Deuxième partie

Thème graphes - Corrigé

Exercice 10 : Degré et connexité

Mines-Télécom, 2025

Q. 1 Quel est le nombre minimal d'arêtes que peut contenir un graphe connexe à n sommets ? Justifier.

Q. 2 Soit G un graphe dont le degré minimal est $\delta \geqslant \frac{n-1}{2}$. Montrer que G est connexe.

Q. 3 Trouver un graphe de degré minimal $\left\lfloor \frac{n}{2} \right\rfloor - 1$ qui n'est pas connexe.

Exercice 11 : Relation d'équivalence

CCINP type A, 2024

On fixe $n \in \mathbb{N}^\star$. Soient $\varphi : [\![1, n]\!] \to [\![1, n]\!]$ et $\psi : [\![1, n]\!] \to [\![1, n]\!]$. Soient u et v deux éléments de $[\![1, n]\!]$. On dit que u et v sont (φ, ψ)-équivalents s'il existe $k \in \mathbb{N}$, un tuple $(w_0, w_1, \cdots, w_{k+1}) \in [\![1, n]\!]^{k+2}$ avec $w_0 = u, w_{k+1} = v$ et vérifiant :

$$\forall i \in [\![0, k]\!], \varphi(w_i) = \varphi(w_{i+1}) \text{ ou } \psi(w_i) = \psi(w_{i+1}).$$

L'objectif est d'écrire un algorithme en pseudo-code permettant de calculer les différentes classes d'équivalence engendrées par cette relation.

Q. 1 Justifier rapidement que "être (φ, ψ)-équivalent" est une relation d'équivalence sur l'ensemble $[\![1, n]\!]$.

Q. 2 Pour cette question, on considère les applications φ et ψ définies par :

$i =$	1	2	3	4	5	6	7	8	9
$\varphi(i) =$	3	2	2	9	6	4	9	5	7
$\psi(i) =$	5	1	3	4	5	1	7	7	4

Calculer les différentes classes d'équivalence.

Q. 3 On revient au cas au général. On définit le graphe $G = (S, A)$ par :

$$S = [\![1, n]\!], A = \{\{x, y\} \in S^2 | x \neq y \text{ et } (\varphi(x) = \varphi(y) \text{ ou } \psi(x) = \psi(y))\}.$$

On fixe x et y deux sommets différents de S. Traduire sur le graphe G le fait que les sommets x et y sont (φ, ψ)-équivalents et en déduire que le calcul des classes d'équivalence de G se traduit en un problème classique sur les graphes que l'on précisera.

Q. 4 Donner en pseudo-code un algorithme permettant de résoudre le problème correspondant sur les graphes.

On fixe n un nombre pair. On considère deux applications φ et ψ de $[\![1, n]\!]$ où tout élément de l'image de φ admet exactement deux antécédents par φ et où tout élément de l'image de ψ admet exactement deux antécédents par ψ.
Pour $f \in \{\varphi, \psi\}$, on note G_f le graphe $(S, \{(x, y) \in S^2 | x \neq y \text{ et } f(x) = f(y)\})$.

Q. 5 Préciser la forme du graphe G_f pour $f \in \{\varphi, \psi\}$.

Q. 6 Expliciter la forme des différentes classes d'équivalence dans le graphe G correspondant.

Exercice 12 : Maximum des degrés

CCINP type B, sujet 0

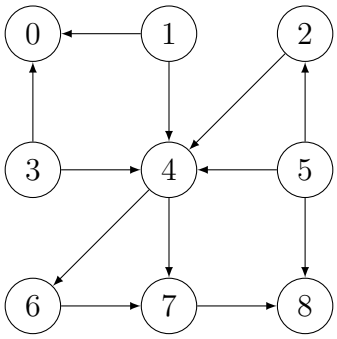
Consignes : L'exercice suivant est à traiter dans le langage C. Cet énoncé est accompagné d'un code compagnon `ccinp_2022_max_degre.c` fournissant le type décrit par l'énoncé et quelques fonctions auxiliaires : il est à compléter en y implémentant les fonctions demandées.

