ENSTA
2024-2025

Résolution de problème algorithmique

Examen final - 1 avril 2025

PRÉPARATION

Consignes générales (À LIRE IMPÉRATIVEMENT)

— **Lisez attentivement les consignes et tout le sujet avant de commencer.**

— Les documents (polys, transparents, TDs, livres . . .) sont autorisés.
— Sont **absolument interdits** : le WEB, le courrier électronique, les messageries diverses et variées, le répertoire des camarades, le téléphone (même pour avoir l'heure puisque vous l'avez sur votre ordinateur), les IA génératives (e.g., Mistral AI, ChatGPT, Copilot, etc.).

— Votre travail sera évalué par un mécanisme automatique. Vous **devez** respecter les règles de **nommage** et autres **consignes** qui vous sont données, en particulier, le format des sorties de vos programmes.

— La connaissance de C est un pré-requis du cours CSC_3IN03_TA. Ainsi, la présence d'erreur(s) à la compilation induira une note de zéro à l'exercice considéré, par le mécanisme de correction automatique.

— Lorsqu'il vous est demandé que votre programme réponde en affichant « **Yes** » ou « **No** », il ne doit **rien** afficher d'autre, et **pas** « Oui » ou « Yes. » ou « no » ou « La réponse est : no ». Donc, pensez à retirer vos affichages de test / debug.

— La totalité des points d'un exercice est donnée si le programme compile sans erreur, son exécution termine sans erreur et le résultat produit pour chaque jeu de tests est conforme aux spécifications. C'est donc une notation binaire : cela fonctionne ou cela ne fonctionne pas, il n'y aura pas de correction manuelle des programmes.

— **IMPORTANT** : Le site de correction automatique ne doit pas être utilisé pour mettre au point vos programmes ! Pour le développement de vos programmes, vous devez utiliser votre machine et le mode BYOD sur lequel la bibliothèque `libin103` est installée. Une fois que vos programmes compilent et s'exécutent correctement sur les jeux de données fournis alors vous pouvez soumettre votre réponse pour évaluation. La correction automatique testera vos programmes sur un ensemble de jeux de données augmenté (au moins un de plus que ceux qui vous ont été fournis).

— Le sujet comporte 10 pages et l'examen dure **2h30**.

Version de la bibliothèque `libin103`

La version $\mathbf{1.4.2}$ est la dernière version en date de la bibliothèque `libin103`. Sauf si vous n'avez pas déjà installée cette bilbiothèqe, voici la procédure :

1. À la racine de votre compte, créez un répertoire nommé `Library` **s'il n'a pas déjà été créé**, puis placez vous dans ce répertoire.

```
mkdir ~/Library; cd ~/Library
```

2. Téléchargez l'archive `libin103-1.4.2.tar.gz` sur le site du cours

   wget
   `https://perso.ensta-paris.fr/~chapoutot/teaching/in103/practical-work/libin103-1.4.2.tar.gz`

3. Désarchivez l'archive

```
tar -xvzf libin103-1.4.2.tar.gz
```

4. Allez dans le répertoire `libin103-1.4.2` et compilez la bibliothèque.
   — Il faut utiliser la commande `make`. À la fin de la compilation vérifiez la présence du fichier `libin103.a` dans le répertoire `source`.

Pour rappel, la documentation de la bibliothèque est accessible sur le site Web du cours

`https://perso.ensta-paris.fr/~chapoutot/teaching/in103/refman/index.html`

L'onglet `fichier` regroupe la documentation de tous les fichiers `.h`.

Matériel pour l'examen

Récupérez l'archive associée à cette séance d'examen à l'adresse :

`https://perso.ensta-paris.fr/~chapoutot/teaching/in103/practical-work/in103-examen-final-2425-material.tar.gz`

Comme pour les TP, un répertoire est associé à chaque exercice dans lequel se trouve un fichier `Makefile` et un ou plusieurs fichiers code source. Ce fichier `Makefile` a été configuré pour utiliser la dernière version de la bibliothèque `libin103` (version 1.4.2).

De plus, dans chaque répertoire des exercices se trouve également un répertoire `tests` qui contient des jeux de données d'entrée et de sorties attendues. Une façon simple de lancer les tests est d'utiliser la directive `test` du fichier `Makefile` ou, plus précisément, en lançant la commande

```
$ make test
```

Cette commande compile le programme et exécute les tests en les comparant la sortie du programme avec la sortie attendue. De manière plus détaillée, la commande exécutée est, par exemple,

```
$ ./mon_programme.x < tests/test1.in | diff - tests/test1.out
```

`mon_programme.x` est le programme qui vous auriez écrit, `tests/test1.in` est un jeu de données d'entrée lu sur l'entrée standard du programme, et `tests/test1.out` est la sortie attendue pour une exécution correcte du programme pour la donnée d'entrée. La commande `diff` permet de comparer deux flux d'informations (venant de l'entrée standard ou d'un fichier). Si aucune différence entre les deux flux n'est détectée, la sortie de la commande `diff` ne produit aucun affichage.

