ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE – LAUSANNE
POLITECNICO FEDERALE – LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY – LAUSANNE

Faculté Informatique et Communications
Cours d'Information, Calcul et Communication, sections MA et PH
Chappelier J.-C.

Information, Calcul et Communication (SMA/SPH) : Examen III

21 décembre 2018

SUJET 1

INSTRUCTIONS (à lire attentivement)

IMPORTANT ! Veuillez suivre les instructions suivantes à la lettre sous peine de voir votre examen annulé dans le cas contraire.

1. Vous disposez de deux heures quarante-cinq minutes pour faire cet examen (13h15 – 16h00).
2. Vous devez **écrire à l'encre noire ou bleu foncée**, pas de crayon ni d'autre couleur.
   N'utilisez **pas non plus de stylo effaçable** (perte de l'information à la chaleur).
3. Vous avez droit à toute documentation papier.
   En revanche, vous ne pouvez pas utiliser d'ordinateur personnel, ni de téléphone portable, ni aucun autre matériel électronique.
4. Répondez aux questions directement sur la donnée, **MAIS** ne mélangez pas les réponses de différentes questions !
   Ne joignez aucune feuilles supplémentaires ; **seul ce document sera corrigé**.
5. Lisez attentivement et *complètement* les questions de façon à ne faire que ce qui vous est demandé. Si l'énoncé ne vous paraît pas clair, ou si vous avez un doute, demandez des précisions à l'un des assistants.
6. L'examen comporte 6 exercices indépendants sur 20 pages, qui peuvent être traités dans n'importe quel ordre, mais qui ne rapportent pas la même chose (les points sont indiqués, le total est de 150 points) :
   1. questions courtes : 26 points ;
   2. problème : 63 points ;
   3. mélange de mots : 12 points ;
   4. compréhension de programme : 16 points
   5. la cache est déterminante : 19 points et
   6. chercher les erreurs : 14 points.

   Tous les exercices comptent pour la note finale.

Question 1 – Questions courtes [26 points]

Question 1.1 [4 points]

Un sac contient 24 billes de 4 couleurs différentes : 12 billes rouges, 8 billes bleues, 2 billes vertes, et 2 billes oranges. Quelle est, en bit, l'entropie du jeu consistant à deviner la couleur d'une bille tirée au hasard ?

Donnez votre réponse sous la forme $a + b \log_2(3)$ (précisez les valeurs de a et b, éventuellement sous forme de fraction) :

$$\frac{1}{2} + \frac{1}{3}\log_2(3) + \frac{1}{6}\log_2(12) = \mathbf{\frac{5}{6}} + \mathbf{\frac{1}{2}}\log_2(3)$$

Question 1.2 [5 points]

A partir d'un alphabet de 33 lettres, on compose un mot X de 128 lettres au total, chacune des lettres de l'alphabet étant présente au moins une fois dans le mot X. Le code de Huffman de ce mot a une longueur moyenne de 5.5 bits.

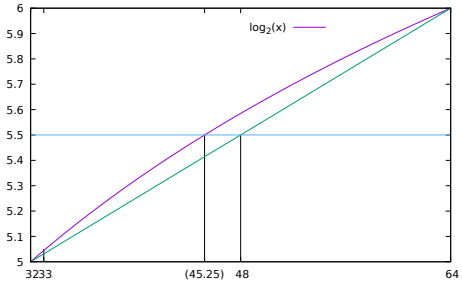
1. Est-ce possible ? Justifiez *brièvement* votre réponse.

   La réponse attendue était : oui ça *semble* possible :

   $$H(X) \leq \log_2(33) < L_c(\text{Huffman}(X))$$

   et

   $$L_c(\text{Huffman}(X)) < \log_2(33) + 1$$

   Note : ceci dit, le choix de 5.5 pour $L_c(\text{Huffman}(X))$ est un peu extrême et, en fait, trop grand. On pourrait par exemple le majorer par un code non-optimal (genre tout le monde à 5 bits, sauf deux à 6 bits) qui donne un majorant inférieur à 5.5 : donc c'est, en fait, impossible d'avoir 5.5 ; mais je n'attends pas un tel niveau de raisonnement.

