Algoritmos glotones continuación...

mat-151

Algoritmos glotones: **códigos de Huffman**

Algoritmos glotones: **códigos de Huffman**

- Son técnicas muy efectivas para comprimir datos.

Algoritmos glotones: **códigos de Huffman**

- Son técnicas muy efectivas para comprimir datos.
- Alcanzan una compresión de entre 20% y 90% típicamente dependiendo de las características de los datos de entrada.

Algoritmos glotones: **códigos de Huffman**

- Son técnicas muy efectivas para comprimir datos.
- Alcanzan una compresión de entre 20% y 90% típicamente dependiendo de las características de los datos de entrada.
- La entrada es una secuencia de caracteres.

Algoritmos glotones: **códigos de Huffman**

- Son técnicas muy efectivas para comprimir datos.
- Alcanzan una compresión de entre 20% y 90% típicamente dependiendo de las características de los datos de entrada.
- La entrada es una secuencia de caracteres.
- Se usa una tabla de frecuencias de ocurrencia de los caracteres para construir la manera óptima de representación de cada carácter con una cadena binaria.

Algoritmos glotones: **códigos de Huffman**

- Son técnicas muy efectivas para comprimir datos.
- Alcanzan una compresión de entre 20% y 90% típicamente dependiendo de las características de los datos de entrada.
- La entrada es una secuencia de caracteres.
- Se usa una tabla de frecuencias de ocurrencia de los caracteres para construir la manera óptima de representación de cada carácter con una cadena binaria.
- Archivo de 100,000 caracteres a almacenar de manera compacta. Se observa que los caracteres en el archivo ocurren con las frecuencias:

Algoritmos glotones: **códigos de Huffman**

- Son técnicas muy efectivas para comprimir datos.
- Alcanzan una compresión de entre 20% y 90% típicamente dependiendo de las características de los datos de entrada.
- La entrada es una secuencia de caracteres.
- Se usa una tabla de frecuencias de ocurrencia de los caracteres para construir la manera óptima de representación de cada carácter con una cadena binaria.
- Archivo de 100,000 caracteres a almacenar de manera compacta. Se observa que los caracteres en el archivo ocurren con las frecuencias:

	a	b	c	d	e	f
Frecuencia (en miles)	45	13	12	16	9	5

Algoritmos glotones: **códigos de Huffman**

- Son técnicas muy efectivas para comprimir datos.
- Alcanzan una compresión de entre 20% y 90% típicamente dependiendo de las características de los datos de entrada.
- La entrada es una secuencia de caracteres.
- Se usa una tabla de frecuencias de ocurrencia de los caracteres para construir la manera óptima de representación de cada carácter con una cadena binaria.
- Archivo de 100,000 caracteres a almacenar de manera compacta. Se observa que los caracteres en el archivo ocurren con las frecuencias:

	a	b	c	d	e	f
Frecuencia (en miles)	45	13	12	16	9	5

- Aparecen 6 caracteres, el caracter “a” ocurre 45,000 veces.

Algoritmos glotones: **códigos de Huffman**

Algoritmos glotones: **códigos de Huffman**

- Se quiere diseñar un código de caracteres binarios (o código) donde cada caracter sea representado por una cadena binaria única.

Algoritmos glotones: **códigos de Huffman**

- Se quiere diseñar un código de caracteres binarios (o código) donde cada caracter sea representado por una cadena binaria única.
- IDEA

Algoritmos glotones: **códigos de Huffman**

- Se quiere diseñar un código de caracteres binarios (o código) donde cada caracter sea representado por una cadena binaria única.
- IDEA
  - podemos usar un código de tamaño fijo donde necesitaremos 3 bits para representar los 6 caracteres: a=000, b=001, ..., f=101.

Algoritmos glotones: **códigos de Huffman**

- Se quiere diseñar un código de caracteres binarios (o código) donde cada caracter sea representado por una cadena binaria única.
- IDEA
  - podemos usar un código de tamaño fijo donde necesitaremos 3 bits para representar los 6 caracteres: a=000, b=001, ..., f=101.
  - este método requiere 300,000 bits para codificar enteramente el archivo.

Algoritmos glotones: **códigos de Huffman**

- Se quiere diseñar un código de caracteres binarios (o código) donde cada caracter sea representado por una cadena binaria única.
- IDEA
  - podemos usar un código de tamaño fijo donde necesitaremos 3 bits para representar los 6 caracteres: a=000, b=001, ..., f=101.
  - este método requiere 300,000 bits para codificar enteramente el archivo.
  - ¿se puede hacer mejor?

Algoritmos glotones: **códigos de Huffman**

- Se quiere diseñar un código de caracteres binarios (o código) donde cada caracter sea representado por una cadena binaria única.
- IDEA
  - podemos usar un código de tamaño fijo donde necesitaremos 3 bits para representar los 6 caracteres: a=000, b=001, ..., f=101.
  - este método requiere 300,000 bits para codificar enteramente el archivo.
  - ¿se puede hacer mejor?

	a	b	c	d	e	f
Frecuencia (en miles)	45	13	12	16	9	5
Código de tamaño constante	0	1	10	11	100	101

Algoritmos glotones: **códigos de Huffman**

Algoritmos glotones: **códigos de Huffman**

- Usando un código de tamaño variable, dando a los caracteres con mayor frecuencia palabras clave cortas y a los menos frecuentes palabras clave más largas.

Algoritmos glotones: **códigos de Huffman**

- Usando un código de tamaño variable, dando a los caracteres con mayor frecuencia palabras clave cortas y a los menos frecuentes palabras clave más largas.
- Por ejemplo:

Algoritmos glotones: **códigos de Huffman**

- Usando un código de tamaño variable, dando a los caracteres con mayor frecuencia palabras clave cortas y a los menos frecuentes palabras clave más largas.
- Por ejemplo:

	a	b	c	d	e	f
Frecuencia (en miles)	45	13	12	16	9	5
Código de tamaño constante	0	1	10	11	100	101
Código de tamaño variable	0	101	100	111	1101	1100

Algoritmos glotones: **códigos de Huffman**

- Usando un código de tamaño variable, dando a los caracteres con mayor frecuencia palabras clave cortas y a los menos frecuentes palabras clave más largas.

- Por ejemplo:

	a	b	c	d	e	f
Frecuencia (en miles)	45	13	12	16	9	5
Código de tamaño constante	0	1	10	11	100	101
Código de tamaño variable	0	101	100	111	1101	1100

- Este código requiere:

Algoritmos glotones: **códigos de Huffman**

- Usando un código de tamaño variable, dando a los caracteres con mayor frecuencia palabras clave cortas y a los menos frecuentes palabras clave más largas.

- Por ejemplo:

	a	b	c	d	e	f
Frecuencia (en miles)	45	13	12	16	9	5
Código de tamaño constante	0	1	10	11	100	101
Código de tamaño variable	0	101	100	111	1101	1100

- Este código requiere:

$$(45 \cdot 1 + 13 \cdot 3 + 12 \cdot 3 + 16 \cdot 3 + 9 \cdot 4 + 5 \cdot 4) \cdot 1000 = 224{,}000 \text{ bits}$$

Algoritmos glotones: **códigos de Huffman**

- Usando un código de tamaño variable, dando a los caracteres con mayor frecuencia palabras clave cortas y a los menos frecuentes palabras clave más largas.

- Por ejemplo:

	a	b	c	d	e	f
Frecuencia (en miles)	45	13	12	16	9	5
Código de tamaño constante	0	1	10	11	100	101
Código de tamaño variable	0	101	100	111	1101	1100

- Este código requiere:

$$(45 \cdot 1 + 13 \cdot 3 + 12 \cdot 3 + 16 \cdot 3 + 9 \cdot 4 + 5 \cdot 4) \cdot 1000 = 224{,}000 \text{ bits}$$

para representar un archivo, ahorrando cerca del 25%.

Algoritmos glotones: **códigos de Huffman**

Algoritmos glotones: **códigos de Huffman**

- Se consideran solo códigos en donde ninguna palabra sea también un prefijo de otra palabra. Este tipo de códigos son llamados códigos-prefijo.

Algoritmos glotones: **códigos de Huffman**

- Se consideran solo códigos en donde ninguna palabra sea también un prefijo de otra palabra. Este tipo de códigos son llamados códigos-prefijo.
- La codificación es un proceso simple utilizando códigos binarios: solo hay que concatenar las palabras que representan cada caracter.

Algoritmos glotones: **códigos de Huffman**

- Se consideran solo códigos en donde ninguna palabra sea también un prefijo de otra palabra. Este tipo de códigos son llamados códigos-prefijo.
- La codificación es un proceso simple utilizando códigos binarios: solo hay que concatenar las palabras que representan cada caracter.
- Por ejemplo, usando el código de la tabla, la codificación de la cadena abc es $0 \cdot 101 \cdot 100 = 0101100$ donde "$\cdot$" denota concatenación.

Algoritmos glotones: **códigos de Huffman**

- Se consideran solo códigos en donde ninguna palabra sea también un prefijo de otra palabra. Este tipo de códigos son llamados códigos-prefijo.
- La codificación es un proceso simple utilizando códigos binarios: solo hay que concatenar las palabras que representan cada caracter.
- Por ejemplo, usando el código de la tabla, la codificación de la cadena abc es $0 \cdot 101 \cdot 100 = 0101100$ donde "$\cdot$" denota concatenación.
- Los códigos-prefijo son deseables ya que eliminan ambigüedades sobre dónde empieza una nueva cadena.

