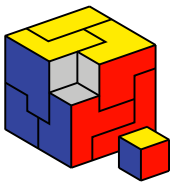
Prologin 2020



Concours national d'informatique

Épreuve écrite d'algorithmique

Alter Égopieurs

1 Préambule

Bienvenue à **Prologin**. Ce sujet est l'épreuve écrite d'algorithmique et constitue la première des trois parties de votre épreuve régionale. Sa durée est de 3 heures. Par la suite, vous passerez un entretien (20 minutes) et une épreuve de programmation sur machine (4 heures).

Conseils

— Lisez bien tout le sujet avant de commencer.
— **Soignez la présentation** de votre copie.
— N'hésitez pas à poser des questions.
— Si vous avez fini en avance, relisez bien, ou préparez votre présentation pour l'entretien.
— N'oubliez pas de passer une bonne journée.

Remarques

— Le barème est donné à titre indicatif uniquement.
— Indiquez lisiblement vos nom et prénom, la ville où vous passez l'épreuve et la date en haut de votre copie.
— Tous les langages sont autorisés, veuillez néanmoins préciser celui que vous utilisez.
— Ce sont des humains qui lisent vos copies : laissez une marge, aérez votre code, ajoutez des commentaires (**seulement** lorsqu'ils sont nécessaires) et évitez au maximum les fautes d'orthographe, sinon ça va barder.
— Le barème récompense les algorithmes les plus efficaces : écrivez des fonctions qui trouvent la solution le plus rapidement possible.
— Si vous trouvez le sujet trop simple, relisez-le, réfléchissez bien, puis dites-le-nous, nous pouvons ajouter des questions plus difficiles.

Comme chaque année, les candidats de Prologin vont plancher pendant plusieurs heures sur leur sujet écrit, et comme chaque année la triche sera de mise, c'en est presque une tradition.

En réaction à cette triche quasi systématique de la part des candidats, les organisateurs ont mis en place plusieurs mesures :

I - Offrir des viennoiseries aux candidats avant et pendant l'épreuve. Le sucre étant un excitant naturel [1], ils ont plus de mal à adopter l'attitude furtive nécessaire pour communiquer avec leurs voisins.

II - Intimider les candidats. Vous pouvez remarquer que pendant l'épreuve certains d'entre eux vont être appelés, puis éloignés des autres pour une dizaine de minutes.

III - Même si les méthodes préventives mentionnées ci-dessus se sont montrées efficaces, elles ne suffisent pas à éliminer tous les cas de triche, il est nécessaire de vérifier pour toutes les paires de copies si l'une ne semble pas trop fortement inspirée de l'autre.

Ce document s'intéresse au troisième point. À cause des nouvelles technologies entre autre, toute paire de candidat est capable de félonie, et nous ne pouvons pas nous contenter de considérer que seuls des voisins sont susceptibles de tricher [2]. Pour une épreuve à Paris qui accueillerait 50 candidats, il y aurait 1225 paires à vérifier. Il devient capital de simplifier cette tâche fastidieuse.

Il va sans dire que ce sujet doit être utilisé discrètement, et **en aucun cas** ne devra tomber entre les mains d'un candidat du concours.

2 Introduction

Pour réduire le nombre de paires de copie que les organisateurs doivent analyser pour détecter la triche, on va considérer que les candidats ne peuvent copier que sur la copie d'un ami (ou ne recevoir une solution que de l'un d'eux) [3]. Nous allons donc nous aider de la liste d'amis de chaque candidat [4] pour sélectionner les paires de copies susceptibles de félonie.

Question 1 (1 point)

> Décrivez comment représenter un candidat ainsi que la liste d'amis de l'ensemble des candidats. Vous pouvez utiliser un langage de programmation ou bien du pseudo code en utilisant des opérations claires.

Question 2 (1 point)

> Écrivez une fonction `test_amis(amities, candidat_1, candidat_2)` qui vérifie si deux candidats sont amis étant donné la liste d'amis de chaque candidat présent.
>
> Donnez une estimation grossière du nombre d'instructions exécutées par votre fonction en fonction du nombre de candidats et liens d'amitiés.

Question 3 (1 point)

> Écrivez une fonction `liste_paires(amities)` qui prend en entrée la liste d'amis de chaque candidat présent, et retourne la liste des paires d'amis parmi eux.
>
> Donnez une estimation grossière du nombre d'instructions exécutées par votre fonction en fonction du nombre de candidats et liens d'amitiés.

