Egyedi felsorolók

Gregorics Tibor

gt@inf.elte.hu

http://people.inf.elte.hu/gt/oep

Felsorolások fajtái

- Eddig nevezetes gyűjtemények elemeinek szokásos, elvárt sorrendben történő, ún. standard felsorolásával találkoztunk. Ilyenkor a felsorolást végző felsoroló gyakran észrevétlen maradt, nem kell a felsorolónak az osztályát megadni, nem kell külön felsoroló objektumot példányosítani.
- Egyedi felsorolás az, amelyik
  - egy nevezetes gyűjteményt nem standard módon sorol fel. Például:
    - mátrix elemeit nem sorfolytonosan járja be
    - nem az elejéről indul, hanem egy korábban (a first() művelettel) megkezdett felsorolást folytat
    - korábban leáll, mielőtt a gyűjtemény összes elemét bejárná
    - egy lépéshez a gyűjtemény több elemét is felhasználja
  - egyszerre több gyűjtemény szimultán bejárására épül
  - egy speciális, esetleg virtuális gyűjteményt jár be

1.Feladat

Adott két függvény, az $f : \mathbb{N} \longrightarrow \mathbb{R}$ és $g : \mathbb{N} \longrightarrow \mathbb{R}$, továbbá egy $e \in \mathbb{R}$ szám. Tudjuk, hogy van olyan i és j természetes számpár, amelyre f(i)+g(j)=e teljesül. Adjunk meg ilyen i-t és j-t!

$A = (e:\mathbb{R} , i:\mathbb{N} , j:\mathbb{N})$

$Ef = (e = e_0 \wedge \exists i,j \in \mathbb{N} : f(i)+g(j)=e)$

$Uf = (Ef \wedge f(i)+g(j)=e)$

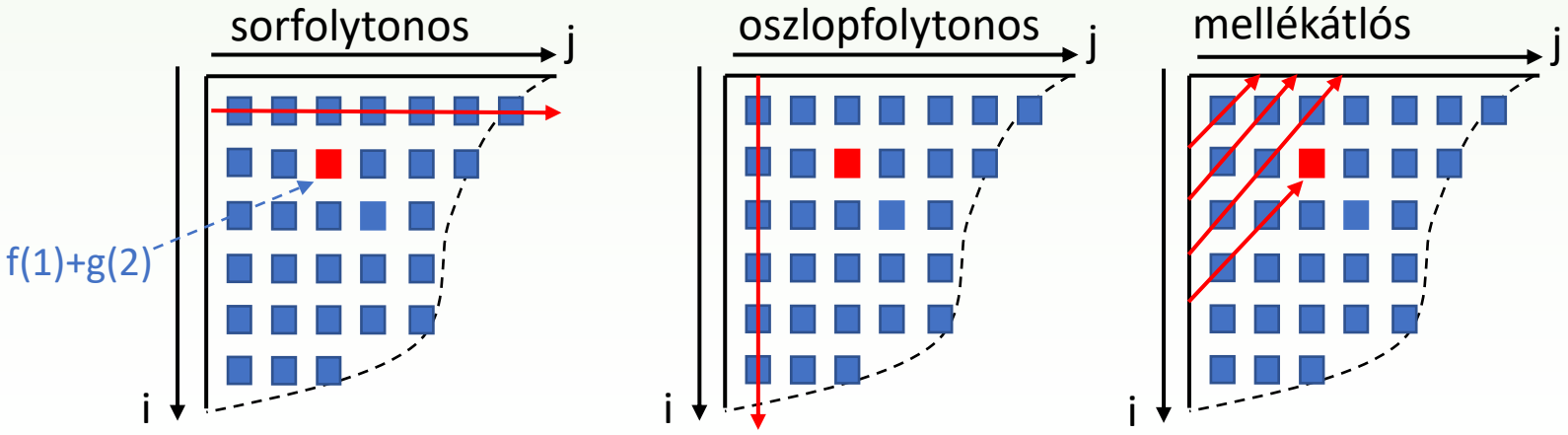
Ez a specifikáció nem sokat segít, nincs benne programozási tételre utaló jel.

Ötlet

Gondolatban rendezzük el az f(i)+g(j) összegeket egy végtelen kiterjedésű 0-tól kezdve indexelt mátrixban úgy, hogy e mátrix i-dik sorának j-edik eleme az f(i)+g(j) érték legyen. Ebben a mátrixban kell megkeresnünk egy adott (e) értéket.

Definiáljunk egy felsorolót, amelyik ennek a képzeletbeli mátrix elemeinek indexpárjait járja be.

