UTBM - LO44

Projet d'algorithmique et programmation

M. El Zaher - C. Cappelle

A2012

1 Objectif du projet

Le projet a pour objectif la définition et la manipulation de types abstraits de données Individu et Population. Un Individu est représenté par une suite de bits et une Population est une suite d'Individus. Il s'agit d'une version simplifiée d'algorithme génétique.

2 Réalisation et évaluation du projet

- Groupe de 1 à 2 personnes maximum.
- Programmation à réaliser en langage C.
- Le projet fera l'objet d'une présentation orale qui se déroulera pendant la séance de TP du vendredi 11 janvier 2013 entre 14h et 16h.
- Le projet devra être documenté au sein d'un rapport écrit qui expose brièvement (15 à 20 pages) les structures de données et algorithmes utilisés ainsi que les optimisations éventuellement introduites, les jeux d'essais, les commentaires sur les résultats et les éventuelles difficultés rencontrées. Le rapport ne devra pas contenir de listing du programme C. Les choix techniques et algorithmiques devront être justifiés.
- Les codes sources fournis devront être commentés et comporter le nom des différents auteurs du projet ainsi qu'une description textuelle précisant son objet.
- Un Makefile devra être créé pour la compilation du code, ainsi que pour le lancement de l'exécutable. Attention, la compilation doit se faire en lançant simplement la commande *make*. Quant au lancement de votre exécutable, elle doit être effectuée par la commande *make launch*.
- Le rapport au format **PDF**, le code source des programmes ainsi que le Makefile sont à fournir au plus tard le lundi 7 janvier 2013 à 18h par mail adressé à `cindy.cappelle@utbm.fr` et `madeleine.el-zaher@utbm.fr` (Objet du mail : [Projet LO44] - NomEtudiant1EnMajuscule et NomEtudiant2EnMajuscule) dans une archive au format **ZIP** ou **TAR.GZ** nommée de la façon suivante :
  [LO44]_NomEtudiant1EnMajuscule_NomEtudiant2EnMajuscule.zip

Tout retard ou non respect de ces directives sera fortement pénalisé.

Les points suivants devront faire l'objet d'une attention particulière lors de l'écriture du programme : le bon fonctionnement du programme sans bug (il vaut mieux ne pas fournir une fonctionnalité plutôt que de la fournir si elle ne fonctionne pas), la lisibilité et la clarté du code (commentaire et indentation), la complexité et l'efficience des algorithmes proposés ainsi que le choix des structures de données, la modularité du code (développement de petites fonctions, répartition du source en plusieurs fichiers regroupant des fonctions de façon logique et nommées de manière adéquate).

3 Travail à réaliser

3.1 Type abstrait « Individu »

Définir un type abstrait de données appelé Individu, représenté par une liste de bits, de longueur donnée (*longIndiv*). Écrire l'algorithme et le sous-programme C correspondant aux opérations suivantes :

- Initialiser aléatoirement la liste de bits (donner une version itérative et une version récursive de cette opération).
- Décoder la liste de bits et donner la valeur entière correspondante.
- Croiser deux listes de bits, c'est-à-dire intervertir les éléments des deux listes selon une probabilité donnée (*pCroise*) pour chaque position dans la liste (tirage aléatoire et comparaison avec la probabilité).
- Calculer à partir de la valeur d'un individu sa qualité, en utilisant la fonction réelle **f1** (1) donnée ci-dessous :

$$f_1(x) = -X^2 \quad avec : \quad \begin{cases} X = \frac{x}{2^{longIndiv}}(B - A) + A \\ A = -1 \\ B = 1 \\ longIndiv = 8 \end{cases} \tag{1}$$

Au niveau de l'implémentation en C on utilisera pour représenter un bit la définition de type suivante : « **typedef unsigned char Bit ;** ». Ainsi, une variable de type **Bit** prendra les valeurs **0** ou **1**. De plus, une suite de bits sera représentée sous forme de liste chaînée. Pour la génération de nombres aléatoires, on utilisera les fonctions **rand**, **srand** et **time** des bibliothèques **stdlib** et **time**.

3.2 Type abstrait « Population »

Définir un type abstrait de données appelé Population, constitué d'une liste d'Individus. Écrire l'algorithme et le sous-programme C correspondant aux opérations suivantes :

- Initialiser aléatoirement la liste d'Individus. Le nombre d'Individus dans la population est donné par le paramètre *taillePop*.
- Trier la liste par Qualité décroissante des Individus au moyen de *Quicksort* (voir ci-dessous). On utilisera pour cela la fonction Qualité définie sur le type abstrait Individu.
- Sélectionner les meilleurs Individus de la Population en tronquant la liste et en la complétant par recopie des *tSelect* premiers éléments.

Par exemple :

tselect = 4							
Qualité des Individus	7	6	5	4	3	2	1
Liste après sélection	7	6	5	4	7	6	5

– Croiser la Population, c'est-à-dire à partir d'une Population P1, créer une seconde Population P2, constituée d'Individus sélectionnés aléatoirement deux à deux dans P1 et croisés entre eux.
– Au niveau de l'implémentation en C, une suite d'Individus sera représentée sous forme de liste chaînée.

3.3 Le tri rapide (*Quicksort*)

Le tri rapide, ou *Quicksort*, est fondé sur le principe « diviser pour mieux régner ». Il est donc composé des deux étapes suivantes :

– Diviser : la liste des éléments à trier est partitionnée en deux sous listes S1 et S2, telles que chaque élément de S1 soit inférieur à tout élément de S2. On procède en inversant lorsque c'est nécessaire les extrémités de la liste.
– Régner : les deux sous-listes sont triées par appels récursifs à *Quicksort*.

Les deux sous-listes étant individuellement triées et les éléments de la première inférieure à ceux de la seconde, la liste est donc globalement triée.

3.4 Programme

En utilisant les précédentes définitions, vous réaliserez le programme correspondant à l'algorithme décrit ci-dessous :

Initialiser la Population
Répéter nGen fois
Début
    Croiser la Population
    Trier la Population
    Sélectionner la Population
Fin
Afficher le meilleur Individu de la Population

Les paramètres à utiliser sont les suivants :

Longueur d'un individu :	$longIndiv = 8$
Probabilité de croisement :	$pCroise = 0.5$
Taille de la Population :	$20 \leq taillePop \leq 200$
Taux de sélection :	$10\% \leq tSelect \leq 90\%$
Nombre de génération :	$20 \leq nGen \leq 200$

3.5 Manipulation

Un fois les types de données et leurs opérations définis et le programme réalisé, tester celui-ci en remplaçant la fonction Qualité **f1** par les fonctions suivantes :

$$f_2(x) = -ln(X) \quad avec : \quad \begin{cases} X = \frac{x}{2^{longIndiv}}(B - A) + A \\ A = 0.1 \\ B = 5 \\ longIndiv = 16 \end{cases} \tag{2}$$

$$f_3(x) = -cos(X) \quad avec : \quad \begin{cases} X = \frac{x}{2^{longIndiv}}(B - A) + A \\ A = -\pi \\ B = \pi \\ longIndiv = 32 \end{cases} \tag{3}$$

On précise que l'on réalise préalablement au calcul de la fonction en lui-même, une mise à l'échelle, au moyen de X. Constater et commenter (rapidement) les résultats obtenus. Observer (rapidement) l'influence des paramètres.