Algoritmos codiciosos

Los algoritmos codiciosos son usados, en general, para resolver problemas de optimización.

Algoritmos codiciosos

Los algoritmos codiciosos son usados, en general, para resolver problemas de optimización.

El principio fundamental de este tipo de algoritmos es lograr un óptimo global tomando decisiones locales que son óptimas.

Algoritmos codiciosos

Los algoritmos codiciosos son usados, en general, para resolver problemas de optimización.

El principio fundamental de este tipo de algoritmos es lograr un óptimo global tomando decisiones locales que son óptimas.

- En general, la intuición detrás de estos algoritmos es simple

Algoritmos codiciosos

Los algoritmos codiciosos son usados, en general, para resolver problemas de optimización.

El principio fundamental de este tipo de algoritmos es lograr un óptimo global tomando decisiones locales que son óptimas.

- En general, la intuición detrás de estos algoritmos es simple

Dos componentes fundamentales de un algoritmo codicioso:

- Una función objetivo a minimizar o maximizar
- Una función de selección usada para tomar decisiones locales óptimas

Algoritmos codiciosos

Lamentablemente en muchos casos los algoritmos codiciosos no encuentran un óptimo global

Algoritmos codiciosos

Lamentablemente en muchos casos los algoritmos codiciosos no encuentran un óptimo global

- Sin embargo, en muchos casos pueden ser usados como heurísticas para encontrar valores de la función objetivo cercanos al óptimo

Algoritmos codiciosos

Lamentablemente en muchos casos los algoritmos codiciosos no encuentran un óptimo global

- Sin embargo, en muchos casos pueden ser usados como heurísticas para encontrar valores de la función objetivo cercanos al óptimo

Una segunda dificultad con los algoritmos codiciosos es que, en general, se necesita de una demostración formal para asegurar que encuentran el óptimo global

Algoritmos codiciosos

Lamentablemente en muchos casos los algoritmos codiciosos no encuentran un óptimo global

- Sin embargo, en muchos casos pueden ser usados como heurísticas para encontrar valores de la función objetivo cercanos al óptimo

Una segunda dificultad con los algoritmos codiciosos es que, en general, se necesita de una demostración formal para asegurar que encuentran el óptimo global

- Aunque pueden ser algoritmos simples, en algunos casos no es obvio por qué encuentran el óptimo global

Algoritmos codiciosos

Lamentablemente en muchos casos los algoritmos codiciosos no encuentran un óptimo global

- Sin embargo, en muchos casos pueden ser usados como heurísticas para encontrar valores de la función objetivo cercanos al óptimo

Una segunda dificultad con los algoritmos codiciosos es que, en general, se necesita de una demostración formal para asegurar que encuentran el óptimo global

- Aunque pueden ser algoritmos simples, en algunos casos no es obvio por qué encuentran el óptimo global

Vamos a ver un ejemplo clásico de algoritmos codiciosos.

Almacenamiento de datos

Dada una palabra w sobre un alfabeto Σ, suponga que queremos almacenar w utilizando los símbolos 0 y 1

Almacenamiento de datos

Dada una palabra w sobre un alfabeto Σ, suponga que queremos almacenar w utilizando los símbolos 0 y 1

- Esta palabra puede ser pensada como un texto si Σ incluye las letras del alfabeto español, símbolos de puntuación y el espacio, por lo tanto puede ser muy larga
- El uso de 0 y 1 corresponde a la idea de almacenar el texto en un computador

Almacenamiento de datos

Dada una palabra w sobre un alfabeto Σ, suponga que queremos almacenar w utilizando los símbolos 0 y 1

- Esta palabra puede ser pensada como un texto si Σ incluye las letras del alfabeto español, símbolos de puntuación y el espacio, por lo tanto puede ser muy larga
- El uso de 0 y 1 corresponde a la idea de almacenar el texto en un computador

Definimos entonces una función $\tau : \Sigma \to \{0,1\}^*$ que asigna a cada símbolo en $a \in \Sigma$ una palabra en $\tau(a) \in \{0,1\}^*$ con $\tau(a) \neq \varepsilon$

Almacenamiento de datos

Dada una palabra w sobre un alfabeto Σ, suponga que queremos almacenar w utilizando los símbolos 0 y 1

- Esta palabra puede ser pensada como un texto si Σ incluye las letras del alfabeto español, símbolos de puntuación y el espacio, por lo tanto puede ser muy larga
- El uso de 0 y 1 corresponde a la idea de almacenar el texto en un computador

Definimos entonces una función $\tau : \Sigma \to \{0,1\}^*$ que asigna a cada símbolo en $a \in \Sigma$ una palabra en $\tau(a) \in \{0,1\}^*$ con $\tau(a) \neq \varepsilon$

- Vamos a almacenar w reemplazando cada símbolo $a \in \Sigma$ que aparece en w por $\tau(a)$

Almacenamiento de datos

Dada una palabra w sobre un alfabeto Σ, suponga que queremos almacenar w utilizando los símbolos 0 y 1

- Esta palabra puede ser pensada como un texto si Σ incluye las letras del alfabeto español, símbolos de puntuación y el espacio, por lo tanto puede ser muy larga
- El uso de 0 y 1 corresponde a la idea de almacenar el texto en un computador

Definimos entonces una función $\tau : \Sigma \to \{0,1\}^*$ que asigna a cada símbolo en $a \in \Sigma$ una palabra en $\tau(a) \in \{0,1\}^*$ con $\tau(a) \neq \varepsilon$

