TP : Le juste prix par dichotomie

Alice choisit un entier au hasard, par exemple entre 1 et $n = 100$. Bob doit deviner le nombre choisi par Alice avec un minimum de réponses. Pour chaque réponse de Bob, Alice lui indique si sa réponse est trop grande, trop petite ou correcte.

Bob utilise une stratégie de dichotomie, dont le principe est expliqué ci-dessous :

Par exemple, si le nombre à deviner est 54 :

- on propose 50, on nous dit c'est plus.
- on propose alors 75 le milieu de 50 et 100, on nous dit c'est moins. Le nombre est entre 50 et 75.
- on propose alors 62 le milieu de 50 et 75, on nous dit c'est moins. Le nombre est entre 50 et 62.
- on propose alors 56 le milieu de 50 et 62, on nous dit c'est moins. Le nombre est entre 50 et 56.
- on propose alors 53 le milieu de 50 et 56, on nous dit c'est plus. Le nombre est entre 53 et 56
- on propose alors 54 le milieu de 53 et 56. C'est gagné.

Nous allons utiliser deux variables a et b initialisées aux valeurs respectives 1 et 100 et maintenir invariant que le juste prix est compris entre a et b. Nous allons itérer tant qu'il y a une réponse possible (c'est-à-dire lorsque $b - a \geqslant 0$) ou jusqu'à ce qu'on tombe sur la bonne réponse.

Une version presque juste

```
n = 100
justeprix = randint(1,n) # on tire un entier au hasard entre 1 et n
a, b = 1, n
nbre_reponses = 0
# on va maintenir invariant que le justeprix est compris au sens large entre a et b
# on va faire décroître strictement la quantité b-a pour sortir de la boucle.

while b-a >= 0: # tant qu'il y a une réponse possible
    rep = (a+b)//2
    nbre_reponses += 1
    if rep == justeprix:
        print("la bonne réponse est:", rep)
        break # # on est tombé sur la bonne réponse, on sort de la boucle
    elif rep > justeprix:
        b = rep
    else:
        a = rep
print("le nbre de réponses est:", nbre_reponses)
print("le juste prix est ", justeprix)
```

On fait des tests, cela semble parfait, pourtant cet algorithme est faux!! En effet, il boucle indéfiniment lorsque le juste prix à trouver est 100 ou 56.

Essayons de comprendre pourquoi :

par exemple, si le nombre à deviner est 100 :

- on propose 50, on nous dit c'est plus.
- on propose alors 75 le milieu de 50 et 100, on nous dit c'est plus. Le nombre est entre 75 et 100.
- on propose alors 87 le milieu de 75 et 100, on nous dit c'est plus. Le nombre est entre 87 et 100.
- on propose alors 93 le milieu de 87 et 100, on nous dit c'est plus. Le nombre est entre 93 et 100.
- on propose alors 96 le milieu de 93 et 100, on nous dit c'est plus. Le nombre est entre 96 et 100.
- on propose alors 98 le milieu de 96 et 100, on nous dit c'est plus. Le nombre est entre 98 et 100.
- on propose alors 99 le milieu de 98 et 100, on nous dit c'est plus. Le nombre est entre 99 et 100.

Arrivé ici, on va à nouveau proposer 99 le milieu de 99 et 100, et boucler indéfiniment. On observe ainsi que la quantité $b-a$ qui est décroissante ne l'est pas strictement, puisqu'elle peut stagner à 1.

La version corrigée

On pourrait penser que la quantité $b-a$ est strictement décroissante car divisé par 2 à chaque itération. Ce n'est pas tout à fait exact, c'est ce qui conduit à notre erreur. En effet, si on note a' et b' la valeur des variables a et b après une itération, on peut avoir $b'-a' = \frac{b-a}{2}$ ou $b'-a' = \frac{b-a}{2} \pm \frac{1}{2}$, ainsi $b'-a' \leqslant \frac{b-a+1}{2}$ et donc $b'-a'-1 \leqslant \frac{b-a-1}{2}$. C'est donc la quantité $b-a-1$ qui est au moins divisée par 2. Ainsi au bout de k itérations, elle est inférieure à $(n-1-1)/2^k$ donc inférieure à 1 dès que 2^k dépasse $n-2$, donc pour $k > \log_2(n-2)$. Dans ce cas, $b-a-1 < 1$, ie $b-a < 2$, c'est-à-dire $b-a \leqslant 1$. Dans ce cas , il reste au maximum deux réponses possibles, il faut s'arrêter d'itérer. La nouvelle condition d'arrêt est donc «tant que $b-a > 1$».

L'algo effectue donc au plus $\log_2(n-2)+1$ itérations, et il lui reste alors deux tests au maximum.

```
n = 100
justeprix = randint(1,n)
a, b = 1, n
nbre_reponses = 0
# on va maintenir invariant que le justeprix est compris au sens large entre a et b
# on va faire décroître strictement la quantité b-a pour sortir de la boucle.
```

```
while b-a > 1:
    rep = (a+b)//2
    nbre_reponses += 1
    if  rep == justeprix:
        print("la bonne réponse est:", rep)
        break # # on est tombé sur la bonne réponse, on sort de la boucle
    elif rep > justeprix:
        b = rep
    else:
        a = rep

# arrivé ici, soit on a breaké car on a trouvé le juste prix qui vaut rep,
#soit b-a <= 1, et le justeprix étant compris entre a et b, c'est soit a, soit b.
if  b-a <= 1:
    if  justeprix == a:
        print("la bonne réponse est:", a)
    else:
        print("la bonne réponse est:", b)

print("le nbre de réponses est:", nbre_reponses)
print(justeprix)
```

Une version un peu meilleure, notre modèle

```
from random import randint

n = 100000
justeprix = randint(1,n)
a = 1
b = n
nbre_reponses = 0
# on va maintenir invariant que le justeprix est compris au sens large entre a et b
# on va faire décroître strictement la quantité b-a pour sortir de la boucle.

while b-a >=0: # tant qu'il y a une réponse possible
    rep = (a+b)//2
    nbre_reponses += 1
    if  rep == justeprix:
        print("la bonne réponse est:", rep)
        break # # on est tombé sur la bonne réponse, on sort de la boucle
    elif rep > justeprix:
        b = rep - 1
    else:
        a = rep + 1
# arrivé ici, il n'y a plus de réponse possible, on a donc trouvé le juste prix
print("le nbre de réponses est:", nbre_reponses)
print(justeprix)
```

A chaque itération, la quantité $b-a+1$ est au moins divisée par 2. En effet, on peut avoir $b'-a' = \frac{b-a}{2} - 1$ (lorsque $b-a$ est pair) ou $b'-a' = \frac{b-a}{2} - 1 \pm \frac{1}{2}$ (lorsque $b-a$ est impair), ainsi $b'-a' \leqslant \frac{b-a-1}{2}$ et donc $b'-a'+1 \leqslant \frac{b-a+1}{2}$. On veut itérer jusqu'à ce $b-a<0$, donc jusqu'à ce que $b-a+1<1$. Or au bout de k itérations, $b-a+1$ est inférieure à $n/2^k$ donc est inférieure à 1 dès que 2^k dépasse n, donc pour $k > \log_2(n)$. L'algo effectue donc au plus $\lceil \log_2(n) \rceil$ itérations.