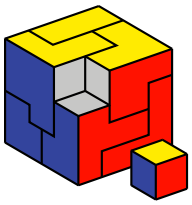
Prologin 2019



Concours National d'Informatique

Correction de la phase de sélection

Correction des qualifications Prologin 2019

Thibault Allançon, Rémi Dupré, Joël Felderhoff, Garance Gourdel*

1 janvier 2019

Table des matières

*Pour l'équipe des correcteurs 2019 : .

0 Questions ouvertes

0.1 Question 1 : IOI

Énoncé Combien de médailles ont été remportées par l'équipe française aux IOI depuis sa création ?

Correction La france a remporté **47** médailles aux IOI depuis sa création, cette information peut être retrouvée sur le site de france-ioi.

0.2 Question 2 : Language fonctionnel français

Énoncé Lequel de ces langages est fonctionnel et français ?

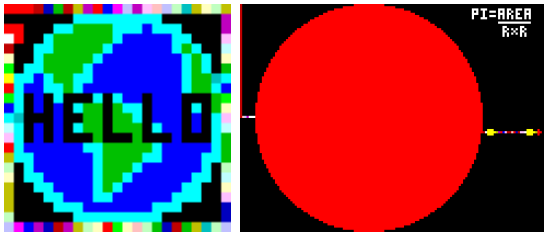
1. OCaml
2. JavaScript
3. GOTO++
4. Haskell

Correction La réponse attendue était **OCaml**.

0.3 Question 3 : Piet

Énoncé Quelles sont les sorties des programmes Piet suivants ?

Écrivez les sorties des deux programmes séparées par un point-virgule : `Sortie du premier programmep;Sortie du second programme`.



Correction La réponse attendue était : **Hello, world !;31405**.

Il existe des interpréteurs de Piet en ligne, il suffisait de leur fournir les images après les avoir téléchargé depuis notre site, ou bien directement leur fournir les URL des images.

Il était également possible de 'contourner' la question en faisant une recherche par image sur un moteur de recherche, on peut alors tomber sur des sites référençant plein de programmes intéressants 'écrits' en Piet.

0.4 Question 4 : Tours de Hanoï

Énoncé Quel est le nombre de déplacements de disque minimal à effectuer pour déplacer une tour de Hanoï à 16 disques ?

Correction Il faut effectuer au minimum **65535** déplacements. On peut trouver qu'en règle générale le problème demande $2^n - 1$ déplacements pour une tour de Hanoï à n disques, on n'a plus qu'à remplacer n par 16.

0.5 Question 5 : Versions TeX

Énoncé Vers quelle célèbre constante mathématique se rapprochent les numéros de version successif de TeX ?

Cette constante a un nom, donnez-le en lettres minuscules.

Correction La constante en question n'est autre que **pi**. La version actuelle de TeX est 3.14159265.

0.6 Question 6 : Bad JavaScript

Énoncé Dans le langage JavaScript, laquelle de ces expressions retourne `k+kcfalse` au lieu de `k+kctrue` ?

1. `k+kctrue o+ k+kctrue o=== k+kctrue o+ l+m+mi1 o k+kctrue o=== l+m+mi1`
2. `l+m+mi1n+nxe16 o=== l+m+mi1n+nxe16 o+ l+m+mi1 o l+m+mi1n+nxe16 o!== l+m+mi1n+nxe16 o+ l+m`
3. `l+m+mi3 o* l+m+mf0.1 o!== l+m+mi2 o* l+m+mf0.15 o l+m+mi3 o* l+m+mf0.01 o=== l+m+mi2 o* l`
4. `o!+p[]o+p[]o+!p[] o=== l+s+s1true o+ p(k+kcNaN o=== k+kcNaNp)`

Correction On attendait la **1ère** réponse. Le plus simple était d'ouvrir une console JavaScript et d'y copier chaque ligne pour voir le résultat. Il est très simple d'accéder à une telle console, la plupart des navigateurs web modernes en intègrent une, sur Firefox il suffit d'appuyer sur F12 ou bien Ctrl+Maj+S puis d'aller à l'onglet *console*.

