INF5130 — Devoir 1

Samuele Giraudo

Session 2025-01

Déclaration. *J'affirme, pour ce devoir, n'avoir reçu aucune aide d'aucune forme que ce soit, incluant les aides par des personnes et par des outils génératifs d'intelligence artificielle. Ceci vaut pour toute la période du devoir, depuis sa date de publication initiale à sa date de rendu finale.*

Je comprends que si j'ai bénéficié de telles aides et que je remplis cette déclaration, elle devient fallacieuse. Cela peut entraîner des sanctions prévues in accordance with what will be decided by the comité des infractions académiques.

Répondre « J'affirme que tout ceci est exact. » et signer par prénom, nom et code permanent. Si ceci manque, le devoir ne pourra pas être évalué.

Réponse — J'affirme que tout ceci est exact. Mélissa Vallée (VALM22549307)

Conventions générales. Tout tableau t de longueur n est indicé de 0 à $n-1$. La notation $t[i]$ désigne la case mémoire à l'indice i de t. La fonction Longueur renvoie la longueur du tableau en argument en temps constant. La fonction CréerTableau prend en entrée un entier naturel n et renvoie un tableau de longueur n dont toutes les cases contiennent la valeur 0. La complexité en temps de cette fonction dans le pire et meilleur cas est linéaire en n.

1 Manipulation de sommes

Question 1.1. *(5 points)* — Soit n un entier naturel. Exprimer la quantité

$$\sum_{i=0}^{n}(2i+1) \tag{1}$$

par une expression polynomiale dépendant de n la plus simplifiée possible. Le calcul doit être détaillé et justifié.

Réponse —
1. On commence par distribuer 2i et 1.

$$\sum_{i=0}^{n}(2i) + \sum_{i=0}^{n} 1 \tag{2}$$

2. Ensuite on réduit l'équation de gauche

$$\sum_{i=0}^{n}(2i) = 2\sum_{i=0}^{n} i = 2\left(\frac{n(n+1)}{2}\right) = n(n+1) \tag{3}$$

3. On réduit maintenant l'équation de droite.

$$\sum_{i=0}^{n} 1 = n+1 \tag{4}$$

4. On peut maintenant assembler l'équation final et la réduire.

$$\begin{aligned}\sum_{i=0}^{n}(2i+1) &= n(n+1)+n+1 && (5)\\ &= (n+1)(n+1) && (6)\\ &= n^2+n+n+1 && (7)\\ &= n^2+2n+1 && (8)\end{aligned}$$

Question 1.2. *(5 points)* — Soient α et β deux entiers tels que $\alpha \leq \beta$. Exprimer la quantité

$$\sum_{i=\alpha}^{\beta} i \tag{9}$$

par une expression polynomiale dépendant de α et de β la plus simplifiée possible. Le calcul doit être détaillé et justifié.

Réponse — 1. Comme on ne connait pas α et β mais $\alpha \leq \beta$, on doit faire l'équation suivante :

$$\sum_{i=\alpha}^{\beta} i = \sum_{i=0}^{\beta} i - \sum_{i=0}^{\alpha-1} i \tag{10}$$

2. On réduit ensuite la partie de gauche :

$$\sum_{i=0}^{\beta} i = \frac{\beta(\beta+1)}{2} \tag{11}$$

3. On réduit la partie de droite :

$$\sum_{i=0}^{\alpha-1} i = \frac{(\alpha-1)(\alpha-1+1)}{2} = \frac{(\alpha-1)\alpha}{2} \tag{12}$$

4. On assemble l'équation finale et on réduit.

$$\sum_{i=0}^{\beta} i - \sum_{i=0}^{\alpha-1} i = \frac{\beta(\beta+1)}{2} - \frac{(\alpha-1)\alpha}{2} = \frac{\beta^2 \ +\beta - \alpha^2 + \alpha}{2} \tag{13}$$

Question 1.3. *(5 points)* — Soit n un entier naturel. Exprimer la quantité

$$\sum_{i=0}^{n}\sum_{j=0}^{m}(i+j) \tag{14}$$

par une expression polynomiale dépendant de n et de m la plus simplifiée possible. Le calcul doit être détaillé et justifié.