2. Si **oui**, donnez les *meilleures* bornes possibles (haute et basse) que vous pouvez pour
   a) l'entropie de ce mot :

   $$L_c(\text{Huffman}(X)) - 1 \leq H(X) \leq \log_2(33)$$

   En fait, en toute rigueur sur 128 lettres, on a l'entropie maximale lorsque 29 lettres apparaissent 4 fois et les 4 autres lettres apparaissent 3 fois, ce qui fait une entropie de $\frac{5\times 29}{32} + \frac{3}{32}\log_2(\frac{128}{3}) = 5.039$. Mais cette réponse n'était pas attendue.

   b) la longueur moyenne d'un code de Shannon-Fano de ce mot :

   $$5.5 \leq L_c(\text{Shannon-Fano}(X)) \leq H(X) + 1 \leq \log_2(33) + 1$$

et si c'est **non**, donnez les *meilleures* bornes possibles (haute et basse) que vous pouvez pour la longueur moyenne d'un code de Huffman de ce mot :

$$\leq L_c(\text{Huffman}(X)) \leq$$

Justifiez brièvement vos réponses.

Justifications :

- théorème de codage de source de Shannon
- optimalité des codes de Huffman

Question 1.3 [5 points]

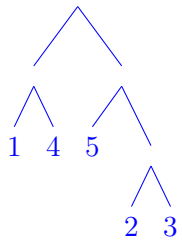
Des séquences de cinq niveaux d'alerte météo doivent être transmises codées (code sans-préfixe et sans perte) sous forme de séquences de pastilles (ronds) rouges ou vertes. La table ci-dessous représente trois propositions de codes possibles. Malheureusement, ce sujet est tiré en noir et blanc ; la couleur verte ou rouge s'est donc perdue...

code I	code II	code III
niveau 1 : ●	niveau 1 : ●●	niveau 1 : ●●●
niveau 2 : ●●●●	niveau 2 : ●●●	niveau 2 : ●
niveau 3 : ●●	niveau 3 : ●●●	niveau 3 : ●●●
niveau 4 : ●●●●	niveau 4 : ●●	niveau 4 : ●
niveau 5 : ●●	niveau 5 : ●●	niveau 5 : ●●

Quel(s) code(s) (d'origine, avec les couleurs) êtes vous néanmoins sûr(e) de ne pas pouvoir utiliser pour la transmission désirée ? Justifiez *brièvement* votre réponse.

Les codes I et III ne peuvent pas être utilisés car ils ont nécessairement des mots qui sont préfixes d'autres : à vérifier soit en utilisant l'inégalité de Kraft, soit simplement en essayant d'affecter des couleurs aux mots les plus courts.

Le code II par contre *pourrait* (mais on n'est pas sûr) être un code utilisable ; p.ex. :



Question 1.4 [7 points]

Soit f une fonction de $\mathbb{R}$ dans $\mathbb{R}$ définie par

$$f(t) = \sum_{m \in \mathbb{Z}} a_m \, \mathrm{sinc}(30\,t - m)$$

où $\mathrm{sinc}(x) = \dfrac{\sin(\pi\,x)}{\pi\,x}$ et $a_m = 7\sin(\frac{2\pi}{3}m + \frac{\pi}{6}) + 3\sin(\frac{4\pi}{5}m + \frac{\pi}{4}) + 2\sin(\frac{2\pi}{6}m)$.
Écrire $f(t)$ comme la somme de trois fonctions sinus :

$$f(t) = 7\sin(20\pi\,t + \frac{\pi}{6}) + 3\sin(24\pi\,t + \frac{\pi}{4}) + 2\sin(10\pi\,t)$$

Justifiez brièvement votre réponse.

Voyons a_m comme les échantillons d'un signal $g(t)$: $a_m = g(m\,T_e) = g(\frac{m}{f_e})$ avec donc

$$g(t) = 7\sin(2\pi\,\frac{f_e}{3}\,t + \frac{\pi}{6}) + 3\sin(2\pi\,\frac{2\,f_e}{5}\,t + \frac{\pi}{4}) + 2\sin(2\pi\,\frac{f_e}{6}\,t)$$

La bande passante de g ($\frac{2\,f_e}{5}$) étant strictement inférieure à $\frac{f_e}{2}$, g sera reconstruite parfaitement par la formule de reconstruction, qui est justement celle utilisée dans f avec un $f_e = 30$ Hz, et on a donc pour tout t, $f(t) = g(t)$.

