Remplissage de rectangles, solution

Dans ce problème, on tente compter le nombre de manières de remplir un rectangle $m \times n$, qu'on va appeler le rectangle principal, avec de plus petits rectangles, non-carrés. Il faut voir le remplissage comme une séquence ORDONNÉE de petits rectangles qu'on va insérer un à un pour jusqu'à ce que le rectangle principal soit rempli. Pourquoi ordonné? Parce qu'on veut pas compter deux fois la même configuration. Mettre d'abord un rectangle 2×3, coin supérieur gauche à la case $(4, 6)$, puis ensuite un rectangle 3×1, coin supérieur gauche à la case $(9, 9)$, ça donne exactement la même configuration que mettre d'abord un rectangle 3×1, coin supérieur gauche à la case $(9, 9)$, puis ensuite un rectangle 2×3, coin supérieur gauche à la case $(4, 6)$.

On doit donc se fixer un unique protocole pour choisir le prochain rectangle. Pour faciliter les calculs, voici la méthode adoptée

1. Soit (i, j) la case vide la plus à gauche (j minimal). En cas d'égalité entre deux cases, prendre la plus haute (i minimal). Alors le prochain rectangle doit couvrir cette case.

De cette manière, on ne comptera pas deux fois la même configuration. De plus grâce à cette manière, on a la garantie qu'une case (i, j) ne sera jamais remplie avant une case (i, k) si $k < j$.

L'algorithme qui suit est un algorithme de programmation dynamique, alors il faut définir ce que sera un état, et dans notre cas, ce sera un état de remplissage.

Soit C l'ensemble des cases à remplir. Un état de remplissage sera défini comme étant un ensemble $Y \in C$ qui contient les cases remplies. À chaque état de remplissage Y, on associe un score $count(Y)$ qui compte le nombre de manières de placer des rectangles de manière à arriver à cet état de remplissage. Par exemple $count(\emptyset) = 1$ et $count(C) = f(m, n)$, la réponse finale.

Maintenant, il faut une formule de propagation, et elle est plutôt simple. Soit A l'ensemble des états de remplissage pour lesquels il est légalement possible d'ajouter un rectangle non-carré et d'aboutir à l'état de remplissage Y. Alors:

$$count(Y) = \sum_{X \in A} count(A)$$

Petite rappel: on peut ajouter un rectangle seulement s'il couvre le coin le plus en haut à gauche disponible, comme expliqué au début.

Ensuite, il faut une structure de donnée pour garder en mémoire tous les états pendant la propagation. Pour cela, une propriété mentionnée plus haut aide. Grâce au fait qu'on priorise le remplissage des cases plus à gauche d'abord, jamais dans un état, une case est remplie si une case plus à gauche de la même rangée ne l'est pas encore. Un état se caractérise donc entièrement par le nombre de cases remplies par rangées, ce qui fait qu'il y a au plus $(n+1)^m$ états créés pendant la propagation. Mieux encore, on peut facilement faire une bijection entre l'intervalle $[0, (n+1)^m[$ et l'ensemble des états possibles. Pour y parvenir,

simplement écrire le nombre de cette intervalle en base $(n+1)$ et le i-ème chiffre indique le nombre de cases remplies dans la rangée i. Grâce à cela, on peut facilement mettre tous ces états dans un tableau de taille $(n+1)^m$.

Cette méthode est assez efficace autant pour le premier problème que pour le troisième problème, car $(11+1)^7$ et $(5+1)^5$ ne sont pas des tailles de tableau excédant la mémoire disponible. Voir Solution.java pour l'implémentation, fonction baseIndexMethod(...)

Toutefois, elle ne l'est pas pour le deuxième problème puisque un tableau de taille $(20203+1)^2$ serait coûteux en espace (et aussi en temps). Il faut donc utiliser un autre méthode: faire des calculs un peu plus théoriques.

Définissons

1. S_n: nombre de manières de remplir un rectangle $1 \times n$, i.e $f(1,n)$
2. T_n: nombre de manières de remplir un rectangle $2 \times n$ tel que les deux cases d'extrême gauche sont couvertes par deux rectangles distincts.
3. G_n: nombre de manières de remplir un rectangle $2 \times n$ tel que les deux cases d'extrême gauche sont couvertes le même rectangle.
4. A_n: nombre de manières de remplir un rectangle $2 \times n$ (i.e.$f(2,n)$)

Alors les équation récursives ci-dessous sont vraies pour tout $n \geq 1$

1. $S_n = \sum_{i=2}^{n-2} S_{i-2}$: car pour remplir un rectangle $1 \times n$, on choisit la longueur i du premier rectangle à gauche (longueur 1 interdite) et il y a ensuite S_{i-2} manières de remplir le reste.
2. $A_n = G_n + T_n$: pour remplir un rectangle $2 \times n$ (i.e.$f(2,n)$), il y a deux possibilités: remplir les deux cases à gauche avec deux rectangles différents ou le même rectangle
3. $G_n = A_{n-1} + \sum_{i=3}^{n} A_{n-i}$: car pour remplir un rectangle $2 \times n$ avec un rectangle de 2 rangées à gauche, on choisit la longueur i de celui-ci (longueur 2 interdite) et on remplit le reste selon n'importe quelle A_{n-i} manière possible.
4. $T_n = \sum_{i=2}^{n-2} S_i^2 G_{n-i}$: car pour remplir un rectangle $2 \times n$ avec deux rectangles différents pour chaque case de gauche, on choisit i, la taille de l'intervalle à partir de la gauche dans lequel on va remplir chaque rangée séparément. Ensuite suivra un rectangle qui couvre les deux rangées, puis autre chose quelconque. Cela fait donc S_i^2 manières de couvrir les deux couches à gauche et G_{n-i} manières de remplir le reste du rectangle.

En débutant avec $S_0 = G_0 = T_0 = A_0 = 1$, ce système récursif est totalement calculable. Cela est fait par le fichier Solution.java, fonction twoLayersCaseCount(int n, int modu). Des opérations modulo sont constamment appliquées.

En conclusion, voici les résponses obtenues:

Solution to first problem is: 81130 Solution to second problem is: 248389853 Solution to third problem is: 271738941100070494