INTELIGENCIA ARTIFICIAL 2

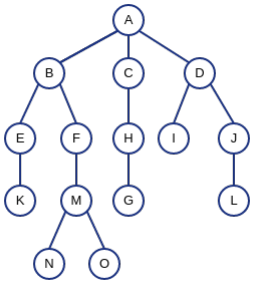
INFORME
TP1

Nicolás Tejerina – Luciano Esperlazza

AÑO 2022

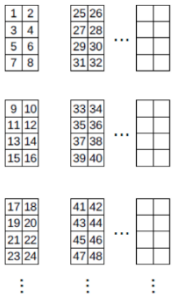
ALGORITMO A*

Características del algoritmo:



- Es un algoritmo de búsqueda global
- Tenemos un consumo de memoria que va creciendo a medida que avanza la búsqueda.
- Cada nodo es un estado y se van generando dichos estados a medida que se van recorriendo los nodos.
- Se convierte exponencial en la profundidad, es decir si en la ramificación está muy abajo la solución.
- Utilizamos una función f(n) = g(n) + h(n). [Esto, por cada nodo nos da una estimación del costo total para llegar a la meta cuando pasamos por cada nodo]
    - g(n) es el costo de la ruta desde el nodo raíz hasta el nodo n.
    - h(n) costo estimado desde el nodo n al nodo objetivo. (Heurística)

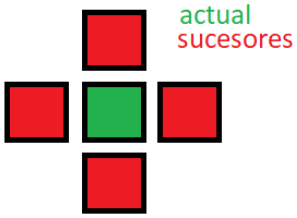
Ejercicio:



Dado un almacén con un layout similar al siguiente, calcular el camino más corto (y la distancia) entre 2 posiciones del almacén, dadas las coordenadas de estas posiciones, utilizando el algoritmo A*

Implementación:

Partimos de un nodo inicial (nodo raíz), este va a ser la entrada al almacén y expandimos sus nodos (van a ser las posibles posiciones a las que nos podemos mover a partir del actual)



Si nosotros estamos parados en el actual, los sucesores serían las posibles celdas a las que podemos movernos.

Vamos a continuar por el camino que tenga un menor valor de f y los demás nodos se van almacenando en una lista.

Tenemos 2 listas:

- LISTA ABIERTA: Para los sucesores de un nodo anterior que están sin explorar
- LISTA CERRADA: Para todos los nodos que ya fueron explorados (esto nos sirve para no explorar subcaminos más de una vez)

Los nodos que no se encuentran en ninguna de las dos listas es porque no los hemos explorados

Al nodo que abro para ver sus sucesores lo paso de la lista abierta a la lista cerrada

Una buena práctica es insertarlos en orden creciente para que tengamos en el primer lugar de la lista el nodo que tenga un menor valor de f por lo tanto va a ser el siguiente en expandir sus sucesores.

Ejecución del código:

1 Mapa del Almacén

```
Mapa = []
############ 0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17
Mapa.append([" P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P"])  # 0 PASILLO
Mapa.append([" P", " 0", " 1", " P", " P", "24", "25", " P", " P", "48", "49", " P", " P", "72", "73", " P", " P", "96"])  # 1
Mapa.append([" P", " 2", " 3", " P", " P", "26", "27", " P", " P", "50", "51", " P", " P", "74", "75", " P", " P", "97"])  # 2
Mapa.append([" P", " 4", " 5", " P", " P", "28", "29", " P", " P", "52", "53", " P", " P", "76", "77", " P", " P", "98"])  # 3
Mapa.append([" P", " 6", " 7", " P", " P", "30", "31", " P", " P", "54", "55", " P", " P", "78", "79", " P", " P", "99"])  # 4
Mapa.append([" P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P"])  # 5 PASILLO
Mapa.append([" P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P"])  # 6 PASILLO
Mapa.append([" P", " 8", " 9", " P", " P", "32", "33", " P", " P", "56", "57", " P", " P", "80", "81", " P", " P", " P"])  # 7
Mapa.append([" P", "10", "11", " P", " P", "34", "35", " P", " P", "58", "59", " P", " P", "82", "83", " P", " P", " P"])  # 8
Mapa.append([" P", "12", "13", " P", " P", "36", "37", " P", " P", "60", "61", " P", " P", "84", "85", " P", " P", " P"])  # 9
Mapa.append([" P", "14", "15", " P", " P", "38", "39", " P", " P", "62", "63", " P", " P", "86", "87", " P", " P", " P"])  # 10
Mapa.append([" P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P"])  # 11 PASILLO
Mapa.append([" P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P"])  # 12 PASILLO
Mapa.append([" P", "16", "17", " P", " P", "40", "41", " P", " P", "64", "65", " P", " P", "88", "89", " P", " P", " P"])  # 13
Mapa.append([" P", "18", "19", " P", " P", "42", "43", " P", " P", "66", "67", " P", " P", "90", "91", " P", " P", " P"])  # 14
Mapa.append([" P", "20", "21", " P", " P", "44", "45", " P", " P", "68", "69", " P", " P", "92", "93", " P", " P", " P"])  # 15
Mapa.append([" P", "22", "23", " P", " P", "46", "47", " P", " P", "70", "71", " P", " P", "94", "95", " P", " P", " P"])  # 16
Mapa.append([" P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P"])  # 17 PASILLO
```

Escogiendo como entrada del almacén la esquina superior izquierda, coordenadas (0,0). Y como meta el producto a buscar (1,5). Vemos que el algoritmo hace la búsqueda y se queda con el camino más corto tal como muestra la ilustración 2.

El camino está representado por las coordenadas que tiene cada posición del almacén

```
Ingrese coordenada Y del inicio (FILA) (Entrada al almacén):
0
Ingrese coordenada X del inicio (COLUMNA)(Entrada al almacén):
0
Ingrese coordenada Y de la meta (FILA) (Producto a buscar):
1
Ingrese coordenada X de la meta (COLUMNA) (Producto a buscar):
5
Nodo Inicial [ 0 ][ 0 ]

Nodo Actual [ 1 ][ 0 ]
Iteracion numero: 1
Nodo Actual [ 0 ][ 1 ]
Iteracion numero: 2
Nodo Actual [ 0 ][ 2 ]
Iteracion numero: 3
Nodo Actual [ 0 ][ 3 ]
Iteracion numero: 4
Nodo Actual [ 1 ][ 3 ]
Iteracion numero: 5
Nodo Actual [ 0 ][ 4 ]
Iteracion numero: 6
Nodo Actual [ 1 ][ 4 ]
META ALCANZADA
CAMINO MAS CORTO:
[0][0]
[0][1]
[0][2]
[0][3]
[1][3]
[1][4]
```

```
## 0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17
[" P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P"]  # 0 PASILLO
[" P", " 0", " 1", " P", " P", "24", "25", " P", " P", "48", "49", " P", " P", "72", "73", " P", " P", "96"]  # 1
[" P", " 2", " 3", " P", " P", "26", "27", " P", " P", "50", "51", " P", " P", "74", "75", " P", " P", "97"]  # 2
[" P", " 4", " 5", " P", " P", "28", "29", " P", " P", "52", "53", " P", " P", "76", "77", " P", " P", "98"]  # 3
[" P", " 6", " 7", " P", " P", "30", "31", " P", " P", "54", "55", " P", " P", "78", "79", " P", " P", "99"]  # 4
[" P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P"]  # 5 PASILLO
[" P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P"]  # 6 PASILLO
[" P", " 8", " 9", " P", " P", "32", "33", " P", " P", "56", "57", " P", " P", "80", "81", " P", " P", " P"]  # 7
[" P", "10", "11", " P", " P", "34", "35", " P", " P", "58", "59", " P", " P", "82", "83", " P", " P", " P"]  # 8
[" P", "12", "13", " P", " P", "36", "37", " P", " P", "60", "61", " P", " P", "84", "85", " P", " P", " P"]  # 9
[" P", "14", "15", " P", " P", "38", "39", " P", " P", "62", "63", " P", " P", "86", "87", " P", " P", " P"]  # 10
[" P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P"]  # 11 PASILLO
[" P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P"]  # 12 PASILLO
[" P", "16", "17", " P", " P", "40", "41", " P", " P", "64", "65", " P", " P", "88", "89", " P", " P", " P"]  # 13
[" P", "18", "19", " P", " P", "42", "43", " P", " P", "66", "67", " P", " P", "90", "91", " P", " P", " P"]  # 14
[" P", "20", "21", " P", " P", "44", "45", " P", " P", "68", "69", " P", " P", "92", "93", " P", " P", " P"]  # 15
[" P", "22", "23", " P", " P", "46", "47", " P", " P", "70", "71", " P", " P", "94", "95", " P", " P", " P"]  # 16
[" P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P"]  # 17 PASILLO
```

