Auxiliar 1 - Algoritmos Greedy

CC4102 - Diseño y Análisis de Algoritmos
Profesor: Pablo Barceló Auxiliar: Jorge Bahamonde

25 de Marzo del 2015

1 Codificación de Huffman

Se tienen reales $p_1 ... p_n$ tales que $\sum_{i=1}^{n} p_i = 1$. Construya un árbol binario que tenga estos valores en sus hojas y que minimice $\sum_{i=1}^{n} d_i p_i$, donde d_i es la profundidad de la hoja con valor p_i.

2 Fórmulas de Horn

Una fórmula de Horn es un conjunto de cláusulas que pueden tomar las siguientes formas:

- Cláusulas de *implicancia*: $(u_1 \wedge u_2 \wedge ... \wedge u_n) \implies v$
- Cláusulas de *disyunción*: $(\bar{u_1} \vee \bar{u_2} \vee ... \vee \bar{u_n})$

con u_i, v variables booleanas.

Construya un algoritmo que determine si existe una asignación de valores para las variables de una fórmula de Horn que haga verdaderas todas las cláusulas de ésta, y encuentre una de estas asignaciones, de existir.

3 Guardias en un Pasillo

Se tienen N cuadros invaluables en el pasillo de un museo, ubicados en los puntos $X = x_1, ...x_N$ de una recta. El director del museo quiere protegerlos a toda costa, pero se encuentra con algunos inconvenientes.

Para empezar, los guardias piden un sueldo altísimo, por lo que quisiera minimizar la cantidad de guardias a contratar. Por otro lado, los guardias resultan ser miopes, por lo que sólo pueden proteger los cuadros que estén a distancia 1 o menor (es decir, un guardia ubicado en un punto y protegerá los cuadros en $[y-1, y+1]$).

Construya un algoritmo que determine la cantidad óptima y posición de los guardias a contratar.

1 Codificación de Huffman

La idea del algoritmo es la siguiente:

- Se parte considerando cada peso p_i como un nodo (suelto) del árbol a construir.
- Mientras tomar los dos árboles i y j del conjunto cuyas raíz tengan los mayores pesos. Unirlos como los dos hijos de un nodo con peso $r_i + r_j$, donde r_i y r_j son los pesos guardados en las raíces i y j.
- El algoritmo termina cuando sólo se tiene un árbol en el conjunto

De manera un poco más formal:

---

function HUFFMAN($P = (p_1, ..., p_N)$)
  $T \leftarrow v_1, ..., v_N$ ▷ N árboles (nodos, por ahora)
  while $|T| > 1$ **do**
    $n_i, n_j \leftarrow$ MIN2(T) ▷ Extrae los dos árboles con los menores valores p_i, p_j, en sus raíces
    $n_{new} \leftarrow$ NEWNODE($p_i + p_j, n_i, n_j$)
    $T \leftarrow T \cup \{n_{new}\}$
  end while
  return t ▷ Donde $T = \{t\}$
end function

Es fácil notar que el algoritmo produce un árbol con los pesos p_i en las hojas. Es necesario, sí, demostrar que minimiza el costo $C = \sum_{i=1}^{n} d_i p_i$ asociado.

Demostración. Por inducción en n (el número de hojas a ubicar en el árbol).

- $n = 2$

En este caso, sólo existen dos hojas con pesos p_1 y p_2, por lo que los únicos árboles posibles están compuestos por un nodo raíz con los nodos p_1 y p_2 como hijos. Sólo existen dos árboles de este tipo, ambos con p_1 y p_2 como hijos de un nodo raíz (y por lo tanto poseen el mismo costo). El algoritmo produce uno de estos árboles, que posee el costo mínimo.

- $n \implies n + 1$

Supongamos que para cualquier conjunto de n pesos, el algoritmo produce el árbol con costo óptimo. Por contradicción, supongamos que el árbol T generado por el algoritmo para $n+1$ pesos tiene un costo mayor al óptimo.

Sean $p_1 ... p_{n+1}$ pesos a ubicar en un árbol. Sin pérdida de generalidad, se puede suponer que se encuentran ordenados. De esta forma, el algoritmo partirá por unir p_1 y p_2 a través de un padre V. Es claro que en el árbol T que el algoritmo produzca, p_1 y p_2 serán los nodos con la máxima profundidad d_{MAX} en el árbol.