Réponse —

1. On commence par prendre la somme intérieur (somme des j) pour la réduire. i sera donc une constante pour l'instant.

$$\sum_{j=0}^{m}(i+j) = \sum_{j=0}^{m} i + \sum_{j=0}^{m} j \tag{15}$$

2. On va ensuite réduire l'expression de gauche considérant que i est une constante.

$$\sum_{j=0}^{m} i = i(m+1) \tag{16}$$

3. Ensuite on réduit l'expression de droite

$$\sum_{j=0}^{m} j = \frac{m(m+1)}{2} \tag{17}$$

4. On assemble maintenant la somme intérieur réduite :

$$\sum_{j=0}^{m} i + \sum_{j=0}^{m} j = i(m+1) + \frac{m(m+1)}{2} \tag{18}$$

5. On doit maintenant faire la somme extérieur de cette expression.

$$\sum_{i=0}^{n}\left(i(m+1) + \frac{m(m+1)}{2}\right) \tag{19}$$

6. On décompose avec maintenant m comme constante

$$\sum_{i=0}^{n}\left(i(m+1)+\frac{m(m+1)}{2}\right)=\sum_{i=0}^{n}i(m+1)+\sum_{i=0}^{n}\frac{m(m+1)}{2} \tag{20}$$

7. On réduit la partie de gauche en sortant $(m+1)$ car c'est une constante

$$\sum_{i=0}^{n}i(m+1)=(m+1)\sum_{i=0}^{n}i=(m+1)\frac{n(n+1)}{2} \tag{21}$$

8. On réduit la partie de droite qui n'est qu'une somme de constante

$$\sum_{i=0}^{n}\frac{m(m+1)}{2}=(n+1)\frac{m(m+1)}{2} \tag{22}$$

9. Finalement on peut assembler et réduire l'équation à sa forme factorisé

$$\sum_{i=0}^{n}\sum_{j=0}^{m}(i+j)=(m+1)\frac{n(n+1)}{2}+(n+1)\frac{m(m+1)}{2} \tag{23}$$

$$=\frac{(m+1)n(n+1)+(n+1)m(m+1)}{2} \tag{24}$$

$$=\frac{(m+1)(n+1)(m+n)}{2} \tag{25}$$

2 Complexité en temps

Question 2.1. *(5 points)* — Considérons l'algorithme décrit par le pseudo-code suivant.

```
1.  Fonction ChercherMot(u, v)
2.      j ← 0
3.      Tant que j ≤ Longueur(v) − Longueur(u) Faire
4.          i ← 0
5.          Tant que i ≤ Longueur(u) − 1 Et u[i] = v[j + i] Faire
6.              i ← i + 1
7.          Fin
8.          Si i = Longueur(u) Alors
9.              Renvoyer j
10.         Fin
11.         j ← j + 1
12.     Fin
13.     Renvoyer −1
14. Fin
```

Cet algorithme prend en entrée deux tableaux u et v dont les valeurs sont des caractères. Donner la description du problème auquel cet algorithme répond. La réponse doit être justifiée. Elle peut s'appuyer sur des exemples bien choisis qui illustrent le comportement de l'exécution de cet algorithme.

Réponse —
En gros cette algorithme permet de regarder si le mot contenu dans le tableau u est inclus dans le mot du tableau v. s'il n'y est pas, cette méthode retourne -1. Ça ressemble un peu à la méthode contains qu'on retrouve dans plusieurs language.

