

Apr 06, 05 13:21

CNtag_main.cpp

Page 1/21

```

#include <map>
#include <hash_map>
#include <math.h>
#include <string>
#include <time.h>
#include <windows.h>
#include <boost/pool/pool_alloc.hpp>

using namespace std;
using namespace stdext;
using namespace boost;

// Överför strängar till interna beteckningar som tal
// Dessutom lagras strängarna som tal
// Observera att denna programversion de facto aldrig skriver ut de valda taggar
na, men
// att detta i princip är trivialt att lägga till
map<string, int> tokens;
map<string, int> tags;
vector<string> tags2int;

int nowtokenID = 1;
int nowtagID = 0;

// Maximal tillåten längd för suffix
const int maxsufflen = 1;

// Tillståndet EFTER anrop med viss input är fullständigt deterministiskt
// Delta-tillstånd mellan före och efter anrop är det däremot inte.
// const innebär det första, men brukar ibland implicera även det andra
// Dessa både funktioner översätter strängar till deras ID-tal
const int tokenCode(string token)
{
    map<string, int>::iterator i;
    if ((i = tokens.find(token)) == tokens.end())
    {
        tokens[token] = nowtokenID;

        return nowtokenID++;
    }

    return i->second;
}

const int tagCode(string tag)
{
    map<string, int>::iterator i;
    if ((i = tags.find(tag)) == tags.end())
    {
        tags[tag] = nowtagID;

        tags2int.push_back(tag);

        printf("%d\n", nowtagID);
        return nowtagID++;
    }

    return i->second;
}

// Klassen word_sequence kapslar i någon mån in detaljerna i att representera en
följd av fyra ord.

```

Wednesday April 06, 2005

Apr 06, 05 13:21

CNtag_main.cpp

Page

```

// Denna konstant används för att packa ord + tagg-ID. Den innebär också be-
ningar vid suffixhantering.
// (2^20 bitar lite lite)
const int masklistan[4] = {0, 1048575, (1 << 30) - 1048575, (1 << 30) - 1};

// En word_sequence kan överföras till 2 64-bitarstal för snabbare indexerin-
g.
typedef pair<long long, long long> wordpair;

class word_sequence
{
public:
    int data[4];

    // Defaultkonstruktor, tämligen tråkig sådan.
    word_sequence()
    {
        for (int x = 0; x < 4; x++)
        {
            data[x] = -1;
        }
    }

    // Castingoperator till long long-par.
    inline operator wordpair() const
    {
        return make_pair( ((long long) data[0]) << 32) + data[1],
ng long) data[2]) << 32) + data[3]);
    }

    // Avsedd för eventuell användning med hash_map. Denna tycks, förvä-
nog, innebära sämre prestanda.
    inline operator size_t() const
    {
        size_t result = 0;
        for (int x = 0; x < 4; x++)
        {
            result = (result << 1) + ((result & (1 << 31)) > 0);
            result ^= data[x];
        }

        return result;
    }

    // Värderna ges till en word_sequence på ett stack-liknande sätt.
    void push_back(int word, int tag)
    {
        for (int x = 1; x < 4; x++)
        {
            data[x - 1] = data[x];
        }
        data[3] = word + (tag << 20);
    }

    // Maskering används för att endast plocka ut särdragen för ett vis-
nster, d.v.s. tagg respektive ord
    // för specifika ord i mönstret
    void mask(int mask, word_sequence& dest)
    {
        for (int x = 0; x < 4; x++)
        {
            dest.data[x] = (data[x] & masklistan[(mask & 3)]);
            mask >>= 2;
        }
    }
}

```

CNtag_main.cpp

Apr 06, 05 13:21

CNtag_main.cpp

Page 3/21

```

    }
}
// Jämförelseoperator.
bool inline operator <(const word_sequence& seq2) const
{
    const word_sequence& seq1 = *this;

    for (int x = 0; x < 4; x++)
    {
        int diff = seq1.data[x] - seq2.data[x];
        if (diff == 0) continue;

        return diff < 0;
    }
    return false;
}
// Jämförelseoperator.
bool operator ==(const word_sequence& seq2) const
{
    const word_sequence& seq1 = *this;

    for (int x = 0; x < 4; x++)
    {
        int diff = seq1.data[x] - seq2.data[x];
        if (diff == 0) continue;
        return false;
    }
    return true;
};

// Cdouble används internt för flyttal. double rekommenderas då precisionen i float kan vara begränsande.
// Dock minskar float minnesanvändningen och kan öka prestanda betydligt beroende på arkitektur.
typedef double Cdouble;

// En partvector innehåller par med tagnummer och frekvenser. En alltför stor träningskorpus kräver
// i praktiken 32-bitarstal här i stället.
typedef vector<pair<unsigned __int8, unsigned __int16>, pool_allocator<pair<unsigned __int8, unsigned __int16>,
default_user_allocator_new_delete,
details::pool::null_mutex,
131072>> > partvector;

// Data lagras i en map mellan (maskerade) mönster och resultatvärden.
typedef map<wordpair, pair<int, partvector >, less<wordpair >,
fast_pool_allocator<pair<wordpair, pair<int, partvector >>,
default_user_allocator_new_delete,
details::pool::null_mutex,
131072>> > DATATYPE;

// Det högsta mönstret vi använder.
const int end = 192;

// Data lagras separat för olika suffixlängder.
DATATYPE data[end * (maxsuffixlen + 1)];

// En lista med vilka mönster som faktiskt används. En sanning med modifikation.

