TDP4 : Récursivité (pyramide)

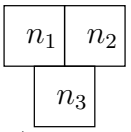
TOP: Techniques and tOols for Programming

Première année

L'objectif de ce TD et du TP associé est de réaliser un algorithme de recherche avec retour arrière pour résoudre ce qu'on appelle le problème de la pyramide.

Présentation du problème On considère une pyramide la tête en bas de hauteur n comme celle représentée plus bas. On cherche à remplir toutes les cases avec chacun des entiers compris entre 1 et $\frac{n(n+1)}{2}$ en respectant les contraintes suivantes :

1. Chaque nombre de $\left[1, \frac{n(n+1)}{2}\right]$ ne figure qu'une fois sur la pyramide
2. La valeur de chaque case est égale à la différence des deux cases placées au dessus d'elle. Ainsi sur la figure ci-contre, $n_3 = |n_1 - n_2|$



Certaines instances du problème (certains n) n'admettent pas de solution (cf. dernier exercice).

★ Exercice 1: Représentation mémoire

La première idée pour représenter la pyramide est d'utiliser la moitié d'un tableau à deux dimensions (`int pyr[ ][ ]`), mais l'autre moitié serait réservée pour rien.

Pour économiser la mémoire, nous allons utiliser un tableau à une dimension en rangeant les différentes «tranches de pyramides» les unes à coté des autres. Selon la façon de couper ces tranches, il existe de nombreuses manières de numéroter les cases. Les figures 2 et 3 présentent deux numérotations différentes.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

FIG. 1 – Représentation mémoire.

10	11	12	13	14
6	7	8	9	
3	4	5		
1	2			
0				

FIG. 2 – Une numérotation par lignes.

0	1	3	6	10
2	4	7	11	
5	8	12		
9	13			
14				

FIG. 3 – Une numérotation par colonnes.

▷ **Question 1:** Écrivez une fonction `indiceLigne(ligne,colonne)` calculant l'indice de la case placée sur la `ligne` et sur la `colonne` indiquée en suivant la numérotation par lignes représentée sur la figure 2. INDICATION : calculez le nombre de case dans la pyramide de taille `ligne`.
`indiceLigne(1,1)=0`; `indiceLigne(2,2)=2`; `indiceLigne(3,2)=4`; `indiceLigne(4,2)=7`.

▷ **Question 2:** Écrivez `indiceColonne(ligne,col)` utilisant la numérotation par colonnes de la figure 3.
`indiceColonne(1,1)=0`; `indiceColonne(2,2)=2`; `indiceColonne(2,3)=4`; `indiceColonne(2,4)=7`.

★ Exercice 2: Algorithme «générer puis tester« (première approche)

La première idée est de générer toutes les pyramides existantes, puis de vérifier à postériori si elles vérifient les contraintes ou non. Il faut donc générer toutes les permutations de la liste des n premiers entiers puis chercher celles vérifiant la seconde contrainte (puisque la première est vérifiée par construction).

▷ **Question 1:** Donnez un algorithme permettant de calculer les permutations des n premiers entiers.

▷ **Question 2:** Écrivez la fonction `correcte()` qui teste récursivement si la pyramide est valide.

▷ **Question 3:** Dénombrez le nombre d'opérations que cet algorithme réalise.

★ Exercice 3: Algorithme de construction pas à pas (deuxième approche)

La solution de l'exercice précédent est inefficace car elle construit toutes les solutions possibles, même celles ne respectant pas les contraintes du problème. Une amélioration possible consiste à vérifier à chaque étape de la construction que ces contraintes sont respectée, et à s'interrompre dès qu'un choix mène à une situation interdite.

Il s'agit donc d'un algorithme récursif avec retour arrière. Pour découvrir le paramètre d'induction, il faut s'interroger sur la façon de couper une pyramide en une pyramide de plus petite taille. La première façon est celle dite «en ligne» (comme sur la figure 4), on ajoute des lignes plus longues. Le paramètre d'induction est alors le numéro de ligne en cours de remplissage. Une autre façon est d'agir «en colonnes» (comme sur la figure 5) on ajoute des «tranches» de la pyramide. Le paramètre d'induction est alors la diagonale en cours de traitement.

