Syeda Rija Ali
(23K-0057).

DAA Assignment 3.

Date: 10/11/25

Q1) 1- (0-0)(2-0) - (3-0)(2-0) = -6

2- (0-0)(1-0) - (3-0)(1-0) = -3

3- (0-0)(1-0) - (3-0)(2-0) = -6

4- (0-0)(0-0) - (3-0)(3-0) = -9

5- (0-0)(2-3) - (0-3)(2-0) = 6

6- (2-0)(1-3) - (2-3)(1-0) = -3

7- (2-0)(1-3) - (2-3)(2-0) = -2

8- (2-0)(0-3) - (2-3)(3-0) = -3

9- (1-2)(3-2) - (1-2)(0-2) = -3

10- (1-2)(1-2) - (1-2)(2-2) = 1

11- (2-2)(3-2) - (1-2)(0-2) = -2

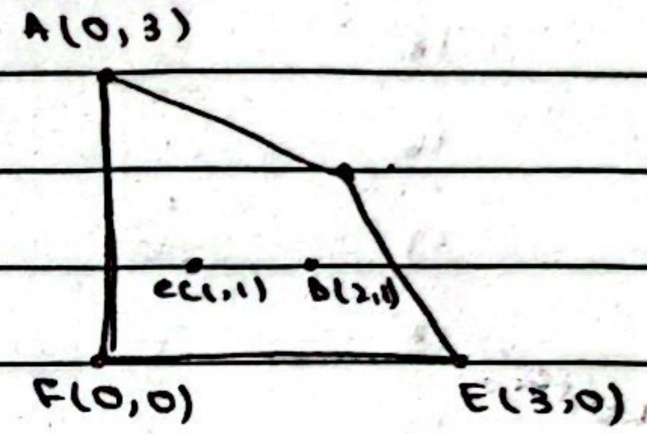
12- (2-2)(0-2) - (1-2)(3-2) = 1

13- (3-2)(3-2) - (0-2)(1-2) = -3

14- (3-2)(1-2) - (0-2)(1-2) = -3

15- (0-3)(2-0) - (3-0)(2-3) = -3

16- (0-3)(1-0) - (3-0)(1-3) = 3



Step	P	Q	R	Orientation	CCW	Newcandidate
1	F	A	C	-6	N	A
2	F	A	D	-3	N	A
3	F	A	E	-6	N	A
4	F	A	C	-9	N	A
5	A	B	C	6	Y	B
6	A	B	D	-3	N	B
7	A	B	E	-2	N	B
8	A	B	F	-3	N	B
9	B	C	A	1	Y	D

Date: _____

Step	P	Q	R	Orientation	CCW	New candidate
10	B	C	D	-2	N	D
11	B	D	A	1	Y	E
12	B	D	E	-3	N	E
13	B	E	A	-3	N	E
14	B	E	C	-3	N	E
15	E	A	B	3	Y	A
16	E	A	C	-1	N	C
17	E	C	D	-3	N	C
18	E	C	A	-3	N	C
19	E	C	F	3	N	F

→ FEBA Ans

F(0,0) → E(3,0) → B(2,2) → A(0,3) → F(0,0).

Q2 i) Brute force Algorithm.

```
for j ← 0 to m-1 do
    F[j] ← 0
    substring ← P[0....j]
    for k ← n-1 down to 1 do
        prefix ← substring [0.--k-1]
        suffix ← substring [n-k .... n-1]
        if prefix = suffix then
            f[j] ← k
            break
            F[j] ← k
            break
        End if
    End for
End for
Return F
```

MIGHTY PAPER PRODUCT

Date: _____

Time Complexity:

- for each j [1 .... n-1] we compare up to j prefix-suffix pairs.
- Each comparison may take $O(j)$ time

$\therefore O(n^2)$.

2. ~~Dmp~~ Optimized KMP Algo:

```
f[0] = 0
k = 0
for j-1 to m-1:
while k>0 ? P[k] != P[j]
  k = f[k-1]
if P[k] = P[j]
  k++
  f[j] = k
return f
```

→ Each character is processed once; even during mismatches, j only decreases (never fully restart).

$\therefore O(n)$.

d2) 3) Brute force Table:

j	substring	Pre = Suff	f[j]
0	a	-	0
1	ab	-	0
2	aba	a	1
3	abab	ab	2
4	ababa	aba	3
5	ababac	-	0
6	abababca	a	1

Date: _____

- KMP Table

i	P[i]	j before	j after	f[i]
0	a	-	-	0
1	b	P[0] ≠ b	0	0
2	a	P[0] = a	1	1
3	b	P[1] = b	2	2
4	a	P[2] = a	3	3
5	c	P[3] = a	0	0
6	a	P[0] = a	1	1

→ Brute force Result → f = [0, 0, 1, 2, 3, 0, 1]

KMP Result ⟶ f = [0, 0, 1, 2, 3, 0, 1]

Both gives same answer.

