make.tol de PDonnelly.CellsMatrix

PDonnelly.CellsMatrix es un programa que reproduce el autómata celular propuesto por Peter James Donnelly y que esta programado en Tol utilizando una estructura matricial, Matrix, de celulas de opinion. Es un programa desarrollado en un solo fichero Tol y que al emplear una version moderna de la funcion FormatMatrix() de 6 parametros, que en las versiones anteriores solo tiene 3, funciona a partir de la version Tol 2.0.1, sin embargo, su adaptacion a versiones anteriores no es compleja.

PDonnelly.CellsMatrix implementa una propuesta de automata celular de Peter James Donnelly del University College de Swansea, Gales, y de Dominic Welsh, de la Oxford University. Alexander Keewatin Dewdney describio en detalle este automata celular en su artículo titulado Five easy pieces for a do-loop and random number generator, Computer recreations, Scientific American, April: 20-30. En este artículo llama Votacion a este automata celular, pues segun A. K. Dewdney reproduce una votación política, si bien, lo que intenta reproducir este automata celular es el cambio de opinion por la influencia de los vecinos.

Cada celda de la matriz de este automata celular contiene un valor, por ejemplo, el 0 o el 1 elegido aleatoriamente, si bien PDonnelly.CellsMatrix puede funcionar hasta con 10 valores diferentes del 0 al 9 Cada valor 0 o 1 puede representar, por ejemplo, la opinión política de la persona residente en esa celda de la matriz u otra opinion sobre cualquier otro tema en particular. En cada paso del programa, se selecciona al azar a una de las personas y su opinion se somete a un posible cambio. Para ello se selecciona al azar uno de sus 8 vecinos, o menos si la celda esta situada en un borde o esquina de la matriz y su opinion se cambia por la de este vecino, fuera cual fuera su anterior opinión, pudiendo ser que no cambie si era la misma que la de su vecino. Este automata celular de, a partir de una matriz con valores aleatorios, tiende a crear zonas de opinion que crecen unas sobre otras y evolucionan por areas de influencia, llegado el caso, tras multiples ciclos, puede que toda la matriz puede terminar siendo de la misma opinion. Una de las posibles variantes de simulacion, que se incluye dentro de este programa PDonnelly.CellsMatrix, es la de la opinion contraria, esto es, que la persona en vez de adoptar la misma opinion que su vecino, adopte justo la contraria. Con esta estrategia, en vez de formarse grupos el proceso es justo el contrario, los grupos de opinion tienden a deshacerse segun avanza el proceso de ejecucion.

PDonnelly.CellsMatrix puede simular un comportamiento biológico como si fuera un ejemplo de la coexistencia y competencia de 2 clases de bacterias en un mismo medio, matriz, donde no falta su alimento, pero hay sobrepoblacion. En cada celda de la matriz hay una bacteria de una de las 2 especies y en cada paso del proceso se escoge al azar una de las bacterias para que muera y el espacio que deja libre sea utilizado por una de sus vecinas, elegida al azar, para reproducirse y que lo ocupe un ejemplar semejante a ella. A partir de la distribucion aleatoria inicial, con ambas clases de bacterias mezcladas, los ciclos de ejecucion, hacen que las bacterias de cada especie se organicen en grande grupos, que se mueven, se amplian o decrecen mientras luchas por sobrevivir. Tras un tiempo de ejecucion es posible que uno de los tipos de bacteria se convierta en dominante y el otro pueda llegar a extinguirse.

Árbol de ficheros

PDonnelly.CellsMatrix autómata celular de opiniones propuesto por Peter James Donnelly

- ← make.tol recrea con una matriz de celulas el automata celular de opiniones
- ← make.bat mandato de ejecucion del automata celular de Peter Donnelly
- simulator directorio del simulador del motor de reglas en Javascript
  - css directorio para css, Cascading Style Sheets, del simulador
    - ← simulator.css css para simular areas de aplicacion de las reglas
  - src directorio de codigo fuente Javascript del simulador de reglas
    - ← simulator.js simula el funcionamiento del automata celular
    - ← simulatorarray.js array con ejemplos de evolucion del automata celular
- → simulator.html simulador del autómata celular de opiniones de Peter J. Donnelly
- → pdonnelly_cellsmatrix.pdf funciones del autómata celular de opiniones de Peter J. Donnelly

Declaraciones

Constantes

- `Real PdcMin`
  Menor valor de la opinion, usualmente un 0.
- `Real PdcMax`
  Meyor valor de la opinion, usualmente un 1.