Question 1.5 [5 points]

Quel(s) problème(s) existant(s) ou potentiel(s) voyez vous dans l'*extrait* de code suivant :

```
int s(0);
for (int n(0); n <= v.size(); ++n) {
    s = s + v[n];
}
double m(s / v.size());
```

- Le `<=` devrait être un `<` ;
- `m` étant en `double`, il faudrait faire attention à la division entière et la faire en `double` ;
- `v` pourrait être vide : il faut donc faire un test pour éviter une division par un `v.size()` nul.
- `n` devrait être un `size_t` pas un `int`.

Question 2 – Problème [63 points]

On s'intéresse ici à transmettre de façon compacte et sécurisée un son sur le réseau Internet. Plusieurs étapes sont donc nécessaires, parmi lesquelles : numériser le son, comprimer sa représentation, l'encrypter et transporter les paquets correspondant via IP.

Question 2.1 – Numérisation [8 points]

Question 2.1.1 – Échantillonnage [6 points]

Le son que l'on souhaite transmettre a été produit par un synthétiseur correspondant au code C++ ci-contre (dans lequel les temps sont en secondes et les fréquences en hertz).

```
struct Composante {
    double amp;
    double freq;
    double phase;
};

const vector<Composante> signal({
    { 1.2,   333.5,  0.78  },
    { 0.78,  611.2,  0.5   },
    { 1.42,  440.0,  0.65  },
    { 0.33,  123.7,  0.44  },
    { 2.18,  432.5,  0.15  },
  });

double son(double t)
{
    double x(0.0);
    for (auto c : signal) {
        x += c.amp * sin( 2 * M_PI * c.freq * t
                          + c.phase );
    }
    return x;
}
```

Pour le numériser, on va prendre des échantillons toutes les 0.8 ms.

1. Peut-on reconstruire correctement le signal ainsi échantillonné en utilisant la formule de reconstruction vue en cours ?
2. Et si l'on échantillonnait toutes les 1 ms ?

Justifiez pleinement vos deux réponses et proposez une solution alternative dans chaque cas de réponse négative.

Réponse :

0.8 ms correspond à un f_e de 1250 Hz. Toutes les fréquences étant strictement inférieures à sa moitié (625 Hz), le signal sera en effet parfaitement reconstruit.

1 ms correspond à un f_e de 1000 Hz, ce qui ne permet pas de reconstruire le signal correctement. Il faudrait filtrer (filtre passe-bas) toutes les fréquences supérieures à 500 Hz.

Question 2.1.2 – Quantification [2 points]

Sachant que l'amplitude de ce signal est toujours comprise entre -5.8 et 5.8, on décide de représenter les valeurs des échantillons en utilisant des nombres entiers signés sur 8 bits (à intervalles réguliers et en utilisant toutes les valeurs possibles). Quelle précision relative par rapport à l'amplitude maximale (que l'on supposera égale à 5.8 ici) a-t-on avec une telle représentation ? Donnez votre réponse sous forme de fraction et justifiez votre réponse.

précison de représentation : $\frac{2\times5.8}{255}$; relative : $\frac{2}{255}$.

Question 2.2 – Compression [27 points]

Question 2.2.1 – Calcul de l'entropie [13 points]

Après numérisation, on se retrouve donc avec une suite de valeurs que l'on peut représenter sur des `char` en C++ (8 bits).

Écrivez ici une *fonction* C++ pour calculer l'entropie (en bit) d'une telle suite de valeurs :

Voir l'exercice 5 de la série 8.

```
double entropie(vector<char> const& signal)
{
  // 1) Compter (note: on pourrait utiliser une map ici, mais non vue en cours)
  array<int, 256> comptes;
  for (auto& x : comptes) x = 0;
  double somme(0.0); // en double pour éviter pb division plus tard
  for (auto c : signal) { /* auto devrait être « unsigned char » mais
                           * trop subtil pour le niveau de cours *et*
                           * pour un examen sur papier !              */
    ++comptes[c];
    ++somme;
  }

  // 2) Calcul de l'entropie
  double H(0.0);
  for (auto& x : comptes) {
    if (x != 0) { // le log(0) n'est pas défini !
      const double p(x/somme); // pas de risque de division par 0 ici
                               // (x non nul => somme non nulle)
      H -= p*log(p);
    }
  }

  return H / log(2.0) ; // ne pas oublier les unités : bit !
}
```

Question 2.2.2 – Taux de compression [2 points]

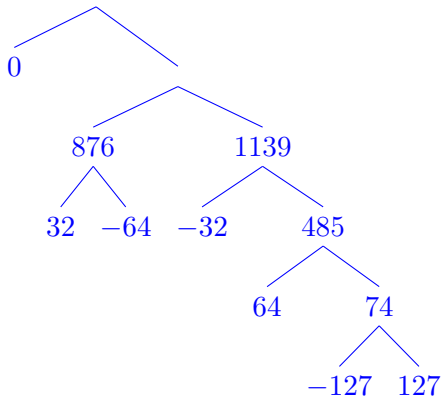
Votre programme précédent vous donne une entropie de 5.18 bit pour le signal considéré. Quel taux de compression pouvez vous alors espérer (bornes) ? Donnez votre réponse sous forme de deux fractions et justifiez brièvement votre réponse.

