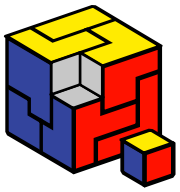
Prologin 2017



Concours National d'Informatique

Correction de la phase de sélection

Correction des questions d'algorithmique

Guillaume Aubian, Rémi Audebert, Théophile Bastian, Théo Pierron*

24 janvier 2017

Table des matières

*Pour l'équipe des correcteurs 2017 : Théophile Bastian, Clément Beauseigneur, Éloi Charpentier, Nicolas Ding, Stéphane Henriot, Théo Pierron, Alexandre Talon.

1 Questions ouvertes

1.1 Question 1 : Bases

Énoncé On note « A en base b » : A_b. Combien font, en base 10, $10_2 + 10_8 + 10_{16}$?

Correction Cette question a pour but de vous échauffer les doigts afin de vous mettre dans une bonne situation pour attaquer les exercices suivants.

Trouvons la solution en Python :

```
>>> 0b10 + 0o10 + 0x10
26
```

Voilà c'est fini, on a utilisé les notations pour la base 2 (`0b`), 8 (`0o`) et 16 (`0x`) de Python et nous lisons la réponse en base 10 : 26. Passons maintenant à quelque chose de tout à fait différent.

Imaginons un instant que vous n'ayez pas Python mais uniquement une imprimante[1], que faire alors ? Une deuxième solution s'offre à vous, elle consiste à utiliser PostScript, dans une adaptation plus utilisable : le Encapsulated PostScript.

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 37 28

/Courier findfont 14 scalefont setfont

2#10 8#10 16#10 add add
3 string cvs

10 10 moveto
show
```

Vous pouvez écrire ce bloc de code dans un fichier et l'envoyer à votre imprimante, elle se se fera un plaisir de le digérer. Après quelques instants une feuille de papier sortira sur laquelle de l'encre sera déposée d'une façon vraisemblablement intelligible, assimilable à ces deux caractères côte à côte : 2 et 6. Il est aussi possible que votre imprimante vous sorte une série d'insultes où l'interpréteur PostScript s'offusque que vous lui ayez envoyé une commande offensante.

En dernier recours, vous pouvez utiliser votre chère calculatrice de bureau GNU[2], pour laquelle notre solution combine élégance et simplicité :

```
$ echo Isb2i10lbi8i10lbi16i10++p | dc
26
$
```

1.2 Question 2 : Comptage

Énoncé Si on écrit tous les nombres de 1 à 999 à la suite en séparant les chiffres par des points (1.2.3.4.5.6.7.8.9.1.0.1.1.1.2…), combien de points seront immédiatement entourés par des chiffres impairs ?

1. Mettez-vous dans la situation inédite d'être employé par une compagnie d'assurance dont les bureaux sont installés sur un bateau, quelque part du côté de Londres. La présence d'une imprimante mais pas de logiciel adéquat est dans ce cas parfaitement concevable.
2. Disponible librement à cette adresse.

Correction Une solution en Python :

```python
a = ''.join(str(i) for i in range(1, 1000))
# On crée deux itérateurs, le second en avance d'un caractère sur le premier
gauche = iter(a)
droite = iter(a)
next(droite)
print(sum(g in '13579' and d in '13579' for g, d in zip(gauche, droite)))
# affiche : 775
```

1.3 Question 3 : Population

Énoncé Quel est le 1000^e entier qui a un nombre premier de 1 dans sa représentation binaire ?

Correction Le nombre de `1` dans la représentation binaire est aussi appelé le poids de Hamming, ou aussi la *population*. Un entier dont la population est un nombre premier est appelé un nombre pernicieux (*pernicious number* en anglais). La liste des nombres pernicieux est disponible dans l'encyclopédie des suites d'entiers en ligne sous la référence A052294.

Une solution en Python utilisant la bibliothèque `pyprimes` ressemble à ceci :

```python
import pyprimes
n = 0
i = 0
while i != 1000:
    n += 1
    if pyprimes.isprime(bin(n).count('1')):
        i += 1
print(n)
# affiche : 1990
```

Le calcul de la population a été intégré dans les processeurs `x86_64` avec l'instruction `popcnt`. On peut s'amuser à résoudre cet exercice en l'utilisant :

```
#include <asm/unistd_64.h>
// Sauvegardez ça dans qcm_3.S, puis lancez :
// $ make qcm_3 && ./qcm_3

.section .rodata
fmtstring:
        .asciz        "%d\n"
// On a précalculé la primalité pour toutes les population possibles sur 64 bits
primality_lut:
        .byte        0 /*  0 */, 0 /*  1 */, 1 /*  2 */, 1 /*  3 */, 0 /*  4 */
        .byte        1 /*  5 */, 0 /*  6 */, 1 /*  7 */, 0 /*  8 */, 0 /*  9 */
        .byte        0 /* 10 */, 1 /* 11 */, 0 /* 12 */, 1 /* 13 */, 0 /* 14 */
        .byte        0 /* 15 */, 0 /* 16 */, 1 /* 17 */, 0 /* 18 */, 1 /* 19 */
        .byte        0 /* 20 */, 0 /* 21 */, 0 /* 22 */, 1 /* 23 */, 0 /* 24 */
        .byte        0 /* 25 */, 0 /* 26 */, 0 /* 27 */, 0 /* 28 */, 1 /* 29 */
        .byte        0 /* 30 */, 1 /* 31 */, 0 /* 32 */, 0 /* 33 */, 0 /* 34 */
        .byte        0 /* 35 */, 0 /* 36 */, 1 /* 37 */, 0 /* 38 */, 0 /* 39 */
        .byte        0 /* 40 */, 1 /* 41 */, 0 /* 42 */, 1 /* 43 */, 0 /* 44 */
        .byte        0 /* 45 */, 0 /* 46 */, 1 /* 47 */, 0 /* 48 */, 0 /* 49 */
        .byte        0 /* 50 */, 0 /* 51 */, 0 /* 52 */, 1 /* 53 */, 0 /* 54 */
        .byte        0 /* 55 */, 0 /* 56 */, 0 /* 57 */, 0 /* 58 */, 1 /* 59 */
        .byte        0 /* 60 */, 1 /* 61 */, 0 /* 62 */, 0 /* 63 */, 0 /* 64 */
```

