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) : Correction de l'Examen final

22 décembre 2022

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 (8h15 – 11h00).
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 7 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 125 points).
   Tous les exercices comptent pour la note finale.

Question 1 – Quelques questions [22 points]

① **[4 points]** Considérons une séquence de lettres X dont l'entropie est de 3.25 bit. Pour *chacune* des valeurs suivantes (en bits) : 3.45, 3.15, 3.25, 4.35 :

- **(a)** *peut*-il exister un code binaire sans-préfixe et sans perte de X ayant une telle longueur moyenne ? (oui ou non)
- **(b)** pensez-vous qu'il existe un code (sans-préfixe et sans perte) pour X ayant une longueur moyenne strictement plus petite ? (oui, non ou peut-être)

Répondez dans le tableau ci-dessous. **Attention !** il y aura des pénalités pour les réponses fausses.

longueur moyenne	(a) peut exister ?	(b) existe-t-il un meilleur code ?
3.45	oui	peut être
3.15	non	non
3.25	oui	non
4.35	oui	oui

② **[3 points]** En utilisant RSA, vous souhaitez signer [1] une information que vous envoyez à un ami dont la clé publique est $(47, 377)$. Votre clé publique est $(101, 247)$ et votre clé privée est 77.
Votre ami, dont la clé privée est 143, déchiffre le message qu'il a reçu et trouve `10100111`.

Quelle est la signature que vous avez envoyée ?

Exprimez votre réponse sous la forme « $x^y \mod z$ » et justifiez pleinement votre réponse.

Réponse et justification :

Le message d'origine (non signé) est donc `10100111`, soit 167 en décimal.

Vous l'avez signé avec votre clé *privée*, et donc envoyé :

$$167^{77} \mod 247$$

1. uniquement signer, pas protéger

Considérez le code assembleur suivant :

```
1: cont_ppe r2 0 9
2: multiplie r3 4 r0
3: charge r0 r1
4: multiplie r1 -5 r1
5: somme r1 r1 r3
6: somme r1 r1 7
7: somme r2 r2 -1
8: continue 1
9: rien (fin)
```

③ **[3 points]** Si `r0` contient 2, `r1` contient 1, et `r2` contient 3, que contient `r0` à la fin du programme ?

Justifiez brièvement votre réponse.

Réponse et justification :

-39 (valeurs intermédiaires de `r0` : 2, 1, 10)

④ **[3 points]** Ecrivez une fonction C++ *récursive* d'au plus deux lignes[2] dont la compilation pourrait donner le code assembleur ci-dessus.

Réponse :

```
int f(int u0, int u1, int k)
{
  if (k <= 0) return u0;
  return f(u1, 4 * u0 - 5 * u1 + 7, k-1);
}
```

Commentaire : Ca n'a pas de sens de passer par référence (et ce n'est d'ailleurs pas possible avec des expressions !).

⑤ **[2 points]** Quelle est la complexité de l'algorithme correspondant ?

Justifiez brièvement votre réponse.

Réponse et justification : $\Theta(\texttt{r2})$ (ou $\Theta(\texttt{k})$ si l'on suit la notation du code C++)

On appelle simplement autant de fois que $k-1$, le reste étant en $\Theta(1)$.

Commentaire : Avoir une boucle ne suffit pas pour avoir une complexité linéaire, encore faut il que le corps de la boucle soit en temps constant.

2. c.-à-d. dont le corps contient au maximum deux ; (points-virgules)

⑥ **[3 points]** En supposant que **taille()** est en $\Theta(1)$, quelle est la complexité de l'algorithme ci-contre, où $\lfloor x \rfloor$ représente la partie entière inférieure de x et $L[i:j]$ représente la sous-liste $(L(i), ..., L(j))$ si $i \leq j$ ou la liste vide si $i > j$?
Justifiez votre réponse.

algo1
entrée : *une liste L non vide*
sortie : *? ?*

```
n ⟵ taille(L)
Si n = 1
    Sortir : L(1)
k ⟵ ⌊n/3⌋
Si k = 0
    k ⟵ 1
p ⟵ algo1(L[1 : k])
q ⟵ algo1(L[k + 1 : n])
Si p < q
    Sortir : p
Sortir : q
```

