Prueba de aula Algorítmica (11593)

15 de noviembre de 2022

1 **2 puntos**

El siguiente algoritmo iterativo de Programación Dinámica resuelve el problema del cálculo de la probabilidad del trayecto más probable en el río Congo:

```
def iterative_maximum_probability(E, p):
   P = [None]*E
   P[0] = 1
   P[1] = p(1,0)
   for i in xrange(2, E):
      P[i] = max( P[i-1] * p(i,i-1), P[i-2] * p(i,i-2) )
   return P[E-1]
```

siendo `E` el número de embarcaderos y `p(i,j)` una función que devuelve la probabilidad de llegar al embarcadero `i` desde `j` (ambos numerados de `0` a `E-1`). Responde a las siguientes cuestiones:

a) Analiza el coste temporal y espacial del algoritmo.
b) Modifica el código para conocer también el camino de máxima probabilidad.

Solución: El coste espacial viene dado por el vector `P`, de talla `E`: $O(E)$, y el temporal por el bucle `for` que se ejecuta $O(E)$ veces, cada una de ellas con un coste constante (es una maximización entre dos elementos, independiente de la talla del prroblema). Por tanto, también $O(E)$.

Para encontrar el camino de mayor probabilidad basta con añadir un vector de punteros hacia atrás y luego hacer sobre él el correspondiente postproceso para recuperar el camino:

```
def iterative_maximum_probability(E, p):
   P,B = [None]*E,[None]*E
   P[0] = 1
   P[1] = p(1,0)
   B[1] = 0
   for i in xrange(2, E):
      P[i],B[i] = max( (P[i-1] * p(i,i-1),i-1), (P[i-2] * p(i,i-2),i-2) )
   path = [E-1]
   while B[path[-1]] != None:
     path.append(B[path[-1]])
   path.reverse()
   return P[E-1],path
```

Tras una catástrofe natural, un continente necesita ayuda humanitaria (víveres, material médico). Se han preparado paquetes *idénticos* con este tipo de ayuda y se deben distribuir a las distintas zonas. Se tienen P paquetes y se plantea cómo distribuirlos entre las Z zonas afectadas. Se ha estimado el beneficio a conseguir en cada zona z, para z entre 1 y Z, dependiendo del número de paquetes p que se envien. Esta estimación se obtiene con la función $b(z, p)$. Los expertos también han decidido que, como mínimo, a cada zona se le deben enviar m paquetes.

En esta situación, deseamos calcular la distribución de los paquetes entre las zonas que maximice el beneficio total. Para ello nos piden:

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 de Programación Dinámica que resuelva el problema de encontrar el máximo beneficio que se puede alcanzar. Indica cuál sería la llamada inicial para resolver el problema.

3. El algoritmo iterativo asociado a la ecuación recursiva anterior para calcular *el beneficio*.

4. Analiza el coste temporal y espacial del algoritmo iterativo.

5. ¿Se puede reducir el coste espacial y/o temporal del algoritmo anterior? ¿En qué condiciones?

Solución:

Este ejercicio se puede resolver, al menos de dos maneras diferentes:

- Modificando el problema de asignación de recursos tal cual se ha visto en teoría para añadir un número mínimo de unidades a cada zona.
- Repartiendo *a priori* los paquetes mínimos entre las zonas y usando el problema estándar de asignación de recursos para repartir el resto.

Soluciones factibles:

$$X = \left\{ (x_1, x_2, \ldots, x_Z) \in \mathbf{N}^Z \;\middle|\; m \leq x_i \leq P, 1 \leq i \leq Z; \quad \sum_{1 \leq i \leq Z} x_i \leq P \right\}.$$

