Szimulációs feladatok programozási versenyeken

Zsakó László[1], Szlávi Péter[1]

[1] {zsako,szlavi}@ludens.elte.hu
ELTE IK

Absztrakt. Egyre gyakrabban jelennek meg szimulációs feladatok középiskolásoknak szóló programozási versenyeken. Mivel a versenyek egy jelentős része automatikus kiértékelésű, ezért sok esetben diszkrét, determinisztikus események szimulációjával találkozhatunk. A cikk áttekinti, hogy a magyarországi informatika versenyeken hogyan jelennek meg ezek a feladatok.

Kulcsszavak: számítógépi szimuláció, közoktatás, versenyfeladat

1. Bevezetés

A modell rendszerint bonyolult, részleteiben nem ismert rendszerek működésének megismerésére készített sematikus elképzelés, amelyből új összefüggésekre lehet következtetni, vagy amely alkalmas arra, hogy a rendszer jelenségei matematikailag leírhatók legyenek. A modell a valódi rendszereknek többnyire csak főbb tulajdonságait tükrözi, egyszerűsített formában.

A modellalkotás első lépésében meg kell határozni a modellben szereplő objektumokat, amelyeket meg kell feleltetni a valós rendszer objektumainak (objektumai egy-egy osztályának). Ez a megfeleltetés általában állapotaik megfeleltetését jelenti. Ahhoz ugyanis, hogy objektumokról külön-külön beszélhessünk, szükség van individuális létezésükre, amelyet állapotaik megadásával helyettesítünk.

Ezután következő feladat a rendszer állapotváltozását (az objektumok számának változását, állapotainak változását) leíró algoritmus elkészítése.

Az állapotváltozás alapján beszélhetünk determinisztikus, illetve sztochasztikus szimulációról. Az automatizált értékelésű versenyeken egyértelműen könnyebb a determinisztikus szimuláció értékelése, emiatt lényegében ezek fordulnak elő a programozási versenyeken. Elvileg elképzelhető lenne olyan sztochasztikus szimuláció is, amely sztochasztikus egyensúlyi állapota a kezdőállapotból egyértelmű, így az eredmény alapján történő ellenőrzés elvégezhető, de a magyar versenyeken ilyenekkel még nem találkoztunk.

A modelleket két nagy osztályba soroljuk:

- Az egyikben a teljes jövőt előre kiszámítjuk (a kezdőállapotból), s azután csak a kiszámított jövő megjelenítése a feladat.
- A másikban az aktuális állapotból csak a következő időegységbeli állapotot határozzuk meg, majd abból számítjuk a következőt ...

Példa

Ha egy bolygó Nap körüli mozgását akarjuk szimulálni, akkor eljárhatunk úgy is, hogy egy pillanatbeli helyzetéből, sebességvektorából, a bolygó és a Nap tömege segítségével meghatározzuk a pálya egyenletét – „tiszta” matematikai megfontolásokkal –, majd ezt megjelenítjük. Eljárhatunk azonban úgy is, hogy az aktuális helyzetből és sebességből a bolygóra ható

pillanatnyi erők segítségével meghatározzuk a bolygó helyét és sebességét a következő időegységben, majd ebből számoljuk az ezt követő időegység adatait.

A programozási versenyeken egyértelműen a második fajtával találkozhatunk.

A valós rendszerek, amelyeket modellezni szeretnénk, olyanok lesznek, hogy a rendszer elemei osztályokba sorolhatók, a rendszer állapotát ezen osztályok elemszáma fogja meghatározni, illetve egyes esetekben ezen osztályok térbeli eloszlása, mintázata.

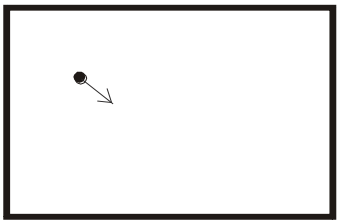
Egyszerűbb esetekben csak egy osztályt vizsgálunk, s minden elemet ebbe sorolunk be, máskor minden osztályban csupán egy elem lesz. A legegyszerűbb pedig az az eset, amikor egy osztály létezik egy elemmel. Ekkor tulajdonképpen egy (néhány) képletet kell folyamatosan számolnunk. [1]

2. Elemi szimuláció

A magyarországi versenyeken először az Izsák Imre Gyula matematika-fizika-informatika versenyen jelent meg ez a szimulációs feladattípus, amit az alábbi két példa illusztrál. (Ezen a háromtusa versenyen a diákok három tárgyból versenyeznek, összetett és tantárgyankénti eredményt is számolnak. A három tantárgy gyakran ad egymáshoz közel álló feladatokat, így pl. az informatikában gyakoriak a fizikai szimulációs feladatok.)

