Enseignement en Programmation Sportive
EPS

TD1 - Introduction, The Online Judge, Gestion I/O

Joël Felderhoff, Quoc Tung Le, Lucas Perotin, Eric Thierry

Travaux pratiques d'algorithmique

Objectif : implémenter des algorithmes classiques sur des exercices originaux issus de concours de programmation

Prérequis : algorithmique de base de notre L3 (ALGO1 déjà vu, synchro avec ALGO2) et un peu de programmation (PROG)

La théorie de la complexité en pratique : trouver de bons compromis temps d'exécution / temps de développement

En prime : un entrainement à la *programmation sportive* (*competitive programming* en VO)

Tout le spectre de ALGO en L3

Thèmes :

- Grands paradigmes : glouton, diviser-pour-règner, programmation dynamique, ...
- Algo des ensembles (structures de données)
- Algo des nombres et d'algèbre
- Algo des mots
- Algo des graphes
- Algo géométrique

Mise en pratique :

- Modéliser les problèmes : énoncés avec des histoires qui ne donnent aucune indication sur la méthode à utiliser
- Concevoir des algorithmes : trouver des résolutions efficaces
- Programmer ces algorithmes : écriture & débugage efficaces

Compétitions de programmation

Challenge : résoudre en temps limité une série de plusieurs problèmes, ou bien un unique et énorme problème

Programmer «efficacement» : solution juste et rapide, temps de développement compris

Variantes :

- Durée du concours (quelques minutes à plusieurs semaines)
- OnLine ou OnSite
- En solo ou en équipe
- Limites sur le temps d'exécution ou pas
- Données en entrée cachées ou pas

Des compétitions internationales annuelles

International Olympiad in Informatics (IOI)		individuel, ≤ 19 ans pas bachelier
Prologin		individuel, ≤ 20 francophone
Google Code Jam	Bientôt en mars	individuel, ≥ 18 ans pas chez Google
Google Hash Code	Urgent en février	équipe de 2 à 4 pas chez Google
Facebook Hacker Cup		individuel, ≥ 18 ans pas chez Facebook
International Collegiate Programming Contest	ACM-ICPC SWERC	BAC+5 et/ou ≤ 23 ans sur l'année

Des sites d'entrainement toujours ouverts

Codeforces	concours très réguliers	individuel
TopCoder	pleins de formats de concours	individuel
ProjectEuler	problèmes très mathématiques	individuel
Online Judge	entrainement au concours ACM-ICPC	individuel
Codin'Game	pleins de formats de concours souvent duels d'IA	individuel francophone
Hackathons Online	concours en temps limité souvent longs ($\geq$ 24h)	individuel ou par équipe

Exemple : 13 problèmes, 5h, équipes de 3

ACM SWERC 2014

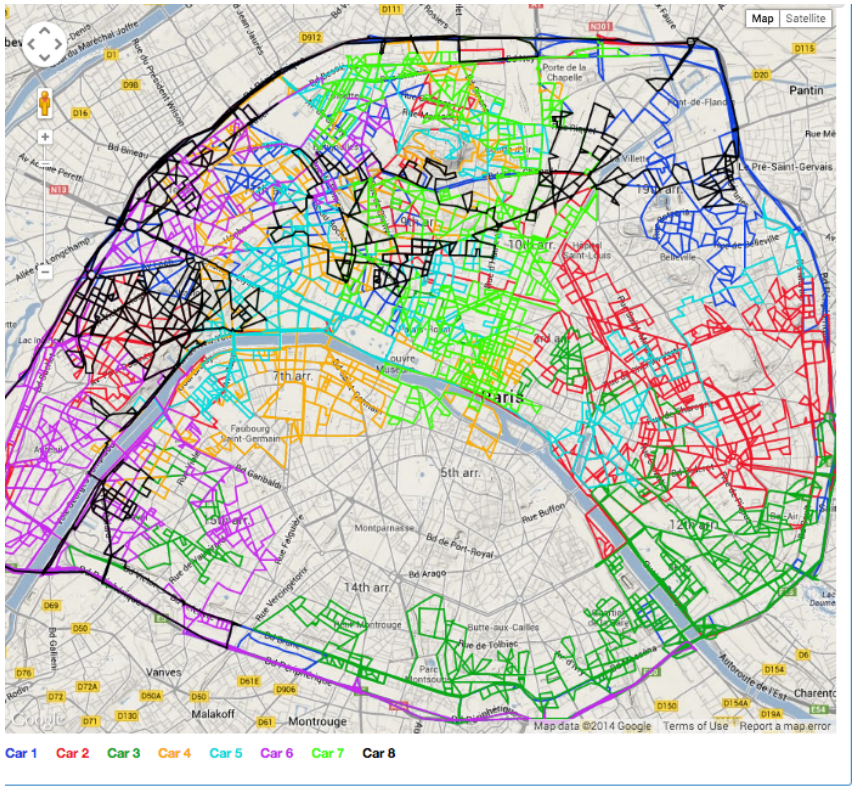
Problème A : dénombrer les solutions à l'égalité GREAT+SWERC=PORTO (remplacer les lettres par des chiffres)

Indication : brute force!

