Programozási alapismeretek előadásjegyzet

Kiss-Bartha Nimród

2023. október 16.

Tartalomjegyzék

Előszó

Ez a jegyzet azért készült, hogy megkönnyítse a *programozási alapismeretek* (vagy közismerten *progalap*) tantárgyra való felkészülést.

Magát a jegyzetet elsősorban azért készítettem, hogy könnyebben elérhető formában, egyetlen helyen meg lehessen találni a tantárgyhoz szükséges elméleti és gyakorlati ismereteket, különös hangsúlyt fektetve a gyakorlati órák követelményeire. Kiemelten ajánlom azoknak, akik soha életükben nem programoztak vagy nem is hallottak pl.: a programozási tételekről – ám bízom benne, hogy a tapasztaltabbak számára is segítséget nyújthat az anyag gyors átböngészésében.

Tartozik hozzá egy *feladatgyűjtemény* is, ami a prezentációkban szereplő feladatokat vezeti le, valamint tartalmaz extra feladatokat is, amiket anno a gyakorlaton vettünk.

Annak ellenére, hogy a 2022/2023/1. félév diasorai alapján állítottam össze a jegyzetet, önmagában **nem helyettesíti a hivatalosan elérhető anyagot**, pusztán segítséget nyújthat az abban való elmélyülésben. A prezentációk mellett felhasználtam más forrásokat is, ezek listáját hátul láthatod.

A jegyzetet ugyan a legjobb tudásom szerint állítottam össze, ám tévedések egészen biztosan lesznek benne (helyesírási hibák, rossz magyarázat, tömb indexében i helyett j-nek kéne szerepelnie, stb.). Ha ilyet találsz, kérlek jelezd nekem ezt e-mailben a `email.address@gmail.com` címen.

Kellemes tanulást és sikeres félévet kívánok!

Nimród

1. fejezet

Specifikáció és struktogram

Ez a fejezet az 1-2. előadás diasorainak tartalmát foglalja össze, illetve rendszerezi.

A programozás folyamatát nem kezdhetjük el anélkül, hogy megértenénk a problémát és megterveznénk az optimális, ideális megoldását. Ennek megragadásához két eszköz áll a rendelkezésünkre: a *specifikáció* és a *struktogram*.

1.1. Specifikáció

Specifikáció

A specifikáció a feladat formális megragadását jelenti. Gyakorlati értelemben egy interface a megbízó és a fejlesztő között. Jellemző rá, hogy

- „egyértelmű”, pontos, teljes
- (formalizált $\rightarrow$) tömör
- érthető, szemléletes

Ezek a tulajdonsáok gyakran ellentmondanak egymásnak.

1.1.1. Leírási eszközök

- szöveges leírás (pszeudokód)
- matematikai megoldás

1.1.2. Összetevői

1. *Definíció* (def) – opcionális, olyankor alkalmazzuk, amikor a bemeneti adatokkal valamilyen műveletet, relációt, tulajdonságunk akarunk tömören és meghivatkozhatóan megfogalmazni, pl.: függvények
2. **Bemenet** (be) – amely adatokkal dolgoznunk kell
3. **Kimenet** (ki) – amely adato(ka)t / eredmény(eke)t meg szeretnénk kapni
4. **Előfeltétel** (ef) – ismeretek a bemenetről, relációk köztük (amolyan kikötések)
5. **Utófeltétel** (uf) – az eredményt meghatározó állítás(ok)

1.1.3. Fajtái

- Feladatspecifikáció
- Programspecifikáció* (később lesz róla szó bővebben)
- Séma* (nem specifikáció a szó szoros értelmében)

1.2. Algoritmus

A következő *elemi* tevékenységekből épül fel: **értékadás**, **beolvasás**, **kiírás**.

1.2.1. Összeállítási módjai

1. **Szekvencia** – egymás utáni végrehajtás
2. **Elágazás** – választás két vagy több tevékenységből
3. **Ciklus** – ismétlés adott darabszámszor vagy adott feltételtől függően
4. **Alprogram** – egy összetett tevékenység, egyedi néven (absztrakció)

1.2.2. Algoritmusleíró nyelvek

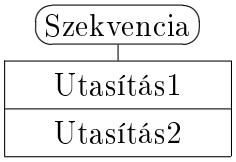
- Szöveges leírás
  - Mondatokkal leírás
  - Mondatszerű elemekkel – **pszeudokód**
- Rajzos leírás
  - Folyamatábra (flowchart)
  - **Struktogram** (Nassi–Shneiderman diagram)

1.3. Struktogram

Szekvenciák, elágazások, ciklusok és alprogramok, vagyis az algoritmusok szemléletes és nyelvfüggetlen ábrázolására használt segédeszköz.

A fejezetben elhelyeztem az algoritmusokat pszeudokódban is, mégha nem is fektetünk rá nagy hangsúlyt. A pszeudokódnál ara kell odafigyelni, hogy a kulcsszavakat félkövéren jelöljük, valamint egyes kulcsszavak nagybetűvel kezdődnek.

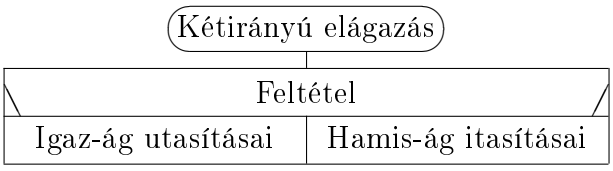
Elementárisan téglalapokból áll – ezek határozzák meg az utasításokat.




```
Utasítás01
Utasítás02
```

1.1. ábra. Szekvencia

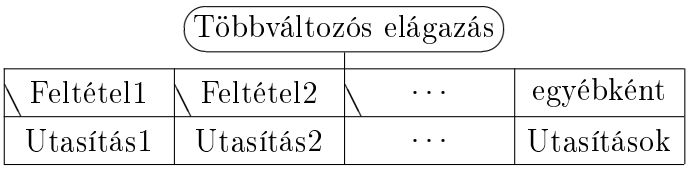
Az elágazások bal alsó sarkába kerül az igaz ág, a jobb alsóba meg a hamis ága – melyek alá rendre 1-1 új utasítás jelenik meg.




```
Ha Feltétel akkor
    Igaz-ág utasításai
Különben
    Hamis-ág utasításai
Elágazás vége
```

1.2. ábra. Kétirányú elágazás

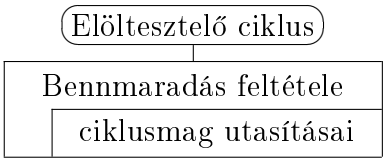
Megjegyzés: a többváltozós elágazás esetén az *egyébként*-ágban (ha van) általában a másik (jobb) sarokba helyezzük el a vonást. Itt az eltérést technikai korlátok okozták.




```
Elágazás
    Feltétel01 esetén Utasítások01
    Feltétel02 esetén Utasítások02
    ...
    egyéb esetekben Utasítások
Elágazás vége
```

1.3. ábra. Többirányú elágazás

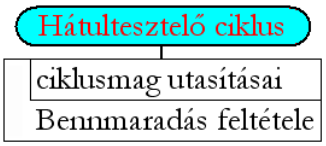
Az elöltesztelő ciklusnak a ciklusmagját felülről, . . .




```
Ciklus amíg Feltétel
    ciklusmag utasításai
Ciklus vége
```

1.4. ábra. Elöltesztelő ciklus

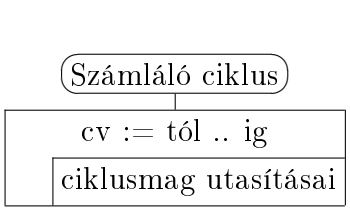
. . . a hátultesztelősnek meg alulról keretezi be a feltétel a bal oldalról. Az ábra eltérőségét a többitől szintén technikai limitációk okozták.

