Algoritmi şi programare - Tema 4

Termen de realizare: săptămâna 14-18 ianuarie 2013

Observaţii:

- Toate probleme sunt obligatorii.
- Vor primi punctaj maxim soluţiile optimale din punct de vedere al complexităţii.

Obiective:

- familiarizarea cu tipurile abstracte de date: arbori binari.
- reprezentarea grafurilor şi digrafurilor şi operaţii asupra acestora.

Probleme propuse

1. Să se creeze un fişier antet `arbore_binar.h` şi un fişier sursă `arbore_binar.c`. Fişierul `arbore_binar.h` conţine definiţia tipului de date `NodArboreBinar` folosind pointeri şi prototipuri pentru următoarele funcţii:

    - Crearea arborelui binar asociat unei secvenţe de caractere. Secvenţa este constituită din litere mari şi caracterele `( , ) $`. Semnificaţia acestora este următoarea:

        - literele reprezintă informaţia dintr-un nod;
        - `(` defineşte un nivel inferior;
        - `,` separă sub-arborele stâng de cel drept;
        - `)` marchează stârşitul nivelului inferior;
        - `$` marchează lipsa sub-arborelui respectiv.

    - Explorarea arborelui în adâncime (DFS), prin parcurgerea:

        - în preordine (RSD);
        - în inordine (SRD);
        - in postordine (SDR).

    - Explorarea arborelui în lățime (BFS).

    Fişierul sursă `arbore_binar.c` conţine implementări ale funcţiilor din `arbore_binar.h`. Creaţi un fişier sursă `demo_arbori.c` care conţine funcţia `main` şi apelează funcţiile din `arbore_binar.h`.

    Exemplu
    Pentru arborele corespunzător secvenței de caractere `A(B,C(D(E,F),$))`, explorările în adâncime vor duce la vizitatea, în ordine, a următoarelor noduri:

    - în preordine (RSD): `A B C D E F`
    - în inordine (SRD): `B A E D F C`
    - în postordine (SDR): `B E F D C A`

    Explorarea în lățime va duce la vizitarea nodurilor:

    `A B C D E F.`

2. Să se adauge în fişierul `arbore_binar.h` prototipuri de funcţii ale căror implementare se va găsi în fișierul `arbore_binar.c` și care realizează următoarele acţiuni:

    - obţinerea versiunii în oglindă a arborelui;
    - afişarea nodurilor de pe un nivel dat din arbore;
    - numărarea nodurilor din arbore;
    - numărarea nodurilor din arbore cu un singur descendent;
    - afişarea celui de-al `k`-lea element din traversarea inordine a arborelui.

3. Să se creeze un fişier antet `digraf.h` care conţine definiţia unui tip de date pentru reprezentarea unui digraf şi următoarele prototipuri de funcţii:

```
// citeste datele dintr-un fisier al carui nume este dat ca parametru
void creeaza(Digraf *, char *);

// pentru un nod dat, calculeaza gradul interior
int grad_int(Digraf, int);

// pentru un nod dat, calculeaza gradul exterior
int grad_ext(Digraf, int);

// determina daca digraful dat la intrare este tare conex
int este_tare_conex(Digraf);
```

    Să se creeze un fişier sursă `digraf.c` care conţine implementările pentru funcţiile de mai sus. Creaţi un fişier sursă `demo_digraf.c` care conţine funcţia `main` şi apelează funcţiile din antet.

4. Adăugaţi în programul de la problema 3 funcţii care realizează următoarele operaţii:
    - Verifică dacă un digraf dat conţine un vârf "groapă" (`i` este "groapă" dacă pentru orice alt vârf `j` ≠ `i` există un arc (`j`, `i`) şi nu există arc de forma (`i`, `j`)). Dacă există un astfel de vârf, atunci funcţia va returna vârful găsit. Dacă nu, va returna `-1`. Utilizaţi funcţiile `grad_int` şi `grad_ext` definite în problema 3.
    - Pentru un digraf `D` şi două noduri date `x` şi `y`, returnează lungimea celui mai scurt drum între acestea în `D`, dacă există; dacă nu, returnează `LONG_MAX`.
    - Pentru un digraf `D`, returnează `D` transpus, și anume graful obţinut prin inversarea sensurilor arcelor lui `D`.
    - Afişează componentele tare conexe ale unui digraf `D`.

5. Se consideră un graf `G`. Scrieți un program care verifică în timp `O(N+M)` dacă graful `G` este arbore. Aici `N` şi `M` reprezintă, respectiv, numărul de vârfuri şi numărul de muchii din graful `G`.