Voici une courte explication pour chaque ligne :

1. `k+kcfalse` : le second membre est faux car le type de `k+kctrue` en javascript est `l+s+s2boolean` et celui de `l+m+mi1` est `l+s+s2number`. L'opérateur `o===` n'autorise pas l'égalité entre deux termes de types différents en javascript.
2. `k+kctrue` : le nombre 10^{16} est assez grand pour que les flottants 64 bits utilisés par JavaScript ne soient pas précis à l'unité près mais suffisament petit pour qu'ils le soient à deux unités près.
3. `k+kctrue` : les flottants donnent lieu à des imprécisions parfois difficiles à prédire. Ici, la plupart des calculs ne comportent pas de difficulté à part `l+m+mi3 o* l+m+mf0.1` qui vaut `l+m+mf0.30000000000000`
4. `k+kctrue` : l'opérateur d'addition de JavaScript échoue rarement, lorsqu'il ne sait pas comment additionner deux éléments, il les convertis en chaîne de caractère et en retourne la concaténation. Dans cet example, les deux membres valent `l+s+s2truefalse`. À noter que le flottant *not a number* n'est égal à rien, même pas à lui même et donc `k+kcNaN o=== k+kcNaN` vaut `k+kcfalse`.

0.7 Question 7 : timestamp UNIX

Énoncé Laquelle de ces dates ne peut pas être représentée à l'aide d'un timestamp sur un système UNIX ?

1. 25 août 1991
2. 2 décembre 1970
3. 2 mars 1969
4. 13 janvier 1997

Correction La bonne réponse était le **2 mars 1969**. En effet, un timestamp est un entier positif représentant le nombre de secondes écoulées entre le premier janvier 1970 à minuit UTC et la date à représenter.

0.8 Question 8 : générateur congruentiel linéaire

Énoncé Quelle est le $333^{\text{ième}}$ nombre aléatoire X_{333} généré par un générateur congruentiel linéaire de la forme $X_{i+1} = 1337X_i + 404 \mod 1000$ en utilisant la graine $X_1 = 42$?

On va ici considérer que X_1 est le premier nombre donné par notre générateur.

Correction La réponse attendue est **322**. Il faut calculer le $333^{\text{ième}}$ terme de la suite $(X_n)_{n\geq 1}$ définie ci-dessus, voici par exemple un script python qui le calcule :

Listing 1 – Un algorithme pour répondre à la question 8 du QCM

```python
# Premier terme X_1 de la suite
x = 42

for i in range(1, 333):
    # x a pour valeur X_i, on peut calculer X_{i+1}
    x = (1337 * x + 404) % 1000

# Finalement, x vaut X_333
print(x)
```

À noter que bien que cette méthode puisse paraître rudimentaire, faire générer un nombre aléatoire par un ordinateur est une réelle difficulté, et avec de meilleurs paramètres ce type de méthode est très utilisé.

1 Arnaque aérienne

1.1 Énoncé

Comme le disait le grand-père de Joseph Marchand, « Le plus beau voyage est celui qu'on n'a pas encore fait ». New York était dans la tête de Joseph depuis plusieurs années maintenant, et il a décidé aujourd'hui d'acheter son billet d'avion.

Quelques secondes avant de cliquer sur le bouton « Acheter ! », son amie Haruhi lui envoie une liste de prix qu'elle a trouvé sur Internet. Joseph est curieux de voir si ces derniers sont moins chers que le billet qu'il s'apprêtait à acheter.

Entrée

Sur la première ligne le prix initial du billet de Joseph.

Sur la deuxième ligne un entier N, correspondant au nombre de billets envoyés par Haruhi. La ligne suivante contient les N prix trouvés par Haruhi.

Sortie

Si Haruhi a trouvé au moins 3 prix strictement moins chers que celui de Joseph, affichez « ARNAQUE ! » pour l'avertir. Sinon « Ok bon voyage, bisous, n'oublie pas de m'envoyer des photos ! ».

Contraintes

- $1 \leq N \leq 100$
- $1 \leq prix \leq 2000$

1.2 Solution

Le sujet nous demande de vérifier si on peut trouver dans une liste d'entiers, au moins trois entiers strictement inférieurs à un nombre P.

Pour cela, nous faisons un simple parcours dans la liste des prix, et nous gardons un compteur que l'on incrémente à chaque fois que la condition $p_i < P$ est vérifiée.

Listing 2 – Une solution de l'exercice 1 en Python

```
1  prix_initial = int(input())
2  nb_prix = int(input())
3  prix = list(map(int, input().split()))
4
5  nb_inf = 0
6  for p in prix:
7      if p < prix_initial:
8          nb_inf += 1
9
10 if nb_inf >= 3:
11     print("ARNAQUE !")
12 else:
13     print("Ok bon voyage, bisous, n'oublie pas de m'envoyer des photos !")
```

2 Méli-mélo temporel

2.1 Énoncé

Joseph et Haruhi sont maintenant partis dans leurs voyages endiablés. Ils ne voyagent pas dans les mêmes pays et sont donc sur des fuseaux horaires différents. Pour toujours savoir combien de décalage ils ont entre eux, ils ont noté le décalage de tous les pays par rapport à la France. Maintenant à chaque fois que l'un ou l'autre arrive dans un pays, ils veulent prévenir l'autre du décalage entre eux.

