

# Sor, Sorozat adattípusok

Horváth Gyula

[horvath@inf.elte.hu](mailto:horvath@inf.elte.hu)

## 3. A Sor adattípus

### 3.1. Sor

**Értékhalma:**  $Sor = \{\langle a_1, \dots, a_n \rangle : a_i \in E, i = 1, \dots, n, n \geq 0\}$

**Műveletek argumentumai:**  $S : Sor, x : E$

---

Előfeltétel	Művelet	Utófeltétel
$S = \langle a_1, \dots, a_n \rangle$	$SorBa(S, x)$	$S = \langle a_1, \dots, a_n, x \rangle$
$S = \langle a_1, \dots, a_n \rangle \wedge n > 0$	$SorBól(S, x)$	$x = a_1 \wedge S = \langle a_2, \dots, a_n \rangle$
$S = \langle a_1, \dots, a_n \rangle$	$Elemszám(S)$	$= n \wedge S = Pre(S)$
$S = S$	$ÜresE(S)$	$= (Pre(S) = \langle \rangle) \wedge S = Pre(S)$
$S = \langle a_1, \dots, a_n \rangle \wedge n > 0$	$Első(S)$	$= a_1 \wedge S = Pre(S)$
$S = \langle a_1, \dots, a_n \rangle \wedge n > 0$	$Töröl(S)$	$S = \langle a_2, \dots, a_n \rangle$
$S = \langle a_1, \dots, a_n \rangle \wedge n > 0$	$Utolsó(S)$	$= a_n \wedge S = Pre(S)$

---

### 3.2. A Sor C++ és C# műveletei

---

Művelet	C++	C#
Típusdef	<i>queue &lt; E &gt;</i>	<i>Queue &lt; E &gt;</i>
<i>SorBa(S,x)</i>	<i>S.push(x)</i>	<i>S.Enqueue(x)</i>
<i>SorBól(S,x)</i>	<i>x = S.front();S.pop()</i>	<i>x := S.Dequeue()</i>
Elemszám	<i>S.size()</i>	<i>S.Count</i>
ÜresE( <i>S</i> )	<i>S.empty()</i>	<i>S.Count == 0</i>
Első( <i>S</i> )	<i>S.front()</i>	<i>S.Peek()</i>
Töröl( <i>S</i> )	<i>S.pop()</i>	<i>S.Dequeue()</i>
Utolsó( <i>S</i> )	<i>S.back()</i>	<i>nincs</i>

---

### 3.3. Feladat: Várólista

Járóbeteg ellátásra egy napon  $n$  beteg jelent meg. Minden beteg érkezésekor feljegyezték, hogy mikor érkezett, és azt is, hogy hány percig tartott a kezelése. A betegeket érkezési sorrendben veszik kezelésre. A kezelést folyamatosan végezte az orvos, tehát ha a  $p$  perckor kezdte meg egy beteg kezelését ami  $k$  percig tartott, akkor a következő beteg kezelését a  $p + k$  perctől végezhetette. Ki kell számítani, hogy legtöbben hányan vára­koztak aznap egy időben kezelésre várva!

#### Bemenet

A standard bemenet első sora egy egész számot tartalmaz, a betegek  $n$  ( $1 \leq n \leq 1000$ ) számát. A következő  $n$  sor mindegyike két egész számot tartalmaz, egy beteg  $e$  érkezési és  $k$  kezelési idejét ( $0 \leq e \leq 720$ ,  $1 \leq k \leq 40$ ). A bemenetben az adatok érkezési sorrendben vannak. Adott időpontban több beteg is érkezhetett.

#### Kimenet

A standard kimenet a nap során a legtöbb egy időben várakozó beteg számát kell írni! A várakozókba nem számít be az, akinek éppen a kezelését végzik.

