DAA Assignment #03

(1)

Step #1 :- (Starting Point):

Smallest x-coordinate

if not present then we will take F(0,0)

Step #02 Iteration

i1 : starting Point F(0,0)

STEP	Point	CANDIDATE	R	Orientation	ccw
1	F(0,0)	A(0,3)	B(2,2)	-6 < 0	No
1.2	F(0,0)	A(0,3)	C(1,1)	-3 < 0	No
1.3	F(0,0)	A(0,3)	D(2,1)	-6 < 0	No
1.4	F(0,0)	A(0,3)	E(3,0)	-9 < 0	No

New Candidate

A (0, 3) ← Next Hull Point

i2 :- iteration :-02

Step	P	Q	R	Orientation	ccw	Next
2.1	A(0,3)	B(2,2)	C(2,1)	-3 < 0	No	
2.2	A(0,3)	B(2,2)	D(2,1)	-2 < 0	No	B(2,2)
2.3	A(0,3)	B(2,2)	E(3,0)	-3 < 0	No	..
2.4	A(0,3)	B(2,2)	F(0,0)	-6 < 0	No	

CamScanner

A_3:- Iteration 03

starting Point B(2,2)

step #	P	Q	R	orientation	more ccw	Next candidate
3-1	B(2,2)	C(1,1)	D(2,1)	1 > 0	Yes	D(2,1)
3-2	B(2,2)	D(2,1)	E(3,0)	1 > 0	Yes	E(3,0)
3-3	B(2,2)	E(3,0)	F(0,0)	-6 < 0	No	
3-4	B(2,2)	E(3,0)	A(0,3)	-3 < 0	No	

Next E(3,0)

A_4:

step	P	Q	R	orientation	ccw	new candidate
4.1	E(3,0)	F(0,0)	A(0,3)	-9 < 0	No	F(0,0)
4.2	E(3,0)	F(0,0)	B(2,2)	-6 < 0	No	
4.3	E(3,0)	F(0,0)	C(1,1)	-3 < 0	No	
4.4	E(3,0)	F(0,0)	D(2,1)	-3 < 0	No	

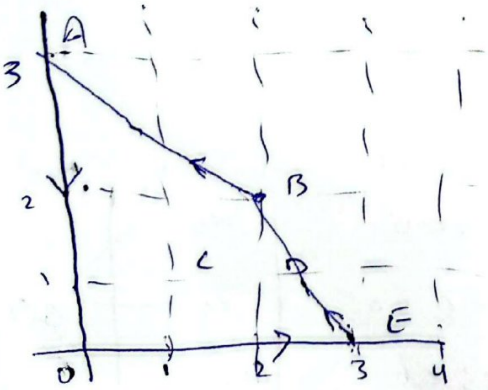
Next to starting Point

order

F → A → B → C → F

counter-clockwise

F → E → B → A → F



2:- Brute Force :-

```
#include <iostream>

void BruteForceFunction (const string &Pattern) int S[]
    int n= Pattern.length();
    for (int j=0; j<n; j++){
     int max_length= INT_MAX;
     string substring = Pattern.substr (0, j+1);
     for (int k=j ; k>0; k--){
      string prefix = substring.substr (0, k)
      string suffix. substr (j-k+1, k);
       if (prefix == suffix)
      { max_length = k ;
        break; } }
      S[j] = max_length;
    }
```

Conclusion:-

Outer loop runs n times for each j
inner loop tests prefix lengths
O(k) in WC
O(n^3)

CamScanner

Q (2.2)

(4)

```
void KMP( const string &,patt , int S[])
{
  int n= patt.length();
   S[0]=0;
   int i=0;
  for(int j=1 ; j<n; j++)
  { while( i>0 && patt[i] != patt[j]) {
          i= S[i-1];
       }
     if (patt[i] ==patt[j]) {
        itt; }
     else {
        i=0; }
     S[j]= i;
  }
}
```

(i) Uses KMP function itself to backtrack.

(ii) Offers mismatching it jumps to S[i-1] instead of brute force

(iii) maintains stability

Conclusion

O(n) linear time.