```
.text
.code64
.global main
.type main, @function
main:
        // On s'assure de la disponibilité de l'instruction popcnt en vérifiant
        // que le bit correspondant de cpuid est présent
        // %eax=1: Processor Info and Feature Bits
        xor        %eax, %eax
        inc        %eax
        cpuid
#define POPCNT_BIT 23
        bt        $POPCNT_BIT, %ecx
        jnc        do_exit_err

        // %r12: le nombre d'entier pernicieux qu'il nous reste à trouver
        // On cherche le 1000e entier
        mov        $1000, %r12
        // %rbx: l'entier courant
        // On commence avec 0
        xor        %rbx, %rbx
        // On a besoin que les octets de poids forts de %rcx soient à 0
        xor        %rcx, %rcx
        // On charge l'adresse de la table des nombres premiers
        lea        primality_lut(%rip), %rax

.Lloop:
        // Nombre suivant
        inc        %rbx

        // On stocke le nombre de bits à 1 de %rbx dans %rsi
        // http://developer.amd.com/wordpress/media/2008/10/24594_APM_v3.pdf
        popcnt        %rbx, %rsi

        // Lecture de l'entrée %rsi dans la table
        // %cl, et par extension %rcx dont on s'est assuré de la nullité des
        // octets de poids forts, vaudra 1 si le nombre est premier
        mov        (%rax,%rsi), %cl

        // Décompte des nombres pernicieux restant à trouver :
        // - si le nombre courant en est un (%rcx=1) on décrémente de 1
        // - sinon (%rcx=0) on décrémente de 0
        sub        %rcx, %r12

        // Est-ce qu'on a trouvé notre 1000e nombre ?
        test        %r12, %r12
        jnz        .Lloop

        // On l'a trouvé, on l'affiche et on quitte
        lea        fmtstring, %rdi
        mov        %rbx, %rsi
        xor        %rax, %rax
```

```
        call        printf

        // return 0
        xor         %rax, %rax
        ret

do_exit_err:
        // return 1
        mov         $1, %rax
        ret
```

1.4 Question 4 : Syracuse

Énnoncé Quel est le plus grand entier strictement inférieur à 421337 qui engendre la plus longue suite de Syracuse ?

Correction Une solution en Python :

```python
def collatz_length(x):
    l = 1
    while x > 1:
        l += 1
        if x % 2 == 0:
            x = x / 2
        else:
            x= 3 * x + 1
    return l
print(max((collatz_length(n), n) for n in range(1, 421338)))
# affiche : 410011
```

1.5 Question 5 : Cyrano

Énoncé Dans combien de scènes le personnage éponyme Cyrano de Bergerac de la pièce d'Edmond Rostand s'exprime-t-il ? Attention : Cyrano parle parfois sans être annoncé !

Correction La solution la plus simple et la plus agréable, soyons francs, est bien évidemment de lire et relire encore la pièce tout en comptant les interventions de Cyrano. Il ne fallait surtout pas s'arrêter à lire simplement les annonces des personnages en début de chaque chapitre, car comme précisé dans l'énoncé, Cyrano se joue du théâtre.

On va utiliser la version Wikisource du texte, et prenant chaque page, les concaténant puis séparant en scènes, vérifier, pour chacune d'elles, si Cyrano y parle.

```python
import requests
def has_cyrano(t):
    return '{{personnaged|cyrano' in t or '{{personnage|cyrano' in t
pages = [requests.get('https://fr.wikisource.org/w/index.php?title=' +
    'Page:Rostand_-_Cyrano_de_Bergerac.djvu/%s&action=raw' % i) for i in range(1, 215)]
pages_text = [p.text.lower() for p in pages]
scenes = '\n'.join(pages_text).split('{{scène')
print(sum(has_cyrano(s) for s in scenes))
# affiche : 37
```

1.6 Question 6 : Babel

Énoncé On trouve dans la bibliothèque de Babel créée par Jonathan Basile sur https://libraryofbabel.info/ des volumes avec des pages dont le contenu est le seul mot « prologin » entouré d'espaces. Quel est le numéro d'une telle page dans le volume titré « ozggclu » de l'hexagone uoqoogwiczo[…] sur le mur 3 au niveau 5 de l'étagère ?

Correction Pour répondre à cette question il n'était pas nécessaire d'écrire du code. Voici les étapes à suivre pour trouver la solution :

1. Aller sur https://libraryofbabel.info/, dans la section « Browse ».
2. Copier dans le champs de texte tous les caractères contenus à l'adresse https://prologin.org/media/quizz/2017-hexagone.txt, c'est le nom de l'hexagone que l'on cherche. Appuyer sur `Entrée`.
3. Sélectionner le mur 3, le niveau 5 de l'étagère qui s'affiche puis le livre marqué « ozggclu » sur la tranche.
4. Une fois dans le livre, vous pouvez lire toutes les pages jusqu'à arriver à celle qui ne contient que « prologin », mais on peut aussi faire plus simplement : en téléchargeant le livre complet (lien « Download ») puis en cherchant « prologin » on voit qu'il apparait à la ligne 3691. Voyant qu'une page fait 40 lignes et qu'une ligne vide sépare les pages, nous déduisons que « prologin » apparait à la page $\lceil 3691/41 \rceil = 91$.

2 Problème de pointure

2.1 Énoncé

Joseph Marchand prépare ses vacances à la montagne. Comme tous les ans, il va skier. Mais comme tous les ans, il doit d'abord vérifier que ses skis sont encore à sa taille.

D'expérience, il sait qu'il peut porter ses skis même s'il y a quelques tailles de différence avec ses pieds (ce qui n'est pas très prudent pour ses chevilles).

Il vous donne la taille de ses pieds et la taille de ses skis. Pouvez-vous lui donner l'écart entre les deux tailles ?

2.2 Solution

On affiche la valeur absolue de la différence entre la taille des pieds de Joseph Marchand et la taille des skis proposés.

Listing 1 – Une solution de l'exercice 1 en Python