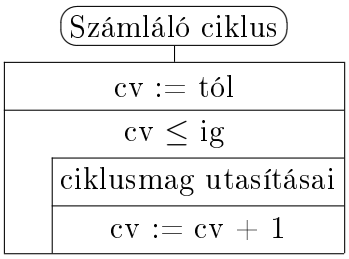



```
Ciklus
    ciklusmag utasításai
amíg Feltétel
Ciklus vége
```

1.5. ábra. Hátultesztelő ciklus

A számláló ciklus feltétele jellemzően így néz ki: `i := elejétől..végéig`, ahol az `i` a *ciklusváltozó* (vagy néha *futóindex*) szerepét tölti be. Pl.: `i := 1 .. n`






```
Ciklus cv=tól-tól ig-ig
    ciklusmag utasításai
Ciklus vége
```

1.6. ábra. Számláló ciklus

Alapelv a specifikáció és a struktomgram használatának megértéséhez: a specifikációban *deklaráljuk* a változókat, rekordokat, függvényeket; a struktogramban *felhasználjuk* és *elhelyezzük* őket az algoritmusban.

1.4. Jelölési konvenciók

Azon „összetevőket”, melyek léteznek a specifikációban és a struktogramban egyaránt (típusok, változók, függvények, stb.), szükséges őket jelölnünk valahogy.[1]

Összetevők	**Specifikáció**	**Struktogram**
változók	$var \in \mathbb{H}$,	var
	$var : \mathbb{H}$	
tömbök	$arr_{1..n} \in \mathbb{H}^n$,	$arr[n]$
	$arr_{1..n} : \mathbb{H}^n$	
dinamikus tömbök	$darr_{1..} \in \mathbb{H}^*$	$darr[], darr := ()$
tömb i-edik eleme	arr_i , $arr[i]$	$arr[i]$
rekordok	$tipus := rec(elem_1 \times \ldots \times elem_n)$,	$osszetett.elem_1$,
	$osszetett \in tipus^n$;	$osszetett.elem_n$
függvények	$f : \mathbb{H}_1 \to \mathbb{H}_2$	–
	$f(x) :=$ függvénytörzs	$f(x)$
	$g : \mathbb{H}_1 \times \mathbb{H}_2 \to \mathbb{H}_g$	–
	$g(x, y) :=$ függvénytörzs	$g(x, y)$
értékadás	$:=$	$:=$
egyenlőségvizsgálat	$=$	$==$, $=$
logikai „és”	$\wedge$, és	&& , és
logikai „vagy”	$\vee$, vagy	|| , vagy
negálás	$\neg$, nem	! , nem

[1] Valójában nincsenek kőbe vésve, hogy az egyes elemeket hogyan kell jelölni, párhuzamosan létezik sok jelölési rendszer – amik sokszor ellentmodanak más rendszereknek. A lényeg, hogy következetesek maradjunk.

Függvények – megjegyzés

Ha közelebbről megvizsgáljuk a függvények definícióját a specifikációban, akkor azt vehetjük észre, hogy kísértetiesen hasonlítanak arra, ahogyan a Haskellban definiáljuk őket.

$$f : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{L}$$
$$f(x, p, q) := (p \leq x) \wedge (q \geq x)$$

```
f :: Int -> Int -> Int -> Bool
f x p q = (p <= x) && (q >= x)
```

Figyeljük meg, hogy míg a matematikai jelölésben $\times$-szel „kötjük össze” a függvény argumentumait és a $\to$ után kezdődik a függvény visszatérési értéke, addig a Haskellban mind a két esetben `->` jelölést alkalmazunk.

1.5. Tanulságok (a feladatokból)

- Ha az utófeltételben $\exists$, $\forall$, vagy $\sum$ jel van, akkor a megoldás mindig **ciklus**!
- Ha az utófeltételben $\exists$ vagy $\forall$ jel van, akkor a megoldás sokszor **feltételes ciklus**!
- Ha az utófeltételben $\sum$ jel van, akkor a megoldás sokszor **számlálós ciklus**! ($\prod$ is...)
- Feltételes $\sum$ esetén a **ciklusban elágazás** lesz.

2. fejezet

Programozási tételek

Ez a fejezet a 3-4. előadás diasorainak tartalmát foglalja össze, illetve rendszerezi.

A programozási tételek **cél**ja, hogy bizonyíthatóan helyes sablont nyújtanak, amelyekre magasabb szinten lehet építeni a megoldást, így a fejlesztés gyorsabbá és biztonságosabbá válik. Érdemes megjegyezni, hogy mindegyiknél a bemenet legalább egy **sorozat** (tömb, halmaz... kontextustól függ).

Szerkezete: absztrakt feladatspecifikáció, absztrakt algoritmus

Felhasználásának menete:

1. a konkrét feladat specifikálása
2. a specifikációban a programozási tétekek megsejtése
3. a konkrét feladat és az absztrakt feladat paramétereinek egymáshoz rendelése
4. a konkrét algoritmus „generálása” a megsejtett programozási tételek absztrakt algoritmusok alapján, a 3. lépés szerint átparaméterezve
5. „hatékonyabbá tevés” programtranszformációkkal

Csoportosításuk:

sorozat → érték	sorozat → sorozat	sorozat → sorozatok	sorozatok → sorozat
összegzés, megszámlálás, maximumkiválasztás, eldöntés, keresés, kiválasztás	másolás, kiválogatás	szétválogatás	halmazos tételek, pl.: metszet, unió (róluk később lesz szó)

2.1. Elemi programozási tételek

Ezek a tételek a **sorozat** $\rightarrow$ **érték** kategóriába tartoznak.

2.1.1. Összegzés / Sorozatszámítás

Objektíva: n valamiből kell kiszámolni 1 valamit.
Kapcsolódó jelölések: $\sum$ (szumma), $\prod$ (produktum), $\cup$ (unió), & (konkatenáció)

Itt látható a sorozatszámítás *általánosítása*, ahol az F függvényt kicserélhetjük a 4 operátor valamelyikjére.

- Bemenet: $N \in \mathbb{N},\ X_{1..N} \in \mathbb{H}^N_{(1)}$
- Kimenet: $S \in \mathbb{H}_{(2)}$
- Előfeltétel: $-$
- Utófeltétel: $S := F(X_{1..N})$
- Definíció: $F : \mathbb{H}^*_{(1)} \rightarrow \mathbb{H}_{(2)}$

$$F(X_{1..N}) := \begin{cases} F_0 & \text{ha } N = 0 \\ f(F(X_{1..N-1}), X_N) & \text{ha } N > 0 \end{cases}$$

$f : \mathbb{H}_{(2)} \times \mathbb{H}_{(1)} \rightarrow \mathbb{H}_{(2)},\ F_0 \in \mathbb{H}_{(2)}$

$S := F_0$
$i := 1..N$
$\quad S := f(S, X[i])$

Mivel a leggyakrabban az elemek *összeadását* értjük az összegzés alatt, ezért kiemelt szerepet tölt be a $\sum$ esete.

- Bemenet: $N \in \mathbb{N}, X_{1..N} \in \mathbb{H}^N$
- Kimenet: $S \in \mathbb{H}$
- Előfeltétel: $-$
- Utófeltétel: $\boxed{S := \sum_{i:=1}^{N} X_i}$

$S := 0$
$i := 1..N$
$S := S + X[i]$

Természetesen, van arra lehetőség, hogy csak azokat az elemeit összegezzük a sorozatnak, amik megfelelnek egy adott T tulajdonságnak. Ilyenkor a tétel: $S := \sum_{\substack{i:=1 \\ T(X_i)}}^{N} X_i$. Erre példát a Feladatgyűjtemény 1. fejezetében találhatunk.

