Subject: String near match algorithm
Posted by Didier on Fri, 31 Jul 2009 19:00:20 GMT
View Forum Message <> Reply to Message

For data importing I have written a small algorithm intended to compare 2 strings and indicate if the two strings are close to each other or not.

I used it to avoid importing into a DB data with misspelled names

```
int correlation(const String& a, const String& b)
{
 int res=0;
 const char* A = a.Begin();
 const char* B = b.Begin();

 const int Al = a.GetLength();
 const int Bl = b.GetLength();
 const int deltaL = Al - Bl;

 int As, Bs, intersectLength;
 int subRes;

 int matchPatternMinLength = max(2, min(b.GetLength(), a.GetLength())/3);


 for (int offset = (matchPatternMinLength-Bl); offset <= (Al-matchPatternMinLength); offset++ )
 {
  // range calculation
  if (offset < 0)
  {
   As = 0;
   Bs = -offset;
   if (offset < deltaL) intersectLength = offset + Bl;
   else                 intersectLength = Al;
  }
  else
  {
   As = offset;
   Bs = 0;
   intersectLength = Al - offset;
   if (offset <= deltaL) intersectLength = Bl;
   else                  intersectLength = Al-offset;
  }

  subRes = 0;
  for (int c = 0; c<intersectLength; c++)
  {
   if( A[As+c] == B[Bs+c]) ++subRes;
  }
  if (subRes >= matchPatternMinLength) res += subRes;
 }

 // taking string length in account
 if ( deltaL > 0 )
 {
  res -= deltaL;
 }
 else
 {
  res += deltaL;
 }
 return res;
}


inline bool CompareDistance(const String& a, const String& b)
{
 if (correlation(a, b) >= max(2, min(a.GetLength(), b.GetLength())*3/5)) return true;
 return false;
}
```

The algorithm is based on a basic signal processing technique like in sonars or radars (correlation) adapted to strings.
It is not very optimized but it works fine. The function CompareDistance() is only here to introduce a threshold value: 3/5 in the code.

How to use it:
If CompareDistance(stringA, stringB) returns true, then the two strings are considered "near match" ==> maybe misspelled names.

Maybe this can be useful to somebody (if it doesn't exist anywhere else) .

Subject: Re: String near match algorithm
Posted by koldo on Sat, 01 Aug 2009 08:53:14 GMT
View Forum Message <> Reply to Message

Thank you Didier.

Do you have some base to get the differences between two strings ?.

Best regards
Koldo

Subject: Re: String near match algorithm
Posted by Didier on Sat, 01 Aug 2009 09:56:22 GMT
View Forum Message <> Reply to Message

What do you meen when saying
Quote:Do you have some base to get the differences between two strings ?.

??

Subject: Re: String near match algorithm
Posted by koldo on Sat, 01 Aug 2009 14:02:20 GMT
View Forum Message <> Reply to Message

Hello Didier

I meant if you have the algorithm or something else to compare two strings and report the differences.

Best regards
Koldo

Subject: Re: String near match algorithm
Posted by Didier on Sat, 01 Aug 2009 14:19:32 GMT
View Forum Message <> Reply to Message

Hello Koldo,

I don't have such an algorithm for the moment but I'll think about it, and I have an idea ...

I'll try it out to see if it works.

Subject: Re: String near match algorithm
Posted by Didier on Sun, 02 Aug 2009 21:39:12 GMT
View Forum Message <> Reply to Message

Hi Koldo,

I wrote a variant that extracts a histogram of letter validity: low score=>bad letter.
The algorithm is mainly intended to detect near matches so it is not well suited to finding which letters are different (a basic compare loop should work better).

