ALGORITMO DE BUSQUEDA DE CONTRASEÑAS CON DIVIDE Y VENCERÁS

EQUIPO: LOS SIN GPU

INTEGRANTES:

Jorge Daniel Hernández Reyes 220027797

Braulio Hurtado Escoto 220426225

Juan Pablo Solís Regin 220468416

DESCRIPCIÓN

Nuestro programa se enfoca en la busqueda de contraseñas con un diccionario predefinido con algunas contraseñas comunes, pero añadimos nuevos elementos. Estas nuevas mejoras a nuestro codigo hace la implementación de Divide y vencerás para hacerlo más eficiente.

Nos permite utilizar múltiples núcleos de CPU de forma efectiva, acelerando significativamente la búsqueda en comparación con una ejecución secuencial.

Al encontrar una contraseña, se registra y la búsqueda puede continuar para otros objetivos, o se puede detener completamente si se alcanza un tope de intentos o tiempo.

DEL ALGORITMO

IMPORTANCIA

Este algoritmo se centra especialmente en el ambito de la ciberseguridad, donde se puede tomar como un ejemplo práctico y educativo de cómo aplicar patrones de programación concurrente y paralela para resolver problemas intensivos en cómputo.

Tambien ayuda a los desarrolladores y usuarios a comprender la vulnerabilidad de las contraseñas cortas. Incluso con hardware modesto y un método bien paralelizado, una contraseña de, por ejemplo, 4 o 5 caracteres de complejidad media puede romperse en segundos.

```
69      def _chunk_worker(self, targets_set, charset, max_len, prefix, start_time, time_budget, attempts_counter, at
70          if len(prefix) > max_len:
71              return
72          for L in range(len(prefix), max_len + 1):
73              if self.is_stopped():
74                  return
75              if time_budget is not None and (time.time() - start_time) >= time_budget:
76                  return
77              if L == len(prefix):
78                  candidate = prefix
79                  with attempts_counter.get_lock():
80                      attempts_counter.value += 1
81                      attempts = attempts_counter.value
82                  if attempts % 10000 == 0:
```

_chunk_worker(self, targets_set, charset, max_len, prefix, ...)

Esta es la función que representa al Worker (hilo o tarea) individual en el patrón "Divide y Vencerás".

Se le asigna una porción del espacio de búsqueda definida por un prefijo (prefix) precalculado (cuya longitud es partition_len).

Su trabajo es generar y probar todas las combinaciones posibles que comiencen con ese prefijo, desde la longitud del prefijo hasta max_len.

```python
    def brute_force_divide_and_conquer(self, targets, charset, max_len, partition_len=1, time_budget=None, max_a
        self._stop_event.clear()
        self._found = {}
        start = time.time()
        targets_set = set(targets)

        attempts_counter = multiprocessing.Value('L', 0)
```

brute_force_divide_and_conquer(self, targets, charset, max_len, partition_len=1, ...)

Calcula Particiones

Inicializa Sincronización

Distribuye el Trabajo

Gestiona el Tiempo

Recolección de Resultados

```python
def start_benchmark(self):
    charset_input = self.entry_charset.get().strip()
    if charset_input:
        charset = list(dict.fromkeys(charset_input))
    else:
        charset = list(string.ascii_lowercase + string.digits)
    try:
        maxlen = int(self.spin_maxlen.get())
        cap = int(self.spin_cap.get())
        partlen = int(self.spin_partlen.get())
        workers = int(self.spin_workers.get())
    except ValueError:
        messagebox.showerror("Error", "Parámetros inválidos.")
        return
```

```python
def _run_benchmark(self, charset, maxlen, partlen, time_budget, cap, workers_param):
    lengths = []
    times = []
    for L in range(1, maxlen + 1):
        if self.worker.is_stopped():
            self.append_log("[*] Benchmark detenido por usuario.")
            break
        target = ''.join([charset[-1]] * L)
        self.append_log(f"[Benchmark] L={L}: probando objetivo='{target}' (peor caso)...")
        start = time.time()
        res = self.worker.brute_force_divide_and_conquer([target], charset, max_len=L, partition_len=partler
        elapsed = time.time() - start
        found = target in res
        lengths.append(L)
        times.append(elapsed)
        if found:
            candidate, attempts, worker_elapsed = res[target]
```

start_benchmark y _run_benchmark

- Ejecutan la fuerza bruta con "Divide y Vencerás" de manera iterativa, probando una contraseña de peor caso para cada longitud (L) desde 1 hasta max_len.
- Miden el tiempo que tarda el algoritmo en encontrar la contraseña para cada longitud.
- Genera una gráfica que ilustra cómo el tiempo de cracking aumenta exponencialmente con la longitud de la contraseña

APLICACIONES

Nuestro codigo se utiliza principalmente en ciberseguridad, teniendo las siguientes aplicaciones:

- **Auditoría de Seguridad (Pruebas de Penetración):** Se utiliza para evaluar la fortaleza de las contraseñas de un sistema.
- **Recuperación de Datos:** Puede ser empleado para recuperar contraseñas de archivos cifrados o carteras digitales (wallets) cuando la contraseña original se ha olvidado
- **Investigación Criptográfica:** El principio de dividir el trabajo se aplica para medir la resistencia de diferentes algoritmos de hashing y cifrado ante ataques de fuerza bruta distribuidos.

