make.tol de WordSearch.LetterSoup

WordSearch.LetterSoup es un programa que busca las palabras de un conjunto dentro de una sopa de letras que se especifica como un rectangulo de caracteres. La busqueda la realiza en todas las direcciones horizontal, vertical y en las 2 diagonales y en todos los sentidos posibles, de izquierda a derecha, de derecha a izquierda, de arriba hacia abajo y de abajo hacia arriba, en total son 8 las posibles formas en las que puede aparecer una palabra. Es un programa desarrollado en un solo fichero Tol y que funciona en todas las versiones del lenguaje de programacion Tol en las que se ha probado.

En WordSearch.LetterSoup, la sopa de letras se representa mediante un texto rectangular, de cualquier tamaño y proporcion, con cada fila en una linea y con las lineas separadas por su salto de linea. Dentro de una misma linea todos los caracteres van consecutivos. A la hora de realizar la busqueda este texto se convierte en una tabla, conjunto de conjuntos, esto es, un conjunto de lineas donde cada linea es un conjunto de caracteres. De la conversion de texto a tabla se encargan las funciones: a) SplitText(), que retorna el conjunto de todas las letras de una palabra o de un texto cualquier. b) CreateTable(), que con el soporte de SplitText(), crea una tabla de caracteres como conjunto de conjunto.

Las palabras que WordSearch.LetterSoup tiene que buscar las recibe como un conjunto de textos y el programa asume que existen 3 posibilidades, funcionando para las 3: a) que la palabra no aparezca en la sopa de letras, b) que aparezca 1 vez en cualquier direccion (vertical, horizontal y diagonales) y en cualquier sentido (hacia la derecha, hacia la izquierda, hacia arriba o hacia abajo y c) que la palabra aparezca varias veces en diferentes o iguales sentidos y direcciones. Los casos de prueba que acompañan a este programa exploran todas estas posibilidades y en uno de ellos hay que buscar la palabra palabra muchas veces encontrandose en todas las variantes de direccion y sentido.

WordSearch.LetterSoup para optimizar su funcionamiento crea, a partir de la tabla que forma la sopa de letras, un pequeño indice, donde para cada letra que aparece en la sopa de letra proporciona informacion de las coordenasas de fila y columna donde esta esa letra. Ello permite que dada una palabra no haya que procesar toda la tabla, sino solo aquellas coordenadas donde aparece su letra inicial. En cada coordenada donde aparece la inicial de la palabra la busqueda se realiza en estrella, en las 8 forma que generan la combinacion de las horizontales, verticales y las 2 diagonales en sus 2 sentidos, esto es, (1+1+2)*2=8 formas: a) CreateIndex(), se encarga de la construccion de este indice que es un conjunto de tablas donde cada tabla se asocia a una letra y en cada fila se informa de la letra, la fila y la columna, para ello se emplea la funcion Tol Classify(). b) FindLetter(), es la funcion que realiza la busqueda de las coordenadas de una letra en el indice, para ello se apoya en la funcion Tol llamada Select(). c) Las funciones WordMatch() y FindWord() se encargan de realizar la busqueda de la palabra conociendo las coordenadas de su letra inicial.

Las direcciones y sentidos de busqueda se codifican, en WordSearch.LetterSoup, mediante 2 numeros de desplazamiento que pueden tomar los valores -1, 0 y 1, de esta forma: a) la escritura normal seria (0, 1), sin cambiar de fila (0) avanzar a la derecha (1) de letra a letra, b) escribir hacia la izquierda como (0, -1), sin cambiar de fila (0) retroceder hacia la izquierda (-1) de letra en letra, c) escribir de arriba hacia abajo como (1, 0), sin cambiar de columna (0), bajar de fila en fila (1). d) en diagonal de arriba hacia abajo y de izquierda a derecha como (1,1), bajando de fila en fila (1), a la derecha de letra en letra (1), e) la escritura en diagonal de abajo hacia arriba y de derecha a izquierda se codifica como (-1,-1), f) etc.

