Table des matières

1 Présentation de l’activité 1 **1**
- 1.1 Enoncé 1
- 1.2 Consignes 1
- 1.3 solution possible : celle qui semble le plus naturel 1
- 1.4 Enoncé 1
- 1.5 solution possible : avec le curseur 2

2 Formalisation des algorithmes **2**
- 2.1 Représentation du jeu et réécriture des algorithmes 2
- 2.2 complexité des algorithmes dans le point 1.5 3
- 2.3 réflexions sur l’implémentation - hors sujet 3

1 Présentation de l’activité 1

1.1 Enoncé

Version 1

On considère un jeu de cartes classique avec le même nombre de cartes , Les cartes forment un paquet battu et sont retournées.

Contrainte : On ne peut retourner que la carte du dessus.

But : être sûr d’avoir regroupé les cartes rouges et noires, et à ce moment on dit "stop".

1.2 Consignes

Vous disposez d’un paquet de 12 cartes. Élaborez votre stratégie, dont vous noterez succinctement les étapes.~~Passez la feuille à votre voisin qui devra trier un paquet avec vos indications.~~

1.3 solution possible : celle qui semble le plus naturel

Avec la création deux paquets dont chacun dédié à une couleur, puis regroupement des deux paquets en un tas par superposition. Voici une version avec un seul deuxième paquet.

```
Données : un jeu battu de cartes retournées
Résultat : le jeu de cartes avec les cartes rouges en premier et les noires ensuite
préparer un nouveau paquet avec la première carte retournée (face cachée sur la table)
tant que notre paquet de départ n'est pas vide faire
    retourner la carte du dessus
    si elle est noire alors
        la mettre au dessus
    sinon
        la mettre sous le nouveau paquet
    fin
fin du tant que
on dit "Stop"
```

Algorithme 1 : À l’aide d’un second paquet

1.4 Enoncé

Version 2

À la contrainte, du premier on ajoute :

Contraintes : On ne peut retourner que la carte du dessus.

On ne peut pas créer un deuxième paquet (par exemple pas de tables à notre disposition), donc la seule opération possible est de l’insérer dans le paquet à n’importe quel endroit.

But : être sûr d’avoir regroupé les cartes rouges et noires, et à ce moment on dit "stop".

1.5 solution possible : avec le curseur

Trouver l'algorithme est plus compliqué maintenant. Les difficultés possibles :
— Pensez à indiquer l'endroit de séparation des rouges et noires
— Bien positionnée la carte noire

Données : un jeu battu de cartes retournées
Résultat : le jeu de cartes avec les cartes rouges en premier et les noires ensuite

Répéter *tant qu'on n'a pas retourné 12 cartes* **faire**
  retourner la carte du dessus
  si *elle est noire* **alors**
    Insérer la carte au-dessus de la carte indiquant la première carte rouge (*si elle n'existe pas la mettre sous le paquet*)
  sinon
    la mettre sous le paquet (*s'il s'agit de la première carte rouge on la fait un peu sortir du paquet*)
  fin
fin
On dit "STOP"

Algorithme 2 : On parcourt tout le paquet

Données : un jeu battu de cartes retournées
Résultat : le jeu de cartes avec les cartes rouges en premier et les noires ensuite

Répéter *tant qu'on n'a pas retourné 6 cartes rouges* **faire**
  retourner la carte du dessus
  si *elle est noire* **alors**
    Insérer la carte au-dessus de la carte indiquant la première carte rouge (*s'elle n'existe pas la mettre sous le paquet*)
  sinon
    la mettre sous le paquet. (*s'il s'agit de la première carte rouge on la fait un peu sortir du paquet*)
  fin
fin
On dit "STOP"

Algorithme 3 : En parcourant le "minimum" de cartes

Remarque On pourrait penser que l'algorithme3 est bien plus rapide, que l'algorithme 2 mais en fait on a une probabilité $\frac{1}{2}$, qu'il fasse exactement le même nombre de répétitions (la dernière carte du paquet de départ est noire). Voir plus loin pour davantage de renseignements

2 Formalisation des algorithmes

Nous ne pouvons pas ou du moins difficilement apporter la preuve que les algorithmes fonctionnent dans tous les cas, même si intuitivement cela semble correct. Il faut formaliser les procédés afin de prouver les algorithmes.

Il faut répondre aux interrogations suivantes :

1. Comment représenter un jeu de cartes ? Les couleurs ?
2. En quoi les boucles **tant que** et **Répeter** ne bouclent pas à l'infini ?
3. l'algorithme retourne-t-il bien un jeu trié ?

2.1 Représentation du jeu et réécriture des algorithmes

Représentation de notre jeu :
Un jeu de cartes est représenté par une liste ordonnées d'entiers, dont la valeur représente une couleur :
$L = (L_0, L_1, L_2, \ldots, L_n) = (L_i)_{0 \leqslant i \leqslant n}$ où n+1 est le nombre de cartes.