Funciones

- `Matrix PdcCreate(Real pdcHei, Real pdcWid, Real pdcMin, Real pdcMax)`
  Crea y retorna una matriz de poblacion con las dimensiones (pdcHei, pdcWid) y con opiniones en el rango (pdcMin, pdcMax), usualmente 0 y 1.
- `Real PdcTrace(Matrix inpMat, Real numSta, Text trcFil, Real trcCtr)`
  Realiza todas las trazas visuales de una evolucion de la matriz y escribe una traza Javascript en el fichero de traza que recibe como parametro.
- `Real PdcJavascript(Text dirTrc, Text dirSrc)`
  Reune todos los casos realizados y los prepara para que su resolucion pueda ser simulada por un simulador visual Javascript. Genera 2 ficheros en lenguaje Javascript: a) un fichero indice con un array en forma de tabña que contiene todos los casos resueltos, cada fila es (array de pasos, comentario), el nombre del array de pasos es el nombre del fichero sin extension y el comentario es igual cambiando el _ por blanco y b) un fichero mas grande, con tantos arrays como casos resueltos, cada uno de estos arrays contiene todos los pasos de resolucion del caso. El nombre de los arrays esta formado por el nombre de los ficheros. Para la generacion de estos 2 ficheros emplea 2 ficheros semilla que tienen la cabecera de comentarios de los programas Javascript. Recibe como parametros: a) el directorio de casos resueltos de entrada y b) el directorio de salida donde se escribira el codigo Javascript, asume que los ficheros semilla Javascript estan en este mismo directorio y los nombres de los ficheros para el array indice y el banco de datos son fijos. Retorna el numero de casos resueltos.

- `Set PdcNeighbors(Matrix inpMat, Real rowNum, Real colNum)`
  Retorna el conjunto del valor de las celdas vecinas de la que esta en la columna colNum y fila rowNum, que no siempre son 8 ya que las de las esquinas y las de los bordes tienen menos vecinos.
- `Real PdcRandNeighbor(Matrix inpMat, Real rowNum, Real colNum, Real opiCtr)`
  Retorna la opinion de un vecino elegido al azar, si opiCtr es falso entonces retorna justo la opinion contraria.
- `Set PdcRandCell(Matrix inpMat)`
  Retorna una celda elegida al azar de la matriz de entrada.
- `Matrix PdcExecute(Matrix inpMat, Real maxCic, Real opiCtr, Text trcFil)`
  Con la matriz de entrada impMat, ejecuta maxCic veces el automata celular, de forma normal si opiCtr o con la opinion contraria si !opiCtr y genera una traza de todos los paso en el fichero trcFil.

Pruebas

- `Text tstCmd`
  Control de los diferentes escenarios.
- `Real tstExe`
  Hace evolucionar al sistema de Peter Donnelly.
- `Real tstJsc`
  Reune los ficheros Javascript, lo prepara para la simulacion y crea la tabla array indice a los diferentes casos.

Constantes

Real PdcMin

```
////////////////////////////////////////////////////////////////////////////
Real PdcMin = 0;
////////////////////////////////////////////////////////////////////////////
PutDescription("Menor valor de la opinion, usualmente un 0.", PdcMin);
////////////////////////////////////////////////////////////////////////////
```

Real PdcMax

```
////////////////////////////////////////////////////////////////////////////
Real PdcMax = 9;
////////////////////////////////////////////////////////////////////////////
PutDescription("Meyor valor de la opinion, usualmente un 1.", PdcMax);
////////////////////////////////////////////////////////////////////////////
```

Matrix PdcCreate()

```
////////////////////////////////////////////////////////////////////////////
Matrix PdcCreate(Real pdcHei, // Numero de filas de la matriz de poblacion
                 Real pdcWid, // Numero de columnas de la matriz
                 Real pdcMin, // Valor minimo de la opinion, usualmente 0
                 Real pdcMax) // Valor maximo de la opinion, usualmente 1
////////////////////////////////////////////////////////////////////////////
```

```
{ Round(Rand(pdcHei, pdcWid, pdcMin, pdcMax)) };
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Crea y retorna una matriz de poblacion con las dimensiones (pdcHei, pdcWid)
y con opiniones en el rango (pdcMin, pdcMax), usualmente 0 y 1.",
PdcCreate);
////////////////////////////////////////////////////////////////////////////
```

