UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Algorítmica (11593)

8 de enero de 2016

etsinf
Escuela Técnica Superior de Ingeniería Informática

NOMBRE:

1 — 3 puntos

Después de la Navidad viene la época de dietas y la cuesta de enero. Nos han dado un cheque regalo para el supermercado con un valor de E euros que no nos permite comprar productos repetidos. Con ese cheque queremos realizar una compra de valor exactamente E euros (no puede sobrarnos ni faltarnos nada) que, además, consiga un carrito con un total de C calorías (ni una más ni una menos). Nos dan una lista de los N productos candidatos del super con sus respectivos precios: $p_1, p_2, \ldots, p_N$ y calorías $c_1, c_2, \ldots, c_N$. Nos piden implementar un algoritmo basado en backtracking o búsqueda con retroceso que nos permita encontrar una combinación de productos (si existe) con las condiciones exigidas. Por ejemplo, para estos valores:

```
p = [5,3,4,2,1]
c = [200,110,100,80,50]
E = 10
C = 350
```

una posible solución de la llamada `calorias(p,c,E,C)` sería el vector:

```
[1,0,1,0,1]
```

Solución: es una modificación de "la suma del subconjunto" (subset-sum) visto en clase. Basta con replicar los pesos y las comprobaciones para llevar euros y calorías al mismo tiempo. Aunque no es imprescindible, la condición de prometedor puede tener en cuenta no solamente que no nos hemos pasado en euros y en calorías sino, además, que es posible alcanzar los valores deseados con el resto de objetos que quedan por considerar. Una posible solución:

```
def calorias(p,c,E,C):
  n = len(p) # numero de productos
  sol = [None]*n
  sum_p = [ sum(p[j] for j in range(i+1,n)) for i in range(n)] # ineficiente
  sum_c = [ sum(c[j] for j in range(i+1,n)) for i in range(n)] # ineficiente

  def backtracking(longSol,E,C):
    if longSol == n: # es completa
      return True # en este caso, no hace falta comprobar factible
    # no completa, ramificar
    # 1) consideramos primero que se compra este objeto:
    if (E>=p[longSol] and p[longSol]+sum_p[longSol]>=E and
        C>=c[longSol] and c[longSol]+sum_c[longSol]>=C): # prometedor
      sol[longSol]=1
      if backtracking(longSol+1,E-p[longSol],C-c[longSol]):
        return True
    # 2) ahora consideramos que no se compra:
    if sum_p[longSol]>=E and sum_c[longSol]>=C: # prometedor
      sol[longSol]=0
      if backtracking(longSol+1,E,C):
        return True
    return False

  if backtracking(0,E,C):
    return sol
  else:
    return None
```

```
if __name__ == "__main__":
  p = [5,3,4,2,1]
  c = [200,110,100,80,50]
  E = 10
  C = 350
  print calorias(p,c,E,C)
```

2 **4 puntos**

Dado un grafo no dirigido $G = (V, A)$ con $N = |V|$ vértices $v_1, \ldots, v_N$ y un conjunto de aristas modelados por una función de adyacencia $A : V \times V \to Bool$ tal que $A(i, j) = true$ sii hay una arista entre v_i y v_j. Nos piden asociar a cada vértice un color entre 1 y C de modo que no haya ninguna arista conectando vértices del mismo color. También nos indican que utilizar un color c en el vértice i tiene un coste dado por $coste(c, i)$. Queremos colorear el grafo de la manera más barata posible (minimizando la suma de los costes de colorear cada vértice). Para ello se pretende utilizar la estrategia de ramificación y poda y nos piden:

a) Expresar formalmente el conjunto de soluciones factibles, la función objetivo a minimizar y la solución óptima buscada.
b) Describir los siguientes conceptos sobre los estados que serán necesarios para el algoritmo:
   1) Representación de un estado (no terminal) y su coste espacial.
   2) Condición para que un estado sea solución.
   3) Identifica el estado inicial que representa todo el conjunto de soluciones factibles.
c) Definir una función de ramificación. Analiza su coste temporal.
d) Diseñar una cota optimista no trivial. Estudia cómo se puede realizar el cálculo de la cota de forma eficiente. ¿Cuál sería su coste temporal? Justifícalo.

