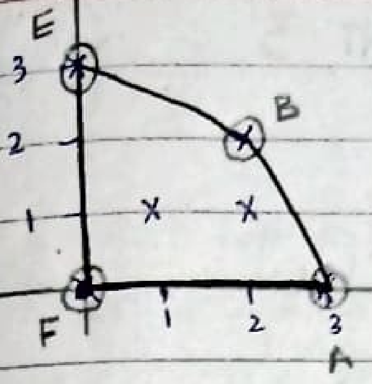
23K-0066
BS(AI)-5A

DAA ASSIGNMENT 3

Q1:

Step	Current point p	Candidate (q)	checking point (r)	orientation (p,q,r)	More CCW?	New Candida
1	F(0,0)	A(0,3)	B(2,2)	(0-0)(2-0) - (3-0)(2-0) = ~~6~~ -6	No	-
2	F(0,0)	A(0,3)	C(1,1)	(0-0)(1-0) - (3-0)(1-1) = 0	No	-
3	F(0,0)	A(0,3)	D(2,1)	(0-0)(1-0) - (3-0)(2-0) = -6	No	-
4	F(0,0)	A(0,3)	E(3,0)	(0-0)(0-0) - (3-0)(3-0) = -9	No	-
5	A(0,3)	B(2,2)	C(1,1)	(2-0)(1-3) - (2-3)(1-0) = -3	No	-
6	A(0,3)	B(2,2)	D(2,1)	(2-0)(1-3) - (2-3)(2-0) = -2	No	-
7	A(0,3)	B(2,2)	E(3,0)	(2-0)(0-3) - (2-3)(3-0) = -3	No	-
8	A(0,3)	B(2,2)	F(0,0)	(2-0)(0-3) - (2-3)(0-0) = -6	No	-
9	B(2,2)	C(1,1)	D(2,1)	(1-2)(1-2) - (1-2)(2-2) = 2	Yes	D
10	B(2,2)	D(2,1)	E(3,0)	(2-2)(0-2) - (1-2)(3-2) = 1	Yes	E
11	B(2,2)	E(3,0)	F(0,0)	(3-2)(0-2) - (0-2)(0-2) = -6	No	-
12	B(2,2)	E(3,0)	A(0,3)	(3-2)(3-2) - (0-2)(0-2) = -3	No	-
13	B(2,2)	E(3,0)	C(1,1)	(3-2)(1-2) - (0-2)(1-2) = -3	No	-
14	E(3,0)	F(0,0)	A(0,3)	= (0-3)(3-0) - (0-0)(0-3) = -9	No	-
15	E(3,0)	F(0,0)	B(2,2)	(0-3)(2-0) - (0-0)(2-3) = -6	No	-
16	E(3,0)	F(0,0)	C(1,1)	(0-3)(1-0) - (0-0)(1-3) = -3	No	-
17	E(3,0)	F(0,0)	D(2,1)	(0-3)(1-0) - (0-0)(2-3) = -3	No	-

$F \rightarrow A \rightarrow B \rightarrow E \rightarrow F$



F → A → B → E → F

Q2)

Brute force :

```
bruteForceFF(P):
    m = length(P)
    F = array of size m
    for j=0 to m-1:
        F[j] = 0
        for k = j down to 1:
            match = True
            for i=0 to k-1:
                if p[i] != p[j-k+1+i]:
                    match = False
                    break
            if match:
                F[j] = k
                break
    return F
```

P = ababaca

j	Substring P[0..j]	longest proper prefix = sofix	F[j]
0	a	—	0
1	ab	—	0
2	aba	a	1
3	abab	ab	2
4	ababa	aba	3
5	ababac	—	0
6	ababaca	a	1

Result: F = [0, 0, 1, 2, 3, 0, 1]

Time Complexity:

outer loop = $O(m)$

middle loop = $O(m)$ in worst case

inner loop = $O(m)$ in worst case

Total: $O(m^3)$ worst case

Optimized KMP:

```
Compute Failure KMP (P):
    m = length (P)
    F = array of size m
    F[0] = 0
    l = 0
```

```
For j=1 to m-1 :
    while i > 0 and P[i] != P[j] :
        i = F[i-1]
    if P[i] == P[j] :
        i = i+1
    else :
        i = 0
    F[j] = i
return F
```

