16. On obtient

$$F(2,3) = 7,$$
$$F(3,2) = 7,$$
$$F(3,3) = 9.$$

17. On additionne les poids individuels des arêtes:

```
def poids_chemin(c):
    p = len(c) - 1
    s = 0
    for k in range(p):
        (i, j) = c[k]
        (a, b) = c[k+1]
        s += P[i, j, a, b]
    return s
```

Variante avec une écriture en compréhension:

```
def poids_chemin(c):
    p = len(c) - 1
    return sum(P[c[k][0], c[k][1],
                 c[k+1][0], c[k+1][1]]
               for k in range(p))
```

18. Un chemin optimal vers (i, j) avec $i > 0$ et $j > 0$ ne peut être obtenu que de deux façons: en arrivant à (i, j) par en-dessous, ou par sa gauche.

    De plus, pour que le chemin vers (i, j) soit optimal, il faut que l'un au moins des chemins vers $(i-1, j)$ et vers $(i, j-1)$ soit optimal (si aucun des deux ne l'est, le chemin vers (i, j) peut être remplacé par un autre chemin de poids strictement inférieur, donc il ne peut pas être optimal).

    Donc la longueur d'un chemin optimal est la plus petite des deux longueurs obtenues en ajoutant $F(i-1, j)$ et $P(i-1, j, i, 1)$ d'une part pour une arrivée avec un dernier pas vers la droite, et en ajoutant $F(i, j-1)$ et $P(i, j-1, i, 1)$ d'une part pour une arrivée avec un dernier pas vers le haut.

19. On obtient

$$F(0,0) = 0,$$
$$\forall i > 0,\ F(i,0) = F(i-1,0) + P(i-1,0,i,0),$$
$$\forall j > 0,\ F(0,j) = F(0,j-1) + P(0,j-1,0,j).$$

20. La méthode récursive telle qu'elle est proposée n'est pas efficace car elle suppose de calculer de nombreuses fois les mêmes valeurs sans tenir compte du fait qu'elles ont déjà été rencontrées par le passé.

Plus précisément, le calcul de $F(i,j)$ par cette méthode nécessite de l'ordre de 2^{i+j} appels récursifs (le nombre total de chemins de $(0,0)$ à (i,j) dans le graphe), alors qu'il n'y a que $(i+1)(j+1)$ sommets à explorer au total.

21. Dans la proposition de programme suivante, chaque ligne ne fait appel qu'à des valeurs déjà calculées dans le tableau.

```
F = np.zeros(n, n)
for i in range(1, n):
    F[i, 0] = F[i-1, 0] + P[i-1, 0, i, 0]
for j in range(1, n):
    F[0, j] = F[0, j-1] + P[0, j-1, 0, j]
for i in range(1, n):
    for j in range(1, n):
        F[i, j] = min(F[i-1, j] + P[i-1, j, i, j],
                      F[i, j-1] + P[i, j-1, i, j])
print("Le poids d'un escalier optimal est ", F[n-1, n-1])
```

22. Le nombre d'additions, d'affectations et de calculs de minimum est $O(n) \cdot O(1) + O(n) \cdot O(1) + O(n) \cdot O(n) \cdot O(1) = O(n^2)$.

23. On peut par exemple procéder comme suit:

```
F = np.zeros(n, n)
D = np.zeros(n, n)
for i in range(1, n):
    F[i, 0] = F[i-1, 0] + P[i-1, 0, i, 0]
    D[i, 0] = 1
for j in range(1, n):
    F[0, j] = F[0, j-1] + P[0, j-1, 0, j]
    D[0, j] = 2
for i in range(1, n):
    for j in range(1, n):
        gauche = F[i-1, j] + P[i-1, j, i, j]
        bas    = F[i, j-1] + P[i, j-1, i, j]
        if gauche < bas:
            F[i, j] = gauche
            D[i, j] = 1
        else:
            F[i, j] = bas
            D[i, j] = 2
```

24. On remonte le chemin optimal en suivant les indications contenues dans la matrice D.

```
def reconstruire(n, D):
    i, j = n-1, n-1
    chemin = [(i, j)]
    while i > 0 or j > 0:
```

```
    if D[i, j] == 1:
        i -= 1
    else:
        j -= 1
    chemin.append((i, j))
chemin.reverse()
return chemin
```