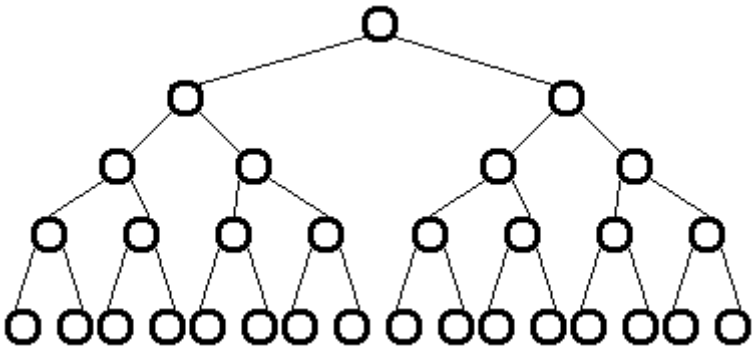
Дървета, пирамидална сортировка, използване на дърветата за съхранение на данни.
Дърветата са сложни структури. Обичайно дърветата са двоични но могат да се срещнат и други видове. По-долу ще разгледаме основните характеристики на едно дърво и различни примери за дървета.

Двоично дърво:



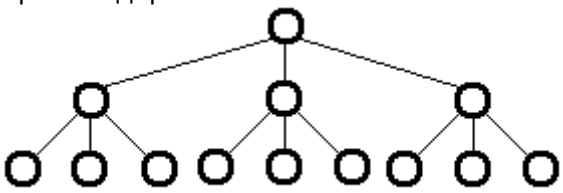
Най-горното кръгче представлява корен и към коренът нямаме входна дъга.

От всеки възел могат да излязат най-много 2 дъги.

Възли който нямат приемници (от тях не излизат дъги) се наричат крайни възли или листа на дървото.

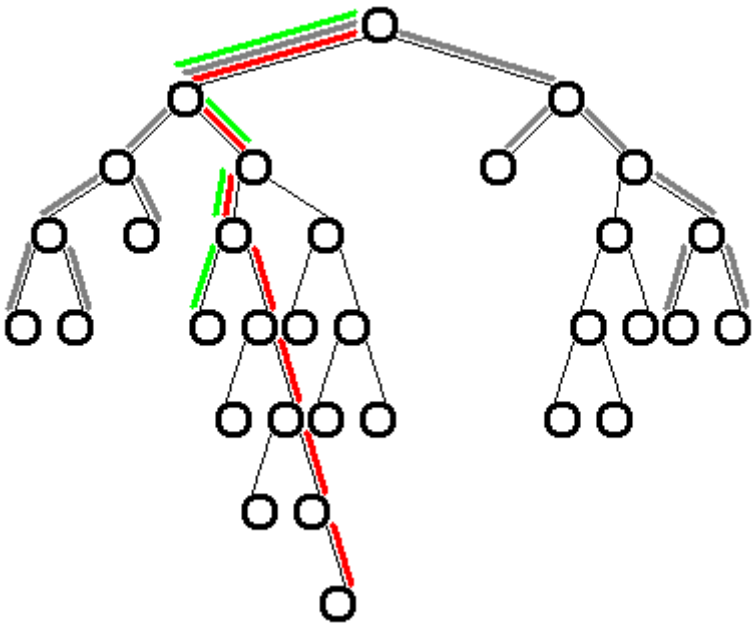
Всяко кръгче представлява възел на двоичното дърво.
Всяка права представлява дъга.
Двоичното дърво представлява ацикличен граф.

Троично дърво:



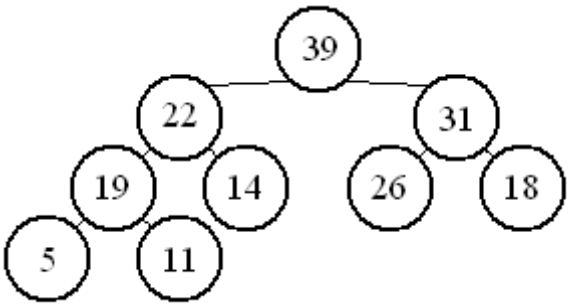
... възможни са всякакви броеве на наследниците на, като това определя видът на дървото.

Височина на възел – максималният път от краен възел до коренът на дървото.
Височина на корена – най-голямата височина = височината на дървото. (H)
Дълбочина на възела – дължина от точно определено избрано листо до корена.