Entrée

L'entrée comprendra :

— Un entier N le nombre de pays que visiteront Joseph et Haruhi.

— Sur les N lignes suivantes, un entier 1 i N le numéro du pays et un entier relatif di le décalage entre la France et le pays i.

— Sur la ligne suivante un entier V, le nombre de voyages que font Joseph et Haruhi.

— Sur les V lignes suivantes, un entier 1 ou 2 pour indiquer un déplacement de Haruhi ou Joseph et un entier positif indiquant le numéro du pays.

Sortie

Pour chaque déplacement de Haruhi ou Joseph, vous afficherez leurs décalages horaire en valeur absolue.

Contraintes

- $1 \leq N \leq 1000$
- $1 \leq V \leq 1000$
- $-12 \leq d_i \leq 12$

2.2 Solution

Il faut après chaque déplacement de Haruhi ou Joseph donner le décalage horaire qu'il y a entre eux deux.

Comme les décalages sont tous donnés par rapport à la France, il suffit de faire la soustraction en valeur absolue du décalage (par rapport à la France) du pays où est Haruhi et celui où est Joseph.

$$décalage = |décalage\ pays\ Haruhi - décalage\ pays\ Joseph|$$

On répète ça pour chaque étape et on obtient la solution suivante :

Listing 3 – Une solution de l'exercice 2 en Python

```
N = int(input())
d = [0] * (N + 1)

for i in range(0, N):
    n, d_n = map(int, input().split())
    d[n] = d_n

V = int(input())
position = {"1": 0, "2": 0}

for i in range(0, V):
    traveler, country = input().split()
    country = int(country)
```

```
14      traveler = str(traveler)
15
16      position[traveler] = country
17      lag = abs(d[position["1"]] - d[position["2"]])
18      print(lag)
```

3 Manhattan maboul

3.1 Énoncé

Ah Manhattan... Le quartier préféré de Haruhi ! Nombreux de ses amis sont d'ailleurs de passage à New York et Haruhi aimerait les revoir depuis son dernier long voyage.

Adepte des finales Prologin, Haruhi a pour objectif de rencontrer en M jours **consécutifs** un nombre maximal de ses amis. Pour cela elle a noté l'arrivée de tout le monde. À noter qu'il est possible que plusieurs amis soient présents un même jour.

Entrée

La première ligne contient deux entiers N et M correspondant respectivement au nombre d'amis passant par New York, ainsi qu'au nombre de jours consécutifs que Haruhi peut utiliser pour rencontrer ses amis.

Sur la ligne suivante N entiers, représentant un jour où un des amis de Haruhi est à Manhattan.

Sortie

Le nombre d'amis maximum que Haruhi peut revoir en M jours consécutifs.

Contraintes

- $1 \leq N \leq 100$
- $1 \leq M \leq 36$
- $1 \leq jour \leq 10^6$

Contraintes de performance

- $1 \leq N \leq 10^6$
- $1 \leq M \leq 10^5$

3.2 Solution

En résumant l'énoncé d'un point de vue strictement algorithmique, on nous demande de trouver le nombre maximal de points couverts par un segment de longueur M.

La solution naïve et évidente consiste simplement à tester à partir de chaque point de début, jusqu'où on peut étendre ce segment et maintenir un compteur au fûr et à mesure. On répète cela pour chaque point et on conserve le maximum global. Cependant, la **complexité en temps** de cet algorithme est de l'ordre de $\mathcal{O}(N \times M)$ ce qui ne passe pas les tests de performance où N et M peuvent aller jusqu'à 10^6 et 10^5 respectivement.

Souvent, une bonne manière de se rendre compte de pourquoi notre algorithme est trop lent est de réaliser un exemple à la main. En utilisant notre idée précédente, lorsqu'on étend notre segment le plus loin possible, on va recommencer le segment à zéro pour le prochain début alors qu'on voit qu'on peut au minimum atteindre la même fin de segment. L'idée pour améliorer notre algorithme est donc de ne faire qu'avancer au lieu de revenir inutilement en arrière à chaque étape.

Complexité Pour trier les jours, nous pouvons utiliser la fonction de la bibliothèque standard, résultant en un tri en $\mathcal{O}(N \log N)$. Le reste de l'algorithme ne constitue qu'un parcours des événements ce qui est de l'ordre de $\mathcal{O}(N)$. À première vue, on pourrait se dire qu'une boucle imbriquée nécessite un temps quadratique pour être complétée, mais si l'on réfléchit à la manière dont cette boucle fonctionne dans sa totalité (on parle plus précisément d'**analyse amortie**), on remarque qu'elle ne prend qu'un temps linéaire, car elle va dans le pire des cas parcourir tous les points (on ne revient jamais en arrière, on ne fait qu'avancer !).

