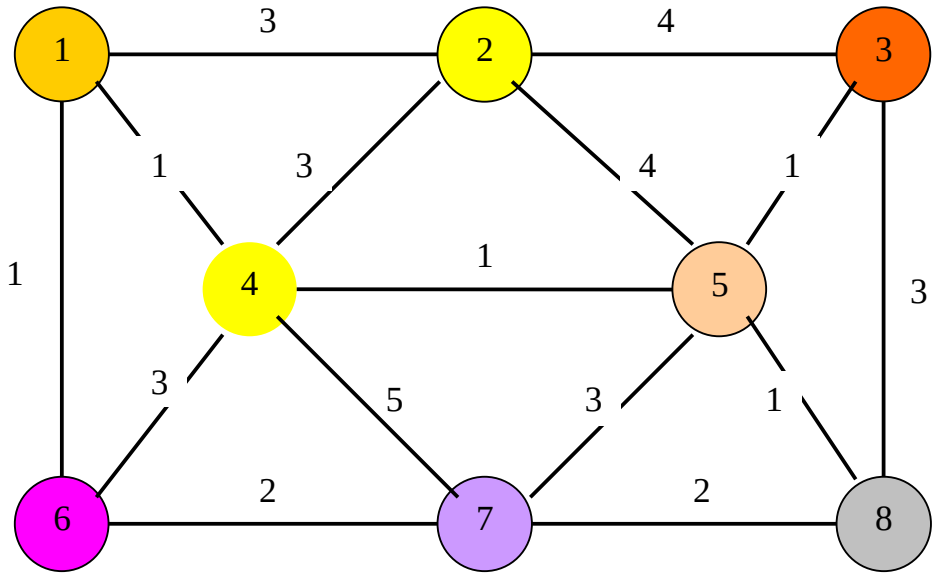
1. Grafuri. Implementarea grafurilor

Definiție:

Un *graf* este o structură formată din noduri şi arce. Dacă arcele sunt orientate, graful este *orientat*.

Pentru aplicații, prezintă interes *grafurile conexe*, adică cele în care orice nod are cel puțin un arc de conexiune cu alt nod.



Nodurile sunt identificate prin nume (1, 2, .. sau a, b, c, . .) dar pot avea asociate valori. Arcele au, de regulă, anumite valori asociate care pot reprezenta costuri, intensități de trafic, tensiuni electrice etc.

Cea mai simplă implementare a grafurilor în memorie este prin utilizarea tablourilor. Numărul de noduri determină dimensiunea tabloului iar valorile asociate arcelor determină elementele tabloului:

Considerăm graful din figură; tabloul asociat poate fi definit astfel:

$$a_{ij} = \begin{cases} v_{ij} \ \textit{daca exista conexiune directa} \\ \infty \ \textit{daca nu exista conexiune de la i la j} \end{cases}$$

Exemplu: Să considerăm o rețea de n oraşe; acestea sunt legate prin şosele de diferite categorii; costul de transport de la nodul i la nodul j este exprimat prin valoarea asociată arcului $i - j$.

O problemă de optimizare poate fi ***determinarea drumului de cost minim între două oraşe***.

Să construim tabloul asociat grafului din figură:

$$A=\begin{pmatrix} 0 & 3 & \infty & 1 & \infty & 1 & \infty & \infty \\ 3 & 0 & 4 & 3 & 4 & \infty & \infty & \infty \\ \infty & 4 & 0 & \infty & 1 & \infty & \infty & 3 \\ 1 & 3 & \infty & 0 & 1 & 3 & 5 & \infty \\ \infty & 4 & 1 & 1 & 0 & \infty & 3 & 1 \\ 1 & \infty & \infty & 3 & \infty & 0 & 2 & \infty \\ \infty & \infty & \infty & 5 & 3 & 2 & 0 & 2 \\ \infty & \infty & 3 & \infty & 1 & \infty & 2 & 0 \end{pmatrix}$$

Pentru determinarea drumului de cost minim între două oraşe se utilizează următoarea observaţie:

Pentru o pereche de noduri ***(i, j),*** *dacă drumul de cost minim de la i la j trece prin k, atunci drumurile* ***(i, k)*** *şi* ***(k, j)*** *sunt minime.*

Vom utiliza această observaţie pentru determinarea unui algoritm şi a unei relaţii de recurenţă pentru rezolvarea problemei.
Pentru aceasta, vom considera pe rând ca noduri intermediare, nodurile *1, 2, 3, . . . ,n,* şi vom calcula de fiecare dată drumurile minime pentru toate perechile ***(i, j)***.

- La pasul 1 considerăm k=1 ca nod intermediar; rezultă A^1 ;
- La pasul 2 considerăm k=2 ca nod intermediar; rezultă A^2 ;
- . . . . . .
- La pasul n considerăm k=n ca nod intermediar; rezultă A^n ;

Relaţia de recurenţă va fi:

$$A^k=\begin{cases} A, & \text{pentru } k=0; \\ \min(a_{ik}^{k-1}+a_{kj}^{k-1}), & \text{pentru } k\neq 0; \end{cases}$$

adică elementele tabloului la pasul k se calculează din cele obţinute la pasul k-1.

