TP05 Conception d'algorithmes simples

Pensez à utiliser GithubDesktop pour faire un « Pull » et récupérer le dossier du TP qui aura été déposé sur Gitlab par le Général Kléber.

Pensez aussi à écrire votre code *en premier lieu* sur feuille pour vous entraîner aux futures épreuves écrites d'informatique.

Partie I

Mise en jambe : beaucoup de bruit pour rien

I.1 Énoncé

C'est le jour de la rentrée des classes et les instituteurs ont décidé d'organiser un jeu dans la cour avec tous les élèves de l'école. Ils sont tous placés en rang sur une ligne de départ d'un côté de la cour. À l'autre bout de la cour, les instituteurs ont disposé de nombreux objets. Au signal du départ tous les élèves se mettent à courir pour récolter le plus d'objets possible qu'ils rapportent ensuite à leur position de départ. Un instituteur parcourt le rang pour inscrire le score de chaque élève. Tous les objets sont remis à leur place et le signal de départ d'une seconde manche est donné. L'instituteur passe à nouveau dans le rang pour compter le nouveau score, et inscrire l'addition. Il ne sait pas alors qu'il a parcouru le rang dans l'autre sens et que ses résultats seront faux !

Par exemple, lors de la première manche les élèves ont obtenu dans l'ordre `3 5 2 8 3`. Lors de la seconde manche ils ont obtenu `3 1 3 2 5`. Le score total devrait donc être `6 6 5 10 8` mais l'instituteur a additionné `5 2 3 1 3` à `3 5 2 8 3` et a donc obtenu `8 7 5 9 6`.

Les objets sont rangés pendant que l'instituteur établit le classement. Les vainqueurs sont annoncés mais d'autres enfants protestent. L'instituteur vérifie ses comptes et s'aperçoit de son erreur. En tant qu'instituteur chargé de l'enseignement de l'informatique, vous suggérez de corriger les scores à l'aide d'un programme, ce qui sera plus rapide que de le faire à la main.

Écrivez un programme `corrige_score(premiere_manche,final_errone)` qui, étant données la liste des scores de la première manche et la liste erronée du score total, donne la liste correcte du score total.

I.2 Commentaires sur un exemple

Il y a 6 élèves qui participent au jeu. Lors de la première manche, leurs scores respectifs sont `[0,1,2,3,4,5]`, dans l'ordre du rang de gauche à droite. Après la deuxième manche, l'instituteur additionne les nouveaux scores (mais à l'envers), et obtient les scores totaux `[1,5,3,6,6,5]`. Ses additions sont les suivantes :

```
  0   1   2   3   4   5   (scores de la première manche, fournis)
+ 1   4   1   3   2   0   (scores inversés de la 2ème manche, non fournis)
--- --- --- --- --- ---
  1   5   3   6   6   5   (résultats obtenus par l'instituteur, fournis)
```

alors qu'il aurait du calculer :

```
  0   1   2   3   4   5
+ 0   2   3   1   4   1   (scores dans le bon sens)
--- --- --- --- --- ---
  0   3   5   4   8   6   (résultats corrects, à renvoyer par votre programme)
```

Votre programme doit renvoyer le résultat correct des additions, soit `[0,3,5,4,8,6]` sur cet exemple.

Stop Gitlab

Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

Partie II

Prison de Sikinia

II.1 Version historique

Dans la prison centrale de Sikinia, il y a 30 cellules numérotées $1, 2, 3, \ldots, 30$, toutes occupées. Les portes des cellules peuvent être dans deux états : ouvertes ou fermées. On peut passer d'un état à l'autre en faisant faire un demi-tour au bouton de la porte. Au moment où commence l'histoire, toutes les portes sont fermées.

Pour fêter le vingtième anniversaire de la république de Sikinia, le président décide d'une amnistie. Il donne au directeur de la prison les ordres suivants : « Tournez successivement d'un demi-tour les boutons :

— de toutes les portes,
— puis d'une porte sur deux, à partir de la deuxième,
— puis d'une porte sur trois, à partir de la troisième,
— ...

Continuez ainsi jusqu'à la dernière cellule. Libérez alors les prisonniers dont la porte de cellule est ouverte. »

Pour des raisons de sécurité, le directeur de la prison aimerait connaître à l'avance quels seront les prisonniers libérés. Pouvez-vous l'aider ? Vous écrirez un programme plus général `liberation_prisonniers(n)` qui prend en argument le nombre de cellules de la prison (numérotées de `1` à `n`) et qui doit renvoyer la liste des numéros des cellules ouvertes. (Problème tiré de *Elements of mathematics*, 1975, St-Louis (Missouri))

Stop Gitlab

Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

II.2 Version moins prévisible

Les plus mathématiciens d'entre vous auront vu qu'en fait les cellules ainsi ouvertes suivent toujours un même motif[1]. On va donc corser un tout petit peu les choses : à présent, la procédure de rotation des boutons est toujours la même, mais on ne part pas de l'état « toutes les cellules sont fermées », mais d'un état intermédiaire où certaines sont ouvertes et certaines sont fermées. Pour stocker cet état, on vous fournit un dictionnaire[2] `cellules` qui a pour clefs les numéros de cellule[3] et pour valeur `True` si la cellule commence ouverte et `False` si la cellule commence fermée.

