Concurso Programacion 2

Tecnicatura Universitaria en Inteligencia Artificial

Dámian Ariel Marotte

Section 1

Fuerza Bruta

Problema

Perdida de contraseña.

Analisis

Intentos desordenados

- m8l
- 6ey7mf
- bc
- r3pe6aeo
- 7

Analisis

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)
```

Caracteristicas

Ventajas

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

Caracteristicas

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.

Caracteristicas

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

```python
alfabeto = "0123456789abcdefghijklmnopqrstuvwxyz"

def es_solucion(intento):
    return intento == "abcde"

def siguiente(intento = ""):
    if intento == alfabeto[-1] * len(intento):
        return alfabeto[0] * (len(intento) + 1)
    elif intento[-1] != alfabeto[-1]:
        return intento[:-1] + alfabeto[alfabeto.find(intento[-1]) + 1]
    else:
        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
- ...
- aaaaa
- ...

Ejemplo

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

Section 2

Greedy

Problema

Dar el vuelto.

Analisis

Observemos que en principio la solucion de fuerza bruta sigue estando disponibles.

Sin embargo a veces es preferible resignar presicion 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)
```

Caracteristicas

Ventajas

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

Caracteristicas

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.

Caracteristicas

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

Iteracion 5

- x = 1.
- candidatos = [1].
- 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

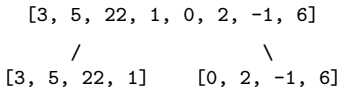
Problema

Ordenar numeros.

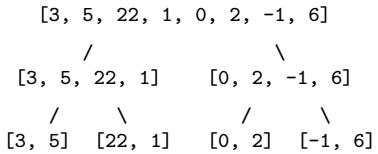
Analisis

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

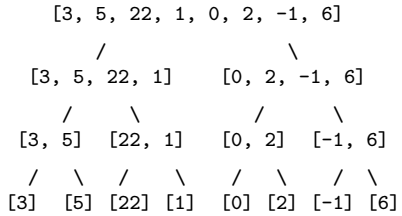
Analisis



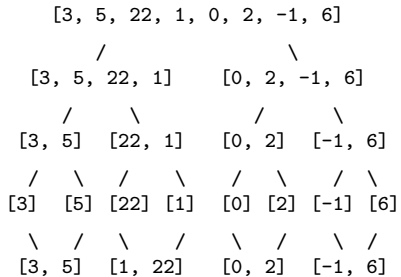
Analisis



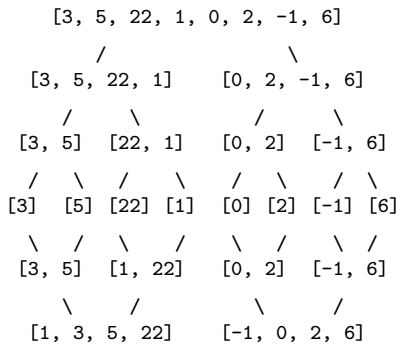
Analisis



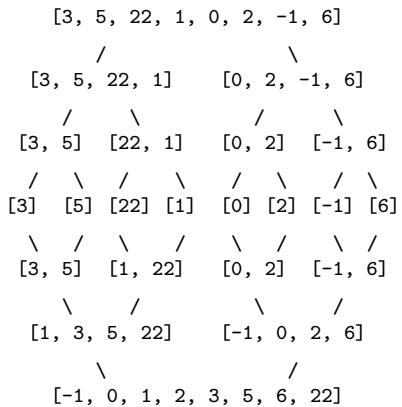
Analisis



Analisis


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

Analisis


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

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

```python
from sort import problema, es_caso_base, dividir, fusionar

def resolver(problema):
    if es_caso_base(problema):
        return problema

    subproblema1, subproblema2 = dividir(problema)
    solucion1, solucion2 = resolver(subproblema1), resolver(subproblema2)

    return fusionar(solucion1, solucion2)

print(resolver(problema))
```

Caracteristicas

Ventajas

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

Caracteristicas

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.

Caracteristicas

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

```python
problema = [3, 5, 22, 1, 0, 2, -1, 6, 7, 11]

def es_caso_base(problema):
    return len(problema) <= 1

def dividir(problema):
    mitad = len(problema) // 2
    return problema[:mitad], problema[mitad:]

def fusionar(solucion1, solucion2):
    solucion = []

    while solucion1 and solucion2:
        if solucion1[0] <= solucion2[0]:
            solucion.append(solucion1.pop(0))
        else:
            solucion.append(solucion2.pop(0))

    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]
```