- Vamos a almacenar w reemplazando cada símbolo $a \in \Sigma$ que aparece en w por $\tau(a)$
- Llamamos a τ una Σ-codificación

Almacenamiento de datos

La extensión $\hat{\tau}$ de una Σ-codificación τ a todas las palabras $w \in \Sigma^*$ se define como:

$$\hat{\tau}(w) \quad = \quad \begin{cases} \varepsilon & w = \varepsilon \\ \tau(a_1) \cdots \tau(a_n) & w = a_1 \cdots a_n \text{ con } n \geq 1 \end{cases}$$

Almacenamiento de datos

La extensión $\hat{\tau}$ de una Σ-codificación τ a todas las palabras $w \in \Sigma^*$ se define como:

$$\hat{\tau}(w) = \begin{cases} \varepsilon & w = \varepsilon \\ \tau(a_1) \cdots \tau(a_n) & w = a_1 \cdots a_n \text{ con } n \geq 1 \end{cases}$$

Vamos a almacenar w como $\hat{\tau}(w)$

Almacenamiento de datos

La extensión $\hat{\tau}$ de una Σ-codificación τ a todas las palabras $w \in \Sigma^*$ se define como:

$$\hat{\tau}(w) = \begin{cases} \varepsilon & w = \varepsilon \\ \tau(a_1) \cdots \tau(a_n) & w = a_1 \cdots a_n \text{ con } n \geq 1 \end{cases}$$

Vamos a almacenar w como $\hat{\tau}(w)$

- Si la Σ-codificación τ está fija (como el código ASCII) entonces no es necesario almacenarla

Almacenamiento de datos

La extensión $\hat{\tau}$ de una Σ-codificación τ a todas las palabras $w \in \Sigma^*$ se define como:

$$\hat{\tau}(w) = \begin{cases} \varepsilon & w = \varepsilon \\ \tau(a_1) \cdots \tau(a_n) & w = a_1 \cdots a_n \text{ con } n \geq 1 \end{cases}$$

Vamos a almacenar w como $\hat{\tau}(w)$

- Si la Σ-codificación τ está fija (como el código ASCII) entonces no es necesario almacenarla
- Si τ no está fija entonces debemos almacenarla junto con $\hat{\tau}(w)$
  - En general $|w|$ es mucho más grande que $|\Sigma|$, por lo que el costo de almacenar τ es despreciable

Almacenamiento de datos

La función $\hat{\tau}$ debe especificar una traducción no ambigua: si $w_1 \neq w_2$, entonces $\hat{\tau}(w_1) \neq \hat{\tau}(w_2)$

- De esta forma podemos reconstruir el texto original dada su traducción

Almacenamiento de datos

La función $\hat{\tau}$ debe especificar una traducción no ambigua: si $w_1 \neq w_2$, entonces $\hat{\tau}(w_1) \neq \hat{\tau}(w_2)$

- De esta forma podemos reconstruir el texto original dada su traducción

Vale decir, $\hat{\tau}$ debe ser una función inyectiva

Almacenamiento de datos

La función $\hat{\tau}$ debe especificar una traducción no ambigua: si $w_1 \neq w_2$, entonces $\hat{\tau}(w_1) \neq \hat{\tau}(w_2)$

- De esta forma podemos reconstruir el texto original dada su traducción

Vale decir, $\hat{\tau}$ debe ser una función inyectiva

- ¿Cómo podemos lograr esta propiedad?

Codificaciones de largo fijo

Obviamente para lograr una traducción no ambigua necesitamos que $\tau(a) \neq \tau(b)$ para cada $a, b \in \Sigma$ tal que $a \neq b$

Codificaciones de largo fijo

Obviamente para lograr una traducción no ambigua necesitamos que $\tau(a) \neq \tau(b)$ para cada $a, b \in \Sigma$ tal que $a \neq b$

Para obtener la misma propiedad para $\hat{\tau}$ podemos imponer la siguiente condición: para cada $a, b \in \Sigma$ se tiene que $|\tau(a)| = |\tau(b)|$

Codificaciones de largo fijo

Obviamente para lograr una traducción no ambigua necesitamos que $\tau(a) \neq \tau(b)$ para cada $a, b \in \Sigma$ tal que $a \neq b$

Para obtener la misma propiedad para $\hat{\tau}$ podemos imponer la siguiente condición: para cada $a, b \in \Sigma$ se tiene que $|\tau(a)| = |\tau(b)|$

- Decimos entonces que τ es una Σ-codificación de largo fijo

Codificaciones de largo fijo

Obviamente para lograr una traducción no ambigua necesitamos que $\tau(a) \neq \tau(b)$ para cada $a, b \in \Sigma$ tal que $a \neq b$

Para obtener la misma propiedad para $\hat{\tau}$ podemos imponer la siguiente condición: para cada $a, b \in \Sigma$ se tiene que $|\tau(a)| = |\tau(b)|$

- Decimos entonces que τ es una Σ-codificación de largo fijo

Ejercicio

Muestre que para tener una Σ-codificación de largo fijo basta con asignar a cada $a \in \Sigma$ una palabra $\tau(a) \in \{0,1\}^*$ tal que $|\tau(a)| = \lceil \log_2(|\Sigma|) \rceil$

Codificaciones de largo variable

Decimos que τ es una Σ-codificación de largo variable si existen $a, b \in \Sigma$ tales que $|\tau(a)| \neq |\tau(b)|$

Codificaciones de largo variable

Decimos que τ es una Σ-codificación de largo variable si existen $a, b \in \Sigma$ tales que $|\tau(a)| \neq |\tau(b)|$

¿Por qué nos conviene utilizar codificaciones de largo variable?

