TD I41 - 1 Compression de Tableau

Dans de grandes applications numériques, les tableaux contiennent un grand nombre de 0. Ceci a pour conséquence de rendre des calculs inutiles et d'occuper de l'espace mémoire inutilement. Afin remédier à ce problème on utilise une méthode de compression de tableau pour obtenir des *tableaux ou vecteurs creux*. Un tableau creux est un tableau dont les 0 ont été ôtés.

On veut définir une arithmétique sur des vecteurs creux qui permettent des faire des calculs sur la version compressée. *Ces calculs doivent correspondre aux calculs que l'on aurait fait sur des vecteurs non compressés et dont on aurait compressé le résultat.*

1 Compression de vecteurs

Pour cela on utilise la structure suivante

```
typedef struct CREUX {int nb ; float V[N] ; int I[N];} creux_t;
```

Pour cette structure, le champs `nb` correspond au nombre de valeurs non nulles, le champs `V` correspond aux valeurs non nulles, le champs `I` est un tableau décrivant la position des éléments dans le tableau initial non compressé.

Exemple : Considérons le tableau `A1={10,0,0,0,20,0,0,30,40}` la version compressée correspond à `C1={ nb=4 ; V={10,20,30,40} ; I={0,4,7,8} }`
Considérons le tableau `A2={100,200,0,0,300,0,0,0,0}` la version compressée correspond à `C2={ nb=3 ; V={100,200,300} ; I={0,1,4} }`

Attention vous ne devez pas décompresser les tableaux pour ces exercices.

1. Faites une fonction `int nonzero(float A[M])` qui compte le nombre d'éléments different de 0.
2. Faites une fonction `void compress(float A[M], creux_t *C)` qui compresse le tableau A pour avoir une représentation creuse C.
3. Faites une fonction `void add(creux_t C1, creux_t C2, creux_t *C3)` qui additionne les deux vecteurs $\vec{c}_1, \vec{c}_2$ compressés. Le résultat est le vecteur $\vec{c}_3$ (ie. $\vec{c}_1 + \vec{c}_2 = \vec{c}_3$.

   Exemple : L'addition de `C1` et de `C2` décrit précédemment donne : `C3 = {nb=5, V=(110,200,320,30,40), I=(0,1,4,7,8)}`.
4. Faites une fonction `void mul(creux_t C1, creux_t C2, creux_t *C3)` qui multiplie les deux vecteurs $\vec{c}_1, \vec{c}_2$ compressés. Le résultat est le vecteur $\vec{c}_3$ (ie. $\vec{c}_1 + \vec{c}_2 = \vec{c}_3$.

   Exemple : La multiplication de `C1` et de `C2` donne `C3 = {nb=2, V={1000,6000}, I={0,4}}`.
5. Faites une fonction `float scalprod(creux_t1 C,float D)` qui calcule le produit scalaire de $a = \sum c_i.d_i$ entre un tableau creux (C) et un tableau dense (D).

2 Compression de matrices

On applique cette approche non plus au vecteurs mais aux matrices (tableau à deux dimensions `A[M][M]`). Pour cela on utilise un format spécial appelé CRS Compressed Row Storage (ou CCS). On utilise la structure suivante :

`struct CRS { float DA[N] ; int CO[N] ; int RO[N] } crs_t`

(note : `M+1` est syntaxiquement faux)

- `DA` : ensemble des données différentes de 0
- `CO` : ensemble des positions colonnes (resp. lignes) où se situent les éléments sur un tableau dense
- `RO` : `RO(`i`)`=le rang minimal sur `DA` possédant une valeur de la ligne i (resp colonne), RO$(i+1)$ sinon.
- `RO[`M`]` correspond au nombre d'éléments non nul.

- Exemple -

$N = 6, M = 4$

$$\begin{pmatrix} 0 & a & 0 & b \\ 0 & c & 0 & d \\ 0 & 0 & 0 & 0 \\ e & 0 & f & 0 \end{pmatrix}$$

DA	CO	i	RO
a	1	0	0
b	3	1	2
c	1	2	4
d	3	3	4
e	0	4	6
f	2		

1. faites une fonction `void mat2crs(crs_t *S,float A[M][M])` compressant la matrice A.
2. faites la foction `matvec(crst_t S,float X[M],float Y[M])` qui calcule la multiplication matrice vecteur en considérant le tableau creux et un vecteur dense.
3. tester le programme creux et dense (temps d'exécution) (cf. time.h) avec des matrices créees aléatoirement.

```
version séquentielle de la multiplication matrice vecteur
for(i=0; i<M;i++)
 {
  Y[i]=0;
  for(j=0;j<M;j++)
  Y[i]+=A[i],[j]*X[J];
 }

/* Corrections du dernier exercice qui est le plus dur  */
/* A supprimer pour les étudiants */
for(i=0;i<M;i++)
  { Y[i]=0;
  for(j=S.RO(i);j<S.RO(i+1);j++)
   Y[i]+=S.DA[j]*X[S.CO[j]];
   }
```