

PROGRAMMATION DYNAMIQUE

1. Entiers atteignables

1. Pour $n = 6$, les deux premiers sauts sont nécessairement vers la droite et amènent en position 3. Le saut suivant peut amener soit en position 0 (saut à gauche), soit en position 6 (saut à droite). Le saut suivant se fait alors dans la direction opposée et amène soit en position 4, soit en position 2. A partir de là, n'importe quel saut de longueur 5 fait sortir de la règle. Ainsi,

L'entier 6 n'est pas atteignable.

2. On peut proposer la séquence de sauts suivante : chaque couple indique la direction du saut et la position à l'issue du saut :

$(D, 1), (D, 3), (D, 6), (G, 2), (D, 7), (G, 1), (D, 8), (G, 0), (D, 9)$

Ainsi,

L'entier 9 est atteignable.

3. Après 0 sauts (respectivement 1 puis 2), on ne peut atteindre que la position 0 (resp. 1, puis 2). Le troisième saut peut en revanche amener soit en position 0, soit (lorsque $n \geq 6$) en position 6. Ainsi,

$L_0 = [\text{True}, \text{False}, \text{False}, \dots, \text{False}]$
 $L_1 = [\text{False}, \text{True}, \text{False}, \dots, \text{False}]$
 $L_2 = [\text{False}, \text{False}, \text{False}, \text{True}, \text{False}, \dots, \text{False}]$
 $L_3 = [\text{True}, \text{False}, \text{False}, \text{False}, \text{False}, \text{False}, \text{True}, \text{False}, \dots, \text{False}]$

4. Pour aboutir en position i à l'issue du k -ième saut, il faut nécessairement être en position $i - k$ ou $i + k$ à l'issue du $(k - 1)$ -ième saut. Il suffit alors de faire un saut vers la droite dans le premier cas, vers la gauche dans le second. On peut donc atteindre la position i si et seulement si l'une des deux positions $i - k$ ou $i + k$ est bien un indice dans $\llbracket 0; n \rrbracket$, et cette position peut être atteinte en $k - 1$ sauts. Ainsi,

$b_{i,k} = (b_{i-k,k-1} \text{ or } b_{i+k,k-1})$

Si $i < k$, alors l'indice $i - k$ est en dehors de l'intervalle $\llbracket 0; n \rrbracket$. On ne peut donc pas atteindre la position i depuis la position $i - k$. Il en est de même si $i + k > n$. On peut donc corriger la formule de la manière suivante :

$$b_{i,k} = \begin{cases} (b_{i-k,k-1} = \text{si } i < k \text{ et } i + k \leq n \\ (b_{i+k,k-1} = \text{si } i \geq k \text{ et } i + k > n \\ \text{False} = \text{si } i < k \text{ et } i + k > n \end{cases}$$

5. Il suffit d'appliquer les formules de la question précédente.

```
1 def next_L(L,k):
2     n=len(L)-1
3     L1=[False for i in range(n+1)]
4     for i in range(n+1):
5         if i-k>=0:
6             L1[i]=L1[i] or L[i-k]
7         if i+k<=n:
8             L1[i]=L1[i] or L[i+k]
9     return L1
```

6. Il suffit d'initialiser une liste L à la valeur de L_0 de la question 3, puis d'utiliser la fonction de la question précédente pour calculer successivement $L_1, \dots, L_n$. Une fois cette dernière calculée, le résultat est simplement donné par $L_n[n] = b_{n,n}$.

```
1 def atteignable(n):
2     L=[False for i in range(n+1)]
3     L[0]=True
4     for k in range(1,n+1):
5         L=next_L(L,k)
6     return L[n]
```

2. Problème du carré maximal

1. Par définition, on a directement

$$(c_{i,j})_{i,j \in \llbracket 0;3 \rrbracket} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 2 \\ 0 & 1 & 1 & 2 \end{pmatrix}$$

2. Par définition, si $i = 0$ ou $j = 0$, il n'existe qu'un seul carré dont la case d'indice (i, j) est le coin inférieur droit et celui-ci ne comporte que la case en question. La valeur de $c_{i,j}$ vaut donc 1 ou 0, suivant que la case contient un 1 ou un 0.

$$\boxed{\text{Si } i = 0 \text{ ou } j = 0, \quad c_{i,j} = a_{i,j}}$$

3. Soient $i, j \geq 1$. Justifions la relation donnée. Le premier cas est évident. Pour le second, on procède par double inégalité (avec un dessin si possible)

