Összefuttatás

Összefuttató felsoroló

- Egy összefuttató felsoroló több másik felsorolás elemeit sorolja fel egyetlen sorozatként.
- A továbbiakban feltesszük, hogy mindig olyan felsorolókat futtatunk össze, amelyek által felsorolt elemek egy közös szempont szerint szigorúan növekedően rendezettek, más szavakkal, minden felsorolás egyértelmű és növekedően rendezett.

Összefuttatás alapfeladata

Adott két, azonos szempont szerint egyértelmű és rendezett felsorolás. Futtassuk össze ezek elemeit a rendezettségük sorrendjében, és dolgozzuk fel őket attól függően, hogy mi egy elem állása, azaz csak az egyik, csak a másik, vagy mindkettő felsorolásban szerepel-e! A feldolgozás bármi lehet (számlálás, maximum kiválasztás, keresés), de most csak egyszerűen fűzzük fel egy sorozatba az állásuktól függően átalakított elemeket.

A : $x : enor(E), y : enor(E), z : F^*$

a felsorolás egyértelmű és rendezett

Ef : $x = x_0 \wedge y = y_0 \wedge x\uparrow \wedge y\uparrow$

összefuttatás jele

Uf : $z = \bigoplus_{e \in x_0 \cup y_0} f(e)$

a felsorolás elemeinek halmaza

ahol $f : E \to F^*$ és

$$f(e) = \begin{cases} f_1(e) & \text{ha } e \in \{x_0\} \wedge e \notin \{y_0\} \\ f_2(e) & \text{ha } e \notin \{x_0\} \wedge e \in \{y_0\} \\ f_3(e) & \text{ha } e \in \{x_0\} \wedge e \in \{y_0\} \end{cases}$$

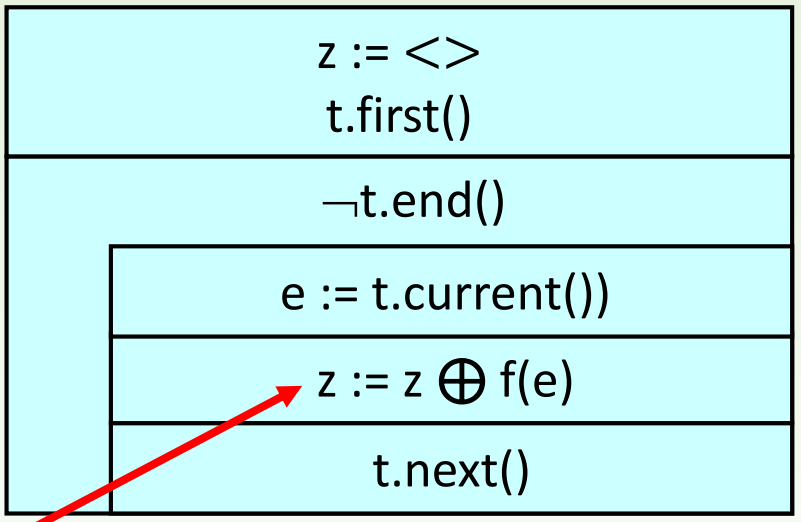
és $f_1, f_2, f_3 : E \to F^*$ adott függvények

Tervezés

Összegzés:

t:enor(E) ~ x∪y:enor(E)
x,y : enor(E)