## Példa bemenet és kimenet

**bemenet**

5

1 3

2 4

3 2

3 5

4 2

**kimenet**

3

## Korlátok

Időlimit: 0.2

Memória limit: 32 MiB

### 3.4. Megoldás

Jelölje  $B[i] = (erk_i, kezel_i)$  az  $i$ -edik beteg adata. Legen  $v_i$  az az időpont, amikor az  $i$ -edik beteg kezelése befejeződik, tehát ekkor kezdődhet a következő beteg kezelése.  $v_i$  a következőképpen számítható.

$$\begin{aligned} v_1 &= B[1].erk + B[1].kezel_i \\ v_i &= \begin{cases} B[i].erk + B[i].kezel_i, & \text{ha } v_{i-1} < B[i].erk \\ v_{i-1} + B[i].kezel_i, & \text{ha } B[i].erk \leq v_{i-1} \end{cases} \end{aligned}$$

Szimuláljuk a történéseket. Vegyük sorra, hogy adott  $v_i$  időben mi történik. Ehhez a következő adatokat számítjuk:

$$\begin{aligned} i &= \text{a következő érkező beteg sorszáma} \\ v &= \text{ekkor fejeződik be az aktuális beteg kezelése} \\ S &= \text{a várakozók sora (érkezési sorrendben)} \\ varok &= \text{az eddigi várakozók maximális száma} \end{aligned}$$

Szimulációs tevékenység a  $v$  időben:

- ha van várakozó ( $S$  nem üres), akkor vegyük  $S$ -ből az elsőt, egyébként a következő érkezőt
- aktualizáljuk a  $v$  értékét
- tegyük a várakozók  $S$  sorába a  $v$  időig beérkezőket
- aktualizáljuk a  $varok$  értékét

### 3.4.1. C++ megvalósítás

```
1 #include <iostream>
2 #include <queue>
3 #define maxN 10000
4 using namespace std;
5 struct Beteg{
6     int erk,kezeli;
7 };
8 Beteg B[maxN];
```

```

9  int main(){
10     queue<Beteg> S;
11     int n,v=0,varok=0,i=0;
12     cin>>n;
13     for(int i=0;i<n;i++){           //adatok be
14         cin>>B[i].erk>>B[i].kezeli;
15     }
16     Beteg ku;                       //az aktuális beteg
17     while(i<n || !S.empty()){       //amíg van beteg
18         if (S.size()>varok)
19             varok=S.size();
20         if (!S.empty()){             //van várakozó
21             ku=S.front(); S.pop();   //vegyük ez első várakozót
22         } else                       //nincs várakozó
23             ku=B[i++];              //vegyük a következő érkezőt
24         if (v<ku.erk)
25             v=ku.erk+ku.kezeli;     //v-ig tart az aktuális kezelése
26         else
27             v+=ku.kezeli;           //eddig tart az aktuális kezelése
28         while(i<n && B[i].erk<v){    //v-ig beérkezők mennek a sorba
29             S.push(B[i]); i++;
30         }
31     }
32     cout<<varok<<endl;
33     return 0;}

```



### 3.4.2. C# megvalósítás

```
1  using System;
2  using System.Collections.Generic;
3
4  class Program{
5      public class Beteg{
6          public int erk; public int kezeli;
7          public Beteg(int x, int y){erk=x; kezeli=y;}
8      }
9      public static void Main(){
10         int n;
11         n=Convert.ToInt32(Console.ReadLine());
12         Queue<Beteg> S = new Queue<Beteg>();
13         Beteg[] B=new Beteg[n+1];
14         string[] sor;
15         int v=0,varok=0,i=0;
```

```

16  for(int ii=0;ii<n;ii++){           //adatok be
17      sor=Console.ReadLine().Split();
18      B[ii]=new Beteg(Convert.ToInt32(sor[0]), Convert.ToInt32(sor[1]));
19  }
20  Beteg ku;                          //az aktuális beteg
21  while(i<n || S.Count!=0){          //amíg van beteg
22      if (S.Count >varok)
23          varok=S.Count;
24      if (S.Count!=0){                //van várakozó
25          ku=S.Dequeue();             //vegyük ez első várakozót
26      }else                           //nincs várakozó
27          ku=B[i++];                  //vegyük a következő érkezőt
28      if (v<ku.erk)                   //v-ig tart az aktuális kezelése
29          v=ku.erk+ku.kezeli;         //v-ig tart az aktuális kezelése
30      else
31          v+=ku.kezeli;                //eddig tart az aktuális kezelése
32      while(i<n && B[i].erk<v){        //v-ig beérkezők mennek a sorba
33          S.Enqueue(B[i]); i++;
34      }
35  }
36  Console.WriteLine(varok);
37  }
38  }