- Par définition, le carré de côté $c_{i,j}$ de coin inférieur droit (i, j) ne contient que des 1. En particulier, cela assure que les 3 carrés de côté $c_{i,j} - 1$ de coin inférieur droit $(i, j - 1)$, $(i - 1, j)$ ou $(i - 1, j - 1)$ ne contiennent que des 1. Par suite, $c_{i,j-1}$, $c_{i-1,j}$ et $c_{i-1,j-1}$ sont tous les trois supérieurs ou égaux à $c_{i,j} - 1$, donc leur minimum également. Ainsi,

$$c_{i,j} \leq 1 + \min \{c_{i,j-1}, c_{i-1,j}, c_{i-1,j-1}\}$$

- Réciproquement, notons p le minimum de $c_{i,j-1}$, $c_{i-1,j}$ et $c_{i-1,j-1}$. Le carré de côté p et de coin inférieur droit $(i, j - 1)$ est inclus dans celui de côté $c_{i,j-1}$ de même coin inférieur droit. Il ne contient donc que des 1. Il en est de même pour celui de côté p et de coin inférieur droit $(i, j - 1)$ ou $(i - 1, j - 1)$. Par conséquent, le carré de côté $p + 1$ et de coin inférieur droit (i, j) ne contient que des 1. Cela prouve que

$$c_{i,j} \geq 1 + \min \{c_{i,j-1}, c_{i-1,j}, c_{i-1,j-1}\}$$

Finalement

$$\boxed{\forall i, j \geq 1, \quad c_{i,j} = 1 + \min \{c_{i,j-1}, c_{i-1,j}, c_{i-1,j-1}\}}$$

4. L'algorithme consiste simplement à calculer les coefficients $(c_{i,j})_{i,j \in \llbracket 1;n \rrbracket \times \llbracket 1;m \rrbracket}$ ligne par ligne ou colonne par colonne en stockant les résultats dans une matrice au fur et à mesure (stratégie bottom-top). Cet ordre de calcul garantit que la formule de la question précédente peut toujours être appliquée : au moment du calcul de $c_{i,j}$, les trois valeurs $c_{i,j-1}$, $c_{i-1,j}$ et $c_{i-1,j-1}$ ont toutes déjà été calculées. Il ne reste ensuite plus qu'à parcourir la matrice à la recherche du plus grand coefficient.

```

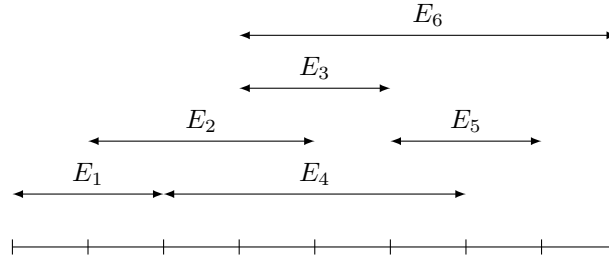
1 def carre_max(A):
2     n=len(A); m=len(A[0])
3     c=[[0 for j in range(m)] for i in range(n)]
4     # initialisation 1ere ligne/colonne
5     for i in range(n):
6         c[i][0]=A[i][0]
7     for j in range(m):
8         c[0][j]=A[0][j]
9     # calcul du reste des coefficients
10    for i in range(1,n):
11        for j in range(1,m):
12            if A[i][j]==0:
13                c[i][j]=0
14            else:
15                c[i][j]=1+min(c[i-1][j],c[i][j-1],c[i-1][j-1])
16    # recherche du maximum
17    (im,jm,cm) = (0,0,c[0][0])
18    for i in range(n):
19        for j in range(m):
20            if c[i][j]>cm:
21                (im,jm,cm) = (i,j,c[i][j])
22    return (im,jm,cm)

```

Chaque calcul de cet fonction se fait en temps constant. Le coût global est donc proportionnel à la longueur des boucles imbriquées, donc en $O(n \cdot m)$.

3. Problème du téléspectateur

1. On peut proposer la représentation graphique suivante :



A partir de ce dessin, on constate facilement qu'on peut regarder E_1 puis E_3 et enfin E_5 sans qu'il soit possible de regarder 4 émissions ou plus. Par suite,

Le nombre maximal d'émissions que le téléspectateur peut regarder est 3.

