2D1465 – Avancerad individuell kurs i datalogi

Stavningskontroll i sökmotor

Rasmus Kjellman, kjellman@kth.se

Abstract

This is a report written within the course “Advanced, Individual Course in Computer Science” (2D1465) at the institution NADA at the Royal Institute of Technology in Stockholm, Sweden. The report explains how a spellchecker for a search engine could be implemented and what problems and conclusions that have been identified during the work of building a prototype spellchecker for Picsearch AB (www.picsearch.com). The prototype corrects spelling errors at both distance one and distance two and assumes a common word is more correctly spelled than a less common word. The words are checked in three indexes built from searches at Picsearch’s website. One index contains every word and how common they are. The second index contains all word pairs, making it easier to correct especially names. The last index contains all search phrases. The results are good but further optimization and evaluations are needed to make use of this product in a professional way.

Sammanfattning

Detta är en rapport skriven inom kursen ”Avancerad individuell kurs i datalogi” (2D1465) vid NADA på Kungliga Tekniska Högskolan i Stockholm, Sverige. Rapporten förklarar hur ett rättstavningsverktyg för en sökmotor kan implementeras och vilka problem och slutsatser som har identifierats under utvecklingen av en rättstavningsprototyp för Picsearch AB (www.picsearch.com). Prototypen korrigerar fel både på avstånd ett och avstånd två och utgår ifrån att ett vanligt ord är mer rättstavat än ett mindre vanligt ord. Orden kontrolleras mot tre index uppbyggda från sökningar på Picsearchs hemsida. Ett index innehåller alla ord och hur vanliga de är. Det andra indexet innehåller alla ordpar, vilket underlättar rättstavning av framförallt namn. Det sista indexet innehåller alla sökfraser. Resultatet är bra men vidare optimering och utvärdering krävs för att kunna använda produkten i en professionell verksamhet.

Innehållsförteckning

Inledning

Denna rapport är skriven inom ramarna för kursen ”Avancerad individuell kurs i datalogi” (2D1465), en fyrapoängskurs som går vid institutionen för numerisk analys och datalogi på Kungliga Tekniska Högskolan i Stockholm. Rapporten riktar sig framförallt till personer som har en hel del kunskap inom datalogi.

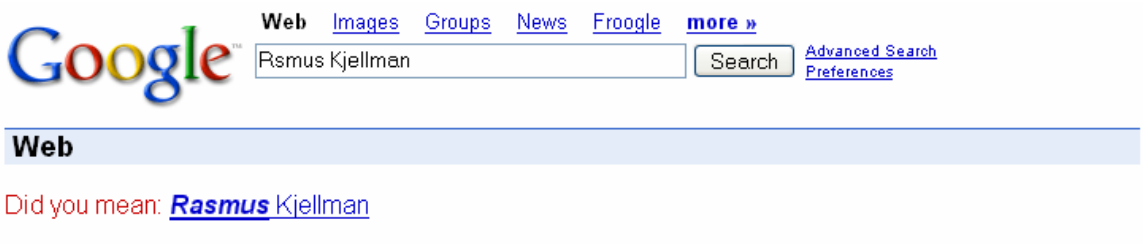
Under kursen har en prototyp till ett rättstavningsverktyg till en sökmotor tagits fram. Denna rapport beskriver erfarenheter, problem och slutsatser som har identifierats under utvecklingsarbetet av detta verktyg. Tillämpningen tillhör ämnet som brukar benämnas ”information retrieval” (IR).

Rapporten är indelad i fem större delar. En bakgrund inleder rapporten. I bakgrunden beskrivs kort söktmotorer i allmänhet, rättstavning i sökmotorer i synnerhet samt Picsearch AB och deras önskemål. I avsnittet som avhandlar utvecklingsarbetet beskrivs på ett personligt och mer populistiskt vis utvecklingsarbetet av ett rättstavningsverktyg för sökmotorer och vilka problem som identifierades under arbetet. Därefter beskrivs implementationen tekniskt. Till sist presenteras resultat och slutsatser som har dragits under arbetets gång.

Bakgrund

Stavningskontroll i sökmotorer

Sökmotorer likt Google (www.google.com), Altavista (www.altavista.com) och MSN search (beta-version beta.search.msn.com) har med Internets snabba tillväxt blivit ett extremt vanligt och nödvändigt verktyg för Internets alla användare. Sökmotorerna har till och med introducerat ord i det svenska språket – idag förstår de flesta vad verbet googla betyder. När sökmotorernas användare blev vana vid motorernas funktion började en del användare på ett klumpigt men funktionellt sätt använda bland annat Google för rättstavning. Genom att söka på ett ord eller fras och därefter söka på samma ord eller fras med en alternativ stavning kunde användaren bedöma vilken stavning som var rätt genom att observera hur många träffar sökningen på de olika stavningarna gav. Till exempel ger en sökning på Google (www.google.com den 13 november 2004) efter ordet ”fobar” (cirka) 914 träffar, en sökning på ”foobar” ger däremot (cirka) 1 230 000 träffar. Det är därför rimligt att anta att ”foobar” är ”rätt” stavning av ordet. Google, med enorma resurser, har ofta gått i bräschen för utvecklingen av funktioner i sökmotorer och uppmärksammade hur användarna valde att använde sökmotorn. Därför presenterade Google i maj 2001 [1] en stavningskontroll i sökmotorn, se figur 1.



