Deklaratív programozás

2. nagy házi feladat (Prolog)
Számtekercs

Tartalomjegyzék

TÖMÖRI PÉTER ANDRÁS

I4RZ0O

2024/25-ÖS TANÉV, ŐSZI FÉLÉV

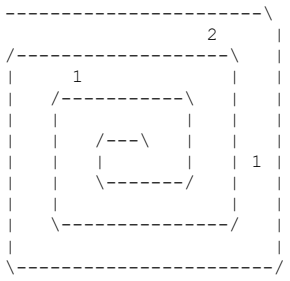
Feladatspecifikáció – a számtekercs feladvány

Adott egy n*n mezőből álló, négyzet alakú tábla, amelynek egyes mezőiben 1 és m közötti számok vannak, *egyes mezők pedig nem tölthetők ki*. A feladat az, hogy *további* 1 és m közötti számokat helyezzünk el a táblában úgy, hogy az alábbi feltételek teljesüljenek:

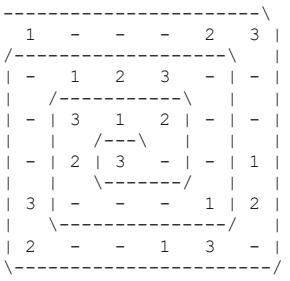
1. Minden sorban és minden oszlopban az 1..m számok mindegyike pontosan egyszer szerepel.
2. A bal felső sarokból induló *tekeredő vonal* mentén a számok rendre az 1,2,...m,1,2,...,m,... sorrendben követik egymást.

A *tekeredő vonalat* a következőképpen definiáljuk. Először a négyzet első sorában haladunk balról jobbra, majd az utolsó oszlopban felülről lefelé. Ezután az utolsó sorban megyünk jobbról balra, majd az első oszlopban alulról fölfelé, egészen a 2. sor 1. mezőjéig. Miután így bejártuk a négyzetes tábla szélső sorait és oszlopait, rekurzívan folytatjuk a bejárást a 2. sor 2. mezőjében kezdődő (n-2)*(n-2) mezőből álló négyzettel.

Az 1. ábra egy feladványt ábrázol, a 2. ábra ennek (egyetlen) megoldását mutatja.



1. ábra. Egy feladvány (n=6, m=3)



2. ábra. A feladvány megoldása

A megvalósítandó tekercs/2 predikátum első, bemenő paramétere a feladványt írja le, a második, kimenő paraméter a feladvány egy megoldása. Az eljárásnak a visszalépések során minden megoldást pontosan egyszer kell kiadnia. Ha a feladványnak nincs megoldása, az eljárás hiúsuljon meg. Egy-egy megoldás a kitöltött táblát írja egész számokból álló listák listájaként. Minden szám a tábla egy mezőjének értéket adja meg: ha 0, a mező üres, egyébként pedig az i értek, ahol $1 \leq i \leq m$

Az 1. ábrán bemutatott feladvány esetén például a tekercs/2 predikátum első paramétere:

```
szt(6,3,[i(1,5,2),i(2,2,1),i(4,6,1)])
```

A 2. ábrán látható megoldást írja le például az alábbi listák listájaként ábrázolt mátrix:

```
[[1,0,0,0,2,3],
 [0,1,2,3,0,0],
 [0,3,1,2,0,0],
 [0,2,3,0,0,1],
 [3,0,0,0,1,2],
 [2,0,0,1,3,0]]
```

Követelmények elemzése

Feladat áttekintése

A feladat egy n*n méretű négyzetes tábla kitöltése olyan módon, hogy:

- Minden sorban és minden oszlopban az 1-től m-ig terjedő számok mindegyike pontosan egyszer szerepeljen.
- A bal felső sarokból induló tekeredő vonal mentén a számok hézagok megengedésével az 1, 2, ..., m, 1, 2, ..., m sorrendben kövessék egymást.

A feladat további korlátozást tartalmaz: A táblán egyes mezők előre megadhatók definiált értékkel, melyek nem módosíthatók.

Paraméterek

A feladat inputja szt(Meret, Ciklus, Adottak), ahol:

- Meret: a tábla mérete (Meret*Meret-es tábla).
- Ciklus: a kitöltendő számsorozatok utolsó eleme.
- Adottak: egy lista, amely mező-érték párokat tartalmaz. Minden listaelem i(Sor, Oszlop, Elem) formátumú, ahol Sor és Oszlop a mező sor- és oszlopindexe, Elem pedig az adott mező értéke, amit nem lehet megváltoztatni.
- A bemenet helyességét a függvény hívásakor már feltételezzük.

Feltételek és korlátozások

- A predikátum a feladvány összes megoldását elő kell, hogy állítsa egyszer.
- Ha nincs megoldás akkor az eljárás meghiúsul.
- A számtekercsnek megfelelő bejárásban a számoknak növekvő sorrendben kell lenni, kivéve az m utáni számot, ami újra 1.
- Nem kell minden mezőn számnak lennie, maradhatnak hézagok: ezt az implementáció során a 0-k jelentik.
- Minden sorban és oszlopban 1-től Ciklus-ig minden szám pontosan egyszer szerepel.
  - ➔ Soronként és oszloponként (Meret-Ciklus) darab 0-s van egy megoldásban
- Adottakat nem szabad módosítani.