We compute F[j] using previously computed F values. We maintain a pointer i which is the length of the current longest prefix which is also a suffix for P[0...j-1]. For next j, we try to extend by comparing P[i] with P[j]. If they match, F[j] = i+1. If not we set i = F[i-1] and repeat until match or i = 0

Time Complexity :
The while loop runs at most O(m) total across all j because i increases at most by 1 per step, and decreases only via i = F[i-1] which is less than i.
So TC = O(m)

j	P[j]	comparison	i_{before}	i_{after}	F[j]
0	-	-	0	-	0
1	~~P~~ b	P[0] != P[1]	0	0	0
2	a	P[0] == P[2]	0	1	1
3	b	P[1] == P[3]	1	2	2
4	a	P[2] == P[4]	2	3	3
5	c	P[3] != P[5]	3	1	
~~6~~		P[1] != P[5]	1	0	
		P[0] != P[5]	0	0	0
6	a	P[0] == P[5]	0	1	1

F = [0, 0, 1, 2, 3, 0, 1]

Comparison:

	Brute - Force	Optimized KMP
Time complexity	$O(m^3)$	$O(m)$
Character comparisons	Upto $O(m^3)$	$O(m)$
Reuse of computed Info	No reuse	Reuses F values via backtracking
Pratical efficiency	Slow for long patterns	Fast, used in real KMP matcher

Q3)

(a)

S - {1, 5, 6, 8}   desired change is 13.

dp[i] → no. of ways to make amount i

dp[0] = 1 → base case (one way: Use no coins)

For each coin c update:

dp[i] += dp[i - c]   for i = c to amount

coins / w	0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
5	1	1	1	1	1	2	$\overset{1+1}{2}$	$\overset{1+1}{2}$	$\overset{1+1}{2}$	$\overset{1+1}{2}$	$\overset{1+2}{3}$	$\overset{1+2}{3}$	$\overset{1+2}{3}$	$\overset{1+2}{3}$
6	1	1	1	1	1	2	$\overset{2+1}{3}$	$\overset{2+1}{3}$	$\overset{2+1}{3}$	$\overset{2+1}{3}$	$\overset{3+1}{4}$	$\overset{3+2}{5}$	$\overset{3+3}{6}$	$\overset{3+3}{6}$
8	1	1	1	1	1	2	3	3	$\overset{3+1}{4}$	$\overset{3+1}{4}$	$\overset{4+1}{5}$	$\overset{5+1}{6}$	$\overset{6+1}{7}$	$\overset{6+2}{8}$

Final dp[13] = 8

There are 8 combinations to make amount 13 with coins {1, 5, 6, 8}

(b)

str1 = KITTEN   str2 = SITTING

dp[i][j] → min operations to convert first i chars of str1 to first j chars of str2

dp[i][0] = i → delete all i chars
dp[0][j] = j   (Insert j chars)   } Base cases

```
if str1[i-1] == str2[j-1]:
    dp[i][j] = dp[i-][j-1]
else:
    dp[i][j] = 1 + min(dp[i-1][j], dp[i][j-1],
                       dp[i-1][j-1])
                       (delete, insert, substitute)
```

	0	S	I	T	T	I	N	G
0	0	1	2	3	4	5	6	7
K	1	1	2	3	4	5	6	7
I	2	2	1	2	3	4	5	6
T	3	3	2	1	2	3	4	5
T	4	4	3	2	1	2	3	4
E	5	5	4	3	2	2	3	4
N	6	6	5	4	3	3	2	3

~~Final~~ [illegible]

Edit distance = 3

Operations:

1) K → S (substitute)
2) E → I (substitute)
3) Insert G at end

(c)

length [ ] = {1, 2, 3, 4, 5, 6, 7, 8}
price [ ] = {1, 5, 8, 9, 10, 16, 18, 20}
Rod length : 8

dp [L] = max value for rod length L
dp [L] = max (price [i] + dp[L-i-1]) for i = 0 to L-1
or simpler : dp [L] = max (dp[L], price [i] + dp [L-i])
with 1- based indexing

dp [0] = 0
~~dp~~ L = 1 : max (1) = 1
L = 2 : max (5, 1+dp[1] = 1+1 = 2) = 5
L = 3 : max (8, ~~1~~ + dp [2] = 1+5, 6 , 5+dp[1] = 5+1 = 6)
= 8