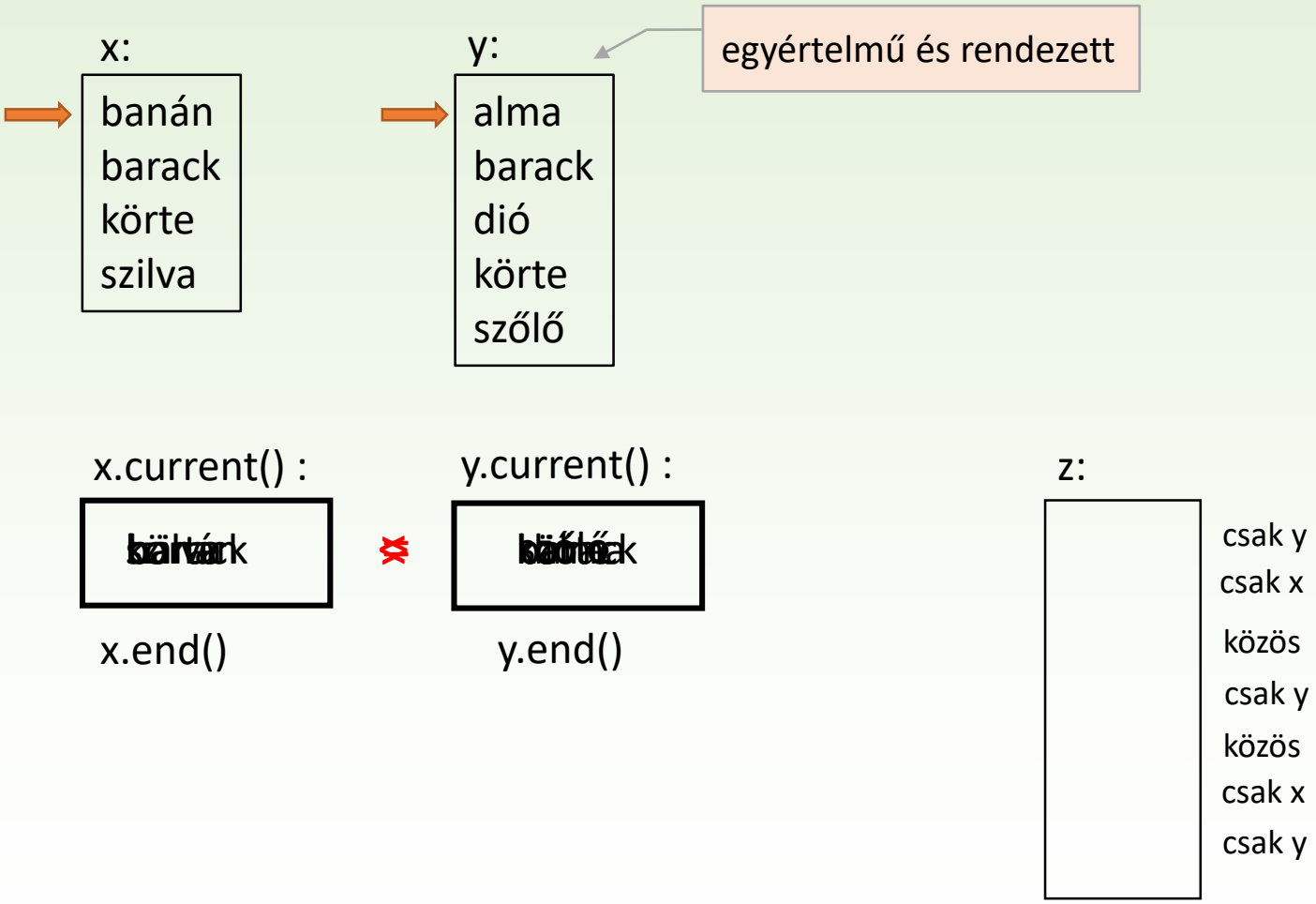
H,+,0 ~ F*, ⊕, < >



$e \in \{x\} \wedge e \notin \{y\}$	$e \notin \{x\} \wedge e \in \{y\}$	$e \in \{x\} \wedge e \in \{y\}$
$z := z \oplus f_1(e)$	$z := z \oplus f_2(e)$	$z := z \oplus f_3(e)$

Ezen feltételek kiértékelése így nem végezhető el hatékonyan. Jó lenne, ha a t.current() azt is megadná, hogy mi az aktuális elem állása felsorolásokban: csak x-ben, csak y-ban, vagy mindkettőben benne van-e.

Összefuttatás ötlete



Összefuttató felsoroló

enor(E)

E*	first()	next()	current()	end()
x : enor(E) y : enor(E)	x.fist() y.first()	lásd lenn	lásd lenn	x.end() ∧ y.end()

e, p := current()

A feltételek megfogalmazásánál kihasználjuk, hogy a felsorolás még tart: ¬x.end() ∨ ¬y.end()

y.end() ∨ (¬x.end() ∧ x.current() < y.current())	x.end() ∨ (¬y.end() ∧ x.current() > y.current())	¬x.end() ∧ ¬y.end() ∧ x.current() = y.current()
e, p := x.current(), ”x”	e, p := y.current(), ”y”	e, p := x.current(), ”xy”

next()

y.end() ∨ (¬x.end() ∧ x.current() < y.current())	x.end() ∨ (¬y.end() ∧ x.current() > y.current())	¬x.end() ∧ ¬y.end() ∧ x.current() = y.current()
x.next()	y.next()	x.next(); y.next()

Összefuttatás ciklusmagja

e, p := t.current()
z := z ⊕ f(e, p)
t.next()

e, p := current()

y.end() ∨ (¬x.end() ∧ x.current()<y.current())	x.end() ∨ (¬y.end() ∧ x.current()>y.current())	¬x.end() ∧ ¬y.end() ∧ x.current()=y.current()
e, p := x.current(), ”x”	e, p := y.current(), ”y”	e, p := x.current(), ”xy”

z := z ⊕ f(e, p)

