Le but de ce td est dans un premier temps de faire le point sur les deux grandes structures de données de Maple que sont les listes et les tableaux (section 1). Dans une deuxième partie, nous verrons plus en détail le cas des matrices. Une troisième section contient quelques applications possibles.

1 Structures de données

L'intérêt des structures de données est fondamental en informatique, partout présentes dans les problèmes pratiques : gestion des billets à la SNCF, ou dans toute autre entreprise de voyage, liste d'attente dans un funérarium, etc... Dans un premier temps nous réviserons les **listes**, nous regarderons ensuite les **tableaux**.

1.1 Listes

La structure de liste est *récursive*, et est définie de la manière suivante : une liste L est
- soit la liste vide : `[]`
- soit la concaténation d'une liste L' et d'un élément a : `L :=[a,op(L')]`

C'est donc une structure dynamique pour laquelle la *taille* - *i.e* le nombre d'éléments de L - peut évoluer au cours des besoins et des manipulations.

1.1.1 Opérations élémentaires

Les exemples qui suivent sont fondamentaux et doivent être parfaitement maîtrisés :
Définition d'une liste : `[> L := [1,4,9,16];`
Liste vide : `[> L := [];`
Longueur d'une liste : `[> nops(L);`
i-ème élément de la liste : `[> L[i];` i est un entier compris entre 1 et longueur(L)
Extraction d'une sous liste : `[> L[i..j];` $1 \leq i \leq j \leq$ longueur(L)
Ajout d'un élément en fin de liste : `[> L :=[op(L),23];`
Ajout d'un élément en début de liste : `[> L :=[23, op(L)];`
Définition "automatique" : `[> L :=[seq(k,k=1..5)];`
Opération sur tous les éléments : `[> map(x->x^2,L);`

Il est essentiel de noter le lien entre listes et séquences (idem, sans les crochets autour). Ainsi on peut construire des listes "complexes" en utilisant le constructeur de séquences `seq`. De plus, les transformations de liste en séquence (`op`) et réciproquement (`[.]`) sont très simples.

1.1.2 Quelques exercices

Exercice 1 *Définir une procédure prenant en entrée un entier n et renvoyant une liste de taille n remplie aléatoirement avec des entiers de 1 à n.(*Indication *: utilisez la fonction* `rand`*).*

Exercice 2 *Écrire une procédure* `div` *renvoyant la liste des diviseurs d'un entier n. On définit un nombre* parfait *comme étant un nombre égal à la somme de ses diviseurs stricts* - i.e. *n non compris. En utilisant* `div`*, donnez un programme* `parfait` *prenant un entier n en entrée, qui renvoie* `true` *si n est parfait et* `false` *sinon. Donnez un exemple d'entier parfait.*

On dit de deux nombres entiers m et n qu'ils sont amicaux *si et seulement si la somme des diviseurs stricts de n* - i.e. *n non compris - est égale à m et si la somme des diviseurs stricts de m vaut n. Programmez en MAPLE une procédure* `amicaux` *ayant deux entiers comme entrée et qui renvoie* `true` *si et seulement si m et n sont amicaux. Exhibez un couple d'entiers amicaux.*

Donnez dans chaque cas une estimation de la complexité de chaque procédure.

1.2 Tableaux

Un *tableau* est un autre type de structure de données fondamental. Concrètement un tableau est créé avec la commande `array` - c'est le moment de se précipiter sur l'aide pour `array`. Les tableaux peuvent être unidimensionnels, auquel cas ils s'apparentent aux listes, au détail près que leur taille reste constante. Ainsi, la structure de tableau n'est pas dynamique. En particulier, il n'existe pas de commande donnant la taille d'un tableau.