```
1 taillePersonne = int(input())
2 tailleSki = int(input())
3
4 print(abs(taillePersonne - tailleSki))
```

3 Les bons skis

3.1 Énoncé

Dans son magasin, Joseph Marchand a loué presque tous ses skis, et certaines tailles sont manquantes. Cependant, il essaie de répondre aux attentes de chaque nouveau client.

Lorsqu'un client entre dans le magasin, Joseph lui demande sa taille et parcourt ensuite les paires de skis qu'il a à sa disposition pour trouver celle qui correspondrait le mieux. Vous pouvez aider Joseph !

Ce dernier vous donne la taille d'une paire de skis désirée par un client (notée A), le nombre de skis qu'il a en réserve (noté N), et une liste de la taille de chacun de ces N skis. En échange vous lui donnez la taille de la paire de skis de son stock la plus proche de la taille de la paire de skis qui correspond à son client. Si plusieurs paires sont à égalité, vous donnerez la plus petite de celles-ci pour économiser du bois ! Il vous en sera très reconnaissant.

3.2 Solution

Il s'agit ici de trouver la paire de skis ayant la taille la plus proche possible de celle de Joseph, et donc de trouver le minimum de différence en valeur absolue (la valeur qu'on cherchait dans l'exercice précédent).

Pour trouver le minimum, on itère sur chaque taille de skis en conservant la meilleure différence et la meilleure taille qu'on ait trouvée jusque là. Si la différence de taille actuelle est meilleure, ou identique mais avec des plus petits skis, on met à jour ces deux valeurs.

Il ne faut pas oublier d'initialiser ces deux variables à une « bonne » valeur avant d'itérer, par exemple la valeur qu'on trouvera pour les premiers skis (c'est ce qu'on fait dans le code ci-dessous).

Complexité On itère une fois sur chaque paire de skis, chaque itération faisant un simple calcul en temps constant. La complexité totale est donc en $\mathcal{O}(N)$.

Listing 2 – Une solution de l'exercice 2 en Python

```
1  def les_bons_skis(nbPersonnes, taillePersonne, skis):
2      meilleure_difference = abs(taillePersonne - skis[0])
3      meilleure_taille = skis[0]
4      for tailleSki in skis:
5          if abs(tailleSki - taillePersonne) < meilleure_difference \
6             or (abs(tailleSki - taillePersonne) == meilleure_difference
7                 and tailleSki < meilleure_taille):
8              meilleure_taille = tailleSki
9              meilleure_difference = abs(tailleSki - taillePersonne)
10     return meilleure_taille
11
12 nbPersonnes = int(input())
13 taillePersonne = int(input())
14 skis = [int(i) for i in input().split()]
15
16 print(les_bons_skis(nbPersonnes, taillePersonne, skis))
```

4 Choix des skis

4.1 Énoncé

Cette année, tout le monde veut faire du ski, mais Joseph Marchand travaille seul et il est débordé ! Il a encore besoin de votre aide. Cette fois vous n'aurez pas à vous occuper d'un client, mais de plusieurs à la fois ! Comme Joseph est prévoyant, il vous a donné un stock de paires de skis aussi grand que le nombre de clients (noté N). Comme Joseph n'a toujours pas toutes les tailles, il vous demande de faire au mieux correspondre ces tailles de skis (notés S_i) aux tailles des clients (notés P_i).

Vous cherchez à distribuer les skis de manière à limiter le plus possible la déception des clients. La déception d'une attribution est la somme, pour chaque client, de la différence entre sa taille et celle de la paire de skis qu'il a reçue. Ainsi Joseph peut être sûr de répondre au mieux aux attentes de tous ses clients.

4.2 Solution

Pour minimiser la déception, on n'a pas d'autre choix de que trier les pointures des clients et les tailles de skis par ordre croissant, puis de les attribuer dans l'ordre.

Pour trier, le plus simple est d'utiliser la fonction de la bibliothèque standard qui existe dans presque tous les langages. On peut aussi coder soi-même un tri efficace, comme le tri fusion ou le tri rapide (il y en a d'autres). Enfin, on attribue les skis au clients dans l'ordre où ils viennent, en retenant la somme des différences.

Complexité Un tri efficace, comme celui de Python, a pour complexité $\mathcal{O}(N \log N)$. On trie deux listes de N éléments, puis on parcourt conjointement ces listes (en ne faisant que des opérations en temps constant), la complexité totale est donc $\mathcal{O}(N \log N)$.

Attention ! La solution consistant à chercher la taille minimisant l'écart pour chaque pointure est en temps $\mathcal{O}(N^2)$. Ici, $N = 100\,000$, c'est donc trop lent pour cet exercice ! Ces algorithmes passaient les tests de correction, mais pas ceux de performance.

Listing 3 – Une solution de l'exercice 3 en Python