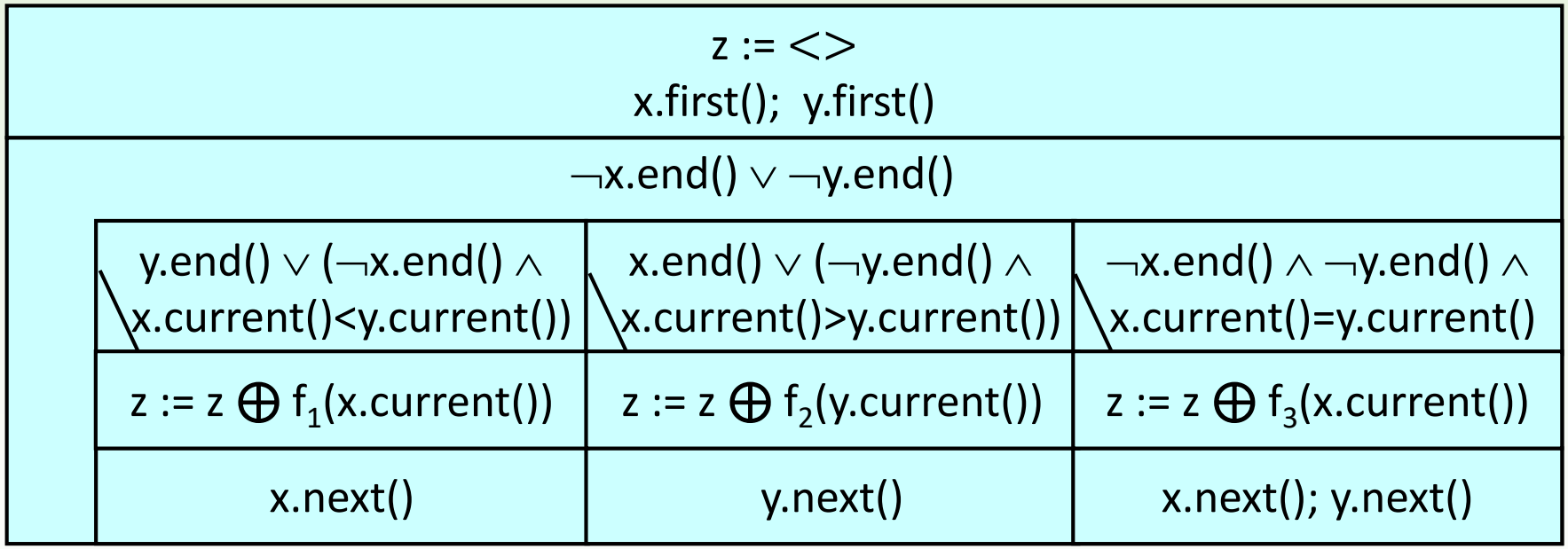
A három elágazás összevonásával a p kiküszöbölhető.

p = ”x”	p = ”y”	p = ”xy”
z := z ⊕ f_1(e)	z := z ⊕ f_2(e)	z := z ⊕ f_3(e)

next()

y.end() ∨ (¬x.end() ∧ x.current()<y.current())	x.end() ∨ (¬y.end() ∧ x.current()>y.current())	¬x.end() ∧ ¬y.end() ∧ x.current()=y.current()
x.next()	y.next()	x.next(); y.next()

Összefuttatás algoritmusa



Az x és y lecserélhető konkrét felsorolókra.

Összefuttatás kódja

```
x.first(); y.first();
while ( !x.end() || !y.end() ) {
    if(y.end() || (!x.end() && x.current()<y.current()) ){
        … x.current() … ; read(x,dx,sx);
    } else if(x.end() || (!y.end() && x.current()>y.current()) ){
        … y.current() … ; read(y,dy,sy);
    } else if(!x.end() && !y.end() && x.current()=y.current()) {
        … x.current() … ; read(x,dx,sx); read(y,dy,sy);
    }
}
```

Ez a változat nem jó, mert nem egy háromágú elágazás kódját, hanem három üres „else” ágú elágazás szekvenciájának kódját tartalmazza: ez x=<”alma”> és y=< > esetén rosszul működik, hiszen feldolgoz egy nem létező y-beli elemet is.

```
x.first(); y.first();
while( !x.end() || !y.end() ) {
    if(y.end() || (!x.end() && x.current()<y.current())
        { … x.current() … ; read(x,dx,sx); }
    if(x.end() || (!y.end() && x.current()>y.current()) ){
        { … y.current() … ; read(y,dy,sy); }
    if(!x.end() && !y.end() && x.current()=y.current()) {
        { … x.current() … ; read(x,dx,sx); read(y,dy,sy); }
}
```

Összefuttatás tesztelése

- Az összefuttatás vizsgálata
  - mindkét felsoroló üres
  - csak az egyik felsoroló üres
  - az x-beli elemek mind kisebbek az y-beli elemeknél, és fordítva
  - az x-beli és az y-beli elemek azonosak
  - vegyes: x = { 1, 2, 5, 6 } és y = { 3, 4, 5 }

1. Feladat

Az Objektumelvű programozás tárgy előfeltétele az új Programozás tárgy vagy a régi Programozási alapismeretek tárgy. Egy szekvenciális input fájl az Objektumelvű programozás kurzust most felvevő (x), egy másik az új Programozás kurzust előző ősszel felvevő (y) hallgatók névsorát tartalmazza. Mindkét fájl egyértelmű és növekedően rendezett.