```

Wednesday April 06, 2005

Apr 06, 05 13:21

CNtag_main.cpp

Page

```

bool goodlist[256];

void makegood()
{
    // Räkna antalet ingående aspekter i mönstret, stoppa graford + tagg
    // samma position
    for (int x = 0; x < 256; x++)
    {
        int count = 0;
        int count2 = 0;
        int y = x;
        for (int z = 0; z < 4; z++)
        {
            if (y & 1)
            {
                count++;
            }
            if (y & 2)
            {
                count2++;
            }
            if ((y & 3) == 3)
            {
                goto end;
            }
            y >>= 2;
        }

        printf("%d %d\n", count, count2);

        if (x & 32) count += 5;

        // Värde under fem i stället hjälper inte så mycket.
        // En gång 6519, 993 mot 6516, 990, repade sig snabbt mot 6519.
993
        goodlist[x] = (count + count2 < 4); // 3 är dock lite lågt.
    }
end:;
}

// Mappning mellan absolut frekvens och ena dimensionen bland de sextion vektorerna i
// gsfacken
const inline int countindex(Cdouble count)
{
    if (count < 4)
    {
        return 0;
    }
    if (count < 10)
    {
        return 1;
    }
    if (count < 50)
    {
        return 2;
    }
    return 3;
}

// Mappning mellan relativ frekvens och ena dimensionen bland de sextion vektorerna i
// gsfacken
const inline int countprob(Cdouble prob)

```

CNtag_main.cpp

Apr 06, 05 13:21

CNtag_main.cpp

Page 5/21

```

{
    if (prob > 0.98) return 0;
    if (prob > 0.82) return 1;
    if (prob > 0.50) return 2;

    return 3;
    /*if (prob > 0.99) return 0;
    if (prob > 0.93) return 1;
    if (prob > 0.70) return 2;
    return 3;*/
}

// wl anger det högsta ordnummer som används i korpusen, ger snabb åtskillnad me
llan kända och okända ord
int wl = 0;

// Viktningsmatrisen, det allra heligaste
Cdouble matrix[(maxsufflen + 1) * end][4][4][2];
// Viktningsmatrisens onde tvillingbror, faktormatrisen som påverkar vilket värd
e optimeringsmatrisen
// provar nästa gång
Cdouble laststep[(maxsufflen + 1) * end][4][4][2];

// Fulsätt att ge värden om antalet lyckade okända, kända och totalt antal kända
till statusrapporteringen
// i main
int gsuccu, gsucck, gknown;

// Kort funktion som ur korpusen slår upp den mest sannolika taggen för ett ord,
med utgångspunkt
// enbart i unigram.
const inline int likelyTac(int toc)
{
    int x = 16;

    word_sequence mainseq, refseq, refseq2;

    mainseq.push_back(toc, 0);
    mainseq.push_back(0, 0);

    mainseq.mask(x, refseq);
    DATATYPE::iterator i;
    if ((i = data[x].find(refseq)) == data[x].end())
    {
        return -1;
    }

    Cdouble sum = i->second.first;
    int maxtag = -1;
    double maxvalue = 0;

    for (partvector::iterator j = i->second.second.begin(); j != i->second.s
econd.end(); j++)
    {
        if (maxvalue < j->second)
        {
            maxvalue = j->second;
            maxtag = j->first;
        }
    }

    return maxtag;
}