Dans cet exercice, tous les graphes seront orientés. On représente un graphe orienté $G = (S, A)$ avec $S = \{0, ..., n-1\}$ en C par la structure suivante :

```c
struct graph_s {
  int n;
  int degre[100];
  int voisins[100][10];
}
```

L'entier `n` correspond au nombre de sommets $|S|$ du graphe. On suppose que $n \leqslant 100$. Pour $0 \leqslant s < n$, la case `degre[s]` contient le degré sortant $d^+(s)$, c'est-à-dire le nombre de successeurs, appelés ici *voisins*, de s. On suppose que ce degré est toujours inférieur à 10. Pour $0 \leqslant s < n$, la case `voisins[s]` est un tableau contenant, aux indices $0 \leqslant i \leqslant d^+(s)$, les voisins du sommet s. Il s'agit donc d'une représentation par listes d'adjacence où les listes sont représentées par des tableaux en C.

La variable `g_exemple` définie dans la fonction `main` du fichier compagnon représente le graphe ci-contre. Pour $s \in S$ on note $\mathcal{A}(s)$ l'ensemble des sommets accessibles à partir de s. Pour $s \in S$, le maximum des degrés d'un sommet accessible à partir de s est noté $d^*(s) = \max\{d^+(s') \mid s' \in \mathcal{A}(S)\}$. Par exemple, pour le graphe ci-contre, $\mathcal{A}(2) = \{2, 4, 6, 7, 8\}$ et $d^*(2) = 2$ car $d^+(4) = 2$. Dans cet exercice, on cherche à calculer $d^*(s)$ pour chaque sommet $s \in S$.
On représente un sous-ensemble $S' \subset S$ par un tableau de booléens de taille n contentant `true` à la case d'indice s' si $s' \in \mathcal{A}(S)$ et `false` sinon.



Q. 1 Écrire une fonction `int degre_max(graph* g, bool* partie)` qui calcule le degré maximal d'un sommet $s' \in S'$ dans un graphe $G = (S, A)$ pour une partie $S' \subset S$ représentée par S, c'est-à-dire qui calcule $\max\{d^+(s) \mid s' \in S'\}$.

Q. 2 Écrire une fonction `bool* accessibles(graph* g, int s)` qui prend en paramètre un graphe et un sommet s et qui renvoie un (pointeur sur) un tableau de booléens de taille n représentant $\mathcal{A}(s)$. Une fonction `nb_accessibles` qui utilise la fonction `accessible` et un test pour l'exemple ci-dessus sont donnés dans le fichier compagnon.

Q. 3 Écrire une fonction `int degre_etoile(graph*g, int s)` qui calcule $d^*(s)$ pour un graphe et un sommet passés en paramètres. Quelle est la complexité de l'approche proposée ?

Q. 4 Linéariser le graphe donné en exemple ci-dessus, c'est-à-dire représenter ses sommets sur une même ligne dans l'ordre donné par un tri topologique, tous les arcs allant de gauche à droite.

Q. 5 Dans cette question, on suppose que le graphe $G = (S, A)$ est acyclique. Décrire un algorithme permettant de calculer tous les $d^*(s)$ pour $s \in S$ en $O(|S| + |A|)$.

Q. 6 On ne suppose plus le graphe acyclique. Décrire un algorithme permettant de calculer tous les $d^*(s)$ pour $s \in S$ en $O(|S| + |A|)$.

Exercice 13 : Chemins simples sans issue *CCINP type B, 2023*

Consignes : Cet exercice est à traiter en OCaml. Le fichier chemins_simples.ml est fourni avec ce sujet. Il est à compléter en y implémentant les fonctions demandées.

L'objectif de cet exercice est de programmer une fonction générant la liste des chemins simples sans issue d'un graphe. On rappelle les définitions d'un graphe, d'un chemin, et on donne leur représentation en OCaml.