A standard (sorfolytonos vagy oszlopfolytonos) felsorolás viszont nem lesz jó, mert a mátrix sorai is, oszlopai is végtelen hosszúak, így ezek a felsorolók csak az első sor vagy oszlop elemeit képesek bejárni.



Indexpárok felsorolója

A felsoroló mögött nem egy valódi, hanem egy virtuális gyűjtemény van.

Kell egy felsoroló, amely egy képzeletbeli végtelen kiterjedésű (0-tól kezdve indexelt) mátrix elemeinek sor-oszlop indexpárjait állítja elő.

$\mathrm{enor}(\mathbb{N}\times\mathbb{N})$

Számpárok végtelen sorozata. Az end() műveletet nem lehet definiálni, de szerencsére nem is lesz rá szükség.

$(\mathbb{N}\times\mathbb{N})^{\infty}$	first()	next()	current() : $\mathbb{N}\times\mathbb{N}$
i, j:$\mathbb{N}$	i, j:=0,0	**if** i>0 **then** i, j := i−1, j+1 **elsif** i=0 **then** i := j+1 ; j := 0	**return** (i, j)

Tervezés

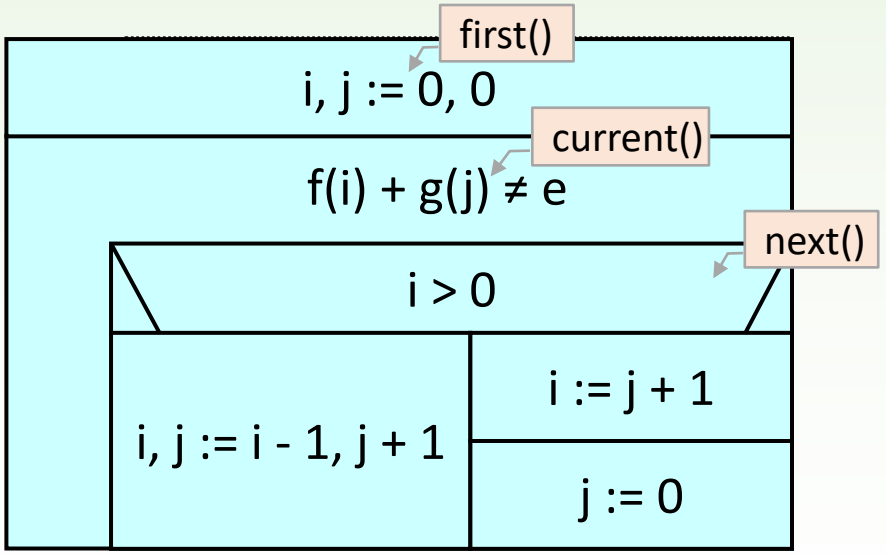
A = (t:enor($\mathbb{N}\times\mathbb{N}$), e:$\mathbb{R}$, i:$\mathbb{N}$, j:$\mathbb{N}$)

Ef = ($t = t_0 \wedge e = e_0 \wedge \exists i,j\in\mathbb{N}: f(i)+g(j)=e$)

Uf = ($e = e_0 \wedge (i,j) = \text{SELECT}_{(i,j)\in t_0}\ (f(i)+g(j)=e)$)

Kiválasztás:

t:enor(E)	~	t:enor($\mathbb{N}\times\mathbb{N}$)
elem, t.current()	~	(i,j)
felt(e)	~	f(i)+g(j) = e



Kiválasztás tesztelése

- A kiválasztás tesztelése a felsoroló tesztelésére korlátozódik. Nem releváns azonban a felsorolás hossza, illetve utolsó eleme szerinti tesztelés. Ezek helyett azokat az eseteket érdemes vizsgálni, amikor a keresett elem
  - az első, második, illetve sokadik indexpárhoz tartozik
  - az első, második, illetve sokadik „mellékátlóba” esik
  - egy „mellékátló” elején, végén, vagy közepén helyezkedik el

Program

```
#include "read.hpp"

bool all(double r) { return true; }

int main()
{
        double e = read<double>("Give a real number: ",
                                "This is not a real number!", all);

        int i, j;
        i = j = 0;
        while( f(i)+g(j)!=e ){
                if(i>0) { --i; ++j; }
                else    { i = j+1; j = 0; }
        }

        cout << "The given number is equal to the sum f("
                << i << ")+g(" << j << ")\n";

        return 0;
}
```

a harmadik paraméter egy logikai függvény, amely eldönti, hogy a paraméterének adott érték megfelelő-e.

Ez egy függvénysablon, amelyet itt valós szám beolvasására használunk, ezért a típust helyettesítő paraméterének a double-t adjuk meg.

