelikož tací většinou při psaní na klávesnici nehledí. Textové editory pro psaní programového kódu tento problém většinou řeší zobrazováním nabízených doplnění v menu nad nebo pod kurzorem (tedy zhruba podobně jako jednořádková vyhledávací a jiná pole). Podobně to řeší i kancelářský balík LibreOffice, který zobrazuje po napsání prvních tří písmen slova jednu predikci nad kurzorem. Výběr predikce se odvíjí od frekvence, přičemž velký důraz je kladen na délku nabízeného slova (obrázek 1 na straně 6). Tento systém uživateli umožňuje udržovat pozornost na relativně malé oblasti obrazovky, čímž se zvyšuje pravděpodobnost, že se rozhodne doplnění použít.

ocitovat



Obrázek 1: Implementace predikce textu v LibreOffice Writer 4.2.8.2

Lze tedy říci, že nabídka slov, která daný systém vyhodnotil jako vhodná doplnění, by neměla být obtěžující, mělo by být snadné ji ignorovat a měla by se nacházet co nejblíže místu, kam je soustředěna uživatelova pozornost. To, jak bylo naznačeno výše, může být odlišné v závislosti na cílové kategorii uživatelů.

3 Smysl automatického doplňování pro uživatele

3.1 Predikce jako způsob zjednodušení zápisu

Důvody, pro které si může uživatel vybrat užívat prediktivní text, se liší podle přístroje, na němž píše, a dalších okolností.

Na mobilních zařízeních, která disponují nejednoznačnou klávesnicí (typicky starší mobilní telefony s 12tlačítkovou klávesnicí) je predikce výhodná proto, že umožňuje zadávat text jediným stisknutím kláves, které reprezentují znaky obsažené v psaném

slově. Bez predikce by byl uživatel nucen v případě písmene, které na klávese není na první pozici, stisknout tutéž klávesu vícekrát po sobě⁷.

Dalším případem jsou mobilní zařízení, která mají dotykový displej, kde je zase problém s přesností, s jakou je uživatel schopen se na klávesy trefit svým prstem nebo stylusem. Tento případ sice spadá do kategorie automatických oprav, ale může být kombinován s prediktivním textem (Kocienda et al., 2012).

Při psaní na jednoznačné fyzické klávesnici (typicky klasická QWERTY klávesnice, jež je pro jednoduchost v této práci považována za reprezentanta všech ostatních lokalizovaných rozložení) obvykle není predikce potřeba, ale například pro uživatele fyzicky postiženého může být výhodné mít k dispozici systém, který mu ušetří stisky kláves nutné k zapsání celého slova.

3.2 Predikce jako inspirace

Vedle původního určení predikce textu, kterým je zrychlit zadávání textu a tím komunikaci na rozhraní člověk–počítač, může pro běžného uživatele automatické doplňování textu plnit ještě několik dalších funkcí.

Takovým případem může být doplňování ve vstupním poli vyhledávače typu Google, případně jeho obdoby v nějaké menší síti, například univerzitní knihovně. (Google, 2015)

Ward et al. ve své studii Autocomplete as a Research Tool uvádí, že studenti, na kterých testoval automatické doplňování ve vyhledávání v rámci informačního systému knihovny, poukázali v následném dotazníku na několik dalších funkcí, které pro ně predikce plnila.

Na otázku, jak by svým přátelům vysvětlil, k čemu je našeptávač dobrý, odpověděl jeden student, že rozhodně jako asistence v pravopisu. Studenti totiž často nemají z ústních zadání v hodinách úplně přesnou představu, jak se např. jméno autora, jehož díla jim jejich vyučující doporučil jako zdroj informací k práci, přesně píše. Autoři studie ze záznamů hledání vyvodili, že někteří studenti si ani nemuseli být vědomi faktu, že začátek jména autora zadali špatně, protože predikce jim nabídla rovnou správnou verzi a oni ji vybrali, aniž by přemýšleli, v čem se nabídka liší od jejich vstupu.

Mezi další výhody, které studenti uvedli ve zmíněné studii, patřilo například to, že automatické doplňování jim potvrdilo, že podobné téma již někdo před nimi hledal a jejich postup je tedy pravděpodobně správný.

S tímto souvisí i jedna odpověď od dalšího účastníka studie, která uvádí, že našeptávač pro ně fungoval jako nástroj pro dotvoření myšlenky, jako brainstorming od počítače.

Z výše uvedeného lze tedy vyvozovat, že automatické doplňování textu na klasickém počítači skutečně nemá význam pouze pro uživatele tělesně či jinak postižené, kteří mohou mít problém se samotným zadáváním textu, ale může posloužit i běžným uživatelům jako nástroj pro kontrolu toho, zda to, co píšou, je správně, k tématu

⁷Více o klávesnicích ITU v kapitole 6.1.

a podobně. V základním rozdílem oproti předchozím případům využití predikce je fakt, že uživatel nemusí přesně vědět, co chce napsat, předtím, než to vidí jako doplnění. (Ward et al., 2012)

4 Jak predikce textu funguje

Prakticky všechny aplikace, které se snaží usnadnit zadávání textu, se snaží o předvídání dalšího slova, jež uživatel může chtít napsat. Existují dva různé přístupy umožňující takového výsledku dosáhnout.

4.1 Slovníkový model

Pokud prediktivní systém využívá slovníkového modelu, znamená to, že má v paměti zařízení uložený slovník (seznam slov) společně s jejich frekvencemi v referenčním korpusu, tedy tzv. frekvenční slovník (detaily v kapitole 4.1.1). Implementace frekvenčních slovníků v aplikacích pro predikci textu se pravděpodobně značně liší a vzhledem k tomu, že jde o uzavřené komerční systémy, není možné konkrétní implementace zjistit. (Henry, 2014)

U systémů na mobilních telefonech s ITU klávesnicí výběr kandidátského slova probíhá tak, že systém vytvoří kombinace všech slov, která mohou vzniknout z písmen na stisknutých klávesách a vzniklá slova porovná se slovníkem. Průnik těchto dvou množin pak byl uživateli nějakým způsobem prezentován.

Systémy s plnou QWERTY klávesnicí na mobilních zařízeních čelily, jak bylo naznačeno výše, jinému problému. Jelikož je velikost zařízení omezená a klávesy klávesnice jsou tudíž relativně malé, je vysoká pravděpodobnost, že se uživatel dopustí chyby při zadávání vstupu. (Kocienda et al., 2012)

Jedno z řešení je následující. Po získání vstupu z klávesnice se vytvoří permutace, které jsou pak porovnávány se slovníkem. Permutace jsou v tomto případě řetězce, jež byly vytvořeny tak, že pro každý znak A v původním zadaném řetězci O byl vytvořen nový řetězec P obsahující na pozici původního znaku A znak A nebo znak, který byl na klávesnici původnímu znaku A sousedem. (Kocienda et al., 2012) Pro slovo *pes* by tak vytvořené permutace při použití klávesnice QWERTY vypadaly takto:

pes, oes, les, ies, kes, prs, pds, pws, pss, per, ped, pew

Dalším řešením, které pravděpodobně bývá kombinováno s výše uvedenou metodou, je hledání slov ve slovníku, která mají od zadaného slova určitou editační vzdálenost. Editální vzdálenost je Levenshteinova vzdálenost, tedy vyjádření míry odlišnosti dvou řetězců minimálním množstvím operací *vložit znak*, *smazat znak*, *prohodit* dva sousední znaky a *nahradit* jeden znak jiným, které je nutno provést, aby z prvního řetězce byl vytvořen druhý. (Levenshtein, 1966) Tato metoda také využívá frekvenčního slovníku, který navíc kombinuje se statistickým modelem pravděpodobnosti výskytu a pravděpodobnosti souvýskytu. (Norvig, 2015)

je to patent applu, urcite toho bude existovat vic, tak se pojistuju xD

4.1.1 Frekvenční slovníky

Frekvenční slovník je slovník, který ke každému heslu uvádí hodnoty spojené s frekvencí daného slova ve zdrojovém korpusu. Obvykle to je hodnota absolutní frekvence daného slova a například Frekvenční slovník češtiny uvádí také průměrnou redukovanou frekvenci, což je hodnota, jež vychází z absolutní frekvence, avšak zohledňuje rozložení daného slova v korpusu. Pokud tedy mají dvě slova stejnou frekvenci, avšak první z nich se vyskytuje pouze v malém množství odborných textů, zatímco výskyty druhého jsou rovnoměrně rozloženy přes celý korpus, druhé slovo dostane vyšší číselnou hodnotu ARF než slovo první. Podle hodnoty ARF tak lze vyvodit, že druhé slovo je obecně známější a užívanější širším publikem. Čermák; Blatná, 2004 V kontextu s predikcí slov to může například znamenat, že je pravděpodobnější, že uživatel chce napsat spíše druhé slovo nežli to první, které je pravděpodobně značně odborné.