Real PdcTrace()

```
////////////////////////////////////////////////////////////////////////////
Real PdcTrace(Matrix inpMat, // Matriz para trazar
              Real   numSta, // Numero de estado 0-inicial, 1, 2, ...
              Text   trcFil, // Ruta del fichero de traza
              Real   trcCtr) // 0 abre fichero, 1 ciclo, 2 cierra fichero
////////////////////////////////////////////////////////////////////////////
{
  // Visualizacion simple por pantalla
  Text linSep = Repeat("_", Columns(inpMat))+"\n"+
                "State = "+FormatReal(numSta, "%.0lf")+"\n";
  Text boxMat = FormatMatrix(inpMat, "", "\n", "", "", "%.0lf"); // Rectangulo
  Text WriteLn(linSep+boxMat);

  // Registro en el fichero de traza

  Text srcIni = If(trcCtr == 0, // Apertura
  {
    Text WriteLn("Cases : "+trcFil);
    Text srcSep = "\n"+Repeat("/", 78)+"\n\n"; // Separador de arrays
    Text srcVar = GetFilePrefix(trcFil);       // Nombre fichero -> variable
    Text srcOpn = srcSep +
                  "var " + srcVar + " = new Array(\n"; // Array Javascript
    Text WriteFile(trcFil, srcOpn);
     ""       // No hay inicio en la apertura Javascript
  }, ",\n"); // En ciclo javascript cierra la linea anterior del array

  If(trcCtr <= 1, // Para 0 y 1 hay un estado que escribir
  {
    Text linMat = FormatMatrix(inpMat, "", "~",  "", "", "%.0lf"); // 1 linea
    Text srcBdy = "  " + Char(34) + linMat + Char(34);

    Text AppendFile(trcFil, srcIni + srcBdy); // Javascript

    TRUE // Seguimos en ciclo
  },
  {
    // 2 o mayor es el fin, no hay estado que escribir, solo se cierra
    Text AppendFile(trcFil, "\n);\n");              // Fin del array
    FALSE // Fin del ciclo
  })
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Realiza todas las trazas visuales de una evolucion de la matriz y escribe
una traza Javascript en el fichero de traza que recibe como parametro.",
PdcTrace);
////////////////////////////////////////////////////////////////////////////
```

Real PdcJavascript()

```
////////////////////////////////////////////////////////////////////////////
Real PdcJavascript(Text dirTrc, // Directorio de casos resueltos
                   Text dirSrc) // Directorio de Javascript
////////////////////////////////////////////////////////////////////////////
{
  Set  filSet = GetDir(dirTrc)[1]; // Todos los ficheros a primer nivel
  Set  filRea = EvalSet(filSet, Text(Text filNam) // Leer todos los fichero
                        { ReadFile(dirTrc+"/"+filNam)+"\n" });
  Text filCod = SetSum(filRea); // Codigo junto de todos los ficheros
```

```
  // Contruir el array de casos, indice
  Set  arrSet = For(1, Card(filSet), Text(Real filPos) // Para los casos
  {
    Text arrNam = GetFilePrefix(filSet[filPos]);
    Text arrLbl = Replace(arrNam, "_", " ");
    Text arrNew = "new Array(" + arrNam + ", '" + arrLbl + "')";
    Text arrEnd = If(Card(filSet)==filPos, "", ",\n"); // El ultimo diferente
    "  "+arrNew+arrEnd
  });
  Text arrTxt = SetSum(arrSet); // Une todos los textos
  Text arrSed = dirSrc+"/simulatorarray.sed"; // Fichero semilla
  Text arrPth = Replace(arrSed,".sed",".js"); // Fichero javascript
  Text WriteFile(arrPth, Replace(ReadFile(arrSed), "_ARR_", arrTxt));

  // Construir el banco de datos de pasos, steps, de los casos
  Text sdbSed = dirSrc+"/simulatordb.sed";
  Text sdbPth = Replace(sdbSed,".sed",".js");
  Text WriteFile(sdbPth, Replace(ReadFile(sdbSed), "_SDB_", filCod));

  Card(filSet) // Numero de casos
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Reune todos los casos realizados y los prepara para que su resolucion pueda
ser simulada por un simulador visual Javascript.
Genera 2 ficheros en lenguaje Javascript:
a) un fichero indice con un array en forma de tabña que contiene todos los
   casos resueltos, cada fila es (array de pasos, comentario),
   el nombre del array de pasos es el nombre del fichero sin extension y
   el comentario es igual cambiando el _ por blanco y
b) un fichero mas grande, con tantos arrays como casos resueltos,
   cada uno de estos arrays contiene todos los pasos de resolucion del caso.
El nombre de los arrays esta formado por el nombre de los ficheros.
Para la generacion de estos 2 ficheros emplea 2 ficheros semilla que tienen
la cabecera de comentarios de los programas Javascript.
Recibe como parametros:
a) el directorio de casos resueltos de entrada y
b) el directorio de salida donde se escribira el codigo Javascript,
   asume que los ficheros semilla Javascript estan en este mismo directorio y
   los nombres de los ficheros para el array indice y
   el banco de datos son fijos.
Retorna el numero de casos resueltos.",
PdcJavascript);
////////////////////////////////////////////////////////////////////////////
```

