Recognition with strings

Let's assume that objects are represented as ordered sequences of discrete elements.
Generally these elements (letters of some alphabet) are nominal data – there is no well defined distance measure between letters.
The next difficulty results from different length of letters – words sequences.

Problems:

- String matching
- Edit distance
- String matching with errors
- String matching with ~ (don't care) symbol

Preparing of these materials was supported by a grant from Liechtenstein, Iceland and Norway through EEA Grants and Norway Grants and co-financed by domestic funds

String matching

String matching consists in finding all appearances of the given *pattern* in *text*.
Pattern is denoted as ***x***=x[0..m-1]; length of pattern equals *m*.
Text is denoted as ***y***=y[0..n-1]; its length is equal *n*.
Pattern and text have letters from alphabet A, of size *a*.
String matching most often uses window (usually of size *m*). For given window position (with specific *shift* with respect to the beginning of the text) algorithm performs character matching *attempt*. (An attempt results in finding pattern in text or not). After an attempt window is moved toward the end of the text.

Differences between string matching algorithms lie in the method for determining shift (increment) after an attempt.

Preparing of these materials was supported by a grant from Liechtenstein, Iceland and Norway through EEA Grants and Norway Grants and co-financed by domestic funds

Practical approach

```
p = y;
while ((p = strstr(p, x)) != 0)
   printf("%d\n", p-y);
```

This solution is optimal regarding implementation time. It can be also quite fast, because `strstr` function is implemented neatly in the standard library.
(It would be instructive, to check implementation of strstr (or its counterpart in your favorite programming language)).

Preparing of these materials was supported by a grant from Liechtenstein, Iceland and Norway through EEA Grants and Norway Grants and co-financed by domestic funds

Naïve solution

```
for (int j=0; j <= n-m; ++j)
{
   for (int i=0; i < m && x[i] == y[i+j]; ++i);
   if (i == m)
      printf("%d\n", j);
}
```

No preprocessing needed.
Comparing from left to right.
Constant shift of 1.
Computational complexity $\boldsymbol{O}(mn)$.
Expected number of comparisons $2n$.

Preparing of these materials was supported by a grant from Liechtenstein, Iceland and Norway through EEA Grants and Norway Grants and co-financed by domestic funds

Boyer-Moore algorithm

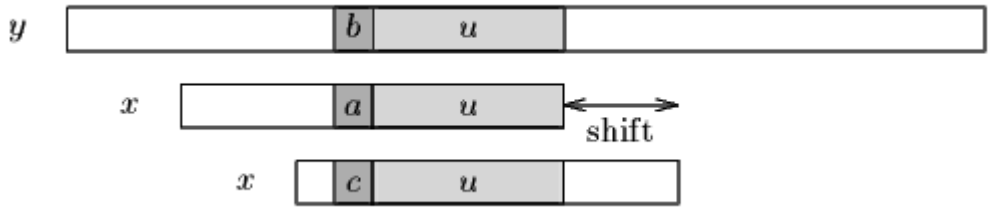
Algorithm has preprocessing phase, during which shift arrays are calculated: good suffix array (Gs) and bad character array (Bc).

```
for (int j=0; j <= n-m;)
{
   for (int i=m-1; i>=0 && x[i] == y[i+j]; --i);
   if (i < 0)
   {
      printf("%d\n", j);
      j += Gs[0];
   }
   else
      j += max(Gs[i], Bc[y[i+j]]-m+1+i);
}
```

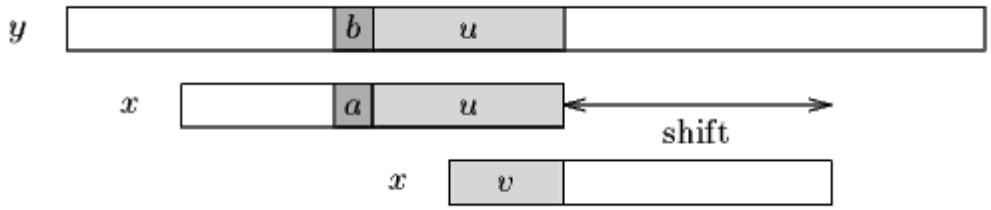

Preparing of these materials was supported by a grant from Liechtenstein, Iceland and Norway through EEA Grants and Norway Grants and co-financed by domestic funds

BM algorithm – good suffix

We have found different characters x[i]=*a* and y[i+j]=*b*. The suffix x[i+1..m-1]=*u* is the same in pattern and text. The shift is computed as the rightmost occurrence of *u* preceded by other than *a* character.



