Nagyhatékonyságú Deklaratív Programozás

Nagy házi feladat: Sudoku távolságkorlátokkal

Készítette: Bobula Dalma

Követelmények elemzése

A feladatunk az volt, hogy egy sudoku solvert valósítsunk távolságkorlátokkal. A később prologban megfogalmazandó szabályaink/korlátaink a következők voltak:

Általános sudoku követelmények:

- Az végeredmény mátrix csak 1..L számokat tartalmazhat, ahol $L = K^2$ (K – cellaméret).
- Minden sor minden 1..L számot csak egyszer tartalmazhat, másképp fogalmazva a sor minden eleme legyen különböző. (A két állítás esetünkben ekvivalens, mivel a sor hossza N).
- Minden oszlop összes eleme legyen különböző.
- Minden cella összes eleme legyen különböző.

A ’D’ infómátrix egy cellája 3 féle dolog bármely kombinációját tartalmazhatja: Számot, s és w karaktert. Emellett bemenetként kapunk egy N távolságkonstanst.

Infómátrix megkötései:

- Ha D egy cellájában egy szám van, akkor a megoldásban is ott annak a számnak kell szerepelnie.
- Ha egy s karakter van a cellában, akkor az eredmény mátrixban ugyanazon a helyen lévő elem és a tőle délre lévő elem különbségének abszolútértéke legyen N. (délre = Egy sorral lejjebb)
- Ha egy w karakter van a cellában, akkor az eredmény mátrixban ugyanazon a helyen lévő elem és a tőle nyugatra lévő elem különbségének abszolútértéke legyen N. (nyugatra = Egy oszloppal balra).
- Ezen megkötések tetszőleges kombinációja a szabályaik ugyanazon kombinációját jelentsék. (Pl. 4w --> Az elem legyen 4 és abs(4-W) = N, ahol W a nyugatra lévő elem)

A fenti távolság szabályok ’komplementerei’ is további követelményeket jelentenek:

- Ha nincs egy cellában s karakter, akkor az elemtől délre lévő elemmel vett különbség se lehet egyenlő N-nel.
- Ha nincs egy cellában w karakter, akkor az elemtől nyugatra lévő elemmel vett különbség se lehet egyenlő N-nel.
- És ezek kombinációjának is teljesülni kell. Tehát akkor a fentebbi példánk ezt fogja jelenteni: 4w --> Az elem legyen 4 és abs(4-W) = N és abs(4-S) \= 4 , ahol S a délre lévő elem

Tervezés, az egyes eljárások specifikálása

```
sudoku(s(N,T), Rows)
```

Ez a feladatban elvárt eljárás fejléce, maga a sudoku solver, mely egy s(N,T) struktúrát vár és eredményét a Rows-ba tölti.

```
apply_dist(M,N,D,I,J,C)
```

Az apply_dist eljárás feladata, hogy a távolság szabályokat felvegye minden elemre.

Input: M – mátrix, N – távolság érték, D – infómátrix, I – aktuális sor indexe, J – aktuális oszlop indexe, C – index/counter

output: - (felvett szabályok)

```
nth0_mtx(M,I,J,E)
```

Egy mátrix I-dik sorának J-dik elemét rakja bele E-be.

```
dist_to_be(X, Y, N) +:
```

FD predikátum, mely azt fejezi ki, hogy X és Y különbségének az abszolút értéke legyen N.

```
dist_not_be(X, Y, N)
```

Ez az eljárás azt köti ki, hogy X és Y különbségének abszolút értéke ne legyen N.

```
dist(D,E,S,W,N)
```

A dist eljárás egy adott E1 cellára köti ki a távolság korlátokat D lista alapján.

input: E – aktuális cella, D – infómátrix E cellájához tartozó mezője, S – déli szomszéd, W – nyugati szomszéd, N – távolság érték

output: - (megkötések)

```
shave(X, Rest, BB0, BB)
```

A borotválásért felelős eljárás, mely az X változó és a többi változó értékét is borotválja.

```
shave_var(V)
```

A V változó tartományát szűkíti borotválás segítségével, tehát azon értékeket, melyek amúgyis meghiúsuláshoz vezetnének, kiveszi.

```
shave_vars(L)
```

L változó lista elemein iterál végig rekurzívan és mindegyikre meghívja a shave_var eljárást

```
get_block(M,I,J,K,L,IC,JC,LR)
```

Ez az eljárás azért felel, hogy az adott koordináták alapján visszaadja oszlop folytonosan az adott blokkot.

```
make_blocks_distinct(S,I)
```

Ez az eljárás azért felel, hogy az összes blokkra kikösse az elemek egyediségének a szabályát. Ehhez először a blokkokat le kell kérni a get_block segítségével.

Megvalósítás

Korlátok megválasztása

A követelményekben megfogalmazott szabályokat fogom itt leírni.

