



# Strukturált adattípusok

## Felvételi felkészítő

Dr. Păţcaş Csaba

Babeş-Bolyai Tudományegyetem  
Magyar Matematika és Informatika Intézet

2024.03.08



- Bemutatókozás
- Online kommunikáció: mindenki lenémítva, kézfelemelés
- $3 \times 50$  perc
- Kérdések



- Mit nevezünk strukturált adattípusoknak?
- Mire jók? Mikor ajánlatos a használatuk?
- Hogyan használjuk őket Pascal és C/C++ nyelvben?



- Különböző típusú adatokat **csoportosít**.
- Ezeket egyetlen adatként kezelhetjük.
- A csoport elemeihez is hozzáférünk, ezeket **mezőknek** nevezzük.
- Az azonos szintű mezők azonosítói (nevei) különbözőek kell legyenek.
- A mezők típusa lehet bármilyen, akár tömb, vagy egy másik struktúra is.
- A mezőazonosító önmagában nem kaphat értéket, hanem egy struktúra típusú változóval együtt használjuk.



- Azonos típusú struktúrák átadhatják egymásnak az összes mező értékét egyetlen **értékadással**.
- Pascalban a `record` kulcsszóval deklaráljuk.
- C/C++-ban általában `struct`-ot használunk, de ez tulajdonképpen egy olyan `class`, melynek minden mezője publikus.



Adott  $n$  személy, akiknek ismerjük a nevét, születési dátumát és a magasságát. Írassuk ki őket magasság szerinti csökkenő (név - magasság párokat), majd kor szerinti növekvő sorrendben (név - betöltött évek - betöltött napok formában).



- Egy év akkor szökőév ha osztható 4-el de nem osztható 100-al, vagy osztható 400-al.
- A C/C++-ba beépített sort függvénnel struktúrákat is rendezhetünk, ha megmondjuk mikor tekintünk egy struktúrát „kisebbnek” a másiknál (vagyis mikor kell korábban megjelenjen a végső sorrendben).
- Ennek a rendezésnek a bonyolultsága legrosszabb esetben  $\Theta(n \log n)$ .
- Pascal-ban nincs ilyen lehetőségünk, itt nekünk kell megírni egy hatékony rendezési algoritmust.
- Írhatunk lineáris idejű rendezésre alapuló megoldást is erre a feladatra.

# Első megoldás I



Strukturált  
adattípusok

Dr. Păţcaş  
Csaba

Bevezető

1. feladat

2. feladat

3. feladat

4. feladat

5. feladat

6. feladat

7. feladat

```
1  #include <vector>
2  #include <string>
3  #include <iostream>
4  #include <fstream>
5  #include <ctime>
6  #include <algorithm>
7
8  using namespace std;
9
10 #define FOR(i, a, b) for(int i = (a); i <= (b); ++i)
11
12 #define BE_ALLOMANY "fel1.in"
13 #define KI_ALLOMANY "fel1.out"
14
15 struct Szemely
16 {
17     string nev;
18     int szul_ev, szul_ho, szul_nap, bet_ev, bet_nap, magassag;
19 };
20
21 int AKT_EV, AKT_HO, AKT_NAP, AKT_SORSZAM;
22 int NAPOK[13] = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
23
24 bool szokoelv(int ev)
25 {
26     return ev % 4 == 0 && ev % 100 != 0 || ev % 400 == 0;
```



# Első megoldás II



Strukturált  
adattípusok

Dr. Pátcs  
Csaba

Bevezető

1. feladat

2. feladat

3. feladat

4. feladat

5. feladat

6. feladat

7. feladat

```
27 }
28
29 int szamolNapSorszam(int ev, int honap, int nap)
30 {
31     if (szokoev(ev)) NAPOK[2] = 29;
32     else NAPOK[2] = 28;
33     int sorszam = 0;
34     FOR(i, 1, honap - 1) sorszam += NAPOK[i];
35     sorszam += nap;
36     return sorszam;
37 }
38
39 void szamolEvekNapok(Szemely& sz)
40 {
41     if (sz.szul_ho < AKT_HO || sz.szul_ho == AKT_HO && sz.szul_nap <= AKT_NAP)
42     {
43         sz.bet_ev = AKT_EV - sz.szul_ev;
44         int szul_sorszam = szamolNapSorszam(AKT_EV, sz.szul_ho, sz.szul_nap);
45         sz.bet_nap = AKT_SORSZAM - szul_sorszam;
46     }
47     else
48     {
49         sz.bet_ev = AKT_EV - sz.szul_ev - 1;
50         int szul_sorszam = szamolNapSorszam(AKT_EV - 1, sz.szul_ho, sz.szul_nap);
51         sz.bet_nap = 365 - szul_sorszam + AKT_SORSZAM;
52         if (szokoev(AKT_EV - 1)) ++sz.bet_nap;
```

