Tabele de dispersie. Programare modulară

25 mai 2004

Problema: regăsirea eficientă a unui obiect

În proiectarea unui program, structurile de date trebuie să permită regăsirea *rapidă* a obiectelor utilizate, pentru prelucrare *eficientă*.
– în *tablou*: *acces direct* la elemente dacă ştim indicele
– sau: structuri cu elemente legate prin pointeri (înlănţuite)
(ex. în grafuri, legături între nodurile şi muchiile corespunzătoare)

Problema:
Accesul la obiecte referite din exteriorul programului prin *nume*
(identificator nenumeric ⇒ nu poate fi folosit direct pentru indexare)

Ex.: graf cu oraşe; utilizatorul introduce un nume (nu număr) de nod
Ex.: la compilare, regăsirea înregistrării unei variabile (la întâlnirea ei)

Metode de regăsire

– *căutare secvenţială* (în tablou sau listă)
durează proporţional cu numărul de elemente
– *căutare binară* (într-un tablou)
durează logaritmic, dar structura trebuie menţinută sortată
(pe ansamblu, efort similar dacă sunt multe inserări / ştergeri)
– căutare în *arbore binar* (tot logaritmic)
dar arborele trebuie menţinut relativ echilibrat - structură complicată

Ideal ar fi regăsirea (accesul) în timp practic constant.
– ideea: găsirea unei funcţii `h` cu o *valoare numerică unică* pentru fiecare obiect considerat, într-un domeniu restrâns (utilizabil ca indice)
⇒ memorăm fiecare obiect `x` într-un tablou la indicele `h(x)`

Tehnica se numeşte *dispersie* (*hashing*):
obiectele cu care lucrăm sunt dispersate într-un tablou (*hash table*).

Funcţii de dispersie (hash functions)

– matematic: o funcţie (parţială) $h : D \to V$, unde D e domeniul tuturor obiectelor posibile, iar domeniul de valori (indici) V e $0, 1, \ldots, N-1$.
– ex. la compilare: D e mulţimea tuturor identificatorilor
– practic, $|D| >> |V|$ (ca dimensiune) deci h nu poate fi injectivă pe D
– dar avem nevoie de valori distincte doar pt. submulţimea obiectelor efectiv utilizate $D_u \subseteq D$ (ex. identificatorii dintr-un program C dat)

Proprietăţi dorite ale funcţiilor de dispersie
– să fie rapid calculabile (pentru eficienţă)
– să aibă o distribuţie de valori cât mai uniformă, pentru a minimiza probabilitatea de *coliziune* (valori egale pentru obiecte diferite

Exemple de funcţii de dispersie

Cel mai frecvent caz: funcţii pentru şiruri de caractere
– se calculează cu (aproape) toate caracterele (deosebire cât mai bună)
– cu deplasări frecvente pe biţi pentru a "amesteca" valoarea obţinută
Exemple concrete (şirul `char *s;` se parcurge secvenţial):

```
for (h=len; len--;) h = ((h<<7) ^ (h<<27)) ^ *s++;     /* Knuth */
for (h=5381; c=*s++; ) h += (h << 5) + c;          /* Bernstein */
for (h=0; c=*s++; ) h = (h<<6) + (h<<16) - h + c;       /* SDBM */
```

La sfârşit, valoarea e luată modulo dimensiunea tabloului: `h %= N`

Pentru alte tipuri de obiecte: se pot face calcule cu octeţii obiectului grupaţi câte 4 (sau 2) şi interpretaţi ca întregi

Şi funcţiile bune au *coliziuni* (valori egale pt. obiecte diferite)
⇒ trebuie rezolvate (dezambiguate) pentru a permite regăsirea corectă

Tabele de dispersie deschise şi închise

Tabele de dispersie închise (closed hashing)
– dacă la indicele `idx=h(x)` se găseşte alt obiect `y`, se caută succesiv după o anumită regulă: *secvenţial* (`idx++`), *liniar* (`idx+=i`), cu a doua funcţie (`idx+=h2(x)`), până se găseşte obiectul sau o intrare vidă
– nu pot conţine mai multe obiecte decât dimensiunea tabloului
⇒ la depăşire, obiectele trebuie redistribuite într-un tablou mai mare
– la ştergere, intrarea în tablou trebuie marcată "şters", nu "vid", pentru a permite căutarea corectă (până la găsire sau "vid")

Tabele de dispersie deschise (open hashing)
– o intrare în tablou: *listă* de obiecte cu aceeaşi valoare pentru `h`
⇒ hashing + căutare liniară în listă (scurtă pentru funcţii bune)
– necesită alocare dinamică pentru elementele listei (v. exemplu)
– tabloul poate fi mai mic decât nr. de obiecte, dar se recomandă să fie comparabil (pt. a avea nr. mic de obiecte cu aceleaşi valori pt. `h`)