Si par exemple on prend :
u = ['B','O','N','J','O','U','R']
v = ['A','U','R','E','V','O','I','R']
L'algorithme va regarder si $u[0] = v[0]$, comme il ne l'est pas, pas d'incrémentation de i mais incrémentation de j pour chercher plus à partir du prochain index de v, et fait ceci jusqu'a temps que le reste de v soit trop court pour que u y soit. Si u avait été compris dans v le programme aurait retourné l'index du premier caractère où le mot commence dans v

Question 2.2. *(5 points)* — En considérant l'algorithme ChercherMot de la question 2.1, identifier une opération barométrique qui permettra de faire une étude de la complexité en temps de cet algorithme en fonction de n, la longueur de u et m, la longueur de v. La réponse doit être justifiée. Il est possible de répondre à cette question même si la question 2.1 n'a pas été (correctement) traitée.

Réponse —
La ligne 5($i \leq$ Longueur(u) $-$ 1 **Et** $u[i] = v[j + i]$) est l'opération barométrique. Parce qu'elle contient une comparaison qui évalue si on continue d'itérer ou pas sur les caractères de u et c'est, dans le pire cas, l'opération effectué le plus grand nombre de fois dans cette algorithme.

Question 2.3. *(5 points)* — En considérant l'algorithme ChercherMot de la question 2.1 et l'opération barométrique identifiée dans la réponse à la question 2.2, donner une expression $T(n, m)$ en fonction de n et m (les longueurs respectives des tableaux u et v) qui renseigne sur le nombre d'opérations barométriques exécutées dans le pire cas. La réponse doit être justifiée. Il est possible de répondre à cette question même si la question 2.1 n'a pas été (correctement) traitée.

Réponse —
On représente le pire cas par $T(n, m)$.
Le pire cas est une somme de somme si on prend en compte la boucle externe de notre opération barométrique et donc le nombre de fois que la ligne 5 est executé dans le pire cas est :

$$\sum_{j=0}^{m-n} \sum_{i=0}^{n-1} 1 = \sum_{j=0}^{m-n} n - 1 + 1 \tag{26}$$

$$= (m - n + 1)n \tag{27}$$

Parce que dans le pire cas le mot est trouvé au dernier index possible de v et on parcours donc le tableau u en entier. et on renvoit j.

3 Écriture d'algorithmes

Question 3.1. *(10 points)* — Soit t un tableau de longueur n. Ce tableau t est une *permutation* si l'ensemble des valeurs de t est $\{1, \ldots, n\}$. Ceci est équivalent au fait que toute valeur entre 1 et n apparaît exactement une fois dans t. Écrire un algorithme EstPermutation qui prend en entrée un tableau t d'entiers et qui renvoie vrai si t est une permutation et faux sinon. Il faut justifier brièvement du fait qu'il répond à ce problème.

Réponse —

```
1.  Fonction EstPermutation(t)
2.      n ← longueur(t)
3.      elements ← [0] * n
4.      pour i ← 1 haut n Faire
5.          nb ← t[i]
6.          Si nb ≤ n et nb > 0 Et elements[nb] = 0 Alors
7.              elements[nb] ← 1
8.          sinon
9.              renvoyer Faux
10.         Fin
11.     Fin
12.     Renvoyer Vrai
13. Fin
```

L'entrée de la fonction est t, un tableau indexé de 1 à n éléments. Comme on sait que les éléments de t doivent être compris entre 1 et n on crée le tableau $elements$,initialisé à des 0, qui va contenir soit 0 si l'élément n'est pas trouvé ou 1 sinon. on peut donc simplement itérer et valider que les éléments sont unique et bien compris dans les bornes, s'ils ne le sont pas on retourne Faux directement et s'ils le sont tous alors on sort de la boucle et retourne Vrai, ce qui répond à ce problème.