# Első megoldás III



Strukturált  
adattípusok

Dr. Pátcaş  
Csaba

Bevezető

1. feladat

2. feladat

3. feladat

4. feladat

5. feladat

6. feladat

7. feladat

```
53     }
54 }
55
56 void beolvas(int& n, vector <Szemely>& a)
57 {
58     ifstream f(BE_ALLOMANY);
59     f >> n;
60     a.resize(n + 1);
61     FOR(i, 1, n)
62     {
63         f >> a[i].nev >> a[i].szul_ev >> a[i].szul_ho >> a[i].szul_nap >> a[i].magassag;
64         szamolEvekNapok(a[i]);
65     }
66 }
67
68 bool magassagSzerint(Szemely& x, Szemely& y)
69 {
70     return x.magassag > y.magassag;
71 }
72
73 bool evSzerint(Szemely& x, Szemely& y)
74 {
75     return x.bet_ev < y.bet_ev || x.bet_ev == y.bet_ev && x.bet_nap < y.bet_nap;
76 }
77
78 ofstream f(KI_ALLOMANY);
```

# Első megoldás IV



Strukturált  
adattípusok

Dr. Pátcaş  
Csaba

Bevezető

1. feladat

2. feladat

3. feladat

4. feladat

5. feladat

6. feladat

7. feladat

```
79
80 void kiirMag(int n, vector <Szemely>& a)
81 {
82     FOR(i, 1, n) f << a[i].nev << " " << a[i].magassag << endl;
83     f << endl;
84 }
85
86 void kiirEv(int n, vector <Szemely>& a)
87 {
88     FOR(i, 1, n) f << a[i].nev << " " << a[i].bet_ev << " " << a[i].bet_nap << endl;
89 }
90
91 void maiDatum()
92 {
93     time_t t = time(0);
94     tm* most = localtime(&t);
95     AKT_EV = most->tm_year + 1900;
96     AKT_HO = most->tm_mon + 1;
97     AKT_NAP = most->tm_mday;
98     AKT_SORSZAM = most->tm_yday + 1;
99 }
100
101 int main()
102 {
103     int n;
104     vector <Szemely> személyek;
```



```
105
106     maiDatum();
107     beolvas(n, személyek);
108     sort(szemelyek.begin() + 1, személyek.end(), magassagSzerint);
109     kiirMag(n, személyek);
110     sort(szemelyek.begin() + 1, személyek.end(), evSzerint);
111     kiirEv(n, személyek);
112     return 0;
113 }
```

# Második megoldás I



Strukturált  
adattípusok

Dr. Păţcaş  
Csaba

Bevezető

1. feladat

2. feladat

3. feladat

4. feladat

5. feladat

6. feladat

7. feladat

```
1  #include <vector>
2  #include <string>
3  #include <iostream>
4  #include <fstream>
5  #include <ctime>
6  #include <algorithm>
7
8  using namespace std;
9
10 #define FOR(i, a, b) for(int i = (a); i <= (b); ++i)
11
12 #define BE_ALLOMANY "fel1.in"
13 #define KI_ALLOMANY "fel1.out"
14
15 struct Szemely
16 {
17     string nev;
18     int szul_ev, szul_ho, szul_nap, bet_ev, bet_nap, magassag;
19 };
20
21 int AKT_EV, AKT_HO, AKT_NAP, AKT_SORSZAM;
22 int NAPOK[13] = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
23
24 bool szokoelv(int ev)
25 {
26     return ((ev % 4 == 0) && (ev % 100 != 0) || (ev % 400 == 0));
```

## Második megoldás II



Strukturált  
adattípusok

Dr. Pátcaş  
Csaba

Bevezető

1. feladat

2. feladat

3. feladat

4. feladat

5. feladat

6. feladat

7. feladat

```
27 }
28
29 int szamolNapSorszam(int ev, int honap, int nap)
30 {
31     if (szokoev(ev)) NAPOK[2] = 29;
32     else NAPOK[2] = 28;
33     int sorszam = 0;
34     FOR(i, 1, honap - 1) sorszam += NAPOK[i];
35     sorszam += nap;
36     return sorszam;
37 }
38
39 void szamolEvekNapok(Szemely& sz)
40 {
41     if (sz.szul_ho < AKT_HO || sz.szul_ho == AKT_HO && sz.szul_nap <= AKT_NAP)
42     {
43         sz.bet_ev = AKT_EV - sz.szul_ev;
44         int szul_sorszam = szamolNapSorszam(AKT_EV, sz.szul_ho, sz.szul_nap);
45         sz.bet_nap = AKT_SORSZAM - szul_sorszam;
46     }
47     else
48     {
49         sz.bet_ev = AKT_EV - sz.szul_ev - 1;
50         int szul_sorszam = szamolNapSorszam(AKT_EV - 1, sz.szul_ho, sz.szul_nap);
51         sz.bet_nap = 365 - szul_sorszam + AKT_SORSZAM;
52         if (szokoev(AKT_EV - 1)) ++sz.bet_nap;
```