Tvorba frekvenčních slovníků byla v dobách před rozvojem počítačů schopných zpracovat velká množství textů poněkud komplikovaná. Vzniklé databáze bylo obtížné aktualizovat, jejich výroba byla náročná a pomalá. Nástup počítačů však většinu těchto problémů odstranil, takže je možné velmi snadno z korpusu vyrobit frekvenční slovník, který bude tak aktuální, jako je samotný korpus. Na rozdíl od ručního zpracování dat navíc počítačové zpracování dovoluje pracovat s mnohem většími korpusy, takže výsledný frekvenční slovník může být i přesnější. Pro srovnání, jeden z prvních frekvenčních slovníků pro angličtinu, The Teachers Word Book of 30,000 words, měl 30 tisíc lemmat (z 13 tisíc slovních rodin (Lorge, 1944)) a byl vyroben z ručně psaného korpusu, který obsahoval 18 milionů slov. (Nation; Waring, 1997) Naproti tomu moderní frekvenční slovník současné americké angličtiny (Davies; Gardner, 2013) byl vytvořen z korpusu současné americké angličtiny, The Corpus of Contemporary American English (COCA), který obsahuje přes 400 milionů slov. Samotný slovník pak sestává z 5000 nejfrekventovanějších slov. Nutno podotknout, že slovník je určen především uživatelům, kteří nejsou rodilí mluvčí a anglicky se učí, což je pravděpodobně důvod, proč obsahuje pouze 5000 nejčastějších slov.

(Alderson, 2007) tvrdí, že ani velké korpusy nemusí být dostačující pro adekvátní zhodnocení frekvenčního rozložení slov v jazyce tak, jak jej vnímají mluvčí daného jazyka. Vychází z výsledků výzkumů zaměřujících se na porovnání hodnocení frekventovanosti určitých slov v jazyce jeho zkušenými uživateli s daty získanými z korpusu. Tyto hodnoty se totiž podle citovaných průzkumů značně rozcházejí a je tedy otázkou, zda z toho vyplývá, že ani zkušení mluvčí nedokážou správně posoudit pravděpodobnost výskytu daného slova, či že data získaná z korpusů nereprezentují zkušenosti mluvčích s jazykem dostatečně přesně.

4.2 Statistický model

Nejak mi porad prijde, ze o tom nic nevim... pls help xD

jest by to chtělo nějaká data o současných frekvenčních slovnících, nebo stačí ty příklady o BNC? Prip. kde je vzít?

ani nevím, jestli jsem to vůbec interpretoval smysluplně a správně

5 Měření účinnosti zadávání textu

Vstupní metody mívají zpravidla dva cíle, jichž se snaží jejich vývojáři dosáhnout. Jedním z nich je omezení počtu nutných úhozů při zadávání textu, druhým pak je rychlost samotného zadávání. Omezování počtu nutných úhozů je aktuální u jednoznačných klávesnic, které zpomalují zadávání textu především nutností stisknout jednu klávesu vícekrát pro zadání některých znaků. Tento problém u jednoznačných QWERTY klávesnic není tak podstatný, takže už spíše záleží na rychlosti zadávání.

Měřit rychlost zadávání textu pak lze několika odlišnými způsoby. Uživatelé, na kterých se testuje, mohou buď psát text předem nepřipravený, tedy ho mohou komponovat sami, nebo ho mohou opisovat, tedy kopírovat. Kopírování se obecně považuje za lepší, protože eliminuje zpoždění nutné k samotnému vymyšlení psaného textu. (Tarvainen, 2010, s. 2)

5.1 KSPC

Metoda měření počtu úhozů na znak, neboli KSPC⁸ (zkratka pro keystrokes per character), je nejužitečnější metrikou pro vyhodnocování těch metod zadávání textu, které se snaží o snížení množství úhozů na znak, jako je například prediktivní systém T9⁹. KSPC tedy udává průměrný počet úhozů nutný k zadání jednoho znaku.

5.1.1 Primitivní výpočet KSPC

Naivní přístup k výpočtu KSPC je takový, že se spočítá počet písmen ve slově (i s mezerou za slovem) a počet úhozů, kterými se na dané klávesnici slovo zadá. Nutno počítat i se stisky speciální klávesy NEXT, kterou je nutno použít v případě, že se dvě po sobě jdoucí písmena nachází na stejné klávese. (Silfverberg et al., 2000, s. 10) Pro metodu multitap na standardní ITU klávesnici by to pro slovo "kniha" byla následující sekvence kláves:

kniha_ : 5566444N442S,

kde N je NEXT a S je znak mezery. KSPC vypočteme jako poměr úhozů na znak:

$$\frac{12}{6} = 2,000.$$

Je ovšem zřejmé, že pokud se nyní stejným způsobem provede výpočet pro slovo "data", bude sekvence úhozů mnohem jednodušší:

data_ : 3282S

a KSPC tedy vyjde:

⁸Přestože název obsahuje stisky kláves, veličina se používá i pro měření účinnosti zadávání textu např. pomocí stylusu na dotykovém zařízení.

⁹Více o predikci T9 v kapitole 6.1.2.

$$\frac{5}{5} = 1,000.$$

Z toho vyplývá, že je nutno provést výpočet tak, aby výsledná hodnota byla průměrná pro daný jazyk. Takový výpočet má dvě prerekvizity. První z nich je jednoznačný popis postupu, kterým lze dojít k zadání jednotlivých znaků, druhou pak jazykový model. Ten je nutný proto, aby výsledné KSPC bylo nezávislé na konkrétním textu v daném jazyce (tedy aby pro daný jazyk bylo průměrné). (MacKenzie, 2002, s. 196)

5.1.2 Jazykový model

Jazykovým modelem je v tomto případě korpus a jeho omezené formy. Tyto formy se liší podle toho, jaká metoda zadávání je testována. Pokud například jde o metodu, která vybírá znaky nezávisle na kontextu (na okolních znacích), stačí mít informace o frekvenčním rozložení jednotlivých písmen v korpusu. Pokud by byl výběr znaku závislý na kontextu jednoho písmene, byl by jazykový model frekvenční slovník bi-gramů. Pro vyhodnocení metod, které využívají k odhadu dalšího písmene kontext celých slov, se použije korpus zredukovaný na unikátní slova a jejich frekvence. (MacKenzie, 2002, s. 197)

5.1.3 Obecný výpočet KSPC

Při výpočtu průměrného KSPC pro daný jazyk a metodu se pak postupuje podle následujícího vzorce:

$$KSPC = \frac{\sum (K_c \times F_c)}{\sum (C_c \times F_c)},$$

kde K_c je počet úhozů nutný pro zadání znaku, C_c je velikost znaku a F_c je frekvence daného znaku v korpusu. Pokud se tedy tento vzorec aplikuje na situaci, kdy metoda zadávání textu vybírá písmeno bez ohledu na kontext, platí, že $C_c = 1$, pokud se vybírá v závislosti na jednom předešlém znaku, je $C_c = 2$. Pokud metoda zohledňuje celá slova, počítá se s průměrnou délkou slov v daném jazyce. Ta se počítá jako vážený průměr počtu znaků ve slovech daného jazyka, který pro BNC vychází na 4,59 znaku; pro srovnání, (Arif; Stuerzlinger, 2009, s. 2) počítá při svých výpočtech s 5 úhozy na slovo, což se považuje za obecný standard. (Tarvainen, 2010, s. 3)

5.1.4 Obecné dělení metod zadávání dle KSPC

Metody zadávání textu lze podle KSPC rozdělit do tří kategorií: ty, pro které je KSPC větší než 1 (např. původní metoda multitap), ty, pro které je rovno 1 (např. QWERTY klávesnice) a nakonec ty, jejichž KSPC dosahuje hodnot nižších než 1 (např. QWERTY klávesnice kombinovaná s predikcí).

Jako základ pro porovnání jednotlivých metod podle KSPC je vhodná standardní klávesnice QWERTY. (Arif; Stuerzlinger, 2009) Uvažují-li se pouze malá písmena anglické abecedy, platí pro ni, že $KSPC = 1$, tedy na jeden znak připadá jeden úhoz, protože pro každý znak má dedikovanou právě jednu klávesu.

5.2 WPM

Počet slov za minutu (WPM, Words per minute) je běžně užívaná metrika pro vyhodnocení rychlosti psaní. (Tarvainen, 2010) Tato metoda nezohledňuje počet úhozů, který během měření byl proveden, pouze výslednou délku přepsaného řetězce. Vzorec pro výpočet WPM (Wobbrock, 2007, s. 48):

$$WPM = \frac{T - 1}{S} \times 60 \times \frac{1}{5},$$

kde T je délka přepsaného řetězce, S počet sekund od prvního úhozu do posledního úhozu. Konstanta 60 reprezentuje počet sekund za minutu a $1/5$ počet znaků na slovo. Aby byla metoda univerzální, počítá se slovo jako pět znaků nezávisle na tom, kolik znaků slova ve výsledném řetězci skutečně mají. Tato standardizace se vyskutuje už od roku 1905. (Yamada, 1980) Odečtení konstanty 1 z počtu znaků ve výsledném řetězci je důležité proto, že většina systémů začíná počítat čas ve chvíli zadání prvního znaku. (Wobbrock, 2007, s. 49)

5.3 Vyhodnocování chyb

Při vyhodnocování rychlosti psaní je důležité zohlednit také chyby ve výsledném textu, jelikož jinak by bylo možné dosáhnout velice dobrých výsledků pouhým rychlým náhodným mačkáním kláves.