Feladat

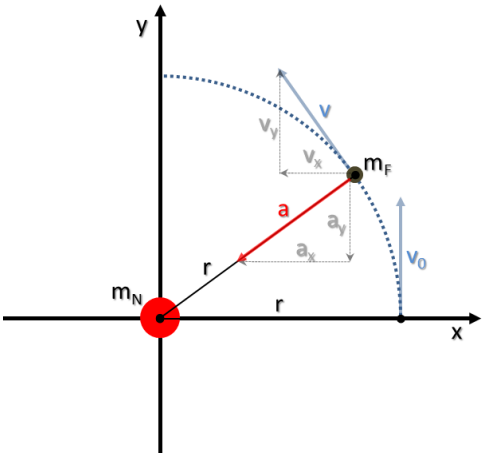
Egy N×M-es ($1 \le N,M \le 1000$) téglalap alakú asztalon egy golyót helyeztünk el az (x,y) pozícióra. A golyó időegységenként (dx,dy) távolságot tesz meg, az asztal széléről szabályosan visszaverődik. A golyó a súrlódás miatt lassul, a sebessége időegységenként L%-kal csökken. Készíts programot, amely követi a golyó útját! [2]



1. ábra: A feladatot magyarázó ábra

Feladat

Készíts programot bolygómozgás szimulálására! A képernyő közepén álljon a nap, melynek tömege: $1{,}989 \cdot 10^{30}$ kg – ez legyen a programban beépített konstans! Vegyünk fel egy bolygót, amelynek megadjuk a Naptól vett távolságát és tömegét (a Föld esetén a távolság 150 millió km és $5{,}972 \cdot 10^{24}$ kg), valamint a pályamenti sebességét (a Föld esetén 29,8 km/másodperc)! Olyan nagyítást kell találni, hogy a Naptól 300 millió km-re levő bolygó még éppen elférjen a képernyőn! [2]



2. ábra: A feladatot magyarázó ábra

Hasonló elemi szimulációs feladatokkal találkozhatunk olyan esetekben, amikor egy eszköz (pl. robot) útját vagy tevékenységét kell követni, azaz egy utasítássort végrehajtani. [3]

Robotversenyeken tipikus lehet ez a feladat, de előfordul Logo programozási versenyeken is. Egy feladat a 2014/15-ös Logo versenyről, 3-4. osztályos tanulók számára. [4]

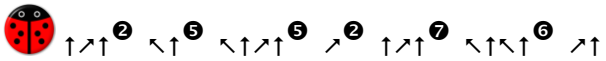
Feladat

A szegedi katicabogarat Muszka Dániel tervezte és építette 1956-57-ben.

3. ábra: A feladatot magyarázó ábra

A elején található három fényérzékelő közül a megfelelőre világítva robotocskánk – helyben maradva – jobbra vagy balra kanyarodik derékszögben, illetve előre halad egy négyzetrácsnyi távolságot.

Rajzold le a katica útját, az alábbi jelsorozat alapján!



3. Sok elem szimulációja

Amivel a továbbiakban foglalkozunk: sok elem mozog térben (síkban) egymástól függően, miközben állapotukat változtathatják. [5]

A folyamatleírás során azt határozzuk meg, hogy egy elemmel mi történhet. Megvalósítási lehetőség:

- időléptetés (minden időegységben történik mindenkivel valami)
  - elemenkénti vizsgálat
  - helyenkénti vizsgálat
- eseményléptetés (a következő esemény időpontjára lépünk, és azt végrehajtjuk, egyes események más eseményeket generálhatnak, más események bekövetkezési időpontját megváltoztathatják)

A párhuzamosság problémája: a valós világ párhuzamosságát a számítógép szekvenciális működésére kell átalakítani úgy, hogy az eseményeket a programbeli sorrend ne befolyásolja!

Feladat – havazás

A szimulációs tér egy mátrix, ahova fentről hópelyhek lépnek be. A hópelyhek időegységenként egyet lépnek lefelé (egyszerre – azaz párhuzamosan). Ha alulra érnek, vagy már lent álló hópehely fölé érnek, akkor 3 jelenség történhet (az alábbi sorrendben):

- ha balra lefelé léphet, akkor oda lép;
- ha jobbra lefelé léphet, akkor oda lép;
- helyben marad.

Megvalósítási lehetőségek:

- időléptetés: Ez a természetes, a hópelyhek időegységenként lépnek egyet
  - elemenkénti vizsgálat: minden hópehelyre – mi történhet?
  - helyenkénti vizsgálat: minden helyre – mi történhet?
- eseményléptetés: Lehetne esemény az, amikor a hópehely a végleges helyére kerül, de ezt nehéz előre kiszámolni. A belépéskor minden hópehely kap egy becsült időt, amikorra megállapodik. Ezt az időpontot menet közben más hópelyhek növelhetik.

