

Egyedi felsorolók

Felsorolások fajtái

- ❑ Eddig nevezetes gyűjtemények elemeinek szokásos, elvárt sorrendben történő, ún. **standard felsorolásával** találkoztunk. Ilyenkor a felsorolást végző felsoroló gyakran észrevétlen maradt, a megvalósításban nem kellett a felsoroló osztályát megadni.
- ❑ **Egyedi felsorolás** az, amelyik
 - nem a szokásos módon járja be a feldolgozandó gyűjtemény elemeit, vagy
 - egy lépésben a gyűjtemény több elemét is felhasználja, vagy
 - több feldolgozandó gyűjtemény elemeit futtathatja össze
- ❑ **Egyedi felsorolásnak tekinthetjük azt is, ha egy standard felsorolás**
 - nem a `first()` művelettel indul, mert már korábban valahogyan a „folyamatban van” állapotba került, vagy
 - korábban áll le, minthogy a gyűjtemény elemei elfogynának, vagy
 - nincs a felsorolás háttérében valódi gyűjtemény

1.Feladat

Adott két függvény, az $f : \mathbb{N} \rightarrow \mathbb{R}$ és $g : \mathbb{N} \rightarrow \mathbb{R}$, továbbá egy $e \in \mathbb{R}$ szám. Tudjuk, hogy van olyan i és j argumentum, 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 nyom.

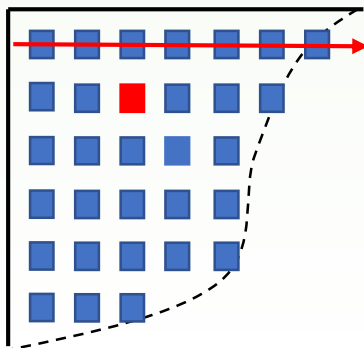
Ötlet

Gondolatban rendezzük el az $f(i)+g(j)$ értékeket egy végtelen kiterjedésű 0-tól kezdődő indexelésű mátrixba úgy, hogy a mátrix i -dik sorának j -edik eleme az $f(i)+g(j)$ érték legyen.

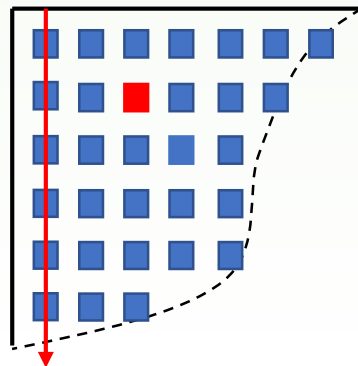
Definiáljunk egy felsorolót, amelyik ennek a képzeletbeli mátrix elemeinek indexpárjait járja be.

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

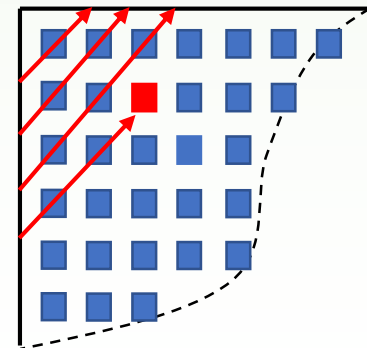
sorfolytonos



oszlopfolytonos



mellékátlós



Tervezés

Ezen felsoroló mögött nincs gyűjtemény, csak egy képzeletbeli mátrix, amelynek indexeit járja be nem a szokott módon.

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

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

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

Kiválasztás:

$t:\text{enor}(E) \sim t:\text{enor}(\mathbb{N} \times \mathbb{N})$

$e \sim (i, j)$

$\text{felt}(e) \sim f(i) + g(j) = e$

$t.\text{first}()$

$\neg \text{felt}(t.\text{current}())$

$t.\text{next}()$

$i, j := t.\text{current}()$

Indexpárok felsorolója

A felsorolással úgy teszünk, mintha egy létező mátrix soron következő sor és oszlop indexét állítanánk elő.

$\text{enor}(\mathbb{N} \times \mathbb{N})$	Olyan sorozat, amelynek elemei index párok.		
$(\mathbb{N} \times \mathbb{N})^*$	<code>first()</code>	<code>next()</code>	<code>current()</code>
$i, j: \mathbb{N}$	$i, j := 0, 0$	if $i > 0$ then $i, j := i - 1, j + 1$ elsif $i = 0$ then $i := j + 1 ; j := 0$	(i, j)

`end()` művelet nem kell

Tervezés

Ezen felsoroló mögött nincs gyűjtemény, csak egy képzeletbeli mátrix, amelyet nem a szokott módon jár be.

