Licence d'informatique
Algorithmique – 3I003
Fascicule de TD

année 2018–2019

Nawal Benabbou
Anne-Elisabeth Falq
Pierre Fouilhoux
Fanny Pascual
Maryse Pelletier
Olivier Spanjaard
Lionel Tabourier

Algorithmique – 3I003

Preuves et analyse de complexité d'algorithmes

Notations Asymptotiques

Exercice 1 – Notations de Landau (exercice de base)

Q 1.1 Démontrer que :

1. $n^2 \in O(10^{-4}n^3)$
2. $25n^4 + 13n^2 \in O(n^4)$
3. $2^{n+100} \in O(2^n)$
4. $n^5 \in \Omega(10n^3)$
5. $20n + 5n^2 \in \Theta(n^2)$

Q 1.2 Montrer que, si a est un réel positif, alors $f(n) = 1 + a + a^2 + \ldots + a^n$ est en :

(*a*) $\Theta(1)$ si $a < 1$.

(*b*) $\Theta(n)$ si $a = 1$.

(*c*) $\Theta(a^n)$ si $a > 1$.

Exercice 2 – Complexité au pire cas (exercice de base)

Le but de cet exercice est de tester votre compréhension de la notion de complexité dans le pire des cas.

Q 2.1 Si je prouve que la complexité dans le pire des cas d'un algorithme est en $O(n^2)$, est-il possible qu'il soit en $O(n)$ sur *toutes* les données ?

Q 2.2 Si je prouve que la complexité dans le pire des cas d'un algorithme est en $\Theta(n^2)$, est-il possible qu'il soit en $O(n)$ sur *certaines* données ?

Q 2.3 Si je prouve que la complexité dans le pire des cas d'un algorithme est en $\Theta(n^2)$, est-il possible qu'elle soit en $O(n)$ sur *toutes* les données ?

Exercice 3 – Échelle de croissance (exercice de base)

Pour classer des fonctions, on utilisera les notations suivantes :

$$\begin{array}{ll} f(n) < g(n) & \text{si } f(n) \in O(g(n)) \text{ et } f(n) \notin \Theta(g(n)) \\ f(n) \equiv g(n) & \text{si } f(n) \in \Theta(g(n)) \end{array}$$

Sur une échelle croissante, classer les fonctions suivantes selon leur comportement asymptotique :

$$\begin{array}{llll} f_1(n) = 2\,n & f_2(n) = 2^n & f_3(n) = \log(n) & f_4(n) = \frac{n^3}{3} \\ f_5(n) = n! & f_6(n) = [\log(n)]^2 & f_7(n) = n^n & f_8(n) = n^2 \\ f_9(n) = n + \log(n) & f_{10}(n) = \sqrt{n} & f_{11}(n) = \log(n^2) & f_{12}(n) = e^n \\ f_{13}(n) = n & f_{14}(n) = \sqrt{\log(n)} & f_{15}(n) = 2^{\log_2(n)} & f_{16}(n) = n\,\log(n) \end{array}$$

Preuves et complexité d'algorithmes

Exercice 4 – Complexité en fonction de deux paramètres (exercice de base)

Déterminer la complexité des algorithmes suivants (par rapport au nombre d'itérations effectuées), où m et n sont deux entiers positifs.

Algorithme : A

$i \leftarrow 1$; $j \leftarrow 1$;
tant que $(i \leq m)$ **et** $(j \leq n)$ **faire**
  $i \leftarrow i + 1$;
  $j \leftarrow j + 1$;

Algorithme : B

$i \leftarrow 1$; $j \leftarrow 1$;
tant que $(i \leq m)$ **ou** $(j \leq n)$ **faire**
  $i \leftarrow i + 1$;
  $j \leftarrow j + 1$;

Algorithme : C

$i \leftarrow 1$; $j \leftarrow 1$;
tant que $(j \leq n)$ **faire**
  si $i \leq m$ **alors** $i \leftarrow i + 1$;
  sinon $j \leftarrow j + 1$;

Algorithme : D

$i \leftarrow 1$; $j \leftarrow 1$;
tant que $(j \leq n)$ **faire**
  si $i \leq m$ **alors** $i \leftarrow i + 1$;
  sinon $j \leftarrow j + 1$; $i \leftarrow 1$;

Exercice 5 – Division Euclidienne (exercice de base)

On considère l'algorithme DivisionEuclidienne (page 6), qui prend en arguments deux entiers a et b et retourne l'entier $\lfloor \frac{a}{b} \rfloor$.

Q 5.1 Exécutez cet algorithme avec $a = 14$ et $b = 3$. Vous indiquerez à chaque itération les valeurs de R et de Q.

Q 5.2 Montrez que cet algorithme se termine et est valide.

Algorithme : DivisionEuclidienne
Entrées : a : entier, b : entier
$R \leftarrow a$; $Q \leftarrow 0$;
tant que $R \geq b$ **faire**
  $R \leftarrow R - b$;
  $Q \leftarrow Q + 1$;
retourner Q

Exercice 6 – Factorielle (exercice de base)

Q 6.1 Ecrire un algorithme récursif qui prend en argument un entier naturel n et retourne $n!$

Q 6.2 Prouvez que votre algorithme se termine et est valide.

