Зад. 2 (Алгоритъм ха Хъфман за компресия на данни – честотни таблици)

Не е нова идеята да се предава информация по възможно най-икономичен начин. Например естествените говорими езици и писмени азбуки неизменно страдат от излишество. При тях обаче икономичното предаване на информация не е най-важната страна; макар и не оптимални от тази гледна точка те са удобни за използване от човек. За оптимално кодиране са разработени специални системи, каквито са например стенографската, морзовата азбука, азбуката за глухи, които са лишени от доста удобства. С навлизането на компютрите се появява възможност автоматично сравнително бързо да се "превежда" даден поток от информация на по-икономична азбука и обратно. Бързо намират приложение алгоритмите за компресиране на информация, а те от своя страна се доразработват и оптимизират, за да навлязат във всекидневна употреба. Всеки е използвал поне една универсална програма за компресиране (ARJ, ZIP, RAR, ACE) и се е възползвал от компресии на мултимедия - звук (MP3, OGG), картина (GIF, JPEG), филмов клип (MPEG), дори и извън всекидневната работа с компютрите (компресия на звук по GSM). Алгоритмите за компресия имат стабилна математическа основа и стават все по-сложни и с по-добра степен на компресия с нуждата от тяхното прилагане.

Алгоритъм на Хъфман

Алгоритъмът на Хъфман, разгледан тук е сравнително прост универсален алгоритъм за компресия без загуба на данни (за разлика от алгоритмите със загуба, стоящи в основата на MP3, например). При него се предполага, че е даден краен поток от числа в някакъв предварително фиксиран интервал. Ще считаме, че става дума за символи, кодирани със ASCII код, т.е. ще разглеждаме информацията като поредица от байтове (числа в интервала 0..255). Алгоритъмът се базира на простата идея, че най-често срещаните символи в поредицата трябва да се записват с най-малък брой битове. Така той построява нова азбука, която следва тази идея и след това превежда информацията в новата азбука. Кодирането е обратимо, тоест по кодираната последователност може да се декомпресира - да се намери първоначалната поредица.

Построяване на дърво на Хъфман

Нека трябва да компресираме даден низ от символи. Искаме да построим двоично дърво, от което ще определим азбука за компресиране.

Алгоритъмът за построяване на дърво се състои от следните стъпки:

1. Създава се честотна таблица на низа - за всеки символ се записва броят на срещанията му.
2. Нека различните символи в низа са n на брой. Създаваме n дървета от по един елемент, където всяко дърво съдържа символ и броя на срещанията му.
3. Намираме двете дървета, които в корените имат най-малко число. Обединяваме дърветата в ново дърво, като в корена записваме сумата от стойностите в двете намерени дървета.
4. Повтаряме стъпка 3, докато не получим само едно дърво - дървото на Хъфман за дадения низ.

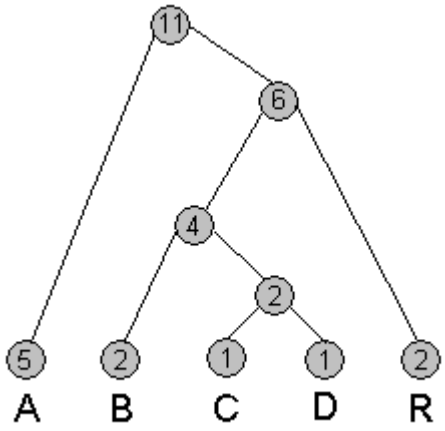
Така построено дървото е двоично и има точно n на брой листа, като на всяко листо отговаря един символ от честотната таблица. По начина на построение се вижда, че по-често срещаните символи се намират по-близо до корена от по-рядко срещаните. Това се вижда и в примера, даден по-долу.

Пример:

Нека имаме низа "ABRACADABRA". Честотната таблица за низа е:

Символ:	Брой срещания:
A	5
B	2
C	1
D	1
R	2

Строим дървото по следния начин:

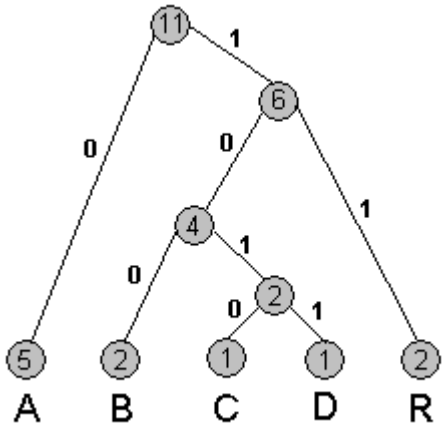


Построяване на азбуката по дървото на Хъфман

На всеки клон от дървото съпоставяме двоична цифра 0 или 1: 0 за ляв клон, 1 за десен клон. Така на всеки път от корена до някое листо отговаря двоичен низ. Тъй като всяко листо е символ от низа, можем да съпоставим на всеки символ двоичната последователност, която съответства на пътя от корена до листото на символа. Тъй като най-често срещаните символи са най-близко до корена, на тях ще отговарят най-къси последователности. Обратно - на рядко срещаните символи съответстват дълги последователности.