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

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

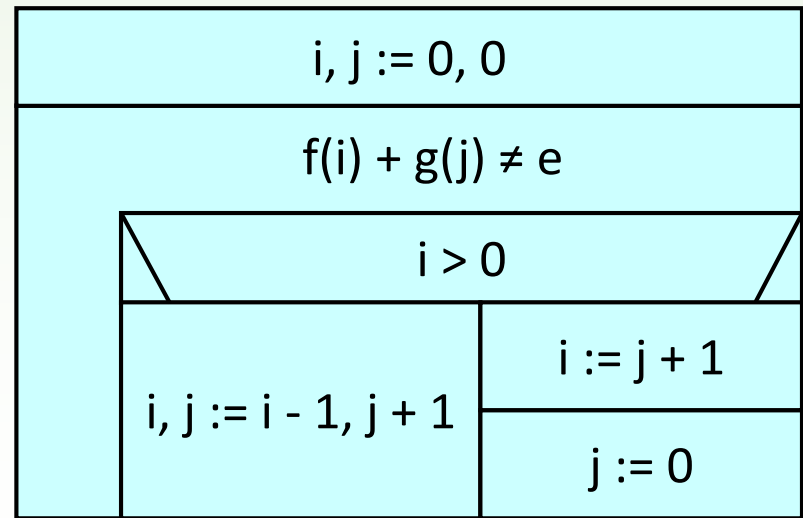
$Uf : e = e_0 \wedge (i, j) = \text{SELECT}_{(i, j) \in t_0} (f(i) + g(j) = e)$

Kiválasztás:

$t:\text{enor}(E) \sim t:\text{enor}(\mathbb{N} \times \mathbb{N})$

$e \sim (i, j)$

$\text{felt}(e) \sim f(i) + g(j) = e$



Kiválasztás tesztelése

- A kiválasztás tesztelése magának a felsorolónak a tesztelésével azonos.
 - A **felsorolás hossza** most mindig végtelen hosszú, de azt vizsgálhatjuk, hogy a keresett elem a felsorolásban az első, második, illetve sokadik indexpárra jelenik-e meg, esetleg ezt a vizsgálatot a mellékátlókra levetítve azt tesztelhetjük, hogy melyik átlóban van a keresett elem: első, második, sokadik.
 - Azt is vizsgálhatjuk, hogy megfelelően működik-e a keresés, ha a keresett elem egy **mellékátlónak az elején, végén**, vagy közepén van.
 - A **felsorolás eleje** szerint: $f(0)+g(0)=e$

Program

```
int main()
{
    bool all(double r) { return true; }

    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;
}
```

függvénysablon

Olvasó függvénysablon

```
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;
}
```

az operator>>-nak az Item típusra értelmezettnek kell lennie

2.Feladat

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

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

$Ef : f = f_0 \wedge |f_0| \geq 2$

$Uf : v = \left(\sum_{\substack{i=2 \dots |f_0| \\ f_0[i] > f_0[i-1]}} 1 \right) / \left(\sum_{i=2 \dots |f_0|} 1 \right)$

Egyszerre kell két egymás utáni elemre hivatkozni, de a szekvenciális fájl olvasással történő bejárása ezt nem támogatja.

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

$Ef : t = t_0 \wedge |t_0| > 0$

$Uf : v = \left(\sum_{\substack{(\text{első}, \text{másod}) \in t_0 \\ \text{első} < \text{másod}}} 1 \right) / \left(\sum_{(\text{első}, \text{másod}) \in t_0} 1 \right)$

Az lenne jó, ha a szekvenciális fájl helyett lenne egy olyan felsorolónk, amelyik a szekvenciális fájl közvetlen egymás utáni elempárjait tudná felsorolni.

Pufferelt felsoroló

A felsorolással úgy teszünk, mintha minden lépésben a soron következő két szomszédos elemet olvasnánk be a szekvenciális inputfájlból.