L'ambiance du concours est tellement conviviale qu'il est bien connu que les cercles d'amis s'étendent à tours de bras. Il faut prendre en compte le fait qu'un candidat est susceptible de communiquer avec les amis de ses amis, les amis de ses amis de ses amis, les amis de ses amis de ses amis de ses amis, les amis de ses amis de ses amis de ses amis de ses amis, les amis de ses amis de ses am [5]

1. À noter que les buveurs de cafés font les pires mécréants, c'est une faille que l'on peut également exploiter.
2. En plus, à cause ~~de l'incompétence~~ du surmenage des organisateurs, ils paument toujours le plan de salle.
3. Tant que les candidats ne sont pas au courant de cette combine, ils ne risquent pas d'adapter leur comportement en fonction.
4. Qui sera bientôt téléchargeable depuis le site du ministère des finances.
5. Un organisateur a cru bon d'interrompre ici les effets de l'incomp... heu... du surmenage d'un autre.

Par exemple, dans la figure 1, comme Paul et Ana sont amis, et que Paul et Luc aussi, on dit qu'il existe une chaîne d'amitiés entre Ana et Luc.

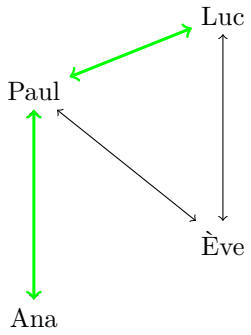


FIGURE 1 – Un exemple de chaîne d'amitiés

Question 4 (4 points)

> Écrivez une fonction `composante(amities, candidat)` qui retourne la liste des candidats telle qu'il existe une chaîne d'amis entre chaque candidats et le candidat donné en entrée.
>
> Donnez une estimation grossière du nombre d'instructions exécutées par votre fonction en fonction du nombre de candidats et liens d'amitiés.

Question 5 (2 points)

> Écrivez une fonction `composantes(amities)` qui retourne la liste de tous les groupes de candidats liés par des chaînes d'amitié.
>
> Donnez une estimation grossière du nombre d'instructions exécutées par votre fonction en fonction du nombre de candidats et liens d'amitiés.

Question 6 (1 point)

On dit qu'une liste d'amitié est *connexe* si pour toute paire de candidats, il existe une chaîne d'amitiés entre ces deux candidats. Par exemple, la chaîne d'amitiés sur la figure 1 est connexe, mais pas celle représentée sur la figure 2.

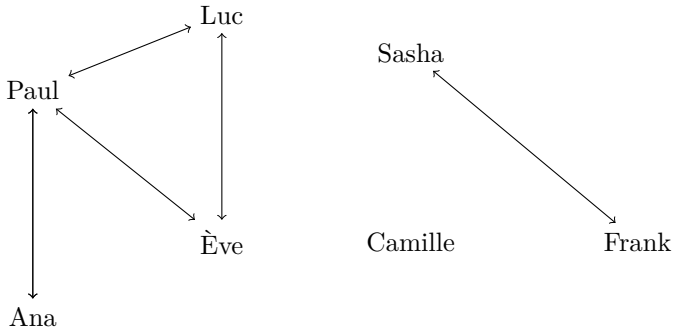


FIGURE 2 – Un exemple de chaîne d'amitiés non connexe

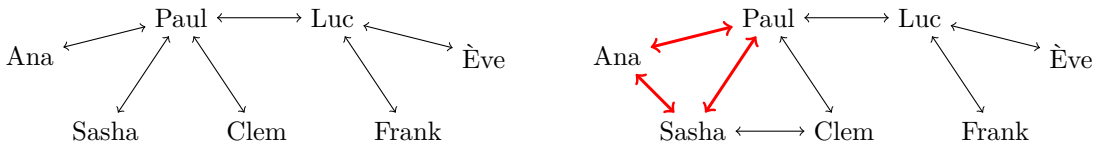
> Comment utiliser les fonctions présentées dans la question 4 ou la question 5 pour vérifier si une liste d'amitié est connexe ?
>
> Donnez une estimation grossière du nombre d'instructions exécutées par votre fonction en fonction du nombre de candidats et liens d'amitiés.

3 Arbres Féloniques

Cette année, les organisateurs se sont enfin documentés sur le sujet [6]. Ils ont pris le temps d'étudier l'état de l'art des recherches sur ce type de situation : *The Untouchables* de Brian de Palma, *The Godfather* de Francis Ford Coppola, ... Une chose certaine est ressortie : s'ils veulent neutraliser efficacement ces réseaux de candidats félons, il va d'abord falloir identifier leurs meneurs !

Pour ce faire, les organisateurs ont envisagé de modéliser les relations d'amitiés en utilisant des *arbres féloniques*. Un arbre félonique est une liste d'amitiés (tel que vue dans la section précédente), qui vérifie les conditions suivantes :

— il n'y a pas de cycles parmi les liens d'amitié entre les candidats représentés dans l'arbre félonique ;
— la liste d'amitiés est connexe (cf. qu. 6).



