Université Bordeaux 1, Sciences et technologies
UFR Informatique et Mathématiques

Maîtrise d'informatique
Session de janvier 2003

Techniques et Fondement de la Programmation (TFP)
(partie "techniques")
Devoir maison dû le 15 janvier à 12 :00
Version préliminaire

Ce devoir est à faire en binômes, ou exceptionnellement seul. Chaque étudiant peut participer à au plus un binôme et peut contribuer au code d'au plus une copie rendue.

Les deux membres d'un binôme collaborent sur le devoir, mais ne doivent pas discuter des questions relatives au devoir avec un membre d'un autre binôme. Les questions de nature générale doivent être posées par courrier électronique à strandh@labri.fr. La réponse sera envoyée (également par courrier électronique) à l'expéditeur normalement au plus 24h après réception de la question. Si la question est jugée d'intérêt général, la question et la réponse seront postées dans emi.general. Veuillez consulter emi.general avant d'envoyer une question.

Vous êtes encouragés à consulter tout document (livre, article, site web, etc) afin de vous aider à produire votre résultat. Les règles habituelles s'appliquent, à savoir que toute inclusion de code trouvé ailleurs doit être déclarée en tant que telle. L'acte de faire passer du code écrit par une autre personne comme étant le vôtre s'appelle "plagiat" et est formellement interdit.

Toute collaboration non permise, plagiat détecté ou soupçonné sera rapportée au président de l'université. La pénalité possible inclut l'exclusion provisoire ou définitive de l'université.

La copie rendue doit être étiquetée par *deux numéros d'anonymat*, ou (exceptionnellement) un seul. Toute copie rendue portant plus de deux numéros d'anonymat sera considérée comme non valide. Le même barème sera appliqué à une copie portant deux ou un seul numéro. Vous avez donc intérêt à travailler en binômes.

Le travail rendu sera jugé selon les critères suivants :

- Clarté et lisibilité du code. Le code doit être le plus clair et le plus lisible possible. Utiliser la construction la plus spécifique applicable dans chaque cas.
- Respect de la culture partagée. Le code doit respecter les traditions de Lisp, y compris indentation, espacement, noms de variables, formats de commentaire et utilisation d'idiomes. Attention à ne pas utiliser l'indentation pour Emacs Lisp à la place de l'indentation pour Common Lisp. Il sera parfois nécessaire de rajouter des règles d'indentation dans Emacs, en particulier concernant des macros éventuelles réalisées.
- La taille du code. Le code doit être le plus compact possible (en nombre d'expressions) toute en respectant la culture partagée. La duplication de code doit être évitée autant que possible grâce à l'utilisation d'abstractions sous la forme d'abstractions de données (classes, structures, etc), d'abstractions de contrôle (fonctions, méthodes, etc) et d'abstractions de syntaxe (macros).
- Performance. Le code doit utiliser des algorithmes et des structures de données performants, sauf si la taille du problème peut être considérée comme faible et que l'algorithme ou la structure de donnée moins efficace est considérablement plus lisible.
- Modularité. Le code doit, autant que possible, être séparé en modules indépendants (dans des fichiers différents ; il n'est pas nécessaire d'utiliser les paquetages).
- Extensibilité. Le code doit permettre des extensions futures, de préférence sans modification du code fourni, et surtout sans qu'une application existante utilisant le code ne doive être modifiée.

Bon courage !

On souhaite simuler le comportement d'un ensemble de files d'attente (dans un magasin par exemple). Le but est de calculer le temps d'attente moyen d'un client, dans le but de modifier le nombre et la nature des files (réservée aux clients avec moins de 10 articles par exemple).

1 Probabilité, distribution, répartition

Afin de déterminer certains paramètres de simulation (comme le nombre d'articles d'un client, ou l'heure d'arrivée d'un client) on utilise la notion de *variable aléatoire*. Une telle variable est soit *discrète* soit *continue*. Une variable aléatoire discrète peut être utilisée par exemple pour indiquer le nombre d'articles dans le panier d'un client, alors qu'une variable aléatoire continue peut être utilisée pour indiquer l'heure d'arrivée d'un client.

Pour une variable aléatoire discrète, disons X, on définit une fonction notée $y = f_X(x)$ qui indique la probabilité que la valeur de la variable X prenne la valeur x. Plus formellement $f_X(x) = P(X = x)$. Dans le cas discret, la fonction $f_X(x)$ est appelée *la distribution de probabilité*.

Notons par $F_X(x)$ la fonction $\sum_{t=-\infty}^{x} f_X(t)$. Cette fonction donne la probabilité $P(X \leq x)$. Clairement $F(\infty) = 1$.

La méthode traditionnelle pour générer une valeur aléatoire selon une distribution de probabilité donnée est de générer une valeur aléatoire continue y telle que $0 < y < 1$ avec une distribution uniforme (rectangulaire), puis de déterminer la plus petite valeur de x telle que $y \leq F_X(x)$. Une suite de valeurs de x calculées de cette façon aura la distribution de probabilité correspondant à la fonction $f_X(x)$.

Pour une variable aléatoire continue, les choses se compliquent. Dans ce cas, on définit une fonction (également notée $y = f_X(x)$) qui indique la valeur de l'expression $lim_{\Delta x \to 0} P(x \leq X < x + \Delta x)/\Delta x$. On appelle cette fonction la *densité* de la variable aléatoire.