Árbol de ficheros

WordSearch.LetterSoup busca palabras, por direcciones y sentidos, en una sopa de letras

- ← make.tol busca palabras en sopas de letras en 4 direcciones x 2 sentidos
- ← make.bat mandato de ejecucion del buscador de palabras en sopas de letras
- → startlog.txt log de solucion con pruebas y sopas de letras ya resueltas
- → wordsearch_lettersoup.pdf documento de funciones de busqueda en sopas de letras

Declaraciones

Funciones

- `Set SplitText(Text linTxt)`
  Retorna el conjunto formado por todas las letras de una palabra.
- `Set CreateTable(Text tabTxt)`
  Retorna una tabla (como conjunto de conjuntos) donde cada elemento son las letras de un texto. Cada fila corresponde a una linea del texto, por lo que rompe por saltos de linea y luego por letras. Selecciona solo las filas que tienen el mismo largo que la primera, para garantizar que la tabla queda perfectamente rectangular.
- `Set CreateIndex(Set tabSet)`
  Retorna una clasificacion, que es un conjunto de tablas, donde cada tabla comienza por la misma letra, a partir de otra donde para cada celda se especifica su contenido, su fila y su columna, por ejemplo: [[[a, 1, 1]] [a, 1, 1] ab -> [a, 1, 1]] cd [[b, 1, 2]] aa [[c, 2, 1]] [[d, 2, 2]]] A esta tabla se la puede denominar tabla indice.
- `Set FindLetter(Set tabInx, Text oneLet)`
  Retorna las posiciones en las que se encuentra una letra, oneLet, dentro de una tabla indice. Retorna el conjunto formado por las posiciones, pares de fila y columna.
- `Set WordMatch(Set tabSet, Text word, Real iniRow, Real iniCol, Real incRow, Real incCol)`
  Busca una palabra en una tabla. La busqueda se realiza a partir de una posicion (iniRow,iniCol) siguiendo un cierto tipo de movimiento lineal que se especifica con los incrementos, positivos, nulo o negativos especificados con (incRow,incCol). Retorna la palabra, la posicion inicial, los incrementos y si la ha encontrado o no.
- `Set FindWord(Set tabSet, Text word, Set tabInx)`

Retorna todas la posibles ocurrencias de una palabra word en una tabla tabSet. Para ello solo busca alrededor de las posiciones donde se encuentra su letra inicial. Para saber donde esta su letra inicial utiliza la tabla indice tabInx.

- `Set SolveSet(Set tabSet, Set wordSet)`
  Retorna la posiciones en las que ha encontrado todas las palabras del conjunto wordSet dentro de la tabla tabSet.
- `Set SolvePrint(Text tabTxt, Set wordSet)`
  Retorna la posiciones en las que ha encontrado todas las palabras del conjunto wordSet dentro de la tabla tabSet. Como efecto lateral visualiza la tabla original y la solucion.

Pruebas

- `Text tstCmd`
  Control de las diferentes pruebas.
- `Real num_17`
  Pequeña prueba solo con numeros.
- `Real palDir`
  La palabra palabra en multiples direcciones.
- `Real aniDom`
  Pequeña prueba con nombres de animales domesticos.
- `Real aniSal`
  Pequeña prueba con nombres de animales salvajes.
- `Real mueHog`
  Pequeña prueba con nombres de muebles del hogar.
- `Real medTra`
  Pequeña prueba con nombres de medios de transporte.
- `Real paiEu7`
  Pequeña prueba con nombres de 7 paises de Europa.
- `Real paiE10`
  Pequeña prueba con nombres de 10 paises de Europa.
- `Real pla__9`
  Prueba 9 planetas, de la lista algunos no son.