Réponse et justification :
Cet algorithme est en $\Theta(n)$ avec n la taille de la liste en entrée.
En effet, tout est en $\Theta(1)$ sauf les appels récursifs, lesquels séparent toujours la liste en deux parties disjointes, qui seront chacune traitées totalement. Chaque élément de la liste sera ainsi traité une et une seule fois.
On peut aussi faire l'arbre des appels : c'est un arbre binaire **in**complet ayant n feuilles, et dont *chaque* nœud est visité (avec, pour chacun de ces nœuds, une complexité ajoutée dans ce nœud qui est en $\Theta(1)$: chaque nœud en lui-même n'a pas une complexité « cachée »). Il faut alors faire attention ici à bien compter les nœuds : il n'y en **n'**a **pas** de l'ordre de n (taille de la base) fois $\log(n)$ (hauteur de l'arbre), mais bien n au total, car l'arbre est largement incomplet.

Notes : (non demandées)
- cet arbre est complet jusqu'à la profondeur $\lfloor \log_k(n/2) \rfloor + 1$, puis incomplet jusqu'à la profondeur $\lceil (\log(n))/(\log(k) - \log(k-1)) \rceil$
- on peut aussi montrer formellement par récurrence que $C(n)$ est une fonction affine, en constant que $C(n) = \lambda + C(k) + C(n-k)$, avec λ la partie constante de chaque appel ; on arrive alors à $C(n) = (C(2) - C(1)) \cdot (n-1) + C(1)$ (et $\lambda = C(2) - 2C(1)$).

⑦ **[4 points]** Le plan du marché de Noël comporte 8 stands de nourriture, 3 stands d'habits chauds, 6 stands de bijoux et 1 stand pour faire des dons à une association carritative.

Quelle est l'entropie du plan des stands du marché de Noël ?

Donnez votre réponse sous la forme $a + b \log_2(3)$, avec a et b deux fractions (non nulles) ; puis *justifiez* la brièvement.

Réponse et justification :

$$a = -\frac{2}{3} \qquad \text{et} \qquad b = \frac{3}{2}$$

$$\frac{8}{18}\log_2\left(\frac{18}{8}\right) + \frac{3}{18}\log_2\left(\frac{18}{3}\right) + \frac{6}{18}\log_2\left(\frac{18}{6}\right) + \frac{1}{18}\log_2\left(\frac{18}{1}\right)$$

$$= \log_2(18) - \frac{1}{18}\left(8\log_2(8) + 3\log_2(3) + 6\log_2(6)\right)$$

$$= 2\log_2(3) - \frac{24}{18} - \frac{6}{18} - \left(\frac{3}{18} + \frac{6}{18}\right)\log_2(3) = -\frac{12}{18} + \frac{27}{18}\log_2(3) = -\frac{2}{3} + \frac{3}{2}\log_2(3)$$

Question 2 – Course à pied [20 points]

Vous organisez une course à pied et vous voulez archiver les temps de la façon la plus compacte possible. Ces temps sont représentés par une valeur en minutes et une valeur en secondes ; p.ex. 30 minutes et 58 secondes. Toutes les minutes entre 26 et 33 sont présentes (huit valeurs possibles) et toutes les secondes entre 00 et 59 aussi.

① Un stagiaire a proposé un premier codage consistant à simplement représenter la valeur entière positive des minutes, suivie de celle des secondes, *chacune* sur le nombre minimum de bits pour celle-ci.

a) [2 points] Combien de bits cette représentation utilise-t-elle pour chaque temps ?

Justifiez votre réponse.

b) [2 points] Concrètement comment est codé le temps 30 minutes et 58 secondes ?

Réponses et justification :

a) Pour coder les minutes (de 0 à 33, donc), il faut 6 bits ($32 < 33 < 64$).
Et il faut 6 bits pour coder les secondes ($32 < 59 < 64$).
Donc 12 bits au total.

b) 30 minutes et 58 secondes est codé :

```
011110  111010
```

② **[1 point]** Une seconde idée suggérée a été de coder les minutes, toujours séparement des secondes, mais sur le nombre minimum de bits pour coder les valeurs de minutes utilisées.

Combien de bits cette représentation utilise-t-elle pour chaque temps ?

Justifiez votre réponse.

