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, section PH
Chappelier J.-C.

Information, Calcul et Communication (SPH) : Correction de l'Examen final

20 décembre 2024

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**.

   Si nécessaire, il y a une page de réponse supplémentaire en fin de copie (page 20).

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(e) des assistant(e)s.

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 128 points).

   Tous les exercices comptent pour la note finale.

Question 1 – Divers [25 points]

① **[4 points]** Qu'affiche le code ci-dessous ?

Justifiez votre réponse (vous pouvez aussi annoter le code).

```
#include <iostream>
using namespace std;

double f(double x) {
  if (x < 0.0) throw -1.2;
  return x + 3.4;
}

double g(double x = -5.6) {
  try {
    return f(x);
  }
  catch(double y) {
    return 7.8 + y;
  }
}

int main()
{
  try {
    const double u( g() );
    cout << u << endl;
  }
  catch (double x) {
    cout << x << endl;
  }
  return 0;
}
```

Réponse et justification :

Il affiche « `6.6` ».

Tout d'abord l'appel de `g()` utilise la valeur par défaut -5.6, qui, donc, appelle `f()` avec cette valeur.

`f()` `throw` alors la valeur -1.2, qui est attrapée par le `catch` de `g()`.

`g()` *retourne* donc $7.8 + -1.2 = 6.6$.

Le `try` du `main()` n'attrape rien puisqu'aucune exception n'est lancée par `g()`.

En utilisant RSA, vous souhaitez transmettre, de façon authentifiée, une information à un ami.

La clé publique de votre ami est $(295, 2773)$.

Votre clé publique est $(1241, 1679)$.

② **[3 points]** Quelle est votre clé privée, sachant que : $1679 = 23 \times 73$, $22 \times 72 = 1584$,
$1584 \times 720 = 1 \mod 1241$, $1241 \times 665 = 1 \mod 1584$,
$1241 \times 1241 = 433 \mod 1584$, $1241 \times 23 = 17 \times 1679$,
$1241 \times 73 = 1606 \mod 1679$?

Justifiez *brièvement* votre réponse.

Notre clé privée d est celle qui vérifie $e \times d = 1 \mod m$ avec $e = 1241$ et $m = 1584$ $((23-1)\times(73-1))$, donc $d = 665$.

Commentaire : Beacoup trop travaillent modulo n ici.

③ **[7 points]** Le message de départ que vous voulez envoyer est `100001001` (en binaire). Quel message lui envoyez vous ?
Exprimez votre réponse soit en binaire, soit sous la forme « $x^y \mod z$ » avec x, y et z en base 10 et **justifiez** pleinement votre réponse.

Réponse et justification :

Tout d'abord, `100001001` représente 265 en binaire (ici forcément convention de nombres entiers **positifs** puisque l'on fait de l'arithmétique modulaire modulo n).

Ensuite, pour authentifier nos messages (c.-à-d. les signer) on utilise notre clé *privée* (665). On envoie donc :

$$265^{665} \mod 1679$$

Ceci dit, vu le cadre décrit ici, le destinataire n'a visiblement aucune chance de savoir quel était vraiement le message de départ. Il semble donc nécessaire ici d'envoyer aussi le message en clair (sinon n'importe quoi plus petit que 1679 (connue) pourrait être envoyé par n'importe qui en prétendant que c'est un message signé par nous...). Il semble donc important ici aussi d'envoyer le message en clair.
Par contre, aucune confidentialité n'est demandée ici. Il n'y a donc pas besoin de crypter pour ça.

Commentaire : Beaucoup chiffrent pour de la confidentialité au lieu de signer.
Et très peu pensent à envoyer le message en clair.

④ **[2 points]** Combien de bits au maximum contiendra le message que vous envoyez (encodé en binaire simplement comme des entiers) ?

Dans le cadre décrit ici, les messages que nous envoyons sont soit en clair, soit signés (mais pas encryptés pour confidentialité). Mais s'ils sont signés, ils sont alors compris entre 0 et $1679 - 1$ (notre n).

On aura donc besoin d'au plus **11 bits** : $1023 < 1678 < 2047$. (Et l'on pourra toujours découper les messages « en clair » (non signés/encryptés) en tranches de 11 bits.)

Commentaire : Plusieurs prennent la borne basse (10 bits) : comment alors représenter les valeurs entre 1024 et 1678 ?

⑤ **[2 points]** Dans un jeu de plateau, un monstre a un score d'attaque qui peut prendre toutes les valeurs entières entre -10 et 10 inclus. Combien de bits sont nécessaires pour représenter ce score ? En supposant qu'on utilise ces bits avec une représentation en complément à 2, comment représente-t-on la valeur d'attaque -6 ?
Justifiez vos réponses.

Réponses et justifications :

Il y a 21 valeurs possibles pour les scores d'attaque. On a donc besoin de $\lceil \log_2(21) \rceil = 5$ bits.

Si on utilise ces 5 bits en complément à 2, on représente 6 en binaire comme `00110`. On inverse tous les bits, ce qui donne `11001`, puis on ajoute 1, et on obtient `11010`.

Alternativement, on peut calculer $2^5 - 6$ et représenter ce nombre sur 5 bits non signés, ce qui revient au même en exploitant l'arithmétique modulaire du complément à 2.

⑥ **[3 points]** Si les monstres ont également un score de « bonus aérien » compris entre 1 et 10 inclus, de combien de bits a-t-on besoin pour représenter les scores d'attaque *et* de bonus d'un monstre ?
Justifiez vos réponses.

Réponses et justifications :

On a désormais 21×10 (ensemble des couples = produit cartésien des deux ensembles), soient 210 valeurs possibles. Pour cela, nous avons besoin de 8 bits.

On aurait pu être tenté d'encoder séparément le score d'attaque sur 5 bits et le bonus aérien sur 4 bits, mais cela donne un résultat plus grand que nécessaire. Or on est totalement *libre* de choisir comment on souhaite coder ces 21×10 valeurs.

Commentaire : Beaucoup trop s'imposent un codage sous-optimal (concaténation) au lieu de simplement calculer le nombre de bits requis pour représenter K informations. La formule « nombre de bits = $\lceil \log_2 K \rceil$ » ne semble donc vraiment pas acquise en pratique (déjà vu au premier examen) :-(.

