IIC2283 — Diseño y Análisis de Algoritmos
Ayudantía 7

Problema 1: Problema de la mochila fraccionario

En el problema de la mochila se tienen como input n objetos, cada uno con peso w_i y valor v_i, además de una capacidad máxima C. La versión fraccionaria del problema consiste en encontrar valores $0 \leq f_i \leq 1$, tales que maximicen:

$$\sum_i f_i \cdot v_i,$$

sujeto a que:

$$\sum_i f_i \cdot w_i \leq C.$$

Describa y analice un algoritmo eficiente que resuelva este problema.

Idea de solución Algoritmo greedy. Se ordenan los items por v_i/w_i decreciente, y luego se agregan en ese orden a la mochila hasta completar la capacidad, o bien hasta agregar todos los objetos.

```
function GreedyKnapsack(C, w⃗, v⃗)
    Ordenar items por v/w decreciente
    F ← 0⃗
    R ← C
    for i = 1..n do
        if w_i ≤ R then
            F[i] ← 1
            R ← R − w_i
        else
            F[i] ← R/w_i
            R ← 0                          ▷ También se puede retornar F directamente
        end if
    end for
    return F
end function
```

Para demostrar la optimalidad de la solución se debe justificar que existe una solución óptima que utiliza la mayor fracción posible del primer objeto, por inducción en la cantidad de objetos. Si solo hay un objeto, y no se utiliza la mayor fracción posible entonces la mochila tiene capacidad disponible, generando una contradicción.

Si se tienen $n+1$ objetos, y hay una solución óptima que no utiliza la mayor fracción posible del primer objeto, entonces se considera si la mochila tiene capacidad disponible. Si tiene capacidad disponible, entonces la solución no es óptima. Si no tiene capacidad disponible, y no se está utilizando la mayor fracción posible del primer objeto, entonces existe algún objeto $j > 0$, con $f_j > 0$, y por lo tanto se puede reducir f_j, y aumentar f_1 para obtener otra solución con igual peso, pero con valor mayor o igual. Como la última solución utiliza una fracción mayor del primer objeto, se puede repetir el procedimiento hasta que la fracción utilizada del

primer objeto sea la mayor posible. Al cumplir esto, se utiliza la hipótesis de inducción considerando los objetos desde el segundo.

Problema 2: Vueltos con tres tipos de monedas

Parte 1 Describa y analice un algoritmo *greedy* que dada una cantidad ilimitada de monedas con denominaciones $\{1, 2, k\}$, para algún $k > 2$, permita dar un vuelto de valor C utilizando la menor cantidad posible de monedas.

Idea de solución Algoritmo greedy. Si denominamos n_1, n_2, n_k a la cantidad de monedas con denominación $1, 2, k$, entonces la solución se obtiene al resolver:

$$C = k \cdot n_k + r$$
$$r = 2 \cdot n_2 + n_1$$

donde $0 \leq r < k$, y $0 \leq n_1 < 2$.

La optimalidad se justifica en dos partes. Primero, se considera la solución greedy con denominaciones $\{1, 2\}$, donde se demuestra que es óptima por que si hay más de dos monedas de 1, entonces se pueden cambiar dos por una de 2, y obtener otra solución que utilice menos monedas.

En el caso general, se considera alguna solución $S = (n'_1, n'_2, n'_k)$ donde $n'_1 < 2$, y se muestra que si la solución greedy es $G = (n_1, n_2, n_k)$, entonces $n_1 + n_2 + n_k \leq n'_1 + n'_2 + n'_k$.