```

Apr 06, 05 13:21

CNtag_main.cpp

Page

```

}

// Värdena nedan används för att förlagra redan beräknade resultat efter fö
iterationen
// maxlen styr hur stor provtexten får vara. Notera att meningsgränsen kan
oväntat mycket
// till detta värde.
const int maxlen = 10000;
const int BC = 3; // beforecount

// prevals innehåller ord-id, tagg-id respektive uppskattad tagg-id för näst
d
int prevals[maxlen][3];

// en lista med själva graforden, behövs i suffixhanteringen
char tokenlist[maxlen][256];

// de gamla beräknade fyroordskontexterna för varje position för varje följd
vector<word_sequence> gamla[BC * BC];

//
Cdouble gamlascores[maxlen][BC * BC][148][2];
int maxbearer[maxlen][BC];
int maxindex[maxlen][BC];

pair<Cdouble, Cdouble> tag(Cdouble* require, int xchange, Cdouble* value)
{
    // Öppna fil med provtext.
    FILE* fil = fopen("eval.txt", "r");
    char tlf[2048];
    word_sequence mainseq[BC * BC], refseq[BC * BC], refseq2[BC * BC], m
eq2[BC * BC];
    Cdouble factor[BC * BC];

    // Initiera värden
    for (int n = 0; n < BC * BC; n++)
        factor[n] = 1;

    // Dessa är NÄSTA ord och tagg, d.v.s. sista positionen i fyraordsk
ten
    int toC = -1;
    int taC = 0;

    // Dessa är de vi faktiskt taggar just nu
    int lasttaC = -1;
    int lasttoC = 0;

    // Totalt antal kända och okända ord
    int succk = 0;
    int succu = 0;

    // Totalt antal ord, används i förlagringsadresseringen
    int tot = 0;
    int known = 0;

    // Tidigare använt värde för poängsättning
    Cdouble corrrsum = 0;

    // Sträng med föregående token, används i suffixhantering
    char oldtoken[256] = {0};
}

```

Apr 06, 05 13:21

CNtag_main.cpp

Page 7/21

```

// Lagring av hur stor listan med gamla sekvenser var innan vi började l
äsa, avgör om detta är första iterationen
int size = gamla[0].size();

// Poäng, två stycken i ett försök att inte överträna
Cdouble scores[2] = {0, 0};

// Antingen fortsätt till rätt storlek, eller läs in nytt tills filen to
m
))))
while ((tot < size) || (size == 0 && !feof(fil) && fgets(tlf, 2048, fil
))
{
    char token[256];
    char tag[256];

    lasttoC = toC;
    lasttaC = taC;

    // Fyll i det aktuella ordet i kontexten. Vi anger ingen tagg hä
r, den vill vi ju ta reda på!
    for (int subindex = 0; subindex < BC * BC; subindex++)
    {
        mainseq[subindex] = mainseq2[subindex];
        mainseq[subindex].push_back(lasttoC, (lasttaC == -1) ? -
1 : -2);
    }

    // Om oläst, läs på riktigt, annars finns det i prevals.
    if (tot >= gamla[0].size())
    {
        if (sscanf(tlf, "%256s %256s", token, tag) == 2)
        {
            prevals[tot][0] = tokenCode(token);
            prevals[tot][1] = taC = tagCode(tag);
            strcpy(tokenlist[tot], token);
        }
        else
        {
            prevals[tot][0] = 0;
            prevals[tot][1] = -1;
        }
        prevals[tot][2] = likelyTac(prevals[tot][0]);
    }

    strcpy(token, tokenlist[tot]);
    toC = prevals[tot][0];
    taC = prevals[tot][1];

    // Är detta ord känt?
    bool thisknown = (lasttoC < wl) && (lasttoC > 0);

    // Lagra i så fall det!
    known += thisknown;
    int len = strlen(oldtoken);

    // Fortsätt på var och en av de BC * BC vägarna.
    for (int subindex = 0; subindex < BC * BC; subindex++)
    {
        // Fyll på nästa ord, med en uppskattad tagg. Denna anta
s ge heuristik för att ge säkrare poäng.
        mainseq[subindex].push_back(toC, (taC == -1) ? -1 : prev

```

Apr 06, 05 13:21

CNtag_main.cpp

Page

```

als[tot][2]);

// En intrikat bitflagga som gör att vi ibland inte
måste slå upp i data för att inse att något inte
// finns
int lastnotfound = 0xff;

// Har vi att göra med en total omräkning, eller så
här vägen i princip likadan ut förra gången, med
// en mindre matrisförändring
// Här kunde vi eventuellt även lägga in stöd för a
värde från tidigare subindex. Ett problem är att
// kopiering av 2 double-värden för varje möjlig ta
ta tar *längre* tid än en omkörning med matching == true!
bool matching = tot < gamla[0].size() && (gamla[sub
]][tot] == mainseq[subindex]) && (xchange <= end * (maxsufflen + 1) + 1);

// Om det inte matchar får vi allt se till att noll.
a.
if (!matching)
{
    for (int x = 0; x < nowtagID; x++)
    {
        gamlascores[tot][subindex][x][0] = 0;
        gamlascores[tot][subindex][x][1] = 0;
    }
}

// Detta betyder i korthet att om vi enbart ändrat
r för okända ord behöver vi inte göra ett dugg om "vårt" ord
// är känt och inte en total omtagning behövs
if (!matching || (thisknown ^ (xchange >= end)))
    // Loopa enbart över de mönster som behöver
as, alla eller ett.
for (int x = 1 + matching * ((xchange % end
)); x < end + (-end + 1 + (xchange % end)) * matching; x++)
{
    // Om mönstret ej används, avbryt f
if (!goodlist[x]) continue;

/*
Detta gäller om man inte har suffix
ring.
if ((x & 16) && !thisknown)
{
    continue;
}*/

// Om ett mönster som är en delmängd
detta mönster inte hittats kan vi ge upp att hitta detta.
// Undviker en uppslagning i data.
if ((x & lastnotfound) == lastnotfound)
{
    continue;
}

// Skall vi titta på suffix? Avgörs
m aktuellt ord ingår i mönstret, samt om ordet är känt, förstås.
bool suffix = (x & 16) && (!thisknown)

// Beroende på om vi har suffixhant
maskerar vi lite olika.