Set PdcNeighbors()

```
////////////////////////////////////////////////////////////////////////////
Set PdcNeighbors(Matrix inpMat, // Mariz de entrada
                 Real   rowNum, // Fila, Y
                 Real   colNum) // Columna, X
////////////////////////////////////////////////////////////////////////////
{
  Real matHei = Rows   (inpMat); // Numero de filas de la matriz de entrada
  Real matWid = Columns(inpMat); // Numero de columnas de la matriz de entrada

  Real rowMin = If(rowNum<=1,           1, rowNum-1); // Primera fila
  Real rowMax = If(rowNum>=matHei, matHei, rowNum+1); // Ultima fila

  Real colMin = If(colNum<=1,           1, colNum-1); // Primera columna
  Real colMax = If(colNum>=matWid, matWid, colNum+1); // Ultima columna

  Set tabNei = For(rowMin, rowMax, Set(Real rowCnt) // Para las filas
  {
    For(colMin, colMax, Real(Real colCnt) // Para las columas
    {
      If(And(rowCnt==rowNum, colCnt==colNum),
         PdcMin-1,                         // La propia persona no cuenta
         MatDat(inpMat, rowCnt, colCnt)) // Valor celda vecina
```

```
    })
  });
  Set setNei = BinGroup("<<", tabNei);      // Pasa de tabla a conjunto lineal
  Select(setNei,                            // Elimina el -1 de la propia celda
         Real(Real neiVal){ neiVal >= PdcMin }) // Selecciona los validos
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el conjunto del valor de las celdas vecinas de la que esta en la
columna colNum y fila rowNum, que no siempre son 8 ya que las de las esquinas
y las de los bordes tienen menos vecinos.",
PdcNeighbors);
////////////////////////////////////////////////////////////////////////////
```

Real PdcRandNeighbor()

```
////////////////////////////////////////////////////////////////////////////
Real PdcRandNeighbor(Matrix inpMat, // Matriz de entrada
                     Real   rowNum, // Fila, Y
                     Real   colNum, // Columna, X
                     Real   opiCtr) // La opinion y si false, la contraria
////////////////////////////////////////////////////////////////////////////
{
  Set  opiSet = PdcNeighbors(inpMat, rowNum, colNum); // Conjunto de opiniones
  Real opiCrd = Card(opiSet); // Numero de opiniones
  If(!opiCrd , -1, // Error, conjunto vacio, no hay opiniones
  {
    Real opiVal = opiSet[Min(opiCrd,
                             Max(1, Round(Rand(0, opiCrd) + 0.5)))]; // 1 al azar
    If(opiCtr, opiVal, !opiVal) // Si opiCtr la opinion, sino la contraria
  })
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna la opinion de un vecino elegido al azar, si opiCtr es falso entonces
retorna justo la opinion contraria.",
PdcRandNeighbor);
////////////////////////////////////////////////////////////////////////////
```

Set PdcRandCell()

```
////////////////////////////////////////////////////////////////////////////
Set PdcRandCell(Matrix inpMat) // Para retornar una celda al azar
////////////////////////////////////////////////////////////////////////////
{
  Real matHei = Rows   (inpMat); // Numero de filas de la matriz de entrada
  Real matWid = Columns(inpMat); // Numero de columnas de la matriz de entrada

  Real rowRnd = Min(matHei, Max(1, Round(Rand(0, matHei) + 0.5)));
  Real colRnd = Min(matWid, Max(1, Round(Rand(0, matWid) + 0.5)));

  SetOfReal(rowRnd, colRnd) // Posicion (x,y) o (fila, columna)
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna una celda elegida al azar de la matriz de entrada.",
PdcRandCell);
////////////////////////////////////////////////////////////////////////////
```

