PETIT Apolline
QUERTON Tristan

TD7B

POLICE SCIENTIFIQUE

Une enquête d'Atticus GORE – Mars 2021

NOTE IMPORTANTE : Les valeurs rendues par les chronomètres peuvent être fausses. En effet, la valeur rendue est parfois négative ou nulle. C'est impossible dans le cas d'une durée de traitement nul car le processeur utilisé à une cadence trop élevée pour q u'un tick de processeur correspondent à la durée du traitement.

Ce sont des erreurs liées à de multiples compilations et exécution en série du programme sur le même processeur. Il suffit de relancer le programme ou d'attendre quelques temps que processeur effectue des traitements différents et d'éliminer de la mémoire vive les potentielles résidus de compilation.

Chapitre 1

Vous incarnez **Atticus GORE**, ingénieur, expert en analyse criminelle et inspecteur de la police de Los Angeles, et enquêtez sur la mystérieuse disparition de **Lena FOWLINGS**, jeune fille de 23 ans que ses parents décriraient comme « gothique » et qui a disparue le 29 Février 2021[1]. (1)

Vous êtes passionné d'informatique et ami avec **Kat KORDELL** (alias votre chargé(e) de TD), détective privée qui vous a déjà aidé à résoudre quelques affaires.

Kat vous suggère de commencer par faire le point sur les fréquentations de **Lena FOWLINGS** en vous plongeant dans ses communications téléphoniques.
Après avoir interrogé les parents de la jeune **Lena** pour obtenir l'information, vous contactez la société BELL PHONE chez qui elle a souscrit son abonnement.
Sur présentation de votre commission rogatoire, la société BELL PHONE vous transmet un fichier texte contenant la liste simplifiée des appels de **Lena** sur les 6 derniers mois, ce qui représente 1000 appels en quelques 180 jours, soit un peu plus de 5 appels par jour en moyenne.

Cette liste est mise à votre disposition à l'adresse suivante : Votre espace de cours MOODLE Informatique 2, section Études de cas « **TP Police scientifique – releve telephonique.txt** »

En accédant, via l'application Bloc-Notes, à cette liste que vous venez de télécharger, vous constatez que l'ensemble des numéros sont affichés au format international, c'est-à-dire en commençant par +33 suivi des 9 chiffres (privé du 0) des numéros français (ex. +33145543212, +33663459808…), suivi de la mention « entrant » ou « sortant » puis du nom du correspondant.

En passionné d'informatique que vous êtes, vous vous précipitez sur votre ordinateur pour coder une petite application qui va rapidement vous permettre de tirer un maximum d'informations de cette liste. C'est parti !

[1] (1) Librement inspiré du Roman UN(e)SECTE de Maxime CHATTAM, chez Albin Michel, 2019

Tout d'abord vous constatez qu'il y a précisément **1000 appels**, émis ou reçus, pas un de plus, pas un de moins.

Cette liste est au format texte, elle est donc constituée de 1000 lignes, c'est à dire 1000 chaînes de caractères qu'il faudra récupérer une par une pour les traiter et extraire les numéros qui s'y trouvent. Il faudra également savoir conserver l'association de ces numéros avec le nom des correspondant. (Consultez les annexes ou revoyez votre cours de C en cas de besoin).

Vous décidez ensuite de mettre un peu **d'ordre dans cette liste en triant par ordre croissant** les numéros de téléphone, toutes catégories confondues. Pour ce faire, vous décidez tout d'abord de coder un tri par permutation 2 à 2, encore appelé tri Bulle (voir Annexes). Amateur de précision que vous êtes, vous affichez l'horloge de votre ordinateur juste avant le tri (avant son démarrage), puis juste après le tri (avant tout éventuel affichage du résultat) afin d'estimer le temps d'exécution que prend cette opération.

Tri à bulle #méthode 1 : 0.010s

Soucieux d'aller vite dans votre enquête et de profiter de cette séance de codage pour vous munir des meilleurs outils pour vos enquêtes futures, vous estimez qu'il existe certainement de meilleurs algorithmes de tri et décidez de coder un « tri par insertion » (voir Annexes). Là encore, soucieux d'évaluer l'efficacité de votre travail, vous affichez l'horloge de la machine juste avant et juste après le tri, avant toute influence de l'affichage.

Tri par insertion #Méthode 1 : 0.007s

Alors ? Quel est le meilleur algorithme de tri en temps mesuré ?
Le tri à bulles plus lent que le tri par insertion. Mais cette différence est minime pour un tri avec si peu de donnée.

