Assignment #3

Design & Analysis of Algorithms

BAI 5-A

23k-0055

SYED UKKASHAH AHMED SHAH

Q1)

Step	Current point (p)	Candidate (q)	Checking point (r)	orientation (p,q,r)	More CCW?	New candidate
1	F(0,0)	A(0,3)	B(2,2)	-6	no	-
	F(0,0)	A(0,3)	C(1,1)	-3	no	-
	F(0,0)	A(0,3)	D(2,1)	-6	no	-
	F(0,0)	A(0,3)	E(3,0)	-9	no	-
	F(0,0)	A(0,3)	F(0,0)	0	no	-
2	A(0,3)	B(2,2)	C(1,1)	-3	no	-
	A(0,3)	B(2,2)	D(2,1)	-2	no	-
	A(0,3)	B(2,2)	E(3,0)	-3	no	-
	A(0,3)	B(2,2)	F(0,0)	-6	no	-
	A(0,3)	B(2,2)	A(0,3)	0	no	-
3	B(2,2)	A(0,3)	C(1,1)	3	yes	C
	B(2,2)	C(1,1)	D(2,1)	1	yes	D
	B(2,2)	D(2,1)	E(3,0)	1	yes	E
	B(2,2)	E(3,0)	F(0,0)	-6	no	-
4	E(3,0)	A(0,3)	B(2,2)	-3	no	-
	E(3,0)	A(0,3)	C(1,1)	3	yes	C
	E(3,0)	C(1,1)	D(2,1)	-1	no	-
	E(3,0)	C(1,1)	F(0,0)	3	yes	F