Figur 1

Efter Googles introduktion av stavningskontroll infördes senare samma funktion på bland annat på Alltheweb (www.alltheweb.com), Altavista och Gigablast (www.gigablast.com) [2]. Hur de olika sökmotorerna har valt att implementera stavningskontrollen är dock hemligt. Google skriver mycket kortfattat att de använder sig av ”algorithms and techniques to construct very large scale Bayesian network models” [3]. Övriga sökmotorer presenterar över huvud taget inte hur de valt att implementera verktyget.

Det finns i botten två (eller tre) olika sätt att implementera en rättstavningsfunktion till en sökmotor. Antingen utgår man från sökningar som andra användare gjort och rättstavar ord till ord som är vanlig i andra användares sökningar. Denna metod tycks Google använda sig av [4] (denna uppgift kommer dock ej från Google). En annan metod är att använda sökmotorns cachade hemsidor som underlag. Gigablast använder denna teknik och prioriterar ett vanligt ord över ett mindre vanligt ord [5]. En sista metod är naturligtvis att använda både andra användares sökningar och cachade sidor som underlag.

Skillnader mellan olika stavningskontroller

Stavningskontroll har funnits i ordbehandlare mycket länge. Redan i mycket tidiga versioner av Microsoft Word (www.microsoft.com) har en rättstavningsfunktion som rättat stavfel och enklare grammatiska misstag funnits. Liknande stavningskontroll finns i program som Ispell (www.gnu.org/software/ispell/ispell.html) och Aspell (aspell.sourceforge.net).

Även om rättstavningsfunktionaliteten i ordbehandlare är lik funktionen i en sökmotor finns mycket stora skillnader. En rättstavningsfunktion i en ordbehandlare utgår ifrån en ordlista som likt Svenska Akademins Ordlista innehåller alla rättstavade ord och korrekta böjningar av dessa ord. Ord som inte finns i ordlistan kommer att anses vara felstavade. I stavningskontrollen till en sökmotor har man ett helt annat underlag. Eftersom man utgår ifrån cachade hemsidor eller andra användares sökningar kommer ordlistan till en sökmotor innehålla felstavade ord såväl som rättstavade. Det som är intressant i sökmotorns ordlista är inte om ordet existerar i ordlistan utan hur många gånger det existerar. Ett vanligt ord är ”mer” rättstavat än ett ovanligt. Dessutom är inte ett korrekt stavat ord i sökmotorn ett egensyfte. Huruvida ordet är stavat rätt eller fel är ointressant så länge användaren presenteras med sökresultat användaren efterfrågade.

Problemen i de två olika rättstavningsfunktionerna är även de helt annorlunda. I ordbehandlarna önskas en ”lagom” stor ordlista [6]. En ordlista med för få ord kommer, naturligtvis, sakna många rättstavade ord. En ordlista med för många ord å andra sidan kommer innehålla ovanliga ord som det finns en risk att felstavade ord sammanfaller med. Ska till exempel ett ovanligt ord som ”got” (person tillhörande forntida germanfolk) finnas i ordlistan, eller ska got rättstavas till ”gott” (eller något annat)? I stavningskontrollen i en sökmotor önskas däremot en så stor ordlista som möjligt. Eftersom ordlistan även innehåller hur vanligt ordet är ger en stor ordlista ett bättre statistiskt underlag. I och med detta kan även ord från flera språk finnas i samma ordlista och flera språk därmed rättstavas. Problemet i sökmotorn blir därför att utveckla ett effektivt sätt att spara och söka i en mycket stor ordlista. Utöver detta blir problemet i en sökmotor att dra gränsen för hur mycket vanligare ett ord måste vara för att anses vara rättstavat.

Picsearch AB

Picsearch är ett svenskt företag som utvecklar en sökmotor för bilder (www.picsearch.com). Företaget grundades 2001 och har idag världens näst största sökmotor för bilder på Internet, överträffade endast av Google.

Utvecklingsarbetet

Uppstart

Eftersom jag tycker om kurser där jag i ett eget snabbt tempo kan utveckla och undersöka tekniker inom datalogi insåg jag tidigt att kursen ”Avancerad individuell kurs i datalogi” skulle passa mig mycket bra. Från den obligatoriska kursen ”Programutvecklingsprojekt med mjukvarukonstruktion” (2D1362) i årskurs tre hade jag mycket goda erfarenheter av Picsearch som tillhandahöll en mycket intressant projektuppgift. Av denna anledning kontaktade jag i början av augusti Nils Andersson, CEO Picsearch AB, och frågade om de hade något nytt intressant ämne de ville ha undersökt. På ett möte i början av september presenterade Nils Andersson problemet. De önskade en stavningskontroll till sin sökmotor. Önskemålen var att stavningskontrollen i största möjliga mån skulle vara språkoberoende och fokus för systemet låg på att ge bra rättstavningsförslag (med andra ord ska inte korrekt stavade ord korrigeras och felstavade ord ska inte föreslås en felaktig stavning). Vidare presenterade Nils Andersson ett par önskemål kring prestanda. Systemet bör föreslå en alternativ stavning på mindre än 100 millisekunder och bör klara av hög last, fler än 1000 rättstavningar per minut. Att nå samtliga mål inom en prototyp var naturligtvis inte ett krav – men ett önskemål och en bra målbild under implementationen.