Matrix PdcExecute()

```
////////////////////////////////////////////////////////////////////////////
Matrix PdcExecute(Matrix inpMat, // Matriz de entrada
                  Real   maxCic, // Numero de ciclos
                  Real   opiCtr, // Control de opinion, si false, la contraria
                  Text   trcFil) // Ruta del fichero de traza
```

```
////////////////////////////////////////////////////////////////////////////
{
  Real PdcTrace(inpMat, 0, trcFil, 0); // Estado inicial 0 e inicio de traza

  Set  exeCic = For(1, maxCic, Real(Real numCic)
  {
    Set  celSet = PdcRandCell(inpMat); // Una celda (row, col) al azar
    Real numRow = celSet[1]; // La fila
    Real numCol = celSet[2]; // La columna

    Real newVal = PdcRandNeighbor(inpMat, numRow, numCol, opiCtr);
    Real oldVal = PutMatDat(inpMat, numRow, numCol, newVal);

    PdcTrace(inpMat, numCic, trcFil,  // Visualiza el nuevo estado
             If(numCic<maxCic, 1, 2)) // Intermedio o final
  });

  inpMat
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Con la matriz de entrada impMat, ejecuta maxCic veces el automata celular,
de forma normal si opiCtr o con la opinion contraria si !opiCtr y
genera una traza de todos los paso en el fichero trcFil.",
PdcExecute);
////////////////////////////////////////////////////////////////////////////
```

Text tstCmd

```
////////////////////////////////////////////////////////////////////////////
Text tstCmd = "tin_09"; // Caso de prueba
////////////////////////////////////////////////////////////////////////////
PutDescription("Control de los diferentes escenarios.", tstCmd);
////////////////////////////////////////////////////////////////////////////
```

Real tstExe

```
////////////////////////////////////////////////////////////////////////////
Real tstExe = Case(
  tstCmd == "tin_09",
  {
    // Pequeña prueba para ver como se transmite la opinion
    Matrix iniMat = PdcCreate(4, 6, 0, 9);
    Matrix PdcExecute(iniMat, 401, TRUE,
                      "trace/reducido_con_muchas_opiniones.js");
    TRUE
  },
  tstCmd == "tin_01",
  {
    // Pequeña prueba para ver como predomina una opinion
    Matrix iniMat = PdcCreate(4, 6, 0, 1);
    Matrix PdcExecute(iniMat, 201, TRUE,
                      "trace/reducido_con_dos_opiniones.js");
    TRUE
  },
  tstCmd == "big_01",
  {
    // Grande con 2 opiniones
    Matrix iniMat = PdcCreate(17, 34, 0, 1);
    Matrix PdcExecute(iniMat, 1001, TRUE,
                      "trace/grande_con_dos_opiniones.js");
    TRUE
  },
  tstCmd == "big_03",
  {
    // Grande con 4 opiniones
    Matrix iniMat = PdcCreate(17, 34, 0, 3);
    Matrix PdcExecute(iniMat, 1001, TRUE,
                      "trace/grande_con_cuatro_opiniones.js");
```

```
    TRUE
  },
  tstCmd == "opo_01",
  {
    // Grande con 2 opiniones
    Matrix iniMat = PdcCreate(17, 34, 0, 1);
    Matrix PdcExecute(iniMat, 1001, FALSE, // Opinion contraria
                      "trace/grande_con_la_opinion_contraria.js");
    TRUE
  },
  TRUE,             FALSE); // No hace nada
////////////////////////////////////////////////////////////////////////
PutDescription("Hace evolucionar al sistema de Peter Donnelly.", tstExe);
////////////////////////////////////////////////////////////////////////
```

Real tstJsc

```
////////////////////////////////////////////////////////////////////////
Real tstJsc = PdcJavascript("trace", "simulator/src"); // Dirs traza y .js
////////////////////////////////////////////////////////////////////////
PutDescription(
"Reune los ficheros Javascript, lo prepara para la simulacion y crea la tabla
array indice a los diferentes casos.",
tstJsc);
////////////////////////////////////////////////////////////////////////
```