Projet BLOC2 Groupe G3D

Thème : **Le Crêpier psychorigide**

Problème : « A la fin de sa journée, un crêpier dispose d'une pile de N crêpes de diamètres différents empilées aléatoirement. Le crêpier étant un peu psychorigide, il décide de trier sa pile de crêpes, de la plus grande (en bas) à la plus petite (en haut). Pour cette tâche, le crêpier peut faire une seule action : glisser sa spatule entre deux crêpes et retourner la ou les crêpes qui sont au-dessus de la spatule. Comment doit-il procéder pour trier toute la pile ? »

Nous distinguerons 2 cas, l'un ou les crêpes ont des faces identiques, l'autre ou les crêpes ont une face brûlée, que l'on voudra cacher (coté brûlé vers le bas)

Capacités du programme :

Tris par insertion, par sélection	Écrire un algorithme de tri. Décrire un invariant de boucle qui prouve la correction des tris par insertion, par sélection.	La terminaison de ces algorithmes est à justifier. On montre que leur coût est quadratique dans le pire cas.

- Prototyper une fonction, décrire les préconditions, vérifier dans l'exécution le respect de l'invariant de boucle (assertion Python)

Prérequis : notion en Python, notion de liste, les boucles, instructions conditionnelles, lire et modifier les éléments d'un tableau grâce à leur index, notation a[i][j]

Objectifs de la séance : résoudre un problème et l'expliquer par un algorithme en langage naturel, le modéliser en liste et le formaliser en langage Python (ou compléter le programme en Python proposé), identifier un invariant de boucle. Revoir notion de boucle, récursivité et instructions conditionnelles.

Analogie avec le tri sélection et l'invariant de boucle associé.

En prolongement, appréhension de la notion de complexité, recherche de meilleur ou pire cas et cas moyen, et de meilleur algorithme

I. 1ERE ACTIVITE : L'ALGORITHME DE TRI ET LA NOTION D'INVARIANT (1ER CAS : LES CREPES ONT DES FACES IDENTIQUES)

Etape 1 : Expliquer les contraintes aux élèves.

https://www.youtube.com/watch?v=tI6uTAlX-_w&feature=youtu.be (du début à 1minute 45 secondes)

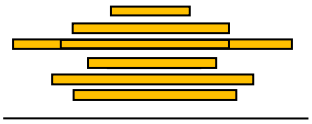
Etape 2 : Manipulation élève

Laisser un certain temps aux élèves pour essayer de résoudre le problème à l'aide de 5 plaques ou de 5 morceaux de papier de tailles différentes

Etape 3 : Etablir un algorithme en langage naturel (langage crépier)

Considérer un camarade comme un ordinateur, et donner lui des instructions simples qu'il exécutera automatiquement pour atteindre l'objectif. Ecrire l'algorithme correspondant.

Correction : Type d'algorithme attendu : (en langage naturel du crêpier)

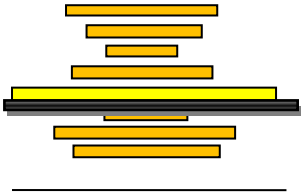


ordonnerCrêpes (k crêpes supérieures)

S'il y a moins de deux crêpes

on ne fait rien

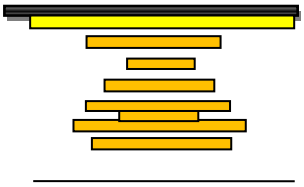
Sinon



On cherche la plus grande dans la pile #(la i-ème parmi k)

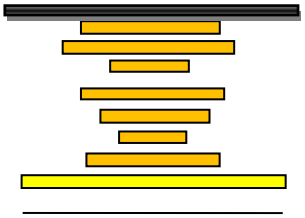
On place la spatule sous la plus grande crêpe

on retourne la pile haute (celle au-dessus de la spatule)



on retourne toute la pile de crêpes

(*)



On recommence avec la pile des k-1 crêpes supérieures

la pile est complètement triée par l'appel de ordonnerCrêpes (N crêpes supérieures)

Etape 4 : invariant de boucle, explication du professeur, recherche par les élèves

Dégager une ou des proprétés en langage naturel

CORRECTION ATTENDUE

(*) la plus grande crêpe de la pile des k crêpes supérieure est en-dessous

(*) invariant de boucle : (version cumulée de la propriété précédente) :

Après n itérations, la pile inférieure des n crêpes inférieures est ordonnée.

II. 1ERE SYNTHESE

Définition : Un algorithme, c'est une suite d'instructions, qui une fois exécutée correctement, conduit à un résultat donné.

