A keresőkkel és adatbázissal ellátott lengyel honlap számos díjat kapott: *Spirit of Delphi '98, Delphi Community Award*, *Poland on the Internet*, *Golden Bagel Award* stb.

Az itt megtalálható komponenseket nemcsak használni lehet, hanem fel is lehet tenni a saját, nyilvános használatra szánt – jól működő – komponenseinket.

Jó böngészést!

firkácska

Érdekes informatika feladatok

XVI. rész

Tömbök tárolása (2.)

Szabályos szerkezetű speciális mátrixok tárolása

A cikksorozat ezen részében a szabályos szerkezetű speciális mátrixok tárolási módszereit mutatjuk be. A specialitás abban rejlik, hogy a mátrixok csak hézagosan vannak kitöltve, az elemek nagyrésze zérós. A szabályos szerkezet pedig abban rejlik, hogy a zérós elemek szabályosan helyezkednek el a mátrixban.

Ilyen mátrixok a:

- ritka mátrixok
- háromszögmátrixok
- szimmetrikus mátrixok
- szalagmátrixok, sávmátrixok

Ritka mátrixok

Azokat a mátrixokat, amelyeknek legtöbb eleme 0-val, vagy valamilyen más, előre ismert elemmel egyenlő, *ritka mátrixoknak* nevezzük.

Egy $n \times m$-es mátrix memóriaigénye, amelynek minden egyes eleme h byte memóriát igényel: $n \times m \times h$.

Ha ténylegesen csak k elem hordoz értékes információt, akkor a memóriaigény $k \times h$.

Ritka mátrixok például a *permutációs mátrixok*. A P permutációs mátrix minden sorában és minden oszlopában pontosan egy 1-es áll, a többi elem 0. Az elnevezés onnan származik, hogy egy vektort ilyen mátrixszal szorozva a vektor elemeinek egy permutációját kapjuk.

Példa permutációs mátrixra:

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Ritka mátrixok esetén érdemes a mátrixot egy rekord szerkezetű vektorban tárolni, pl. az alábbi módon:

```
type
   TRitkaMatrix = array[1..k] of record
                                   x, y: integer;
                                   elem: real;
                                  end;
```

Vagy ha a legtöbb elem nem 0, hanem egy más érték, akkor:

```
type
   TRitkaMatrix = record
                     legtobb: real;
                     tobbi: array[1..k] of record
                                             x, y: integer;
                                             elem: real;
                                           end;
                  end;
```

Az *x* és az *y* a mátrixelem koordinátáit (oszlop, sor) jelenti.

A módszer hátrányai:

- A mátrix valamely *x*, *y* elemének elérése e pillanattól kezdve egy algoritmuson keresztül történik, amely bináris vagy lineáris keresés segítségével „megnézi" a tárolt vektorban, hogy a keresett elem éppen megvan-e, s ha igen, mennyi az értéke. Mindkét keresésre jellemző, hogy nem minden elem esetén egyforma a keresés ideje, vagyis bizonyos elemeket hamarabb, másokat később ér el a mátrixon belül. Tehát ezen adatszerkezet elérése már nem véletlen idejű. Az elérés már nem közvetlen, mivel szükség van erre a keresésre. Ha annak a valószínűségét, hogy egy elem értéke ne legyen 0, p_k-val jelöljük, akkor a kereséskori tesztelések száma: $p_k(n \times m + 1) / 2$.
- Ezzel a módszerrel maximum *k* mátrixbeli elem tárolása lehetséges. Ezt úgy küszöbölhetjük ki, hogy áttérünk az első részben már bemutatott dinamikus tárolási módszerek valamelyikére.
- Egy mátrixbeli elem tárolása már nem *h* byte-ba kerül, hanem *h'* byte-ba, ahol *h'* = h + 2 * (egész típus tárigénye). Tehát a $n \times m \times h$ helyett $k \times h'$. Ezt a módszert használni tehát akkor éri meg, ha a $k \times h'$ jóval kisebb mint $n \times m \times h$.

A következő példaprogram feltölt egy ritka mátrixot, majd visszatéríti egy ritka mátrix elemét:

```
const
  k = 5;

type
  TRitkaMatrix = record
                    legtobb: real;
                    tobbi: array[1..k] of record
                                            x, y: integer;
                                            elem: real;
                                           end;
                 end;
```

```
function Ritka(m: TRitkaMatrix; i, j: integer): real;
var sz: integer;
begin
  Ritka := m.legtobb;
  for sz := 1 to k do
   if (m.tobbi[sz].x = j) and (m.tobbi[sz].y = i) then
    begin
     Ritka := m.tobbi[sz].elem;
     break;
    end;
end;

var
  m: TRitkaMatrix;
  mr: array[1..10, 1..10] of real;
  i, j, sz: integer;

begin
   {A ritka matrix feltoltese, a legtobb elem 3-as}
   for i := 1 to 10 do
    for j := 1 to 10 do
     mr[i, j] := 3;
   mr[1, 5] := 2;
   mr[3, 7] := 5.8;
   mr[6, 9] := 1.23;
   mr[8, 8] := 5;
   mr[9, 2] := 6.89;
   for i := 1 to 10 do
    begin
     for j := 1 to 10 do
      write(mr[i, j]:6:2);
     writeln;
    end;

   {A ritka matrix abrazolasa rekorddal}
   m.legtobb := 3;
   sz := 0;
   for i := 1 to 10 do
    for j := 1 to 10 do
     if mr[i, j] <> 3 then
      begin
        inc(sz);
        m.tobbi[sz].x := j;
        m.tobbi[sz].y := i;
        m.tobbi[sz].elem := mr[i, j];
      end;

   writeln(Ritka(m, 1, 2):6:2);
   writeln(Ritka(m, 6, 9):6:2);
   writeln(Ritka(m, 8, 6):6:2);
   writeln(Ritka(m, 9, 2):6:2);
end.
```