Par le théorème de Shannon (les deux parties) : entre (8-5.18)/8 et (8-6.18)/8 (soit 25 à 35%).

Question 2.2.3 – Compression [12 points]

Pour simplifier ici, supposons que l'on n'ait finalement que les sept valeurs suivantes : -127, -64, -32, 0, 32, 64, 127 observées respectivement 21, 432, 654, 1324, 444, 411 et 53 fois.

Proposez ici un code de compression optimal pour ces valeurs :



distribution des longueurs de mots de code :

-127	-64	-32	0	32	64	127
5	3	3	1	3	4	5

☞ En utilisant votre code, codez la séquence d'échantillons -64, 0, -32, 0, 32 :

doit être conforme à la distribution de longueurs ci-dessus : 3 bits, 1 bit, 3 bits, 1 bit, 3 bits.

Question 2.3 – Sécurité [11 points]

Pour éviter tout vol de votre musique (droit d'auteur), vous souhaitez la transmettre en utilisant RSA.

Question 2.3.1 – Réencodage [2 points]

Pour cela, vous regroupez les mots de codes précédents pour en former à nouveau des entiers, en remplissant si nécessaire la fin par des 0.

Par exemple, si la séquence à transmettre est codée `10`, `0111`, `000`, `11`, `001`, vous regroupez les bits de sorte à transmettre `10011100`, `01100100`.

A quelle valeur décimale correspond le premier mot transmis ci-dessus (`10011100`), interprété comme entier non signé ?

$128 + 16 + 8 + 4 = 156$.

Question 2.3.2 – Encryptage [3 points]

On interprète donc maintenant le message à transmettre comme une suite d'entiers positifs que l'on code chacun en utilisant RSA.

Votre clé publique est $(79, 899)$; votre clé privée est 319. La clé publique de votre destinataire est $(485, 667)$.

Comment transmettez vous l'entier 238 ? Donnez la réponse sous la forme « $a^b \mod c$ » :

$$238^{485} \mod 667$$

Si vous recevez l'entier 532, en quoi le décodez vous ? Donnez la réponse sous la forme « $x^y \mod z$ » :

$$532^{319} \mod 899$$

Question 2.3.3 – Encryptage en C++ [6 points]

Écrivez ici une fonction C++ capable de calculer la valeur (entier non signé) de $x^y \mod z$. Par exemple pour $x = 8$, $y = 27$ et $z = 55$, cette fonction retourne la valeur 2.

Voir transparent 37/51 de la leçon IV.1.

```
unsigned int exp(unsigned int x, unsigned int y, unsigned int z)
{
  if (y == 1)
    return x % z;

  else if (y % 2 == 0)
    return exp( ((x % z) * (x % z)) % z, y / 2, z) % z;

  else
    return (x % z) * exp( ((x % z) * (x % z)) % z, (y - 1) / 2, z) % z;
}
```

Une simple boucle sur des multiplications ne fonctionne pas car on aurait rapidement un dépassement de capacité des `int`.

Question 2.4 – Transmission TCP/IP [17 points]

Il est maintenant temps de transmettre effectivement nos informations sur le réseau Internet (qui utilise le protocole TCP/IP).

Pour simplifier, nous ne mettrons qu'un seul entier par paquet réseau. Pour rappel, la couche TCP ajoute comme information à chaque paquet :

- une indication si c'est un paquet de donnée ou un paquet d'acquittement ;
- un numéro de séquence pour remettre les paquets dans l'ordre.

