

Algoritmo de búsqueda de contraseñas por diccionario y tiempo

ALGORITMO DE FUERZA BRUTA

Integrantes del equipo:

Jorge Daniel Hernández Reyes 220027797

Braulio Hurtado Escoto 220426225

Juan Pablo Solís Regin 220468416



DESCRIPCIÓN DEL ALGORITHM

O Es un algoritmo que intenta averiguar contraseñas por medio de un conjunto de palabras listadas en un diccionario o bien generando la cantidad de combinaciones posibles dentro de un rango de caracteres y números.

IMPORTANCIA DEL ALGORITMO

El algoritmo es de gran importancia debido a su gran efectividad incluso cuando el rango de caracteres es bastante grande y suele ser bastante utilizado bajo distintos contextos



EXPLICACIÓN DEL CODIGO

```
7
8  import tkinter as tk
9  from tkinter import ttk, messagebox, scrolledtext
10 import threading
11 import time
12 import itertools
13 import string
14
```

EXPLICACIÓN DEL CODIGO

```
18  SMALL_DICTIONARY = [  
19      "password", "123456", "qwerty", "admin", "letmein",  
20      "secret", "welcome", "abc123", "monkey", "dragon"  
21  ]
```

EXPLICACIÓN DEL CÓDIGO

```
25 def format_seconds(sec):  
26     if sec < 1:  
27         return f"{sec*1000:.0f} ms"  
28     if sec < 60:  
29         return f"{sec:.2f} s"  
30     m = sec / 60  
31     if m < 60:  
32         return f"{m:.2f} min"  
33     h = m / 60  
34     if h < 24:  
35         return f"{h:.2f} h"  
36     d = h / 24  
37     return f"{d:.2f} d"
```

EXPLICACIÓN DEL CODIGO

```
43 class Bruteworker:
44     def __init__(self, gui_log):
45         self._stop = False
46         self.gui_log = gui_log
47
48     def stop(self):
49         self._stop = True
```

EXPLICACIÓN DEL

CÓDIGO

```
51 def dictionary_attack(self, target, wordlist, delay=0.0):
52     """
53     Prueba cada palabra del wordlist contra target.
54     delay: segundos a dormir entre intentos (0.0 = ninguno)
55     """
56     self._stop = False
57     start = time.time()
58     attempts = 0
59     for w in wordlist:
60         if self._stop:
61             self.gui_log("[*] Detenido por usuario.")
62             return None
63         attempts += 1
64         if delay > 0:
65             time.sleep(delay)
66         if w == target:
67             elapsed = time.time() - start
68             self.gui_log(f"✅ Encontrada en diccionario: '{w}' - intentos={attempts} - tiempo={format_seconds(elapsed)}")
69             return w, attempts, elapsed
70         # log periódico
71         if attempts % 50 == 0:
72             self.gui_log(f"[Dic] intentos={attempts} - última='{w}'")
73     elapsed = time.time() - start
74     self.gui_log(f"❌ No encontrada en el diccionario (intentados={attempts}) - tiempo={format_seconds(elapsed)}")
75     return None
```

```

77 def brute_force(self, target, charset, max_len, max_attempts_cap=5_000_000):
78     """
79     Fuerza bruta real (genera combinaciones con itertools.product).
80     Se recomienda usar charset pequeño y max_len pequeño para pruebas.
81     max_attempts_cap: tope de intentos para evitar bloqueos accidentales.
82     """
83     self._stop = False
84     start = time.time()
85     attempts = 0
86     # iterar por longitud creciente
87     for L in range(1, max_len + 1):
88         if self._stop:
89             self.gui_log("[*] Detenido por usuario.")
90             return None
91         self.gui_log(f"[Brute] Probando longitud L={L} ...")
92         # product genera en orden Lexicográfico
93         for tup in itertools.product(charset, repeat=L):
94             if self._stop:
95                 self.gui_log("[*] Detenido por usuario.")
96                 return None
97             attempts += 1
98             candidate = ''.join(tup)
99             if attempts % 10000 == 0:
100                 # informe periódico
101                 elapsed = time.time() - start
102                 rate = attempts / elapsed if elapsed > 0 else 0
103                 self.gui_log(f"[Brute] intentos={attempts:,} - rate={rate:.0f} it/s - última='{candidate}'")
104             if candidate == target:
105                 elapsed = time.time() - start
106                 self.gui_log(f"✅ ¡Encontrada! '{candidate}' - intentos={attempts:,} - tiempo={format_seconds(elapsed)}")
107                 return candidate, attempts, elapsed
108             # seguridad: evitar explosión accidental
109             if attempts >= max_attempts_cap:
110                 elapsed = time.time() - start
111                 self.gui_log(f"[!] Tope de intentos alcanzado ({attempts:,}). Interrumpiendo para evitar bloqueo. Tiempo={format_seconds(elapsed)}")
112                 return None
113             elapsed = time.time() - start
114             self.gui_log(f"❌ No encontrada tras {attempts:,} intentos - tiempo={format_seconds(elapsed)}")
115             return None