Ilustración 2

Problemas que tuvimos:

- En varias ocasiones se produjo la búsqueda a través de las “paredes” del almacén, una búsqueda tipo pacman. Solucionamos esto imponiendo restricciones en los bordes del almacén.
- Utilizamos una función para ordenar los valores de los costos en la lista, esto es un problema debido a que consume más recursos y hace el código más lento.

TEMPLE SIMULADO

Características del algoritmo:

- Es un algoritmo de búsqueda local
- Tiene un consumo de memoria constante (inician y terminan con el mismo consumo de memoria)
- Mantienen un único estado (o conjunto pequeño)

Ejercicio:

Dada una orden de pedido, que incluye una lista de productos del almacén anterior que deben ser despachados en su totalidad, determinar el orden óptimo para la operación de picking mediante Temple Simulado. ¿Qué otros algoritmos pueden utilizarse para esta tarea?

Implementación:

- Tenemos como objetivo buscar el menor costo entre varios puntos, es decir, debo buscar en qué orden conviene buscar los productos.
- Para cada combinación (orden para buscar los productos), debemos calcular el costo óptimo.
- Nos ayudamos del algoritmo A*, para enviarle dos posiciones determinadas y nos calcule el camino óptimo entre las posiciones.
- Iniciamos el algoritmo con una configuración aleatoria de orden de búsqueda de los productos.
- El espacio de búsqueda es n!. Donde n es la cantidad de productos.
- La cantidad de estados vecinos está conformada por (n-1)!. Se corresponden con una única permutación entre el orden de 2 productos de la configuración anterior.
- Si el nuevo estado es mejor que el anterior, lo aceptamos. Si el nuevo estado es peor que el anterior, lo vamos a aceptar con una cierta probabilidad que depende de la temperatura T (cantidad de iteraciones) y ΔE (delta de energía, que tan peor es el nuevo estado respecto del estado anterior).
- El algoritmo tiene un componente estocástico. Tiende a moverse a estados peores con mayor fuerza al principio y con menor fuerza sobre el final.
- Para hacer el afinamiento, podemos variar T y la función de enfriamiento (como decrece el valor de T). Esto puede ser lineal, logarítmico, exponencial. Otra forma de optimizar es probar como funciona el algoritmo al hacer que t varíe de forma proporcional a la cantidad de productos.

Ejecución del código:

Tenemos una lista de productos y debemos buscar el orden óptimo desde la puerta del almacén, recoger todos los productos y volver a la puerta del almacén.