Inläsning

Efter att ha fått problemet presenterat för mig inledde jag en inläsningsperiod. Under september och oktober läste jag igenom rapporter, hemsidor och diskussioner kring sökmotorer och deras stavningskontroller samt rättstavning i ordbehandlare. I början av oktober hade jag ett möte med Viggo Kann, professor i datalogi på NADA, och tilldelas Magnus Rosell, doktorand i datalogi på NADA, som handledare i kursen. Av Viggo Kann fick jag tips om ytterligare litteratur inom ämnet som jag läste.

Utveckling

Efter inläsningen hade jag en bild av hur en första version av en simpel stavningskontroll skulle kunna se ut. Litteraturen [7] visade att 80 till 90 procent av alla skrivfel på en dator hade sitt ursprung i något av felen i tabell 1.

Typ	Beskrivning	Exempel
1	En bokstav för mycket	kunnglig
2	En bokstav för lite	kuglig
3	En felaktig bokstav	kumglig
4	Ett bokstavspar har bytt plats (transponerats)	kugnlig

Tabell 1

Att utifrån ett sökord generera samtliga ovan beskrivna fel och jämföra mot ett index med samtliga ord och antalet förekomster av orden är inte särskilt svårt. Antalet fel som behöver genereras beror på två faktorer, dels på hur långt sökordet är och dels på vilket alfabete man använder sig av. Jag väljer att benämna sökordets längd n och antalet

bokstäver i alfabetet benämns *a*. Fel av typ ett rättas enkelt genom att ta bort en bokstav. Det finns *n* möjliga sätt att göra detta. Fel av typ två rättas genom att skjuta in en bokstav mellan de befintliga bokstäverna (samt först och sist i ordet). Det finns *n + 1* olika placeringar av en ny bokstav och det finns *a* olika bokstäver. Med andra ord finns *(n + 1) a* olika fel att generera för att kontrollera felstavningar av typ två. En felaktig bokstav, typ tre, kontrolleras enkelt genom att ersätta varje bokstav i ordet med varje bokstav (utom den ursprungliga) i alfabetet. Detta ger *n (a -1)* möjligheter. Antalet transponeringar av bokstavspar i ett ord är *n - 1*. I tabell 1 sammanställs antalet fel.

Typ	**Antal möjliga fel**
1	*n*
2	*(n + 1) a*
3	*n (a - 1)*
4	*n – 1*
Totalt	***2na + n + a -1***

Tabell 2

För ett fem bokstäver långt ord med ett alfabete med 29 bokstäver ger detta att 323 stavningsalternativ behöver generas och kontrolleras om de är vanligare än sökordet man utgick ifrån.

Jag implementerade en strängmanipulerare som givet en sträng generar samtliga felen i tabell 1. Jag valde att göra implementationen i C#.Net eftersom jag, för egen del, har mest rutin på det språket och den utvecklingsmiljön just nu. Huruvida C# är ett lämpligt språk för en slutgiltig implementation för ett Linux-system låter jag vara osagt.

Efter att strängmanipuleraren var klar behövde jag ett index att slå upp de alternativa stavningarna av ord i. Av Picsearch fick jag då tillgång till en delmängd av deras cachade hemsidor. Datamängden jag fick var komprimerat nästan 4 GB och jag har inte tillgång till någon partition med tillräckligt utrymme för att spara en okomprimerad version av filen (som var större än 50 GB). Genom att utnyttja verktyget, SharpZipLib (http://www.icsharpcode.net/OpenSource/SharpZipLib/Default.aspx) som gör det möjligt att streama data rakt ut från den komprimerade filen, kunde jag använda den stora datamängden.