Exercice 7 – Fonctions F et G mystères (exercice d'entraînement)

Q 7.1 Soit la fonction récursive F suivante, qui prend en argument un entier positif n :

Algorithme : Fonction $F(n)$
si $n = 0$ **alors**
  retourner 2;
sinon
  retourner $F(n-1) \times F(n-1)$;

Que calcule cette fonction ? Le prouver.

Q 7.2 Déterminer la complexité de la fonction F. Comment améliorer cette complexité ?

Q 7.3 Soit la fonction itérative G suivante :

Algorithme : Fonction $G(n)$
$R \leftarrow 2$;
pour i *variant de* 1 *à* n **faire**
  $R \leftarrow R * R$;
retourner R;

Que calcule cette fonction ? Le prouver.

Q 7.4 Déterminer la complexité de la fonction G.

Soient a et b deux nombres entiers positifs. On souhaite effectuer la multiplication de a par b. Les deux exercices suivants ont pour but d'étudier et de comparer deux algorithmes résolvant ce problème.

Exercice 8 – Multiplication intuitive (exercice d'entraînement)

On considère l'algorithme suivant :

Algorithme : Multiplie1(entier a, entier b)
$p \leftarrow 0$;
$m \leftarrow 0$;
tant que $m < a$ **faire**
  $p \leftarrow p + b$;
  $m \leftarrow m + 1$;
retourner p;

Q 8.1 Prouvez que l'algorithme se termine et est valide.

Q 8.2 Quelle est la complexité de cet algorithme ?

Exercice 9 – Multiplication russe (exercice d'approfondissement)

On peut améliorer l'algorithme précédent en multipliant a par deux et en divisant b par deux chaque fois que sa valeur est paire. Les opérations de multiplication et de division par deux sont simples pour un codage binaire puisqu'elles consistent à décaler un bit vers la gauche ou vers la droite. L'algorithme est le suivant :

Algorithme : Multiplie2(entier a, entier b)
$prod \leftarrow 0$;
$x \leftarrow a$;
$y \leftarrow b$;
tant que $y \neq 0$ **faire**
  si y *est impair* **alors**
    $prod \leftarrow prod + x$;
    $y \leftarrow y - 1$
  $x \leftarrow 2 \times x$;
  $y \leftarrow y/2$;
retourner $prod$;

Q 9.1 Donner la trace de l'algorithme (valeurs de x, y et $prod$) quand $a = 4$ et $b = 11$.

Q 9.2 Démontrer que la propriété suivante est un invariant de boucle : *À la fin de la i-ème itération,* $prod + x \times y = a \times b$. Autrement dit : pour toute itération i, on a $prod_i + x_i \times y_i = a \times b$, où x_i, y_i et $prod_i$ sont les valeurs de x, y et $prod$ à la fin de l'itération i. En déduire la correction de l'algorithme.

Q 9.3 Quelle est la complexité de cet algorithme ?

Exercice 10 – Les tours de Hanoi (exercice d'approfondissement)

Petit divertissement mathématique mis au point par Édouard Lucas en 1883 : au départ, on dispose de 3 piquets. Le premier porte n disques de tailles toutes différentes et empilés du plus grand (en bas) au plus petit (en haut). Le problème consiste à déplacer tous ces disques jusqu'au troisième piquet sachant que l'on respecte les règles suivantes :

— on ne déplace qu'un disque à la fois ;

— on ne peut déplacer qu'un disque se trouvant en haut d'une pile ;

— aucun disque ne doit être empilé sur un disque de diamètre inférieur.

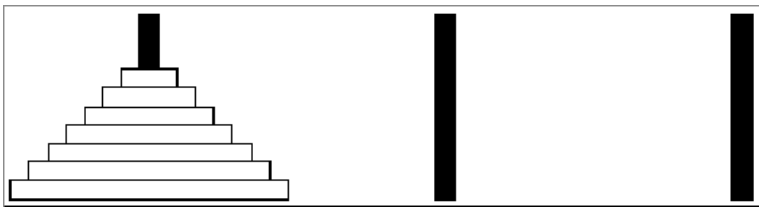


FIGURE 1 – état initial

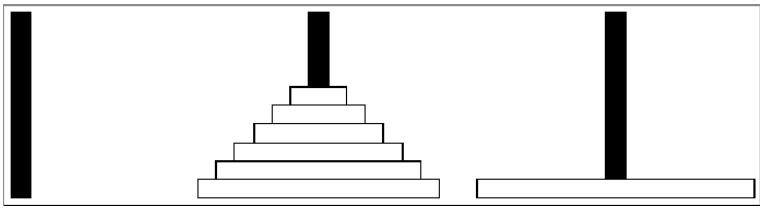


FIGURE 2 – état intermédiaire

Pour résoudre ce problème, l'algorithme récursif *Deplacer* est proposé.