Données : $L = (L_0, L_1, L_2, \ldots, L_n)$
Résultat : Pour $i < \frac{n}{2}$ on a $L_i = 0$ et pour $i \geqslant \frac{n}{2}$ on a $L_i = 1$

$N \leftarrow ()$
Pour i *allant de 0 à n* **faire**
  si $L_i = 1$ **alors**
    $N \leftarrow (N_0, ..., L_i)$
  sinon
    $N \leftarrow (L_i, N_0, ...)$
  fin
fin
on dit "Stop"

Algorithme 4 : version formalisée

Données : un jeu battu de cartes retournées
Résultat : le jeu de cartes avec les cartes rouges en premier et les noires ensuite

préparer un nouveau paquet avec la première carte retournée (face cachée sur la table)
tant que *notre paquet de départ n'est pas vide* **faire**
  retourner la carte du dessus
  si *elle est noire* **alors**
    la mettre sous le nouveau paquet
  sinon
    la mettre au dessus
  fin
fin du tant que
on dit "Stop"

Maintenant on va prouver que l'algorithme fonctionne :

Données : $L = (L_0, L_1, L_2, \ldots, L_n)$
Résultat : Pour $i < \frac{n}{2}$ on a $N_i = 0$ et pour $i \geqslant \frac{n}{2}$ on a $N_i = 1$ avec...

$N \leftarrow ()$
 # Variant de boucle $i \leqslant n$
 # Invariant de boucle $\mathscr{P}(i)$: pour $j < cur$, on a $N_j = 0$ et pour $cur \leqslant j < i$, on a $N_j = 1$
 # $cur \leftarrow -1$
Pour *i allant de 0 à n* **faire**
  si $L_i = 1$ **alors**
    $N \leftarrow (N_0, \ldots, L_i)$
  sinon
    # $cur \leftarrow cur + 1$
    $N \leftarrow (L_i, N_0, \ldots)$
  fin
fin
on dit "Stop"

Algorithme 5 : version formalisée

Preuve La preuve s'effectue en deux parties, la terminaison et la preuve partielle.

La terminaison : la suite i est une suite d'entier naturel strictement croissante, donc elle dépassera n et la condition $i \leqslant n$ n'étant plus satisfaite, la boucle se termine.

La preuve partielle : supposons qu'à l'étape i, on ait $\mathscr{P}(i)$ c'est à dire pour $j < cur$ on a $N_j = 0$ et pour $cur \leqslant j < i$, on a $N_j = 1$.

À fin de l'étape i deux cas se présentent :

$L_i = 1$: dans ce cas pour $j < cur$ on a $N_j = 0$ (car l'élément a été ajouté à la fin)
pour $cur \leqslant j < i$, on a toujours $N_j = 1$ (hypothèse de récurrence ?)
pour $j = i$ on a $N_j = L_i = 1$, donc
pour $cur \leqslant j \leqslant i < i + 1$, on a toujours $N_j = 1$.

$L_i = 0$: ... difficile de rédiger sans introduire N' la nouvelle valeur de N...

2.2 complexité des algorithmes dans le point 1.5

Pour l'algorithme 2, on a toujours 12 répétitions.

Pour la suite on peut noter X la variable aléatoire qui égale au nombre minimal de tirages pour avoir le paquet trié. Pour l'algorithme 3 :

1. Dans le meilleur des cas : 6 répétitions avec un paquet déjà trié (rouges puis noires). $P(X = 6) = \frac{6!6!}{12!} = \frac{1}{924}$
2. Dans le pire des cas, la dernière carte du tas est noire et donc on a 12 répétitions. $P(X = 12) = \frac{1}{2}$
3. En moyenne : la probabilité d'avoir $6 + k$ tirages est $P(X = 6 + k) = \binom{5+k}{5} \times \frac{6!6!}{12!}$

Nbr répétitions ($X = 6 + k$)	6	7	8	9	10	11	12
probabilité	$\frac{1}{924}$	$\frac{1}{154}$	$\frac{1}{44}$	$\frac{2}{33}$	$\frac{3}{22}$	$\frac{3}{11}$	$\frac{1}{2}$

Donc la moyenne est $\dfrac{78}{7} \simeq 11,14$

Pour l'algorithme 3(bis) : il y a aucune raison de favoriser une couleur par rapport à une autre, et donc la condition devient on s'arrête dés que l'on a retourné 6 cartes d'une même couleur.

1. Dans le meilleur des cas : 6 répétitions, paquet déjà trié. $\frac{2 \times 6!6!}{12!} = \frac{1}{462}$
2. Dans le pire des cas et tout dépend du choix de la couleur mise en dessous, si elle est rouge on la met sous le paquet et si elle noire on n'a pas à bouger la carte, donc la probabilité est $\dfrac{1}{2}$.
3. En moyenne : ...