Kik azok az Objektumelvű programozás kurzust felvevő hallgatók, akik az előfeltételt nem előző ősszel teljesítették?

A : x : infile(String), y : infile(String), z : outfile(String)

Ef : $x = x_0 \wedge y = y_0 \wedge x\uparrow \wedge y\uparrow$

Uf : $z = \bigoplus_{e \in x_0 \cup y_0} f(e)$

inputfájlok összefuttató felsorolása

ahol $f : String \rightarrow String^*$ és

$$f(e) = \begin{cases} <e> \text{ ha } e \in \{x_0\} \wedge e \notin \{y_0\} \\ <> \text{ ha } e \notin \{x_0\} \wedge e \in \{y_0\} \\ <> \text{ ha } e \in \{x_0\} \wedge e \in \{y_0\} \end{cases}$$

Algoritmus

Összegzés:

t:enor(E) ~ x∪y:enor(String)

x,y : infile(String)

f(e) ~
- <e> ha $e \in \{x_0\} \wedge e \notin \{y_0\}$
- < > ha $e \notin \{x_0\} \wedge e \in \{y_0\}$
- < > ha $e \in \{x_0\} \wedge e \in \{y_0\}$

H,+,0 ~ String*, $\oplus$, < >

A szekvenciális inputfájlok összefuttató felsorolása az sx,dx,x:read és sy,dy,y:read olvasásokra épül.

f: String → String*

z := <>

sx, dx, x : read : sy, dy, y : read

sx=norm ~~∨ sy=norm~~

Az algoritmus a konkrét feladat ismeretében egyszerűsíthető.

sy=abnorm∨ (~~sx=norm~~ ∧ dx < dy)	~~sx=abnorm ∨~~ (sy=norm ∧ dx > dy)	~~sx=norm~~ ∧ sy=norm ∧ dx = dy
z : write(dx)	–	–
sx, dx, x : read	sy, dy, y : read	sx, dx, x : read; sy, dy, y : read

Program

```
bool read(ifstream &f, string &df, Status &sf){
     f >> df;
     sf = f.eof() ? abnorm : norm;
     return st==norm;
}
```

```
Status = enum{ abnorm, norm);
bool read(ifstream &f, string &df, Status &st);
void write(ofstream &f, const string &df);

int main()
{
     ifstream x("inp1.txt");  // hiányzik az ellenőrzés
     ifstream y("inp2.txt");  // hiányzik az ellenőrzés
     ofstream z("out.txt");   // hiányzik az ellenőrzés

     read(x,dx,sx); read(y,dy,sy);
     while ( norm == sx) {
          if( abnorm == sy || dx < dy) ){
               write(z,dx);
               read(x,dx,sx);
          }else if( norm == sy &&  dx > dy ){
               read(y,dy,sy);
          }else if( norm == sy && dx == dy ){
               read(x,dx,sx); read(y,dy,sy);
          }
     }
     return 0;
}
```

```
void write(ofstream &f, const string &df){
     f << df << endl;
}
```

2. Feladat

Az Objektumelvű programozás tárgy előfeltétele az új Programozás tárgy vagy a régi Programozási alapismeretek tárgy. Egy szekvenciális input fájl az Objektumelvű programozás kurzust most felvevő (x), egy másik az új Programozás kurzust előző ősszel felvevő (y) hallgatók névsorát tartalmazza. Mindkét fájl egyértelmű és növekedően rendezett.

Hány olyan hallgató van, aki mindkét fájlban szerepel, de a neve rövidebb, mint 7 betű?

A : x : infile(String), y : infile(String), c : $\mathbb{N}$

Ef : $x = x_0 \wedge y = y_0 \wedge x\uparrow \wedge y\uparrow$

Uf : $c = \sum_{\substack{e \in x_0 \cup y_0 \\ e \in \{x_0\} \wedge e \in \{y_0\} \wedge |e|<7}} 1$

Algoritmus

A szekvenciális inputfájlok összefuttató felsorolása az sx,dx,x:read és sy,dy,y:read olvasásokra épül.

Számlálás:

t:enor(E) ~ x∪y:enor(String)
x,y : infile(String)

felt(e) ~ e∈{x} ∧ e∈{y} ∧ |e|<7

felt: String → 𝕃 és

$$felt(e) = \begin{cases} hamis & ha\ e \in \{x_0\} \wedge e \notin \{y_0\} \\ hamis & ha\ e \notin \{x_0\} \wedge e \in \{y_0\} \\ igaz & ha\ e \in \{x_0\} \wedge e \in \{y_0\} \wedge |e| < 7 \\ hamis & ha\ e \in \{x_0\} \wedge e \in \{y_0\} \wedge |e| \geq 7 \end{cases}$$

c := 0
sx, dx, x : read; sy, dy, y : read

sx=norm ∧ sy=norm

~~sy=abnorm∨ (sx=norm ∧ dx < dy)~~	~~sx=abnorm ∨ (sy=norm ∧ dx > dy)~~	~~sx=norm ∧ sy=norm~~ ∧ dx = dy
–	–	|dx|<7
		c := c+1 / –
sx, dx, x : read	sy, dy, y : read	sx, dx, x : read; sy, dy, y : read

Az algoritmus a konkrét feladat ismeretében egyszerűsíthető.