Si les différences ne vous semblent pas probantes, pensez-vous que cela puisse venir du faible nombre de valeurs triées ?
Le tri à bulle est plus efficace que le tri par insertion quand il s'agit d'un grand nombre de données. Le tri par insertion est efficace pour des petites quantités de données ou des listes presque triées. En augmentant la quantité de données à trier, un constate en effet que tri à bulle est plus efficace.

Nous devons donc être dans le cas ou le nombre de données à traiter est trop petit pour que le tri à bulles soit plus rapide que le tri par insertion.

Ou finalement la complexité d'un tri bulle n'est-elle pas vraiment meilleure que celle d'un tri par insertion ?
La complexité de ces deux tris est identique. Mais comme expliqué précédemment, chaque tri est plus efficace que l'autre dans une situation précise.

Notez ici la complexité au pire d'un tri par permutation dit tri bulle : n^2
Et celle d'un tri par insertion : n^2

Au fait, vous vous y êtes pris comment pour gérer ces numéros de téléphone ?
Nous créons une structure pour chaque appel. Il est donc nécessaire de créer un tableau de 1000 structures d'appels. Notre structure est composée de

N'y avait-il pas plusieurs façons de voir ces numéros ?
Il aurait aussi été possible de créer un tableau de 1000 chaines de caractères. Une chaine contient toutes les informations d'un appel, elle est composée de 4 chaînes de caractères : Nom, Prénom, Type d'appel et numéro de téléphone. Nous avons choisi de laisser le numéro en chaîne de caractère car cela demande moins d'espace mémoire et pour trier il nous suffit d'utiliser la fonctions *strcmp()*, de la bibliothèque <string.h>

Et si vous percevez ces différentes approches des numéros, cela a-t-il une influence sur le temps d'exécution de votre algorithme de tri ? A tester !
Notez le temps d'exécution des différentes approches ici :

Nous avons donc créé un tableau de 1000 chaines de 60 caractères contenant toutes les informations d'un appel. Nous réalisons à nouveau le relevé des durées de tri :

Tri à bulle #méthode 2 : 0.016s

Tri par insertion #méthode 2 : 0.017s

On observe que la méthode 2 à un temps d'exécution plus long, cependant celle-ci demande moins d'espace mémoire car elle ne nécessite pas l'utilisation d'une structure en plus.
Les durées des tris sont si petites quelle ne sont pas vraiment représentative de l'efficacité d'un tri par rapport à un autre.

[optionnel]

Toujours pas satisfait de l'efficacité de votre algorithme de tri ? Vous pourrez chez vous continuer votre recherche de performance en temps de calcul en testant les algorithmes de tri par pivot, tri par fusion, le « quick sort », algorithmes stables ou instables…
Consultez le net.

Nous avons ajouté 2 tris à notre enquête, le tri par fusion et par sélection.

Tri par fusion : 0.002s

Tri par sélection : 0.003s

PETIT Apolline TD7B
QUERTON Tristan

Chapitre 2

Kat est ravie que vous ayez pu récupérer et traiter cette liste !

Mais d'après elle on a besoin d'aller plus loin dans son utilisation. **Kat** est persuadée qu'on peut pousser nos investigations jusqu'à savoir si un correspondant à plus échangé que les autres avec **Lena** ces six derniers mois.

C'est vous l'expert informatique. Vous lui proposez immédiatement de coder un outil algorithmique qui vous révèlera cette information : Vous avez trouvé lequel ? Un **histogramme** évidemment.

Un histogramme d'un ensemble de valeurs consiste à dénombrer les occurrences de chaque instance de valeur (donc de compter !).

Cet histogramme vous en dira plus sur la fréquence des appels reçus et émis par **Lena**. Il vous révélera le nom de celui ou celle qui a harcelé Lena dernièrement.

Mais quel est donc le nom de cet individu ? Noté le ici :
Oscar RIOTTO, il y a eu 140 appels entre Léna est lui ce qui est grandement supérieur aux autres interlocuteurs téléphoniques.

Bravo ! Grâce aux suggestions de **Kat**, vous avez désormais une piste !!!
Mais dites-donc, vous venez de découvrir le soi-disant…*harceleur*. Êtes-vous sûr ? Si ça se trouve c'est Lena qui harcelait cette personne, non ?
Comment pourriez-vous en être sûr ? Y a-t-il un moyen à votre disposition pour en avoir le cœur net ? Oui ??!!
Il est possible de le savoir en étudiant le type d'appels : entrants ou sortants.
Alors codez-le vite. Et répondez à cette question : harceleur ou harcelé ? **Oscar harcelait Lena.**

Remarque : on observe que trois autres individus ont appelé Léna 20 fois chacun sans qu'elle ne les rappelle jamais en retour. Il s'agit de : Andre Sanfraper, Edgar Palatujaine et Vanessa Feimal.
Notre coupable aurait-il des complices ?

