

Concurso Programacion 2

Tecnicatura Universitaria en Inteligencia Artificial

Dámian Ariel Marotte

Section 1

Fuerza Bruta

Problema

Un usuario guarda informacion importante adentro de un archivo comprimido. Por desgracias, ha olvidado la contraseña y ahora no puede acceder a sus archivos.

Analisis

Intentos desordenados

- m8l
- 6ey7mf
- bc
- r3pe6aeo
- 7

Analysis

Intentos desordenados

- m8l
- 6ey7mf
- bc
- r3pe6aeo
- 7

Intentos ordenados

- a
- b
- c
- d
- e

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]
```

Ejemplo

- a

Ejemplo

- a
- b

Ejemplo

- a
- b
- c

Ejemplo

- a
- b
- c
- ...
- z

Ejemplo

- a
- b
- c
- ...
- z
- aa

Ejemplo

- a
- b
- c
- ...
- z
- aa
- ab

Ejemplo

- a
- b
- c
- ...
- z
- aa
- ab
- ac
- ...

Ejemplo

- a
- b
- c
- ...
- z
- aa
- ab
- ac
- ...
- aaaaaa
- ...

Ejemplo

- a
- b
- c
- ...
- z
- aa
- ab
- ac
- ...
- aaaaaa
- ...
- abcdd
- abcde

Section 2

Greedy

Problema

El cajero de un país que cuenta con monedas de denominación 4, 3 y 1 tiene que dar un vuelto de seis unidades. Se pregunta en qué forma debe hacerlo para minimizar la cantidad de monedas que debe entregar.

Analisis

Observemos que en principio la solución de fuerza bruta sigue estando disponible.

Sin embargo a veces es preferible resignar precisión con el objetivo de ganar velocidad.

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] * 2 + [3] * 2 + [1] * 2
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
```

Ejemplo

Iteracion 0

- `candidatos = [4, 4, 3, 3, 1, 1].`

Ejemplo

Iteracion 0

- `candidatos = [4, 4, 3, 3, 1, 1].`

Iteracion 1

- `x = 4.`
- `candidatos = [4, 3, 3, 1, 1].`
- `eleccion_actual = [4].`

Ejemplo

Iteracion 0

- `candidatos = [4, 4, 3, 3, 1, 1].`

Iteracion 1

- `x = 4.`
- `candidatos = [4, 3, 3, 1, 1].`
- `eleccion_actual = [4].`

Iteracion 2

- `x = 4.`
- `candidatos = [3, 3, 1, 1].`
- `eleccion_actual = [4].`

Ejemplo

Iteracion 0

- `candidatos = [4, 4, 3, 3, 1, 1].`

Iteracion 1

- `x = 4.`
- `candidatos = [4, 3, 3, 1, 1].`
- `eleccion_actual = [4].`

Iteracion 2

- `x = 4.`
- `candidatos = [3, 3, 1, 1].`
- `eleccion_actual = [4].`

Iteracion 3

- `x = 3.`
- `candidatos = [3, 1, 1].`
- `eleccion_actual = [4].`

Ejemplo

Iteracion 4

- $x = 3$.
- `candidatos = [1, 1]`.
- `eleccion_actual = [4]`.

Ejemplo

Iteracion 4

- $x = 3$.
- $\text{candidatos} = [1, 1]$.
- $\text{eleccion_actual} = [4]$.

Iteracion 5

- $x = 1$.
- $\text{candidatos} = [1]$.
- $\text{eleccion_actual} = [4, 1]$.

Ejemplo

Iteracion 4

- `x = 3.`
- `candidatos = [1, 1].`
- `eleccion_actual = [4].`

Iteracion 5

- `x = 1.`
- `candidatos = [1].`
- `eleccion_actual = [4, 1].`

Iteracion 6

- `x = 1.`
- `candidatos = [].`
- `eleccion_actual = [4, 1, 1].`

Section 3

Divide & Conquer

Problema

Se han guardado en una lista una cantidad de valores numericos. Para poder trabajar con ellos adecuadamente, es necesario ordenarlos.