Feladatok

- Ismerjük egy ember havi bevételeit és kiadásait. Adjuk meg, hogy év végére mennyivel nőtt a vagyona!
- Ismerjük egy autóversenyző körönkénti idejét. Adjuk meg az átlagkörének idejét!
- Adjuk meg az n számhoz az n faktoriális értékét!
- Ismerjük egy iskola szakköreire járó tanulóit, szakkörönként. Adjuk meg, kik járnak szakkörre!
- Ismerünk n szót. Adjuk meg a belőlük összeállított mondatot!

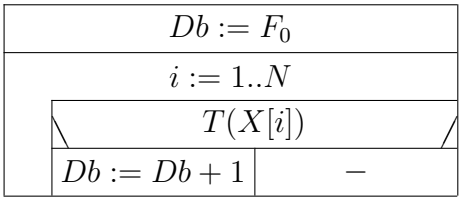
Megjegyzések

1. A konkrét feladat előfeltétele lehet erősebb, mint a programozási tételé.
2. A konkrét feladat utófeltétele lehet gyengébb, mint a programozási tételé (lesz ilyen).
3. Az 1-től N-ig indexelt tömb helyett lehet E-től U-ig indexelt tömb.
4. Egyetlen tömb elemei helyett lehet a tételben szereplő „i-edik elem” értékét kiszámító kifejezés (több tömbből, több tömbelemből; vagy tömbtől független függvény).

2.1.2. Megszámolás

Objektíva: n darab „valamire” kell megadni, hogy hány adott tulajdonságú van közöttük.

- Definíció: $T : \mathbb{H} \to \mathbb{L}$
- Bemenet: $N \in \mathbb{H},\ X_{1..N} \in \mathbb{H}^N$
- Kimenet: $Db \in \mathbb{H}$
- Előfeltétel: $-$
- Utófeltétel: $\boxed{Db := \sum_{\substack{i:=1 \\ T(X_i)}}^{N} 1}$



Megjegyzés

A T tulajdonság egy **logikai függvény**ként adható meg. X (sőt $\mathbb{H}$) minden elemről megvizsgálható, hogy rendelkezik-e az adott tulajdonsággal vagy sem.

A diasorban a T tulajdonságfüggvény a *bemenet*nél szerepel, de hagyományosan külön szoktuk megadni a *definíció*ban. Ilyenkor a szignatúra és a függvénytörzs a feladattól függően más-más lesz.

Feladatok

1. Ismerjük egy ember havi bevételeit és kiadásait. Adjunk meg, hogy hány hónapban nőtt a vagyona!
2. Adjuk meg egy természetes szám osztói számát!
3. Adjuk meg egy ember nevében levő „a” betűk számát!
4. Adjunk meg az éves statisztika alapján, hogy hány napon fagyott!
5. Adjuk meg n születési hónap alapján, hogy közöttük hányan születtek télen!

2.1.3. Maximumkiválasztás

Objektíva: n darab „valami” közül kell megadni a legnagyobbat (vagy a legkisebbet).

A „valamik” között értelmezhető egy rendezési reláció. Ha **legalább 1** „valamink” van, akkor legnagyobb (legkisebb) is biztosan van közöttük!

- Be: $N \in \mathbb{N},\ X_{1..N} \in \mathbb{H}^N$
- Ki: $MaxInd \in \mathbb{N},\ MaxÉrt \in \mathbb{H}$
- Ef: $N > 0$
- Uf_1: $1 \leq MaxInd \leq N\ \wedge\ \forall i\ (1 \leq i \leq N) : X_{MaxInd} \geq X_i\ \wedge\ MaxÉrt := X_{MaxInd}$
- Uf_2: $\boxed{(MaxInd, MaxÉrt) := \operatorname*{Max}_{i:=1}^{N} X_i}$
- Uf_3 (csak az index): $\boxed{MaxInd := \operatorname*{Max}_{i:=1}^{N} X_i}$ vagy $MaxInd := \operatorname*{MaxInd}_{i:=1}^{N} X_i$
- Uf_4 (csak az érték): $\boxed{MaxÉrt := \operatorname*{Max}_{i:=1}^{N} X_i}$ vagy $MaxÉrt := \operatorname*{MaxÉrt}_{i:=1}^{N} X_i$

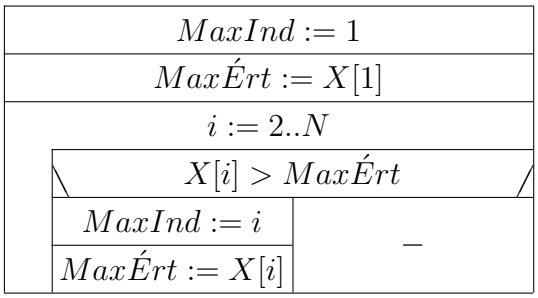
Megjegyzések

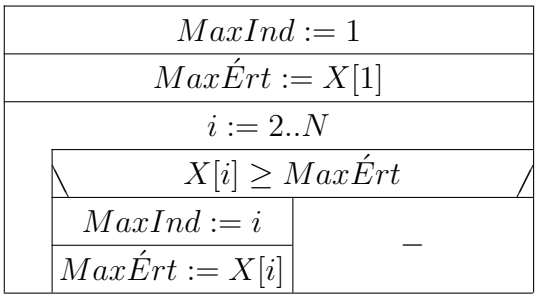
1. Uf_1 – Léteznie kell a $\geq: \mathbb{H} \times \mathbb{H} \to \mathbb{L}$ **rendezési reláció**nak!
2. Uf_2 – Ugyanaz, mint az Uf_1, csak egy szummával azonos tömörségű operátort vezettünk be.
3. Uf_3, Uf_4 – A kettő kifejezés ugyanazt takarja, az utóbbi csak tovább konkretizálja, hogy melyik értéket fogjuk visszakapni.
4. Abban az esetben, amikor vagy csak az indexet, vagy csak az értéket akarjuk megkapni; a struktogramban csak az a változó fog szerepelni, amelyikre kíváncsiak vagyunk.

A minimumkiválasztásra nem létezik önálló tétel. Egyedül a változók neveiből meg a struktogramból tudhatjuk meg, hogy éppen a minimumra vagy a maximumra vagyunk kíváncsiak ($MaxInd \to MinInd$, $MinInd := \operatorname*{Max}_{i:=1}^{N} X_i$, stb...).

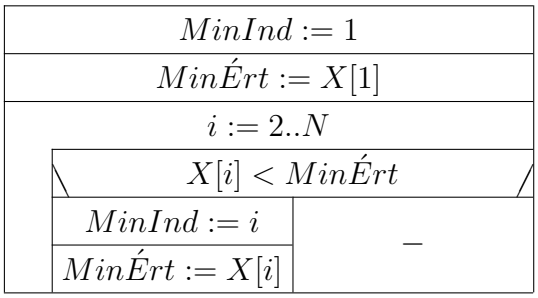
Ha több maximális érték is szerepel a sorozatunkban, megadhatjuk, hogy az első találatot vagy az utolsót adja vissza a programunk. Ha a struktogramban $>$, $<$ található, akkor az elsőt fogja visszaadni, ha pedig $\geq$, $\leq$, akkor az utolsó.

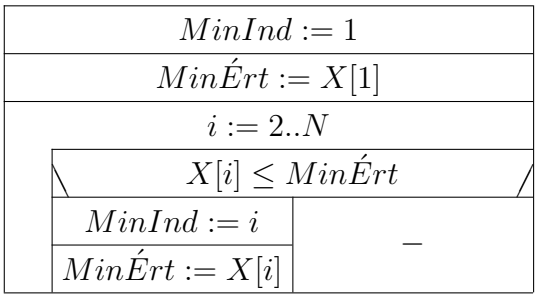
Maximumkiválasztás: első és utolsó maximum





Minimumkiválasztás: első és utolsó minimum





Feladatok

1. Ismerjük egy ember havi bevételeit és kiadásait. Adjunk meg, hogy melyik hónapban nőtt legjobban a vagyona!
2. Adjuk meg n ember közül az ábécében utolsót!
3. Adjuk meg n ember közül azt, aki a legtöbb ételt szereti!
4. Adjunk meg az éves statisztika alapján a legmelegebb napot!
5. Adjuk meg n születésnap alapján azt, akinek idén először van születésnapja!