Un *graphe orienté* est un couple (V, E) où V est un ensemble fini (ensemble des sommets), E est un sous-ensemble de $V \times V$ où tout élément $(v_1, v_2) \in E$ vérifie $v_1 \neq v_2$ (ensemble des arcs). Étant donné un graphe $G = (V, E)$ un *chemin non vide* de G est une suite finie $s_0, \cdots, s_n$ de sommets de V avec $n \geq 0$ et vérifiant $\forall i \in \{0, \cdots, n-1\}, (s_i, s_{i+1}) \in E$. On dit que ce chemin est *simple* si $s_0, \cdots, s_n$ sont distincts deux à deux. On dit qu'il est *sans issue* si pour tout s_{n+1} sommet tel que $(s_n, s_{n+1}) \in E$, s_{n+1} appartient à $\{s_0, \cdots, s_n\}$. Dans la suite, les graphes considérés sont définis sur un ensemble de sommets de la forme $\{0, 1, \cdots, n-1\}$. Pour représenter un graphe en OCaml, on utilise le type suivant :

```
type graphe = int list array
```

qui correspond à un encodage par un tableau de listes d'adjacence. Par exemple, le graphe

$$G_1 = (\{0,1,2,3\}, \{(0,1),(0,3),(2,0),(2,1),(2,3),(3,1)\})$$

est représenté par le tableau `[|[1;3];[];[0;1;3];[1]|]`. L'ordre dans lequel sont écrits les éléments dans les listes importe peu. Par contre, l'emplacement des listes dans le tableau est important. Par exemple, `[|[];[0];[0;3;1];[1]|]` représente le graphe

$$G_2 = (\{0,1,2,3\}, \{(1,0),(2,0),(2,1),(2,3),(3,1)\})$$

On rappelle différentes fonctions pouvant être utiles :

- `List.filter : ('a -> bool) -> 'a list -> 'a list` où l'expression `List.filter f l` est la liste obtenue en gardant uniquement les éléments `x` de `l` vérifiant `f`.
- `List.iter :('a -> unit) -> 'a list -> unit` où `List.iter f l` correspond à `(f a0);(f a1); ...;(f an)` dans le cas où on a `l = a0::a1::...::an::[]`.
- `List.rev : 'a list -> 'a list` est une fonction qui renvoie le retourné d'une liste. Par exemple, `List.rev [3;1;2;2;4]` est égal à `[4;2;2;1;3]`.
- `Array.length : 'a array -> int` est une fonction qui renvoie la longueur d'un tableau.

Les questions de programmation sont à traiter dans le fichier chemins_simples.ml. L'utilisation d'autres fonctions de la bibliothèque que celles mentionnées sont à reprogrammer.

Q. 1 Écrire une fonction `est_sommet : graphe -> int -> bool` où `est_sommet g a` est égal à `true` si `a` est un sommet du graphe `g` et `false` sinon.

Q. 2 Écrire une fonction `appartient : 'a list -> 'a -> bool` où `appartient l x` est égal à `true` si `x` est un élément de `l` et `false` sinon.

Q. 3 Écrire une fonction `est_chemin : graphe -> int list -> bool`, où `est_chemin g l` est égal à `true` si `l` est un chemin de `g` et `false` sinon. On suppose que la liste vide représente le chemin vide, qui est bien un chemin et que les éléments de `l` sont bien des sommets du graphe `g`.

Q. 4 Compléter la fonction `est_chemin_simple_sans_issue : graphe -> int list -> bool`, où `est_chemin g l` est égal à `true` si `l` est un chemin simple sans issue de `g` et `false` sinon. On supposera que les éléments de `l` sont des sommets du graphe `g` et que le chemin vide n'est pas simple sans issue.

Q. 5 On cherche à écrire une fonction qui construit la liste des chemins simples sans issue d'un graphe. Pour cela, on procède à l'aide de parcours en profondeur et d'un algorithme de retour sur trace. Compléter le code de la fonction `genere_chemins_simples_sans_issue` présent dans le fichier `chemins_simples.ml` et qui permet de générer la liste des chemins simples sans issue d'un graphe.