Olvasó függvénysablon

```
#include <iostream>

template <typename Item>
Item read( const std::string &msg, const std::string &err, bool valid(Item))
{
      Item n;
      bool wrong;
      do{
            std::cout << msg;
            std::cin >> n;
            if((wrong = std::cin.fail())) std::cin.clear();
            std::string tmp = "";
            getline(std::cin, tmp);
            wrong = wrong || tmp.size()!=0 || !valid(n);
            if(wrong) std::cout << err << std::endl;
      }while(wrong);
      return n;
}
```

Függvénysablon, amely definíciójának egyik típusa egy később megadandó paraméter.

függvényt váró paraméter

Az Item helyébe olyan típust lehet írni, amelyre értelmezett az operator>>()

read.hpp

2.Feladat

Egy kiránduláson adott távolságonként (lépésenként) mértük a felszín tengerszint feletti magasságát (méterben), és az adatokat egy szekvenciális inputfájlban rögzítettük.
A kirándulás hány százalékában vezetett az út felfelé?

$A = (f:\text{infile}(\mathbb{Z}), v:\mathbb{R})$

$Ef = (f = f_0 \wedge |f_0| \geq 2)$

$Uf = (v = (100 \cdot \sum_{\substack{i=2\,..\,|f_0| \\ f_0[i] > f_0[i-1]}} 1) / (\sum_{i=2\,..\,|f_0|} 1))$

Itt a szekvenciális inputfájlnak egyszerre két egymás utáni elemére kell hivatkozni, amit a fájl standard felsorolása nem támogat.

Úgy kell a szekvenciális fájl elemeit felsorolni, hogy minden lépésben annak két közvetlenül egymás utáni elemét tudjuk elérni.

$A = (t:\text{enor}(\mathbb{Z}\times\mathbb{Z}), v:\mathbb{R})$

$Ef = (t = t_0 \wedge |t_0| > 0)$

$Uf = (v = (100 \cdot \sum_{\substack{(első,második)\in t_0 \\ első<második}} 1) / (\sum_{(első,második)\in t_0} 1))$

Pufferelt felsoroló

A felsorolás minden lépésben a soron következő két szomszédos elemet olvassa be a szekvenciális inputfájlból.

enor($\mathbb{Z}\times\mathbb{Z}$)

Olyan sorozat, amelynek elemei az eredeti inputfájl szomszédos elemeiből álló számpárok.

$(\mathbb{Z}\times\mathbb{Z})^*$	first()	next()	current() : $\mathbb{Z}\times\mathbb{Z}$	end() : $\mathbb{L}$
f: infile($\mathbb{Z}$) első, másod : $\mathbb{Z}$ st : Status	st,első,f : read st,másod,f : read	első := másod st,másod,f : read	**return** (első, másod)	**return** st=abnorm

Tervezés

A = (t: enor($\mathbb{Z}\times\mathbb{Z}$), v:$\mathbb{R}$)

Ef = ($t = t_0 \wedge |t_0|>0$)

Uf = ($v = (100\cdot\sum_{(első,másod)\in t_0,\ első<másod} 1) / (\sum_{(első,másod)\in t_0} 1)$)

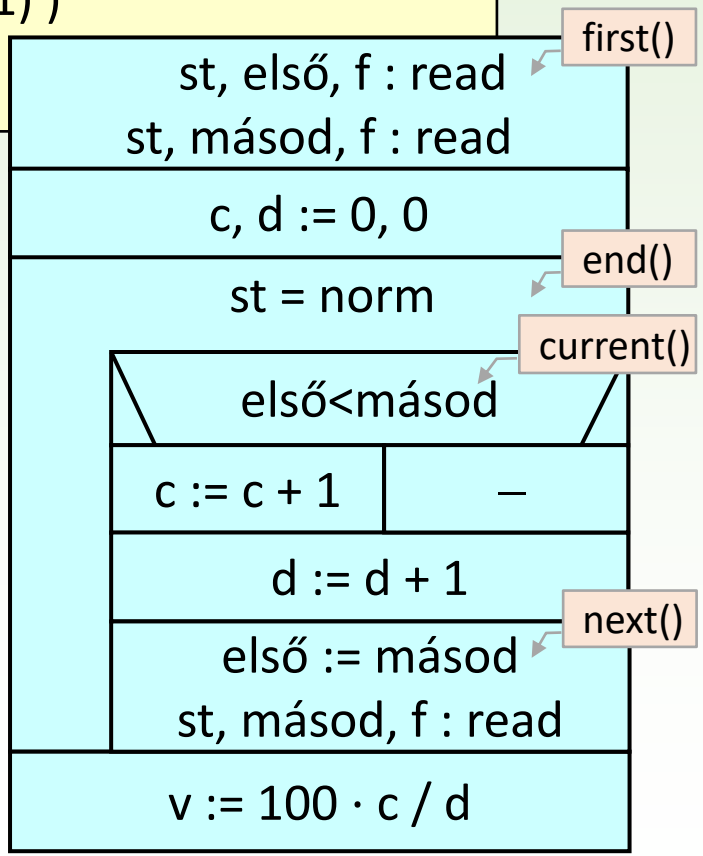
A számlálás és az összegzést közös ciklusba vonjuk össze.