Final convex hull => F, A, B, E, F

Step 1

orientation (F, A, B) = (0-0)(2-0) - (3-0)(2-0) = -6 < 0 → CW

orientation (F, A, C) = (0-0)(1-0) - (3-0)(1-0) = -3 < 0 → CW

orientation (F, A, D) = (0-0)(1-0) - (3-0)(2-0) = -6 < 0 → CW

orientation (F, A, E) = (0-0)(0-0) - (3-0)(3-0) = -9 < 0 → CW

orientation (F, A, F) = (0-0)(0-0) - (3-0)(0-0) = 0 → colinear

Step 2

orientation (A, B, C) = (2-0)(1-3) - (2-3)(1-0) = -3 < 0 → CW

orientation (A, B, D) = (2-0)(1-3) - (2-3)(2-0) = -2 < 0 → CW

orientation (A, B, E) = (2-0)(0-3) - (2-3)(3-0) = -3 < 0 → CW

orientation (A, B, F) = (2-0)(0-3) - (2-3)(0-0) = -6 < 0 → CW

orientation (A, B, A) = colinear with self → ignore

Step 3

orientation (B, A, C) = (0-2)(1-2) - (3-2)(1-2) = 3 > 0 → CCW

orientation (B, C, D) = (1-2)(1-2) - (1-2)(2-2) = 1 > 0 → CCW

orientation (B, D, E) = (2-2)(0-2) - (1-2)(3-2) = 1 > 0 → CCW

orientation (B, E, F) = (3-2)(0-2) - (0-2)(0-2) = -6 < 0 → CW

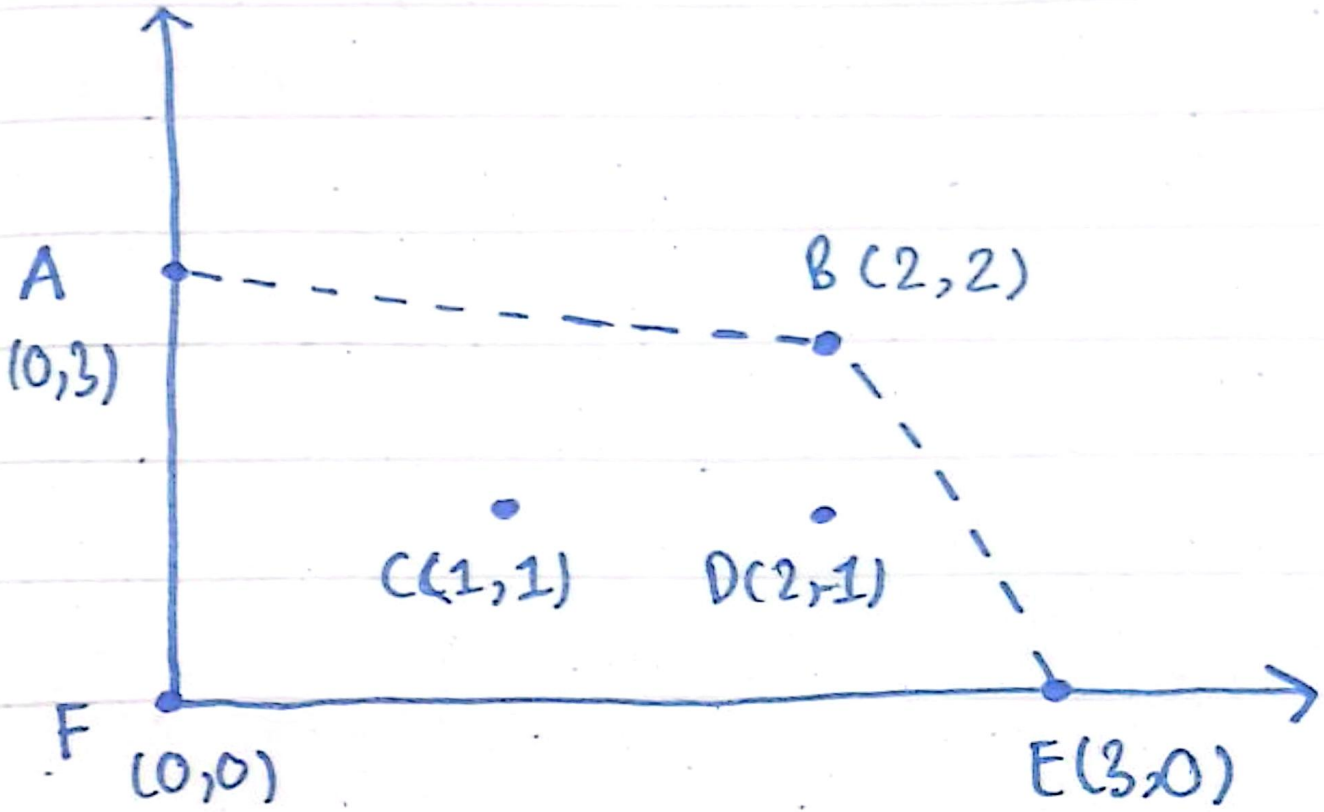
Step 4

orientation (E, A, B) = (0-3)(2-0) - (3-0)(2-3) = -3 < 0 → CW

orientation (E, A, C) = (0-3)(1-0) - (3-0)(1-3) = 3 > 0 → CCW

orientation (E, C, D) = (1-3)(1-0) - (1-0)(2-3) = -1 > 0 → CW

orientation (E, C, F) = (1-3)(0-0) - (1-0)(0-3) = 3 > 0 → CCW



Convex Hull
Diagram

[F, A, B, E, F]

Q2) 1. BF_FailureFunction (P):

```
m = len(P)
F = array of zeroes w/ size m
For(j=0; j<=m-1; j++){
    F[j] = 0
~~For(k=j; k<=~~
For(k=j; j>=1; j--){
    prefix = P[0:k-1]
    suffix = P[j-k+1:j]
    if prefix == suffix:
        F[j] = k
        break
return F
```

Time complexity → $O(m^2)$

2. KMP_FailureFunction (P): Time complexity → O(m)

```
m = len(P)
F = [0]*m
k = 0
for j in range(i,m):
    while k>0 && P[k] != P[j]:
        k = F[k-1]
    if P[k] == P[j]:
        k++
    F[j] = k
return F
```