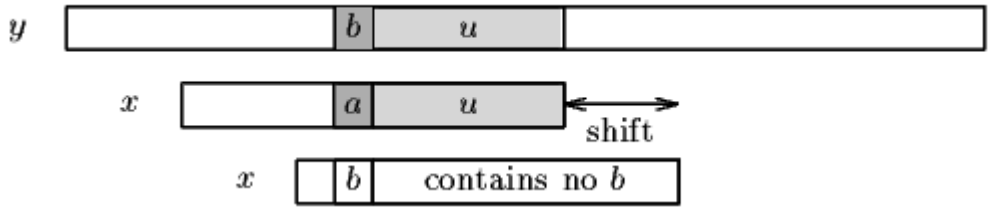
If *u* occurs in *x* only at the end, the shift results from the longest match between suffix *u* as the prefix of pattern *x*.



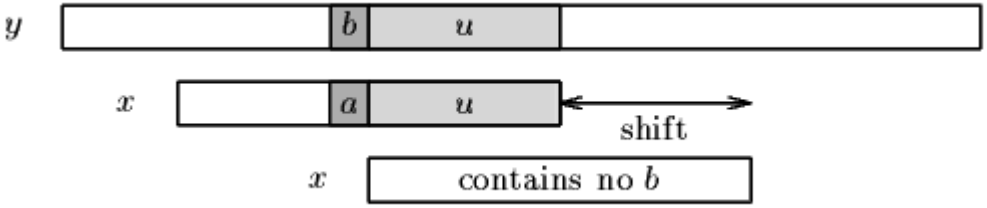
Preparing of these materials was supported by a grant from Liechtenstein, Iceland and Norway through EEA Grants and Norway Grants and co-financed by domestic funds

BM algorithm – bad character

Bad character shift aligns character *b*=y[i+j] with the rightmost occurrence of *b* in *x*[0..m-2].



If *b* does not occur in *x* at all, we shift the beginning of *x* after the occurrence of *b* in *y*, i.e. to the position i+j+1.



Preparing of these materials was supported by a grant from Liechtenstein, Iceland and Norway through EEA Grants and Norway Grants and co-financed by domestic funds

Boyer-Moore algorithm

BM algorithm is considered the most effective matching algorithm in ordinary applications.
Comparisons performed from right to left.
Preprocessing is $\boldsymbol{O}(m+a)$ in time and memory.
Computational complexity of matching $\boldsymbol{O}(mn)$.
Expected number of comparisons in the worst case: $3n$ (for aperiodic pattern).
Best case complexity $\boldsymbol{O}(n/m)$.

Example
Alphabet: a c g t
Pattern: `gcagagag`
Text: `gcatcgcagagagtatacagtacg`

Preparing of these materials was supported by a grant from Liechtenstein, Iceland and Norway through EEA Grants and Norway Grants and co-financed by domestic funds

Boyer-Moore - example

```
Bc:   a c g t                          Gs:   0 1 2 3 4 5 6 7
      1 6 2 8                                7 7 7 2 7 4 7 1

1. gcatcgcagagagtatacagtacg
   .......g                        s=1  (Gs[7]=Bc[a]-8+8)

2. gcatcgcagagagtatacagtacg
   . ....gAG                       s=4  (Gs[5]=Bc[c]-8+6)

3. gcatcgcagagagtatacagtacg
        GCAGAGAG                   s=7  (Gs[0])

4. gcatcGCAGAGAGtatacagtacg
               .....gAG            s=4  (Gs[5]=Bc[c]-8+6)

5. gcatcGCAGAGAGtatacagtacg
                     ......aG      s=7  (Gs[6])
```

5 attempts, 17 character comparisons.

