Algoritmo para buscar la mejor opción de hardware para un juego

Integrantes:
Axel Gutierrez Vazquez
Axxel Gael Hernandez Macias
Laura Vanessa Quintero Arreola

El siguiente algoritmo tiene como finalidad ayudar a un usuario a encontrar la mejor opción de hardware en función de un videojuego y su presupuesto

El algoritmo resulta bastante eficaz, ya que logra resolver una tarea compleja con un número reducido de líneas de código. Esta eficiencia no se debe únicamente a la lógica implementada, sino también a la forma en que está estructurado. Al estar basado en API's de servidores, se aprovechan recursos externos que permiten reducir significativamente la cantidad de código necesario, evitando repeticiones y simplificando el desarrollo. Gracias a este enfoque, se consigue un sistema más compacto, y fácil de implementar en códigos complejos.

```python
import requests #necesario par importar urls
import re #se utiliza para extraer cpu, gpu y ram

def LimpiarHTML(texto_html): #Funcion que elimina las etiquetas HTML
    # Reemplaza etiquetas de lista <li> por un guion y un salto de linea para formatear
    texto = texto_html.replace('<li>', '\n- ').replace('<br>', '\n')
    # Usa una expresión regular para encontrar y eliminar cualquier otra etiqueta HTML (<...>)
    limpio = re.sub(r'<.*?>', '', texto)
    # Elimina espacios o líneas en blanco al inicio y al final
    return limpio.strip()

def LocalizarJuego(nombre): #Funcion para buscar los juegos, pasamos el parametro
    url = "https://store.steampowered.com/api/storesearch/" #Api de steam que nos permite obtener informacion de los juegos
    parametros = {"term": nombre, "cc": "mx", "l": "spanish"} #parametros a utilizar, term es lo que inserta el ususario, cc - es la region, y l es el idioma de la respuesta
    #request pide los datos a un servidor, en este caso a la "url" que le pasamos que seria el api de steam definida previamente
    r = requests.get(url, params=parametros, timeout=10) #esto convierte la url a algo como (url)../?term=Portal&cc=mx&l=spanish
    data = r.json() #esto convierte la respuesta a un formato JSON aceptable para python
    if data.get("total", 0) == 0: #Si el resultado indica que no encontro ningun juego
        return None #devuelve None
    item = data["items"][0] #En caso de encotnrar el juego
    return {"appid": item["id"], "name": item["name"]} #regresa  el "id" que steam usa para identificar el juego y el nombre(name)

def ObtenerDetalles(appid): #funcion para obtener los detalles mediante el id del juego
    url = f"https://store.steampowered.com/api/appdetails/" #Api de steam que permite obtener detalles de un juego
    parametros = {"appids": appid, "cc": "mx", "l": "spanish"} #los parametros a utilizar son el id del juego, region, y lenguaje (esp)
    r = requests.get(url, params= parametros, timeout=10) #le pide los datos al servidor especificado
    data = r.json() #convierte los datos a formato json

    if not data.get(str(appid), {}).get("success", False): #si la api no da detalles
        return None  #regresa none

    detalles = data[str(appid)]["data"] #obtiene la informacion del juego del json
    nombre = detalles.get("name", "Desconocido") #obtiene el nombre del juego de detalles
    precio = detalles.get("price_overview", {}) #obtiene el precio del juego si no existe retorna un diccionario vacio
    if precio: #si existe
        precio_str = f'{precio.get("final_formatted", "desconocido")}' #extrae el precio en formato con moneda y si no existe imprime "desconocido"
    else: #si no hay informacion del precio
        precio_str = "Gratis / no disponible" #imprime que el juego es gratis o no se encontro el precio