$\text{enor}(\mathbb{R} \times \mathbb{R})$

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

$(\mathbb{R} \times \mathbb{R})^*$	<code>first()</code>	<code>next()</code>	<code>current()</code>	<code>end()</code>
<code>f: infile(\mathbb{R}) első, másod : \mathbb{R} st : Status</code>	<code>st, első, f : read st, másod, f : read</code>	<code>első:= másod st, másod, f : read</code>	<code>(első, másod)</code>	<code>st =abnorm</code>

Tervezés

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

$Ef : t = t_0 \wedge |t_0| > 0$

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

A számlálás és az összegzést közös ciklusba vonjuk össze.

Számlálás:

$t: \text{enor}(E) \sim t: \text{enor}(\mathbb{R} \times \mathbb{R})$
 $e \sim (\text{első}, \text{másod})$
 $\text{felt}(e) \sim \text{másod} > \text{első}$

Összegzés:

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

$st, \text{első}, f : \text{read}$
 $st, \text{másod}, f : \text{read}$

$c, d := 0, 0$

$st = \text{norm}$

$\text{első} < \text{másod}$

$c := c + 1$

—

$d := d + 1$

$\text{első} := \text{akt}$
 $st, \text{másod}, f : \text{read}$

$v := 100 \cdot c / d$

Számlálás és összegzés tesztelése

- ❑ A számlálás és összegzés ugyanazt a felsorolót használja. E szerint egyben vizsgálható a
 - 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 szerint:
 - 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";
        exit(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;
}
```

The diagram illustrates the expansion of the `for` loop and a note about the division operation. An arrow points from the `for` loop's body to a box containing the expanded loop logic. Another arrow points from the `(100.0*c)/d` expression to a box explaining that it is a floating-point division because the numerator is a `double`.

```
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?

$A : f:\text{infile}(\mathbb{R}), \text{max}:\mathbb{N}$

$Ef : f = f_0 \wedge$

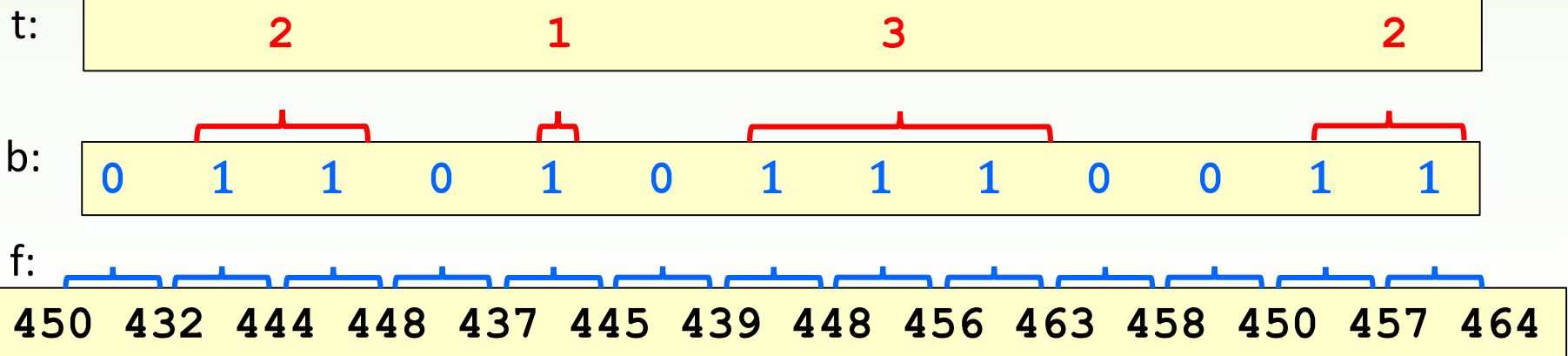
$\exists i \in [1 .. |f|] : f[i] > f[i-1]$

$Uf : ?$

Egy maximum kiválasztást kellene specifikálni az egybefüggően emelkedő szakaszok hosszai között, de ezek nem olvashatók ki közvetlenül az inputfájlból, ezért a feladatot nehéz precízen specifikálni.