Si $S = G$, entonces el resultado es trivial. Si son distintas, pero en ambos casos $n_1 < 2$, entonces se tiene que $2n'_2 + n'_1 \geq k$, por lo que hay tres casos:

- Si $k = 2p$ es par, entonces $n'_2 \geq p$, y se pueden cambiar $\lfloor \frac{n_2}{p} \rfloor \cdot p$ monedas con denominación 2, por $\lfloor \frac{n_2}{p} \rfloor$ monedas de denominación k, obteniendo la solución greedy con una cantidad menor de monedas.
- Si $k = 2p + 1$ es impar, y $n'_1 = 1$, entonces nuevamente $n'_2 \geq p$, y se pueden cambiar p monedas de denominación 2 y una de denominación 1, por una moneda de k, obteniendo otra solución que tiene menos monedas en total, pero que utiliza más monedas de denominación k.
- Si $k = 2p + 1$ y $n'_1 = 0$, entonces $n'_2 \geq p + 1$, por lo que se pueden intercambiar $p + 1$ monedas con denominación 2, por una de denominación 1 más una de denominación k, obteniendo otra solución, con a lo más la misma cantidad de monedas, pero con más monedas de denominación k.

En el primer caso se obtiene la solución greedy directamente, y en los otros se puede repetir el procedimiento según el caso que corresponda, hasta obtener la solución greedy, sin aumentar la cantidad total de monedas. Finalmente, como esto funciona para cualquier solución S con $n'_1 < 2$, entonces se obtiene que la solución greedy es óptima.

Parte 2 Muestre que existen enteros c y d tales que si las denominaciones son $\{1, c, d\}$, entonces el algoritmo anterior no entrega la solución óptima.

Solución Con $\{1, 4, 5\}$, el algoritmo greedy utiliza 4 monedas para dar un vuelto de valor 8, cuando el óptimo son 2.

Parte 3 Demuestre que si $d = q \cdot c + r$, con $q > 0$ y $0 \leq r < c$, y además $0 < r < c - q$, entonces el algoritmo greedy no genera la solución óptima.

Idea de solución Se considera $T = c + d - 1$. Para ese valor, el algoritmo greedy genera la solución:

$$n_1 = c - 1, n_c = 0, n_d = 1,$$

con c monedas, mientras que también existe la solución:

$$n_1 = 0, n_c = q + 1, n_d = c - 1,$$

que utiliza $q + r$ monedas. Luego, si $r < c - q$, entonces $c < r + q$ y la solución greedy no es óptima.

Parte 4 Describa y analice un algoritmo que permita dar un vuelto óptimo, para valores c, d arbitrarios, en tiempo $O(C)$.

Idea de solución Por programación dinámica. Se define la cantidad mínima de monedas como:

$$N(C) = \begin{cases} \infty & C < 0 \\ 0 & C = 0 \\ 1 + \min\{N(C-1), N(C-c), N(C-d)\} & C > 0 \end{cases}$$

que luego se calcula de forma bottom-up por programación dinámica. En este caso, la subestructura óptima se obtiene por que una solución que incluye una moneda de denominación m debe incluir una solución óptima para $C - m$, por que de lo contrario se podrían intercambiar las soluciones, y obtener otra solución con una menor cantidad de monedas. Las cantidades de monedas de cada tipo se pueden obtener modificando el algoritmo.

Problema 3: Árbol de cobertura mínimo alternativo

Parte 1 Demuestre la siguiente propiedad: Dado un ciclo en un grafo, donde e es el arco de mayor peso del ciclo, entonces existe un MST que no contiene e.

Idea de solución Dado un MST que incluye el arco e, entonces al eliminarlo se desconecta el árbol, y se obtienen dos componentes. Luego se consideran los otros arcos del ciclo.Entre estos debe existir alguno con un extremo en cada componente, con peso menor o igual a e, y por lo tanto al agregarlo a las dos componentes obtenidas antes, se obtiene un árbol de cobertura. Como ese arco tiene peso menor o igual al de e, entonces el árbol obtenido es un árbol de cobertura con peso menor o igual al original, pero que no incluye al arco e.

Parte 2 Demuestre que el siguiente algoritmo es correcto:

```
function MST(G = (V, E))
    Ordenar los arcos por peso, en orden decreciente
    for cada arco e ∈ E do
        if e es parte de un ciclo then
            Remover e del grafo
        end if
    end for
    return G
end function
```

Idea de solución Por inducción en la cantidad de ciclos. Si el grafo no tiene ciclos, entonces el algoritmo funciona trivialmente.