Tervezés

Alapgondolatok

Fontos részlet, hogy az általam tervezett implementációban előre definiált értékként megadható 0 is, így az ilyen jellegű bemenetek megoldására is képes.

A keresési feladatok megoldásának egy fontos módszere a korlát-programozás (Constraint Programming, CP), illetve ennek egy részterülete, a korlát-logikai programozás véges tartományokon (Constraint Logic Programming on Finite Domains, CLPFD). Ez utóbbi módszernek az az alapgondolata, hogy bizonyos megszorításokkal (kényszerekkel, korlátozásokkal, korlátokkal) definiált feladatok megoldása során a feladatban szereplő változók *tartományát* (azaz a lehetséges értékek halmazát) tartjuk nyilván, és ezeket a tartományokat a korlátozások alapján próbáljuk folyamatosan szűkíteni. A házi feladat elkészítése során ezt a módszert használtam annak érdekében, hogy a lehető leggyorsabban megoldásokat találjak.
Tehát kezdetnek felvettem egy kiindulási mátrixot, aminek minden eleme egy tartomány a lehetséges értékekkel. Adottakat beállítottam a megfelelő helyre, így azoknál az az egy lehetséges elem maradt. A feladatmegoldás során a véglegesített mezőket, azaz amiknek egy lehetséges értékük marad számként jelölöm, nem egyelemű tömbként. Adottak meghatározása után pedig különböző szűkítési algoritmusokkal szűkítem az elemek tartományát amíg lehet. Ha a szűkítéssel nem jutottam megoldáshoz, akkor a generate-and-test módszerhez hasonló műveletet végzek: A számtekercs bejárásának megfelelően haladok a mezőkön, így számontartva, hogy éppen milyen érték kerülhet az ellenőrzött mezőhöz. Ha egy olyan mezőhöz érek, aminek még nincs végleges értéke, akkor elágazok maximum 2 irányba: ha felveheti azt az értéket, amit a bejárás szerint kell felvennie, akkor azt adom neki értéknek, ismét szűkítek amíg lehetséges és haladok tovább. Továbbá, ha felveheti a mező a 0 értéket, akkor azzal az értékadással is létrehozok egy elágazást, szintén szűkítéssel és továbblépéssel folytatva. Ha már van ott egy érték, ami az adott ágon elvárt tovább lépek a következő mezőre a megfelelő értékekkel. Ha pedig nem az elvárt érték van ott, akkor azt az ágat elvágom, mert nem vezethet helyes megoldáshoz.

Megvalósítandó algoritmus

Az algoritmus az alábbi lépésekből áll:

1. **Adatok Inicializálása**: A kiinduló mátrix létrehozása Adottak beállításával
2. **Szűkítési algoritmusok elvégzése**: A különböző szűkítési algoritmusokat elvégzem, és ha az elvégzésük után a Mátrix módosult, akkor ismételten ellenőrzöm a szűkítés lehetőségét (2. lépés) Ha a szűkítések által kiderült, hogy az adott mátrixnak nincs megoldása, akkor az adott ág eljárása meghiúsul. Ha még van hátra mező a tekercs szerinti bejárásban akkor tovább lépek, ha viszont nem, akkor minden mező biztosan egyelemű, így találtam egy megoldást.

3. **Aktuális mező fixálása, esetleges elágazás**: Ha az adott mezőn fix érték van tovább lépek a következő fixálandó mezőre. (3. lépés). Ha nem, akkor potenciálisan 2 ágon indítok további esetleges megoldáshoz vezető utakat: Az aktuális mező értékét beállítom a várt értékre vagy 0-ra ha lehet, majd szűkítéseket végzet (2. lépés)

