Programmation Efficace

François Laroussinie
francoisl@irif.fr
&
Arnaud Sangnier
sangnier@irif.fr

INTRODUCTION

Quelques informations

- Un cours/tp par semaine
- Programmation en groupe de 5 étudiants
- Page web du cours :

  https://www.irif.fr/~sangnier/enseignement/prog-eff.html
- Évaluation : (peut changer)
  - 1ère session : 50 % Controle Continu + 50% Examen Final
    - Controle Continu :
      - Participation aux exercices
      - Évaluation des exercices rendus
      - La présence au cours est prise en compte dans cette note
      - La clarté du rendu sur git aussi
    - Examen :
      - Concours de programmation en groupe à faire en 3h
  - 2ème session : 50 % Controle Continu + 50% Examen

Organisation

- Les exercices sont à faire en groupe
- A priori les groupes ne changent pas
- Nombre de personnes par groupes : 5
- Utilisation de **Git** pour soumettre les solutions et pour travailler en commun
- Avant la fin de la première semain, il faut nous envoyer (**sangnier@irif.fr** et **françoisl@irif.fr**) :
  - La composition de votre groupe
  - Je vous enverrai un numéro de groupe et vous devrez créer sur **Git** **u**n nouveau projet peff23-k où k sera le numéro de votre groupe
- **Langages autorisés : Java, Python et C**

À propos de Git

- **Git** : logiciel de versions
- Permet de travailler en commun et en parallèle
- Commandes qui vont vous être utiles
  - **git clone git@gaufre.informatique.univ-paris-diderot.fr:sangnier/peff21-1.git** -> pour récupérer le répertoire du groupe 1 pour la première partie du semestre
  - **git add file.txt** -> pour ajouter un fichier au git
  - **git commit -a -m "Message"** -> pour sauvegarder vos modifications
  - **git push** -> pour envoyer vos modifications sur le serveur distant
  - **git pull** -> pour récupérer la version du serveur distant
  - **git tag etiquette** -> pour créer une étiquette correspondant à votre version
  - **git checkout etiquette** -> pour récupérer la version correspondant à une étiquette
  - **git checkout master** -> pour revenir la branche principale (quand on a vu une étiquette)
  - **git push origin --tags** -> pour envoyer votre version avec les étiquettes
- **Même si Git permet de les gérer, faîtes attention aux conflits qui peuvent faire perdre du temps**

Objectifs du cours

- Le but de ce cours est d'apprendre à **fournir rapidement un programme efficace répondant à un problème donné**
- Autre but : S'entraîner aux concours de programmation du style Hashcode ou Swerc
- Les programmes ne sont pas forcément les mieux en ce qui concerne :
  - L'organisation du code
  - La modularité
  - La réutilisation
  - L'optimisation du code

  Ils ne respectent pas forcément les critères attendus en Génie Logiciel
- Mais ils doivent satisfaire les contraintes suivantes :
  - Terminer sans erreur
  - Donner des solutions correctes
  - Efficacité de l'algorithme (les jeux de données peuvent être grands)

  Un programme ne respectant pas les deux premiers points est éliminatoire

Prérequis, acquis et méthodologie

- **Prérequis**
  - Savoir programmer (sans erreur) dans un des trois langages
  - Connaître la manipulation des entrées/sorties (lecture/ écriture sur les entrées/sorties standard et dans des fichiers)
  - Connaître quelques bibliothèques de structures de données (file, pile, table de hachage) ou savoir les trouver
  - Être capable de programmer un algorithme
- **Acquis**
  - Méthodologie pour approcher des problèmes de programmation
  - Pratique de la programmation rapide en équipe
  - Développement de nouveaux algorithmes
  - Rappels algorithmiques
- **Méthodologie**
  - Être présent et participer aux cours/tp
  - Finir chez soi les exercices donnés
  - S'entraîner chez soi en groupe ou tout seul

Ressources

- Le cours d'Algorithmique du premier semestre
- Le livre ***Programmation Efficace*** de Chirstoph Dürr et Jilll-Jênn Vie
- Les sites des concours de programmation (regarder les années passées)
  - Google Hashcode : **https://codingcompetitions.withgoogle.com/**
  - ACM ICPC Swerc : **https://swerc.eu/**
- Des sites d'entraînement à la programmation:
  - Sphere Online Judge : **https://www.spoj.com/**
  - Code Forces : **https://codeforces.com/**
  - Et d'autres (n'hésitez pas à chercher par vous même et à nous les signaler)
  - **C'est une bonne idée de participer au HashCode 2023 (il y a un hub virtuel à Université de Paris) (13 Avril 2023)**

Fonctionnement du cours/TP

- Différents types de séances:
  - Cours (rappels de notions algortithmiques + nouvelles notions) + Exercices de programmation liés
  - Uniquement programmation avec un ou des problèmes à résoudre (soit dans le temps imparti, soit en les finissant chez vous)
- Certaines séances spéciales:
  - Séance type google Hashcode
  - Séance sur la présentation de solutions par les étudiants et discussions
  - Dans la deuxième partie du semestre, il y aura une séance de type Swerc et présentation de solutions

Quels types de problèmes

- Souvent on va distinguer deux types de problèmes :