⑦ **[4 points]** Lorsqu'un monstre attaque depuis les airs, on *multiplie* son score d'attaque et son bonus aérien. Considérez un monstre avec un score d'attaque de -8 et un bonus aérien de 9. Si on calcule -8×9 en complément à 2 avec le nombre de bits que vous avez utilisé en ⑤, quel résultat obtient-on ?

Qu'en est-il si on utilise le nombre de bits que vous avez utilisé en ⑥ ?

Exprimez vos réponses **en décimal** et **justifiez-les**.

Réponse et justification :

La réponse mathématique est $-8 \times 9 = -72$. Sur 5 bits, on cherche donc $r \in [-16, 15]$ tel que $r \equiv -72 \pmod{32}$ et on obtient $r = -72 + 2 \cdot 32 = -8$.

Sur 8 bits, on peut correctement stocker -72, et c'est donc le résultat.

Commentaire : Beaucoup ne font pas l'effort de calculer la valeur effectivement représentée en cas de dépassement de capacité.

Question 2 – Manipulations de listes [18 points]

On s'intéresse au problème suivant : soit une liste de valeurs considérées deux à deux ; on souhaite regrouper (sommer) toutes les secondes valeurs qui sont juste après la même première valeur.
On s'intéresse donc à des listes de la forme $(a_1, x_1, a_2, x_2, \cdots, a_n, x_n)$ où l'on souhaite regrouper tous les a_i identiques en sommant leurs x_i. De telles listes ont donc forcément un nombre pair d'éléments.

Par exemple, à partir de la liste $\Big($ 3,5, 1, 8, 5, 2, 1, −2, 2, 1, 3,2 $\Big)$, on souhaite produire la liste $(3, 7, 1, 6, 5, 2, 2, 1)$: après 3 apparaît la somme des valeurs 5 et 2 qui sont les valeurs apparaissant partout après 3 dans la première liste ; de même, après 1 apparaît la somme des valeurs 8 et −2 qui sont les valeurs apparaissant partout après 1 dans la première liste, etc.
À noter que la première des deux valeurs (a_i) peut apparaître plus que deux fois dans la liste de départ. Par exemple, à partir de la liste $(5, 3, 5, 2, 5, 1, 5, 3)$, on produira la liste $(5, 9)$ (car $3+2+1+3=9$).

Il n'est pas nécessaire que les premières valeurs (a_i) de la liste produite comme solution apparaissent dans le même ordre que dans la liste d'origine. Ainsi dans le premier exemple, les listes $(3, 7, 1, 6, 5, 2, 2, 1)$, $(5, 2, 1, 6, 2, 1, 3, 7)$, $(1, 6, 2, 1, 3, 7, 5, 2)$ ou $(3, 7, 5, 2, 1, 6, 2, 1)$ (par exemples) sont toutes aussi acceptables comme solution au problème considéré.

① **[12 points]** Écrivez un algorithme *récursif* résolvant le problème décrit ci-dessus.

Réponse :

Il y a plein de façons d'écrire un tel algorithme. Voici un premier algorithme, simple :

```
algoV1
entrée : liste L telle que décrite
sortie : liste L' « comprimée » comme indiquée
    n ⟵ taille(L)
    Si n ≤ 2
    |   Sortir : L
    L' ⟵ algoV1(L[1], ··· , L[n − 2])
    n' ⟵ taille(L')
    Pour i de 1 à n' − 1 de 2 en 2
    |   Si L[n − 1] = L'[i]
    |   |   L'[i + 1] ⟵ L'[i + 1] + L[n]
    |   |   Sortir : L'
    Sortir : L' ⊕ (L[n − 1], L[n]) // ajout en fin de liste
```

Note : on peut aussi utiliser l'algorithme de recherche (linéaire, liste non-ordonnée), mais il faut alors bien préciser la convention de recherche (indices impairs) et celle de retour (pour pouvoir récupérer le bon indice ou une indication claire (p.ex. 0) que l'on n'a pas trouvé).

Voici une autre solution, aussi assez simple :

algoV2

entrée : *liste L telle que décrite*
sortie : *liste « comprimée » comme indiquée*

$n \longleftarrow$ **taille**(L)
Si $n \leq 2$
  Sortir : L
$L' \longleftarrow ()$ *// liste vide*
Pour i de 3 à $n-1$ de 2 en 2
  Si $L[1] = L[i]$
    $L[2] \longleftarrow L[2] + L[i+1]$
  Sinon
    $L' \longleftarrow L' \oplus (L[i], L[i+1])$
Sortir : $(L[1], L[2]) \oplus$ **algoV2**(L')

Voici enfin une autre solution, plus sophistiquée :

algoV3

entrée : *liste L telle que décrite*
sortie : *liste L' « comprimée » comme indiquée*

$n \longleftarrow$ **taille**(L)
Si $n \leq 2$
  Sortir : L
Sortir : algo_merge$\Big($**algoV3**$\big(L[1 : 2\lfloor \frac{n}{4} \rfloor]\big)$, **algoV3**$\big(L[2\lfloor \frac{n}{4} \rfloor + 1 : n]\big)\Big)$

algo_merge

entrée : L_1, L_2, *deux listes de nombres telles que décrites, mais triées sur les premières des deux valeurs*
sortie : *liste L' fusion* ***triée*** *« comprimée » de L_1 et L_2*

$n_1 \longleftarrow$ **taille**(L_1)
Si $n_1 = 0$
  Sortir : L_2
$n_2 \longleftarrow$ **taille**(L_2)
Si $n_2 = 0$
  Sortir : L_1
Si $L_1[1] < L_2[1]$
  Sortir : $(L_1[1], L_1[2]) \oplus$ **algo_merge**$\big(L_1[3 : n_1],\ L_2\big)$
Si $L_1[1] > L_2[1]$
  Sortir : $(L_2[1], L_2[2]) \oplus$ **algo_merge**$\big(L_1,\ L_2[3 : n_2]\big)$
Sortir : $(L_1[1], L_1[2] + L_2[2]) \oplus$ **algo_merge**$\big(L_1[3 : n_1],\ L_2[3 : n_2]\big)$

où $L_1(a : b)$ désigne la sous-liste $(L(a), \cdots, L(b))$ si $a \leq b$, et la liste vide sinon.