Ötlet

- Milyen jó lenne, ha az eredeti fájl adatai helyett egyből az egybefüggően emelkedő szakaszok hosszait sorolnánk fel! Ezek között már könnyű lenne a legnagyobbat megtalálni.
- Az emelkedő szakaszok hosszainak megadásához viszont az eredeti fájl adatai helyett elég volna azt látni, hogy mely lépések emelkedtek, melyek nem.
- Ezen jelzéseket az eredeti fájl alapján könnyű felsorolni.



Tervezés

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

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

$Ef : t = t_0 \wedge |t_0| > 0$

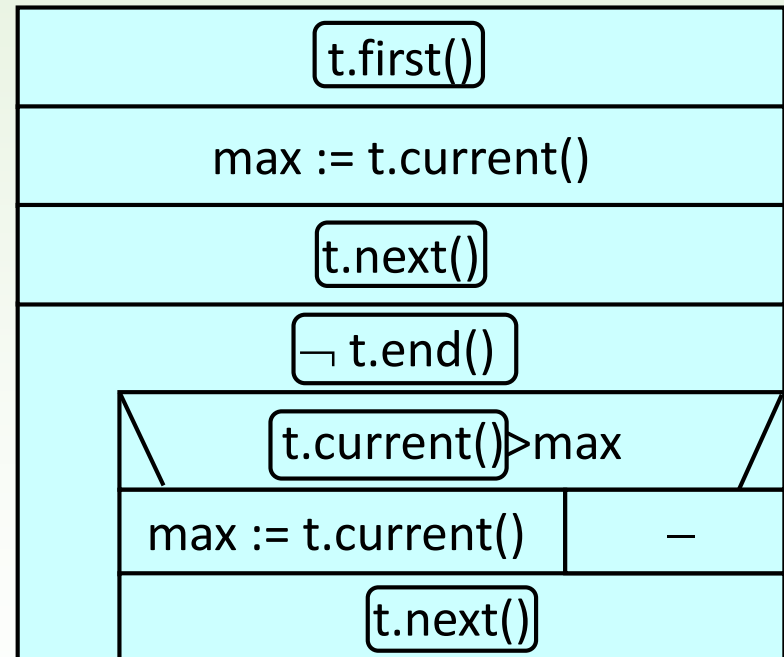
$Uf : \text{max} = \mathbf{MAX}_{e \in t_0} e$

Maximum kiválasztás

$t:\text{enor}(E) \sim t:\text{enor}(\mathbb{N})$

$f(e) \sim e$

$H, > \sim \mathbb{N}, >$



Maximum kiválasztás tesztelése

❑ Maximum kiválasztás felsorolója szerint:

- felsorolás hossza: egy, kettő vagy hosszabb
- felsorolás eleje: legelső érték a legnagyobb
- felsorolás vége: legutolsó érték a legnagyobb

