Algoritmia básica
3^o Grado en Ingeniería Informática
Curso 2024/25

Práctica 1: Algoritmos Voraces

1. Consideraciones generales

1.1. Entrega de la práctica

- La entrega de la práctica se realizará en moodle, enviando un fichero practica1.zip en la tarea habilitada para ello.
- El fichero `practica1.zip` contendrá un directorio denominado `practica1_NIA1_NIA2` (siendo NIA1 y NIA2 los números identificadores de cada estudiante asignados por la Universidad de Zaragoza, y NIA1 será el NIA menor. En el caso de un grupo de práctica formado por un único alumno, el directorio tendrá como nombre `practica1_NIA` (con el identificador de ese alumno).
- El directorio incluirá los siguientes ficheros de texto:
  - Descripción general del directorio: cómo está organizado, instrucciones de compilación y ejecución, instrucciones para repetir las pruebas, etc. (tiene que llamarse `LEEME`).
  - Listados del código debidamente comentados. Deberán seguir una estructura lógica para poder encontrar y navegar adecuadamente cada una de las partes de la práctica.
  - Un programa para la *shell* (*shellscript*) que automatice la compilación y ejecución de los programas entregados con los casos de prueba. Deberá funcionar en `lab000` (tiene que llamarse `ejecutar.sh`).
  - Los ficheros auxiliares de entrada necesarios para ejecutar las pruebas del punto anterior (estarán en un subdirectorio se llamará `pruebas`).
  - El ejecutable no tiene menús que permitan la interactividad, todas las acciones posibles se controlan a través de la llamada (su nombre se indica más abajo).
  - El directorio incluirá también un informe con la presentación y análisis de resultados (fichero PDF, máximo 3 páginas sin portada). Indicar: nombre, apellidos y NIA de cada miembro del grupo de práctica.
- **La fecha límite de entrega para la primera convocatoria** es el sábado siguiente a la segunda sesión de prácticas del grupo correspondiente. Es decir:

Grupo	Fecha y hora
Jueves A	15/02/2025 8:00AM
Viernes A	15/02/2025 8:00AM
Jueves B	22/02/2025 8:00AM
Viernes B	22/02/2025 8:00AM

1.2. Evaluación

- En la calificación se tendrán en cuenta los siguientes aspectos: documentación, diseño e implementación, diseño de casos de prueba, análisis de las pruebas realizadas y facilidad para la repetición de las pruebas por los profesores.
- Se aplicarán las reglas de tratamiento de casos de plagio explicadas en la presentación de la asignatura.

2. Enunciado

Tarea 1. Diseño. Diseñad un algoritmo *voraz* que toma como entrada un texto y construye el árbol y compacta el fichero. Diseñad el algoritmo para descompactar.

Tarea 2. Implementación. Se debe desarrollar un programa compactador/descompactador `huf` que implemente el método de compactación de ficheros basado en el código de Huffman. Las formas de ejecución deberán ser las siguientes:

- Para compactar:

```
lab000:> huf -c <nombre de fichero>
```

  donde: `<nombre de fichero>` es el nombre de un fichero cualquiera (de texto o binario).

  El programa generará el fichero compactado `<nombre de fichero>.huf`.

- Para descompactar:

```
lab000:> huf -d <nombre de fichero>
```

  donde:

  `<nombre de fichero>` es el nombre de un fichero compactado utilizando nuestro algoritmo. El programa generará el fichero original.

Tarea 3. Mejorando algunos casos. Este método puede funcionar mal si hay un número significativo de símbolos que tienen frecuencia baja. Para mejorar esta situación se utilizan códigos de Huffman de longitud limitada. Esto es, el código es de longitud variable, como antes, pero ahora hay una constante L que es la longitud máxima de la codificación. El algoritmo que proponemos fue descrito en [LH90][1].

Para aplicar este método se utiliza un algoritmo de combinación de paquetes (*package-merge algorithm*) que permite reorganizar los caracteres para tener en cuenta la nueva limitación.

Utilizaremos una propiedad que sirve para verificar que la longitud máxima es suficiente para codificar los caracteres (necesitamos un código libre de prefijos). Esto se hace aplicando la desigualdad de Kraft-McMillan[2]:

> Si tenemos un alfabeto $S = \{c_1, c_2, \ldots, c_n\}$ que vamos a codificar con un alfabeto de r símbolos y tal que las longitudes $l_1, l_2, \ldots, l_n$ entonces debe cumplirse que $\sum_{i=1}^{n} r^{-l_i} \leq 1$.

[1] `https://ics.uci.edu/~dhirschb/pubs/LenLimHuff.pdf`
[2] `https://en.wikipedia.org/wiki/Kraft-McMillan_inequality`

En nuestro caso el alfabeto de destino tiene dos símbolos, por lo tanto $r = 2$.

No verificaremos que una longitud es suficiente (aunque si no lo es tendremos problemas para descompactar) y para calcular las nuevas codificaciones procederemos de la siguiente manera:

1. Recorremos los caracteres que queremos codificar de mayor a menor frecuencia y actualizamos su valor con el mínimo entre la frecuencia que tenían originalmente y la longitud máxima de codificación:

   $$fNueva[c_i] = min(f[c_i], L), \forall 1 \leq i \leq n$$

   Esto es, para los que exceden la longitud L la sustituimos por ese valor.

