UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Algorítmica (11593)

Recuperación del Primer Parcial
23 de enero de 2018

etsinf
Escuela Técnica Superior de Ingeniería Informática

NOMBRE:

1 **4.5 puntos**

Una discográfica planea editar una recopilación. Ha seleccionado un total de N grupos y, para cada grupo i ($1 \leq i \leq N$), un número de canciones $c(i)$. Cada canción tiene una puntuación $p(i, j)$ siendo i el grupo y j la canción ($1 \leq j \leq c(i)$). De manera similar, $d(i, j)$ indica la duración (en minutos) de dicha canción. Deseamos seleccionar una canción de cada grupo de manera que se maximize la puntuación total sin que el álbum supere los T=90 minutos de duración. 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. Dada una instancia ¿cuál debe de ser la condición para que exista alguna solución factible?
3. Plantea una ecuación recursiva que devuelva la máxima puntuación alcanzable. ¿Qué expresión, utilizando la ecuación recursiva anterior, deberías hacer para resolver el problema?
4. Algoritmo iterativo (función `elegir` en Python3) asociado a la ecuación anterior para calcular *únicamente la mejor puntuación*. La función recibirá los enteros `N` y `T` y las funciones `c`, `p` y `d`.
5. El coste temporal y espacial del algoritmo iterativo, justificando (brevemente) las respuestas.

Solución:

1. El espacio de búsqueda se puede modelar como un conjunto de tuplas de longitud N donde la componente i-ésima de cada tupla representa la canción del grupo i-ésimo, sujeto a que la duración total no supere los T minutos:

$$X = \{(x_1, \ldots, x_N) | 1 \leq x_i \leq c(i), 1 \leq i \leq N, \sum_{i=1}^{N} d(i, x_i) \leq T\}$$

   La función objetivo a maximizar sería $f((x_1, \ldots, x_N)) = \sum_{i=1}^{N} p(i, x_i)$ mientras que la solución óptima buscada es $\hat{x} = \text{argmax}_{x \in X} f(x)$.

2. Para que exista una solución factible debe de ser posible tomar una canción de cada grupo y que la duración total no exceda el valor T. Sería suficiente que

$$\sum_{i=1}^{N} \left(min_{j=1}^{c(i)} d(i, j) \right) \leq T$$

3. Sea $P(g, t)$ la mayor puntuación que se puede conseguir asignando t minutos a los primeros g grupos musicales:

$$P(g, t) = \begin{cases} 0 & \text{si } g = 0 \wedge t = 0 \\ -\infty & \text{si } g = 0 \wedge t \neq 0 \\ \text{máx}_{1 \leq j \leq c(g)} P(g-1, t - d(g, j)) + p(g, j) & \text{en otro caso} \end{cases}$$

   Para calcular lo que nos pide el problema podemos realizar la siguiente expresión que hace uso de la ecuación recursiva anterior:

$$max_{\left(\sum_{i=1}^{N} min_{j=1}^{c(i)} d(i,j)\right) \leq t \leq T} P(N, t)$$

4. Algoritmo iterativo:

```
def elegir(N,T,c,p,d,infty=2**31):
  Tmin = sum(min(d(g,cancion) for cancion in range(c(g))) for g in range(N))
  if Tmin>T:
    return -infty
  M = {}
  for t in range(1,T+1):
    M[0,t] = -infty
  M[0,0] = 0
  for g in range(N):
    for t in range(T+1):
      M[g][t]=max((M[g-1,t-d(g,cancion)]+p(g,cancion) for cancion in range(c(g))
                  if t>=d(g,cancion)), default=-infty)
  return max(M[g][t] for t in range(Tmin,T+1))
```

5. El coste espacial sería $O(N \cdot T)$, aunque se podría implementar con reducción del coste espacial utilizando únicamente coste $O(T)$. El coste temporal viene dominado por los 3 bucles anidados que recorren N, T y, para cada grupo, su lista de canciones. Por tanto sería $O(N \cdot T \cdot M)$ siendo $M = \text{máx}_{1 \leq i \leq N} c(i)$, aunque sería posible ajustar más dicho coste y sustituir $T \cdot M$ por $(T + \sum_{i=1}^{N} c(i))$.

2 **2 puntos**

La siguiente ecuación recursiva resuelve la versión del cambio de monedas que cuenta el número de formas distintas de realizar el cambio de la cantidad Q usando los N tipos de moneda $v_1, \ldots, v_N$:

$$M(i,q) = \begin{cases} 1 & \text{si } i = 0, q = 0 \\ 0 & \text{si } i = 0, q > 0 \\ \sum_{x_i=0}^{\lfloor q/v_i \rfloor} M(i-1, q - x_i * v_i) & \text{en otro caso} \end{cases}$$

Implementa la versión iterativa (en Python) de este problema en la versión **con reducción del coste espacial**. Indica el coste espacial y temporal del algoritmo realizado justificando (brevemente) la respuesta. Ejemplo de llamada: `cuentaCambios(8,[1,2,5,10,20]))`

Solución:

```
def cuentaCambios(Q,tipos):
  V = [0]*(Q+1) # 0 tipos de moneda solamente es válido para 0 euros
  V[0] = 1      # que tiene 1 forma válida (0 monedas)
  for tipo in tipos: # ahora vamos probando tipo de moneda por tipo:
    V = [sum((V[q-x*tipo] for x in range(0,(q//tipo)+1))) for q in range(Q+1)]
  return V[Q]
```

Queremos asignar unos eslóganes a los países de un continente. Un mismo eslogan puede estar repetido en varios países. Sin embargo, hay combinaciones de esglóganes que no pueden utilizarse en países limítrofes porque aumentaría la hostilidad entre ellos. Elabora una función que enumere las formas de asociar eslóganes a países a partir de estos datos:

- `P` es la lista de N países, ejemplo: `P = ['Andoria','Nibiru','Regulus']`
- `G` es una matriz simétrica y cuadrada $N \times N$ indicando qué países son limítrofes. Ejemplo:

```
G = [[False, True,  False], # Andoria linda con Nibiru
     [True,  False, True],  # Nibiru linda con Andoria y Regulus
     [False, True,  False]] # Regulus linda con Nibiru
```

- Una lista `S` de eslóganes tipo:

```
S = ['los mejores','los hubo peores','preparados para la guerra',
  'patriotas cuando conviene', 'ni fu ni fa','yo me quiero ir de aquí']
```

- Una función `compatibles` que recibe dos eslóganes y devuelve un booleano.

Se pide: utilizar búsqueda con retroceso o *backtracking* para devolver todas las posibles asignaciones válidas de eslóganes a países. Ejemplo: la llamada `print(asignar(P,S,compatibles))` produciría:

```
[[('Andoria','los mejores'),('Nibiru','ni fu ni fa'),('Regulus','no nos ganarán')],
  ...
]
```

Solución:

```
def asignar(P,S,compatibles):
  N=len(P) # número de países
  solutions = []
  currentsol = [None]*N
  def backtracking(longSol):
    if longSol==N:
      solutions.append([(P[i],slogan) for i,slogan in enumerate(currentsol)])
    else:
      for slogan in S:
        if all((compatibles(slogan,currentsol[i]) for i in range(longSol)
                if G[i][longSol])):
          currentsol[longSol] = slogan
          backtracking(longSol+1)
  backtracking(0)
  return solutions
```