Codificaciones de largo variable

Decimos que τ es una Σ-codificación de largo variable si existen $a, b \in \Sigma$ tales que $|\tau(a)| \neq |\tau(b)|$

¿Por qué nos conviene utilizar codificaciones de largo variable?

- Podemos obtener representaciones más cortas para la palabra que queremos almacenar

Codificaciones de largo variable

Ejemplo

Suponga que $w = aabaacaab$

- Para $\tau_1(a) = 00$, $\tau_1(b) = 01$ y $\tau_1(c) = 10$ tenemos que:

$$\hat{\tau}_1(w) \quad = \quad 000001000010000001$$

- Para $\tau_2(a) = 0$, $\tau_2(b) = 10$ y $\tau_2(c) = 11$ tenemos que:

$$\hat{\tau}_2(w) \quad = \quad 001000110010$$

Por lo tanto $|\hat{\tau}_2(w)| = 12 < 18 = |\hat{\tau}_1(w)|$

Codificaciones de largo variable

Ejemplo

Suponga que $w = aabaacaab$

- Para $\tau_1(a) = 00$, $\tau_1(b) = 01$ y $\tau_1(c) = 10$ tenemos que:

$$\hat{\tau}_1(w) \quad = \quad 000001000010000001$$

- Para $\tau_2(a) = 0$, $\tau_2(b) = 10$ y $\tau_2(c) = 11$ tenemos que:

$$\hat{\tau}_2(w) \quad = \quad 001000110010$$

Por lo tanto $|\hat{\tau}_2(w)| = 12 < 18 = |\hat{\tau}_1(w)|$

Pero nos falta responder una pregunta: ¿por qué $\hat{\tau}_2$ es inyectiva?

Codificaciones de largo variable

Lema

Si existen $w_1, w_2 \in \Sigma^*$ *tales que* $w_1 \neq w_2$ *y* $\hat{\tau}(w_1) = \hat{\tau}(w_2)$*, entonces existen* $a, b \in \Sigma$ *tales que* $a \neq b$ *y* $\tau(a)$ *es un prefijo de* $\tau(b)$

Codificaciones de largo variable

Lema

Si existen $w_1, w_2 \in \Sigma^*$ *tales que* $w_1 \neq w_2$ *y* $\hat{\tau}(w_1) = \hat{\tau}(w_2)$*, entonces existen* $a, b \in \Sigma$ *tales que* $a \neq b$ *y* $\tau(a)$ *es un prefijo de* $\tau(b)$

Demostración: Suponga que $w_1 \neq w_2$, $\hat{\tau}(w_1) = \hat{\tau}(w_2)$ y

$$\begin{array}{lcll} w_1 & = & a_1 \dots a_m & m \geq 1 \\ w_2 & = & b_1 \dots b_n & n \geq 1 \end{array}$$

Además, sin perdida de generalidad suponga que $m \leq n$

Si w_1 es prefijo propio de w_2 entonces $\hat{\tau}(w_1)$ es prefijo propio de $\hat{\tau}(w_2)$

- Puesto que $\tau(a) \neq \varepsilon$ para cada $a \in \Sigma$

Dado que $\hat{\tau}(w_1) = \hat{\tau}(w_2)$, tenemos entonces que w_1 no es prefijo propio de w_2

Codificaciones de largo variable

Sea $k = \min_{1 \leq i \leq m} a_i \neq b_i$

- k está bien definido puesto que $w_1 \neq w_2$ y w_1 no es prefijo propio de w_2

Dado que $\hat{\tau}(a_1 \cdots a_{k-1}) = \hat{\tau}(b_1 \cdots b_{k-1})$ y $\hat{\tau}(w_1) = \hat{\tau}(w_2)$, concluimos que $\hat{\tau}(a_k \cdots a_m) = \hat{\tau}(b_k \cdots b_n)$

Tenemos entonces que $\tau(a_k) \cdots \tau(a_m) = \tau(b_k) \cdots \tau(b_n)$, de lo cual se deduce que $\tau(a_k)$ es un prefijo de $\tau(b_k)$ o $\tau(b_k)$ es un prefijo de $\tau(a_k)$

- Lo cual concluye la demostración puesto que $a_k \neq b_k$ □

Codificaciones libres de prefijos

Decimos que una Σ-codificación τ es libre de prefijos si para cada $a, b \in \Sigma$ tales que $a \neq b$ se tiene que $\tau(a)$ no es un prefijo de $\tau(b)$

Codificaciones libres de prefijos

Decimos que una Σ-codificación τ es libre de prefijos si para cada $a, b \in \Sigma$ tales que $a \neq b$ se tiene que $\tau(a)$ no es un prefijo de $\tau(b)$

Ejemplo

Para $\Sigma = \{a, b, c\}$ y $\tau(a) = 0$, $\tau(b) = 10$, $\tau(c) = 11$, se tiene que τ es libre de prefijos.

Codificaciones libres de prefijos

Decimos que una Σ-codificación τ es libre de prefijos si para cada $a, b \in \Sigma$ tales que $a \neq b$ se tiene que $\tau(a)$ no es un prefijo de $\tau(b)$

Ejemplo

Para $\Sigma = \{a, b, c\}$ y $\tau(a) = 0$, $\tau(b) = 10$, $\tau(c) = 11$, se tiene que τ es libre de prefijos.

Corolario

Si τ es una codificación libre de prefijos, entonces $\hat{\tau}$ es una función inyectiva.

