The University of Newcastle
School of Information and Physical Sciences

COMP2230/COMP6230 Algorithms
Tutorial Week 12 Solutions

18th – 22nd October 2021

1. The following is Rabin-Karp algorithm that searches for an occurrence of a pattern p in a text t. It returns the smallest index i such that $t[i..i+m-1] = p$, or -1 if no such index exists.

```
Input Parameters: p, t
Output Parameters: None
rabin_karp_search(p, t) {
        m = p.length
        n = t.length
        q = prime number larger than m
        r = 2^(m-1) mod q
        // computation of initial remainders
        f[0] = 0
        pfinger = 0
        for j = 0 to m-1 {
                f[0] = 2 * f[0] + t[j] mod q
                pfinger = 2 * pfinger + p[j] mod q
        }
        i = 0
        while (i + m ≤ n) {
                if (f[i] == pfinger)
                        if (t[i..i+m-1] == p) // this comparison takes time O(m)
                                return i
                f[i + 1] = 2 * (f[i]- r * t[i]) + t[i + m] mod q
                i = i + 1
        }
        return -1
}
```

1.1 Trace the algorithm on pattern "101" and text "011010011001110" with $q=2$. How many comparisons between the pattern and text symbols are made?

SOLUTION:

```
m=3, n=15 , r=0
pfinger=1
     f=[1, -, -, -, -, -, -, -, -, -, -, -, -, -, -]
i=0, f=[1, 0, -, -, -, -, -, -, -, -, -, -, -, -, -]
i=1, f=[1, 0, 1, -, -, -, -, -, -, -, -, -, -, -, -]
Index: 2 (total comparisons: 4)
```

1.2 Trace the algorithm on pattern "101" and text "011010011001110" with *q=5*. How many comparisons between the pattern and text symbols are made?

SOLUTION:

```
m=3, n=15 , r=4
pfinger=0
st: f=[3, -, -, -, -, -, -, -, -, -, -, -, -]
i=0, f=[3, 1, -, -, -, -, -, -, -, -, -, -, -]
i=1, f=[3, 1, 0, -, -, -, -, -, -, -, -, -, -]
Index: 2 (total comparisons: 3)
```

1.3 Show that the worst case running time of the algorithm is *O(m(n-m+1))*.

SOLUTION:

In the worst case every substring of the text has the same fingerprint as the pattern, and all but the last character is the same.
In that case we check all characters of the pattern for every character in the text. We do not check substrings at the end of the text that are too small, so we only check the pattern for $-(\quad -1) = -+1$ characters of the text. In total this gives us $\times(\quad -+$ 1) comparisons.

This situation occurs in the case that, for example, we are using the English alphabet with characters having values 0 to 25 (for a to z respectively) , and we are searching for the pattern "aaf" in the text "aaaaaaaaaaaaaaaaa" (for any number of a's) if we use q=5. There are many other similar examples.

2. The following is a pseudocode of the algorithm that computes the shift table for a pattern p, to be used in Knuth–Morris–Pratt(KMP) algorithm; *shift[k]*, *kÎ[-1,…,p.length–1]*, is the smallest positive integer s such that *p[0 . . k-s] = p[s . . k]*.

Input Parameters: *p*
Output Parameters: *shift*

```
knuth_morris_pratt_shift(p,shift)
{ m = p.length
shift[-1]=1 // if p[0] ¹ t[i] we shift by one position
shift[0]=1
i = 1
j = 0
while (i + j < m)
        if (p[i+j] = = p[j]) {
                shift[i+j]=i
                j = j + 1
        }
        else {
                if (j = = 0)
                  shift[i]=i+1
                i = i + shift[j-1]
                j = max(j-shift[j-1],0)
        }
}
```

