3. gyakorlat

Tematika: lengyel forma / lengyel forma kiértékelés, quick sort megbeszélése, lejátszása

## Lengyel forma[1]

Infix, prefix és postfix alak megbeszélése. Mi a gond az infix alakkal? Lengyel forma: egy aritmetikai kifejezés postfix alakja. Jellemzői:

- Nincsenek benne zárójelek, a kiértékelés mégis egyértelmű, és könnyen elvégezhető,
- Operandusok sorrendje nem változik, az infix kifejezéshez képest,
- Operátorok sorrendje: az elvégzésük sorrendjében szerepelnek,
- Minden operátort közvetlen megelőznek az operandusai. Az operandus lehet változó, konstans, de lehet postfix kifejezés is.

| infix kifejezés | lengyel forma (postfix alak) | Megjegyzés |
|---|---|---|
| a+b | ab+ | műveleti jel az operandusai mögött áll |
| a+b*c | abc*+ | műveletek rangsorának hatása: (prec(*) > prec(+) |
| a*b+c | ab*c+ | műveletek rangsorának hatása: (prec(*) > prec(+) |
| a*(b+c) | abc+* | zárójelezés felülbírálhatja a műveletek rangsorát |
| a/b*c | ab/c* | azonos rangú műveletek általában balról jobbra sorrendben végzendők el |
| a^b^c | abc^^ | a fenti szabály alól akad néhány kivétel, például az egymást követő hatványozás sorrendje jobbról balra értendő |

**Feladat:**

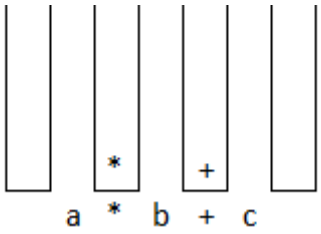
A) ( a + b ) * ( c – d ) / f ^ ( g – h ) + j – l – i kifejezés lengyel formára hozása. (Próbáljuk meg közösen felírni, még nem az algoritmust használva.)
B) Értékadó operátor hatása, Hova illik az értékadó operátor?
x = ( a + b ) * ( c – d ) / f ^ ( g – h ) + j – l – i

**Megoldás:**

**x** a b + c d - * f g h - ^ / j + l – i - =

## Lengyel formára hozás verem segítségével – bemutatás példákon keresztül

**1. a * b + c**

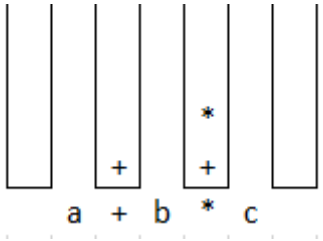



**Kimenet:** a b * c +

[1] Az eredeti prefix jelölési formát, **Jan Łukasiewicz** lengyel matematikus javasolta 1920-ban, később az ausztrál filozófus, **Charles Leonard Hamblin** javasolta a postfix alakot (1950), melyet emiatt „fordított lengyel formának" is szokás nevezni. (forrás: wikipedia)

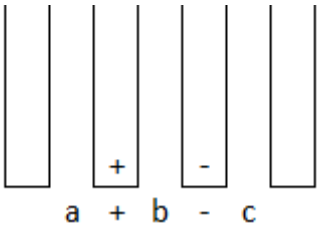
# 3. gyakorlat

**2. a + b * c**




**Kimenet:** a b c * +

**3. a + b – c**




**Kimenet:** a b + c –

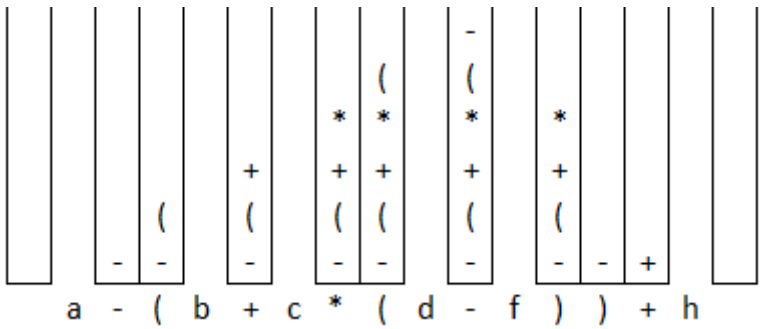
**Precedencia hatása:**

- Minden beolvasott műveleti jel bekerül a verembe, hogy „megvárja”, míg az operandusai kiíródnak, de előtte a veremben várakozó műveleti jelek vizsgálata történik:
- ha azonos rangú a beolvasott és a verem tetején lévő műveleti jel, kiírjuk a veremben lévőt (balról jobbra sorrend esetén) – 3. példa,
- ha a veremben magasabb prioritású művelet szerepel, mint ami bekerülne, kiírjuk – 1. példa,
- ha a verem tetején alacsonyabb rangú van, mint az olvasott, akkor bekerül a verembe – 2. példa.