Set SplitText()

```
////////////////////////////////////////////////////////////////////////
Set SplitText(Text linTxt)
////////////////////////////////////////////////////////////////////////
{
  Real lenTxt = TextLength(linTxt);
  For(1, lenTxt, Text(Real pos) { Sub(linTxt, pos, pos) })
};
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el conjunto formado por todas las letras de una palabra.",
SplitText);
////////////////////////////////////////////////////////////////////////
```

Set CreateTable()

```
////////////////////////////////////////////////////////////////////////////
Set CreateTable(Text tabTxt)
////////////////////////////////////////////////////////////////////////////
{
  Set linSet = Tokenizer(tabTxt,"\n");

  Set tabSet = EvalSet(linSet, Set(Text linTxt) { SplitText(linTxt) });

  Select(tabSet, Real(Set rowSet) { Card(rowSet)==Card(tabSet[1]) })
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna una tabla (como conjunto de conjuntos) donde cada elemento son las
letras de un texto.
Cada fila corresponde a una linea del texto, por lo que rompe por saltos de
linea y luego por letras.
Selecciona solo las filas que tienen el mismo largo que la primera,
para garantizar que la tabla queda perfectamente rectangular.",
CreateTable);
////////////////////////////////////////////////////////////////////////////
```

Set CreateIndex()

```
////////////////////////////////////////////////////////////////////////////
Set CreateIndex(Set tabSet)
////////////////////////////////////////////////////////////////////////////
{
  Real maxRow = Card(tabSet);
  Real maxCol = Card(tabSet[1]);
  Set  posSet = For(1, maxRow, Set(Real row)
  {
    For(1, maxCol, Set(Real col)
    {
      [[ tabSet[row][col], row, col ]]
    })
  });

  Set  binGrp = BinGroup("<<",posSet); // Convierte de tabla a conjunto

  Classify(binGrp, Real(Set a, Set b) { Compare(a[1],b[1]) }) // Clasifica
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna una clasificacion, que es un conjunto de tablas, donde cada tabla
comienza por la misma letra, a partir de otra donde para cada celda se
especifica su contenido, su fila y su columna, por ejemplo:
     [ [ [a, 1, 1] ]
         [a, 1, 1]
 ab ->   [a, 1, 1] ]
 cd    [ [b, 1, 2] ]
 aa    [ [c, 2, 1] ]
       [ [d, 2, 2] ] ]
A esta tabla se la puede denominar tabla indice.",
CreateIndex);
////////////////////////////////////////////////////////////////////////////
```

Set FindLetter()

```
////////////////////////////////////////////////////////////////////////////
Set FindLetter(Set  tabInx,
               Text oneLet)
////////////////////////////////////////////////////////////////////////////
```

```
{
  Set iniSet = Select(tabInx, Real(Set cla) { cla[1][1] == oneLet });

  If(NE(Card(iniSet),1), Empty, // Si existe solo tendria que haber uno
     EvalSet(iniSet[1], Set(Set iniPos)
             { SetOfReal(iniPos[2], iniPos[3]) })) // Fila y columna
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna las posiciones en las que se encuentra una letra, oneLet, dentro de
una tabla indice.
Retorna el conjunto formado por las posiciones, pares de fila y columna.",
FindLetter);
////////////////////////////////////////////////////////////////////////////
```

Set WordMatch()

```
////////////////////////////////////////////////////////////////////////////
Set WordMatch(Set  tabSet,
              Text word,
              Real iniRow,
              Real iniCol,
              Real incRow,
              Real incCol)
////////////////////////////////////////////////////////////////////////////
{
  Set  wrdSet = SplitText(word);
  Real wrdLen = Card(wrdSet);
  Real maxRow = Card(tabSet);
  Real maxCol = Card(tabSet[1]);
  Real match  = Copy(TRUE);
  Real end    = Copy(FALSE);
  Real row    = Copy(iniRow);
  Real col    = Copy(iniCol);
  Real wrdPos = 1;
  Real While(Not(end),
  {
    If(wrdSet[wrdPos] != tabSet[row][col],
    {
      Real(end   := Copy(TRUE));
      Real(match := Copy(FALSE));
      FALSE
    },
    {
      Real(row    := row+incRow);
      Real(col    := col+incCol);
      Real(wrdPos := wrdPos + 1);
      If(GT(wrdPos,wrdLen),  { Real(end:=Copy(TRUE)), TRUE },
      If(Or(LT(row,     1),
            LT(col,     1),
            GT(row,maxRow),
            GT(col,maxCol)), { Real(end:=Copy(TRUE));
                               Real(match:=Copy(FALSE));
                               FALSE },
                             { TRUE }))
    })
  });
  [[word, iniRow, iniCol, incRow, incCol, match]]
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Busca una palabra en una tabla. La busqueda se realiza a partir de una
posicion (iniRow,iniCol) siguiendo un cierto tipo de movimiento lineal que
se especifica con los incrementos, positivos, nulo o negativos especificados
con (incRow,incCol).
Retorna la palabra, la posicion inicial, los incrementos y si la ha encontrado
o no.",
WordMatch);
////////////////////////////////////////////////////////////////////////////
```