Algorithme : $Deplacer(nombre, piquet1, piquet3, piquet2)$
si $nombre > 0$ **alors**
  $Deplacer(nombre - 1, piquet1, piquet2, piquet3)$;
  $Bouger\_un\_disque(piquet1, piquet3)$;
  $Deplacer(nombre - 1, piquet2, piquet3, piquet1)$;

où *nombre* : nombre de disques utilisés,
et $Bouger\_un\_disque(origine, destination)$: déplace le disque se trouvant en haut de la pile du piquet *origine* vers le piquet *destination*.

Q 10.1 Prouver la correction de l'algorithme récursif proposé. (Indication : penser à vérifier que chaque déplacement de disque respecte les règles du jeu.)

Q 10.2 On appellera $D(n)$ le nombre de déplacements de disques effectués par l'algorithme pour n disques. Donner la complexité de cet algorithme en nombre de déplacements de disques et le prouver.

En supposant qu'un être humain met une seconde à déplacer un disque, combien de temps lui faudra-t-il pour résoudre ce casse-tête pour une tour de Hanoi comportant 36 disques ?

Q 10.3 Montrer que tout algorithme résolvant le problème des tours de Hanoï nécessite au minimum $2^n - 1$ déplacements de disques.

Exercice 11 – Calcul de l'écart minimum dans un tableau (exercice d'approfondissement)

Soit $T[1..n]$ un tableau d'entiers. Si $1 \leq i < j \leq n$, l'écart $e(i, j)$ entre les cellules i et j est le nombre d'éléments entre ces deux cellules : il est donc défini par $j - i - 1$. Le but de l'exercice est la détermination du *plus petit écart* noté $emin(T)$ entre 2 cellules qui *contiennent le même entier*. Si tous les éléments du tableau sont différents, alors par convention $emin(T) = +\infty$.

Q 11.1 Soit T le tableau d'entiers suivant : | 1 | 2 | 4 | 5 | 2 | 7 | 8 | 1 | 5 | 3 | .
Quelle est la valeur de $emin(T)$?

On considère l'algorithme suivant, qui prend en argument un tableau T de n éléments, et retourne un nombre entier représentant l'écart minimum dans le tableau T.

```
Algorithme : TrouveEcartMin(T)
EcartMin ← +∞;
pour i variant de 1 à (n − 1) faire
    j ← i + 1;
    tant que j ≤ n et T[j] ≠ T[i] faire
        j ← j + 1;
    si j ≠ n + 1 et (EcartMin > j − i − 1) alors
        EcartMin ← j − i − 1;
retourner EcartMin;
```

Q 11.2 Exprimer (sans le démontrer) un invariant de boucle $I(k)$ vérifié à la fin de l'itération k de la boucle Pour de l'algorithme TrouveEcartMin. En déduire que l'algorithme TrouveEcartMin est correct.

Q 11.3 Quelle est la complexité temporelle pire-cas de l'algorithme TrouveEcartMin ? Justifiez votre réponse.

Q 11.4 La complexité pire-cas que vous avez trouvée dans la question précédente peut-elle être atteinte, ou est-elle seulement une borne supérieure de la complexité pire-cas de l'algorithme ? Justifiez votre réponse. Quelle est la complexité temporelle pire-cas de votre algorithme ?

Q 11.5 On suppose maintenant que les valeurs du tableau T sont n entiers dont les valeurs sont comprises entre 1 et k, avec $k \leq n$. Expliquer en quelques phrases comment avoir un algorithme de meilleure complexité temporelle pire-cas pour ce problème.

Algorithmique – 3I003

Programmation récursive

Programmation récursive

Exercice 1 – Analyse d'une récurrence (exercice de base)

Soit un algorithme dont la complexité $T(n)$ est donnée par la relation de récurrence :

$$\begin{aligned} T(1) &= 1, \\ T(n) &= 3\,T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n^2. \end{aligned}$$

Q 1.1 En supposant que l'entier n est une puissance de 2, calculer $T(n)$ en résolvant la récurrence.

Q 1.2 En utilisant le théorème maître, donner un ordre de grandeur asymptotique pour $T(n)$.

Exercice 2 – Application du théorème maître (exercice de base)

Le but de cet exercice est d'utiliser le théorème maître pour donner des ordres de grandeur asymptotique pour des fonctions définies par récurrence.

En utilisant le théorème maître, donner un ordre de grandeur asymptotique pour $T(n)$ où $\frac{n}{2}$ peut valoir $\lceil \frac{n}{2} \rceil$ ou $\lfloor \frac{n}{2} \rfloor$:

Q 2.1 $T(1) = 1, \quad T(n) = 2\,T\left(\frac{n}{2}\right) + n^2$;

Q 2.2 $T(1) = 0, \quad T(n) = 2\,T\left(\frac{n}{2}\right) + n$;

Q 2.3 $T(1) = 1, \quad T(n) = 2\,T\left(\frac{n}{2}\right) + \sqrt{n}$;

Q 2.4 $T(1) = 0, \quad T(n) = T\left(\frac{n}{2}\right) + c \,\text{avec}\, c > 0$

Exercice 3 – Lequel est le plus rapide (exercice de base)

Le but de cet exercice est de choisir l'algorithme de type « diviser pour régner » le plus rapide pour un même problème.