2.3 réflexions sur l'implémentation - hors sujet

Cette section n'a pas d'intérêt dans le cadre de l'algorithmique, mais la question de comment implémenter l'algorithme et le coup en temps est intéressant. Voici quelques résultats (Cf Annexe 2), on crée mille fois un paquet de mille cartes qu'on trie :

TRI avec la méthode du drapeau hollandais inversion de deux cartes 0.7092394939973019
TRI avec la méthode du drapeau hollandais création d'une nouvelle liste 06.852948751999065
TRI avec la création d'une seconde liste 0.7922300749996793

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@authors: Pascal,Patrick
"""

from random import randrange,shuffle

class cartes:
    """
        modélisation d'un jeu de cartes
    """
    def __init__(self,n,couleurs = 2, equi = True):
        self.couleurs=couleurs
        self.taille = n
        if equi == True:
            p = self.taille//couleurs
            r = n - p*couleurs # si la taille n'est pas un multiple de la couleur
            self.main = [i for j in range(p) for i in range(couleurs)]
            if r !=0:
                for i in range(r):
                    self.main.append(couleurs-1)
            shuffle(self.main) # on mélange le paquet
        else:
            self.main = [randrange(0,couleurs) for i in range(self.taille)]

    def melanger(self):
        shuffle(self.main)

    def tri2A(self):
        """
        Tri des couleurs avec une seule liste en mémoire à la manière du
        drapeau hollandais
        Seulement deux couleurs
        """
        pr = self.taille # indice à partir duquel on est sûr d'avoir des rouges
        # on l'initialise au successeur du dernier terme
        pa = 0
        while pa < pr:
            if self.main[pa] == 0: # on fait le choix que 0 représente la couleur noire
                pa += 1
            else:
                pr -= 1
                self.main[pa], self.main[pr] = self.main[pr], self.main[pa]
        return self.main

    def tri2B(self):
        """
        Tri des couleurs avec une seule liste en mémoire
        mais on ne retourne que la carte du dessus
        Seulement deux couleurs
        """
        pn = self.taille-1 # indice à partir duquel on range les noires

        for i in range(self.taille):
            c=self.main[0] # la première carte
            if c == 0: # on fait le choix que 0 représente la couleur noire
                # insertion de la carte à sa place
                self.main = self.main[1:pn+1]+[c]+self.main[pn+1:]
            else: # la carte est rouge
                self.main =self.main[1:]+[c] # on la met à la fin
            pn -= 1 # l'indice diminue
        return self.main

    def triB(self):
        """
        Tri des couleurs avec une seule liste en mémoire en insérant les couleurs au bon endroit,
        Nombre de couleurs quelconque
        """
        lp=[self.taille-1 for i in range(self.couleurs)]
        # indices des indices où insérer les cartes pour chaque espace de couleur
        for i in range(self.taille):
            c=self.main[0] # la première carte
            # insertion de la carte à sa place
            self.main = self.main[1:lp[c]+1]+[c]+self.main[lp[c]+1:]
            for p in range(c+1):
                lp[p]-=1 # Les indices des couleurs inférieures diminuent
        return self.main

    def tri2C(self):
        """
        Tri des couleurs avec la construction d'une nouvelle liste
        Seulement 2 couleurs
        """
        nv = []
        nv.append(self.main[0])
        for pa in range(1,self.taille):
            if self.main[pa] == 0:
                nv.insert(0, self.main[pa])
            else:
                nv.append(self.main[pa])
        return nv
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Jun 10 10:17:43 2019

"""

from random import randrange,shuffle
from cartes import *

print("Jeu avec 2 couleurs")
jeu = cartes(20,2)
print(jeu.main)
print("Tri 2C (2 couleurs) - ne modifie pas la main")
print(jeu.tri2C()) # ne modifie pas la main
print("Tri 2A (2 couleurs)")
print(jeu.tri2A())
print("mélange du jeu de carte")
jeu.melanger()
print(jeu.main)
print("Tri 2B (2 couleurs)")
print(jeu.tri2B())

print("Jeu avec 4 couleurs")
jeu4 = cartes(32,4)
print(jeu4.main)
print("Tri B avec 4 couleurs")
print(jeu4.triB())

print("jeu non equi")
jeuF = cartes(32,4,equi=False)
print(jeuF.main)
print("Tri B avec 4 couleurs (cas non equi)")
print(jeuF.triB())


import timeit
test1 = timeit.Timer("jeu = cartes.cartes(1000,2);jeu.tri2A()", "import cartes;")
print("TRI A",test1.timeit(1000))
test2 = timeit.Timer("jeu = cartes.cartes(1000,2);jeu.tri2B()", "import cartes;")
print("TRI B",test2.timeit(1000))
test3 = timeit.Timer("jeu = cartes.cartes(1000,2);jeu.tri2C()", "import cartes;")
print("TRI C",test3.timeit(1000))
```