Question 3.2. *(10 points)* — Un indice i d'un tableau t d'entiers de longueur $n \geq 1$ est un *maximum local* si pour tout indice j de t telle que $i < j$, $t[i] > t[j]$. Par exemple, le tableau $t = (6, 0, 6, 0, 2, 1)$ est telle que les indices 2, 4 et 5 sont des maximums locaux. Écrire un algorithme NbMaxLocaux qui prend en entrée un tableau t d'entiers et qui renvoie le nombre de maximums locaux dans t. En considérant le tableau t de l'exemple précédent, l'algorithme renvoie 3. Il est demandé que la complexité en temps dans le pire cas de l'algorithme proposé soit linéaire en fonction de la longueur de t. Il faut justifier brièvement du fait qu'il répond à ce problème et que sa complexité est bien linéaire.

Réponse —

```
1.  Fonction NbMaxLocaux(t)
2.      resultat ← 0
3.      n ← longueur(t)
4.      maxVal ← −∞
5.      pour n − 1 bas i Faire
6.          Si t[i] > maxVal Alors
7.              resultat ← resultat + 1
8.          Fin
9.          maxVal ← max(resultat, t[i])
10.     Fin
11.     Renvoyer resultat
12. Fin
```

Cet algorithme parcourt le tableau t dans le sens inverse, c'est-à-dire de $n - 1$ à 0. Il initialise la valeur maximale $maxVal$ à $-\infty$, puis cherche les nouveaux maximums au fur et à mesure du parcours du tableau. À chaque maximum trouvé, la variable *resultat* est incrémentée. Une fois tout le tableau parcouru, le résultat final est retourné. L'algorithme ne parcourt le tableau qu'une seule fois, ce qui confirme une complexité linéaire, soit $\mathcal{O}(n)$.

4 Croissance des fonctions

Question 4.1. *(5 points)* — Soient les fonctions $f : \mathbb{N} \to \mathbb{R}^+$ et $g : \mathbb{N} \to \mathbb{R}^+$ telles que pour tout $n \in \mathbb{N}$, $f(n) = n^4 + 9n^3 - 6n^2 + n - 7$ et $g(n) = n^4$. Démontrer que $f \in \mathcal{O}(g)$ en donnant explicitement des constantes n_0 et c telles que pour tout $n \geq n_0$, $f(n) \leq cg(n)$.

Réponse —
1. On doit trouver des constantes n_0 et c telles que pour tout $n \geq n_0$, $f(n) \leq cg(n)$.

$$n^4 + 9n^3 - 6n^2 + n - 7 \leq cn^4 \tag{28}$$

2. On pose

$$\begin{aligned} n^4 + 9n^3 - 6n^2 + n - 7 &\leq n^4 + 9n^3 + 6n^2 + n + 7 && \text{(Pour tout } n \geq 0) \\ &\leq n^4 + 9n^4 + 6n^4 + n^4 + 7n^4 && \text{(Pour tout } n \geq 1) \\ &\leq 24n^4 && \text{(Pour tout } n \geq 1) \end{aligned}$$

3. On a donc que $c = 24$ et $n_0 = 1$ comme constantes conviennent.

Question 4.2. *(5 points)* — Une relation binaire $\mathcal{R}$ sur un ensemble A est un *préordre* si $\mathcal{R}$ est réflexive et transitive.

Soit la relation binaire $\mathcal{R}_\mathcal{O}$ sur l'ensemble F des fonctions de domaine $\mathbb{N}$ et de codomaine $\mathbb{R}^+$ telle que pour toutes fonctions $f_1, f_2 \in F$, $f_1 \,\mathcal{R}_\mathcal{O}\, f_2$ si $f_1 \in \mathcal{O}(f_2)$. Démontrer que $\mathcal{R}_\mathcal{O}$ est un préordre.

