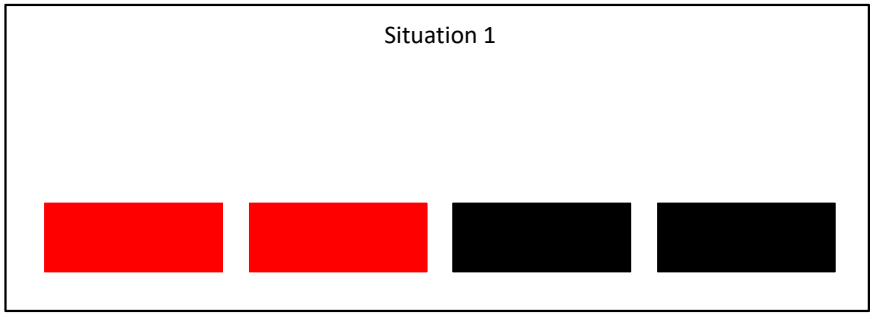
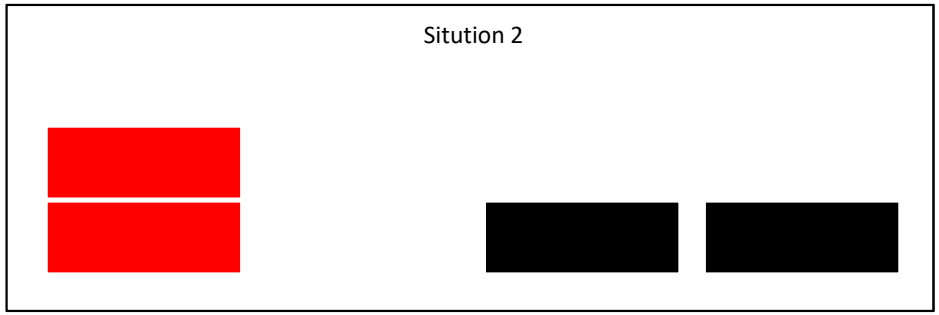
Q1-Etat de la liste x **dans les 2 situations**





- Nombre de tablettes /couleur : N=2
- Couleurs : ci=0 pour le rouge, ci=1 pour le noir

- Situation 1: x = [[0,1] , [0,1] , [1,1] , [1,1]]
- Situation 2: x = [[0,2],[1,1],[1,1]]

Q2-Fonction empile(x, i, j) prenant en paramètres la liste de situations x et les indices i,j de 2 piles pi et pj tels que i≠j.

La fonction retourne la liste de listes LXij des situations atteignables par empilement.

```
def empile( x,i,j ):
    """ empile ( x : list, i : int, j : int )-> list
        entrees : x, liste de listes(ci,hi), represente la situation initiale.
                : i, j, entiers, indices des piles que l'on veut empiler
        sortie; LXij , liste de situations possibles après empilement           """
    LXij = []
    ci , hi = x[i]
    cj , hj = x[j]
    if ci==cj or hi==hj : #empilement possible
        reste = x[:i]+x[i+1:j]+x[j+1:]
        pij = [ci,hi+hj] # pile ci sur cj (arbitrairement)
        X = reste + [pij]
        X.sort()
        LXij.append(X)
        if ci!=cj: # si couleur différente => ajouter la situation avec cj au dessus
            pji = [cj,hi+hj] # pile cj sur ci
            X = reste + [pji]
            X.sort()
            LXij.append(X)
    return LXij
```

- Q3 **Fonction coups(x)** qui prend en paramètre une situation x (liste) et retourne une liste

```
def coups(x):
    """ coups(x : list) -> list
      entree: x, liste de liste(ci,hi), situation initiale
      sortie: ensemble des situations possibles avec ttes les combinaisons d'empilement possibles
    """
    LX = []
    for i in range( len(x) ):
        for j in range( i+1 , len(x) ):
            LXij = empile( x,i,j )
            for X in LXij:
                if X not in LX: # On n'ajoute la situation que si elle n'a pas déjà été stockée
                    LX.append(X)
    return LX
x = [ [0,2],[0,3],[1,1],[1,2] ]
lesCoups1 = coups(x)
print( lesCoups1 )
# affiche [[[0, 5], [1, 1], [1, 2]], [[0, 3], [0, 4], [1, 1]], [[0, 3], [1, 1], [1, 4]],
[[0, 2], [0, 3], [1, 3]]]
x = [[0,1],[0,1],[1,1],[1,1]]
lesCoups2 = coups(x)
print( lesCoups2 )
# affiche [[[0, 2], [1, 1], [1, 1]], [[0, 1], [0, 2], [1, 1]], [[0, 1], [1, 1], [1, 2]],
[[0, 1], [0, 1], [1, 2]]]
```

