CSIT5500 Advanced Algorithm HW2

LIN Jialiang 20656855

Question 1

Computing the next table:

1. Initialize the next(0)=-1

0	1	2	3	4	5	6	7	8	9	10	11	12
c	g	t	a	c	g	t	t	c	g	t	a	c
-1												

2. The procedure:

k	i	comparison	k
-1	1	p[0] p[1] no	-1
-1	2	p[0] p[2] no	-1
-1	3	p[0] p[3] no	-1
-1	4	p[0] p[4] yes	0
0	5	p[1] p[5] yes	1
1	6	p[2] p[6] yes	2
2	7	p[3] p[7] no	next(2) = -1
-1	7	p[0] p[7] no	-1
-1	8	p[0] p[8] yes	0
0	9	p[1] p[9] yes	1
1	10	p[2] p[10] yes	2
2	11	p[3] p[11] yes	3
3	12	p[4] p[12] yes	4

Therefore, the next table should be:

0	1	2	3	4	5	6	7	8	9	10	11	12
c	g	t	a	c	g	t	t	c	g	t	a	c
-1	-1	-1	-1	0	1	2	-1	0	1	2	3	4

Question 2

Constructing the suffix array:

Here I assume $ replacing with the null characters, and **$ is smaller than any other symbols thus having -1 order**.

J=0:

index	symbol
0	m
1	i
2	n
3	i
4	m
5	i
6	z
7	e

sort

index	symbol	rank
7	e	0
1	i	1
3	i	1
5	i	1
0	m	4
4	m	4
2	n	6
6	z	7

Rank array:

index	rank
0	4
1	1
2	6
3	1
4	4
5	1
6	7
7	0

J=1:

index	symbol	order
0	mi	4,1
1	in	1,6
2	ni	6,1
3	im	1,4
4	mi	4,1
5	iz	1,7
6	ze	7,0
7	e$	0,-1

sort

index	order	rank
7	0,-1	0
3	1,4	1
1	1,6	2
5	1,7	3
0	4,1	4
4	4,1	4
2	6,1	6
6	7,0	7

Rank array:

index	rank
0	4
1	2
2	6
3	1
4	4
5	3
6	7
7	0

J=2:

index	symbol	order
0	mini	4,6
1	inim	2,1
2	nimi	6,4
3	imiz	1,3
4	mize	4,7
5	ize$	3,0
6	ze$$	7,-1
7	e$$$	0,-1

sort

index	order	rank
7	0,-1	0
3	1,3	1
1	2,1	2
5	3,0	3
0	4,6	4
4	4,7	5
2	6,4	6
6	7,-1	7

Rank array:

index	rank
0	4
1	2
2	6
3	1
4	5
5	3
6	7
7	0

J=3:

Rank array:

index	symbol	order
0	minimize	4,5
1	inimize$	2,3
2	nimize$$	6,7
3	imize$$$	1,0
4	mize$$$$	5,-1
5	ize$$$$$	3,-1
6	ze$$$$$$	7,-1
7	e$$$$$$$	0,-1

sort

index	order	rank
7	0,-1	0
3	1,0	1
1	2,3	2
5	3,-1	3
0	4,5	4
4	5,-1	5
2	6,7	6
6	7,-1	7

index	rank
0	4
1	2
2	6
3	1
4	5
5	3
6	7
7	0

Another way to solve the problem is to **treat $ as rank 0, so the rank of other symbols beginning from 1**. Here is the process: (basically adding 1 to the above process)

J=0:

Rank array:

index	symbol
0	m
1	i
2	n
3	i
4	m
5	i
6	z
7	e

sort

index	symbol	rank
	$	0
7	e	1
1	i	2
3	i	2
5	i	2
0	m	5
4	m	5
2	n	7
6	z	8

index	rank
0	5
1	2
2	7
3	2
4	5
5	2
6	8
7	1

J=1:

Rank array:

index	symbol	order
0	mi	5,2
1	in	2,7
2	ni	7,2
3	im	2,5
4	mi	5,2
5	iz	2,8
6	ze	8,1
7	e$	1,0

index	rank
0	5
1	3
2	7
3	2
4	5
5	4
6	8
7	1

J=2:

Rank array:

index	symbol	order
0	mini	5,7
1	inim	3,2
2	nimi	7,5
3	imiz	2,4
4	mize	5,8
5	ize$	4,1
6	ze$$	8,0
7	e$$$	1,0

⇒

index	rank
0	5
1	3
2	7
3	2
4	6
5	4
6	8
7	1

J=3:

Rank array:

index	symbol	order
0	minimize	5,6
1	inimize$	3,4
2	nimize$$	7,8
3	imize$$$	2,1
4	mize$$$$	6,0
5	ize$$$$$	4,0
6	ze$$$$$$	8,0
7	e$$$$$$$	1,0

⇒

index	rank
0	5
1	3
2	7
3	2
4	6
5	4
6	8
7	1

Question 3

According to the problem description, I have the following definition:

s[m⋯n] (string): the substring beginning from index m to index n, where the whole string begins from index 1, aka, s[1⋯n].

V[i] (boolean value): whether substring from index=1 to index=i, aka s[1⋯i], can be reconstituted as a sequence of valid words.

- Boundary condition:

$$V[0] = 1$$

Which means I treat null characters as valid words and it would help the algorithm.

- Recurrence relation:

$$V[i] = \sum_{k\in[0,i-1]} V[k] \cdot dict(s[k+1 \dots i])$$

Here, the summation and multiply operation are all Boolean operation.

The recurrence relationship indicates that: (Consider V[i])

1. If V[i-1]==1 and dict(s[i⋯i])==1: then V[i]=1; which means the substring s[1⋯i-1] can be reconstituted as a sequence of valid words, and s[i] is also a valid word through calling dict() function, thus substring s[1⋯i] can be reconstituted as well. Both conditions are supposed to be satisfied.
2. Else I look for V[i-2], V[i-3], ⋯, until V[0], one of the above cases satisfy the two requirements, then ubstring s[1⋯i] can be reconstituted, and V[i] should be 1, aka, true.

The whole dynamic programming is as followed:

```
function DP(s):
V[0] = 1;
for i = 1:len(s)
  V[i] = 0;
  for k = 0:i-1
    if V[k] && dict(s.substring[k+1,i]): V[i]=1;break;
return V[len(s)]
```

According to the problem, the dict() function takes unit time. Therefore, two for-loop take O(n*n) time. The total running time of above dynamic programming algorithm is $O(n^2)$.