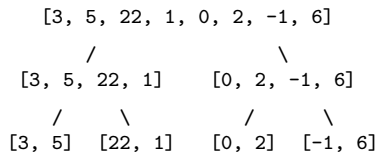
Analisis

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

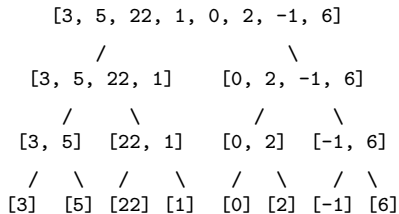
Analisis

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

Analisis



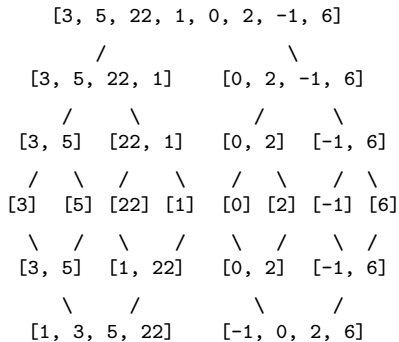
Analisis



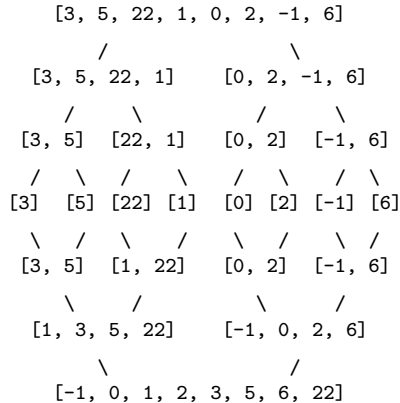
Analisis

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

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)
```

Ejemplo

Iteracion 0

```
solucion1 = [1, 3, 5, 22]  
solucion2 = [-1, 0, 2, 6]  
solucion = []
```

Ejemplo

Iteracion 0

```
solucion1 = [1, 3, 5, 22]  
solucion2 = [-1, 0, 2, 6]  
solucion = []
```

Iteracion 1

```
solucion1 = [1, 3, 5, 22]  
solucion2 = [0, 2, 6]  
solucion = [-1]
```

Ejemplo

Iteracion 0

```
solucion1 = [1, 3, 5, 22]  
solucion2 = [-1, 0, 2, 6]  
solucion = []
```

Iteracion 1

```
solucion1 = [1, 3, 5, 22]  
solucion2 = [0, 2, 6]  
solucion = [-1]
```

Iteracion 2

```
solucion1 = [1, 3, 5, 22]  
solucion2 = [2, 6]  
solucion = [-1, 0]
```

Ejemplo

Iteracion 0

```
solucion1 = [1, 3, 5, 22]  
solucion2 = [-1, 0, 2, 6]  
solucion = []
```

Iteracion 1

```
solucion1 = [1, 3, 5, 22]  
solucion2 = [0, 2, 6]  
solucion = [-1]
```

Iteracion 2

```
solucion1 = [1, 3, 5, 22]  
solucion2 = [2, 6]  
solucion = [-1, 0]
```

Iteracion 3

```
solucion1 = [3, 5, 22]  
solucion2 = [2, 6]  
solucion = [-1, 0, 1]
```

Ejemplo

Iteracion 4

```
solucion1 = [3, 5, 22]  
solucion2 = [6]  
solucion = [-1, 0, 1, 2]
```

Ejemplo

Iteracion 4

```
solucion1 = [3, 5, 22]  
solucion2 = [6]  
solucion = [-1, 0, 1, 2]
```

Iteracion 5

```
solucion1 = [5, 22]  
solucion2 = [6]  
solucion = [-1, 0, 1, 2, 3]
```

Ejemplo

Iteracion 4

```
solucion1 = [3, 5, 22]  
solucion2 = [6]  
solucion = [-1, 0, 1, 2]
```

Iteracion 5

```
solucion1 = [5, 22]  
solucion2 = [6]  
solucion = [-1, 0, 1, 2, 3]
```

Iteracion 6

```
solucion1 = [22]  
solucion2 = [6]  
solucion = [-1, 0, 1, 2, 3, 5]
```


Ejemplo

Iteracion 4

```
solucion1 = [3, 5, 22]  
solucion2 = [6]  
solucion = [-1, 0, 1, 2]
```

Iteracion 5

```
solucion1 = [5, 22]  
solucion2 = [6]  
solucion = [-1, 0, 1, 2, 3]
```

Iteracion 6

```
solucion1 = [22]  
solucion2 = [6]  
solucion = [-1, 0, 1, 2, 3, 5]
```

Iteracion 7

```
solucion1 = [22]  
solucion2 = []  
solucion = [-1, 0, 1, 2, 3, 5, 6]
```