Típusok specifikálása

`feladvany_leiro`: `szt(meret, ciklus, list(adott_elem)` a feladvány leírásához szükséges elemeket tartalmazza.
`meret`: Egész szám, a feladvány táblázatának mérete.
`ciklus`: A számsorozat hossza. 1-től `meret`-ig tartó szám lehet
`adott_elem`: `i(sorszam, oszlopszam, elem)` a feladvány egy előre megadott számához szükséges adatok.
`sorszam`: a sor száma, 1-től `meret`-ig tartó szám lehet.
`oszlopszam`: az oszlop száma, 1-től `meret`-ig tartó szám lehet.
`hatar`: 1-től `meret`-ig tartó egész szám, a tekercs szerinti bejárásban határozza meg, hogy az adott rétegben milyen sor és oszlopszámokon belül kell tartózkodni (legalább `meret-hatar`, legfeljebb `hatar`)
`elem`: egy tényleges szám a tekercsben, 1-től `ciklus`-ig tartó szám
`megoldas`: `meret` darab `sor`ból álló lista, a feladvány egy megoldása
`sor`: `meret` darab `ertek`-ből álló lista, a feladvány egy megoldásának egy sora
`ertek`: 0-tól `ciklus`-ig tartó szám, a feladvány mezőjének egy végleges értéke
`t_matrix`: `meret` darab `t_sor`ból álló lista, a feladvány egy folyamatban lévő megoldása.
`t_tekercs`: `t_ertek`-ekből álló lista, a `t_matrix` tekercs szerinti bejárásának egydimenziós kiterítése
`t_sor`: `meret` darab `t_ertek`-ből álló lista, a feladvány egy folyamatban lévő megoldásának egy sora
`t_ertek`: a feladvány folyamatban lévő megoldásában egy mező értéke. Ha már végleges akkor lehet egy `ertek`, ha nem, akkor `ertek`-ek listája, mely a még lehetséges értékek tartományát jelenti
`szukites`: a kizárásos szűkítés módszer eredményét adja, amennyiben egy sorban/oszlopban talált érték szerint szűkítési lehetőséget, akkor `sor(sorszam,ertek)` / `oszl(oszlopszam,ertek)`, ha viszont ezzel a szűkítési módszerrel nem talált szűkítési lehetőséget akkor értéke `nem`.

Predikátumok specifikálása

`tekercs(feladvany_leiro::in, megoldas::out).`

Előállítja a számtekercset reprezentáló mátrixot a kezdő tartományokkal. Ezután elvégzi a lehetséges kezdeti szűkítéseket, majd az ertek_meghatarozasa predikátumra bízza a számtekercs sorrendjének megfelelő értékbeállításokat. Az utolsó érték meghatározása után a helyes megoldásokat egyesével visszaadja.

```
kezdotabla(feladvany_leiro::in, t_matrix::out).
```

kezdotabla/2 az szt feladvány-leíró alapján előállítja az annak megfelelő legáltalánosabb Mx tartomány-mátrixot.

```
tartomany(meret::in, ciklus::in, list(integer)::out).
```

tartomany/3 a feladvány mérete és ciklusszáma alapján meghatározza az alapvető tartományát a mezőknek

```
kezdo_matrix(meret::in, ciklus::in list(adott_elem)::in, list(integer)::in,
t_matrix::out).
```

kezdo_matrix/5 létrehozza a kiindulási tartománymátrixot az adott értékekre szűkítéssel

```
adottak_szukitese(meret::in, ciklus::in, list(adott_elem)::in,
t_matrix::in, t_matrix::out).
```

Előre megadott értékek beállítása a mátrixban és kivétele a szükséges tartományokból. Kezeli azt is ha 0-t adnak meg előre!

```
szukites(meret::in, ciklus::in, t_matrix::in, t_matrix::out).
```

Leszűkíti a mátrixelemek tartományát az egyelemű tartományok alapján amíg tudja, majd végigmegy a sorokon kizárásos szűkítést keresve, majd a kizaras_ellenorzese_sorban predikátumra bízza ennek eredményét, esetleges további szűkítés keresését az oszlopokban. Ezután a tekercs menti szűkítést tekercs_szukites-sel elvégzi, végül a szukites_ellenorzese predikátummal értékeli ki a szűkítési fázist.

```
ismert_szukites_iteracio(meret::in, ciklus::in, t_matrix::in,
t_matrix::out).
```

A szukites_vegrehajtas-t meghívva egyszer végigmegy a mátrixon. Amennyiben a mátrix módosul, újra meghívja magát ezáltal elölről kezdve a mátrix bejárását. Ha nem módosult, akkor visszatér a kapott Mátrix-szal.

```
szukites_vegrehajtas(sorszam::in, oszlopszam::in, meret::in, ciklus::in,
t_matrix::in, t_matrix::out).
```

Végigmegy a mátrixon és az egyelemű tartományok alapján szűkíti a tartományokat. Minden szűkítés után visszalép és újrakezdi a bejárást.

```
egyelemu_ellenorzes(meret::in, ciklus::in, sorszam::in, oszlopszam::in,
t_ertek::in, t_matrix::in, t_matrix::out).
```

Ellenőrzi, hogy az adott elem tartománya egyelemű-e. Ha igen, akkor módosítja az elemet a mátrixban, elvégzi a szükséges módosításokat a sorban és oszlopban, majd visszatér a módosított mátrixszal. Ha nem egyelemű tartomány van, akkor továbblép a mátrixban a következő elemre.

```
elemek_modositasa(list(integer)::in, sorszam::in, t_ertek::in,
t_matrix::in, t_matrix::out).
```

Az mátrix adott sorában kicseréli a listában kapott indexek értékét a kapott értékre.

```
elem_modositasa(sorszam::in, oszlopszam::in, integer::in, t_matrix::in,
t_matrix::out).
```

Módosítja az adott elemet a mátrixban a megadott értékre.

```
elem_csere(sorszam::in, integer::in, t_sor::in, t_sor::out).
```

A megadott lista megadott indexén az elemet lecseréli a megadott értékre és visszatér a módosított listával.

```
sor_es_oszlop_frissites(meret::in, ciklus::in, sorszam::in, oszlopszam::in,
integer::in, t_matrix::in, t_matrix::out).
```

