IFT-3001 (Automne 2020) Travail 1

Enseignant : Claude-Guy Quimper
Date de remise : 18 octobre 2020, 23h59.

- Les travaux remis en retard ne seront pas corrigés et se verront attribuer la note de zéro.
- À moins de spécifications contraires, vous devez toujours pleinement justifier vos réponses.
- Il s'agit d'un travail individuel. Toute communication ou entraide avec une autre personne au sujet de ce travail est interdite.
- Vous devez présenter un travail original : aucun emprunt provenant de l'Internet ou de toute autre source n'est permis.
- Le code de programmation doit pouvoir compiler avec g++ 5.4.0. À partir d'une connexion VPN, vous pouvez compiler votre code sur cette page web : http://geyron.fsg.ulaval.ca.

Format de remise :

- Vous devez remettre un fichier zip contenant un fichier PDF par question. La structure du fichier zip ne doit pas contenir de sous-répertoire et doit seulement contenir les fichiers : Question1.pdf, division.cpp, Question2.pdf, Question3.pdf, Question4.pdf et multiplication.cpp.
- Votre nom, prénom et matricule doivent apparaître dans l'en-tête de chaque page.

Question # 1: [25 points]

Considérons deux polynômes $a(x)$ et $b(x)$. L'opérateur de comparaison $a(x) \prec b(x)$ est vrai si et seulement si le degré du polynôme $a(x)$ est strictement plus petit que le celui du polynôme $b(x)$. Soit $a(x)$ le dividende et $b(x)$ le diviseur, nous voulons calculer le quotient $q(x)$ et le reste $r(x)$ tels que $a(x) = q(x)b(x) + r(x)$ et $r(x) \prec b(x)$. En d'autres termes, $q(x)$ est le résultat de la division du polynôme $a(x)$ par $b(x)$ et $r(x)$ est le reste de la division. Ce reste doit être de degré inférieur au diviseur $b(x)$. Voici un exemple de la division de $a(x) = 2x^2 + x - 1$ par $b(x) = x + 2$. Nous obtenons $q(x) = 2x - 3$ et $r(x) = 5$.

$$\begin{array}{rrr|rr} 2x^2 & +x & -1 & x & +2 \\ \cline{4-5} 2x^2 & +4x & & 2x & -3 \\ \cline{1-3} & -3x & -1 & & \\ & -3x & -6 & & \\ \cline{2-3} & & 5 & & \end{array} \quad \text{reste } 5$$

La première partie de cette question consiste à programmer l'algorithme de division. La deuxième partie consiste à calculer le plus grand commun diviseur entre deux polynômes $m(x)$ et $n(x)$; c'est-à-dire le polynôme ayant le plus grand degré qui divise les polynômes $m(x)$ et $n(x)$.

Implémentez l'algorithme d'Euclide. Utilisez votre algorithme calculant le reste d'une division à la place de l'opérateur de modulo.

Notez que le plus grand commun diviseur entre deux polynômes n'est pas unique. Par exemple, le plus grand commun diviseur entre $m(x) = x^2 + 2x + 1$ et $n(x) = x^2 - 1$ peut être $x + 1$ puisque nous pouvons factoriser les deux polynômes en faisant ressortir le facteur $x + 1$: $m(x) = (x + 1)^2$ et $n(x) = (x + 1)(x - 1)$. Cependant, le polynôme $k(x + 1)$, pour $k \neq 0$, est aussi un diviseur commun puisque $m(x) = \big(k(x + 1)\big)\big(\frac{1}{k}(x + 1)\big)$ et $n(x) = \big(k(x + 1)\big)\big(\frac{1}{k}(x - 1)\big)$.