  1) Les problèmes qui attendent une unique solution

  - Pour ces problèmes le but est de développer un algorithme exact qui répond à la question
  - La difficulté réside souvent dans l'efficacité du programme
  - Les instances peuvent être grandes et il faut souvent être malins dans les algorithmes pour résoudre en pratique le problème
  - Le concours **SWERC** est basé sur ce genre de problèmes

  2) Les problèmes qui admettent une solution optimale qu'il est difficile d'obtenir en pratique dans le temps imparti et il faut donner les meilleures solutions approchées

  - Le but ici est déjà de réussir à produire une solution
  - Ensuite de pouvoir améliorer une solution fournie
  - Ici on ne veut pas l'algorithme qui donnerait la solution optimale, car celui-ci serait trop couteux en temps
  - Il s'agit de développer des heuristiques pour s'approcher le mieux possible de la solution optimale
  - Selon les cas, différentes heuristiques peuvent être données
  - Le concours **Hashcode** est basé sur ce genre de problèmes

Exemple I

- Pour les épreuves de type **SWERC**
  - On donne un problème précis :
    - Un fichier de n lignes avec 0 < n < 1000000
    - Chaque ligne contient un nombre flottant e avec deux chiffres après la virgule tel que 0.00 < e < 1000000.00
    - Il faut afficher la somme sous la forme d'un nombre flottant avec deux chiffres après la virgule
  - **On soumet le programme** en C++, Java ou Python
  - Un juge automatique teste le programme sur différentes instances (non connues des participants)
  - Le juge renvoie une réponse du type CORRECT, TIMEOUT, COMPILATIN ERROR, RUN ERROR, WRONG ANSWER,...
  - Plus on soumet de programmes, plus on est pénalisé
  - On doit résoudre plusieurs problèmes de ce type pendant un temps donné

Exemple II

- Pour les épreuves de type **Google HashCode**
  - On donne un problème admettant plusieurs solutions il faut trouver la meilleure possible :
    - On dispose de N sac à dos, chaque sac à doc à une capacité limite C en kg et peut contenir des objest de type A, B et C
    - Pour chaque sac à dos, on a des contraintes, par exemple le sac 1 ne doit pas contenir plus d'objets de type A que de type B, etc
    - On a ensuite une liste d'objets avec leurs poids en kg
    - Le but est de remplir le plus de sac à dos possible
  - On programme avec le langage que l'on veut
  - On ne soumet pas le programme au juge pendant la compétition (mais à la fin)
  - **On dispose de différentes instances du programme et pour chacune on soumet un fichier contenant une solution possible**
  - Un juge automatique teste la solution et donne des points (plus la solution est bonne plus on a de points)
  - Il ne faut résoudre qu'un seul problème, mais selon les entrées, la solution qui marche le mieux peut changer
  - Un algorithme qui donnerait la meilleure solution dans tous les cas est ou **trop long à écrire dans le temps imparti** ou **trop long à s'exécuter**

Entrée/Sortie

- Pour les épreuves de type **SWERC**
  - On lit sur l'entrée standard une suite de lignes
  - On écrit sur la sortie standard les réponses
- Pour les épreuves de type **Google HashCode**
  - On lit dans un fichier
  - On écrit dans un fichier
- Il faut éviter de perdre trop de temps à écrire du code pour lire l'entrée
- **Pensez à avoir des morceaux de code déjà prêts à réutiliser pour:**
  - Lire les lignes d'un fichier
  - Récupérer les différents éléments d'une ligne séparé par des espaces, des virgules, des points virgules,etc
  - Traduire des chaînes de caractères récupérées en d'autres types (souvent il s'agira d'entiers)
  - Note: Pour le Swerc, vous devrez les reprogrammer rapidement

Conseils en vrac I

- Lisez calmement l'énoncé jusqu'à ce qu'il n'y ait plus d'ambiguïté possible
- Réfléchissez avant de programmer et définissez proprement votre algorithme
  - Il faut être capable d'expliquer l'algorithme à vos co-équipiers
- Communiquez avec votre équipe et séparez vous les tâches
- Maîtriser certains algorithmes 'classiques'
- Maitrisez vos entrée/sortie pour éviter les bugs
- Limitez l'utilisation de raccourcis syntaxiques qui dans la précipitation amènent à des bugs (du style écrire i=i++ en Java)
- Initialisez vos variables
- Faites attention quand vous copiez des structures de données
  - On a parfois besoin de copier une liste, assurez-vous que vous ne faites pas que des copies d'adresse mais que vous dupliquer bien vos données
- Pour le **Hashcode**, programmer un testeur de solutins
- Pour le **Swerc**, pensez à faire des exemples en plus de l'unique fourni

Conseils en vrac II

- Pour le **Hashcode:**
  - programmer un testeur de solutions
  - connaître certaines bibliothèques d'un langage
  - commencer par programmer un programme simple chercheur de solutions (même si il est trop basique, vous pouvez le laisser tourner pendant que vous chercher un meilleur algorithme)
- Pour le **Swerc:**
  - pensez à faire des exemples en plus de l'unique fourni
  - parcourir les différents problèmes pour voir si certains sont plus simples que d'autres
  - faîtes attention au codage que vous utilisez (parfois la clef du problème est la taille des données)