4) Time Complexity: The brute force algorithm has a high time complexity because it checks all pairs / prefix / suffix combinations for each position in pattern. Leads to repeated comparison & can go upto $O(n^2)$ time. In contrast, KMP algorithm which is optimised avoids repetition by reusing earlier results ; getting $O(n)$ time

Number of Character Comparisons:

Brute-force repeatedly compares the same characters, causing many redundant checks. KMP compares each character at most twice, greatly reducing comparisons

Reuse of computed info :-

Brute force recomputes everything from scratch. KMP reuses the previously calculated failure values to skip unnecessary work

Date: 1

Q3 a) S = {1, 5, 6, 8}

desired change = 13

each coin can be used unlimited times.

	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	2	2	2	2	2	3	3	3	3
{1,5,6}	1	1	1	1	2	3	3	3	3	4	5	6	6
{1,5,6,8}	1	1	1	1	2	3	3	4	4	5	6	7	(8)

↳ 8 combinations to achieve 13.

i	using coin
1	13(1)
2	1(5) + 8(1)
3	2(5) + 3(1)
4	1(6) + 7(1)
5	1(6) + 1(5) + 2(1)
6	2(6) + 1(1)
7	1(8) + 1(5)
8	1(8) + 5(1)

b) Str 1 = kitten ⇒ Rows

Str 2 = sitting ⇒ columns

Date: _____

		S	T	T	T	I	N	G
	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

Position	str1	str2	Action
1	K	S	subs K → S
2	I	I	Match
3	T	T	Match
4	T	T	Match
5	E	I	Sub E → I
6	N	N	Match
7	-	G	Insert G at the end.

↳ 3 operations from which 2 substitutions (K→S, E→I) and one insertion (G).

Q3 c)

Length (L)	1	2	3	4	5	6	7	8
Price (P)	1	5	8	9	10	16	18	20

Date: _____

	0	1	5	8	9	10	16	18	20
0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	1
2	0	1	5	5	5	5	5	5	5
3	0	1	5	8	8	8	8	8	8
4	0	1	5	8	9	9	9	9	9
5	0	1	5	8	9	13	13	13	13
6	0	1	5	8	9	13	16	16	16
7	0	1	5	8	9	13	16	18	18
8	0	1	5	8	9	13	16	18	21

↳ The combination that yields, max amount using 0/1 knapsack we figured out that it is 21 through the combination at $2,6] which gives $5,16] respectively.

d3 d) Input s= 'ilikeapple'

WordDict = {'i', 'like', 'ice', 'cream', 'icecream', 'mobile', 'apple'}.

Algorithm:

1) Initialize consegment[0] = true (empty str can segment).
2) for each index i from 1 → len(s).

check all possible previous splits j < i

o if canSegment[j] == true & s[j,i] is in the dictionary then mark canSegment[i] = true.

3) Find answer → canSegment[len(s)].

Date: _____

Index (i)	Substring s[0:i]	can segment?	How it is formed.
0	''	✓	Base case
1	'i'	✓	'i' ∈ dict
2	'il'	×	no match
3	'ili'	×	no match
4	'ilik'	×	no match
5	'ilike'	✓	'i' (true) + 'like' (dict)
6	'ilikea'	×	no valid split
7	'ilikeap'	×	no valid split
8	'ilikeapp'	×	no valid split
9	'ilikeappl'	×	no valid split
10	'ilikeapple'	✓	'ilike' (true) + 'apple' (dict)

can segment[10] = true, so thus 'ilikeapple' can be segmented into 'i like apple'.

d4) Monthly budget planning problem.

↳ A person has a fixed monthly budget (like 30K pkr) and a list of possible expenses or items they can spend on (groceries, rent, utilities, outings, gadgets etc.)

Each expense provides certain benefits / satisfaction (like importance or priority level) & costs a certain amount of money.

Since budget is set, goal is to maximize total satisfaction while keeping the total cost within the budget.

Date: _____

Example Data:

Expense	Cost (PKR)	satisfaction Value
Rent	25K	10
Groceries	10K	6
Utilities	8K	5
Outings	7K	4
New Headphone	12K	7

Total budget = 30K PKR.

↳ The problem directly maps onto 0/1 Knapsack.

	0	Rent (25K,10)	Groceries (10K,6)	Utilities (8K,5)	Outings (7K,4)	Headphones (12K,7)
0	0	0	0	0	0	0
7K	0	0	0	0	4	4
10K	0	0	(6)	6	6	6
15K	0	0	6	6	0	9
20K	0	0	6	11	11	(12)
25K	0	10	10	11	11	13
30K	0	10	10	11	15	(18)

Max Satisfaction = 18

chosen items = Groceries (10K, 6) + Utilities (8K, 5) + Headphones (12K, 7)

⇒ 30K & 18 satisfaction.

The approach ensures (Dynamic Programming):-

- never exceed total budget.
- Always choose combination of expense with the highest total satisfaction
- each expense is only chosen 0/1 knapsack.