Metoda KSPC může být použita k vyčíslení chybovosti tak, že porovná počet znaků zadaných a počet znaků ve výsledném řetězci. Jednou z nevýhod je fakt, že žádným způsobem nerozlišuje chybné znaky od správných, takže vyčísluje pouze množství oprav.

Dalším způsobem, jak změřit chybovost, s jakou uživatel text zadal, je vyhodnocení Levenshteinovy vzdálenosti originálního a přepsaného řetězce. (Soukoreff; MacKenzie, 2001)

6 Rozšířené metody zadávání textu

6.1 Klávesnice ITU

Klávesnice ITU je textová klávesnice na tónovém telefonu, která má na jednu klávesu přiřazeno více písmen. V průběhu vývoje se vyskytovalo několik různých způsobů, jak byla písmena na klávesnici rozřazena, avšak rozdělení písmen podle mezinárodního standardu ITU E.161/ISO 9995-8 je následující:

nejsem schopen to v
tom dokumentu najít -
<http://www.itu.int/rec/T-REC-E.161/en>



Obrázek 2: Klávesnice ITU

6.1.1 Multitap

Na těchto klávesnicích musí uživatel stisknout klávesu s požadovaným písmenem tolikrát, jaké je pořadí daného písmena na klávese. Například na klávese 3 jsou písmena D, E, F, tudíž chce-li uživatel zadat písmeno F, musí klávesu 3 stisknout třikrát. Např. pro napsání slova „nejlepsi_“ tak je nutno stisknout 21 kláves:

nejlepsi : 66335N555337N7777444S,

kde N značí klávesu NEXT a S zadání mezery na konci slova.

Tento systém zadávání se nazývá multitap. Při psaní tímto systémem může KSPC dosahovat hodnot přes 2, protože většina kláves musí být stisknuta vícekrát než jednou. (MacKenzie, 2002, s. 202)

Aby šlo zadávání u klávesnic, které mají méně kláves než je písmen v abecedě, zbavit tohoto problému, bylo nutné nějakým způsobem disambiguovat vstup vzniklý tak, že uživatel vždy stiskne klávesu, na níž se vyskytuje potřebné písmeno, jen jednou. Termín predikce textu je v tomto případě poněkud nepřesný, ale je použitelný, protože systém na základě nejednoznačného vstupu předvídá, odhaduje, co uživatel chtěl napsat. Toto odhadování může být prováděno dvěma způsoby – na základě slovníku a na základě prefixů.

6.1.2 Slovníková disambiguace

Slovníková disambiguace umožňuje uživateli zadávat text stiskem každé klávesy s požadovaným písmenem pouze jednou. Systém potom vyhledává ve slovníku záznamy

korespondující se sekvencí stisknutých kláves, tzn. slova, která obsahují písmena ze stisknutých kláves. Pokud sekvenci stisknutých kláves odpovídá více slov, je uživateli prezentován seznam těchto slov seřazený podle frekvence výskytu.

Tento způsob disambiguace samozřejmě není dokonalý, protože mnoho různých slov sdílí stejné kombinace kláves. Tato situace je obvykle řešena prezentací seznamu možných slov uživateli, který musí pomocí speciálních kláves požadované slovo vybrat. (MacKenzie et al., 2001) Například kombinace kláves 7725 má v češtině možné min. tři interpretace: *prak*, *pral*, *psal*.

Taková slova, která se zapisují stiskem stejných kláves, se nazývají textonyma (Zorn, 2007) a jsou jimi v angličtině např. slova „book“ a „cool“. Zatímco obvykle tento jev pouze snižuje efektivitu zápisu pomocí T9, v roce 2010 záměna dvou textonym v textové zprávě vedla ve Spojených státech k vraždě. (The Bolton News, 2011)

První disambiguační systém byl určen pro lidi s částečně nebo zcela poškozeným slyšením. Tento systém, navržený v roce 1985, si kladl za cíl umožnit neslyšícím telefonní komunikaci, a to za přijatelnou pořizovací cenu (na rozdíl od předchozích existujících řešení) a s tím, že speciální přístroj bude potřebovat pouze neslyšící strana. (Feinson, 1988)

Podle citovaného návrhu se zpráva neslyšící straně předává tak, že protistrana na svém přístroji stiskne pro každé písmeno ve zprávě klávesu, na níž se písmeno vyskytuje (pouze jednou), tyto stisky se převedou na tón o odpovídající frekvenci (podobně jako u tónové volby) a tento tón se přenese po telefonní lince neslyšící straně. Tam je sekvence tónů převedena pomocí přístroje vybaveného mikroprocesorem na sekvenci skupin znaků, které odpovídají daným klávesám, a provede se prohledání slovníku lokálně uloženého v přístroji. Nalezená shoda je pak uživateli zobrazena na obrazovce telefonního přístroje.

T9 – akronym pro *Text on 9 keys* – je systém, který se společně s větším rozvojem mobilních telefonů s klasickými 12klávesovými klávesnicemi se stal všeobecně známým pojmem pro prediktivní text. Byl vyvinut společností Tegic Communications (Nuance Communications, 2007), patentovaný ve Spojených státech (Grover et al., 1998).

Systém T9 umožňuje uživateli rozšiřovat vestavěný slovník tak, že uživatel nové slovo zadá klasickým způsobem multitap a nové slovo se pak uloží do uživatelské databáze (UDB). (Nuance Communications, 2015) Problém nastává ve chvíli, kdy uživatel zjistí, že slovo není ve slovníku, až poté, co zadá příslušnou sekvenci kláves, která by k onomu slovu měla vést. V tu chvíli totiž musí smazat zadaný řetězec, který je buď nesmyslný, nebo to není požadované slovo, a zadat slovo znova pomocí multitapu.

Celkově je však T9 velmi jednoduchá na pochopení a použití a pro její nezávislost na kontextu u ní lze očekávat vždy stejný výstup po stisknutí určité sekvence kláves, takže se zkušený uživatel nemusí při psaní dívat na klávesnici. (MacKay, n.d.)

iTap je systém konkurenční k systému T9 vyvinutý firmou Motorola. Přestože oba systémy využívají slovníkového modelu, iTap je na rozdíl od T9 kontextově závislý a jeho slovník obsahuje kromě jednotlivých slov i běžně užívané fráze.

(MacKay, n.d.) dále uvádí, že ve srovnání se systémem iTap je systém T9 pro uživatele výhodnější v tom smyslu, že nejsou nuceni činit rozhodnutí ohledně toho, zda v určitém momentu zvolit nabízenou predikci, či pokračovat v manuálním psaní. Pro nezkušené uživatele navíc může u systému iTap být demotivující, když jim predikce nabídne slova, které nemínili napsat. Systém iTap je také celkově mírně náročnější na ovládání.

6.1.3 Prefixová disambiguace

6.1.4 LetterWise

LetterWise je systém prediktivního psaní vyvinutý a patentovaný firmou Eatoni Ergonomics. Jeho distinktivním rysem oproti ostatním systémům pro psaní na klávesnicích typu ITU je fakt, že nepracuje se slovníkem, ale s databází pravděpodobností prefixů pro každý používaný jazyk. Které písmeno ze stisknuté klávesy zvolit tedy systém vybírá porovnáním pravděpodobností, s jakými mohou písmena z dané klávesy následovat po předcházejících již zadaných písmenech. V angličtině je například nepoměrně pravděpodobnější, že po sekvenci písmen T a H bude následovat při stisku klávesy 3 písmeno E, nikoliv D či F. Prediktivní systém LetterWise má tedy v paměti zařízení uloženy pouze tyto pravděpodobnosti, nikoliv celá slova. (MacKenzie et al., 2001) To jej činí zajímavým oproti ostatním slovníkovým systémům ze dvou důvodů.

Prvním z nich jsou nízké paměťové nároky. Jazykový model zabírá v zařízení typicky 2–10 kB pro každý jazyk a pro běh potřebuje LetterWise operační paměť o kapacitě v řádu stovek bytů. (Eatoni Ergonomics, 2012)

Druhým, pro uživatele výrazně citelnějším důvodem, je možnost relativně jednoduše zapisovat i neslovníková slova. Pokud se uživatel pokouší totiž zadat slovo, které by sice ve slovníku neexistovalo, ale má skladbu hlásek odpovídající jazyku, jehož databázi prefixů používá, je pravděpodobné, že systém vybere správné písmeno. Další vlastností implicitně vyplývající z faktu, že systém nepoužívá slovník, je, že se uživatel nikdy nemusí uchýlovat k zápisu slov klasickým způsobem (tedy že stiskne klávesu 3 dvakrát, chce-li zadat písmeno E). Při zápisu slov, jejichž posloupnost písmen se neshodává s jazykem databáze, musí uživatel pouze v případě špatného odhadu následujícího písmena stisknout klávesu pro volbu dalšího písmene. Tento systém tedy uživatele nenutí v případě zjištění, že jím požadované slovo není ve slovníku, celé slovo smazat a zadat jej klasickým způsobem. (MacKenzie et al., 2001) (Ghayoomi; Momtazi, 2009, s. 5234)

6.1.5 WordWise

Firma Eatoni Ergonomics vedle systému LetterWise vyvinula ještě jeden méně známý systém – WordWise. Ten je na rozdíl od systému LetterWise založený na doplňo-

vání slov ze slovníku, v čemž je podobnější systému T9. Liší se od něj ale tím, že se vrací k systému LetterWise, pokud se uživatel pokusí zadat slovo, které nemá WordWise ve slovníku. U něj, jak je uvedeno výše, záleží počet nutných stisků kláves na podobnosti slova s právě používaným jazykovým modelem.