**Feladat:**

a – ( b + c * ( d – f ) ) + h kifejezés lengyel formára hozása verem segítségével. A verem tartalmát folyamatosan tartsuk nyilván!

**Megoldás:**




Elvégzett verem műveletek
v.push(„-”); v.push(„(”); v.push(„+”);
v.push(„*”); v.push(„(”); v.push(„-”);
v.pop(); // Hívások száma: 2
// Első „(” jelig.
v.pop(); // Hívások száma: 3
// Első „(” jelig.
v.pop(); v.push(„+”); v.pop();

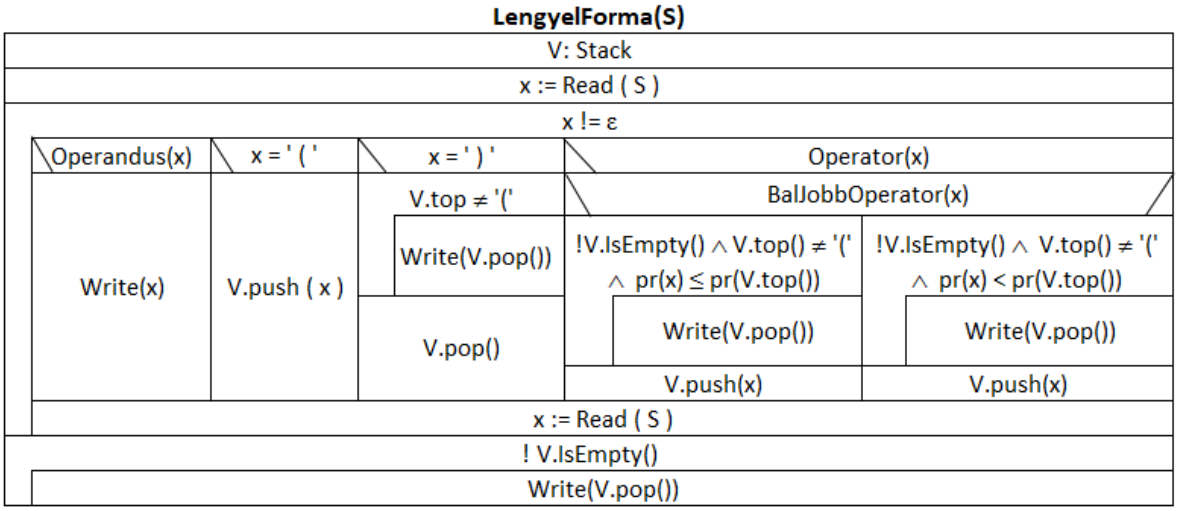
**Kimenet:** a b c d f - * + - h +

Ha időnk engedi, kidolgozhatunk a csoporttal egy kicsit bonyolultabb kifejezést, például:

**x = a + ( - b ^ c ^ 2 + d * e ) / ( ( f + g ) * h / - k ) - p * z**

**Algoritmus:**

Bemenet: egy helyesen zárójelezett kifejezés: S




**Megemlítendő:**

- Vannak jobbról balra sorrendű operátorok, ezek feldolgozása hogyan történik?
  y = x = a ^ b ^ c
- Gondoljuk meg, hogy az egy operandusú operátorok (pl. negatív előjel –a^b, vagy ++i*x) hogyan illeszthetők be a lengyel formába?
- Egyszerű függvények bevonása. pl: x = z * sin(y/w)^2
- Javasoljuk a hallgatóknak, hogy keressék meg az interneten a C++ nyelv operátorait és precedenciájukat, például: http://www.cplusplus.com/doc/tutorial/operators/

## Lengyel forma kiértékelése

**Feladat:**

Hozzuk lengyel formára a következő kifejezést, majd verem segítségével értékeljük ki:
( a + b ) * c – d

**Lengyel forma: a b + c * d –**

Tegyük fel, hogy a változók az alábbi értékkel rendelkeznek, számítsuk ki a kifejezés értékét a lengyel formából!
a = 2, b = 4, c = 3, d = 1