Exemple : 1 problème, 4h, équipes de 2 à 4

Google Hash Code 2014

Itinéraires de 8 voitures StreetView dans Paris pour couvrir le plus de rues possibles en 15 heures



Indication : algorithme du Postier Chinois sur le graphe des rues de Paris

Les vertus du sport

- Se perfectionner en programmation / algorithmique sur des problèmes amusants
- Langages variés suivants les concours : C, C++, Java, Python, Caml, Haskell, Ruby, ...
- Une porte d'entrée pour postuler dans certaines entreprises / universités (p.ex. sponsors de concours)
- Ce que ne sont pas les concours :
  • de la recherche pure
  • des projets de développement à grande échelle (encore qu'il existe des Hackathons ambitieux)

Organisation du cours

Site web du cours : https ://perso.ens-lyon.fr/eric.thierry/EPS

Notre principal site de musculation : Online Judge (ex-UVa)

- Online Judge : le serveur-juge de Valladolid
  https ://onlinejudge.org
- uHunt : vos stats et préférences
  https ://uhunt.onlinejudge.org
- uDebug : des exemples d'entrées/sorties pour tester
  https ://udebug.com

Evaluations : rendu des TD complétés à la maison (12 pts), un partiel (4 pts) et un examen (4 pts) dans des conditions «concours»

Exos Bonus : optionnels, mini bonus si faits (max +2 pts), un critère de sélection si bcp d'étudiants veulent participer au SWERC

Choix du langage de programmation

«Libre» mais contraint par les serveurs juges :

- Online Judge : C, C++, Java, Python, Pascal
- ACM SWERC : C, C++, Java, Kotlin, Python, OCaml
- ACM ICPC : C, C++, Java, Kotlin, Python

Choix privilégié pour ce cours : C++

- quelques rappels/infos de syntaxe, un suivi à la demande
- pas un cours de prog OO, programmes courts
- profiter de la STL (Standard Template Library)

Des références pour C++ :

- documentation en ligne : cplusplus.com
- le livre du créateur : le Soustrup

Java, Python : parfois des avantages (p.ex bibliothèques)

⚠ le Judge est difficile avec Python sur les temps d'exécution

Choix de l'environnement de programmation

Environnement de développement «libre» :

- Combo éditeur préféré + commandes à la console
- IDE favori pour le langage choisi : Visual Studio Code, Visual Studio, Code : :Blocks, Eclipse, PyCharm, Spyder ...

Conseil :

- gardez votre environnement préféré, en particulier si adopté depuis longtemps ou pendant PROG
- faites-nous signe si vous hésitez

Remarques :

- vos choix (langage + environnement) peuvent influencer les outils de débugage/profilage disponibles
- si nous traitons ces sujets, nous ferons notre possible pour que tout le monde puisse expérimenter

Notre évaluation de vos solutions

Algo de notation d'un exercice (sur 5 pts) :

- **Exercice validé par le Judge :** 5
- **Exercice non validé par le Judge :** additionner
  - Modélisation (parfois plusieurs valides) : +1 si correcte (nécessite un code ou une idée complète envoyés), 0 sinon
  - Algo (correction & complexité) : +2 si correct et rapide, +1 si oubli ou échec sur un cas marginal ou si complexité un ordre de grandeur sous-optimale, 0 si oubli ou échec sur un cas majeur ou si algo complétement faux ou si complexité très mauvaise
  - Implémentation (sur les cas où algo correct) : +2 si correcte et performante, +1 si erreur mineure (erreur = bug syntaxe, entrée ou sortie erronée) ou performances inégales (p.ex. faibles sur grosses entrées), 0 sinon
- **Malus :** -1 si code sale, -1 si deadline non respecté

