Programmation algorithmique
TP1 - Structure de données unidimensionnelle
25%

420-FC6-AG

Date de remise : 16 novembre 2018

Le but de ce TP est d'implémenter une structure de données unidimensionnelle ainsi que tous les algorithmes rattachés et de l'utiliser dans une simulation discrète.

À faire seul.

Remettez tout les fichiers de code source et d'en-tête documentés de votre projet dans un dossier compressé sur LÉA.

Vous devez respecter les règles et les conventions de nomenclature, d'indentation, de gestion des nombres magiques et de documentation. Votre projet doit être bien structuré en fichiers. Les fonctions doivent être courtes et avoir une tâche précise.

Le plagiat, partiel ou total, entraînera la note de 0. Référez-vous à l'annexe A pour les définitions officielles.

1 La structure de données “Queue”

Vous allez implémenter la structure de données `Queue`. Une queue suit le principe “premier arrivé, premier servi”.

La queue sera composée de pointeurs vers des instances de la structure `Client` (dont les membres sont précisés en plus de détails à la section 2.3). La queue doit implémenter les fonctionnalités `offrirClient`, `coupDOeilTete`, `obtenirTete`, `longueurQueue` et `imprimerQueue`.

```
typedef struct Client Client;
struct Client{ /* ... */ };

typedef struct Queue Queue;
struct Queue{ /* ... */ };

// Ajoute le client a la fin de queue.
void offrirClient(Queue* queue, Client* client);

// Retourne le premier client dans la queue.
// Si la queue est vide, NULL est retourne.
Client* coupDOeilTete(Queue queue);

// Retire et retourne le premier client dans la queue.
// Si la queue est vide, NULL est retourne.
Client* obtenirTete(Queue* queue);

// Retourne le nombre de clients dans la queue.
int longueurQueue(Queue queue)

// Imprime chaque client de la queue.
void imprimerQueue(Queue queue)
```

Vous êtes libre de choisir l’implémentation de la `Queue`, soit sous forme d’un tableau dynamique, soit sous forme d’une liste chaînée (recommandé). Votre code doit permettre les queues arbitrairement longues, utiliser une quantité raisonnable d’espace et de temps, et libérer toute la mémoire qu’il réserve.

Fournissez le pseudo-code pour les fonctions `offrirClient`, `obtenirTete` et `longueurQueue` dans leur en-tête de commentaire d’implémentation.

2 La simulation

Vous allez simuler les caisses dans une épicerie en utilisant vos structures `Queue` et `Client`. Une épicerie a *4* caisses ayant chacune une file d'attente.

La simulation dure *200* cycles.

À chaque cycle:

1. Il y a une chance sur *6* qu'un nouveau client entre dans le système. Quand un client entre, il s'ajoute à la file d'attente la plus courte. Les clients ont en moyenne *10* articles dans leur panier. Du code vous est fourni à la section 2.3 pour générer des paniers au hasard.

2. Chaque caissier traite un article du client à la première position dans sa file. Si le client à la caisse n'a plus d'articles, ce dernier quitte l'épicerie et le prochain client dans la file passe à la caisse. Si sa file est vide, le caissier ne fait rien.

3. Le système pause pendant *1000* millisecondes (Utilisez la fonction `sleep`).

2.1 L'état des caisses

À chaque *1* cycle, l'état des files d'attente et des caisses est affiché à la console. Vous devez imprimer le contenu de toutes les files d'attente.

```
ÉTAT DES CAISSES AU CYCLE #56
Caisse 1 : [{cycle #1, 1 articles}, {cycle #21, 51 articles}, {cycle #55,
 3 articles}]
Caisse 2 : [{cycle #12, 42}]
Caisse 3 : []
Caisse 4 : [{cycle #18, 9 articles}, {cycle #31, 13 articles}]
```

N'hésitez pas à modifier la valeur des paramètres de la simulation (le nombre de caisses *[4]*, le nombre de cycles *[200]*, le taux d'arrivée des nouveaux clients *[6]*, le nombre d'articles moyen par panier *[10]*, le temps de pause par cycle *[1000]* et le taux d'impression de l'état des caisses *[1]*).

2.2 Le rapport

À la fin de la simulation, un rapport est affiché, contenant, au minimum, les informations suivantes :

```
STATISTIQUES :
 Nombre de clients entrés  : 2000
 Nombre d'articles vendus  : 21487
 Temps d'attente moyen     : 36 cycles
 Temps d'attente maximum   : 214 cycles
 Nombre d'articles maximum : 82
 Nombre d'articles minimum : 1
 Nombre d'articles moyen   : 10
```

2.3 Les paniers des clients

Les informations suivantes doivent être conservées par client : le numéro du cycle auquel il est arrivé et le nombre d'articles dans son panier. Le comportement des clients est simple : une fois qu'ils sont ajoutés à une file, ils ne quittent pas tant qu'ils ne sont pas servis. Ils ne changent pas de file et ne perdent pas patience.

Le nombre d'articles dans le panier d'un client suit une distribution exponentielle. La fonction `double randExpo(double lambda)` retourne un naturel au hasard, selon la distribution exponentielle d'espérance `1/lambda` :

```c
#include <math.h>
#include <stdlib.h>

double randExpo(double lambda)
{
  double u = rand() / (RAND_MAX + 1.0);
  return -log(1 - u) / lambda;
}
```

Sur certains systèmes d'exploitation, vous devrez utiliser l'option `lm` pour compiler.

```
gcc *.c -lm
```

3 Bonus

3.1 La patience a ses limites [0 ou 5 pts]

Modifiez votre programme pour qu'un client qui a attendu trop longtemps abandonne la file. Affichez le nombre d'abandons dans le rapport.