3. Mapa del almacén con sus productos y una lista de productos dada por un pedido en un orden cualquiera:

```
#  0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17
Mapa.append([" P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P"]) # 0 PASILLO
Mapa.append([" P", " 0", " 1", " P", " P", "24", "25", " P", " P","48", "49", " P", " P", "72", "73", " P", " P", "96"]) # 1
Mapa.append([" P", " 2", " 3", " P", " P", "26", "27", " P", " P","50", "51", " P", " P", "74", "75", " P", " P", "97"]) # 2
Mapa.append([" P", " 4", " 5", " P", " P", "28", "29", " P", " P","52", "53", " P", " P", "76", "77", " P", " P", "98"]) # 3
Mapa.append([" P", " 6", " 7", " P", " P", "30", "31", " P", " P","54", "55", " P", " P", "78", "79", " P", " P", "99"]) # 4
Mapa.append([" P", " P", " P", " P", " P", " P", " P", " P", " P","P", " P", " P", " P", " P", " P", " P", " P", " P"]) # 5 PASILLO
Mapa.append([" P", " P", " P", " P", " P", " P", " P", " P", " P","P", " P", " P", " P", " P", " P", " P", " P", " P"]) # 6 PASILLO
Mapa.append([" P", " 8", " 9", " P", " P", "32", "33", " P", " P","56", "57", " P", " P", "80", "81", " P", " P", " P"]) # 7
Mapa.append([" P", "10", "11", " P", " P", "34", "35", " P", " P","58", "59", " P", " P", "82", "83", " P", " P", " P"]) # 8
Mapa.append([" P", "12", "13", " P", " P", "36", "37", " P", " P","60", "61", " P", " P", "84", "85", " P", " P", " P"]) # 9
Mapa.append([" P", "14", "15", " P", " P", "38", "39", " P", " P","62", "63", " P", " P", "86", "87", " P", " P", " P"]) # 10
Mapa.append([" P", " P", " P", " P", " P", " P", " P", " P", " P","P", " P", " P", " P", " P", " P", " P", " P", " P"]) # 11 PASILLO
Mapa.append([" P", " P", " P", " P", " P", " P", " P", " P", " P","P", " P", " P", " P", " P", " P", " P", " P", " P"]) # 12 PASILLO
Mapa.append([" P", "16", "17", " P", " P", "40", "41", " P", " P","64", "65", " P", " P", "88", "89", " P", " P", " P"]) # 13
Mapa.append([" P", "18", "19", " P", " P", "42", "43", " P", " P","66", "67", " P", " P", "90", "91", " P", " P", " P"]) # 14
Mapa.append([" P", "20", "21", " P", " P", "44", "45", " P", " P","68", "69", " P", " P", "92", "93", " P", " P", " P"]) # 15
Mapa.append([" P", "22", "23", " P", " P", "46", "47", " P", " P","70", "71", " P", " P", "94", "95", " P", " P", " P"]) # 16
Mapa.append([" P", " P", " P", " P", " P", " P", " P", " P", " P","P", " P", " P", " P", " P", " P", " P", " P", " P"]) # 17 PASILLO

Inicio = [0, 0]
NodosAux = crearNodos(Mapa, Inicio, [1, 1]) # LO UTILIZO PARA BUSCAR LA UBICACION DE LOS PRODUCTOS
orden = [" 2", " 9", "36", "25", "11", " 4","54","62","88","94","44","40","70","82"]
```

Para la ejecución se obtuvo el orden de picking enumerado en rojo (mejor estado):