Tabloul A se modifică mai mult sau mai puţin la fiecare pas; în final el va conţine valorile minime ale drumurilor dintre oricare două noduri.

```
// Determinarea drumurilor de cost minim intr-un graf
#include<stdio.h>
#include<stdlib.h>
          int n=5;
          int    a[5][5]=   { { 0, 1, 99, 4,   5},
                              { 1, 0,  3, 1,   8},
                              {99, 3,  0, 2, 100},
                              { 4, 1,  2, 0,  12},
                              { 2, 4,  7, 100, 0}};

     void main()
     {
          int i,j,k;
          for(k=0;k<n;k++)
               for(i=0;i<n;i++)
                    for(j=0;j<n;j++)
                    if(a[i][j]>a[i][k]+a[k][j]) a[i][j]=
a[i][k]+a[k][j];

          for(i=0; i<n; i++){
               for(j=0;j<n;j++)
                    printf("%4d ",a[i][j]);
               printf("\n\n");
          }
     }
```

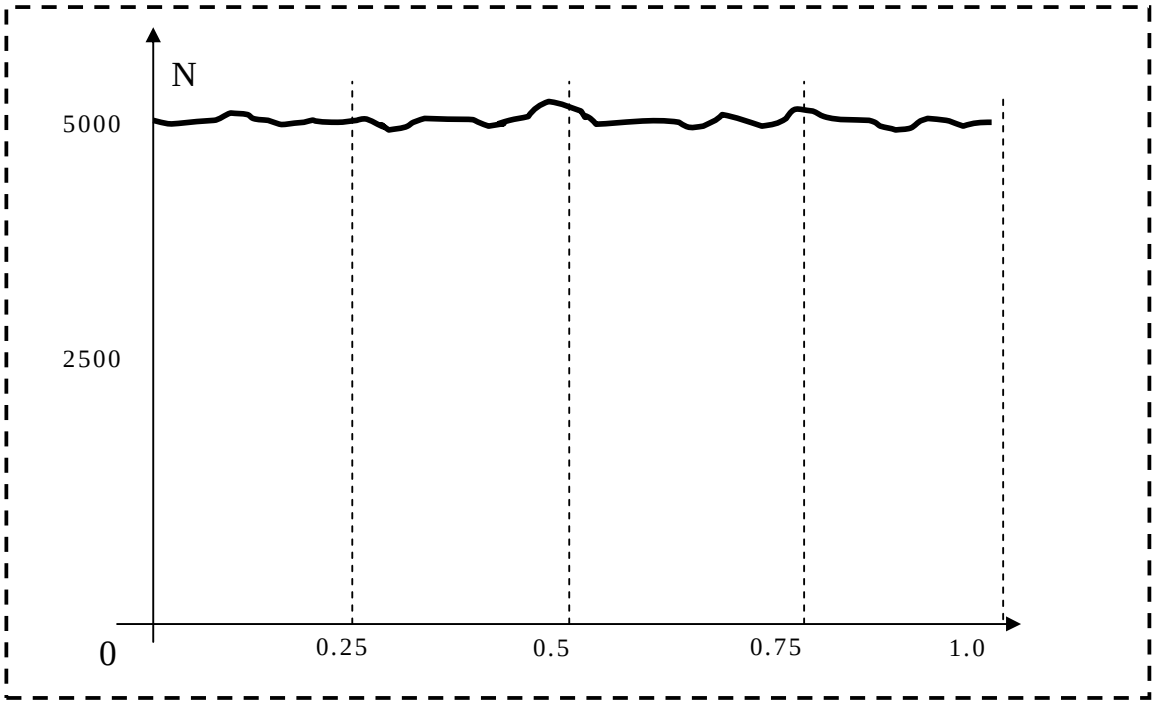
2. Generarea numerelor aleatoare

Atributul ***aleator*** caracterizează relația dintre o anumită valoare şi celelalte valori ale unei *secvențe* caracterizate de o anumită funcție de distribuție. Principala caracteristică a unei secvențe aleatoare este lipsa oricărei *corelații* între valori, ceea ce face imposibilă predicția valorii următoare (la o secvență periodică, valoarea următoare este perfect predictibilă).

Secvențe aleatoare pot rezulta numai din anumite procese fizice naturale, ca de exemplu, dezintegrarea nucleelor dintr-o probă radioactivă. Deoarece nu se poate prevedea care dintre nuclee se va dezintegra şi nici momentul la care se va produce acest fenomen, o mărime cu caracter aleator este intervalul de timp dintre două dezintegrări succesive.

Numeroase aplicații, de modelare a fenomenelor fizice naturale, necesită generarea unor secvențe aleatoare de valori numerice.