Algoritmos glotones: **códigos de Huffman**

- Se consideran solo códigos en donde ninguna palabra sea también un prefijo de otra palabra. Este tipo de códigos son llamados códigos-prefijo.
- La codificación es un proceso simple utilizando códigos binarios: solo hay que concatenar las palabras que representan cada caracter.
- Por ejemplo, usando el código de la tabla, la codificación de la cadena abc es $0 \cdot 101 \cdot 100 = 0101100$ donde "$\cdot$" denota concatenación.
- Los códigos-prefijo son deseables ya que eliminan ambigüedades sobre dónde empieza una nueva cadena.
- Así, la cadena 0010111101 se puede leer de manera única como $0 \cdot 0 \cdot 101 \cdot 1101$ que se decodifica como aabe.

Algoritmos glotones: **códigos de Huffman**

- Se consideran solo códigos en donde ninguna palabra sea también un prefijo de otra palabra. Este tipo de códigos son llamados códigos-prefijo.
- La codificación es un proceso simple utilizando códigos binarios: solo hay que concatenar las palabras que representan cada caracter.
- Por ejemplo, usando el código de la tabla, la codificación de la cadena abc es $0 \cdot 101 \cdot 100 = 0101100$ donde "$\cdot$" denota concatenación.
- Los códigos-prefijo son deseables ya que eliminan ambigüedades sobre dónde empieza una nueva cadena.
- Así, la cadena 0010111101 se puede leer de manera única como $0 \cdot 0 \cdot 101 \cdot 1101$ que se decodifica como aabe.
- Se necesita una representación conveniente para la decodificación de tal manera que se identifique rápidamente la palabra código inicial.

Algoritmos glotones: **códigos de Huffman**

Algoritmos glotones: **códigos de Huffman**

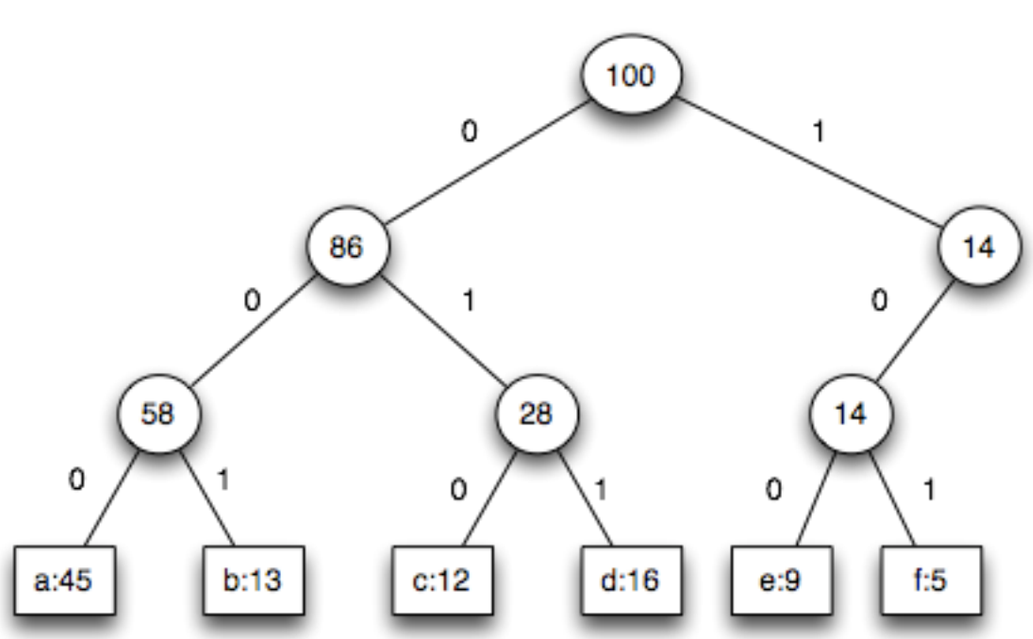
- Representación: árbol binario donde las hojas son los caracteres.

Algoritmos glotones: **códigos de Huffman**

- Representación: árbol binario donde las hojas son los caracteres.
- Se interpreta el código binario para un caracter como el camino desde la raíz hasta el caracter, donde 0 significa “ir al hijo izquierdo” y 1 significa “ir al hijo derecho”.

Algoritmos glotones: **códigos de Huffman**

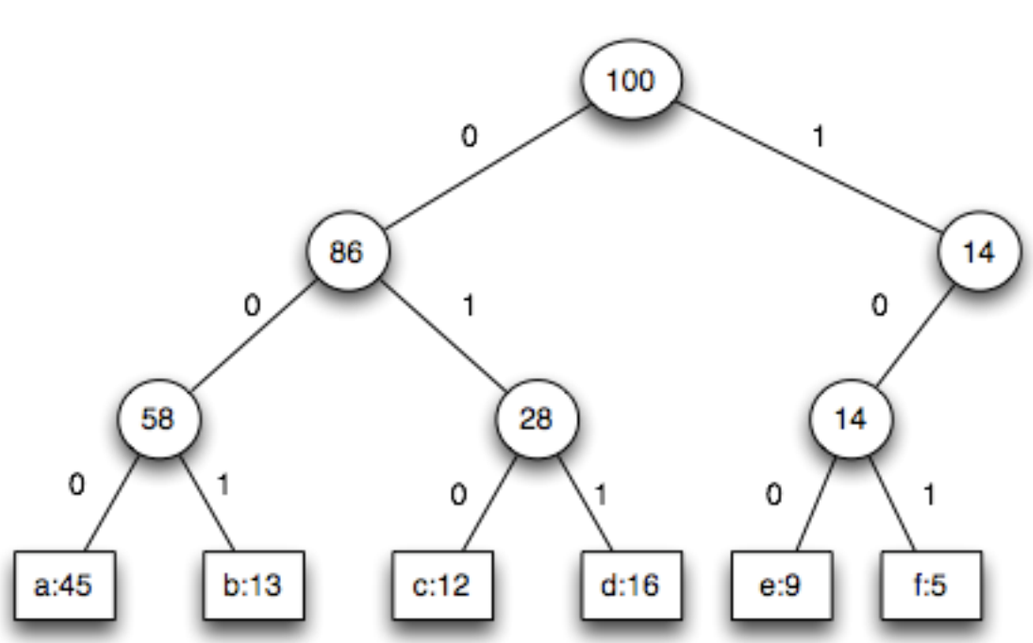
- Representación: árbol binario donde las hojas son los caracteres.
- Se interpreta el código binario para un caracter como el camino desde la raíz hasta el caracter, donde 0 significa “ir al hijo izquierdo” y 1 significa “ir al hijo derecho”.



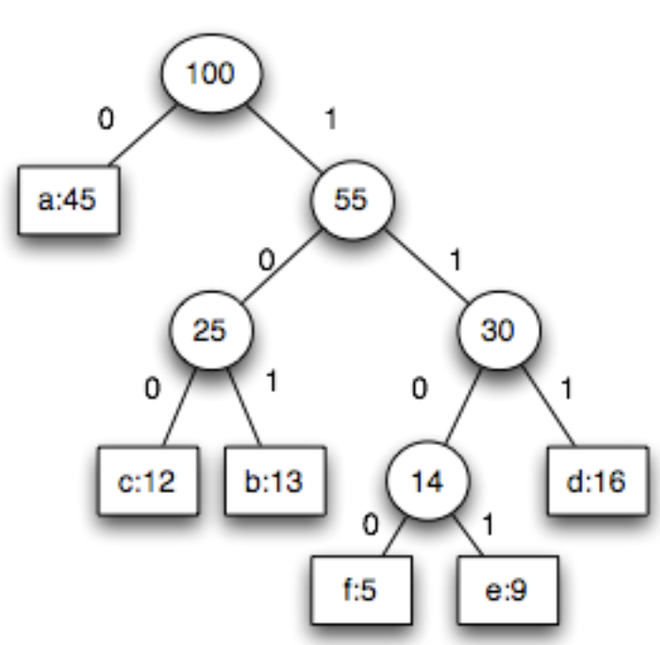
código de tamaño fijo

Algoritmos glotones: **códigos de Huffman**

- Representación: árbol binario donde las hojas son los caracteres.
- Se interpreta el código binario para un caracter como el camino desde la raíz hasta el caracter, donde 0 significa “ir al hijo izquierdo” y 1 significa “ir al hijo derecho”.



código de tamaño fijo



código de tamaño variable

Algoritmos glotones: **códigos de Huffman**

Algoritmos glotones: **códigos de Huffman**

- El código óptimo para un archivo siempre será un árbol binario lleno.

Algoritmos glotones: **códigos de Huffman**

- El código óptimo para un archivo siempre será un árbol binario lleno.
- C es el alfabeto de donde se toman los caracteres.

Algoritmos glotones: **códigos de Huffman**

- El código óptimo para un archivo siempre será un árbol binario lleno.
- C es el alfabeto de donde se toman los caracteres.
- El árbol de código-prefijo óptimo tiene exactamente hojas.

Algoritmos glotones: **códigos de Huffman**

- El código óptimo para un archivo siempre será un árbol binario lleno.
- C es el alfabeto de donde se toman los caracteres.
- El árbol de código-prefijo óptimo tiene exactamente hojas.
- El árbol de código-prefijo óptimo tiene exactamente nodos internos.