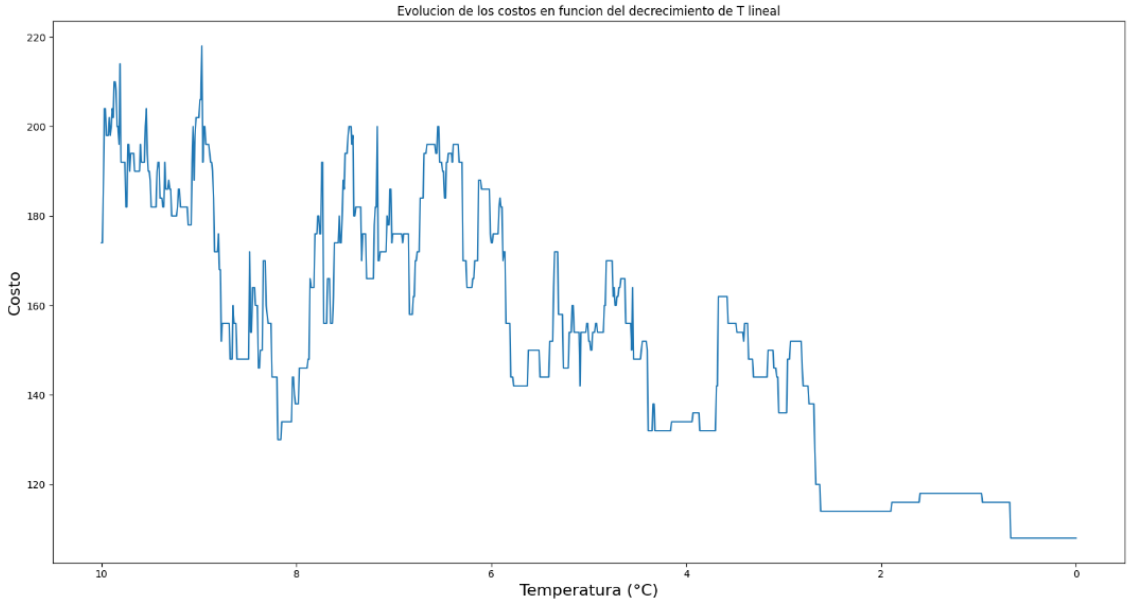
```
ESTADO Y COSTO FINAL [['25', '82', '88', '94', '70', '54', '62', '40', '44', '36', '11', ' 9', ' 2', ' 4'], 108]
MEJOR ESTADO Y COSTO ENCONTRADO [['25', '82', '88', '94', '70', '54', '62', '40', '44', '36', '11', ' 9', ' 2', ' 4'], 108]
```

```
#  0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17
Mapa.append([" P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P", " P"]) # 0 PASILLO
Mapa.append([" P", " 0", " 1", " P", " P", "24", "25", " P", " P","48", "49", " P", " P", "72", "73", " P", " P", "96"]) # 1
Mapa.append([" P", " 2", " 3", " P", " P", "26", "27", " P", " P","50", "51", " P", " P", "74", "75", " P", " P", "97"]) # 2
Mapa.append([" P", " 4", " 5", " P", " P", "28", "29", " P", " P","52", "53", " P", " P", "76", "77", " P", " P", "98"]) # 3
Mapa.append([" P", " 6", " 7", " P", " P", "30", "31", " P", " P","54", "55", " P", " P", "78", "79", " P", " P", "99"]) # 4
Mapa.append([" P", " P", " P", " P", " P", " P", " P", " P", " P","P", " P", " P", " P", " P", " P", " P", " P", " P"]) # 5 PASILLO
Mapa.append([" P", " P", " P", " P", " P", " P", " P", " P", " P","P", " P", " P", " P", " P", " P", " P", " P", " P"]) # 6 PASILLO
Mapa.append([" P", " 8", " 9", " P", " P", "32", "33", " P", " P","56", "57", " P", " P", "80", "81", " P", " P", " P"]) # 7
Mapa.append([" P", "10", "11", " P", " P", "34", "35", " P", " P","58", "59", " P", " P", "82", "83", " P", " P", " P"]) # 8
Mapa.append([" P", "12", "13", " P", " P", "36", "37", " P", " P","60", "61", " P", " P", "84", "85", " P", " P", " P"]) # 9
Mapa.append([" P", "14", "15", " P", " P", "38", "39", " P", " P","62", "63", " P", " P", "86", "87", " P", " P", " P"]) # 10
Mapa.append([" P", " P", " P", " P", " P", " P", " P", " P", " P","P", " P", " P", " P", " P", " P", " P", " P", " P"]) # 11 PASILLO
Mapa.append([" P", " P", " P", " P", " P", " P", " P", " P", " P","P", " P", " P", " P", " P", " P", " P", " P", " P"]) # 12 PASILLO
Mapa.append([" P", "16", "17", " P", " P", "40", "41", " P", " P","64", "65", " P", " P", "88", "89", " P", " P", " P"]) # 13
Mapa.append([" P", "18", "19", " P", " P", "42", "43", " P", " P","66", "67", " P", " P", "90", "91", " P", " P", " P"]) # 14
Mapa.append([" P", "20", "21", " P", " P", "44", "45", " P", " P","68", "69", " P", " P", "92", "93", " P", " P", " P"]) # 15
Mapa.append([" P", "22", "23", " P", " P", "46", "47", " P", " P","70", "71", " P", " P", "94", "95", " P", " P", " P"]) # 16
Mapa.append([" P", " P", " P", " P", " P", " P", " P", " P", " P","P", " P", " P", " P", " P", " P", " P", " P", " P"]) # 17 PASILLO
```