Q 3.1 Pour résoudre un problème manipulant un nombre important de données, on propose deux algorithmes :

1. un algorithme qui résout un problème de taille n en le divisant en 2 sous-problèmes de taille $n/2$ et qui combine les solutions en temps quadratique,
2. un algorithme qui résout un problème de taille n en le divisant en 4 sous-problèmes de taille $n/2$ et qui combine les solutions en $\sqrt{n}$ opérations.

Lequel faut-il choisir ?

Q 3.2 Même question avec les algorithmes suivants :

1. un algorithme qui résout un problème de taille n en le réduisant à un sous-problème de taille $n/2$ et qui combine les solutions en temps $O(\sqrt{n})$,
2. un algorithme qui résout un problème de taille n en le divisant en 2 sous-problèmes de taille $n/2$ et qui combine les solutions en temps constant.

Exercice 4 – Arbre des appels récursifs (exercice de base)

Considérons un algorithme dont le nombre d'opérations sur une entrée de taille n est donné par la relation de récurrence suivante :

$$T(1) = 0 \quad \text{et} \quad T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2\,n}{3}\right) + n.$$

Q 4.1 Construire un arbre représentant les appels récursifs de l'algorithme. En déduire la complexité de $T(n)$.

Q 4.2 Peut-on retrouver le résultat de la question précédente à l'aide du théorème maître ?

Exercice 5 – Recherche dans un tableau (exercice de base)

Proposer un algorithme de recherche d'un élément dans un tableau *trié dans l'ordre croissant* qui renvoie l'indice de l'élément lorsqu'il est trouvé ou la valeur -1 sinon, et indiquer sa complexité au pire cas en fonction de la taille du tableau :

Q 5.1 en utilisant une recherche séquentielle,

Q 5.2 en utilisant le principe « diviser pour régner ».

Exercice 6 – Pièces d'or (exercice d'entraînement)

On dispose de n pièces d'or toutes de même poids, sauf une défectueuse qui est plus légère. Le problème consiste à retrouver la pièce défectueuse en utilisant une balance de type Roberval (balance à deux plateaux qui permet de dire uniquement s'il y a un plateau plus lourd que l'autre).

On part d'un ensemble de $n = 3^p$ pièces, dans lequel on cherche à déterminer la pièce la plus légère.

Q 6.1 On suppose que les n pièces sont réparties en trois ensembles de même taille. Montrer qu'à l'aide d'une balance de type Roberval, on peut décider dans quel ensemble se trouve la pièce la plus légère en une seule pesée.

Q 6.2 Proposer un algorithme de type « diviser pour régner » pour résoudre ce problème.

Q 6.3 On note $T(n)$ le nombre de pesées nécessaires pour trouver la pièce défectueuse dans un ensemble de $n = 3^p$ pièces.

1. Donner l'expression de la récurrence vérifiée par $T(n)$.
2. Résoudre cette récurrence d'une part directement et d'autre part en utilisant le théorème maître.

Exercice 7 – Majoritaire (exercice d'entraînement)

Un élément x est majoritaire dans un ensemble E de n éléments si E contient strictement plus de $n/2$ occurrences de x. La seule opération dont nous disposons est la comparaison (égalité ou non) de deux éléments.

Le but de l'exercice est d'étudier le problème suivant : étant donné un ensemble E, représenté par un tableau, existe-t-il un élément majoritaire, et si oui quel est-il ?

Q 7.1 Voilà une idée pour un algorithme naïf.

1. Écrire un algorithme NombreOccurrences qui, étant donné un tableau E, un élément x et deux indices i et j, calcule le nombre d'occurrences de x entre $E[i]$ et $E[j]$. Quelle est sa complexité (en nombre de comparaisons) ?

2. Écrire un algorithme Majoritaire qui vérifie si un tableau E contient un élément majoritaire, et si oui le retourne. Quelle est sa complexité ?

Q 7.2 Décrire le principe d'un algorithme de type « diviser pour régner » pour résoudre ce problème.

Q 7.3 Écrire un algorithme reposant sur le principe décrit à la question précédente. Analyser sa complexité.

Q 7.4 Donner le principe d'un algorithme permettant de résoudre le problème en $O(n)$ si on suppose que les éléments de E sont à valeurs dans $\{1, \cdots, k\}$, k étant une constante.

Exercice 8 – Algorithme de Stooge (exercice d'entraînement)

On considère l'algorithme récursif ci-dessous, qui trie le sous-tableau $A[i \dots j]$.

```
Algorithme : Stooge(tableau A, entier i, entier j)
si A[i] > A[j] alors
    échanger le contenu de A[i] et A[j];
si j − i > 1 alors
    k ← (j − i + 1)/3 ; // quotient de la division entière
    Stooge(A, i, j − k) ; // premier 2/3 du tableau
    Stooge(A, i + k, j) ; // dernier 2/3 du tableau
    Stooge(A, i, j − k) ; // premier 2/3 à nouveau
retourner A;
```

L'appel initial est *Stooge(A,1,n)*, où A est un tableau indicé de 1 à n. Pour simplifier, on suppose dans tout l'exercice que n est une puissance de 3 (autrement dit, $n = 3^p$ pour un entier $p \geq 0$), ce qui garantit que le reste de la division entière est nul dans tous les appels récursifs.