Q. 6 Écrire des expressions donnant les listes des chemins simples pour les graphes G_1 et G_2.

Q. 7 Expliciter la complexité des fonctions `appartient` et `est_chemin_simple_sans_issue`.

Exercice 14 : Clôture transitive

CCINP type B, 2024

À toutes fins utiles, on rappelle que le module `Array` fournit une fonction `Array.make_matrix` de signature `int -> int -> 'a -> 'a array array`.

On considère des graphes orientés booléens, on représente ces graphes en OCaml par le type suivant. `type graphe = bool array array`. Si g est un tel graphe, l'arc `(u, v)` est présent dans ce graphe si et seulement si `g.(u).(v)` vaut `true`.

On définit la **clôture réflexive transitive** d'un graphe orienté $G = (S, A)$ comme étant le graphe orienté $G' = (S, A')$ où A' est l'ensemble des arcs (u, v) tels qu'il existe un chemin entre u et v dans G.

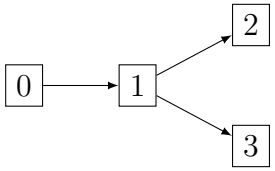


Figure 1 – Graphe G_{ex}

Q. 1 Représenter la clôture réflexive transitive de G_{ex}.

Q. 2 Définir en OCaml deux valeurs `g_ex` et `g_ex_ct` représentant respectivement le graphe G_{ex} et sa clôture réflexive transitive.

Q. 3 Coder en OCaml une fonction `somme` de signature `graphe -> graphe -> graphe` qui calcule la somme de deux matrices booléennes carrées de même taille. La somme de matrices booléennes attendue est semblable à la somme de matrices réelles, il suffit d'utiliser la loi $+$ sur $\mathbb{B}$ (les booléens) au lieu de l'addition réelle.

Q. 4 Coder en OCaml une fonction `produit` de signature `graphe -> graphe -> graphe` qui calcule le produit deux matrices booléennes carrées de même taille. Le produit de matrices booléennes attendu est semblable au produit de matrices réelles, il suffit d'utiliser les lois $+$ et $\cdot$ sur $\mathbb{B}$ au lieu de l'addition et de la multiplication réelles.

Q. 5 Coder en OCaml une fonction `puissance` de signature `graphe -> int -> graphe` calculant A^n pour $n \geqslant 1$.

Q. 6 Compléter le code de la fonction `puissance` pour traiter le cas $n \geqslant 0$. Justifier le résultat proposé.

Q. 7 On suppose que le graphe G représente une relation réflexive, c'est-à-dire que G contient tous les arcs de la forme (u, u). Montrer qu'alors il existe un chemin de u à v dans G si et seulement s'il existe un chemin de taille exactement $n - 1$ de u à v dans G.

Q. 8 À l'aide des fonctions et remarques précédentes, coder une fonction `cloture` de signature `graphe -> graphe` qui calcule la cloture réflexive transitive d'un graphe.

Q. 9 Quelle est la complexité temporelle de la fonction `cloture` ?
Quelle est sa complexité spatiale ?

Q. 10 Proposer un algorithme plus efficace permettant de calculer plus efficacement la clôture réflexive transitive et préciser sa complexité spatiale et sa complexité temporelle.

