Grand Devoir 1

Structures de données et algorithmes
Piles, Files d'attente et Listes (Stacks, Queues, Lists)

Mentions générales

• Pour ce projet vous allez travailler **par équipes de 2 personnes (ou seuls ☹).**

• Le projet est à rendre **sur la plateforme Moodle** (curs.upb.ro). S'il y a des problèmes lors du téléchargement vers ou depuis la plateforme, contactez par Teams ou par mail votre assistant des travaux pratiques: Iulia Stanica (iulia.stanica@gmail.com) ou Mara Chirașcu (mara.chirascu@gmail.com)

• Le projet est à rendre au plus tard le **24.04.2023, à 08:00**. Aucun retard ne sera accepté.

• Vous recevrez des questions concernant votre solution durant la séance suivante du TP. Des devoirs qui **ne seront pas présentés au TP suivant ne seront pas notés!**

• La rendue finale du projet comportera une archive nommée Etudiant1Nom_Etudiant1Prenom_ Etudiant2Nom_Etudiant2Prenom _HW1 avec:

- **les fichiers du code source (.cpp et .h)** et non pas des fichiers objets (i.e. *.o), des fichiers exécutables (i.e. *.exe) ou des fichiers codeblocks (i.e. *.cbp); SVP **mettez chaque exercice dans un dossier séparé !**
- **un fichier de type README** où vous allez spécifier (dans quelques mots) votre approche, avec des instructions pour exécuter les exercices. Au contraire, s'il y a des exigences qui ne sont pas fonctionnelles, vous pouvez proposer des idées pour une solution possible que vous allez expliquer en classe aussi (et peut-être obtenir des points supplémentaires).

• Pour toute question concernant le sujet du projet ou les exigences, envoyer un mail/message sur Teams à vos assistants ou **utiliser le canal de SDA de Teams**

• Attention : notre équipe utilise des logiciels détectant le plagiat (Moss du Stanford). Dans le cas malheureux de **plagiat, le projet sera noté par un 0 (zéro**).

! Remarque : Vous pouvez utiliser les files, les piles et les listes qu'on a fait ensemble en classe (seront représentées par des fichiers en-tête (headers), en utilisant des chablons (<template>)). Comme alternative, vous pouvez utiliser les implémentations standard du C++ POUR pile ou file d'attente, mais PAS d'autres implémentations prises de l'Internet. Pour l'exo avec liste, vous devez créer votre propre header.

1. Pile (Stack) (3p)

Vous êtes Ethan Hunt et le IMF vous a envoyé en mission pour protéger l'ingénieur nucléaire, Homer Simpson, d'un éventuel enlèvement par la société maléfique appelée The Syndicate. Homer va participer à une conférence importante. Vous rassemblez les informations sur votre cible mais vous découvrez rapidement que le IMF n'a aucune image de lui, vous ne savez donc pas à quoi ressemble votre cible. *Ce que vous savez, c'est qu'il est un spécialiste réputé et que tout le monde lui parlera. Ce que vous savez également, c'est le fait qu'Homer est un gars très timide et qu'il n'essaiera de parler à personne en retour car il ne connaît personne de l'événement.*

Ce soir, il y a un dîner formel pour tous les participants et vous êtes envoyé pour recueillir des informations sur l'interaction entre les personnes afin d'identifier votre cible. Après l'événement, vous ramenez vos notes au camp de base et vous examinez toutes les interactions qui se sont produites entre les personnes sous une matrice de forme **N x N, où 0 signifie que la personne i n'a pas parlé à la personne j et 1 signifie cette personne i a parlé à la personne j.**
À l'aide d'un Stack (pile), trouvez votre cible et sauvez-la. (Info extra: la complexité va être O(n))

PS: Il est possible qu'Homer soit si timide qu'il ait décidé de sauter le dîner. Si cela se produit, vous devez créer un autre événement afin de recueillir à nouveau des informations.

Obs : Vous pouvez utiliser la matrice d'entrée UNIQUEMENT pour sauvegarder des données, pour vérifier des valeurs des positions données (i, j), ou au MAXIMUM de traverser une SEULE ligne ± colonne. Donc vous DEVEZ utiliser pile (stack) pour obtenir la solution.

Exemple entrée: **Exemple sortie:**

```
MATRIX = { {0, 0, 1, 1, 0, 1},
           {0, 0, 1, 1, 0, 1},
           {0, 1, 0, 1, 0, 1},
           {0, 0, 0, 0, 0, 0},
           {1, 0, 1, 1, 0, 1},
           {0, 0, 0, 1, 0, 0},
```

Homer et la personne avec l'id 3

Exemple entrée: **Exemple sortie:**

```
MATRIX = { {0, 0, 1, 1, 0, 1},
           {0, 0, 1, 1, 0, 1},
           {0, 1, 0, 1, 0, 1},
           {0, 1, 0, 0, 0, 0},
           {1, 0, 1, 1, 0, 1},
           {0, 0, 0, 1, 0, 0},
```

Tout le monde a discuté avec au moins une autre personne. Homer n'était pas présent.

Points:

- 0.5p lecture de données et construction matrice
- 2p algorithme principal AVEC pile
- 0.5p toutes les validations, affichage, y compris le cas "impossible".