CamScanner

"ababaca"

j		P:s	s[j]
0	a	—	0
1	ab	—	0
2	aba	a	1
3	abab	ab	2
4	ababa	aba	3
5	ababac	~~ab~~	0
6	ababaca	a	1

*KMP Not succeeded :-

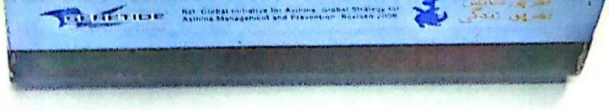
0	a	—	0	
01	ab	—	0	p[1] ≠ p[0]
02	aba	a	1	p[2] = p[0]
03	abab	ab	2	p[3] = p[1] (i=2)
04	ababa	aba	3	p[4] = p[2] (i=3)
05	ababac	-	0	p[5] ≠ p[3]
6	ababaca	a	1	p[6] = p[0] (i=1)

No of Comparision = 8

{ 0,0, 1, 2, 3, 0, 1 }

CamScanner

Q(2.4) :-



(6)

T.C

B.F → $O(n^3)$ - Cubic

KMP → $O(n)$ - Linear Time

B.F → 18 Comparision

KMP → 8 comparision

B.F → ~~Checks~~ form zero after missmatch

KMP → we use builds

KMP > B.F

3:-

(A) Coin:-

```
#include<iostream>
int coin_ways( int coins[], int n, int m)
{   int sol[n+1] = {0};
    sol[0] = 1;
    for(int i=0; i<m; i++) {
        for(int j= coins[i]; j<n; j++) {
            sol[j] += sol[j- coins[i]];
        }
    }
    return sol[n];
}
```

Driver function:-

```
int main() {
coins[] = {1, 5, 6, 8}
int n = sizeof(coins)/
        sizeof(coins[0]);
int amount=3;
//Number of ways
coin_ways(coins, n);

//output 8
```


CamScanner

B:- CODE :-

```
int e_distance (string str1, string str2) {
    int m = str1.length();
    int n = str2.length();
    int sol[m+1][n+1];
for (int i=0; i<=m; i++)
 { for (int j=0; j<=n; j++) {
    if (i==0)
    sol[i][j]=j;
  else if (str2[i-1] == str1[j-1]) {
     sol[i][j] = sol[i-1][j-1]
    else
    sol[i][j] = 1 + min (sol[i][j-1], sol[i-1][j], sol[i-1][j-1])
  }}
    return sol[m][n];
}
```

Driver function:-
string str1 = "kitten" str2 = "SITTING";
e_distance (str1, str2);
Output = 3

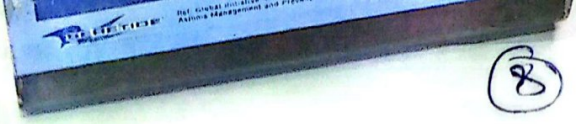
C:- CODE :-

```
int rodCutting (int price[], int n)
{   int sol[n+1];
    sol[0]=0;
    for (int i=1; i<=n; i++) {
        int max_value = -1
      for (int j=0; j<i; j++)
        { max_value = max (max_value, price[j]+sol[i-j-1]);
        } sol[i] = max_val; }
      return sol[n];
}
```

Driver function;
int P[] = {1, 5, 8, 9, 10, 16, 18, 20}
int len = 8;
rodCutting (P, len);
Output => 22

CamScanner

141



(8)

Q:-

```
bool words(string s, string dict[], int n)
{  int m=s.length();
   bool sol[m+1] = {false};
   sol[0] = true;
   for(int i=1; i<=n; i++) {
      for(int j=0; j<n; j++)
      {  string w= dict[j];
         int len = w.length();
      if (i>=len && dic[i-len] && s.substr(i-len, len==w)
      {
         sol[i]= true;
         break;
      }}}
      return sol[n];
   }
```

Driver function.

```
string dic[]= {"i", "like", "ice", "cream", "icecream",
                "mobile", "apple"};
string check= "ilikeapple";
int n= 7;
words(check, dic, n);
```

Output // Yes

CamScanner

(4)

#include <iostream>

Using namespace std;

before writing code let's map it first

w[i] study hours of course

v[i] credit of course

Capacity(w) Maximum hours student can spend

e.g:-

Course	Credits	Hours
C_1	5	4
C_2	3	2
C_3	6	5
C_4	4	3

We will chose only those which gives total credit ≤ 8 hours.

$\{C_1, C_2, C_3, C_4\}$

1 0 0 1 : 9 & 7 hours ≤ 8

Algorithm Selection of Courses (credits [ ], h[ ], max H)

```
int sol[n+1][maxH+1];

for (i=0 ; i<=n ; i++)
{ for (int j=0 ; j<= maxH ; j++)
    { if (i==0 || j==0)
        { sol[i][j]=0 ; }
    elseif ( h[i-1] <= j)
        sol[i][j] = max ( credits[i-1] + sol[i-1][j-h[i-1]],
                          sol[i-1][j])

    else {
    sol[i][j] = sol[i-1][j]
}}
return sol[n][maxH];
```

}

:- Algo Type :- 0/1 knapsack

T.C => O(n·m)

H	0	C_2	C_2	C_3	C_4
0	0	0	0	0	0
2	0	0	3	3	3
3	0	0	3	3	4
4	0	5	5	5	5
5	0	5	5	6	6
6	0	5	8	8	8
7	0	5	8	9	9
8	0	5	8	9	9

CamScanner