Set FindWord()

```
////////////////////////////////////////////////////////////////////////
Set FindWord(Set  tabSet,
             Text word,
             Set  tabInx)
////////////////////////////////////////////////////////////////////////
{
  Set findIni = FindLetter(tabInx, Sub(word,1,1));
  Set match   = EvalSet(findIni, Set(Set posSet)
  {
    Real row = posSet[1];
    Real col = posSet[2];
//  Text WriteLn("Searching <"+word+"> en: ("+FormatReal(row,"%.0lf")+", "+
//                                           FormatReal(col,"%.0lf")+")");
    [[ WordMatch(tabSet, word, row, col,-1,-1),
       WordMatch(tabSet, word, row, col,-1, 0),
       WordMatch(tabSet, word, row, col, 0,-1),
       WordMatch(tabSet, word, row, col, 0, 1),
       WordMatch(tabSet, word, row, col, 1, 0),
       WordMatch(tabSet, word, row, col, 1, 1),
       WordMatch(tabSet, word, row, col, 1,-1),
       WordMatch(tabSet, word, row, col,-1, 1) ]]
  });
  // Retornar solo las encontradas res[6]==TRUE
  Select(BinGroup("<<",match), Real(Set res) { res[6] })
};
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna todas la posibles ocurrencias de una palabra word en una tabla
tabSet. Para ello solo busca alrededor de las posiciones donde se encuentra
su letra inicial. Para saber donde esta su letra inicial utiliza la tabla
indice tabInx.",
FindWord);
////////////////////////////////////////////////////////////////////////
```

Set SolveSet()

```
////////////////////////////////////////////////////////////////////////
Set SolveSet(Set tabSet,
             Set wordSet)
////////////////////////////////////////////////////////////////////////
{
  Set tabInx = CreateIndex(tabSet);
  Set resSet  = EvalSet(wordSet, Set(Text word)
  { FindWord(tabSet, word, tabInx) });
  BinGroup("<<", resSet)
};
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna la posiciones en las que ha encontrado todas las palabras del
conjunto wordSet dentro de la tabla tabSet.",
SolveSet);
////////////////////////////////////////////////////////////////////////
```

Set SolvePrint()

```
////////////////////////////////////////////////////////////////////////
Set SolvePrint(Text tabTxt, Set wordSet)
////////////////////////////////////////////////////////////////////////
{
  Text WriteLn(Repeat("_",20)+"\nProblema:");
  Text WriteLn(tabTxt);
  Set tabSet = CreateTable(tabTxt);
  Set solSet = SolveSet(tabSet, wordSet);
```

```
  Set inxTre = EvalSet(solSet, Set(Set sol)
  {
    Text word   = sol[1];
    Real wrdLen = TextLength(word);
    Real iniRow = sol[2];
    Real iniCol = sol[3];
    Real incRow = sol[4];
    Real incCol = sol[5];
    For(1, wrdLen, Set(Real pos)
    {
      Text let = Sub(word,pos,pos);
      Real row = iniRow+(incRow*(pos-1));
      Real col = iniCol+(incCol*(pos-1));
      [[ let, row, col ]]
    })
  });

  Set inxCla = Classify(BinGroup("<<",inxTre), Real(Set a, Set b)
  {
    Real comRow = Compare(a[2],b[2]);
    If(NE(comRow,0), comRow, Compare(a[3],b[3]))
  });

  // Eliminar las letras que son comunes a palabras
  Set inxSor = EvalSet(inxCla, Set(Set cla) { cla[1] });

  Real inxCnt = 1;
  Text WriteLn("Solucion:");
  Real maxRow = Card(tabSet);
  Real maxCol = Card(tabSet[1]);
  Set For(1, maxRow, Real(Real row)
  {
    Set For(1, maxCol, Real(Real col)
    {
      If(GT(inxCnt,Card(inxSor)), { Text Write("·"); FALSE },
      If(And(EQ(row,inxSor[inxCnt][2]),EQ(col,inxSor[inxCnt][3])),
      {
        Text Write(inxSor[inxCnt][1]);
        Real(inxCnt:=inxCnt+1);
        TRUE
      },
      { Text Write("."); FALSE }))
    });
    Text WriteLn("");
    TRUE
  });

  Text WriteLn("");
  solSet
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna la posiciones en las que ha encontrado todas las palabras del
conjunto wordSet dentro de la tabla tabSet. Como efecto lateral visualiza
la tabla original y la solucion.",
SolvePrint);
////////////////////////////////////////////////////////////////////////////
```