Réponse et justification :

Comme il y a huit valeurs, **3 bits** suffisent pour coder les minutes (coder $x - 26$) ; soient 9 bits en tout.

③ **[2 points]** Une troisième idée proposée consiste à coder les minutes et les secondes ensembles sur le nombre minimum de bits pour coder toutes les valeurs de temps possibles.

Combien de bits cette représentation utilise-t-elle pour chaque temps ?

Justifiez votre réponse.

Réponse et justification :

Il y a donc $8 \times 60 = 480$ situations possibles. Il faut donc **9 bits** pour les coder : $256 < 480 < 512$.

Note : pour avoir une différence avec la réponse précédente, il faudrait 17, 33 ou 34 valeurs différentes pour les minutes.

④ Finalement, vous prennez en charge ce codage et commencez par constater que les secondes sont uniformément distribuées, mais que les minutes ne le sont pas du tout. Sur les statistiques du passé, vous constatez les comptes suivants :

26	27	28	29	30	31	32	33
2	2	16	16	64	16	10	2

a) [6 points] Proposez, complètement, de façon détaillée et justifiée, un code pour les temps.

b) [2 points] Concrètement quel est votre code pour le temps 30 minutes et 58 secondes ?

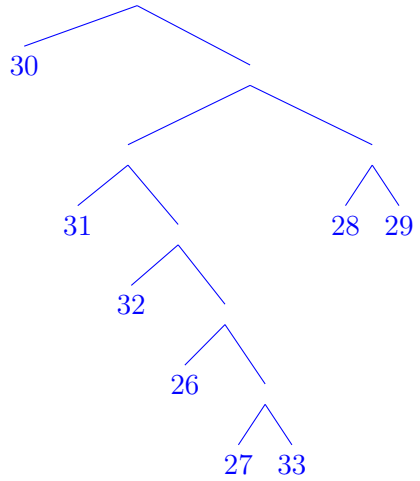
c) [5 points] Par rapport à la proposition du stagiaire faite en ①, combien gagnez vous de bits sur la représentation des temps de 1000 participants ? **Justifiez** votre réponse.

Réponses et justifications :

a)

30	28	29	31	32	26	27	33
64	16	16	16	10	2	2	2

L'idée est bien sûr de proposer le meilleur code possible, donc on fait un code de Huffman. Par exemple :



Par exemple :
30 : `1` (1),
28 : `001` (3),
29 : `000` (3),
31 : `011` (3),
32 : `0101` (4),
26 : `01001` (5),
27 : `010001` (6),
33 : `010000` (6),

30	28	29	31	32	26	27	33
$\frac{1}{2}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{5}{64}$	$\frac{1}{64}$	$\frac{1}{64}$	$\frac{1}{64}$
1	3	3	3	4	5	6	6

$$L(C) = \frac{1}{2}+\frac{9}{8}+\frac{37}{64} = \frac{141}{64} = 2+\frac{13}{64} \simeq 2+\frac{12.8}{64} = 2.2$$

Et l'on coderait toujours les secondes sur 6 bits comme avant, car étant équiprobables, leur entropie est proche de 6 bits ; il n'y a rien à y gagner.

Note : tel quel, ce n'est pas un code sans-prefixe ; mais comme la taille du code des secondes est fixée, on peut néanmoins le décoder sans ambiguité : on sépare les 6 derniers bits et l'on retrouve alors le code sans-prefixe pour les minutes.

b) Donc 30 minutes et 58 secondes se coderait :

`1 111010`

c) cf ci-dessus pour la longueur moyenne : on utilise donc en moyenne 8.2 bits, au lieu de 12. Sur 1000 messages, on gagne donc environ **3800 bits** (475 octets).

Question 3 – C'est par où ? [11 points]

Vous envoyez un fichier par FTP depuis votre ordi (D, comme départ) à une amie (A, comme amie ou comme arrivée) via Internet.

Voici quelques **extraits** de tables de routage (aucune information pertinente n'est cachée ; toutes les informations nécessaires pour répondre à la question sont données) :

Table de D :

dest.	dir.	dist
A	X	3
W	Z	2

Table de X :

dest.	dir.	dist
D	D	1
A	Y	2
W	Z	2

Table de Y :

dest.	dir.	dist
D	X	2
A	A	1
W	W	1

Table de Z :

dest.	dir.	dist
D	D	1
A	W	2
W	W	1

Table de W :

dest.	dir.	dist
D	Z	2
A	A	1
X	Z	2

Table de A :

dest.	dir.	dist
D	W	3
X	Y	2

① **[2 points]** Quelle(s) route(s) suivent les paquets de votre fichier ? Justifiez votre réponse.