Algoritmos glotones: **códigos de Huffman**

- El código óptimo para un archivo siempre será un árbol binario lleno.
- C es el alfabeto de donde se toman los caracteres.
- El árbol de código-prefijo óptimo tiene exactamente hojas.
- El árbol de código-prefijo óptimo tiene exactamente nodos internos.
- El número de bits necesarios para codificar el archivo (el costo del árbol T) se calcula:

Algoritmos glotones: **códigos de Huffman**

- El código óptimo para un archivo siempre será un árbol binario lleno.
- C es el alfabeto de donde se toman los caracteres.
- El árbol de código-prefijo óptimo tiene exactamente $|C|$ hojas.
- El árbol de código-prefijo óptimo tiene exactamente nodos internos.
- El número de bits necesarios para codificar el archivo (el costo del árbol T) se calcula:

Algoritmos glotones: **códigos de Huffman**

- El código óptimo para un archivo siempre será un árbol binario lleno.
- C es el alfabeto de donde se toman los caracteres.
- El árbol de código-prefijo óptimo tiene exactamente $|C|$ hojas.
- El árbol de código-prefijo óptimo tiene exactamente $|C| - 1$ nodos internos.
- El número de bits necesarios para codificar el archivo (el costo del árbol T) se calcula:

Algoritmos glotones: **códigos de Huffman**

- El código óptimo para un archivo siempre será un árbol binario lleno.
- C es el alfabeto de donde se toman los caracteres.
- El árbol de código-prefijo óptimo tiene exactamente $|C|$ hojas.
- El árbol de código-prefijo óptimo tiene exactamente $|C| - 1$ nodos internos.
- El número de bits necesarios para codificar el archivo (el costo del árbol T) se calcula:

$$B(T) = \sum_{c \in C} f(c) d_T(c)$$

Algoritmos glotones: **códigos de Huffman**

- El código óptimo para un archivo siempre será un árbol binario lleno.
- C es el alfabeto de donde se toman los caracteres.
- El árbol de código-prefijo óptimo tiene exactamente $|C|$ hojas.
- El árbol de código-prefijo óptimo tiene exactamente $|C| - 1$ nodos internos.
- El número de bits necesarios para codificar el archivo (el costo del árbol T) se calcula:

$$B(T) = \sum_{c \in C} f(c) d_T(c)$$

donde $f(c)$ es la frecuencia de c en el archivo y $d_T(c)$ es la profundidad de la hoja c en el árbol (y la longitud de la palabra código para c).

Algoritmos glotones: **códigos de Huffman**

Algoritmos glotones: **códigos de Huffman**

- Huffman inventó un algoritmo glotón que construye el código-prefijo óptimo llamado código Huffman.

Algoritmos glotones: **códigos de Huffman**

- Huffman inventó un algoritmo glotón que construye el código-prefijo óptimo llamado código Huffman.

```
HUFFMAN(C)

1. n←|C|
2. Q←C
3. for i ← 1 to n-1
4.       do crear un nuevo nodo z
5.            left[z] ← x ← EXTRACT-MIN(Q)
6.            right[z] ← y ← EXTRACT-MIN(Q)
7.            f[z] ← f[x] + f[y]
8.            INSERT(Q,z)
9. return EXTRACT-MIN          // regresa la raíz del árbol
```

Algoritmos glotones: **códigos de Huffman**

- Huffman inventó un algoritmo glotón que construye el código-prefijo óptimo llamado código Huffman.

```
HUFFMAN(C)

1. n←|C|
2. Q←C
3. for i ← 1 to n-1
4.       do crear un nuevo nodo z
5.            left[z] ← x ← EXTRACT-MIN(Q)
6.            right[z] ← y ← EXTRACT-MIN(Q)
7.            f[z] ← f[x] + f[y]
8.            INSERT(Q,z)
9. return EXTRACT-MIN          // regresa la raíz del árbol
```

C : conjunto de n caracteres.

c en C : un objeto caracter.

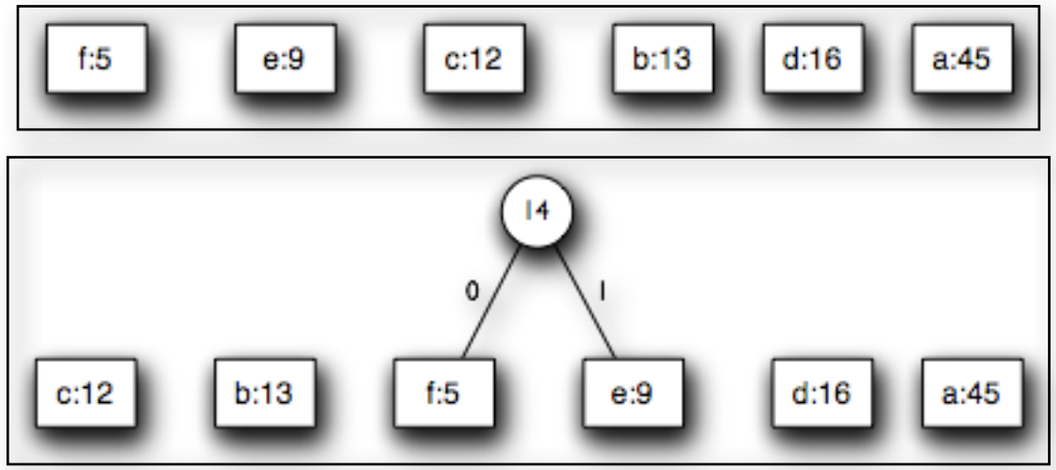
f(c) : frecuencia de ocurrencia.

T : árbol binario del código.

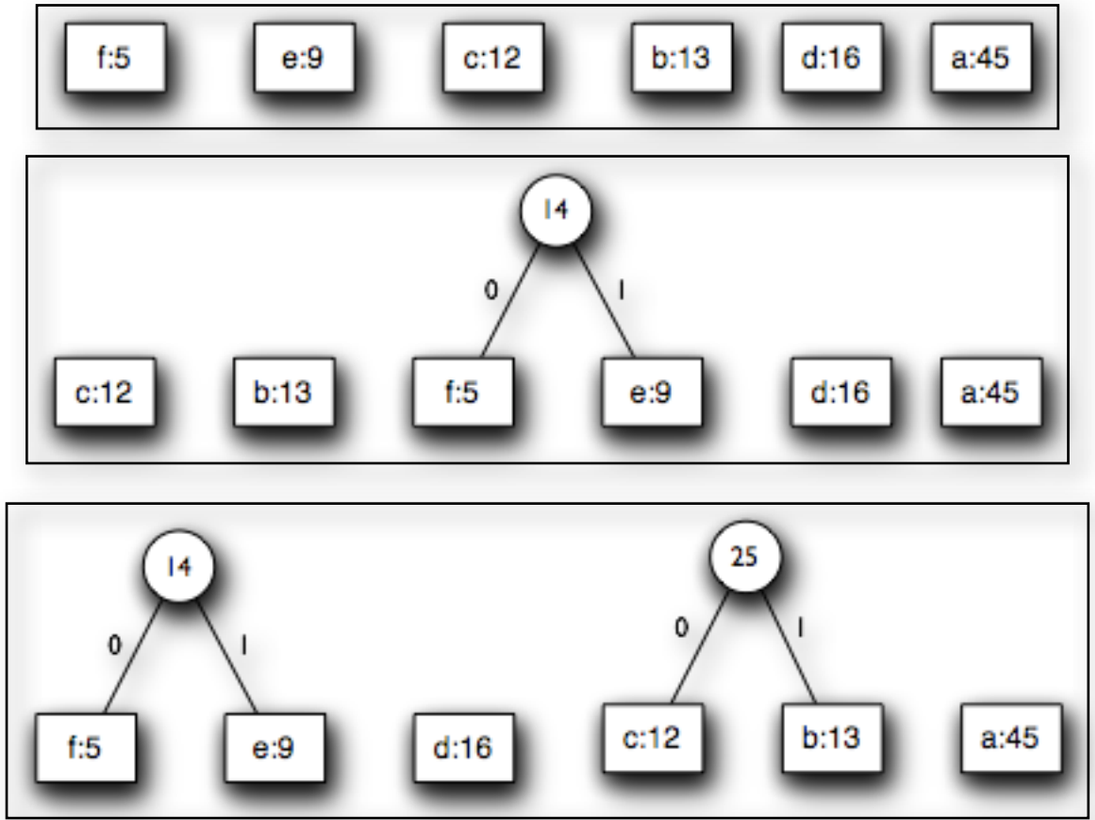
Q : cola de prioridad **min** con llave **f**.