Program

```
Status = enum{ abnorm, norm);
bool read(ifstream &f, string &df, Status &st);

int main()
{
     ifstream x("inp1.txt");   // hiányzik az ellenőrzés
     ifstream y("inp2.txt");   // hiányzik az ellenőrzés
     int c = 0;
     read(x,dx,sx); read(y,dy,sy);
     while ( norm == sx && norm == sx ) {
          if( dx < dy ){
               read(x,dx,sx);
          }else if( dx > dy ){
               read(y,dy,sy);
          }else if( dx == dy ){
               if(dx.size()<7) ++c;
               read(x,dx,sx); read(y,dy,sy);
          }
     }
     cout << "Mindket fajlban szereplo hallgatok szama: " << c << endl;
     return 0;
}
```

```
bool read(ifstream &f, string &df, Status &sf){
     f >> df;
     sf = f.eof() ? abnorm : norm;
     return st==norm;
}
```

3. Feladat

Adott két egész számokat tartalmazó tömb, mindkettő egyértelmű és növekedően rendezett.

Keressünk bennük olyan páros számot, amely vagy csak az egyik, vagy csak a másik tömbben található!

A : $x : \mathbb{Z}^n, y : \mathbb{Z}^m, l : \mathbb{L}, k : \mathbb{Z}$

Ef : $x = x_0 \wedge y = y_0 \wedge x\uparrow \wedge y\uparrow$

Uf : $l, k = \textbf{SEARCH}_{e \in x_0 \cup y_0} \left(((e \in \{x_0\} \wedge e \notin \{y_0\}) \vee (e \in \{x_0\} \wedge e \notin \{y_0\})) \wedge 2 \mid e\right)$

tömbök összefuttató felsorolása

tömb elemeinek halmaza

Algoritmus

tömbök összefuttató felsorolása a tömbök indexelésére épül.

Lineáris keresés:

t:enor(E) ~ $x \cup y$:enor($\mathbb{Z}$)

$x : \mathbb{Z}^n, y : \mathbb{Z}^m$

felt(e) ~ $((e \in \{x\} \wedge e \notin \{y\}) \vee (e \in \{x\} \wedge e \notin \{y\})) \wedge 2 \mid e$

felt: $\mathbb{Z} \rightarrow \mathbb{L}$ és

$$\text{felt}(e) = \begin{cases} 2 \mid e & \text{ha } e \in \{x_0\} \wedge e \notin \{y_0\} \wedge 2 \mid e \\ 2 \mid e & \text{ha } e \notin \{x_0\} \wedge e \in \{y_0\} \wedge 2 \mid e \\ \text{hamis} & \text{ha } e \in \{x_0\} \wedge e \in \{y_0\} \end{cases}$$

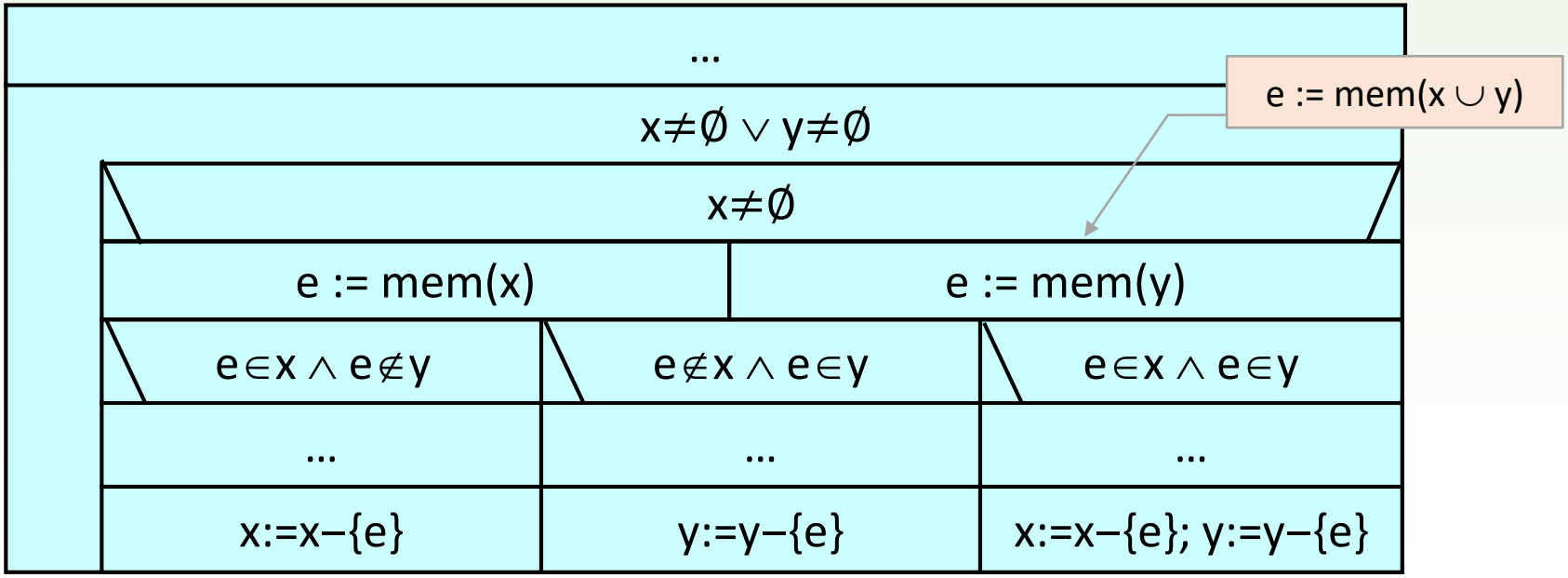
l := hamis i, j := 1, 1		
$\neg l \wedge (i \le n \vee j \le m)$		
$j > m \vee (i \le n \wedge dx < dy)$	$i > n \vee (j \le m \wedge dx > dy)$	$i \le n \wedge j \le m \wedge dx = dy$
$l, k := 2 \mid dx, dx$	$l, k := 2 \mid dy, dy$	–
i := i+1	j := j+1	i , j := j+1