Modifikací tohoto systému je také systém WordWise Shift, který využívá jednu klávesu na telefonu, které není přiřazeno žádné písmeno, jako klávesu Shift, a na každé klávese se skupinou písmen je vybráno jedno písmeno, které lze napsat pouze se stiskem klávesy Shift a dané klávesy. Toto má podle výrobce za cíl minimalizovat problémy s predikcí slov, která sdílejí stejnou kombinaci kláves. Společnost Eatoni na svých stránkách uvádí, že uživatel při používání tohoto systému nemusí vůbec sledovat display svého zařízení, a přesto bude moci jím napsaný text být zcela správně, protože chyba v predikcích slov nastává jednou za 440 slov.

citeWard2012

Paměťová náročnost jazykového modelu pro angličtinu je v aplikaci WordWise zhruba 75 kB.

6.2 QWERTY klávesnice na chytrých telefonech

S rozvojem chytrých telefonů s velkými dotykovými obrazovkami dostaly do širšího používání plné QWERTY klávesnice. Tyto klávesnice, které se zobrazují na displeji zařízení a nemají tudíž po hmatu rozlišitelné klávesy, se také nazývají *soft keyboards* nebo softwarové klávesnice.

Pokud by se účinnost takové klávesnice jako metody vstupu textu porovnávala pouze podle hodnoty KSPC, byly by plné QWERTY klávesnice na chytrých telefonech stejně účinné jako standardní klávesnice k počítačům. Realita je ovšem jiná, protože velikost klávesnic na chytrých telefonech je v řádu centimetrů namísto desítek centimetrů, a také proto, že na rozdíl od fyzické klávesnice nejsou jednotlivé klávesy na dotykovém displeji rozpoznatelné po hmatu. Zadávat textu na takové klávesnici proto vyžaduje ze strany uživatele větší zkušenosti a pozornost. Text navíc nemůže zadávat všemi deseti prsty, ale při nejlepším dvěma prsty, což rychlost psaní také rapidně snižuje. Nemožnost poznat hranice kláves po hmatu navíc vede k zásadnímu nárůstu chybovosti (překlepů), které je uživatel nucen opravovat. Ačkoliv je pravděpodobně nereálné snažit se bez empirického výzkumu zahrnout tyto faktory do výpočtu teoretického výpočtu KSPC pro tyto klávesnice, lze logickým úsudkem dospět k tomu, že i pro tuto metodu vstupu je žádoucí zvýšit rychlost a účinnost zadávání nějakým typem predikce textu.

6.2.1 Barevné zvýrazňování kláves

Jedním z navrhovaných způsobů, jak pomoci uživatelům snížit množství překlepů při psaní na softwarových QWERTY klávesnicích, bylo barevně odlišovat nejpravděpodobnější následující klávesy. Tento způsob byl navržen především pro starší uživatele, kteří jednak nemusejí být seznámeni s rozložením QWERTY klávesnice, a jednak nemají takovou přesnost motoriky a rychlost reakcí, jako uživatelé mladí.



Obrázek 3: QWERTY – barevné zvýrazňování

Klávesy byly odlišeny různými odstíny šedivé barvy, a to lineárně podle toho, jak pravděpodobné bylo dané písmeno v souvislosti s předchozím zadaným znakem. Vedle barevného odlišení pozadí klávesy byl ještě zvětšen popisek klávesy (obrázek 3 na straně 17).

nejaka cisla z toho clanku

Výsledky studie, která byla provedena na 15 ženách a 5 mužích, ovšem předpokládá, že zvýrazňování kláves starším uživatelům pomůže, popřely. Ukázalo se, že snaží-li se klávesnice vizuálně nějakým způsobem odlišit některé prvky a přitáhnout k nim pozornost uživatele, působí to na něj spíše rušivě a výrazně to snižuje rychlost zadávání textu.

6.2.2 Predikce celých slov

Druhým navrhovaným způsobem ve výše zmíněné studii byla metoda, kterou používá v současné době většina klávesnic na chytrých telefonech.

Jde o systém, kde po zadání znaku či znaků je ze slovníku známých slov vybráno několik slov, která jsou v souvislosti s již zadanými znaky nejpravděpodobnějšími řetězci, jež uživatel chce zadat (obrázek 4 na straně 17). Tento seznam je obvykle nabízen ve formě horizontálního seznamu nad klávesnicí a uživatel může ušetřit několik stisků kláves tak, že vybere požadované slovo ještě před tím, než jej celé dopíše (za předpokladu, že jej systém našel ve svém slovníku a správně navrhl).

citace? je to takovy obecně znamy fakt, prijde mi



Obrázek 4: QWERTY – seznam predikcí

Oproti zvýrazňování jednotlivých nejpravděpodobnějších kláves má tento systém

minimálně dvě výhody. Jednak nevyžaduje, aby si uživatel nestínil svými prsty, kterými píše, výhled na klávesnici, a jednak uživateli zobrazuje celá slova, ne pouze další znak z nich.

Nevýhodou této metody na druhou stranu je fakt, že je na uživatele kladena vyšší kognitivní zátěž v tom smyslu, že je nucen procházet zrakem seznam navrhovaných slov a přemýšlet, zda je mezi nimi i to, které zamýšlel napsat. Z tohoto důvodu je nutné délku seznamu návrhů omezit na optimum - některé práce uvádí, že ve své implementaci používají 6 návrhů.

6.2.3 Swype

Další systém zadávání textu, který je poněkud odlišný od všech dosud zmíněných, používají mj. aplikace Swype, Fleksy, oficiální klávesnice od Google, etc. Jde o systém původně vyvinutý firmou Swype Inc., nyní vlastněný a vyvíjený firmou Nuance Communications, určený výhradně pro dotyková zařízení. Uživatel v něm nezadává text ťukáním na jednotlivé klávesy, ale přejíždí prstem či stylusem po klávesnici od prvního písmena zadávaného slova k poslednímu tak, aby na cestě přejel pokud možno přes všechna písmena obsažená ve slově.

Swype se skládá ze tří základních částí – analyzátoru trasy dotyku, systému na vyhledávání slov v databázi výrazů a uživatelského prostředí. V raných verzích databáze obsahovala 65 000 výrazů.

damn.. tady mi zase utekla citace nekam do haje

Nutno podotknout, že firma Nuance, jež systém Swype vyvíjí, nezahrnuje do své klávesnice pouze tento systém zadávání, ale i klasickou metodu zadávání s nabídkou predikcí v horizontálním seznamu. Navzdory faktu, že hlavním důvodem popularity klávesnicové aplikace Swype je systém zadávání Swype, i klasická klávesnice v této aplikaci kombinuje různé technologie na zlepšení uživatelského komfortu. Mezi ně patří například schopnost rozšiřovat virtuální (nikoliv vizuální) velikost klávesy podle toho, jaké klávesy předcházely. Pokud uživatel tedy např. často tapne na klávesu N, smaže zadaný znak N a tapne na klávesu B a pokračuje v psaní, pak se to klávesnice naučí a příště, pokud je tapnutí ne zcela jednoznačné, vybere pravděpodobnější klávesu, v tomto příkladu tedy B.

Swype také používá Advance Language Model (AML), „pokročilý jazykový model“, který pracuje s bigramy a trigramy vytvořenými mimo jiné na základě textů, které už uživatel zadal. To aplikaci umožňuje účinněji disambiguovat slova, která vychází jako možná řešení uživatelova vstupu či předvídat další slovo, které by mohlo přijít po již napsaném. Oficiální dokumentace uvádí, že díky AML aplikace dokáže na základě kontextu správně rozlišit slova, která mají stejnou trasu tahu, tedy např. „put“, „pit“ a „pot“.

6.2.4 Predikce na fyzických QWERTY klávesnicích

Jak bylo naznačeno v kapitolách výše, většina implementací predikce textu, automatického doplňování textu a automatických oprav založených na predikci textu je určena pro zařízení buď s nejednoznačnou klávesnicí, kde má predikce eliminovat

nutnost opakovaných stisků stejné klávesy, nebo pro zařízení s dotykovou obrazovkou a softwarovou klávesnicí, na nichž si systémy kladou za cíl zvýšit rychlost a snížit chybovost zadávání textu.