```

Apr 06, 05 13:21

CNtag_main.cpp

Page 9/21

```

        if (suffix)
        {
            mainseq[subindex].mask(x - 16, r
efseq[subindex]);
        }
        else
        {
            mainseq[subindex].mask(x, refseq
[subindex]);
        }

        // Faktor för aktuell suffixposition, vi
modulerar teckenkoder med 32 för att rymmas någorlunda i 20 bitar
        int snum = 1;

        // Behandla antingen suffixlängd 0 (orde
t känt/ingår ej) eller de möjliga suffixen
        for (int suff = suffix; suff < 1 + maxsu
fflen * suffix; suff++)
        {
            // Om vi har suffix måste vi fyl
la på i refseq, den sekvens vi slår upp.
            if (suffix)
            {
                if (len - suff <= 0) bre
ak;
                refseq[subindex].data[2]
+= (((int) oldtoken[len - suff]) + 256) % 32) * snum;
                snum *= 32;
            }
            DATATYPE::iterator i;
            // Var vi slår upp beror på om d
et är suffix eller ej.
            * suffix);
            fseq[subindex])) == data[dataindex].end())
            {
                if (data[dataindex].size
() && suff <= 1) lastnotfound = x;
                if (suff > 1)
                {
                    refseq[subindex]
.data[2] /= 32;
                }
                else
                    break;
            }
            // Matrisvikter är separata för
okända ord med mönster där aktuellt ord inte används, och
// kända ord där det inte använd
s. MEN, korpusdata är identiska. Däremot gäller inte detta
// för suffixdata. Alltså är dat
aindex inte samma sak som index.
            int index = x + end * (!(thiskno
wn) + (suff - 1) * suffix);

            // Här kan vi komma fram till at
t vi är på fel nivå, jämfört med det som har ändrats. Fortsätt.

```

Wednesday April 06, 2005

Apr 06, 05 13:21

CNtag_main.cpp

Page 1

```

        if (matching && index != xcl
) continue;
        // Vi har summan av totala
et förekomster förlagrad.
        Cdouble sum = i->second.fir
// Iterera över alla förekom
e taggar, lägg till poäng i enlighet med vikterna. Algoritmens hjärta.
        for (partvector::iterator j
>second.second.begin(); j != i->second.second.end(); j++)
        {
            int i = countprob(j
ond / sum) * 4 + countindex(j->second);
            for (int i2 = 0; i2
i2++)
            {
                Cdouble add
= j->second * (matching ? value[i * 2 + i2] : (matrix[index][0][i][i2] * !
ing));
                gamlascores
[subindex][j->first][i2] += addition;
            }
        }
        // Fyll på oldtoken
strcpy(oldtoken, token);
        // Förläng "gamla", om nödvändigt.
        if (tot + 1 > gamla[0].size())
        {
            for (int subindex = 0; subindex < BC * BC; subindex
gamla[subindex].resize(tot + 1);
        }
        // Fyll på själva värdena i gamla
        for (int subindex = 0; subindex < BC * BC; subindex++)
            gamla[subindex][tot] = mainseq[subindex];
        // De BC högsta poängvärdena.
        // För var och en av dessa lagras sedan ett maxbearer-värde
ken tagg max-värdet hör till.
        // Detta används senare, bl.a. under backtracking.
        // maxindex anger för varje maxbearer vilken av de föregåen
garna som mest sannolikt ledde dit,
        // ungefär.
        Cdouble max[BC];
        // Nollställ.
        for (int q = 0; q < BC; q++)
        {
            max[q] = 0;
            maxbearer[tot][q] = -1;
            maxindex[tot][q] = 0;
        }
        // Slit inte i onödan om vi är mitt i en meningsgräns.