Дърветата се изграждат с указатели, като всеки възел може да съдържа толкова указатели колкото дъги излизат от него. За да изградим едно дърво без да използваме указатели, е необходимо:

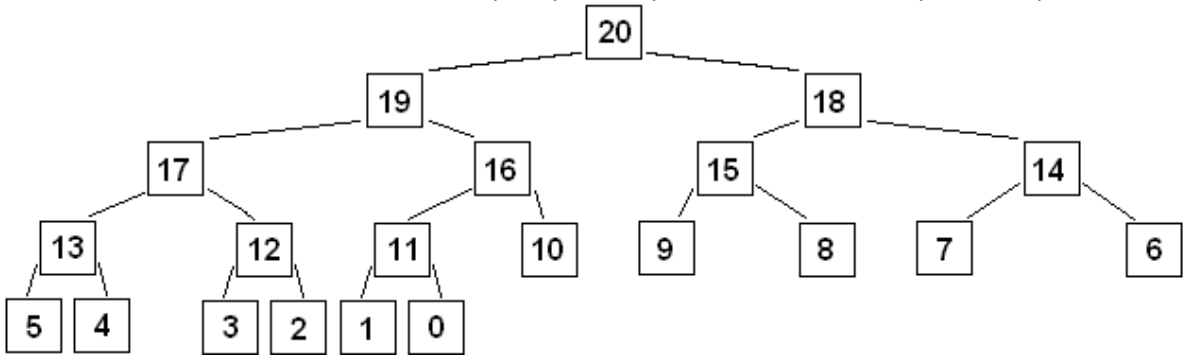
- Произволен краен възел трябва да е с дължина (H) или (H-1)
- Произволен краен възел с дълбочина (H) трябва да е по наляво от (H-1) в структурата с която е записан.
- Стойностите на произволен възел трябва да е по-голяма от стойностите на неговите приемници, за да бъде подредено дървото.



В нашите представи дърветата могат да изглеждат по горните начини, но всяка последователност от данни може да бъде подредена като дърво. Ако вземем следната последователност която може да бъде списък, масив или друго.

20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

И ако тази последователност искаме да представим като дърво (примерно 2-ично). То тогава трябва да използваме формулата X -> 2*X ; 2*X +1. Където X е поредният номер на елементът до който сме достигнали. В нашия случай поредния номер на елемент със стойност 20 е номер 1, поредния номер на елемент със стойност 15 е номер 6, поредният номер на елемент със стойност 0 е 21. За да разберем кои от елементите на кой са наследници. Използвайки формулата => първият елемент със стойност 20 има наследници с позиции- 2*1 = 2-ра позиция (стойност 19) и 2*1 +1 = 3-та позиция (стойност 18). Примерно елемент 6-ти със стойност 15 има наследници – 2*6 = 12-ти елемент (стойност 9), 2*6+1 = 13-ти елемент (стойност 8). Елемент 8-и със стойност 13 има наследници - 8*2 = 16-ти елемент със стойност 5 и 8*2 +1 = 17-ти елемент със стойност 4. Можем да проверим твърдението си като направим дървото:



Виждаме че:
елемент 1 със стойност 20 има наследници елементи със стойности 19 и 18,
елемент 6 със стойност 15 има наследници елементи със стойности 9 и 8,
елемент 8 със стойност 13 има наследници елементи със стойности 5 и 4.

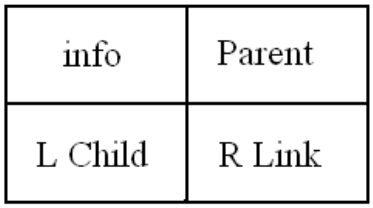
Не е необходимо да правим дървото всеки път за да видим кои елементи на кой са наследници. Във формулата както обяснихме X е поредния номер на елемента а дали да добавим стойност 1 или не ни изгражда съответно двете различни позиции на наследяващите елементи. Стойността за умножение 2 зависи от това какво е дървото („ичноста" на дървото – 2-ично, 2-ично ...). В по-общ вид тази формула може да бъде N*X+a. Където N е „ичноста" на дървото, X е поредният номер на достигнатият елемент, а „а" е стойност в интервал от 0 до N-1. Примерно за 3-йчно дърво => N=3, a=0,1,2. X -> 3*X , 3*x +1 , 3*x +2.

Пирамидална сортировка.

