



EXAMEN DEL 3DO PARCIAL

Calcule el camino óptimo para trasladarse de la esquina superior izquierda hasta la esquina inferior derecha con programas usando las siguientes técnicas algorítmicas:

- A) Fuerza bruta (33/100 puntos)
- B) Voraz (33/100 puntos)
- C) Programación Dinámica (33/100 puntos)

	3.1	2.07	2.01	3.1	2.06	3.04
3.02		2.04		2.09		2.04
	3.04		2.02		3.1	2.09
2.04		2.05		2.01		2.1
	2.02		3.05		3.01	2.09
3.05		2.09		2.07		2.08
	2.07		3.05		2.01	2.01
3.05		3.09		3.08		3.03
	2.05		2.1		2.09	2.05
2.03		3.05		3.06		2.09
	2.05		2.08		3.03	2.09
2.08		2.07		3.1		2.02
	2.08		3.06		2.01	2.1

Definir las matrices de distancias en el lenguaje de programación Python:

```
wr = [ [3.02,2.04,2.09,2.04,3.03,2.05,2.05], #Matriz que representa las filas
        [2.04,2.05,2.01,2.1,3.1,2.04,2.06],
        [3.05,2.09,2.07,2.08,3.03,2.02,3.05],
        [3.05,3.09,3.08,3.05,3.03,3.1,2.06],
        [2.03,3.05,3.06,2.09,2.07,3.02,3.04],
        [2.08, 2.07, 3.1, 2.02,3.05,2.1,2.07] ]
wc = [ [3.1,2.07,2.01,3.1,2.06,3.04], #Matriz que representa las columnas
        [3.04,2.02,3.1,2.09,2.08,2.03],
        [2.02,3.05,3.01,2.09,2.09,2.04],
        [2.07,3.05,2.01,2.01,3.04,3.05],
        [2.05,2.1,2.09,2.05,3.08,3.07],
        [2.05,2.08,3.03,2.09,3.09,3.1],
        [2.08,3.06,2.01,2.1,2.02,2.06] ]
```

A)

B) Estrategia Voraz:

Representación de la solución en Python

```
16 def ManhattanTouristVoraz(dis, x, y):
17     wc=dis[0]
18     wr=dis[1]
19     n=len(wr)
20     m=len(wc)
21     s=np.zeros((n+1,m))
22     ver=False
23     i=0
24     j=0
25     r=0
26     while (ver!=True):
27         if (i==x):
28             r=r+wr[i][j+1]
29             ver=True
30         elif(j==y):
31             r=r+wr[i+1][j]
32             ver=True
33         else:
34             r, var=Max2(r,s[i][j] + wr[i][j],s[j][i] + wc[j][i])
35             if var==True:
36                 j=j+1
37             else:
38                 i=i+1
39     print(s)
40     return r
41
```

Definición en la clase main del programa:

```
70     print("Estrategia Voraz:")
71     s=ManhattanTouristVoraz(dis,1,1)
72     print(s)
```

C)

Respuesta:

El más optimo tiene un valor de: 6.06(lo cual no se cumple)

```
Estrategia Voraz:
[[ 0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.]
 5.15
```

D) Programación Dinámica:

Representación de la solución en Python

```
4 def ManhattanTouristDP(wc, wr, n, m):
5     s=np.zeros((n+1,m))
6     for i in range(1,n+1):
7         s[i][0]= s[i-1][0] + wr[i-1][0]
8     for j in range(1,m):
9         s[0][j]= s[0][j-1] + wc[0][j-1]
10    for i in range(1, n+1):
11        for j in range(1,m):
12            s[i][j] = Max(s[i-1][j] + wr[i-1][j] , s[i][j-1] + wc[i][j-1] )
13    print(s)
14    return s[n][m-1]
15
```

Definición en la clase main del programa:

```
36     print("Programación Dinámica:")
37     s=ManhattanTouristDP(wc, wr, len(wr), len(wc))
38     print(s)
```

Respuesta:

El más óptimo tiene un valor de: 33.77

```
Programación Dinámica:
[[ 0.    3.1   5.17  7.18  10.28  12.34  15.38]
 [ 3.02  6.06  8.08  11.18  13.31  15.39  17.43]
 [ 5.06  8.11  11.16  14.17  16.41  18.5   20.54]
 [ 8.11  10.2  13.25  16.25  19.44  22.48  25.53]
 [ 11.16 13.29 16.33  19.3   22.47  25.58  28.65]
 [ 13.19 16.34 19.39  22.42  24.54  28.6   31.7 ]
 [ 15.27 18.41 22.49  24.5   27.59  30.7   33.77]]
33.77
```