At least it reuses the "near match" information.

```
int correlation(const String& a, const String& b, int* aHistogram, int* bHistogram)
{
 int res=0;
 const char* A = a.Begin();
 const char* B = b.Begin();

 const int Al = a.GetLength();
 const int Bl = b.GetLength();
 const int deltaL = Al - Bl;

 int As, Bs, intersectLength;
 int subRes;

 int matchPatternMinLength = max(2, min(b.GetLength(), a.GetLength())/3);

 // init de l'histogramme
 for (int c = 0; c<Al; c++)
 {
  aHistogram[c] = 0;
 }

 for (int c = 0; c<Bl; c++)
 {
  bHistogram[c] = 0;
 }

 for (int offset = (matchPatternMinLength-Bl); offset <= (Al-matchPatternMinLength); offset++ )
 {
  // determination des bornes
  if (offset < 0)
  {
   As = 0;
   Bs = -offset;
   if (offset < deltaL) intersectLength = offset + Bl;
   else                 intersectLength = Al;
  }
  else
  {
   As = offset;
   Bs = 0;
   intersectLength = Al - offset;
   if (offset <= deltaL) intersectLength = Bl;
   else                  intersectLength = Al-offset;
  }

  subRes = 0;
  for (int c = 0; c<intersectLength; c++)
  {
   if( A[As+c] == B[Bs+c]) ++subRes;
  }
  if (subRes >= matchPatternMinLength)
  {
   res += subRes;

   // creation de l'histogramme
   for (int c = 0; c<intersectLength; c++)
   {
    if( A[As+c] == B[Bs+c])
    {
     aHistogram[As+c] += subRes;
     bHistogram[Bs+c] += subRes;
    }
   }
  }
 }

 // prise en compte du nbr de caracteres
 if ( deltaL > 0 )
 {
  res -= deltaL;
 }
 else
 {
  res += deltaL;
 }
 return res;
}
```