Program

```
int main()
{
     vector<int> x; …       // x feltöltése
     vector<int> y; …       // y feltöltése
     unsigned int i,j;
     int k;
     bool l = false;
     i = j = 1;
     while ( !l && (i<x.size() || j<y.size() ) {
          if( j>=y.size()  || (i<x.size() && x[i]<x[j]) ){
               l = x[i]%2 == 0; k = x[i];
               ++i;
          }else if( i>=x.size() || (j<y.size() && x[i]>x[j]) ){
               l = x[i]%2 == 0; k = x[i];
               ++j;
          }else if( i<x.size() && j<y.size() && x[i]==x[j] ){
               ++i; ++j;
          }
     }
     if (l) cout << "Csak egyik tombben levo paros szam: " << k << endl;
     else   cout << "Nincs csak egyik tombben levo paros szam.\n" ;
     return 0;
}
```

Halmazok összefuttatása

- Egy halmaz elemeinek felsorolásához használt e:=mem(x) (x:set(E), e:E) művelet nem garantálja a halmaz elemeinek rendezett felsorolását.
- De két halmaz összefuttatásához ($e \in x \cup y$) nem kell a rendezettség, mert hatékony művelet az „eleme-e” lekérdezés: l:=e∈x (x:set(E), e:E, l:𝕃).