Пример:

Продължаваме примера отгоре. Дървото, отбелязано с 0 и 1 изглежда така:



Таблицата за кодиране е:

Символ:	Код:
A	0
B	100
C	1010
D	1011
R	11

След като получим таблицата за кодиране, извършваме кодиране на низа - всеки символ заместваме с неговия код. Така получаваме последователност от 0 и 1. Ако разбием на блокове по 8 бита, можем да получим и изход от байтове.

Пример:

```
ABRACADABRA -->
0 100 11 0 1010 0 1011 0 100 11 0 -->
01001101010010110100110 -->
01001101 01001011 0100110 -->
77 75 38
```

От 11 символа (байта) = 8*11 бита = 88 бита получихме 23 бита компресирана информация - около 26% от оригиналния обем. Получихме четири пъти по-малко описание на "ABRACADABRA".

Декомпресиране на компресирана информация

Разкомпресирането на данните става лесно при условие, че имаме дървото на Хъфман. Вървим едновременно по двоичния низ и по дървото, като всеки път като срещнем 0 завиваме наляво, а при 1 - надясно. Когато стигнем до листо, записваме съответния символ и рестартираме от корена. Така стъпка по стъпка получаваме първоначалния низ.

Пример:

```
0100110101001010110100110 -->
0 100 11 0 1010 0 1011 0 100 11 0 -->
A B R A C A D A B R A
```

Задача

Разглеждаме алгоритъмът на Хъфман в частта му свързана с построяване на честотна таблица на входния поток от информация. Задачата е да се напише програма, която строи честотна таблица на даден двоичен или текстов (достатъчно голям) файл. Програмата да разпределя по подходящ начин работата за построяване на честотната таблица между две или повече нишки (задачи);

Изискванията към програмата са следните:

(о) Чете името на входния файл от подходящо избран команден параметър – например **"-f file.dat"**;

(о) Втори команден параметър задава максималния брой нишки (задачи) на които разделяме работата по построяването на честотната таблица – например **"-t 1"** или **"-tasks 3"**;

(о) Извежда подходящи съобщения на различните етапи от работата си, както и времето отделено построяване на честотната таблица на входния файл;

(о) Да се осигури възможност за „quiet“ режим на работа на програмата, при който се извежда само времето отделено за построяване на, отново чрез подходящо избран друг команден параметър – например **"-q"**;

ЗАБЕЛЕЖКА:

(о) При желание за направата на подходящ графичен потребителски интерфейс (GUI) с помощта на класовете от пакета **javax.swing** задачата може да се изпълни от **двама души**; Разработването на графичен интерфейс не отменя изискването Вашата програма да поддържа изредените командни параметри. В този случай към функцията на параметъра параметъра **"-q"** се добавя изискването **да не запуска** графичния интерфейс. Причината за това е, Вашата програма да може да се тества отдалечено.

(о) Задачата може да се реши и с помощта на RMI (**java.rmi**). За целта трябва да се помисли за разпределения достъп до общия ресурс в случая файл – чрез копие на всяка от машините извършващи преброяването или чрез подходящ интерфейс към клиентското приложение запускащо отдалечените пресмятания.

Заключителни бележки

Описаният алгоритъм е един от най-простите алгоритми за компресиране. Това е универсален алгоритъм без загуба на информация за кодиране с променлива дължина. За декодиране е необходимо да се пази допълнителна структура - в случая честотна таблица или дърво на Хъфман. За сравнение има алгоритми (LZ77, LZ78), които не се нуждаят от допълнителна структура, а строят такава динамично по време на компресия и декомпресия въз основа на самата информация. Направени са много подобрения на алгоритъма на Хъфман, подобряващи степента на компресиране. Последното е за сметка на усложняване на алгоритъма.

Уточнения:

(о) В условието на задачата се говори за разделянето на работата на две или повече нишки. Работата върху съответната задача на една нишка ще служи за еталон, по който да измерваме евентуално ускорение (T1). Тоест в кода реализиращ решенията на задачите

трябва да се предвиди и тази възможност – задачата да бъде решавана от единствена нишка (процес); Пускайки програмата да работи върху задачата с помощта на единствена нишка, ще считаме че използваме серийното решение на задачата; Измервайки времето за работа на програмата при работа с „p“ нишки - Tp, изчисляваме Sp. Представените на защитата данни за работата на програмата, трябва да отразят и ефективността от работата и, тоест да се изчисли и покаже Ep.

(o) Командните аргументи (параметри) на терминална (конзолна) Java програма, получаваме във масива String args[] на main() метода, на стартовия клас. За „разбирането“ им (анализирането им) може да ползвате и външни библиотеки писани специално за тази цел . Един добър пример за това е: Apache Commons CLI (http://commons.apache.org/cli/).