Si el grafo tiene $n + 1$ ciclos, entonces al encontrar y eliminar el primer arco que pertenezca a un ciclo, se obtiene un grafo con menos de $n + 1$ ciclos (no necesariamente son n), en el cual el algoritmo funciona por hipótesis de inducción.

Parte 3 En cada iteración el algoritmo debe revisar si existe un ciclo que contenga un arco específico. Describa un algoritmo de tiempo lineal para este problema, justificando su corrección.

Idea de solución Eliminar el arco, y buscar un nodo desde el otro (por ejemplo con DFS).

Parte 4 Analice el tiempo de ejecución del algoritmo de la parte 2.

Idea de solución La búsqueda es de tiempo $O(|V| + |E|)$, y se debe realizar por cada arco, por lo tanto el algoritmo funciona en tiempo $O(|E|(|V| + |E|))$.

Problema 4: Puntos de articulación

Decimos que un grafo dirigido P es una *palmera* si sus arcos se pueden dividir en dos conjuntos disjuntos, denotados por $v \to w$ y $v \dashrightarrow w$, tales que:

- El subgrafo T obtenido con los arcos $v \to w$ es un árbol de cobertura de P,
- Los arcos $\dashrightarrow$ son un subconjunto de $(\overset{*}{\to})^{-1}$, es decir si $a \dashrightarrow b$, entonces $b \overset{*}{\to} a$

Parte 1 Demuestre que el grafo dirigido generado al aplicar DFS en un grafo no dirigido conexo es una palmera.

Para esto, considere la siguiente versión de DFS:

```
INTEGER i
function DFS(v, u)                    ▷ El nodo u es el padre de v en el árbol de cobertura
    i ← i + 1
    NUMBER(v) ← i
    for w en la lista de adyacencia de v do
        if w no tiene número then
            Agregar arco v → w a P
            DFS(w, v)
        else if NUMBER(w) < NUMBER(v) y w ≠ u then
            Agregar arco v ⇢ w a P
        end if
    end for
end function
i ← 0
DFS(s, 0)
```

Referencia Este problema está basado en el algoritmo para componentes biconexas descrito en [1], el cual está disponible en `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.327.8418`.

La parte 1 corresponde al teorema 1.

Un grafo $G = (V, E)$ se dice *biconexo* si para cualquier triple de nodos v, w, a en V, existe un camino entre v y w que no pasa por a. Adicionalmente, decimos que un nodo p es un *punto de articulación* si existen nodos s, t tales que todo camino entre s y t pasa por p.

Luego, definimos una relación de equivalencia en el conjunto de arcos, donde dos arcos son equivalentes ssi pertenecen a algún ciclo en común, tras lo cual consideramos sus clases de equivalencia. Dada la clase E_i, definimos G_i como el subgrafo $G_i = (V_i, E_i)$, donde V_i son los vértices adyacentes a algún arco de E_i.

Parte 2 Demuestre las siguientes afirmaciones:

1. G_i es biconexo para todo i.
2. No existe ningún subgrafo biconexo de G, que tenga algún G_i como subgrafo propio.
3. Cada punto de articulación aparece más de una vez entre los V_i, y cada punto que no es de articulación aparece exactamente una vez.
4. El conjunto $V_i \cap V_j$ contiene a lo más un punto para $i \neq j$. Tal punto es un punto de articulación.

Solución Esta parte corresponde al lema 3 de la referencia, pero no están las demostraciones:

- Por definición. Dados tres nodos del subgrafo, u, a, w, entonces hay dos casos: ningún camino entre u y w pasa por a, o existe un camino por ese nodo. Si existe el camino, entonces se consideran los dos arcos en torno a, los que son equivalentes, y por lo tanto parte de un ciclo. Luego, se pueden reemplazar ambos arcos por el resto del camino, obteniendo otro camino entre u y w que no pasa por a.
- Por contradicción. Si existe un grafo G biconexo, tal que G_i sea un subgrafo propio de G, entonces debe existir algún arco entre un nodo a de G_i y un nodo v de $G \backslash G_i$. Como G_i se construyó desde una clase de equivalencia de arcos, entonces debe existir algún nodo $w \in G_i$ conectado con a. Luego, como G es biconexo, existe un camino entre v y w que no pasa por a, y por lo tanto, los arcos v, a y a, w pertenecen a un ciclo común, lo que genera una contradicción, por que $v \notin G_i$.
- Si a es un punto de articulación, entonces por definición existen nodos v, w tales que todo camino entre v y w pasa por a. Si consideramos los dos arcos en torno a a en alguno de esos caminos, estos no pueden ser equivalentes, por que en ese caso se podría construir otro camino que no pasa por a. Luego, ambos arcos pertenecen a clases de equivalencia distintas, entonces el nodo a pertenece a al menos dos componentes.