Question 2.4.1 – Paquet TCP [2 points]

Proposez ici un type C++ pour représenter les paquets TCP :

```
struct paquet {
  bool acquit;
  unsigned int seq;
  unsigned int data;
};
```

Question 2.4.2 – Ordre des paquets TCP [3 points]

En utilisant le type proposé ci-dessus, écrivez une fonction C++ qui, pour deux paquets donnés, indique s'ils sont dans le bon ordre ou non : elle retourne `true` si le paquet passé en premier argument arrive avant celui passé en second argument.

```
bool operator<(paquet const& a, paquet const& b) {
  return a.seq < b.seq;
}
```

Question 2.4.3 – Réception TCP [12 points]

Écrivez la fonction C++ qui gère la réception TCP des paquets : elle reçoit comme arguments un paquet P, un numéro i et une liste L de paquets ordonnés par numéro de séquence décroissant, et elle :

- ne fait rien et retourne i si P est un paquet d'acquittement ;
- crée le paquet d'acquittement correspondant à P (même contenu, seule l'information d'acquittement change) et l'envoie en utilisant une fonction `send()` que l'on suppose exister ;
- si le paquet n'est pas le suivant direct du dernier paquet reçu, dont le numéro est indiqué par l'argument i, elle ajoute P à sa place dans L et retourne i ; l'insertion de P à sa place dans L se fait en utilisant une fonction `insert()` que l'on suppose exister ;
- si le paquet P est le suivant direct du dernier paquet reçu (dont le numéro est indiqué par l'argument i), elle l'affiche en utilisant une fonction `affiche_paquet()` (au singulier), que l'on suppose exister ;
  si, de plus, le dernier paquet de L est celui attendu juste après P, elle affiche également tous les paquets jusqu'au prochain paquet manquant et les supprime de L (exemple ci-dessous) ;
  dans ce cas, la fonction retourne le numéro de séquence du dernier paquet affiché ;

  par exemple, si i vaut 42, et que le numéro de séquence de P est 43 et que L contient, dans cet ordre, des paquets dont les numéros de séquence sont 55, 50, 46, 45, 44, cette fonction affichera alors P et les paquets 44 à 46 (dans cet ordre) ; elle retournera 46 et la liste L ne contiendra alors plus que les paquets 55 et 50 (dans cet ordre).

```
unsigned int tcp_receive(paquet const& p, unsigned int i, vector<paquet>& L)
{
  if (p.aquit) return i;

  paquet ack(p);
  ack.aquit = true;
  send(ack);

  if (p.seq == i + 1) {
    affiche_paquet(p);

    size_t j(0);
    ++i;
    while (not L.empty() and (L.back().seq == i + 1)) {
      affiche_paquet(L.back());
      L.pop_back();
      ++i;
    }
  } else {
    insert(L, p);
  }

  return i;
}
```

Question 3 – Mélange de mots [12 points]

Question 3.1 [3 points]

Quelle est l'entropie (telle que définie en cours) d'un mot X constitué de n fois la lettre `'F'` et m fois la lettre `'G'` : « $\underbrace{\texttt{F}\ldots\ldots\texttt{F}}_{n}\underbrace{\texttt{G}\ldots\ldots\texttt{G}}_{m}$ » ? Soit $p = \frac{n}{n+m}$. Utilisez p pour exprimer cette entropie comme une fonction de p uniquement :

$$H(X) = h(p) = = -\frac{n}{n+m}\ \log_2(\frac{n}{n+m}) - \frac{m}{n+m}\ \log_2(\frac{m}{n+m}) = p\ \log_2(\frac{1}{p}) + (1-p)\ \log_2(\frac{1}{1-p})$$

Indication : que vaut $1-p$?

Justifiez votre réponse simplement en explicitant le calcul effectué.

Question 3.2 [9 points]

Soient A un mot de longueur n et B un mot de longueur m n'ayant *aucune* lettre en commun avec A. Démontrez que l'entropie (telle que définie en cours) du mot « AB » , constitué du mot B collé derrière le mot A vaut

$$H(AB) = h(p) + p\,H(A) + (1-p)\,H(B)$$

où $p = \frac{n}{n+m}$ et $H(A)$, $H(B)$ sont respectivement l'entropie du mot A et du mot B (telle que définie en cours).

Indication : imaginez le calcul de $H(A)$, celui de $H(B)$ et faites le calcul de $H(AB)$.

Soient x_i, $1 \leq i \leq a$, les différentes lettres de A et y_i, $1 \leq i \leq b$, les différentes lettres de B ; et soient q_i et r_i leur compte (nombre d'apparitions) respectif.

