TP06 Algorithmes plus complexes

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

Applications directes

I.1 Recherche dans une liste

Implémentez la fonction `recherche(element, liste)` qui, si `element` appartient à `liste`, renvoie l'indice où l'on peut trouver cette élément. Renvoie `None` sinon. Attention, vous n'avez pas le droit à la méthode `index`, il faut le faire « à la main »

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.

I.2 Recherche d'un maximum

Implémentez la fonction `maximum(liste)` qui renvoie le maximum d'une liste en complexité linéaire. Attention, vous n'avez pas le droit à la fonction `max` ni le droit de trier votre liste pour prendre le dernier élément (ce n'est d'ailleurs pas linéaire car le tri a un coût qu'il ne faut pas oublier).

Stop Gitlab

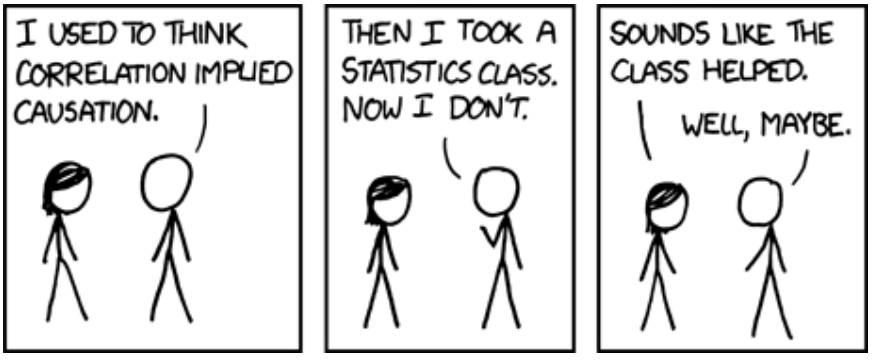
Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

I.3 Calcul de la moyenne et de la variance

Implémentez la fonction `moyenne_et_variance(liste)` qui, à partir d'une liste de nombres, renvoie la moyenne ($\langle u \rangle$) et la variance ($\langle (u - \langle u \rangle)^2 \rangle$) de la liste. Il peut être utile de se faire une fonction `moyenne` annexe que l'on pourra utiliser un nombre raisonnable de fois dans la fonction demandée.

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.




```
Correlation doesn't imply causation, but it does waggle its eyebrows suggestively
and gesture furtively while mouthing 'look over there'.
```

Partie II

Triangle de Pascal

1. Implémenter une procédure `bino(n,p)` permettant de calculer le coefficient binomial $\binom{n}{p}$ défini par

$$\binom{n}{p} = \begin{cases} \dfrac{n!}{p!(n-p)!} & \text{si } p \in [\![0\,;\, n]\!] \\ 0 & \text{sinon} \end{cases}$$

On pourra définir une fonction annexe `factoriel(n)` qui renvoie la factorielle de l'entier `n`.

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.

2. On souhaite construire le triangle de Pascal qui peut s'écrire de la façon suivante

$$\begin{array}{cccc} \binom{0}{0} & & & \\ \binom{1}{0} & \binom{1}{1} & & \\ \binom{2}{0} & \binom{2}{1} & \binom{2}{2} & \\ \binom{3}{0} & \binom{3}{1} & \binom{3}{2} & \binom{3}{3} \end{array} \quad \text{ou encore} \quad \begin{array}{cccc} 1 & & & \\ 1 & 1 & & \\ 1 & 2 & 1 & \\ 1 & 3 & 3 & 1 \end{array}$$

En d'autre termes, il faut écrire une procédure `pascal(n)` qui définisse ce triangle (sous forme d'une liste de listes de longueurs variables) jusqu'à la valeur `n` fournie. L'exemple précédent est donc le résultat de `pascal(3)`, qui devrait renvoyer la liste `[[1],[1,1],[1,2,1],[1,3,3,1]]`.

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.

3. Plutôt que de faire des boucles de boucles, il peut être intéressant de procéder de manière récursive (pensez à la fonction `triangle_haut` du TP03). Écrivez une fonction `pascal_rec(n)` qui produit la même chose que la fonction précédente, mais de manière récursive. Comme la dernière fois, la fonction doit s'appeler elle-même, les boucles imbriquées sont proscrites et il faut limiter l'usage des instructions conditionnelles au minimum.

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.

4. Il y a néanmoins une manière plus facile de calculer les termes successifs du triangle en utilisant la formule de Pascal qui relie le terme de la ligne n aux termes de la lignes $n-1$ qui lui sont au-dessus et à gauche. Visuellement, cela donne

$$\begin{array}{lcc} & \text{colonne } p-1 & \text{colonne } p \\ \text{ligne } n-1 & \binom{n-1}{p-1} + & \binom{n-1}{p} \\ & & \downarrow \\ \text{ligne } n & & \binom{n}{p} \end{array}$$

   En utilisant la formule précédente, implémentez une fonction `pascal2(n)` qui renvoie le triangle de Pascal sans utiliser d'appel à la fonction `bino(n,p)`. Comparez la vitesse d'exécution des deux fonctions `pascal(n)` et `pascal2(n)` pour $n = 1, 10, 50, 100, 200, 300, 400$.

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.

5. À présent que l'on sait construire les éléments du triangle de Pascal facilement, on peut résoudre le problème 203 du projet Euler. Les 8 premières lignes ($n = 7$) du triangle de Pascal s'écrivent

```
1
1  1
1  2  1
1  3  3  1
1  4  6  4  1
1  5  10 10 5  1
1  6  15 20 15 6  1
1  7  21 35 35 21 7  1
```

   On peut voir que ces huit premières lignes du triangle contiennent douze nombres distincts : 1, 2, 3, 4, 5, 6, 7, 10, 15, 20, 21 et 35. Un entier positif n est dit « squarefree » si aucun carré d'un nombre n'est parmi les diviseurs de n. Des douze nombres distincts de ces huit premières lignes du triangle de Pascal, tous sont « squarefree » mis à part 4 et 20. La somme de tous ces nombres « squarefree » des huit premières lignes vaut alors 105. Écrivez un programme `squarefree_pascal(n)` qui calcule la somme des nombres « squarefree » distincts présents dans les `n+1` premières lignes du triangle de Pascal. On pourra utiliser un ensemble (« `set` ») pour s'assurer que chaque nombre n'est compté qu'une seule fois (voir `help(set)` pour plus de détails).

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

Pour réfléchir un peu (Projet Euler 18 et 67), programmation dynamique

Considérons le triangle suivant :

3
7 4
2 **4** 6
8 5 **9** 3

En partant du sommet et en descendant uniquement selon les chiffres adjacents sur la ligne du dessous, la somme maximale que l'on peut obtenir est 3+7+4+9 = 23. Écrivez une fonction `somme_maximale(triangle)` qui, étant donné un triangle proposé comme une liste de liste comme ceux de la section précédente, calcule la somme maximale sur un des chemins qui mène du sommet à la base.

Une petite mise en garde néanmoins : pour un triangle de N lignes, il existe $2^N - 1$ chemins différents du sommet à la base. S'il est plausible de tous les explorer par une méthode « force brute » pour de faibles valeurs de N (disons jusqu'à 20 environ), cela n'est plus possible pour un triangle de 100 lignes. La méthode se doit d'être un peu plus réfléchie. [1] ;-)

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. Une idée (SPOILER) : utilisez un triangle « dual » de même forme que le triangle initial mais qui, pour chaque case, contient la somme du chemin maximal permettant d'atteindre cette case (pas besoin de se souvenir du chemin, seule la somme maximale suffit).