Frissíti a megadott elem sorát és oszlopát az adott elem értékétől függően. Ha az érték 0, akkor a 0-k frissítése történik meg sor0_frissites és oszlop0_frissites segítségével. Ha az érték nem 0, akkor a tartományok frissítése történik meg sor_tart_frissites és oszlop_tart_frissites segítségével.

```
sor_tart_frissites(meret::in, ciklus::in, sorszam::in, oszlopszam::in,
integer::in, t_matrix::in, t_matrix::out).
```

```
oszlop_tart_frissites(meret::in, ciklus::in, sorszam::in, oszlopszam::in,
integer::in, t_matrix::in, t_matrix::out).
```

sor_tart_frissites/7 és oszlop_tart_frissites/7 predikátumok frissítik a sorban/oszlopban a tartományokat az adott elem tartományból kivételével. A sorok és oszlopok frissítése során ha egy elem már nem tömb, akkor azt kihagyjuk. A két predikátum a soron/oszlopon elemenként megy végig 1-től Meret-ig.

```
elem_kivetele(integer::in, t_ertek::in, t_ertek::out).
```

Kiveszi az adott elemet a tartományból. Ha nem tartomány az elem, hanem érték, akkor nem csinál semmit.

```
sor0_frissites(meret::in, ciklus::in, sorszam::in, oszlopszam::in,
t_matrix::in, t_matrix::out).
```

```
oszlop0_frissites(meret::in, ciklus::in, sorszam::in, oszlopszam::in,
t_matrix::in, t_matrix::out).
```

sor0_frissites/6 és oszlop0_frissites/6 frissítik a sorban/oszlopban az összes [0]-t 0-ra, majd megszámolja a 0-kat és ha a szám megegyezik a Meret-Ciklus értékével, akkor a sorban/oszlopban a tartományokból kiveszi a 0-t, ha pedig nagyobb, mint Meret-Ciklus, akkor a mátrixot kicseréli üres tömbre ezzel jelezve, hogy az adott mátrix nem vezethet helyes megoldáshoz. A két predikátum a soron/oszlopon elemenként megy végig 1-től Meret-ig.

```
egyelemu_0(t_ertek::in, t_ertek::out).
```

egyelemu_0/2 lecseréli a [0]-t 0-ra, egyébként nem módosítja az elemet.

```
nullak_szama_sorban(integer::in, meret::in, ciklus::in, sorszam::in,
t_matrix::in, t_matrix::out).
```

```
nullak_szama_oszlopban(integer::in, meret::in, ciklus::in, oszlopszam::in,
t_matrix::in, t_matrix::out).
```

Ha az adott sorban/oszlopban n-m nulla van, akkor az adott sorban/oszlopban a tartományokból kiveszi a 0-t, ha több, akkor a mátrixot kicseréli üres tömbre, ha kevesebb, akkor nem csinál semmit.

```
nullak_szamolasa_oszlopban(meret::in, sorszam::in, oszlopszam::in,
t_matrix::in, integer::in, integer::out).
```

Megszámolja az adott oszlopban levő 0-k számát. Végigmegy az oszlopon 1-től Meret-ig és akkumulátorban tárolja az eddigi 0-k számát.

```
elem_0_ellenorzes(t_ertek::in, integer::out).
```

elem_0_ellenorzes/2 ellenőrzi, hogy az adott elem 0-e, ha igen akkor 1-t ad vissza, ha nem akkor 0-t.

```
vonalmenti_bejaras(sorszam::in, meret::in, t_matrix::in, ertek::in,
ciklus::in, t_matrix::out, sorszam::out, ertek::out).
```

Sorok/oszlopok bejárása minden lehetséges értékkel a szűkítéshez.

```
ertek_elofordulasai(t_sor::in, ertek::in, integer::in, list(integer)::out,
list(integer)::out).
```

Kigyűjti egy listába a kapott listának minden olyan indexét (1-től indexelve), aminek értéke vagy a kapott érték, vagy pedig egy olyan lista, amiben szerepel a kapott érték. Továbbá egy másik listába kigyűjti csak azokat, amik tartományban tárolják az értéket

```
vonal_szukites(sorszam::in, meret::in, t_matrix::in, ertek::in, ciklus::in,
list(integer)::in, list(integer)::in, t_matrix::out, sorszam::out,
ertek::out).
```

Adott sor/oszlop szűkítésének ellenőrzése adott értékre.

```
kizaras_ellenorzese_sorban(meret::in, ciklus::in, t_matrix::in,
sorszam::in, ertek::in, t_matrix::out, szukites::out).
```

A sor alapú vonalmenti bejárás eredményének kiértékelése, esetleges szűkítéskeresés oszlopokban.

```
kizaras_ellenorzese_oszlopban(meret::in, ciklus::in, t_matrix::in,
oszlopszam::in, ertek::in, t_matrix::out, szukites::out).
```