***4**. Representación del camino con el mejor estado encontrado*

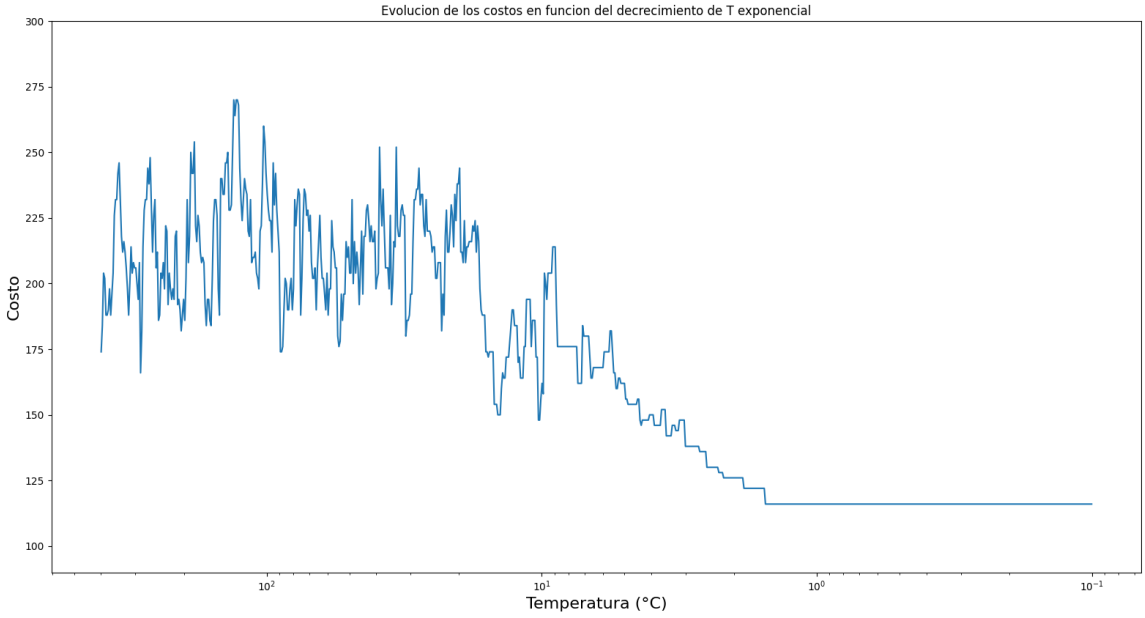
Gráficas obtenidas

Para un decrecimiento de T lineal (decrece paulatinamente la posibilidad de aceptar una peor calidad):



5. Este gráfico se corresponde con la ejecución del código del orden representado anteriormente

Para un decrecimiento de T exponencial (malo al principio y luego rápidamente encuentra una mejor calidad):



6. Esta gráfica se corresponde con otra ejecución del código la cual expresa el siguiente resultado óptimo de orden de picking:

```
ESTADO Y COSTO FINAL [['25', '82', '62', '40', '44', ' 9', '11', '36', '88', '94', '70', '54', ' 4', ' 2'], 116]
MEJOR ESTADO Y COSTO ENCONTRADO [['25', '82', '62', '40', '44', ' 9', '11', '36', '88', '94', '70', '54', ' 4', ' 2'], 116]
```

Problemas que tuvimos:

- Entre pruebas que hicimos sobre T y el paso con el que va descendiendo obtuvimos un mejor resultado del algoritmo al hacer que desde T=10 descienda de forma lineal en pasos de 0,1 a que descienda desde T=100 con pasos de 1.