Q 8.1 Montrer par récurrence forte que *Stooge(A,1,n)* trie le tableau A en ordre croissant.

Q 8.2 Soit $T(n)$ le nombre d'échanges au pire réalisé par *Stooge* sur un tableau de taille n. Ecrire la relation de récurrence permettant de calculer $T(n)$ pour $n > 2$. En déduire la complexité de l'algorithme à l'aide du théorème maître.

Q 8.3 Est-ce que *Stooge* est un bon algorithme de tri ?

Exercice 9 – Le pic (exercice d'entraînement)

Soit A un tableau contenant n entiers distincts ($n \geqslant 3$). La suite définie par $A[1], A[2], \dots, A[n]$ est unimodale, c'est-à-dire qu'elle est croissante (sur au moins 2 éléments) puis décroissante (sur au moins 2 éléments). Par exemple, $1, 3, 5, 4, 2$ est unimodale alors que $1, 3, 2, 5, 4$ ne l'est pas. On souhaite représenter graphiquement les éléments de la suite en représentant sur l'axe des abscisses l'indice i et sur l'axe des ordonnées la valeur $A[i]$ correspondant. Afin de définir l'échelle des ordonnées, il est nécessaire avant de commencer le tracé de connaître la valeur maximale du tableau A.

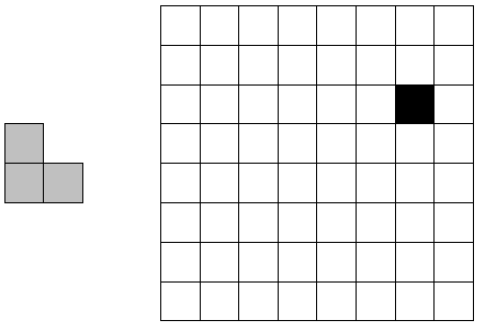
Q 9.1 Soit A un tableau unimodal. Résoudre le problème de la recherche du maximum de A, à l'aide

d'une approche de type « diviser pour régner ». Donner le pseudo-code.

Q 9.2 Étudier la complexité de votre algorithme.

Exercice 10 – Triominos (exercice d'entraînement)

Un *triomino* est une tuile en forme de L formée de trois carrés 1×1. Le problème est de couvrir par des triominos un échiquier "troué" $2^n \times 2^n$. Les triominos peuvent être orientés de façon arbitraire, mais ils doivent couvrir toutes les cases de l'échiquier (exceptée la case manquante) sans chevauchement.



Q 10.1 Proposer une méthode de type « diviser pour régner » pour résoudre ce problème.

Q 10.2 Prouver la validité de la méthode proposée.

Q 10.3 On considère maintenant un échiquier carré comportant un nombre pair de cases, privé de deux angles diagonalement opposés, qu'on cherche à couvrir de dominos sans chevauchement. Peut-on trouver un algorithme ?

Exercice 11 – Produit matriciel (exercice d'approfondissement)

Le but de cet exercice est de réaliser un produit de deux matrices carrées X et Y de dimension $n \times n$ en utilisant l'approche *diviser pour régner*. Un produit de matrices est particulièrement facile à diviser en sous-problèmes car il peut être réalisé *par blocs*. Pour voir ce que cela signifie, partitionnons X en quatre blocs $n/2 \times n/2$, ainsi que Y :

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, \quad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}.$$

Le produit XY peut être réalisé par blocs de la façon suivante :

$$XY = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

Q 11.1 On suppose que l'addition de deux scalaires se fait en $\Theta(1)$. Quelle est la complexité (en fonction de n) de l'opération de fusion des réponses aux sous-problèmes ?

Q 11.2 Soit $T(n)$ le nombre d'additions de scalaires à effectuer pour réaliser un produit de matrices

carrées $n \times n$. Ecrire l'équation de récurrence vérifiée par $T(n)$. En déduire la complexité de la méthode en fonction de n.

On suppose désormais qu'on divise le problème en réalisant les calculs suivants :

$$XY = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$$

où

$$\begin{array}{lll} P_1 = A(F-H) & & P_5 = (A+D)(E+H) \\ P_2 = (A+B)H & & P_6 = (B-D)(G+H) \\ P_3 = (C+D)E & & P_7 = (A-C)(E+F) \\ P_4 = D(G-E) & & \end{array}$$

Q 11.3 On vérifie facilement que la complexité de l'opération de fusion reste inchangée. Ecrire l'équation de récurrence vérifiée par $T(n)$ pour cette nouvelle méthode de produit matriciel. En déduire la complexité.