2.1.4. Eldöntés

Objektíva: döntsük el, hogy n „valami” között van-e adott tulajdonsággal rendelkező elem!

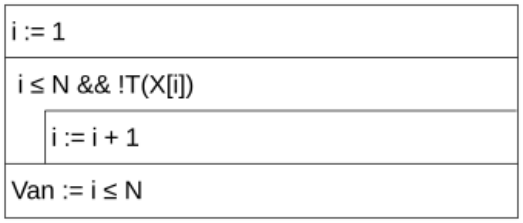
Feladatok

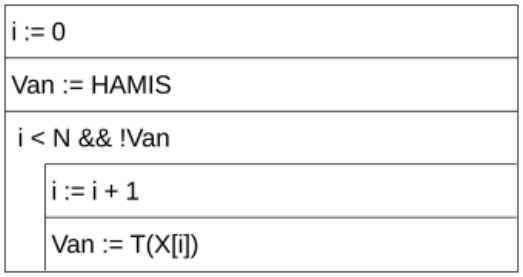
1. Egy természetes számról döntsük el, hogy prímszám-e!
2. Egy szóról mondjuk meg, hogy egy hónapnak a neve-e!
3. Egy tanuló év végi osztályzatai alapján állapítsuk meg, hogy bukott-e!
4. Egy szóról adjuk meg, hogy van-e benne magánhangzó!
5. Egy számsorozatról döntsük el, hogy monoton növekvő-e!
6. Egy tanuló év végi jegyei alapján adjuk meg, hogy kitűnő-e!

A feladatokból megállapíthatjuk, hogy két esetet különböztethetünk meg.

Létezik-e olyan? ($\exists$)

- Def: $T : \mathbb{H} \to \mathbb{L}$
- Be: $N \in \mathbb{N},\ X_{1..N} \in \mathbb{H}^N$
- Ki: $Van \in \mathbb{L}$
- Ef: $-$
- Uf_1: $Van := \exists i\ (1 \leq i \leq N) : T(X_i)$
- Uf_2: $Van := \exists i \in [1..N] : T(X_i)$
- Uf_3: $\boxed{Van := \overset{N}{\underset{i:=1}{\exists}} T(X_i)}$





Egy kis magyarázat az algoritmushoz: az algoritmusban egy `while`-ciklus szerepel, ami *pesszimistán* figyeli, hogy az adott tulajdonság érvényesül-e. Feltételezzük, hogy az első eleme a sorozatnak nem rendelkezik T tulajdonsággal, így addig kell mennünk, ameddig meg nem találjuk az első elemet, amire érvényesül. Ekkor kiugrunk a ciklusból.

A második variáns is hasonló logikával működik, csupán a változók más sorrendben szerepelnek és a Van a ciklusban mindig kiértékelődik. Amint igazzá válik, kiugrik a ciklusból.

Mivel a programozási nyelvek a tömböket 0-tól indexelik, emiatt némileg át kell írnunk egyes részleteket ahhoz, hogy megfelelően le tudjuk írni az algoritmust a kívánt programozási nyelven (esetünkben C#-ban):

1. algoritmus

```
bool Van;
int i = 0;

while (i < N && !T(X[i]))
    i++;

van = i < N;
```

2. algoritmus

```
// 1. variáns

bool Van = false;
int i = -1;

while (i < (N - 1) && !Van)
{
    i++;
    Van = T(X[i]);
}
```

```
// 2. variáns

bool Van = false;
int i = 0;

while (i < N && !Van)
{
    Van = T(X[i]);
    i++;
}
```

Mindegyik-e olyan? ($\forall$)

- Def: $T : \mathbb{H} \to \mathbb{L}$
- Be: $N \in \mathbb{N},\ X_{1..N} \in \mathbb{H}^N$
- Ki: $Mind \in \mathbb{L}$
- Ef: $-$
- Uf_1: $Mind := \forall i\ (1 \leq i \leq N) : T(X_i)$
- Uf_2: $Mind := \forall i \in [1..N] : T(X_i)$
- Uf_3: $\boxed{Mind := \overset{N}{\underset{i:=1}{\forall}} T(X_i)}$

Itt pontosan fordítva közelítjük meg a problémát: optimistán azt vizsgáljuk, hogy az összes elemre teljesül-e T tulajdonság. Ha i N előtt kiugrik, akkor megvizsgálja, hogy $i > N$. Figyeljük meg, hogy a relációsjel az előző esetnek a tagadását jelenti:

$$\neg(i \leq N) \iff i > N$$

Tegyük fel, hogy az összes elemre igaz a T tulajdonság. Végigmegyünk az összesen, és amikor a ciklus az utolsó lépéshez ér (ergo: $i == N$), akkor hozzáad +1-et az i-hez, így biztosak lehetünk, hogy az $i > N$ igaz lesz.

Az algoritmus C#-ban:

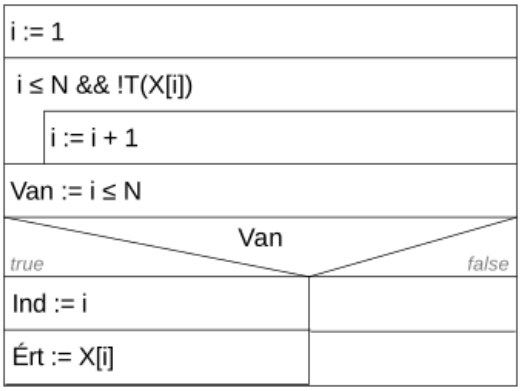
```
bool Mind;
int i = 0;

while (i < N && T(X[i]))
    i++;

Mind = i >= N;
```

2.1.5. Keresés

Objektíva: n darab „valami” közül kell megadni egy adott tulajdonságút, ha nem tudjuk, hogy ilyen elem van-e.

Vegyük észre, hogy ez a tétel tartalmazza a korábbi eldöntés tételt.

- Def: $T : \mathbb{H} \to \mathbb{L}$
- Be: $N \in \mathbb{N},\ X_{1..N} \in \mathbb{H}^N$
- Ki: $Van \in \mathbb{L},\ Ind \in \mathbb{N},\ \acute{E}rt \in \mathbb{H}$
- Ef: $-$
- Uf_1: $Van := \exists i\ (1 \leq i \leq N) : T(X_i)\ \wedge$
  $Van \implies 1 \leq Ind \leq N \wedge T(X_{Ind}) \wedge \acute{E}rt := X_{Ind}$
- Uf_2: $\boxed{(Van, Ind, \acute{E}rt) := \underset{\substack{i:=1 \\ T(X_i)}}{\overset{N}{\mathrm{Keres}}}\, i}$



Többlet tudás: a megoldás az első adott tulajdonságú elemet adja meg.

Feladatok

1. Ismerjük egy ember havi bevételeit és kiadásait. Év végére nőtt a vagyona. Adjunk meg egy hónapot, amikor nem nőtt a vagyona!
2. Adjuk meg egy természetes szám egy 1-től és önmagától különböző osztóját!
3. Adjuk meg egy ember nevében egy „a” betű helyét!
4. Adjunk meg egy tanulóra egy tárgyat, amiből megbukott!
5. Adjuk meg egy számsorozat olyan elemét, amely nagyobb az előzőnél!