```

### 3.5. Feladat: Vizeskannák

Egy gazdának három kannája van. Kezdetben az első kanna tele van, a másik kettő pedig üres. Szeretne kimérni pontosan  $L$  liter vizet átöntésekkel. Egy átöntés során át tud tölteni egy tetszőleges kannából egy tetszőleges másik kannába vizet, addig amíg előbbi ki nem ürül, vagy utóbbi tele nem lesz (amelyik előbb bekövetkezik). Készíts programot, amely megadja, hogy minimum hány átöntéssel lehet kimérni pontosan  $L$  liter vizet!

#### Bemenet

Bemenet A standard bemenet első sorában a három kanna mérete van ( $1 \leq K1, K2, K3 \leq 10000$ ). A második sorban a kimérendő víz mennyisége található ( $1 \leq L \leq 10000$ ).

#### Kimenet

Kimenet A standard kimenet első és egyetlen sorába az átöntések minimális számát kell írni, amelylyel kimérhető pontosan  $L$  liter víz! Ha nem lehet kimérni, akkor -1-et kell kiírni!

#### Példa bemenet és kimenet

**bemenet**

8 5 3  
4

**kimenet**

6

#### Korlátok

Időlimit: 0.2

Memória limit: 32 MiB

### 3.5.1. Megoldás

Állapot:  $(a, b, c)$  ennyi víz van rendre a három kannában. Egy öntéssel egy másik állapot keletkezik.

A lehetséges öntögetéseket addig kell folytatni, amíg valamelyik kannában  $L$  liter nem lesz.

De, számom kell tartani, hogy egy adott állapotot korábban elértünk-e!

Észrevétel: Minden  $(a, b, c)$  állapotra igaz, hogy  $a + b + c = K1$ , ezért egyértelműen megadható az  $(a, b)$  párossal, mert  $c = K1 - (a + b)$

A korábban elért  $(a, b)$  állapotok állapotok reprezentálása:  $AB[a] := \{b : (a, b)\}$ ;

### 3.5.2. C++ megvalósítás

```
1  #include <bits/stdc++.h>
2  #define maxL 10001
3  using namespace std;
4  struct Allapot{
5      int a,b, l;
6      Allapot(int x1, int x2, int x3){a=x1; b=x2; l=x3;}
7      Allapot(){a=0;b=0;l=0;}
8  };
9  vector<int> AB[maxL];
10 int main(){
11     int K[3], L, mego=-1, ont;
12     cin>>K[0]>>K[1]>>K[2]>>L;
13     if(K[0]==L){cout<<"0\n"; return 0;}
14     Allapot ka=Allapot(K[0],0,0); Allapot q;
15     int qa[3], uja[3];
16     queue<Allapot> S;
17     S.push(ka);
18     AB[K[0]].push_back(0);
```

```

19 while(S.size()>0){
20     q=S.front(); S.pop();
21     qa[0]=q.a; qa[1]=q.b; qa[2]=K[0]-(q.a+q.b);
22     for(int i=0;i<3;i++){
23         for(int j=0;j<3;j++){
24             if(i==j) continue;
25             for(int x=0;x<3;x++) uja[x]=qa[x];
26             ont=min(qa[i], K[j]-qa[j]);
27             uja[i]-=ont;
28             uja[j]+=ont;
29             if(L==uja[0] || L==uja[1] || L==uja[2]){
30                 cout<<q.l+1<<endl; return 0;
31             }
32             bool volt=false;
33             for(int x:AB[uja[0]]) if(x==uja[1]){ volt=true; break;}
34             if(!volt){
35                 AB[uja[0]].push_back(uja[1]);
36                 S.push({uja[0],uja[1], q.l+1});
37             }
38         }
39     }
40 }
41 cout<<mege<<endl;
42 return 0;
43 }

```

### 3.6. Feladat: Elárasztás

Nagy vízhozamú forrás tört fel, amely elárasztja a környező területet. Azt szeretnénk megtudni, hogy hány nap alatt árasztja el a forrás a környékét. A probléma megoldáshoz a terület négyzetrácsos modelljét használhatjuk, ismerjük a forrás helyét és a forrást körülvevő, a forrás helyénél magasabban fekvő részeket. A víz egy nap alatt a négy (balra, jobbra, felfelé és lefelé lévő) szomszédos cella területét árasztja el, ha azok nem magasabbak a forrás helyénél.