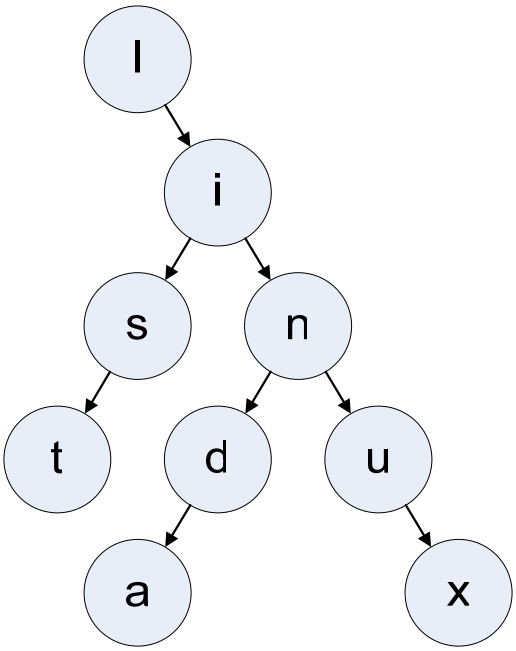
Till en början valde jag att lagra mitt index i en vanlig Hashtable. Jag lagrade endast ord som var längre än tre bokstäver och som avskiljare mellan ord specificerade jag en lång lista med tecken. För varje unikt ord räknade jag upp hur många gånger det förekom i datat. Eftersom jag hade tillgång till en enormt stor mängd hemsidor som underlag la jag in en gräns för hur många unika ord som skulle indexeras. För min dator med 512 MB minne visade det sig lagom med ett index på ungefär 3 miljoner unika ord (vilket gav ett underlag på totalt cirka 500 miljoner ord från 250 000 hemsidor).

Nu hade jag en första version jag kunde göra små tester på för att utvärdera förändringar och tillägg. Min strängmanipulerare visade sig vara tillräckligt snabb och uppslagningen

mot indexet (en Hashtable) gick även det mycket snabbt. Ett flertal problem kvarstod dock. En sökning efter till exempel ”Bill Clinton” i denna tidiga version föreslår ”will clinton” eftersom det engelska verbet will är oerhört mycket vanligare än namnet Bill. Detta var ett alvarligt problem, inte minst eftersom de allra flesta sökningarna på Picsearch görs på namn (idoler, kändisar, etc.). För att undanröja detta problem byggde jag upp ett nytt index över grannord i rådatat med inledande stor bokstav.

Med ett namn-index till hands förbättrades resultatet avsevärt. Nu stavades namn rätt i en betydligt större utsträckning. Eftersom jag tyckte att idén med att använda ordets kontext var bra byggde jag upp ett tredje index med ”vanliga” ord i par. Min förhoppning var att ord som är starkt kopplade i till exempel olika uttryck som ”bed room” inte ska rättstavas till ”red room” även om ordet ”red” är betydligt vanligare än ”bed”.

Med tre index som jag alla ville göra mycket stora önskade jag ett effektivare sätt att lagra index. Rättstavningsverktyg till ordbehandlare använder med fördel ett bloom-filter. Men eftersom jag utöver ordet behöver lagra hur ofta ordet förekommer är detta ej ett alternativ. Jag valde istället att försöka med en Trie (se figur 2).



Figur 2

I Trien kan jag utan problem i varje node lagra hur vanligt ett ord som slutar på den bokstaven är. Eftersom .Net saknar implementationer av Tries behövde jag skriva en själv. Jag implementerade till sist sex varianter av en Trie. En med en Hashtable i varje node, en med en ArrayList och en med en vanlig array. Dessutom gjorde jag två varianter av varje sådan Trie – en utan maximalt djup och en som maximerade djupet till en bestämd konstant (så att trädet inte kunde bli hur djupt som helst). Resultatet var dock dåligt. Mina Tries behövde mer minne än vad en Hashtable behövde. Huruvida detta

beror på att jag gjort dåliga implementationer eller ej låter jag vara osagt. Jag ser det inte som inom denna kursens ramar att undersöka detta.

Jag var fortfarande inte helt nöjd med resultatet. En felstavning som ”Christina Aguliara” (korrekt stavning: Christina Aguilera) skulle inte kunna korrigeras eftersom Aguilera har stavats fel på två sätt; för det första har ”l” och ”i” bytt plats och för det andra har ”a” blivit ”e”. Genom att kombinera alla varianter av typer i tabell 1 med varandra skulle fel på ”avstånd två” kunna rättas.

Typ 1	**Typ 2**	**Antal möjliga fel**
1	1	n^2
1	2	$an(n + 1)$
1	3	$n^2(a - 1)$
1	4	$n(n -1)$
2	1	$an(n + 1)$
2	2	$((n + 1)a)^2$
2	3	$an(n + 1)(a -1)$
2	4	$a(n + 1)(n – 1)$
3	1	$n^2(a - 1)$
3	2	$an(n + 1)(a -1)$
3	3	$n^2(a - 1)$
3	4	$n(a - 1)(n - 1)$
4	1	$n(n -1)$
4	2	$a(n + 1)(n - 1)$
4	3	$n(a - 1)(n - 1)$
4	4	$(n-1)^2$

Tabell 3

Utan att summera kan sägas att antalet varianter av fel ökar väldigt mycket. Min implementation av rättstavning av ord på ”avstånd två” tar drygt två sekunder, vilket naturligtvis är alldeles för lång tid. Efter en del optimering av koden blev processen snabbare (drygt en sekund), men fortfarande allt för långsam. Optimeringen gjordes främst genom att byta alla strings mot StringBuilders. Genom att begränsa felen som rättas på avstånd två till ungefär 50% erhölls önskad svarstid. Framförallt tar det lång tid att införa nya bokstäver (typ 2) och att ändra en bokstav (typ 3) därför valde jag att ej använda mig av dessa i kombination med varandra.