2. Il est clair que $m_1 = 1$ (il suffit de regarder l'émission E_1). Pour E_2 , si $a_2 \geq b_1$, alors l'émission E_2 commence après l'émission E_1 , on peut donc les regarder toutes les deux (en zappant très vite si $a_2 = b_1$). Dans ce cas, $m_2 = 2$. Sinon, il est impossible de les regarder toutes les deux, et dans ce cas, $m_2 = 1$.

$$m_1 = 1 \quad \text{et} \quad m_2 = \begin{cases} 2 & \text{si } a_2 \geq b_1 \\ 1 & \text{sinon} \end{cases}$$

3. On dit que deux émissions E_p et E_q sont compatibles lorsque les intervalles $]a_p; b_p[$ et $]a_q; b_q[$ sont disjoints. Séparons l'ensemble des sélections d'émissions deux à deux compatibles dans $\{E_1, \dots, E_i\}$ en deux sous-ensembles :

- celui des sélections qui incluent l'émission E_i . Puisque l'émission E_i débute à l'heure a_i , une telle sélection ne contient aucune émission parmi $E_{k(i)+1}, \dots, E_{i-1}$ (qui terminent après le début de E_i par définition de $k(i)$). Si on ôte E_i de la sélection, on obtient donc une sélection d'émissions deux à deux compatibles dans $\{E_1, \dots, E_{k(i)}\}$.
- celui des sélections qui n'incluent pas l'émission E_i . Ils s'agit donc de sélection d'émissions deux à deux compatibles dans $\{E_1, \dots, E_{i-1}\}$.

Ainsi, les sous-ensembles de cardinal maximal sont à chercher parmi

- les sous-ensembles de $\{E_1, \dots, E_{k(i)}\}$ de cardinal maximal, auxquels on ajoute l'émission E_i , qui sont ainsi par définition de cardinal $m_{k(i)} + 1$.
- les sous-ensembles de $\{E_1, \dots, E_{i-1}\}$ de cardinal maximal, lequel vaut alors m_{i-1} par définition.

Par conséquent

$$m_i = \max(m_{i-1}, m_{k(i)} + 1)$$

4. La suite $(b_i)_{i \in [1; n]}$ étant triée par ordre croissant, la recherche de l'élément $k(i)$ peut se faire par dichotomie. Une version en pseudo-code pourrait prendre la forme suivante. On maintient deux indices d et f tels que $b_d \leq a_i < b_f$ (avec la convention $b_0 = -\infty$).

```

Poser  $d = 0$  et  $f = i$ 
Tant que  $d + 1 < f$  Faire
    Poser  $m = (d + f)/2$ 
    Si  $b_m > a_i$ 
        Poser  $f = m$ 
    Sinon
        Poser  $d = m$ 
Fin tant que
Renvoyer  $d$ 

```

5. Si on veut seulement le nombre maximal d'émissions, on peut se contenter de calculer les $(m_i)_{0 \leq i \leq n}$ par une simple boucle.

```
Poser  $m_0 = 0$  et  $m_1 = 1$   
Pour tout  $i$  dans  $\llbracket 2; n \rrbracket$  Faire  
    Trouver le plus grand entier  $k(i)$  tel que  $b_{k(i)} \leq a_i$   
    Poser  $m_i = \max(m_{i-1}, m_{k(i)} + 1)$   
Fin tant que  
Renvoyer  $m_n$ 
```

6. Si on veut une sélection optimale, on peut modifier le code de la manière suivante :

```
Poser  $X_0 = \emptyset$  et  $X_1 = \{E_1\}$   
Pour tout  $i$  dans  $\llbracket 2; n \rrbracket$  Faire  
    Trouver le plus grand entier  $k(i)$  tel que  $b_{k(i)} \leq a_i$   
    Si  $|X_{k(i)}| + 1 > |X_{i-1}|$   
        Poser  $X_i = X_{k(i)} \cup \{E_{k(i)}\}$   
    Sinon  
        Poser  $X_i = X_{i-1}$   
Fin tant que  
Renvoyer  $X_n$ 
```

La recherche de l'entier $k(i)$ se fait par recherche dichotomique, pour un coût logarithmique à chaque étape. La boucle s'exécute n fois. Par conséquent, la complexité est bien quasi-linéaire.