L = 4 : max (9, 1+dp[3] = 1+8 = 9 , 5+dp [2] = 5+5 = 10 ,
8 + dp[1] = 8+1 = 9) = 10

L = 5 : max (10, 1+dp [4] = 1+10 = 11, 5+dp[3] = 5+8 = 13,
8+ dp [2] = 8+5 = 13, 9+dp [1] = 9+1 = 10) = 13

L = 6 : max (16, 1+dp [5] = 1+13 = 14 , 5+dp [4] = 5+10 = 15,
8+dp[3] = 8+8 = 16 , 9+dp [2] = 9+5 = 14,
10+ dp [1] = 10+1 = 11) = 16

L = 7 : max (18, 1+dp [6] = 1+16 = 17, 5+dp [5] = 5+13 = 18,
8+ dp [4] = 8+10 = 18, 9+ dp [3] = 9+8 = 17,
10+ dp [2] = 10+5 = 15 , 16 + dp [1] = 16+1 = 17)
= 18

L = 8 : max (20, 1+dp[7] = 1+18 = 19,
5 + dp[6] = 5+16 = 21,
8 + dp[5] = 8+13 = 21,
9 + dp[4] = 9+10 = 19,
10 + dp[3] = 10+8 = 18,
16 + dp[2] = 16+5 = 21,
18 + dp[1] = 18+1 = 19)

L = 8 : 21

Rod length L	Max value dp[L]	Optimal cuts
0	0	-
1	1	1
2	5	2
3	8	3
4	10	2+2
5	13	2+3
6	16	6
7	18	2+5 or 1+6
8	21	2+6 or 3+5

Maximum value = 21
Optimal cuts : (2,6) or (3,5) or (5,3) or (6,2)
all give 5+16 = 21 or 8+13 = 21

(d)

word Dict = {i, Like, ice, cream, icecream, mobile, apple}

Input: iLikeapple

```
dp[i] = true if s[0...i-1] can be segmented
dp[0] = true (empty string) → base case
For i=1 to n:
    For j=0 to i-1:
        if dp[j] is true and s[j..i-1] in dict,
        then dp[i] = true
```

dp[0] = true

i=1: s[0..0] = "i" in dict → dp[1] = true

i=2: check "iL" not in dict, dp[1] && "l" no → False

i=3: "iLi" no, dp[1] && "Li" no, dp[2] && "i" no → False

i=4: "iLik" no, dp[1] && "Lik" no, dp[2] && "ik" no,
 dp[3] && "k" no → false

i=5: "iLike" no, dp[1] && "Like" yes → dp[5] = true

i=6: "iLikea" no, dp[5] && "a" no → false

i=7: "iLikeap" no, dp[5] && "ap" no → false

i=8: "iLikeapp" no, dp[5] && "app" no → false

i=9: "iLikeappl" no, dp[5] && "appl" no → false

i=10: "iLikeapple" no, dp[5] && "apple" yes → dp[10] = True

True, string can be segmented as "i like apple"

Q4)

Course selection to maximize credits within study hours limit.

Mapping this problem to 0/1 knapsack:

study hours	=	weight
credits	=	value
each cource	=	item (include or exclude)

dp[h] = max credits acheivable with exactly h hours

```
Loop over courses: for h from max_hours down to course_hours:
                       dp[h] = max(dp[h], dp[h-hours] + credits)
```

1	credits = 4	hours = 5	
2	credits = 3	hours = 4	max hours = 10
3	credits = 5	hours = 7	

hours

Courses	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	4	4	4	4	4	4
2	0	0	0	0	3	4	4	4	7	7	7
3	0	0	0	0	3	4	4	5	7	7	8

maximum credits = 7

optimal selection : Course 1 (4 credits, 5 hours) +
Course 2 (3 credits, 4 hours)

Total: 7 credits, 9 hours