2. File d'attente (Queue) (3p)

Dans un restaurant, on utilise une file d'attente pour sauvegarder les commandes des clients – on a un seul chef. Chaque commande est représentée par deux nombres : le temps t (quand le client est arrivé dans le restaurant) et la durée d (durée de préparation de son plat). Les commandes doivent être triées en ordre croissante, en fonction de leur temps d'arrivée. Les commandes sont sauvegardées dans une file d'attente dans l'ordre de leur arrivée. Chaque fois qu'un plat a été préparé, la commande correspondante est éliminée de la file d'attente et le chef commence à préparer le plat suivant (s'il y en a un). Le restaurant va être fermé au temps T.

a) (0.5p) Choisissez une méthode convenable pour représenter les temps et les durées des commandes (Bonus pour des structs / classes). Lisez du clavier les nombre N, T et ensuite N paires (temps, durée).
b) (0.75p) Affichez les moments quand la queue devient vide (ou les intervalles de temps quand elle est vide) durant le programme de fonctionnement du restaurant.
c) (0.5p) Affichez la durée maximale des commandes reçues.
d) (0.75p) Pour chaque commande, afficher le temps théorique de sa complétion (ce que le client attendait) et le temps vrai de sa complétion (quand le restaurant a servi en réalité le plat).
e) (0.5p) Déterminer s'il y a des commandes qui ont été complétées après la fin du programme de fonctionnement du restaurant.

Ex :
N = 6, T = 20, Les commandes sont :
t = 0, d = 5
t = 1, d = 3
t = 10, d = 3
t = 11, d = 2
t = 12, d = 4
t = 18, d = 5

Ordre 1 : temps attendu complétion = 5, temps vrai = 5 ;
Ordre 2 : temps attendu complétion = 4, temps vrai = 8 ;
Entre les moments 8 et 10, la file d'attente est vide (on n'a pas d'ordres) etc.

Notre restaurant est ouvert jusqu'au moment T = 20. Après les calculs, on obtient que la dernière commande a été complétée au moment t = 24, donc on a des commandes qui ont été complétées après la fin du programme de fonctionnement du restaurant.

3. Listes (Lists) (3p)

Suite à une grande découverte, vous réussissez à voyager dans le temps avec votre ami. Vous arrivez dans **l'Empire romain**, mais à cause d'erreurs de calcul, vous vous retrouvez dans différents coins de ce grand empire. Mais souvenez-vous que tous les chemins mènent à Rome… ou au moins c'est ce qu'ils vous ont dit. Vous êtes tous les deux situés quelque part dans l'empire, sachant qu'entre chacun d'entre vous deux et Rome, il doit y avoir au moins une route. Les routes seront certainement intersectées quelque part…

Toi et ton ami connaissez tous les deux l'informatique, vous savez donc que toute route menant à Rome peut être écrite comme une **liste linéaire simplement chaînée** (singly-linked linear list), où les nœuds sont les villes traversées par la route. Tenant compte que vous connaissez tous les deux votre emplacement dans l'empire (la ville où vous êtes situé) et la destination (Rome), vous devez trouver la ville où vos 2 routes s'intersectent.

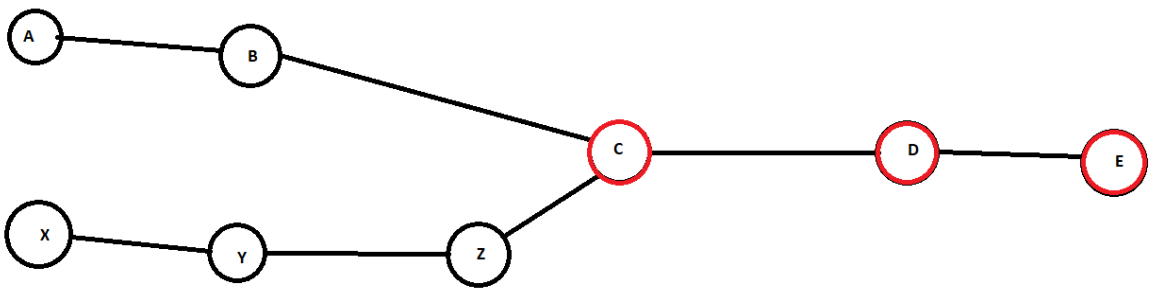
Étant donné les pointeurs des premiers nœuds des 2 listes qui représentent vos routes, votre tâche consiste à trouver le point d'intersection des vos 2 routes et à imprimer le nom de la ville. Vous devez représenter les villes sous forme de structure / classe!

Exemple entrée:

votre route: Alexandria -> Athens -> Tomis -> Sarmizegetusa -> Aquileia -> Rome
la route de votre ami: Napoca -> Sarmizegetusa -> Aquileia -> Rome

En regardant les 2 listes de l'exemple, on peut observer que les 2 routes vont se croiser à **Sarmizegetusa** (votre sortie).

Votre tâche principale consiste à écrire la fonction **findRoadIntersection** qui prend comme paramètres 2 pointeurs vers les têtes des 2 listes linéaires simplement chaînées et de retourner les "infos" du point d'intersection entre les 2 listes. *!!! Attention !!!* Il peut y avoir aussi le cas où il n'y a pas d'intersections.



Dans cet exemple, la fonction renverra "C", puisque c'est le premier nœud où les 2 listes se croisent.

Points:

- 0.5p representation ville sous forme de struct / class (a vous de décider paramètres, méthodes etc.)
- 0.5p création du header contenant une liste linéaire simplement chaînée (singly-linked linear list)
- 0.5 construction listes des 2 routes
- 1p fonction correcte findRoadIntersection
- 0.5p toutes les validations, affichage, y compris le cas "impossible"

1 p extra si tous les exercices s'exécutent sans erreurs de compilation