Réponse et justification :

Il suffit de suivre la route en lisant les tables de proche en proche : D $\longrightarrow$ X $\longrightarrow$ Y $\longrightarrow$ A

Par contre, la route aller n'est pas la même que la route des paquets d'acquittement, lesquels transitent par : A $\longrightarrow$ W $\longrightarrow$ Z $\longrightarrow$ D

Note : le seul dessin du graphe complet n'est pas une route (et est inutile ici).

Commentaire : Les tables de routage étant *données*, les nœuds n'ont pas connaissance d'autres routes possibles.

② **[9 points]** Sachant que :

- la liaison entre X et Y perd un paquet sur 10,
- la liaison entre Z et W perd un paquet sur 15,
- la transmission de votre fichier nécessite 150 paquets,
- et que les temps de traitement [3] d'un paquet par les nœuds sont les suivants (en ms) :

D	X	Y	Z	W	A
10	2	3	5	5	15

Combien de temps va prendre la transmission de votre fichier ?
Justifiez pleinement votre réponse.

Note : on pourra faire l'approximation $24/150 = 4/25 \simeq 1/5$

Réponse : 1 paquet sur 10 sera perdu à l'aller et 1 paquet d'aquittement sur 15 sera perdu. Donc globalement D considère que $10 + 15 - 1 = 24$ paquets sont perdus sur 150.

$$\left(1 - \frac{14}{15} \cdot \frac{9}{10} = \frac{24}{150}\right)$$

3. par « traitement », nous entendons, traitement *complet* par le nœud suivant ce qu'il a à faire (émission, réception, attente, retransmission, etc.)

D va donc envoyer

$$150 + 24 + 4 + 1 = 179$$

paquets.

Note : si on fait l'approximation suggérée ($24/150 = 4/25 \simeq 1/5$), on trouve alors :

$$150 + 30 + 6 + 1 = 187$$

En ce qui concerne le temps :

- la version la plus simple tolérée est de donner la priorité à la note de bas de page sans autre réflexion et donc compter simplement le temps total par paquet : $10 + 2 + 3 + 15 = 30$ ms à l'aller et $15 + 5 + 5 + 10 = 35$ ms pour les acquittements, soient 65 ms en tout par paquet ; et de simplement multiplier cela par le nombre de paquets
- une version un tout petit peu plus élaborée serait de dire qu'il n'est pas nécessaire de compter l'acquittement du dernier paquet (celui, seul, de la dernière « salve » de renvoi) et donc soustraite 35 ms à la solution précédente
- des versions plus avancées pourraient utiliser le parallélisme des envois et ne compter donc que 10 ms par paquet à l'aller et 15 ms au retour puis *ajouter* (et non pas multiplier) les temps ci-dessus (30 ms et 35 ms) comme latence (initiale, ou mieux, finale).

Question 4 – Le retour de la machine [14 points]

On considère la machine de Turing ayant pour table de transition :

	0	1	ε
1	(1, 0, +)	(2, 0, −)	(8, ε, +)
2	(2, 0, −)	(1, ε, −)	(3, ε, −)
3	(4, 1, −)	(3, 0, −)	(4, 1, −)
4	(4, 0, −)	(4, 1, −)	(5, ε, +)
5	(5, 0, +)	(5, 1, +)	(1, ε, +)

① **[6 points]** Quel est l'état de la bande et la position de la tête de lecture lorsque la machine s'arrête, si elle a démarré avec sa tête de lecture positionnée comme suit :

$\cdots\varepsilon$	0	ε	0	1	0	1	1	0	1	$\varepsilon\cdots$
			↑							

② **[8 points]** Justifiez votre réponse en deux ou trois phrase(s), puis dites, en une courte phrase, ce que fait cette machine.

Réponses : ①

$\cdots\varepsilon$	1	0	0	ε	0	0	0	0	0	0	0	ε	$\varepsilon\cdots$
													↑

Commentaire : Pourquoi ne pas donner la position de la tête de lecture alors que c'est explicitement demandé ?