   Ahora el árbol que podríamos construir con las nuevas frecuencias ya no sería libre de prefijos, puesto que hemos añadido caracteres de alguna longitud inferior a la que teníamos, y necesitamos redistribuir el resto de (longitudes de los) códigos.

2. Volvemos a recorrer la lista de caracteres en el mismo orden (con las nuevas frecuencias) y para aquellos que tengan longitud de codificación (la nueva frecuencia) menor que la máxima $fNueva[c_i] < L$, la incrementamos en 1 mientras se cumpla la desigualdad.

   Ahora puede ser que hayamos cambiado valores pequeños por otros más grandes de manera innecesaria, puesto que no hemos hecho ninguna verificación en ese sentido.

3. Hacemos otro recorrido, en esta ocasión de menor a mayor frecuencia, reduciendo su tamaño de codificación tanto como sea posible: ahora se trata de disminuir la longitud de codificación de los caracteres mientras se siga cumpliendo la desigualdad de Kraft-McMillan.

   $$fNueva[c_i] = fNueva[c_i] - 1$$

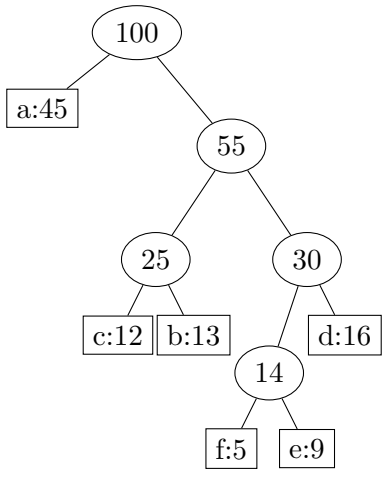
Ayuda: Es buena idea ir actualizando el valor de la desigualdad en los tres pasos, para que cada cambio que hacemos en las frecuencias nos permita actualizarla sin tener que hacer todas las operaciones.

Ejemplo

En el ejemplo de clase, vamos a suponer que queremos tener códigos de longitud, como máximo, 3 ($L = 3$).

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

El árbol que obteníamos era:



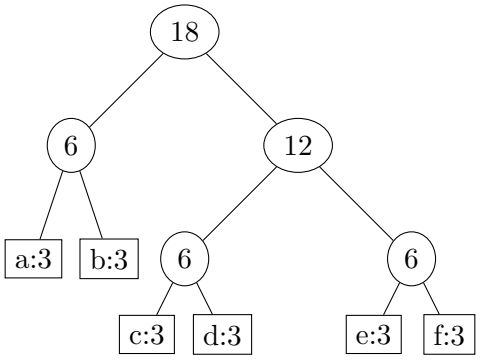
El primer paso de nuestro algoritmo cambiaría la tabla de frecuencias por:

	a	b	c	d	e	f
longitud	3	3	3	3	3	3

El segundo paso no tendría efecto en este caso (todos los valores son iguales a L), y el tercero, reduciría los dos primeros a dos:

	a	b	c	d	e	f
longitud	2	2	3	3	3	3

Construyendo el árbol (con el algoritmo voraz de Huffman) llegaríamos a la codificación:



El fichero resultante comprimido tendría un tamaño de:

$$2 \cdot (45 + 13) + 3 \cdot (12 + 16 + 9 + 5) = 242 \text{ mil.}$$

Las nueva forma de ejecución deberá ser las siguientes:

- Para compactar:

```
lab000:> huf -l 3 -c <nombre de fichero>
```

  donde:

  - `<nombre de fichero>` es el nombre de un fichero cualquiera (de texto o binario).
  - `-l 3` indica la longitud máxima para la codificación.

  El programa generará el fichero compactado `<nombre de fichero>.huf`.

- Para descompactar:

```
lab000:> huf -d <nombre de fichero>
```

  Igual que en el caso anterior.

Tarea 4. Experimentación. Verificad mediante casos de prueba la corrección del programa y analiza la eficiencia (tiempos de ejecución) de los algoritmos implementados.

- El programa debería funcionar con textos escritos, al menos, en texto castellano (con tildes, eñes, ...). Idealmente debería poder compactar y descompactar cualquier fichero (incluso binarios).
- Debemos probar con ficheros pequeños, largos[3] y algunos casos límite:
  - Todos los caracteres con un número parecido de apariciones
  - Unos pocos caracteres con muchas apariciones y el resto pocas.
  - Unos pocos caracteres con pocas apariciones y el resto muchas.

Referencias

[LH90] Lawrence L. Larmore and Daniel S. Hirschberg. A fast algorithm for optimal length-limited huffman codes. *J. ACM*, 37(3):464–473, July 1990.

[3] Por ejemplo, `https://www.cervantesvirtual.com/obra-visor/el-ingenioso-hidalgo-don-quijote-de-la-mancha-6/html/05f86699-4b53-4d9b-8ab8-b40ab63fb0b3_2.html`