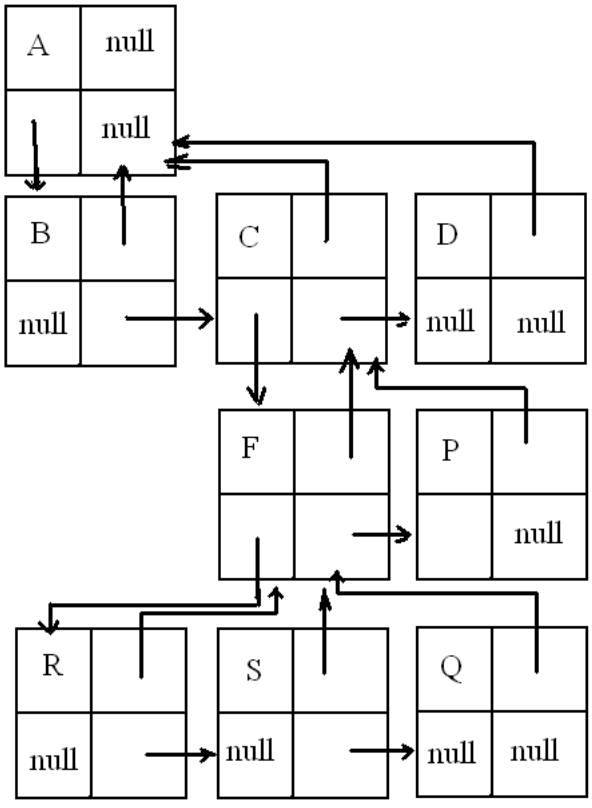
Пирамидалната сортировка е метод за бързо сортиране на едно дърво. Нейната сложност е N.log N, където N е броят на възлите в дървото, а основата на логаритъма зависи от „ичноста" на дървото. В пирамидалната сортировка всеки елемент се сравнява с неговите наследници ако има такива и ако един от тях или и повече са по-големи по стойност от родителя си се взима този с най голяма стойност и се разменят мястото на взетия и родителят му. Това се прави за всички елементи който имат наследници (като все пак се обхождат всички N елементи))и се прави (logN) пъти. От където се получава сложност N*logN.

Начин на задаване на произволно дърво – структура, предложена от Кнут.

Основен елемент на структурата:



Реализация на примерно дърво:



Структура на един възел в дърво.

```
struct TreeNode {
   int item;          // The data in this node.
   TreeNode *left;    // Pointer to the left subtree.
   TreeNode *right;   // Pointer to the right subtree.
}
```

Начини за обхождане на възлите в дърво.

Preorder:

```
void preorderPrint( TreeNode *root ) {
   if ( root != NULL ) {
      cout << root->item << " ";       // Print the root item.
      preorderPrint( root->left );     // Print items in left subtree.
      preorderPrint( root->right );    // Print items in right subtree.
   }
}
```

Postorder:

```
void postorderPrint( TreeNode *root ) {
   if ( root != NULL ) {
      postorderPrint( root->left );     // Print items in left subtree.
      postorderPrint( root->right );    // Print items in rightsubtree.
      cout << root->item << " ";        // Print the root item.
   }
}
```

Inorder:

```
void inorderPrint( TreeNode *root ) {
   if ( root != NULL ) {
      inorderPrint( root->left );     // Print items in left subtree.
      cout << root->item << " ";      // Print the root item.
      inorderPrint( root->right );    // Print items in right subtree.
   }
}
```

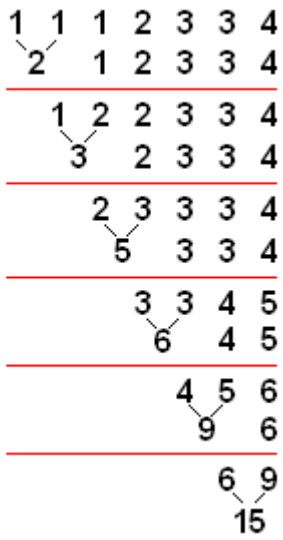
Хафманово кодиране. Хафмановото кодиране се явява компресия без загуби. За да се получи това е необходимо количество информация да запази своето значение но да намали обема си. Практически информацията би трябвало да не може да намали обема си без да се наруши целостта и защото тя (информацията) се представя само от определения брой символи нужен за нейното представяне без излишни данни. Но нека не забравяме че начините за представяне на информацията са предвидени да бъдат универсални за всякаква информация, докато в нашия случай ние ще вземем определено съобщение. Съобщението ни може да не съдържа разновидна информация а може да съдържа повтарящи се елементи. Ние можем да вземем някой от тези елементи да го представим по определен начин и в следващият момент в който достигнем до същия елемент информация не е необходимо да го представяме на ново, а можем да използваме първият, който вече сме представили. Как става това? На машинен език съществуват различни стандарти, Който включват различен брой символи и заради големия брой са необходими големи количества информация за представянето на отделните символи. За да не съвпада представянето на различните символи, те могат да бъдат представени с 8,16,32,64 и тн. бита. Ако в нашето съобщение имаме примерно само 30 символа. То те ще могат да се кодират с log 30 при основа 2 със закрагление на горе на получения резултат, тоест могат да бъдат кодирани с 5 бита.