Preparing of these materials was supported by a grant from Liechtenstein, Iceland and Norway through EEA Grants and Norway Grants and co-financed by domestic funds

Edit distance (Levenshtein distance)

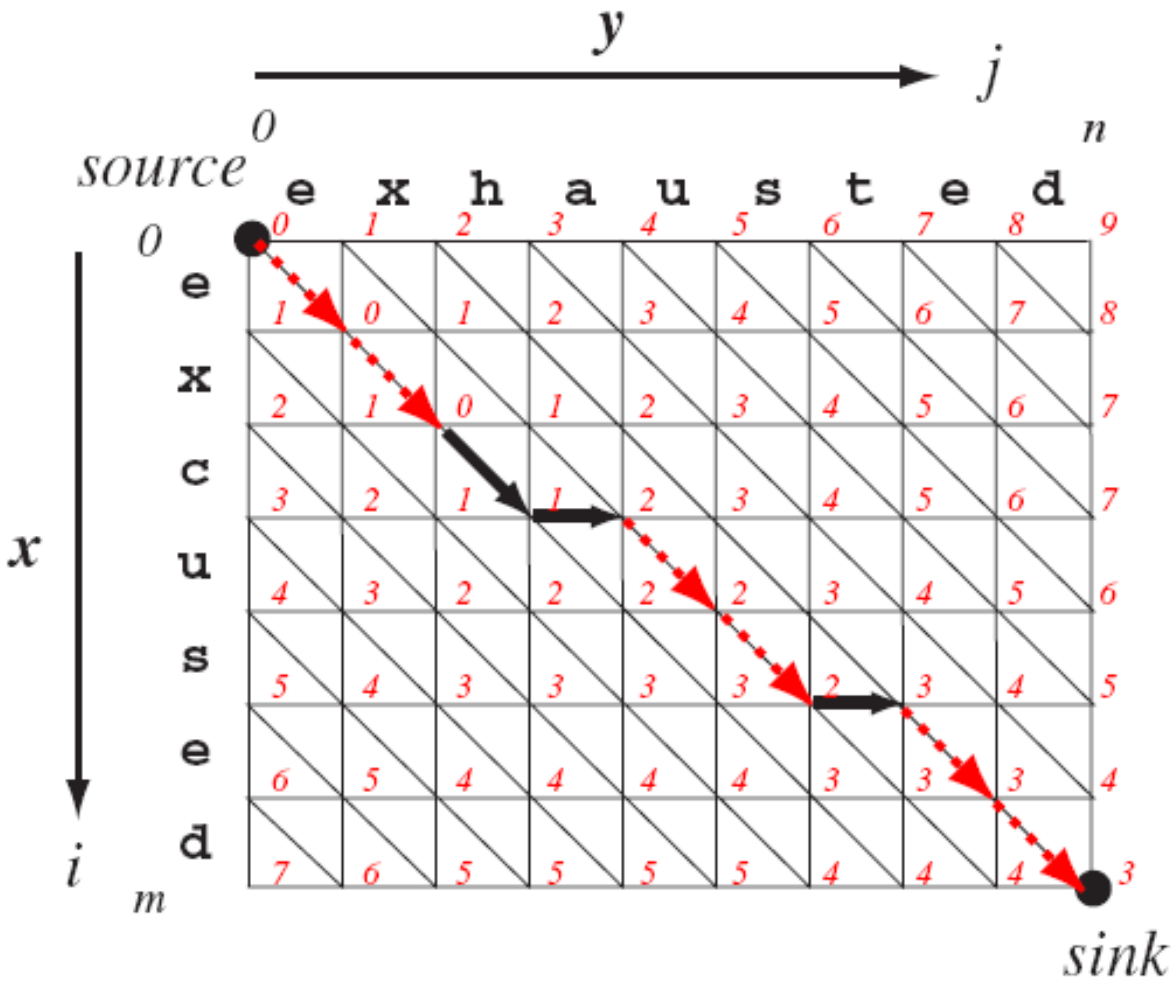
The distance between two strings is simply the number of elementary edit operations transforming string *x* into *y*. These elementary operations are:

- substitution
- deletion
- insertion

```
// initialization C: C[i,0]=i; C[0,j]=j
for (int i=0; i<m; i++)
   for (int j=0; j<n; j++)
      C[i+1, j+1] = min(C[i, j+1] + 1,// insertion
                        C[i+1, j] + 1, // deletion
                        C[i,j] + 1 - x[i] == y[j]);
                                       // subst.?
// C[m, n] is an edit distance
```


Preparing of these materials was supported by a grant from Liechtenstein, Iceland and Norway through EEA Grants and Norway Grants and co-financed by domestic funds

Edit distance



Preparing of these materials was supported by a grant from Liechtenstein, Iceland and Norway through EEA Grants and Norway Grants and co-financed by domestic funds