```
1  def choix_des_skis(nbPersonnes, personnes, skis):
2      personnes.sort()
3      skis.sort()
4      difference = 0
5      for i in range(nbPersonnes):
6          difference += abs(personnes[i] - skis[i])
7      return difference
8
9  nbPersonnes = int(input())
10 personnes = [int(i) for i in input().split()]
11 skis = [int(i) for i in input().split()]
12
13 print(choix_des_skis(nbPersonnes, personnes, skis))
```

5 Maximise ton fun

5.1 Énoncé

Joseph Marchand en a marre du boulot ! Pour se changer les idées, il décide d'aller au ski.

Après une matinée à essayer toutes les pistes, Joseph se rend compte qu'il perd parfois son temps sur des pistes qui le déprime. Il se demande alors s'il ne peut pas optimiser son fun.

Pour cela, il dispose de la carte de la station. La station est composée de différentes plateformes, reliées par des pistes orientées : l'existence d'une piste partant de la plateforme 0 vers la plateforme 1 n'implique pas forcément l'existence d'une piste partant de la plateforme 1 vers la plateforme 0. Pour chacune d'elles, Joseph a recueilli la quantité de fun, éventuellement négative, qu'elle lui apporte.

Partant de sa plateforme initiale avec une quantité de fun nulle, il va emprunter une succession de pistes et ajouter à sa quantité de fun la dose de fun apportée par chacune de ces pistes. Sa dose de fun peut éventuellement devenir négative. De plus, comme rien ne lui interdit de repasser par une piste qu'il a déjà empruntée, il se peut, dans certains cas, qu'il puisse rendre sa quantité personnelle de fun aussi grande qu'il le désire. De plus, il est tout-à-fait possible qu'il reste sur sa plateforme initiale, sans bouger.

Votre but va être de trouver, étant donnée la liste des pistes, et sachant qu'il démarre de la première des plateformes (numérotée 0), quel est le fun maximal que peut atteindre Joseph Marchand, ou d'indiquer s'il peut avoir autant de fun qu'il le désire.

5.2 Correction

Dans ce problème, on demandait de renvoyer la longueur maximale d'une marche (i.e. après avoir emprunté une suite de sommets reliés par des pistes) dans le graphe des stations. On pourra restreindre notre graphe uniquement aux stations accessibles depuis le point de départ de Joseph Marchand (pour ce faire, on peut faire un simple parcours depuis cette station et garder uniquement les sommets visités).

Très rapidement, on remarque que ce problème se ramène à celui du plus court chemin dans un graphe : il suffit de changer le signe du poids de chaque arête. On est alors tenté d'appliquer un algorithme classique de plus court chemin, comme par exemple l'algorithme de Dijkstra.

Le problème, c'est que l'algorithme de Dijkstra ne fonctionne que sur les graphes dont toutes les arêtes ont un poids positif, ce qui n'est pas le cas de notre graphe (sur la figure, le chemin trouvé par l'algorithme de Dijkstra est sv de poids 1, alors que suv est de poids 0).

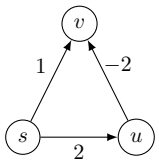


FIGURE 1 – Échec de l'algorithme de Dijkstra

Soit, trouvons donc un algorithme pour trouver le plus long chemin d'un graphe orienté pondéré dans le cas général ! S'il existe bien une solution en $\mathcal{O}(n2^n)$ (un simple algorithme dynamique qui stocke pour chaque sous-ensemble de sommets du graphe le poids maximal d'un chemin sur ces sommets), cela est bien trop lent.

Et pour cause : si on avait une solution polynomiale pour ce problème, on pourrait, pour tout graphe orienté, facilement résoudre le problème du chemin hamiltonien sur celui-ci en pondérant par 1 toutes ses arêtes. Malheureusement, ce problème est connu comme NP-complet, ce qui semble compromettre nos chances de succés [3].

3. Prologin est prêt à payer 500000$ tout candidat qui aurait un tel algorithme

L'astuce, c'est que si le problème de trouver un plus long chemin dans un graphe est NP-complet, celui de trouver une plus longue marche ne l'est pas, et c'est bien ce que le sujet vous demande ! Pour rappel, un chemin est une marche qui emprunte chaque sommet au plus une fois : en particulier, un chemin est fini, et donc sa longueur aussi.

Mais comment calculer une plus longue marche dans un graphe orienté pondéré ?

Il faut remarquer que soit il y a des marches de longueurs arbitrairement grandes, soit, si ce n'est pas le cas, il existe une marche de longueur maximale, qui correspond exactement àla longueur du plus long chemin dans le même graphe.

On pourrait alors agir ainsi :
— chercher s'il y a des marches arbitrairement grandes
— si ce n'est pas le cas, chercher un plus long chemin

Nous avons vu que le second problème est difficile dans le cas général, voyons dans quels cas nous aurons à nous en occuper, puisque cela pourrait nous restreindre à tel point que le problème devienne polynomial : cherchons donc quand on peut trouver des marches de plus en plus grandes !

En générant plusieurs graphes comme exemples et en exhibant les plus longues marches, l'intuition arrive vite : il existe des marches arbitrairement grandes si et seulement si le graphe admet un circuit de poids strictement positif. En réalité, il n'y a besoin pour notre problème de ne démontrer qu'une seule des deux implications.

Pour ce faire, on remarque que s'il y a un circuit de poids strictement positif, il suffit d'aller jusqu'à celui-ci, puis de l'emprunter autant de fois que l'on veut : chaque tour du cycle augmente strictement le poids de la marche. On peut donc trouver des marches de plus en plus longues.

On peut donc supposer que notre graphe ne possède pas de cycle de poids strictement positif. Or, il existe un algorithme qui permet de résoudre le problème du plus long chemin dans un tel graphe : l'algorithme de Bellman-Ford [4].

Si, comme Dijkstra, il peut être intéressant de connaître cet algorithme, il est néanmoins très facile à retrouver si on ne le connaît pas : il suffit de remarquer que dans un tel graphe, on peut reconstruire facilement tous les plus longs chemins passant par au plus r sommets en connaissant les plus longs chemins passant par au plus $r-1$ sommets.

On va donc simplement faire un algorithme dynamique stockant, pour chaque sommet du graphe et chaque $r \in \{1, ..., n\}$, le plus gros poids d'un chemin passant par au plus r sommets de la source vers celui-ci.

En prime, on se rend compte qu'on peut détecter la présence d'un cycle positif simplement en regardant s'il existe un chemin passant par au plus $r+1$ sommets de poids supérieur :

Listing 4 – Une solution de l'exercice 4 en OCaml