Commentaire : Encore énormément de difficultés à écrire des algorithmes propres ; en particulier la gestion de la récursion :

— les cas terminaux sont souvent oubliés / mal gérés ; typiquement le cas de la liste vide n'est pas toujours traité, alors même que la récursion écrite finira par faire un appel avec une liste vide ;

— utilisation de listes locales non initialisées/déclarées ;
— oubli du « **Sortir** » pour utiliser le résultat de l'appel récursif ;
— oubli d'utiliser (typiquement concaténer) la partie laissée de cotée avant l'appel récursif (typiquement les deux premiers ou les deux derniers éléments de la liste) ;
— utilisation de la notation $\ominus$ sans la définir (alors que l'on a insisté plusieurs fois en séances sur l'*ambiguïté* d'une telle notation : confusion entre *valeur* et *indice*).
— utilisation en entrée de variables auxiliaires (non conforme au problème de départ), sans utiliser un second algorithme qui, lui, répondrait au problème posé et, surtout, initialiserait ces variables auxiliaires.

② **[6 points]** Quelle est la complexité de votre algorithme proposé en ① ? **Justifiez** votre réponse.

La complexité dépend bien sûr de l'algorithme proposé.

Les deux premiers algorithmes proposés sont en $\Theta(n^2)$, n étant la taille de la liste en entrée.
Le pire cas est lorsqu'il n'y a rien à changer à la liste. On fera alors à chaque fois la boucle de recherche ($\Theta(n)$), laquelle ne contient que des opérations élémentaires. On fait cela $n/2$ fois (on enlève à chaque fois 2 éléments à la liste pour l'appel récursif).
Formellement :

$$C(n) = a + C(n-2) + b\,(n-2)$$

(en supposant **taille**() en $\Theta(1)$).

Le troisième algorithme est en $\Theta(n \log n)$. C'est l'adaptation à ce problème de l'algorithme « tri fusion » (merge-sort). La justification de cette complexité peut se faire par l'arbre des appels ($\log n$ étages, complexité $\Theta(n)$ à chaque *étage* de l'arbre), ou formellement :

$$D(n) = a + 2\,D(n/2) + M(n)$$

avec $D()$ la complexité de l'algorithme proposé et $M()$ celle de l'algorithme « merge » (où n représente bien la taille totale de son entrée, somme des tailles de ses *deux* listes).
M est en $\Theta(n)$: tout l'algorithme est en $\Theta(1)$, sauf l'appel récursif. Cet appel récursif ne se produit qu'une seule fois à chaque appel (en raison du « **Si** ») et est lancé sur une entrée de taille $n-2$ (dans le pire cas).

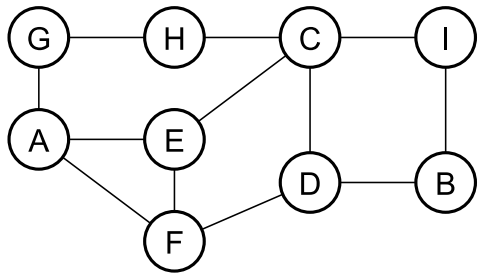
Commentaire : Un oubli fréquent ici était de préciser le pire cas (lorsque c'était pertinent). Typiquement, la plupart des algorithmes proposés ne font pas *systématiquement* toute la boucle, mais uniquement dans le pire des cas.

Un autre oubli est de ne pas préciser que la boucle ne fait que des opérations élémentaires (ce n'est pas parce qu'une boucle tourne n fois que sa complexité est en $\Theta(n)$: encore faut-il qu'à l'intérieur il n'y ait que du $\Theta(1)$).

Question 3 – Entre amis [32 points]

Note : (comme toujours, mais particulièrement ici) nous vous conseillons, afin de faire les bons choix, de lire *entièrement* cette question avant de commencer.

Alice, **B**ob et **C**arole sont trois amis qui souhaitent s'échanger des informations en utilisant le protocole TCP/IP. Ils font partie du réseau informatique suivant où chaque noeud représente un routeur :



- **A**lice
- **B**ob
- **C**arole
- **D**avid
- **È**ve
- **F**rance
- **G**eorges
- **H**ugues
- **I**nès

① **[4 points]** Donnez les lignes correspondant à **A**lice et **B**ob dans la table de routage de **C**arole.

Justifiez votre réponse en annotant (simplement) le graphe ci-dessus.

Réponse :

Puisqu'il y a deux chemins de longueur 2 entre **C**arole et **B**ob, il y a plusieurs réponses possibles. Donner l'une ou l'autre ou les deux voisins suivants possibles sont des solutions valables. Par contre, pour aller chez **A**lice, le seul plus court chemin est *via* **È**ve.

Destination	Voisin suivant	Longueur
Alice	**È**ve	2
Bob	**D**avid ou **I**nès	2

② **[2 points]** On suppose qu'il existe un type de données C++ `AdresseIP` qui représente une adresse IP (si cela vous aide, vous pouvez considérer qu'il s'agit d'un entier non signé). Définissez, en C++, un type de données `TableRoutage` qui permet de stocker la table de routage d'un nœud.

Expliquez brièvement vos choix d'implémentation.

Réponse :

```
struct LigneRoutage {
  AdresseIP destination;
  AdresseIP voisinSuivant;
  unsigned int distance;
};

typedef vector<LigneRoutage> TableRoutage;
```

Note : la solution consistant à faire une `struct` avec trois tableaux est très mauvaise car elle oblige ces trois tableaux à avoir la même taille et à garantir que leurs informations sont rangées strictement dans le même ordre. Cette façon de structurer les données (tableaux « synchronisés ») est vraiment à proscrire.

Commentaire : En plus de l'erreur mentionnée par la note ci-dessus : pourquoi ne pas utiliser `AdresseIP` comme un type en tant que tel au lieu d'en faire un `unsigned int` ou un `char` ou une `string` (rien ne dit que ce soit ça) ?