Exercice 12 – Sous-tableau maximum (exercice d'approfondissement)

Dans cet exercice, on s'intéresse au *problème du sous-tableau maximum*. Etant donné un tableau $A[1, \ldots, n]$, le problème du sous-tableau maximum de A est de déterminer :

$$\text{stm}(A) = \max\{\sum_{k=i}^{j} A[k] : 1 \leq i \leq j \leq n\}$$

Par exemple, on a $\text{stm}([-3, 4, 5, -1, 2, 3, -6, 4]) = 13$, valeur que l'on obtient pour le sous-tableau $[4, 5, -1, 2, 3]$. Dans un premier temps, on considère l'algorithme de résolution ci-dessous :

```
max = max{A[1], ..., A[n]};
pour i = 1 ··· n faire
    pour j = i ··· n faire
        temp ← 0;
        pour k = i ··· j faire
            temp ← temp + A[k];
        si temp > max alors
            max ← temp;
retourner max;
```

Q 12.1 Quelle est la complexité de cet algorithme ?

Q 12.2 Proposer une variante en $\Theta(n^2)$ de cet algorithme.

Dans la suite de l'exercice, on s'intéresse à des algorithmes de type « diviser pour régner » pour ce problème. Soit $m = \lfloor (n+1)/2 \rfloor$, $A_1 = A[1, \ldots, m]$ et $A_2 = A[m+1, \ldots, n]$. Le cas de base est le cas d'un tableau de taille 1, qu'on sait bien sûr résoudre en temps constant. Soit $\text{stm}_1 = \text{stm}(A_1)$,

$\text{stm}_2 = \text{stm}(A_2)$ et stm_3 la valeur maximum d'un sous-tableau qui contient à la fois $A[m]$ et $A[m+1]$.

Q 12.3 Comment calculer stm_3 efficacement ? Quelle est la complexité en fonction de n ?

Q 12.4 Dans le cadre d'une méthode « diviser pour régner », supposons que l'on a calculé stm_1 et stm_2 (par des appels récursifs). En vous aidant de la réponse à la question précédente, donner le principe d'une opération de fusion des résultats des deux appels récursifs, et indiquer la complexité de cette opération.

Q 12.5 Soit $T(n)$ la complexité de la méthode « diviser pour régner » pour un tableau de taille n. Donner l'équation de récurrence satisfaite par $T(n)$. A l'aide du théorème maître, en déduire la complexité de la méthode.

On s'intéresse désormais à une autre approche, toujours de type « diviser pour régner », afin de réduire encore la complexité de l'algorithme. Pour ce faire, on définit :

$$\begin{array}{rcl} \text{pref}(A) & = & \max\{\sum_{k=1}^{i} A[k] : 1 \leq i \leq n\}, \\ \text{suff}(A) & = & \max\{\sum_{k=i}^{n} A[k] : 1 \leq i \leq n\}, \\ \text{tota}(A) & = & \sum_{k=1}^{n} A[k], \end{array}$$

Autrement dit, $\text{pref}(A)$ (resp. $\text{suff}(A)$) est la valeur maximum d'un sous-tableau préfixe (resp. suffixe) de A, et $\text{tota}(A)$ est la somme des cases de A.

Q 12.6 Supposons qu'on vise à déterminer non plus simplement $\text{stm}(A)$, mais le triplet $\text{stm}(A)$, $\text{pref}(A)$, $\text{suff}(A)$. Soit $(\text{stm}_1, \text{pref}_1, \text{suff}_1)$ et $(\text{stm}_2, \text{pref}_2, \text{suff}_2)$ les triplets retournés par les appels récursifs sur A_1 et A_2. Expliquer pourquoi ce serait faux de faire la fusion en retournant $(\max\{\text{suff}_1 + \text{pref}_2, \text{stm}_1, \text{stm}_2\}, \text{pref}_1, \text{suff}_2)$ pour A.

Q 12.7 Supposons maintenant qu'on vise à déterminer le quadruplet $(\text{stm}(A), \text{pref}(A), \text{suff}(A), \text{tota}(A))$. Soit $(\text{stm}_1, \text{pref}_1, \text{suff}_1, \text{tota}_1)$ et $(\text{stm}_2, \text{pref}_2, \text{suff}_2, \text{tota}_2)$ les quadruplets retournés par les appels récursifs sur A_1 et A_2. Donner la formule permettant de déterminer le quadruplet pour A à partir de stm_1, pref_1, suff_1, tota_1, stm_2, pref_2, suff_2, tota_2. Quelle est la complexité de cette opération de fusion ?

Q 12.8 Soit $T(n)$ la complexité de la méthode « diviser pour régner » correspondante pour un tableau de taille n. Donner l'équation de récurrence satisfaite par $T(n)$. A l'aide du théorème maître, en déduire la complexité de cette nouvelle méthode.

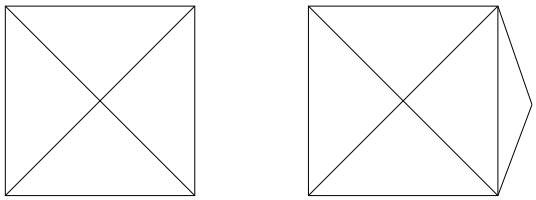
Algorithmique – 3I003

Introduction aux graphes et parcours

Introduction aux graphes

Exercice 1 – Casse-têtes (exercice de base)

Q 1.1 Est-il possible de tracer les figures suivantes sans lever le crayon et sans passer deux fois sur le même trait ?