2.1.6. Kiválasztás

Objektíva: n „valami” közül kell megadni egy adott tulajdonságút, ha tudjuk, hogy ilyen elem biztosan van.

Ez a keresés programozási tétel olyan változata, amelyben nem kell felkészülnünk arra, hogy a keresett elemet nem találjuk meg.

- Def: $T : \mathbb{H} \to \mathbb{L}$
- Be: $N \in \mathbb{N},\ X_{1..N} \in \mathbb{H}^N$
- Ki: $Ind \in \mathbb{N},\ \acute{E}rt \in \mathbb{H}$
- Ef: $N > 0\ \wedge\ \exists i\ (1 \leq i \leq N) : T(X_i)$
- Uf_1: $1 \leq Ind \leq N\ \wedge\ T(X_{Ind})\ \wedge\ \acute{E}rt := X_{Ind}$
- Uf_2: $(Ind, \acute{E}rt) := \underset{\substack{i:=1 \\ T(X_i)}}{\overset{N}{\text{Kiválaszt}}}\, i$

```
i := 1
!T(X[i])
    i := i + 1
Ind := i
Ért := X[i]
```

Többlet tudás: a megoldás az első adott tulajdonságú elemet adja meg – a program tudhat többet annál, mint amit várunk tőle. Hogy kellene az utolsót megadni?

Feladatok

1. Ismerjük egy ember havi bevételeit és kiadásait. Év végére nőtt a vagyona. Adjunk meg egy hónapot, amikor nőtt a vagyona!
2. Adjuk meg egy 1-nél nagyobb természetes szám egytől különböző legkisebb osztóját!
3. Adjuk meg egy magyar szó egy magánhangzóját!
4. Adjuk meg egy hónapnévről a sorszámát!

2.2. Összetett programozási tételek

Ezek a tételek a **sorozat** → **sorozat** kategóriába tartoznak, kivéve a szétválogatást – ami a **sorozat** → **sorozatok** csoport tagja.

2.2.1. Másolás

Objektíva: n darab „valamihez” kell hozzárendelni másik n darab „valamit”, ami akár az előbbitől különböző típusú is lehet. A darabszám, a sorrend is marad. Az elemeken operáló függvény ugyanaz.

Feladatok

1. Egy számsorozat tagjainak adjuk meg az abszolút értékét!
2. Egy szöveget alakítsunk át csupa kisbetűssé!
3. Számoljuk ki két vektor összegét!
4. Készítsünk függvénytáblázatot a $\sin(x)$ függvényről!
5. Ismerünk n dátumot `'éé.hh.nn'` alakban, adjuk meg őket `'éé. hónapnév nn.'` alakban!

A tétel általánosított alakja:

- Definíció: $f : \mathbb{H}_1 \to \mathbb{H}_2$
- Bemenet: $N \in \mathbb{N},\ X_{1..N} \in \mathbb{H}_1^N$
- Kimenet: $Y_{1..N} \in \mathbb{H}_2^N$
- Előfeltétel: $-$
- Utófeltétel$_1$: $\forall i \in [1..N] : Y_i := f(X_i)$
- Utófeltétel$_2$: $\boxed{Y_{1..N} := f(X_{1..N})}$

```
i := 1 .. N
    Y[i] := f(X[i])
```

```
i := 1 .. N
    Y[p(i)] := f(X[i])
```

Megjegyzés

Nem feltétlenül kell ugyanaz az i index a két tömbhöz, pl.:

$$\forall i\ (1 \leq i \leq N) : Y_{p(i)} := f(X_i)$$

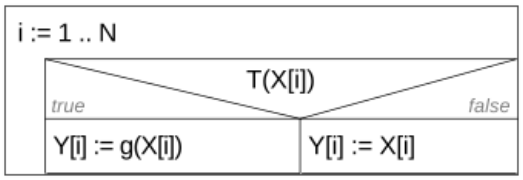
$p(i)$ lehet pl. $2 \cdot i,\ N-i+1, \ldots$ (megfelelő Y tömb mérettel, ill. indexintervallummal definiálva; Y részsorozata a kimenet; p injektív).

Egy gyakori speciális eset – csak bizonyos esetekben hajtunk végre módosításokat:

- Def: $T : \mathbb{H} \to \mathbb{L}$

  $f : \mathbb{H} \to \mathbb{H}$

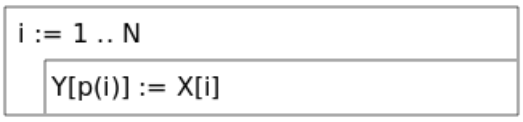
  $$f(x) := \begin{cases} g(x) & \text{ha } T(x) \\ x & \text{egyébként} \end{cases}$$



- Be: $N \in \mathbb{N},\ X_{1..N} \in \mathbb{H}^N$
- Ki: $Y_{1..N} \in \mathbb{H}^N$
- Ef: $-$
- Uf_1: $\boxed{\forall i\ (1 \leq i \leq N) : Y_i := f(X_i)}$
- Uf_2: $\forall i\ (1 \leq i \leq N) : (\ T(X_i) \implies Y_i := g(X_i)\ \wedge\ \neg T(X_i) \implies Y_i := X_i\ )$

És az identikus másolás esete (másolás a szó szoros értelmében):

- Be: $N \in \mathbb{N},\ X_{1..N} \in \mathbb{H}^N$
- Ki: $Y_{1..N} \in \mathbb{H}^N$
- Ef: $-$
- Uf: $\boxed{\forall i \in [1..N] : Y_i := X_i}$



Megjegyzés

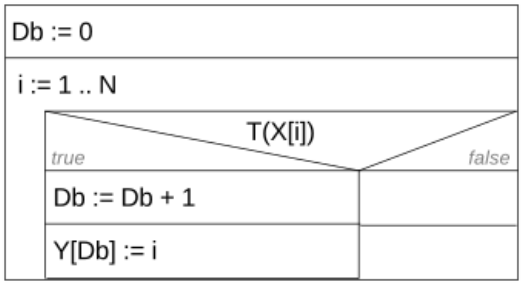
1. Nincs f függvény, helyesebben identikus ($f(x) := x$)
2. Az $Y := X$ értékadással helyettesíthető, ha a két tömb azonos típusú. Ha az indexek különbözőek (p nem identikus), akkor a második algoritmus érvényesül

2.2.2. Kiválogatás

Objektíva: n darab „valami” közül kell megadni az összes, adott T tulajdonsággal rendelkezőt!

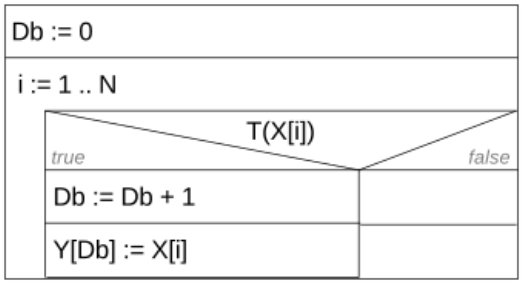
Kiválogatás statikus tömbbe

Az indexek kiválogatása:



- Def: $T : \mathbb{H} \to \mathbb{L}$
- Be: $N \in \mathbb{N},\ X_{1..N} \in \mathbb{H}^N$
- Ki: $Db \in \mathbb{N},\ Y_{1..N} \in \mathbb{N}^N$
- Ef: $-$
- Uf_1: $Db := \sum_{\substack{i:=1 \\ T(X_i)}}^{N} 1 \ \wedge\ \forall i \in (1 \leq i \leq Db) : T(X_{Y_i}) \ \wedge\ Y \subseteq (1, 2, \ldots, N)$
- Uf_2: $\boxed{(Db, Y) := \underset{\substack{i:=1 \\ T(X_i)}}{\overset{N}{\text{Kiválogat}}}\, i}$

Maguknak az elemek kiválogatása:



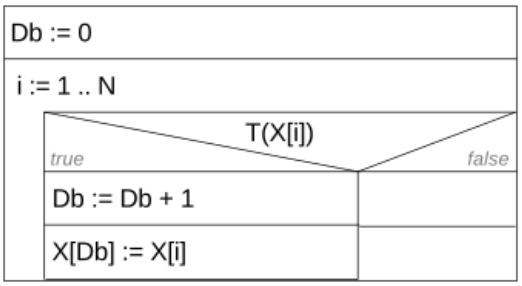
- Def: $T : \mathbb{H} \to \mathbb{L}$
- Be: $N \in \mathbb{N},\ X_{1..N} \in \mathbb{H}^N$
- Ki: $Db \in \mathbb{N},\ Y_{1..N} \in \mathbb{H}^N$
- Ef: $-$
- Uf_1: $Db := \sum_{\substack{i:=1 \\ T(X_i)}}^{N} 1 \ \wedge\ \forall i\ (1 \leq i \leq Db) : T(Y_i) \ \wedge\ Y \subseteq X$
- Uf_2: $\boxed{(Db, Y) := \underset{\substack{i:=1 \\ T(X_i)}}{\overset{N}{\text{Kiválogat}}}\, X_i}$

Vegyük észre az apró eltéréseket: a kimeneti tömb típusa ($\mathbb{N}$, $\mathbb{H}$) és az utófeltétel paraméterei (i, X_i) megváltoznak.

Kiválogatás helyben

Ennél úgy közelítjük meg a problémát, hogy ugyanabba a bemeneti tömbbe tároljuk el a kiválogatott elemeket, mint amiből kikeressük.

- Def: $T : \mathbb{H} \to \mathbb{L}$
- Be: $N \in \mathbb{N},\ X_{1..N} \in \mathbb{H}^N$
- Ki: $Db \in \mathbb{N},\ X'_{1..N} \in \mathbb{H}^N$
- Ef: $-$



- Uf: $\boxed{Db := \sum_{\substack{i:=1 \\ T(X_i)}}^{N} 1 \ \wedge \ X'_{1..Db} \subseteq X_{1..N} \ \wedge \ \forall i \in [1..Db] : T(X'_i)}$

Ilyenkor nem az indexeket, hanem az elemeket rendezzük át a tömbön belül.

Az aposztrófffal (**'**) különböztetjük meg, hogy a tömb bemeneti vagy kimeneti állapotáról van szó.

Kiválogatás dinamikus tömbbe

A programozás a tömb típuson kívül sokféle sorozat típust ismer. Közülük az egyik egy olyan indexelhető típus, aminek az elemszáma **futás közben növelhető** (ebből a szempontból a szöveg típusra hasonlít).

Figyelem: e típus használata jelentősen megnövelheti a program futási idejét!

Műveletek

- $Hossz(S)$ – az S sorozat és a neki megfelelő tömb elemeinek száma
- $Végére(S, x)$ – az S tömb végére egy új elemet, az x-et illeszti
- $S[i]$ – az S tömb i-edik eleme

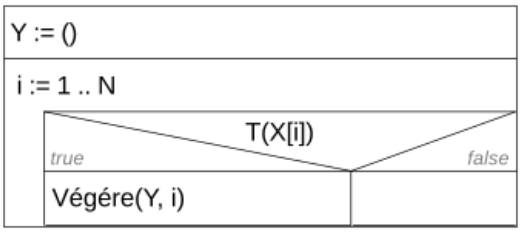
További műveletek is lehetnek, most nem térünk ki rá.

Dinamikus tömb C#-ban

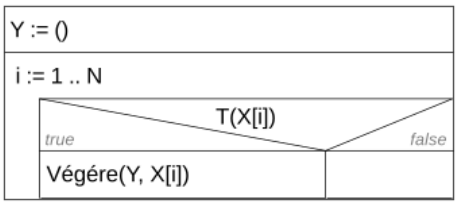
```
// deklaráció és létrehozás
List<TElem> S = new List<TElem>();

// Műveletek
int size = S.Count;         // Hossz(S)
S.Add(x);                   // Végére(S, x)
Console.WriteLine(S[i]);    // Ki: S[i]
```

Az indexek kigyűjtése:

- Def: $T : \mathbb{H} \to \mathbb{L}$
- Be: $N \in \mathbb{N},\ X_{1..N} \in \mathbb{H}^N$
- Ki: $Y_{1..} \in \mathbb{N}^*$
- Ef: $-$
- Uf: $\boxed{Hossz(Y) := \sum_{\substack{i:=1 \\ T(X_i)}}^{N} 1 \ \wedge\ \forall y \in Y : T(X_y) \ \wedge\ Y \subseteq (1, 2, \ldots, N)}$



A sorszám általánosabb, mint az érték. Ha mégis érték kellene, akkor $Végére(Y, X[i])$ szerepelne. (Ekkor a specifikációt is módosítani kell!)



Feladatok

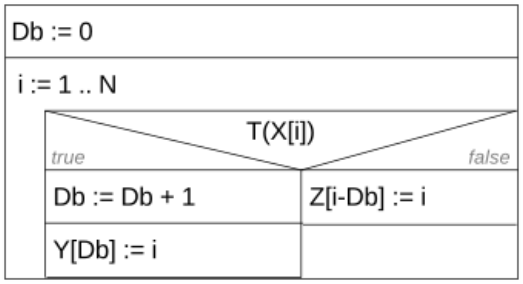
1. Adjuk meg egy osztály kitűnő tanulóit!
2. Adjuk meg egy természetes szám összes osztóját!
3. Adjuk meg egy mondat magas hangrendű szavait!
4. Adjuk meg emberek egy halmazából a 180 cm felettieket!
5. Adjuk meg egy év azon napjait, amikor délben nem fagyott!
6. Soroljuk föl egy szó magánhangzóit!

2.2.3. Szétválogatás

Objektíva: n darab „valami” közül kell megadni az összes, adott T tulajdonsággal rendelkezőt, illetve nem rendelkezőt! Azaz az összes bemeneti elemet „besoroljuk” a kimenet valamely sorozatába. A többfelé szétválogatás visszavezethető a kétfelé szétválogatásra.

Szétválogatás két statikus tömbbe

Indexek szétválogatása:



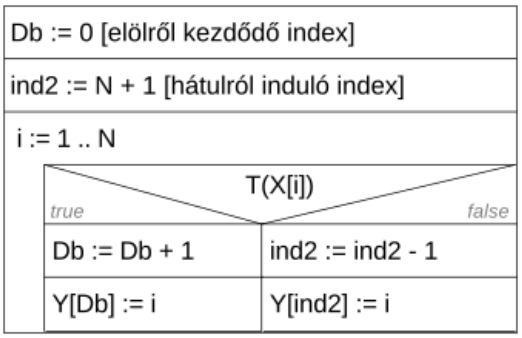
- Def: $T : \mathbb{H} \to \mathbb{L}$
- Be: $N \in \mathbb{N},\ X_{1..N} \in \mathbb{H}^N$
- Ki: $Db \in \mathbb{N},\ Y_{1..N} \in \mathbb{N}^N,\ Z_{1..N} \in \mathbb{N}^N$
- Ef: $-$
- Uf_1: $Db := \sum_{\substack{i:=1 \\ T(X_i)}}^{N} 1 \ \wedge\ \forall i\ [1..Db] : T(X_{Y_i}) \ \wedge\ \forall i\ [1..N - Db] : \neg T(X_{Z_i}) \ \wedge$
  $Y \subseteq (1, 2, \ldots, N) \ \wedge\ Z \subseteq (1, 2, \ldots, N)$
- Uf_2: $\boxed{(Db, Y, Z) := \overset{N}{\underset{\substack{i:=1 \\ T(X_i)}}{\text{Szétválogat}}}\, i}$

Értékek szétválogatásánál, mint korábban, figyeljünk arra, hogy a jelölések megváltoznak: $\mathbb{N}$-ből $\mathbb{H}$ lesz, az utófeltétel meg $(Db, Y, Z) := \overset{N}{\underset{\substack{i:=1 \\ T(X_i)}}{\text{Szétválogat}}}\, X_i$. A struktogramban az alábbi értékadás fog megváltozni: `Y[Db]:= i` helyett `Y[Db]:= X[i]`.