Une autre erreur trop fréquente consiste à confondre table et ligne de la table.

③ **[4 points]** On suppose qu'il existe une fonction C++

```
void envoieSurLien(AdresseIP voisin, AdresseIP destination, string const& paquet);
```

que vous ne devez pas implémenter. Elle transfère au nœud `voisin` direct donné, *via* la couche de lien, un `paquet` à destination de `destination`.

Implémentez, en C++, la fonction `envoieSurReseau()` qui effectue le transfert d'un paquet au niveau de la couche réseau. Elle reçoit la table de routage du nœud courant (telle que vous l'avez définie à la sous-question précédente), un destinaire et un paquet. Elle doit transférer le paquet au prochain nœud via la couche de lien. On suppose que le destinaire du paquet n'est *pas* le nœud courant.

Conseil : n'oubliez pas qu'il vous est toujours possible de définir des fonctions annexes. Dans ce cas, vous devez bien sûr fournir l'implémentation de celles-ci.

Réponse :

```
void envoieSurReseau(TableRoutage const& table, AdresseIP destination,
    string const& paquet) {
  // Chercher la destination dans la table
  for (const auto& ligne : table) {
    if (ligne.destination == destination) {
      // Quand on l'a trouvée, transférer le paquet au voisin suivant
      envoieSurLien(ligne.voisinSuivant, destination, paquet);

      // Optionnel : éviter de chercher dans le reste de la table
      return;
    }
  }
}
```

Commentaire : Vos fonctions C++ doivent être robustes ; ici par exemple se comporter correctement (ne rien faire) même si `voisin` n'est pas trouvé dans la table de routage.

Et même commentaire que précédemment concernant le type `AdresseIP`.

④ **[15 points]** On suppose qu'il existe une fonction C++

```
void envoieDonneesRoutageSurLien(
    AdresseIP voisin, AdresseIP destination, unsigned int distance);
```

que vous ne devez pas implémenter. Elle communique un paquet de données de routage – les paquets qui établissent les tables de routage, pas ceux qui sont réellement envoyés pour le compte des couches supérieures – au `voisin` direct donné, concernant une `destination` et une `distance` donnée.

Implémentez, en C++, la fonction `recoitDonneesRoutageDepuisLien()` qui applique la *réception* d'un tel paquet de données de routage. Elle reçoit l'adresse du voisin direct qui a envoyé le paquet, ainsi que les informations de `destination` et `distance`. Elle reçoit également la table de routage du nœud courant, ainsi qu'une liste des adresses IP de tous les voisins directs.

Par exemple, si **F**rance appelle `envoieDonneesRoutageSurLien()` vers son voisin **D**avid, alors votre fonction s'exécutera chez **D**avid en recevant l'adresse IP de **F**rance comme voisine, comme table de routage celle de **D**avid, et la liste des adresses IP de **B**ob, **C**arole et **F**rance (les voisins de **D**avid).

Votre fonction doit appliquer les changements nécessaires à la table de routage et/ou utiliser la fonction `envoieDonneesRoutageSurLien()`.

Réponse :

```
void recoitDonneesRoutageDepuisLien(TableRoutage& table,
    vector<AdresseIP>& voisins, AdresseIP voisinSource,
    AdresseIP destination, unsigned int distance) {

  // On met potentiellement à jour la table
  if (mettreAJourTable(table, destination, voisinSource, distance)) {
    /* Si quelque chose a changé, c'est que cette route est la nouvelle
     * meilleure. Il faut la partager avec nos voisins directs.
     * Il faut ajouter 1 à la distance pour prendre en compte ce noeud.
     */
    envoieATousLesVoisins(voisins, destination, distance + 1);
  }
}

/* Met à jour la table avec une nouvelle information de routage.
 * S'il y a déjà une entrée dans la table pour `destination` de
 * distance inférieure ou égale à `distance`, alors cette fonction
 * ne fait rien et renvoie `false`.
 * Sinon, elle met à jour l'entrée ou en crée une nouvelle, puis
 * renvoie `true`.
 */
bool mettreAJourTable(TableRoutage& table, AdresseIP destination,
    AdresseIP voisinSuivant, unsigned int distance) {
  // D'abord on cherche s'il y a déjà une entrée pour la destination
  for (auto& ligne : table) {
    if (ligne.destination == destination) {
      // Si oui, on la met à jour seulement si on a une meilleure route
      if (distance < ligne.distance) {
        ligne.voisinSuivant = voisinSuivant;
        ligne.distance = distance;
        return true; // on a modifié quelque chose
      } else {
        // On ne change rien ; il faut retourner `false` ici
        return false;
      }
    }
  }

  // Si on n'avait pas encore d'entrée pour destination, on ajoute une ligne
  LigneRoutage nouvelleLigne { destination, voisinSuivant, distance };
  table.push_back(nouvelleLigne);
  return true;
}

/* Envoie un paquet de méta-données à tous les voisins directs donnés.
 * Idéalement, on pourrait s'épargner d'envoyer l'information au voisin
 * qui vient de nous contacter. Mais ce n'est pas très important.
 */
void envoieATousLesVoisins(vector<AdresseIP> const& voisins,
    AdresseIP destination, unsigned int distance) {
  for (auto voisin : voisins) {
    envoieDonneesRoutageSurLien(voisin, destination, distance);
  }
}
```

⑤ **[3 points]** **È**ve aime **É**couter aux portes. En particulier, elle raffole des ragots qui concernent **A**lice, **B**ob et **C**arole.

Étant donné le réseau défini plus haut, **È**ve est-elle en mesure « d'écouter » les messages que s'envoient **A**lice, **B**ob et **C**arole ? Répondez par **oui**, **non** ou **peut-être** dans chaque case. « peut-être » signifie que les informations données ne permettent pas de décider.
Attention : les réponses laissées blanches valent 0, mais les réponses fausses valent $-1/4$.