Función objetivo:

$$f((x_1, x_2, \ldots, x_Z)) = \sum_{1 \leq i \leq Z} b(i, x_i).$$

Solución óptima:

$$\hat{x} = \operatorname*{argmax}_{x \in X} f((x_1, x_2, \ldots, x_Z))$$

El problema a resolver es calcular el beneficio máximo que se puede conseguir al repartir como mucho P paquetes entre las Z zonas, con las restricciones impuestas (a cada zona, como mínimo, se debe hacer llegar m paquetes). Sea este beneficio $B(Z, P)$. Supongamos $Z = 3$ zonas, un total de $P = 8$ paquetes, y cada zona debe tener, como mínimo, $m = 1$ paquete. El problema a resolver es $B(3, 8)$. En este caso, a la zona última se le podrá asignar: p'=1 (es el mínimo), 2, 3, ..., 8 paquetes; con un beneficio de $b(3, p')$ para esa zona a la que habría que añadir el beneficio máximo del problema que quedaría por resolver, que es $B(2, 8 - p')$. En realidad, se puede afinar un poco más porque no se le pueden asignar 8 paquetes a la zona 3, hemos de dejar al menos un paquete para las otras zonas, por lo que a la zona 3 se le podrían asignar $m \leq p' \leq P - 2m$.

Con todo esto:

$$B(i,p) = \begin{cases} 0, & \text{si } p = 0; \\ \max\limits_{m \leq p' \leq P - m*(i-1)} \{B(i-1, p-p') + b(i,p')\}, & \text{si } i > 0 \text{ y } p > 0. \end{cases}$$

La llamada al algoritmo recursivo sería: $B(Z, P)$. Algoritmo iterativo: grafo de dependencias (se deben tener resueltos todos los problemas $B(i-1, p)$ antes de resolver el problema $B(i, p)$).

```
def beneficio_zonas(Z, P, m, b):
   B = {}
   for i in xrange(Z+1):
      B[i, 0] = 0
   for i in xrange(1, Z+1):
      B[i, 0] = 0
      for p in xrange(1, P+1):
         B[i, p] = max( B[i-1, p-p'] + b[i, p'] for p' in xrange(m,P-m*(i-1)+1))
   return B[Z, P]
```

El coste temporal del algoritmo es $O(ZP^2)$. El coste espacial es $O(ZP)$ (reducible a dos columnas $O(P)$ si no se necesita la asignación óptima de paquetes a cada zona, sólo el beneficio).

3 **3 puntos**

En el problema de la existencia de un ciclo Hamiltoniano, dado un grafo $G = (V, E)$, deseamos encontrar un camino que parta de un vértice, termine en el mismo vértice y visite todos los vértices del grafo una sola vez (excepto en el caso del vértice de partida, que deberá coincidir con el de llegada).

Se pide que implementes un algoritmo que, utilizando la técnica de Búsqueda con Retroceso, devuelva la solución encontrada, si es que existe. Utiliza una representación de grafo en forma de listas de adyacencia. Recuerda que, como el ciclo hamiltoniano puede empezar en cualquier vértice, es una práctica habitual empezar la comprobación por el primer vértice del grafo, $G.V[0]$. Para ello:

1. Explica brevemente la estrategia a seguir.
2. Escribe un algoritmo (en seudo-código o python) que implemente la estrategia anterior.

Solución:

```
# un grafo dirigido representado mediante listas de adyacencia seria
# simplemente una lista de listas del tipo:

G = [[1,2,3],       # del vertice 0 vamos a los vertices 1,2,3
     [0,3,4],       # del vertice 1
     [0,3,5],       # del vertice 2
     [0,1,2,4,5,6], # del vertice 3
     [1,3,7],       # del vertice 4
     [2,3,6,8],     # del vertice 5
     [3,5,7,8,9],   # del vertice 6
     [4,6,9],       # del vertice 7
     [5,6,9],       # del vertice 8
     [6,7,8]]       # del vertice 9

# para este tipo de grafo se puede definir el siguiente codigo de backtracking:

def hamiltonian_cycle(G):
  def backtracking(path):
    if len(path)==len(G):
      if path[0] in G[path[-1]]: return path+[0]
    else:
      for v in [x for x in G[path[-1]] if x not in path]:
        found = backtracking(path+[v])
        if found!=None: return found
    return None
  return backtracking([0])

# ejemplo de uso:
print hamiltonian_cycle(G)
```