

# Actividad 1.1 Algoritmos de fuerza bruta

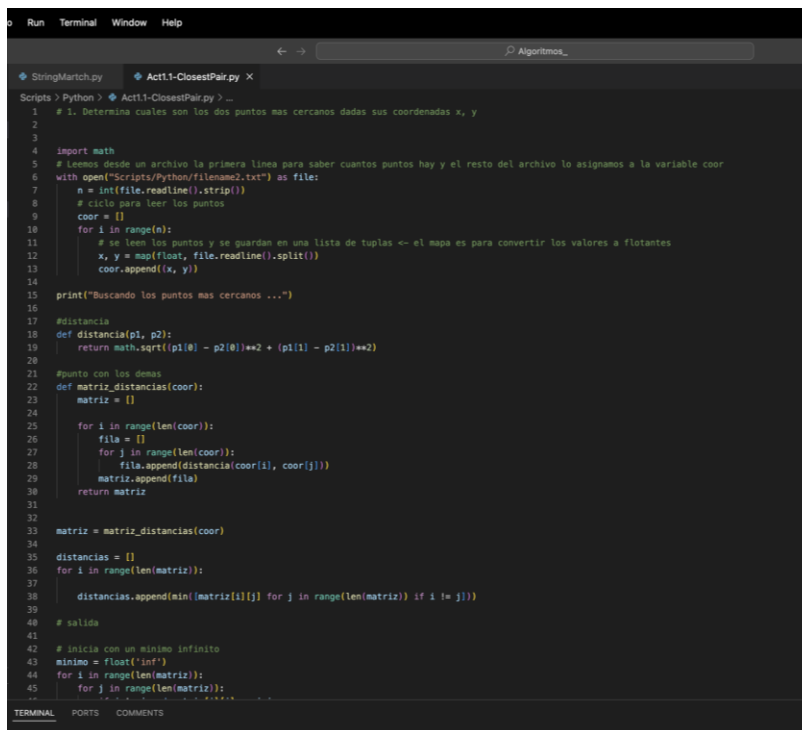
Victor Misael Escalante Alvarado

A01741176

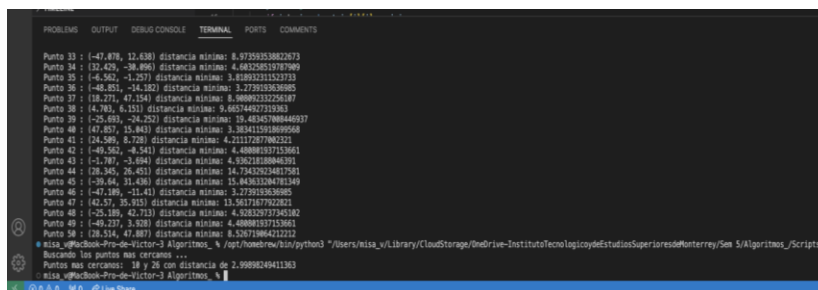
## Problemas trabajados en clase :

1. **Closest pair.** Por cada archivo de prueba (disponibles en Teams, clase 02), muestra las coordenadas de los puntos más cercanos y su distancia.

## Capturas de pantalla del código funcionando



```
1 # 1. Determina cuales son los dos puntos mas cercanos dadas sus coordenadas x, y
2
3
4 import math
5 # Leeos desde un archivo la primera linea para saber cuantos puntos hay y el resto del archivo lo asignamos a la variable coor
6 with open("Scripts/Python/fileName2.txt") as file:
7     n = int(file.readline().strip())
8     # ciclo para leer los puntos
9     coor = []
10    for i in range(n):
11        # se leen los puntos y se guardan en una lista de tuplas <- el mapa es para convertir los valores a flotantes
12        x, y = map(float, file.readline().split())
13        coor.append((x, y))
14
15    print("Buscando los puntos mas cercanos ...")
16
17    #distancia
18    def distancia(p1, p2):
19        return math.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)
20
21    #punto con los demas
22    def matriz_distancias(coor):
23        matriz = []
24
25        for i in range(len(coor)):
26            fila = []
27            for j in range(len(coor)):
28                fila.append(distancia(coor[i], coor[j]))
29            matriz.append(fila)
30        return matriz
31
32
33    matriz = matriz_distancias(coor)
34
35    distancias = []
36    for i in range(len(matriz)):
37
38        distancias.append(mini([matriz[i][j] for j in range(len(matriz)) if i != j]))
39
40    # salida
41
42    # inicia con un minimo infinito
43    minimo = float("inf")
44    for i in range(len(matriz)):
45        for j in range(len(matriz)):
46            if matriz[i][j] < minimo:
47                minimo = matriz[i][j]
```