```

```
39
40      plataformas = detalles.get("platforms", {}) #obtiene la informacion de las plataformas para las que esta disponible
41      plataformas_str = ", ".join([k for k,v in plataformas.items() if v]) #convierte el directorio en una cadena separada por comas
42
43      requisitos_min_html = detalles.get("pc_requirements", {}).get("minimum", "no disponible") #obtiene de detalles los requisitos minimos a traves de <pc_requirements>
44      requisitos_rec_html = detalles.get("pc_requirements", {}).get("recommended", "No disponible") #obtiene de detalles los requisitos recomendados a traves de <pc_requirements>
45
46      cpu_min, gpu_min, ram_min = ObtenerRequisitos(requisitos_min_html) #extrae cpu, gpu y ram de los requisitos minimos
47      cpu_rec, gpu_rec, ram_rec = ObtenerRequisitos(requisitos_rec_html) #extrae cpu, gpu y ram de los requisitos recomendados
48
49      return { #devuelve un directorio con toda la informacion que se va a necesitar
50          "nombre": nombre,
51          "precio": precio_str,
52          "plataformas": plataformas_str,
53          "requisitos_minimos": LimpiarHTML(requisitos_min_html),
54          "requisitos_recomentados": LimpiarHTML(requisitos_rec_html),
55          "cpu_min": cpu_min,
56          "gpu_min": gpu_min,
57          "ram_min": ram_min,
58          "cpu_rec": cpu_rec,
59          "gpu_rec": gpu_rec,
60          "ram_rec": ram_rec
61      }
62
63  def ObtenerRequisitos(texto): #funcion para obtener el gpu, cpu y ram de un juego
64      cpu = None
65      gpu = None
66      ram = None
67
68      if texto != "no disponible" and texto != "No disponible": #si hay texto de requisitos
69          cpu_match = re.search(r"CPU:\s*([^\n]+)", texto) #Busca la linea que contiene la CPU y extrae su nombre
70          gpu_match = re.search(r"(?:GPU|Tarjeta grafica|Graphics):\s*([^\n]+)", texto) #Busca la linea que contiene la GPU y extrae su nombre
71          ram_match = re.search(r"RAM:\s*([^\n]+)", texto) #Busca la linea que contiene la RAM y extrae la cantidad
72          if cpu_match:
73              cpu = cpu_match.group(1).strip() #Si encontro CPU, guarda el texto limpio sin espacios
74          if gpu_match:
75              gpu = gpu_match.group(1).strip() #Si encontro GPU, guarda el texto
76          if ram_match:
```

```python
def ObtenerRequisitos(texto): #funcion para obtener el gpu, cpu y ram de un juego
        if ram_match:
            ram = ram_match.group(1).strip() #Si encontro RAM, guarda el texto
    return cpu, gpu, ram #regresa los daatos obtenidos

def CargarComponentes(): #devuelve listas de cpus, gpus y precios para ram en MXN
    # cada componente: nombre, benchmark (score), precio_mxn
    cpus = [
        {"name":"Intel Core i5-2500","score":5200,"price":1200},
        {"name":"Intel Core i7-6700","score":8400,"price":2500},
        {"name":"AMD Ryzen 5 3600","score":13000,"price":4000},
        {"name":"AMD Ryzen 5 5600","score":18000,"price":6000},
        {"name":"Intel Core i5-10400","score":11000,"price":2200}
    ]
    gpus = [
        {"name":"NVIDIA GeForce GTX 970","score":6700,"price":2000},
        {"name":"NVIDIA GeForce GTX 1060","score":8000,"price":3200},
        {"name":"NVIDIA GeForce RTX 2060","score":15000,"price":5000},
        {"name":"NVIDIA GeForce RTX 3060","score":22000,"price":9000},
        {"name":"NVIDIA GeForce GTX 1650","score":6000,"price":2200}
    ]
    ram_options = {8:800, 16:1500, 32:3000} #precio aproximado en MXN
    return cpus, gpus, ram_options

def BuscarComponentes(lista, nombre): #busca componente por coincidencia simple
    if not nombre:
        return None
    nombre_l = nombre.lower()
    mejores = []
    for c in lista:
        n = c["name"].lower()
        if nombre_l in n or n in nombre_l: #coincidencia por substring
            mejores.append(c)
    if mejores:
        # elegir el de mayor score entre coincidencias
        mejores.sort(key=lambda x: x["score"], reverse=True)
        return mejores[0]
    return None
```

```python
def Build(presupuesto, cpu_req_name, gpu_req_name): #seleccion por fuerza bruta
    cpus, gpus, ram_options = CargarComponentes() #datos locales de ejemplo
    cpu_obj = BuscarComponentes(cpus, cpu_req_name) #mapear requisito cpu a componente
    gpu_obj = BuscarComponentes(gpus, gpu_req_name) #mapear requisito gpu a componente
    cpu_target = cpu_obj["score"] if cpu_obj else 0
    gpu_target = gpu_obj["score"] if gpu_obj else 0

    mejores = None
    mejor_val = -1

    # iterar todas las combinaciones CPU x GPU x RAM
    for cpu in cpus:
        for gpu in gpus:
            for ram_size, ram_price in ram_options.items():
                total_price = cpu["price"] + gpu["price"] + ram_price
                if total_price > presupuesto: #descartar combos fuera de presupuesto
                    continue
                # calcular metrica: ponderar GPU mas que CPU y RAM simple
                score = (0.6 * gpu["score"]) + (0.3 * cpu["score"]) + (0.1 * (ram_size * 1000))
                # normalizar por precio para calidad/precio
                val = score / max(1, total_price)
                # preferir combos que al menos cumplan objetivo (si hay objetivo)
                cumple_objetivo = (cpu["score"] >= cpu_target) and (gpu["score"] >= gpu_target)
                # si cumple objetivo, darle ventaja
                if cumple_objetivo:
                    val *= 1.15
                if val > mejor_val:
                    mejor_val = val
                    mejores = {"cpu": cpu, "gpu": gpu, "ram": {"size": ram_size, "price": ram_price}, "total": total_price}

    # si no encontro ninguna combinacion dentro del presupuesto, devolver el mas barato posible (fallback)
    if not mejores:
        # elegir componentes mas baratos
        cpus_sorted = sorted(cpus, key=lambda x: x["price"])
        gpus_sorted = sorted(gpus, key=lambda x: x["price"])
        min_ram = min(ram_options.items(), key=lambda x: x[1])
        cpu_choice = cpus_sorted[0] if cpus_sorted else None
        gpu_choice = gpus_sorted[0] if gpus_sorted else None
        ram_choice = {"size": min_ram[0], "price": min_ram[1]}
```

```python
def Build(presupuesto, cpu_req_name, gpu_req_name): #seleccion por fuerza bruta
        ram_choice = {"size": min_ram[0], "price": min_ram[1]}
        total_price = (cpu_choice["price"] if cpu_choice else 0) + (gpu_choice["price"] if gpu_choice else 0) + ram_choice["price"]
        mejores = {"cpu": cpu_choice, "gpu": gpu_choice, "ram": ram_choice, "total": total_price}

    return mejores

# ============main======= ( `-_ ›-)
if __name__=="__main__":
    # pedir juego y presupuesto al inicio
    nombre = input("Escribe el nombre del juego\n->:") #la variable nombre guardara los datos ingresados por el usuario atraves de la peticion del input
    pres_raw = input("Ingresa tu presupuesto aproximado en MXN (ej: 6000)\n->:") #pedir presupuesto antes de buscar
    try:
        presupuesto = int(float(pres_raw))
    except:
        presupuesto = 6000 #fallback si el usuario mete algo raro

    res = LocalizarJuego(nombre) #crea la variable res de respuesta

    if res: #si res tiene datos imprime el nombre y el id del juego
        print("\nJuego --> ", res["name"])
        print("id del juego: ", res["appid"])

        detalles = ObtenerDetalles(res["appid"]) #obtiene los detalles del juego usando el appid
        if detalles: #si detalles tiene datos imprime toda la informacion
            print("\nPrecio:", detalles["precio"])
            print("Plataformas:", detalles["plataformas"])
            print("Requisitos minimos:\n", detalles["requisitos_minimos"])
            print("CPU minimo:", detalles["cpu_min"] or "No disponible")
            print("GPU minima:", detalles["gpu_min"] or "No disponible")
            print("RAM minima:", detalles["ram_min"] or "No disponible")
            print("Requisitos recomendados:\n", detalles["requisitos_recomentados"])
            print("CPU recomendada:", detalles["cpu_rec"] or "No disponible")
            print("GPU recomendada:", detalles["gpu_rec"] or "No disponible")
            print("RAM recomendada:", detalles["ram_rec"] or "No disponible")
            # recomendar por fuerza bruta
            print("\nBuild sugerida (MXN):", presupuesto)
            resultado = Build(presupuesto, detalles["cpu_rec"], detalles["gpu_rec"])
            print("=====- Recomendacion-====")
```

```
            if resultado["gpu"]:
                print("GPU:", resultado["gpu"]["name"], "| price MXN:", resultado["gpu"]["price"])
            else:
                print("GPU: No disponible")
            if resultado["cpu"]:
                print("CPU:", resultado["cpu"]["name"], "| price MXN:", resultado["cpu"]["price"])
            else:
                print("CPU: No disponible")
            print("RAM:", f'{resultado["ram"]["size"]}GB', "| price MXN:", resultado["ram"]["price"])
            print("Precio total aprox MXN:", resultado["total"])
        else:
            print("No se pudieron obtener los detalles del juego.")

    else: #si res no tiene datos, imprime un mensaje que indica que no encontro nada
        print("no se encontro el juego ( ´◠`)")