Secvențele de numere produse cu ajutorul unui calculator sunt doar *pseudoaleatoare*, în sensul că succesiunea de numere este complet determinată de primul dintre ele, însă gradul redus de corelație secvențială face ca ele să poată substitui cu succes şiruri de numere aleatoare veritabile în numeroase aplicații.



Generarea numerelor aleatoare cu distribuție uniformă reprezintă blocul funcțional fumdamental pentru orice algoritm bazat pe variabile aleatoare. Distribuția este *uniformă* dacă numerele generate acoperă uniform întreg domeniul de definiție.

Cel mai utilizat algoritm pentru realizarea secvențelor aleatoare, implementat de generatoarele standard ale majorității limbajelor de programare evoluate, se bazează pe *metoda congruenței liniare*, care pornind de la *un număr inițial* x_0, generează o secvență aleatoare prin utilizarea unei relații de recurență de forma:

$$x_{i+1} = (ax_i + c) \bmod m, \qquad (1)$$

unde *a, c* şi *m* sunt numere întregi, numite *multiplicator*, *increment* şi respectiv, *modul,* dintre care ultimul este de regulă foarte mare. Evident, utilizând acelaşi set de numere, se obține aceeaşi secvență de numere aleatoare.

Numerele din secvența generată cu relația (1) sunt cuprinse între *0* şi *(m-1)* şi apar într-o succesiune oarecare, indiferent cu care dintre ele începe secvența. Pentru a obține un şir de valori reale din intervalul [0, 1], se pot utiliza numerele x_{i+1}/m.

Utilizarea relației (1) impune două cerințe contradictorii: pe de o parte, modulul *m* trebuie să fie suficient de mare pentru a se genera secvențe cât mai lungi neperiodice, pe de altă parte, modulul *m* şi multiplicatorul *a* trebuie să fie suficient de mici pentru ca produsul lor să nu depăşească limita de reprezentare a numerelor întregi.

Un exemplu de reprezentare a metodei congruenței liniare este funcția Ran, listată mai jos, care returnează numere aleatoare reale, cu distribuție uniformă în intervalul [0, 1].

Defirea şi utilizarea funcției Ran()

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<time.h>
    float Ran(long *i)
    {
       const long a=8121;
       const long c=28411;
       const long m=13446;
       *i=(a*(*i)+c)%m;
       return (float) (*i)/m;
```

```
}

void main()
{
float x; char c;
int p, n;
long *k, m;
clrscr();

printf("Introduceti numarul: n= "); scanf("%d",&n);
for(p=3; p<n; p++)
{
 m=p; k=&m;  x=1/Ran(k);
 printf("%d ", (int)(10*x));

}
getch();
do {
randomize(); printf("\n");
for(p=3; p<n; p++){
printf("%d ",random(100));
}
c=getch(); } while (c=='g');
 }
```

Funcții standard pentru generarea secvențelor aleatoare

1. random <STDLIB.H>

Macro that returns an integer

Declaration:

Macro: **random(num);**

Function: **int random(int num);**

Remarks:

random returns a random number between 0 and (num-1).

random(num) is a macro defined in STDLIB.H.

Return Value: Returns an integer between 0 and (num - 1).

Example:

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

/* prints a random number in the range 0 to 99 */
int main(void)
```

```
{
   randomize();
  printf("Random number in the 0-99 range: %d\n",
random (100));
   return 0;
}
```

---

2. randomize <STDLIB.H>

Macro that initializes random number generator

Declaration:

```
void randomize(void);
```

Remarks:

randomize initializes the random number generator with a random value.

Because randomize is implemented as a macro that calls the time function prototyped in TIME.H, you should include TIME.H when you use this routine.

Return Value: None

See Also:

rand random srand

Example:

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
int main(void)
{
   int i;
   randomize();
   printf("Ten random numbers from 0 to 99\n\n");
   for(i=0; i<10; i++)
    printf("%d\n", rand() % 100);
   return 0;
}
```

---

3. rand <STDLIB.H>

Random number generator

Declaration: **int rand(void);**

Remarks:

rand uses a multiplicative congruential random number generator with period 232 to return successive pseudo-random numbers in the range 0 to RAND_MAX.

Return Value:

rand returns the generated pseudo-random number.

See Also:

random randomize srand

Example:

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
   int i;

   printf("Ten random numbers from 0 to 99\n\n");
   for(i=0; i<10; i++)
      printf("%d\n", rand() % 100);
   return 0;
}
```

---

4. srand <STDLIB.H> Initializes random number generator

Declaration: **void srand(unsigned seed);**

Remarks:
The random number generator is reinitialized by calling srand with an argument value of 1.

The generator can be set to a new starting point by calling rand with a given seed number.

Return Value: None

See Also:
rand random randomize

Example:

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

int main(void)
{
   int i;
   time_t t;
   srand((unsigned) time(&t));
   printf("Ten random numbers from 0 to 99\n\n");
   for(i=0; i<10; i++)
       printf("%d\n", rand() % 100);
   return 0;
}
```