Sea T' un árbol idéntico a T salvo que V es reemplazado por una única hoja con valor $p_1 + p_2$. Notemos que este árbol es el árbol producido por el algoritmo propuesto si reemplazamos p_1 y p_2 por $p_1 + p_2$ en P. Luego el costo de T' es óptimo, gracias a la hipótesis de inducción (el nuevo conjunto P tendría n elementos).

Los costos de T y T' cumplen la siguiente relación:

$$
\begin{aligned}
C(T) &= \sum_{i=3}^{n+1} d_i p_i + d_{\text{MAX}} p_1 + d_{\text{MAX}} p_2 \\
&= \sum_{i=3}^{n+1} d_i p_i + (d_V + 1)(p_1 + p_2) \\
&= \sum_{i=3}^{n+1} d_i p_i + d_V (p_1 + p_2) + (p_1 + p_2) \\
&= C(T') + (p_1 + p_2)
\end{aligned}
$$

En este punto se hace necesario

Lema 1. *Para cualquier conjunto de pesos $p_1 ... p_n$, con $n > 2$, existe un árbol que alcanza el costo óptimo en el que p_1 y p_2 son dos hojas hermanas con la mayor profundidad del árbol.*

Demostración. Supongamos que existe un árbol A y un $k > 2$ tal que $d_k > d_1$ en A (el caso con d_2 es análogo). Dado que $p_k >= p_1$, el árbol A' obtenido al intercambiar las hojas 1 y k poseerá un costo menor o igual al de A.

Por otro lado, supongamos se tiene un árbol óptimo en el que las hojas p_1 y p_2 no son hermanas. De lo anterior, puede llevarse este árbol a uno en que ambas hojas poseen la altura máxima de acuerdo al párrafo anterior. Si estas hojas no son hermanas, existe una tercera hoja con la misma profundidad que alguna de estas dos, que puede intercambiarse con una de las hojas p_1 y p_2 de modo de "hermanarlas". □

Así, el mismo proceso que se realizó sobre T puede realizarse sobre un árbol Z que también tiene a p_1 y p_2 como hojas hermanas de máxima profundidad, de modo que Z tiene un costo óptimo. Entonces se obtiene Z' tal que

$$C(Z) = C(Z') + (p_1 + p_2)$$

Luego, y utilizando el hecho de que T no posee el costo óptimo:

$$
\begin{aligned}
C(T') &= C(T) - p_1 - p_2 \\
&< C(Z) - p_1 - p_2 \\
&= C(Z') \\
&= C(T')
\end{aligned}
$$

pues T' sí tiene el costo óptimo (dada la hipótesis de inducción). Esto es una contradicción ($C(T') < C(T')$), por lo que T debe tener el costo óptimo.

□

2 Fórmulas de Horn

El algoritmo consiste en partir asignando F a todas las variables (lo que hace que todas las cláusulas de disyunción sean satisfechas); a continuación, para cada cláusula de implicancia que no esté ya satisfecha se setea la variable a la derecha de la implicancia como V (lo que hace que se cumpla).

Luego de hacer esto para cada cláusula de implicancia, se chequean las cláusulas de disyunción. Si todas se encuentran satisfechas, el algoritmo retorna la asignación generada; de lo contrario, el algoritmo responde que no existe una asignación válida.

Un poco más formalmente:

function HORN
  $u \leftarrow \text{F},\ \forall u$
  while $\exists C_{\text{implicancia}} = (u_1 \wedge u_2 \wedge ... \wedge u_n) \implies v$ y $v = \text{F}$ **do**
    if $u_i = \text{V},\ \forall i$ **then**
      $v \leftarrow \text{V}$
    end if
  end while
  for each $C_{\text{disyuncion}} = (\bar{u_1} \vee \bar{u_2} \vee ... \vee \bar{u_n})$ **do**
    if $(\bar{u_1} \vee \bar{u_2} \vee ... \vee \bar{u_n}) = \text{F}$ **then**
      return $\perp$
    end if
  end for
  return $u_i, \forall i$ ▷ Se retorna todo el conjunto de variables con sus valores.
end function

Demostración del funcionamiento del algoritmo. Para empezar, el primer ciclo termina, pues en cada iteración se setea una variable como verdadera; así, se hace una cantidad de iteraciones a lo más igual a la cantidad total de variables en la fórmula de Horn.

Es fácil notar que si el algoritmo retorna una asignación, ésta satisface todas las cláusulas, ya que todas las asignaciones y chequeos que realiza imponen esto. Es necesario demostrar, sí, que el algoritmo está en lo correcto cuando responde que no existe una asignación válida.