- Legyen az eredmény mátrix mérete akkora, mint az infó mátrixé:

```
length(T, L), length(Rows, L),
```

- Az eredmény mátrix minden sora legyen olyan hosszú, mint magának a mátrix mérete:

```
maplist(same_length(Rows), Rows),
```

- A mátrix kilapított listájának domain-je 1..L legyen, ahol L a mátrix mérete:

```
append(Rows, Vs), domain(Vs, 1, L),
```

- A mátrix minden sorában lévő elemek legyenek különbözőek:

```
maplist(all_distinct, Rows),
```

- A mátrix minden oszlopában lévő elemek legyenek különbözőek:

```
transpose(Rows, Columns),
maplist(all_distinct, Columns),
```

- A mátrix minden blokkjában lévő elemek legyenek különbözőek:

```
make_blocks_distinct(Rows, 1),
```

Most pedig jöjjenek a távolságra megfogalmazott szabályok, amelyeket az apply_dist hoz érvénybe. Én a távolság szabályokat 5 külön esetre bontottam fel: [], [s], [w], [s,w], [A|Tail]

```
%Ha nincs semmilyen infó, akkor a komplementer szabályok legyenek érvényben
dist([],E,S,W,N):-
    dist_not_be(E, S, N),
    dist_not_be(E, W, N).
%Ha egy s van, akkor déli szomszédra teljesüljön a távolság korlát, viszont a
nyugatinál semmilyen esetben ne.
dist([s],E,S,W,N):-
    dist_to_be(E, S, N),
    dist_not_be(E, W, N).
%Mindkettő szabály teljesüljön a távolságra vonatkozóan
dist([s,w],E,S,W,N):-
    dist_to_be(E, S, N),
    dist_to_be(E, W, N).
%Csak a nyugati teljesülsön, a déli ne
dist([w],E,S,W,N):-
    dist_not_be(E, S, N),
    dist_to_be(E, W, N).
%Ha szám és vmi egyéb van az infó mezőben akkor az elemet kössük le erre az
értékre és az egyéb feltélekkel pedig hívjunk vissza egy feljebb definiált
változatra
dist([A|Tail],E,S,W,N):-
    integer(A),
    E #= A,
    dist(Tail, E, S, W, N).
```

```
%segéd eljárások amiket a dist hív:
%Skaláris szorzat segítségével fejezzük ki két elem különbségét, aminek az
abszolút értékét már nyugodt szívvel le lehet kötni, tartomány-konzisztensen
szűkít az eljárás.
dist_to_be(X, Y, N) :-
    scalar_product([1,-1],[X,Y],#=,D,[consistency(domain)]),
    abs(D) #= N.

dist_not_be(X, Y, N) :-
    scalar_product([1,-1],[X,Y],#=,D,[consistency(domain)]),
    abs(D) #\= N.
```

Algoritmusok ismertetése

sudoku: Bemeneti és kimeneti paramétere ismert és műveleteit a korlátoknál meg is fogalmaztuk.

```
sudoku(s(N,T), Rows) :-
    length(T, L),
    length(Rows, L),
    maplist(same_length(Rows), Rows),
    append(Rows, Vs),
    domain(Vs, 1, L),
     %Minden sor elem legyen egyedi
    maplist(all_distinct, Rows),
     %Minden oszlop elem legyen egyedi
    transpose(Rows, Columns),
    maplist(all_distinct, Columns),
     %Minden blokk elem legyen egyedi
    make_blocks_distinct(Rows, 1),
     %Teljesüljenek a távolság sszabályok
    apply_dist(Rows,N,T,0,0,0),
     %Futassuk a kiértékelőt First fail, borotválás módszerrel a hatékonyság
miatt
    labeling([ff, value(shave)], Vs).
```

apply_dist: Végig iterál a mátrix elemein egy countert használva, és minden elemre meghívja a dist eljárást. Túlindezelés elkerülése végett 4 blokk alakul ki annak függvényében, hogy első oszlop és/vagy utolsó sorban vagyunk e. (Szomszédok lekérése miatt)

shave:

```
shave(X, Rest, BB0, BB):-
     BB0 = BB,
     %Minden további változóra borotváljunk
    shave_vars(Rest),
     %Aktuális változóra borotváljunk
     shave_var(X),
     %Kössük le X változó tartományát (Szűkítés)
     indomain(X).


shave_var(V):-
     %Kérjük le V változó tartományát
     fd_set(V, Dom),
     %Vegyük ennek a tartomány elemeit egyesével (C – konstans)
     fdset_member(C, Dom),
     ( \+ V = C -> V #\= C % meghiúsulás esetén zárjuk ki a tartományból C-t
     ; fail
     ), !.
```

```
shave_var(_).
```

dist: Bemutatásra került feljebb működése.

Tesztelési megfontolások

Amíg nem volt helyesen működő a sudoku solverem, addig nem tudtam a keretprogramot tesztelésre használni hatékonyan, ezért saját teszteseteket írtam az egyes eljárásaim tesztelésére. Viszont mikor már volt egy működő verzióm onnantól kezdve végig a keretprogramot használtam, hogy mindig ellenőrizni tudjam, hogy a refaktorálások során nem változott-e a működés. Továbbá, a kód hatékonyságának a növeléséhez sokszor figyeltem, hogy gyorsulnak-e az egyes tesztesetek lefutási sebességeik.

Kipróbálási tapasztalatok

Nagyon hasznos volt a házifeladathoz kapott keretprogram, sokat könnyített a helyesség és gyorsaság ellenőrzéseken. Jó érzés volt nézni, ahogy a felvett szabályok által egyszer csak kiesik a helyes megoldás(ok) a kiértékelőnek hála.