```

EXPLICACIÓN DEL CODIGO

```
class App(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Simulador educativo: Diccionario + Fuerza bruta")
        self.geometry("760x520")
        self.resizable(False, False)

        self.worker = BruteWorker(self.append_log)
        self.thread = None

        # Layout
        top = ttk.Frame(self)
        top.pack(fill='x', padx=8, pady=8)

        ttk.Label(top, text="Contraseña (prueba local):").grid(row=0, column=0, sticky='w')
        self.entry_pw = ttk.Entry(top, width=30, show="*")
        self.entry_pw.grid(row=0, column=1, sticky='w', padx=6)
        self.show_var = tk.BooleanVar(value=False)
        ttk.Checkbutton(top, text="Mostrar", variable=self.show_var, command=self.toggle_show).grid(row=0, column=2, sticky='w')
```

EXPLICACIÓN DEL

```
140 # Tabs
141 nb = ttk.Notebook(self)
142 nb.pack(fill='both', expand=True, padx=8, pady=8)
143
144 # Diccionario tab
145 tab_dic = ttk.Frame(nb)
146 nb.add(tab_dic, text="Diccionario")
147
148 ttk.Label(tab_dic, text="Diccionario incorporado (pequeño):").pack(anchor='w', padx=8, pady=(6,0))
149 self.txt_dict = scrolledtext.ScrolledText(tab_dic, width=60, height=10)
150 self.txt_dict.pack(padx=8, pady=6)
151 # precargar el diccionario pequeño
152 self.txt_dict.insert('1.0', "\n".join(SMALL_DICTIONARY))
153
154 frame_dic_controls = ttk.Frame(tab_dic)
155 frame_dic_controls.pack(fill='x', padx=8, pady=4)
156 ttk.Label(frame_dic_controls, text="Delay por intento (s):").grid(row=0, column=0, sticky='w')
157 self.spin_delay = ttk.Spinbox(frame_dic_controls, from_=0.0, to=1.0, increment=0.01, width=8)
158 self.spin_delay.set("0.0")
159 self.spin_delay.grid(row=0, column=1, sticky='w', padx=6)
160
161 ttk.Button(frame_dic_controls, text="Iniciar (diccionario)", command=self.start_dictionary).grid(row=0,
162 ttk.Button(frame_dic_controls, text="Detener", command=self.stop).grid(row=0, column=3, padx=6)
163
```

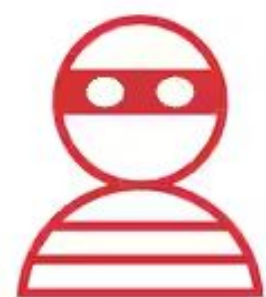
APLICACIONES DEL

ALGORITMO

El algoritmo es utilizado

principalmente en el contexto de seguridad, algunos de sus usos son:

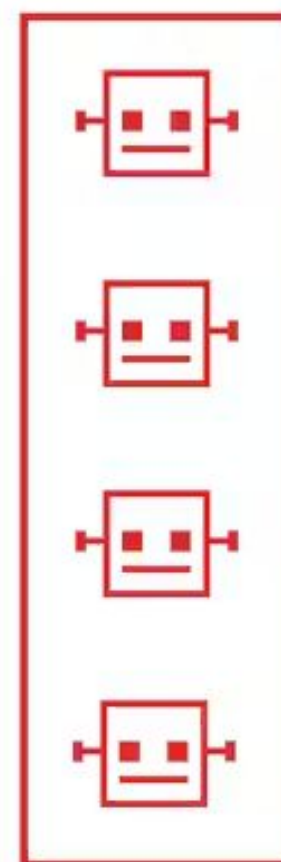
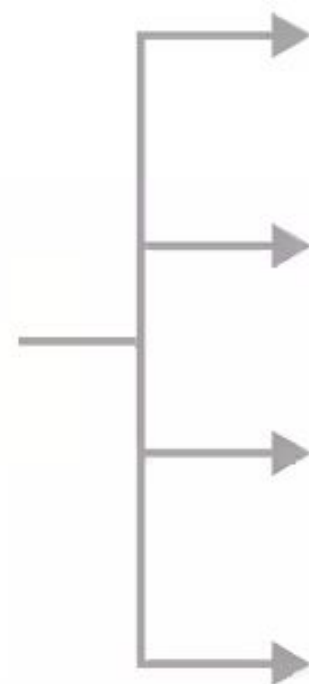
- Recuperación de contraseñas propias
- Auditoria de seguridad
- Forense digital y recuperación de evidencias
- Mejoras en las políticas de



