UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Algorítmica (11593)

Primer Parcial

7 de noviembre de 2017

etsinf

Escuela Técnica Superior de Ingeniería Informática

NOMBRE:

1 — 4.5 puntos

Un ladrón planea robar en las casas de una misma calle. Para no levantar sospechas decide no robar nunca en 2 casas consecutivas.

Sabemos que hay N casas, numeradas de 1 a N, y que lo que puede robar en cada casa i tiene un beneficio de $B(i)$ euros, que siempre será un valor no negativo. Se desea maximizar la ganancia del ladrón. Para ello **se pide**:

1. Especificar formalmente el conjunto de soluciones factibles X, la función objetivo a maximizar f y la solución óptima buscada $\hat{x}$.
2. Como ayuda/sugerencia te proponemos *resolver un problema auxiliar*:

   > $R(n)$ representa el máximo beneficio que conseguiría el ladrón robando las casas entre 1 y n condicionado a que entra a robar en la número n.

   Se pide una ecuación recursiva que resuelva este problema. **También** se pide que indiques cómo resolver el problema original haciendo uso de $R(n)$.
3. El algoritmo iterativo (preferiblemente en Python3) asociado a la ecuación recursiva anterior para calcular *el mayor beneficio* (no hace falta determinar la elección concreta de casas a robar).
4. El coste temporal y espacial del algoritmo iterativo.

2 — 2.5 puntos

El siguiente código calcula el mínimo número de monedas necesario para devolver la cantidad `Q` usando `N` tipos de moneda siendo `v` y `m` vectores de longitud `N` que contienen, respectivamente, el valor y la cantidad de monedas de cada tipo:

```
def number_of_coins(Q, v, m, infinity=2**31):
  N = len(v) # numero de tipos de moneda, N==len(m) también
  M = {} # M[i,q] es el numero de monedas para representar cantidad q
         # usando los i primeros tipos de moneda
         # con 0 tipos de moneda solamente podemos representar la cantidad 0
  M[0,0] = 0
  for q in range(1, Q+1):
    M[0,q] = infinity
  # para más de 0 tipos de moneda, desde 0 hasta N inclusive:
  for i in range(1,N+1):
    for q in range(Q+1): # probar cualquier cantidad entre 0 y Q inclusive
      M[i,q] = min(M[i-1,q-x*v[i-1]]+x for x in range(min(q//v[i-1], m[i-1])+1))
  return M[N,Q]

Q = 24
v = [1, 2, 5, 10, 20, 50]
m = [3, 1, 4,  1,  2,  1]
print(number_of_coins(Q, v, m))
```

Se pide: que realices los cambios necesarios que consideres para que la función devuelva, en lugar del mínimo número de monedas utilizada, las monedas que consiguen ese mínimo en forma de un vector de talla `N` donde en la posición i-ésima esté el número de monedas a utilizar de valor `v[i]`.

Una pizzería tiene una lista de ingredientes y desea enumerar todas las posibles formas de combinar 4 de esos ingredientes de manera que no aparezcan simultáneamente dos ingredientes de una lista de ingredientes que no combinan:

```
ingredientes = ['mozzarela','anchoa','alcaparra','jamon','atun',
                'champinyon','aceituna','chorizo']
no_combinan = [('mozzarela','anchoa'),('atun','chorizo'),('jamon','atun'),
               ('anchoa','chorizo'),('anchoa','champinyon')]
```

La lista de ingredientes es una lista Python de cadenas, mientras que la lista `no_combinan` está representada por una lista de tuplas donde cada tupla tiene 2 ingredientes en el mismo orden relativo que aparecen en la lista `ingredientes` (por ejemplo, aparece la tupla `('atun','chorizo')` donde `'atun'` aparece antes en `ingredientes` que `'chorizo'`).

Se pide: utilizar la estrategia de búsqueda con retroceso o *backtracking* para mostrar por salida estándar todas las combinaciones de 4 ingredientes que combinen. Por ejemplo, la llamada:

```
listar(ingredientes,no_combinan)
```

produciría el siguiente resultado por pantalla:

```
mozzarela, alcaparra, jamon, champinyon
mozzarela, alcaparra, jamon, aceituna
mozzarela, alcaparra, atun, champinyon
mozzarela, alcaparra, atun, aceituna
mozzarela, alcaparra, champinyon, aceituna
mozzarela, jamon, champinyon, aceituna
mozzarela, atun, champinyon, aceituna
anchoa, alcaparra, jamon, aceituna
anchoa, alcaparra, atun, aceituna
alcaparra, jamon, champinyon, aceituna
alcaparra, atun, champinyon, aceituna
```

Como pista comentar que para mostrar `['a','b','c']` como arriba se puede utilizar:

```
>>> print(", ".join(['a','b','c']))
a, b, c
```

y para sacar la posición de un ingrediente en la lista se puede utilizar el método `index`:

```
>>> ingredientes.index('atun')
4
```

Es **importante** no sacar varias veces por pantalla una misma combinación de ingredientes (incluso si aparecen en un orden diferente).

Nota: Aunque no es la solución más deseable, si te resulta más fácil puedes suponer que en lugar de una lista de ingredientes se proporciona el número de ingredientes N y que la lista de ingredientes que no combinan hace referencia a ellos usando enteros entre 0 y $N-1$. De manera que

```
listarN(7,[(0,1),(4,7),(3,4),(1,7),(1,5)])
```

produciría la siguiente salida:

```
[0, 2, 3, 5]
[0, 2, 3, 6]
[0, 2, 4, 5]
[0, 2, 4, 6]
[0, 2, 5, 6]
[0, 3, 5, 6]
[0, 4, 5, 6]
[1, 2, 3, 6]
[1, 2, 4, 6]
[2, 3, 5, 6]
[2, 4, 5, 6]
```