```

CNtag_main.cpp

Apr 06, 05 13:21

CNtag_main.cpp

Page 11/21

```

        if (lasttoC != 0)
        {
            // Poängsumma för de BC olika värdena på närmast föregående tagg för de BC största taggarna.
            Cdouble sums[BC][BC];

            Cdouble sum = 0;
            // Iterera genom alla kända taggar.
            for (int x = 0; x < nowtagID; x++)
            {
                // De verkliga poängvärdena från var och en av vägarna för denna tagg.
                Cdouble vals[BC * BC];

                // Totalsumman, denna taggs poäng.
                Cdouble val = 0;

                // Beräkna poängvärdena.
                for (int subindex = 0; subindex < BC * BC; subindex++)
                {
                    val += (vals[subindex] = gamlascores[tot][subindex][x][1] * factor[subindex]);
                }

                // Är detta större än något värde i listan med BC presumtiva största?
                for (int index = 0; index < BC; index++)
                {
                    // Just det, ställ just den frågan!
                    if (val > max[index])
                    {
                        // Flytta ner nödvändiga element
                        for (int index2 = BC - 1; index2 > index; index2--)
                        {
                            max[index2] = max[index2 - 1];
                            maxbearer[tot][index2] = maxbearer[tot][index2 - 1];
                            maxindex[tot][index2] = maxindex[tot][index2 - 1];
                        }
                        for (int subindex = 0; subindex < BC; subindex++)
                        {
                            sums[index2][subindex] = sums[index2 - 1][subindex];
                        }
                        // Lagra nya värdena.
                        max[index] = val;
                        maxbearer[tot][index] = x;

                        // Beräkna summor, av BC * BC vägar finns det bara BC olika föregående taggar, vi vill veta vilken tagg, inte vilken väg, som bidrog mest.
                        for (int pindex = 0; pindex < BC; pindex++)
                    }
                }
            }

```

Wednesday April 06, 2005

Apr 06, 05 13:21

CNtag_main.cpp

Page 1

```

            sums[index][pindex]
            for (int index2 = 0; index2 < BC; index2++)
            {
                sums[index2][pindex] += vals[pindex * BC + index2];
            }
            // Finn max bland dessa.
            maxindex[tot][index] = 0;
            for (int pindex = 0; pindex < BC; pindex++)
            {
                if (sums[index][pindex] > sums[index][maxindex[tot][index]])
                {
                    maxindex[tot][index] = pindex * BC;
                }
            }
            break;
        }
        sum += val;
    }
    // Vid konstrerandrat av nästa stegs mainseq2 måste vara intakta.
    // Lagra tillfälligt i mainseq3.
    word_sequence mainseq3[BC * BC];

    // Summa av viktning mellan vägar. Används för nästa steg.
    Cdouble factorsum = 0;

    for (int pindex = 0; pindex < BC; pindex++)
    {
        // Här räknar vi antalet kända, oavsett om de mest sannolika.
        succk += (maxbearer[tot][pindex] == lasttoC) ? (*BC - pindex) : (!pindex);
        succu += (maxbearer[tot][pindex] == lasttoC) ? (*BC - pindex) : (!thisknown) ? (*BC - pindex) : (!pindex);
        for (int subindex = 0; subindex < BC; subindex++)
        {
            // Den kommenterade raden är helt åtgärden. Vad tänkte jag när jag skrev den?
            //mainseq3[pindex * BC + subindex] = mainseq2[maxindex[tot][pindex] + subindex];
            mainseq3[pindex * BC + subindex] = mainseq2[subindex * BC];
            mainseq3[pindex * BC + subindex].push_back(lasttoC, maxbearer[tot][pindex]);
        }
        // Den ger upphov till måååånga dubbelräkningar här, kort sagt begränsades antalet vägar.
        // *if (subindex && mainseq3[pindex * BC + subindex - 1])
        {
            printf("%d %d %d\n", pindex, index, tot);
        }
    }

```

CNtag_main.cpp

Apr 06, 05 13:21

CNtag_main.cpp

Page 13/21

```

        }*/
        // Värst av allt, poängen var inte rättv
isande, då de var korrekta!
        factorsum += (factor[pindex * BC + subin
dex] = sums[pindex][subindex]);
    }
}
// Summan har kanske blivit 0. Då blir alla vidare sanno
likheter 0. Inte roligt.
if (factorsum == 0)
{
    for (int index = 0; index < BC * BC; index++)
    {
        factor[index] = 1;
    }
    factorsum = BC * BC;
}
// Det är enklare om vi kan tro att den viktigaste allti
d kommer först, när vi har backtracking med
// flera alternativ
for (int pindex = 1; pindex < BC; pindex++) // Maxfaktor
n först
{
    word_sequence sekv;
    Cdouble aFactor = factor[pindex];

    if (aFactor > factor[0])
    {
        factor[pindex] = aFactor;
        sekv = mainseq3[pindex];
        mainseq3[0] = sekv;
        factor[0] = aFactor;
    }
}
// Normera och kopiera tillbaka till mainseq2
// Ge värde till värderingsfunktionen, värdera placering
som mest sannolik högre. Klokt?
for (int pindex = 0; pindex < BC * BC; pindex++)
{
    factor[pindex] /= factorsum;
    scores[tot & 1] += (-1 + 2 * (maxbearer[tot][pin
dex] == lasttaC)) * factor[pindex] * (1 + (pindex == 0));
    mainseq2[pindex] = mainseq3[pindex];
}
// Lite "asserts", låsta till vissa BC-värden
/*if (factor[0] < factor[1])
{
    printf("%d\n", tot);
}
if (factor[0] + factor[1] + factor[2] < factor[3] + fact
or[4] + factor[5])
{
    printf("Q: %d\n", tot);
}*/
// En gång i tiden valde vi bara en maximal...