Számlálás:

t:enor(E)	~	t:enor($\mathbb{Z}\times\mathbb{Z}$)
e	~	(első, másod)
felt(e)	~	első<másod

Összegzés:

t:enor(E)	~	t:enor($\mathbb{Z}\times\mathbb{Z}$)
e	~	(első, másod)
f(e)	~	1
H,+,0	~	$\mathbb{N}$,+,0



Számlálás és összegzés tesztelése

- A számlálás és összegzés ugyanazt a felsorolót használja, amelyet az alábbi szempontok szerint tesztelünk:
  - felsorolás hossza: egy, kettő vagy hosszabb (legyen végig emelkedő)
  - felsorolás eleje: csak az elején van egy emelkedés
  - felsorolás vége: csak a végén van egy emelkedés
- A számlálás eredménye szerinti tesztesetek:
  - nincs terep-emelkedés
  - egyetlen terep-emelkedés van
  - több terep-emelkedés van
- Az összegzés terheléses vizsgálata itt nem érdekes.

Program

```
int main()
{
	ifstream f("input.txt");
	if(f.fail()){
		cout << "Wrong file name!\n";
		return 1;
	}

	int first, second;
	int c = 0; int d = 0;
	for( f >> first >> second; !f.fail(); first = second, f >> second){
		if( first < second ) ++c;
		++d;
	}

	cout.setf(ios::fixed);
	cout.precision(2);
	cout << "Rate of the uphill part of the trip: " << (100.0*c)/d << "%" << endl;

	return 0;
}
```

```
f >> first >> second;
 while( !f.fail() )
	if( first < second ) ++c;
	++d;
	first = second; f >> second;
}
```

valós osztás, mert a számlálóból double típusú érték lett

3.Feladat

Egy kiránduláson adott távolságonként mértük a felszín tengerszint feletti magasságát, és az adatokat egy szekvenciális inputfájlban rögzítettük. Milyen hosszú volt a leghosszabb egybefüggően emelkedő szakasza a túrának?

A = (f:infile($\mathbb{Z}$), max:$\mathbb{N}$)

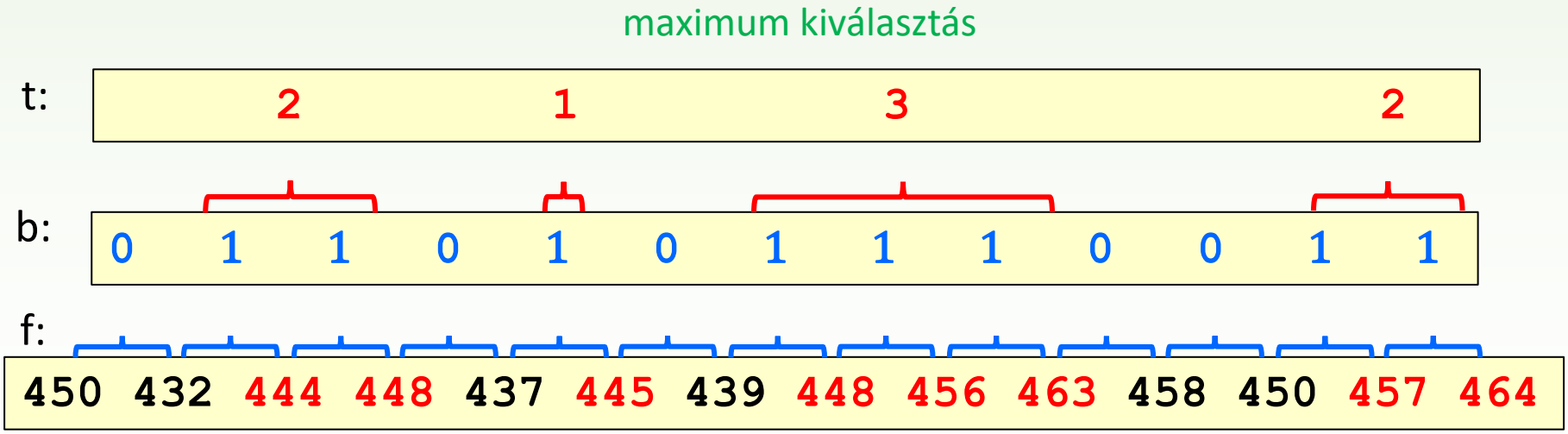
Ef = ($f = f_0 \wedge \exists i \in [1 .. |f|]: f[i] > f[i-1]$)

Uf = (MAX ?)