Párhuzamosság megoldása: előbb léptetjük azt, ami a mozgásával másokat akadályozhat.

3.1. Szimuláció megvalósítása – objektumonkénti szemlélet

Az előbbi, „havazás” feladatot először hópelyhenként szemlélve oldjuk meg.

Ábrázolás:

`N,M` – a tér méretei

`DB` – a hópelyhek száma

`H(1..DB)` – a hópelyhek sor- és oszlopkoordinátái

Párhuzamosság: ha a hópelyheket belépési idejük szerinti sorrendben vizsgáljuk, akkor az akadályozó előbb léphet, mint az akadályozott.

```
Eljárás Szimulációs lépés:
  Ciklus i=1-től DB-ig
    Ha H(i).sor<N akkor
      Ha szabad(H(i).sor+1,H(i).oszlop)
         akkor H(i).sor:=H(i).sor+1
      különben ha H(i).oszlop>1 és szabad(H(i).sor+1,H(i).oszlop-1)
         akkor H(i).sor:=H(i).sor+1; H(i).oszlop:=H(i).oszlop-1
      különben ha H(i).oszlop<M és szabad(H(i).sor+1,H(i).oszlop+1)
         akkor H(i).sor:=H(i).sor+1; H(i).oszlop:=H(i).oszlop+1
  Ciklus vége
  Belépés az 1. sorba
Eljárás vége.

Függvény szabad(sor,oszlop):
  j:=1
  Ciklus amíg j≤DB és nem(sor=H(j).sor és oszlop=H(j).oszlop))
    j:=j+1
  Ciklus vége
  szabad:=j>DB
Függvény vége.
```

A belépés balról jobbra sorrendben történjen!

```
Eljárás Belépés az 1. sorba:
  Ciklus j=1-től M-ig
    Ha van belépés akkor DB:=DB+1; H(DB).sor:=1; H(DB).oszlop=j
  Ciklus vége
Eljárás vége.
```

Megjegyzés: a belépés vizsgálata egy függvény, igaz értékű, ha az adott időpontban van belépő hópehely a `j`. oszlopba. Megvalósítása a szimuláció szempontjából lényegtelen, így itt nem foglalkozunk vele.

3.2. Szimuláció megvalósítása – helyenkénti szemlélet

Érdemes egy másik megoldást is meggondolni. A szemléletmód változik most: a hely lesz a „vezérlő gondolat”.

Ábrázolás:

`N,M` – a tér méretei

`T(1..N,1..M)` – a szimulációs tér, belépés az első sorba

`T(1..N+1,1..M)` – az első alatti sor kitöltve „álló” hópelyhekkel, így a hópelyhek megállása egységesen kezelhető

`T(i,j)=0`, ha nincs ott hópehely; `=1`, ha van ott hópehely.

Párhuzamosság: ha a teret alulról felfelé haladva vizsgáljuk, akkor ami akadályozhat, azt előbb mozgatjuk, mint azt, amit akadályoz.

```
Eljárás Szimulációs lépés:
  Ciklus i=N-1-től 1-ig -1-esével
    Lefelé lépés az i. sorból
    Balra lefelé lépés az i. sorból
    Jobbra lefelé lépés az i. sorból
  Ciklus vége
  Belépés az 1. sorba
Eljárás vége.
```

A mozgás sorrendje a szabályok sorrendjének felel meg.

Kérdés: mi a teendő, ha van olyan hely, ahova egyszerre jönnének balról és jobbról is? Egy valósághű szimulációban véletlenszerűen döntenénk a két lehetőség között. A programozási versenyeken, az objektív értékelés miatt azonban az nem lehet megoldás. A feladat szövege szerint – a valóságos jelenségtől némileg eltérően – a balra lépésnek elsőbbsége van.

```
Eljárás Lefelé lépés az i. sorból
  Ciklus j=1-től M-ig
    Ha T(i,j)=1 és T(i+1,j)=0 akkor T(i+1,j):=1; T(i,j):=0
  Ciklus vége
Eljárás vége.

Eljárás Balra lefelé lépés az i. sorból
  Ciklus j=1-től M-ig
    Ha T(i,j)=1 és T(i+1,j-1)=0 akkor T(i+1,j-1):=1; T(i,j):=0
  Ciklus vége
Eljárás vége.

Eljárás Jobbra lefelé lépés az i. sorból
  Ciklus j=1-től M-ig
    Ha T(i,j)=1 és T(i+1,j+1)=0 akkor T(i+1,j+1):=1; T(i,j):=0
  Ciklus vége
Eljárás vége.

Eljárás Belépés az 1. sorba:
  Ciklus j=1-től M-ig
    Ha van belépés akkor T(1,j):=1
  Ciklus vége
Eljárás vége.
```

4. Automata szimuláció

A fajta szimulációnak van két klasszikusa: a Conway-féle „életjáték” és az Ulam nevével fémjelzett „szimmetriajáték”.