Réponses :

Source du message	Destinataire	Votre réponse
de **A**lice	à **B**ob	non
de **B**ob	à **A**lice	non
de **A**lice	à **C**arole	oui
de **C**arole	à **A**lice	oui
de **B**ob	à **C**arole	non
de **C**arole	à **B**ob	non

⑥ **[4 points]** Si **È**ve devient réellement malhonnête, elle voudra peut-être tenter *d'influencer* les autres routeurs du réseau. Proposez *une* modification *de nature technologique* qu'**È**ve peut implémenter pour lui permettre de mieux espionner **A**lice, **B**ob et/ou **C**arole.

Votre proposition doit permettre de transformer au moins un « non » ou « peut-être » de votre réponse précédente en un « oui ».

On peut supposer qu'**È**ve a connaissance du réseau complet, mais qu'elle ne peut pas changer les connexions physiques.

Expliquez votre choix : comment ça fonctionne et quels « non » ou « peut-être » cela peut-il impacter.

Réponse et explications : **È**ve peut mentir lorsqu'elle envoie ses données de routage à ses voisins. Elle peut leur faire croire qu'elle est plus proche de **B**ob qu'elle ne l'est vraiment.

Ève communique donc à **A**lice qu'elle peut rejoindre **B**ob à une distance de seulement 2 (au lieu de 4). Ainsi **A**lice pensera que **È**ve est sa meilleure prochaine voisine pour envoyer des paquets à **B**ob. En effet 2 est strictement mieux que la route de distance 3 qu'elle reçoit de **F**rance. La ligne « de **A**lice à **B**ob » devient donc un « oui ».

Ève peut aussi envoyer la même information à **C**arole. Cependant chez **C**arole cette route resterait à égalité avec les deux routes existantes, de distance 2. Elle pourrait alors choisir **È**ve comme prochaine voisine pour contacter **B**ob, mais ce n'est pas certain. La ligne « de **C**arole à **B**ob » devient alors un « peut-être ».

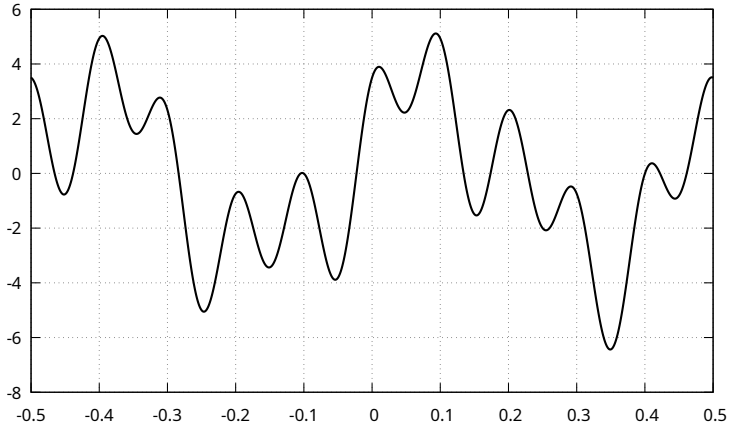
Ève pourrait être tentée de publier une distance de 1 seulement à **C**arole pour rejoindre **B**ob. Mais **C**arole sait que ce serait un mensonge : avec une distance de 1, on peut seulement aller jusqu'à **C**arole et jamais plus loin.

Commentaire : Beaucoup de réponses qui modifient les connexions du réseau alors que c'est explicitement proscrit.

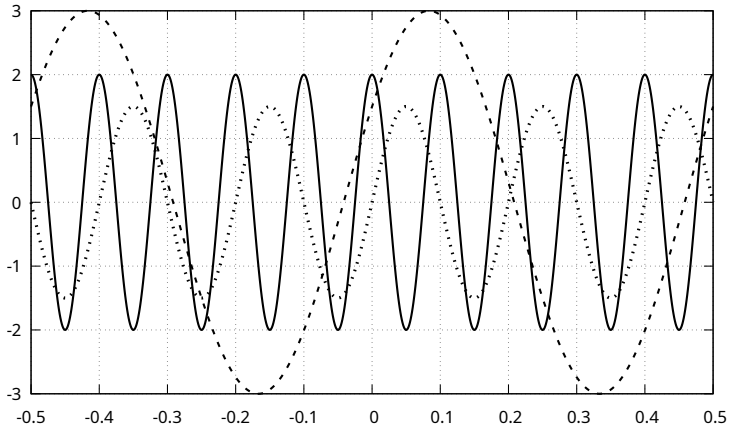
Aussi beaucoup de propositions qui n'ont pas compris le fonctionnement de TCP/IP (pour lequel le routage ne dépend pas de la qualité de service, que ce soit vitesse (débit ou latence) ou taux d'erreur).

Question 4 – Quel son ? [16 points]

On considère le signal $X(t)$ suivant (échelle en secondes) :



composé de la somme des trois fonctions suivantes :



dont les amplitudes sont parmi :

$$0.5 \quad 1 \quad 1.5 \quad 2 \quad 2.5 \quad 3$$

dont les fréquences (en Hz) sont parmi :

$$0 \quad 0.1 \quad 0.2 \quad 0.25 \quad 0.5 \quad 1 \quad 2 \quad 2.5 \quad 4 \quad 5 \quad 10 \quad 15 \quad 20$$

et dont les déphasages sont parmi :

$$0 \quad \frac{\pi}{6} \quad \frac{\pi}{4} \quad \frac{\pi}{2} \quad \frac{2\pi}{3} \quad \frac{3\pi}{4}$$

① **[3 points]** Écrivez la formule mathématique de $X(t)$:

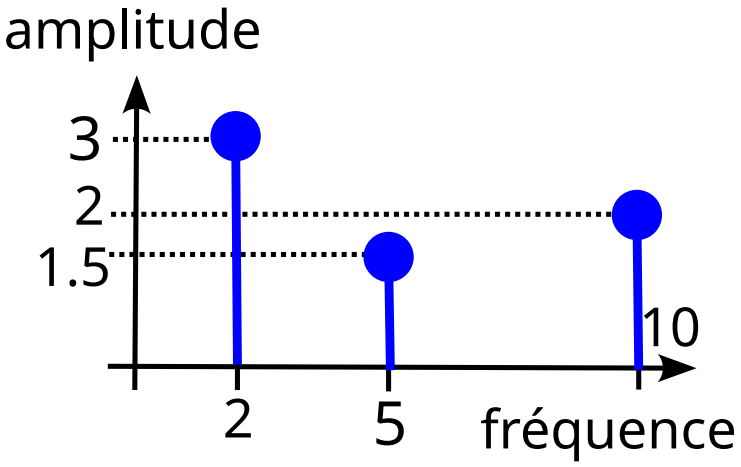
$X(t) = 3\,\sin(4\pi\,t + \frac{\pi}{6}?) + 1.5\,\sin(10\pi\,t) + 2\,\sin(20\pi\,t + \frac{\pi}{2})$

et **justifiez votre réponse** : On peut voir

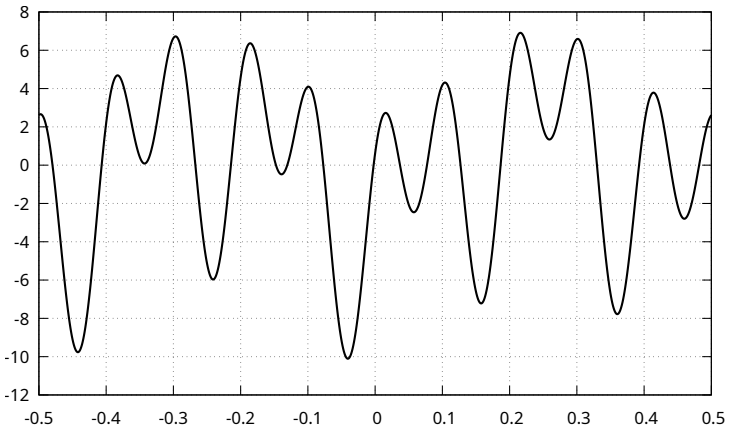
- une composante d'amplitude 3 et de période 0.5 s (donc fréquence 2 Hz) ;
- une composante d'amplitude 2 et de période 0.1 s (donc fréquence 10 Hz) ;
- et une composante d'amplitude 1.5 et de période 0.2 s (donc fréquence 5 Hz).

Pour les phases : on voit facilement le 0 et le $\frac{\pi}{2}$, reste à savoir pour le dernier entre $\frac{\pi}{4}$ et $\frac{\pi}{6}$ (on voit un petit déphasage), mais ce n'est pas très important car sans impact sur la suite.

② **[3 points]** Dessinez le spectre de $X(t)$:



On considère maintenant également le signal $Y(t) = X(t) + 1.5\,X(t + \frac{\pi}{4})$:

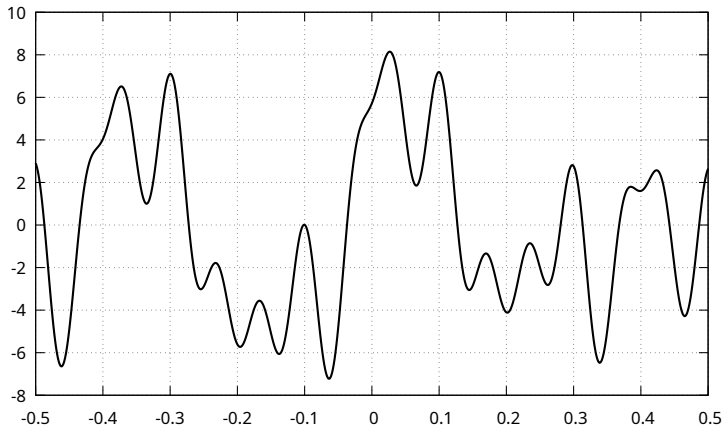


③ **[3 points]** Si l'on échantillonne ces deux signaux à une fréquence d'échantillonnage de $f_e = 22.5$ Hz, et que l'on reconstruit les signaux à partir de ces échantillons (suivant les formules vues en cours), lequel de ces deux signaux sera le mieux reconstruit ?

Justifiez pleinement votre réponse.

Réponse et justification : Y a les mêmes fréquences que X, donc la même bande passante, 10 Hz, qui est strictement inférieure à $f_e/2$. Ces deux signaux sont donc également reconstruits : à l'identique.

On considère maintenant le signal $Z(t) = X(t) + X(1.5\,t + \frac{\pi}{3})$:



④ **[3 points]** Si l'on échantillonne les signaux $X(t)$ et $Z(t)$ à une fréquence d'échantillonnage de $f_e = 27$ Hz, et que l'on reconstruit les signaux à partir de ces échantillons (suivant les formules vues en cours), lequel de $X(t)$ et $Z(t)$ sera le mieux reconstruit ?

Justifiez pleinement votre réponse.

Réponse et justification : les fréquences de Z sont donc les mêmes que celles de X plus 3, 7.5 et 15 Hz (multipliées par 1.5). Sa bande passante est 15 Hz qui est strictement plus grand que la moitié de la fréquence d'échantillonnage.

Z n'est donc pas parfaitement reconstuit alors que X l'est.

⑤ **[4 points]** Soit $\widehat{Z}(t)$ le signal $Z(t)$ après filtrage par un filtre passe-bas idéal de fréquence de coupure $f_c = 6$ Hz. Écrivez la formule mathématique de $\widehat{Z}(t)$:

$$\widehat{Z}(t) = 3\,\sin(4\pi\,t + \frac{\pi}{6}) + 3\,\sin(6\pi\,t + \frac{4\pi^2}{3} + \frac{\pi}{6}) + 1.5\,\sin(10\pi\,t)$$

et **justifiez votre réponse** : Les fréquences conservées ne sont donc que 2, 3 et 5 Hz, celle de 3 Hz venant de 2 Hz de X multipliée par 1.5 ; sa phase est donc $4\,\pi\,\frac{\pi}{3} + \frac{\pi}{6} = \frac{4\pi^2}{3} + \frac{\pi}{6}$

Question 5 – Un peu de broderie [24 points]

On s'intéresse ici à stocker des informations à propos de motifs de broderie. Un *motif* est une suite de couleurs et de jokers, que l'on choisit parmi rouge (R), vert (V), bleu (B), jaune (J) et joker (?). Une *réalisation* d'un motif est une suite de couleurs uniquement ; elle doit respecter les couleurs prescrites par le motif, mais peut utiliser une des quatre couleurs au choix pour chaque joker du motif.