Az oszlop alapú vonalmenti bejárás eredményének kiértékelése.

```
tekercs_szukites(meret::in, ciklus::in, t_matrix::in, t_matrix::out).
```

Tartományok szűkítése a tekeredő vonal mentén mindkét irányba.

```
matrix_tekercs(t_matrix::in, tekercs::out).
```

A mátrix transzponálása és a két mátrix segítségével tekercs létrehozása.

```
matrix_tekercs(t_matrix::out, tekercs::in).
```

Tekercs visszaalakítása mátrixszá.

```
matrix_kiterites(matrix::in, matrix::in, list(integer)::out)
```

Mátrix kiterítése listába a spirális bejárás mentén. Az első és utolsó sort a mátrix segítségével adjuk hozzá, az első és utolsó oszlopot pedig a mátrix transzponáltját használva. A külső réteg levétele után rekurzív hívással határozzuk meg a tekercs hátralevő részét.

```
matrix_karcsusitas(matrix::in, matrix::out)
```

Mátrix frissítése: minden sor első és utolsó elemének kivétele.

```
lista_levagas(list(integer)::in, list(integer)::out)
```

Lista első és utolsó elemének eltávolítása.

```
utolso_pozitiv(ciklus::in, t_tekercs::in, t_elem::in, t_tekercs::out).
```

Adott elemre a tekercsben előző értékek általi szűrés.

```
ciklikus_rakovetkezok(ciklus::in, t_elem::in, t_elem::out).
```

Minden elemre meghatározzuk az utána várt nem 0 értéket.

```
kovetkezo_ertek(ciklus::in, elem::in, elem::out).
```

Meghatározza az elemet követő hozzáadandó értéket: ha elértük a Ciklus méretét, akkor 1-t ad vissza, egyébként 1+Értéket.

```
kovetkezo_utolso_pozitiv(t_elem::in, t_elem::in, t_elem::out).
```

Következő elem megengedett értékeinek meghatározása. Ha az aktuális mező értéke nem lehet 0, akkor csak az aktuális mező megengedett értékei határozzák meg a következő elem megengedett értékeit. Ha az aktuális mező értékei közt van a 0 is, akkor a jelenlegi lehetséges értékek (a 0-t kivéve), és az előző megengedett lehetséges értékei is meghatározhatják a következő mező lehetséges értékeit.

```
utolso_pozitiv_forditott(ciklus::in, t_tekercs::in, t_elem::in,
t_tekercs::out).
```

Adott elemre a tekercsben előző értékek általi szűrés, de fordított irányban, tehát mindig kisebbnek kell következnie.

```
ciklikus_rakovetkezok(ciklus::in, t_elem::in, t_elem::out).
```

Minden elemre meghatározzuk az utána várt nem 0 értéket, a fordított sorrend szerint!

```
elozo_ertek(ciklus::in, ertek::in, ertek::out).
```

Adott nem nulla szám előtti legelső nem nulla érték a tekercs szerinti bejárásban.

```
matrix_inicializalas(meret::in, matrix::out)
```

N*N-es mátrix létrehozása változókkal a visszatekeréshez

```
sor_inicializalas(meret::in, t_sor::out)
```

Meret hosszú lista létrehozása változókkal

```
szukites_ellenorzese(meret::in, ciklus::in, t_matrix::in, szukites::in,
t_matrix::out).
```

Megnézi, hogy a szűkítések előtti és a szűkítések utáni mátrix megegyeznek-e. Ha nem találtunk szűkítési lehetőséget visszatérünk a kapott mátrixszal (ez igaz az üres mátrixra is) Ha találtunk szűkítési lehetőséget, akkor rekurzívan meghívjuk a szukites/4-t.

```
ertek_meghatarozasa(meret::in, ciklus::in, sorszam::in, oszlopszam::in,
hatar::in, elem::in, t_matrix::in, megoldas::out).
```

A kapott helyre beállítja vagy a meghatározott értéket vagy a 0-t, valamint ellenőrzi, hogy a bejárás véget ért-e.

```
ertek_beallitasa(meret::in, ciklus::in, sorszam::in, oszlopszam::in,
hatar::in, elem::in, t_ertek::in, t_matrix::in, megoldas::out).
```

A kapott helyre beállítja a megadott értéket amennyiben lehetséges/szükséges, majd ismételten leellenőrzi a lehetséges szűkítéseket, és továbblép a következő mezőre.

```
nulla_beallitasa(meret::in, ciklus::in, sorszam::in, oszlopszam::in,
hatar::in, elem::in, t_ertek::in, t_matrix::in, megoldas::out).
```

A kapott helyre beállítja a 0-t amennyiben lehetséges/szükséges, majd ismételten leellenőrzi a lehetséges szűkítéseket, és továbblép a következő mezőre.

```
kovetkezo_lepes(meret::in, sorszam::in, oszlopszam::in, hatar::in,
sorszam::out, oszlopszam::out, hatar::out).
```

A tekercsbeli jelenlegi helyzetünk alapján meghatározza a számtekercs bejárása szerinti következő mezőt.