```
let _ =
  let ($) a b = if min a b = min_int/2 then min_int/2 else a+b in
  let n, m = Scanf.scanf "%d %d" (fun n m -> n, m) in
  let l = Array.(to_list
    init m (fun _ -> Scanf.scanf " %d %d %d" (fun u v w -> (u,v,w))))
  in
```

4. Il permet aussi de résoudre le problème du plus court chemin dand un graphe sans cycle négatif, ce qui est équivalent à notre problème en remplaçant les poids par leur opposé.

```
7    let dist = Array.init n (fun i -> if i > 0 then min_int / 2 else 0) in
8      for k = 1 to n do
9        List.iter (fun (u,v,w) -> dist.(v) <- max dist.(v) (dist.(u)$w)) l
10     done;
11     let x = Array.copy dist in
12       List.iter (fun (u,v,w)-> dist.(v) <- max dist.(v) (dist.(u)$w)) l;
13       if x <> dist
14         then print_string "OVERDOSE DE FUN"
15         else print_int (Array.fold_left max min_int x)
```

On aurait pu s'intéresser à ce problème dans le cas où notre graphe n'était pas orienté : il resterait polynomial, mais deviendrait beaucoup plus dur puisqu'on se ramènerait alors à un problème de couplage maximal dans un graphe général !

6 Taxi des neiges

6.1 Énoncé

Joseph Marchand a décidé de se lancer dans une nouvelle aventure, qu'il espère la plus lucrative possible : le taxi-motoneige, avec sa SUPERBE motoneige Nimbus 4242™.

Pour ce faire, il reçoit un certain nombre M de requêtes de clients, qu'il doit aller chercher à leur position actuelle, puis emmener à destination. Heureusement pour lui, les clients savent la chance qu'ils ont de pouvoir admirer une motoneige Nimbus 4242™, et n'en voudront en conséquence pas à Joseph s'il prend son temps pour les chercher ou s'il les transporte à leur destination de manière particulièrement inefficace. Si, durant une course, Joseph Marchand veut s'arrêter pour observer sa motoneige Nimbus 4242™, il le peut : quel veinard !

Il peut aussi, quand il le désire, s'arrêter dans différentes stations-services situées sur la carte (au nombre de N) pour se recharger en carburant. Comme les pompistes admirent sa motoneige Nimbus 4242™, le carburant est gratuit pour notre chauffeur de taxi.

Intéressons-nous maintenant à sa motoneige Nimbus 4242™. Il s'agit d'une motoneige quatre places : c'est à dire qu'elle peut accueillir Joseph et au plus trois clients. De plus, elle fonctionne beaucoup moins bien quand elle a moins de carburant : plus exactement, s'il parcourt une certaine distance en kilomètres entre deux passages dans des stations-service, il aura consommé un litrage de carburant égal au carré de cette distance (calculée, évidemment, à vol d'oiseau, car la motoneige Nimbus 4242™ ne s'embarasse pas des obstacles).

Devant changer son réservoir, Joseph aimerait bien savoir quel est le volume minimal, en litres, qu'il doit supporter pour pouvoir répondre à toutes les requêtes, sachant qu'il commence dans la première station décrite, qu'il doit rentrer dans icelle après avoir satisfait toutes les requêtes et qu'il n'existe que des réservoirs avec des contenances entières.

6.2 Correction

L'enrobage de ce problème était en soi une difficulté, mais plusieurs remarques permettent de se rapprocher d'un problème connu.

Tout d'abord, remarquons que nos déplacements ne se font que par des lignes droites : en effet, on n'a rien à gagner à faire des trajectoires courbes, vu que les déplacements seront tous de la forme station/départ de requête/arrivée de requête-station/départ de requête/arrivée de requête. On caractérise donc totalement notre solution uniquement avec une liste de stations/départs de requête/arrivées de requête.

Tous nos déplacements seront de la forme station-station, départ/arrivée de requête-station ou station-départ/arrivée de requête. Ceci se démontre très facilement par récurrence sur la longueur du chemin optimal en cours :

- s'il finit par une station, tous les déplacements restent autorisés
- supposons qu'il finisse par un départ/arrivée de requête alors, d'après l'hypothèse de récurrence, on avait précedemment visité une station : d'après l'inégalité triangulaire [5] on n'a rien à perdre à d'abord rentrer sur cette station avant d'atteindre un départ/arrivée de requête suivant.

De plus, la limite du nombre de personnes sur la motoneige est totalement inutile : notre efficacité est mesurée par le volume de carburant maximal entre deux stations successives, on n'empire donc pas notre solution en effectuant plusieurs fois de suite notre cycle-solution (puisqu'il faut finir sur le lieu de départ).

Plus important, on peut totalement dissocier les départs et arrivées de requêtes. En effet, le seul problème serait qu'on ait une arrivée avant le départ de la requête idoine. Il suffirait alors de ré-effectuer notre cycle solution, mais dans le sens opposé !

5. Dans un triangle la longueur de chaque côté est inférieure à la somme des deux autres.

On peut donc simplement doubler le nombre de requêtes, et considérer que chaque requête est un point du plan. En fait, on peut même simplement traiter chaque requête séparément, on factorisera la partie importante et lente du code pour ne pas la répéter.

Soit T l'ensemble des stations visitées dans une solution optimale. D'après l'inégalité triangulaire, on ne perd rien à ne pas emprunter de requête pour les visiter. On peut donc facilement associer à chaque station i de T le score minimal T_i pour accéder jusqu'à cette station depuis le sommet initial ; il suffit de procéder comme durant l'exécution de l'algorithme de Prim. On part d'un ensemble de stations réduit à la station initiale. À chaque étape, on cherche la station i non encore visitée qui minimise la distance aux stations de l'ensemble. On ajoute alors cette station à l'ensemble et on obtient la valeur de T_i comme étant $\max(T_j, \text{dist}(i,j))$ où j est la station de l'ensemble la plus proche de i. On réitère ensuite jusqu'à ce que toutes les stations soient visitées.

Étant donné un point-requête x que l'on veut atteindre en minimisant notre score, on peut simplement regarder pour chaque station i de T laquelle minimise $\max(T_i, 2\text{dist}(i,x))$, ce qui peut se faire en $\mathcal{O}(NM)$.

En effectuant l'opération pour chaque x et en gardant le maximum, on obtient donc notre score final.

Le seul problème consiste en la création de T qui prendrait un temps quadratique. Néanmoins, on peut réduire ce temps à du semi-linéaire en le nombre de stations en utilisant une hypothèse qu'on a négligée jusque là. Imaginons que lors de la création de T, on ait relié à chaque étape les stations i et j. On peut montrer que le graphe[6] obtenu est un sous-graphe du graphe induit par la triangulation de Delaunay[7] des stations. Autrement dit, si les stations i et j sont reliées dans T, c'est qu'elles sont reliées dans une triangulation de Delaunay des stations.

Un algorithme possible est alors de calculer cette triangulation, ce qui peut se faire en $\mathcal{O}(N\log(N))$, puis d'utiliser l'algorithme de Prim (décrit plus haut) sur le graphe obtenu. En remarquant que le graphe issu de la triangulation est planaire[8], on sait qu'il suffit de considérer seulement un nombre réduit d'arêtes (un nombre linéaire au lieu de quadratique). La complexité pour trouver T est donc réduite à $\mathcal{O}(N\log(N))$.

Il n'était toutefois pas nécessaire de faire cette triangulation pour passer le système de test. Une construction de cet arbre en $\mathcal{O}(N^2)$ était acceptée, et toute solution du problème en $\mathcal{O}(N(N+M))$ passait donc les tests.

Voici une solution en OCaml en $\mathcal{O}(N(\log(N)+M))$:

Listing 5 – Une solution de l'exercice 5 en OCaml