Génial !

Fort de ces nouvelles informations, Atticus attrape son blouson et entraine Kat hors de son bureau. Il est temps d'aller discuter avec l'individu identifié pour en savoir plus sur ses rapports avec Lena FOWLING.

Cette enquête s'arrête là pour cette fois. La suite, qui vous permet de poursuivre les séances de TP, fait référence à des notions d'optimisation… d'efficacité…
Posez-vous la question : OK, je gagne du temps d'exécution, mais est-ce que je gagne aussi de la place mémoire ?

Chapitre 3

L'étape précédente de votre enquête qui a consisté à calculer l'histogramme, a pris du temps. Mais combien précisément ?
Soucieux de précision, vous intégrez comme pour l'étape de tri un affichage du temps avant et après le calcul de l'histogramme et un affichage de durée de traitement par différence entre les deux valeurs.

Durée de calcul de l'histogramme ici : 0.004s

Cette durée vous interpelle. C'est long… Trop long !

Alors vous réfléchissez.
Il doit y avoir une méthode plus efficace.
Afin de trouver cette méthode, vous codez un générateur de 1000 entiers entre 0 et 24, bornes comprises.
Au fait, pourquoi cette plage de valeurs ?
Il y a 25 contacts dans la liste des appels, donc 25 suspects. Ainsi, il nous faut un tableau de 25 places allant de 0 à 24.

Si ! Si ! Si vous suivez attentivement cette enquête depuis le début, vous êtes en mesure de répondre. Mais il faut avoir réussi ce qui précède.

Vous appliquez à cette liste de nombres le même algorithme de calcul d'histogramme qu'à la liste de numéros téléphoniques.
Cette fois-ci, on sait qu'il existe 25 suspects, on peut donc utiliser directement un tableau fixe de 25 places. Cela nous fait gagner de l'espace de stockage ainsi que du temps d'exécution.

Vous relevez le temps d'exécution. Notez-le ici : **Durée pour la méthode I : 0.000s**

Et vous réfléchissez… Il y a certainement une façon plus efficace de coder cet algorithme… Combien de fois chaque nombre apparaît-il ?...
Ces nombres sont générés aléatoirement donc d'après la loi de la probabilité, plus il y a de nombre dans le tableau plus le pourcentage d'apparition de chaque nombre va tendre vers 1/25 = 4%
Vous avez trouvé ? Alors codez vite cette nouvelle méthode et évaluez son temps d'exécution par différence d'horloge comme vous le faites depuis le début. Pour rendre le programme plus efficace, on peut attribuer.
Les nombres sont générés aléatoirement de 0 à 24, et les cases de notre tableau vont également de 0 à 24. Ainsi, pour diminuer la complexité de notre calcul d'histogramme, il nous suffit d'attribuer la valeur de la case au numéro de la casse.
C'est à dire : tab[NUMsuspect] = NUMsuspect.
Ainsi, nous parcourons une fois le tableau de 1000 nombres et c'est tout ! Notre tableau d'histogramme possède une seule dimension qui est le nombre de occurrences du nombre.

Notez le temps ici : **Durée pour la méthode II : 0.000s**

Alors, plus efficace ? **C'est deux valeurs sont incohérente, en effet avec 2 boucle imbriquée dans la première méthode, le temps d'exécution de cette partie du code devrait excéder 1 tick de processeur.**

Auriez-vous pu appliquer cette nouvelle méthode, peut être meilleure, à votre liste de numéros téléphoniques ?

Expliquez ici :
Cette méthode fonctionne car les nombres, générés entre 0 et 24 correspondent toujours à une case du tableau du même indice.
Cependant, les numéros de téléphone sont compris entre 0 et 109 si on ne compte pas le +33 au début de chaque numéro. Ainsi, pour être sûre que les numéros ont tous un indice de tableau différent, il nous faudrait un tableau de 1 000 000 000 places. On gagnerait certes en vitesse d'exécution mais on perdrait tellement en espace de stockage que cela n'est absolument pas rentable.
Après étude des 25 numéros différents grâce au chapitre précédent, on remarque que les deux derniers chiffres de chaque numéro sont tous différents, on pourrait donc se dire qu'il suffit d'un tableau de 100 entiers. Cependant en prenant un autre relevé téléphonique il se peut que cela ne fonctionne pas. Voulant que notre code fonction dans tous les cas avec 0% d'erreurs
Cette méthode ne peut pas être utilisée.

Pour conclure, on pourrait utiliser la méthode utilisée précédemment (avec les entiers compris entre 0 et 24), mais cette méthode ne serait pas rentable tellement elle demande d'espace de stockage.