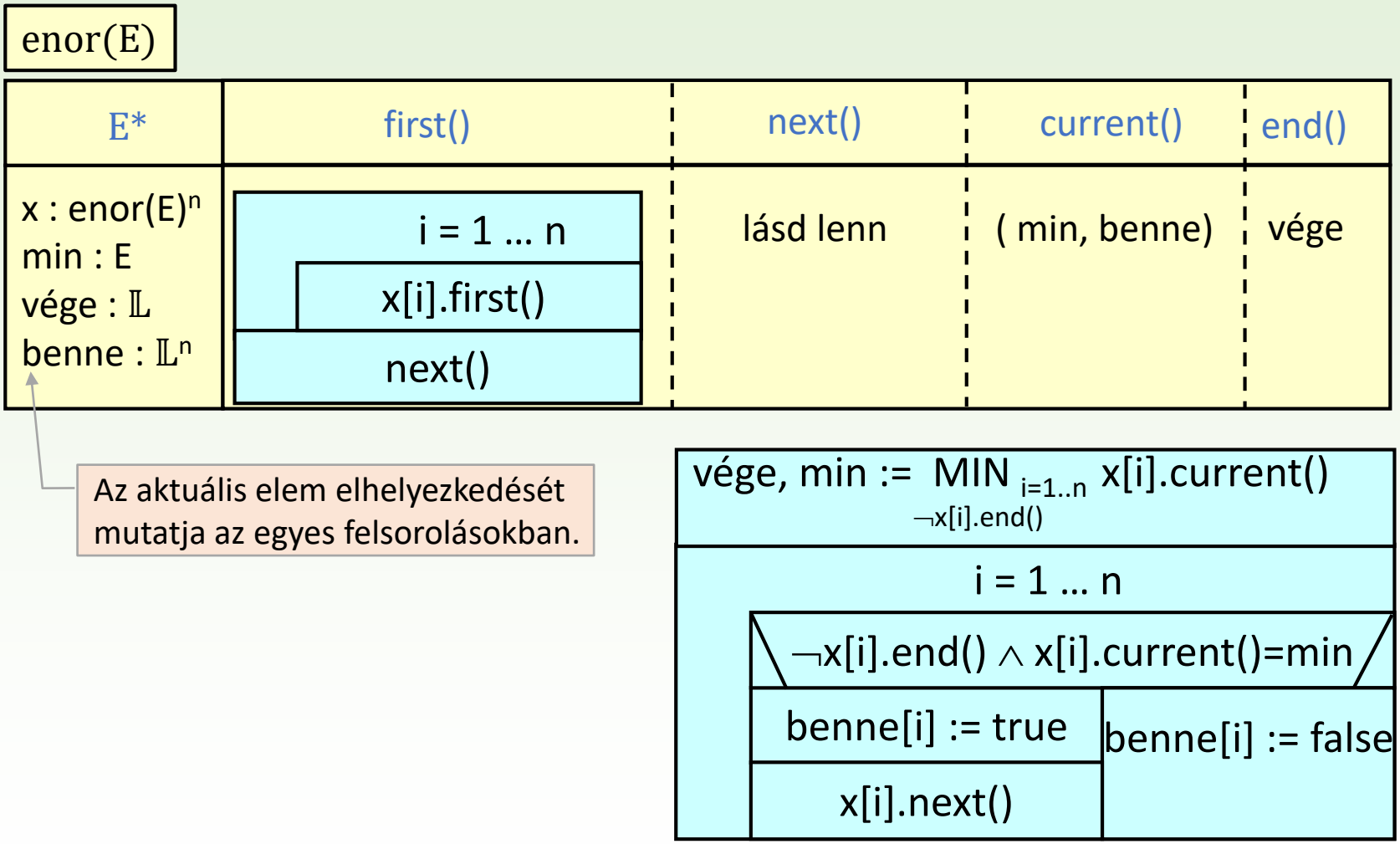
Vegyes összefuttatás

- Nem csak azonos típusú felsorolókat lehet összefuttatni, hanem például egy inputfájl és egy tömb felsorolásait is (feltéve, hogy a felsorolt elemek azonos típusúak).
- Problémát ez csak akkor okoz, ha az egyik felsoroló egy halmazt jár be, a másik pedig egy sorozat szerkezetű gyűjteményt. Ekkor
  - vagy a halmaz elemeit kell rendezett módon felsorolni,
  - vagy egy sorozatról kell tudni eldönteni, hogy egy elem benne van-e. Ez utóbbi azonban végezhető el konstans időben, ezért inkább az alábbi egyedi módszert érdemes használni:
    - Először a sorozat elemeit dolgozzuk fel annak megfelelően, hogy azok a halmaznak elemei-e vagy sem, ha igen, ki is vesszük ezeket a halmazból.
    - Ezután a halmazban maradt elemeket (ezek biztosan nem voltak a sorozatban) dolgozzuk egyenként fel.

Három vagy még több felsoroló egyidejű összefuttatása

- Nemcsak két, hanem három vagy még több felsoroló egyszerre történő összefuttatásáról is lehet beszélni.
- De amíg két felsoroló összefuttatása 3 esetet különböztet meg (a feldolgozandó elem csak az egyik, csak a másik, mindkettő felsorolásban szerepel), addig három felsoroló összefuttatása már 7 eset, n darab felsorolóé 2^n-1 eset kezelését igényli. Nem tűnik kényelmesnek ilyen monumentális elágazások használata.
- Szóba jöhetne egy „cascade” jellegű feldolgozás, amely először két felsorolót futtat össze, majd az így kapott sorozat felsorolását a harmadik felsorolóval, és így tovább.
- Elegánsabb megoldást kínál a következő dia.

n darab felsoroló összefuttatása



4. Feladat

Adott n darab egész számokat tartalmazó szekvenciális inputfájl, mindegyik egyértelmű és növekedően rendezett.

Soroljuk fel a fájlokban levő egész számokat, de mindegyiket csak egyszer!

A : $f : infile(\mathbb{Z})^n, z : outfile(\mathbb{Z})$

Ef : $f = f_0 \wedge \forall i \in [1..n]: f[i]\uparrow$

Uf : $z = \bigoplus_{e \in U_{i=1..n} f_0[i]} <e>$

az összefuttató felsorolásnak itt nem kell megadnia, hogy egy elem melyik felsorolás része, ezért nem kell a benne[] tömböt használni.