# Második megoldás III



Strukturált  
adattípusok

Dr. Pátcaş  
Csaba

Bevezető

1. feladat

2. feladat

3. feladat

4. feladat

5. feladat

6. feladat

7. feladat

```
53     }
54 }
55
56 void beolvas(int& n, vector<Szemely>& a)
57 {
58     ifstream f(BE_ALLOMANY);
59     f >> n;
60     a.resize(n + 1);
61     FOR(i, 1, n)
62     {
63         f >> a[i].nev >> a[i].szul_ev >> a[i].szul_ho >> a[i].szul_nap >> a[i].magassag;
64         szamolEvekNapok(a[i]);
65     }
66 }
67
68 ofstream f(KI_ALLOMANY);
69
70 void kiirMag(int n, vector<Szemely> a)
71 {
72     vector<vector<Szemely>> rendezve(301);
73     FOR(i, 1, n) rendezve[a[i].magassag].push_back(a[i]);
74     for (int i = 300; i >= 0; --i)
75         if (rendezve[i].size())
76             FOR(j, 0, rendezve[i].size() - 1)
77                 f << rendezve[i][j].nev << " " << rendezve[i][j].magassag << endl;
78     f << endl;
```

# Második megoldás IV



Strukturált  
adattípusok

Dr. Pátcaş  
Csaba

Bevezető

1. feladat

2. feladat

3. feladat

4. feladat

5. feladat

6. feladat

7. feladat

```
79 }
80
81 void kiirEv(int n, vector<Szemely> a)
82 {
83     vector<vector<vector<Szemely>>> rendezve(151, vector<vector<Szemely>>(367));
84     FOR(i, 1, n) rendezve[a[i].bet_ev][a[i].bet_nap].push_back(a[i]);
85     FOR(i, 0, 150)
86         FOR(j, 0, 366)
87             if (rendezve[i][j].size())
88                 FOR(k, 0, rendezve[i][j].size() - 1)
89                     f << rendezve[i][j][k].nev << " " << rendezve[i][j][k].bet_ev << " " << rendezve[i][j][k].bet_nap << endl;
90 }
91
92 void maiDatum()
93 {
94     time_t t = time(0);
95     tm* most = localtime(&t);
96     AKT_EV = most->tm_year + 1900;
97     AKT_HO = most->tm_mon + 1;
98     AKT_NAP = most->tm_mday;
99     AKT_SORSZAM = most->tm_yday + 1;
100 }
101
102 void main()
103 {
```





```
104     int n;  
105     vector<Szemely> személyek;  
106  
107     maiDatum();  
108     beolvas(n, személyek);  
109     kiirMag(n, személyek);  
110     kiirEv(n, személyek);  
111 }
```



Olvassunk be  $n$  komplex számot és számítsuk ki az összegüket és szorzatukat, majd rendezzük őket az origótól való távolság szerinti növekvő sorrendbe. Ha vannak az origótól azonos távolságra lévő számok, csoportosítsuk ezeket és írjuk ki mindegyik csoportra a benne szereplő számok sorszámait (az eredeti sorrendet figyelembe véve, 1-től számozva).