#### Bemenet

A standard bemenet első sora négy egész számot tartalmaz, a négyzetrács sorainak  $m$  ( $0 < m \leq 1000$ ) és oszlopainak  $n$  ( $0 < n \leq 1000$ ) és a forrás  $fs$  és  $fo$  koordinátáit ( $1 \leq fs \leq m$   $1 \leq fo \leq n$ ). A négyzetrács bal felső cellájának koordinátái  $(1, 1)$ . A következő  $m$  sor írja le a területet, azon mezők, amelyek magasabbak a forrás helyénél 1, a többi 0 értékkel jelölt.

#### Kimenet

A standard kimenet első sorába azt a legkisebb  $k$  számot kell írni, ahány nap alatt a forrás elárasztja a területet!

## Példa bemenet és kimenet

**bemenet**

8 10 4 3

1 1 1 1 1 1 1 1 1 1

1 0 0 0 0 0 1 0 0 1

1 0 0 1 0 0 1 0 0 1

1 0 0 0 0 1 0 0 0 1

1 0 0 0 1 0 0 1 0 1

1 0 0 0 0 1 0 0 0 1

1 0 1 0 0 0 1 0 0 1

1 1 1 1 1 1 1 1 1 1

**kimenet**

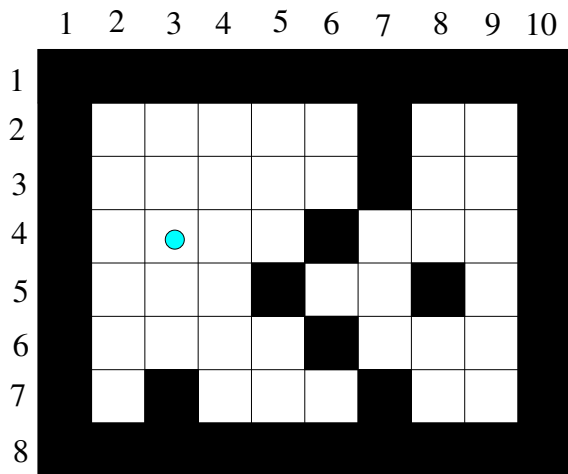
7

## Korlátok

Időlimit: 0.2

Memória limit: 32 MiB





1. ábra. A terület négyzetrácsos leírása

### 3.6.1. Megoldás

Tekintsük a celláknak a következő halmazait.

$$T(1) = \{(fs, fo)\}$$

$$T(k) = \{c : c \notin (T(1) \cup \dots \cup T(k-1)) \wedge \text{van olyan } cc \in T(k-1) \text{ cella, amely szomszéda } c\text{-nek}\}$$

A feladat megoldása az a legnagyobb  $k$ , amelyre  $T(k) \neq \emptyset$



2. ábra. Az S sor tartalma

	1						

2

3

4

5

6

7

8

	2						
2	1	2					
	2						

2

3

4

5

6

7

8

	3						
3	2						
2	1	2	3				
3	2	3					
	3						

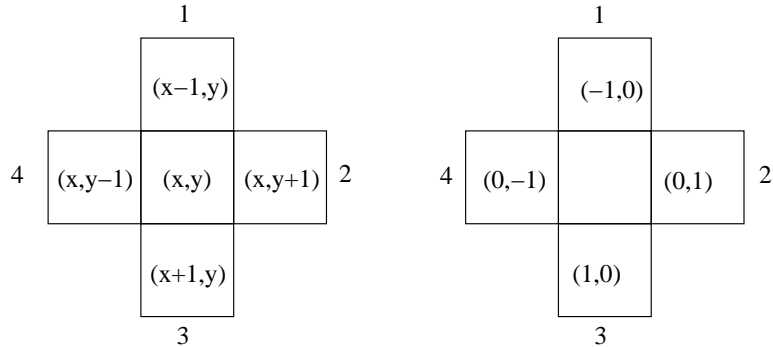
[illegible]