Q 1.2 Est-il possible de dessiner, dans le plan, 9 segments de telle manière que chaque segment coupe exactement 3 autres segments ?

Exercice 2 – Réseaux de capteurs (exercice d'entraînement)

On dispose de n capteurs sans fil représentés par n points dans un plan. Deux capteurs x et y peuvent communiquer si la distance qui les sépare est d'au plus 500 mètres ou s'il existe une chaîne de capteurs entre x et y tel que la distance entre deux capteurs adjacents de cette chaîne est d'au plus 500 mètres.

On impose la contrainte suivante : chaque capteur doit être à une distance d'au plus 500 mètres d'au moins $\frac{n}{2}$ autres capteurs. On suppose que n est pair.

Cette contrainte est-elle suffisante pour permettre à tout couple de capteurs de communiquer ?

Q 2.1 Formuler la question posée par une propriété dans un graphe non orienté à n sommets (n pair).

Q 2.2 Si la proposition est vraie, la prouver. Sinon, fournir un contre-exemple.

Exercice 3 – Rencontres et graphe triangulé (exercice d'entraînement)

Le duc de Densmore connut une fin tragique puisqu'il périt dans l'explosion qui détruisit son château. Son testament fut détruit ; or celui-ci avait tout pour déplaire à l'une de ses sept ex-femmes.

Peu avant le crime, elles étaient toutes venues au château et elles jurèrent que ce fut la seule fois où elles s'y étaient rendues. Elles peuvent donc toutes être coupables, mais le dispositif ayant provoqué l'explosion avait été dissimulé dans une armure dans la chambre du duc, et sa pose avait nécessité plus d'une visite. Donc la coupable a menti : elle est venue plusieurs fois. On dispose des déclarations suivantes :

1. Ann dit avoir rencontré Betty, Charlotte, Félicia et Georgia ;
2. Betty dit avoir rencontré Ann, Charlotte, Edith, Félicia et Helen ;
3. Charlotte dit avoir rencontré Ann, Betty et Edith ;
4. Edith dit avoir rencontré Betty, Charlotte et Félicia ;
5. Félicia dit avoir rencontré Ann, Betty, Edith et Helen ;
6. Georgia dit avoir rencontré Ann et Helen ;
7. Helen dit avoir rencontré Betty, Félicia et Georgia.

Q 3.1 Tracer le graphe des rencontres, dont les sommets représentent les sept femmes et où l'on trace une arête entre deux sommets lorsque deux femmes se sont rencontrées.

Q 3.2 Montrer que si personne ne ment alors il ne peut pas exister de cycle de longueur 4 sans corde (arête reliant deux sommets non consécutifs du cycle) dans le graphe de rencontres.

Q 3.3 En déduire qui a tué le duc de Densmore (en considérant que les six innocentes n'ont pas menti !).

Exercice 4 – Deux prisonniers (exercice d'entraînement)

Deux prisonniers sont soumis à une épreuve. Dans une pièce se trouve une grande table et sur la table sont posées 50 boîtes fermées contenant des cartes numérotées de 1 à 50. Il y a une carte pour chaque numéro de 1 à 50 et chaque boîte contient une carte. La distribution des cartes dans les boîtes a été faite aléatoirement et elle est inconnue des prisonniers. Les deux prisonniers peuvent convenir d'une stratégie avant, mais ne peuvent plus se parler le moment venu de l'épreuve. Le premier prisonnier rentre dans la pièce, regarde le contenu de toutes les boîtes et, s'il le désire, il inverse le contenu de deux boîtes (mais seulement deux). Le deuxième prisonnier entre alors dans la pièce (le premier est sorti), les gardiens de la prison lui assignent un numéro (aléatoirement) entre 1 et 50, et il doit trouver ce numéro en ouvrant au maximum 25 boîtes. S'il trouve le numéro qui lui a été donné, les deux prisonniers sont libé- rés, sinon, ils sont exécutés.

Q 4.1 Le problème semble défier l'entendement, car on a du mal à comprendre la stratégie que pourrait adopter le premier prisonnier sans avoir connaissance du numéro qui va être attribué au second. Pourtant, s'ils sont malins, les deux prisonniers sont certains d'être libérés. Comment font-ils ?

Exercice 5 – Représentation des graphes et primitives élémentaires (exercice de base)

Il existe différentes façons de représenter les graphes. Nous citons les deux représentations suivantes :

- la représentation par *matrice d'adjacence* consiste en une matrice carrée M d'ordre n telle que $M_{ij} = 1$ si $\{i, j\}$ est une arête (respectivement si (i, j) est un arc) de G, $M_{ij} = 0$ sinon ;
- la représentation par *liste d'adjacence* consiste en un tableau Adj de n listes. Pour tout sommet i de S, la liste $Adj[i]$ contient la liste des sommets j de S tels que $\{i, j\}$ est une arête (respectivement (i, j) est un arc) de G.

Q 5.1 Donner la structure de données correspondant aux deux types de représentations.