3. Brute force

j	P[0:j]	Proper prefix	Proper Suffix	Largest prefix = suffix	F[j]
0	a	–	–	–	0
1	ab	a	b	–	0
2	aba	a, ab	a, ba	a	1
3	abab	a, ab, aba	b, ab, bab	ab	2
4	ababa	a, ab, aba, abab	a, ba, aba, baba	aba	3
5	ababac	a, ab, aba, abab, ababa	c, ac, bac, abac babac	–	0
6	ababaca	a, ab, aba, abab, ababa, ababac	a, ca, aca, baca, abaca, babaca	a	1

KMP

j	P[j]	K (prev F)	P[K] = P[j] ?	F[j]
0	a	0	–	0
1	b	0	P[0]=a vs P[1]=b X	0
2	a	0	P[0]=a vs P[2]=a ✓	1
3	b	1	P[1]=b vs P[3]=b ✓	2
4	a	2	P[2]=a vs P[4]=a ✓	3
5	c	3	P[3]=b vs P[5]=c X	0
6	a	0	P[0]=a vs P[6]=a ✓	1

Brute force = [0, 0, 1, 2, 3, 0, 1]

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

Brute Force	KMP	
$O(m^2)$	$O(m)$	← Time complexity
$\frac{m(m-1)}{2}$ comparisons in worst case	$2m-1$ in worst case	← No. of comparisons
Every j is computed independently	Uses previous failure values to fall back when mismatch occurs	← Reuse of computed info

Q3) a) $dp[0] = 1 \rightarrow$ # of ways (sum)

For i from c to n:

$$dp[i] += dp[i-c]$$

Coin = 1 Coin = 5

i	dp[i]	i	dp[i] (before)	dp[i-5]	dp[i] (after)
1	1	5	1	dp[0] = 1	2
2	1	6	1	dp[1] = 1	2
3	1	7	1	dp[2] = 1	2
4	1	8	1	dp[3] = 1	2
5	1	9	1	dp[4] = 1	2
6	1	10	1	dp[5] = 2	3
7	1	11	1	dp[6] = 2	3
8	1	12	1	dp[7] = 2	3
9	1	13	1	dp[8] = 2	3
10	1				
11	1				
12	1				
13	1				

Coin = 6

i	dp[i] (before)	dp[i-6]	dp[i] (after)
6	2	1	3
7	2	1	3
8	2	1	3
9	2	1	3
10	3	1	4
11	3	2	5
12	3	3	6
13	3	3	6

Coin = 8

i	dp[i] (before)	dp[i-8]	dp[i] (after)
8	3	1	4
9	3	1	4
10	4	1	5
11	5	1	6
12	6	1	7
13	6	2	8

dp[13] = 8

Q3b) str1 = I T T E N

str2 = S I T T I N G

if str1[i-1] == str2[j-1]:

dp[i][j] = dp[i-1][j-1] #same char

else:

dp[i][j] = 1 + min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1])

i/j	^	S	I	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

dp[6][7] = 3

1. Substitute K → S : SITTEN
2. Substitute E → I : SITTIN
3. Insert G at the end : SITTING

i = 1 (K) j = 1 (S)

K ≠ S → 1 + min (dp[0][2], dp[1][0], dp[0][0])

=> 1 + min (1, 1, 0) = 1

i = 1 (K), j = 2 (I)

K ≠ I → 1 + min (dp[0][2], dp[1][1], dp[0][1])

=> 1 + min (2, 1, 2) = 2

(Similar computations for the rest)

Q3 c) $dp[i]$ = max revenue for length j

$dp[j] = \max\{1 \le i \le j\} (dp[i] + dp[j-1])$

$cut[j]$ = argmax_i above

$j = 1$:

↳ $i = 1 \Rightarrow$ $p1 + dp[0] = 1 + 0 = 1$

$dp[1] = 1$, $cut[1] = 1$

$j = 2$:

↳ $i = 1$: $p1 + dp[1] = 1 + 1 = 2$

↳ $i = 2$: $p1 + dp[0] = 5 + 0 = 5$

$dp[2] = 5$, $cut[2] = 2$

$j=3$:

↳ $i=1$: $1+dp[2]=1+5=6$

↳ $i=2$: $5+dp[1]=5+1=6$

↳ $i=3$: $8+dp[0]=8+0=8$

$dp[3]=8$, $cut[3]=8$

$j = 4$:

↳ $i = 1$: $1 + dp[3] = 1 + 8 = 9$

↳ $i = 2$: $5 + dp[2] = 5 + 5 = 10$

↳ $i = 3$: $8 + dp[1] = 8 + 1 = 9$

↳ $i = 4$: $9 + dp[0] = 9 + 0 = 9$

$dp[4] = 10$, $cut[4] = 2$

j=3:

↳ i=2: 1+dp[4] = 11

↳ i=2: 3+dp[3] = 13

↳ i=3: 8+dp[2] = 13

↳ i=4: 9+dp[1] = 10

↳ i=5: 10+dp[0] = 10

dp[3] = 13, cut[3] = 2
(or 5)

$j = 6$:

↳ $i = 2$: $1 + dp[5] = 14$

↳ $i = 2$: $5 + dp[4] = 15$

↳ $i = 3$: $8 + dp[3] = 16$ ✓

↳ $i = 4$: $4 + dp[2] = 14$

↳ $i = 5$: $10 + dp[1] = 11$

↳ $i = 6$: $16 + dp[0] = 16$ (tie)

$j-1$)

$j=7$:

↳ $i=1$: $1+dp[6]=17$

↳ $i=2$: $5+dp[5]=18$

↳ $i=3$: $8+dp[4]=18$ (tie)

↳ $i=4$: $9+dp[3]=17$

↳ $i=5$: $10+dp[2]=15$

↳ $i=6$: $16+dp[1]=17$

↳ $i=7$: $18+dp[0]=18$ (tie)

$dp[7]=18$, $cut[7]=2$

(or 3 or 7)

dp = [0, 1, 5, 8, 10, 13, 16, 18, 21]

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

j = 8 → cut[8] = 2 　　8 − 2 = 6

j = 6 → cut[6] = 3 　　6 − 3 = 3

j = 3 → cut[3] = 3 　　3 − 3 = 0

Optimal Cutting = 2 + 3 + 3

Total Value ⇒ p2 + p3 + p3 = 5 + 8 + 8 = 21

Q3d) wordDict = {i, like, ice, cream, icecream, mobile, apple}

S = likeapple

dp[0] = true

dp[1] = dp[0] + "i" → true

dp[5] = dp[5] + "like" → true

dp[10] = dp[7] + "apple" → False

dp[j] = True if dp[i] == True & s[1:j-1] in wordDict

index(j)	dp[j]	substring s[0:j-1]	Valid Split
0	True	""	base case
1	True	"i"	"i"
2	False	"il"	–
3	False	"ili"	–
4	False	"ilik"	–
5	True	"ilike"	"i + like"
6	False	"iliked"	–
7	False	"ilikeap"	–
8	False	"ilikeapp"	–
9	False	"ilikeappl"	–
10	True	"ilikeapple"	"ilike + apple"

segment possible → "i + like + apple"

Q4) Scenario: Meal Planning for a Fitness goal

i/j	0	100	200	300	400	500
None (0)	0	0	0	0	0	0
Egg (70)	1	0	6	12	18	24	30
Chicken (200)	2	0	6	25	31	50	56
Milk (150)	3	0	6	25	31	50	56
Oats (100)	4	0	6	25	31	50	56

Max Protein = 56

1 egg + 2 chicken => 470 calories, 56g protein