ALGORITMO GENÉTICO

Características del algoritmo:

- Para este algoritmo hacemos uso de los algoritmos de temple simulado (para elegir el mejor camino entre varios productos) y del algoritmo A* (para calcular el mejor camino entre 2 posiciones). Con el algoritmo genético necesitamos evaluar la calidad de cada configuración del almacén (FITNESS)
- Utilizamos mecanismos evolutivos con permutaciones:
    - Crossover
    - Mutación
- Como parámetros de afinación podemos ir probando:
    - El tipo de mecanismo del crossover
    - La probabilidad de permutación que usamos
    - Criterios de parada
    - Formas alternativas de agregar los valores de las órdenes para calcular el fitness
    - Tamaño de la población

Ejercicio:

Implementar un algoritmo genético para resolver el problema de optimizar la ubicación de los productos en el almacén, de manera de optimizar el picking de los mismos. Considere que

- El layout del almacén está fijo (tamaño y ubicación de pasillos y estanterías), solo debe determinarse la ubicación de los productos
- Cada orden incluye un conjunto de productos que deben ser despachados en su totalidad
- El picking comienza y termina en una bahía de carga, la cual tiene ciertas coordenadas en el almacén (por generalidad, puede considerarse la bahía de carga en cualquier borde del almacén)
- El “costo” del picking es proporcional a la distancia recorrida

Implementación:

Dadas m ordenes de n productos, tenemos que buscar la ubicación óptima de los productos con la finalidad de minimizar el tiempo de picking **promedio** de las órdenes.

Inicializamos la población en forma aleatoria, es decir un orden aleatorio de la posición de cada producto en el almacén. Los productos no se pueden repetir.

Este es un problema de diseño del almacén el cual se lo hace 1 vez en mucho tiempo.

Para el cálculo de la función fitness: Usamos el algoritmo con un 1 orden, calculamos su energía/costo. Luego usamos una medida de agregación:

- Suma de todos los costos
- Promedio de todos los costos

Criterios de parada que podemos usar:

- Cantidad de iteraciones
- Tiempo de ejecución

- Convergencia
- Condición híbrida de parada

Ejecución del código:

Teniendo un listado de ordenes con sus respectivos productos:

ordenes.txt

```
Order 1
P22
P27
P29
P31
P33
P46
P47
P70
P72
P97

Order 2
P12
P20
P24
P25
```

Se generan 10 individuos con configuración aleatoria de las posiciones de todos los productos en el almacén

```
for i in range(0,10):
    productos=[]
    productosorden=[]
    for numeros in range(0,100):
        productos.append(str(numeros))  #crea una lista con productos del 0 al 99
    for i in range(0,100):
        elegido=random.choice(productos) #elige un elemento de la lista productos al azar
        productosorden.append(elegido) #guardamos ese producto en una lista auxiliar
        productos.remove(elegido) #eliminamos el producto de la lista creada en el bucle for anterior
    matriz.append(productosorden)
########### MATRIZ CONTIENE 10 ORDENAMIENTOS AL AZAR (población inicial de 10 individuos) ##########
```

