Universidad Autónoma de Nuevo León
Facultad de Ciencias Físico Matemáticas
San Nicolás de los Garza

IA

Tarea 3

Inteligencia Artificial

Docente: JUAN PABLO ROSAS BALDAZO

Matrícula	Nombre
1947480	FELIX YAHVEH ALANIS CANDELARIA
1963196	JUAN CARLOS DIAZ GONZALES
1962111	JOHANN JOSEPH VELÁZQUEZ ANTONIO

San Nicolás de los Garza, a 24 de febrero del 2023

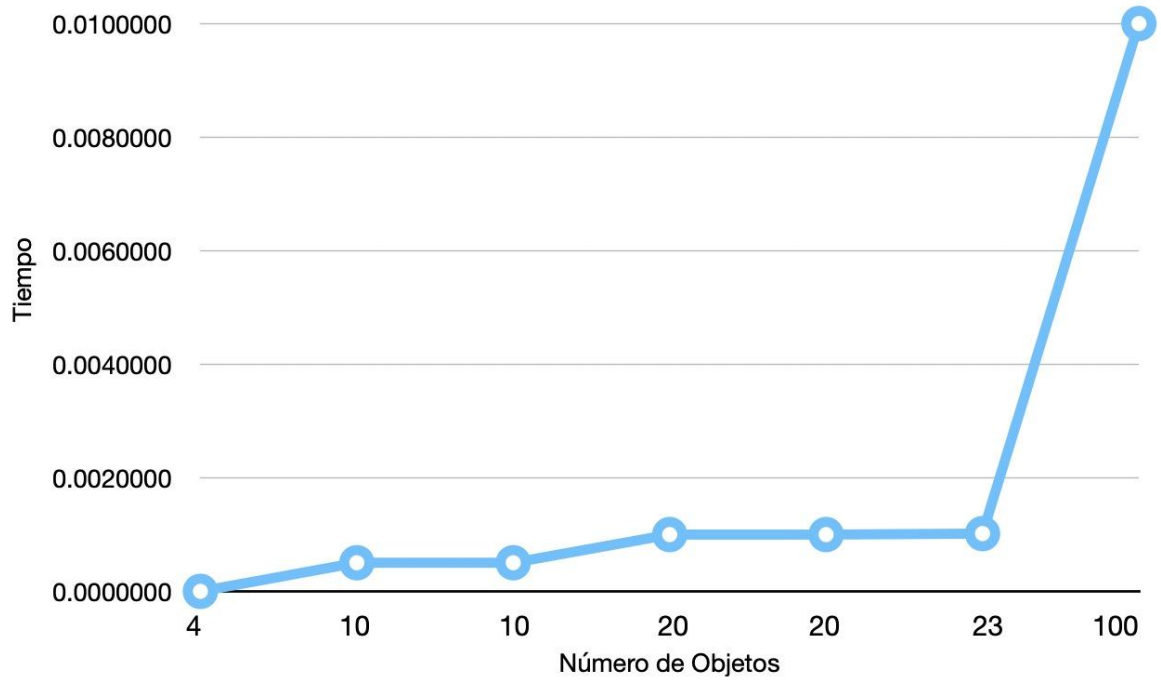
IA

GITHUB: https://github.com/Johann-28/IA

Ejercicio1:

```
items <- [    {"value": 10, "weight": 269},
          {"value": 55, "weight": 95},
          {"value": 10, "weight": 4},
          {"value": 47, "weight": 60},
          {"value": 5, "weight": 32},
          {"value": 4, "weight": 23},
          {"value": 50, "weight": 72},
           {"value": 8, "weight": 80},
          {"value": 61, "weight": 62},
          {"value": 85, "weight": 65},
           {"value": 87, "weight": 46},]

max_weight <- 300
stack <- []

Funcion knapsack_dfs(items, max_weight)
    Definir value, weight como enteros
    Definir taken_items, best_items como matriz de enteros
    value <- 0
    weight <- 0
    taken_items <- []
    best_value <- 0
    best_items <- []
    stack <- [(0, 0, [])]
    Mientras stack sea no vacío hacer
        (value, weight, taken_items) <- Desapilar(stack)
        Si weight > max_weight entonces
            Continuar
        Fin Si
        Si value > best_value entonces
            best_value <- value
            best_items <- taken_items
        Fin Si
        Para i <- 0 hasta Longitud(items)-1 hacer
            item <- items[i]
            Si i no esta en taken_items entonces
                new_value <- value + item["value"]
                new_weight <- weight + item["weight"]
                new_items <- tomados + [i]
                Apilar(stack, (new_value, new_weight, new_items))
            Fin Si
        Fin Para
    Fin Mientras
```