En outre, on ne va pas forcément démarrer à la toute première porte, mais on donne un second argument `i_depart` qui dit qu'on démarre à cette porte `i_depart` pour tourner les boutons à toutes les cellules multiples de `i_depart` en allant jusqu'à la dernière cellule accessible[4]. On continue ainsi de suite jusqu'à atteindre le(s) multiple(s) du numéro de la dernière cellule.

En résumé, il faut ici compléter la fonction `sikinia_difficile(cellules, i_depart)` qui prend en argument le dictionnaire `cellules` donnant l'état actuel des cellules et le numéro `i_depart` de la cellule à partir de laquelle on réapplique le protocole de la section II.1 précédente.

Stop Gitlab

Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

1. Il peut même être intéressant de réfléchir au pourquoi mathématique de la chose, mais je vous laisse en discuter avec vos profs de maths respectifs ou essayer de le démontrer par vous-même.
2. Une liste aurait pu marcher aussi, en prenant pour position le numéro de la cellule et en rajoutant une cellule numérotée 0, mais c'est pour s'entraîner dans la manipulation des dictionnaires.
3. Ces numéros sont bien des entiers qui sont des clefs valides de dictionnaires, au même titre que des chaînes de caractères.
4. Pour trouver le nombre de cellules total jusqu'auquel on va devoir aller, vous pouvez utiliser la fonction `len` qui donne le nombre de clefs présentes dans le dictionnaire. Les clefs en question étant les entiers de 1 à n, il y en a bien n au total.

II.3 Version récursive

La structure d'appel adoptée à la section II.2 précédente facilite grandement la mise en place d'une implémentation récursive `sikinia_recursif(cellules, i_depart, final)` qui doit de la même manière renvoyer la liste des cellules desquelles on libèrera les prisonniers après application de la procédure complète. On ajoute néanmoins un argument (`final`) qui est un booléen mis à `True` s'il s'agit de l'appel original à la fonction (qui doit donc renvoyer la fameuse liste des prisonniers à libérer) et `False` s'il s'agit des appels internes de la fonction (rien ne sert de calculer qui va être libéré tant que toutes les manipulations n'ont pas été faites). À noter que votre fonction devra *modifier* le dictionnaire `cellules` qu'on lui donne en argument car sinon les copies successives lors des appels récursifs vont plomber l'efficacité de la procédure.

Pour passer les tests, votre fonction devra forcément :

— s'appeler elle-même [5] ;

— comporter un maximum de cinq instructions conditionnelles, donc cinq `if` (ou `elif`) accompagnés ou non de leur `else`, à votre convenance [6] ;

— comporter une unique boucle [7] au lieu des deux dont vous avez eu besoin dans les deux sections précédentes.

Stop Gitlab

Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

Partie III

Feux de forêt

La foudre se déchaîne sur les Vosges. Les éclairs déchirent le ciel, offrant un spectacle à la fois sublime et effrayant. Néanmoins, les habitants du coin savent que la menace principale provient des feux de forêt : chaque éclair s'abattant sur la forêt engendre un départ de feu. Vous décidez d'aider les pompiers à s'organiser dans leur combat contre les flammes.