Algoritmus implementációk

```
adottak_szukitese(meret::in, ciklus::in, list(adott_elem)::in,
t_matrix::in, t_matrix::out).
```

Sorban halad a megadott elemeken, és minden i(Sorszam, Oszlopszam, Ertek)-re meghívja az elem_modositasa(Sorszam, Oszlopszam, Ertek, Mx0, Mx1) eljárást. Minden módosítás után elvégzi a sor_es_oszlop_frissites-t, hogy az első szukites meghívásakor ezek már szűkítve legyenek.

```
vonalmenti_bejaras(sorszam::in, meret::in, t_matrix::in, ertek::in,
ciklus::in, t_matrix::out, sorszam::out, ertek::out).
```

A mátrixból kiveszi az adott vonalszámú vonalat és ertek_elofordulasai-val megnézi hogy hányszor van benne tartományban illetve egyértelmű elemként. Ezt átadva a vonal_szukites-nek történik meg az esetleges szűkítés

```
ertek_elofordulasai(t_sor::in, ertek::in, integer::in, list(integer)::out,
list(integer)::out).
```

Sorban végighalad a vonalon, Meretig növeli az Indexet. Ha az adott elem lista és benne van a keresett elem akkor hozzáadja a mező indexét a vonalon az Elofordulasokhoz is és az ElofordulasokListaban-hoz is. Ha egyelemű elemként van ott akkor csak az

Elofordulasokhoz. Ha nincs benne, akkor nem adja hozzá egyikhez sem. Megy a következő Indexre ha van még.

```
vonal_szukites(sorszam::in, meret::in, t_matrix::in, ertek::in, ciklus::in,
list(integer)::in, list(integer)::in, t_matrix::out, sorszam::out,
ertek::out).
```

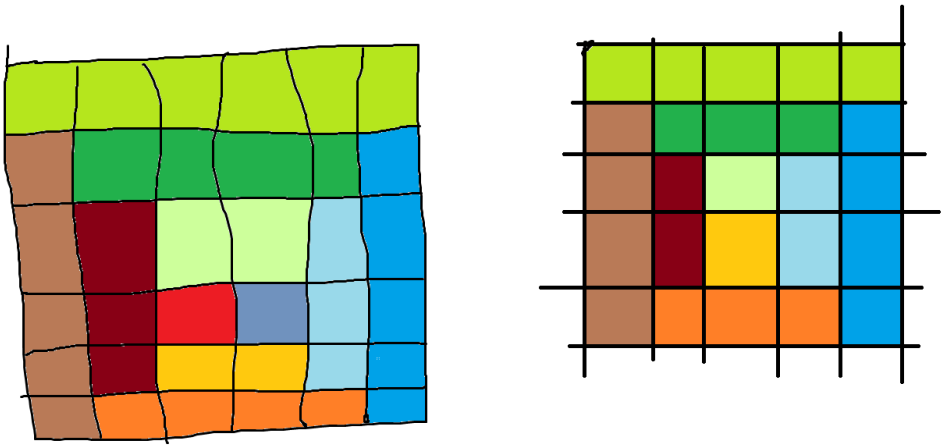
Megadott vonal szűkítésének ellenőrzése a megadott értékre. Ha az érték 0 és Elofordulasok mérete kisebb mint Meret-Ciklus, akkor a feladatnak nincs megoldása. Ha az érték 0, Elofordulasok mérete egyenlő Meret-Ciklussal, és ElofordulasokListaban mérete nagyobb, mint 0, akkor minden listát, ami tartalmazza a 0-t kicseréljük [0]-ra. Ezt a frissítést a mátrixban is megtéve visszatérünk vele és a szűkítés értékeivel. Ha az érték nem 0 és Elofordulasok mérete 0, akkor a feladatnak nincs megoldása. Ha az érték nem 0, Elofordulasok mérete 1, és ElofordulasokListaban mérete is 1, akkor átírjuk [Ertek]-re, frissítjük a mátrixot és visszatérünk vele. (Itt akár lehetne rögtön berakni az értéket és meghívni a sor_tart _frissites-t). Megjegyzés: a Szukites-t a feladat nem használja fel, de bizonyos körülmények között hasznos lehet (lásd: khf 6). Egyébként pedig haladunk tovább a vonalmenti bejárásban a következők szerint: Ha még nem érte el a vonal utolsó értékét, akkor az adott vonalat nézzük eggyel nagyobb értékkel. Ha elérte az utolsó értéket, de még nem az utolsó vonalon van, akkor a következő vonalat nézzük 0 értékkel. Ha elérte az utolsó vonalat az utolsó értékkel úgy, hogy nem talált szűkítést, visszatér a Mátrixszal

```
kizaras_ellenorzese_sorban(meret::in, ciklus::in, t_matrix::in,
sorszam::in, ertek::in, t_matrix::out, szukites::out).
```

vonalmenti_bejaras/8 után érdemes hívni annak az eredményét átadva, így ez leellenőrzi, hogy a sorokban sikerült-e szűkíteni a módszerrel. Ha nem sikerült, akkor transzponálja a mátrixot, így az oszlopok lesznek a sorok, és a vonalmenti_bejaras újbóli meghívásával már az oszlopokat is tudja ellenőrizni. Ebben az esetben a szűkítés eredménye után visszatranszponálja a mátrixot és a kizaras_ellenorzese_oszlopban-ra bízza az oszlopmenti szűkítés eredményét. (Jelen feladatban nincs jelentősége.)

```
matrix_kiterites(matrix::in, matrix::in, list(integer)::out)
```