- Egy  $x + yi$  alakú komplex szám távolsága az origótól  $\sqrt{x^2 + y^2}$  (Pitagorasz tétele).
- Két komplex szám összege  $(x, y) + (u, v) = (x + u, y + v)$
- Két komplex szám szorzata  $(x, y) \cdot (u, v) = (xu - yv, xv + yu)$
- C++-ban újradefiniálhatjuk egy adott struktúrán definiált operátorokat.
- Két valós szám egyenlőségét soha ne ellenőrizzük direkt módon az egyenlőség operátorral.



```
1  #include <vector>
2  #include <fstream>
3  #include <algorithm>
4  #include <cmath>
5  #include <iomanip>
6
7  using namespace std;
8
9  #define FOR(i, a, b) for(int i = (a); i <= (b); ++i)
10
11 #define BE_ALLOMANY "fel2.in"
12 #define KI_ALLOMANY "fel2.out"
13
14 #define EPS 0.0000001
15
16 struct Komplex
17 {
18     double valos, imag, tav;
19     int index;
20
21     Komplex operator+(Komplex x)
22     {
23         Komplex s;
24         s.valos = valos + x.valos;
25         s.imag = imag + x.imag;
26         return s;
27     }
28 }
```



```
27     }
28
29     Komplex& operator+=(const Komplex& x)
30     {
31         valos += x.valos;
32         imag += x.imag;
33         return *this;
34     }
35
36     Komplex& operator*=(const Komplex& x)
37     {
38         double regiValos = valos;
39         valos = valos * x.valos - imag * x.imag;
40         imag = regiValos * x.imag + imag * x.valos;
41         return *this;
42     }
43
44     bool operator<(Komplex& x)
45     {
46         return tav < x.tav;
47     }
48
49     bool operator==(Komplex& x)
50     {
51         return abs(tav - x.tav) < EPS;
52     }
```



```
53 };
54
55 ofstream fki(KI_ALLOMANY);
56
57 void szamolTav(Komplex& k)
58 {
59     k.tav = sqrt(k.valos * k.valos + k.imag * k.imag);
60 }
61
62 void beolvas(int& n, vector <Komplex>& a)
63 {
64     ifstream fbe(BE_ALLOMANY);
65     fbe >> n;
66     a.resize(n + 1);
67     FOR(i, 1, n)
68     {
69         fbe >> a[i].valos >> a[i].imag;
70         szamolTav(a[i]);
71         a[i].index = i;
72     }
73 }
74
75 void kiirOsszegSzorzat(int n, vector <Komplex>& a)
76 {
77     Komplex s, p;
78     s.valos = 0;
```



```
79     s.imag = 0;
80     //FOR(i, 1, n) s = s + a[i];
81     FOR(i, 1, n) s += a[i];
82     p.valos = 1;
83     p.imag = 0;
84     FOR(i, 1, n) p *= a[i];
85     fki << "Osszeg= " << fixed << setprecision(2) << s.valos << " + " << s.imag << "i" << endl;
86     fki << "Szorzat= " << fixed << setprecision(2) << p.valos << " + " << p.imag << "i" << endl;
87 }
88
89 void kiirTav(int n, vector <Komplex>& a)
90 {
91     int i = 2;
92     while (i <= n)
93     {
94         if (a[i] == a[i - 1])
95         {
96             fki << endl << a[i].tav << " tavolsagra: " << a[i - 1].index;
97             while (i <= n && a[i] == a[i - 1])
98             {
99                 fki << " " << a[i].index;
100                 ++i;
101             }
102         }
103         ++i;
104     }
```



```
105 }  
106  
107 void main()  
108 {  
109     int n;  
110     vector <Komplex> szamok;  
111  
112     beolvas(n, szamok);  
113     kiirOsszegSzorzat(n, szamok);  
114     sort(szamok.begin() + 1, szamok.end());  
115     kiirTav(n, szamok);  
116 }
```





Adott  $n$  egész számokat tartalmazó mátrix, azzal a tulajdonsággal, hogy minden mátrix oszlopainak száma (kivéve az utolsót), egyenlő a rákövetkező mátrix sorainak számával. Számoljuk ki a mátrixsorozat szorzatát.

*Bónusz:* A szorzásokat milyen sorrendben érdemes végrehajtani ahhoz, hogy a műveletek össz-száma minimális legyen?



```
1  #include <vector>
2  #include <fstream>
3
4  using namespace std;
5
6  #define FOR(i, a, b) for(int i = (a); i <= (b); ++i)
7
8  struct Matrix
9  {
10     int sor, oszlop;
11     vector < vector <int> > a;
12 };
13
14 Matrix operator*(Matrix& x, Matrix& y)
15 {
16     Matrix p;
17     p.sor = x.sor;
18     p.oszlop = y.oszlop;
19     p.a.resize(x.sor + 1, vector <int>(y.oszlop + 1));
20     FOR(i, 1, x.sor)
21         FOR(j, 1, y.oszlop)
22             FOR(k, 1, x.oszlop)
23                 p.a[i][j] += x.a[i][k] * y.a[k][j];
24     return p;
25 }
26
```



```
27 ifstream fin("fel3.in");
28 ofstream fout("fel3.out");
29
30 void beolvas(int &n, vector<Matrix> &matrixok)
31 {
32     fin >> n;
33     matrixok.resize(n + 1);
34     FOR(i, 1, n)
35     {
36         int sor, oszlop;
37         fin >> sor >> oszlop;
38         matrixok[i].sor = sor;
39         matrixok[i].oszlop = oszlop;
40         matrixok[i].a.resize(sor + 1, vector<int>(oszlop + 1));
41         FOR(j, 1, sor)
42             FOR(k, 1, oszlop)
43                 fin >> matrixok[i].a[j][k];
44     }
45 }
46
47 void kiirSzorzat(int n, vector<Matrix>& matrixok)
48 {
49     Matrix p = matrixok[1];
50     FOR(i, 2, n) p = p * matrixok[i];
51
52     fout << "Szorzat:" << endl;
```



```
53     FOR(i, 1, p.sor)
54     {
55         FOR(j, 1, p.oszlop)
56             fout << p.a[i][j] << " ";
57         fout << endl;
58     }
59
60 }
61
62 void main()
63 {
64     int n;
65     vector<Matrix> matrixok;
66
67     beolvas(n, matrixok);
68     kiirSzorzat(n, matrixok);
69 }
```



Adott egy  $n$  csúcsú konvex sokszög, melynek az origó garantáltan a belsejében található. Rendezzük a sokszög pontjait az origó körüli trigonometrikus sorrendbe, majd határozzuk meg a sokszög területét!

*Bónusz:* Hogyan számíthatjuk ki egy konkáv sokszög területét?