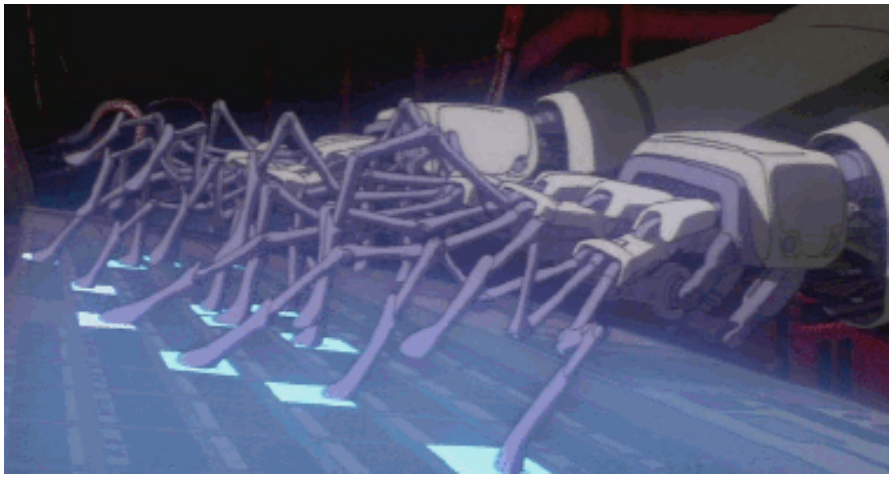
Aujourd'hui

- Inscription sur le site Online Judge puis sur le notre (respectez bien cet ordre !)
- Premiers exercices et exemples
- Familiarisation avec Online Judge, uHunt, uDebug
- Gestion C++ des entrées/sorties avec la STL
- Utilisation de structures de données de la STL

Interagissez avec nous pendant la séance
Ne restez pas bloqués
Posez des questions

Plus pointu ...

- Eplucher les codes sources des antisèches/booklets autorisées pendant les compétitions ACM
- S'entrainer en équipe → savoir communiquer et se répartir les tâches
- Tester sa vitesse de frappe (typing speed test)



Briefing : comment fonctionne Online Judge ?

Fonctionnement du Online Judge (OJ) :

- Serveur Linux avec compilateurs/interpréteurs idoines et base de données des exercices
- C++ : `prog.cpp` compilé en exécutable `prog.out` puis `prog.out < entrée_secrète > sortie_vérifiée`
- Java : `prog.java` compilé en interprétable `prog.class` puis `java prog < entrée_secrète > sortie_vérifiée`
- Python : exécution du code `prog.py` avec `python prog.py < entrée_secrète > sortie_vérifiée`
- Temps d'exécution : faire précéder par la commande `time`

Flux de redirection : les chevron < et > servent respectivement à redéfinir entrée/sortie standards, càd de lecture/écriture, du programme (typiquement des fichiers).

Briefing : conséquences du fonctionnement de OJ

Entrée OJ secrète : nom/lieu/contenu du fichier sont secrets

- Empêche d'utiliser les fonctions I/O pour gérer les fichiers
- Empêche d'analyser l'entrée pour optimiser votre code
- Complique le débugage

Entrée/sortie via flux de données (stream) : gestion I/O avec fonctions capables de traiter des flux en entrée/sortie

Cas général : entrée secrète = plusieurs instances (testcases) énumérées selon différents schémas

- première ligne = nombre de testcases
- nombre de testcases non fourni, savoir repérer EOF
- nombre de testcases non fourni, savoir repérer un code de fin

Briefing : conséquences du fonctionnement de OJ

Entrée OJ secrète : nom/lieu/contenu du fichier sont secrets

- Empêche d'utiliser les fonctions I/O pour gérer les fichiers
- Empêche d'analyser l'entrée pour optimiser votre code
- Complique le débugage → uDebug offre des entrées

Entrée/sortie via flux de données (stream) : gestion I/O avec fonctions capables de traiter des flux en entrée/sortie

Cas général : entrée secrète = plusieurs instances (testcases) énumérées selon différents schémas

- première ligne = nombre de testcases
- nombre de testcases non fourni, savoir repérer EOF
- nombre de testcases non fourni, savoir repérer un code de fin

Briefing : import depuis des bibliothèques

Langage C++ :

- `#include<iostream>`
- `#include<string>`

Langage Python :

- `import sys`
- `from math import sqrt`

Langage Java :

- `import java.util.Scanner`
- `import java.math.BigInteger`

Bibliothèques autorisées dans ce cours : comme Online Judge

- Exemple en C++ : STL ok mais pas BOOST
- Exemple en Python : pas `numpy` ...