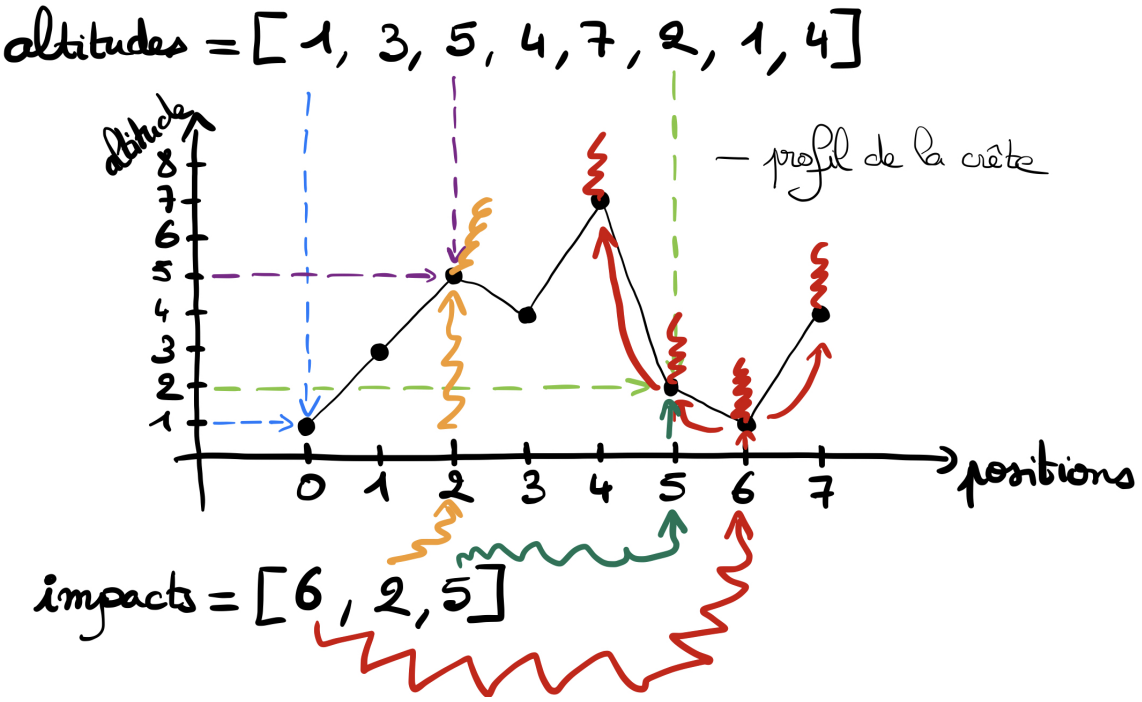
La forêt, qui s'étend tout le long de la crête (donc à 1 dimension) de la montagne [8], peut être représentée par une suite de cases, chacune à une altitude donnée. Lorsque la foudre s'abat sur une case, celle-ci s'enflamme, puis se propage ensuite aux cases voisines, toujours en montant. Autrement dit, le feu se propage d'une case aux cases voisines de même altitude ou d'altitude supérieure.

Écrivez un programme `feux_de_foret(altitudes, impacts)` qui détermine le nombre de cases de forêt qui risquent de brûler connaissant la liste `altitudes` des altitudes des cases successives le long de la crête et la liste `impacts` des positions (variant de `0` à `len(altitudes)-1`) où frappent les éclairs (ces positions étant données dans un ordre a priori quelconque).

Votre programme doit renvoyer un unique entier : le nombre de cases de forêt menacées par les flammes.

Sur l'exemple ci-dessous, le programme doit renvoyer `5` puisqu'il y a 5 cases brûlées au total, l'éclair qui tombe sur la position 5 ne pouvant plus faire de dégats puisque la case était déjà brûlée par le tout premier éclair.

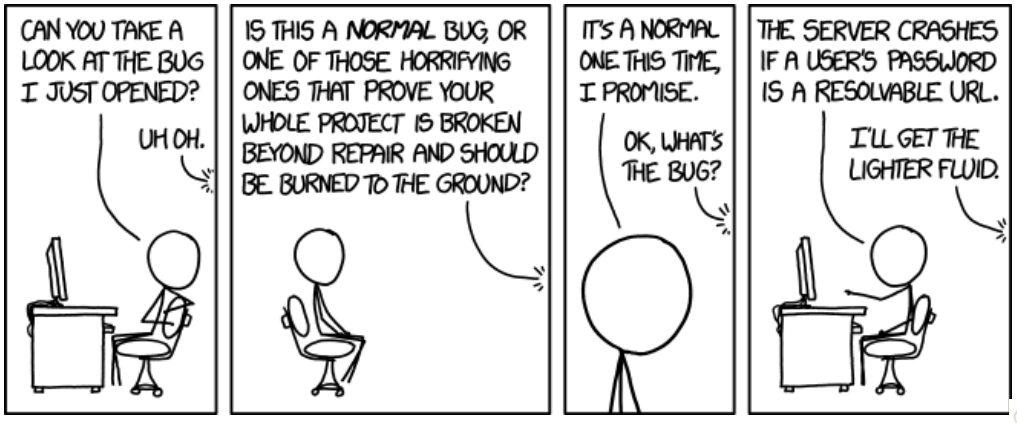
5. C'est le principe même d'une fonction récursive : elle fait appel à elle-même en changeant juste les paramètres d'appel.
6. Réfléchissez bien à la condition d'arrêt qu'il faut implémenter. Il est possible de s'en sortir avec seulement trois `if` pour ceux qui sont joueurs
7. `for` ou `while`, à votre convenance, même si la boucle `for` sera plus sûre.
8. Enfin des collines, quoi, c'est les Vosges...