Existují ale případy, kdy je predikce textu vhodná i pro použití s klávesnicemi fyzickými. Lze usuzovat, že se tyto případy nebudou týkat uživatelů, kteří nemají fyzický nebo mentální handicap a jsou na fyzickou QWERTY klávesnici zvyklí, ale spíše uživatelů, kteří z nějakého důvodu nemohou zadávat text normální rychlostí. Takoví uživatelé pravděpodobně trpí nějakým typem omezení motoriky rukou či nemocemi typu dyslexie. Pro ně může predikce textu mít smysl v tom, že nemusí zadávat celá slova, ale stačí zadat jeho menší část a poté vybrat požadované slovo z nabídky předvídaných slov. To pro ně patrně bude jednodušší, než muset zadávat celá slova, ať už z důvodu neschopnosti dostatečně rychlého psaní, či problémům se správným psaním. Taková predikce textu může napomoci např. lepší sociální interakci těchto uživatelů, protože rychlost komunikace může být značnou překážkou v účinné výměně informací s protistranou, která takovým handicapem netrpí, a nemá tudíž například trpělivost čekat na odpověď v chatu nadstandardní dobu.

Část II

Praktická část

Jak bylo naznačeno v úvodu, cíl této práce leží v úmyslu pokusit se implementovat tradiční, byť poněkud zjednodušený, model automatického doplňování textu pro uživatele s fyzickou QWERTY klávesnicí s tím, že na rozdíl od tradičních systémů popsaných v práci výše nebude mít tato aplikace lokální slovník nebo databázi n-gramů, ale veškeré operace produkující možná doplnění budou prováděny na vzdáleném serveru.

Základní výhodou tohoto modelu je fakt, že na vzdáleném serveru, na němž se provádí všechny operace spojené s analýzou vstupu a tvorbou predikcí, je možné skladovat výrazně vyšší množství jazykových dat, které slouží jako databáze známých slov.

Při umístění jazykových dat na vzdálený server je sice nutno počítat s neustálým připojením k internetové síti, na druhou stranu lze počítat s tím, že v implementaci určené pro klasické počítače, která je součástí této práce, by to neměl být omezující faktor.

7 Analýza

7.1 Backend

Zde popisovaná aplikace je závislá na serverovém skriptu, který tvoří takzvaný backend, tedy aplikaci, která se stará o produkci doplnění. Tento backend byl, společně s předchozími verzemi popisované aplikace, vyvinut již dříve a zcela nezávisle na této práci.

V této práci je tedy backendem skript naprogramovaný v dynamickém programovacím jazyce Python, který s webovou aplikací komunikuje pomocí protokolu CGI.

Jako zdrojová data pro skript produkující doplnění slouží předem vygenerované n-gramy o délce 3 až 12 tokenů. Tyto n-gramy pochází z nejrozsáhlejšího českého korpusu czTenTen, který je kolekcí textů z různých webových zdrojů. Texty v korpusu neprošly korekturami, takže nemusí nutně obsahovat pouze spisovnou češtinu, což může být závažným problémem, pokud by aplikace pro predikci textu měla být využívána pro kontrolu správnosti psaní. Vzhledem k tomu, že to ale není primární cíl a ani to přímo nesouvisí s cílem praktické části této práce, nebyl na tuto skutečnost během vzniku zde popisované aplikace brán zřetel. Nicméně v případě, že by byl zájem aplikaci užívat za takovým účelem, stačí vyměnit korpus, z něhož se generují predikce, za takový, který bude vyhovovat potřebám užití.

proc jich je tolik, když to stejně funguje jenom s bigrama?

Vygenerované n-gramy jsou vyfiltrovány podle různých kritérií s cílem vytvořit databázi n-gramů, která bude mít přijatelné pokrytí textů korpusu se zachováním rozumné datové velikosti databáze.

tady jeste popsat neco o tech FSA + prostor pro popis, jak se ty n-gramy vazi (ani o jednom nejak nic moc nevim.. too techy for me ww)

Vstupem backendového skriptu jsou posledních několik slov (záleží na konfiguraci aplikace) z textového editoru na frontendu. Serverový skript pracuje v základu ve dvou režimech, a to v závislosti na tom, zda vstup, který dostává, končí mezerou, či nikoliv. Pokud končí mezerou, předpokládá se, uživatel poslední slovo vstupu dokončil a chystá se psát další. Pokud mezera na konci vstupu není, je rozumné předpokládat, že uživatel slovo ještě nedokončil a chystá se v něm pokračovat. Podle tohoto jsou také rozdělené režimy, ve kterých serverový skript pracuje.

V prvním případě, tedy pokud na konci vstupu mezera je, jsou výstupem skriptu nejpravděpodobnější celá slova, která se v bigramech vyskytovala za slovem, které je posledním slovem vstupu. Pokud mezera na konci vstupu není, výstupem skriptu jsou slova, která se v bigramech vyskytovala na pozici za předposledním slovem vstupu a zároveň začínají řetězcem, který je shodný s posledním (nedokončeným) slovem na vstupu. Je důležité si povšimnout, že výstupem nejsou pouze doplnění posledního nedokončeného slova, ale celá slova obsahující i vstupní řetězec. Tímto se druhý režim odlišuje od prvního, který vrací doplnění, která lze jednoduše připojit za text bez nutnosti odstraňovat část v editoru již existujícího textu.

7.1.1 Forma komunikace

Co se formy vstupu a výstupu týče, má tento serverový skript velmi jednoduché rozhraní. Vstupní řetězec přijímá přes HTTP požadavek, v němž jsou data přenášena metodou GET, tedy jako parametry URL, například `http://nlp.fi.muni.cz/projekty/predictive/predict.py?input=sk%C3%A1kal+pes+`. Vstupem je v uvedeném příkladu tedy *skákal_pes_* (podtržítka značí mezery). Jelikož posledním znakem je mezera, očekávaným výstupem budou slova, která lze doplnit za *pes*.

Skript pak vrací aplikaci, z níž byl požadavek odeslán, text s maximálně 13 doplněními, která jsou oddělená mezerou.

Díky tomuto jednoduchému konceptu komunikace tedy stačí, aby aplikace, která serverového skriptu využívá, byla schopna odesílat HTTP požadavky na konkrétní URL adresu a parsovat prostý text.

7.2 Frontend

Frontend, neboli část aplikace, se kterou interaguje uživatel, byla v této práci pojata jako zásuvný modul do webového editoru CKEditor, což je vizuální HTML (HyperText Markup Language) editor, jehož cílem je zjednodušit produkci obsahu pro webové stránky. Editor umožňuje pracovat s formátováním ve značkovacím jazyce HTML, který se používá pro tvorbu webových stránek a webového obsahu. Práce s formátováním je založena na principu WYSIWYG, což znamená, že uživatel nemusí mít žádné znalosti HTML, protože pro aplikaci formátování lze použít vizuální prvky ve formě tlačítek a rozbalovacích menu, podobně jako v klasických

kancelářských textových procesorech typu LibreOffice Writer nebo Microsoft Office Word.

CKEditor je open source aplikace, což znamená, že jeho zdrojový kód je veřejně přístupný a každý jej může libovolně používat, modifikovat a distribuovat za dodržení jistých podmínek.

Výhodou CKEditoru je jeho modularita, která spočívá v možnosti rozšiřovat jej pomocí zásuvných modulů. Tvorba zásuvných modulů není vyhrazena pouze pro vybrané autory, ale díky otevřenosti kódu editoru se jí může zabývat takřka kdokoliv. Proto je CKEditor vhodným výběrem pro účely této práce.

8 Návrh

Jelikož uživatelské rozhraní zde popisované aplikace je určeno pro použití ve webovém prohlížeči, byly při vývoji využity výhradně webové technologie, tedy JavaScript v kombinaci s knihovnou AJAX (asynchronní JavaScript a XML (Extensible Markup Language)), značkovací jazyk HTML a CSS (kaskádové styly – Cascading Style Sheets).

8.1 JavaScript

JavaScript je dynamický, interpretovaný, objektově orientovaný programovací jazyk široce využívaný pro vývoj webových aplikací. Z hlediska komunikačního modelu klient-server se jedná o jazyk pro skriptování na straně klienta, takže se používá pro programování uživatelského rozhraní webových stránek.

Jeho vývoj, který probíhá od roku 1995 dodnes, začal ve společnosti Netscape Communications Corporation. V roce 1997 se JavaScript stal průmyslovým standardem poté, co společnost Ecma International vydala první standardizovanou verzi JavaScriptu pojmenovanou ECMAScript.

Ve zde popisovaném zásuvném modulu slouží JavaScript k obstarání hlavní logiky rozhraní, tedy extrakci textových dat editoru a jejich zpracování do formy vhodné k odeslání serverovému skriptu, sledování aktivity uživatele a případné zajištění, že data budou odeslána. Po přijetí odpovědi serveru zpracovává JavaScript přijatý text, zajišťuje uživateli nabídku doplnění a jejich případně vložení na správné místo v editoru.

8.2 AJAX

AJAX, neboli Asynchronous JavaScript and XML, je souhrnné označení webových technologií, které se používají pro komunikaci webových aplikací se serverem. Hlavním využitím AJAXu bývají právě aplikace, které potřebují komunikovat se vzdáleným serverem bez nutnosti načítat celou webovou stránku znova.