Egy maximum kiválasztásra lenne szükség, amely az egybefüggően emelkedő szakaszok hosszai között, de ezek a szakaszhosszok nem olvashatók ki közvetlenül az inputfájlból, ezért nehéz a feladatot specifikálni.

Absztrakciós szintek

- Könnyű lenne a feladat megoldása, ha az eredeti fájl adatai helyett a folyamatosan emelkedő szakaszok hosszait tudnánk felsorolni, mert ezen hosszok között egyszerűbb a legnagyobb értéket megtalálni.
- Az emelkedő szakaszok hosszainak megadásához viszont az eredeti fájl adatai helyett egyszerűbb volna csak azt látni, hogy mely lépések emelkedtek, melyek nem.



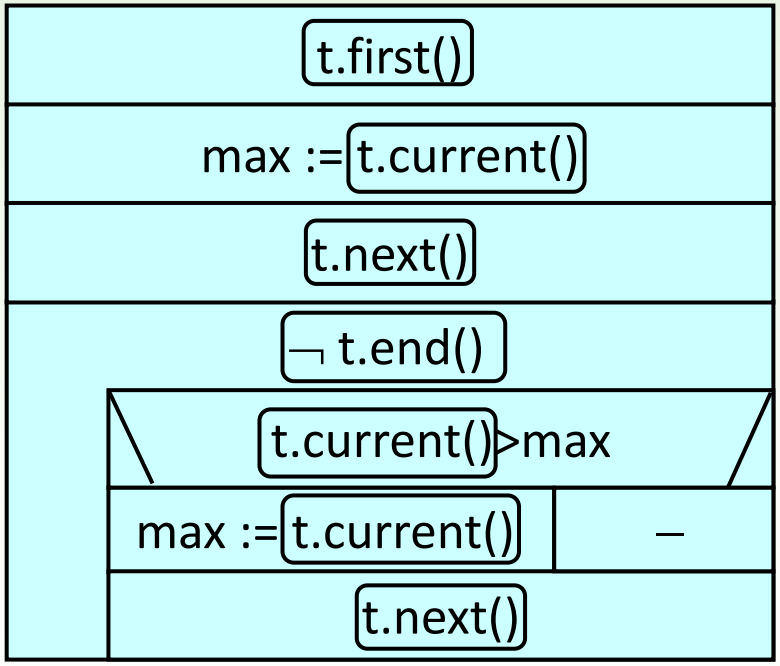
Tervezés

Tegyük fel, hogy van olyan felsorolónk, amely felsorolja az egybefüggően emelkedő szakaszok hosszait.

A = (t:enor($\mathbb{N}$), max:$\mathbb{N}$)

Ef = ($t = t_0 \wedge |t_0|>0$)

Uf = (max = $\mathbf{MAX}_{e \in t_0}$ e)