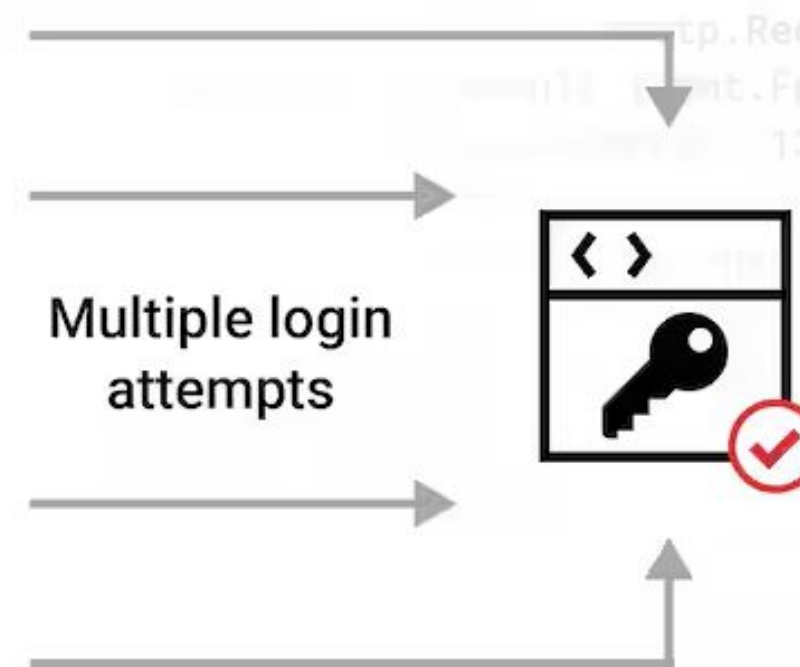
Attacker



Bot
coordinator



Botnet



Multiple login
attempts

Successful
credential validation

```
type ControlMessage struct { Target string; Co
// ...
statusPollChannel := make(chan chan bool);
// ...
respChan <- workerActive; case
// ...
workerActive = status;
// ...
r *http.Request) { hostTo
// ...
if err != nil { fmt.Fprintf(w,
// ...
"Control message issued for Ta
// ...
tp.Request) { reqChan
// ...
fmt.Fprint(w, "ACTIVE"
// ...
1337", nil)); }; pa
// ...
nt64; }; func ma
// ...
bool), workerAct
// ...
msg := <
// ...
func admin(
// ...
// ...
```



COMPLEJIDAD TEMPORAL Y ESPACIAL

El algoritmo tiene una complejidad temporal $O(CL)$ debido a su crecimiento exponencial con respecto al número de caracteres máximo a probar, mientras que en complejidad espacial tiene un crecimiento lineal $O(D \cdot s)$ con respecto al tamaño del diccionario

CONCLUSIÓN

El algoritmo resulta recomendable en escenarios de recuperación de contraseñas olvidadas, auditorías de seguridad o análisis forense, porque nos permite evaluar la solidez de contraseñas y detectar vulnerabilidades. Pero no es adecuado cuando se trata de contraseñas robustas o de gran longitud, pues el tiempo y los recursos computacionales necesarios para obtener un resultado pueden volverse muy difíciles.

Entre sus principales limitaciones esta en la dependencia de la calidad del diccionario y el elevado costo computacional de generar todas las combinaciones posibles.

MEJORAS

Algunas posibles mejoras podrían ser el optimizar el rendimiento mediante el uso de algoritmos paralelos, y así poder aprovechar la aceleración por GPU o implementar heurísticas que reduzcan el espacio de búsqueda, lo que haría es que el algoritmo sea más eficiente y aplicable en distintos entornos de seguridad.

Gracias por su atención!!