Topological Sort

Zip

A restaurer dans \Project9

Utilisation dans un projet

(1) Ajouter au projet
   a. \Project9\Tools\Prog\Dependency.prg
   b. \Project9\Tools\Prog\QueueBits.prg
   c. \Project9\Tools\Prog\StackBits.prg
   d. \Project9\Tools\Prog\Tools_Base.prg
   e. \Project9\Tools\Prog\Tools_Classes_base.prg
   f. \Project9\Tools\Prog\Tools_Object_base.prg
   g. \Project9\Tools\Include\Tools_base.h (a exclure)

(2) Set proc edure to \Project9\Tools\Prog\Tools_Base additive

(3) If (`Tools_Init()`)
  && ok
else
  && probleme
endif

(4) Tools_init()
   a. Va ajouter Dependency() et QueueBits() a Set Procedure
   b. Ajouter un objet a _screen

Contenu

Dependency

Contient le Topolgical sort a la Foxpro

L'algorithme se trouve ici http://www.brpreiss.com/books/opus4/html/page557.html

QueueBits

Contient une classe Queue

StackBits

Contient une classe Stack

Tools_Base:

Contient quelques fonctions et quelques classes de base

Tools_Classes_base

quelques classes de base

Tools_Object_base

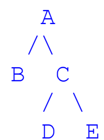
Quelques function – ea la creation objet

Utilisation de la classe Dependency

```
#if FALSE
      * a = b + c
       * c = d + e
       *
       * should get b d e c a (or d e c b a)
       */
#endif

Il faut évaluer
   -  b avant a
   -  c avant a
   -  d avant c
   -  e avant c

On a un arbre
```




```
function Test_02()

      local obj, QueueObj, CycleInfoString, s

      && on cree l'objet
      assert Dependency_Object(@m.obj)

      && ajouter les dependences : Successeur, Predecesseur
      assert m.obj.Edge("a", "b")
      assert m.obj.Edge("a", "c")
      assert m.obj.Edge("c", "e")
      assert m.obj.Edge("c", "d")

      && demander le tri
      assert m.obj.Dependency(@m.QueueObj, @m.CycleInfoString)

      && l'ordre d'evaluation
      do while m.QueueObj.DeQueue(@m.s)
            ?m.s
      enddo
      && b
      && e
      && d
      && c
      && a

S'il y a un (ou plusieurs) cycle(s) , la methode Dependency() retournera
FALSE et CycleInfoString contiendra un cycle

Endfunc
```

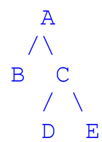
Test avec un cycle

```
#if FALSE
      * a = b + c
       * c = d + e
       *
       * should get b d e c a (or d e c b a)
       */
#endif
```

Il faut evaluer

- b avant a
- c avant a
- d avant c
- e avant c

On a une foret d'un arbre




```
function Test_03()

      local obj, QueueObj, CycleInfoString, s

      && on cree l'objet
      assert Dependency_Object(@m.obj)

      && ajouter les dependences : Successeur, Predecesseur
      assert m.obj.Edge("a", "b")
      assert m.obj.Edge("a", "c")
      assert m.obj.Edge("c", "e")
      assert m.obj.Edge("c", "d")
      assert m.obj.Edge("e", "a") && Le cycle

      && demander le tri, retourne FALSE
      assert m.obj.Dependency(@m.QueueObj, @m.CycleInfoString)

      et CycleInfoString contient :
            e,c,a


S'il y a un (ou plusieurs) cycle(s) , la methode Dependency() retournera
FALSE et CycleInfoString contiendra un cycle

Endfunc
```