Căutare în grafuri. Compilare separată

11 mai 2004

Parcurgerea grafurilor

În găsirea unei căi din cursul anterior, se continua imediat căutarea din ultimul nod atins, adică parcurgerea cât mai "în adâncime" a grafului.
Există însă şi alte strategii de parcurgere.
O procedură generală de parcurgere a tuturor nodurilor:

procedura parcurge
  n = nod iniţial
  marchează n ca atins
  introdu n în lista de aşteptare
  cât timp lista de aşteptare nu e vidă
    scoate un nod n din listă
    vizitează n /* fă prelucrarea necesară */
    pentru fiecare succesor s al lui n
      dacă s n-a fost atins
        marchează s ca atins
        introdu s în lista de aşteptare

Parcurgerea grafurilor: discuţie

Marcarea unui nod e necesară pentru evitarea *ciclurilor infinite*.
– e necesar un câmp corespunzător în structura de date a nodului (sau un tablou auxiliar de fanioane, indexat după noduri)
– pe parcursul algoritmului, un nod se află succesiv în *trei faze*:
  – nemarcat: nu a fost încă decoperit (atins) de algoritm
  – marcat, în lista de aşteptare: descoperit dar neprelucrat
  – marcat, ieşit din lista de aşteptare: prelucrare încheiată

Câmpul "marcat" poate avea valori în funcţie de scopul parcurgerii (ex. culoarea nodului, predecesorul în parcurgere, etc.)

Într-un graf neconectat, se repetă pentru fiecare componentă conexă.
Obs: pentru mai multe prelucrări repetate ale grafului, el trebuie din nou "demarcat" (acelaşi algoritm, cu inversarea sensului marcării)

Căutarea în adâncime şi prin cuprindere

Căutarea în adâncime
– recursiv, prin continuarea parcurgerii din *ultimul* nod atins
– nerecursiv, dacă lista de aşteptare e o *stivă*
  ⇒ următorul nod scos pentru prelucrare e ultimul nod atins

Căutarea prin cuprindere
– dacă lista de aşteptare e o *coadă*
  ⇒ elementele sunt scoase în ordinea în care au fost introduse
  ⇒ parcurgere în ordinea distanţei (nr. muchii) de nodul iniţial (ex. pt. drumul cel mai scurt ca nr. de muchii traversate)

Pentru arbori:
Căutarea în adâncime corespunde traversării în preordine
Căutarea prin cuprindere corespunde traversării pe nivele.
– nu e necesară marcarea nodurilor, arborii sunt grafuri *aciclice*.

Programe compuse din mai multe fişiere

Implicit, obiectele declarate la nivel de fişier sunt *unice* într-un program (două declaraţii ale aceluiaşi identificator în fişiere diferite reprezintă *acelaşi obiect*, v. curs 4).
⇒ obiectul va fi *definit* într-un singur fişier, *declarat* în toate fişierele ce-l utilizează. Declaraţii care nu sunt definiţii:
– pentru variabile: cu specificatorul `extern`
– pentru funcţii, doar prototipul (antetul), nu şi corpul funcţiei

Fazele compilării:
– compilarea în fişiere *obiect* `.c -> .o`
(cod maşină, dar conţine încă nume de variabile în loc de adrese fixe)
– editarea de legături (linkeditarea): referinţele la un identificator (*simbol*) din toate fişierele obiect înlocuite prin aceeaşi adresă

Obiectele cu specificatorul `static` nu sunt vizibile în afara fişierului
⇒ acelaşi identificator poate fi refolosit pentru obiecte diferite

Structurarea programelor din mai multe fişiere

– câte un fişier pentru porţiunile de cod care formează o entitate logică
– cu un minim de interacţiune (fără variabile globale nenecesare, etc.)
– declaraţiile de tipuri, funcţii şi variabile ce trebuie exportate se pun într-un fişier antet `.h`
– acesta e inclus de fiecare fişier `.c` care îl necesită
– pentru a nu include/declara în duplicat, se poate încadra în

```
#ifndef __FISIERULMEU_H
#define __FISIERULMEU_H
/* aici vine continutul propriu-zis */
#endif
```

Tipuri de date abstracte

TDA = un model matematic cu un set de operații asupra lui
⇒ o structură de date + funcții care operează pe ea
⇒ noțiunea de *clasă* din programarea orientată pe obiecte

Pentru implementarea TDA în C:
– tipul de date (ex. structura) e ascuns în partea de implementare
– în fișierul `.h` se declară doar un `typedef` pt. tip (sau pointer)

```
typedef struct node {      /* în fişierul .c cu implementarea */
  int info;                /* sau/şi alte câmpuri */
  struct node *nxt;
} node_t;
typedef struct node *list; /* în fişierul .h */
```

– utilizatorul, care include doar fișierul `.h` nu are acces la structura internă a tipului; accesul e permis doar prin funcții care citesc/modifică componentele unei variabile de acest tip (ca și pt. `FILE`)

Tipuri de date abstracte (cont.)

Decizii de proiectare:
– ce operații că fie incluse
– dacă se transmit obiecte sau doar pointeri la obiecte
(pointerii sunt necesari pentru funcții care modifică obiectul)
– dacă rezultatul unei operații e returnat (eventual alocat dinamic), sau depus într-un obiect specificat (deja alocat) transmis ca parametru
– dacă funcția returnează un obiect, sau un cod de succes/eroare (și obiectul e depus la adresa dată de un pointer parametru)

Vezi exemple de cod pentru:
– cozi și stive implementate cu tablouri sau liste
– mulțimi (de întregi), memorate cu câte un bit pe element