När jag studerade resultatet från körningar i detta läge identifierade jag ett par återkommande fel. Ord som påminde om ”head”, ”meta”, ”node” och andra ord som var mycket vanliga i html-koden jag byggde upp indexet ifrån rättstavades, naturligtvis, ofta till just dessa ord. Eftersom orden var så överrepresenterade i rådatat fick de en stor genomslagskraft i rättstavningen. Av denna anledning införde jag två sorters filter. Det ena filtret innehöll ord som inte skulle rättstavas. Det andra filtret innehöll ord som inga ord skulle rättstavas till. Oavsett hur avancerat och bra rättstavningsverktyg man

utvecklar finns risken att något ord korrigeras på ett så felaktigt sätt att man önskar stoppa det från att rättstavas – med dessa filter skapas den möjligheten.

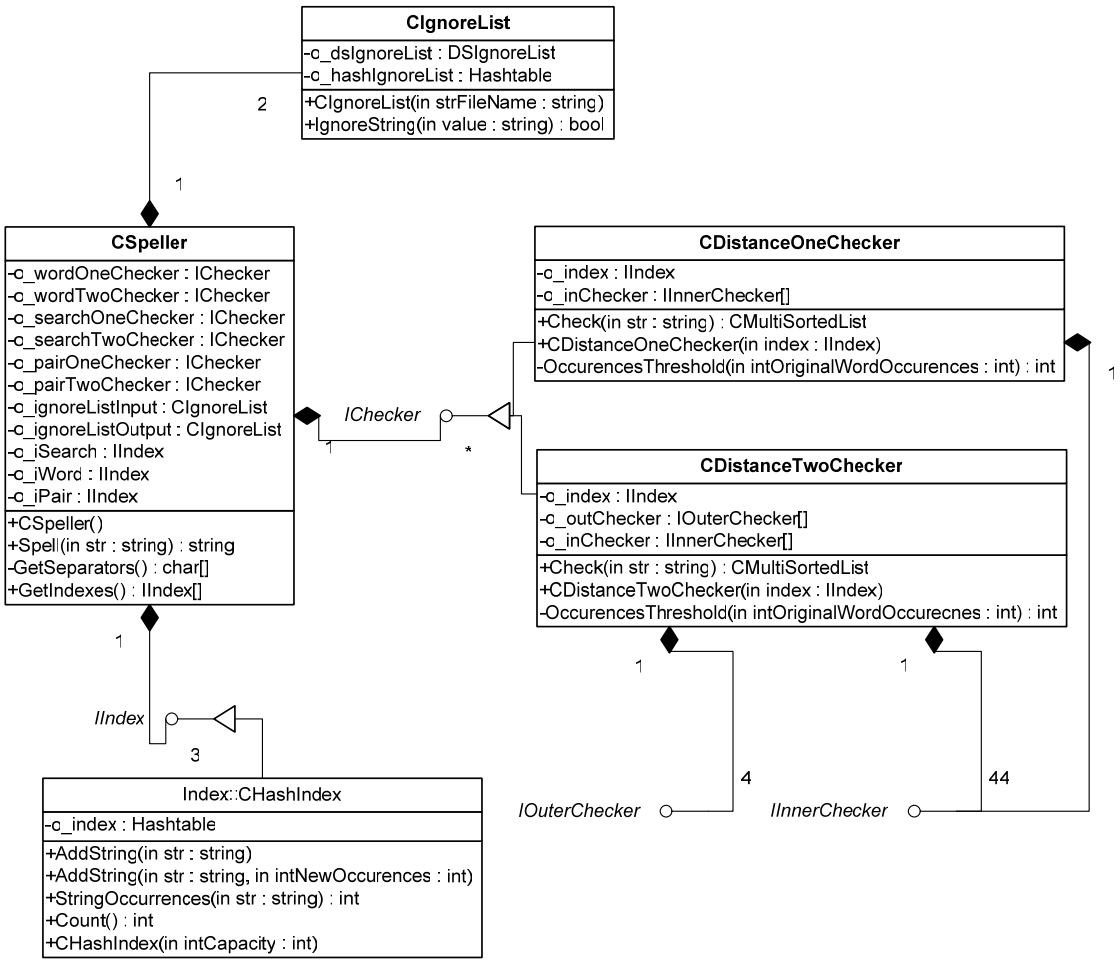
Även om filter hjälpte var jag medveten om att indexet innehöll väldigt mycket ”dålig” data (i form av delar av html-taggar, javascript, etc.). För att lösa detta bad jag Picsearch om tillgång till stora mängder av deras sökloggar. Ett dygn senare hade jag fått en fil med fem miljoner unika sökningar på Picsearch samt hur vanlig varje sökfras var. Med denna rådata som grund ersatte jag mina gamla index med tre stycken nya. Det första indexet byggdes upp runt hur vanligt varje ord var i sökningarna. Det andra indexet byggdes upp med parvisa ord (både ord och namn) och det sista indexet byggdes upp med hela sökfrasen. De nya indexena utan dålig data förbättrade resultatet avsevärt. Dessutom, tror jag, det kan finnas andra vinster med att bygga upp indexet från gamla sökningar. Framförallt rättstavar man sökningar med hjälp av just sökningar – det är rimligt att anta att sökningar är lika varandra och därför kommer utgöra ett bättre underlag än hemsidor, som inte i samma utsträckning liknar en sökning. Å andra sidan är det hemsidor användaren letar efter, att försöka efterlikna det man söker borde vara en fördel. Jag är dock av meningen att hemsidor med löptext innehåller allt för mycket ”onödig” text (om vi bortser från html-taggar, etc. som skulle gå att filtrera bort) i form av ord som inte är intressanta för den som söker. Min gissning är till exempel att personliga pronomen och prepositioner är oerhört mycket vanligare i hemsidor än vad de är i sökningar. En idé, som Picsearch kom med, är att försöka lokalisera ord i hemsidor som är viktiga för den som söker (till exempel skulle ALT-taggen för bilder kunna vara intressant i Picsearchs fall) och bygga upp index från dessa. Jag har dock ej haft möjlighet att inom denna kurs undersöka hur detta skulle fungera.