Maximum kiválasztás

t:enor(E)	~	t:enor($\mathbb{N}$)
f(e)	~	e
H, >	~	$\mathbb{N}$, >

Maximum kiválasztás tesztelése

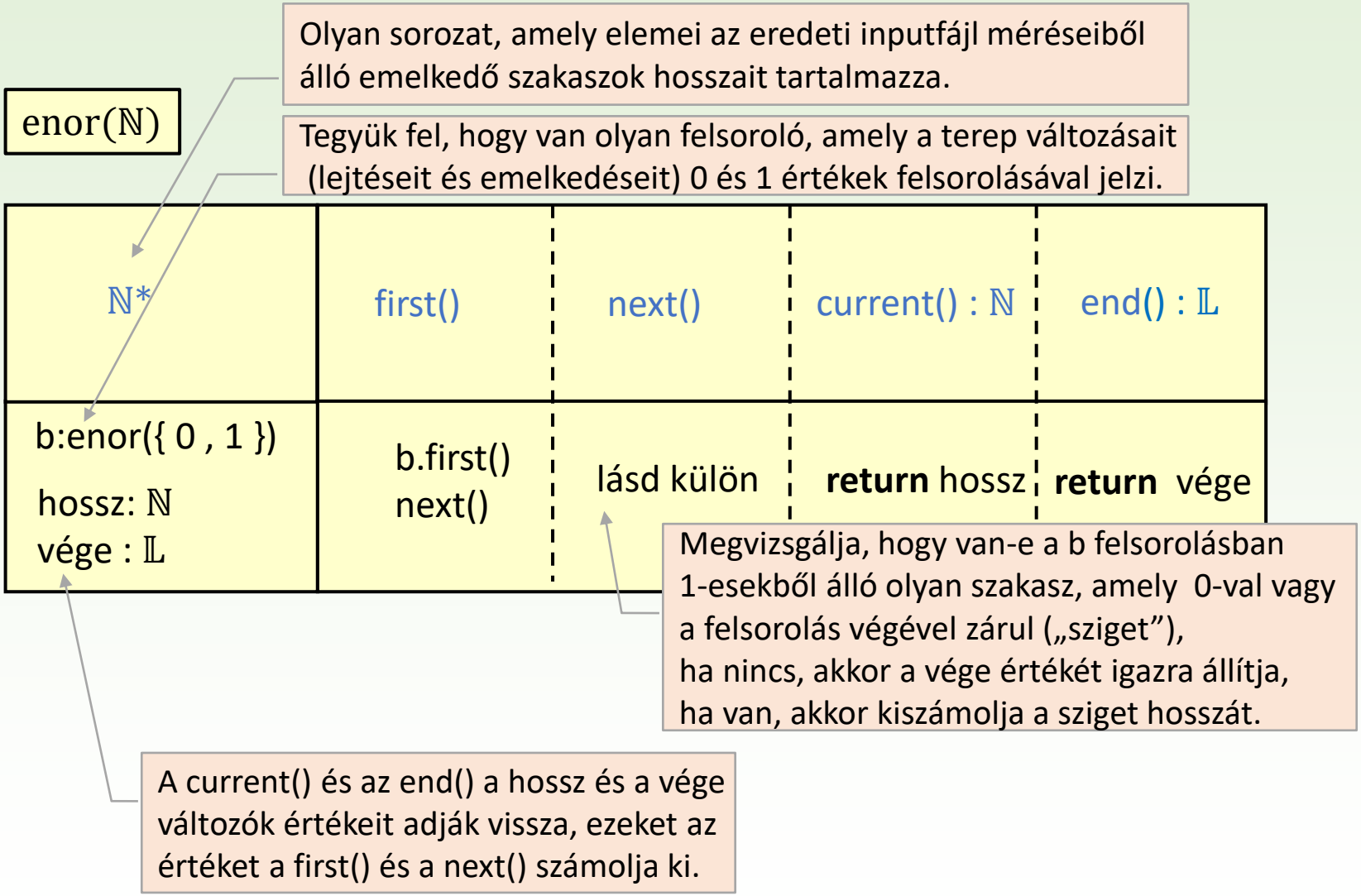
- Maximum kiválasztás felsorolója szerinti tesztesetek:
  - felsorolás hossza: egy, kettő vagy hosszabb
  - felsorolás eleje: legelső érték a legnagyobb
  - felsorolás vége: legutolsó érték a legnagyobb
- Maximum kiválasztás eredménye szerinti tesztesetek :
  - egyetlen legnagyobb érték
  - több egyformán legnagyobb érték

Főprogram

```
int main()
{
	try{
		LengthEnumerator t("input.txt");
		t.first();
		int max = t.current();
		for( t.next(); !t.end(); t.next() ){
			if( max < t.current() ) max = t.current();
		}
		cout << "The length of the longest uphill part: " << max << endl;
	} catch(Errors e) {
		if(FILE_ERROR==e) cout << "Wrong file name!\n" ;
	}
	return 0;
}
```

a megoldáshoz kitalált, a kirándulás emelkedő szakaszainak hosszát felsoroló objektum osztálya

Hosszúságok felsorolója



Next() művelet

Megadja a b felsorolás soron következő 1-esekből álló, 0-val vagy a felsorolás végével határolt szakaszának hosszát, és hogy van-e ilyen.

A = (b:enor({ 0 , 1 }), hossz:$\mathbb{N}$, vége:$\mathbb{L}$)

Ef = (b = b' $\wedge$ b' „folyamatban van")

Uf = ((l, elem, b") = $\text{SEARCH}_{e \in (b'.\text{current}(),\ b')}$ (e=1) $\wedge$ (vége=$\neg$l) $\wedge$

$\wedge$ ($\neg$vége $\rightarrow$ (hossz, b = $\sum_{e \in (b".\text{current}(),\ b")}^{e=1}$ 1)))

b' – a b változó kiinduló értéke
b" – a b változó értéke a 0-k átlépése után
b – a b változó értéke a megoldás végén

Keressük az első 1-es elemet.

Nem kell first(), a b' felsorolását korábban elkezdtük, az aktuális elemhez hozzáférünk.

Az összegzés addig tart, amíg $\neg$b.end() és b.current()=1.

Nem kell first(), hiszen a b" felsorolását már korábban elkezdtük, így hozzáférünk az aktuális eleméhez, sőt b".current() = 1.