Cependant, pour clarifier le message, le lancement de la commande `make test` a été associé à un affichage joli qui permet d'afficher `OK` quand le jeu de test est conforme aux attentes et `KO` quand la sortie produite par le programme n'est pas conforme aux attendus. Par exemple,

```
exo1 $ make test
gcc -Wall -Werror -I$(HOME)/Library/libin103-1.4.2/include gain2.c -o gain2.x \
                              -L$(HOME)/Library/libin103-1.4.2/source -lin103
Test 1: OK
Test 2: OK
Test 3: OK
Test 4: OK
Test 5: OK
```

La première commande (sur 2 lignes) est associée à la compilation du programme, les suivantes sont les différents tests qui sont effectués et leur status.

Pour la mise au point de vos programmes, la commande

```
$ ./mon_programme.x < tests/test1.in
```

vous permettra d'analyser le comportement de votre programme avec un jeu de données précis sans comparaison avec la sortie attendue.

À SOUMETTRE

Exercice 1 – Mutliplication (2 pts)

Le matériel pour cet exercice est donné dans le répertoire `exo1`.

Description L'objectif de cet exercice est d'écrire un programme qui lit une liste d'entiers positifs sur son entrée standard (un par ligne) et affiche sur sa sortie standard (un par ligne) le produit de l'entier multiplier par le le numéro de ligne.

Exemple On considère que le programme est un fichier nommé `gain.x`. On considère le premier fichier de test, `test1.in` qui se situe dans le répertoire `tests`,

```
$ cat tests/test1.in
1
2
3
4
5
6
7
8
9
10
11
$ ./gain.x < tests/test1.in
1
4
9
16
25
36
49
64
81
100
121
$
```

Spécifications des entrées L'entrée est un entier positif n compris entre zéro et `INT_MAX` qui vaut 2147483647. Cet entier est lu sur l'entrée standard.

Spécifications des sorties La sortie est un entier compris entre `INT_MIN` et `INT_MAX` qui vaut 2147483647. Le résultat des calcul est affiché sur la sortie standard avec un saut de ligne en fin.

Exercice 2 – Générateur de séquences d'entiers (2 pts)

Le matériel pour cet exercice est donné dans le répertoire `exo2`.

Description L'objectif de cet exercice est de programmer est de lire sur son entrée strandard deux entiers n et m tels que $n < m$ et affiche, un par ligne, la séquence d'entiers pairs dans l'ordre croissant, suivi de la liste des entiers impairs dans l'ordre décroissant.

Exemple On considère que le programme est un fichier nommé `pair-impair.x`. Et nous considérons les deux premiers fichiers de test, `test1.in` et `test2.in` qui se situent dans le répertoire `tests`,

```
$ cat tests/test1.in
0 10
$ ./pair-impair.x < tests/test1.in
0
2
4
6
8
10
9
7
5
3
1
$ cat tests/test2.in
11 24
$ ./pair-impair.x < tests/test2.in
12
14
16
18
20
22
24
23
21
19
17
15
13
11
```

Spécifications des entrées Le programme lit sur son entrée standard deux entiers sous la forme

$$x \quad y$$

où x est la borne de début de séquence et y est la borne de fin de la séquence. De plus, x, y peuvent prendre des valeurs entre `INT_MIN` (-2147483648) et `INT_MAX` (2147483647).

Spécifications des sorties Affichez les nombres un par ligne suivi d'un saut de ligne.

Exercice 3 – Fréquence (2 pts)

Le matériel pour cet exercice est donné dans le répertoire `exo3`.

Description L'objectif de cet exercice est de lire sur son entrée standard une séquence de caractères (un par ligne) et compter le nombre d'occurrences de chaque caractère.

Exemple On considère que le programme est un fichier nommé `frequency.x`. Et nous considérons le premier fichier de test, `test1.in` qui se situe dans le répertoire `tests`,

```
$ cat tests/test1.in
s
a
l
u
t
$./frequency.x < tests/test1.in
a:1
l:1
s:1
t:1
u:1
```

Spécifications des entrées Le programme lit sur son entrée standard une séquence de caractères. Chaque caractère de la séquence occupe une ligne de l'entrée standard. Les caractères considérés sont uniquement les lettres minuscules ou majuscules sans accent et des symboles de ponctuation.