Dans le cas continu, la fonction $F_X(x)$ est définie comme ceci : $F_X(x) = \int_{-\infty}^{x} f(t)dt$. On l'appelle alors la *fonction de répartition*. De la même manière que pour le cas continu, afin de générer des valeurs selon une fonction de densité donnée, il suffit de générer une valeur aléatoire continue y telle que $0 < y < 1$ avec une densité uniforme (rectangulaire), puis de déterminer la plus petite valeur de x telle que $y \leq F_X(x)$. Une suite de valeurs de x calculées de cette façon aura la distribution de probabilité correspondant à la fonction $f_X(x)$.

La complication du cas continu provient du fait que l'intégrale de la fonction de densité peut ne pas être triviale à calculer. Dans le cas général, il faut faire une approximation numérique sous forme de somme.

Votre programme doit pouvoir traiter des variables aléatoire discrètes et continues. Les fonctions de densité et de distribution de probabilité traitées dans le cadre du devoir auront toutes la propriété suivante : $\forall x < 0, f_X(x) = 0$, mais il faut permettre une extension future pour des fonctions de densité n'ayant pas cette propriété.

Une fonction densité ou de distribution de probabilité est dite *finie* si et seulement si $\exists k$ tel que $\forall x > k, f_X(x) = 0$. Sinon elle est dite *infinie*.

Deux représentations différentes d'une distribution de probabilité (cas discret) doivent être gérées par votre programme, selon qu'elle est finie ou infinie. Dans le cas fini, on peut la représenter par une table de valeurs de $f_X(x)$ pour $0 \leq x \leq k$. Dans le cas infini, on se servira d'une fonction Common Lisp (créée par une expression `lambda` par exemple) capable de calculer $f_X(x)$. Un exemple d'une telle fonction serait $f_X(x) = 2^{-x-1}$. Un cas particulier de cette deuxième représentation serait une fonction spécifique *bien connue* dans le monde de la probabilité, peut-être avec des paramètres (exemples : gaussienne, exponentielle, χ^2, etc). Ce cas particulier ne sera pas nécessaire pour votre travail, mais vous devez expliquer comment faire pour étendre votre programme à telles fonctions.

Comme dans le cas discret, deux représentations différentes d'une fonction de densité continue doivent être gérées par votre programme. Si la distribution est finie et définie par une suite de segments de droites, il est possible de la représenter sous la forme d'une table contenant les points d'intersection des droites. Sinon, la densité sera représentée par une fonction Common Lisp comme dans le cas discret. Comme dans le cas discret, on prévoit la possibilité de travailler avec des fonctions de densité *bien connues*. Pour certaines de ces fonctions, l'*inverse de la fonction de répartition* ($x = F_X^{-1}(y)$) existe. Si c'est le cas, calculer la plus petite valeur de x telle que $y \leq F_X(x)$ revient à calculer $F_X^{-1}(y)$.

En fait, si $F_X^{-1}(y)$ n'existe pas, c'est parce que $f_X(x)$ vaut 0 dans un ou plusieurs intervalles et donc que $F_X(x)$ est constante dans ces intervalles. Dans ce cas, on peut quand même dire que l'inverse existe, car il suffit de supprimer ces intervalles de la définition de la fonction $F_X(x)$. La probabilité d'obtenir une valeur dans ces intervalles est par définition 0 de toute façon. La fonction $F_X^{-1}(y)$ aura dans ces cas des discontinuités.

2 Simulation du temps

Votre programme doit contenir un gestionnaire du temps. L'essence d'un tel gestionnaire est une *file de priorités* contenant des *événements* triés par date de déclenchement. Il doit être possible de rajouter des événements et de déclencher tous les événements (dans l'ordre) dont la date de déclenchement est inférieure à une date donnée.

Votre programme doit pouvoir traiter plusieurs types d'événements, par exemple un événement indiquant l'arrivée d'un client, ou la fin de la simulation. Il doit également permettre une extension future concernant le type des événements traités.

3 Application

Afin de montrer que votre logiciel marche, vous devez programmer une application dont l'objectif est de simuler un magasin avec trois files de caisse, l'une des trois réservée aux clients avec au plus 10 articles.

Un client est caractérisé par
– sa date d'arrivée devant les caisses,
– le nombre d'articles achetés,
– le temps de payement.
Ces trois paramètres doivent être choisis aléatoirement selon des distributions raisonnables.

Le *temps total de traitement* d'un client est une constante (paramètre) fois le nombre d'articles achetés plus le temps de payement. Un client avec au plus 10 articles qui arrive va choisir la file réservée à ce type de client si et seulement si sa longueur ne dépasse pas deux fois celle de la plus courte des deux autres. Sinon, il va choisir la plus courte des deux autres. Un client avec plus de 10 articles qui arrive va choisir la file permise la plus courte.

Vous devez :

- tester votre programme (et rendre le résultat des tests avec le programme) sur 100 clients,
- choisir les paramètres pour que la simulation soit intéressante,
- éviter par exemple une simulation qui génère presque toujours des clients avec un nombre d'articles supérieur à 10, ainsi que des temps de traitement trop brefs par rapport à la fréquence d'arrivée des clients,
- fournir non seulement le résultat de la simulation, mais aussi une trace contenant, sur chaque ligne, le déclenchement d'un événement.

4 Documentation

Fournir *au plus une page* de texte expliquant ce que vous avez fait. Ne pas expliquer le code, mais les choix considérés mais non retenus (et pourquoi) concernant les représentations de données, les algorithmes choisis et l'organisation du programme. Expliquer également comment étendre votre programme selon les exigences indiquées ci-dessus.