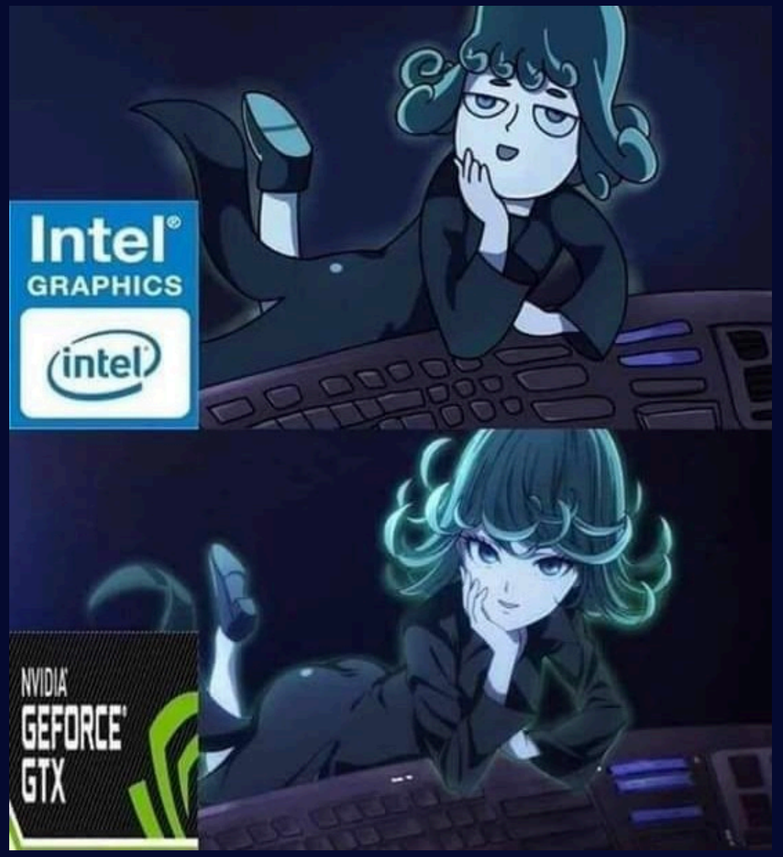
OTRAS APLICACIONES

- **Computación Científica:** Resolver grandes sistemas de ecuaciones o simulaciones físicas (por ejemplo, dividiendo un área geográfica en cuadrículas y asignando el cálculo a diferentes núcleos).
- **Búsqueda en Espacios Grandes:** Problemas como el "Viajero" o la búsqueda de soluciones en inteligencia artificial (IA).
- **Procesamiento de Datos Masivos:** Dividir un gran conjunto de datos para que múltiples máquinas o núcleos lo procesen simultáneamente (como en MapReduce).

MEJORAS

Conocemos que el codigo podría evolucionar y ser mucho más eficiente y mejor con algunas mejoras como las siguientes:

- Uso de GPU (Unidades de Procesamiento Gráfico): En vez de usar hilos, implementar el uso de librerías que aprovechen la GPU de una computadora (si la tiene, claro).
- Implementación de Ataques Basados en Hashing: en lugar de comparar la contraseña candidata directamente, calcule el hash de la candidata y lo compare con una lista de hashes objetivo (simulando un ataque a un archivo shadow o base de datos de contraseñas).



CONCLUSIÓN

Nuestro codigo original se basaba en un ataque de fuerza bruta secuencial , donde un único hilo probaba las combinaciones una tras otra. Con la implementación de Divide y vencerás el espacio de búsqueda se particiona lógicamente usando prefijos, y el trabajo se distribuye entre múltiples hilos, logrando una aceleración casi lineal en el rendimiento al aprovechar los múltiples núcleos de la CPU. Nuestro codigo nos ayuda a entender las vulnerabilidades que pueden existir en nuestras propias contraseñas, y llendonos más lejos, en las vulnerabilidades de seguridad en empresas, escuelas y más. Con los recursos necesarios, este codigo puede aspirar a temas de hacking o detección de vulnerabilidades de manera oportuna.

GRACIAS