

Concurso Programacion 2

Tecnicatura Universitaria en Inteligencia Artificial

Dámian Ariel Marotte

Section 1

Fuerza Bruta

Problema

Perdida de contraseña.

Analisis

Prueba

- m8l
- 6ey7mf
- bc
- r3pe6aeo

Analisis

Prueba

- m8l
- 6ey7mf
- bc
- r3pe6aeo

Prueba

- a
- b
- c

Ingredientes

- Funcion siguiente que dado el intento actual, determina el siguiente.

Ingredientes

- Funcion `siguiente` que dado el intento actual, determina el siguiente.
- Funcion `es_solucion` que dado el intento actual, determina si es solucion.

Algoritmo

```
1  from password import es_solucion, siguiente
2
3  intento_actual = siguiente()
4
5  while not es_solucion(intento_actual):
6      intento_actual = siguiente(intento_actual)
7
8  print(intento_actual)
```


Características

Ventajas

- Es facil de implementar.
- Si existe una solucion, la encuentra.
- Si existen varias, encuentra todas.
- Si existe una solucion optima la encuentra.

Características

Ventajas

- Es facil de implementar.
- Si existe una solucion, la encuentra.
- Si existen varias, encuentra todas.
- Si existe una solucion optima la encuentra.

Desventajas

- Es muy costoso.

Características

Ventajas

- Es facil de implementar.
- Si existe una solucion, la encuentra.
- Si existen varias, encuentra todas.
- Si existe una solucion optima la encuentra.

Desventajas

- Es muy costoso.

Casos de uso

Solo tiene sentido cuando no se conoce una mejor forma de resolver el problema.

Detalles

```
1  alfabeto = "0123456789abcdefghijklmnopqrstuvwxyz"
2
3  def es_solucion(intento):
4      return intento == "abcde"
5
6  def siguiente(intento = ""):
7      if intento == alfabeto[-1] * len(intento):
8          return alfabeto[0] * (len(intento) + 1)
9      elif intento[-1] != alfabeto[-1]:
10         return intento[:-1] + alfabeto[alfabeto.find(intento[-1]) + 1]
11     else:
12         return siguiente(intento[:-1]) + alfabeto[0]
```

Section 2

Greedy

Problema

Dar el vuelto.

Analisis

Ingredientes

- Un conjunto de candidatos de donde elegir.

Ingredientes

- Un conjunto de candidatos de donde elegir.
- Funcion `elegir_candidato` que elige la mejor opcion del momento.

Ingredientes

- Un conjunto de candidatos de donde elegir.
- Funcion `elegir_candidato` que elige la mejor opcion del momento.
- Funcion `es_factible` que determina si una determinada eleccion puede llevar a la solucion.

Ingredientes

- Un conjunto de candidatos de donde elegir.
- Funcion `elegir_candidato` que elige la mejor opcion del momento.
- Funcion `es_factible` que determina si una determinada eleccion puede llevar a la solucion.
- Funcion `es_solucion` que dada la seleccion actual, determina si es una solucion al problema.

Algoritmo

```
1  from coins import es_solucion, elegir_candidato, candidatos, es_factible
2
3  eleccion_actual = []
4
5  while not es_solucion(eleccion_actual):
6      x = elegir_candidato()
7      candidatos.remove(x)
8      if es_factible(eleccion_actual + [x]):
9          eleccion_actual.append(x)
10
11 print(eleccion_actual)
```

Características

Ventajas

- Es facil de implementar.
- Si existe una solucion, la encuentra.
- Es un algoritmo rapido.
- Usa poca memoria.

Características

Ventajas

- Es facil de implementar.
- Si existe una solucion, la encuentra.
- Es un algoritmo rapido.
- Usa poca memoria.

Desventajas

- No siempre encuentra la mejor solucion.

Características

Ventajas

- Es facil de implementar.
- Si existe una solucion, la encuentra.
- Es un algoritmo rapido.
- Usa poca memoria.

Desventajas

- No siempre encuentra la mejor solucion.

Casos de uso

Podemos usar este algoritmo cuando necesitamos una buena solucion, aunque no necesariamente la mejor.

Detalles

```
1 total = 6
2 candidatos = [4] * total + [3] * total + [1] * total
3
4 def es_solucion(eleccion_actual):
5     return sum(eleccion_actual) == total
6
7 def elegir_candidato():
8     return max(candidatos)
9
10 def es_factible(eleccion):
11     return sum(eleccion) <= total
```


Section 3

Divide & Conquer

Problema

Ordenar numeros.