**Kiértékelés:**

| | | | | | | |
|---|---|---|---|---|---|---|
| | 4 | | 3 | | 1 | |
| 2 | 2 | 6 | 6 | 18 | 18 | **17** |
| 2 | 4 | + | 3 | * | 1 | - |

# 3. gyakorlat

**Az algoritmus:**

Bemenet: egy lengyel formájú kifejezés: S

**lengyel_kiertekeles(S)**

| | | |
|---|---|---|
| V:Stack; x:=read(S) | | |
| x ≠ ε | | |
| | Operator(x) | Operandus(x) |
| | jobb:=V.pop() | V.push(x) |
| | bal:=V.pop() | |
| | V.push(bal ⊗ jobb) | |
| | x:=read(S) | |
| write(V.pop()) | | |

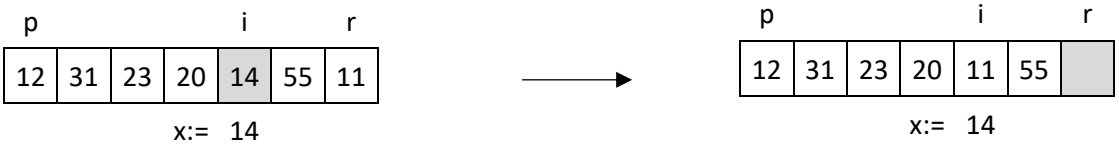
S - lengyel formájú kifejezés

Megjegyzések az Operator(x) ághoz:

- bal ⊗ jobb jelölés: elvégezzük az x műveletet,
- feltettük, hogy az operárorok két operandusúak, de ez könnyen kiterjeszthető egy oprandusú műveletekre
- az algoritmus kiszámíthatja az értéket, vagy fordító program esetén generálhatja a kiszámítás kódját.

## Quick sort

1. lépés: pivot elem véletlenszerű kiválasztása (indexe: i). Kiírjuk a választott elemet az x segédváltozóba, majd a résztömb utolsó elemét az i-edik helyre másoljuk. A tömbrész végén egy „lyuk”-at képzünk.

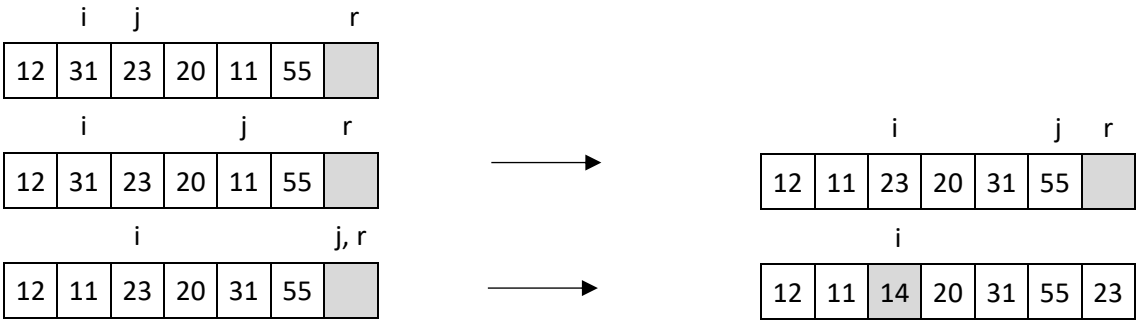



2. lépés: p-ről elindítva i-t, megkeressük az első olyan elemet, ami nagyobb, mint a pivot elem. i az algoritmus során mindig egy olyan elemre fog mutatni, amelyről tudjuk, hogy nagyobb, mint a pivot elem (ha van ilyen), és garantált, hogy előtte a pivotnál kisebbegyenlő értékek vannak. Ha nem találunk a pivot elemnél nagyobbat, akkor az azt jelenti, hogy a pivot elemnek választott elem a legnagyobb, ekkor a pivot elemet betesszük a tömb végén lévő lyukba, és vége a particionálásnak.