Algoritmos glotones: **códigos de Huffman**

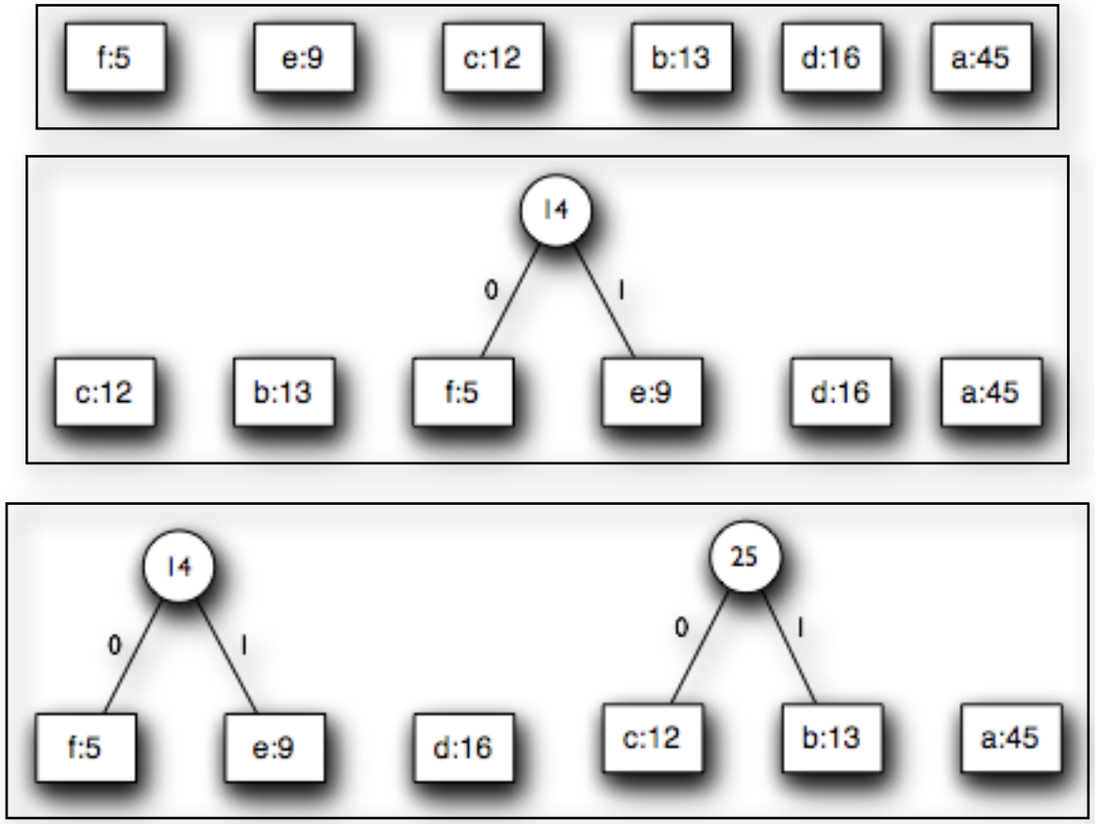
Algoritmos glotones: **códigos de Huffman**



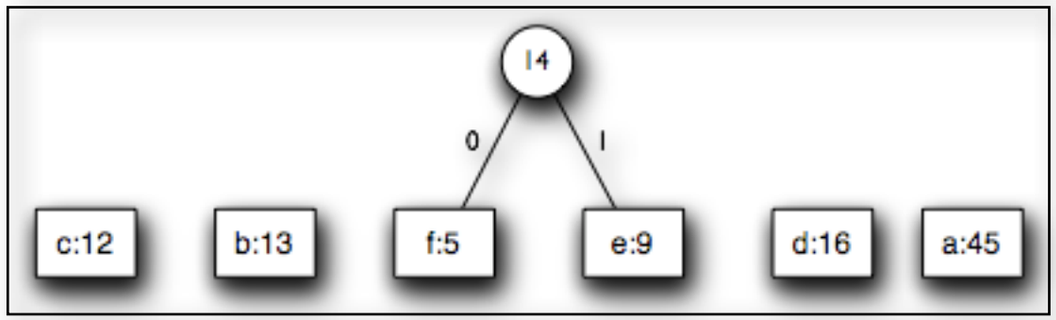
Algoritmos glotones: **códigos de Huffman**

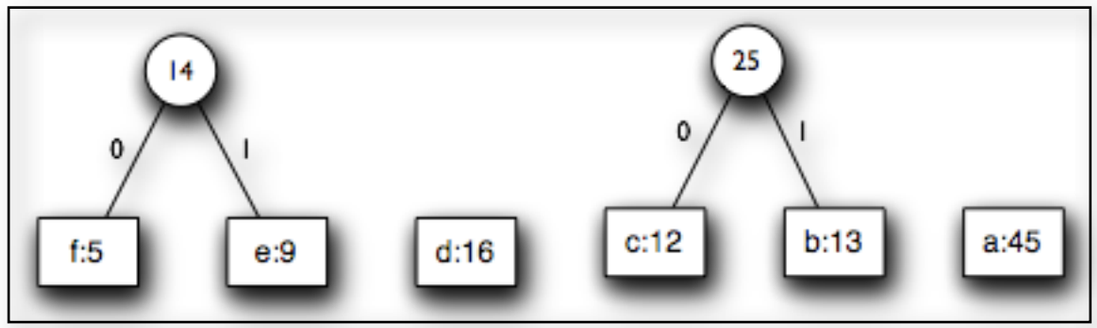


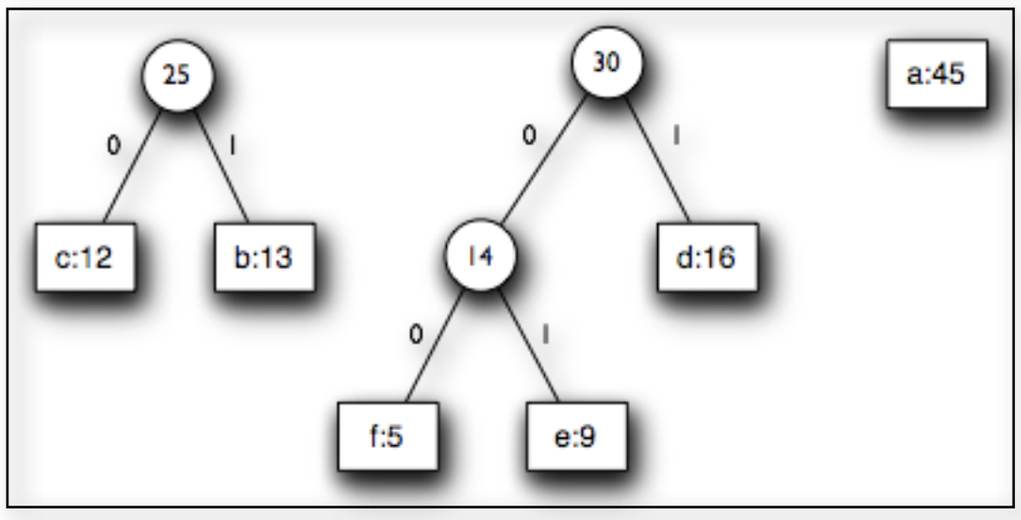
Algoritmos glotones: **códigos de Huffman**