② Cette machine compte le nombre de 1 présents dans l'entrée (partie droite du ruban).

Elle est composée de quatre parties :

- état 1 : un « détecteur et effaceur » de 1 sur l'entrée (partie droite)
- état 2 : remonter en tête de l'entrée (et un cran de plus)
- états 3 et 4 : un incrémenteur (ajoute 1 en binaire – cf exercices pendant semestre)
- état 5 : sauter le compte pour retourner au début de l'entrée

suite au dos ☞

Question 5 – Le compte est bon [27 points]

On s'intéresse ici à écrire des *parties* d'un programme C++ permettant de résoudre le problème du « compte est bon » : à partir de 6 nombres entiers donnés et d'un résultat désiré (aussi entier), essayer de combiner un sous-ensemble quelconque de ces 6 nombres, pris chacun qu'une seule fois, pour obtenir le résultat.

Par exemple, peut-on trouver le résultat 329 à partir des nombres $(2, 5, 5, 8, 25, 50)$?
La réponse est « oui » : $(5 + 2) \times (50 - (8 - 5)) = 329$.

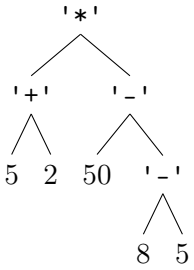
① **[3 points]** Dans un premier temps, on s'intéresse à représenter une suite de calculs arithmétiques. Pour cela, un *calcul* est défini comme :

- une opération ($+$, $-$, $\times$ ou $/$) (un `char` suffira en C++) ;
- un résultat (entier positif) ;
- un calcul, opérande de droite ;
- un calcul, opérande de gauche.

Par exemple, le calcul $(5 + 2) \times (50 - (8 - 5))$ peut se représenter comme :

- opération : `'*'` ;
- résultat : 329 ;
- calcul de gauche : « $(5 + 2)$ » ;
- calcul de droite : « $(50 - (8 - 5))$ ».

Évidemment, les deux derniers éléments sont également des *calculs*, chacun avec leurs éléments respectifs. Le calcul « $(5 + 2) \times (50 - (8 - 5))$ » complet pourrait donc se représenter (sans les résultats) comme ci-contre :



Proposez un type `Calcul` pour représenter en C++ les calculs tels que définis ci-dessus.

Réponse :

```
typedef unsigned int Nb;
typedef char         Operation;

struct Calcul {
  Operation op;
  Nb result;
  const Calcul* first;
  const Calcul* second;
};
```

Les `typedef` sont une bonne pratique, optionnelle ici (mais pensez-y).

② **[3 points]** Comment représentez-vous les nombres de base utilisés dans les calculs (comme p.ex. 8, 5, 50 ou 2 dans le calcul précédent) ?

Définissez p.ex. en C++ la variable représentant le nombre 50 du calcul précédent.

Réponse : Ils sont représentés avec une opération peut être n'importe quel caractère sauf ceux des opérateurs arithmétiques.

Et ses deux « sous-calculs » sont des pointeurs nuls.

```
const Calcul cinquante({' ', 50, nullptr, nullptr});
```

③ **[7 points]** On souhaite afficher joliment ces calculs. Par exemple, on souhaiterait afficher le calcul précédent $((5+2)\times(50-(8-5))=329)$ comme ci-contre, où le décalage de chaque nouveau sous-calcul est de 3 espaces vers la droite.
Définissez la fonction C++ *récursive* `display()` permettant de faire un tel affichage.

```
329 =
  7 =
      5
      +
      2
  *
  47 =
      50
      -
      3 =
          8
          -
          5
```

Réponse :

```
void display(Calcul const& r, int indent = 0)
{
  cout << setw(indent) << ' ' << r.result;
  if (r.first) cout << " = ";
  cout << endl;

  if (r.first) {
    constexpr int increment(3);
    const int next_level(indent + increment);
    display(*r.first, next_level);
    cout << setw(next_level) << ' ' << r.op << endl;
    if (r.second) display(*r.second, next_level);
  }
}
```

On ne va pas s'intéresser à tous les détails du programme complet, mais se focaliser uniquement sur la résolution « de haut niveau » dans les trois prochaines questions.