Étapes :

1. Téléchargez le code de programmation pour cette question.
2. Implémentez les fonctions `division` et `plus_grand_commun_diviseur` dans le fichier `division.cpp`. Aucun autre fichier ne doit être modifié. Votre implémentation doit avoir un complexité en pire cas la plus petite possible.
3. Testez votre code sur le serveur de compilation (voir l'en-tête de l'énoncé)
4. Dans un fichier nommé `Question1.pdf`, présentez l'analyse de la fonction `division`. Si votre implémentation a un comportement différent en pire et meilleur cas, vous devez présenter une analyse en pire et une en meilleur cas. L'analyse en cas moyen n'est pas demandée.
   (a) Utilisez la degré du dividende pour la taille de l'instance.
   (b) Si vous avez un pire cas, caractérisez une instance de taille n s'exécutant en pire cas.
   (c) Si vous avez un meilleur cas, caractérisez une instance de taille n s'exécutant en meilleur cas.

(d) Exprimez les résultats de vos analyses à l'aide de la notation Θ. Certaines fonctions sont déjà analysées dans le code de programmation. N'hésitez pas à réutiliser ces résultats d'analyse.

À quoi sert de calculer le plus grand commun diviseur de deux polynômes ? Le plus grand commun diviseur est utilisé dans les algorithmes numériques calculant les zéros d'un polynôme. En effet, certains algorithmes numériques ont de la difficulté à détecter les zéros de multiplicité plus grande que 1. Par exemple, dans le polynôme $p(x) = (x-2)(x-3)^5$, le zéro 3 a une multiplicité de 5. Afin d'éliminer ces zéros, on peut diviser $p(x)$ par le plus grand commun divisieur de $p(x)$ et sa dérivé $p'(x)$. Nous avons :

$$PGCD(p(x), p'(x)) = PGCD((x-2)(x-3)^5, (x-3)^5 + 5(x-2)(x-3)^4) \quad (1)$$

$$= (x-3)^4 \quad (2)$$

$$\frac{p(x)}{PGCD(p(x), p'(x))} = (x-2)(x-3) \quad (3)$$

Remarquez que dans le polynôme $(x-2)(x-3)$, tous les zéros ont une multiplicité de 1, ce qui les rend plus faciles à détecter par les algorithmes numériques.

Question # 2: [25 points] Pour détecter si un prélèvement contient le virus de la COVID-19, on doit le mélanger à une dose de réactif. S'il y a une réaction, c'est que le virus se retrouve dans le prélèvement. La façon la plus simple d'analyser n échantillons est de mélanger chaque échantillon avec une dose de réactif. Toutefois, cela requiert n doses de réactif. Lors d'une pénurie de réactif et lorsque la proportion des prélèvements testés positifs est faible, on peut procéder comme l'algorithme 1 afin de réduire la quantité de doses utilisées. On mélange des prélèvements pour les soumettre à une seule dose de réactif. Si le mélange est testé négatif, c'est qu'aucun prélèvement du mélange ne contient le virus. Si le mélange est testé positif, c'est qu'au moins un prélèvement est infecté par le virus. Il faut alors procéder à des analyses additionnelles sur ces prélèvements.

Tous les algorithmes ci-dessous ont été simplifiés afin de ne retourner que le premier prélèvement testé positif. Vous devez les analyser en pire cas et en meilleur cas. L'analyse en cas moyen n'est pas demandée.

1. Pour chaque algorithme, procédez à l'analyse en meilleur cas.
   (a) Utilisez n pour la taille de l'instance.
   (b) Utilisez *Analyse* pour l'opération baromètre. En effet, il s'agit de l'opération critique puisqu'elle consomme une dose de réactif. De plus, l'analyse peut prendre plusieurs heures afin que le réactif réagisse. Pour cette raison, toutes les autres opérations ont un temps d'exécution négligeable, incluant la fonction *Mélanger*.
   (c) Identifiez une instance de taille n qui constitue le meilleur cas.
   (d) Écrivez la sommation ou la récurrence qui dénombre les appels à la fonction *Analyse*.
   (e) Résolvez la sommation ou la récurrence trouvée à l'étape précédente. Portez attention au traitement des fonctions $\lfloor x \rfloor$ et $\lceil x \rceil$.
   (f) Exprimez le temps d'exécution en utilisant la notation asymptotique Θ.
2. Répétez les étapes au point 1, mais pour l'analyse en pire cas.

