Homework 6

1.
$T = t_1 t_2 t_3 ......t_n$
$P = p_1 p_2 p_3 ......p_m$

Simple solution: $O(nm)$*time*
  − *Try every* t_i *as a starting point*.

Efficient algorithm:

For example:

```
T = x   y  y  x  z  x y  y  x  y  y  x  z
                           \
     ||   ||  || ||    comparison failed here.
                           /
P = x   y   y  x  y y  z
                 x  y y  x y y  z
                    |
                   second comparison starts from here since we know t2 and t3 are not
                   "x".
```

At first, we have to create a "next array" for P to eliminate unnecessary comparisons.

next array
- This array shows how much of the beginning of the string matches up to the portion immediately preceding a failed comparison.

```
Algorithm Next_Array (B, m)
begin:
        next[1] := -1;
        next[2] := 0;
        for  i = 3 to m do
                j := next[i – 1] + 1;
                while B[j] != B[i – 1] and j > 0 do
                        j := next[j] + 1;
                next[i] = j;
        end for
end
```

example:

	1	2	3	4	5	6	7	8	9	10
P =	a	b	a	b	a	b	a	b	c	a
Next:	-1	0	0	1	2	3	4	5	6	0

The numbers in the next array indicate the number of reuse.
P[3] (a) and P[4](b) are the same as the beginning of the string P[1] and P[2] so next[5] = 2.

After creating the next array for string P, we can use the algorithm below to find the longest prefix of P which is a substring of T.

```
Algorithm Longest_Prefix (A, n, B, m)
begin
    i := 1;
    j := 1;
    Start := 0;
    MaxLength := 0;
    while MaxLength < m and i <= n do
        if A[i] = b[j] then
            i := i + 1;
            j := j + 1;
        else
            if(j - 1) > MaxLength then
                MaxLength := j – 1;
                Start := i – j + 1;
            j := next[j] + 1;
            if j = 0 then
                j := 1;
                i := i + 1;
        if j = m + 1 then
            MaxLength := m;
            Start := i – m + 1;
    end while
end
```

“Start” is the position in T which the longest prefix of P(a substring of T) starts at.
“MaxLength” is the length of the longest prefix of P being a substring of T.

To create a next array, it takes O(m) time and to find the longest prefix of string P that matches a substring of T, it takes O(n) time. Therefore, the total time complexity is O(n + m).

2.

At first, append T at the end of T to check if P is a circular substring of T.

For example:

T = abcdefg
becomes
T' = abcdefgabcdefg

Algorithm to find if P is a substring of T'(we use the next array for this algorithm as well).

```
Algorithm Substring (A, n, B, m)
begin
        i := 1;
        j := 1;
        Start := 0;
        while start = 0 and i <= n do
                if A[i] =  B[j] then
                        i := i + 1;
                        j := j + 1;
                else
                        j := next[j] + 1;
                        if  j = 0 then
                                j := 1;
                                i := i + 1;
                If j = m + 1 then
                        Start := i – m;
        end while
end
```

The length of the string T' is 2n and we have to check until the (n + m – 1)th character in T'. Therefore, the total time complexity is O(n + m –1 + m) = O(n + 2m).

Following figures show the computing of the shortest path from C to E step by step.