Laborator Algoritmi și Structuri de Date
Tema 10

Tema săptămânii 10.

1 Arbori binari

(+2p) 1. Pentru un arbore binar, scrieți o funcție care afișează toate nodurile de pe nivelul k din arbore, numerotat de la rădăcină, și în care rădăcina este pe nivelul 0.

În această notație, nivelul k înseamnă toate nodurile aflate la distanța de k noduri de la rădăcină (trebuie parcurse k noduri interne ale arborelui pentru a ajunge la nodul respectiv).

Sugestie: modificați o parcurgere recursivă și mai adăugați un parametru *nivel* care crește cu +1 la fiecare apel recursiv.

2 Liste & Cozi

(3p) 2. Creați o coadă (double-ended queue = dequeue) în care să se poată *insera* și *extrage* de la ambele capete în timp constant. Folosiți o implementare de listă dublu înlănțuită de tipul

```
struct nod{int val;nod* prev;nod* next;} *stanga,*dreapta;
```

(2p) 3. Implementați o coadă de priorități (priority queue) folosind un heap.

(Structura aceasta se mai folosește și la construcția arborelui Huffman)

Coada de priorități este formată din elemente de tipul $(valoare, prioritate)$. Ea trebuie să suporte *extragerea* elementului de prioritate cea mai importantă (unul dintre ele, pentru că pot fi mai multe are au aceeași prioritate) și *inserția* unui element după prioritatea acestuia.

Într-o implementare bazată pe cozi (queue) vom obține timp constant pentru extragerea elementului de prioritate cea mai importantă (va fi primul în coadă) și timp $O(n)$ ca să inserăm un element în coadă (pentru că e ca la inserarea sortată, trebuie inserat sortat după prioritate, deci căutată poziția sa).

Când folosim un heap ambele operații vor fi în timp $O(\log n)$. Implementați un min-heap sau max-heap după interpretarea dvs. a priorității. De exemplu la unele sisteme de operare se folosesc întregi pozitivi, unde cu cât este mai mic întregul respectiv asociat unui proces, cu atât acesta este mai important.