Un algorithme doit donc contenir uniquement des instructions compréhensibles par celui qui devra l'exécuter. Il faut être méthodique et rigoureux, il faut aussi systématiquement se mettre mentalement à la place de la machine qui va l'exécuter.

La vérification méthodique, pas à pas, de l'algorithme représente plus de la moitié du travail à accomplir... et le gage de vos progrès.

Un invariant décrit une propriété qui lie les variables entre elles , après l'exécution d'une instruction donnée de l'algorithme. La correction de l'algorithme nécessite le respect de l'invariant après chaque exécution de cette instruction.

III. IMPLEMENTATION PYTHON

1. Modélisation et Implémentation Python.

Faire écrire ou compléter (selon le niveau), le programme en Python (modélisation des crêpes en liste d'entiers, assertion….)

Correction : Exemple de programme en langage Python :

L'état de la pile de crêpes est modélisé par :

- un tableau, tas[], de n entiers
  tas [i] désigne le diamètre (entier) de la (i-1)ème crêpe
  tas [0] désigne le diamètre (entier) de la crêpe tout en bas

```python
#bas du tas à gauche, haut du tas droite

tas=[20,18,15,14,11,9,6,5,3,2]

N=len(tas)

def echanger(i,j):
    global count
    aux=tas[i]
    tas[i]=tas[j]
    tas[j]=aux
    count+=3

def idMax(n):
    global count
    idM=n
    for i in range(n+1,N): #n+1 pour éviter redondance
        if tas[idM]<tas[i]:
            idM=i
            count+=1
    return idM

def renverser(i0,i1):
    global count

    assert(i0<=i1)
    #for i in range((i1-i0)//2+1):
    for i in range((i1-i0)//2+1):
        echanger(i0+i,i1-i)

count=0
def ordonnerCrepe(n):
    if n < N:
        idM=idMax(n) # n
        print(tas)
        renverser(idM,N-1) #n
        print(tas)
        renverser(n,N-1)
        #M.append(tas.copy())
        if n>0 : assert(max(tas[n-1:])==tas[n-1])
        ordonnerCrepe(n+1)

ordonnerCrepe(0)
print(count)
```

2. Expression des propriétés en python : l'instruction `assert` *

Avant l'appel récursif :

```
if n>0 : assert(max(tas[n-1 :])==tas[n-1])
```

3. Respect de l'invariant :

On propose une implémentation Python volontairement buggée. L'exécution du programme fait apparaitre le viol de l'invariant.

On demande le diagnostic du bug et une correction possible.

On met en regard la conformité du programme à sa spécification par un jeu de test adapté et dans le même temps le respect de l'invariant (absence d'exception Python provoquée par le non respect de l' « assert »)

IV. Activite 2 : Optionnel (2eme cas, on veut maintenant que la face brulee des crepes soit vers le bas)

On demande d'adapter l'algorithme précédent de manière à ce que la face brûlée de chaque crêpe soit cachée (face tournée vers le bas)

1) Comment modéliser les crêpes selon leur diamètre et leur coté brulé ou pas
2) Quel test doit-on rajouter dans l'algorithme précédent?
3) Ouvrir le fichier « crepebrulee.py » **(le programme bugué)**
4) Identifier dans le code, le test proposé en question 2
5) Faites tourner le programme, Que se passe-t-il ?
6) Chercher le problème **(notion d'assertion)**
7) Ouvrir le fichier « crepebruleedebug.py » et constater qu'il répond à la question

V. PROLONGEMENTS 1 : Un tri classique analogue et la preuve de l'invariant (analogue)

1. Tri par sélection : algorithme

On cherche à trier dans l'ordre croissant un tableau `tab` d'entiers, de longueur N.

L'état du tableau pendant le tri est caractérisé de la manière suivante :

• une partie gauche triée (vide au début) et une partie droite non triée telle que les valeurs de la partie gauche triée soient toutes inférieures ou égales à celles de la partie droite non triée.

• On **sélectionne** la **plus petite** valeur de la partie non triée et la place à la fin de la partie triée par une procédure d'échange des valeurs dans le tableau.

• On recommence cette opération tant qu'il reste plus de deux « cartes » dans la partie non triée.

Exo 1 : Appliquer l'algorithme de tri par sélection à la main pour trier la liste d'entiers : {72, 39, 29, 59, 17, 54}. Décrire la liste modifiée à chaque étape de votre tri.

Constat attendu : des analogies avec le tri des crêpes.

☞ *Implémentation de l'algorithme*

On considère un tableau de taille `N`. On rappelle que l'index du dernier élément est `N−1`.