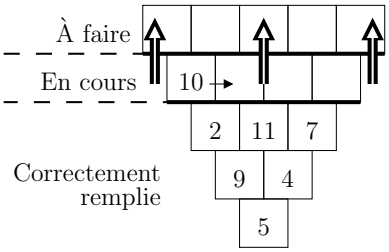


FIG. 4 – Remplissage partiel par lignes.

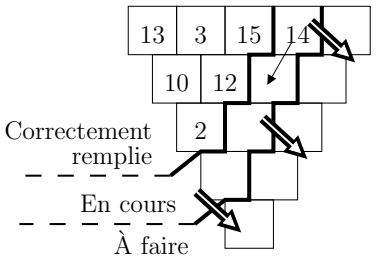


FIG. 5 – Remplissage partiel par colonnes.

L'approche «en colonnes» est préférable car la seconde contrainte lie un nombre aux deux placés au dessus de lui. Il est donc naturel de chercher à traiter le nombre du dessous juste après un nombre donné. Cela permet de s'assurer que toute solution correcte aux étapes précédentes de la récursion ne gênera pas le respect de la seconde contrainte. Au contraire, il est possible que l'approche «en ligne» mène à une situation de blocage due à la seconde contrainte à l'étage n nécessitant de modifier les étages inférieurs.

▷ **Question 1:** Écrire le corps de la méthode `void remplir()`. Elle lance une fonction récursive dotée de plus de paramètres en initialisant les paramètres convenablement.

★ Exercice 4: Écriture de la fonction récursive

Après avoir trouvé le schéma général de la récursion, il nous faut écrire la fonction récursive elle-même.

Le paramètre de la récursion est la diagonale en cours de traitement ; la condition d'arrêt est le dépassement de la taille de la pyramide : si le paramètre `diag` dépasse la hauteur de la pyramide, le problème est résolu et la pyramide est complètement correctement résolue.

Si on parvient à remplir correctement la diagonale `diag`, il convient de réaliser un appel récursif avec `diag+1` pour tenter de remplir le reste de la pyramide. Sinon, on effectue un retour arrière.

On constate que l'on peut calculer (par une série de soustraction) la diagonale entière à partir de la solution partielle et du nombre placé sur la première ligne de de la diagonale. La fonction récursive `remplir(diag)` consiste donc à tester chaque entier en première position de la diagonale, puis à tenter de le «propager», *i.e.* à vérifier que la diagonale induite par ce nombre est valide. On échoue si on est amenés à réutiliser un nombre déjà utilisé dans la pyramide.

▷ **Question 1:** Écrire la méthode `boolean propage(int diag, int val)` qui tente de propager une valeur donnée sur une diagonale donnée.

▷ **Question 2:** Écrire la méthode `boolean remplir(int diag)` qui tente de remplir la diagonale donnée.

★ Exercice 5: Pour aller plus loin

La figure ci-contre donne les temps de calcul (en secondes sur un centrino 1.5Ghz) et les taux de remplissage obtenus pour des pyramides de différentes tailles. La dernière ligne indique donc que la recherche des pyramides de taille 12 a duré plus de 28h, et que la meilleure solution ne remplit que 73% du tableau.

Rang	Remplissage	Temps
2	3/3	0,002
3	6/6	0,002
4	10/10	0,003
5	15/15	0,006
6	20/21	0,12
7	25/28	0,9
8	31/36	6
9	37/45	70
10	43/55	948
11	49/66	11551
12	57/78	103671

Il semble donc que ce problème n'admette pas de solution pour $n > 5$. On peut donc le généraliser de la façon suivante : on s'autorise à laisser de coté des nombres lors du remplissage de la pyramide. Il s'agit alors de minimiser le numéro M de la plus grande valeur utilisée dans une pyramide à n étages n'utilisant que des nombres entiers strictement positifs distincts tels que chaque valeur (sauf celles de la dernière ligne) soit la différence des deux valeurs situées immédiatement au-dessous.

Pour $n = 5$, on a alors $M = \frac{n(n+1)}{2} = 15$

Si vous trouvez une solution pour $n > 5$, merci de la communiquer à l'équipe enseignante (qui n'en connait pas).