1.2.1 Opérations élémentaires

- Définition :
  `[> A :=array(0..10,1..20);` crée un tableau non initialisé de 11 lignes et 20 colonnes indexées respectivement de 0 à 10 et de 1 à 20.
  `[> A :=array(1..3,[1,2,3]);` crée un tableau à une dimension initialisé avec le vecteur $[1, 2, 3]$.
- Accès aux valeurs/Modification des valeurs :
  Accès à un élément : `[> a :=A[1,2];`
  Modification d'un élément : `[> A[1,2] :=3;`
- Affichage : `[> print(A);`
- Problème de copie d'un tableau : `evalm()`
  Comparez les deux séquences d'instructions suivantes :

```
[> A:=array(1..3,[4,5,6]);          [> A:=array(1..3,[4,5,6]);
[> B:=A;                            [> B:=evalm(A);
[> B[1]:=0;                         [> B[1]:=0;
[> print(B);                        [> print(B);
[> print(A);                        [> print(A);
```

- Opération sur tous les éléments : `[> map(x->x^2,A);`

1.2.2 Un exercice d'application

Exercice 3 (Carrés Magiques)

1. *Un tableau* `T` *à* `n` *lignes et* `p` *colonnes étant donné, écrire une procédure* `ligne(T,p,i)` *qui calcule la somme des éléments de* `T` *situés en ligne* `i` .
2. *Un tableau* `T` *à* `n` *lignes et* `p` *colonnes étant donné, écrire une procédure* `colonne(T,n,j)` *qui calcule la somme des éléments de* `T` *situés en colonne* `j` .
3. *Un tableau carré* `T` *à* `n` *lignes et* `n` *colonnes étant donné, écrire une procédure* `diagonale1(T,n)` *qui calcule la somme des éléments de* `T` *situés sur la première diagonale.*
4. *Un tableau carré* `T` *à* `n` *lignes et* `n` *colonnes étant donné, écrire une procédure* `diagonale2(T,n)` *qui calcule la somme des éléments de* `T` *situés sur la seconde diagonale.*
5. *Un tableau carré* `T` *à* `n` *lignes et* `n` *colonnes est dit magique si la somme des éléments de n'importe quelle ligne, de n'importe quelle colonne, et de n'importe quelle diagonale est la même .*
   Utiliser les procédures précédentes pour écrire une fonction booléenne `magique(T,n)` *rendant la valeur* `true` *si* `T` *est magique, et* `false` *sinon .*

2 Matrices

Pour travailler sur des matrices, on utilisera la librairie `linalg` qui définit un constructeur `matrix` et de nombreuses fonctions classiques sur les matrices. Il est important de remarquer que l'objet `matrix` de Maple n'est qu'un cas particulier de l'objet `array` : la dimension vaut 2 et les lignes et les colonnes sont toujours numérotées à partir de 1.

2.1 Définition

```
[> with(linalg);
[> A:=matrix(5,5):print(A);
[> B:=matrix(2,2,[[2,-3],[1,4.5]]);
[> B[1,2]:= 0; print(B);
[> C:=matrix(2,4,[1,0,5,6,-9,0,0,3]);
[> E:=diag(1,2,3);
[> F:=matrix(8,8,(i,j)->if i=j then 0 else i+j fi);
[> G:=matrix(4,6,3);
```

Exercice 4 *Définissez les matrices suivantes :*

$$N = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \text{ et } \qquad M = \begin{pmatrix} 1 & 2 \\ 0 & 1 \\ 2 & 0 \end{pmatrix}$$

Exercice 5 *Écrivez une procédure* `Id(n)` *retournant la matrice identité de taille* $M_n(\mathbb{K})$.

2.2 Quelques outils concernant les matrices

2.2.1 Matrices comme tableaux

Comme dit plus haut, les matrices sont des tableaux particuliers. Leur utilisation est donc proche de celle des tableaux. Ainsi, les mêmes difficultés apparaissent pour la copie de la matrice. Il faut dans ce cadre utiliser sans retenue l'opérateur `evalm`.

D'une façon générale. il faut être vigilant lors des affectations. Ainsi, pour effectuer une opération de réaffectation comme `[> M :=3*M;`, il faut taper `[> M :=evalm(3*M);`.

2.2.2 Matrices avec `linalg`

De nombreux outils matriciels d'algèbre linéaire sont disponibles dans le package `linalg` qui se charge en tapant l'instruction `[> with(linalg);`. On ira donc voir avec intérêt l'aide sur ce package.