Réponse —
1. On doit d'abord démontrer que $\mathcal{R}_\mathcal{O}$ est réflexive. Pour démontrer si une relation $\mathcal{R}_\mathcal{O}$ est réflexive, on doit démontrer que $f \in \mathcal{O}(f)$, soit f est en relation avec lui-même. On cherche donc des constantes c et n_0 telles que :

$$\begin{aligned} f(n) &\leq cf(n) && \text{(Pour tout } n \geq 0) \\ &\leq 1 * f(n) && \text{(Pour tout } n \geq 0) \end{aligned}$$

On peut donc prendre $c = 1$ et $n_0 = 1$, qui est toujours vrai.
2. Pour démontrer que la relation est transitive. On doit prendre f_1, f_2, $f_3 \in F$ et prouver que si $f_1 \in \mathcal{O}(f_2)$ et $f_2 \in \mathcal{O}(f_3)$ alors $f_1 \in \mathcal{O}(f_3)$.
On a déjà dans la définition de $\mathcal{R}_\mathcal{O}$ que $f_1, f_2 \in F$, $f_1 \,\mathcal{R}_\mathcal{O}\, f_2$ si $f_1 \in \mathcal{O}(f_2)$. On peut donc déjà confirmer que $f_1 \in \mathcal{O}(f_2)$. qui peut se réécrire avec une constante c_0

$$\begin{aligned} f_1(n) &\leq c_0 f_2(n) && \text{(Pour tout } n \geq 0) \\ &\leq 1 * f(n) && \text{(Pour tout } n \geq 0) \end{aligned}$$

Avec $c = 1$ et $n_0 = 1$.
3.On doit maintenant démontrer que $f_2 \in \mathcal{O}(f_3)$. On peut cette fois prendre une constante c_1.

$$\begin{aligned} c_0 f_2(n) &\leq c_1 c_0 f_3(n) && \text{(Pour tout } n \geq 0) \\ &\leq 1 * c_0 f_3(n) && \text{(Pour tout } n \geq 0) \end{aligned}$$

On prend encore $c_1 = 1$ et $n_1 = 1$. Qui est toujours vrai.
4. Comme on a démontré que $f_1 \in \mathcal{O}(f_3)$ et $f_2 \in \mathcal{O}(f_3)$. On peut démontrer que $f_1 \in \mathcal{O}(f_3)$ en prenant une constante c_2 cette fois.

$$f_1(n) \leq c_1 c_0 f_3(n) \qquad \text{(Pour tout } n \geq 0)$$

$c_2 = c_1 * c_0$, qui pourrait encore une fois être 1, et qui est vrai pour tout. On a donc démontré que $\mathcal{R}_{\mathcal{O}}$ est un préordre.

5 Équations de récurrence et approche « diviser pour régner »

Question 5.1. *(5 points)* — En appliquant la méthode de résolution des équations de récurrence aux différences finies, résoudre la récurrence

$$T(n) = T(n-1) + n^2, \quad \text{si } n \geq 1, \tag{29}$$

$$T(0) = 0. \tag{30}$$

L'expression donnée pour $T(n)$ ne doit pas être récursive. En se basant sur l'expression obtenue, démontrer que $T \in \Theta(n^3)$.

Réponse —
Alors on pose d'abord $T(0) = 0$, ensuite on cherche a définir de manière similaire $T(1)$ jusqu'à $T(3)$ pour avoir un echantillon assé gros.

$$T(0) = 0 \tag{31}$$
$$T(1) = T(0) + 1^2 = 0 + 1 = 1 \tag{32}$$
$$T(2) = T(1) + 2^2 = 1 + 4 = 5 \tag{33}$$
$$T(3) = T(2) + 3^2 = 5 + 9 = 14 \tag{34}$$

Ce qui nous donne cette expression :

$$T(n) = \sum_{i=0}^{n} i^2, n \geq 0 \tag{35}$$

On doit maintenant la réduire pour prouver que $T \in \Theta(n^3)$

$$\sum_{i=0}^{n} i^2 = \frac{(n(n+1)(2n+1)}{6} \tag{36}$$
$$= \frac{(n^2+n)(2n+1)}{6} \tag{37}$$
$$= \frac{2n^3+3n^2+n}{6} \tag{38}$$

Ce qui démontre que $T \in \Theta(n^3)$

Question 5.2. *(10 points)* — Considérons l'algorithme décrit par le pseudo-code suivant.