Text tstCmd

```
////////////////////////////////////////////////////////////////////////////
Text tstCmd = "all"; // Caso de prueba
////////////////////////////////////////////////////////////////////////////
PutDescription("Control de las diferentes pruebas.", tstCmd);
////////////////////////////////////////////////////////////////////////////
```

Real num_17

```
////////////////////////////////////////////////////////////////////////////
Real num_17 = If(!(tstCmd <: [["num_17","all"]]), FALSE,
{
  Set resSet = SolvePrint(
    "unocsaszx\n"+
    "dwseueidc\n"+
    "ocrsraeti\n"+
    "stigbyten\n"+
    "henujmerc\n"+
    "sonulpano\n",
    [["uno","dos","tres","cuatro","cinco","seis","siete"]]);
  Card(resSet)
});
////////////////////////////////////////////////////////////////////////////
PutDescription("Pequeña prueba solo con numeros.",num_17);
////////////////////////////////////////////////////////////////////////////
```

Real palDir

```
////////////////////////////////////////////////////////////////////////////
Real palDir = If(!(tstCmd <: [["palDir","all"]]), FALSE,
{
  Set resSet = SolvePrint(
    "pqpqwawffeoacapaa\n"+
    "azwaurqueiroztarh\n"+
    "lxpwlbooebzpablbg\n"+
    "awpalabraqazqaaaa\n"+
    "beafglblracdwlblf\n"+
    "rdldeaarbdedearad\n"+
    "pcakopuratfarpapa\n"+
    "avbrloaxlolrttras\n"+
    "lfrobkuiajybybblz\n"+
    "ararbalapdfauaaaq\n"+
    "btuuhgloozslillbw\n"+
    "rygeerfavbnaoaarq\n"+
    "eheemnbvpcxppppap\n",
    [["palabra"]]);
  Card(resSet)
});
////////////////////////////////////////////////////////////////////////////
PutDescription("La palabra palabra en multiples direcciones.",palDir);
////////////////////////////////////////////////////////////////////////////
```

Real aniDom

```
////////////////////////////////////////////////////////////////////////////
Real aniDom = If(!(tstCmd <: [["aniDom","all"]]), FALSE,
{
  Set resSet = SolvePrint(
    "gallina\n"+
    "abvacac\n"+
    "saogato\n"+
    "orrubit\n"+
    "sorirgo\n"+
    "eoelarñ\n"+
    "lipateu\n",
    [["gallina","perro","gato","cabra","vaca","burro"]]);
  Card(resSet)
});
////////////////////////////////////////////////////////////////////////////
PutDescription("Pequeña prueba con nombres de animales domesticos.",aniDom);
////////////////////////////////////////////////////////////////////////////
```

Real aniSal

```
////////////////////////////////////////////////////////////////////////
Real aniSal = If(!(tstCmd <: [["aniSal","all"]]), FALSE,
{
  Set resSet = SolvePrint(
    "gallina\n"+
    "abvacac\n"+
    "saogato\n"+
    "norubit\n"+
    "osrirgo\n"+
    "eoelarñ\n"+
    "lipateu\n",
    [["tigre","leon","oso","ñu","tiburon","aguila"]]);
  Card(resSet)
});
////////////////////////////////////////////////////////////////////////
PutDescription("Pequeña prueba con nombres de animales salvajes.",aniSal);
////////////////////////////////////////////////////////////////////////
```