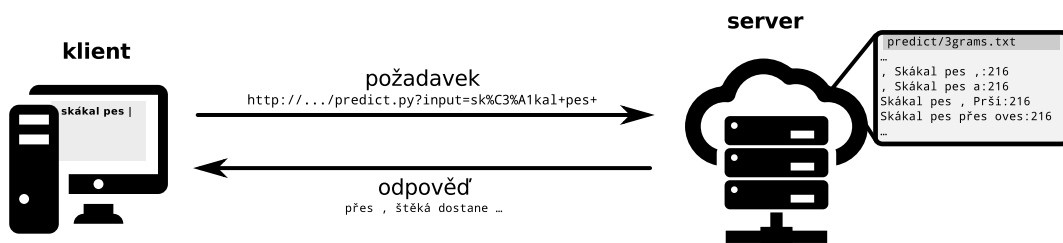
Asynchronní výměna dat se serverem probíhá pomocí aplikačního rozhraní XMLHttpRequest (XHR), které skriptovacím jazykům na straně klienta, jako je například

JavaScript, umožňuje odesílat data na vzdálený server. Po přijetí odpovědi se pak stará od dodání dat zpět do skriptu.

Pro komunikaci se serverem se využívá vzhledem k velmi malým objemům dat, které je nutno přenášet, prostého textu, a to ve dvou formách.

První z forem se používá při odesílání požadavku, kde jsou data serveru dodávána v rámci URL jako parametry. Jedná se vždy o posledních několik slov (záleží na konfiguraci zásuvného modulu), která jsou zakódována globální funkcí JavaScriptu `encodeURIComponent()`, aby byla zajištěna správná funkčnost i v případě výskytu zvláštních znaků, jako jsou například písmena s diakritikou nebo jakékoliv jiné znaky, které v URL adrese být přímo nemohou. Jednotlivá slova jsou oddělována pomocí znaku `+`, který v URL adrese může nahrazovat mezeru.

Druhá forma prostého textu se využívá v odpovědích serveru. Skript na požadavky odpovídá textovým řetězcem, který obsahuje navrhovaná slova oddělená mezerou. Tento textový řetězec je odesílán klientu jako dokument obsahující prostý text, který neobsahuje žádné přídatné strukturní nebo designové prvky, takže je opět velmi snadno zpracovatelný.



Obrázek 5: Schéma komunikace klienta a serveru¹⁰

8.3 jQuery

jQuery je knihovna navržená pro usnadnění programování v jazyce JavaScript. Zaměřuje se především na zjednodušení zpracování prvků v HTML dokumentu, práce s událostmi (např. stisk klávesy) a komunikace mezi klientem a serverem pomocí AJAXu.

8.4 HTML

HTML, neboli HyperText Markup Language, je standardní značkovací jazyk velice podobný jazyku XML široce používaný pro tvorbu webových stránek pro webové prohlížeče, jako je kupříkladu Mozilla Firefox¹¹ či ¹²Google Chrome. Webové prohlížeče jsou schopny HTML zpracovat a zobrazit uživateli výslednou stránku, která obsahuje vizuální prvky a textové formátování.

¹⁰Schéma obsahuje ikony od Freepik vytvořené pod licencí Creative Commons BY 3.0

¹¹<https://www.mozilla.org/en-US/firefox/desktop/>

¹²<https://www.google.com/chrome/>

Standard HTML vychází ze obecného značkovacího jazyka SGML (Standard Generalized Markup Language), který je založen na dvou principech, a to že by značkovací jazyk měl být deklarativní, tedy popisovat strukturu a další atributy, spíše než definovat způsob, jakým je dokument zpracováván, a že by definice v dokumentu měly být jednoznačné a pevně definované, aby byl dokument zpracovatelný stejně jako například programový kód.

HTML popisuje strukturu stránky po sémantické stránce, stejně jako definuje (částečně či úplně) její vzhled. Rozdíl mezi sémantickou informací a informací pouze o tom, jak by měla prezentace vypadat, lze ilustrovat na dvojici značek, jejichž výsledná prezentace je zpravidla identická, a to `` a ``. Značka `` webovému prohlížeči říká, že text uvnitř této značky je důležitější než okolní text a měl by podle toho také být naformátován (tedy zpravidla tučným řezem písma), zatímco značka `` pouze indikuje, že by text v ní uzavřený měl být naformátován tučně.

Zde popisovaná aplikace využívá HTML relativně málo, protože se jedná o zásuvný modul, který ze své podstaty nesmí ovlivňovat vzhled stránky, do níž si uživatel vloží CKEditor s tímto modulem. V rámci editačního pole se však vyskytla potřeba HTML použít pro zobrazení nabídky s výběrem jednotlivých možných doplnění, z nichž si uživatel může vybrat. Jakkoliv tedy co do množství kódu je HTML zastupováno nejmenší částí ze zde popisovaných technologií, je jeho použití důležité pro komfort koncového uživatele.

8.5 CSS

Druhou technologií, která je relativně málo využita, ale pro koncového uživatele je zásadní z hlediska uživatelské přívětivosti rozhraní, jsou kaskádové styly (CSS). CSS je také jednou ze základních technologií používaných pro webové prezentace a zajišťuje definice vizuálního vzhledu stránek napsaných ve značkovacím jazyce.

Hlavním smyslem CSS je oddělení obsahu webové stránky od její vizuální prezentace. To je důležité především pro snazší udržování stránky a větší flexibilitě. Použití CSS navíc umožňuje zobrazit jednu stránku různými způsoby, jedním z nichž může být například možnost zobrazení pro barvoslepeho uživatele.

Ve zde popisované aplikaci je tedy CSS využíváno k definici vizuální prezentace nabídky alternativních doplnění.

9 Implementace

9.1 Uživatelské rozhraní

Mluvit o uživatelském rozhraní jako celku je v tomto případě poněkud problematické, protože jde o zásuvný modul a není tudíž možné předvídat, jak přesně bude koncový uživatel modulu mít CKEditor nakonfigurován. Pro účely této práce však postačí uvažovat základní konfiguraci CKEditoru tak, jak je poskytována na ofi-

ciálních webových stránkách¹³. Tam jsou k dispozici ke stažení předpřipravené tři balíčky, jež se liší množstvím zásuvných modulů, které jsou obsahují. Pro účely vývoje zde popisovaného pluginu i pro účely samotného popisu jsme vybrali nejmenší balíček *Basic Package*, který obsahuje 17 zásuvných modulů obstarávající základní funkcionalitu. V popisu práce s editorem pak bude uvažována webová stránka, na níž se bude nacházet pouze CKEditor v konfiguraci *Basic Package* s přidaným pluginem, který je předmětem této práce.

Práce v takto nakonfigurovaném editoru probíhá tak, že uživatel zadává text do editoru pomocí klasické klávesnice a vybírá si (případně nevybírá) doplnění, která jsou mu v průběhu nabízena. Doplnění jsou nabízena vždy, když na určitou dobu uživatel přestane vyvíjet aktivitu (pro účely testování se jako vhodné ukázalo být 300 ms), a když jsou pro danou kombinaci slov a písmen před pozicí kurzoru nalezena doplnění. Pokud nalezena nejsou, nabídka se nezobrazuje. Ve chvíli, kdy si uživatel některé doplnění z nabídky vybere, skript jej v závislosti na situaci vhodným způsobem doplní do editoru, přidá za nově doplněné slovo mezeru a posune na její místo kurzor. Pokud uživatel poté nevyvine žádnou další aktivitu po 300 ms, zobrazí se mu další doplnění zvolené na základě předchozího slova zadaného (či vybraného) slova. V ideálním případě by tak mělo na základě prvního slova být možné doplnit určité standardní věty pouhým vybíráním vhodných predikcí. Tomu ovšem zabráňuje nastavení aplikace, které filtruje z nabídky predikcí příliš krátká slova (méně než čtyři znaky), protože ta většinou bývá rychlejší napsat celá manuálně.

9.2 Zpracování textu v editoru

Aby mohla být serverovému skriptu odeslána data, na jejichž základě budou vyhledána možná doplnění, musí být na straně klienta zpracován text, který uživatel napsal do editoru. To probíhá v několika krocích, jejichž výsledkem je řetězec předem definovaného počtu posledních slov v editoru, který je vhodný k odeslání na server.

Všechny operace spojené s modifikacemi textu v editoru i prezentací nabídky predikcí prováděny pouze v případě aktivity uživatele. Touto aktivitou se myslí javascriptové události `keydown` a `keyup`, které reprezentují stisky kláves – konkrétně stisknutí klávesy (`keydown`) a její uvolnění (`keyup`).

Prvním krokem je rozdělení obsahu editoru na jednotlivá slova. To zajišťuje funkce `sliceContent()`, jež přijímá text editoru ve formě řetězce (datový typ `String`). Text rozděluje postupně na řádky podle tagu `
`, kterým končí v CKEditoru každý řádek, a dále zpracovává pouze poslední řádek. To sice znemožňuje používat predikci na jiném než posledním řádku, což ale nevádí, protože momentálně celkově plugin nepodporuje práci s predikcemi na jiných místech než na úplném konci vkládaného textu. To je jeden ze známých nedostatků, který bude rozebrán ve vyhodnocení. Následně funkce HTML tagy odstraní, nahradí nedělitelné mezery normálními a rozdělí zpracováváný řádek na jednotlivá slova. Jako datový typ pro uložení řetězce rozděleného na slova bylo zvoleno pole (`Array`), jež má v jazyce JavaScript připraveno

¹³<http://ckeditor.com/>

velké množství metod vhodných pro tuto aplikaci. Další operace prováděné funkcí `sliceContent()` na výsledném poli jsou čistě technického rázu.