1. **Fonction** ChercherÉlément(t, x, α, β)
2. **Si** $\alpha > \beta$ **Alors**
3. **Renvoyer** -1
4. **Fin**
5. $m \leftarrow \left\lfloor \frac{\alpha+\beta}{2} \right\rfloor$
6. **Si** $t[m] = x$ **Alors**
7. **Renvoyer** m

8. **Sinon Si** $t[m] > x$ **Alors**
9. **Renvoyer** ChercherÉlément(t, x, α, $m-1$)
10. **Sinon**
11. **Renvoyer** ChercherÉlément(t, x, $m+1$, β)
12. **Fin**
13. **Fin**

Cet algorithme prend en entrée un tableau t dont les valeurs sont des entiers. Ce tableau est de plus trié dans l'ordre croissant de la gauche vers la droite. Il peut contenir plusieurs occurrences d'une même valeur. L'algorithme prend de plus en entrée un entier x, et deux indices α et β tous deux compris entre 0 et la longueur de t moins 1. La fonction ChercherÉlément renvoie un indice i telle que $t[i] = x$ s'il existe et renvoie -1 dans le cas contraire (c'est-à-dire lorsque t ne possède aucune occurrence de x).

Donner, en la justifiant, la fonction de coût $T(n)$ de la forme

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \tag{39}$$

qui mesure la complexité en temps dans le pire cas de cet algorithme, où n est la longueur du tableau t. Il faut ainsi donner les valeurs des constantes a et b, et donner une définition de la fonction $f : \mathbb{N} \to \mathbb{N}$.

Utiliser ensuite le théorème général de résolution des équations de récurrence aux divisions finies pour donner une fonction $g : \mathbb{N} \to \mathbb{R}^+$ telle que $T \in \Theta(g)$.

Réponse —
Comme le tableau t est divisé en deux à chaque appel de la fonction et que l'appel récursif est dans un return ce qui fait qu'elle est appelé une seule fois par appel de fonction, on peut en déduire que $a = 1$ car il n'y a qu'un seul sous-problème. On peut aussi déduire que $b = 2$ car le tableau t est divisé en 2 à chaque fois avant que la fonction ne soit appelé. Ce qui donne l'équation suivante :

$$T(n) = T\left(\frac{n}{2}\right) + f(n) \tag{40}$$

La complexité du reste des opérations de la fonction est $\mathcal{O}(1)$ car ce n'est que des assignations. et donc $f(n) \in \mathcal{O}(1)$.
Théorème général :
1.On doit d'abord trouver c avec la formule

$$c = \log_b(a) = \log_2(1) = 0 \tag{41}$$

2. Ensuite on doit trouver l'exposant d qui est la complexité de la recombinaison des résultats. On cherche d telle que

$$f(n) = \theta(n^d) \tag{42}$$

et on a que $f(n) \in \mathcal{O}(1)$ donc $\mathcal{O}(n^0)$ ce qui fait que $f(n) \in \theta(n^0)$ alors $d = 0$.
3. On cherche maintenant à trouver la catégorie de l'ordre de croissance de T. Comme $c = d$ alors on a qu'il y a équilibre entre les appels récursifs et la recombinaison et donc :

$$T \in \theta(n^c \log(n)) \tag{43}$$

$$T \in \theta(\log n) \tag{44}$$

4. Ce qui nous donne qu'une fonction $g : \mathbb{N} \to \mathbb{R}^+$ telle que $T \in \Theta(g)$ est $g(n) = \log n$

6 Programmation dynamique

Question 6.1. *(15 points)* — Considérons une grille de dimension $n \times m$ avec $n, m \in \mathbb{N}$. Un *chemin* dans cette grille est une suite de pas *Nord* (noté N, qui est un segment vertical de longueur 1) ou *Est* (noté E, qui est un segment horizontal de longueur 1), débutant à l'origine $(0, 0)$ et arrivant au point (n, m). Un chemin est *valide* s'il reste au dessus du segment connectant l'origine et le point (n, m) de la grille. La notion d'« être au dessus » est prise au sens large : un chemin peut toucher la diagonale tout en restant valide — il faut simplement qu'il ne la traverse pas.