- **Partie 2 : Création du graphe du jeu**
- Q4-**Fonction init(C, N)** prenant en argument le nombre de couleurs C et le nombre de tablettes par couleur N et renvoyant la liste x0 situation initiale du jeu triée

```
def init(C,N):
    """ init( C : int, N: int ) -> list
        entrees : C, entier positif, nombre de couleurs differentes
                : N , entier positif, nombre de tablettes par couleur
        sortie : x0, liste de listes (ci,hi) correspondant à la situation initiale
    """
    x0 = []
    for c in range(C):
        for _ in range(N):
            x0.append([c,1])
    x0.sort() # Inutile compte tenu de ma programmation avec c croissant
    return x0

test = init( 3, 4 )
print(test)
# affiche [[0, 1], [0, 1], [0, 1], [0, 1], [1, 1], [1, 1], [1, 1], [1, 1], [2, 1], [2, 1]
, [2, 1], [2, 1]]
```

- Q5-F**onction récursive Tuple(L)** réalisant cette transformation de liste en Tuple.

```
def Tuple(L):
    if L==[]:
        return ()
    elif type(L[0])!=list:
        return tuple(L)
    else:
        Res = []
        for l in L:
            Res.append(Tuple(l))
        return tuple(Res)
```

- Q6- **Fonction graphe(C , N)** réalisant le parcours en largeur des possibilités.

```
from collections import deque
def graphe(C,N):
    """ graphe(C : int ,N:int )-> dict
        entrees : C, entier positif, nombre de couleurs differentes
                : N , entier positif, nombre de tablettes par couleur
        sortie : G, dictionnaire, representant le graphe :clé : tuple de la situation, valeur :
                                liste de listes de situations atteignables   """
    # constitution de la situation initiale
    x0 = init(C,N)
    G = {} # initialisation du dictionnaire G qui représente le graphe
    file = deque()
    file.append(x0) # initialisation de la file avec la situation initiale x0
    while len(file) != 0:
        x = file.popleft()  # recuperation du 1er element de la file
        Cle = Tuple(x)      # génération de la cle sous forme de tuple
        Lc = coups(x)       # recuperation de la liste des situations atteignables depuis x
        Valeur = Lc
        G[Cle] = Valeur # ajout dans le graphe du couple clé, valeur
        #ajout ds la file des differentes situations(si pas déjà été stockées)pr traitt ultérieur
        for c in Lc:
            if Tuple(c) not in G:
                file.append(c)
    return G
```

Q7- Que représentent N1 et N2 ?

```
N1 = len(Graphe)
N2 = sum([len(Graphe[x]) for x in Graphe])
```

N1 : nombre de sommets (12 sur l'exemple)
N2 : nombre d'arêtes (16 sur l'exemple)

Q8- Remplissage tableau :

	N =	1	2	3	4
C = 1	Sommets	1	2	3	5
	Arêtes	0	1	2	5
	Temps (s)	$2{,}1.10^{-5}$	$8{,}0.10^{-6}$	$1{,}6.10^{-5}$	$2{,}9.10^{-5}$
C = 2	Sommets	3	12	43	133
	Arêtes	2	16	90	386
	Temps (s)	$1{,}1.10^{-5}$	$8{,}8.10^{-5}$	$1{,}0.10^{-3}$	$2{,}8.10^{-2}$
C = 3	Sommets	7	92	696	4220
	Arêtes	6	234	2832	23 487
	Temps (s)	$2{,}6.10^{-5}$	$3{,}9.10^{-3}$	1,2	6560
C = 4	Sommets	23	728		
	Arêtes	48	3040		
	Temps (s)	$9{,}1.10^{-4}$	1,6		