Poté, co má skript k dispozici slovo z aktuálního (posledního) řádku v editoru, se v případě, že současné slovo u kurzoru má délku alespoň tři znaky, zjišťuje pomocí funkce `findLastSimiliarWord()`, zda se v editoru už nevyskytuje slovo, které začíná na řetězec shodný se současným slovem a má délku alespoň pět znaků. Pokud ano, přidá se do seznamu predikcí, který později bude uživateli nabídnut. Tento krok se provádí proto, že lze předpokládat, že pokud uživatel napsal jednou nějaké delší slovo a znova začal psát slovo, které začíná stejnými znaky, bude chtít napsat téže slovo. Jde o funkcionálnost podobnou té z textového procesoru LibreOffice Writer (obrázek 1 na straně 6).

Po tomto kroku už přichází na řadu funkce `getSuggestions()`, která provádí AJAXové volání na server za účelem získání predikcí vytvořených z korpusových dat. Funkce `getSuggestions()` přijímá několik hodnot, a to pole se slovy z posledního řádku, počet slov, která mají být odeslána na server a dále callback na funkci `handleSuggestion()`, která zpracovává získaná doplnění.

Ve funkci `getSuggestions()` pak proběhne připravení řetězce se slovy, která mají být odeslána na server. To znamená, že funkce metodou `slice()` vyřízne z pole posledních N slov, kde N je počet slov odesílaných na server a výsledné pole N prvků spojí do řetězce s tím, že mezi jednotlivé prvky (slova) vloží znak + (plus). Znak + byl zvolen proto, že slova jsou odesílána, jak již bylo popsáno v kapitole 8, metodou GET, tedy jako parametry adresy URL, a v rámci dat odesílaných metodou GET lze reprezentovat mezeru buď znakem + nebo řetězcem `%20`, přičemž tyto způsoby jsou v praxi rovnocenné.

Nyní funkce čeká předem nastavenou dobu, zda nenastane další aktivita uživatele, a v případě, že nenastane, odešle AJAXový požadavek na server.

V případě, že AJAXové volání na server uspělo (návrátový kód 202 `Success`), zavolá se pomocí callbacku, který byl funkci `getSuggestions()` předán dříve, funkce `handleSuggestion()`. Pokud volání z jakéhokoliv důvodu skončilo chybou (příkladem budiž nedostupný server, což způsobí, že volání skončí návratovým kódem 404 `Not Found`), vypíše se chyba do konsoly a z hlediska uživatele se nic nestane.

Funkce `handleSuggestion()` přijímá jako argumenty mj. současné slovo pod kurzorem, jeho případné doplnění z existujícího textu v editoru a odpověď serveru. Odpověď serveru je řetězec znaků, který obsahuje slova oddělená od sebe znakem mezery. Funkce odpověď rozdělí na jednotlivá slova a vyřadí slova kratší než 3 znaky včetně (z důvodů uvedených v kapitole 9.1). Pokud byla dříve nalezena predikce z textu, je doplněním ze serveru předřazena, aby uživateli byla prezentována jako první.

Poté jsou predikce nabídnuty uživateli.

9.3 Zobrazení predikcí

Nabídka predikcí sestává ze dvou různých částí. První z nich je predikce (jíž může být celé slovo po mezeře, či část slova doplňující část slova již zadanou v editoru), která

jak se správně říká to, že to jsou prostě nepodstatné blbosti, kterými spravuju různé divné chování CKEditoru? xD

se zobrazuje méně výrazně (šedě) ihned za zadávaným textem v hlavním editoru (obrázek 6 na straně 27). Takové doplnění lze přijmout pouhým stiskem tabulátoru. Tento způsob zobrazení doplnění je inspirován řešením zobrazování automatického doplňování ve vyhledávači Google, které funguje na podobném principu.

Pro tento způsob zobrazení predikce je využíváno tzv. stínového editoru, který je pomocí CSS umístěn přesně pod hlavní editor, v němž uživatel zadává text. Do něj se pomocí JavaScriptové funkce `copyContent()` ve zde popisovaném zásuvném modulu kopíruje text z hlavního editoru. Pro zobrazení predikce je pak využíváno toho, že text zkopírovaný z hlavního editoru má ve stínovém editoru identické prostorové vlastnosti jako text hlavního editoru a lze na jeho konec snadno připojit aktuální predikci, která se tak zobrazuje uživateli fontem stínového editoru (ve zde popisované implementaci má stínový editor šedou barvu písma a hlavní editor černou).



Obrázek 6: Zobrazení predikcí

9.3.1 Alternativní predikce

Vzhledem k tomu, že predikce nejsou vždy dokonalé nebo není možné jednoznačně rozhodnout, které slovo uživatel zamýšlí napsat, ukázalo se být nutným zobrazovat i alternativní predikce.

Tyto alternativní predikce jsou prezentovány pod pozicí kursoru ve formě seznamu. Pokud chce uživatel vyžít některou predikci zobrazenou v seznamu, stačí mu opětovně stisknout klávesu tabulátor, pomocí níž lze v seznamu jednosměrně cyklovat mezi položkami. Aktuální položka se pak zároveň kopíruje do hlavního editoru, takže po dosažení požadované predikce uživatel stiskne klávesu mezerník a může pokračovat v psaní. Tato metoda volby byla navržena především s ohledem na původní cíl aplikace, tedy usnadnit psaní uživatelům, kteří z nějakého důvodu problém zadávat text na QWERTY klávesnici. Cílem tedy není zrychlení psaní, jako tomu bývá u implementací prediktivního textu například pro dotyková zařízení, ale co nejvíce snížit množství stisků různých kláves pro volby predikce.

Nabídka alternativních predikcí je zobrazována vždy pod kursorem, aby uživatel nebyl nucen dělit svou poroznost na různá místa obrazovky. O zobrazení nabídky v místě kursoru se stará funkce `showAlternatives()`, která je závislá na funkci `getSelectionCoords()`. Z pohledu problematiky implementace je zajímavá především funkce `getSelectionCoords()`, která pomocí zjišťování pozice aktuálního výběru textu získává pixelovou pozici kursoru, jež je nutná pro správné zobrazení alternativ. Implementace výběrů je velmi závislá na konkrétním prohlížeči, avšak lze obecně říci, že aktuální výběr vždy končí v místě, kde je právě umístěn kurzor. S ohledem na tuto skutečnost a na to, že API (application program interface) moderních prohlížečů umožňuje získat pixelové pozice začátku a konce výběru, lze pak

pixelovou pozici konce aktuálního výběru považovat za pixelovou pozici kursoru, na níž se má zobrazit nabídka alternativních doplnění.

bezprisorní odstavec o problematice vyberu

Výběry mají v aplikaci ještě další způsob využití, a to přesunutí kursoru na korektní místo poté, co je vložena zvolená predikce. Je zřejmé, že kursor nemůže zůstat na původním místě, na němž byl před vložením predikce, protože pak by uživatel buď psal další text před vloženou predikci, nebo by musel manuálně přejít na konec vkládaného textu. Tento problém je sice snadno řešitelný přemístěním kursoru o délku predikce dále, ale v praxi se ukazuje, že existuje nezanedbatelné množství případů, kdy výběr není korektně umístěn v textu, a tudíž ani pozice kursoru na jeho umístění závislá není správná. Je to způsobeno především odlišnou a nedokonalou implementací v různých prohlížečích, protože nejde o standardizovanou součást API prohlížečů. Zajistit tak například korektní umístění kursoru při vkládání predikce nikoliv na konec, ale doprostřed textu, se ukázalo být velice problematické, a proto od toho v rámci této práce bylo upuštěno.

nejsem si uplne jist, jak moc zabíhat do podrobnosti o selections a tehle vecich, protože jejich vysvetlení pro filozofy by mohlo byt docela obtizne.. a vlastne pro vsechny, protože je to dost silena vec

9.4 Vkládání predikcí

Poté, co si uživatel vybere predikci tím, že stiskne klávesu tabulátor, zbývá ji vložit na správné místo, tedy na pozici kursoru.

V případě vkládání celého slova, nikoliv doplnění zbytku již zadaného slova, je vložení predikce relativně triviální. Jde pouze o připojení dané predikce za pozici kursoru, přidání mezery a přesun kursoru za ni. Jak bylo naznačeno výše, pokud se uživatel rozhodne cyklovat v seznamu alternativních predikcí, vkládají se tyto do hlavního editoru automaticky pomocí funkce `printCompletions()`. Chce-li využít první predikce, použije se při stisku klávesy tabulátor stejná funkce, `printCompletions()`.