Implementation

För dokumentation av systemet hänvisas i första hand till källkoden som är både väldokumenterad och lättläst och i andra hand till projektets web-dokumentation som går att nå på: http://kjellman.dyndns.org/speller/Speller/Speller.HTM.

Viktiga klasser och interface

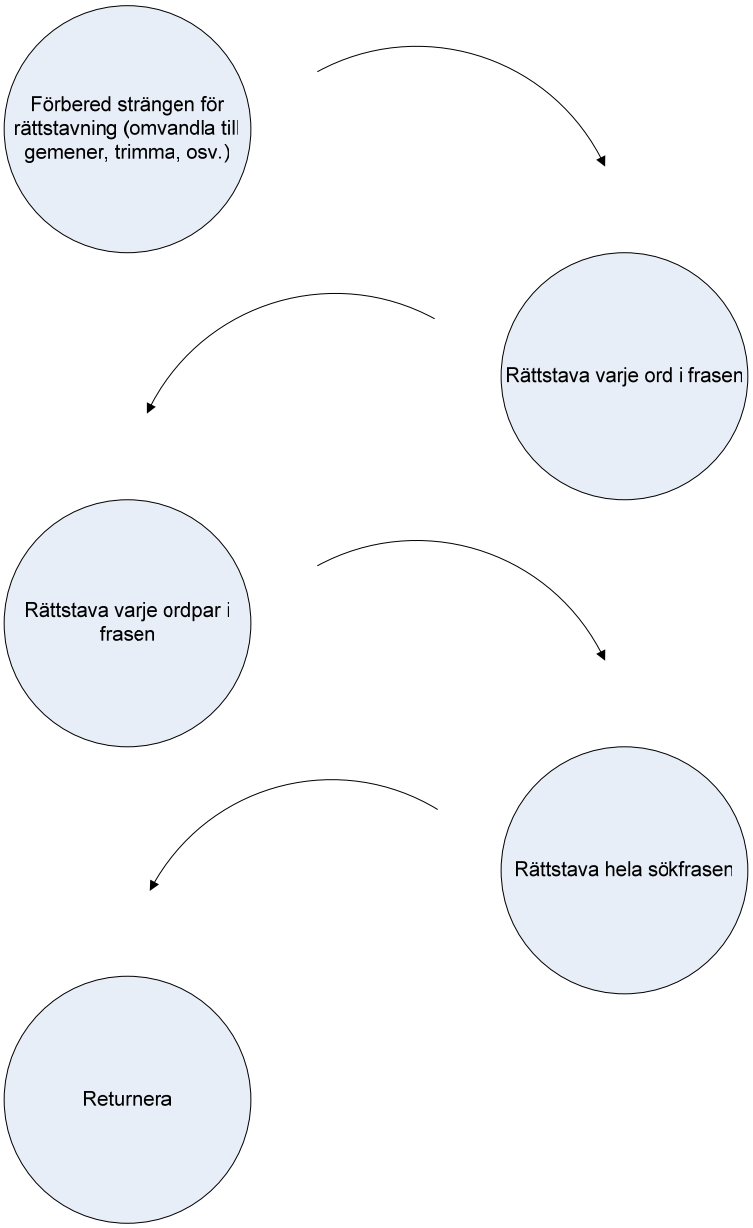
I figur 3 nedan illustreras de viktigaste klasserna. CSpeller är huvudklassen som håller tre stycken IIndex i minnet och låter ICheckers arbeta på indexena. Därutöver finns två ignore-lists som möjliggör filtrering av in- respektive ut-data.