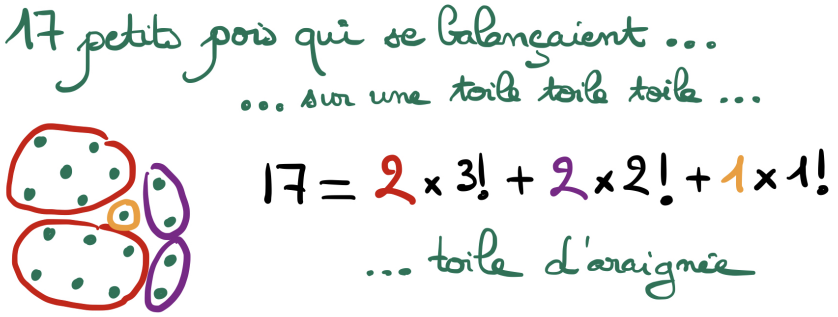
Pour passer tous les tests, votre programme doit être aussi efficace que possible (limitations en temps à 1 s sachant que le nombre de cases et le nombres d'éclairs peuvent varier dans $[\![1\,;\,40\,000]\!]$).

Stop Gitlab

Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.




```
There's also a unicode-handling bug in the URL request library,
and we're storing the passwords unsalted ...
so if we salt them with emoji, we can close three issues at once!
```



Partie IV

Des histoires de petits pois

Un marchand de légumes très maniaque souhaite ranger ses petits pois en les regroupant en boîtes de telle sorte que chaque boîte contienne un nombre factoriel de petits pois. On rappelle qu'un nombre est factoriel s'il est de la forme 1, 1×2, $1 \times 2 \times 3$, $1 \times 2 \times 3 \times 4$... et qu'on les note sous la forme suivante : $n! = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$

Il souhaite également utiliser le plus petit nombre de boîtes possible. Ainsi, s'il a 17 petits pois, il utilisera :

— 2 boîtes de $3! = 6$ petits pois, soit 12 petits pois rangés ;

— 2 boîtes de $2! = 2$ petits pois, soit 4 petits pois rangés ;

— 1 boîte de $1! = 1$ petit pois, soit 1 petit pois rangé.

ce qui donne bien $2 \times 3! + 2 \times 2! + 1 \times 1! = 12 + 4 + 1 = 17$.

D'une manière générale, s'il a `nb_petits_pois`, il doit trouver une suite $a_1, a_2, \ldots, a_p$ d'entiers positifs ou nuls avec $a_p > 0$ et telle que

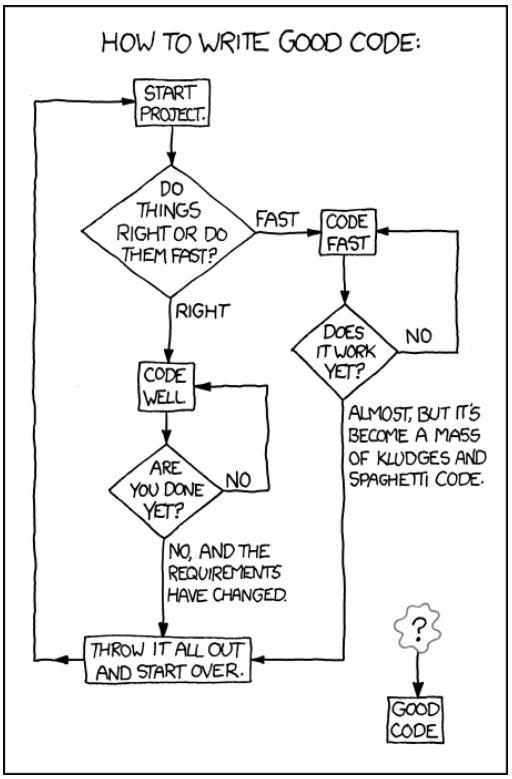
$$\texttt{nb\_petits\_pois} = a_1 \times 1! + a_2 \times 2! + \cdots + a_p \times p!$$

avec $a_1 + \cdots + a_p$ minimal.

Votre programme `range_petits_pois(nb_petits_pois)` doit renvoyer la liste `[a1, a2, ..., ap]` (unique) qui permettra à l'assistant du marchand de préparer les boîtes avant d'y ranger ses petits pois. Bien entendu, tout comme les factoriels, le nombre de petits pois à ranger peut vite devenir astronomique [9] et il ne faut pas que le programme prenne plus de 1 s pour donner sa réponse.

Stop Gitlab

Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.



`You can either hang out in the Android Loop or the HURD loop.`

xkcd.com

9. Le marchand a tendance à faire ses achats en gros, mais on assure que ce nombre n'excèdera pas 10^{1700}