Mátrix kiterítése listába a spirális bejárás mentén. Megkapja az aktuális mátrixot és annak transzponáltját. A mátrixból leveszi az első sort (réteg teteje), a transzponáltból is leveszi az első sort az első elemének kivételével, majd sor megfordításával (réteg bal széle), a mátrixból leveszi az utolsó sort az első elem kivételével, majd megfordítja és az így keletkező első elemet is kiveszi (réteg alja). A transzponált utolsó sorát is kiveszi és leveszi róla az első elemet (réteg jobb széle). Ezeket a megfelelő sorban összefűzve (teteje, jobb széle, alja, bal széle) megkapja az adott réteget. Ezt fogja hozzáfűzni ahhoz, amit a rekurzív hívásból kap, de ahhoz előbb létre kell hozni a mátrixot ami a réteg belsejében maradt. Már magánál az illesztésnél leszedtük az első sorokat -> Mtx és MtxT és amikor ezeknek az utolsó sorát meghatároztuk, megkaptuk az utolsó sorok nélküli mátrixot is AlacsonyMtx és AlacsonyMtxT. Mindkettőre meghívva a matrix_karcsusitast, ami minden sorból kiveszi az első és utolsó elemet, létre is jön az n-2 x n-2 méretű belső mátrix, amire lehet indítani a rekurzív hívást.



3. és 4. ábra: Mátrix kiterítésének illusztrálása páros és páratlan méretű mátrix esetén.

```
ertek_meghatarozasa(meret::in, ciklus::in, sorszam::in, oszlopszam::in,
hatar::in, elem::in, t_matrix::in, megoldas::out).
```

Ha a Hatar -1, azaz a végére értünk, és a Mátrix nem üres tömb, akkor visszatér a kapott megoldással. Különben viszont kiveszi a sor-és oszlopszám szerinti értéket a mátrixból és azt továbbadva meghívja az ertek_beallitasa és a nulla_beallitasa predikátumokat két külön ágon!

```
ertek_beallitasa(meret::in, ciklus::in, sorszam::in, oszlopszam::in,
hatar::in, elem::in, t_ertek::in, t_matrix::in, megoldas::out).
```

Ha a kapott helyen már van érték, és az van ott, amit be akartunk állítani, akkor meghatározzuk a következő értéket és mezőt és meghívjuk az ertek_meghatarozasat. Ha olyan tartomány van az adott helyen amiben benne van az elemünk, akkor módosítjuk azt az értékre, és meghívjuk a szukites-t. (A gyorsítás érdekében először a sor_tart_frissitest és oszlop_tart_frissitest). Ezután következő érték és mező meghatározása, majd ertek_meghatarozasa hívása.

```
nulla_beallitasa(meret::in, ciklus::in, sorszam::in, oszlopszam::in,
hatar::in, elem::in, t_ertek::in, t_matrix::in, megoldas::out).
```

Ha a kapott helyen már van érték, és az 0, akkor meghatározzuk a következő mezőt és meghívjuk az ertek_meghatarozasat azzal az értékkel, ami jelenleg is volt. Ha olyan tartomány van az adott helyen amiben benne van a 0, akkor módosítjuk azt 0-ra, és meghívjuk a szukites-t. (A gyorsítás érdekében először a sor_0_frissitest és oszlop_0_frissitest). Ezután következő mező meghatározása, majd ertek_meghatarozasa hívása.

```
kovetkezo_lepes(meret::in, sorszam::in, oszlopszam::in, hatar::in,
sorszam::out, oszlopszam::out, hatar::out).
```

A tekercsbeli jelenlegi helyzetünk alapján meghatározza a számtekercs bejárása szerinti következő mezőt. Az alap gondolat az, hogy tároljuk a jelenlegi mezőt és egy értéket, hogy a mátrixban hanyadik oszlopig illetve hanyadik sorig mehet el (Hatar). Ezt felhasználjuk a minimum sor és a minimum oszlop (Meret-Hatar+1) meghatározásához is.Az alábbi esetek lehetségesek, az implementáció garantálja, hogy a felsorolás sorrendjében csak a legelső illeszkedő eset fut le.