• L'algorithme contient une boucle **`Pour`** qui parcourt le tableau entre l'index 0 et l'avant-dernier `N−2` : son compteur `i` indique la première position de la partie non triée.

• Une fonction `indexMinimum` retourne l'index noté `iMin` de la plus petite valeur du tableau entre l'index `i` et `N−1`.

• Dans le tableau les valeurs d'index `iMin` et `i` sont échangées.

Exo 2 : Programmer en Python l'algorithme de tri par la sélection du minimum.

```
def indexMinimum(i) qui renvoie l'indice du minimum de tab[i+1 :n]

définition echangerValeurs(i,j) qui echange les valeurs du tableau aux indices
i et j

Définition triSelection()

    Pour i de 0 à N-2

        iMin=indexMinimum(i)

        echangerValeurs(i,iMin)

    Fin_Pour

    afficherTableau()
```

2. Enoncé et Preuve de l'invariant

(*) propriété $\min(Tab[i:]) = Tab[i]$

(*) invariant de boucle : version cumulée :

Après i itérations, Tab[0 :i] est contient les i plus petites valeurs triées de Tab.

Preuve de la propriété $\min(Tab[i:]) = Tab[i]$,

puis description intuitive de l'invariant de boucle (version cumulée) (nécessite un raisonnement par récurrence Hors programme en 1ere)

VI. Autres Prolongements possibles : Notion de complexite

- déterminer un ordre de grandeur en complexité du programme (1^{er} ou $2^{ème}$ cas).
- insérer des compteurs d'étapes pour voir si l'algorithme est efficace.
- proposition d'amélioration de l'algorithme (nouvelle assertion, éviter des étapes si la liste est déjà triée, chercher des exemples de listes entrainant des pires cas…

1. Choix de l'unité de complexité

Afin de déterminer un ordre de grandeur en complexité du programme, choisit-on le nombre d'accès au tableau ou le nombre de coups de spatules ?

On peut appréhender la complexité avec le nombre de coups de spatules mais la comparaison avec la complexité de tris classiques (exprimés en accès tableau) devra tenir compte d'un facteur n introduit par chaque retournement de tableau provoqué par la spatule.

2. Meilleur cas et pire cas

- Questionnement : Quel est le meilleur cas ?
  Réponse attendue : tas déjà trié
  Que fait l'algorithme ? il fait le travail quand même…
- et le pire cas ? trié à l'envers ? pas sûr…
  Que fait l'algorithme ? il fait le travail quand même alors qu'il pourrait le faire en un coup de spatule

Constat : algorithme brutal dont la complexité est complètement déterministe en fonction de N

3 . Comment améliorer l'algorithme ?

Un test sur l'invariant permettra de ne pas faire le travail si l'invariant est respecté à l'entrée de la fonction. (Un tel test ne modifie pas la correction de l'algorithme.)

ordonnerCrêpes (k crêpes supérieures)

S'il y a moins de deux crêpes
on ne fait rien
Sinon

On cherche la plus grande dans la pile #(la i-ème parmi N)

Si (la plus grande crêpe de la pile des k crêpes supérieure n'est pas déjà en-dessous)
On place la spatule sous la plus grande crêpe
on retourne la pile haute (celle au-dessus de la spatule)

on retourne toute la pile de crêpes

(*)

On recommence avec la pile des N-1 crêpes supérieures

la pile est complètement triée par l'appel de ordonnerCrêpes (N crêpes supérieures)

Le cas d'un tas trié à l'envers au départ devient alors presque un meilleur cas.

La complexité n'est alors plus une fonction de la seule variable N (taille du tableau) mais aussi du tableau Tas[] lui-même.

4 . Recherche de cas défavorables :

On pourra insérer des compteurs de spatules et générer des piles à configurations « complexes »,

5 . Recherche statistiques de cas moyens :

on pourra insérer des compteurs de spatules et générer des piles à configurations « aléatoires »,

Remarque :
La complexité de ce problème a fait l'objet d'un article de B.Gates en 1979 (cf wikipedia)

Un majorant de P, nombre minimal de manipulations a été déterminé en 2008 par $\frac{18N}{11}$

6 . Comment montrer que l'implémentation du tri des crêpes n'est pas le meilleur algorithme possible (si c'est le cas…)?

En partant d'une pile ordonnée, on génère aléatoirement N_{alea} retournements de spatules inverses de ceux déclenchés par l'algorithme.
Le décompte N_{algo} des coups de spatules générés par l'algorithme permet la comparaison.
S'il existe un $N_{alea} < N_{algo}$, alors l'algorithme n'est pas le meilleur possible.