Troisième partie

Thème programmation dynamique - Corrigé

Exercice 15 : Sac à dos

CCINP type B, sujet 0

L'exercice suivant est à traiter dans le langage C.
On dispose de $n \geqslant 1$ objets $\{o_0, ..., o_{n-1}\}$ de valeurs respectives $(v_0, ..., v_{n-1}) \in \mathbb{N}^n$ et de poids respectifs $(p_0, ..., p_{n-1}) \in \mathbb{N}^n$. On souhaite transporter dans un sac de poids maximum p_{max} un sous-ensemble d'objets ayant la plus grande valeur possible. Formellement, on souhaite donc maximiser

$$\sum_{i=0}^{n-1} x_i v_i$$

sous les contraintes

$$(x_0, ..., x_{n-1}) \in \{0,1\}^n \text{ et } \sum_{i=0}^{n-1} x_i p_i \leqslant p_{max}$$

Intuitivement, la variable x_i vaut 1 si l'objet o_i est mis dans le sac et 0 sinon.
On propose d'utiliser un algorithme glouton dont le principe est de considérer les objets $o_0, o_1, ...o_{n-1}$ dans l'ordre et de choisir à l'étape i l'objet i (donc poser $x_i = 1$) si celui-ci rentre dans le sac avec la contrainte de poids maximal respectée et ne pas le choisir (donc poser $x_i = 0$) sinon. *On remarque que les valeurs $v_0, v_1, ..., v_{n-1}$ ne sont pas directement utilisées par cet algorithme, elle le seront lors du tri éventuel des objets.*

Q. 1 Proposer un type de données pour implémenter, pour n objets, leurs valeurs, leurs poids et les indicateurs $x_0, x_1, ...x_{n-1}$.

Q. 2 Écrire une fonction qui implémente la méthode gloutonne décrite ci-dessus à partir de n, p_{max} et $p_0, p_1, ..., p_{n-1}$ et qui permet de renvoyer les indicateurs $x_0, x_1, .., x_{n-1}$ pour le choix glouton.

Q. 3 Écrire un programme complet qui permet de lire sur l'entrée standard (au clavier par défaut) un entier $n \geqslant 1$, puis un entier naturel p_{max}, puis n entiers naturels correspondant aux valeurs $v_0, v_1, ..., v_{n-1}$, puis n entiers naturels correspondant aux poids $p_0, p_1, ..., p_{n-1}$ et qui affiche sur la sortie standard (l'écran par défaut) sous une forme de votre choix les indicateurs $x_0, x_1, ..., x_{n-1}$, la valeur de la solution $\sum_{i=0}^{n-1} x_i v_i$ et le poids utilisé $\sum_{i=0}^{n-1} x_i p_i$. On rappelle que le spécificateur de format pour lire ou écrire un entier est `%d`.

Q. 4 L'algorithme glouton ci-dessus donne-t-il toujours une solution optimale,
- $a)$ si on ne suppose rien sur l'ordre des objets a priori ?
- $b)$ si les objets sont triés par ordre de valeur décroissante ?
- $c)$ si les objets sont triés par ordre de poids croissant ?
- $d)$ si les objets sont triés par ordre décroissant des quotients v_i/p_i ?

Justifier chaque réponse à l'aide d'un contre-exemple ou d'une démonstration.

Q. 5 Le problème du sac à dos étudié dans cet exercice est un problème d'optimisation. Donner le problème de décision associé en utilisant une valeur v_{seuil}. Montrer que ce problème de décision est dans la classe NP.

Q. 6 Quelle serait la complexité d'une méthode qui examinerait tous les choix possibles pour retenir le meilleur ? Quelles stratégies d'élagage pourrait-on mettre en œuvre pour réduire l'espace de recherche ?

Exercice 16 : Récolte dynamique de fleurs *CCINP type B, 2023*

Consignes : Cet énoncé est accompagné d'un code compagnon en C ccinp_2023_prog_dyn_fleurs.c fournissant certaines des fonctions mentionnées dans l'énoncé : il est à compléter en y implémentant les fonctions demandées. La ligne de compilation `gcc -o main.exe *.c -lm` *vous permet de créer un exécutable main.exe à partir du ou des fichiers C fournis.*

Une petite fille se trouve en haut à gauche (case A) d'un champ modélisé par un tableau rectangulaire de taille $m \times n$ et doit se rendre dans la case B en bas à droite du champ où réside sa grand-mère (figure ci-dessous).