Figur 3

En sökfras flöde genom systemet

Figur 4 visar en sökfras flöde genom systemet. Huruvida ordningen av korrigeringar är den bästa eller inte är svårt att svara på. Tanken är att börja med rättstavning av varje ord i frasen för att i ett senare läge, vid till exempel rättstavning av ordpar, kunna ändra tillbaka den tidigare korrigeringen eftersom den visade sig ovanlig i det aktuella sammanhanget. Risken finns dock att man vid rättstavning av ord rättstavar så att det eftersökta ordet ligger på avstånd tre och därmed ej kommer att rättas. Hur vanligt detta är får framtida undersökningar visa.



Figur 4

Resultat

Det är mycket svårt att utvärdera om rättstavningsverktyget är bra eller dåligt. Det behövs mycket mer testning och utvärdering än vad som kunnat vara möjlig inom denna kurs ramar. Det som använts för att utvärdera resultatet är tre sökloggar från Picsearch. Den första sökloggen innehåller felstavade sökningar efter ”Britney Spears”. Den andra loggen innehåller felstavade sökningar efter ”Christina Aguilera”. Dessa loggar har extraherats ur Picsearchs totala söklogg genom att efterfråga sökfraser med inledande ”B” avslutande ”s” och med en längd på plus/minus två tecken från ”Britney Spears”. Den sista loggen innehåller tjugotusen slumpmässiga sökfrågor till Picsearch. De två första loggarna har utvärderats utifrån hur många av fraserna som kom att korrigeras på ett korrekt sätt, hur många som kom att korrigeras på ett felaktigt sätt och hur många fraser som inte rättstavades över huvud taget. Ur den sista loggen finns endast möjlighet att göra godtyckliga urval och utifrån dessa ge ett betyg för helhetsintrycket.

Britney Spears

Sökloggen med felstavningar innehöll olika fraser som alla påminde om ”Britney Spears”. Resultatet från körningen var:

	#
Totalt antal unika stavningar	330
Ursprungliga sökningar med korrekt stavning	122 300
Ursprungliga sökningar med fel stavning	4 461
Korrekt korrigerade stavningar	254
Felaktigt korrigerade stavningar	58
Stavningar utan rättstavningsförslag	17
Korrekt korrigerade sökningar	3 726
Felaktigt korrigerade sökningar	145
Sökningar utan rättstavningsförslag	589
Procent korrekt korrigerade sökningar	**84%**
Procent felaktigt korrigerade sökningar	**3%**
Procent sökningar utan korrigering	**13%**

Tabell 4

Christina Aguilera

Sökloggen med felstavningar innehöll olika fraser som alla påminde om ”Christina Aguilera”

	#
Totalt antal unika stavningar	661
Ursprungliga sökningar med korrekt stavning	92 681
Ursprungliga sökningar med fel stavning	6 029
Korrekt korrigerade stavningar	478
Felaktigt korrigerade stavningar	142
Stavningar utan rättstavningsförslag	40
Korrekt korrigerade sökningar	5 264
Felaktigt korrigerade sökningar	254
Sökningar utan rättstavningsförslag	511
Procent korrekt korrigerade	**87%**
Procent felaktigt korrigerade	**9%**
Procent utan korrigering	**4%**

Tabell 5

Söklogg

Sökloggen bestod av 20 000 sökningar på Picsearch. Utav dessa kom 1858 stycken fraser (9%) att ges ett förslag till alternativ stavning. Hur många av dessa korrigeringar som är korrekta eller felaktiga är omöjligt att säga. Ett slumpmässigt urval av 20 sökningar ger följande resultat:

Sök fras	Stavningsförslag	Rätt	Fel	Osäkert
don jones, guest book	dow jones guest book		X	
Serene	Serena			X
Venessa	Vanessa	X		
Revolutionary war- americans and british	revolutionary war americans and british			X
freedom isnt free	Freedom sint free		X	
jerusalem,	jerusalem	X		
surreal & serene	surreal serena			X
aamer khan	amir khan	X		
Hood up cars	hooked up cars		X	
Does pipette look like?	does pipette look like	X		
daniel hendrex	daniel hendrix	X		
Does pipette look like?	does pipette look like	X		
Seiko sbj	seiko ssj			X
carpal tunnel syndrome,	carpal tunnel syndrome	X		
franck balestracci	france balestracci		X	
lesani bosy	lesani boys	X		
Tug	Pug			X
towns and cites landscapes	towns and cities landscapes	X		
Does pipette look like?	does pipette look like	X		
Vampire	vampire	X		

Tabell 6

Om en korrigering är korrekt eller ej är inte lätt att svara på eftersom man inte kan fråga användaren som ställde frågan vad den egentligen sökte efter. Därför är tabell sex ytterst godtycklig och resultatet måste undersökas ytterligare för att kunna användas vidare.

Prestanda

Kravet som Picsearch hade på ett rättstavningsverktyg var att det inom 100 ms skulle ge ett svar. Problem med prestanda får man först när man försöker rättstava ord på avstånd två. I dagsläget uppfyller inte prototypen prestandakraven. En uppslagning tar ungefär 500 ms (på en AMD Athlon XP 2500+). Jag anser detta dock som tillräckligt nära kravet för att det ska gå att, vid behov, optimera till 100 ms. Ett byte av programmeringsspråk kan förbättra svarstiden avsevärt.