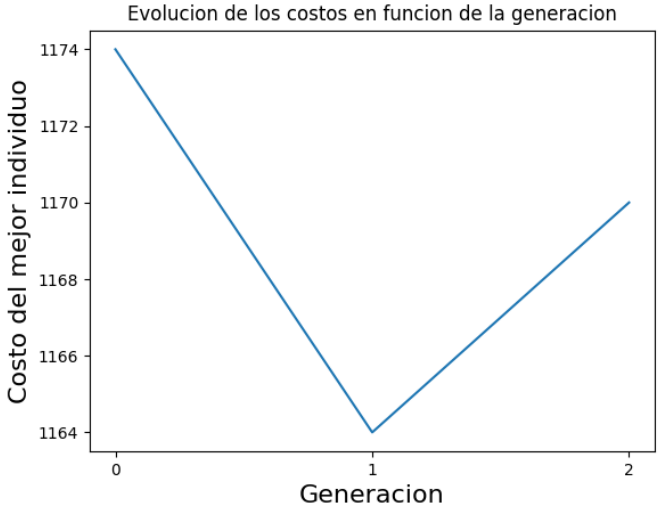
Luego vamos seleccionando individuos de la población según su fitness y creando nuevas generaciones (10 individuos más). Para finalmente quedarnos con el que tenga un mejor fitness:

```
Generacion 0 creada...
##########

Proceso de Crossover...
Generacion 1 creada...
##########

Proceso de Crossover...
Generacion 2 creada...
##########

Proceso de Crossover...
Generacion 3 creada...
El mejor ordenamiento es ['24', '71', '48', '74', '1', '50', '56', '57', '93', '32', '19', '98', '77', '38', '60', '73', '7', '30', '49', '76', '41', '69', '40', '26', '65', '95', '28', '6', '37', '90', '83', '35', '27', '50', '25', '84', '72', '5', '39',
 '29', '53', '62', '8', '94', '31', '15', '81', '43', '79', '54', '51', '78', '3', '70', '88', '23', '91', '85', '45', '47', '22', '21', '2', '87', '67', '63', '58', '61', '96', '34', '46', '64', '17', '92', '75', '0', '12', '80', '82', '55', '13', '20',
'97', '66', '10', '36', '89', '18', '68', '99', '86', '11', '33', '9', '16', '14', '44', '42', '52', '4'] y su costo es 1164
```

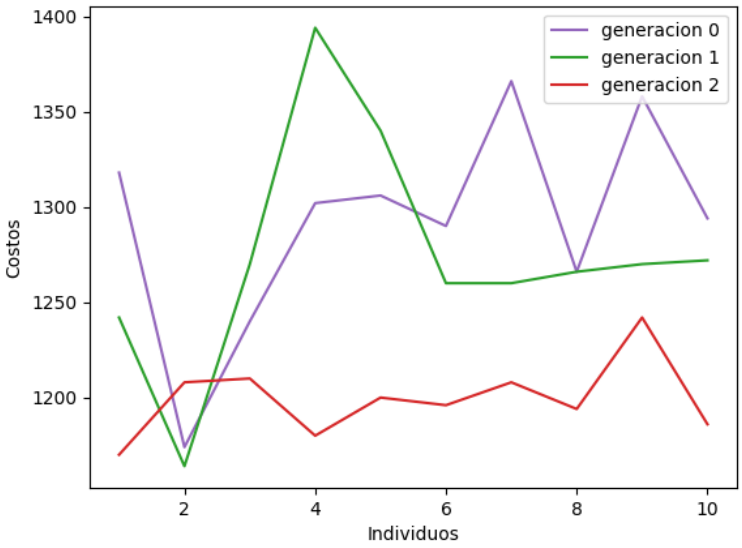
Gráficas obtenidas

```
El mejor ordenamiento es ['18', '62', '10', '98', '45', '31', '28', '55', '52', '99', '42', '40', '60', '61', '97', '39', '80', '87', '33', '22', '15', '53', '78', '51', '81', '13', '21', '64', '24', '69', '93', '92', '46', '8', '70', '82', '12', '7', '76', '47
', '63', '34', '29', '79', '89', '5', '8', '86', '30', '14', '84', '75', '73', '96', '48', '66', '38', '36', '94', '68', '56', '67', '50', '43', '74', '83', '58', '85', '3', '20', '57', '16', '65', '32', '1', '95', '90', '54', '49', '37', '77', '11', '59', '27'
, '9', '26', '17', '6', '71', '88', '72', '91', '41', '44', '23', '35', '4', '25', '19', '2'] y su costo es 1164
```

Menor costo que tiene un individuo de su población por generaciones



Costos en función de los individuos para las 3 generaciones:



Se observarían mejores conclusiones con mayores cantidades de generaciones. Pero podemos observar que a medida que pasamos a siguientes generaciones, el costo tiende a disminuir.

Problemas que tuvimos:

- La lista de m órdenes de n productos que tenemos (en nuestro caso 10 órdenes de 10 productos), no es la más adecuada para obtener una comprobación eficaz de que el orden de productos en el almacén sea el mejor debido a que no se tiene demasiada repetibilidad en ciertos productos para que estos se ubiquen más cerca de la puerta del almacén.