A	❀	❀	❀	❀	❀	❀
❀	❀	❀	❀	❀	❀	❀
❀	❀	❀	❀	❀	❀	❀
❀	❀	❀	❀	❀	❀	B

Chaque case du tableau, *y compris les cases A et B*, contient un certain nombre de fleurs. La petite fille, qui connaît depuis sa position initiale le nombre de fleurs de chaque case, doit se déplacer vers B de case en case, les seuls mouvements autorisés étant vers le bas ou vers la droite. À chaque déplacement, elle récolte les fleurs de la case atteinte. L'objectif pour elle est alors de faire le bouquet avec le plus de fleurs possible lors de son déplacement pour l'offrir à sa grand-mère.

Q. 1 On considère le champ suivant :

0 (A)	1	2	3
1	2	3	4
2	3	4	0
3	4	0	1 (B)

Donner le nombre maximal de fleurs cueillies par la petite fille.

Q. 2 On note $n(i,j)$ le nombre maximum de fleurs que la petite fille peut récolter en se déplaçant de A à la case (i,j). Exprimer $n(i,j)$ en fonction de $n(i-1,j)$ et $n(i,j-1)$. En déduire une fonction récursive de prototype `int recolte(int champ[m][n], int i, int j)` qui, étant données les coordonnées `i`, `j` d'une case, calcule le nombre maximum de fleurs cueillies par la petite fille de A à la case (`i`, `j`).

Q. 3 On suppose $m = n = 4$ et on effectue donc un appel à `recolte(champ,3,3)` pour résoudre le problème posé. Donner le nombre de fois où votre fonction calcule le nombre de fleurs maximum cueillies dans la case $(1,1)$ (deuxième case de la diagonale).

D'une manière générale, le nombre d'appels à la fonction récursive est important. On a donc intérêt à transformer l'algorithme récursif en algorithme dynamique. On propose de déclarer dans le programme principal un tableau `fleurs` dont la case (i,j) est destinée à contenir la récolte maximale que la petite fille peut obtenir en cheminant de A vers la case (i,j).

Q. 4 Dans quel ordre remplir le tableau `fleurs` de sorte à éviter de recalculer une valeur ?

Q. 5 Écrire une fonction de prototype `int recolte_iterative(int champ[m][n], int i, int j, int fleurs[m][n])` qui calcule, stocke dans `fleurs[i][j]` et retourne la cueillette maximale obtenue en parcourant le champ de A à la case (`i`, `j`).

La fonction `recolte_iterative` permet de déterminer la cueillette maximale en (`i`, `j`) mais ne précise pas le chemin parcouru pour l'obtenir.

Q. 6 Écrire la fonction de prototype `void deplacements(int fleurs[m][n], int i, int j)` qui affiche la suite des déplacements effectués par la petite fille sur un chemin permettant de récolter le nombre maximum de fleurs entre (0,0) et (`i`, `j`).

Q. 7 Insérer un appel de `deplacements` dans la fonction `recolte_iterative` pour afficher le chemin parcouru.

Exercice 17 : Justification de texte en OCaml

CCINP type B, 2023

Consignes : Ce sujet est à traiter en OCaml en complétant le fichier compagnon qui vous est fourni, nommé `ccinp_2023_justification.ml`. Outre des définitions de types et de valeurs, ce fichier contient un jeu de tests pour chaque fonction à coder.

On s'intéresse à l'affichage d'un texte justifié sur plusieurs lignes. La longueur des lignes, donnée en nombre de caractères, est la même pour toutes les lignes. Sur chaque ligne, on cherche à occuper toute la longueur en ajoutant des espaces entre les mots de manière équilibrée, sachant qu'il y a toujours au moins un espace entre deux mots.

```
Ceci  est  un  exemple
simple      de      texte
a                   afficher
```

```
Ceci      est      un      exemple
simple          de          texte
a                           afficher
```

FIGURE 2 – Exemple de texte justifié sur 24 ou 31 caractères

On représente le texte à afficher en OCaml sous la forme d'un tableau de mots, d'où le type suivant.

```
type texte = string array
```

Le fichier compagnon contient plusieurs exemples de tels textes, notamment le texte affiché ci-dessus est représenté par `t1`.

```
let t1 = [|"Ceci"; "est"; "un"; "exemple"; "simple"; "de"; "texte"; "a";
↪  "afficher"|]
```

On a volontairement omis les accents pour des raisons de simplicité (les lettres accentuées sont codées sur deux caractères, mais s'affichent sur un seul).