```

Wednesday April 06, 2005

Apr 06, 05 13:21

CNtag_main.cpp

Page 1

```

        //corrsum += (lasttaC == maxbearer) * (max / sum);
if (sum == 0) sum = 1;
        // corrsum += (gamlascores[tot][lasttaC][0] * gamla
s[tot][lasttaC][1] + (-1 + 1 * (lasttaC == maxbearer)) * max) / sum;
        /* if (lasttoC >= wl)
        {
            succu += lasttaC == maxbearer;
        }
        else
        {
            succk += lasttaC == maxbearer;
        }*/
    }
else
    {
        // Om noll, nollställ!
for (int subindex = 0; subindex < BC * BC; subindex++)
        {
            mainseq2[subindex].push_back(lasttoC, -1);
            factor[subindex] = 1.0 / BC * BC;
        }
    }
    tot++;

}
lasttaC = -1;
// Nu skall vi backtracka!
succk = 0;
succu = 0;

for (int x = 0; x < 2; x++)
{
    // Vikta ned, relativt.
    scores[x] /= 128;
    // Kanske borde man vikta med hur många som täcks in bland
daterna?
    //scores[x] += succk + succu;
}
// Gå bakåt
for (tot--; tot > 0; tot--)
{
    // Gå mellan de möjliga kandidaterna
for (int i = 0; i < BC; i++)
    {
        // Om vi har markeringen "-1" vet vi inte, förra va
ingsslut. Ta första bästa.
        // Då är det en fördel om första också är bästa, se
eringen ovan.
if (lasttaC == -1 || maxbearer[tot][i] == lasttaC)
        {
            // Ta den tagg som denna innehåller.
            lasttaC = maxbearer[tot][i];
        }
        // Är ordet känt?
        bool thisknown = (prevals[tot - 1][0] && pre
[tot - 1][0] <= wl);
        succk += thisknown * (lasttaC == prevals[tot
][1]);
        succu += !thisknown * (lasttaC == prevals[tot

```

CNtag_main.cpp

Apr 06, 05 13:21

CNtag_main.cpp

Page 15/21

```

1][1]) * ((bool) prevals[tot - 1][0]);
// Om detta inte är en meningsavskiljare, poängs
ätt!
ttaC == prevals[tot - 1][1]);
// Gå bakåt längs den väg som bidrog mest till d
enna, plocka ut dess tagg.
lasttaC = gamla[maxindex[tot][i]][tot].data[1] >
> 20;
int toc = gamla[maxindex[tot][i]][tot].data[1] &
1048575;
// Om taggen var 0, skall den senaste taggkoden
vara -1, oavsett allt annat.
if (toc == 0) lasttaC = -1;
break;
}
}
fclose(fil);
gsuccu = succu;
gsucck = succk;
gknown = known;
return make_pair(scores[0], scores[1]);
}
}
}

int main(int argc, char** argv)
{
    makegood();

    // Öppna träningsfilen.
    FILE* fil = fopen("train.txt", "r");
    char tlf[2048];
    word_sequence mainseq, refseq;
    int line = 0;
    char oldtoken[256] = {0};

    int tt = 0;

    while (!feof(fil) && fgets(tlf, 2048, fil))
    {
        line++;
        char token[256];
        char tag[256];
        int toC;
        int taC;

        if (sscanf(tlf, "%256s%256s", token, tag) == 2)
        {
            toC = tokenCode(token);
            taC = tagCode(tag);
        }
        else
        {
            toC = 0;
            taC = -1;
        }
    }
}

```

Wednesday April 06, 2005

Apr 06, 05 13:21

CNtag_main.cpp

Page 1

```

int len = strlen(oldtoken);

mainseq.push_back(toC, taC);
for (int x = 1; x < end; x++)
{
    if (!goodlist[x]) continue;
    mainseq.mask(x, refseq);
    int snum = 1;

    for (int suff = 0; suff <= maxsufflen; suff++)
    {
        int count = (bool) (x & 1);
        count += (bool) (x & 4);
        count += (bool) (x & 64);

        int count2 = (bool) (x & 2);
        count2 += (bool) (x & 8);
        count2 += (bool) (x & 128);
        // Minnesanvändningen kan minskas genom att
a mindre information om suffix.
        // Samtidigt är all information bra informa
        // Här är ett exempel på villkor.
        //if (suff >= 1 && count * 2 + count2 >= 4)
k;

        // Om vi har suffix, men ett mönster utan s
, sluta!

        if (suff >= 1 && !(x & 16))
        {
            break;
        }
        else
        {
            // Fyll i sekvensdata.
            if (suff == 1)
            {
                mainseq.mask(x - 16, refseq);
            }
            if (suff >= 1)
            {
                if (len - suff <= 0) break;
                refseq.data[2] += (((int) c
                snum *= 32;
            }
        }

        // Finns posten? Om inte, skapa den.
        DATATYPE::iterator i;
        if ((i = data[x + end * suff].find(refseq))
ata[x + end * suff].end())
        {
            i = data[x + end * suff].insert(mak
r(refseq, make_pair(0, 0))).first;
        }

        // Leta upp om det finns data för vår tagg,
till där, annars skapa ny.
        int newtaC = mainseq.data[2] >> 20;