- Az  $\text{atan2}(y, x)$  visszatéríti radiánban  $\frac{y}{x}$  arkusztangensét.
- Figyelembe veszi ezek előjelét is, így egy  $[-\pi, \pi]$  intervallumból való értéket térít vissza.
- Egy koordinátái által megadott háromszög területét determinánssal számolhatjuk ki.
- Ez negatív számot is visszaadhat, figyelembe véve a megadott pontok sorrendjét.
- Ezt felhasználva a háromszögelésre alapuló területszámítás konkáv sokszögekre is működik.



```
1  #include <vector>
2  #include <fstream>
3  #include <cmath>
4  #include <algorithm>
5
6  using namespace std;
7
8  #define FOR(i, a, b) for(int i = (a); i <= (b); ++i)
9
10 struct Pont
11 {
12     double x, y, szog;
13     void szamolSzog()
14     {
15         szog = atan2(y, x);
16     }
17
18     bool operator<(Pont &x)
19     {
20         return szog < x.szog;
21     }
22 };
23
24 double haromszogTerulet(Pont p1, Pont p2, Pont p3)
25 {
26     return (p1.x - p3.x) * (p2.y - p3.y) - (p2.x - p3.x) * (p1.y - p3.y);
```



```
27 }
28
29 struct Sokszog
30 {
31     int n;
32     vector <Pont> pontok;
33     void rendezPontok()
34     {
35         sort(pontok.begin() + 1, pontok.end());
36     }
37     double terület()
38     {
39         double t = 0;
40         FOR(i, 2, n - 1) t += haromszogTerulet(pontok[1], pontok[i], pontok[i + 1]);
41         return t / 2;
42     }
43 };
44
45 ifstream fin("fel4.in");
46 ofstream fout("fel4.out");
47
48 void beolvas(Sokszog &sokszog)
49 {
50     fin >> sokszog.n;
51     sokszog.pontok.resize(sokszog.n + 1);
52     FOR(i, 1, sokszog.n)
```





```
53     {
54         fin >> sokszog.pontok[i].x >> sokszog.pontok[i].y;
55         sokszog.pontok[i].szamolSzog();
56     }
57 }
58
59 void kiir(Sokszog &sokszog)
60 {
61     fout << "A pontok rendezve:" << endl;
62     FOR(i, 1, sokszog.n)
63         fout << sokszog.pontok[i].x << " " << sokszog.pontok[i].y << endl;
64     fout << "A sokszog terulete: " << sokszog.terulet();
65 }
66
67 Sokszog sokszog;
68 void main()
69 {
70     beolvas(sokszog);
71     sokszog.rendezPontok();
72     kiir(sokszog);
73 }
```



Minden személynek van születési dátuma és magassága. Egy focistának ezen kívül van még pozíciója is ahol játszik (kapus, védő, középpályás, vagy csatár), míg egy edzőnek van kedvenc taktikai felállása. Egy focicsapat 11 játékosból és egy edzőből áll. Egy bajnokságban 16 csapat szerepel. A bajnokság 15 fordulóból áll, mindegyik fordulóban 8 mérkőzésen két csapat játszik egymás ellen és mindegyik csapat rúg valamennyi gólt. Tervezzük meg milyen adatstruktúrákkal tárolnánk a bajnokság csapatait és a mérkőzések eredményeit! Hogyan iratnánk ki az  $i$ . forduló  $j$ . mérkőzéséhez tartozó adatokat (csapatok, edzők, végeredmény)?



Minden személynek van születési dátuma és magassága. Egy focistának ezen kívül van még pozíciója is ahol játszik (kapus, védő, középpályás, vagy csatár), míg egy edzőnek van kedvenc taktikai felállása. Egy focicsapat 11 játékosból és egy edzőből áll. Egy bajnokságban 16 csapat szerepel. A bajnokság 15 fordulóból áll, mindegyik fordulóban 8 mérkőzésen két csapat játszik egymás ellen és mindegyik csapat rúg valamennyi gólt. Tervezzük meg milyen adatstruktúrákkal tárolnánk a bajnokság csapatait és a mérkőzések eredményeit! Hogyan iratnánk ki az  $i$ . forduló  $j$ . mérkőzéséhez tartozó adatokat (csapatok, edzők, végeredmény)?

Pl.

```
bajnoksag.fordulok[i].merkozes[j].golok1
```



Minden személynek van születési dátuma és magassága. Egy focistának ezen kívül van még pozíciója is ahol játszik (kapus, védő, középpályás, vagy csatár), míg egy edzőnek van kedvenc taktikai felállása. Egy focicsapat 11 játékosból és egy edzőből áll. Egy bajnokságban 16 csapat szerepel. A bajnokság 15 fordulóból áll, mindegyik fordulóban 8 mérkőzésen két csapat játszik egymás ellen és mindegyik csapat rúg valamennyi gólt. Tervezzük meg milyen adatstruktúrákkal tárolnánk a bajnokság csapatait és a mérkőzések eredményeit! Hogyan iratnánk ki az  $i$ . forduló  $j$ . mérkőzéséhez tartozó adatokat (csapatok, edzők, végeredmény)?