Listing 4 – Une solution de l'exercice 3 en Python

```
n, m = map(int, input().split())
jours = [int(j) for j in input().split()]
jours.sort()

nb_amis_max = 0
debut = 0
for fin in range(n):
    while debut < fin and jours[fin] - jours[debut] > m:
        debut += 1
    nb_amis_max = max(nb_amis_max, fin - debut + 1)

print(nb_amis_max)
```

4 La Meilleure Ville

4.1 Énoncé

Joseph Marchand est arrivé à Busland. C'est un pays où la majorité des déplacements entre les différentes villes se font par bus. Heureusement, dans ce pays, les bus sont très organisés : dans chaque ville on attend toujours un bus un temps fixé avant qu'il arrive, le temps d'attente dépend de la ville. Les routes aussi sont très bien organisées : il n'y a jamais aucun bouchon. On met toujours le même temps pour aller d'une ville à l'autre, ce temps dépend de la distance entre les deux villes.

Les routes entre les villes sont toujours à sens unique, en conséquence, avoir un bus direct de la ville A à la ville B ne veut pas dire qu'il y aura un bus direct de B à A. On ne prendra pas non plus nécessairement le même temps pour aller de A à B que de B à A.

Pour éviter d'encombrer le centre ville avec des voyageurs qui ne font que transiter, certaines villes ont mis en place des gares routières. Ces gares routières ont un avantage, les bus en partent en continu, il n'y a donc aucune attente au départ des gares routières.

Depuis la gare routière d'une ville A on peut atteindre les mêmes destinations que depuis la ville A (et les temps de transit sont les mêmes), en revanche, il est impossible de se rendre au centre ville de la ville A depuis la gare routière de la ville A. Pour arriver dans la gare routière d'une ville A, il faut prendre un bus spécial à partir d'une autre ville, ce bus aura son propre temps de trajet.

Joseph veut savoir quelles sont les villes les plus interconnectées à Busland, et pour cela il a trouvé un moyen simple : il associe à chaque ville A un score. Ce score, c'est la moyenne prise sur chaque ville B accessible depuis A, du temps minimal qu'il mettra pour aller du centre ville de A au centre ville de B.

Votre rôle est d'aider Joseph Marchand à calculer les scores de chaques villes de Busland.

Entrée

Sur la première ligne trois entiers :

— n, le nombre de villes ;

— m, le nombre de lignes de bus joignant les villes ;

— g, le nombre de lignes de bus à directions de gares routières.

Sur les n prochaines lignes, le temps d'attente du bus pour la ville numéro i (on commence à $i = 1$).

Puis, m lignes de la forme : a b t indiquant qu'il faut t temps pour aller de la ville a vers la ville b.

Enfin, g lignes de la forme : a b t' correspondant au temps t' requis pour aller de la ville a à la gare routière de la ville b.

Sortie

Les scores des villes, dans l'ordre de leurs numéros, un par ligne.

Si une ville ne permet d'atteindre aucune autre, son score est de -1.

Le score de chaque ville (si différent de -1), arrondi à l'entier inférieur.

Contraintes

- $1 \leq n \leq 150$ et $1 \leq m \leq 200000$
- $0 \leq m \leq 20000$
- $0 \leq g \leq 20000$
- $0 \leq temps \leq 1000$

4.2 Solution

Le sujet consistait à faire des opérations de plus courts chemins d'un point à un autre pour chaque point sur un graphe un peu modifié, puis de calculer la moyenne sur les sommets accessibles.

Pour réussir à résoudre ce problème, on peut essayer de le résoudre dans des cas simples, puis de complexifier ces cas au fur et à mesure.

4.2.1 Une version plus simple du problème : on ne considère que les routes

Pour commencer, regardons la methode pour résoudre dans le cas où notre réseau routier (nous l'appellerons dans la suite « graphe ») n'a que des temps d'attentes nuls, et aucune gare routière.

Dans ce cas, on cherche les plus courts chemins de tous les noeuds du graphe à tous les noeuds du graphe. C'est un problème classique.

4.2.2 Résolution naive

Une première approche est de calculer le plus court chemin d'un point à tous les autres, puis de recommencer. Calculer le plus court chemin d'un point à un autre peut se faire par l'algorithme de Dijkstra, car tous les poids de nos arêtes sont positifs. Si on note n le nombre de sommets et a le nombre d'arêtes du graphe, cet algorithme fait $O(a + n\log(n))$ opérations. On peut alors le répéter pour chaques paires de noeuds, ce qui nous donne un nombre d'opération de $O(n^2(a + n\log(n)))$. Ce qui est trop.