Frecuencias relativas de los símbolos

Palabra a almacenar: $w \in \Sigma^*$

Frecuencias relativas de los símbolos

Palabra a almacenar: $w \in \Sigma^*$

Para $a \in \Sigma$ definimos $\text{fr}_w(a)$ como la frecuencia relativa de a en w, vale decir, el número de apariciones de a en w dividido por el largo de w

- Por ejemplo, si $w = aabaacaab$, entonces $\text{fr}_w(a) = \frac{2}{3}$ y $\text{fr}_w(c) = \frac{1}{9}$

Frecuencias relativas de los símbolos

Palabra a almacenar: $w \in \Sigma^*$

Para $a \in \Sigma$ definimos $\text{fr}_w(a)$ como la frecuencia relativa de a en w, vale decir, el número de apariciones de a en w dividido por el largo de w

- Por ejemplo, si $w = aabaacaab$, entonces $\text{fr}_w(a) = \frac{2}{3}$ y $\text{fr}_w(c) = \frac{1}{9}$

Para una Σ-codificación τ definimos el largo promedio para w como:

$$\text{lp}_w(\tau) \quad = \quad \sum_{a \in \Sigma} \text{fr}_w(a) \cdot |\tau(a)|$$

Frecuencias relativas de los símbolos

Palabra a almacenar: $w \in \Sigma^*$

Para $a \in \Sigma$ definimos $\text{fr}_w(a)$ como la frecuencia relativa de a en w, vale decir, el número de apariciones de a en w dividido por el largo de w

- Por ejemplo, si $w = aabaacaab$, entonces $\text{fr}_w(a) = \frac{2}{3}$ y $\text{fr}_w(c) = \frac{1}{9}$

Para una Σ-codificación τ definimos el largo promedio para w como:

$$\text{lp}_w(\tau) \quad = \quad \sum_{a \in \Sigma} \text{fr}_w(a) \cdot |\tau(a)|$$

Tenemos entonces que $|\hat{\tau}(w)| = \text{lp}_w(\tau) \cdot |w|$

Frecuencias relativas de los símbolos

Palabra a almacenar: $w \in \Sigma^*$

Para $a \in \Sigma$ definimos $\text{fr}_w(a)$ como la frecuencia relativa de a en w, vale decir, el número de apariciones de a en w dividido por el largo de w

- Por ejemplo, si $w = aabaacaab$, entonces $\text{fr}_w(a) = \frac{2}{3}$ y $\text{fr}_w(c) = \frac{1}{9}$

Para una Σ-codificación τ definimos el largo promedio para w como:

$$\text{lp}_w(\tau) \quad = \quad \sum_{a \in \Sigma} \text{fr}_w(a) \cdot |\tau(a)|$$

Tenemos entonces que $|\hat{\tau}(w)| = \text{lp}_w(\tau) \cdot |w|$

- Por lo tanto queremos una Σ-codificación τ que minimice $\text{lp}_w(\tau)$

Frecuencias relativas de los símbolos y la función objetivo

Problema de optimización a resolver

Dado $w \in \Sigma^*$, encontrar una Σ-codificación τ libre de prefijos que minimice el valor $\mathrm{lp}_w(\tau)$

Frecuencias relativas de los símbolos y la función objetivo

Problema de optimización a resolver

Dado $w \in \Sigma^*$, encontrar una Σ-codificación τ libre de prefijos que minimice el valor $\mathrm{lp}_w(\tau)$

La función objetivo del algoritmo codicioso es entonces $\mathrm{lp}_w(x)$

- Queremos minimizar el valor de esta función

Frecuencias relativas de los símbolos y la función objetivo

La función $\mathrm{lp}_w(x)$ se define a partir de la función $\mathrm{fr}_w(y)$

Frecuencias relativas de los símbolos y la función objetivo

La función $\mathrm{lp}_w(x)$ se define a partir de la función $\mathrm{fr}_w(y)$

- No se necesita más información sobre w. En particular, no se necesita saber cuál es el símbolo de w en una posición específica

Frecuencias relativas de los símbolos y la función objetivo

La función $\mathrm{lp}_w(x)$ se define a partir de la función $\mathrm{fr}_w(y)$

- No se necesita más información sobre w. En particular, no se necesita saber cuál es el símbolo de w en una posición específica

Podemos entonces trabajar con funciones de frecuencias relativas en lugar de palabras

- La entrada del problema no va a ser w sino que fr_w

Frecuencias relativas de los símbolos y la función objetivo

Decimos que $f : \Sigma \to (0, 1)$ es una función de frecuencias relativas para Σ si se cumple que $\sum_{a \in \Sigma} f(a) = 1$

Frecuencias relativas de los símbolos y la función objetivo

Decimos que $f : \Sigma \to (0, 1)$ es una función de frecuencias relativas para Σ si se cumple que $\sum_{a \in \Sigma} f(a) = 1$

Dada una función f de frecuencias relativas para Σ y una Σ-codificación τ, el largo promedio de τ para f se define como:

$$\mathrm{lp}_f(\tau) \quad = \quad \sum_{a \in \Sigma} f(a) \cdot |\tau(a)|$$

Frecuencias relativas de los símbolos y la función objetivo

Decimos que $f : \Sigma \to (0, 1)$ es una función de frecuencias relativas para Σ si se cumple que $\sum_{a \in \Sigma} f(a) = 1$