4.1. Életjáték

A Conway-féle életjátékban az N×N-es négyzetrács mezőit celláknak, a korongokat sejteknek nevezzük. Egy cella környezete a hozzá legközelebb eső 8 mező (tehát a cellához képest „átlósan” elhelyezkedő cellákat is figyelembe vesszük). Egy sejt/cella szomszédjai a környezetében lévő sejtek. A játék körökre osztott, a kezdő állapotban tetszőleges számú (egy vagy több) cellába sejteket helyezünk. Ezt követően a játékosnak nincs beleszólása a játékmenetbe. Egy sejttel (cellával) egy körben a következő három dolog történhet:

- A sejt túléli a kört, ha két vagy három szomszédja van.

- A sejt elpusztul, ha kettőnél kevesebb (elszigetelődés), vagy háromnál több (túlnépesedés) szomszédja van.
- Új sejt születik minden olyan cellában, melynek környezetében pontosan három sejt található.

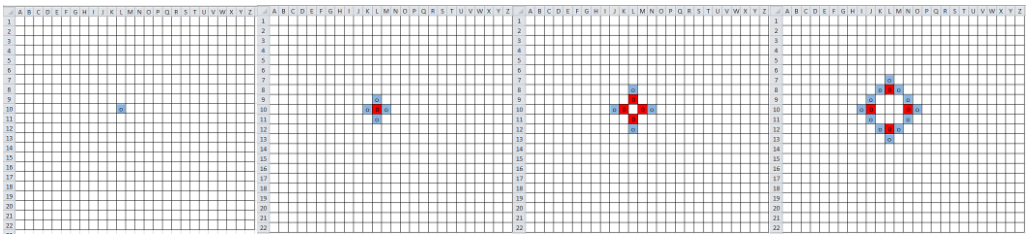
Fontos, hogy a változások csak a kör végén következnek be, tehát az „elhalálozók” nem akadályozzák a születést és a túlélést (legalábbis az adott körben), és a születések nem mentik meg az „elhalálozókat”. [6], [7]

4. ábra: A Conway-féle „életjáték” illusztrálása – egy táblázatkezelőben készített szimuláció; a sejttér kezdő és a 41. lépésbeli állapota.

4.2. Szimmetriajáték

Stanislaw Ulam szimmetriajátéka négyzetrács alakú táblán játszódik. Kezdetben a négyzetrács 1 vagy 2 pontján van kis o betű. Ezután minden következő lépésben:

- minden olyan üres helyen, amelynek „oldalszomszédságában” (a jobbra levő ábrán a satírozott mező „oldalszomszédai” látszanak) egyetlen kis o vagy nagy O betű található, megjelenik egy kis o betű;
- minden eddigi kis o betűből nagy O betű lesz;
- minden eddigi nagy O betű eltűnik a tábláról. [8]



5. ábra: Az Ulam-féle szimmetriajáték illusztrálása – egy táblázatkezelőben készített szimuláció; állapot a 0-3. időegységben.

Mindkét feladat megoldásának lényege, hogy a szimulált teret (azaz táblázatot) két példányban tároljuk, az egyik állapotból állítjuk elő minden lépésben a következő állapotot.

Ennek megvalósítására kiválóan alkalmas egy *táblázatkezelő*, amely mindennapos eszköze és hangsúlyos témája a mai magyarországi közoktatásnak. A fenti ábrák is egy táblázatkezelőben (Microsoft® Excel® 2010-ben) készültek. A szimuláció egyetlen nehézségét a körkörös hivatkozás okozza. Ennek kézenfekvő feloldása egy olyan makró (Visual Basic szubrutin) elkészítése, amely az „aktív” (azonos szemantikájú formulákkal feltöltött cellájú) oldal állapotának a „passzív” (az előző állapotot megőrző) lapra másolását végzi, pl. gomblenyomásra. Egy másik megoldás: a tisztán Visual Basic® (pontosabban Visual Basic for Application) nyelvű szubrutin megírása, amelynek elegendő egyetlen táblázatkezelőbeli lap is. A táblázatkezelő lényegében a szimulációs tér megjelenítését végzi, persze nyilvánvalóan a VBA program végrehajtásán túl. További „szépsége” e megközelítésnek, hogy a két szélsőséges (a lényegében táblázatkezelő függvényeken alapuló vs. a tisztán VBA-s) megoldás között számos köztes út létezik a függvényes vs. hagyományos programnyelvi betétek arányát tekintve.

