String matching

Sunday algorithm

Idea

The Boyer-Moore-algorithm uses for its bad-character heuristics the text symbol that has caused a mismatch. The Horspool-algorithm uses the rightmost symbol of the current text window. It was observed by Sunday [Sun 90] that it may be even better to use the symbol directly *right of* the text window, since in any case this symbol is involved in the next possible match of the pattern.

Example:

```
0 1 2 3 4 5 6 7 8 9 ...     0 1 2 3 4 5 6 7 8 9 ...     0 1 2 3 4 5 6 7 8 9 ...
a b c a b d a a c b a       a b c a b d a a c b a       a b c a b d a a c b a
b c a a b                   b c a a b                   b c a a b
  b c a a b                         b c a a b                       b c a a b

(a)  Boyer-Moore            (b)  Horspool               (c)  Sunday
```

In this example, t_0, ..., t_4 = a b c a b is the current text window that is compared with the pattern. Its suffix a b has matched, but the comparison c-a causes a mismatch. The bad-character heuristics of the Boyer-Moore algorithm (a) uses the "bad" text character c to determine the shift distance. The Horspool algorithm (b) uses the rightmost character b of the current text window. The Sunday algorithm (c) uses the character directly right of the text window, namely d in this example. Since d does not occur in the pattern at all, the pattern can be shifted past this position.

Like the Boyer-Moore and the Horspool algorithm, the Sunday algorithm assumes its best case if every time in the first comparison a text symbol is found that does not occur at all in the pattern. Then the algorithm performs just $O(n/m)$ comparisons.

In contrast to the Boyer-Moore and the Horspool algorithm the pattern symbols need not be compared from right to left. They can be compared in an arbitrary order. For instance, this order can depend on the symbol probabilities, provided they are known. Then the least probable symbol in the pattern is compared first, hoping that it does not match, so that the pattern can be shifted

The following example shows the comparisons performed if symbol c of the pattern is compared first..

Example:

```
0 1 2 3 4 5 6 7 8 9 ...
a b c a b d a a c b a
b c a a b
            b c a a b
```

Preprocessing

The occurrence function *occ* required for the bad-character heuristics is computed in the same way as in the Boyer-Moore algorithm.

Given a pattern p, the following function *sundayInitocc* computes the occurrence function; it is identical to the function *bmInitocc*.

```
void sundayInitocc()
{
    int j;
    char a;

    for (a=0; a<alphabetsize; a++)
        occ[a]=-1;

    for (j=0; j<m; j++)
    {
            a=p[j];
            occ[a]=j;
        }
    }
```

Searching algorithm

Using a function *matchesAt* that compares the pattern with the text window in a certain manner depending on the implementation, the searching algorithm looks as follows:

```
void sundaySearch()
{
    int i=0;
    while (i<=n-m)
    {
        if (matchesAt(i)) report(i);
        i+=m;
        if (i<n) i-=occ[t[i]];
    }
}
```

After statement `i+=m`, it is necessary to check if the value of i is at most n-1, since subsequently $t[i]$ is accessed.

References

[Sun 90]	D.M. SUNDAY: A Very Fast Substring Search Algorithm. Communications of the ACM, 33, 8, 132-142 (1990)
[Web 1]	http://www-igm.univ-mlv.fr/~lecroq/string/

Next:

H.W. Lang   Hochschule Flensburg   lang@hs-flensburg.de   Impressum   ©   Created: 31.01.2005   Updated: 29.05.2016