В нашето съобщение някои символи могат да се срещат много по често от други. Ако ние направим по-често срещаните символи с по къси кодове а по-рядко срещаните символи с по дълги кодове можем да спестим още от големината на цялото съобщение. В това се крие същината на Хафмановото кодиране.

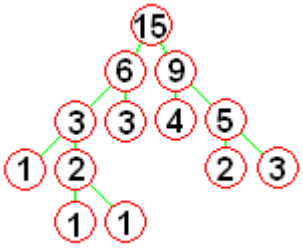
Пример: Пипи пее песни. Ще направим честотен анализ на това съобщение като запишем всеки от срещаните символи и съответно колко често се среща (без оглед на главни букви).
п - 4 // и - 3 // интервал – 2 // е -3 // с – 1 // н – 1 // точка - 1

Ще подредим символите според броя на това колко пъти са се срещнали (тоест тяхната честота в съобщението): п, и, е, _, с, н, .

Получаваме следната подредба на честотите 4,3,3,2,1,1,1. След като сме направили честотен анализ можем д а изградим дърво в листата на което да запишем елементите на съобщението. За да направим това по близо до корена на дървото ще записваме по-често срещаните елементи и те ще бъдат с по-къси записи и с по-малък брой дъги преди себе си. Дървото ни трябва да е 2-йчно защото един бит може да има само 2 стойности. Стойностите ще се явяват нашите изходни дъги. Всеки възел ще има максимум две изходни дъги тоест 2 възможности както един бит. За да изградим дървото започваме от най-рядко срещаните елементи.
1 1 1 2 3 3 4 – е нашата последователност записана във възходящ ред.



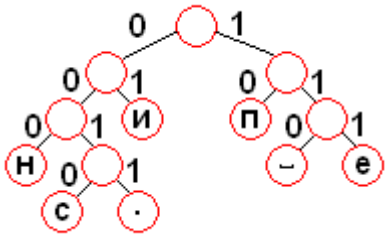
Събираме първите 2 елемента (тоест тези с най малка стойност), след което от получените стойности записваме отново поредица с нарастващи стойности, това се повтаря докато остане само 1 елемент.



За да представяне на дървото ще започнем разпишем получената структура на обратно.

В листата на така полученото дърво ще подредим поставим стойностите който са имали съответните честоти на срещане.

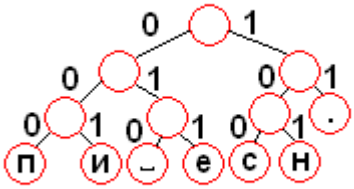
След като сме поставили символите в листата избираме лявата или дясната дъга за 1-ца и съответно другата за нула и чрез обхождане на дъгите водещи до всяко от листата получаваме неговата стойност с която ще го кодираме.



При обхождането получаваме:
н = 000,
с = 0010,
точка = 0011,
и = 01,
п =10,
интервал = 110,
е = 111.

След като знаем с колко елемента може да се кодира всеки символ, правим пресмятане колко ще бъде голямо цялото ни съобщение. Умножаваме броя на елементите за симвула с броя на неговите срещания в съобщението.
н - 3.1=3 | с - 3.1=3 | точка - 3.1=3 | и - 3.2=6 | п - 4.2 = 8 | интервал - 2.3 = 6 |е - 3.3 =9
и сумарно за цялото съобщение получаваме: 3+3+3+6+8+6+9=38.

Ако бяхме записали буквите в нормално разпределено дърво без да се съобразяваме с тяхната честота на срещане щеше да получим:



При което всички символи без точката щяха да се нуждаят от по 3 символа за кодиране а точката щеше да се нуждае от два. При което съобщението щеше да има следната големина:

п – 4.3 =12 | и – 3.3=9 | интервал – 2.3=6 | е -3.3=9 | с – 1.3=3 | н – 1.3=3 | точка – 1.2=2
Сумарно: 12+9+6+9+3+3+2 = 44. Подобрението е 44 – 38 = 6 бита по-малко при предаване на съобщението. 6/44 = 13.6 % подобрение. Всички параметри който използвахме са строго индивидуални и зависят от самото съобщение и подреждането на символите при работа.

В същност когато направим кодиране на едно съобщение ние използваме стойности получени на момента за неговите елементи, за да разкодираме съобщението тези стойности трябва да се добавят към крайното съобщение с което то нараства. Полза от кодирането което ще направим се наблюдава по-ясно в по-големи съобщения.

Опитайте да направите Хафманово кодиране на ваши съобщения.
.....

Някои източници на информация.
http://math.hws.edu/eck/cs225/s03/binary_trees/
http://en.wikipedia.org/wiki/Heapsort
http://cslibrary.stanford.edu/110/BinaryTrees.html
Не се ограничавайте само с тези.