5. Mozgás táblázatban

Ezt a fajta szimulációt legjobban jellemzi az alábbi feladat. Jól látszik, hogy a központi szerepet valamilyen táblázat játssza.

Egy játéktáblán a 0. időegységben L bábu van. Mindegyiket elindítjuk valamerre. Egy időegység alatt mindegyik a neki megfelelő távolságra mozdul el, a tábla széléről visszafordulnak. Lehetséges, hogy előbb-utóbb két bábu összeütközik: ugyanarra a helyre lépnének vagy átlépnének egymáson – egydimenziós térben feltétlenül, két dimenzióban nem biztos.

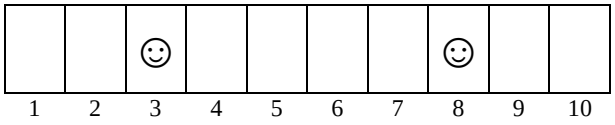
Feladat

Írj programot, amely megadja, hogy K időegységen belül mikor ütközik legelőször két bábu!

Példa

A tábla hossza: 10
A bábuk száma: 2
Az időtartam: 10
1. bábu helye: 3, iránya: B
2. bábu helye: 8, iránya: B

Ütközés időpont: 5



Ez olyan feladat, aminek a megoldása elvileg ki is számítható, de a bábuk számának növekedésével bonyolulttá válik. A bábuk egyenkénti mozgatása azonban 7-8. osztályos tanulók számára is megoldható.

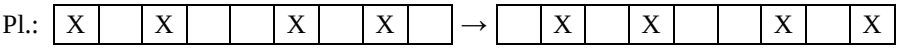
Hasonló feladattal találkozunk például az UVA online feladatbank 50. feladatában is. [9]

A Nemes Tihamér programozási verseny gyakori feladata a közlekedési szimuláció, mint a következő két példából látható.

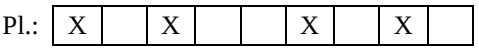
Feladat

Egy utat középen egy gyalogosátkelő két szakaszra oszt, a zebrához közlekedési lámpát helyeztek. Az útszakaszokat négyzetes cellákra osztjuk. N cella van a lámpa előtt, 1 cella a zebra, újabb N cella van a lámpa mögött. A mozgás szabályai:

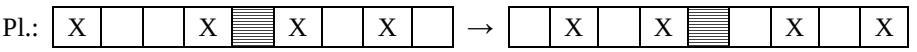
- egy autó egy időegység alatt egy cellával mozdulhat el;



- egy útszakaszon két autó között mindig kell lenni legalább 1 üres cellának (akkor is, ha sűrűbben érkeznének);



- a közlekedési lámpa minden P időtartam végén levő U időegységben piros, a többiben zöld; piros lámpaállásnál autó nem léphet a zebrára.

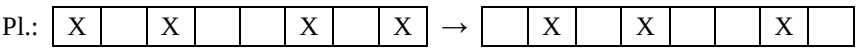


Készíts programot, amely megadja, hogy az egyes autók mikor jutnak ki az útszakasz végén! [10]

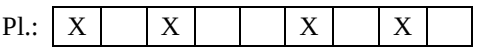
Feladat

Két út középen keresztezi egymást. Az egyiken csak balról jobbra, a másikon csak felülről lefelé haladhatnak autók. Az útszakaszokat négyzetes cellákra osztjuk, a kereszteződés előtt és mögött is N cella van. A mozgás szabályai:

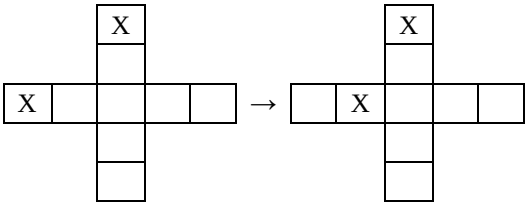
- egy autó egy időegység alatt egy cellával mozdulhat el;



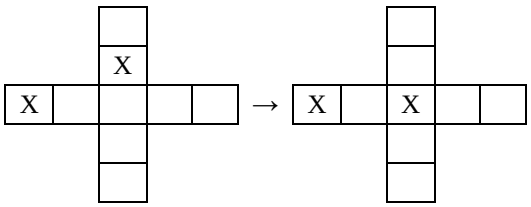
- két autó között mindig kell lenni legalább 1 üres cellának;



- a kereszteződésben jobbkéz-szabály van: ha a kereszteződés előtti cellákba egyszerre lépne 2 autó, akkor a balról jövő léphet, a felülről jövő nem;



- ha a felülről jövő a kereszteződésbe lép, a balról jövőnek tartania kell az 1 cella távolságot.



