Lab 6 Report

1. Given a string, calculate next function, print the result out.

(1) Firstly, for a next_array, the first two elements should be -1 and 0.

So I set a list likes: [-1,0,.....,0,0]. I also set 2 int : k and j.

The number j means the location of current element waiting for get a value in the next_array, while the k is used to calculate the that value. While string[j] != string[k], the k = next_array[k] and join the judge again until string[j] == string[k], when next_array[i]=k.

At the end we get a complete next_array.

(2) Code in Python:

```
def get_next_array(the_str):
    j = 0
    k = -1
    next_array = [-1] + [0] * (len(the_str)-1)
    while j < len(the_str) - 1:
        if k == -1 or the_str[j] == the_str[k]:
            j += 1
            k += 1
            next_array[j] = k
        else:
            k = next_array[k]
    return next_array
```

(3) Test:

```
print(get_next_array("abaabcac"))
```

```
[-1, 0, 0, 1, 1, 2, 0, 1]
```

2. Implement KMP algorithm

(1) Since we have gotten the next_array, while aim_str[i] != model_str[j], I just change the j to next_array[j] until aim_str[i] == model_str[j]. While the j == length of model_str, a copy of model_str in aim_str is found and change the j to (j-1).

(2) Code in Python:

```
def kmp(aim_str, model_str):
    aim_len = len(aim_str)
    model_len = len(model_str)
    re = []

    if aim_len >= model_len:
        i = 0
        j = 0
        next_array = get_next_array(model_str)
        while i < aim_len:
            if j == -1 or aim_str[i] == model_str[j]:
                i += 1
                j += 1
            else:
                j = next_array[j]
            if j == model_len:
                re.append(i - model_len)
                # i = i - model_len + 1
                # j = 0
                # this kind of change also work
                j = j - 1

    return ["time: " + str(len(re))] + re
```

(3) Test:

```
print(kmp("AAATTTAAATTTAAATTTAAATTTAA","TTT"))
```

```
['time: 4', 3, 9, 15, 21]
```

3. Given two strings S1, S2, find out how many time S2 appears in S1.

Print out corresponding location when a search is successful.

S1: ATCGCCGCGATCGTATTTAATCGATTGCATCG

S2: ATCG

Result: ATCGCCGCGATCGTATTTAATCGATTGCATCG

Test:

```
print(kmp("ATCGCCGCGATCGTATTTAATCGATTGCATCG","ATCG"))
```

Output:

```
['time: 4', 0, 9, 19, 28]
```