Háromszögmátrixok tárolása

A háromszögmátrix egy olyan négyzetes mátrix ($n \times n$), amelyben a főátló fölötti elemek 0-val egyenlők (a[i, j] =0 , j >i).

$$\begin{bmatrix} a_{11} & 0 & \dots & 0 \\ a_{21} & a_{22} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

A nullától különböző elemek száma: $n \times (n + 1) / 2$ – első sorban 1 elem, második sorban 2 elem, …, *n.* sorban *n* elem.

Természetesen a háromszögmátrixok tárolásánál helyet spórolhatunk meg, ha a klasszikustól eltérő ábrázolásmódot használunk. Így ilyen tárolási módszer:

Sor	*Elem*	*Cím*
1	a_{11}	$b + 0 \times h$
2	a_{21}	$b + 1 \times h$
2	a_{22}	$b + 2 \times h$
3	a_{31}	$b + 3 \times h$
3	a_{32}	$b + 4 \times h$
3	a_{33}	$b + 5 \times h$
4	a_{41}	$b + 6 \times h$
...	...	...
n	a_{nn}	$b + (n + 2) \times (n - 1) / 2 \times h$

A *b* az első elem címét jelöli, a *h* pedig egy elem hosszát (a típusának megfelelő byte-ok számát).

Ez a tárolás $h \times n \times (n - 1) / 2$ byte-ot mentesít azáltal, hogy a 0-val egyenlő elemeket nem tárolja.

A háromszög mátrixok optimális kezelése érdekében egy akkora vektort kell deklarálni, amely mérete lehetővé teszi a mátrix értékes (nem zérós) elemeinek eltárolását.

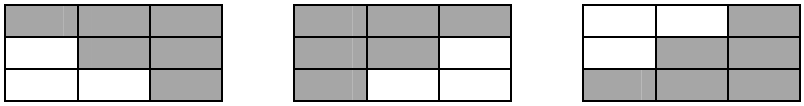
Tehát az `m: array[1..n, 1..n] of integer;`-ből például `v: array [1..n*(n+1)/2] of integer;` lesz. A vektor elemei folytonosan tárolódnak el a memóriában.

Ahhoz, hogy megírhassuk a megfelelő címkiszámoló függvényeket (az *m* mátrix *ij* eleme a *v* tömb mely eleme lesz – és fordítva) vegyük észre a következőket:

- `v(index) = (i-1) * n - ((((i - 1) * i) / 2) + i - 1) + j - 1 + 1;`
- ha egy elem címére vagyunk kíváncsiak: `cím(a[i, j]) = b + h × i × (i – 1)/2 + (j – 1) × h;`

Írjuk meg a visszafelé számoló képleteket is!

Természetesen háromszögmátrix lehet a következő alakok valamelyike is:



Írjuk meg ezekre is az átalakító függvényeket!

Szimmetrikus mátrixok

Egy M mátrix szimmetrikus, ha $M = M^T$, ahol M^T az M mátrix transzponáltja, amelyet a mátrix sorainak és oszlopainak a felcserélésével kapunk meg. Értelemszerűen, ha egy mátrix szimmetrikus, akkor négyzetes is.

Példa szimmetrikus mátrixra:

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 6 & 4 \\ 3 & 4 & 5 \end{bmatrix}$$

Természetesen a szimmetria függvényében elég csak az egyik részt tárolni, a másikat pedig kiszámolni.

Definiáljunk adatszerkezetet szimmetrikus mátrixok tárolására!

Szalagmátrixok, sávmátrixok

Egy M mátrixot akkor nevezünk m sávszélességű szalagmátrixnak, ha létezik $1 \leq m < n$-1 úgy, hogy a főátlótól m-nél „messzebb" lévő elemek mind zérósak, azaz:

$$a_{ij} = 0, \text{ ha } |i - j| > m$$

Ha a sávszélesség 1, akkor *tridiagonális* mátrixról beszélünk. A tridiagonális mátrixok másik neve: *kontinuáns mátrixok*.

A főátló melletti átlókat *alsó* (i = j + 1) és *felső* (i + 1 = j) *mellékátló*nak nevezzük.

Példa szalagmátrixra:

$$\begin{bmatrix} 0 & 0 & 3 & 7 \\ 0 & 2 & 1 & 6 \\ 1 & 2 & 8 & 0 \\ 1 & 4 & 0 & 0 \end{bmatrix}$$

Definiáljunk adatszerkezetet szalagmátrixok tárolására!

Kovács Lehel István