Ovšem v případě doplňování pouze části slova jde vzhledem k tomu, že serverový skript ve všech případech vrací celá slova, o úkon, který obsahuje zásah do již existujícího textu v hlavním editoru. Pokud tedy uživatel napíše jako poslední slovo řetězec „p“ a předcházející slovo je „vážený“, navracená slova jsou „paciente“, „pane“, etc. Je tedy zřejmé, že nelze zvolenou predikci (např. „paciente“) vložit přímo za existující text, ale je nutné nejdříve začátek slova, pro který je predikce určena, smazat, a pak až predikci vložit. Tento problém je řešen ve funkci `insertCompletion()` smazáním již napsané části slova z predikce, jejíž zbytek je pak vložen za napsaný text. Do hlavního editoru se tedy nevkládá celá predikce „paciente“, nýbrž pouze její zbývající část po odtržení již napsané části, tedy „aciente“.

10 Závěr

Při testování hotového zásuvného modulu se projevilo několik problémů, jejichž obrysy byly naznačeny v předcházejících kapitolách.

Cíl práce, tedy vyvinout aplikaci, která by umožňovala zadávání textu bez nutnosti stisku všech kláves korespondujících s písmeny ve slovech zadávaného textu,

byl splněn. Pokud uživatel chce zadat text, jehož slova existují v korpusu, z něhož jsou vybírány predikce, má možnost z těchto predikcí vybrat vhodně ty, které odpovídají jeho záměru, a snížit tak KSPC na hodnotu menší než jedna. Vzhledem k velikosti korpusu, z něhož jsou predikce vybírány, je rozumné předkládat, že takového výsledku lze dosáhnout téměř s libovolným textem v českém jazyce. Současná implementace navíc umožňuje snížit počet stisknutých kláves na počet znaků v textu také tím, že mezi predikce zařazuje již zadaná delší slova. Tato funkcionalita může být užitečná například pro uživatele, kteří zadávají text s větším množstvím dlouhých odborných termínů (dobrým příkladem budiž lékařské texty).

Co se problémů týče, hlavním z nich je fakt, že momentální implementace neumožňuje práci vprostřed těla textu, ale pouze na jeho konci. Není tak možné mít text, do jehož části chce uživatel vložit odstavec, ve kterém by využíval predikce textu. Zde je tedy značný prostor pro zlepšení, a to hlavně v případě, že selepší implementace výběrů textu v prohlížečích.

problemy s formatováním?

Závěr



Seznam literatury

- ALDERSON, J Charles, 2007. Judging the frequency of English words. *Applied Linguistics*, roč. 28, č. 3, s. 383–409.
- ARIF, Ahmed Sabbir; STUERZLINGER, Wolfgang, 2009. Analysis of text entry performance metrics. In: *Science and Technology for Humanity (TIC-STH), 2009 IEEE Toronto International Conference*. s. 100–105.

- ČERMÁK, František; BLATNÁ, Renata, 2004. *Frekvenční slovník češtiny*. Praha : Lidové Noviny.
- DAVIES, Mark; GARDNER, Dee, 2013. *A frequency dictionary of contemporary American English: Word sketches, collocates and thematic lists*.
- EATONIERGONOMICS, 2012. *FAQ:For Manufacturers*. Dostupný z WWW: https://web.archive.org/web/20120212172428/http://www.eatoni.com/wiki/index.php/FAQ:For_Manufacturers.
- FEINSON, Roy W, 1988. *Interpretive tone telecommunication method and apparatus*. US Patent 4,754,474.
- GHAYOOMI, Masood; MOMTAZI, Saeedeh, 2009. An overview on the existing language models for prediction systems as writing assistant tools. In. *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on* [online]. [cit. 2015-10-24], s. 5233–5237. Dostupný z WWW: [http://hpsg.fu-berlin.de/~ghayoomi/Publishedpapers/ghayoomi-2009\[3\].pdf](http://hpsg.fu-berlin.de/~ghayoomi/Publishedpapers/ghayoomi-2009[3].pdf).
- GOOGLE, 2015. *Automatické doplňování* [online]. [cit. 2015-10-23]. Dostupný z WWW: <https://support.google.com/websearch/answer/106230?hl=cs%5C&vid=1-635809595515389355-3739668669>.
- GROVER, D.L. et al., 1998. *Reduced keyboard disambiguating computer* [online]. [cit. 2015-10-24]. US Patent 5,818,437. Dostupný z WWW: <https://www.google.com/patents/US5818437>.
- HENRY, Alan, 2014. How Predictive Keyboards Work (and How You Can Train Yours Better). *Lifehacker* [online], [cit. 2015-10-23]. Dostupný z WWW: <http://lifehacker.com/how-predictive-keyboards-work-and-how-you-can-train-yo-1643795640>.
- KOCIENDA, K. et al., 2012. *Method, device, and graphical user interface providing word recommendations for text input* [online]. [cit. 2015-10-23]. US Patent 8,232,973. Dostupný z WWW: <https://www.google.com/patents/US8232973>.
- LEVENSHTEIN, V. I., 1966. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady* [online], roč. 10, [cit. 2015-10-23], s. 707. [Překlad Левенштейн, 1965]. Dostupný z WWW: <http://adsabs.harvard.edu/abs/1966SPhD...10..707L>.
- LORGE, Irving, 1944. The teacher's word book of 30,000 words. *New York: Teachers College, Columbia University*.
- MACKAY, David. *Some notes on iTap / T9 / "predictive text"*. Dostupný z WWW: <http://www.inference.phy.cam.ac.uk/itprnn/itap/>.
- MACKENZIE, I Scott, 2002. KSPC (keystrokes per character) as a characteristic of text entry techniques. In. *Human Computer Interaction with Mobile Devices*. s. 195–210.

- MACKENZIE, I Scott et al., 2001. LetterWise: prefix-based disambiguation for mobile text input. In. *Proceedings of the 14th annual ACM symposium on User interface software and technology* [online]. [cit. 2015-10-22], s. 111–120. Dostupný z WWW: <http://www.yorku.ca/mack/uist01.pdf>.
- NATION, Paul; WARING, Robert, 1997. Vocabulary size, text coverage and word lists. *Vocabulary: Description, acquisition and pedagogy*, roč. 14, s. 6–19.
- NORVIG, Peter. *How to Write a Spelling Corrector* [online]. [cit. 2015-10-23]. Dostupný z WWW: <http://norvig.com/spell-correct.html>.
- NUANCE COMMUNICATIONS, 2007. *Nuance Closes Acquisition of Tegic Communications* [online]. [cit. 2015-10-24]. Dostupný z WWW: http://www.nuance.com/news/pressreleases/2007/20070824_tegic.asp.
- NUANCE COMMUNICATIONS, 2015. *T9® – The Global Standard for Mobile Text Input* [online]. [cit. 2015-10-24]. Dostupný z WWW: <http://www.nuance.com/for-business/by-product/t9/index.htm>.
- SILFVERBERG, Miika et al., 2000. Predicting text entry speed on mobile phones. In. *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. s. 9–16.
- SKETCH ENGINE, 2015a. *Corpus czTenTen12 – statistics and info* [online]. [cit. 2015-10-22]. Dostupný z WWW: https://ske.fi.muni.cz/bonito/run.cgi/corp_info?corpname=preloaded/cztenten12_8.
- SKETCH ENGINE, 2015b. *Corpus enTenTen [2013] – statistics and info* [online]. [cit. 2015-10-22]. Dostupný z WWW: https://ske.fi.muni.cz/bonito/run.cgi/corp_info?corpname=preloaded/ententen13.
- SOUKOREFF, R William; MACKENZIE, I Scott, 2001. Measuring errors in text entry tasks: an application of the Levenshtein string distance statistic. In. *CHI'01 extended abstracts on Human factors in computing systems*. s. 319–320.
- TARVAINEN, Jussi, 2010. Beginner Performance with the GKOS Chorded Keyboard. [online], [cit. 2015-10-24], s. 2–5. Dostupný z WWW: <http://urn.fi/urn:nbn:fi:uta-1-20834>.
- THE BOLTON NEWS, 2011. *Indefinite sentence for killing his friend* [online]. [cit. 2015-10-24]. Dostupný z WWW: http://www.theboltonnews.co.uk/news/8950811.Indefinite_sentence_for_killing_his_friend/.
- WARD, David et al., 2012. Autocomplete as research tool: A study on providing search suggestions. *Information Technology and Libraries*, roč. 31, č. 4, s. 6–19.
- WOBBROCK, Jacob O., 2007. Measures of text entry performance. In MACKENZIE, I.S.; TANAKA-ISHII, K. (ed.). *Text Entry Systems: Mobility, Accessibility, Universality*. San Francisco, CA : Morgan Kaufmann, s. 47–74.
- YAMADA, Hisao, 1980. A historical study of typewriters and typing methods: from the position of planning Japanese parallels. *Journal of information processing*, roč. 2, č. 4, s. 175–202.

ZORN, Eric, 2007. *Slang early-warning alert: ‘Book’ is the new ‘cat’s pajamas’* [online]. [cit. 2015-10-24]. Dostupný z WWW: http://blogs.chicagotribune.com/news_columnists_ezorn/2007/01/slang_earlywarn.html.

ЛЕВЕНШТЕЙН, Владимир Иосифович, 1965. Двоичные коды с исправлением выпадений, вставок и замещений символов. *Доклады Академий Наук СССР*, роč. 163, č. 4, s. 845–848.