Modélisation d'un tableur

Un tableur est un programme permettant de gérer un *tableau* constitué d'une matrice de valeurs et d'une matrice de formules. Voici un exemple de tableau :

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

`valeurs`

	0	1	2
0		A_{22}	A_{00}
1	$2A_{20}$	$A_{20} + A_{22}$	$A_{10} + A_{21}$
2	A_{02}	$A_{11} + A_{01}$	

`formules`

Chaque formule indique comment calculer l'élément de la matrice des valeurs ayant même position, par exemple la cellule ligne 1, colonne 2 contient la formule $A_{10} + A_{21}$ ce qui signifie que la valeur ligne 1, colonne 2 doit être égale à la somme des valeurs ligne 1, colonne 0 et ligne 2, colonne 1. Manifestement ce n'est pas le cas et le rôle du tableur est de remplacer les éléments de la matrice des valeurs qui sont associés à une formule par l'évaluation de la formule en question. A priori il suffit d'évaluer chaque formule et de modifier en conséquence la matrice des valeurs, mais il se pose un problème délicat : dans quel ordre doit-on évaluer les formules ?

Ce problème est appelé *tri topologique :* étant donné un ensemble fini d'objets et un ensemble fini de relations de dépendances entre ces objets, classer les objets de sorte qu'aucun ne dépende d'un objet situé après lui dans le classement. L'algorithme du tri topologique est le suivant :

1. Noter pour chaque objet o de combien d'objets il dépend et quels sont les objets qui dépendent de lui. Ces objets seront appelés successeurs *de o. Initialiser la liste résultat à* $\varnothing$.
2. Tant qu'il reste des objets ayant un compte de dépendance nul faire :
2.1. Sélectionner un objet ayant un compte de dépendance nul, le placer dans la liste résultat et diminuer d'une unité les comptes de dépendance de tous ses successeurs.
3. S'il reste des objets non classés le problème n'a pas de solution, sinon retourner la liste résultat.

La terminaison de cet algorithme est évidente, la correction l'est un peu moins, on y réfléchira. L'objet de ce TP est d'implémenter l'algorithme du tri topologique dans le cas particulier d'un tableur. On utilisera les déclarations Caml suivantes :

```
type lcell  == (int*int) list;;          (* liste de cellules     *)
type formule = Rien | Somme of lcell;;
type tableau = {
          n  : int; p:int;               (* dimensions            *)
          v  : int vect vect;            (* valeurs des cellules  *)
          f  : formule vect vect;        (* formules              *)
          dep: int vect vect;            (* nombre de dépendances *)
          suc: lcell vect vect;          (* successeurs           *)
  mutable lt : lcell                     (* ordre de calcul       *)
};;
```

Si `t` est une variable de type `tableau` et i, j sont deux entiers on aura :

– `t.n` est le nombre de lignes du tableau ;
– `t.p` est le nombre de colonnes du tableau ;
– `t.v.(`i`).(`j`)` est la valeur ligne i, colonne j ;
– `t.f.(`i`).(`j`)` est la formule ligne i, colonne j ;
– `t.dep.(`i`).(`j`)` est le compte de dépendance de la cellule ligne i, colonne j ;
– `t.suc.(`i`).(`j`)` est la liste des successeurs de la cellule ligne i, colonne j ;
– `t.lt` est la liste des cellules triée par ordre topologique.
– Si `t.f.(`i`).(`j`) = Rien` alors la valeur `t.v.(`i`).(`j`)` n'est soumise à aucune contrainte.
– Si `t.f.(`i`).(`j`) = Somme[`$c_1; \ldots; c_k$`]` alors la valeur `t.v.(`i`).(`j`)` doit être égale à la somme des valeurs des cellules $c_1, \ldots, c_k$.

Ici on s'est limité aux formules de type addition de cellules, d'autres types de formules pourraient être naturellement implémentées. Le tableau donné en exemple est défini par :

```
let t = {
  n = 3; p = 3;
  v = [| [|  1;   2;   3  |];
         [|  4;   5;   6  |];
         [|  7;   8;   9  |] |];
  f = [| [|Rien;            Somme[2,2];      Somme[0,0]      |];
         [|Somme[2,0; 2,0]; Somme[2,0; 2,2]; Somme[1,0; 2,1] |];
         [|Somme[0,2];      Somme[1,1; 0,1]; Rien            |] |];
  dep = make_matrix 3 3 0;
  suc = make_matrix 3 3 [];
  lt  = []
};;
```

1) Programmer l'algorithme du tri topologique. Voici à titre indicatif une décomposition possible des opérations à effectuer :

`dépendances : tableau -> unit`
: calcule la matrice `dep` d'un tableau à partir de sa matrice `f`.

`successeurs : tableau -> (int*int) -> lcell -> unit`
: ajoute la cellule spécifiée en deuxième argument (couple ligne,colonne) aux listes de successeurs de toutes les cellules dont la liste est fournie en troisième argument.

`place : tableau -> (int*int) -> unit`
: place une cellule dans la liste `lt` et décrémente le compte de dépendance de chaque successeur de la cellule placée.

`décompte : tableau -> lcell -> unit`
: décrémente les comptes de dépendance des cellules transmises en deuxième argument et appelle `place` pour chaque cellule dont le compte de dépendance devient nul.

`tri_topo : tableau -> unit`
: effectue le tri topologique du tableau transmis.

`place` et `décompte` sont deux fonctions mutuellement récursives, elles devront être déclarées de façon conjointe par : `let rec place t (i,j) = ... and décompte t liste = ...;;`

2) Maintenant que le tableau est « trié topologiquement » il reste à évaluer les formules dans l'ordre et à modifier en conséquence la matrice `v` du tableau. Programmer celà. On doit trouver pour le tableau servant d'exemple les valeurs suivantes :

	0	1	2
0	1	9	1
1	2	10	21
2	1	19	9

3) Complément : lorsque l'on modifie la valeur d'une cellule qui n'est pas soumise à contraintes, il faut réévaluer toutes les formules qui font intervenir cette cellule directement ou indirectement. Une solution naïve consiste à recalculer tout le tableau, mais on peut faire mieux en notant pour chaque cellule non contrainte la liste des cellules à recalculer, triée par ordre topologique.