Here are some test results:

```
COMPARE  (123456789 , 123456789 )     corr= 9   ==>  NEAR_MATCH
Histogram:  - 9 - 9 - 9 - 9 - 9 - 9 - 9 - 9 - 9
Histogram:  - 9 - 9 - 9 - 9 - 9 - 9 - 9 - 9 - 9
```

```
COMPARE  (123456789 , 12345678 )     corr= 7   ==>  NEAR_MATCH
Histogram:  - 8 - 8 - 8 - 8 - 8 - 8 - 8 - 8 - 0
Histogram:  - 8 - 8 - 8 - 8 - 8 - 8 - 8 - 8

COMPARE  (123456789 , 1234567 )     corr= 5   ==>  NEAR_MATCH
Histogram:  - 7 - 7 - 7 - 7 - 7 - 7 - 7 - 0 - 0
Histogram:  - 7 - 7 - 7 - 7 - 7 - 7 - 7

COMPARE  (123456789 , 123456 )     corr= 3   ==>  NEAR_MATCH
Histogram:  - 6 - 6 - 6 - 6 - 6 - 6 - 0 - 0 - 0
Histogram:  - 6 - 6 - 6 - 6 - 6 - 6

COMPARE  (123456789 , 12345 )     corr= 1   ==>  -------
Histogram:  - 5 - 5 - 5 - 5 - 5 - 0 - 0 - 0 - 0
Histogram:  - 5 - 5 - 5 - 5 - 5

COMPARE  (123456789 , 1234 )     corr= -1   ==>  -------
Histogram:  - 4 - 4 - 4 - 4 - 0 - 0 - 0 - 0 - 0
Histogram:  - 4 - 4 - 4 - 4

COMPARE  (123456789 , 123 )     corr= -3   ==>  -------
Histogram:  - 3 - 3 - 3 - 0 - 0 - 0 - 0 - 0 - 0
Histogram:  - 3 - 3 - 3

COMPARE  (123456789 , 12 )     corr= -5   ==>  -------
Histogram:  - 2 - 2 - 0 - 0 - 0 - 0 - 0 - 0 - 0
Histogram:  - 2 - 2

COMPARE  (123456789 , 1 )     corr= -8   ==>  -------
Histogram:  - 0 - 0 - 0 - 0 - 0 - 0 - 0 - 0 - 0
Histogram:  - 0

COMPARE  (123456789 , 12345  89 )     corr= 7   ==>  NEAR_MATCH
Histogram:  - 7 - 7 - 7 - 7 - 7 - 0 - 0 - 7 - 7
Histogram:  - 7 - 7 - 7 - 7 - 7 - 0 - 0 - 7 - 7

COMPARE  (123456789 , 1234589 )     corr= 5   ==>  NEAR_MATCH
Histogram:  - 5 - 5 - 5 - 5 - 5 - 0 - 0 - 2 - 2
Histogram:  - 5 - 5 - 5 - 5 - 5 - 2 - 2

COMPARE  (123456789 , 12  56789 )     corr= 7   ==>  NEAR_MATCH
Histogram:  - 7 - 7 - 0 - 0 - 7 - 7 - 7 - 7 - 7
Histogram:  - 7 - 7 - 0 - 0 - 7 - 7 - 7 - 7 - 7

COMPARE  (123456789 , 1256789 )     corr= 5   ==>  NEAR_MATCH
Histogram:  - 2 - 2 - 0 - 0 - 5 - 5 - 5 - 5 - 5
Histogram:  - 2 - 2 - 5 - 5 - 5 - 5 - 5
```

```
===========
COMPARE  (123 , 123 )     corr= 3   ==>  NEAR_MATCH
Histogram:  - 3 - 3 - 3
Histogram:  - 3 - 3 - 3

COMPARE  (123 , 234 )     corr= 2   ==>  NEAR_MATCH
Histogram:  - 0 - 2 - 2
Histogram:  - 2 - 2 - 0

COMPARE  (123 , 345 )     corr= 0   ==>  -------
Histogram:  - 0 - 0 - 0
Histogram:  - 0 - 0 - 0

COMPARE  (123 , 3456 )     corr= -1   ==>  -------
Histogram:  - 0 - 0 - 0
Histogram:  - 0 - 0 - 0 - 0

COMPARE  (123 , 2345 )     corr= 1   ==>  -------
Histogram:  - 0 - 2 - 2
Histogram:  - 2 - 2 - 0 - 0

COMPARE  (123 , 01234 )     corr= 1   ==>  -------
Histogram:  - 3 - 3 - 3
Histogram:  - 0 - 3 - 3 - 3 - 0

COMPARE  (123 , 120023 )     corr= 1   ==>  -------
Histogram:  - 2 - 4 - 2
Histogram:  - 2 - 2 - 0 - 0 - 2 - 2

COMPARE  (SMITH , SMITH )     corr= 5   ==>  NEAR_MATCH
Histogram:  - 5 - 5 - 5 - 5 - 5
Histogram:  - 5 - 5 - 5 - 5 - 5

COMPARE  (SMITH , SMITHE )     corr= 4   ==>  NEAR_MATCH
Histogram:  - 5 - 5 - 5 - 5 - 5
Histogram:  - 5 - 5 - 5 - 5 - 5 - 0

COMPARE  (SMITH , SMISS )     corr= 3   ==>  NEAR_MATCH
Histogram:  - 3 - 3 - 3 - 0 - 0
Histogram:  - 3 - 3 - 3 - 0 - 0

COMPARE  (SMITH , SMIF )     corr= 2   ==>  NEAR_MATCH
Histogram:  - 3 - 3 - 3 - 0 - 0
Histogram:  - 3 - 3 - 3 - 0

COMPARE  (SMITH , SMICH )     corr= 4   ==>  NEAR_MATCH
Histogram:  - 4 - 4 - 4 - 0 - 4
```

```
Histogram:  - 4 - 4 - 4 - 0 - 4

COMPARE  (SMITH , SWITH )     corr= 4   ==>  NEAR_MATCH
Histogram:  - 4 - 0 - 4 - 4 - 4
Histogram:  - 4 - 0 - 4 - 4 - 4

COMPARE  (SMITH , SMATH )     corr= 4   ==>  NEAR_MATCH
Histogram:  - 4 - 4 - 0 - 4 - 4
Histogram:  - 4 - 4 - 0 - 4 - 4

COMPARE  (SMITH , CMITE )     corr= 3   ==>  NEAR_MATCH
Histogram:  - 0 - 3 - 3 - 3 - 0
Histogram:  - 0 - 3 - 3 - 3 - 0

COMPARE  (SMITH , CEMISS )     corr= 1   ==>  -------
Histogram:  - 0 - 2 - 2 - 0 - 0
Histogram:  - 0 - 0 - 2 - 2 - 0 - 0

COMPARE  (REPETITION , REPETITION )     corr= 16   ==>  NEAR_MATCH
Histogram:  - 10 - 13 - 10 - 13 - 13 - 13 - 13 - 13 - 10 - 10
Histogram:  - 10 - 13 - 10 - 13 - 13 - 13 - 13 - 13 - 10 - 10

COMPARE  (REPETITION , REPETETION )     corr= 13   ==>  NEAR_MATCH
Histogram:  - 9 - 13 - 9 - 13 - 13 - 4 - 9 - 9 - 9 - 9
Histogram:  - 9 - 9 - 9 - 13 - 9 - 4 - 13 - 13 - 9 - 9

COMPARE  (REPETITION , REPETECION )     corr= 11   ==>  NEAR_MATCH
Histogram:  - 8 - 11 - 8 - 11 - 8 - 3 - 0 - 8 - 8 - 8
Histogram:  - 8 - 8 - 8 - 11 - 8 - 3 - 0 - 11 - 8 - 8

COMPARE  (REPETITION , REPETESION )     corr= 11   ==>  NEAR_MATCH
Histogram:  - 8 - 11 - 8 - 11 - 8 - 3 - 0 - 8 - 8 - 8
Histogram:  - 8 - 8 - 8 - 11 - 8 - 3 - 0 - 11 - 8 - 8

COMPARE  (REPETITION , REPETESSION )     corr= 7   ==>  NEAR_MATCH
Histogram:  - 5 - 5 - 5 - 5 - 5 - 0 - 0 - 3 - 3 - 3
Histogram:  - 5 - 5 - 5 - 5 - 5 - 0 - 0 - 0 - 3 - 3 - 3

COMPARE  (REPETITION , REPETISSION )     corr= 11   ==>  NEAR_MATCH
Histogram:  - 6 - 6 - 6 - 9 - 6 - 6 - 3 - 6 - 3 - 3
Histogram:  - 6 - 9 - 6 - 6 - 9 - 9 - 0 - 0 - 3 - 3 - 3

COMPARE  (REPETITION , RIPITISSION )     corr= 9   ==>  NEAR_MATCH
Histogram:  - 4 - 0 - 4 - 0 - 4 - 7 - 3 - 6 - 3 - 3
Histogram:  - 4 - 0 - 4 - 3 - 7 - 7 - 0 - 0 - 3 - 3 - 3

COMPARE  (REPETITION , RIPPITTISSION )     corr= 4   ==>  -------
Histogram:  - 4 - 0 - 4 - 0 - 0 - 0 - 4 - 7 - 3 - 3
```