Algoritmus

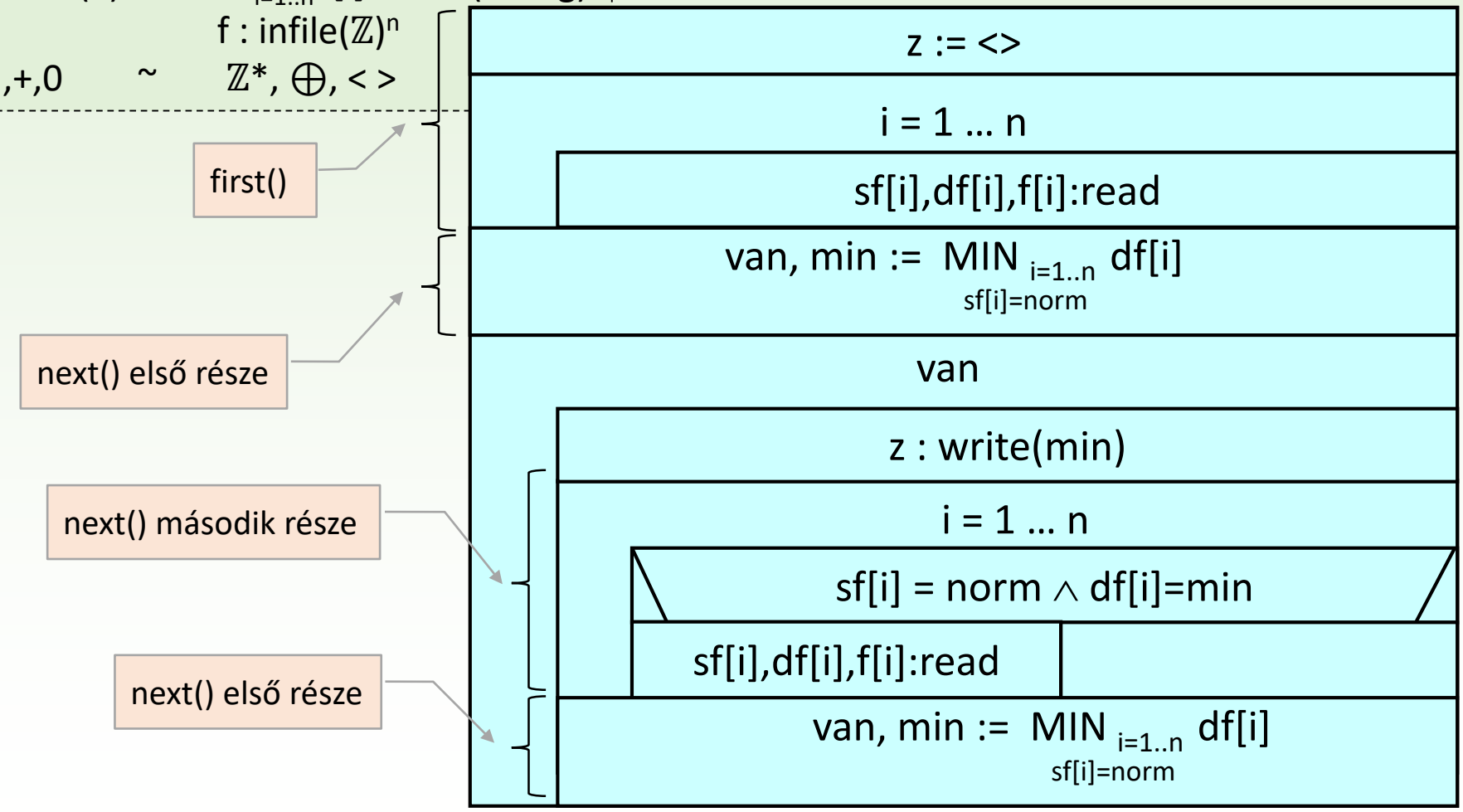
A szekvenciális inputfájlok összefuttató felsorolása az sf[i],df[i],f[i]:read olvasásokra épül.

Összegzés:

t:enor(E) ~ $U_{i=1..n}$ f[i] : enor(String)

f : infile($\mathbb{Z}$)^n

H,+,0 ~ $\mathbb{Z}^*$, $\oplus$, < >



first()

next() első része

next() második része

next() első része

Program

```
int main()
{
     string str;
     cout << "A leiro allomany neve: "; cin >> str;
     ifstream fname(str.c_str());
     if(fname.fail()) { cout << "Hiba\n"; return 1;}

     int n;
     fname >> n;
     vector<ifstream> f(n);
     for(int i = 0; i < n; ++i){
          fname >> str;
          f[i].open(str.c_str());
          if(f[i].fail()) { cout << "Hiba\n"; return 1;}
     }

     vector<int> df(n);
     vector<Status> sf(n);

     …
```

Program

```
    …

    vector<int> df(n);
    vector<Status> sf(n);
    ofstream z("output.txt");
    bool more;
    int min;
    for(int i = 0; i<n; ++i){
        read(f[i], df[i], sf[i]);
    }
    more = condmin(sf, df, min);
    while(more){
        z << min << endl;
        for(int i = 0; i<n; ++i){
            if(sf[i]==norm && df[i]==min) {
                read(f[i], df[i], sf[i]);
            }
        }
        more = condmin(sf, df, min);
    }
    return 0;
}
```