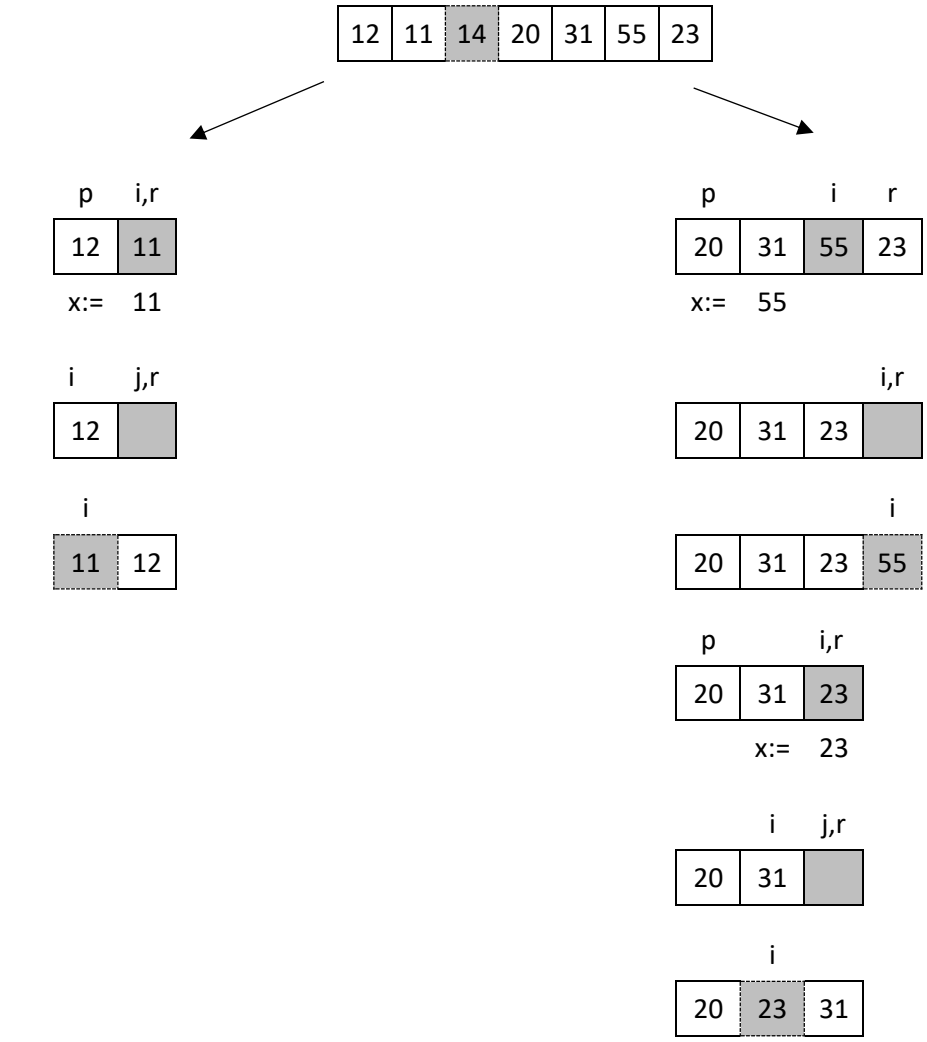
3. lépés: egy másik változóval, j-vel az i utáni elemről indulva lépegetünk. Ha j-vel egy a pivot elemnél kisebb elemhez érünk, felcseréljük az i és j indexű elemeket. i-t ilyenkor eggyel tovább léptetjük, majd j-vel folytatjuk a tömb bejárását. A p..i-1 elemek kisebbegyenlők a pivotnál, i..j-1 elemek pedig nagyobbegyenlők. i tehát mindig a vízválasztó index.

Végül j-vel elérünk az r indexig, ami a résztömb végét jelenti. r-et már nem vizsgáljuk, mert ott a „lyuk” van! Mivel i az első olyan elem indexe, ami nagyobb a pivotnál, azt kell az r indexű helyre beírni, a pivot elemet pedig betsszük az i-edik indexre. A p..r tömbrész ketté osztása az i indexnél történt, ezzel tér vissza az algoritmus.