```

```
Escribe el nombre del juego
->:bioshock
Ingresa tu presupuesto aproximado en MXN (ej: 6000)
->:4036

Juego -->  BioShock Infinite
id del juego:  8870

Precio: Mex$ 269.99
Plataformas: windows, mac, linux
Requisitos minimos:
 Minimum:

- OS *: Windows Vista Service Pack 2 32-bit

- Processor: Intel Core 2 DUO 2.4 GHz / AMD Athlon X2 2.7 GHz

- Memory: 2GB

- Hard Disk Space: 20 GB free

- Video Card: DirectX10 Compatible ATI Radeon HD 3870 / NVIDIA 8800 GT / Intel HD 3000 Integrated Graphics

- Video Card Memory: 512 MB

- Sound: DirectX Compatible
CPU minimo: No disponible
GPU minima: No disponible
RAM minima: No disponible
```

```
Requisitos recomendados:
 Recommended:

- OS *: Windows 7 Service Pack 1 64-bit

- Processor: Quad Core Processor

- Memory: 4GB

- Hard Disk Space: 30 GB free

- Video Card: DirectX11 Compatible, AMD Radeon HD 6950 / NVIDIA GeForce GTX 560

- Video Card Memory: 1024 MB

- Sound: DirectX Compatible
CPU recomendada: No disponible
GPU recomendada: No disponible
RAM recomendada: No disponible

Build sugerida (MXN): 4036
=====- Recomendacion-====
GPU: NVIDIA GeForce GTX 970 | price MXN: 2000
CPU: Intel Core i5-2500 | price MXN: 1200
RAM: 8GB | price MXN: 800
Precio total aprox MXN: 4000
PS C:\Users\axelg>
```

Aplicaciones en la Vida Real

Puede ser útil para usuarios que realmente no conozcan mucho sobre componentes de una computadora, pero quieran ajustar su experiencia de juego a que sea lo más cómoda posible.

También puede ser útil para que las empresas generen presupuestos a la hora de hacer computadoras con menores o mayores capacidades.

Conclusión

Consideramos que el algoritmo es muy básico para contemplarlo como una opción viable, pues es muy limitado, pero aunque limitado se puede emplear para una persona que ocupe de muy poca información, de ahí en más si tiene la posibilidad seria mejor asesorarse de un profesional, aunque de igual forma puede darle una pauta al usuario como estan los precios. La principal limitación del algoritmo es que para actualizarse ocupa manteniendo directo en el código.