UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Algorítmica (11593)

Simulacro de Primer Parcial
25 de octubre de 2017

etsinf
Escuela Técnica Superior de Ingeniería Informática

NOMBRE:

1 — 6.5 puntos

Un restaurante se plantea reducir el precio de su famosa "paella". Para ello quiere conseguir la paella de mayor calidad que no supere su presupuesto tope P. La "paella" tiene N ingredientes y, para cada ingrediente, se han buscado K alternativas (marcas, proveedores,...):

- El valor $p_{i,j}$ representa el *precio* (en céntimos, valor entero) que tendrá el ingrediente i con la alternativa j, siendo $1 \leq i \leq N$ y $1 \leq j \leq K$.
- Por otra parte, el valor $c_{i,j}$ representa el *beneficio* que aporta a la "paella" la alternativa j del ingrediente i, de nuevo siendo $1 \leq i \leq N$ y $1 \leq j \leq K$.

Queremos maximizar la suma de beneficio de los N ingredientes de la "paella" sin superar nuestro presupuesto tope de P (en céntimos, valor entero). **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. Una ecuación recursiva que resuelva el problema y la llamada inicial.
3. El algoritmo iterativo asociado a la ecuación recursiva anterior para calcular *la mayor calidad* (no hace falta determinar la elección concreta de la calidad de cada ingrediente).
4. El coste temporal y espacial del algoritmo iterativo.

Solución:

1. Especificar formalmente el conjunto de soluciones factibles X, la función objetivo a maximizar f y la solución óptima buscada $\hat{x}$.

   Podemos representar las soluciones como una tupla de longitud N donde cada elemento indique el tipo de alterantiva del ingrediente asociado a esa posición:

   $$X = \left\{ (x_1, \ldots, x_N) \in \{1, \ldots, K\}^N \mid \sum_{i=1}^{N} p_{i,x_i} \leq P \right\}$$

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

2. Una ecuación recursiva que resuelva el problema y la llamada inicial.

   Vamos a considerar que $F(n, w)$ calcula el mejor beneficio que se puede conseguir con los n primeros ingredientes y con un presupuesto w para los mismos.

   $$F(n,w) = \begin{cases} 0 & \text{si } n = 0 \\ -\infty & \text{si } n > 0 \text{ y } p_{n,j} > w \ \forall 1 \leq j \leq K \\ \max\limits_{\substack{1 \leq j \leq k \\ p_{n,j} \leq w}} F(n-1, w - p_{n,j}) + c_{n,j} & \text{en otro caso} \end{cases}$$

   La llamada inicial sería $F(N, P)$.

3. El algoritmo iterativo asociado a la ecuación recursiva anterior para calcular *la mayor calidad* (no hace falta determinar la elección concreta de la calidad de cada ingrediente).

   Vamos a suponer que tanto p como c son diccionarios Python que asocian a cada tupla `(i,j)` su valor.

```
def paella(N,K,P,p,c,inf=2**31):
  m = {}
  for w in range(P+1):
    m[0,w]=0
  for n in range(1,N+1):
    for w in range(P+1):
      m[n,w] = max((m[n-1,w-p[n,j]]+c[n,j]
                    for j in range(1,K+1) if w>=p[n,j]),
                   default=-inf)
  return m[N,P]
```

4. El coste temporal y espacial del algoritmo iterativo.

   El coste espacial de este algoritmo es $O(N \cdot P)$, mientras que el coste temporal es $O(N \cdot P \cdot K)$.

Un supersticioso nos pide imprimir por salida estándar todas las permutaciones de los N primeros números enteros *exceptuando las que tengan ciertas combinaciones de números consecutivas.*

La entrada de tu programa ha de ser el valor N y una lista de parejas como en el siguiente ejemplo:

```
N=4
prohibidas=[(2,3),(4,1),(3,1),(1,2)]
no_prohibidas(N,prohibidas)
```

daría el siguiente resultado (en otro orden sería correcto si salieran las mismas soluciones):

```
[1, 3, 2, 4]
[1, 3, 4, 2]
[1, 4, 3, 2]
[2, 1, 3, 4]
[2, 1, 4, 3]
[3, 2, 1, 4]
[3, 4, 2, 1]
[4, 2, 1, 3]
[4, 3, 2, 1]
```

Observa que, por ejemplo, no se ha generado la secuencia `[1,2,3,4]` porque contiene la pareja `2,3`. Ten en cuenta que el orden de los números en las prohibidas es relevante. Así, por ejemplo, la secuencia `[1, 3, 2, 4]` sí aparece a pesar de tener `1,3` porque lo que está prohibido es `3,1`.

Solución:

```
def permutaciones_no_prohibidas(N,prohibidas):
  def backtracking(nodo):
    if len(nodo)==N: # is complete
      print(nodo)
    else:
      for i in range(1,N+1):
        if (i not in nodo and
            (nodo[-1],i) not in prohibidas):
          backtracking(nodo+[i])
  for i in range(1,N+1):
    backtracking([i])
```

Observa que esta solución no tiene por qué ser la única correcta. Así, por ejemplo, en lugar de realizar un bucle for en la función principal para situar el primer elemento de la permutación es posible realizar una llamada inicial `backtracking([])` y tratar el caso de un solo dígito como especial en el tratamiento de *"prometedor"*:

```
def permutaciones_no_prohibidas(N,prohibidas):
  def backtracking(nodo):
    if len(nodo)==N: # is complete
      print(nodo)
    else:
      for i in range(1,N+1):
        if (i not in nodo and
            (len(nodo)==0 or
             (nodo[-1],i) not in prohibidas)):
          backtracking(nodo+[i])
  backtracking([])
```