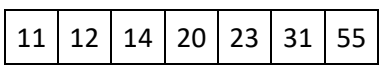
# 3. gyakorlat



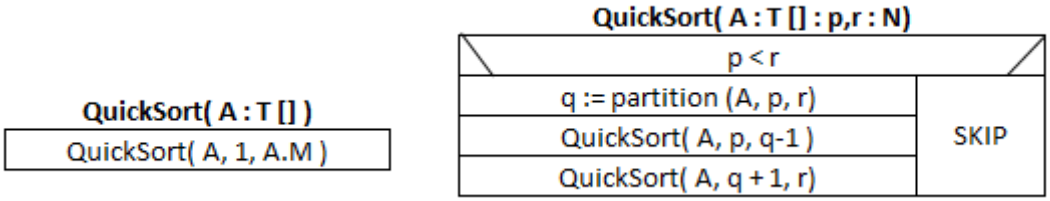

Folytatódik a particionálás a p..i-1 és i+1..r tömbrészen:

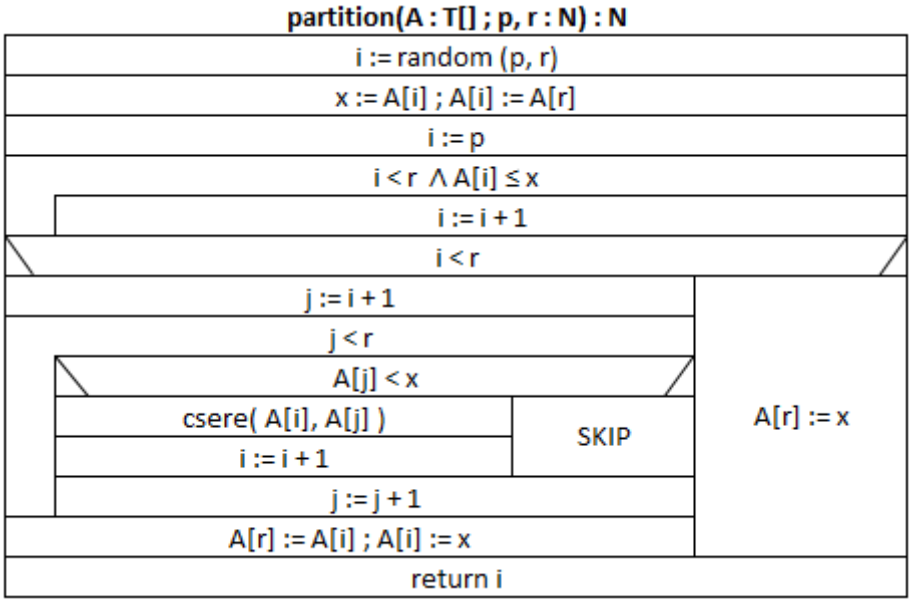



A rendezett tömb.



**Az algoritmus (előadáson szerepel):**




## Javasolt házifeladatok:

Egyet érdemes feladni az (1)-(3) feladatokból, jobb csoportoknál fel lehet adni a (4)-est is.

(1) Teljesen és helyesen zárójelezett kifejezésből hogyan állítható elő a lengyel forma.

(2) Teljesen és helyesen zárójelezett kifejezésből hogyan értékelhető ki verem segítségével a kifejezés.

(3) Lengyel formából hogyan állíthatjuk elő a teljesen zárójelezett alakot.

(4) A quick sort algoritmusban a particionáló eljárásnak számos változata van. Egy érdekes megoldás a következő:

- A tanult algoritmushoz hasonlóan kiválasztjuk a strázsa elemet, majd a tömb rész végén képzünk egy lyukat, ugyanúgy ahogy a tanult algoritmusban láttuk.
- Ez a lyuk fog vándorolni. Amikor hátul van a lyuk, akkor elölről indulva, a még nem vizsgált elemek között keresünk egy olyat, amelyik nagyobb, mint a strázsa. Ha találunk, akkor azt áthelyezzük a lyukba, így a lyuk elölre kerül.
- Most hátulról indulva, a még nem vizsgált elemek közül keresünk egy olyat, amelyik a strázsa elemnél kisebb. Ha találunk, akkor azt betesszük a lyukba, így a lyuk ismét hátulra kerül.
- Akkor van vége, ha a kereső ciklus változója elérkezik a lyukig, Ekkor a strázsa elem épp a lyukba illik.

Készítsék el a felvázolt algoritmust!

3. gyakorlat

# Megoldások:

(1) Teljesen és helyesen zárójelezett kifejezésből a lengyel forma előállítása.

**lengyelforma(Inp,Out)**

| V:Stack; x:=read(Inp) | | | |
|---|---|---|---|
| x ≠ ε | | | |
| x = '(' | Operandus(x) | Operator(x) | x = ')' |
| skip | Write(Out,x) | V.push(x) | Write(Out,V.pop()) |
| x:=read(Inp) | | | |

Inp - egy teljesen és helyesen zárójezett aritmetikai kifejezés

Out - a kifejezés lengyel formája

(2) Teljesen és helyesen zárójelezett kifejezésből hogyan értékelhető ki verem segítségével a kifejezés.