2.1 Trace the algorithm on pattern “pappar”.

SOLUTION:

```
m=6
i=1, j=0, shift=[1, 1, -, -, -, -, -]
i=2, j=0, shift=[1, 1, 2, -, -, -, -]
i=2, j=1, shift=[1, 1, 2, 2, -, -, -]
i=3, j=0, shift=[1, 1, 2, 2, -, -, -]
i=3, j=1, shift=[1, 1, 2, 2, 3, -, -]
i=3, j=2, shift=[1, 1, 2, 2, 3, 3, -]
i=5, j=0, shift=[1, 1, 2, 2, 3, 3, -]
i=6, j=0, shift=[1, 1, 2, 2, 3, 3, 6]
```

2.2 Trace the algorithm on pattern “ababbcabab”.

SOLUTION:

```
m=10
i=1, j=0, shift=[1, 1, -, -, -, -, -, -, -, -, -]
i=2, j=0, shift=[1, 1, 2, -, -, -, -, -, -, -, -]
i=2, j=1, shift=[1, 1, 2, 2, -, -, -, -, -, -, -]
i=2, j=2, shift=[1, 1, 2, 2, 2, -, -, -, -, -, -]
i=4, j=0, shift=[1, 1, 2, 2, 2, -, -, -, -, -, -]
i=5, j=0, shift=[1, 1, 2, 2, 2, 5, -, -, -, -, -]
```

```
i=6, j=0, shift=[1, 1, 2, 2, 2, 5, 6, -, -, -, -]
i=6, j=1, shift=[1, 1, 2, 2, 2, 5, 6, 6, -, -, -]
i=6, j=2, shift=[1, 1, 2, 2, 2, 5, 6, 6, 6, -, -]
i=6, j=3, shift=[1, 1, 2, 2, 2, 5, 6, 6, 6, 6, -]
i=6, j=4, shift=[1, 1, 2, 2, 2, 5, 6, 6, 6, 6, 6]
```

2.3 Show that *knuth_morris_pratt_shift* algorithm correctly computes the shift array in time *O(m)*.

SOLUTION:

We choose the line "`if (p[i+j] = = p[j])`" as a barometer statement.

Observe that each entry of the `shift[]` array is only written to once, and each time a value is set the value of + is increased by exactly 1—either by i being incremented because = 0, and so `shift[j-1]` = 1 or by j being incremented.
If all the characters of the pattern are the same (e.g., p="aaaaa") the barometer statement will be executed exactly times (once for each 'true' branch of the main if statement, as j increments from 0 to − 1).
If all the characters of the pattern are different (e.g., p="abcde") observe that it will always be the case. The barometer statement will be executed exactly times (once for each 'false'—i.e, else—branch of the main if statement as increases from 1 to).

If neither of the above two cases occur, then it must be the case that there are some repeated sub-patterns in the pattern. Suppose, then, that we have found a substring in the pattern that is the same as the substring at the beginning of the pattern. The value of will be the length of that substring. Let = + (and so `shift[k]` is has not yet been set). In this scenario, it must be the case that the barometer statement was executed exactly times in the previous iterations, as the 'true' branch of the main if statement is executed. This will continue so long as `p[k]==p[j]`.
We consider, then, what happens when `p[k]` ≠ `p[j]`. In this case ≠ 0, so the value of `shift[k]` will not be written. Instead the value of is reduced by `shift[j-1]` (note that the value of is increased by the same value and as such the value of = + will remain unchanged). This will continue until = 0 at which point the value of shift[k] is written. In the worst case will be reduced by exactly 1 each iteration, and so the barometer statement will have been executed exactly more times when the value of `shift[k]` is written, and + is incremented. As such, for the entire substring, the barometer statement is executed exactly 2 times.

Note that if = 0 and `p[i+j]`≠`p[j]` then the barometer statement is executed exactly once for the corresponding setting of the `shift[i]` value. The worst case, then, happens when `p[i+j]==p[j]` every time = 0, and `shift[j-1]==1` every time `p[i+j]`≠`p[j]`. If so the barometer statement is executed exactly 2 times (twice for each character in the pattern) and so is ( ).
This worst case scenario happens, for example, with the pattern "aaaab".