3.2 $n-$caisses/$n-$queues VS $n-$caisses/$1-$queue [0 ou 5 pts]

Est-ce que le système de caisses serait plus efficace s'il y avait une seule file d'attente commune à toutes les caisses? Dans ce modèle, dès qu'un caissier est disponible, il sert le prochain client dans la file commune. Pour répondre à cette question, programmez une deuxième simulation qui fonctionne parallèlement à la première et comparez les rapports.

4 Critères d'évaluation

- **[20 pts]** Algorithmes corrects et efficaces sur la structure de données unidimensionnelle.
- **[12 pts]** Programmation correcte de la solution algorithmique.
- **[10 pts]** Utilisation correcte de la solution algorithmique.
- **[8 pts]** Application rigoureuse des règles de syntaxe et de lisibilité du code `C`.
- **[6 pts]** Documentation appropriée.
- **[5 pts]** Rédaction appropriée du pseudo-code pour un algorithme sur la structure de données unidimensionnelle.
- **[5 pts]** Établissement correct des entrées et des sorties et des conditions de succès.
- **[4 pts]** Éclatement approprié en fonctions et en fichiers.

Pour un total de 70 points qui seront ramenés sur 25.

A Annexe - PIEA

Politique de retard

À partir de la PIEA, pour l'enseignement aux adultes, la politique de retard dans les remises de travaux a été précisée de la manière suivante :

2.3.3.1 L'étudiant·e qui remet une production écrite (rapport de laboratoire, analyse, dissertation, recherche, etc.) ou une production concrète (vidéo, œuvre d'art, montage, etc.) en retard est pénalisé·e.
2.3.3.2 Pour les travaux dont le délai de production est d'une semaine (7 jours) ou moins, le professeur peut refuser le travail et inscrire la note zéro.
2.3.3.3 Pour les travaux dont le délai de production est de plus d'une semaine (8 jours et plus), le professeur peut enlever jusqu'à 10 % des points par jour de retard, incluant les jours de fin de semaine.
2.3.3.4 Aucun travail ne sera accepté après la remise des travaux corrigés.

Politique de plagiat

À partir de la PIEA, pour l'enseignement aux adultes, la politique de plagiat dans les remises de travaux a été précisée de la manière suivante :

3.3 Le plagiat (Dionne, 2013, p.199)
Le plagiat est l'acte de faire passer pour siens des textes, des contenus, des réponses ou des idées d'autrui, sans citer la source. Par exemple :
3.3.1 Copier le travail d'une autre personne en totalité ou en partie.
3.3.2 Utiliser l'œuvre d'autrui, des passages ou des idées de celle-ci sans en citer la source.
3.3.3 Copier une page (ou un segment de page) sur le Web sans en mentionner la source.
3.3.4 Résumer l'idée originale d'un auteur en l'exprimant dans ses propres mots, mais en omettant d'en indiquer la source.
3.3.5 Traduire partiellement ou totalement un texte sans en mentionner la provenance.
3.3.6 Utiliser le travail d'une autre personne et le présenter comme le sien, et ce, même si cette personne a donné son accord.
3.3.7 Paraphraser un texte sans mentionner la source.

3.4 La fraude, la tricherie et la tentative de tricherie

La fraude est un acte de tromperie qui vise l'obtention d'un avantage personnel, parfois au détriment des autres. Par exemple :
3.4.1 Obtenir les questions ou les réponses d'un examen avant l'évaluation.
3.4.2 Posséder du matériel non autorisé (calculatrice, formulaire, notes).
3.4.3 Obtenir une aide quelconque inappropriée ou non autorisée.
3.4.4 Utiliser du matériel non autorisé.
3.4.5 Consulter la copie d'un·e autre étudiant·e qu'il·elle soit complice ou non.
3.4.6 Réutiliser un travail produit dans un autre cours sans avoir obtenu au préalable l'accord du professeur.
3.4.7 Inventer des données dans le cadre d'un travail.
3.4.8 Remplacer un·e étudiant·e lors d'un examen.
3.4.9 Se faire remplacer par une autre personne lors de l'examen.
3.4.10 Acheter un travail et le déposer comme étant le sien.

3.5 Conséquences
Tout plagiat, toute fraude, toute tentative de plagiat ou de fraude, toute coopération à un plagiat ou à une fraude et toute présence de matériel non autorisé entraînent la note zéro pour l'activité concernée et peuvent entraîner une réévaluation des résultats antérieurs. S'il y a récidive, le cas est automatiquement signalé au comité d'admission; l'étudiant·e pourrait alors se voir exclu·e du Collège et ce dernier pourrait ne pas recommander la sanction de cet·te étudiant·e. (La définition de plagiat et tricherie est donnée à la partie 3.3 de la PIEA)

Politique linguistique

Le travail doit être produit en langue française (commentaires et documentation).

Politique de contenu

Sont à proscrire tous les contenus à caractère irrespectueux, diffamatoire ou explicite, qu'ils soient sexuels, racistes, religieux ou autres. Tout contenu devra être validé par le professeur avant d'être diffusé.