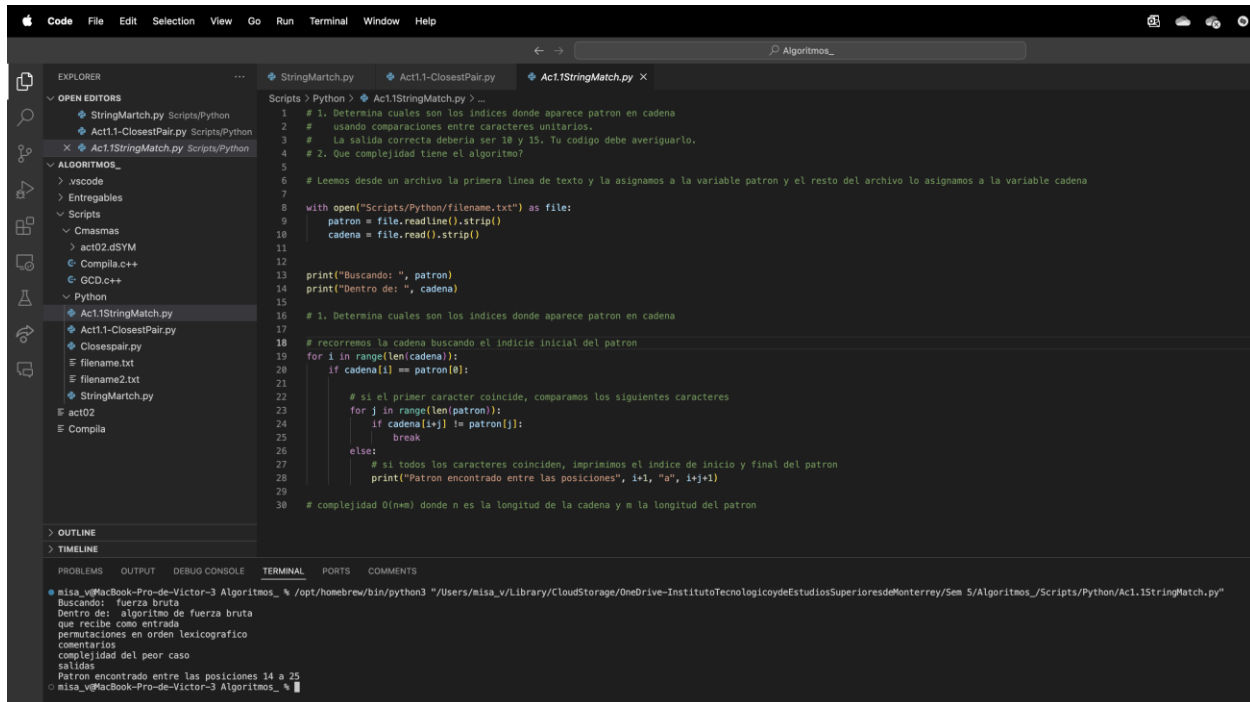
```
Punto 33 : (-47.478, 12.438) distancia minima: 8.97359358822673
Punto 34 : (32.429, -38.896) distancia minima: 4.603256519787969
Punto 35 : (-4.562, -1.257) distancia minima: 3.81893215237319
Punto 36 : (-48.853, -34.182) distancia minima: 3.2739153636985
Punto 37 : (18.272, 47.154) distancia minima: 8.98692322242497
Punto 38 : (4.789, 6.151) distancia minima: 9.65746027319363
Punto 39 : (-25.693, -34.253) distancia minima: 19.483457888446937
Punto 40 : (47.837, 15.843) distancia minima: 3.354115188699564
Punto 41 : (24.569, 8.708) distancia minima: 4.21172877862321
Punto 42 : (-48.582, -4.543) distancia minima: 4.48881837133661
Punto 43 : (-2.787, -3.984) distancia minima: 4.93612188643631
Punto 44 : (28.345, 26.451) distancia minima: 14.734326234817581
Punto 45 : (-39.44, 31.438) distancia minima: 15.843631284781549
Punto 46 : (-47.396, -12.433) distancia minima: 13.7791263636895
Punto 47 : (42.57, 35.853) distancia minima: 13.5617387792821
Punto 48 : (-25.189, 42.713) distancia minima: 4.9283797740182
Punto 49 : (-45.237, 3.933) distancia minima: 4.48881837133661
Punto 50 : (28.334, 47.887) distancia minima: 8.526719864212212
Buscando los puntos mas cercanos ...
Puntos mas cercanos: 38 y 26 con distancia de 2.9989249411363
```

2. **String-matching.** Muestra cada patrón (archivos disponibles en Teams, clase 02) buscado y los índices de inicio y fin en los que aparece. En el archivo de patrones,

cada línea es un caso de prueba.