Par exemple, pour $n = 4$ et $m = 9$, le chemin $NENNEEEEENEEE$ est valide tandis que le chemin $NNEEEEEENNEEE$ est invalide. Il est fortement conseillé de dessiner la grille et ces deux chemins pour se familiariser avec ces notions (ces dessins ne font cependant pas partie du rendu attendu).

Proposer un algorithme CheminsValides qui prend en entrées les entiers naturels n et m, et qui renvoie le nombre de chemins valides dans la grille $n \times m$. Il faut utiliser le principe de la programmation dynamique en enregistrant, pour tout point (k, ℓ) de la grille avec $0 \leq k \leq n$ et $0 \leq \ell \leq m$, le nombre de débuts de chemins valides qui commencent en $(0, 0)$ et s'arrêtent en (k, ℓ).

En implantant cet algorithme et en l'exécutant, donner les valeurs renvoyées par CheminsValides(n, n) pour tout $0 \leq n \leq 8$ ainsi que par CheminsValides(n, $2n$) pour tout $0 \leq n \leq 8$. Le programme utilisé pour ces calculs ne doit pas faire partie du rendu.

Réponse —

```
1.  Fonction CheminsValides(n, m)
2.      dp[m][n] ← Toutes les cases initialisés à 0
3.      Pour x ← 0 haut m − 1 faire
4.          Pour y ← 0 haut n − 1 faire
5.              Si x = 0 Et y = 0 Alors
6.                  dp[x][y] ← 0
7.              Fin
8.              Si x > 0 Et y ≥ (n/m)x Alors
9.                  dp[x][y] ← dp[x][y] + dp[x − 1][y]
10.             Fin
11.             Si y > 0 Et y ≥ (n/m)x Alors
12.                 dp[x][y] ← dp[x][y] + dp[x][y − 1]
13.             Fin
14.         Fin
15.     Fin
16.     Renvoyer dp[m][n]
17. Fin
```

1.Pour CheminsValides(n, n) :

$$\textsc{CheminsValides}(0,0) = 0 \tag{45}$$
$$\textsc{CheminsValides}(1,1) = 1 \tag{46}$$
$$\textsc{CheminsValides}(2,2) = 2 \tag{47}$$
$$\textsc{CheminsValides}(3,3) = 5 \tag{48}$$
$$\textsc{CheminsValides}(4,4) = 14 \tag{49}$$
$$\textsc{CheminsValides}(5,5) = 42 \tag{50}$$
$$\textsc{CheminsValides}(6,6) = 132 \tag{51}$$
$$\textsc{CheminsValides}(7,7) = 429 \tag{52}$$
$$\textsc{CheminsValides}(8,8) = 1430 \tag{53}$$
$$\tag{54}$$

2. pour les chemins CheminsValides(n,$2n$)

$$\textsc{CheminsValides}(0,0) = 0 \tag{55}$$
$$\textsc{CheminsValides}(1,2) = 1 \tag{56}$$
$$\textsc{CheminsValides}(2,4) = 3 \tag{57}$$
$$\textsc{CheminsValides}(3,6) = 12 \tag{58}$$
$$\textsc{CheminsValides}(4,8) = 55 \tag{59}$$
$$\textsc{CheminsValides}(5,10) = 273 \tag{60}$$
$$\textsc{CheminsValides}(6,12) = 1428 \tag{61}$$
$$\textsc{CheminsValides}(7,14) = 7752 \tag{62}$$
$$\textsc{CheminsValides}(8,16) = 43263 \tag{63}$$
$$\tag{64}$$