Algorithme 1 : AnalyseParLotsDeDix($E[0..n-1]$)

```
pour i ← 0 à ⌈n/10⌉ − 1 faire
    z ← min(10i + 9, n − 1);
    M ← Mélanger(E[10i], E[10i + 1], ..., E[z]);
    si Analyse(M) = positif alors
        pour j = 10i à z − 1 faire
            si Analyse(E[j]) = positif alors
                retourner E[j];
        retourner E[z];
return Aucun échantillon positif;
```

Algorithme 2 : AnalyseDichotomique($E[0..n-1]$)

si $n = 1 \wedge$ *Analyse*$(E[0]) =$ *positif* **alors**
  retourner $E[0]$;
si $n \leq 1$ **alors**
  retourner Aucun échantillon positif;
$M_1 \leftarrow$ Mélanger$(E[0], E[1], \ldots E[\lfloor \frac{n}{2} \rfloor - 1])$;
si *Analyse*(M_1) = *positif* **alors**
  $E' \leftarrow [E[0], E[1], \ldots E[\lfloor \frac{n}{2} \rfloor - 1]]$;
  retourner AnalyseDichotomique(E');
$M_2 \leftarrow$ Mélanger$(E[\lfloor \frac{n}{2} \rfloor], E[\lfloor \frac{n}{2} \rfloor + 1], \ldots, E[n-1])$;
si *Analyse*(M_2) = *positif* **alors**
  $E' \leftarrow [E[\lfloor \frac{n}{2} \rfloor], E[\lfloor \frac{n}{2} \rfloor + 1], \ldots, E[n-1]]$;
  retourner AnalyseDichotomique(E');
retourner Aucun échantillon positif;

Algorithme 3 : AnalyseInsensée($E[0..n-1]$)

pour $i = 0$ **à** $\lceil \sqrt{n} \rceil$ **faire**
  pour $j = i$ **à** i^2 **faire**
    si *Analyse*$(E[(42 + j) \mod n])$ = *positif* **alors**
      retourner $E[(42 + j) \mod n]$;
retourner Aucun échantillon positif;

Question # 3: [25 points] Résolvez la récurrence suivante avec la substitution à rebours.

$$C(n) = \begin{cases} 0 & \text{si } n \leq 1 \\ 2C(\lfloor \frac{n}{3} \rfloor) + n & \text{si } n > 1 \end{cases}$$

Vous devez faire au moins **deux** substitutions. Vous pouvez supposer que n est une puissance d'un entier. Vous devez trouver la formule exacte de la récurrence et non pas son ordre de croissance. N'utilisez que les sommations de l'aide-mémoire pour simplifier votre résultat. Le résultat final doit être dans sa forme la plus simple possible, donc ne laissez pas de logarithmes dans les exposants.

Question # 4: [25 points]

Préambule Les algorithmes de cryptographie encodent les messages circulant sur l'Internet. Chaque message est une séquence de bits qui peut être interprétée comme un très grand entier. Les algorithmes de cryptage, s'appuyant sur la théorie des nombres, utilisent donc des opérations arithmétiques pour coder et décoder les messages. Pour cette question, vous devrez concevoir un algorithme qui multiplie deux entiers. Les deux entiers pourront avoir un nombre arbitraire de bits. Pour concevoir votre algorithme, vous pouvez utiliser les opérateurs arithmétiques du processeur en sachant que ceux-ci ne peuvent manipuler que des entiers de w bits (où w est généralement 32 ou 64 bits).

