

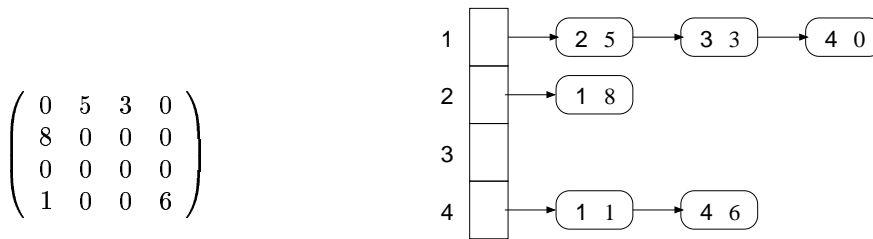
Licence Miage - Programmation 1
Examen de janvier 2002

Durée : 3 heures.
 Documents de cours autorisés. Calculatrices non autorisées

Les deux exercices sont indépendants. Pour chaque question, même si ce n'est pas précisé, pensez à traiter proprement les situations pathologiques.

Exercice 1 : Matrices creuses - le retour (8 points)

Une *matrice creuse* est une matrice où la majorité des éléments sont égaux à 0. En TD, vous avez vu qu'il était possible de représenter une matrice creuse par une liste chaînée. Le but de cet exercice est d'étudier une représentation alternative, utilisant un tableau de listes chaînées : une matrice $n \times m$ est représentée par un tableau de n listes chaînées, chaque liste correspondant à une colonne. Par exemple, voici une matrice avec la représentation correspondante.



Remarquez que (4,0) permet de connaître la dimension de la matrice.

Question 1. À quoi vous fait penser cette structure de tableau de listes chaînées?

Question 2. Écrivez le type `Matrice_creuse`.

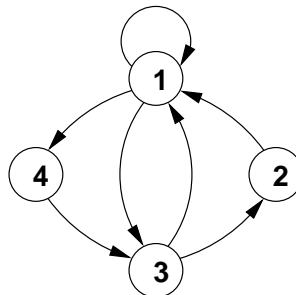
Question 3. Écrire une procédure `Get(M: Matrice_creuse; i, j: Natural; N: out Natural)` qui donne la valeur de $M(i, j)$, l'élément de M sur la ligne i et la colonne j .

Question 4. Écrire une procédure `Put(M: in out Matrice; i, j: Natural; N: Natural)` qui affecte la valeur N à $M(i, j)$.

Question 5. Écrire une fonction `+` qui fait la somme de deux matrices creuses.

Exercice 2 : Les graphes (12 points)

Un graphe orienté est une structure déterminée par un ensemble de nœuds, d'une part, et un ensemble d'arcs reliant un nœud à un autre, d'autre part. Quand il existe un arc menant d'un nœud i à un nœud j , on dit que j est un *successeur* de i . Par exemple, ce graphe compte quatre nœuds et sept arcs. Les successeurs du nœud 1 sont 1, 3 et 4, et ainsi de suite.



Par souci de simplicité, on suppose pour l'instant que si un graphe a n nœuds, ces derniers sont numérotés de 1 à n . Pour représenter un graphe constitué de n nœuds et p arcs, on définit deux tableaux

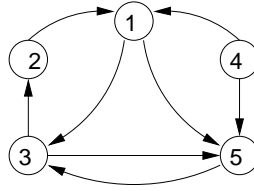
- un premier tableau T contenant n éléments,

• un second tableau T contenant p éléments

tels que les successeurs du nœud i sont énumérés dans le tableau P entre les indices $T(i)$ et $T(i + 1)$, ou $T(i)$ et $P'last$ si i est égal à $T'last$. Par exemple, pour le graphe ci-dessus, les tableaux T et P sont

T	1	4	5	7			
P	1	3	4	1	1	2	3

Question 6. Construire les tableaux T et P pour le graphe ci-dessous.



Question 7. Écrire un type `Graphe` qui permette de représenter un graphe. Attention, ce type devra pouvoir apparaître comme un type `private`.

Question 8. Écrire une fonction `Arc(G: graphe; i, j: Natural) return Boolean` qui teste si j est successeur de i dans le graphe G .

On se pose maintenant la question de savoir s'il existe un chemin menant d'un nœud i à un nœud j dans un graphe G . On vous propose la fonction récursive suivante `Chemin`.

```

if Arc(G, i, j) then
    return TRUE;
else
    for x in 1..N loop
        if Arc(G, i, x) and Chemin(G, x, j) then
            return TRUE;
        end if;
    end loop;
    return FALSE;
end if;

```

Question 9. Expliquer pourquoi cette approche ne fonctionne pas. Vous pouvez vous aider des exemples des deux graphes au-dessus.

En fait, il existe un algorithme itératif standard qui permet de déterminer s'il existe un chemin entre un nœud i et un nœud j . L'algorithme procède en construisant une matrice de booléens M de dimension $n \times n$ telle que

$$M(i, j) = \text{TRUE} \Rightarrow \text{il existe un chemin menant de } i \text{ à } j.$$

On définit la *longueur* d'un chemin comme le nombre d'arcs qui le compose. La matrice M est remplie successivement, en trouvant les chemins de longueur 1, puis en ajoutant ceux de longueur 2, de longueur 3 etc. Les règles sont

- Initialisation: $M(i, j) = \text{TRUE}$ si, et seulement si, il existe un arc menant de i à j . On a ainsi les chemins de longueur 1.
- Pour les étapes suivantes, on utilise la propriété : il existe un chemin de longueur k entre les nœuds i et j s'il existe un nœud x tel que
 1. il existe un chemin de longueur $< k$ entre x et i , c'est-à-dire $M(i, x) = \text{TRUE}$, et
 2. il existe un chemin de longueur $< k$ entre x et j , c'est-à-dire $M(x, j) = \text{TRUE}$.

Au premier passage après l'initialisation, on a ainsi trouvé tous les chemins de longueur 2, etc.

- On s'arrête quand on a rempli la matrice jusqu'aux chemins de longueur $n - 1$. En effet, comme la matrice a n nœuds, s'il existe un chemin entre deux nœuds, il en existe un de longueur inférieure à n .

Question 10. Écrire une fonction `Chemin` qui reprend cet algorithme. Quelle est la complexité ?

Question 11. Quelles autres structures de données (plus simples) aurait-on pu choisir? Quelles auraient été les avantages et les inconvénients par rapport à la représentation proposée? (Vous pouvez donner deux exemples)

Question 12. On souhaite modifier le type `Graphe`, pour permettre de manipuler des graphes dont les nœuds sont les plus généraux possibles, tout en gardant la même structure de données sous-jacente (deux tableaux). Que faut-il faire ?