Készíts programot, amely megadja, hogy az egyes autók mikor jutnak ki az útszakasz végén! (Ha pl. csak balról jobbra jönnek autók, akkor a kilépési idejük pontosan 2*N+1-gyel több, mint a belépési idejük.) [11]

Ezekhez a feladathoz kétféleképpen is hozzáállhatunk. (Különbség közöttük csak az adatok ábrázolásában és a vizsgálandó feltételek számában van.)

Elképzelhető, hogy az adatokat helyenként ábrázoljuk, annyi vektorban, ahány útszakaszunk van. Minden útszakasz pontjait haladási iránnyal ellentétes sorrendben vizsgálva, az útszakaszon levő autók a peremfeltételeknek megfelelően elmozdíthatók vagy helyben hagyandók.

Elképzelhető, hogy az autók helyét vagy az adott útszakaszról várható kilépési idejét tároljuk. Ekkor annyi sort veszünk fel, ahány útszakasz van. Az autók a bemeneti adatok alapján valamely sorokba lépnek be, egyes sorokból kilépő autók más sorokba lépnek be, illetve eredménybe írható a kilépési idejük. A sorokban pedig minden autóhoz azt tároljuk, hogy mikor léphetne ki leghamarabb az adott sorból, kilépni pedig egy időegységben csak egy autót engedünk.

6. Párhuzamos megközelítés

Sok olyan szimulációs jellegű feladat is létezik, amelyben a végrehajtást tervezéskor akkor is érdemes *párhuzamos folyamat*ként elképzelni, ha erre a versenyen használt programozási nyelv nem ad lehetőséget

Feladat

Egy metróállomásra N időegységben érkeznek utasok, a K hosszú mozgólépcsőre legfeljebb ketten léphetnek egyszerre (azaz az érkezők közül ketten azonnal a mozgólépcső legfelső fokára kerülnek), a lépcsőn nincs mozgás – időegységenként mindenki egyet halad lefelé. A lépcső egy L utast befogadni képes váróterembe érkezik, az i-edik időegységben váróterembe lépőt ugyanabban az időegységben nem viheti el a metró. A metró M időegységenként jön és elviszi az összes várakozó utast. A beszállás 1 időegység alatt megtörténik. Kezdetben (a 0. időegységben) a lépcső és a váróterem is üres, az első metró az M. időegységben érkezik. Ha a váróterembe nem férnek be az utasok, akkor a metróállomást leállítják. Készíts programot, amely megadja, hogy az egyes metrószerelvények hány utast visznek el! A végrehajtás vagy N+K+M időegység után fejeződjön be, vagy akkor, amikor a váróterem megtelik! [11]

Itt a feladat megoldása pl. 4 párhuzamos folyamatra bontható:

- a metróállomás fenti váróterme – lefelé menni szándékozók kezelése;
- a lefelé menő lépcső – a lépcsőn lefelé haladók kezelése;
- a metróállomás lenti váróterme – metróra várakozók kezelése;
- a metró – felszállók kezelése

Itt például a lefelé menő lépcsőt kezelő eljárás a lépcső állapotát egy tömbben ábrázolja. Minden időegységben egy hellyel lépteti a tömbben levő utasokat, a legalsó lépcsőről lelépők számát átadja a lenti várótermet kezelő eljárásnak, a fenti várótermet kezelő eljárástól pedig átveszi a legfelső lépcsőre újonnan lépők számát. A fenti váróterembe érkeznek valahányan kívülről, s közülük egy időegységben maximum 2 utas átlép a mozgólépcsőre. Az alábbi két algoritmusban a ciklusfeltételt megfogalmazatlanul hagytuk, mert az most számunkra lényegtelen.

```
Eljárás Fenti váróterem:
  Db:=0
  Ciklus …
    Belépés(X); Db:=Db-X
    Ha Db>1 akkor  Küld(Lefelé lépcső,2); Db:=Db-2
          különben Küld(Lefelé lépcső,Db); Db:=0
  Ciklus vége
Eljárás vége.

Eljárás lefelé lépcső:
  Lépcső:=(0,...,0)
  Ciklus …
    Küld(Lenti váróterem,Lépcső(N))
    Ciklus i=N-től 2-ig -1-esével
      Lépcső(i):=Lépcső(i-1)
    Ciklus vége
    Vesz(Fenti váróterem,Lépcső(1))
  Ciklus vége
Eljárás vége.
```

Két különleges utasítást használtunk – `Küld(kinek,mit)`, `Vesz(Kitől,mit)` – amelyek megvalósítják a párhuzamos folyamatok közötti üzenetátadást (randevúval).

