Faculté des Sciences de Sfax
Département Informatique et des Communications

Année Universitaire : 2020-2021
Section : Licence en Sciences de l'Informatique
Niveau: 1ère année
Matière : Atelier de Programmation
Enseignant : Mohamed Ali Hadj Taieb

TP4 - Structures de données: Liste Linéaire

Objectifs:

- Pratiquer la programmation modulaire
- Représenter des structures de données sous forme TDA
- Utiliser les listes chainées pour implémenter des structures plus complexes
- Réutiliser des services implémentés
- Adapter des services implémentés

Exercice 1

Concevoir et réaliser un module permettant de définir le TDA **liste linéaire bidirectionnelle** à travers une représentation chainée, contenant des entiers, en exportant les services suivants :

- `creer_liste` : créer une liste linéaire vide.
- `liste_vide` : tester si la liste linéaire est vide ou non
- opérations d'adjonction d'un élément à une liste linéaire :
  - en tête : avant le premier
  - en queue : après le dernier
  - après un élément référencé
  - avant un élément référencé
- opérations de suppression d'un élément de la liste linéaire :
  - premier élément
  - dernier élément
  - un élément référencé
- `recherche` : vérifier l'existence d'un élément dans une liste linéaire. Le point de départ peut être fourni comme paramètre.
- `visiter` : visiter tous les éléments de la SD LL en effectuant pour chaque élément visité une action donnée (paramètre).
- `trier_liste` : trier les éléments de la liste linéaire dans l'ordre croissant en inspirant du tri insertion.

Ajouter **un programme de test** permettant de créer deux listes linéaires, les remplir respectivement par n et m éléments, les trier et les fusionner dans une troisième liste linéaire triée dans l'ordre croissant.

Exercice 2

Concevoir et réaliser un module permettant de gérer des **listes linéaires circulaires unidirectionnelles** à travers une représentation chainée, contenant des entiers, en exportant les services suivants :

- Création d'une liste circulaire unidirectionnelle vide
- Adjonction après un élément référencé
- Suppression d'un élément référencé
- Recherche d'un élément à partir d'une clé donnée
- Visite de tous les éléments de la liste en effectuant un traitement donné comme paramètre.

Exercice 3

Soit le type *Personne* défini par un nom, un prénom et une date de naissance.

- Définir les TDAs **Personne** et **DateNaissance** avec les services nécessaires (création, getters et setters).
- Ajouter au TDA **Personne :**
  - un service permettant de calculer et retourner l'âge d'une personne.
  - un service pour définir la relation d'ordre suivant le *nom*
  - un service pour définir la relation d'ordre suivant le *prénom*
  - un service pour définir la relation d'ordre suivant l'*âge*.
- Reprendre l'exercice 1 pour que le TDA liste linéaire bidirectionnelle soit capable de stocker des personnes.
- Adapter le service `trier_liste` pour qu'il reçoit un paramètre fonctionnel définissant la relation d'ordre à appliquer pour trier une liste linéaire de personnes.

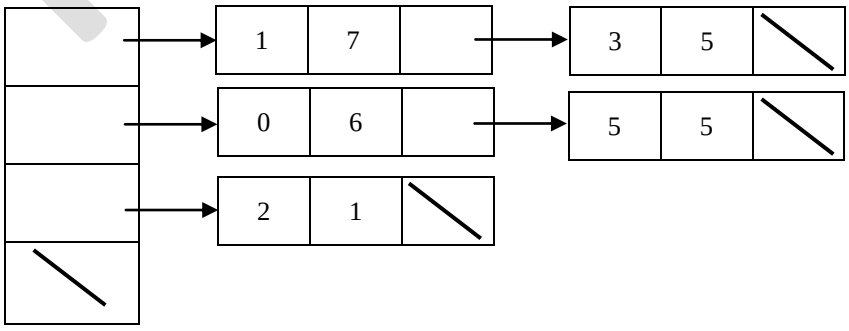
Exercice 4

Matrice Creuse

On définit une matrice creuse comme étant une matrice dont plus que la moitié de ses éléments sont nuls.

0	7	0	5	0	0
6	0	0	0	0	5
0	0	1	0	0	0
0	0	0	0	0	0

Pour minimiser l'espace occupé par la matrice on choisit de la représenter sous forme d'un tableau de listes linéaires chaînées, de telle façon à ce que la i^{ème} liste chaînée contient les éléments non nuls de la ligne *i* de la matrice et chacun d'eux accompagné du numéro de la colonne où il se trouve.