Aucun des x_i n'étant un y_i, l'entropie de AB s'écrit donc (en décomposant sa définition sur les deux ensembles disjoints de x_i et de y_i) :

$$H(AB) = -\sum_{i=1}^{a} \frac{q_i}{n+m} \log(\frac{q_i}{n+m}) - \sum_{i=1}^{b} \frac{r_i}{n+m} \log(\frac{r_i}{n+m})$$

que l'on réécrit comme

$$H(AB) = -\frac{n}{n+m} \sum_{i=1}^{a} \frac{q_i}{n} \left(\log(\frac{q_i}{n}) + \log(\frac{n}{n+m}) \right) - \frac{m}{n+m} \sum_{i=1}^{b} \frac{r_i}{m} \left(\log(\frac{r_i}{m}) + \log(\frac{m}{n+m}) \right)$$

c.-à-d.

$$H(AB) = -p \sum_{i=1}^{a} \frac{q_i}{n} \log(\frac{q_i}{n}) - p \sum_{i=1}^{a} \frac{q_i}{n} \log(p) - (1-p) \sum_{i=1}^{b} \frac{r_i}{m} \log(\frac{r_i}{m}) - (1-p) \sum_{i=1}^{b} \frac{r_i}{m} \log(1-p)$$

Comme $H(A) = -\sum_{i=1}^{a} \frac{q_i}{n} \log(\frac{q_i}{n})$, $H(B) = -\sum_{i=1}^{b} \frac{r_i}{m} \log(\frac{r_i}{m})$, $\sum_{i=1}^{a} \frac{q_i}{n} = 1$ et $\sum_{i=1}^{b} \frac{r_i}{m} = 1$, on arrive à la formule proposée.

Question 4 – Compréhension de programme [16 points]

L'extrait de programme suivant ne contient pas d'erreur :

```
bool f(double a, vector<double> const& v, size_t i = 0, double prec = 1e-10)
{
    return (i < v.size()) and
           ( (abs(a - v[i]) < prec)  or  f(a, v, i+1, prec) );
}

vector<double> g(vector<double> const& v1, size_t i, vector<double> const& v2)
{
    vector<double> res;
    if (i == 0) return res;
    --i;
    res = g(v1, i, v2);
    if ((i < v1.size()) and not f(v1[i], v2)) {
        res.push_back(v1[i]);
    }
    return res;
}

vector<double> g(vector<double> const& v1, vector<double> const& v2)
{
    return g(v1, v1.size(), v2);
}
```

Question 4.1 [6 points]

Expliquer *brièvement* ce que font chacune des fonctions `f` et `g` et proposer un nom plus pertinent pour chacune d'entre elles.

`f` teste récursivement si `a` est contenu, à une précision `prec`, dans `v` à partir de la position `i`. `i` prend la valeur `0` par défaut (recherche depuis le début de `v`) et `prec` la valeur 10^{-10} par défaut.
On pourrait la nommer par exemple `appartient()`, `est_dans()`, `contient()`, `est_contenu()`, `possede()`, ...

Il y a deux fonctions `g` (ce qui est possible grace à la surchage), la seconde n'étant que la généralisation à `v1` tout entier de la première.

La première fonction `g` retourne la soustraction ensembliste de `v2` à « `v1` jusqu'au rang `i` exclu » ; c.-à-d. qu'elle retourne la liste des éléments de `v1`, de son début jusqu'au rang `i` exclu, qui ne sont pas contenus dans `v2` (à la précision 10^{-10} près).

On pourrait la nommer par exemple `supprime()`, `soustrait()`, `difference()`, ...

Question 4.2 [10 points]

Si la complexité de `size()` et de `push_back()` pour un `vector` sont en $\mathcal{O}(1)$, quelle est la complexité de l'algorithme équivalent à la fonction `g` en fonction de la taille de `v1` ? en fonction de la taille de `v2` ? Justifiez vos réponses.

« La » fonction `g` ne présente plus d'ambiguité dans cette question du moment que l'on a compris le rôle de `i` (cf sous-question précédente) : il n'a de sens dans cette question que s'il représente la taille de `v1` (pire cas).

Ayant à faire à des fonctions récursives, il faut en toute rigueur commencer par vérifier qu'elles terminent bien dans tous les cas. C'est bien le cas de `f` car soit `i` est plus grand que la taille de `v` et `f` termine immédiatement, soit l'on rappelle `f` en augmentant `i` de `1`, qui finira bien par atteindre la taille de `v`.

`g` se termine également, par un argument similaire : soit `i` vaut `0` et on s'arrête immédiatement, soit on rappelle `g` en décrémentant `i` (lequel est forcément positif).