Solución:

a) Expresar formalmente el conjunto de soluciones factibles, la función objetivo a minimizar y la solución óptima buscada.
El conjunto de soluciones factibles coincide con el problema del coloreado de grafos que normalmente se estudia en búsqueda con retroceso y que se puede modelar de la manera siguiente:

$$X = \{(c_1, \ldots, c_N) \in \{1, \ldots, C\}^N \mid A(i, j) \to c_i \neq c_j \ \forall 1 \leq i < j \leq N\}$$

La función objetivo a minimizar es:

$$f((c_1, \ldots, c_N)) = \sum_{i=1}^{N} coste(c_i, i)$$

Y la solución óptima buscada es:

$$\hat{x} = \operatorname*{argmin}_{x \in X} f(x)$$

b) Describir los siguientes conceptos sobre los estados que serán necesarios para el algoritmo:
   1) Representación de un estado (no terminal) y su coste espacial.
   Una solución parcial puede representarse como un prefijo de una solución completa. Es decir, mediante una tupla $(c_1, \ldots, c_k, ?)$ de longitud $k < N$, su coste espacial es lineal con la longitud de la misma (una cota superior puede ser $O(N)$). **Nota:** No hemos discutimos el coste con una representación de los estados en forma de árbol de prefijos donde cada estado se representa mediante el último valor y un puntero al padre.
   2) Condición para que un estado sea solución.
   Para que un estado o solución parcial $(c_1, \ldots, c_k, ?)$ sea solución basta con que $k = N$.
   3) Identifica el estado inicial que representa todo el conjunto de soluciones factibles.
   El estado inicial que representa implícitamente todo el conjunto de soluciones factibles se corresponde a la secuencia de longitud 0 que normalmente denotamos con (?).

c) Definir una función de ramificación. Analiza su coste temporal.

$$branch((c_1,\ldots,c_k,?)) = \{(c_1,\ldots,c_k,c_{k+1},?)\,|\,c_{k+1}\in\{1,\ldots,C\},\ A(i,k{+}1)\rightarrow c_i\neq c_{k+1}\ \forall 1\leq i\leq k\}$$

Es decir, basta con añadir un nuevo color a la tupla (asociado al vértice $k+1$) de modo que ese color no coincida con el color ya puesto en algun vértice adyacente a $k+1$. El coste espacial para generar cada hijo, cuando no se utiliza una representación con árbol de prefijos y puntero al padre, es $O(k)$ que se puede acotar con $O(N)$.
El coste temporal de generar un estado hijo es, como poco, el coste de generarlo (escribirlo), si bien hay un coste asociado a los estados no prometedores que finalmente ni se generan. El coste de ramificar todos los hijos se puede hacer como sigue: 1) recorrer los vértices v_1 hasta v_k comprobando cuáles son adyacentes a v_{k+1} y marcar ese color para no utilizarlo, 2) usar los colores restantes para ramificar. Con una representación explícita de los estados el coste total se puede acotar por $O(C\times N)$, mientras que con una representación con árbol de prefijos y puntero al padre se puede conseguir un coste $O(C+N)$.
Nota: También es posible considerar una condición de prometedor más restrictiva que la anterior viendo si alguno de los vértices que todavía no se han coloreado tienen colores disponibles, ignorando las incompatibilidades entre vértices no coloreados.

d) Diseñar una cota optimista no trivial. Estudia cómo se puede realizar el cálculo de la cota de forma eficiente. ¿Cuál sería su coste temporal? Justifícalo.
Una cota optimista no trivial (es decir, que depende de la solución parcial particular) sería sumar al coste de la parte conocida una estimación optimista de lo que queda por colorear. Podemos quitar la restricción sobre el color y pensar que podemos colorear cada vértice restante con el color que resulte más barato:

$$opt((c_1,\ldots,c_k)) = \sum_{i=1}^{k} coste(c_i,i) + \sum_{i=k+1}^{N} \min_{c=1}^{C} coste(c,i)$$