Considérez le motif suivant :

$$M = \texttt{RRV??J?RJB?J}$$

et la chaîne

$$L = \texttt{RRVVBJVRJBRJ}$$

L est une *réalisation* possible de M. En effet, on peut obtenir L en remplaçant les quatre `?` de M par `V`, `B`, `V` et `R` respectivement.

En revanche, `RRVVB R VRJBRJ` n'est pas une réalisation possible de M, car en 6e position nous avons un `R`, qui ne correspond pas au `J` prescrit par le motif M. `RRVVBJVRJBRJ RRRR` n'est pas non plus valide car les longueurs ne correspondent pas.

Voici quelques approximations dont vous pourriez avoir besoin au long de cette question :

$$\log_2(3) \simeq 1.58 \qquad \log_2(5) \simeq 2.32 \qquad \log_2(7) \simeq 2.81$$

① **[6 points]**

a) Quelle est l'entropie maximale d'une *réalisation* (quelconque) ?

b) Quelle est, en bits, l'entropie maximale d'un *motif* considéré en tant que tel comme une chaîne de caractères ? Les jokers (`?`) font partie des caractères possibles de cette chaîne.

c) Quelle est l'entropie du motif M ?
Donnez votre réponse à c) sous la forme $a + b\log_2(3) + c\log_2(5) + d\log_2(7)$ avec a, b, c et d des nombres rationnels (potentiellement entiers voire nuls).

Justifiez vos trois réponses.

Réponses et justifications :

L'alphabet d'une réalisation comporte 4 lettres, et donc son entropie maximale vaut $\log_2 4 = 2$.

Pour le motif, nous avons 5 lettres (`?` est une lettre de ce point de vue), et nous obtenons donc $\log_2 5$.

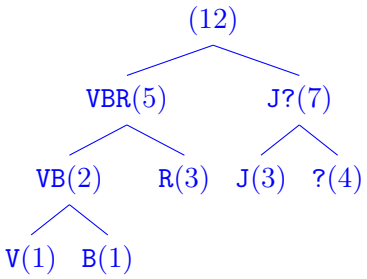
Pour le motif M, nous calculons via la formule de la définition de l'entropie. Pour cela il nous faut les probabilités relatives des différentes lettres. Nous avons 2 fois 1/12, 2 fois 3/12, et une fois 4/12. On a donc

$$\begin{aligned} H(M) &= 2 \cdot \frac{1}{12}\log 12 + 2 \cdot \frac{3}{12}\log 4 + \frac{4}{12}\log 3 \\ &= \frac{2}{12}(\log 3 + \log 4) + \frac{6}{12}\log 4 + \frac{4}{12}\log 3 \\ &= \left(\frac{2}{12} + \frac{4}{12}\right) \cdot \log 3 + \left(\frac{2}{12} + \frac{6}{12}\right) \cdot \log 4 \\ &= \frac{1}{2} \cdot \log 3 + \frac{2}{3} \cdot 2 \\ &= \frac{4}{3} + \frac{1}{2} \cdot \log 3 \end{aligned}$$

② **[6 points]** Proposez, en le justifiant, un code de Huffman pour M. Quelle est sa longueur moyenne ? **Justifiez** et **commentez** votre résultat.

Réponses et justifications :

On regroupe `V` et `B` ensemble pour un total de 2. Puis on regroupe `VB` avec `R` pour un total de 5. On regroupe `J` et `?` pour un total de 7. Et finalement nous regroupons ces deux derniers sous-arbres pour le total attendu de 12. En mettant les 1 à gauche et les 0 à droite, on obtient la table.



V	111
B	110
R	10
J	01
?	00

La longueur moyenne est

$$\frac{1}{12}(3+3+3\cdot 2+3\cdot 2+4\cdot 2)=\frac{26}{12}=\frac{13}{6}$$

Nous vérifions que 13/6 est bien compris entre $H(M)$ et $H(M)+1$. Nous avons

$$H(M)=\frac{4}{3}+\frac{1}{2}\cdot\log 3=\frac{8+3\log 3}{6}\simeq\frac{8+3\cdot 1.58}{6}=\frac{12.74}{6}$$

Nous avons donc bien

$$\frac{12.74}{6}\leq\frac{13}{6}<\frac{18.74}{6}=\frac{12.74}{6}+1$$

Il y a plusieurs arbres corrects et donc plusieurs codes de Huffman pour cette question. Toutes les réponses valides partagent les propriétés suivantes : `V` et `B` utilisent 3 bits dont les 2 premiers sont les mêmes ; `R`, `J` et `?` utilisent chacun deux bits ; le premier bit de `?` est *différent* du premier bit (commun) de `V` et `B`.

Commentaire : Bien que ce soit explicitement demandé, plusieurs ne justifient pas leur construction de code ou ne commentent pas les résultats obtenus ; parfois aucun des deux.

Il y en a aussi plusieurs qui ne savent pas calculer la longueur moyenne.

③ **[2 points]** Quelle est, en bits, l'entropie *maximale* d'une réalisation valide de M ?

Donnez votre réponse sous la forme $a+b\log_2(3)+c\log_2(5)+d\log_2(7)$ avec a, b, c et d des nombres rationnels (potentiellement entiers voire nuls).

Réponse et justification :

Pour obtenir l'entropie maximale, nous devons choisir des remplacements pour les `?` de telle sorte à rendre les différentes lettres les plus équiprobables possibles. Nous pouvons obtenir 3 occurrences partout en replaçant les `?` par deux `V` et deux `B`. Les 4 lettres ont donc toutes une probabilité de 3/12, et nous obtenons l'entropie maximale possible de $4\cdot\frac{1}{4}\log 4=2$ bit.

④ **[4 points]** Quelle est, en bits, l'entropie *minimale* d'une réalisation valide de M ?

Donnez votre réponse sous la forme $a+b\log_2(3)+c\log_2(5)+d\log_2(7)$ avec a, b, c et d des nombres rationnels (potentiellement entiers voire nuls).