Pl.

```
bajnoksag.fordulok[i].merkozes[j].golok1  
csapatok[bajnoksag.fordulok[i].merkozes[j].csapatIndex1].jatekosok[k].pozicio  
csapatok[bajnoksag.fordulok[i].merkozes[j].csapatIndex2].edzo.taktika
```



Adott  $n$  intervallum, melyeknek ismerjük a végpontjait. Határozzuk meg az intervallumok egyesítésének összhosszát!



```
1  #include <vector>
2  #include <fstream>
3  #include <iostream>
4  #include <algorithm>
5
6  #define FOR(i, a, b) for(int i = (a); i <= (b); ++i)
7  #define X first
8  #define Y second
9
10 #define BE_ALLOMANY "a.in"
11
12 using namespace std;
13
14 void beolvas(int& n, vector <pair <int , int> >& a)
15 {
16     ifstream fin(BE_ALLOMANY);
17     fin >> n;
18     a.resize(n + 1);
19     FOR(i, 1, n) fin >> a[i].X >> a[i].Y;
20 }
21
22 int egyesites(int n, vector <pair <int , int> > a)
23 {
24     vector < pair <int , bool> > vegpontok;
25     //true - intervallum kezdopont
26     //false - intervallum vegpont
```



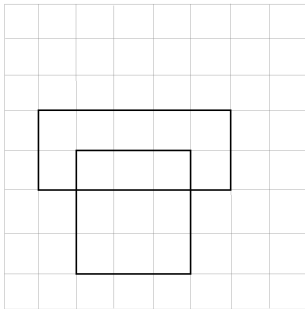
```
27     FOR(i, 1, n)
28     {
29         vegpontok.push_back(make_pair(a[i].X, true));
30         vegpontok.push_back(make_pair(a[i].Y, false));
31     }
32
33     sort(vegpontok.begin(), vegpontok.end());
34
35     int nyitva = 0, hossz = 0;
36     FOR(i, 0, 2 * n - 1)
37     {
38         if (vegpontok[i].Y) ++nyitva;
39         else --nyitva;
40         if (nyitva) hossz += vegpontok[i + 1].X - vegpontok[i].X;
41     }
42     return hossz;
43 }
44
45 int main()
46 {
47     vector <pair <int, int>> intervallumok;
48     int n;
49     beolvas(n, intervallumok);
50     cout << egyesites(n, intervallumok) << endl;
51     return 0;
52 }
```



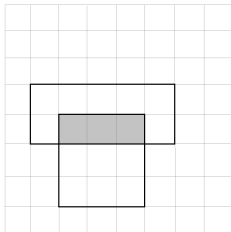




Adott  $n$  téglalap, melyek oldalai párhuzamosak a koordinátatengelyekkel és melyeket a bal alsó és a jobb felső sarkaik egész koordinátaival adunk meg. Határozzuk meg a téglalapok által lefedett területet!



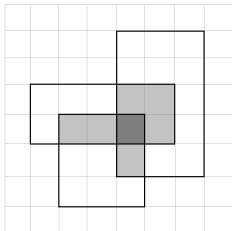
A fenti esetben a téglalapok által lefedett terület  $= 9 + 10 - 3 = 16$



- A fenti számításmódból jöhet az első ötlet: adjuk össze a téglalapok területét, majd vonjuk ki a közös területeket, melyeket páronként lefednek a téglalapok!
- Van-e ezzel az ötlettel valamilyen probléma?

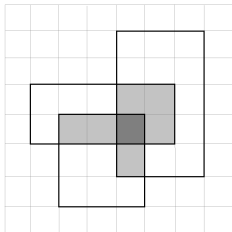


- Azokat a területeket, melyeket egyszerre három téglalap fed le, többször vontuk ki, így újra hozzá kellene adni a megoldáshoz.
- Ha léteznek olyan területek, melyeket egyszerre négy, öt,  $\dots$ ,  $n$  téglalap is lefed, azokat is kezelni kell, így ez a megoldás exponenciális időbonyolultságú lenne  $n$ -ben.
- A képlet amit ebben az esetben alkalmaznunk kellene a **szitaformula** nevet viseli (angolul **inclusion-exclusion principle**, románul **principiul includerii și excluderii**).



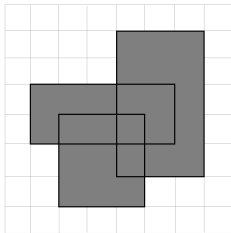


- Legyen  $m$  a koordináták maximális értéke, vagyis ha egy téglalap sarkai  $(x_1, y_1)$  és  $(x_2, y_2)$ , akkor írhatjuk, hogy  $0 \leq x_1, y_1, x_2, y_2 \leq m$
- Találhatunk-e egyszerű megoldást a feladatra, ha  $m$  értéke nem túl nagy, pl.  $m \leq 1000$ ?