En modifiant légèrement l'algo de Dijkstra, on peut obtenir assez facilement avec le même nombre d'opérations la distance d'un noeud à tous les autres, ce qui nous donne une complexité de $O(n(a + n\log(n)))$. C'est toujours trop si le nombre d'arêtes est élevé.

4.2.3 Résolution efficace par l'algorithme de Floyd Warshall

Un meilleur algorithme pour résoudre ceci est l'algorithme de Floyd Warshall.

Cet algorithme est un exemple de programmation dynamique. Il prend en entrée un graphe G avec n sommets $(0, \ldots, n-1)$ et renvoie en sortie un tableau à double entrée M avec $M[i, j]$ la longueur du chemin minimal entre le sommet i et le sommet j.

Le principe est de calculer à l'étape t (t va de 0 à $n-1$) la longueur du chemin le plus court en empruntant seulement les sommets $0, \ldots, t$. On appelle cette valeur M[i, j, t].

A l'étape $t = 0$, on met $M[i, j, 0]$ à infini (qu'on peut représenter avec une très grande valeur) si i et j ne sont pas connectés, et au poids de i à j sinon. En d'autres termes, $M[i, j, 0]$ est la matrice d'adjacence de G.

A l'étape $t \geq 1$, on utilise le tableau calculé à l'étape d'avant : le chemin le plus court entre i et j utilisant les $t+1$ premiers sommets est soit le chemin le plus court utilisant les t premiers sommets, soit le chemin le plus court de i à $t+1$ (utilisant des t premiers sommets) concaténé avec le chemin le plus court de $t+1$ à j (utilisant des t premiers sommets). On a donc :

$$M[i, j, t+1] = min(M[i, j, t], M[i, t+1, t] + M[t+1, j, t])$$

Cet algorithme effectue un nombre d'opérations compatible avec les attentes du sujet : $O(n^3)$.

4.2.4 On complexifie le problème : les temps d'attente

On rajoute maintenant des temps d'attentes aux villes. On va essayer de se ramener à la version simple du problème. Une première solution est de dire que les temps d'attente sont simplement des temps à prendre en compte dans les chemins (on va rajouter le temps d'attente du noeud i à tous les chemins qui partent de i).

Une autre solution, qui va être celle que l'on va privilégier, c'est de transformer chaque noeud du graphe en deux noeuds « internes » : un noeud « entrée » et noeud « sortie », et de dire qu'un temps d'attente t est un chemin de longueur t entre ces deux noeuds.

Une fois que le nouveau graphe est construit, on se ramène à un problème avec uniquement des routes.

4.2.5 Résolution totale du problème : les gares routières

Ici, on va utiliser la deuxième construction de la partie précédente. On peut remarquer qu'une gare routière n'est autre qu'un noeud « sortie » de notre graphe. Les routes qui vont directement aux gares routières ne sont finalement que des noeuds allant directement aux noeuds « sortie » de notre graphe.

On rajoute donc ces arêtes à notre nouveau graphe, et on a alors encodé le problème en un problème de plus courts chemins dans un graphe, que l'on sait résoudre : on cherche la moyenne sur les sommets « entrée » atteints à partir de chaque sommet « entrée ».