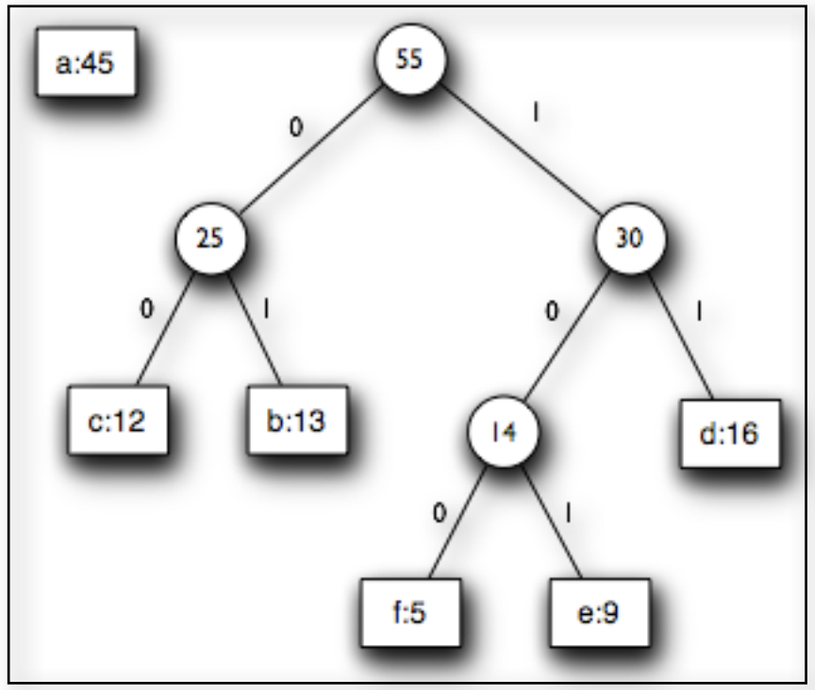


Algoritmos glotones: **códigos de Huffman**

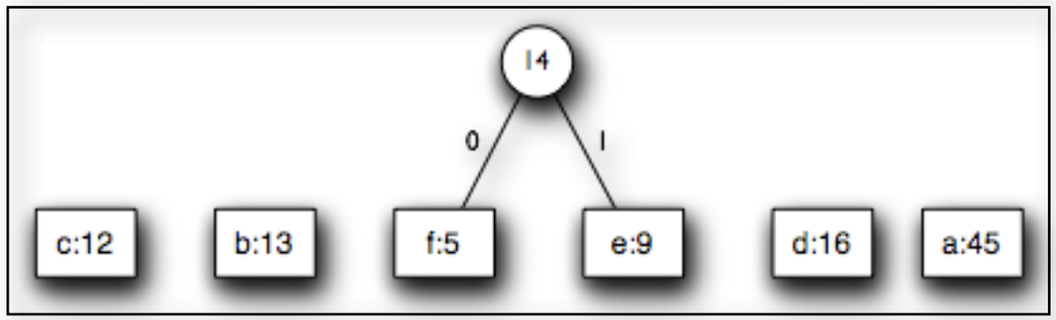


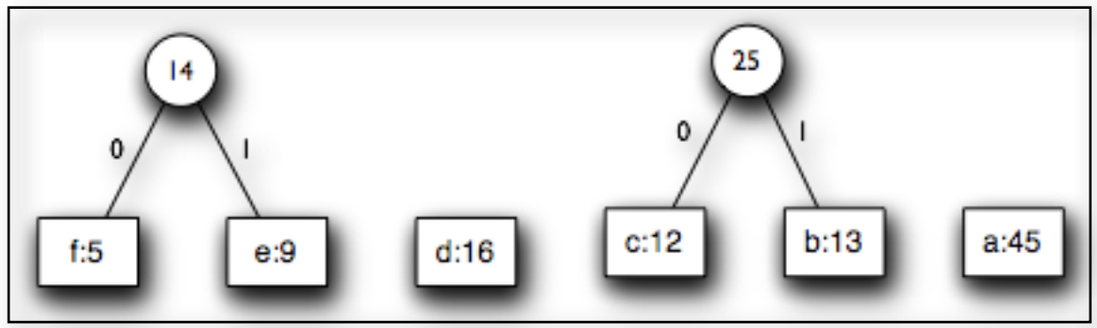


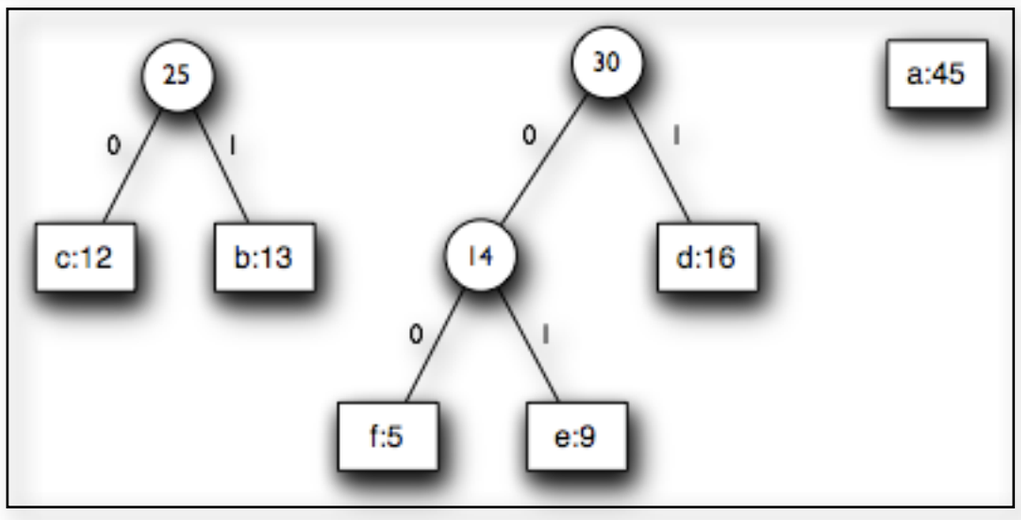


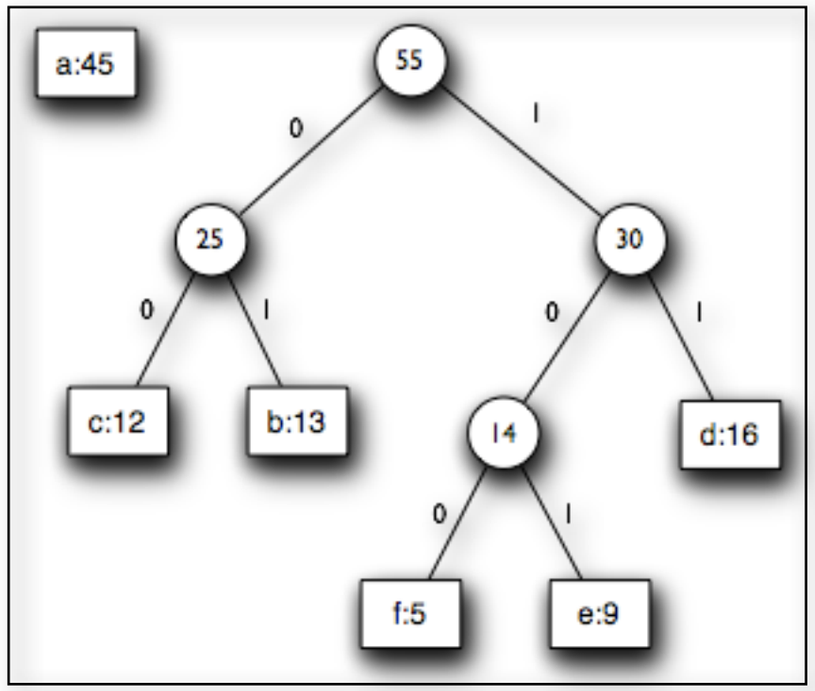


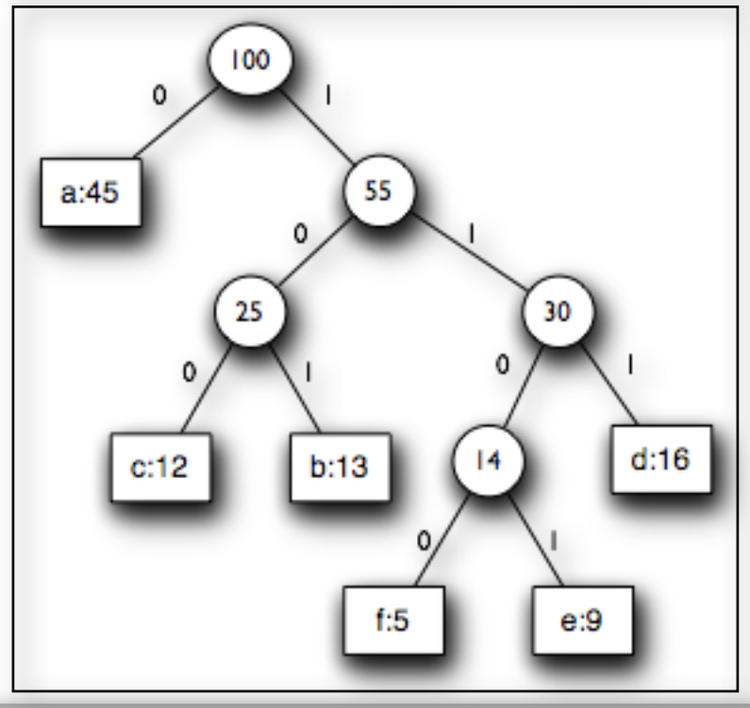
Algoritmos glotones: **códigos de Huffman**