```
let (min_abs, max_abs, min_ord, max_ord) = (-10001,10001,-10001,10001)

(* Calcule le déterminant de deux vecteurs *)
let det (w,x) (y,z) = w*z - x*y

(* Calcule le produit scalaire de deux vecteurs *)
let dot (w,x) (y,z) = w*y + x*z

(* Arrondit à l'unité *)
let round x =
```

6. qui est un arbre !
7. Pour plus de détails, voir https://fr.wikipedia.org/wiki/Triangulation_de_Delaunay
8. On peut le dessiner sans croisement d'arêtes.

```
11      if x -. floor x < 0.5
12          then floor x
13          else ceil x
14
15  (* Calcule la distance euclidienne entre deux points du plan *)
16  let dist ((a,b),(c,d)) =
17     [a-c;b-d]
18     |> List.map (fun i -> i*i)
19     |> List.fold_left (+) 0
20     |> float
21     |> sqrt
22
23  (* Calcule le déterminant d'une matrice 3*3 *)
24  let det_three mat =
25      let a = mat.(0).(0) and b = mat.(0).(1) and c = mat.(0).(2)
26      and d = mat.(1).(0) and e = mat.(1).(1) and f = mat.(1).(2)
27      and g = mat.(2).(0) and h = mat.(2).(1) and i = mat.(2).(2) in
28          a*e*i + b*f*g + c*d*h - g*e*c - h*f*a - i*d*b
29
30  (* Renvoie le signe de x ie si x = 0, cela renvoie 0, sinon x / |x| *)
31  let sgn x =
32      if x > 0
33          then 1
34          else if x = 0
35              then 0
36              else -1
37
38  (* La triangulation *)
39  (* Je ne vais pas trop commenter la partie "triangulation de Delaunay" qui est
        ↪ classiquissime.
40  Remarquons simplement que chaque triangle a un field "emplacement" qui sera utile
        ↪ après la triangulation *)
41  type vertex_opp = {mutable coord: int*int; mutable opp: triangle ref}
42  and triangle = {mutable emplacement : int; mutable is_empty: bool;
43  mutable suiv: triangle ref list; mutable ver: vertex_opp array}
44
45  let init_tri (coord_a,opp_a) (coord_b,opp_b) (coord_c,opp_c) =
46      {emplacement= -1;is_empty=false; suiv = [];
47      ver = [|{coord = coord_a;opp = opp_a};
48              {coord = coord_b;opp = opp_b};
49              {coord = coord_c;opp = opp_c}|]}
50
51  let in_circumscribed (a,b) ((c,d),(e,f),(g,h)) =
52      sgn  det (e-c,f-d) (g-c,h-d) * sgn (det_three
53      [|[|c-a;e-a;g-a|];
54      [|d-b;f-b;h-b|];
55      [|c*c-a*a + d*d-b*b;e*e-a*a + f*f-b*b;g*g-a*a+h*h-b*b|]|])
56
57  let have_to_be_switched triangle_to_test i =
58      let tri = !triangle_to_test in
59      let vois_opp = !(tri.ver.(i).opp) in
60          (not vois_opp.is_empty) && (in_circumscribed tri.ver.(i).coord
61          (vois_opp.ver.(0).coord,vois_opp.ver.(1).coord,vois_opp.ver.(2).coord) > 0)
```