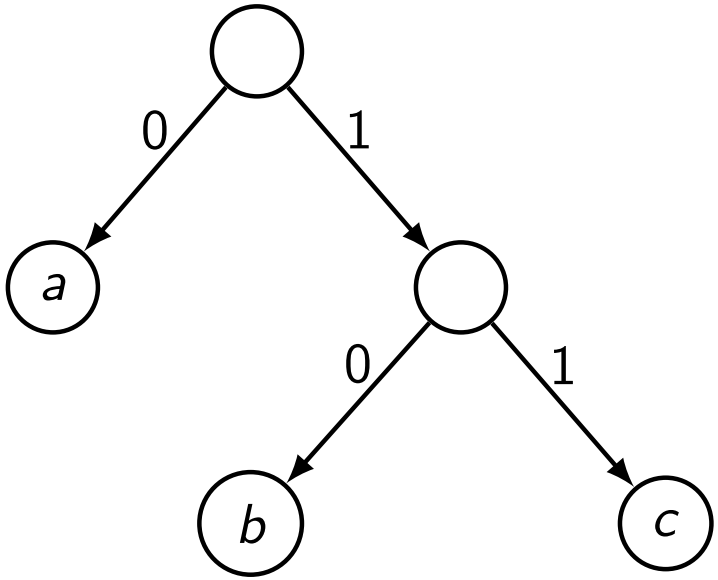
Dada una función f de frecuencias relativas para Σ y una Σ-codificación τ, el largo promedio de τ para f se define como:

$$\mathrm{lp}_f(\tau) \quad = \quad \sum_{a \in \Sigma} f(a) \cdot |\tau(a)|$$

La entrada del problema es entonces una función f de frecuencias relativas para Σ, y la función objetivo a minimizar es $\mathrm{lp}_f(x)$

Un ingrediente fundamental: codificaciones como árboles

Representamos la codificación $\tau(a) = 0$, $\tau(b) = 10$, $\tau(c) = 11$ como un árbol binario:



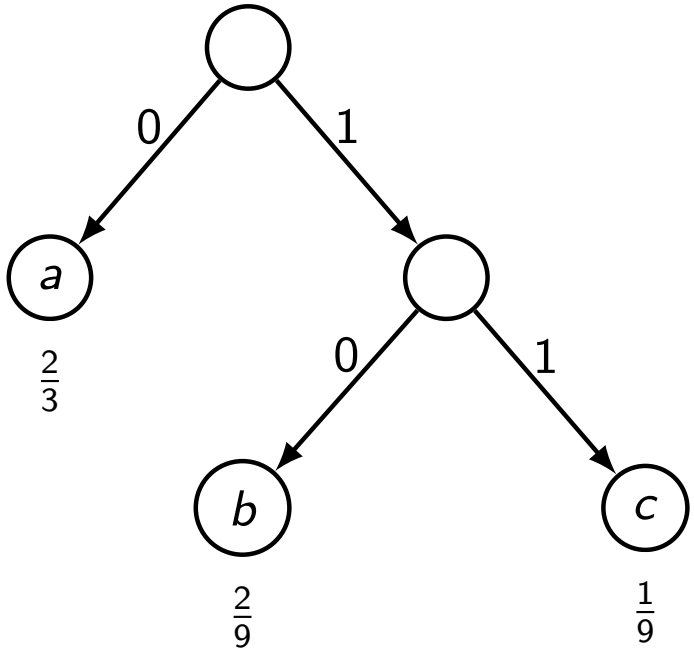
Un ingrediente fundamental: codificaciones como árboles

Si una Σ-codificación τ es libre de prefijos, entonces el árbol que la representa satisface las siguientes propiedades.

- Cada hoja tiene como etiqueta un elemento de Σ, y estos son los únicos nodos con etiquetas
- Cada símbolo de Σ es usado exactamente una vez como etiqueta
- Cada arco tiene etiqueta 0 ó 1
- Si una hoja tiene etiqueta e y las etiquetas de los arcos del camino desde la raíz hasta esta hoja forman una palabra $w \in \{0,1\}^*$, entonces $\tau(e) = w$

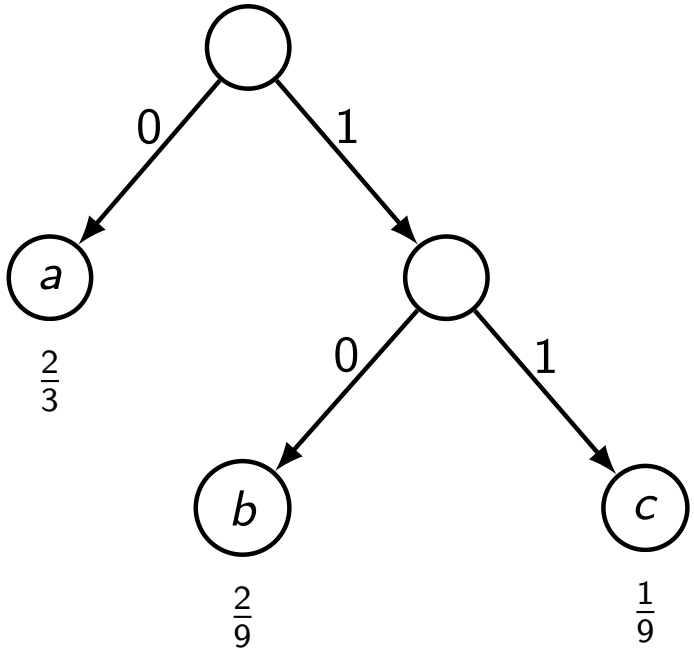
Codificaciones como árboles y las frecuencias relativas

