Ibrahim Johar Farooqi
23K-0074 (BAI-5A)
Design & Analysis & Algorithms – Assignment 3.

cross prod < 0 → CW
" > 0 → CCW
= 0 → collinear

Day / Date:

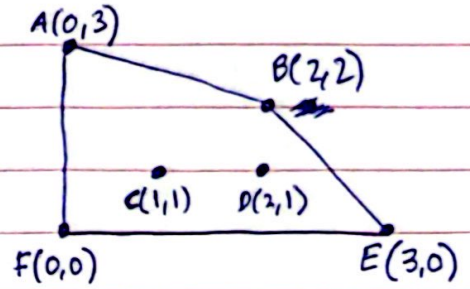
Q.1)

↳ orientation(F,A,B) = (0-0)(2-0) - (3-0)(2-0) = -6 CW
↳ orientation(F,A,C) = (0-0)(1-0) - (3-0)(1-0) = -3 CW
↳ orientation(F,A,D) = (0-0)(1-0) - (3-0)(2-0) = -6 CW
↳ orientation(F,A,E) = (0-0)(0-0) - (3-0)(3-0) = -9 CW
↳ orientation(F,A,F) = (0-0)(0-0) - (3-0)(0-0) = 0 collinear
↳ orientation(A,B,C) = (2-0)(1-3) - (2-3)(1-0) = -3 CW
↳ orientation(A,B,D) = (2-0)(1-3) - (2-3)(2-0) = -2 CW
↳ orientation(A,B,E) = (2-0)(0-3) - (2-3)(3-0) = -3 CW
↳ orientation(A,B,F) = (2-0)(0-3) - (2-3)(0-0) = -6 CW
↳ orientation(A,B,A) → collinear with itself
↳ orientation(B,A,C) = (0-2)(1-2) - (3-2)(1-2) = 3 CCW
↳ orientation(B,C,D) = (1-2)(1-2) - (1-2)(2-2) = 1 CCW
↳ orientation(B,D,E) = (2-2)(0-2) - (1-2)(3-2) = 1 CCW
↳ orientation(B,E,F) = (3-2)(0-2) - (0-2)(0-2) = -6 CW
↳ orientation(E,A,B) = (0-3)(2-0) - (3-0)(2-3) = -3 CW
↳ orientation(E,A,C) = (0-3)(1-0) - (3-0)(1-3) = 3 CCW
↳ orientation(E,C,D) = (1-3)(1-0) - (1-0)(2-3) = -1 CW
↳ orientation(E,C,F) = (1-3)(0-0) - (1-0)(0-3) = 3 CCW

day/date:

Step	current Point(P)	Candidate (q)	checking point(i)	Orientation (p,q,r)	CCW more?	New Candidate
1	F	A	B	-6	No	—
2	F	A	C	-3	No	—
3	F	A	D	-6	No	—
4	F	A	E	-9	No	—
5	F	A	F	0	No	—
6	A	B	C	-3	No	—
7	A	B	D	-2	No	—
8	A	B	E	-3	No	—
9	A	B	F	-6	No	—
10	A	B	A	0	No	—
11	B	A	C	3	Yes	C
12	B	C	D	1	Yes	D
13	B	D	E	1	Yes	E
14	B	E	F	-6	No	—
15	E	A	B	-3	No	—
16	E	A	C	3	Yes	C
17	E	C	D	-1	No	—
18	E	C	F	3	Yes	F

↳ final convex hull is **F, A, B, E, F**



day / date:

Q.2)

1) Brute-Force Algo {

```
for (j=0 to m-1)
    F[j] = 0
    substring = P[0...j]
    for (k=n-1  down to 1)          // for testing all possible suffix-prefix lengths
        prefix = substring[0...k-1]
        suffix = substring[n-k ... n-1]
        if (prefix == suffix)
            then F[j] = k           // the largest k that matches will be stored
            break
        end if
    end for
end for
return F
}
```

↳ time complexity:

↳ for each of j[1...n-1], we will compare upto [j] (prefix-suffix) pairs

↳ string comparision → each takes O(j)