4.2.6 Implémentation en python

Listing 5 – Une solution de l'exercice 4 en python

```python
#!/usr/bin/python
"""
Le poids le plus grand qu'on puisse obtenir est : nombre de sommets * taille maximale * 2 (pour les poids des sommets)
"""
PLUS_INFINITY = 200*2*1000+1

"""
On construit un graphe de taille 2n, avec pour chaque noeud un noeud entrant et un noeud sortant.
La distance entre ces noeuds est le poids du noeud de départ.
On termine avec un nouveau graphe avec 2n noeuds.
Le noeud i se transforme en noeud entrant en position i, et en noeud sortant en position n+i
On le construit sous forme matricielle pour faciliter l'algo de plus court chemin
"""
def construire_graphe(temps_dattente, lignes_bus, lignes_gares_routieres, n):
    graphe = [[PLUS_INFINITY]*(2*n) for _ in range(2*n)]

    for i in range(n): # Du noeud entrant au noeud sortant
        graphe[i][i+n] = temps_dattente[i]

    for (ville_1, ville_2, distance) in lignes_bus: #D'un noeud sortant à un noeud entrant
        graphe[ville_1-1+n][ville_2-1] = distance

    for (ville_1, ville_2, distance) in lignes_gares_routieres: #D'un noeud sortant à une autre noeud sortant
        graphe[ville_1-1+n][ville_2+n-1] = distance

    return graphe

"""
Algorithme de Floyd-Warshall classique.
Référence : https://fr.wikipedia.org/wiki/Algorithme_de_Floyd-Warshall
"""
def distance_tous_a_tous(graphe):
    n = len(graphe)
    for k in range(n):
        for i in range(n):
            for j in range(n):
                graphe[i][j] = min(graphe[i][j], graphe[i][k] + graphe[k][j])
    return graphe

"""
On a les distances entre tous les sommets de notre graphe modifié.
```

```
Il faut maintenant prendre les distances qui nous interessent,
c'est à dire des sommets DE DÉPART (0 <= i <= n-1) aux sommets DE DÉPART
En oubliant pas de ne PAS compter les sommets qu'on n'atteint pas, et le poids propre des sommets
"""
def moyenne_villes(distances_matrix, n):
    moyennes = [-1]*n
    for i in range(n):
        nombre_atteinte = 0
        somme = 0
        for j in range(n):
            if j == i:
                continue
            if distances_matrix[i][j] != PLUS_INFINITY:
                somme += distances_matrix[i][j]
                nombre_atteinte += 1
        if nombre_atteinte != 0:
            moyennes[i] = somme/nombre_atteinte
    return moyennes


"""
Lecture d'entrée
"""
if __name__ == "__main__":
    n, m, g = [int(a) for a in input().split(' ')]
    temps_dattente = [0]*n
    for i in range(n):
        attente_ville = int(input())
        temps_dattente[i] = attente_ville

    lignes_bus = [(0, 0, 0)]*m
    for i in range(m):
        ville_1, ville_2, distance = [int(a) for a in input().split(' ')]
        lignes_bus[i] = (ville_1, ville_2, distance)

    lignes_gares_routieres = [(0, 0, 0)]*g
    for i in range(g):
        ville_1, ville_2, distance = [int(a) for a in input().split(' ')]
        lignes_gares_routieres[i] = (ville_1, ville_2, distance)

    graphe = construire_graphe(temps_dattente, lignes_bus, lignes_gares_routieres, n )
    distances_matrix = distance_tous_a_tous(graphe)
    moyennes = moyenne_villes(distances_matrix, n)

    for m in moyennes:
        if m != -1:
            #on arrondis à l'inférieur à 1 chiffre après la virgule
            arrondi = int(m)
            print(arrondi)
        else:
            print(m)
```

5 Statuettes

5.1 Énoncé

C'est maintenant la fin des vacances pour Haruhi et Joseph, Haruhi est à l'île de pâque et cherche un cadeau pour son groupe d'amis. Elle tombe sur une guirlande composée de statuettes de l'île de pâque de taille variable. Son groupe d'amis a une photo emblématique et pour y faire un clin d'œil elle aimerait avoir un bout de cette guirlande où les statues représentent toutes un des amis du groupe, on les reconnaitra par leurs tailles relatives respectives.

Entrée

L'entrée comprendra :

— Deux entiers, n le nombre d'amis sur la photo, et m la taille totale de la guirlande (le nombre de statuettes sur la guirlande).

— Sur la ligne suivante, n entiers $1 \leq p_i \leq n$ donnent la position du $i^{\text{ème}}$ ami le plus petit. Ces tailles forment une permutation de $[1, n]$.

— Si $p_i = k$ alors l'ami en position k est le $i^{\text{ième}}$ plus petit des n amis sur la photo.

— Sur la ligne suivante m entiers h_j, la taille de la statuette en position j.

Sortie

Donner toutes les positions où Haruhi peut trouver une apparition des tailles relatives de ses amis. Sur la première ligne le nombre d'apparitions a, sur la suivante a entiers, les positions de début des apparitions.

Contraintes

- $1 \leq n \leq 1000000$ et $1 \leq m \leq 200000$
- $1 \leq p_i \leq n$ pour $1 \leq i \leq n$
- $1 \leq h_j \leq 10^9$ pour $1 \leq j \leq m$

5.2 Solution

Il y a plusieurs façons de résoudre ce problème mais nous présentons ici celle-ci. Une autre solution utilisant les FFT et les arbres binaires était possible mais elle ne sera pas détaillée ici.

L'idée est de définir une définition de « match » où on reconnaît un pattern dans un texte (comme de la recherche de mots dans un livre) puis de faire un précalcul pour pouvoir, à partir d'un match partiel, savoir si on peut l'étendre, et après d'appliquer un algorithme de matching classique de type KMP (Knuth–Morris–Pratt).

