

CSC 226 SUMMER 2019 - SOLUTION
ALGORITHMS AND DATA STRUCTURES II
ASSIGNMENT 4 - WRITTEN
UNIVERSITY OF VICTORIA

1. The table below contains the lengths of the longest common subsequences:

LLCS[][]		L	U	L	L	A	B	Y	B	A	B	I	E	S
	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0	0	0	0
U	0	0	1	1	1	1	1	1	1	1	1	1	1	1
L	0	1	1	2	2	2	2	2	2	2	2	2	2	2
L	0	1	1	2	3	3	3	3	3	3	3	3	3	3
A	0	1	1	2	3	4	4	4	4	4	4	4	4	4
N	0	1	1	2	3	4	4	4	4	4	4	4	4	4
D	0	1	1	2	3	4	4	4	4	4	4	4	4	4
B	0	1	1	2	3	4	5	5	5	5	5	5	5	5
O	0	1	1	2	3	4	5	5	5	5	5	5	5	5
N	0	1	1	2	3	4	5	5	5	5	5	5	5	5
E	0	1	1	2	3	4	5	5	5	5	5	5	6	6
S	0	1	1	2	3	4	5	5	5	5	5	5	6	7

So, the longest common subsequence is “ullabes”.

2. What follows is one way of doing it. Most will strictly use the entries in the llcs[][] and that is okay.

Algorithm printLCS(Array llcs[], String x, String y)

Input: Array of lengths of the longest common subsequences and the two strings

Output: String s, the lcs of x and y

```

n ← x.length()
m ← y.length()
String s

k ← m*n
i ← n
j ← m
while (i > 0 && j > 0) do {
    if x[i] = y[j] then
        s[k] = x[i]
        i ← i-1
        j ← j-1
        k ← k-1
    else if llcs[i-1][j] > llcs[i][j-1] then
        i ← i-1
    else
        j ← j-1
}
print(s)

```

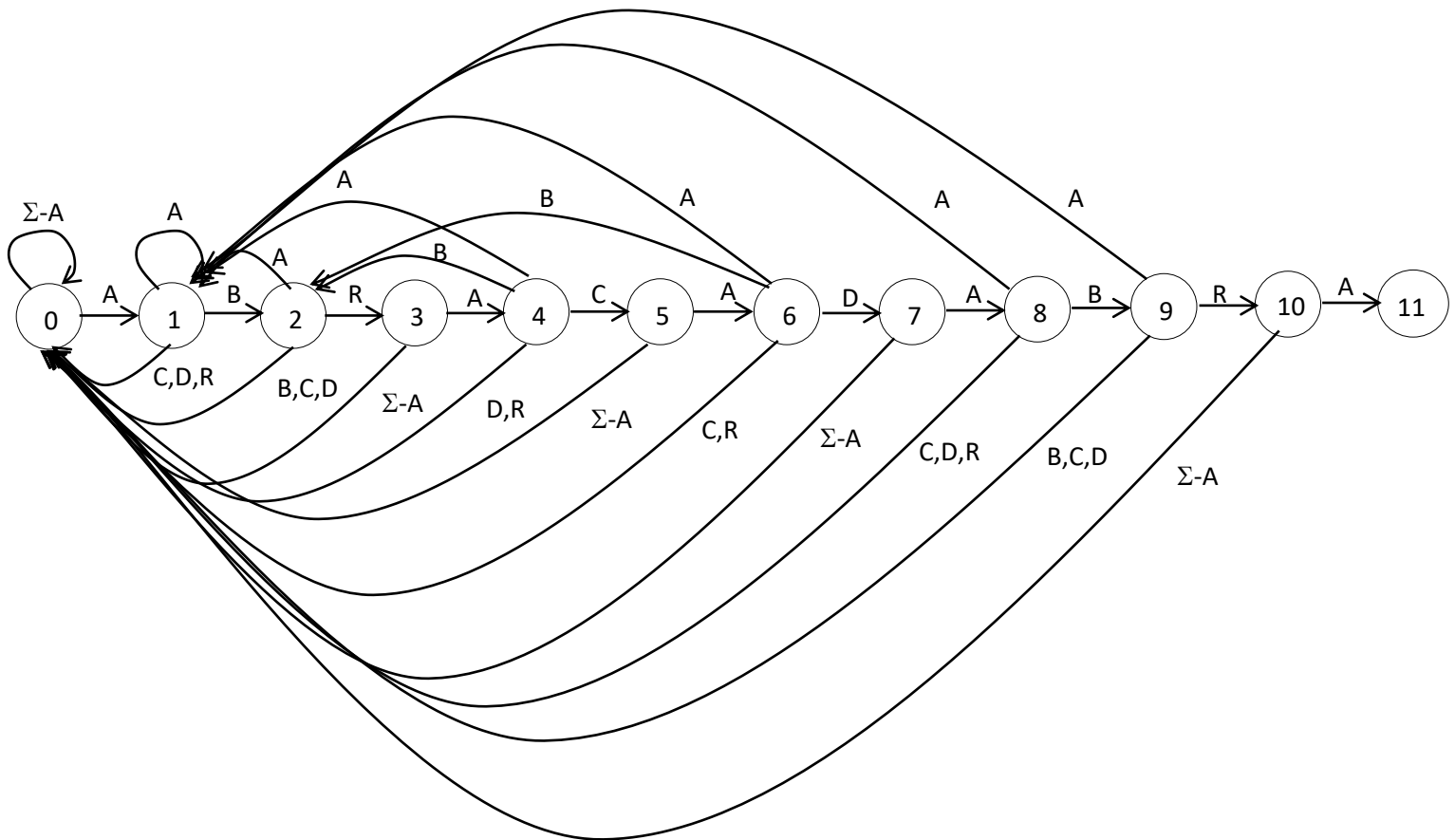
3. The final array is as follows:

right[]	10	8	4	6	9
	A	B	C	D	R

4. First the array

	j	0	1	2	3	4	5	6	7	8	9	10
pat.charAt(j)		A	B	R	A	C	A	D	A	B	R	A
dfa[][j]	A	1	1	1	4	1	6	1	8	1	1	11
	B	0	2	0	0	2	0	2	0	9	0	0
	C	0	0	0	0	5	0	0	0	0	0	0
	D	0	0	0	0	0	0	7	0	0	0	0
	R	0	0	3	0	0	0	0	0	0	10	0

Transition diagram – Note $\Sigma-A = B,C,D,R$



5. There seems to be two ways to do this, either they ignore the wildcard when hashing, only accounting for its position or they calculate two hash values, left and right of the wildcard. In both cases there will be some adjustments to the Rabin-Karp algorithm for calculating the rolling hash values.

They will also have to account for when the wildcard is in position 0 or M-1 as special cases.