Podemos agregar al árbol binario que representa una codificación τ la información sobre las frecuencias relativas dadas por una función f:



Codificaciones como árboles y las frecuencias relativas

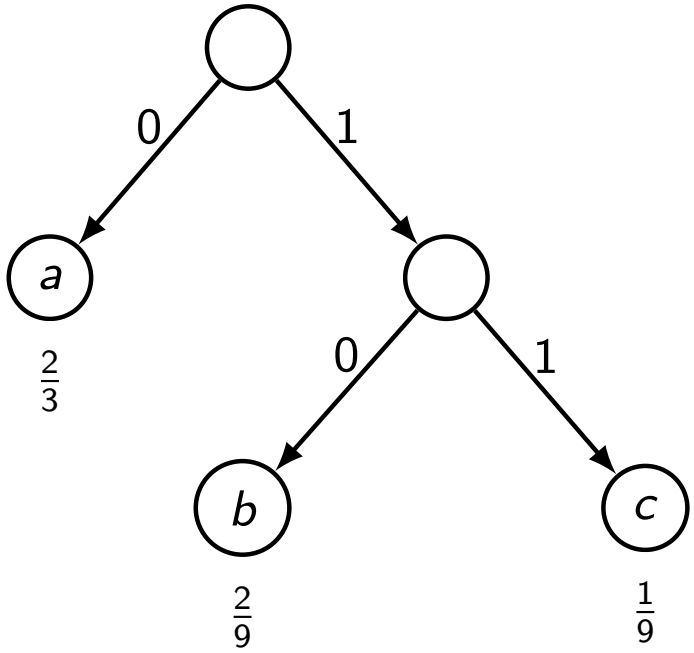
Podemos agregar al árbol binario que representa una codificación τ la información sobre las frecuencias relativas dadas por una función f:



Llamamos a este árbol abf(τ, f)

Codificaciones como árboles y las frecuencias relativas

Podemos agregar al árbol binario que representa una codificación τ la información sobre las frecuencias relativas dadas por una función f:



Llamamos a este árbol abf(τ, f)

- ¡Estos árboles son muy útiles!

Alguna propiedades importantes

Ejercicios

Sea f una función de frecuencias relativas para Σ y τ una Σ-codificación libre de prefijos que minimiza la función $\mathrm{lp}_f(x)$

1. Sean u y v dos hojas en $\mathrm{abf}(\tau, f)$ con etiquetas a y b, respectivamente. Demuestre que si el camino de la raíz a u es más corto que el camino de la raíz a v, entonces $f(a) \geq f(b)$

2. Demuestre que cada nodo interno en $\mathrm{abf}(\tau, f)$ tiene dos hijos

3. Sean $a, b \in \Sigma$ tales que $a \neq b$, $f(a) \leq f(b)$ y $f(b) \leq f(e)$ para todo $e \in (\Sigma \setminus \{a, b\})$. Demuestre que existe una Σ-codificación τ' libre de prefijos tal que $\mathrm{lp}_f(\tau') = \mathrm{lp}_f(\tau)$ y las hojas con etiquetas a y b en $\mathrm{abf}(\tau', f)$ son hermanas
   - Vale decir, existe $w \in \{0,1\}^*$ tal que $\tau'(a) = w0$ y $\tau'(b) = w1$

Calculando el mínimo de $\mathrm{lp}_f(x)$

Vamos a ver un algoritmo codicioso para calcular una Σ-codificación τ libre de prefijos que minimiza $\mathrm{lp}_f(x)$

- El algoritmo calcula la codificación de Huffman

Calculando el mínimo de $\mathrm{lp}_f(x)$

Vamos a ver un algoritmo codicioso para calcular una Σ-codificación τ libre de prefijos que minimiza $\mathrm{lp}_f(x)$

- El algoritmo calcula la codificación de Huffman

El algoritmo tiene los ingredientes mencionados de un algoritmo codicioso.

- Función objetivo a minimizar: $\mathrm{lp}_f(x)$
- Función de selección: elige los dos símbolos de Σ con menor frecuencia relativa, los coloca como hermanos en el árbol binario que representa la Σ-codificación óptima, y continua la construcción con el resto de los símbolos de Σ

Calculando el mínimo de $\mathrm{lp}_f(x)$

Vamos a ver un algoritmo codicioso para calcular una Σ-codificación τ libre de prefijos que minimiza $\mathrm{lp}_f(x)$

- El algoritmo calcula la codificación de Huffman

El algoritmo tiene los ingredientes mencionados de un algoritmo codicioso.

- Función objetivo a minimizar: $\mathrm{lp}_f(x)$
- Función de selección: elige los dos símbolos de Σ con menor frecuencia relativa, los coloca como hermanos en el árbol binario que representa la Σ-codificación óptima, y continua la construcción con el resto de los símbolos de Σ

Además, es necesario realizar una demostración para probar que el algoritmo es correcto.

El algoritmo de Huffman

En el siguiente algoritmo representamos a las funciones como conjuntos de pares ordenados, y suponemos que f es una función de frecuencias relativas que al menos tiene dos elementos en el dominio.

El algoritmo de Huffman

En el siguiente algoritmo representamos a las funciones como conjuntos de pares ordenados, y suponemos que f es una función de frecuencias relativas que al menos tiene dos elementos en el dominio.