Réponse et justification :

Pour obtenir l'entropie minimale, il faut créer la plus grande pique de probabilité possible. On peut le faire en remplaçant les quatre `?` par la même lettre qui a déjà une probabilité de 3/12, par exemple `R`. On a donc 7/12 pour `R`, tandis que les autres probabilités restent comme dans M. On obtient alors

$$\begin{aligned} H(X) &= 2\frac{1}{12}\log 12 + \frac{3}{12}\log 4 + \frac{7}{12}\log\frac{12}{7} \\ &= \frac{1}{12}\left(2(\log 3 + \log 4) + 3\log 4 + 7(\log 3 + \log 4 - \log 7)\right) \\ &= \frac{1}{12}\left(12\log 4 + 9\log 3 - 7\log 7\right) \\ &= 2 + \frac{3}{4}\log 3 - \frac{7}{12}\log 7 \end{aligned}$$

⑤ **[6 points]**

Considérons que l'on doit transmettre plusieurs réalisations valides de M. On suppose que l'émetteur et le récepteur *se mettent d'accord à l'avance* pour ne transmettre que des réalisations valides de M, et savent que toutes les couleurs sont équiprobables pour les `?` de M. Proposez un code (sans préfixe et sans perte) pour transmettre de telles réalisations.

Quelle est la longueur de votre code pour la réalisation L ?

Ce résultat est-il cohérent avec vos réponses aux questions précédentes ?

Réponses et justifications :

On peut être tenté d'utiliser un code de Huffman pour un message X construit en remplaçant les quatre `?` par quatre couleurs différentes, afin d'obtenir des choix équiprobables pour les jokers. Cependant, on peut faire *beaucoup* mieux. En effet, puisqu'on sait qu'on ne transmettra que des réalisations valides de M, nous n'avons pas besoin d'encoder les parties fixes de M du tout ! En termes de notre « jeu » des questions, on a besoin de zéro question pour déterminer que la première lettre sera `R`, par exemple.

Nous ne devons envoyer des données que pour les morceaux inconnus de X, qui sont les 4 couleurs qui prennent la place des jokers. Chaque couleur étant équiprobable, on utiliser un code élémentaire avec 2 bits par couleur.

La longueur du code pour L est donc seulement de $4 \cdot 2 = 8$ bits.

Cette longueur voudrait donc dire que nous utilisons « en moyenne » $8/12 = 2/3$ bits par lettre de X. Cela peut sembler incohérent avec l'entropie minimale de $\log 5 \simeq 2.32$ que nous avions calculée plus haut. Mais c'est normal, on exploite ici énormément d'information entièrement *connue*, qui réduit donc l'entropie effective de X dans ce cas précis.

Commentaire : Très peu ont *vraiment* compris la question et pensent à ne transmettre que les couleurs des 4 jokers. (Ce qui est connu est connu !)

Question 6 – Premiers [13 points]

Écrivez un programme C++ qui crée un fichier nommé `premiers.csv` contenant la liste, un par ligne, des nombres premiers et de leur carré, séparés par une virgule.
Le début du fichier `premiers.csv` sera donc :

```
2,4
3,9
5,25
7,49
```

Votre programme devra commencer par demander jusqu'à quel nombre maximum (pas nécessairement premier) on souhaite aller (inclus si c'est un nombre premier qui est donné). Par exemple :

```
Jusqu'à quel nombre souhaitez-vous aller ?
1234
```

Dans ce cas, la fin du fichier `premiers.csv` sera alors :

```
1229,1510441
1231,1515361
```

(car le prochain nombre premier après 1231 est 1237).

On supposera de plus qu'une fonction

```
unsigned long int prochain_premier(unsigned long int n);
```

est fournie (vous **n'**avez donc **pas** à l'écrire).
Cette fonction donne le prochain nombre premier *strictement* plus grand qu'un nombre `n`. Par exemple `prochain_premier(5)` retourne `7`, et `prochain_premier(1229)` retourne `1231`.

(Rappel : le premier nombre premier est 2.)

Réponse :

Voici une première version simple (qui n'a pas tous les points) :

```
#include <iostream>
#include <fstream>
using namespace std;

// PAS ATTENDU
unsigned long int prochain_premier(unsigned long int n);

int main()
{
  ofstream sortie("premiers.csv");
  if (sortie.fail()) {
    cout << "Impossible d'ouvrir le fichier" << endl;
    return 0;
  }

  unsigned long int last;
  cout << "Jusqu'à quel nombre souhaitez vous aller ?" << endl;
  cin >> last;

  unsigned long int p(2);
  while (p <= last) {
    sortie << p << ',' << p*p << endl;
    p = prochain_premier(p);
  }

  return 0;
}
```

Voici une autre version, mieux aboutie :

```
#include <iostream>
#include <fstream>
#include <string>
#include <limits>
using namespace std;

typedef unsigned long int entier;

entier prochain_premier(entier n); // PAS DEMANDÉ

entier demander_nombre()
{
  entier lu(0);
  do {
    cout << "Jusqu'à quel nombre (> 0) souhaitez vous aller ?" << endl;
    cin >> lu;
    if (cin.fail()) {
      cout << "Je vous ai demandé d'entrer un nombre, "
           << "pas du charabia !" << endl;
      cin.clear();
      cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }
  } while (lu == 0);
  return lu;
}

int main()
{
  const string nom_fichier("premiers.csv");
  ofstream sortie(nom_fichier);
  if (sortie.fail()) {
    cerr << "Impossible d'ouvrir le fichier \""
         << nom_fichier << "\" en écriture" << endl;
    return 1;
  }

  const entier last(demander_nombre());

  for (entier p(2); p <= last; p = prochain_premier(p))
    sortie << p << ',' << p*p << endl;

  sortie.close(); // optionnel en C++...

  return 0;
}
```

Commentaire : Plusieurs ne traitent pas du tout l'aspect fichier mais écrivent dans `cout`.

Plusieurs aussi ne donnent pas un programme complet (comme demandé).

Et très peu pensent à faire une saisie utilisateur robuste.

Enfin, un peu dans le même esprit qu'avec les `AdresseIP` de la Question 3, pourquoi ne pas uiliser les types demandés, ici `unsigned long int` ?