Spécifications des sorties Le programme affiche sur sa sortie standard (dans l'ordre du code ASCII), un par ligne, le caractère suivi du caractère " :", suivi de l'entier représentant le nombre d'occurences.

Exercice 4 – Tri d'entiers (2 pts)

Le matériel pour cet exercice est dans le répertoire `exo4`.

Description Dans cet exercice, on considère un flot de valeurs entières qui est lu sur l'entrée standard. L'objectif est de trier ces valeurs dans l'ordre croissant et d'afficher sur la sortie standard les entiers à une certaine position. Cette position est donnée par une séquence de deux nombres n et m lus en début de séquence. L'affichage sera alors : le plus petit nombre suivi (position 1), du nombre en position $1+n$ suivi du nombre $1+n+m$, suivi du nombre $1+n+m+n$, ..., jusqu'à arrivé à la dernière valeur de la séquence (afficher uniquement si elle fait partie des positions à afficher).

Exemple On considère que le programme est un fichier nommé `sorting.x`. Et nous considérons les deux premiers fichiers de test, `test1.in` et `test2.in`, qui se situent dans le répertoire `tests`,

```
$ cat tests/test1.in
1 1
1
6
7
8
4
3
9
10
2
5
$ ./sorting.x < tests/test1.in
1
2
3
4
5
6
```

```
7
8
9
10
$ cat tests/test2.in
2 4
12
3
10
16
18
19
7
14
13
20
2
1
11
6
17
4
5
8
9
15
$ ./sorting.x < tests/test2.in
1
3
7
9
13
15
19
```

Spécifications des entrées Le programme lit sur son entrée standard des nombres qui sont positionnés un par ligne. Chaque nombre est dans les bornes de valeurs entre `INT_MIN` (-2147483648) et `INT_MAX` (2147483647). Au maximum, il y aura 1000 de données à traiter.

Spécifications des sorties Le programme affiche sur sa sortie standard la suite de nombre dans l'ordre croissant en considérant que les positions exigées en placant un nombre sur chaque ligne. La fin de la séquence est suivie par un saut de ligne.

Exercice 5 – Retour de la multiplication (2 pts)

Le matériel pour cet exercice est dans le répertoire `exo5`.

Description L'objectif de ce programme est de multiplier une séquence de nombres entiers positifs par le numéro de ligne dans l'ordre décroissant de valeurs. S'il y a k valeurs $n_1, n_2, \ldots$alors on aura les produits $k \times n_1$, $(k-1) \times n_2, \ldots$

Exemple On considère que le programme est un fichier nommé `gain2.x`. Et nous considérons un fichier de test, `test1.in` qui se situe dans le répertoire `tests`,

```
$ cat tests/test1.in
1
2
3
4
5
$ ./gain2.x < tests/test1.in
5
8
9
8
5
```

Spécifications des entrées Le programme lit sur son entrée standard des nombres qui sont positionnés un par ligne. Chaque nombre est dans les bornes de valeurs entre `INT_MIN` (-2147483648) et `INT_MAX` (2147483647). Au maximum, il y aura 1000 de données à traiter.

Spécifications des sorties La lise d'entiers résultat de l'opération précisée précédemment.

Exercice 6 – Entrelacement v2 (2 pts)

Le matériel pour cet exercice est donné dans le répertoire `exo6`.

Description Des chaînes de caractères ont été entrelacées et il faut pouvoir les isoler. L'objectif de ce programme est de lire des chaînes de caractères entrelacées, lues sur l'entrée standard, et d'afficher les chaînes reconstituées sur la sortie standard. Petite variation, les chaines de caractères ont été insérées par la fin ("salut" => "tulas").

Exemple On considère que le programme est un fichier nommé `interplay.x`. Et nous considérons le fichier de test, `test2.in` qui se situe dans le répertoire `tests`,

```
$ at tests/test2.in
2
l
s
i
i
n
r
a
o
d
j
$ ./interplay-reverse.x < tests/test2.in
danil
joris
```

Spécifications des entrées Le format de l'entrée sera le suivant. Sur une ligne un entier n qui donne le nombre de chaînes qui sont entrelacées. Les lignes suivantes seront composée d'un caractère appartenant à chaque chaîne de caractères. Remarque, pour un problème donné, les chaînes auront forcément la même longueur et seront constituées uniquement de caractères en minuscule sans accent.

Spécifications des sorties Le programme devra afficher les chaînes de caractères reconstituées, une par ligne et terminera l'affichage par un saut de ligne.

Exercice 7 – Bigfoot :-((2 pts)

Le matériel pour cet exercice est donné dans le répertoire `exo7`.