CalcularCodificaciónHuffman(f)

Sea Σ el dominio de la función f

if $\Sigma = \{a, b\}$ **then return** $\{(a, 0), (b, 1)\}$

else

Sean $a, b \in \Sigma$ tales que $a \neq b$, $f(a) \leq f(b)$ y
$f(b) \leq f(e)$ para todo $e \in (\Sigma \setminus \{a, b\})$

Sea c un símbolo que **no** aparece en Σ

$g := (f \setminus \{(a, f(a)), (b, f(b))\}) \cup \{(c, f(a) + f(b))\}$

$\tau^\star :=$ **CalcularCodificaciónHuffman**(g)

$w := \tau^\star(c)$

$\tau := (\tau^\star \setminus \{(c, w)\}) \cup \{(a, w0), (b, w1)\}$

return τ

La correctitud de **CalcularCodificaciónHuffman**

Teorema

Si f es una función de frecuencias relativas, Σ es el dominio de f y $\tau =$ **CalcularCodificaciónHuffman**(f)*, entonces τ es una Σ-codificación libre de prefijos que minimiza la función $lp_f(x)$*

La correctitud de **CalcularCodificaciónHuffman**

Teorema

Si f es una función de frecuencias relativas, Σ es el dominio de f y $\tau =$ **CalcularCodificaciónHuffman**(f)*, entonces τ es una Σ-codificación libre de prefijos que minimiza la función $lp_f(x)$*

Demostración: Vamos a realizar la demostración por inducción en $|\Sigma|$

La correctitud de **CalcularCodificaciónHuffman**

Teorema

*Si f es una función de frecuencias relativas, Σ es el dominio de f y $\tau =$ **CalcularCodificaciónHuffman**(f), entonces τ es una Σ-codificación libre de prefijos que minimiza la función $lp_f(x)$*

Demostración: Vamos a realizar la demostración por inducción en $|\Sigma|$

Si $|\Sigma| = 2$, entonces la propiedad se cumple trivialmente

- ¿Por qué?

La correctitud de **CalcularCodificaciónHuffman**

Teorema

*Si f es una función de frecuencias relativas, Σ es el dominio de f y $\tau =$ **CalcularCodificaciónHuffman**(f), entonces τ es una Σ-codificación libre de prefijos que minimiza la función $lp_f(x)$*

Demostración: Vamos a realizar la demostración por inducción en $|\Sigma|$

Si $|\Sigma| = 2$, entonces la propiedad se cumple trivialmente

- ¿Por qué?

Suponga entonces que la propiedad se cumple para un valor $n \geq 2$, y suponga que $|\Sigma| = n + 1$

La demostración del teorema

Sean a, b, c, g, $\tau^\star$ y τ definidos como en el código de la llamada **CalcularCodificaciónHuffman**(f), y sea Γ el dominio de g

Como $|\Gamma| = n$ y $\tau^\star =$ **CalcularCodificaciónHuffman**(g), por hipótesis de inducción tenemos que $\tau^\star$ es una Γ-codificación libre de prefijos que minimiza la función $\mathrm{lp}_g(x)$

La demostración del teorema

Sean a, b, c, g, $\tau^\star$ y τ definidos como en el código de la llamada **CalcularCodificaciónHuffman**(f), y sea Γ el dominio de g

Como $|\Gamma| = n$ y $\tau^\star =$ **CalcularCodificaciónHuffman**(g), por hipótesis de inducción tenemos que $\tau^\star$ es una Γ-codificación libre de prefijos que minimiza la función $\mathrm{lp}_g(x)$

Dada la definición de τ es simple verificar las siguientes propiedades:

- τ es una codificación libre de prefijos
- Para cada $e \in (\Sigma \setminus \{a, b\})$ se tiene que $\tau(e) = \tau^\star(e)$
- $|\tau(a)| = |\tau(b)| = |\tau^\star(c)| + 1$

La demostración del teorema

Por contradicción suponga que τ no minimiza el valor de la función $\mathrm{lp}_f(x)$

- Vale decir, existe una Σ-codificación τ' libre de prefijos tal que τ' minimiza la función $\mathrm{lp}_f(x)$ y $\mathrm{lp}_f(\tau') < \mathrm{lp}_f(\tau)$

La demostración del teorema

Por contradicción suponga que τ no minimiza el valor de la función $\mathrm{lp}_f(x)$

- Vale decir, existe una Σ-codificación τ' libre de prefijos tal que τ' minimiza la función $\mathrm{lp}_f(x)$ y $\mathrm{lp}_f(\tau') < \mathrm{lp}_f(\tau)$

Por la definición de a, b y los ejercicios anteriores podemos suponer que existe $w \in \{0,1\}^*$ tal que $\tau'(a) = w0$ y $\tau'(b) = w1$

- Nótese que $w \neq \varepsilon$ puesto que $|\Sigma| \geq 3$

La demostración del teorema

Por contradicción suponga que τ no minimiza el valor de la función $\mathrm{lp}_f(x)$

- Vale decir, existe una Σ-codificación τ' libre de prefijos tal que τ' minimiza la función $\mathrm{lp}_f(x)$ y $\mathrm{lp}_f(\tau') < \mathrm{lp}_f(\tau)$

Por la definición de a, b y los ejercicios anteriores podemos suponer que existe $w \in \{0,1\}^*$ tal que $\tau'(a) = w0$ y $\tau'(b) = w1$

- Nótese que $w \neq \varepsilon$ puesto que $|\Sigma| \geq 3$

A partir de τ' defina la siguiente Γ-codificación τ'':

$$\tau'' \quad = \quad (\tau' \setminus \{(a, \tau'(a)), (b, \tau'(b))\}) \cup \{(c, w)\}$$

La demostración del teorema

Por contradicción suponga que τ no minimiza el valor de la función $\mathrm{lp}_f(x)$