Leálláskor a b-vel hivatkozott felsoroló még tartalmazhat további 0, 1 elemeket.

Az elem-re nincs szükségünk, hiszen $\neg$b".end() esetén ez a b".current(), amely ráadásul biztosan 1-es.

Next() művelet kicsit másképp

Megadja a b felsorolás soron következő 1-esekből álló, 0-val vagy a felsorolás végével határolt szakaszának hosszát, és hogy van-e ilyen.

A = (b:enor({ 0 , 1 }), hossz:$\mathbb{N}$, vége:$\mathbb{L}$)

b’ – a b változó kiinduló értéke
b” – a b változó értéke a 0-k átlépése után
b – a b változó értéke a megoldás végén

Ef = (b = b’ $\wedge$ b’ „folyamatban van”)

Uf = ((elem, b”) = SELECT$_{e \in (b'.current(),\ b')}$ (b”.end() $\vee$ e=1) $\wedge$ (vége=b”.end()) $\wedge$

Keressük az első 1-es elemet vagy a végét.

Nem kell first(), a b’ felsorolását korábban elkezdtük, az aktuális elemhez hozzáférünk.

$\wedge$ ($\neg$vége $\rightarrow$ ((hossz, b) = 1 + $\sum_{e \in b''}^{e=1}$ 1)))

Az összegzés addig tart, amíg $\neg$b.end() és b.current()=1.

Leálláskor a b-vel hivatkozott felsoroló még tartalmazhat további 0, 1 elemeket.

b”.current() = 1

$\neg$vége $\rightarrow$ ((hossz, b) = $\sum_{e \in (b''.current(),\ b'')}^{e=1}$ 1)

Az elem-re nincs szükségünk, hiszen $\neg$b”.end() esetén ez a b”.current(), amely ráadásul biztosan 1-es.

Specifikációs jelölések

Felsorolás végig:

t:enor(E)	i:[m .. n]	x:infile(E)
$r = \boxtimes_{e \in t_0} f(e)$	$r = \boxtimes_{i=m..n} f(i)$	$r = \boxtimes_{dx \in x_0} f(dx)$
$r, t = \boxtimes_{e \in t_0} felt(e)$	$r, i = \boxtimes_{i=m..n} felt(i)$	$r, (sx,dx,x) = \boxtimes_{dx \in x_0} felt(dx)$

Σ, MAX

SEARCH, SELECT

Az sx, dx, x (akárcsak i) változók, amelyeknek a feldolgozás végi értéke is hasznos lehet.

Felsorolás feltétel fenn állásáig:

t:enor(E)	i:[m .. n]	x:infile(E)
$\ldots = \boxtimes_{e \in t_0}^{tart(e)} f(e)$	$\ldots = \boxtimes_{i=m..n}^{tart(i)} f(i)$	$\ldots = \boxtimes_{dx \in x_0}^{tart(dx)} f(dx)$

$\neg t.end() \wedge tart(e)$

$i \leq n \wedge tart(i)$

$sx = norm \wedge tart(dx)$

Korábban abbahagyott felsorolás folytatása:

t:enor(E)	i:[m .. n]	x:infile(E)
$\ldots = \boxtimes_{e \in (t'.current(), t')} f(e)$	$\ldots = \boxtimes_{i=i'+1..n} f(i)$	$\ldots = \boxtimes_{dx \in (dx', x')} f(dx)$

Next() művelet

$$\text{elem, b''} = \text{SELECT}_{e \in (b'.\text{current}(),\ b')}\ (b.\text{end}() \vee e=1)$$

$$\wedge\ (\ \text{vége} = b''.\text{end}()\ )$$

$$\wedge\ (\ \neg\text{vége} \rightarrow ((\text{hossz},b) = \sum_{e \in (b''.\text{current}(),b'')}^{e=1} 1\ )))$$