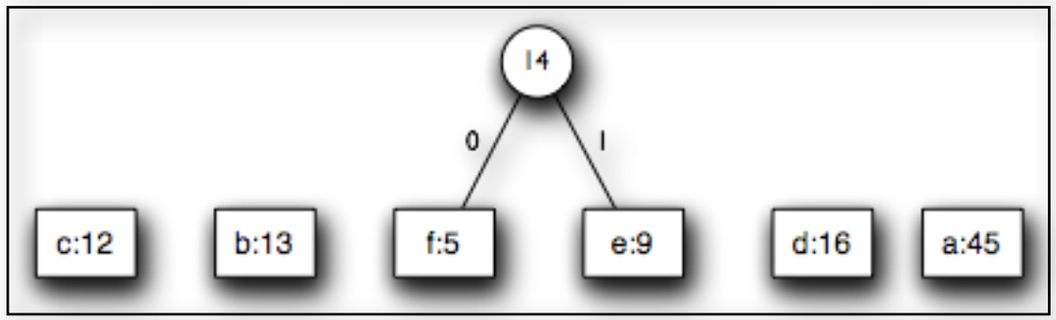


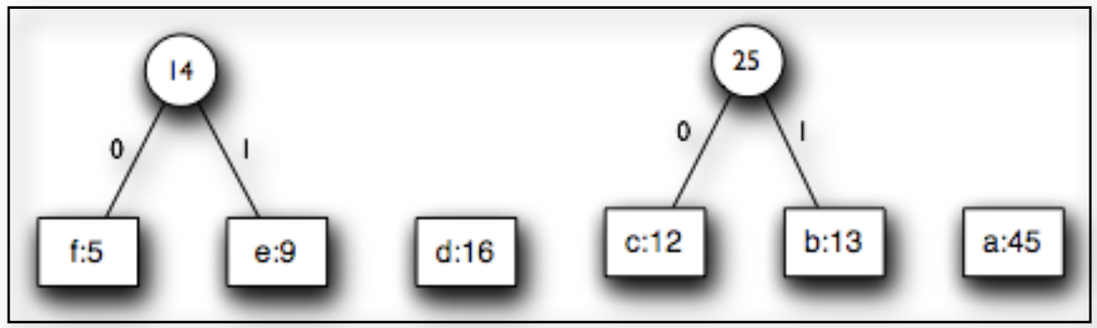


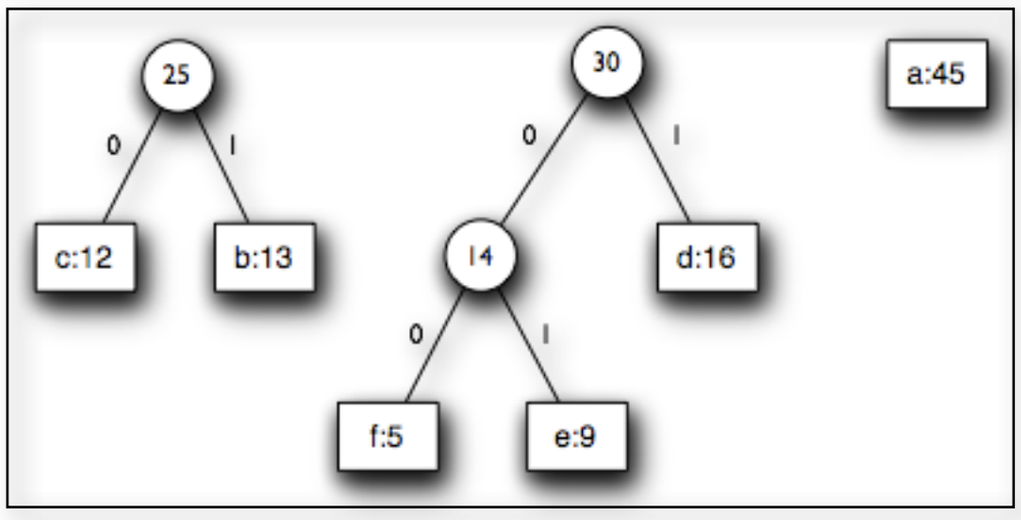


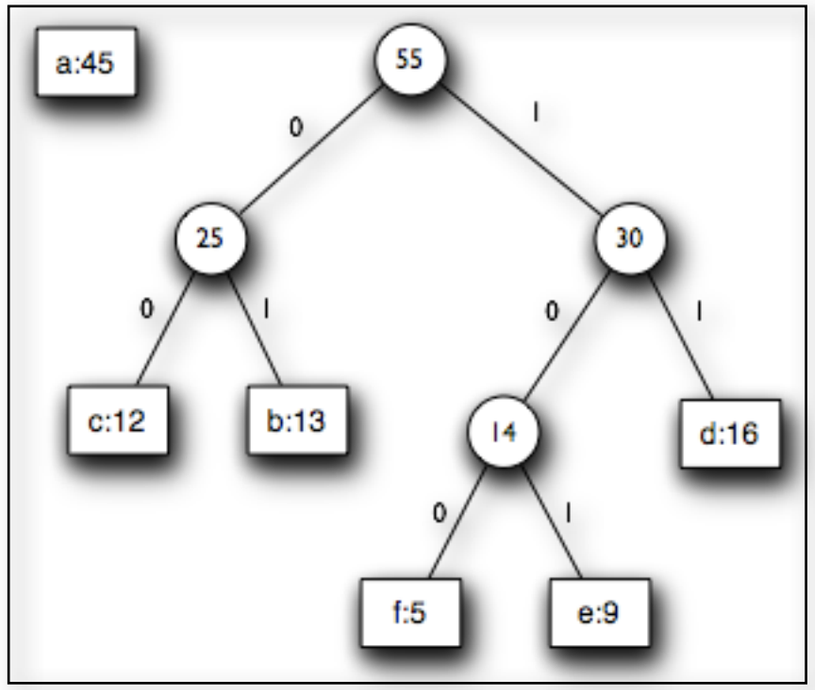


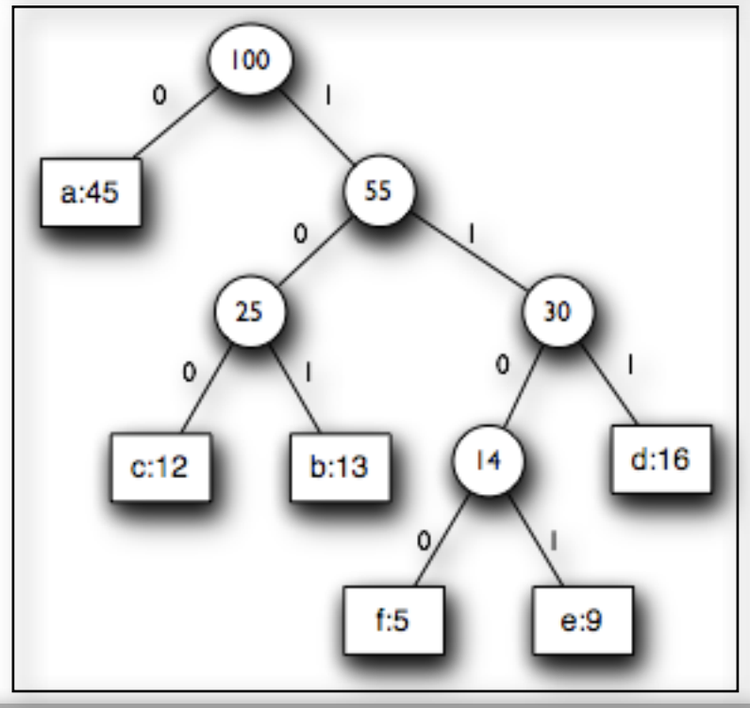
Algoritmos glotones: **códigos de Huffman**











a: 0
b: 101
c: 100
d: 111
e: 1101
f: 1100

Algoritmos glotones: **códigos de Huffman**

Algoritmos glotones: **códigos de Huffman**

- El análisis de tiempo de ejecución supone que Q ha sido implementado como un min-heap binario.

Algoritmos glotones: **códigos de Huffman**

- El análisis de tiempo de ejecución supone que Q ha sido implementado como un min-heap binario.
- Para un conjunto C de n caracteres, la inicialización de Q se puede hacer en O(n).

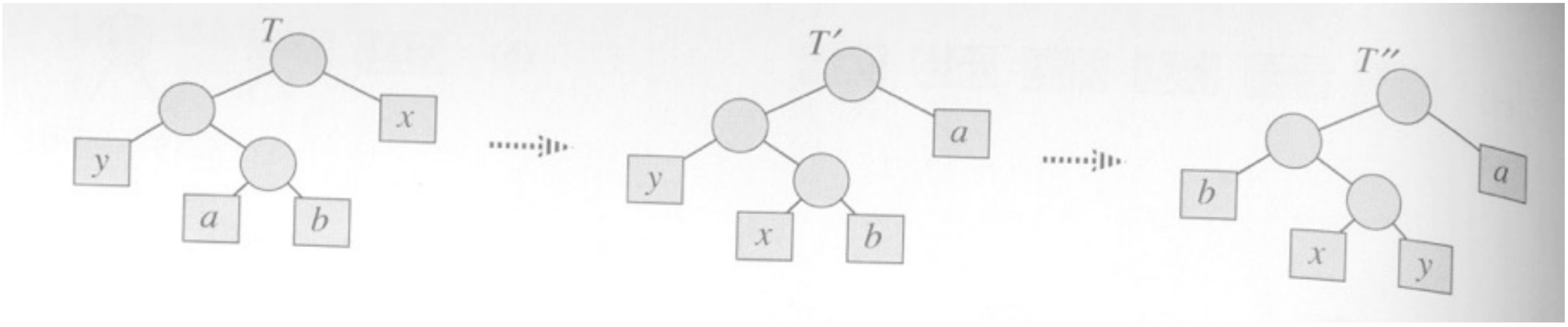
Algoritmos glotones: **códigos de Huffman**

- El análisis de tiempo de ejecución supone que Q ha sido implementado como un min-heap binario.
- Para un conjunto C de n caracteres, la inicialización de Q se puede hacer en O(n).
- El ciclo **for** de las líneas 3-8 se ejecuta n-1 veces y como la operación heap toma O(logn) el cíclo contribuye O(nlogn) al tiempo de ejecución.

Algoritmos glotones: **códigos de Huffman**

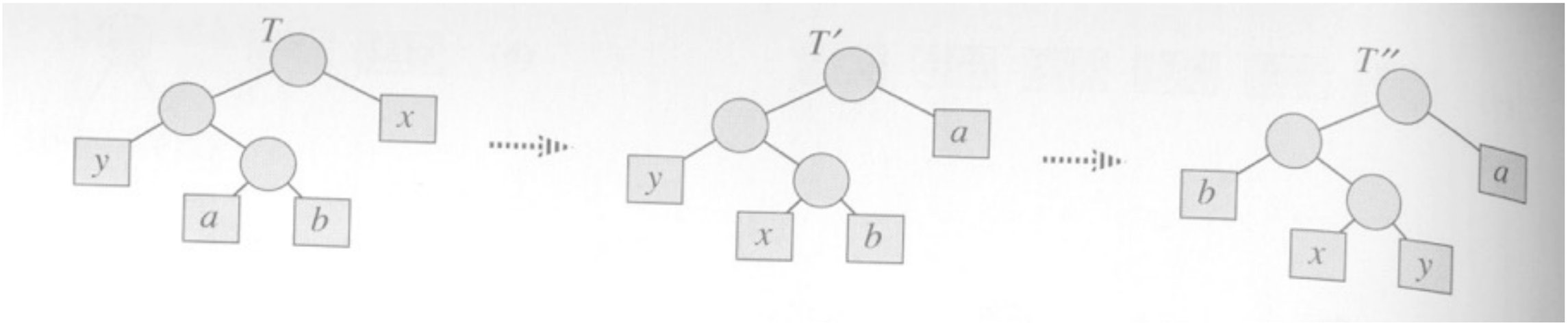
- El análisis de tiempo de ejecución supone que Q ha sido implementado como un min-heap binario.
- Para un conjunto C de n caracteres, la inicialización de Q se puede hacer en O(n).
- El ciclo **for** de las líneas 3-8 se ejecuta n-1 veces y como la operación heap toma O(logn) el cíclo contribuye O(nlogn) al tiempo de ejecución.
- El tiempo de ejecución total de HUFFMAN en un conjunto de n caracteres es O(nlogn).