```
62
63 let replace_by_in replaced replacing tri =
64    if not tri.is_empty
65       then for i = 0 to 2 do
66          if tri.ver.(i).opp == replaced
67             then tri.ver.(i).opp <- replacing;
68           done
69
70 let index_opp i tri =
71    let tri' = (!tri).ver.(i).opp in
72       if (!tri').is_empty
73          then -1
74          else if (!tri').ver.(0).opp == tri
75             then 0
76             else if (!tri').ver.(1).opp == tri
77                then 1
78                else 2
79
80 let switch tri i =
81    let search_opp coordonnees tri =
82       if (!tri).ver.(0).coord = coordonnees
83          then (!tri).ver.(0).opp
84          else if (!tri).ver.(1).coord = coordonnees
85             then (!tri).ver.(1).opp
86             else (!tri).ver.(2).opp
87    in
88    let tri' = (!tri).ver.(i).opp in
89    let j = index_opp i tri in
90    let (a,b,c,d) = ((!tri).ver.(i).coord ,(!tri).ver.((i+1) mod 3).coord ,(!tri).
         ↪ ver.((i+2) mod 3).coord,(!tri').ver.(j).coord) in
91    let (e,f,g,h) = (search_opp b tri',search_opp c tri',search_opp b tri,search_opp
         ↪  c tri) in
92    let tri''' = ref {emplacement= -1;is_empty=false;suiv = [];ver = [|{coord=a;opp=
         ↪ f};{coord=b;opp=tri'};{coord=d;opp=h}|]}
93    and tri'''' = ref {emplacement= -1;is_empty=false;suiv = [];ver = [|{coord=a;opp
         ↪ =e};{coord=c;opp=tri};{coord=d;opp=g}|]} in
94       (!tri''').ver.(1).opp <- tri'''';
95       (!tri'''').ver.(1).opp <- tri''';
96       replace_by_in tri tri'''' !g;
97       replace_by_in tri' tri'''' !e;
98       replace_by_in tri' tri''' !f;
99       replace_by_in tri tri''' !h;
100      (!tri).suiv <- [tri''';tri''''];
101      (!tri').suiv <- [tri''';tri''''];
102      [tri]
103
104 let split_in_three x tri =
105    let (coor_a,coor_b,coor_c,vois_a,vois_b,vois_c) = ((!tri).ver.(0).coord,(!tri).
         ↪ ver.(1).coord,
106    (!tri).ver.(2).coord,(!tri).ver.(0).opp,(!tri).ver.(1).opp,(!tri).ver.(2).opp)
         ↪ in
107    let tri1 = ref {emplacement= -1;is_empty=false;suiv=[];ver=[||]} and tri2 = ref
         ↪ {emplacement= -1;is_empty=false;suiv=[];ver=[||]}
```

```
108     and tri3 = ref {emplacement= -1;is_empty=false;suiv=[];ver=[||]} in
109         (!tri1).ver <- [|{coord=coor_a;opp=tri3};{coord=coor_b;opp=tri2};{coord=x;opp
            ↪ =vois_c}|];
110         (!tri2).ver <- [|{coord=coor_a;opp=tri3};{coord=coor_c;opp=tri1};{coord=x;opp
            ↪ =vois_b}|];
111         (!tri3).ver <- [|{coord=coor_c;opp=tri1};{coord=coor_b;opp=tri2};{coord=x;opp
            ↪ =vois_a}|];
112         replace_by_in tri tri1 !vois_c;
113         replace_by_in tri tri2 !vois_b;
114         replace_by_in tri tri3 !vois_a;
115         (!tri).suiv <- [tri1;tri2;tri3];
116         [tri1;tri2;tri3]
117
118 let is_in_triangle x tri =
119     let direction (a,b) ((c,d),(e,f)) =
120         let determinant = det (a-e,b-f) (c-e,d-f) in
121               determinant >= 0, determinant <=0
122     in
123     let (c,d,e) = (tri.ver.(0).coord,tri.ver.(1).coord,tri.ver.(2).coord) in
124     let flag,drap = direction x (c,d) and flag',drap' = direction x (d,e)
125     and flag'',drap'' = direction x (e,c) in
126         (flag && flag' && flag'') || (drap && drap' && drap'')
127
128 let rec in_which_triangle x = function
129     |t::q ->if is_in_triangle x !t
130                then if (!t).suiv = []
131                    then t
132                    else in_which_triangle x  (!t).suiv
133                else in_which_triangle x q
134     | _   -> failwith "Empty Triangle List"
135
136 let insert x l =
137     let rec aux = function
138         | [] -> ()
139         | tri::q -> begin
140               if (!tri).suiv = []
141                   then if have_to_be_switched tri 0
142                       then aux  (switch tri 0)qelse if have_to_be_switched tri 1then
                            aux  (switch tri 1)q
143                           else if have_to_be_switched tri 2
144                               then aux  (switch tri 2)qelse aux qelse aux ((!tri).suiv
                                    q)
145                   end
146     in
147     let tri = in_which_triangle x l in
148     let to_verify = split_in_three x tri in
149         aux to_verify
150
151 (* La fonction finale pour trianguler, qui ne fait qu'insérer les points les uns
    ↪ après les autres.
152 L'astuce est de prendre pour cas initial les triangles ((-10001,-10001)
    ↪ ,(-10001,10001),(10001,10001))
153 et ((10001,-10001),(10001,10001),(-10001,-10001)), étant donné que l'énoncé
    ↪ contraint les points à être
```