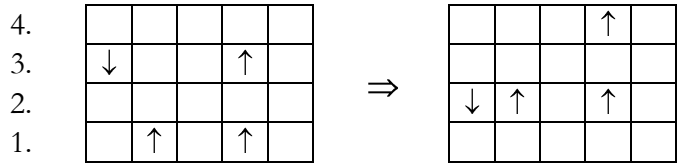
7. Párhuzamosság feloldása

A párhuzamosság a megvalósításban persze legtöbbször nem alkalmazható (a versenyzők tudása, programozási nyelvi ismeretei miatt), ezért egymásutánisággá kell alakítani.

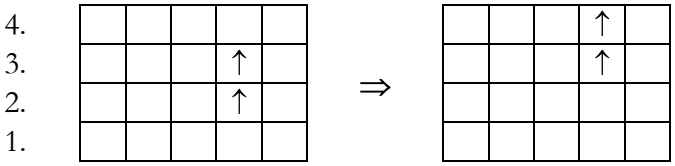
Feladat

Egy útszakaszra zebrát festettek az ábrának megfelelően. A zebrát N sorból és M oszlopból álló négyzetráccsal fedtük le. A négyzetrács egyes pontjaiban egyszerre egy ember tartózkodhat. Lentről és fentről is K időegységben érkeznek gyalogosok a szélső cellákba. Egyszerre 1 cella távolságra léphetnek, azaz legalább N+1 időegység kell az úttest túloldalára jutáshoz. A lépésekhez az alábbi szabályokat kell betartani, a leírás sorrendjében:

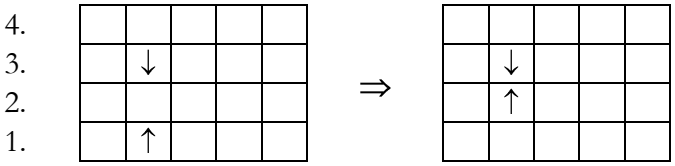
- ha egy gyalogos előtt legalább 2 szabad hely van, akkor előre léphet egyet (akkor is léphet, ha 1 szabad hely és egy vele egy irányban haladó van előtte);



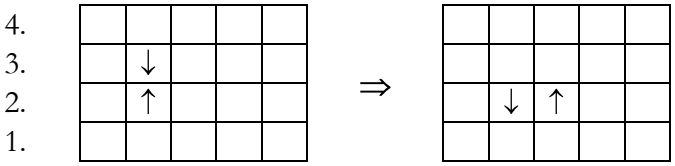
- ha valaki előtt vele egy irányba haladó van, és az ellép onnan, akkor a helyére szabad lépni;



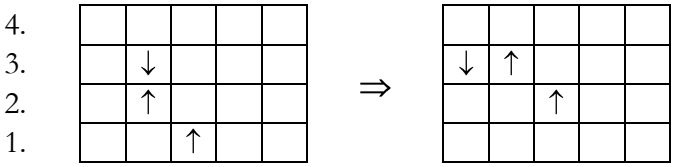
- ha alulról és felülről is ugyanarra a cellára lépne gyalogos, akkor az alulról jövőnek van elsőbbsége, ő léphet, míg a felülről jövő helyben marad;



- ha két gyalogos szemben áll egymással, akkor az alulról jövő jobbra kitérhet, azaz jobbra léphet egyet, ha arra a cellára abban az időegységben nem lépne más alulról vagy felülről; ekkor a felülről jövő egyet előre léphet;



- ha a két szemben álló közül az alulról jövő nem tud lépni, akkor a felülről jövő jobbra kitérhet (nála a jobbra kitérés persze az ábrán a baloldali szomszédra lépést jelenti), ha arra a cellára abban az időegységben nem lépne más; ekkor az alulról jövő egyet előre léphet;



- ha valaki előtt vele egy irányban haladó áll és nem lép el előle, akkor ő sem léphet.

Készíts programot, amely kiszámítja, hogy az N+1., N+2., …, N+2*K. időegységben hány gyalogos ér át az úttest másik oldalára! [11]

Itt a szabályok szigorúan leírják, hogy mit milyen sorrendben kell tenni. Egyértelmű belőlük, hogy a felfelé menőknek elsőbbségük van a lefelé haladókkal szemben. Emiatt a szimuláció két egymás utáni lépésre bontható: először a felfelé menőket, utána pedig a lefelé menőket vizsgáljuk. A kettő azonban nem teljesen független egymástól (utolsó előtti szabály), azaz a két lépés között információátadásra van szükség.

Az adott irányba haladók vizsgálatának sorrendjét pedig az határozza meg, hogy mindenki léphet az előző nyomába (második szabály), azaz az adott irányba haladókat érdemes az irányukkal ellentétes sorrendben vizsgálni.

8. Összefoglalás: a szimulációs feladatok érdekességéről

A téma lezárásaként pár gondolatot szentelünk a szimulációs feladatok érdekességének.