[illegible]

[illegible]



[illegible]



3. ábra. Egy cella szomszédai relatív koordinátákkal

### 3.6.2. C++ megvalósítás

```
1  #include <iostream>
2  #include <queue>
3  #define maxNM 1001
4  using namespace std;
5  int n,m,fs,fo;
6  int T[maxNM][maxNM];
7  struct Cella{
8      int x,y;
9      Cella(){};
10     Cella(int a, int b){x=a; y=b;}
11 };
12 Cella D[4]={ Cella(-1,0), Cella(0,1), Cella(1,0), Cella(0,-1)};
13
14 void Beolvas(){
15     int h;
16     cin>>m>n>fs>>fo;
17     for(int x=1;x<=m;x++){
18         for(int y=1;y<=n;y++){
19             cin>>h;
20             T[x][y]=h;
21         }
22     }
23 }
```

```
24 int main () {
25     Beolvas();
26     queue<Cella> S;
27     T[fs][fo]=1;
28     Cella c=Cella(fs,fo);
29     Cella cc;
30     S.push(c);
31     int mego=1;
32     while (!S.empty()) {
33         c=S.front(); S.pop();
34         for (int i=0; i<4; i++) {
35             cc.x=c.x+D[i].x; cc.y=c.y+D[i].y;
36             if (T[cc.x][cc.y]==0) {
37                 T[cc.x][cc.y]=T[c.x][c.y]+1;
38                 S.push(cc);
39                 if (T[cc.x][cc.y]>mego) mego=T[cc.x][cc.y];
40             }
41         }
42     }
43     cout<<mego<<endl;
44     return 0;
45 }
```

### 3.6.3. C# megvalósítás

```
1  using System;
2  using System.Collections.Generic;
3
4  class Program{
5      public class Cella{
6          public int x, y;
7          public Cella(int a, int b){x=a; y=b;}
8      }
9      public static void Main(){
10         string[] sor=Console.ReadLine().Split();
11         int m,n,fs,fo,h;
12         m=Convert.ToInt32(sor[0]);
13         n=Convert.ToInt32(sor[1]);
14         fs=Convert.ToInt32(sor[2]);
15         fo=Convert.ToInt32(sor[3]);
16         int[,] T=new int[m+2,n+2];
17         Queue<Cella> S = new Queue<Cella>();
18
19         for(int x=1;x<=m;x++){
20             sor=Console.ReadLine().Split();
21             for(int y=1;y<=n;y++){
22                 h=Convert.ToInt32(sor[y-1]);
23                 T[x,y]=h;
```

```
24     }
25 }
26 T[fs, fo]=1;
27 Cella[] D={new Cella(-1,0), new Cella(0,1), new Cella(1,0), new Cella
28 Cella c=new Cella(fs,fo);
29 Cella cc=new Cella(0,0);
30 S.Enqueue(c);
31 int mego=0;
32 while(S.Count!=0){
33     c=S.Dequeue();
34     for(int i=0;i<4;i++){
35         cc=new Cella(c.x+D[i].x, c.y+D[i].y);
36         if(T[cc.x, cc.y]==0){
37             T[cc.x, cc.y]=T[c.x, c.y]+1;
38             S.Enqueue(cc);
39             if(T[cc.x, cc.y]>mego) mego=T[cc.x, cc.y];
40         }
41     }
42 }
43 Console.WriteLine(mego);
44 }
45 }
```