```
154 dans l'un de ces deux triangles *)
155 let rec delaunay = function
156     | [] -> begin
157         let nul = ref {emplacement= -1;is_empty=true; suiv = [];ver= [||]} in
158         let tri1 = ref  init_tri ((min_abs,max_ord),nul)
159   ((min_abs,min_ord),nul) ((max_abs,max_ord),nul)
160   and tri2 = ref  init_tri ((max_abs,min_ord),nul)
161   ((max_abs,max_ord),nul) ((min_abs,min_ord),nul)
162   in (!tri1).ver.(0).opp <- tri2;
163           (!tri2).ver.(0).opp <- tri1;
164           [tri1;tri2]
165         end
166     | t::q -> let l = delaunay q in
167     (insert t l;l)
168
169 (*
170 type vertex_opp = {mutable coord: int*int; mutable opp: triangle ref}
171 and triangle = {mutable emplacement : int; mutable is_empty: bool;
172 mutable suiv: triangle ref list; mutable ver: vertex_opp array}
173 *)
174 let vert_tri tri =
175     if tri.is_empty
176         then []
177         else Array.(to_list (map (fun x -> x.coord) tri.ver))
178
179 let all_tri tri =
180    List.filter (fun x -> not x.is_empty)
181    (tri::(Array.to_list (Array.map (fun x -> !(x.opp)) tri.ver)))
182
183 (* Given a delaunay list and a point, outputs the three nearest points
184 of the delaunay list *)
185 let three_nearest l point =
186     let tri = !(in_which_triangle point l) in
187         List.(flatten (map vert_tri (all_tri tri)))
188
189 let edges tri =
190     let tab = Array.map (fun x -> x.coord) tri.ver in
191         [tab.(0),tab.(1);tab.(1),tab.(2);tab.(2),tab.(0)]
192
193 (* Fin de la triangulation *)
194 (* Etant donnée une liste de points, renvoie le tableau de la triangulation de
        ↪ Delaunay correspondante, où chaque triangle a été
195 numéroté (numérotation conservée au niveau de la valeur "emplacement") *)
196 let pts_to_tab list_obj list_points =
197     let counter = ref 0 and res = ref [] and l = delaunay list_points
198     in
199     let rec aux = function
200         | t::q when (!t).emplacement = -1 -> begin
201               if (!t).suiv = []
202               then (res := t::!res; (!t).emplacement <- !counter; incr counter)
203               else ((!t).emplacement <- 0; aux  (!t).suiv);
204               aux q
205                                       end
```

```
206         | _                           -> ()
207     in aux l;
208     List.map (!) !res
209
210
211 (*
212 ---------------------------------------------------------------------------
213 *)
214
215 module Int =
216 struct
217   type t = int
218   let compare = Pervasives.compare
219 end
220
221 module Pair (X : Set.OrderedType) (Y : Set.OrderedType) =
222 struct
223   type t = X.t * Y.t
224   let compare (a,b) (c,d) =
225     match X.compare a c with
226       | 0 -> Y.compare b d
227       | x -> x
228 end
229
230 module IntInt = Pair(Int)(Int)
231 module IntIntInt = Pair(Int)(IntInt)
232 module IISet = Set.Make(IntInt)
233 module IIISet = Set.Make(IntIntInt)
234 module IIMap = Map.Make(IntInt)
235
236 let map_max x map =
237     IIISet.(fold (fun (y,z) -> add (max x y,z)) map empty)
238
239 let prim grph init =
240   let rec aux seen heap lst_edg =
241       if IIISet.is_empty heap
242       then lst_edg else
243       let (a,b) as e = IIISet.min_elt heap in
244         aux (IISet.add b seen)
245         IIISet.(remove e (if IISet.mem b seen then heap else union (map_max a (
              ↪ IIMap.find b grph)) heap))
246         (if IISet.mem b seen then lst_edg else IIMap.add b a lst_edg)
247   in
248     aux IISet.empty (IIISet.singleton (0,init)) IIMap.empty
249
250 let sq_dist (a,b) (c,d) = ((a-c)*(a-c)) + ((b-d)*(b-d))
251
252 let add_edge (x,y) grph =
253     let open IIMap in
254     let d = sq_dist x y in
255     let nei_x = try find x grph with Not_found -> IIISet.empty in
256     let nei_y = try find y grph with Not_found -> IIISet.empty in
257     let new_x = IIISet.add (d,y) nei_x in
```

```
258     let new_y = IIISet.add (d,x) nei_y in
259         add x new_x (add y new_y grph)
260
261 let knowing_grph grph init near =
262   let lst_edg = prim grph init in
263     near
264     |> IIMap.mapi (fun x nei -> IISet.fold (fun y ->
265       min (max (IIMap.find y lst_edg) (4 * sq_dist x y))) nei max_int)
266     |> (fun map -> IIMap.fold (fun _ -> max) map 0)
267
268 (* Bootstrap both parts *)
269 let final list_obj list_stat init =
270     let del = pts_to_tab list_obj list_stat in
271     let nearest = List.map (fun x -> x, list_stat) list_obj in
272     let add_tri tri = List.fold_right add_edge (edges tri) in
273     let grph = List.fold_right add_tri del IIMap.empty in
274     let add_nei (x,l) =
275         IIMap.add x IISet.(List.fold_right add l empty)
276     in
277         knowing_grph grph init
278         (List.fold_right add_nei nearest IIMap.empty)
279 let _ =
280     let (nb_stat,nb_req) = Scanf.scanf "%d %d" (fun i j -> (i,j)) in
281     let read_coup _ = Scanf.scanf " %d %d" (fun i j -> (i,j)) in
282     let stations = Array.(to_list (init nb_stat read_coup)) in
283     let req = Array.(to_list (init (2 * nb_req) read_coup)) in
284         print_int (final req stations (List.hd stations))
```

*
**

Félicitations à tous les participants !

Nous avons tenté de rédiger une correction aussi claire que possible. Néanmoins, si vous avez des questions, n'hésitez pas à nous contacter à l'adresse info@prologin.org.