```

CNtag_main.cpp

```

Apr 06, 05 13:21          Cntag_main.cpp          Page 17/21
    for (partvector::iterator j = i->second.second.b
egin(); j != i->second.second.end(); j++)
    {
        if (j->first == newtaC && j->second < 65
535)
        {
            i->second.first++;
            j->second++;

            if (j->second == 65535) printf("
Hej!\n");

            goto dontadd;
        }
        if (newtaC >= 0)
        {
            i->second.second.push_back(make_pair(new
taC, 1));
        }
        else
        {
            // Här ser vi till att data om suffix in
te ökas om det för just detta ord redan
            // fanns en representation av just detta
mönster. Motivering till detta finns i
uppsatsen.
            if (suff == 0)
            {
                break;
            }
        }
    }

    strcpy(oldtoken, token);

    if (toC == 0)
    {
        mainseq.push_back(0, -1);
        mainseq.push_back(0, -1);
        mainseq.push_back(0, -1);
        mainseq.push_back(0, -1);
    }

    // Skriv ut lite information
    for (int x = 0; x < 256; x++)
        if (goodlist[x]) printf("%d %d %d\n", tokens.size(), x, data[x].s
ize());

    // "Nollställ" matrisen. Lägre vikt till multiplikationsdelen, låt ettan
i den nollställda poänglistan
    // väga över.
    for (int x = 0; x < (maxsufflen + 1) * end; x++)
    {
        for (int y = 0; y < 4; y++)
        {
            for (int z = 0; z < 4; z++)
            {
                for (int i2 = 0; i2 < 2; i2++)
                {
                    matrix[x][y][z][i2] = 1 / (1.0 + 16383.0

```

```

Apr 06, 05 13:21          Cntag_main.cpp          Page 1
    * i2);
                                laststep[x][y][z][i2] = -1.5;
    }
    }
    }
    // Vi kan läsa in en färdig matris
    if (argc > 1)
    {
        FILE* ut = fopen(argv[1], "rb");
        printf("VAL:%d\n", fread(matrix, sizeof(Cdouble) * ((maxsufflen
1) * end) * 16 * 2, 1, ut));
        fclose(ut);
    }

    wl = nowtokenID;
    int pass = 0;
    Cdouble setback = 1.0001f;
    Cdouble maxlimit = 4096;

    srand(time(0));

    // En gräns för hur stora värden får bli. Lite väl stor?
    // Kanske, men när påtvingad på en befintlig matris skedde försämri
    // i tillförlitligheten om den sattes lägre!
    Cdouble lim = 1024*1024*1024.0f*262144;

    while (true)
    {
        // value är en matris som beskriver alla förändringar i en
ll "matrisrad"
        Cdouble value[32];
        for (int x = 0; x < 32; x++)
            value[x] = 0;

        pair<Cdouble, Cdouble> last = tag(require, end * (maxsufflen
) + 1, value);
        pass++;

        // Tala om hur det går.
        printf("Entering: %d, %d, %lf, %d, %d, %d\n", 0, pass, last.first + 1
second, gsuccu, gsucck, gknown);
        double sum = 0;
        double sum2 = 0;
        int c = 0;
        int bc = 0;

        for (int x = 1; x < end * (maxsufflen + 1); x++)
        {
            //if ((goodlist[x % end]) && x != end && (x < 2 * e
(x & 16)) && data[x].size() || ()
            // Ändra bara på värden som faktiskt används, ungef
            if ((goodlist[x % end]) && (data[x].size() || (x / e
= 1 && data[x - end].size()))
            {
                // 4 * 4 vikter
                for (int i = 0; i < 16; i++)
                {
                    // Både serie 1 och serie 2
                    for (int i2 = 0; i2 < 2; i2++)
                    {

