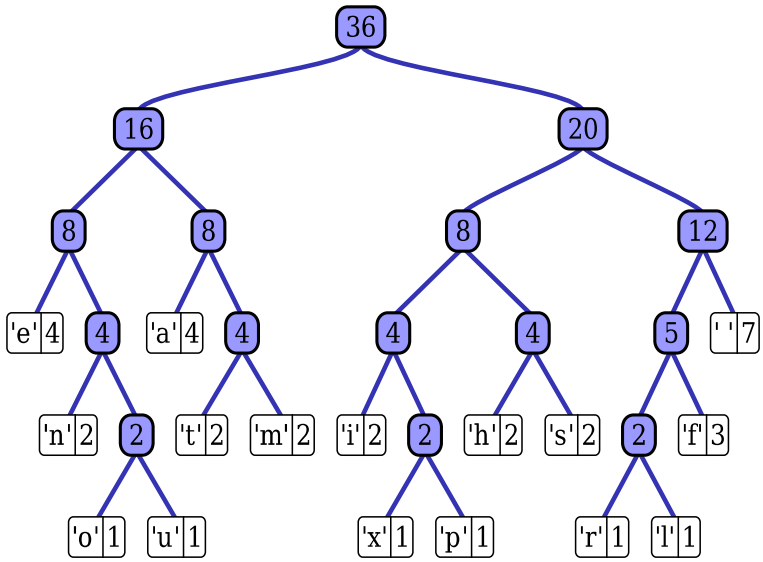
Codificare Huffman

Laboratorul 10

Codificarea Huffman (https://en.wikipedia.org/wiki/Huffman_coding) reprezinta o modalitate eficienta prin care se pot comprima si decomprima secvente de informatie. Algoritmul genereaza un arbore de prefixe ce pot fi folosite pentru a codifica optim informatia.



Arbore Huffman pentru "this is an example of a huffman tree"

Intr-un arbore Huffman, nodurile contin doua elemente – caracterul codificat si frecventa de aparitie a acestuia. Caracterul codificat este relevant doar in cazul nodurilor frunza, acesta fiind ignorat in nodurile interne.

Pentru a construi un arbore Huffman, se executa urmatorii pasi (mai multe detalii in pagina de pe Wikipedia):

- se calculeaza frecventa de aparitie a fiecarui caracter din text
- se creeaza cate un nod frunza pentru fiecare caracter (valoare = caracter, frecventa = frecventa de aparitie calculata anterior)
- se introduc toate nodurile intr-un minheap
- cat timp mai sunt minim doua noduri in minheap:
  - se extrag primele doua noduri (removeMin)
  - se creeaza un nod nou ce le are ca fii (nod intern → valoare = orice, frecventa = suma frecventelor fiilor)
  - se insereaza noul nod in heap
- ultimul nod ramas in minheap este radacina arborelui Huffman

Avand arborele, el este interpretat astfel: coborarea “la stanga” este “0”, coborarea “la dreapta” este “1”. Astfel, pe exemplul de mai sus, 'o' se codifica '00110'. Avand aceasta informatie, orice text poate fi codificat si decodificat. Din nou, pentru mai multe detalii consultati pagina de Wikipedia.

Aveti de implementat urmatoarele functii:

void computeFreqs(char *text, int size, int freqs[256]): numara aparitiile fiecarui caracter din **text** in **freqs**

HuffmanNode *makeTree(int freqs[256], int *size): creeaza arborele Huffman pentru vectorul de frecvente dat ca parametru; arborele este reprezentat ca un vector de noduri. **Size** este setat la numarul de noduri din vectorul intors
- un nod contine caracterul pe care il reprezinta (**value**) si indicii din vectorul anterior mentionat ai fiilor stanga si dreapta (**left** si **right**)
- recomandare: introduceti frunzele la inceput in vectorul de noduri, iar apoi pe masura ce creati noi noduri (algoritmul Huffman), adaugati-le la sfarsitul vectorului; astfel, radacina arborelui va fi pe pozitia **size – 1**
- va veti folosi de heap-ul creat in laboratorul precedent (este inclus un **.h** deja, cu diferenta ca de data aceasta este min heap) pentru a implementa algoritmul Huffman; astfel, puteti folosi campul **prior** pentru a salva frecventa de aparitie si campul **content** pentru a salva indexul nodului in vectorul ce reprezinta arborele Huffman

void makeCodes(int index, HuffmanNode *nodes, char **codes, char *code): functie **recursiva** ce introduce codurile corespunzatoare caracterelor din frunze in **codes**; caracterul prelucrat la fiecare pas este **nodes[index]**
- recomandare: folositi **code** pentru a construi pe parcurs codul atunci cand coborati prin arbore (de exemplu, cand coborati pe fiul stang, concatenati (hint: **strcat**) “0” la **code**; nu uitati sa anulati aceasta schimbare la revenirea din recursivitate

char *compress(char *text, int textSize, HuffmanNode *nodes, int numNodes): functie ce se foloseste de **makeCodes** pentru a codifica **text**

char *decompress(char *text, int textSize, HuffmanNode *nodes, int numNodes): functie ce se foloseste de arborele Huffman (**nodes**) pentru a decodifica **text**