Q. 1 Écrire en OCaml une fonction `largeur_suffisante : texte -> int -> bool` qui teste si chacun des mots du texte passé en premier paramètre tient sur le nombre de caractères passé en second paramètre.

La répartition des mots du texte sur les différentes lignes est représentée en OCaml par une liste de couples `(i, j)` indiquant l'intervalle d'indice des mots de chaque ligne, d'où le type suivant.

```
type disposition = (int * int) list
```

Par exemple le texte `t1` tel qu'il est affiché ci-dessus (que ce soit sur 24 ou 31 caractères) correspond à la disposition `d1_1`. La fonction `affichage_justifie` fournie dans le fichier compagnon réalise de tels affichages pour un texte, un nombre de caractères et une disposition donnés. Cette fonction nécessite d'avoir traité la question **Q. 2**.

```
let d1_1 = [(0, 3); (4, 6); (7, 8)]
```

Le but de cet exercice est de calculer, pour un texte t et une largeur l fixés, la meilleure disposition. On cherche une disposition de coût minimal où le coût d'une disposition valide D est défini comme suit.

$$c(D) = \sum_{(i,j)\in D} e(i,j)^2$$

où $e(i,j)$ désigne le nombre d'espaces à ajouter aux mots de t de l'intervalle $[\![i,j]\!]$ pour remplir une ligne de longueur l. Par exemple, $e_{0,3} = 8$ pour le texte `t1` et une largeur $l = 24$, comme on peut le voir sur la première ligne de la figure de gauche de la figure 2.

Q. 2 Compléter la fonction `nb_espaces (t: texte) (lg: int) (i: int) (j: int) : int` qui calcule le nombre d'espaces à ajouter aux mots de `t` d'indices compris dans l'intervalle $[\![$`i, j`$]\!]$ pour remplir une ligne de longueur `lg` si cela est possible♣. Dans le cas contraire cette fonction renvoie `-1`.

Afin de trouver une disposition de coût minimal, on utilise une approche de programmation dynamique. On cherche à calculer, pour chaque $i \in [\![0,n]\!]$ où n est le nombre de mots du texte, le coût minimal d'une disposition pour les mots de l'intervalle $[\![i,n[\![$. On note $C(i)$ cette quantité.

Q. 3 Justifier que la quantité C vérifie la relation de récurrence suivante.

$$C(n) = 0$$
$$\forall i \in [\![0,n[\![, C(i) = \min \left\{ e(i,j)^2 + C(j+1) \;\middle|\; \begin{array}{l} j \in [\![i,n[\![\text{ tel que les mots du texte de} \\ \text{l'intervalle } [\![i,j]\!] \text{ tiennent sur une ligne} \end{array} \right\}$$

Q. 4 Compléter le code de la fonction `calcule_c` qui calcule le tableau des valeurs de C pour chaque $i \in [\![0,n]\!]$.

Q. 5 Compléter le code de la fonction `calcule_best_j` qui calcule, pour chaque $i \in [\![0,n]\!]$ un indice j réalisant le minimum dans la relation de récurrence de la question **Q. 3**.

Q. 6 Compléter enfin la fonction `dispo_min` une disposition de coût minimal pour le texte passé en premier paramètre sur la longueur passée en second paramètre.

Q. 7 Quelle est la complexité de la fonction `dispo_min` en fonction de la taille du texte ?

♣. On rappelle qu'il faut au moins un espace entre deux mots.