L'algorithme précis sera donné en question ⑤, mais le principe de résolution général est de réduire, par essais successifs, le problème de départ jusqu'à atteindre un calcul dont le résultat est le résultat désiré au départ. On procédera pour cela de proche en proche (récursivement) en remplaçant deux nombres du problème par le résultat de leur opération.

Par exemple, la résolution « $(5+2)\times(50-(8-5))=329$ » du problème de départ « peut-on trouver le résultat 329 à partir des nombres $(2,5,5,8,25,50)$? » aura été obtenue par résolution des problèmes suivants :

- « peut-on trouver 329 à partir de $(2,5,5,8,25,50)$? »
- « peut-on trouver 329 à partir de $(7,5,8,25,50)$? » (combinaison de 5 et 2 en $7=5+2$)
- « peut-on trouver 329 à partir de $(7,3,25,50)$? » (combinaison de 8 et 5 en $3=8-5$)
- « peut-on trouver 329 à partir de $(7,47,25)$? » (combinaison de 50 et 3 en $47=50-3$)
- « peut-on trouver 329 à partir de $(329,25)$? » (combinaison de 7 et 47 en $329=7\times47$)

dont la réponse est « oui ».

Il est clair que pour pouvoir reconstruire toute la solution, la liste donnée en seconde partie de chaque problème est plus qu'une liste de nombres : c'est une liste de `Calcul` (tels que définis en ①).

④ **[1 point]** Proposez un type C++ nommé `Objectif` contenant

- une liste de `Calculs` (ou, suivant votre choix, de pointeurs sur des `Calculs`), que nous appellerons « candidats » ;
- et un nombre entier, que nous appellerons « cible ».

Réponse :

```
struct Objectif
{
  vector<Calcul*> candidats;
  Nb cible;
};
```

`vector` car la taille varie en fonction de l'évolution des calculs ; les 6 plaques au départ.

⑤ **[7 points]** L'algorithme général de résolution est alors le suivant :

solve

entrée : *un* `Objectif`
sortie : *trouvé ou non ? (booléen)*

```
n ⟵ nombre de candidats
Si n ≤ 1
    Sortir : Faux
Pour i de 1 à n − 1
    Pour j de i + 1 à n
        Pour o parmi toutes les opérations (+, −, ×, /)
            Si o est − ou /
                tester l'opération o avec le candidat j et le candidat i
                Si le test est positif
                    Sortir : Vrai
            tester l'opération o avec le candidat i et le candidat j
            Si le test est positif
                Sortir : Vrai
Sortir : Faux
```

où le sous-algorithme **tester** consiste à :

- vérifier que l'opération est valide (on ne peut par exemple pas effectuer de division dont le résultat ne soit pas un entier, ni de soustraction qui donne un résultat négatif) **sortir** « Faux », sinon ;
- calculer le résultat de l'opération testée ;
- **si** le résultat dépasse (strictement) la cible, **sortir** « Faux » ;
- **si** le résultat égale la cible, **sortir** « Vrai » ;
- relancer la recherche (algorithme **solve** ci-dessus) avec comme nouvel objectif, la même cible, mais une liste de candidats réduite, où les deux candidats considérés ont été supprimés et le premier a été remplacé par le résultat du calcul de l'opération testée.

Quelle est la complexité de cet algorithme de résolution (**solve**) ? **Justifiez** votre réponse.

Réponse et justification : C'est un algorithme extrêmement « gourmand » : pour sa partie principale (c.-à-d. hors termes en $\Theta(1)$) il teste toutes les résolutions possibles pour toutes les paires (i, j) (avec i de 1 à $n-1$, n étant le nombre de candidats, et j de $i+1$ à n ; soient $\frac{(n-1)n}{2}$ paires) et toutes les opérations (donc 4).

On a donc :

$$C(n) = 4\,\frac{(n-1)n}{2}\,C(n-1) + K$$

(K regroupe les opérations en $\Theta(1)$)

Ce qui donne (pour le terme dominant) :

$$C(n) \in \Theta\left(4^{n-1}\frac{2\cdot 3}{2} \times \cdots \times \frac{(n-2)(n-1)}{2} \times \frac{(n-1)n}{2}\right) = \Theta\left(4^{n-1}\frac{(n-1)!}{2^{n-1}} \times \frac{n!}{2}\right) = \Theta\left(2^n \times n! \times (n-1)!\right)$$

Les réponses $\Theta(2^n\, n^{2n})$ et $\Theta(\left(\frac{2}{e^2}\right)^n\, n^{2n})$ (ou similaires) sont aussi valables au niveau de ce cours.