Histogram: - 4 - 0 - 4 - 0 - 0 - 0 - 4 - 4 - 0 - 0 - 3 - 3 - 3

COMPARE (REPETITION , RIKKITISSION)   corr= 4   ==> -------
Histogram: - 0 - 0 - 0 - 0 - 0 - 3 - 3 - 6 - 3 - 3
Histogram: - 0 - 0 - 0 - 0 - 3 - 3 - 3 - 0 - 0 - 3 - 3 - 3

Subject: Re: String near match algorithm
Posted by koldo on Sun, 02 Aug 2009 22:36:18 GMT
View Forum Message <> Reply to Message

Hello Didier

Perhaps this is something close to:

COMPARE (REPETITION , REPETETION)
Patterns:
1. REPET
2. E
3. I
4. TION
REPETITION = 1+3+4
REPETETION = 1+2+4

I have seen two interesting papers from Colin Percival (second is his PhD Thesis). This is if we have an insane interest in this area
- http://www.daemonology.net/papers/bsdiff.pdf
- http://www.daemonology.net/papers/thesis.pdf

Best regards
Koldo

Subject: Re: String near match algorithm
Posted by Mindtraveller on Sat, 26 Dec 2009 20:56:04 GMT
View Forum Message <> Reply to Message

Your test results above look very promising.
Does this library support UTF-8 encoded text?

Subject: Re: String near match algorithm
Posted by koldo on Sun, 27 Dec 2009 07:58:16 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Sat, 26 December 2009 21:56Your test results above look very promising. Does this library support UTF-8 encoded text?

Well, in fact it is designed for binary files so it seems it can gets raw data.