  Por otra parte, si un nodo a no es de articulación, hay dos casos. El nodo tiene grado 1, en cuyo caso solo pertenece a una componente, formada solo por ese arco, o el nodo tiene grado mayor a 1. Si consideramos dos arcos adyacentes a a –(a, v) y (a, w)–, entonces estos deben ser equivalentes, por de lo contrario sería un punto de articulación al considerar los nodos v, w.
- Si el conjunto $V_i \cap V_j$ contiene dos nodos distintos, entonces por cada uno hay al menos un arco hacia V_i y otro hacia V_j. Si consideramos los dos pares de arcos, los dos arcos hacia V_i son equivalentes, y por lo tanto pertenecen a algún ciclo común, y de igual forma, los dos arcos hacia V_j también son equivalentes. Al considerar ambos, se puede construir un ciclo que cubre los 4 arcos, mostrando que son equivalentes entre sí, y obteniendo una contradicción, por que están en clases de equivalencia distintas.

  El punto es de articulación por que se consideran los arcos hacia cada componentes, y luego el único camino entre esos nodos es a través de a.

Los subgrafos G_i descritos antes se conocen como las *componentes biconexas de G*. Luego, si P es la palmera obtenida al aplicar DFS en el grafo, y enumeramos los vértices en el orden en que se revisan, entonces para cada vértice v definimos $\text{LOWPT}(v)$ como:

$$\text{LOWPT}(v) = \text{mín}\{\text{NUMBER}(w) | w = v \text{ ó } v \xrightarrow{*} \cdot \dashrightarrow w\}$$

Parte 3 Demuestre que si G es un grafo no dirigido, P la palmera obtenida desde G al aplicar DFS, y T el árbol de cobertura de P, entonces: Si $p : v \overset{*}{\Rightarrow} w$ es un camino en G, entonces p contiene un punto que es un antecesor de v y w en T.

Referencia Corresponde al lema 4.

Parte 4 Sea G un grafo no dirigido, P la palmera obtenida desde G al aplicar DFS, y T el árbol de cobertura de P. Demuestre que si a, v, w son vértices distintos de G tales que $(a, v) \in T$, y w no es un descendiente de v en T, entonces si $\text{LOWPT}(v) \geq a$, entonces a es un punto de articulación de G, y eliminarlo desconecta v y w. Además, si a es un punto de articulación, entonces existen vértices v y w que cumplen las propiedades anteriores.

Referencia Corresponde al lema 5.

Parte 5 Demuestre que el siguiente algoritmo encuentra los puntos de articulación de un grafo no dirigido:

```
INTEGER i
SET pts ← ∅
function ArticulationPoints(G)
    for cada vértice w de G do
        if w no tiene número then
            ArticulationPoints(G, w, 0)
        end if
    end for
end function
function ArticulationPoints(G, v, u)
    i ← i + 1
    NUMBER(v) ← i
    for cada nodo w en la lista de adyacencia de v do
        if w no tiene número then
            ArticulationPoints(G, w, v)
            LOWPT(v) ← mín(LOWPT(v), LOWPTw)
            if LOWPT(v) ≥ NUMBER(v) then
                pts ← pts ∪ {v}
            end if
        else if NUMBER(w) < NUMBER(v) and w ≠ u then
            LOWPT(v) ← mín(LOWPT(v), NUMBER(w))
        end if
    end for
end function
```

Solución Corolario del teorema 8.

Referencias

[1] R. E. Tarjan, *Depth-first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.