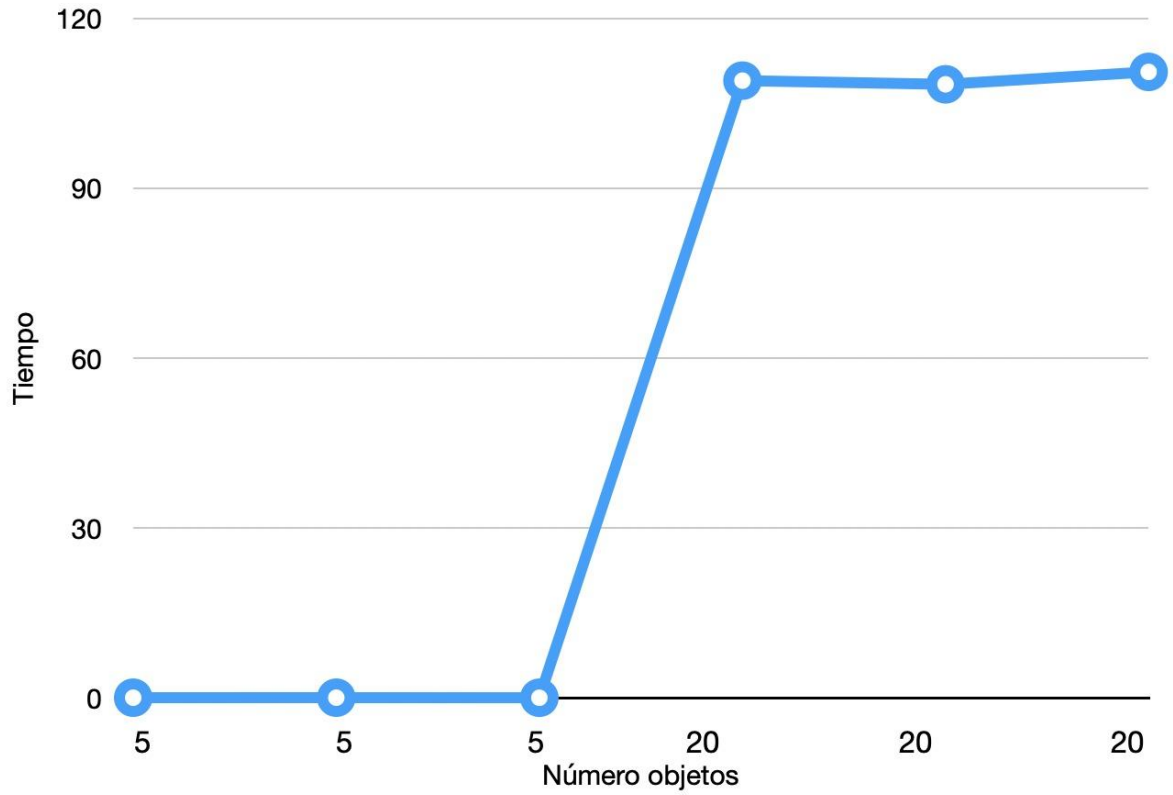
```
    Devolver (best_value, best_items)
Fin Funcion

Mejor_beneficio, Mejor_items <- knapsack_dfs(items, max_weight)

Escribir("Mejor beneficio:", Mejor_beneficio)
Escribir("Items:", Mejor_items)
```



Ejercicio2:

```
Definir clase Mochila
    Definir subproceso __init__(self, valor, peso, capacidad)
        Definir self.valor como valor
        Definir self.peso como peso
        Definir self.capacidad como capacidad
        Definir self.f como una lista vacía #Valores heurísticos
        Definir self.objetos como una lista vacía
        Definir self.beneficio como 0
    FinDefinir

    Definir subproceso calcularF(self)
        Definir valores como una lista vacía
        Para i desde 0 hasta longitud(self.valor) hacer
            Agregar (peso[i] + ConvertirAReal(valor[i])/ConvertirAReal(peso[i])) a valores
        FinPara
```

```
        Devolver valores
    FinDefinir

    Definir subproceso escogerMinimo(self)
        Definir self.f como calcularF()
        Definir min como f[0]
        Definir index como 0
        Para i desde 0 hasta longitud(self.f) hacer
            Si self.f[i] <= min entonces
                Definir min como self.f[i]
                Definir index como i
            FinSi
        FinPara
        self.capacidad -= self.peso[index]
        Si self.capacidad < 0 entonces
            Devolver
        FinSi
        Agregar valor[index] a self.objetos
        Quitar self.f[index] de self.f
        Definir self.beneficio como self.beneficio + self.valor[index]
        Quitar valor[index] de self.valor
        Quitar peso[index] de self.peso
    FinDefinir

    Definir subproceso aEstrella(self)
        Mientras self.capacidad >= 0 hacer
            escogerMinimo()
        FinMientras
    FinDefinir

FinDefinir
```



> Las conclusiones satisafcen la hipótesis de que según crecía el numero de objetos el tiempo de computo sería mayor

> Se realizaron 3 iteraciones en cada configuración de grafo, y se graficó el promedio utilizado.
> Todas las pruebas fueron echas bajo el mismo ambiente del IDE Visual Studio Code.
> Las características del equipo en que se corrió fueron las siguientes:
> SO: Windows 11 Home Single Language x64
> Procesador: AMD Ryzen 5 5600H with Radeon Graphics

> 3.30 GHz
> RAM: 16,0 GB

En las instancias dadas no se presentaba una capacidad de la mochila por lo que no se pudo tener una comparación bajo las mismas condiciones, pero se buscó establecer una capacidad proporcional a la cantidad de objetos y al peso de cada uno, la grafica nos muestra que a mayor numero de objetos más tiempo(en segundos) toma, lo que era de esperarse viendo la naturaleza de los algoritmos.