```

```

Apr 06, 05 13:21          CNtag_main.cpp          Page 19/21
// Begränsa.
[x][0][i][i2], 1.0f/lim);
matrix[x][0][i][i2] = max(matrix
[x][0][i][i2], lim);
matrix[x][0][i][i2] = min(matrix
// Här följer lite godtyckliga r
egler för hur laststep skall förändras, samt återförs till 1
// när det har blivit stooort.
Cdouble abs = fabs(laststep[x][0
][i][i2]) / maxlimit;
if (abs > 32)
{
    abs = 0;
    laststep[x][0][i][i2] =
setback * ((rand() / (RAND_MAX / 2)) ? 1 : -1);
    maxlimit *= 1.00001f;
}
Cdouble oldstep = laststep[x][0
][i][i2];
if (abs > 1.0/64.0)
{
    laststep[x][0][i][i2] *=
-7.0f;
}
else if (abs > 1.0/256.0)
{
    laststep[x][0][i][i2] *=
-5.0f;
}
else if (abs > 1.0/1024.0)
{
    laststep[x][0][i][i2] *=
-4.0f;
}
else if (abs > 1.0/3072.0)
{
    laststep[x][0][i][i2] *=
-2.5f;
}
else if (abs > 1.0/4000.0)
{
    laststep[x][0][i][i2] *=
-1.2f;
}
else
{
    laststep[x][0][i][i2] *=
-1.1f;
}
//printf("%f ", laststep[x][0][i
]);
// Slumpa ett värde i intervalle
t.
laststep[x][0][i][i2] = ((lasts
tepx[x][0][i][i2] + oldstep) / 2) / RAND_MAX * rand()) - oldstep;
// Beräkna en summa av alla last

```

Wednesday April 06, 2005

```

Apr 06, 05 13:21          CNtag_main.cpp          Page 2
step
sum2 += fabs(laststep[x][0
2]);
// Räkna hur många laststep
// Här följer lite godtyckliga r
egler för hur laststep skall förändras, samt återförs till 1
// när det har blivit stooort.
Cdouble abs = fabs(laststep[x][0
][i][i2]) / maxlimit;
if (abs > 32)
{
    abs = 0;
    laststep[x][0][i][i2] =
setback * ((rand() / (RAND_MAX / 2)) ? 1 : -1);
    maxlimit *= 1.00001f;
}
Cdouble oldstep = laststep[x][0
][i][i2];
if (abs > 1.0/64.0)
{
    laststep[x][0][i][i2] *=
-7.0f;
}
else if (abs > 1.0/256.0)
{
    laststep[x][0][i][i2] *=
-5.0f;
}
else if (abs > 1.0/1024.0)
{
    laststep[x][0][i][i2] *=
-4.0f;
}
else if (abs > 1.0/3072.0)
{
    laststep[x][0][i][i2] *=
-2.5f;
}
else if (abs > 1.0/4000.0)
{
    laststep[x][0][i][i2] *=
-1.2f;
}
else
{
    laststep[x][0][i][i2] *=
-1.1f;
}
//printf("%f ", laststep[x][0][i
]);
// Slumpa ett värde i intervalle
t.
laststep[x][0][i][i2] = ((lasts
tepx[x][0][i][i2] + oldstep) / 2) / RAND_MAX * rand()) - oldstep;
// Beräkna en summa av alla last

```

CNtag_main.cpp

Apr 06, 05 13:21

CNtag_main.cpp

Page 21/21

```

                                if (bc != (int) next.fir
st)
                                {
                                    bc = next.first;
                                    printf("%d\t%d\t%
d\t%f\t%f\t%f\t%f\t%d\n", x, i, i2, matrix[x][0][i][i2], laststep[x][0][i][i2], next
.first + next.second, gsucck, gsuccu);
                                    fflush(stdout);
                                    last = next;
                                    // Öka inte steg
et lika fort om det gick bra. Kanske borde man återföra till nära 1?
                                    // I CNtags begy
nnelse gav det dock inte lika bra resultat. Systemet betar sig emellertid annorl
unda
                                    // på flera sätt
nu.
                                    laststep[x][0][i
][i2] = oldstep;
                                    char tlf[255];
                                    sprintf(tlf, "%l
f", last);
                                    FILE* ut = fopen
("utfil.tbl", "wb");
                                    fwrite(matrix, s
izeof(Cdouble) * ((maxsufflen + 1) * end) * 16 * 2, 1, ut);
                                    fclose(ut);
                                }
                                }
                                sum += matrix[x][0][i][i2];
                                }
                                // När vi växlar till nästa matrisrad måste vi n
eutralisera den sista ändringen.
                                // Det är tänkbart att alla rader i value redan
är 0, men vinsten är högst 1/33.
                                tag(require, x, value);
                                for (int x = 0; x < 32; x++)
                                {
                                    value[x] = 0;
                                }
                                }
                                // Skriv lite statistik.
                                printf("Vikt!%f\t%f\t%f\t%d\n", sum, sum2, c, maxlimit);
                                // Om vi nått 54 pass avslutar vi. Då vi har trasslat lite med m
innet genom Boost kan en del
                                // destruktörer bli ledsna. Vi hoppar över dem här...
                                if (pass == 54) ExitProcess(0);
                                }
}

```