⑥ **[6 points]** En supposant qu'il existe une fonction

```
bool tester(char operation, const Calcul* first, const Calcul* second,
            size_t i, size_t j, Objectif const& obj)
```

qui effectue l'algorithme **tester** sur l'opération `op` avec les deux candidats `first` et `second` (situés aux positions `i` et `j` des candidats de l'`Objectif obj`), écrivez le code C++ de la fonction `solve()` qui implémente l'algorithme **solve** décrit en ⑤.

Réponse :

```
bool solve(Objectif const& obj)
{
  if (obj.candidats.size() <= 1) return false;

  for (size_t i(0); i < obj.candidats.size()-1; ++i) {
    for (size_t j(i+1); j < obj.candidats.size(); ++j) {
      for (char op : { '+', '-', '*', '/' }) {
        if (((op == '-') or (op == '/')) and
            tester(op, obj.candidats[j], obj.candidats[i], i, j, obj))
          return true;

        if (tester(op, obj.candidats[i], obj.candidats[j], i, j, obj))
          return true;
      }
    }
  }

  return false;
}
```

Question 6 – Variations algorithmiques [14 points]

Note : Avant de répondre à ①, lisez la sous-question ③ (au dos).

On cherche à écrire un algorithme permettant de sortir, en ordre décroissant, les k valeurs maximales (k fixé, p.ex. $k = 5$) de l'ensemble des produits des valeurs de deux listes.

Par exemple, pour les listes $L_1 = (5, 2, 8, 3)$ et $L_2 = (4, 7)$, et pour $k = 5$, un tel algorithme sortira la liste $(56, 35, 32, 21, 20)$, puisque ce sont là les 5 valeurs maximales des produits des éléments de L_1 par des éléments de L_2.

① **[4 points]** Écrivez un algorithme pour résoudre ce problème.

Réponse :

Voici un premier algorithme, simple :

algo1
entrée : *deux listes* L_1 *et* L_2
sortie : *les* k *valeurs maximales [...]*

```
L' ⟵ ()
Pour tout a de L1
    Pour tout b de L2
        L' ⟵ L' ⊕ (a × b)
trier_décroissant(L')
Sortir : L'[1 : k]
```

avec $\oplus$ l'ajout d'un élément en fin de liste et $L[1 : k]$ la sous-liste initiale d'au plus k éléments (donne toute la liste si sa taille est plus petite que k).

Le tri se fait bien sûr en ordre *décroissant* ici.

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

Réponse : Si l'on considère $\oplus$ comme élémentaire, la complexité de l'algorithme précédent est dominée par le tri de la liste ayant $n_1 \times n_2$ éléments (avec n_i la taille de L_i)

Donc une complexité en $\Theta(n^2 \log n)$ avec n la plus grande des tailles des deux listes.

③ **[8 points]** Écrivez un algorithme de complexité temporelle $\Theta(n \log n)$ pour résoudre le problème proposé, avec n la plus grande des tailles des deux listes.

Si votre réponse à ① est déjà en $\Theta(n \log n)$, vous n'avez rien à faire ici (et serez, bien entendu, noté(e) sur la somme des points des deux sous-questions).

Réponse : L'idée pour améliorer l'algorithme précédent est de ne pas faire plus de calculs que nécessaire : il n'est pas nécessaire faire les $n_1 \times n_2$ produits, seuls $k \times k$ suffisent : les k plus grands se trouvent dans le produit des k plus grands de L_1 avec les k plus grands de L_2.

Voici un algorithme en $\Theta(n \log n)$ pour résoudre cette tâche :

algo2

entrée : *deux listes* L_1 *et* L_2
sortie : *les* k *valeurs maximales [...]*

trier_décroissant(L_1)
trier_décroissant(L_2)
$L' \longleftarrow ()$
Pour tout a de $L_1[1:k]$
  Pour tout b de $L_2[1:k]$
    $L' \longleftarrow L' \oplus (a \times b)$
trier_décroissant(L')
Sortir : $L'[1:k]$

Les tris se font bien sûr en ordre *décroissant* ici.