Ha az aktuális réteg felső sorában vagyunk és még nem értünk az utolsó oszlophoz, akkor jobbra haladunk. (S = M-H+1 és O < H → O = O+1)

Ha az aktuális réteg jobb szélén vagyunk és még nem értünk az utolsó sorhoz, akkor lefelé haladunk. (O = H és S < H → S = S+1)

Ha az aktuális réteg alsó sorában vagyunk és még nem értünk az utolsó oszlophoz, akkor balra haladunk. (S = H és O > M-H+1 → O = O-1)

Ha az aktuális réteg bal szélén vagyunk és még nem értünk a második sorhoz, akkor felfelé haladunk. (O = M-H+1 és S > M-H+2 → S = S-1)

Ha az aktuális réteg bal szélén elértük az utolsó elemet is, és ez nem egy páros mátrix utolsó mezője (ezt úgy lehet leellenőrizni, hogy páros mátrix esetén 2x2-es mátrix az utolsó a bal alsó mezőjével, tehát ott az O eggyel kisebb mint a H), akkor a következő rétegre lépünk jobbra, a határt csökkentjük.
(O = M-H+1 és S = M-H+2 és H != O+1 → O = O+1 és H = H-1)

Különben pedig a mátrix végére értünk → O = -1, S = -1, H = -1

Tesztelési megfontolások

A megoldás implementálását egyértelműen könnyítette, hogy az egyelemű tartományokat nem listaként, hanem integerként tároltuk. Ezáltal elég volt az adott elem lista voltát ellenőrzini.

A ’vakon’ tesztelés nehezítette a feladatmegoldást, ugyanis a kapott tesztek inkonzisztensek voltak a feladatkiírással (tesztekben adott_elem értéke többször is 0 volt, a specifikáció pedig azzal kezd, hogy adott nxn-es mátrix 1 és m közötti számokkal, illetve a kiadott bemenet ellenőrző függvény is így ellenőrzött kezdetben).
A tesztesetek kiadását követően egészen hamar sikerült észrevenni a hibát és apró javítással elérni a helyes működést (eredetileg mindig csak kivettem a sorból és oszlopból az értéket, ezután már az értéket és 0-t külön kezelő sor_es_oszlop_frissites-t futtattam). Habár így a határidőhöz közeledve kellett megoldani a feladatot és stresszesebb volt, érdekes megfigyeléseket tettem (lásd következő fejezet).

A trace haasználata habár a sok függvényhívás és rekurzió miatt nem mindig hatékony, de egyszerűbb példákon kifejezetten hasznos, hogy apróbb hibákat észre lehessen venni.

A felkiáltójelek használata, habár nagyban könnyíti a klózok írását (nem kell mindenhova diszjunkt feltételeket írni), azért óvatosan kell használni, főleg akkor, ha az ember esetleg szeretné, hogy több klózra is illeszkedjen az eljárás folyamata. (ertek_meghatarozasa)

Kipróbálási tapasztalatok

A nagyházi feladatot kezdetben úgy terveztem megoldani, hogy csak az ismert szűkítést és a kizárásos szűkítést végzem el rajta, illetve esetszétválasztok. Mivel az előre megadott 0-kat nem teszteltem így hibás teszteredményeket is kaptam, de a hiba forrását a tesztek kiadásáig nem fejtettem meg. Így hát alternatív megoldásba kezdtem, és elkészítettem egy olyan programot, ami szinte semennyire sem szűkít (csak az ismert szűkítés szerint és arra se kimerítően) és a 0-kat is máshogy kezeli (egy egy listában tárolom a sorokban, illetve oszlopokban található 0-k számát, így a tartományban egyáltalán nem volt 0). És leginkább a generate_and_test optimalizált verziójára hagyatkozik (lásd: nhf1 Elixir Számtekercs). Mivel a 0-kat itt sem kezeltem ez is hibás volt (és persze lassú is). A tesztesetek kiadását követően hamar észrevettem a hibát és javítottam is, először a khf-eket nem felszhasználó programomban. A hibák javítását követően már nem volt hibás teszt, csak olyan ami az időkorlátot túllépte. Megpróbáltam ezután optimalizálni a kizárásos szűkítéssel, de a struktúra nem volt a legalkalmasabb hozzá, így visszatértem az eredeti programomhoz. A hiba javítását követően abban is minden sikertelen teszt időtúllépés miatt lett sikertelen.

A meglepő tapasztalatom viszont az volt, hogy a generate_and_test optimalizált módszere a helyesen megoldott feladatokat jelentősen gyorsabban oldotta meg (~1s), mint az ismert és kizárásos szűkítéssel implementáld clpfd program (~7-10s).

Ugyanakkor határozottan kevés volt mindkét változat, a tekercsmenti szűkítésre (khf7) mindenképp szükség volt a teljes tesztkészlet sikeres végrehajtásához (~1s) tehát valószínűleg a három szűkítés együttesen elegendő ahhoz, hogy a feladatmegoldás ne legyen exponenciális probléma, hanem csak polinomiális. (De mindenképp érdemes lehet megnézni, hogy a 3 szűkítésből csak 2 használata is elegendő-e.)

További érdekes megfigyelés, hogy volt olyan teszt, ami a tekercsmenti szűkítéssel is 7s-ba telt, míg sikerült megoldani, a tekercsmenti fordított irányba tett szűkítéssel együtt ugyanakkor ez is leredukálódott egy másodperc alá.