↳ time complexity: $O(n^2)$

2) Optimized KMP algo

```
Algorithm KMP(P){
    F = [0] * length(P)
    j = 0
    for (i in range (1, length(P)))
        while (j>0 and P[i] != P[j])
            j = F[j-1]
        if( P[i] == P[j])
            j += 1
        F[i] = j
return F
}
```

day / date:

3) Brute Force Table

j	P[0..j]	proper prefix	proper suffix	longest 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, bcbaca	a	1

KMP Table

j	P[j]	k (prev F)	P[k] = P[j] is ?	F[j]
0	a	0	–	0
1	b	0	a ≠ b	0
2	a	0	a = a	1
3	b	1	b = b	2
4	a	2	a = a	3
5	c	3	b ≠ c	0
6	a	0	a = a	1

↳ Brute Force = [0, 0, 1, 2, 3, 0, 1]

↳ KMP = [0, 0, 1, 2, 3, 0, 1]

day / date:

4)

	Brute Force	KMP
Time Complexity	$O(n^2)$	$O(n)$
Num of Comparisons	worst case → $\frac{n(n-1)}{2}$ comparisons	(2n-1) → worst case (comparisons)
Reuse of Computed Information	each j instance is computed individually, no reuse	uses the value of the previous failure to go back to, when there is a mismatch, essentially reuse of computed info when mismatch

Q.3) a) $S = \{1, 5, 6, 8\}$, desired change = 13

↳ using unbounded knapsack, unlimited use of coins

using:	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 possible to achieve the desired change (13)

combinations

num	combinations
1	1+1+1+1+1+1+1+1+1+1+1+1+1
2	5+1+1+1+1+1+1+1+1
3	5+5+1+1+1
4	6+1+1+1+1+1+1+1
5	6+5+1+1
6	6+6+1
7	8+5
8	8+1+1+1+1+1

day / date:

Q.3) b)

str1 = "KITTEN"

str2 = "SITTING"

		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

position	str1	str2	Action
1	K	S	substitute K → S
2	I	I	match
3	T	T	match
4	T	T	match
5	E	I	substitute E → I
6	N	N	match
7	(nil)	G	G inserted at the end

day / date:

c) L = [1,2,3,4,5,6,7,8]
P = [1,5,8,9,10,16,18,20]

	0	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	1	1
5	2	1	(5)	6	6	6	6	6	6
8	3	1	5	8	9	13	14	14	14
9	4	1	5	8	9	13	14	17	18
10	5	1	5	8	9	10	14	17	18
16	6	1	5	8	9	10	16	17	(21)
18	7	1	5	8	9	10	16	18	21
20	8	1	5	8	9	10	16	18	21

↳ selected items are {2,6}
↳ essentially {5,16}

d) input s = "ilikeapple"
word dict = {'i', 'like', 'ice', 'cream', 'mobile', 'apple'}

Algorithm:

```
n = length(s)
array dp[0...n] , initialise all to False
dp[0] = True      //empty string is valid
for (i from 1 to n)
   for(j from 0 i-1)
      if (dp[j] = True and substring(s,j,i) belongs to wordDict then)
         dp[i] = True
         break
return dp[n]
```

day / date:

i	substring		
0	""	✓	empty string, valid
1	"i"	✓	in dict
2	"il"		
3	"ili"		
4	"ilik"		
5	"ilike"	✓	in dict
6	"ilikea"		
7	"ilikeap"		
8	"ilikeapp"		
9	"ilikeappl"		
10	"ilikeapple"	✓	in dict

Q4) scenario: grocery shopping w/ limited budget
- ↳ each grocery item has a cost & nutrional benefit
- ↳ goal is to maximise total nutritional value
- ↳ budget → $10
- ↳ items: [Apple, Banana, Milk, Eggs]
- ↳ cost: [$3, $2, $5, $6]
- ↳ nutrion: [4, 3, 7, 9] calories

	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	4	4	4	4	4	4	4	apple (4)
2	0	0	0	3	4	4	4	7	7	7	7	banana (3)
3	0	0	0	3	4	4	4	7	7	7	(10)	milk (7)
4	0	0	0	3	4	4	4	7	7	9	10	eggs (9)

items selected: {milk, banana}
- ↳ maximising nutritional value