I had this code abandoned in my Desktop for months... I am just including it in Functions4U (it has BSD license).

Best regards
Koldo

Subject: Re: String near match algorithm
Posted by Mindtraveller on Sun, 27 Dec 2009 12:25:45 GMT
View Forum Message <> Reply to Message

OK. If I search for a word in text, I should split text into words and apply near search algorithm to each word.
1. Is it right?
2. Which value should I compare function result to in each case?

Subject: Re: String near match algorithm
Posted by koldo on Sun, 27 Dec 2009 15:37:26 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Sun, 27 December 2009 13:25OK. If I search for a word in text, I should split text into words and apply near search algorithm to each word.
1. Is it right?
2. Which value should I compare function result to in each case?

Sorry Mindtraveller

BSdiff and BSpatch work like diff and patch but for binary files instead of for text files.

To compare strings you should have to go to the original Didier algorithms. Perhaps he has something newer .

Best regards
Koldo

Subject: Re: String near match algorithm
Posted by Didier on Mon, 28 Dec 2009 11:12:24 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Sun, 27 December 2009 13:25OK. If I search for a word in text, I should

split text into words and apply near search algorithm to each word.
1. Is it right?
2. Which value should I compare function result to in each case?

Hello Mindtraveller and Koldo,

My small algoritm can compare whatever you like if you modify it a bit. But it is originally intended for string comparison.

and it compares the complete texts ==> this means that if you want to find a near match inside a phrase you will have to compare all the words individually
===> 1: YES

===> 2: The following function is what I use to determine if it is a near match or not.

```
inline bool CompareDistance(const String& a, const String& b)
{
 if (correlation(a, b) >= max(2, min(a.GetLength(), b.GetLength())*3/5)) return true;
 return false;
}
```

The (3/5) value is a threshold value that you can tune to your needs but this one works pretty well.
The max() and min() functions are to treat corner cases where the words become very small, in fact it is directly linked to the following code in the correlation() function: int matchPatternMinLength = max(2, min(b.GetLength(), a.GetLength())/3)'

I'm gonna make a zipped project with all in it.

Subject: Re: String near match algorithm
Posted by koldo on Mon, 28 Dec 2009 15:25:58 GMT

View Forum Message <> Reply to Message

Hello all

Definitely Didier String comparison and BSDiff binary file comparison are not comparable, but indeed they are interesting.

Implemented BSDiff in Functions4U, there have been compared original and slightly changed files: theide.exe, a .xls file, a .doc file and a .txt file.

For all of them following appears the size of new file, compressed and bsdiff patch.

This is not a serious benchmark but it seems that, if size really matters, the patch files are much smaller than the .7z compressed files, so BSDiff is doing it very well for binary and for text files too (200 times better in this case) .

See how:

- TheIde.exe size comparison
New version: 4,854,784 bytes
New version .7z compressed: 2,032,605 bytes (42% size reduction)
BSDiff patch file: 284,739 bytes (6% size reduction)
-> So BSDiff file is 7 times smaller than .7z file

- .xls file size comparison
New version: 10,399,232 bytes
New version .7z compressed: 692,397 bytes (7% size reduction)
BSDiff patch file: 3,744 bytes (0.04% size reduction)
-> So BSDiff file is 184 times smaller than .7z file

- .doc file size comparison
New version: 259,072 bytes
New version .7z compressed: 150,212 bytes (58% size reduction)
BSDiff patch file: 3,123 bytes (1% size reduction)
-> So BSDiff file is 48 times smaller than .7z file

- .txt file size comparison
New version: 1,248,891 bytes
New version .7z compressed: 34,461 bytes (3% size reduction)
BSDiff patch file: 174 bytes (0.01% size reduction)
-> So BSDiff file is 198 times smaller than .7z file

Best regards
Koldo

Subject: Re: String near match algorithm
Posted by Didier on Mon, 28 Dec 2009 18:27:37 GMT
View Forum Message <> Reply to Message

Here is a package containing a self contained header that implements all I have.

I did a little cleaning and introduced some template parameters (to enable arbitrary string type) and documentation.

It is intended to be put in Bazaar

File Attachments

1) NearMatchCompare.zip, downloaded 344 times