Kiválasztás megkezdett felsorolóval

t:enor(E) ~ b:enor({ 0 , 1 })
first() nélkül

felt(e) ~ b.end() ∨ b.current()=1

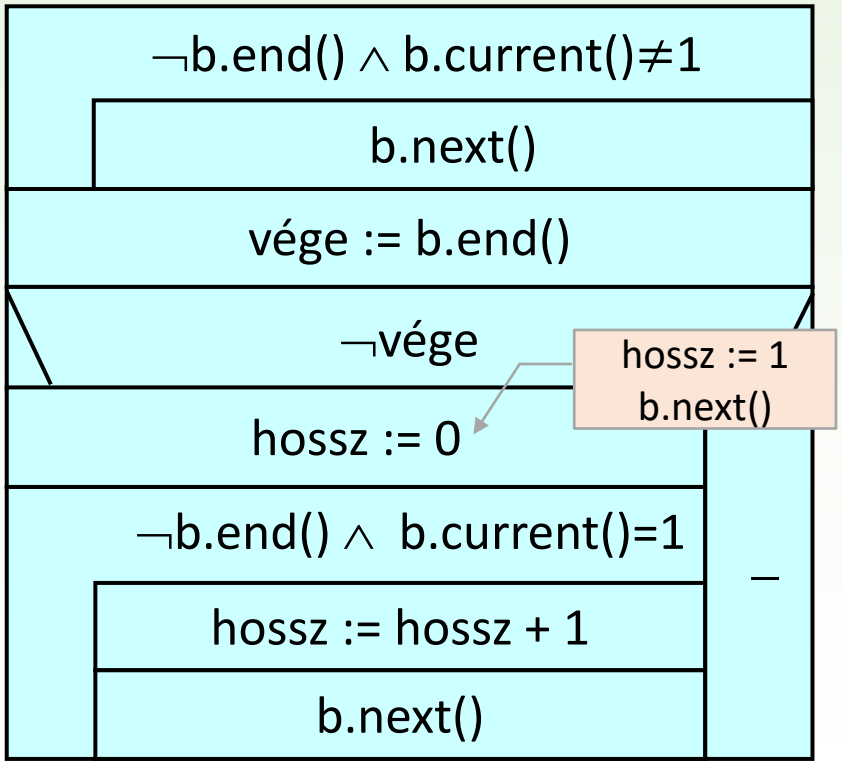
Feltételig tartó összegzés megkezdett felsorolóval

t:enor(E) ~ b:enor({ 0 , 1 })
first() nélkül
amíg b.current()=1

s ~ hossz

f(e) ~ 1

H,+,0 ~ ℕ,+,0



Next() szürkedoboz tesztelése

- Kiválasztás tesztesetei:
  - felsorolás hossza szerint: a fájl elején nulla, egy, vagy hosszabb nem emelkedő szakasz van
  - felsorolás eleje szerint: a fájl elején elején rögtön van emelkedés
  - felsorolás vége szerint: csak a fájl legvégén van egy lépéses emelkedés
  - A kiválasztás feltétele szerint: a fájlban nincs emelkedés illetve van

- Feltételig tartó összegzés tesztesetei:
  - felsorolás hossza szerint : a fájl elején nulla, egy, vagy hosszabb emelkedő szakasz van
  - felsorolás eleje szerint : csak a fájl elején van egy emelkedés
  - felsorolás vége szerint : csak a fájl végén van egy lépéses emelkedés
  - Az összegzés skálázása: itt nem érdekes

Hosszúságok felsoroló osztálya

```
class LengthEnumerator{
public:
	LengthEnumerator(const std::string &fname) : _b(fname){}
	void first() { _b.first(); next(); }
	int current()  const { return _length; }
	bool end()     const { return _end; }
	void next();
private:
	BitEnumerator _b;
	int _length;
	bool _end;
};
```

a megoldáshoz kitalált, a kirándulás emelkedő lépéseit 1-gyel, egyéb lépéseit 0-val helyettesítő felsorolás objektumának osztálya

```
void LengthEnumerator::next()
{
	for( ; !_b.end() && !_b.current(); _b.next() );
	if ( (_end = _b.end()) ) return;
	for( _length = 0 ; !_b.end() && _b.current(); _b.next() ) ++_length;
}
```

Lépések felsorolója

enor({ 0 , 1 })

Olyan sorozat, amely a kirándulás emelkedő lépéseit 1-gyel, egyéb lépéseit 0-val helyettesíti. A felsorolás hossza az eredeti inputfájl hosszánál eggyel kisebb (hiszen minden eleme az eredeti fájl két szomszédos mérésének felel meg.

{0, 1}*	first()	next()	current() : {0, 1}	end() : 𝕃
f: infile(ℤ) első, másod:ℤ st:Status	st, első, f : read st, másod, f : read	első:= másod st, másod, f : read	**if** első < másod **then return** 1 **else return** 0	**return** st=abnorm

Lépések felsoroló osztálya

```
enum Errors { FILE_ERROR };

class BitEnumerator{
public:
	BitEnumerator(const std::string &fname){
		_f.open(fname);
		if(_f.fail()) throw FILE_ERROR;
	}
	void first() { _f >> _first >> _second; }
	void next()  { _first = _second; _f >> _second; }
	int current() const { return (_first < _second ? 1 : 0); }
	bool end()    const { return _f.fail(); }
private:
	std::ifstream _f;
	int _first, _second;
};
```

feltételes kifejezés:
ha _first < _second fenn áll.
akkor az értéke 1, különben 0