Oktatási érdekessége egy részről az **oktathatósága**. Mivel konkrét képhez, jól ismert jelenséghez kapcsolódik, így *könnyű a probléma megértése*, nem okoz gondot a tanulónak. Sőt *mozgósító erejű*, mivel egy jelenség „felszíne alatti", belső működését kell elképzelnünk, sőt akkurátusan megfogalmaznunk. Ez egy izgalmas, teremtő tevékenység.

Oktatandó, ugyanis a szimuláció-alapú oktatás *erősíti az absztrakciós képességet*, hiszen konkrét dolgokból elvont fogalmakat (osztályokat) kell megalkotni; konkrét dolgok jellemzőinek felismerésétől kell eljutni ezek – modellezés szempontjából fontos – állapotaihoz, a jellemzők változásától az állapotváltozás formalizálásáig. [12]

Egy feladat hátterének megvilágításában olvashatjuk a valladolidi egyetem online feladatbankjában az alábbit: „*Simulation is an important application area in computer science involving the development of computer models to provide insight into real-world events. There are many kinds of simulation including (and certainly not limited to) discrete event simulation and clock-driven simulation. Simulation often involves approximating observed behavior in order to develop a practical approach.*" [9]

A Bebras nemzetközi informatikai verseny leírásában is találkozhatunk a verseny kívánatos, jó feladattípusaival, köztük a szimulációval. Criteria for Good Bebras Tasks [13]:

„*Have easy understandable problem statements – A problem statement should be presented as easy as possible: easy understandable wording, easy understandable presentation of the problem (maybe use of pictures, examples, embedded in a proper story, use of a simulation or an interactive solving process), a problem statement should never be misleading.*

Should have interactive elements (simulations, solving activities, etc) – Multiple-choice is in many cases not adequate. Sometimes it is appropriate to input a number or a word or have a choice of a list of possibilities. Often the result can be produced by operating a simulation of a machine that should be operated properly."

Irodalom

1. Szlávi Péter, Zsakó László: *Szimulációs modellek táblázatkezelővel.* INF.O.'97 Informatika és számítástechnika tanárok konferenciája, Békéscsaba, 1997. november 20-22.
2. *Izsák Imre Gyula Komplex Természettudományi Verseny.* http://www.zmgzeg.sulinet.hu/izsak (utoljára megtekintve: 2014.10.31.)
3. Caitlin Buckhaults: *Increasing Computer Science Participation.* In the FIRST Robotics Competition with Robot Simulation. Proceeding ACM-SE 47 Proceedings of the 47th Annual Southeast Regional Conference. Article No. 19, ACM New York, NY, USA (2009)
4. *Logo Országos Számítástechnikai Tanulmányi Verseny 2015*. (2014) http://logo.inf.elte.hu (utoljára megtekintve: 2014.10.31.)
5. M. Eigen, R. Winkler: *A játék.* Gondolat (1981)
6. Charles Seife: *Impressions of Conway*. In The Sciences (1994) http://www.users.cloud9.net/~cgseife/conway.html (utoljára megtekintve: 2014.10.31.)
7. Martin Gardner: *The fantastic combinations of John Conway's new solitaire game "life"*. In Scientific American 223 (October 1970): 120-123. http://web.archive.org/web/20090603015231/http://ddi.cs.uni-potsdam.de/HyFISCH/Produzieren/lis_projekt/proj_gamelife/ConwayScientificAmerican.htm (utoljára megtekintve: 2014.10.31.)

8. Schrandt, R.G., Ulam, S.M.: *On Patterns of Growth of Figures in Two Dimensions*. In Notices of the American Mathematical Society 7 (1960)
9. *UVA problem set.* http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=50 (utoljára megtekintve: 2014.10.31.)
10. *Nemes Tihamér Országos Informatikai Tanulmányi Verseny.* http://nemes.inf.elte.hu (utoljára megtekintve: 2014.10.31.)
11. *Informatika Országos Középiskolai Tanulmányi Verseny.* http://nemes.inf.elte.hu (utoljára megtekintve: 2014.10.31.)
12. Szlávi Péter: *A programkészítés didaktikai kérdései.* Doktori disszertáció, 2005 http://www.inf.elte.hu/karunkrol/szolgaltatasok/konyvtar/Lists/Doktori%20disszertcik%20adatbzisa/Attachments/32/Szlavi_Peter_Ertekezes.pdf (utoljára megtekintve: 2014.10.31.)
13. Valentina Dagiene, Gerald Futschek: *Bebras International Contest on Informatics and Computer Literacy: Criteria for Good Tasks.* In R.T. Mittermeir and M.M. Sysło (Eds.): ISSEP 2008, LNCS 5090, pp. 19–30