5.2.1 Gestion de l'entrée

Le format d'entrée des tailles des amis sur la photo est un peu contre intuitif car il n'est pas le même que pour le collier de statuettes, commençons par changer ça. Il est facile de se ramener à une liste de tailles assez facilement : le plus petit est de taille 1, le deuxième plus petit est de taille 2 et ainsi de suite…Voilà un pseudocode qui règle ce problème :

```
Algorithm Gèrer l'entrée

    Récupèrer les entiers n et m ;
    Enregistrer l'ordre des amis dans order ;
    Enregistrer les tailles des statuettes dans height.
    Initialiser le tableau pat de taille n ;

    for i := 1 to n do
        pat[order[i]] = i ;
```

5.2.2 Précalcul

On peut formaliser la définition donnée dans l'énoncé en terme de plus petit, deuxième plus petit, ..., ainsi, x match y (un bout de la photo match un bout de la guirelande) noté $x \approx y$ si et seulement si :

$$\forall 1 \leqslant i, j \leqslant |x|, x_i \leqslant x_j \iff y_i \leqslant y_j$$

On a défini un critère pour trouver le pattern de la photo des amis dans le collier de statuettes, maintenant avant de pouvoir appliquer un algorithme de recherche classique, on a besoin, étant donné que $pat[1...i]$ match, d'avoir une méthode en $O(1)$ pour savoir si $pat[1...i+1]$ match.

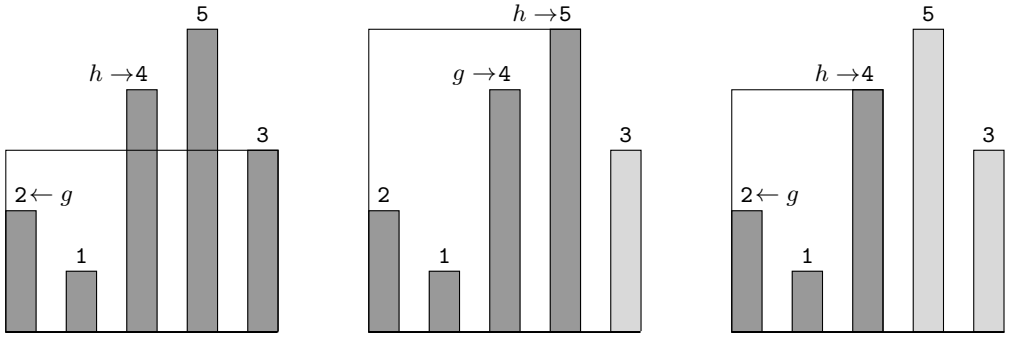
On définit les fonctions suivantes :

— $g[i] = j$ si $pat[j] = \max\{pat[k] | k \in [1, i-1]$ et $pat[k] < pat[i]\}$, s'il n'y en a pas, $g[i] = i$.
En français, $g[i]$ est la taille de la plus grande statuette, parmis les $i-1$ premières statuettes qui sont plus petites que la $i^{\text{ème}}$ statuette.

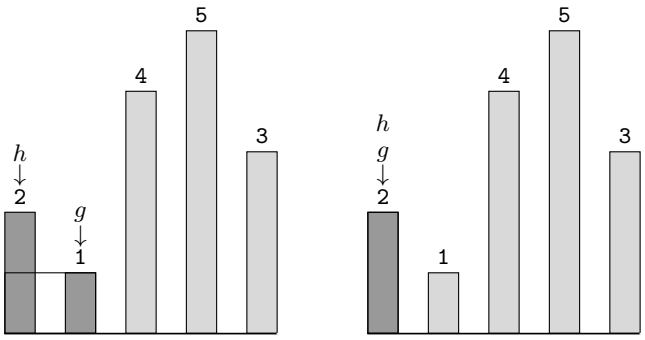
— $h[i] = j$ si $pat[j] = \max\{pat[k] | k \in [1, i-1]$ et $pat[k] > pat[i]\}$, s'il n'y en a pas, $h[i] = i$.
En français, $h[i]$ est la taille de la plus petite statuette, parmis les $i-1$ premières statuettes qui sont plus grandes que la $i^{\text{ème}}$ statuette.

Le calcul se comprends mieux sur un example :

i	1	2	3	4	5
$w[i]$	2	1	4	5	3
$g[i]$	1	2	1	3	1
$h[i]$	1	1	3	4	3



$g[5] = 1 \;\; h[5] = 3$ $g[4] = 3 \;\; h[4] = 4$ $g[3] = 1 \;\; h[3] = 3$



$g[2] = 2 \;\; h[2] = 1$ $g[1] = 1 \;\; h[1] = 1$

On peut les calculer facilement à l'aide de pointeurs, voilà par exemple un pseudo code :

`Algorithm` Calculer de g et h

Initialiser la table *pat* suivant la gestion de l'entrée décrit au dessus ;
Faire la liste *pattern* des elements de *pat*

Initialiser les tables g et h avec des zeros ;