Problème Vous devez concevoir un algorithme de type diviser pour régner qui prend deux tableaux de bits $A[0..n-1]$ et $B[0..m-1]$ représentant les nombres $A = \sum_{i=0}^{n-1} A[i]2^i$ et $B = \sum_{i=0}^{m-1} B[i]2^i$ et qui retourne un tableau de $n+m$ bits $C[0..n+m-1]$ représentant le nombre $C = A \times B$. Pour concevoir votre algorithme, vous disposez des fonctions ADDITION$(X[0..n-1], Y[0..m-1])$ et SOUSTRACTION$(X[0..n-1], Y[0..m-1])$ qui prennent deux nombres de n et m bits et qui retournent un nombre d'au plus $\max(n,m)+1$ bits représentant la somme $X+Y$ et la différence $X-Y$. Les algorithmes ADDITION et SOUSTRACTION s'exécutent en temps $\Theta(n+m)$.

Vous devez concevoir un algorithme de type diviser pour régner. Divisez le problème de la façon suivante. Prenez les $\lfloor \frac{n}{2} \rfloor$ bits les plus faibles de l'entier A et enregistrez-les dans un tableau de $\lfloor \frac{n}{2} \rfloor$ bits que vous appelez $\underline{a}$. Vous prenez ensuite les $\lceil \frac{n}{2} \rceil$ bits les plus forts pour créer un tableau $\overline{a}$. Nous avons donc la relation $A = 2^{\lfloor n/2 \rfloor} \times \overline{a} + \underline{a}$. Vous faites de même avec B que vous exprimez en fonction de $\overline{b}$ et $\underline{b}$. L'algorithme classique de multiplication procède ainsi.

$$
\begin{array}{cccc}
 & & \overline{a} & \underline{a} \\
\times & & \overline{b} & \underline{b} \\
\hline
 & & \overline{a}\underline{b} & \underline{ab} \\
+ & \overline{a}\overline{b} & \underline{a}\overline{b} & \\
\hline
 & \overline{a}\overline{b} & (\overline{a}\underline{b} + \underline{a}\overline{b}) & \underline{ab}
\end{array}
$$

Nous avons donc la relation

$$A \times B = 2^{2\lfloor n/2 \rfloor}\overline{a}\overline{b} + 2^{\lfloor n/2 \rfloor}(\underline{a}\overline{b} + \overline{a}\underline{b}) + \underline{ab} \tag{4}$$

Lorsque n est pair, cette solution necessite de faire quatre multiplications de nombres à $\frac{n}{2}$ bits de même que des opérations d'addition. Or, il est possible de réduire le nombre de multiplications. Pour calculer la valeur de $\underline{a}\overline{b} + \overline{a}\underline{b}$, vous pouver utiliser cette équation.

$$\underline{a}\overline{b} + \overline{a}\underline{b} = (\overline{a} + \underline{a})(\overline{b} + \underline{b}) - \overline{a}\overline{b} - \underline{ab} \tag{5}$$

1. Téléchargez le code de programmation pour cette question.
2. Dans le fichier `multiplication.cpp`, programmez l'algorithme décrit ci-dessus.

3. Dans le fichier `Question4.pdf`, analysez votre algorithme [1] et exprimez son efficacité à l'aide de la notation Θ. Pour l'analyse, vous pouvez supposer que les deux nombres que vous multipliez ont exactement n bits. Si vous avez bien suivi toutes les étapes de l'algorithme, vous devriez obtenir un temps d'exécution dans $O(n^2)$, mais pas dans $\Theta(n^2)$.

Remarque : Quand on écrit le contenu d'un vecteur x sur une feuille de papier, on place généralement l'élément $x[0]$ à gauche. Or, lorsque l'on écrit un nombre, le chiffre le moins significatif (celui à la position 0) est placé à droite. C'est de cette façon, avec le bit le moins significatif à droite, que les tests unitaires affichent les nombres. Ne vous faites pas piéger en déboguant votre code !

1. Si l'efficacité de votre algorithme ne dépend pas uniquement de la taille de l'instance, seule l'analyse en pire cas est demandée