La manera más barata de colorear cada vértice se puede precalcular:

$$costemin(i) = \min_{c=1}^{C} coste(c,i)$$

Y la forma más eficiente de calcular esta cota sería de manera incremental:

$$opt((c_1,\ldots,c_{k+1})) = opt((c_1,\ldots,c_k)) - costemin(k+1) + coste(c_{k+1},k+1)$$

De esta manera, el coste temporal pasaría de lineal a constante.
Nota: Una cota algo más informada sería tener en cuenta la incompatibilidad de colores con los vértices ya coloreados (valga la redundancia) pero no tenerla en cuenta entre los vértices que quedan por colorear:

$$opt((c_1,\ldots,c_k)) = \sum_{i=1}^{k} coste(c_i,i) + \sum_{i=k+1}^{N} \min_{\substack{c\in\{1,\ldots,C\}\\ \neg A(c_i,c)\forall 1\leq i<k}} coste(c,i)$$

3 **3 puntos**

Debemos enviar N=4 productos idénticos. Cada producto está en una ciudad diferente y va a un destino también distinto. Nos dan una matriz con el coste de enviar el producto de cada posible origen a cada posible destino. Se desea calcular el reparto más barato (la mejor asignación de ciudades origen a ciudades destino).

Se pide: realizar una traza de un algoritmo de ramificación y poda basada en *poda explícita* que calcule el reparto más barato utilizando como cota optimista suponer que los paquetes no enviados pueden enviarse al destino más barato sin importar si dicho destino ya tiene envío.

La traza se debe mostrar como la *secuencia de listas de estados activos* de cada iteración del algoritmo indicando la cota optimista de cada estado (ej: como superíndice) y subrayando el estado que se selecciona para la siguiente iteración. Hay que indicar también si se actualiza la variable mejor solución y la poda explícita.

Puedes no utilizar una solución inicial o utilizar como solución inicial un reparto voraz que recorra los orígentes (de arriba hacia abajo) enviando al destino más barato de los disponibles (de izquierda a derecha).

*orig**dest*	1	2	3	4
1	3	1	4	2
2	3	3	4	2
3	5	5	4	3
4	7	8	5	6

Solución:

Para la cota optimista, calculamos la mejor manera de enviar el paquete desde cada origen:

origen	1	2	3	4
coste	1	2	3	5

La cota inicial es $1+2+3+5 = 11$. Obtenemos una solución inicial utilizando el algoritmo voraz que elije, para cada origen, el mejor destino que quede disponible: $\hat{x} = (2, 4, 3, 1)$, $\hat{f} = 1+2+4+7 = 14$.

- $A_0 = \{\underline{(?)^{11}}\}$
- $A_1 = \{(1,?)^{13}, \underline{(2,?)^{11}}, \cancel{(3,?)^{14}}, (4,?)^{12}\}$
- $A_2 = \{(1,?)^{13}, (2,1,?)^{12}, (2,3,?)^{13}, \underline{(2,4,?)^{11}}, (4,?)^{12}\}$
- $A_3 = \{(1,?)^{13}, (2,1,?)^{12}, (2,3,?)^{13}, (2,4,1,?)^{13}, \underline{(2,4,3,?)^{12}}, (4,?)^{12}\}$
- se genera $(2,4,3,1)$ que coincide con la solución voraz inicial
- $A_4 = \{(1,?)^{13}, \underline{(2,1,?)^{12}}, (2,3,?)^{13}, (2,4,1,?)^{13}, (4,?)^{12}\}$
- $A_5 = \{(1,?)^{13}, (2,1,3,?)^{13}, \underline{(2,1,4,?)^{12}}, (2,3,?)^{13}, (2,4,1,?)^{13}, (4,?)^{12}\}$
- se genera $x = (2,1,4,3)$ con valor $f(x) = 12$ que mejora la mejor solución hasta el momento, con lo que $\hat{x} = (2,1,4,3)$ y $\hat{f} = 12$ y procedemos a realizar la poda explícita:
- $A_6 = \{\cancel{(1,?)^{13}}, \cancel{(2,1,3,?)^{13}}, \cancel{(2,3,?)^{13}}, \cancel{(2,4,1,?)^{13}}, \cancel{(4,?)^{12}}\} = \{\}$
  Como la lista de estados activos queda vacía, el algoritmo termina ☺, la mejor solución tiene valor 12.