Se mostrará lo siguiente:

Lema 2. *Para toda fórmula de Horn, cualquier variable que el algoritmo fije como* V *debe tomar este valor en todas las asignaciones que satisfacen la fórmula.*

Demostración. Se hará inducción en el número de pasos del ciclo que asigna valores V a las variables.

- $n = 1$

La primera vez que el algoritmo fija una variable como V, lo hace para una cláusula de la forma $\implies u$ (pues todas las variables tienen valores F). Es claro que todas las posibles asignaciones de valores de verdad que hagan válida la fórmula de Horn tendrán esta variable con un valor V, pues es la única forma de satisfacer la cláusula $\implies u$.

- $n \implies n + 1$

Supongamos en un ciclo $n + 1$ el algoritmo se encuentra con una cláusula del tipo $(u_1 \wedge u_2 \wedge ... \wedge u_k) \implies v$ que no se encuentra satisfecha. Esto implica que a $u_1...u...k$ se les asignó V en iteraciones anteriores del algoritmo. Por hipótesis de inducción, sabemos que todas estas variables son verdaderas en *todas* las asignaciones válidas de valores para la fórmula. Luego necesariamente v debe tomar el valor V en todas las asignaciones válidas para que la cláusula se cumpla. □

Supongamos que el algoritmo responde que no existe asignación posible. Esto sucederá porque una cláusula de disyunción C no se cumple, debido a que todas sus variables fueron seteadas como verdaderas por el algoritmo. Por el lema mostrado, todas las posibles asignaciones válidas para los variables de la fórmula *deben* asignar verdadero a esas variables. Luego la cláusula C no puede ser cumplida, pues en toda asignación válida sus variables tomarán un valor verdadero.

□

3 Guardias en un Pasillo

El algoritmo consiste en barrer el pasillo, ubicando un guardia a una distancia 1 a la derecha del primer cuadro desprotegido (buscando "aprovechar al máximo" el nuevo guardia), para luego seguir barriendo.

---

function WARD($X = \{x_1, ..., x_N\}$) ▷ s.p.g, están ordenados.
  $G \leftarrow \emptyset$
  while $x_j \leftarrow$ MIN(UNWARDED(X)) **do**
    $G \leftarrow G \cup \{(x_k + 1)\}$
  end while
end function

Es claro que el algoritmo logra proteger todos los cuadros, pues en cada iteración protege al menos uno de los cuadros desprotegidos. Es necesario mostrar que el algoritmo posiciona la cantidad óptima (es decir, mínima) de guardias.

Demostración. Sea G el conjunto de guardias generado por el algoritmo y G' un conjunto con el mínimo número de guardias necesario.

Supongamos que $|G| > |G'|$. Supongamos que G y G' se encuentran ordenados. Es claro que G y G' deben discrepar en algún elemento (es decir, existen $g_j \in G, g'_j \in G'$ tales que $g_j \neq g'_j$). De lo contrario, G' simplemente tendría menos guardias que G. Pero quitar un guardia de G dejará al menos un cuadro sin proteger, por lo que G' no sería una solución correcta.

Distinguimos dos casos:

- $g_j < g'_j$

Nótese que de acuerdo al algoritmo mostrado, el guardia g_j está ubicado de modo de proteger el cuadro ubicado en $g_j - 1$; además, ninguno de los guardias en $g_1...g_{j-1}$ pueden ver este cuadro. Dado que el guardia j es el primero para el cual hay una discrepancia entre G y G', tampoco los guardias en $g'_1...g'_{j-1}$ pueden ver este cuadro. Finalmente, el guardia en g'_j (ni los que le siguen) tampoco puede ver el cuadro, pues $g'_j > g_j$. Luego G' no puede ser una solución correcta.

- $g_j > g'_j$

En este caso, es claro que los guardias en $g_1...g_j$ son capaces de proteger tantos o más cuadros que los guardias ubicados en $g'_1...g'_j$. En otras palabras, el guardia en g'_j puede ser reemplazado por uno en g_j manteniendo la optimalidad. Así, puede reemplazarse G' por un G'' que también tiene una cantidad mínima de guardias y en el que $g''_j = g_j$.

Este proceso puede ser repetido, hasta que todos los guardias de G coincidan con los guardias de una solución óptima. Luego G es una solución óptima al problema.

□