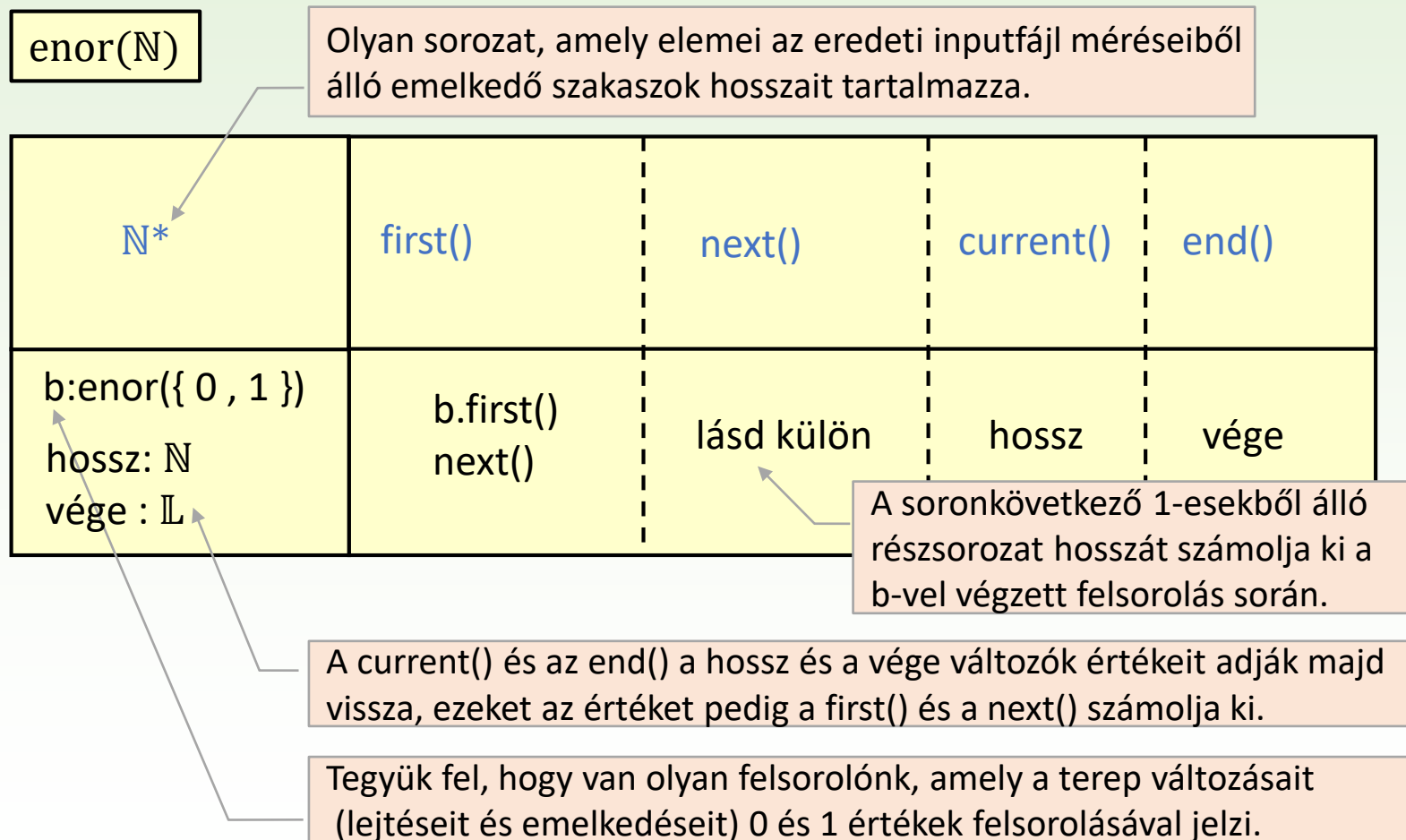
❑ Maximum kiválasztás eredménye szerint:

- Egyetlen legnagyobb érték
- Több egyformán legnagyobb érték

Főprogram

```
int main()
{
    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;
    return 0;
}
```

Hosszúságok felsorolója



Next() művelet

Megszámolja, hogy a 0 és 1-esek már elkezdett felsorolásában milyen hosszú a soron következő 1-esekből álló szakaszt, feltéve, hogy van ilyen.

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

$Ef : b = b' \wedge b' \text{ „folyamatban van”}$

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

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

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

Leálláskor a b felsoroló még tartalmazhat elemeket

Amikor találunk 1-esekből álló szakaszt, akkor $e'' = b''.\text{current}() = 1$ és $\neg b''.\text{end}()$.

b' – a b változó kiinduló értéke

b'' – a b változó értéke a 0-k átlépése után

Keresi a b' felsorolás soron következő 1-esekből álló szakaszának elejét vagy a felsorolás végét.

Ez a jelölés ($e \in b'$ helyett) arra utal, hogy a $b'.\text{current}()$ értékét közvetlenül – a $b.\text{first}()$ alkalmazása nélkül – elérjük, hiszen a b' felsorolása már folyamatban van.

A soron következő 1-esekből álló szakasz hosszát olyan összegzés adja meg, amely addig tart, amíg $e = 1$ (itt $e = b.\text{current}()$).

Egy már folyamatban levő felsorolást folytatunk: az $e \in (e'', b'')$ arra utal, hogy a $b''.\text{current}()$ értékét a $b.\text{first}()$ alkalmazása nélkül elérjük.

