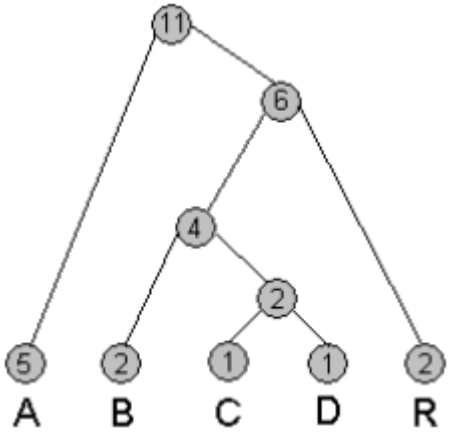
ХЪФМАНОВО КОДИРАНЕ

Да се реализира класическия алгоритъм за Хъфманово кодиране и декодиране.

Алгоритъмът на Хъфман е сравнително прост и универсален алгоритъм за компресия без загуба на данни, т.е. след кодиране и декодиране на даден файл резултатът съвпада с оригинала. Този алгоритъм е способен да кодира всякакви данни, тъй като те винаги могат да бъдат представени като поредица от байтове. За целта на този проект ще приемаме, че данните ни представляват списък от произволни обекти – числа, стрингове, булеви стойности или други. Не е задължително списъкът да бъде хомогенен. Стринговете също ще считаме за списъци от букви, както в езика Haskell. Елементите на този списък ще наричаме за простота ‚символи' в по-долния пример, но това не е ограничение за тях. Алгоритъмът за кодиране работи в четири основни стъпки:

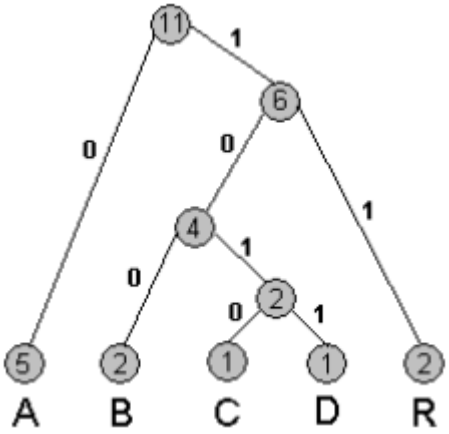
1. Генериране на честотен списък от данните, където на всеки елемент се съпоставя броя негови срещания. За думата "abracadabra" (списък от букви) в Haskell честотният списък би представлявал `[('a',5), ('b',2), ('c',1), ('d',1), ('r',2)]`. Подредбата в този списък не е от значение.
2. Създаване на дърво на Хъфман на базата на честотния списък. В началото започваме с множество дървета с по 1 връх, като всяко едно от тях отговаря на конкретен символ (обект) от списъка и съдържа в корена си броя срещания на този символ. След това от множеството се изваждат две от дърветата с най-ниски стойности в корените и се обединяват в ново дърво, където в корена му се запазва сумата от тези две стойности. Това дърво после се добавя обратно в множеството. Тази стъпка се повтаря, докато остане само едно дърво – него наричаме дървото на Хъфман за дадения списък. За същия пример по-горе едно възможно коректно дърво на Хъфман би изглеждало така:



Когато има повече от две дървета с еднаква най-ниска стойност в множеството, няма значение кои две от тях ще бъдат избрани за обединение. Също така няма значение кое дърво става ляво поддърво на новото и кое – дясно. Единственото условие, което е

задължително да се спазва, е на всяка стъпка да се обединяват две дървета с най-ниските стойности в корените си. В последствие най-често срещаните в оригиналния списък символи ще бъдат най-близко до корена, и обратното.

3. Намиране на кодовете на всеки символ, използвайки построеното дърво. Нека на всяко ребро в дървото се съпостави символ 0 или 1 в зависимост от това дали е ребро към ляво поддърво или дясно поддърво по следния начин:



Всяко едно листо на дървото тогава ще притежава уникален двоичен код, отговарящ на единствения път от корена до това листо. Нещо повече, нито един от кодовете за дадена буква няма да е префикс на кода за друга буква. За всички букви тогава може да се изгради таблица с кодовете, която в същия пример би изглеждала така:

`[('a',"0"), ('b',"100"), ('c',"1010"), ('d',"1011"), ('r',"11)]`.

Тъй като по-често срещаните символи в списъка са разположени по-близо до корена, то те ще имат по-къси кодове.

4. Кодиране на оригиналния списък – всеки символ от него се замества със своя код, слепвайки всичко в последователност от нули и единици:

`"abracadabra" →`

`0 100 11 0 1010 0 1011 0 100 11 0 ->`

`"01001101010010110100110"`.

В този пример сведохме данни от 78 бита (приемайки, че един символ е един байт) до последователност от 23 бита. За целите на този проект все пак ще работим с компресираните данни като стринг, а не битова последователност.

Резултатът от кодирането трябва да включва и Хъфмановото дърво заедно с компресираните данни. Без дървото, дори само с честотния списък, данните не могат да бъдат възстановени еднозначно.

Процесът на декодиране е значително по-прост и може да се осъществи само с помощта на изграденото дърво и компресираните данни. Започваме да ги обхождаме, символ по символ, и успоредно с това обхождаме и дървото от корена към листата. Всеки път, когато срещнем '0', преминаваме на лявото поддърво, а при '1' – на дясното. В момента, в който достигнем листо, запазваме неговия съответстващ елемент от оригиналните данни и започваме наново от корена.

Целта на проекта е да бъдат реализирани двете основни функции – за кодиране и декодиране. Функцията за кодиране трябва да приема освен самите данни и предикат за сравнение на отделните елементи – напр. `eq?` или `equal?` в Scheme, (==) в Haskell, или друг специално дефиниран такъв. Предикатът служи за разпознаване на еднаквите елементи в оригиналните данни. Тази функция трябва да връща само построеното Хъфманово дърво в избрано от Вас представяне и компресираните данни под формата на стринг. Функцията за декодиране трябва да приема само Хъфмановото дърво и компресираните данни и по тях да възстановява оригиналните такива. Всички решения, които си „подпомагат“ с таблицата с кодове при декодиране, ще получават малко по-малко точки. Очевидно е, че ако резултатът от функцията за кодиране подадем на функцията за декодиране, то трябва да получим съвършено копие на оригиналните данни. Кодирането на празен списък би трябвало да връща празно дърво и празен стринг, а тяхното декодиране да връща обратно празен списък, което е коректно обработване.

Ако четенето на данните или писането на резултата се осъществяват с файл, текстови или двоичен, ще бъдат присъдени бонус точки за проекта. Всякаква допълнителна функционалност ще бъде също поощрявана, стига да не нарушава основния алгоритъм.