String matching

Horspool algorithm

Idea

The Boyer-Moore algorithm uses two heuristics in order to determine the shift distance of the pattern in case of a mismatch: the bad-character and the good-suffix heuristics. Since the good-suffix heuristics is rather complicated to implement there is a need for a simple algorithm that is based merely on the bad-character heuristics. Due to an idea of Horspool [Hor 80], instead of the "bad character" that caused the mismatch, in each case the rightmost character of the current text window is used for determining the shift distance.

Example:

```
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
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

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 pattern can be shifted until the rightmost occurrence of b in the pattern matches the text character b, where the occurence at the last position of the pattern does not count.

Like the Boyer-Moore algorithm, the Horspool 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.

Preprocessing

The function *occ* required for the bad-character heuristics is computed slightly different as in the Boyer-Moore algorithm. For every alphabet symbol a, the function value *occ*(p, a) is equal to the rightmost position of a in p_0 ... p_{m-2}, or -1, if a does not occur at all. Observe that the last symbol p_{m-1} of the pattern is not taken into account.

Example:

occ(text, x) = 2

occ(text, t) = 0

occ(next, t) = -1

The occurrence function for a certain pattern p is stored in an array *occ* that is indexed by the alphabet symbols. For every symbol $a \in A$ the entry *occ*[a] holds the corresponding function value *occ*(p, a).

Given a pattern p, the following function *horspoolInitocc* computes the occurrence function.

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

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

    for (j=0; j<m-1; j++)
    {
        a=p[j];
        occ[a]=j;
```

```
    }
}
```

Searching algorithm

As in the Boyer-Moore algorithm, the pattern is compared from right to left with the text. After a complete match or in case of a mismatch, the pattern is shifted according to the precomputed function *occ*.

```
void horspoolSearch()
{
    int i=0, j;
    while (i<=n-m)
    {
        j=m-1;
        while (j>=0 && p[j]==t[i+j]) j--;
        if (j<0) report(i);
        i+=m-1;
        i-=occ[t[i]];
    }
}
```

References

[Hor 80] R.N. Horspool: Practical Fast Searching in Strings. Software - Practice and Experience 10, 501-506 (1980)

[Web 1] http://www-igm.univ-mlv.fr/~lecroq/string/

[Web 2] http://www.inf.fh-flensburg.de/lang/algorithmen/pattern/stringmatchingclasses/HorspoolStringMatcher.java
Horspool algorithm as a Java class source file

Next: [Sunday algorithm] or