2.3 Calculs matriciels

Vous avez appris en cours que $M_n(\mathbb{K})$ est une algèbre, ce qui veut dire que concrètement, vous pouvez faire la somme et le produit de deux éléments - rappelez ces définitions -, et qu'il existe un produit extérieur entre un élément $\lambda \in \mathbb{K}$ et une matrice $M \in M_n(\mathbb{K})$. Ainsi le produit de deux matrices se fait en utilisant '&∗', la puissance d'une matrice s'obtient avec à '^', soyez attentif à l'effet de '^0'.

Exercice 6 *Soit* $A = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$.

Calculez A^k *pour les premières valeurs de* k*, puis déduisez la valeur générale de* A^k *pour* k *quelconque. Généralisez cette constatation pour*

$$A_n = \begin{pmatrix} 0 & 1 & \cdots & 1 \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & 1 \\ 0 & \cdots & \cdots & 0 \end{pmatrix}$$

Exercice 7 *Soient* $M = \begin{pmatrix} -57 & 674 & -393 \\ -644 & 1808 & -956 \\ -899 & 1718 & -851 \end{pmatrix}$ *et* $N = \begin{pmatrix} 29 & 22 & 21 \\ 87 & 66 & 63 \\ 145 & 110 & 105 \end{pmatrix}$. *Calculez* $M * N$ *et* $N * M$. *Qu'observez-vous ? Quelle propriété fondamentale de l'anneau des entiers n'est pas conservée dans celui des matrices ?*

Exercice 8 *Soit* $B = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$. *B est-elle inversible ? Si oui, donnez* B^{-1}.

3 Applications

3.1 Matrices

Exercice 9 *Donnez le rang des matrices suivantes. Lorsque c'est possible, donnez-les inverses - en précisant éventuellement les cas où l'on peut inverser :*

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix}, \begin{pmatrix} 1 & a & b \\ b & 1 & a \\ a & b & 1 \end{pmatrix}.$$

Exercice 10 *On considère la matrice* 5×5, $M = (m_{ij})_{1 \leq i,j \leq 5}$ *telle que :* $m_{ij} = \begin{cases} 0 & \text{si } i > j \\ j - i + 1 & \text{si } i \leq j \end{cases}$.

Calculer M^{-1}. *Faites de même avec la matrice* $M = (min(i,j))_{i \leq i,j \leq 5}$.

Exercice 11 *Soit* $A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -2 & -9 \\ 0 & 1 & 4 \end{pmatrix}$. *Calculez* $(A - Id_3)^2$. *En déduire une méthode pour calculer* A^n *pour tout entier* n. *De manière générale, trouver un* ***polynôme annulateur*** *d'une matrice est une méthode très efficace pour calculer les puissances successives de cette matrice.*

3.2 Tableaux

Exercice 12 (parcours en escalier d'un tableau) *Soit* n *un entier naturel non nul. On désigne par* M *le tableau de* $n+1$ *lignes et* $n+1$ *colonnes défini par* $M_{i,j} = 2^i 3^j$ *pour* $0 \leq i, j \leq n$. *Remarque cruciale quant à l'indexation : la première ligne et la première colonne sont d'indice* 0.

1. *Donnez une fonction de création du tableau* M *dépendant de* n. *On veillera bien à ce que la complexité soit un* $O(n^2)$.
2. *Écrivez une fonction qui prend en argument quatre entiers* i, j, k, l *entre* 0 *et* n *et renvoie le pgcd des coefficients* $M_{i,j}$ *et* $M_{k,l}$ *de* M. *La complexité de cette fonction devra être de* ***deux comparaisons*** *entre entiers au plus.*
3. *Soit* p *un entier compris entre* 1 *et* $2^n 3^n$. *Donnez une fonction* `borneinf`(p) *qui renvoie le plus grand minorant de* p *parmi les coefficients de* M. *Pour réduire la complexité autant que possible, on effectuera la constatation suivante : si sur la ligne* i *le plus grand minorant de* p *est en colonne* j, *alors sur la ligne* $i+1$, *le plus grand minorant de* p *se trouve en position :*
   - $(i+1, j)$ *si* $M_{i+1,j} \leq p$.
   - $(i+1, j-1)$ *sinon, car dans ce cas :* $M_{i+1,j-1} = \frac{2}{3} M_{i,j} \leq p$.

Vous préciserez le nombres de comparaisons effectuées par la fonction `borneinf`.