Specifikációs jelölések

Felsorolás végig:

az $sx, dx, x : \text{read}$
(sx, dx) párokat sorol,
de $(sx, dx) \in x_0$ helyett
ezt a $dx \in x_0$ jelöli

$t:\text{enor}(E)$ Σ, MAX	$h:\text{set}(E)$	$i:[m \dots n]$	$x:\text{infile}(E)$
$r = \boxtimes_{e \in t_0} f(e)$	$r = \boxtimes_{e \in h_0} f(e)$	$r = \boxtimes_{i=m..n} f(i)$	$r = \boxtimes_{dx \in x_0} f(dx)$
$r, t = \boxtimes_{e \in t_0} \text{felt}(e)$	$r, h = \boxtimes_{e \in h_0} \text{felt}(e)$	$r, i = \boxtimes_{i=m..n} \text{felt}(i)$	$r, (sx, dx, x) = \boxtimes_{dx \in x_0} \text{felt}(dx)$

SEARCH, SELECT

Az sx, dx, x (akárcsak h vagy i) változók, amelyek feldolgozás végi értéke is fontos lehet.

Felsorolás feltétel fenn állásáig:

$t:\text{enor}(E)$	$h:\text{set}(E)$	$i:[m \dots n]$	$x:\text{infile}(E)$
$\dots = \boxtimes_{e \in t_0}^{\text{tart}(e)} f(e)$	$\dots = \boxtimes_{e \in h_0}^{\text{tart}(e)} f(e)$	$\dots = \boxtimes_{i=m..n}^{\text{tart}(i)} f(i)$	$\dots = \boxtimes_{dx \in x_0}^{\text{tart}(dx)} f(dx)$
$\neg t.\text{end()} \wedge \text{tart}(e)$	$h \neq \emptyset \wedge \text{tart}(e)$	$i \leq n \wedge \text{tart}(i)$	$sx = \text{norm} \wedge \text{tart}(e)$

Korábban abbahagyott felsorolás folytatása:

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

Next() művelet

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

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

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

Kiválasztás megkezdett felsorolóval

$t:\text{enor}(E) \sim b:\text{enor}(\mathbb{L})$

first() nélkül

$\text{felt}(e) \sim b.\text{end}() \vee b.\text{current}()=1$

Feltételig tartó összegzés

megkezdett felsorolóval

$t:\text{enor}(E) \sim b:\text{enor}(\mathbb{L})$

first() nélkül

amíg $b.\text{current}()=1$

$s \sim \text{hossz}$

$f(e) \sim 1$

$H, +, 0 \sim \mathbb{N}, +, 0$

$\neg b.\text{end}() \wedge b.\text{current}() \neq 1$

$b.\text{next}()$

$\text{vége} := b.\text{end}()$

$\neg \text{vége}$

$\text{hossz} := 0$

$\neg b.\text{end}() \wedge b.\text{current}()=1$

$\text{hossz} := \text{hossz} + 1$

$b.\text{next}()$

Next() szűrkedoboz tesztelése

❑ Kiválasztás megkezdett felsorolóval:

- felsorolás hossza: nulla, egy, vagy hosszabb nem emelkedő az elején
- felsorolás eleje: az elején rögtön van emelkedés
- felsorolás vége: csak a végén van egy emelkedés
- A kiválasztás feltétele: nincs emelkedés illetve van

❑ Feltételig tartó összegzés megkezdett felsorolóval:

- felsorolás hossza: nulla, egy, vagy hosszabb emelkedő az elején
- felsorolás eleje: csak az elején van egy emelkedés
- felsorolás vége: csak a végén van egy emelkedés
- Az összegzés terhelése: 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;  
};
```

```
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, amelynek hossza az eredeti inputfájl hosszánál eggyel kisebb, mert minden eleme az eredeti fájl két szomszédos mérésének felel meg. Egy elem értéke 1, ha a megfelelő a szomszédos mérések emelkedést mutatnak, különben 0.

<div>$\{0, 1\}^*$</div>	<div>first()</div>	<div>next()</div>	<div>current()</div>	<div>end()</div>				
<div>f: infile(\mathbb{R}) első, másod:\mathbb{R} st:Status</div>	<div>st, első, f : read st, másod, f : read</div>	<div>első:= másod st, másod, f : read</div>	<div><table><tr><td colspan="2">első < másod</td></tr><tr><td>1</td><td>0</td></tr></table></div>	első < másod		1	0	<div>st = abnorm</div>
első < másod								
1	0							

Lépések felsoroló osztálya

```
class BitEnumerator{
public:
    enum Errors { FILEERROR };
    BitEnumerator(const std::string &fname){
        _f.open(fname);
        if(!_f.fail()) throw FILEERROR;
    }
    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;
};
```