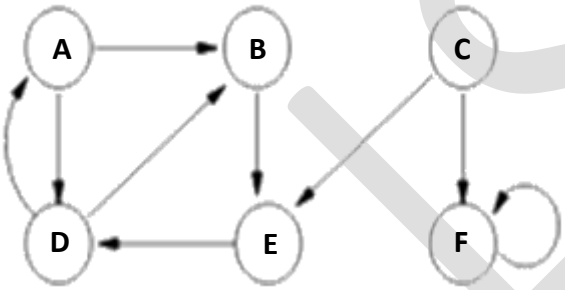
Concevoir et réaliser un module en C permettant de matérialiser la TDA MATRICE_CREUSE. Un tel module doit exporter les opérations suivantes :

- `creer_matrice` : créer une matrice creuse ayant *n* lignes.
- `ajouter_element` : ajouter un élément non nul à une ligne i et une colonne j.
- `consulter_element` : retourner l'élément [i, j]
- `existe` : vérifier l'existence d'un élément [i, j]
- `modifier_element` : modifier le contenu de l'élément [i, j]
- `afficher_matrice` : affiche le contenu d'une matrice

Exercice 5

Graphe orienté

Un graphe orienté **G** est une structure de données composé par un ensemble de nœuds **Noeuds** et un ensemble d'arcs **Arcs**. Un graphe est dit orienté lorsque le lien entre deux nœuds possède un sens.



Exemple d'un graphe orienté G<Nœuds, Arcs>

Le graphe **G** est définir par **Nœuds** l'ensemble des nœuds et **Arcs** l'ensemble des arcs tel que :

Nœuds={A, B, C, D, E, F}

Arcs={(A,B), (A,D), (B,E), (C,E), (C,F), (D,A), (D,B), (E,D), (F, F)}

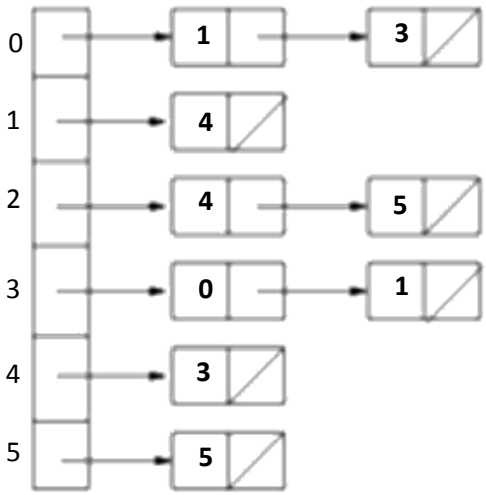
Un graphe orienté peut être représenté par :

Une matrice d'adjacence **Madj** de taille |Nœuds|x |Nœuds| avec **Madj[i][j]=1** si il existe un arc reliant les nœuds $\mathbf{n_i}$ et $\mathbf{n_j}$. Sinon, **Madj[i][j]=0**.

	A	B	C	D	E	F
A	0	1	0	1	0	0
B	0	0	0	0	1	0
C	0	0	0	0	1	1
D	1	1	0	0	0	0
E	0	0	0	1	0	0
F	0	0	0	0	0	1

Matrice d'adjacence du graphe G.

Le graphe peut être aussi représenté par un ensemble de listes linéaires dont chaque liste contient les nœuds adjacents à un nœud **n**. Ces listes sont connus sous le nom listes d'adjacences



Représentation du graphe G par les listes d'adjacence

le tableaux de listes linéaires correspondent aux nœuds avec les indices de 0 à 5 représente respectivement les nœuds de A à F. Par exemple, la première liste avec les éléments dans l'ordre croissant 1 et 3 signifie l'existence des arcs de A vers B et de A vers D.

Travail demandé:

Ecrire une solution modulaire TDA pour la représentation d'un graphe orienté défini par un ensemble de nœuds et un ensemble d'arcs.

Les services à exporter sont :

- creer_graphe : créer un graphe orienté initialement vide.
- ajouter_noeud : ajouter un nœud à un graphe donné.
- supprimer_noeud : supprimer un nœud d'un graphe donné.
- ajouter_arc : ajouter un arc entre deux nœuds appartenant à un graphe.
- supprimer_arc : supprimer un arc d'un graphe donné.
- affiche_graphe : afficher les informations d'un graphe.
- degre_noeud : retourner le degré d'un nœud dans un graphe, sachant que le degré d'un nœud est le nombre d'arcs entrants et sortants d'un nœud.
- matrice_adjacence : retourner la matrice d'adjacence d'un nœud

Remarque : L'insertion dans la liste linéaire se fait en respectant l'ordre croissant des indices attribués aux nœuds pour minimiser le cout de l'opération de recherche.

Questions :

1. Proposer une représentation physique
2. Implémenter les services cités
3. Implémenter un programme de test