**Kiertekeles_teljesenzarojelezettbol(Inp)**

| V:Stack; W:Stack; z:=read(Inp) | | | |
|---|---|---|---|
| z ≠ ε | | | |
| z = '(' | Operandus(z) | Operator(z) | z = ')' |
| skip | V.push(z) | W.push(z) | jobb:=V.pop() |
| | | | bal:=V.pop() |
| | | | x:=W.pop() |
| | | | V.push(bal ⊗ jobb) |
| z:=read(Inp) | | | |
| Write(V.pop()) | | | |

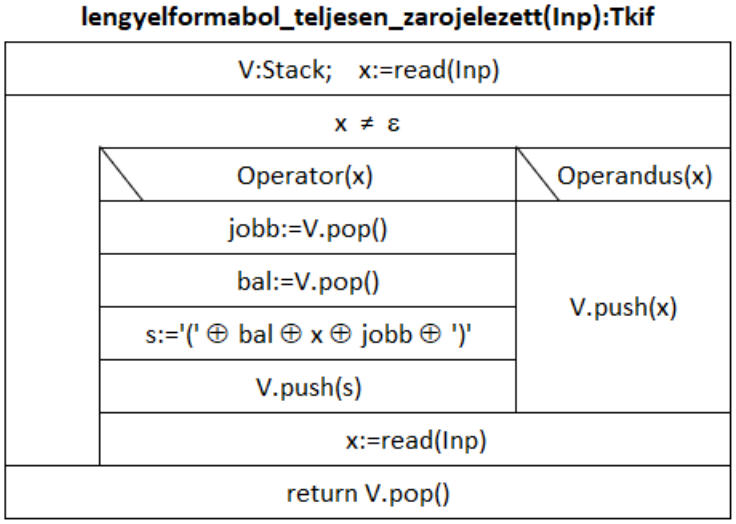
V - operandusokat tároló verem

W - operátorokat tároló verem

Inp - egy teljesen és helyesen zárójelezett kifejezés.

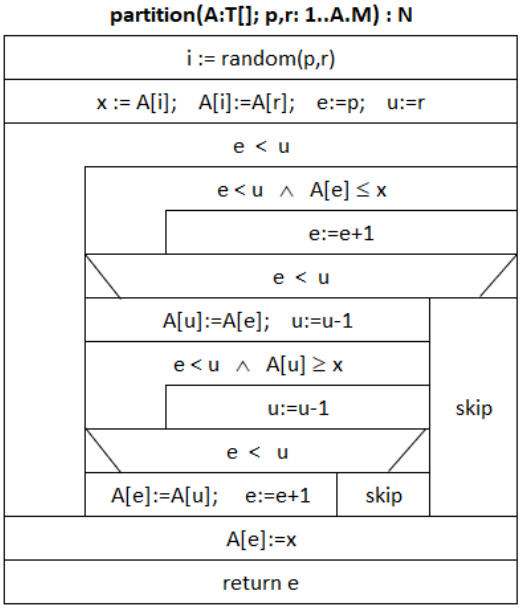
z=')' ágra ugyanazok a megjegyzések vonatkoznak, mint a lengyel forma kiértékelő algoritmusra

(3) Lengyel formából hogyan állíthatjuk elő a teljesen zárójelezett alakot.




Inp - egy lengyel formájú kifejezés.
Eredményül ennek a kifejezésnek a teljesen zárójelezett alakját kapjuk (Tkif ezt jelöli).
⊕szimbólum az összefűzést jelöli.

(4) quick sort particionáló algoritmus




"hátul" képzünk egy lyukat, e-elölről lépeget, u-hátulról

hátul van a lyuk (az u indexen), elölről keresünk a lyukba illő elemet

ha találtunk,

betesszük a lyukba, így elöl keletkezik a lyuk (az e indexen)

most hátulról indulva keresünk egy a lyukba illő elemet

ha találtunk,

betesszük a lyukba, így a lyuk megint hátul lesz (az u indexen)

végül e és u egyenlő lesz, így mindkettő a lyuk helyét jelenti

*Készült az „Integrált kutatói utánpótlás-képzési program az informatika és számítás-tudomány diszciplináris területein” című EFOP 3.6.3-VEKOP-16-2017-00002 azonosítójú projekt támogatásával.*