- Vale decir, existe una Σ-codificación τ' libre de prefijos tal que τ' minimiza la función $\mathrm{lp}_f(x)$ y $\mathrm{lp}_f(\tau') < \mathrm{lp}_f(\tau)$

Por la definición de a, b y los ejercicios anteriores podemos suponer que existe $w \in \{0,1\}^*$ tal que $\tau'(a) = w0$ y $\tau'(b) = w1$

- Nótese que $w \neq \varepsilon$ puesto que $|\Sigma| \geq 3$

A partir de τ' defina la siguiente Γ-codificación τ'':

$$\tau'' \quad = \quad (\tau' \setminus \{(a, \tau'(a)), (b, \tau'(b))\}) \cup \{(c, w)\}$$

Tenemos que τ'' es una Γ-codificación libre de prefijos

- ¿Por qué?

La relación entre $\mathrm{lp}_g(\tau^\star)$ y $\mathrm{lp}_f(\tau)$

$$
\begin{aligned}
\mathrm{lp}_g(\tau^\star) &= \sum_{e \in \Gamma} g(e) \cdot |\tau^\star(e)| \\
&= \left(\sum_{e \in (\Gamma \setminus \{c\})} g(e) \cdot |\tau^\star(e)| \right) + g(c) \cdot |\tau^\star(c)| \\
&= \left(\sum_{e \in (\Sigma \setminus \{a,b\})} f(e) \cdot |\tau(e)| \right) + (f(a) + f(b)) \cdot |\tau^\star(c)| \\
&= \left(\sum_{e \in (\Sigma \setminus \{a,b\})} f(e) \cdot |\tau(e)| \right) + f(a) \cdot |\tau^\star(c)| + f(b) \cdot |\tau^\star(c)| \\
&= \left(\sum_{e \in (\Sigma \setminus \{a,b\})} f(e) \cdot |\tau(e)| \right) + f(a) \cdot (|\tau(a)| - 1) + f(b) \cdot (|\tau(b)| - 1) \\
&= \left(\sum_{e \in \Sigma} f(e) \cdot |\tau(e)| \right) - (f(a) + f(b)) \\
&= \mathrm{lp}_f(\tau) - (f(a) + f(b))
\end{aligned}
$$

La relación entre $\mathrm{lp}_g(\tau'')$ y $\mathrm{lp}_f(\tau')$

$$
\begin{aligned}
\mathrm{lp}_g(\tau'') &= \sum_{e \in \Gamma} g(e) \cdot |\tau''(e)| \\
&= \left(\sum_{e \in (\Gamma \setminus \{c\})} g(e) \cdot |\tau''(e)| \right) + g(c) \cdot |\tau''(c)| \\
&= \left(\sum_{e \in (\Sigma \setminus \{a,b\})} f(e) \cdot |\tau'(e)| \right) + (f(a) + f(b)) \cdot |\tau''(c)| \\
&= \left(\sum_{e \in (\Sigma \setminus \{a,b\})} f(e) \cdot |\tau'(e)| \right) + f(a) \cdot |\tau''(c)| + f(b) \cdot |\tau''(c)| \\
&= \left(\sum_{e \in (\Sigma \setminus \{a,b\})} f(e) \cdot |\tau'(e)| \right) + f(a) \cdot (|\tau'(a)| - 1) + f(b) \cdot (|\tau'(b)| - 1) \\
&= \left(\sum_{e \in \Sigma} f(e) \cdot |\tau'(e)| \right) - (f(a) + f(b)) \\
&= \mathrm{lp}_f(\tau') - (f(a) + f(b))
\end{aligned}
$$

Obteniendo una contradicción

Tenemos entonces que

$$\begin{aligned} \mathrm{lp}_g(\tau'') &= \mathrm{lp}_f(\tau') - (f(a) + f(b)) \\ &< \mathrm{lp}_f(\tau) - (f(a) + f(b)) \\ &= \mathrm{lp}_g(\tau^\star) \end{aligned}$$

Obteniendo una contradicción

Tenemos entonces que

$$\begin{aligned} \mathrm{lp}_g(\tau'') &= \mathrm{lp}_f(\tau') - (f(a) + f(b)) \\ &< \mathrm{lp}_f(\tau) - (f(a) + f(b)) \\ &= \mathrm{lp}_g(\tau^\star) \end{aligned}$$

Concluimos entonces que $\tau^\star$ no minimiza la función $\mathrm{lp}_g(x)$, lo cual contradice la hipótesis de inducción. □

Dos comentarios finales

La siguiente función calcula la codificación de Huffman teniendo como entrada una palabra w:

CalcularCodificaciónHuffman(w)
    if $w = \varepsilon$ **then return** $\emptyset$
    else
        $\Sigma :=$ conjunto de símbolos mencionados en w
        if $\Sigma = \{a\}$ **then return** $\{(a, 0)\}$
        else return CalcularCodificaciónHuffman(fr_w)

Dos comentarios finales

La siguiente función calcula la codificación de Huffman teniendo como entrada una palabra w:

CalcularCodificaciónHuffman(w)
    if $w = \varepsilon$ **then return** $\emptyset$
    else
        $\Sigma :=$ conjunto de símbolos mencionados en w
        if $\Sigma = \{a\}$ **then return** $\{(a, 0)\}$
        else return CalcularCodificaciónHuffman(fr_w)

¿Cuál es la complejidad de **CalcularCodificaciónHuffman**(f)?

- ¿Qué estructura de datos debería ser utilizada para almacenar f?