Description Récemment le serveur hébergent les vidéos des channels a été arrêté pour des raisons de sécurité. Pour ne pas perdre la possbilité de stocker les données des ordinateurs (au nombre de n) ont été placés dans les chambres des étudiants et étudiants d'une même résidence pour dupliquer les possibilités d'accès. On suppose que chaque ordinateurs peut être reliés à n'importe quel autre ordinateur avec des coûts plus ou moins élevés en fonction de la longueur du cable Ethernet. L'objectif est de calculer le métrage minimale de longueur de cable permettant de connecter tous les ordinateurs.

Exemple On considère que le programme est un fichier nommé `bigfoot.x`. Et nous considérons le fichier de test, `test1.in` qui se situe dans le répertoire `tests`,

```
$ cat tests/test1.in
5
1 1 2
5 7 1
1 2 3
2 4 4
2 8 5
3 5 5
3 3 4
4 6 5
$ ./bigfoot.x < tests/test1.in
[ 1 --(1.0)-- 2 ], [ 1 --(2.0)-- 3 ], [ 3 --(3.0)-- 4 ], [ 3 --(5.0)-- 5 ],
```

Spécifications des entrées Le programme lira sur son entrée standard sur une ligne un nombre entiers positifs indiquant le nombre d'ordinateurs dans la résidence. Un certain nombre de lignes suivront et auront la forme suivante :

$$x \quad y \quad z$$

x et z représente deux numéros d'ordinateurs (entiers strictement positifs) et z représente le métrage de cable nécessaire pour relier les deux ordinateurs x et y.

Spécifications des sorties Le programme affichera la liste des ordinateurs suivi d'un saut de saut. Plus précisément chaque lien entre deux ordinateurs aura la forme

```
[ x --(p)-- y ]
```

avec x et y des entiers et p le métrage afficher avec un chiffre après la virgule. Chaque lien sera suivi par une virgule même le dernier.

Exercice 8 – Percolation (2 pts)

Le matériel pour cet exercice est donné dans le répertoire `exo8`.

Description En cours un algorithme de percolation a été présenté succintement pour savoir si un matériaux est poreux ou pas. L'objectif de cet exercice est de mettre en oeuvre cet algorithme.

Exemple On considère que le programme est un fichier nommé `percolation.x`. Et nous considérons le fichier de test, `test1.in` qui se situe dans le répertoire `tests`,

```
$ cat tests/test1.in
3
101
101
101
$ ./percolation.x < tests/test1.in
PERCOLATION
```

Spécifications des entrées L'entrée suivra le format suivant :
- sur la première ligne un entier strictement positif n, et dont la valeur maximale sera 20, donnant la dimension d'un côté d'un carré représentant la surface du matériau ;
- une matrice de dimension $n \times n$ rempi de 1 et 0, 1 représentant un site plein et 0 un site vide.

Spécifications des sorties Le programme affichera sur une seule ligne suivie d'un retour le message :
- $PERCOLATION$ si on détecte que des cases de la ligne 0 sont connectées aux cases de la lignes $n-1$.
- $PASPERCOLATION$ sinon

Exercice 9 – Tout à l'égout (2pts)

Le matériel pour cet exercice est donné dans le répertoire `exo9`.

Description Dans une ville ancienne, les maisons ne sont pas reliées au tout à l'égout. Les habitants doivent donc se déplacer pour se débarasser des eaux usées. Cependant, il existe un lieu dans la ville où on peut se débarasser des eaux usées. L'objectif de cet exercice est de calculer le chemin le plus rapide entre les maisons et le lieu de vidange.

Exemple On considère que le programme est un fichier nommé `percolation.x`. Et nous considérons le fichier de test, `test1.in` qui se situe dans le répertoire `tests`,

```
$ cat tests/test1.in
5 4
0 1 0 0 3
2 0 1 0 0
0 3 0 1 0
0 1 3 0 1
1 0 0 2 0
$ ./sewage.x < tests/test1.in
Maison 1: 2->5->
Maison 2: 3->1->
Maison 3: 4->
Maison 4: 5->2->
Maison 5: 1->4->
```

Spécifications des entrées L'entrée suivra le format suivant :
- sur la première ligne deux strictement positif n et m, n est le nombre de maison (incluant également le lieu de vidange) et m est le numéro associé au lieu de vidange.
- une matrice de dimension $n \times n$ rempi d'entiers positifs : 0 indiquant une absence de chemin entre deux maison et un entier strictement positif indiquant la distance pour rejoindre une maison à partir d'une autre.

Spécifications des sorties Le programme affichera sur plusieurs lignes les maisons par lesquelles passées pour vider les eaux usées.