⚠ Règles du jeu selon les concours de programmation

Briefing : structure des codes

Langage C++ : court, en général ≤ 100 lignes

```
// Imports
...
using namespace std;
...
// Fonctions
...
// Main
int main()
{
    ...
    return 0; // valeur attendue par Online Judge
// facultatif car 0 renvoyé par défaut si succès
}
```

Briefing : structure des codes

Langage Python : court, en général ≤ 100 lignes

```
# Imports
...
# Fonctions
...
# Main
...
```

Remarque : pas obligatoire d'utiliser la syntaxe

```
if __init__ == "__main__" :
```

Briefing : gestion I/O sur OJ en C++

Flux d'entrée :

- utiliser l'entrée standard `cin` avec `#include<iostream>` qui avale le flux de différentes manières
- `cin»a` remplit la variable `a` en sautant les blancs avant, utilisable en série `cin»a»b»c` avec différentes options de découpage
- `getline(cin,s)` lit et stocke une ligne dans un string `s`
- utiliser des string streams avec `# include<sstream>` pour certains découpages

⚠ pleins de combo mais des surprises, cf rubrique cours du site

Flux de sortie :

- utiliser la sortie standard `cout` avec `# include<iostream>`, très souple d'utilisation et de syntaxe similaire
- `cout«a` pour envoyer la valeur de `a` en sortie
- différentes options de formatage, cf rubrique cours du site

Briefing : gestion I/O sur OJ en Python

Flux d'entrée : plusieurs options

- utiliser `input()` qui avale le flux jusqu'au newline suivant (`\n`) et le renvoie sous forme d'un string (en éliminant `\n`)
- utiliser l'entrée `sys.stdin` comme un `file` avec `import sys`
  - `for line in sys.stdin:` pour du ligne par ligne (`\n` compris)
  - `sys.stdin.readline()` pour récupérer une ligne (`\n` compris) ou juste un nombre fixé de caractères
  - `sys.stdin.read()` pour récupérer l'entrée en intégralité

Flux de sortie : plusieurs options

- utiliser `print()` l'affichage des solutions
- utiliser `sys.stdout` comme un `file` avec `sys.stdout.write()` pour l'affichage
- deux découpages possibles : afficher les solutions au fur et à mesure, ou construire un grand string mettant en page toutes les solutions et affiché en une fois à la fin

Briefing : gestion I/O sur OJ en C++

Schéma où nombre de testcases en première ligne :

```
#include<iostream>
cin » n
for (i = 0; i < n; i++)
{
    cin » data
    process_from(data)
}
```

Schéma où nombre de testcases inconnu, fin en EOF :

```
#include<iostream>
while (cin » data)
{
    process_from(data)
}
```

Briefing : gestion I/O sur OJ en Python

Flux d'entrée : plusieurs options

- utiliser `input()` qui avale le flux jusqu'au newline suivant (`\n`) et le renvoie sous forme d'un string (en éliminant `\n`)
- utiliser l'entrée `sys.stdin` comme un `file` avec `import sys`
  - `for line in sys.stdin:` pour du ligne par ligne (`\n` compris)
  - `sys.stdin.readline()` pour récupérer une ligne (`\n` compris) ou juste un nombre fixé de caractères
  - `sys.stdin.read()` pour récupérer l'entrée en intégralité

Flux de sortie : plusieurs options

- utiliser `print()` l'affichage des solutions
- utiliser `sys.stdout` comme un `file` avec `sys.stdout.write()` pour l'affichage
- deux découpages possibles : afficher les solutions au fur et à mesure, ou construire un grand string mettant en page toutes les solutions et affiché en une fois à la fin

Briefing : gestion I/O sur OJ en Python

Schéma où nombre de testcases en première ligne : sans `sys`

```
n = int(input())
for i in range(n):
    data = input()
    process_from(data)
```

Schéma où nombre de testcases en première ligne : avec `sys`

```
import sys
n = int(sys.stdin.readline())
for i in range(n):
    data = sys.stdin.readline()
    process_from(data)
```

Briefing : gestion I/O sur OJ en Python

Schéma où nombre de testcases inconnu, fin par EOF : sans `sys`

```
while True:
    try:
        data = input()
    except EOFError:
        break
    process_from(data)
```

Schéma où nombre de testcases inconnu, fin par EOF : avec `sys`

```
import sys
for line in sys.stdin:
    process_from(line[:-1])
```

⚠ avec `sys`, boucle `for` ne marche pas toujours quand le nombre de lignes des testcases est variable, nécessaire alors d'utiliser une boucle `while True:` et le repérage EOF par gestion d'exception.

Attention au format de sortie

Erreurs de format :

- bien lire la description attendue pour la sortie
- Online Judge parfois très sensible au moindre écart
- erreurs typiques et «invisibles» : des caractères blancs en trop ou en moins, un retour à la ligne de trop ...

Partagez vos (bonnes ou mauvaises) expériences sur le serveur Discord ou le pad du cours :
https ://pads.aliens-lyon.fr/p/EPS