Szétválogatás egy tömbbe

Felmerülő probléma: Y-ban és Z-ben együtt csak n darab elem van, azaz elég lenne egyetlen n elemű sorozat.

- Def: $T : \mathbb{H} \to \mathbb{L}$
- Be: $N \in \mathbb{N},\ X_{1..N} \in \mathbb{H}^N$
- Ki: $Db \in \mathbb{N},\ Y_{1..} \in \mathbb{N}^N$
- Ef: $-$

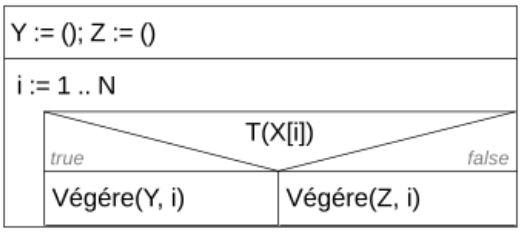


- Uf$_1$: $Db := \sum_{\substack{i:=1 \\ T(X_i)}}^{N} 1 \ \wedge\ \forall i \in [1..Db] : T(X_{Y_i}) \ \wedge\ \forall i \in [(Db+1)..N] : \neg T(X_{Y_i}) \wedge$
  $Y \in Permutáció(1, 2, \ldots, N)$
- Uf$_2$: $\boxed{(Db, Y, Z) := \text{Szétválogat}_2 \sum_{\substack{i:=1 \\ T(X_i)}}^{N} i}$

Ha az értékeket akarjuk szétválogatni, figyeljünk oda a módosításokra!

Szétválogatás két dinamikus tömbbe

- Def: $T : \mathbb{H} \to \mathbb{L}$
- Be: $N \in \mathbb{N},\ X_{1..N} \in \mathbb{H}^N$
- Ki: $Y_{1..} \in \mathbb{N}^*,\ Z_{1..} \in \mathbb{N}^*$
- Ef: $-$

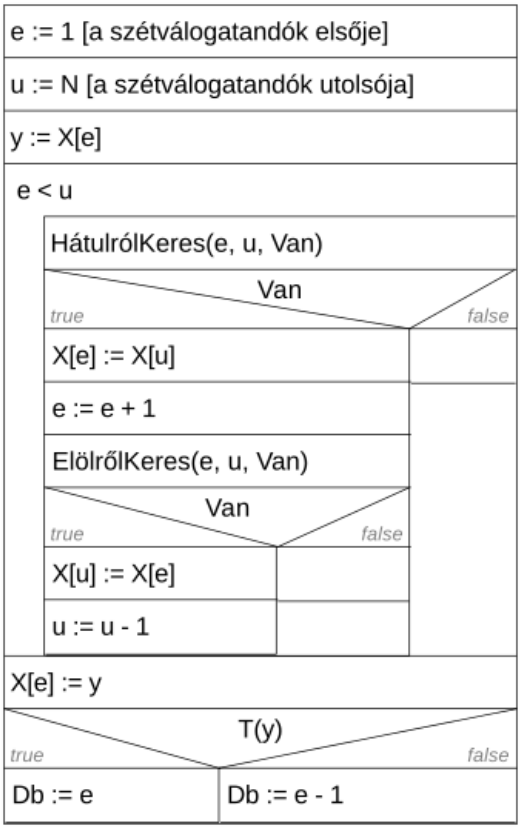


- Uf: $Hossz(Y) := \sum_{\substack{i:=1 \\ T(X_i)}}^{N} 1 \ \wedge\ Y \subseteq (1, 2, \ldots, N) \ \wedge\ \forall y \in Y : T(X_y) \ \wedge$
  $Hossz(Z) := \sum_{\substack{i:=1 \\ \neg T(X_i)}}^{N} 1 \ \wedge\ Z \subseteq (1, 2, \ldots, N) \ \wedge\ \forall z \in Z : \neg T(X_z)$

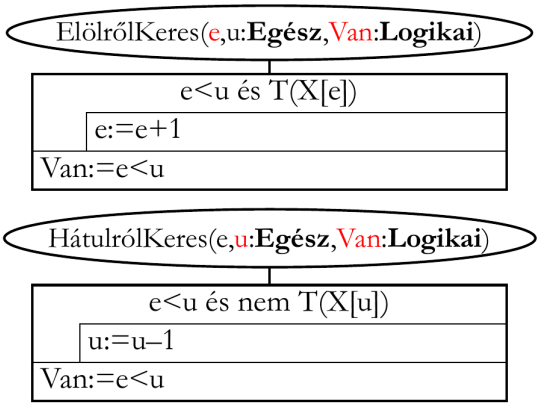
Ha az értékeket akarjuk szétválogatni, figyeljünk oda a módosításokra!

Szétválogatás helyben

- Def: $T : \mathbb{H} \to \mathbb{L}$
- Be: $N \in \mathbb{N},\ X_{1..N} \in \mathbb{H}^N$
- Ki: $Db \in \mathbb{N},\ X'_{1..N} \in \mathbb{H}^N$
- Ef: $-$
- Uf: $Db := \sum_{\substack{i:=1 \\ T(X_i)}}^{N} 1 \wedge$
  $X' \in Permutáció(X) \wedge$
  $\forall i\ (1 \leq i \leq Db) : T(X'_i) \wedge$
  $\forall i\ (Db + 1 \leq i \leq N) : \neg T(X'_i)$



A segédfüggvények definíciói:



Magyarázat az algoritmushoz

1. Vegyük ki (másoljuk le) a sorozat első elemét:
   Oxxxxxxxxxxxx
2. Keresünk hátulról egy elemet, aminek elől a helye (mert T tulajdonságú, nem odavaló):
   Oxxxxxx**x**xxxxx
3. A megtalált elemet tegyük az előbb keletkezett lyukba:
   ⊗xxxxxx**O**xxxxx
   A lyuk mögött és az 1. elemmel már rendben vagyunk.
4. Most keletkezett egy lyuk hátul. Az előbb betöltött lyuktól indulva elölről keressünk hátra teendő (nem odavaló: nem T tulajdonságú) elemet:
   ⊗xx**x**xx⊗xxxxx
5. A megtalált elemet tegyük a hátul levő lyukba, majd újra hátulról kereshetünk!
   ⊗xx**O**xx⊗xxxxx
   Az elől keletkezett lyuk előttiek és a hátrébb mozgatott elemmel kezdve rendben vagyunk.
6. ... és így tovább ...
7. Befejezzük a keresést, ha valahonnan elértük a lyukat.
   xxxxOxxxxxxxx
8. Erre a helyre az 1. lépésben kivettet visszatesszük.

Megjegyzés

Az X változóról az algoritmus végrehajtása közben különböző állításokat mondhatunk:

1. kezdetben a bemenetbeli sorozat;
2. a futás végén a bemeneti X permutációja a szétválogatás utófeltétele szerint;
3. közben e-ig (ún. ciklusinvariáns) T tulajdonságú elemek, u-tól nem T tulajdonságú elemek, köztük nem vizsgált elemek.

Feladatok

1. Adjuk meg egy számsorozatból a páros és a páratlan számokat is!
2. Adjuk meg egy év azon napjait, amikor délben fagyott és amikor nem fagyott!
3. Adjuk meg egy angol szó magán- és mássalhangzóit!
4. Adjuk meg emberek egy halmazából a 140 cm alattiakat, a 140 és 180 cm közöttieket és a 180 cm felettieket!
5. Adjuk meg emberek egy halmazából a télen, tavasszal, nyáron, illetve ősszel születetteket!