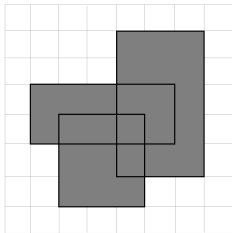
- Egy  $m \times m$  méretű logikai mátrixban megjelölhetjük azokat az  $1 \times 1$ -es négyzeteket IGAZ értékkel, melyeket lefed legalább egy téglalap.
- A kért megoldás az IGAZ értékek száma.



- Mennyi ennek az algoritmusnak a bonyolultsága?



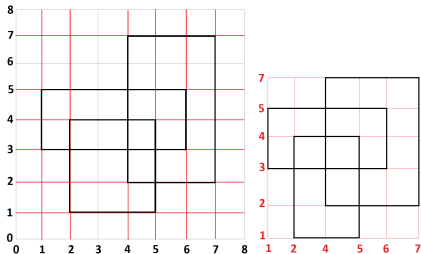
- Egy  $m \times m$  méretű logikai mátrixban megjelölhetjük azokat az  $1 \times 1$ -es négyzeteket IGAZ értékkel, melyeket lefed legalább egy téglalap.
- A kért megoldás az IGAZ értékek száma.



- Mennyi ennek az algoritmusnak a bonyolultsága?
- Idő:  $O(nm^2)$ , memória:  $O(m^2)$



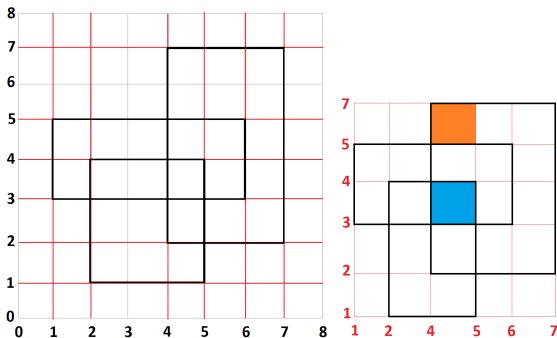
- Az előbbi megoldás akkor válik problémássá, amikor  $m$  értéke nőni kezd, pl. a koordináták 32 bites egész számok lehetnek.
- Ezt kiküszöbölhetjük a **normalizálás**nak nevezett módszerrel.
- Vegyük észre, hogy csak azok a koordináták fontosak, amelyeken egy téglalap kezdődik, vagy végződik (mind a vízszintes, mind a függőleges koordinátákat tekintve).







Miben különbözik a narancssárgával és a kékkel színezett négyzet?





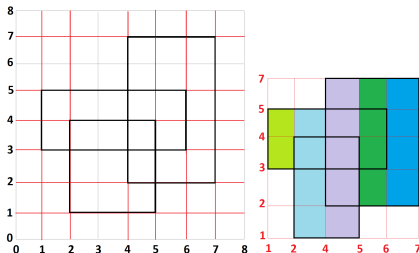
- Mivel a legrosszabb esetben  $2n$  különböző  $x$  és  $2n$  különböző  $y$  koordinátánk lehet, az  $m \times m$  méretű logikai mátrixot helyettesíthetjük egy legtöbb  $2n \times 2n$  méretűvel.
- A normalizáláshoz elegendő rendezni külön-külön egy-egy tömböt, melyek az  $x$  illetve  $y$  koordinátákat tartalmazzák ismétlődések nélkül.
- Az ismétlődések elkerülésére C++-ban használhatjuk a `set` vagy `unordered_set` típusokat (template-eket).
- Hogyan alakul hatékonyság szempontjából az algoritmus?



- Mivel a legrosszabb esetben  $2n$  különböző  $x$  és  $2n$  különböző  $y$  koordinátánk lehet, az  $m \times m$  méretű logikai mátrixot helyettesíthetjük egy legtöbb  $2n \times 2n$  méretűvel.
- A normalizáláshoz elegendő rendezni külön-külön egy-egy tömböt, melyek az  $x$  illetve  $y$  koordinátákat tartalmazzák ismétlődések nélkül.
- Az ismétlődések elkerülésére C++-ban használhatjuk a `set` vagy `unordered_set` típusokat (template-eket).
- Hogyan alakul hatékonyság szempontjából az algoritmus?
- Ennek az algoritmusnak az időbonyolultsága  $O(n^3)$ , memóriabonyolultsága  $O(n^2)$



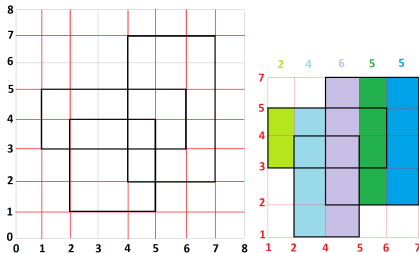
- A feladat megoldásához alkalmazhatjuk a **síkseprő egyenes (sweep line)** módszerét.
- Képzeljünk el egy függőleges egyenest, mely „átseper” a teljes síkon balról jobbra.
- Tárolnunk kell az egyenes aktuális **állapot**ával kapcsolatos információkat, a mi esetünkben, hogy éppen mely téglalapokat metszi.
- Észrevehetjük, hogy az állapot csak a normalizált  $x$  koordinátákban változik, ezeket **eseményeknek** nevezzük.