Algoritmos glotones: **códigos de Huffman**



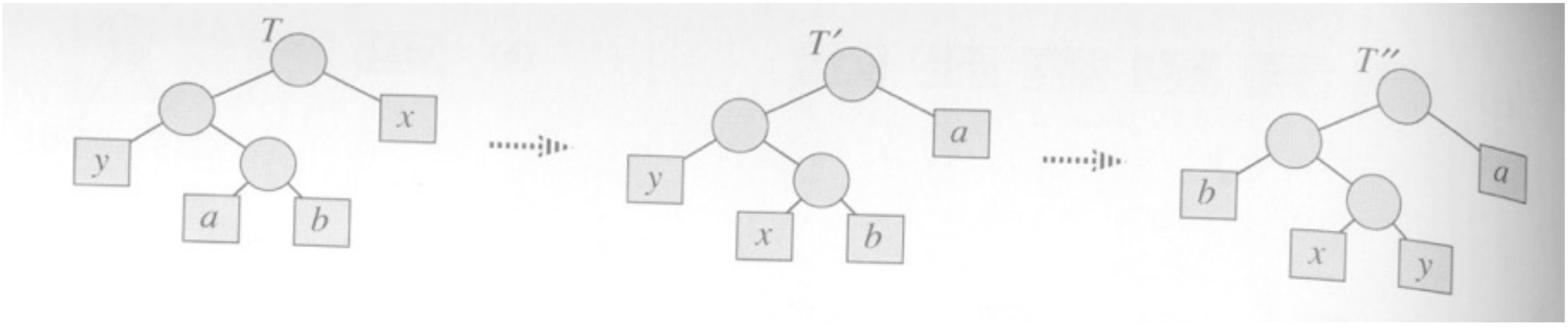
Algoritmos glotones: **códigos de Huffman**

- El algoritmo solo es correcto si exhibe las propiedades de elección glotona y subestructura óptima.



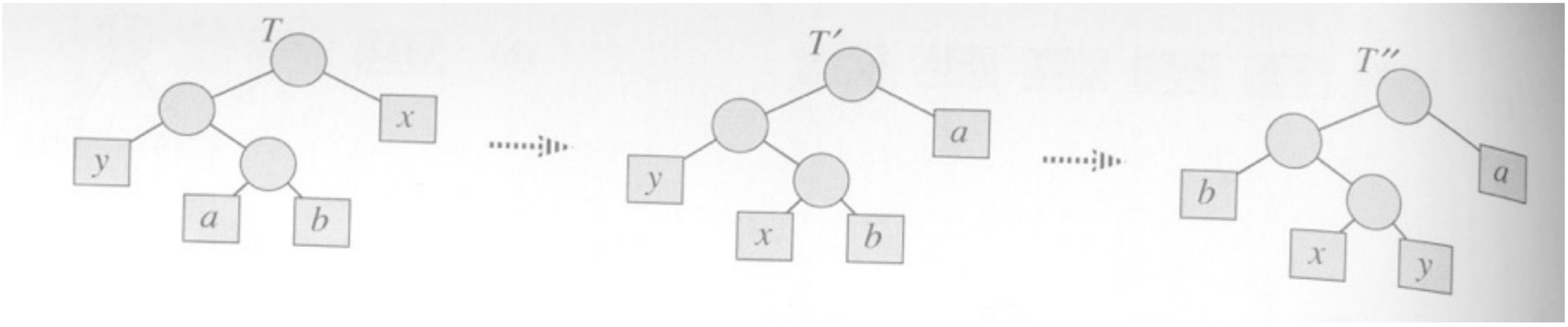
Algoritmos glotones: **códigos de Huffman**

- El algoritmo solo es correcto si exhibe las propiedades de elección glotona y subestructura óptima.
- Propiedad de la elección glotona:



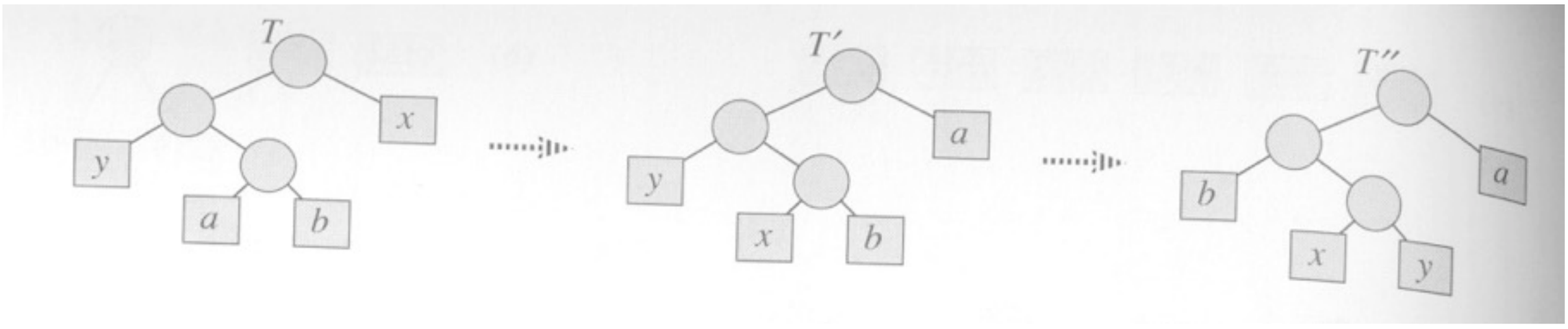
Algoritmos glotones: **códigos de Huffman**

- El algoritmo solo es correcto si exhibe las propiedades de elección glotona y subestructura óptima.
- Propiedad de la elección glotona:
  - Sea C el alfabeto en donde cada caracter c en C tiene una frecuencia f[c].



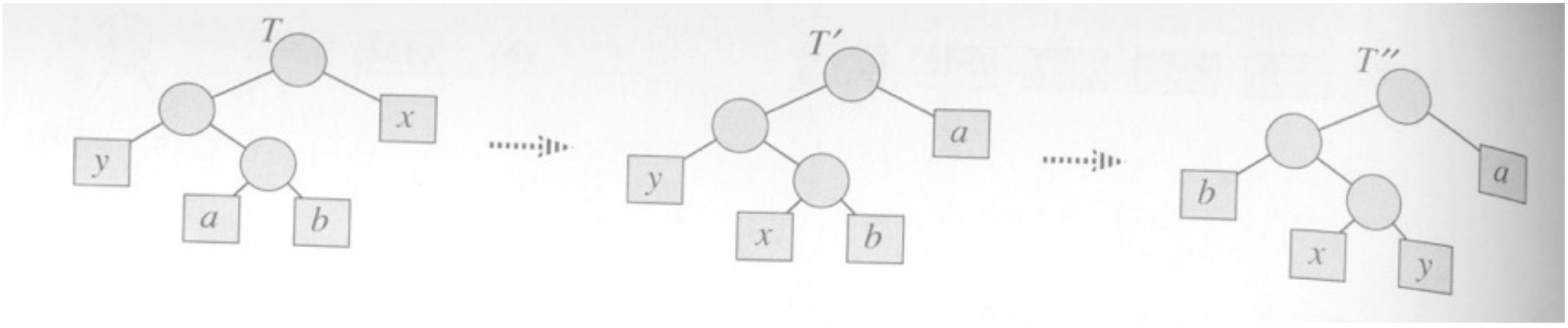
Algoritmos glotones: **códigos de Huffman**

- El algoritmo solo es correcto si exhibe las propiedades de elección glotona y subestructura óptima.
- Propiedad de la elección glotona:
  - Sea C el alfabeto en donde cada caracter c en C tiene una frecuencia f[c].
  - Sean x y y los caracteres en C con menor frecuencia.



Algoritmos glotones: **códigos de Huffman**

- El algoritmo solo es correcto si exhibe las propiedades de elección glotona y subestructura óptima.
- Propiedad de la elección glotona:
  - Sea C el alfabeto en donde cada caracter c en C tiene una frecuencia f[c].
  - Sean x y y los caracteres en C con menor frecuencia.
  - Existe un código-prefijo óptimo para C en el cuál las palabras para x y y tengan la misma longitud y difieran solamente en el último bit.