(a) Un exemple d'arbre félonique

(b) Cette liste d'amitiés n'est pas un arbre félonique car elle comporte un cycle

FIGURE 3 – Un exemple d'arbre félonique et de graphe d'amitiés qui n'est pas un arbre félonique

Question 7 (2 points)

Grâce à cette structure, une fois que le meneur aura été identifié, il va être plus simple pour les organisateur d'identifier la hiérarchie des tricheurs et en particulier de savoir qui a triché sur qui :

— les candidats adjacents au meneur dans l'arbre félonique copient sur le meneur, ce sont les candidats de rang 1 ;
— les candidats adjacents à un candidat de rang 1 dans l'arbre félonique (autres que le meneur) copient sur lui ;
— les candidats adjacents à un candidat de rang 2 dans l'arbre félonique (mais qui ne sont pas de rang $\leq$ 1) copient sur lui ;
— etc...

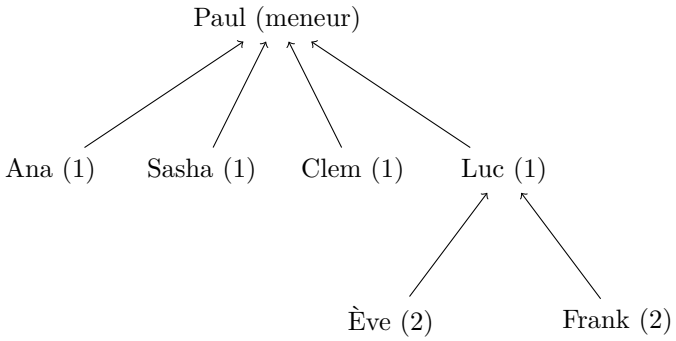


FIGURE 4 – L'arbre de la figure 3a, mais en mettant en valeur la hiérarchie où Paul est le meneur

> Écrivez une fonction, `copieurs(arbre_felonique, meneur)` qui retourne une structure `copie_sur` telle que pour tout candidat, `copie_sur[candidat]` retourne le candidat directement au dessus de lui dans la hiérarchie ou `"Rien"` si `candidat` est le meneur.

Question 8 (2 points)

Écrivez la fonction réciproque de celle introduite dans la question précédente : si `copie_sur` est la structure obtenue comme résultat de la fonction `copieurs`, alors `arbre_felonique(copieurs)` retournera une paire `(meneur, amities)` correspondant au meneur identifié et à l'arbre félonique correspondant, sous forme de liste d'amitié.

6. Nous remercions aussi Mme Bertrand, professeure d'art plastique en 5ème b qui a partagé avec nous son expertise sur le sujet.

Question 9 (1 point)

> Montrez que s'il y a n candidats et $n-1$ liens d'amitiés, alors il y a forcément un candidat qui a au plus un ami.

Question 10 (3 points)

> Montrer que s'il y a n candidats, $n-1$ liens d'amitiés et que la liste des liens d'amitiés est connexe, alors la liste des liens d'amitiés est un arbre félonique.
>
> **NOTE** Cette question est un peu plus dure que les précédentes, voici un brin d'aide pour que nos chers candidats n'aient pas à regarder par dessus l'épaule du voisin [7] : cette question peut être résolue par récurrence.

Question 11 (4 points)

> En déduire que les arbres félonniques sont les listes d'amitiés connexes, ayant $n-1$ liens d'amitiés pour n candidats.

Question 12 (2 points)

> Écrivez une fonction `arbre_felonique(amities)` qui retourne `"Vrai"` si la liste d'amitiés donnée en entrée est un arbre félonique, et `"Faux"` sinon.

Question 13 (2 points)

> Sauriez-vous écrire un algorithme qui détecte qu'une liste d'amitiés est un arbre félonique sans utiliser le théorème ?
>
> Donnez une estimation grossière du nombre d'instructions exécutées par votre fonction en fonction du nombre de candidats et liens d'amitiés.

Question 14 (4 points)

> Écrivez une fonction `foret_couvrante(amities)` qui prend en entrée une liste d'amitiés et retourne un arbre félonique pour chaque composante de la liste d'amitiés (au sens de la question 5, qui ne contiens que des liens d'amitiés présent dans la composante `amities` donnée en entrée et dont tous les candidats de cette composante font partie.
>
> Donnez une estimation grossière du nombre d'instructions exécutées par votre fonction en fonction du nombre de candidats et liens d'amitiés.

4 Les noyaux durs

En réalité, les candidats de Prologin sont bien trop désorganisés [8] pour réussir à former une forte structure de hiérarchie, c'était une hypothèse irréaliste.