A különböző színekkel jelölt függőleges „sávokban” lesz a síkseprő egyenes állapota más és más.

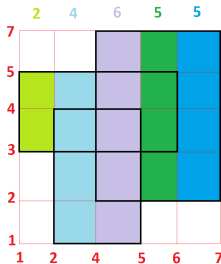
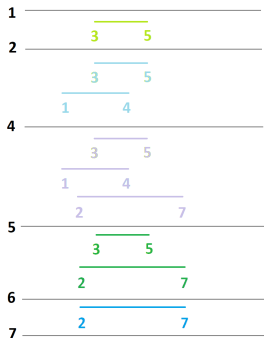


- A végeredmény egy összeg lesz, melynek minden tagja egy-egy sáv területe.
- Egy sáv területét úgy kapjuk, hogy összeszorozzuk a sáv szélességét az egyenes által lefedett téglalapok összmagasságával.





A téglalapok összmagassága, tulajdonképpen az aktív téglalapok  $y$  koordinátái által meghatározott intervallumok egyesítésének a hossza.





- Az egyesítések hosszának meghatározásához, alkalmazhatjuk az előbbi feladatnál látottakat.
- Így az időbonyolultságot  $O(n^2 \log n)$ -re, a memóriabonyolultságot  $O(n)$ -re csökkentettük.
- *Bónusz:* Lehet-e jobban?





```
1  #include <vector>
2  #include <fstream>
3  #include <cmath>
4  #include <algorithm>
5  #include <set>
6
7  using namespace std;
8
9  #define FOR(i, a, b) for(int i = (a); i <= (b); ++i)
10 #define X first
11 #define Y second
12
13 struct Teglalap
14 {
15     pair<int, int> p1, p2;
16 };
17
18 ifstream fin("fel7.in");
19 ofstream fout("fel7.out");
20
21 void beolvas(int& n, vector<Teglalap>& a, vector<pair<int, int>>& esemenyek)
22 {
23     fin >> n;
24     a.resize(n + 1);
25     FOR(i, 1, n)
26     {
```



```
27     fin >> a[i].p1.X >> a[i].p1.Y >> a[i].p2.X >> a[i].p2.Y;
28     esemenyek.push_back(make_pair(a[i].p1.X, i));
29     esemenyek.push_back(make_pair(a[i].p2.X, -i));
30 }
31 }
32
33 int egyesites(set < pair <int, int> >& intervallumok)
34 {
35     vector < pair <int, bool> > vegpontok;
36     //true - intervallum kezdopont
37     //false - intervallum vegpont
38     for (set < pair <int, int> >::iterator it = intervallumok.begin(); it != intervallumok.end();
39         ++it)
40     {
41         vegpontok.push_back(make_pair(it->X, true));
42         vegpontok.push_back(make_pair(it->Y, false));
43     }
44     sort(vegpontok.begin(), vegpontok.end());
45
46     int nyitva = 0, hossz = 0;
47     FOR(i, 0, (int)vegpontok.size() - 1)
48     {
49         if (vegpontok[i].Y) ++nyitva;
50         else --nyitva;
51         if (nyitva) hossz += vegpontok[i + 1].X - vegpontok[i].X;
```



```
52     }
53     return hossz;
54 }
55
56 int szamolTerulet(vector <Teglalap>& teglalapok, vector <pair <int, int> >& esemenyek)
57 {
58     set < pair <int, int> > intervallumok;
59     int terület = 0;
60     FOR(i, 0, esemenyek.size() - 1)
61     {
62         if (i > 0 && esemenyek[i].X > esemenyek[i - 1].X)
63         {
64             int szel = esemenyek[i].X - esemenyek[i - 1].X;
65             terület += szel * egyesites(intervallumok);
66         }
67         if (esemenyek[i].Y > 0)
68         {
69             int index = esemenyek[i].Y;
70             intervallumok.insert(make_pair(teglalapok[index].p1.Y, teglalapok[index].p2.Y));
71         }
72         else
73         {
74             int index = -esemenyek[i].Y;
75             intervallumok.erase(make_pair(teglalapok[index].p1.Y, teglalapok[index].p2.Y));
76         }
77     }
```



```
78     return terület;
79 }
80
81 int main()
82 {
83     int n;
84     vector <Teglalap> teglalapok;
85     vector <pair <int , int> > esemenyek;
86
87     beolvas(n, teglalapok , esemenyek);
88     sort(esemenyek.begin(), esemenyek.end());
89     fout << "Terület = " << szamolTerulet(teglalapok , esemenyek);
90
91     return 0;
92 }
```