Förbättringsmöjligheter

Utvärdering

Genom att noga utvärdera sökningar och resultat kan säkerligen stora förbättringar göras utan särskilt stora förändringar i systemet. Genom att anpassa konstanten som bestämmer hur mycket vanligare ett ord i indexet måste vara för att det ska anses som rättstavat kan säkerligen stora framsteg göras. Eventuellt bör man förändra ordningen som rättstavningen görs på. Idag korrigeras först varje ord, därefter varje ordpar och tillsist hela frasen. En annan ordning kanske kan förbättra resultatet mycket.

Optimering

Dagens prestandaproblem går säkerligen att optimera bort. Till att börja med bör man undersöka hur mycket snabbare en implementation i C/C++ skulle bli. Därefter bör man utvärdera hur snabba/långsamma sträng-operationer är i språket man gjort implementationen.

Skalbarhet

Om man önskar använda systemet i stor skala och i professionella syften ökar kraven på produkten. I det läget är det viktigt att ha mycket stora index. För att kunna ha det behöver man dela applikationen över flera datorer. För just denna applikation är detta mycket enkelt. Genom att låta en dator vara ansvarig för alla fraser som börjar med bokstaven ”a”, en annan för alla fraser som börjar på ”b” och så vidare sprider man snabbt lasten till lika många datorer som man har bokstäver i alfabetet. Man kan vidare tänka sig att en dator blir ansvarig för alla fraser som börjar på ”a” och som är mellan tre och fem tecken långa, en annan för fraser som börjar på ”a” och som är mellan fyra och sex tecken långa och så vidare.

Soundex

Soundex är ett sätt att indexera ord (eller namn) baserat på hur ordet låter (jämför: fonetisk skrift) istället för hur det stavas. Att addera ett soundex-index kan möjligen förbättra resultatet. Det finns dock en stor risk att allt för många ord kommer att sammanfalla och att detta gör indexet oanvändbart.

Index

Man bör undersöka andra datastrukturer för att lagra indexet. Att ha ett mycket stort index att göra uppslag mot är viktigt och därför bör Tries och andra datatyper undersökas och implementeras. De borde ta mindre minne i anspråk vid lagring av index utan att hastigheten blir allt för låg.

Självlärande

Idag byggs indexet upp från gamla sökningar. Utan större svårigheter skulle indexet kunna vara levande i den betydelsen att varje sökning rättstavningsverktyget får in läggs in i indexet (lämpligen i en annan, lågt prioriterad, tråd).

Cache

Att implementera en cache i rättstavningsverktyget som kommer ihåg vanliga fraser och vad de ska rättstavas till ökar säkerligen svarstiden för majoriteten av frågor vilket lämnar mer processortid över till ovanliga sökfraser.

Slutsatser

Att implementera ett rättstavningsverktyg till en sökmotor som ger mycket bra resultat är inte en lätt uppgift. Genom att undersöka och rätta fel på avstånd ett och avstånd två av typerna: en bokstav för mycket, en bokstav för lite, en felaktig bokstav och ett bokstavspar har transponerats erhålls bra resultat. Resultatet kan säkerligen förbättras med små justeringar av parametrar som till exempel hur mycket vanligare ett ord behöver vara för att det ska anses som rättstavat. Att använda sig av flera index har visat sig bra. Implementationen använder sig av tre index. Det första indexet innehåller enstaka ord, det andra indexet innehåller parvisa ord och det tredje indexet innehåller hela sökfraser från tidigare användare. Speciellt indexet med parvisa ord har visat sig vara mycket användbart då det underlättar rättstavning av personnamn och uttryck med närliggande ord. Denna prototyp visar att det enkla angreppssättet till problemet har stora möjligheter att ge bra resultat men betydligt mer utvärdering av systemet krävs innan det kan användas i ett skarpt system.

Källförteckning

[1] Google's Feature History
http://www.googleguide.com/feature_history.html (3/12 2004)

[2] Gigablast Adds Spelling Suggestions
http://www.searchengineshowdown.com/newsarchive/000746.shtml (3/12 2004)

[3] Why You Should Work at Google
http://labs.google.com/why-google.html (3/12 2004)

[4] Spelling Corrections (Suggestions)
http://www.googleguide.com/spelling_corrections.html (3/12 2004)

[5] Spellchecker Finally Finished
http://www.gigablast.com/rants.html#spellchecker (3/12 2004)

[6] Missplel – ett generellt verktyg för generering av stavfel
Linus Ericsson
Royal Institute of Technology, Stockholm Sweden

[7] Detection of Spelling Errors in Swedish Not Using a Word List En Clair
Rickard Domeij, Joachim Hollman och Viggo Kann
Royal Institute of Technology, Stockholm Sweden

[8] Evaluating a spelling support in a search engine
Hercules Dalianis
Royal Institute of Technology, Stockholm Sweden

[9] Improving precision and recall using a spell checker in a search engine
Monsour Sarr
Royal Institute of Technology, Stockholm Sweden

[10] Search Query Spellchecking
Avi Rappoport