Real mueHog

```
////////////////////////////////////////////////////////////////////////
Real mueHog = If(!(tstCmd <: [["mueHog","all"]]), FALSE,
{
  Set resSet = SolvePrint(
    "csillon\n"+
    "asofala\n"+
    "miavion\n"+
    "alnapbo\n"+
    "slehcoc\n"+
    "earemlr\n"+
    "metroga\n"+
    "anucrab\n",
    [["sillon","cama","sofa","mesa","silla","cuna"]]);
  Card(resSet)
});
////////////////////////////////////////////////////////////////////////
PutDescription("Pequeña prueba con nombres de muebles del hogar.",mueHog);
////////////////////////////////////////////////////////////////////////
```

Real medTra

```
////////////////////////////////////////////////////////////////////////
Real medTra = If(!(tstCmd <: [["medTra","all"]]), FALSE,
{
  Set resSet = SolvePrint(
    "csillon\n"+
    "asofala\n"+
    "miavion\n"+
    "alnapbo\n"+
    "slehcoc\n"+
    "earemlr\n"+
    "metroga\n"+
    "anucrab\n",
    [["avion","globo","barco","coche","metro","tren"]]);
  Card(resSet)
});
////////////////////////////////////////////////////////////////////////
PutDescription("Pequeña prueba con nombres de medios de transporte.",medTra);
////////////////////////////////////////////////////////////////////////
```

Real paiEu7

```
////////////////////////////////////////////////////////////////////////
Real paiEu7 = If(!(tstCmd <: [["paiEu7","all"]]), FALSE,
```

```
{
  Set resSet = SolvePrint(
    "españapo\n"+
    "ailatirt\n"+
    "belgicau\n"+
    "gholanda\n"+
    "greciaal\n"+
    "lagutrop\n"+
    "refouftu\n",
    [["portugal","españa","francia","holanda","belgica","italia","grecia"]]);
  Card(resSet)
});
////////////////////////////////////////////////////////////////////////////
PutDescription("Pequeña prueba con nombres de 7 paises de Europa.",paiEu7);
////////////////////////////////////////////////////////////////////////////
```

Real paiE10

```
////////////////////////////////////////////////////////////////////////////
Real paiE10 = If(!(tstCmd <: [["paiE10","all"]]), FALSE,
{
  Set resSet = SolvePrint(
    "ageuronpap\n"+
    "luxemburgo\n"+
    "españanaml\n"+
    "mholandaao\n"+
    "aidnalnifn\n"+
    "nmeadnalri\n"+
    "inglaterra\n"+
    "austriaxic\n",
    [["españa","luxemburgo","irlanda","inglaterra","alemania","austria",
      "holanda","noruega","finlandia","polonia"]]);
  Card(resSet)
});
////////////////////////////////////////////////////////////////////////////
PutDescription("Pequeña prueba con nombres de 10 paises de Europa.",paiE10);
////////////////////////////////////////////////////////////////////////////
```

Real pla__9

```
////////////////////////////////////////////////////////////////////////////
Real pla__9 = If(!(tstCmd <: [["pla__9","all"]]), FALSE,
{
  Set resSet = SolvePrint(
    "jopitertierrav\n"+
    "riosaturntulpe\n"+
    "arcaooemnutpen\n"+
    "runtnboolndnbu\n"+
    "rceuaoesifhaus\n"+
    "errrretipujrlj\n"+
    "ieonutpenotulp\n"+
    "mmromarteuclab\n",
    [["plutón","neptuno","urano","saturno","jupiter","marte","tierra","luna",
      "venus","mercurio","sol"]]);
  Card(resSet)
});
////////////////////////////////////////////////////////////////////////////
PutDescription("Prueba 9 planetas, de la lista algunos no son.",pla__9);
////////////////////////////////////////////////////////////////////////////
```