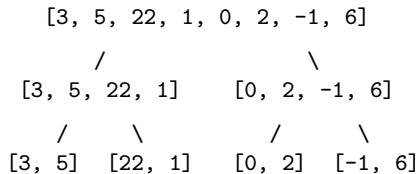
Analisis

[3, 5, 22, 1, 0, 2, -1, 6]

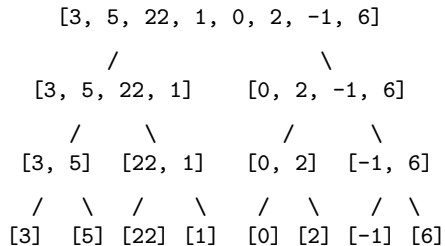
Analisis

$$\begin{array}{ccc}
 & [3, 5, 22, 1, 0, 2, -1, 6] & \\
 & / \qquad \qquad \qquad \backslash & \\
 [3, 5, 22, 1] & & [0, 2, -1, 6]
 \end{array}$$

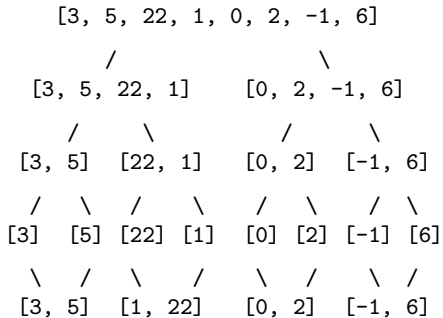
Analysis



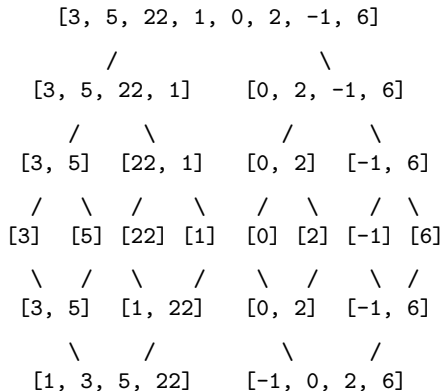
Analisis



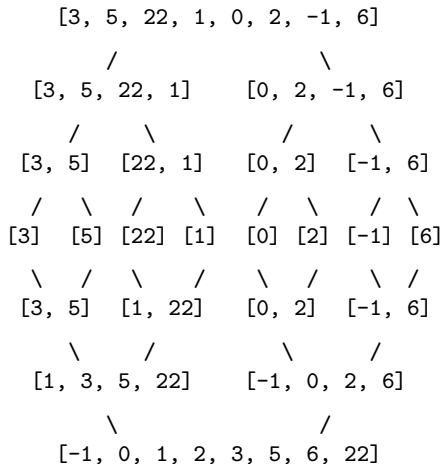
Analisis



Analysis



Analysis



Ingredientes

- Funcion `es_caso_base` que dado un problema determina si el problema es trivial.

Ingredientes

- Funcion `es_caso_base` que dado un problema determina si el problema es trivial.
- Funcion `dividir` que dado un problema, lo divide en dos problemas mas pequeños.

Ingredientes

- Funcion `es_caso_base` que dado un problema determina si el problema es trivial.
- Funcion `dividir` que dado un problema, lo divide en dos problemas mas pequeños.
- Funcion `fusionar` que dados dos problemas resueltos, los combina en una unica solucion.

Algoritmo

```
1  from sort import problema, es_caso_base, dividir, fusionar
2
3  def resolver(problema):
4      if es_caso_base(problema):
5          return problema
6
7      subproblema1, subproblema2 = dividir(problema)
8      solucion1, solucion2 = resolver(subproblema1), resolver(subproblema2)
9
10     return fusionar(solucion1, solucion2)
11
12 print(resolver(problema))
```

Características

Ventajas

- Es facil de implementar.
- Suele ser un algoritmo rapido.
- Si existe una solucion, la encuentra.

Características

Ventajas

- Es facil de implementar.
- Suele ser un algoritmo rapido.
- Si existe una solucion, la encuentra.

Desventajas

- No puede aplicarse siempre.
- Puede consumir mucha memoria.

Características

Ventajas

- Es facil de implementar.
- Suele ser un algoritmo rapido.
- Si existe una solucion, la encuentra.

Desventajas

- No puede aplicarse siempre.
- Puede consumir mucha memoria.

Casos de uso

Solo puede usarse si es factible dividir el problema y combinar las soluciones independientes para formar la solucion general.

Detalles

```
1  problema = [3, 5, 22, 1, 0, 2, -1, 6, 7, 11]
2
3  def es_caso_base(problema):
4      return len(problema) <= 1
5
6  def dividir(problema):
7      mitad = len(problema) // 2
8      return problema[:mitad], problema[mitad:]
9
10 def fusionar(solucion1, solucion2):
11     solucion = []
12
13     while solucion1 and solucion2:
14         if solucion1[0] <= solucion2[0]:
15             solucion.append(solucion1.pop(0))
16         else:
17             solucion.append(solucion2.pop(0))
18
19     return solucion + (solucion1 if solucion1 else solucion2)
```