Para demostración:

$$
\begin{aligned}
B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\
&= f[x] d_T(x) + f[a] d_T(a) - f[x] d_{T'}(x) - f[a] d_{T'}(a) \\
&= f[x] d_T(x) + f[a] d_T(a) - f[x] d_T(a) - f[a] d_T(x) \\
&= (f[a] - f[x])(d_T(a) - d_T(x)) \\
&\geq 0 ,
\end{aligned}
$$

Para demostración:

- Suponemos que T representa un codigo prefijo óptimo arbitrario

$$
\begin{aligned}
B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\
&= f[x] d_T(x) + f[a] d_T(a) - f[x] d_{T'}(x) - f[a] d_{T'}(a) \\
&= f[x] d_T(x) + f[a] d_T(a) - f[x] d_T(a) - f[a] d_T(x) \\
&= (f[a] - f[x])(d_T(a) - d_T(x)) \\
&\geq 0 ,
\end{aligned}
$$

Para demostración:

- Suponemos que T representa un codigo prefijo óptimo arbitrario
- f(a)<=f(b) y f(x)<=f(y)

$$
\begin{aligned}
B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\
&= f[x] d_T(x) + f[a] d_T(a) - f[x] d_{T'}(x) - f[a] d_{T'}(a) \\
&= f[x] d_T(x) + f[a] d_T(a) - f[x] d_T(a) - f[a] d_T(x) \\
&= (f[a] - f[x])(d_T(a) - d_T(x)) \\
&\geq 0 ,
\end{aligned}
$$

Para demostración:

- Suponemos que T representa un codigo prefijo óptimo arbitrario
- f(a)<=f(b) y f(x)<=f(y)
- f(x) <= f(a) y f(y) <=f(b)

$$
\begin{aligned}
B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\
&= f[x] d_T(x) + f[a] d_T(a) - f[x] d_{T'}(x) - f[a] d_{T'}(a) \\
&= f[x] d_T(x) + f[a] d_T(a) - f[x] d_T(a) - f[a] d_T(x) \\
&= (f[a] - f[x])(d_T(a) - d_T(x)) \\
&\geq 0 ,
\end{aligned}
$$

Para demostración:

- Suponemos que T representa un codigo prefijo óptimo arbitrario
- f(a)<=f(b) y f(x)<=f(y)
- f(x) <= f(a) y f(y) <=f(b)

$$
\begin{aligned}
B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\
&= f[x] d_T(x) + f[a] d_T(a) - f[x] d_{T'}(x) - f[a] d_{T'}(a) \\
&= f[x] d_T(x) + f[a] d_T(a) - f[x] d_T(a) - f[a] d_T(x) \\
&= (f[a] - f[x])(d_T(a) - d_T(x)) \\
&\geq 0 ,
\end{aligned}
$$

Para demostración:

- Suponemos que T representa un codigo prefijo óptimo arbitrario
- f(a)<=f(b) y f(x)<=f(y)
- f(x) <= f(a) y f(y) <=f(b)

$$
\begin{aligned}
B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\
&= f[x] d_T(x) + f[a] d_T(a) - f[x] d_{T'}(x) - f[a] d_{T'}(a) \\
&= f[x] d_T(x) + f[a] d_T(a) - f[x] d_T(a) - f[a] d_T(x) \\
&= (f[a] - f[x])(d_T(a) - d_T(x)) \\
&\geq 0 ,
\end{aligned}
$$

Para demostración:

- Suponemos que T representa un codigo prefijo óptimo arbitrario
- f(a)<=f(b) y f(x)<=f(y)
- f(x) <= f(a) y f(y) <=f(b)

$$
\begin{aligned}
B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\
&= f[x] d_T(x) + f[a] d_T(a) - f[x] d_{T'}(x) - f[a] d_{T'}(a) \\
&= f[x] d_T(x) + f[a] d_T(a) - f[x] d_T(a) - f[a] d_T(x) \\
&= (f[a] - f[x])(d_T(a) - d_T(x)) \\
&\geq 0 ,
\end{aligned}
$$

Para demostración:

- Suponemos que T representa un codigo prefijo óptimo arbitrario
- f(a)<=f(b) y f(x)<=f(y)
- f(x) <= f(a) y f(y) <=f(b)

$$
\begin{aligned}
B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\
&= f[x] d_T(x) + f[a] d_T(a) - f[x] d_{T'}(x) - f[a] d_{T'}(a) \\
&= f[x] d_T(x) + f[a] d_T(a) - f[x] d_T(a) - f[a] d_T(x) \\
&= (f[a] - f[x])(d_T(a) - d_T(x)) \\
&\geq 0 ,
\end{aligned}
$$

Para demostración:

- Suponemos que T representa un codigo prefijo óptimo arbitrario
- f(a)<=f(b) y f(x)<=f(y)
- f(x) <= f(a) y f(y) <=f(b)

$$
\begin{aligned}
B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\
&= f[x] d_T(x) + f[a] d_T(a) - f[x] d_{T'}(x) - f[a] d_{T'}(a) \\
&= f[x] d_T(x) + f[a] d_T(a) - f[x] d_T(a) - f[a] d_T(x) \\
&= (f[a] - f[x])(d_T(a) - d_T(x)) \\
&\geq 0 ,
\end{aligned}
$$

Para demostración:

- Suponemos que T representa un codigo prefijo óptimo arbitrario
- f(a)<=f(b) y f(x)<=f(y)
- f(x) <= f(a) y f(y) <=f(b)

$$
\begin{aligned}
B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\
&= f[x] d_T(x) + f[a] d_T(a) - f[x] d_{T'}(x) - f[a] d_{T'}(a) \\
&= f[x] d_T(x) + f[a] d_T(a) - f[x] d_T(a) - f[a] d_T(x) \\
&= (f[a] - f[x])(d_T(a) - d_T(x)) \\
&\geq 0 ,
\end{aligned}
$$

- Y se hace lo mismo para la transicion a T”

Algoritmos glotones: **códigos de Huffman**

Algoritmos glotones: **códigos de Huffman**

- Propiedad de subestructura óptima.

Algoritmos glotones: **códigos de Huffman**

- Propiedad de subestructura óptima.
  - Sea C un alfabeto dado con la frecuencia f[c] definida para cada caracter c en C.

Algoritmos glotones: **códigos de Huffman**

- Propiedad de subestructura óptima.
  - Sea C un alfabeto dado con la frecuencia f[c] definida para cada caracter c en C.
  - Sean x y y dos caracteres en C con frecuencia mínima.

Algoritmos glotones: **códigos de Huffman**

- Propiedad de subestructura óptima.
  - Sea C un alfabeto dado con la frecuencia f[c] definida para cada caracter c en C.
  - Sean x y y dos caracteres en C con frecuencia mínima.
  - Sea C' = C - {x,y} U {z} donde f[c en C'] se define para C' como en C, excepto que f[z]=f[x]+f[y].

Algoritmos glotones: **códigos de Huffman**

- Propiedad de subestructura óptima.
  - Sea C un alfabeto dado con la frecuencia f[c] definida para cada caracter c en C.
  - Sean x y y dos caracteres en C con frecuencia mínima.
  - Sea C' = C - {x,y} U {z} donde f[c en C'] se define para C' como en C, excepto que f[z]=f[x]+f[y].
  - Sea T' cualquier árbol que represente un código-prefijo OPTIMO para el alfabeto C'. Entonces el árbol T, que se obtiene de T' al reemplazar el nodo hoja z por un nodo interno que tenga a x y y como hijos, representa un código-prefijo optimo para el alfabeto C.

Algoritmos glotones: **códigos de Huffman**

- Propiedad de subestructura óptima.
  - Sea C un alfabeto dado con la frecuencia f[c] definida para cada caracter c en C.
  - Sean x y y dos caracteres en C con frecuencia mínima.
  - Sea C' = C - {x,y} U {z} donde f[c en C'] se define para C' como en C, excepto que f[z]=f[x]+f[y].
  - Sea T' cualquier árbol que represente un código-prefijo OPTIMO para el alfabeto C'. Entonces el árbol T, que se obtiene de T' al reemplazar el nodo hoja z por un nodo interno que tenga a x y y como hijos, representa un código-prefijo optimo para el alfabeto C.
    - Ver demostración en *Introduction to Algorithms* de Thomas H. Cormen, sección 16.3

Ejemplo de Problema

Supón que quieres ir de Tiripetio, Michoacán a Anchorage, Alaska en un viaje de exploración. Tu coche puede almacenar $C > 0$ litros de gasolina y recorre $m > 0$ kilómetros por cada litro de gasolina (C y m son números enteros). Preocupado/a por el alto precio del combustible, has investigado el lugar donde se encuentran las n gasolineras en tu ruta y el precio al que vende la gasolina en cada una. Sean d_i la distancia de la i-ésima gasolinera a partir de Tiripetio (d_i también es un entero) y c_i el costo (pesos/litro, no necesariamente un entero) de la gasolina en la i-ésima estación. Además, puedes suponer que para cualquier par de gasolineras i y j, la distancia $d_i - d_j$ entre las dos es un múltiplo de m y en tus casos de prueba asegúrate que esta distancia es menor a la distancia maxima que puede recorrer el auto con el tanque lleno. Empiezas con el tanque vacío en la estación 1. Tu destino final es la gasolinera n a donde deberás llegar con al menos 0 litros de gasolina.

Implementa un algoritmo de programación dinámica que te haga gastar menos en gasolina. Recuerda que no puedes avanzar si el tanque tiene 0 o menos litros de gasolina. Una vez que decides parar en una gasolinera **debes llenar el tanque**.