Les vrais irréductibles groupes de tricheurs sont des noyaux de candidats très soudés [9]. Pour simplifier et parce qu'il faut bien commencer quelque part nous allons rechercher des noyaux de ce type qui comportent seulement 3 candidats.

On va considérer que les noyaux durs dont il faut se méfier sont les groupes de 3 candidats tels que chacun de ces candidats est amis avec les deux autres. Sur la figure 3b, il y a deux groupes de ce type : {Ana, Paul, Sasha} et {Clem, Paul, Sasha}.

7. En effet, nous n'avons pas de quoi traiter les torticolis sur place.
8. Bien au contraire des organisateurs !
9. Ce sont les pires plaies selon Mme Bertrand, elle nous a recommandé de les éradiquer au plus vite.

Question 15 (2 points)

Écrivez une fonction `triplets_naif(amities)` qui retourne la liste des triangles qui peuvent être trouvés dans la liste des candidats.

Donnez une estimation grossière du nombre d'instructions exécutées par votre fonction en fonction du nombre de candidats et liens d'amitiés.

Question 16 (3 points)

Pour la suite de l'énoncé on fixe un candidat `meneur`, la liste d'amitiés `amities` et on pose `arbre_felonique = foret_couvrante(amities)`, ainsi que `parent = copieurs(arbre_felonique, meneur)`.

Montrez que s'il existe un triangle de trois amis dans la liste d'amitiés dont une paire fait partie de l'arbre **alors** il existe une paire d'amis `(x, y)` qui ne fait pas partie de l'arbre félonique, et telle que `(parent(x), y)` fait aussi partie des liens d'amitiés.

Question 17 (3 points)

Montrez que s'il existe une paire d'amis `(x, y)` qui ne fait pas partie de l'arbre félonique, et telle que `(parent(x), y)` fait aussi partie des liens d'amitiés, **alors** il existe un triangle de trois amis dans la liste d'amitiés dont une paire fait partie de l'arbre.

Question 18 (2 points)

Implémentez l'algorithme suivant, donné par un pseudo-code très succinct :

Algorithme 1 : Algorithme mystère

```
Résultat : Vrai s'il existe un triangle dans la liste d'amitiés
tant que amities n'est pas vide faire
    foret ← foret_couvrante(amities) ;
    Utiliser les questions 16 et 17 pour chercher s'il existe un triangle dans le graphe dont une
     arrête appartient à la forêt couvrante ;
    si on a trouvé un triangle alors
        retourner Vrai ;
    sinon
        Enlever de amities toutes les arêtes présentes dans foret
    fin
fin
retourner Faux ;
```

Question 19 (3 points)

Expliquez pourquoi l'algorithme 1 termine et retournera vrai si et seulement s'il y a un triangle.

Question 20 (2 points)

Comment adapter cet algorithme pour compter le nombre de triangles dans le graphe ?

Question 21 (2 points)

On va maintenant supposer que pour n candidats, il y a au plus $3n - 6$ liens d'amitié [10].
Prouver qu'on enlève au moins 1/3 des arêtes à chaque itération.

10. Ce critère n'est pas tiré du chapeau, c'est le nombre d'arêtes dans un graphe planaire.

Question 22 (4 points)

> Donnez une estimation grossière du nombre d'instructions exécutées par l'algorithme 1 en fonction du nombre de candidats dans ce cas.

5 Soyez amis des licornes [11]

Question bonus 23 (1 point)

Rappelez-vous du meilleur conseil que ne vous ai jamais donné Mme Bertrand : ń Un vrai dessinateur ne lève jamais le crayon de sa feuille, il perd en précision et en continuité artistique ż.

> En appliquant les conseils de Mme Bertrand [12], dessinez une licorne qui fait un câlin à un organisateur de Prologin.

Question bonus 24 (-100 point)

> Copiez la réponse à cette question sur votre voisin de diagonale avant-droite [13].

Question bonus 25 (1 point)

> Faite un pari sur les quantités suivantes suivantes :
> — nombre de fotes d'orthographe de l'énoncé
> — nombre de fotes grammaticales de l'énoncé
> — nombre de questions où il faut prouver un résultat faux
> — nombre de questions faites de pur bonheur

Question bonus 26 (0 point)

> Dessinez un problème d'échecs intéressant [14].

Question bonus 27 (0 point)

> Trouvez une anagramme cocasse à votre prénom.

11. Comme ça on aura des licornes dans le graphe d'amitiés.
12. On vous le rappelle parce-qu'évidement, vous n'écoutiez pas.
13. Copiez la réponse sur votre voisin de gauche si vous n'avez pas la réponse. Si vous n'avez pas de voisin de gauche vous n'aurez pas les points de cette question.
14. Il peut être cocasse ou tout simplement challenging.