Pour calculer la complexité de `g`, nous avons besoin de celle de `f`, laquelle est en $\mathcal{O}(m)$ avec m la taille de `v`, puisque l'on réappelle `f` à chaque fois avec un élément de moins (et que le reste est en $\mathcal{O}(1)$).

Comme `g` ne fait pas d'autre usage de `v2` que de le passer à `f` (sans le modifier), la complexité de `g` par rapport à `v2` est donc en $\mathcal{O}(m)$ avec m la taille de `v2`.

Vis à vis de `v1`, `g` seule suffit (l'appel à `f` ne dépend que de `v2` : il est $\mathcal{O}(1)$ vis à vis de `v1`) et l'on se convainc facilement de façon similaire à `f` (ou l'on démontre à l'aide de l'écriture récursive d'une suite arithmétique) que la complexité de `g` par rapport à `v1` est en $\mathcal{O}(n)$ avec n la taille de `v1`.

Question 5 – La cache est déterminante [19 points]

On considère l'extrait de programme suivant (que l'on suppose correct) :

```
double det2(double a, double b, double c, double d) {
    return a*d - b*c;
}
double det3(array<array<double, 3>, 3> const& mat) {
    return det2(mat[1][0], mat[1][1], mat[2][0], mat[2][1]) * mat[0][2]
         - det2(mat[2][1], mat[2][0], mat[0][1], mat[0][0]) * mat[1][2]
         + det2(mat[0][0], mat[0][1], mat[1][0], mat[1][1]) * mat[2][2];
}
```

On suppose que les `double` utilisent 2 mots mémoire, que la cache a deux blocs de 4 mots chacun et que `matrice` est un tableau 3x3 stocké par lignes : si la valeur (en mot) de `&(matrice[0][0])` est a, alors la valeur (en mot) de `&(matrice[0][1])` est $a+2$ et `&(matrice[1][0])` est $a+6$.

Le but de cette question est de calculer le nombre de défauts de cache lors du calcul de `det3(matrice)`.

Question 5.1 Schéma mémoire [7 points]

Dessiner la cache (sans valeur) et les éléments en mémoire avec la convention qu'un rectangle correspond à un mot. Écrivez simplement M_{10} pour « `matrice[1][0]` », etc. Donnez également les adresses (p.ex. « $a+6$ » au dessus de quelques cases pertinentes). La ligne « accès : » au dessous de la case n'est utile que pour la question suivante.

Cache :

Mémoire (off-chip) :

adresse :	a	$a+1$	$a+2$	$a+3$	$a+4$	$a+5$	$a+6$	$a+7$	$a+8$	$a+9$
contenu :	M_{00}	M_{00}	M_{01}	M_{01}	M_{02}	M_{02}	M_{10}	M_{10}	M_{11}	M_{11}
accès :	9, 11		8, 12		5		1, 13		2, 14	

adresse :	$a+10$	$a+11$	$a+12$	$a+13$	$a+14$	$a+15$	$a+16$	$a+17$		
contenu :	M_{12}	M_{12}	M_{20}	M_{20}	M_{21}	M_{21}	M_{22}	M_{22}		
accès :	10		3, 7		4, 6		15			

Question 5.2 Ordre d'appel [4 points]

En supposant que les expressions sont évaluées dans l'*ordre de lecture* du code C++ précédent, indiquez ci-dessus (dans la ligne « accès ») l'ordre d'accès (de 1 à 15) aux différents éléments mémoire. Mettez simplement le ou les numéros d'ordre d'accès sous chaque case mémoire correspondante.

Par exemple, si la donnée que vous avez mise dans la 3e case ci-dessus est la 5e et la 8e à être accédée lors du calcul de l'appel `det3(matrice)`, mettez simplement « 5, 8 » sous la 3e case ci-dessous.

Question 5.3 Nombre de défauts de cache [8 points]

En supposant que l'adresse a =`&(matrice[0][0])` est alignée sur les blocs de la cache,

1. marquez (clairement) les accès mémoire qui provoquent un défaut de cache parmi les 15 accès mémoire effectués pour l'appel `det3(matrice)` :

   1 **2** **3** 4 **5** 6 7 **8** 9 **10** 11 12 **13** **14** **15**

   expliquez/justifiez votre réponse ;
2. concluez en donnant le nombre total de défauts de cache.

Les deux premiers accès sont des défauts de cache car on suppose que la cache ne contient rien de `matrice` au départ. Ensuite, la séquence d'accès indiquée à la question précédente montre facilement lorsque deux blocs de 4 mots sont accédés successivement et donc en cache (qui ne peut avoir que 2 blocs de 4 mots).