3. fejezet

Programozási eszköztár

Ez a fejezet az 5-6. előadás diasorainak tartalmát foglalja össze, illetve rendszerezi.

3.1. Tömbök fajtái

A tömböt mindenki ismeri, nem kell bemutatni. De azért mégis tegyük meg: azonos típusú, véges számú* adat tárolására képes.

3.1.1. Hagyományos tömb

Meghatározott méretű, nem lehet módosítani.

Ne feledjük deklarálni a méretét előtte az alsó indexben. A tömb halmazának felső indexébe tegyük ki az elemszámot.

Tömb használata

Deklarációja: $n \in \mathbb{N}$, $array_{1..n} \in \mathbb{H}^n$
i-edik elem elérése: $array_i$ vagy $array[i]$

Ne feledjük, hogy a specifikációban a tömb elemeit 1-től n-ig számozzuk (az n beleszámít) ($1 \leq i \leq n$), míg a programozási nyelvekben 0-tól $(n-1)$-ig ($0 \leq i < n$).

3.1.2. Dinamikus tömb*

A dinamikus tömbökre már nem igaz, hogy nem módosítható a méretük. Pontosan erre utal a neve: futattási időben módosítható.

Dinamikus tömb használata

Deklarációja: $dynamic \in \mathbb{H}^*$
i-edik elem elérése: $dynamic_i$ vagy $dynamic[i]$
Speciális műveletek: *dynamic.Végére(elem)*, *dynamic.Hossza()*

Hagyományosan nem szoktunk dinamikus tömbökről beszélni, mivel kezdetben nem léteztek ilyesmik a programozási nyelvekben (a C-ben a mai naig nincs). Éppen ezért nem használjuk túl gyakran a specifikációkban, ám jó, ha tudunk róluk.

A C#-ban létezik ún. dinamikus tömb, csak ezt *listá*nak hívják (`List<>`). A későbbi fejezetekben lesz róla szó.

3.2. Típusdefiníció – Rekord (Struktúra)

Összetett típusokat tudunk vele létrehozni. Jellemzően olyan feladatoknál hasznos, amikor több, különféle típusú adatot tárolunk valamilyen rendszerezés szerint – például: egy bolt feljegyzi az árusított termékeinek nevét (*szöveg*), árát (*egész*), és hogy árstoppos-e az adott termék (*logikai*).

Rekord használata

Definíciója: $typename := rec(elem_1 \in \mathbb{H}_1 \times elem_2 \in \mathbb{H}_2 \times \ldots \times elem_n \in \mathbb{H}_n)$
Változó deklarálása: $var \in typename$
Tömb deklarálása: $array_{1..n} \in typename^n$
„Összetevő” elérése: $var.elem_1$, $array_i.elem_2$, stb.

Megfelel a C-ben (C++-ban, C#-ban, stb.) a `struct`-nak. Éppen ezért mind a két elnevezés használatos, ám jellemzően a rekordot használunk, amikor specifikációról van szó, míg minden más esetben struktúrát.

A specifikációnkban van lehetőségünk rekordtípusú tömb létrehozására is. Erre gyakran szükségünk lesz a feladatok megoldása során. Értelemszerűen mindig a *definíció* részében definiáljuk.

3.3. Mátrixok

Mátixoknak nevezzük az olyan tömböket, amik több dimenzióból állnak. Arra alkalmasak, hogy azonos típusú adatokat tároljon többféle szempont szerint – például: adott n település, ahol m napon keresztül feljegyeztük a napi csapadékot (itt az adat a csapadék, amit a település és a nap szempontja szerint rögzítettünk).

A *bemenet*ben szoktuk deklarálni. Emellett ne feledkezzünk meg előtte deklarálni külön a dimenzióit is, valamint jelölni ezeket a halmaz jobb felső sarkában.

Mátrix használata

$n \in \mathbb{N}$, $m \in \mathbb{N}$, $matrix_{1..n,1..m} \in \mathbb{H}^{n \times m}$

Első ránézésre gondolhatnánk, hogy használhatnánk helyette rekordot is – ami igaz, bizonyos esetekben. Sajnos, a legtöbbször nem tudjuk, hogy mégis mennyi n-re és m-re számíthatunk, ebből kifolyólag elég kényelmetlen lenne n vagy m darab elemet felírni a rekordba.

3.4. Szöveg és tömb*

Nem gyakran fogunk szövegtípussal találkozni, de jó, ha ismerjük.

3.5. Összetett típusok – Halmazok*

Az összetett típusok közé tartoznak még a halmaztípusok, melyeknek 3 fajtája van: **halmaz**, **vektorhalmaz** és **multihalmaz**

Velük a gyakorlaton nem fogunk foglalkozni, egy feladatban sem fognak szerepelni. Akit érdekel, az a későbbi fejezetekben tud róluk olvaskatni bővebben.

3.6. Függvények

Rendkívül fontos összetevő a programozási eszköztárunkban, nélkülük valószínűleg még mindig bányákban élnénk (ne feledjük a funckionális programozáson annyiszor belénksulykolt bölcsességet: *minden függvény*).

Függvények használata

tartalom...

3.7. Programtranszformációk*

Ez a 6. előadás bevezető anyaga, és némimileg átvezetésül szolgál a programozási tételek összeépítésébe.

A gyakorlaton nem fektetünk rájuk túl nagy hangsúlyt – ahogy az előadáson sem –, csupán itt egy helyen össze lesznek gyűjtve, hogy később könnyebb legyen visszakeresni őket.

Azonban egyáltalán nem haszontalanok. Ugyanis előfordul, hogy egy-két ilyen trükköt el kell sütnünk ahhoz, hogy a programunk az időlimiten belül képes legyel lefutni (a Bíróról szóló fejezetben erre bővebben kitérek).

4. fejezet

Programozási tételek összeépítése

Ez a fejezet az 6. előadás diasorainak tartalmát foglalja össze, illetve rendszerezi.

4.1. Tétel01 + Tétel02

5. fejezet

Programozás élesben – A Visual Studio és a C#

Ez a fejezet egyik előadás anyagát sem dolgozza fel, csupán a gyakorlathoz szükséges eszközök használatát mutatja be.

Ahogy az közismert, a tárgyon 2022 óta C#-ban írunk és Microsoft Visual Studio fejlesztőkörnyezetet használunk a programozáshoz. Korábban C++-ban és Code::Blocksban történt ugyanez, azonban a C++ bonyolultságára sok panasz érkezett, emiatt lecserélték.

5.1. A C#-ról röviden*

5.2. Visual Studio – Első lépések

6. fejezet

A Bíró (és a Mester)

Ez a fejezet egyik előadás anyagát sem dolgozza fel, csupán egy, a gyakorlathoz szükséges eszközt mutat be: a Bírót.

6.1. Mi az a Bíró?

6.2. Bírós feladatok felépítése

6.3. Specifikáció és struktogram Bírós feladat alapján

6.4. Program Bírós feladat alapján

6.4.1. C#-os trükkök

6.4.2. Időlimit-túllépés kiküszöbölése

6.4.3. Hibás kimenet

7. fejezet

Dokumentáció írása

Ez a fejezet egyik előadás anyagát sem dolgozza fel, csupán a gyakorlathoz szükséges, pontosabban a komplex beadandó elkészítéséhez elegedhetetlen összetevőt mutatja be: a dokuentációt.

A dokumentáció minden programozó rémálma.

7.1. Mi az, aminek benne kell lennie?

7.2. Fejlesztői dokumentáció

7.3. Felhasználói dokumentáció*