Para los dos problemas anteriores, usa las instancias de prueba disponibles en Teams, clase 02

## Capturas de pantalla del código funcionando



The screenshot shows a VS Code editor with a Python script named `Ac1StringMatch.py` open. The script implements a function to find the starting index of a pattern in a string. The terminal output shows the execution of the script, which reads from a file named `filename.txt` and prints the results of the string matching algorithm.

```
1 # 1. Determina cuales son los indices donde aparece patron en cadena
2 # usando comparaciones entre caracteres unitarios.
3 # La salida correcta deberia ser 10 y 15. Tu codigo debe averiguarlo.
4 # 2. Que complejidad tiene el algoritmo?
5
6 # Leemos desde un archivo la primera linea de texto y la asignamos a la variable patron y el resto del archivo lo asignamos a la variable cadena
7
8 with open("Scripts/Python/filename.txt") as file:
9     patron = file.readline().strip()
10    cadena = file.read().strip()
11
12 print("Buscando: ", patron)
13 print("Dentro de: ", cadena)
14
15 # 1. Determina cuales son los indices donde aparece patron en cadena
16
17 # recorremos la cadena buscando el indice inicial del patron
18 for i in range(len(cadena)):
19     if cadena[i] == patron[0]:
20
21         # si el primer caracter coincide, comparamos los siguientes caracteres
22         for j in range(len(patron)):
23             if cadena[i+j] != patron[j]:
24                 break
25         else:
26             # si todos los caracteres coinciden, imprimimos el indice de inicio y final del patron
27             print("Patron encontrado entre las posiciones", i, i, "a", i+j+1)
28
29 # complejidad O(n*m) donde n es la longitud de la cadena y m la longitud del patron
```

Terminal Output:

```
Buscando: fuerza bruta
Dentro de: algoritmo de fuerza bruta
permutaciones en orden lexicografico
comentarios
complejidad del peor caso
salidas
Patron encontrado entre las posiciones 14 a 25
```

## Problemas de combinatoria y permutaciones

1. Algoritmo de generación de permutaciones en orden lexicográfico. La salida es la lista de las permutaciones. Imprime una permutación por línea a la vez.

```

1 // Permutaciones.cpp
2 #include <iostream>
3 #include <vector>
4 #include <algorithm>
5 using namespace std;
6
7 // Función para generar y copiar todas las permutaciones en orden lexicográfico
8 // Se utiliza la función swap de std::vector
9 void generate_permutations(vector<int> &arr)
10 {
11     // especificamos ordenado
12     bubble_sort(arr);
13     // Imprime la primera permutación y repite si se pueden mas
14     if (arr.size() > 1)
15     {
16         // Se genera la primera permutación
17         for (int i = 0; i < arr.size(); i++)
18             cout << arr[i] << " ";
19         cout << endl;
20         // Se genera la siguiente permutación
21         while (next_permutation(arr.begin(), arr.end()))
22             generate_permutations(arr);
23     }
24 }
25
26 int main()
27 {
28     // Se genera el array
29     vector<int> arr;
30     arr.push_back(1);
31     arr.push_back(2);
32     arr.push_back(3);
33     // Se genera las permutaciones del array
34     generate_permutations(arr);
35     return 0;
36 }

```

Output:

```

1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

```

2. Algoritmo de generación de subsets. La salida es solo la lista de binarios que representa a todos los posibles subsets. Imprime un subset por línea a la vez.

```

1 // Subsets.cpp
2 #include <iostream>
3 #include <vector>
4 #include <algorithm>
5 using namespace std;
6
7 // Función para generar y copiar todos los subsets en orden lexicográfico
8 // Se utiliza la función swap de std::vector
9 void generate_subsets(vector<int> &arr)
10 {
11     // especificamos ordenado
12     bubble_sort(arr);
13     // Imprime la primera permutación y repite si se pueden mas
14     if (arr.size() > 1)
15     {
16         // Se genera la primera permutación
17         for (int i = 0; i < arr.size(); i++)
18             cout << arr[i] << " ";
19         cout << endl;
20         // Se genera la siguiente permutación
21         while (next_permutation(arr.begin(), arr.end()))
22             generate_subsets(arr);
23     }
24 }
25
26 int main()
27 {
28     // Se genera el array
29     vector<int> arr;
30     arr.push_back(1);
31     arr.push_back(2);
32     arr.push_back(3);
33     // Se genera los subsets del array
34     generate_subsets(arr);
35     return 0;
36 }

```

Output:

```

0
1
2
3
4
5
6
7

```

Explica en el código detalladamente que hace. ¿Habría forma de hacerlo más eficiente?

Liga a repositorio : <https://replit.com/join/bpiqhcqlww-vmisa>