### 3.7. Sorozat (kétvégű sor)

**Értékhalma:**  $Sorozat = \{\langle a_0, \dots, a_{n-1} \rangle : a_i \in E, i = 1, \dots, n, n \geq 0\}$

**Műveletek argumentumai:**  $A : Sorozat, x : E$

---

	Előfeltétel	Művelet	Utófeltétel
$A = \langle a_0, \dots, a_{n-1} \rangle$		Elejére( $A, x$ )	$A = \langle x, a_0, \dots, a_{n-1} \rangle$
$A = \langle a_0, \dots, a_{n-1} \rangle$		Végére( $A, x$ )	$A = \langle a_1, \dots, a_{n-1}, x \rangle$
$A = \langle a_0, \dots, a_{n-1} \rangle \wedge n > 0$		Első( $A$ )	$= a_0 \wedge A = Pre(A)$
$A = \langle a_0, \dots, a_{n-1} \rangle \wedge n > 0$		Utolsó( $A$ )	$= a_{n-1} \wedge A = Pre(A)$
$A = \langle a_0, \dots, a_{n-1} \rangle \wedge n > 0$		ElsőTöröl( $A$ )	$A = \langle a_1, \dots, a_{n-1} \rangle$
$A = \langle a_0, \dots, a_{n-1} \rangle \wedge n > 0$		UtolsóTöröl( $A$ )	$A = \langle a_0, \dots, a_{n-2} \rangle$
$A = \langle a_0, \dots, a_{n-1} \rangle$		Elemszám( $A$ )	$= n \wedge A = Pre(A)$
$A = A$		ÜresE( $A$ )	$= (Pre(A) = \langle \rangle) \wedge A = Pre(A)$
—		—	—
$A = \langle a_0, \dots, a_{n-1} \rangle \wedge n > 0$		Elem( $A, i$ )	$a_i \wedge A = Pre(A)$
$A = \langle a_0, \dots, a_{n-1} \rangle \wedge 0 \leq i \wedge i < n$		Módosít( $A, i, x$ )	$A = \langle x, a_0, \dots, a_{i-1}, x, a_{i+1}, \dots, a_{n-1} \rangle$
$A = \langle a_0, \dots, a_{n-1} \rangle$		Beszúr( $A, i, x$ )	$A = \langle x, a_0, \dots, a_{i-1}, x, a_i, \dots, a_{n-1} \rangle$
$A = \langle a_0, \dots, a_{n-1} \rangle$		Töröl( $A, i$ )	$A = \langle x, a_0, \dots, a_{i-1}, a_{i+1}, \dots, a_{n-1} \rangle$
$A = \langle a_0, \dots, a_{n-1} \rangle$		Üresít( $A$ )	$A = \langle \rangle$

---

### 3.8. A Sorozat C++ műveletei

---

Művelet	C++	C#
Típusdef	<i>deque &lt; E &gt;</i>	
Elejére( <i>A</i> , <i>x</i> )	<i>A.push_front(x)</i>	<i>A.Insert(0,x)</i>
Végére( <i>A</i> , <i>x</i> )	<i>A.push_back(x)</i>	<i>A.Add(x)</i>
Első( <i>A</i> )	<i>A.front()</i>	<i>A[0]</i>
Utolsó( <i>A</i> )	<i>A.back()</i>	<i>A[A.Count - 1]</i>
ElsőTöröl( <i>A</i> )	<i>A.pop_front()</i>	<i>A.RemoveAt(0)</i>
UtolsóTöröl( <i>A</i> )	<i>A.pop_back()</i>	<i>A.RemoveAt(A.Count - 1)</i>
Elemszám( <i>A</i> )	<i>A.size()</i>	<i>A.Count</i>
ÜresE( <i>A</i> )	<i>A.empty()</i>	<i>A.Count == 0</i>
<i>Elem(A,i)</i>	<i>A.item(i)</i>	—
<i>Elem(A,i)</i>	<i>A[i]</i>	<i>A[i]</i>
Módosít( <i>A,i,x</i> )	<i>A[i] = x</i>	<i>A[i] = x</i>
Beszúr( <i>A,i,x</i> )	<i>A.insert(A.begin() + i,x)</i>	<i>A.Insert(i,x)</i>
Töröl( <i>A,i</i> )	<i>A.erase(A.begin() + i)</i>	<i>A.RemoveAt(i)</i>
Üresít( <i>A,i</i> )	<i>A.clear()</i>	<i>A.Clear()</i>

---