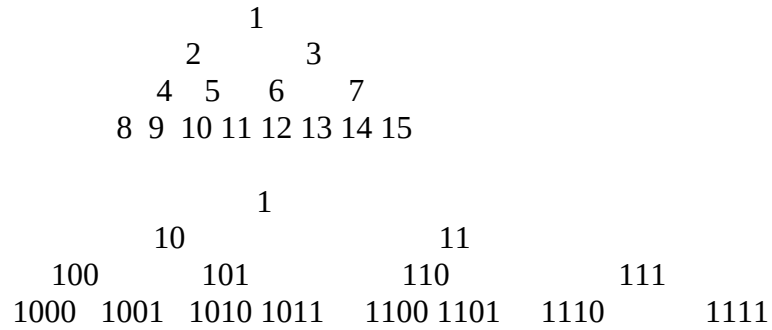
Heap-uri. Compresie Huffman.

Marti 16-18

Heapurile sunt arbori binari cu proprietatea ca fiecare nod are o valoare asociata mai mare (sau mai mica, daca avem minHeap) decat toate valorile din subarborele asociat. Este o structura de date utila pentru a mentine o coada de prioritati – la fiecare pas in radacina se va afla cea mai mare (cea mai mica) valoare din coada, insertia si stergerea radacinii fiind in complexitate logaritmica.
Heapurile au doua operatii importante:
- adaugarea unui nod:

Heapul este un arbore binar echilibrat, nodurile se adauga pe rand, pentru a completa nivelul curent. Se trece la urmatorul nivel doar cand arborele este complet (are 2^(level)-1 noduri). Ordinea in care se adauga se poate intelege din desenul urmator:



La fiecare pas se adauga un nod pe urmatoarea pozitie valida, si cat timp valoarea nodului este mai mare decat valoarea parintelui, se face swap cu parintele.
Din acest motiv, in general implementarea unui heap se face cu vectori – pentru un nod adaugat la pozitia i, parintele se afla la [i/2] iar copiii la 2*i, respectiv 2*i+1. Pentru laboratorul de azi, vom folosi implementare fara vectori – se va retine numarul total de noduri, de exemplu n=6. Se tine reprezentarea acestui numar in baza 2. Daca numarul total de noduri devine 6, in urma unei noi insertii, 6 in baza 2 este 110. Ignoram prima cifra (1), cele 2 cifre, 1 si 0, ne vor spune directiile in care vom merge in arbore pentru a insera noul nod: pornim de la radacina, mergem la dreapta, apoi mergem la stanga.
- stergerea unui nod:

Se poate sterge orice nod, nu doar nodul radacina, dar in practica ne intereseaza sa scoatem radacina. Pentru a scoate un nod, i se atribuie o valoare foarte mare, se face swap cu parintii pana nodul ajunge radacina (ca la insert, dar valoarea fiind foarte mare e sigur ca nodul va deveni radacina), apoi se scoate radacina. Pentru a scoate radacina, se muta valoarea ultimului nod existent in radacina, apoi se face swap cu copilul cu cea mai mare valoare, cat timp valoarea din radacina este mai mica decat valoarea unuia dintre copii.

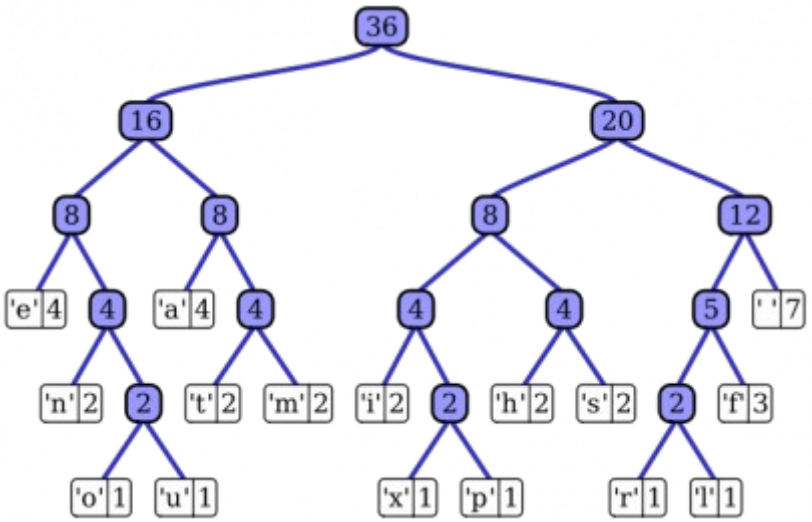
Avand structura:

```
   typedef struct Heap {
        int no_nodes;
        struct node *root;
} Heap;
   typedef struct node {
        int val;
        struct node *left, *right;
} node;
```

Sa se scrie functiile insert(Heap *h, int val) si remove_root(Heap *h). Pentru testare, se introduc noduri random in heap, apoi se scot pe rand si trebuie sa fie afisate in ordine descrescatoare.

Aplicatie – Compresie Huffman.

Compresia Huffman atribuie unei litere un cod care poate fi mai mic de 8 biti (cat are un char), in functie de cat de des e folosita litera in textul respectiv.
Daca stim ce caractere exista in text si de cate ori apar, pentru fiecare caracter se creeaza o frunza in arborele Huffman asociat, avand ca valoare numarul de aparitii ale literei in text. La fiecare pas, se cauta cele mai mici 2 noduri si se unesc intr-un parinte avand ca valoare suma copiilor. La final, se va obtine o radacina care contine toate nodurile. Exemplu grafic:



sora

10110011011000010

Pentru a extrage cele mai mici 2 noduri existente, se poate tine un vector cu toate nodurile curente (care n-au fost inca unite intr-un parinte) si se cauta secvential, sau se poate folosi un heap in care tinem noduri din arborele Huffman. La inceput se aloca memorie pentru frunze, se baga toate frunzele in heap, apoi la fiecare pas se extrag cele mai mici 2 noduri din heap, se creeaza un nod nou care are ca si copii stanga si dreapta aceste 2 noduri extrase din heap, apoi se adauga in heap nodul nou creat. In momentul in care a ramas un singur nod in heap, acesta este radacina arborelui Huffman.
Pentru a vedea care este noul cod pentru un caracter, se parcurge arborele, se retine ‘0’ daca am mers la stanga, respectiv ‘1’ daca am mers la dreapta, si cand am ajuns intr-o frunza, valoarea asociata caracterului este valoarea codului parcurs. In figura de mai sus, pentru ‘e’ aveam 000, pentru ‘a’ avem 010, iar pentru ‘x’ avem 10010. Insumand valoarea nodurilor interne (adica execeptand frunzele), vom obtine noua lungime a textului codificat.
Heapurile trebuie sa retina acum noduri din arborele Huffman, deci structurile vor fi:

```
   typedef struct node {
        struct HuffNode * node;
        struct node *left, *right;
} node;
```

```
typedef struct HuffNode {
        char ch; // caracterul, daca nodul e frunza, altfel ch = 0
        int val; // frecventa
        struct HuffNode *left, *right;
} HuffNode;
```

Se citeste din fisierul “huff.in” un numar N, reprezentand numarul de caractere din text, apoi N perechi caracter, frecventa. Sa se construiasca arborele huffman si sa se afiseze la stdout fiecare caracter si codul asociat in arbore.

Resurse:
https://www.geeksforgeeks.org/binary-heap/
https://www.infoarena.ro/problema/huffman
https://infoarena.ro/problema/heapuri