Initialiser la table ref avec des pointeurs tel que $ref[k]$ pointe vers i_k tel que $pattern[i_k] = k$;

for k :=n `down to` 1 **do**
  $i := ref[k]$;
  if i est le dernier élément de *pattern* **then**
    $h[k] = k$;
  else
    $h[k] = next\_element(i, pattern)$;
  if i est le premier élément de *pattern* **then**
    $g[k] = k$;
  else
    $g[k] = previous\_element(i, pattern)$;
  Supprime la position i de *pattern* ;

Ces calculs nous permettent d'étendre un match en $O(1)$: si $P[1..i]$ match on a seulement à comparer le nouvel élément dans le texte à celui correspondant aux positions $g[i+1]$ et $h[i+1]$ pour voir s'il s'insère correctement !

5.2.3 Algorithme complet

`Algorithm` Resolution problème 5

Gérer l'input ;
Précalculer les fonctions g et h à partir de pat ;

(* Première partie de l'algorithme KMP *)
Initialiser la table $bord$ avec des zeros ;
$bord[0] = j = -1$;
for $i := 1$ `to` n **do**
  while $j \geq 0$ `and` $((pat[k] < pat[k-j+g[j]])$ `or` $(pat[k] > pat[k-j+h[j]]))$ **do**
    $j = bord[j]$;
  $j = j + 1$;
  $bord[i] = j$

(* Deuxième partie de l'algorithme KMP *)
j = 0 ;
for i := 1 `to` n **do**
  while $0 \leq j$ `and` $(height[i-j+g[j]] > height[i])$ `or` $(height[i-j+h[j]] < height[i])$ **do**
    $j = bord[j]$;
  $j = j + 1$;
  if $j == n$ **then**
    i est une solution ;
    $j = border[n]$;

5.2.4 Implémentation en C++

Listing 6 – Une solution de l'exercice 5 en C++

```
1  //g++
2  #include <vector>
3  #include <list>
4  #include <tuple>
5  #include <iostream>
6  #include <algorithm>
7  using namespace std;
8
9  int main()
10 {
11   /* INPUT */
12   int n, m ;
13   cin >> n >> m ;
14   int pattern [n];
15   int height [m];
16   for (int i = 0; i < n; ++i)
17   {
18     cin >> pattern[i];
19   }
20   for (int i = 0; i < m; ++i)
21   {
22     cin >> height[i];
23   }
```

```
24
25  /* REVERSE VIEW OF THE PATTERN */
26
27  int order [n];
28  for (int i = 0; i<n; i++)
29  {
30    order[pattern[i] - 1] = i+1;
31  }
32
33
34  /* PRECOMPUTATION OF g AND h */
35  int g [n];
36  int h [n];
37
38  list<int> pattern_copy = {} ;
39  vector<list<int>::iterator> references(n);
40  for(int i = 0; i < n; i++)
41  {
42    pattern_copy.push_back(pattern[i]);
43    references[pattern[i]-1] = pattern_copy.end();
44    --references[pattern[i]-1];
45  }
46
47  for (int k = n; k > 0; --k)
48  {
49    list<int>::iterator i = references[k-1];
50    ++i;
51    if (i==pattern_copy.end()) h[k-1] = k;
52    else h[k-1] = *i;
53    --i;
54    if (i==pattern_copy.begin()) g[k-1] = k;
55    else
56    {
57      --i;
58      g[k-1] = *i;
59      ++i;
60    }
61    pattern_copy.erase(i);
62  }
63
64  /* KMP ALGORITHM */
65
66  vector<int> border(n+1);
67  int j = -1;
68  border[0] = -1;
69
70  for (int k = 1; k <= n; ++k)
71  {
72    int i = order[k-1];
73    while ( (j>=0) && ((i < order[k-1-j+g[j]-1]) || (i > order[k-1-j+h[j]-1]))) //k
        ↪ -> k-1
74    {
75      j = border[j];
```

```
76      }
77      j += 1;
78      border[k] = j;
79    }
80
81    j = 0;
82    int nb_soluce = 0;
83    list<int> soluce = {};
84    for (int i = 0; i < m; ++i)
85    {
86      while ((j >=0) &&
87          ((height[ i - j+ g[j] -1 ] > height[i]) ||
88          (height[ i - j+ h[j] -1 ] < height[i])))
89      {
90        j = border[j];
91      }
92
93      j += 1;
94      if (j == n )
95          {
96            nb_soluce += 1;
97            soluce.push_back(i+1 - (n - 1) );
98            j = border[n];
99          }
100   }
101
102   cout << nb_soluce << endl;
103   for(int pos:soluce)
104   {
105     cout << pos << " ";
106   }
107   cout << endl;
108 }
```

---

*
**

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.