La grosse différence est donc que L' n'a qu'au plus k^2 éléments et donc le trier est en $\Theta(1)$ (par rapport à n).

Note : on pourrait faire encore mieux, en $\Theta(n)$ (mais ce n'était pas demandé), en extrayant de façon linéaire les k plus grandes valeurs de L_1 puis de L_2 dans les deux premières étapes ci-dessus (pas besoin de trier pour cela).

Question 7 – Taux de CO2 [17 points]

Une de vos amies veut créer un système de relevé automatique du taux de CO2 dans sa chambre (entre 0 et 100). Elle sait que son capteur électronique a une précision de 0.1 et que chaque mesure sera stockée avec une indication du temps à laquelle elle aura été prise, exprimé en secondes depuis le 1er janvier 2000.

Le tout sera stocké en C++ dans un `vector` de `struct` ayant deux champs (dans cet ordre) :

- le taux comme un nombre à virgule flottante sur 32 bits avec 23 bits de mantisse (ordre : signe, exposant – ici directement en binaire non signé – puis mantisse)
- et le temps comme un `int` sur 32 bits.

① **[3 points]** Tout les combien de temps (au minimum) est-ce que son système doit sauvegarder une mesure si votre amie veut pouvoir reconstruire exactement la courbe des taux mesurés et que l'on suppose que ces mesures ne varient pas à plus de 0.025 Hz.

Justifiez votre réponse.

Réponse et justification :

échantilloner à au moins $2f$, soit au moins toutes les 20 secondes (on peut bien sûr le faire plus souvent).

② **[7.5 points]** Sachant que votre amie a commencé à enregistrer ses mesures à partir du temps écrit en binaire `0010 1000 0101 0000 0001 0010 1110 1011`,

a) à quoi correspond l'enregistrement écrit en binaire sur le disque comme :
`0000 0001 1010 0100 0000 0000 0000 0000, 0010 1000 0101 0000 0001 0101 0111 1111`
Quel échantillon est-ce ? (c.-à-d. combien de temps après le début ?)

b) Quel est le taux de CO2 correspondant ?

Justifiez vos réponses.

Réponses et justification :

11 min. (= 660 s) après le début de ses mesures, la taux était de 10.25%.

11 min. (= 660 s) se trouve en faisant soit l'addition/soustraction des valeurs correspondantes aux bits différents, soit en détaillant le calcul comme suit :
`0010 1000 0101 0000 0001 0101 0111 1111` − `0010 1000 0101 0000 0001 0010 1110 1011`
soit `101 0111 1111` − `010 1110 1011` (biffer les parties égales)
soit 4 fois `101011111` − `010111010` (il y a 2 zéros à la fin, que je décale, donc multiplication par 4)
soit 4 fois `101011111` + `101000110` (complément à deux)
c.-à-d. 4 fois `010100101`, c.-à-d. 4×165 =11 min. (= 660 s).

Le taux de 10.25% se trouve en décomposant `0001 1010 0100` comme `0 0011 0100100...`, donc `0011` d'exposant (c.-à-d. 3, donc la valeur 8) et `1.01001` pour la mantisse ;
ce qui donne : $8 + 0 + 2 + 0 + 0 + 0.25 = 10.25$.

③ **[6.5 points]** Comment pourrait faire votre amie pour réduire la taille des informations stockées, sans perdre d'information ?

Essayez de lui proposer un système optimal. Expliquez votre démarche en détails.

Réponse :

Si elle est sûre de prendre ses mesures à intervalles réguliers, sans en perdre, alors rien ne sert de stocker

le temps à part le temps de départ, peut être.

Et sinon, on peut de toutes façons stocker les temps à partir de 0 (et la date de départ) au lieu de stocker chaque temps à partir du premier janvier 2000 et gagner plusieurs bits sur chaque date.

Pour les taux entre 0 et 100, à raison de 0.1 cela fait 1001 mesures possibles, soient 10 bits.

Ensuite, on pourrait étudier l'entropie des taux de CO2 et certainement y gagner encore (car ils sont certainement loin d'être équiprobables) avec un code de Huffman. On pourrait aussi envisager de coder les écarts entre valeurs successives au lieu des valeurs elles-mêmes.