Typiquement le 4e accès ne fait pas de défaut de cache puisque continue en mémoire avec le 3e accès.

Par ailleurs, la cache ayant deux bloc, tout bloc ayant des accès à moins de 2 ne font pas non plus de défaut de cache. Par exemple, les blocs aux adresses $a + 12$ à $a + 15$ ne font pas de défaut de cache aux accès 6 et 7.

Cela fait donc **9** défauts de cache au total comme indiqué ci-dessus.

Question 6 – Chercher les erreurs [14 points]

Considérer le programme C++ suivant (sur deux pages) :

```
1   #include <iostream>
2   #include <string>
3   #include <vector>
4   #include <array>
5   #include <fstream>
6   using namespace std;
7
8   typedef vector<string> Texte;
9
10  // ============================================================================
11  string encodage(Texte const& en_clair)
12  {
13    const array<string, 16> code({
14        "01",   "110",   "101",   "1111",
15        "0100", "1010",  "00101", "1110",
16        "00",   "0011",  "11001", "111000",
17        "0101", "11"  ,  "01011", "101010"
18      });
19
20    string resultat;
21    for (auto mot : en_clair) {
22      for (auto c : mot) {
23        while (c > 16) {
24          resultat += code[c % 16];
25          c -= 16;
26        }
27        resultat += code[c];
28      }
29    }
30    return resultat;
31  }
32
33  // ============================================================================
34  Texte lire_fichier(string const& nom)
35  {
36    Texte retour;
37    ifstream fichier(nom);
38    string mot;
39    size_t i(0);
40    while (fichier << mot) {
41      retour[i] = mot;
42      i += 1;
43    }
44    return retour;
45  }
46
```

```
47  // ========================================================================
48  string demander_nom_fichier()
49  {
50    string retour("test.txt");
51    cout << "Entrez un nom de fichier : ";
52    getline(cin, retour);
53    return retour;
54  }
55
56  // ========================================================================
57  int main()
58  {
59    cout << encodage(lire_fichier(demander_nom_fichier()))
60         << endl;
61    return 0;
62  }
```

Question 6.1 – But général [2 points]

Bien que ce programme contienne plusieurs erreurs, indiquer (*brièvement* mais *clairement*) le but et le fonctionnement général attendu de ce programme (c.-à-d. l'intention du programmeur).

Ce programme encode le contenu d'un fichier dont le nom est entré par l'utilisateur en encodant chaque caractères par une séquence de mots de codes fixés au départ.

Question 6.2 – Compilation [3 points]

Ce programme ne compile pas correctement. Le compilateur n'indique qu'une seule erreur :

```
pgm.cc:40:18: error: no match for ‘operator<<’ (operand types are ‘std::ifstream’
              {aka ‘std::basic_ifstream<char>’} and ‘std::__cxx11::string’
              {aka ‘std::__cxx11::basic_string<char>’})
```

Que cela signifie-t-il ? (quelle est l'erreur ?) Comment corriger ?
Répondez sur le code lui-même en marquant votre reponse d'un « Question 6.2 ».

Le signe de lecture >> doit être utilisé ligne 40.

Question 6.3 – Analyse critique [9 points]

Ce programme comporte encore au moins quatre autres erreurs de différente nature (théorie, conception, implémentation, mauvaises pratiques, etc.)

Indiquez ces erreurs puis corrigez les.
Vous pouvez répondre sur le code lui-même (sans autre marque) ou sur la page au dos.

Attention ! Il y aura des pénalités pour des indications d'erreur qui n'en sont pas vraiment (mais seront tolérents sur les mauvaises pratiques).

Ce programme présente plusieurs autres problèmes en plus de celui mentionné à la question précédente :

— théorie : le code utilisé dans la fonction `encodage()` est un code ambigu : plusieurs mots de code sont préfixe des autres ;

— ligne 23 ou ligne 27 : attention au cas particulier `16` : soit mettre `>=` en ligne 23, soit mettre `c % 16` ligne 27 ;
— ligne 37 (ou 38) : il manque un test d'ouverture correcte du fichier
— ligne 41 : il faut faire des `push_back()` ici, `resultat[i]` n'existe pas !
— mauvaise pratique : la valeur `16` est répétée plusieurs fois explicitement dans la fonction `encodage()` ; il faudrait utiliser `code.size()` à la place.