Q 5.2 Quelle est la taille en mémoire requise pour représenter un graphe de n sommets et m arêtes (respectivement arcs) selon les deux représentations ?

Q 5.3 Si G est non orienté, déterminer pour chacune des représentations, la complexité des primitives suivantes :

1. fonction $Adjacent(i, j : sommet)$: *booléen*, qui rend "vrai" si i et j sont adjacents ;
2. fonction $Degré(i : sommet)$: *entier*, qui calcule le degré du sommet i.

Q 5.4 Si G est orienté, déterminer pour chacune des représentations, la complexité des primitives suivantes :

1. fonction $Successeur(i, j : sommet)$: *booléen*, qui rend "vrai" si j est un successeur de i ;
2. fonction $Prédécesseur(i, j : sommet)$: *booléen*, qui rend "vrai" si j est un prédécesseur de i ;
3. fonction $Degré\_Extérieur(i : sommet)$: *entier*, qui calcule le degré extérieur du sommet i ;

4. fonction *Degré_Intérieur*(i : *sommet*) : *entier*, qui calcule le degré intérieur du sommet i.

Q 5.5 Évaluer, selon la représentation choisie, la complexité d'un algorithme qui calcule le nombre de sommets de degré impair d'un graphe non orienté.

Exercice 6 – Tri topologique (exercice d'approfondissement)

On rappelle que dans une liste topologique L, pour tout arc (x, y) du graphe, x est placé avant y dans la liste. Un graphe (orienté) a une liste topologique si et seulement s'il est sans circuit.

Q 6.1 Proposer un algorithme de construction d'une liste topologique basé sur la propriété suivante : tout graphe sans circuit possède un sommet sans prédécesseur. Calculer sa complexité.

Q 6.2 Appliquer l'algorithme proposé sur le graphe représenté ci-dessous :

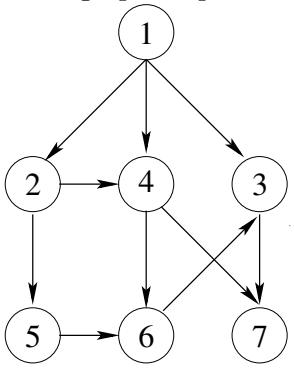


FIG. 3– Un graphe sans circuit.

Parcours de graphes

Exercice 7 – Parcours générique (exercice de base)

On considère le graphe $G_1 = (S, A)$ représenté par la figure 4.

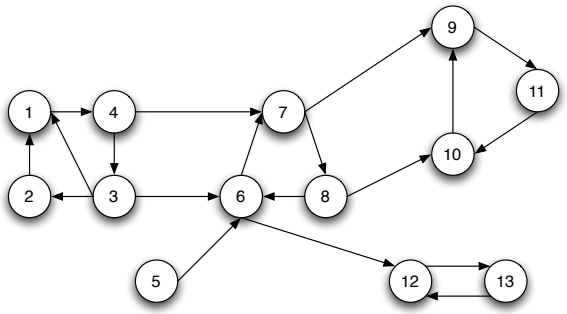


FIG. 4– Graphe G_1

Q 7.1 Déterminer les composantes fortement connexes de G_1 et le graphe réduit $\hat{G}_1$ de G_1.

Q 7.2 Montrer que la liste $L = (6, 7, 9, 11, 10, 8, 12, 13, 5, 4, 3, 2, 1)$ est un parcours de G_1. Quels sont les points de régénération de L ? Donner une forêt couvrante associée à L.

Q 7.3 Expliquer pourquoi tout parcours de G_1 possède au moins 2 points de régénération.
Donner un parcours de G_1 possédant exactement 2 points de régénération.
Expliquer pourquoi tout parcours de G_1 possède au plus 5 points de régénération.
Donner un parcours de G_1 possédant exactement 5 points de régénération.

Q 7.4 Dans le cas général, donner le nombre de points de régénération minimum et maximum du parcours d'un graphe orienté.

Exercice 8 – Reconnaissance de graphe biparti (exercice d'entraînement)

Soit $G = (S, A)$ un graphe non orienté *connexe*. L'objet de cet exercice est de concevoir un algorithme qui détermine si G est *biparti*. Un graphe est biparti si l'ensemble S peut être partitionné en deux sous-ensembles S_1 et S_2 (avec $S_1 \cup S_2 = S$ et $S_1 \cap S_2 = \emptyset$) tel qu'il n'existe pas d'arêtes entre sommets d'un même sous-ensemble (par exemple, si $(x, y) \in S_1^2$ alors il n'existe pas d'arête entre x et y).

Soit $L = (s_1, \ldots, s_n)$ un parcours de G à partir d'un sommet quelconque s_1 de S et $F(L)$ la forêt couvrante associée à L. On colore (en rouge ou bleu) tous les sommets de S selon la règle de coloration suivante :

- le sommet s_1 est bleu ;
- pour $i = 2, \ldots, n$, le sommet s_i est bleu (resp. rouge) si son père dans $F(L)$ est rouge (resp. bleu).

Le but de cet exercice est de montrer que G est biparti si et seulement si il n'existe pas d'arête de A entre deux sommets de même couleur.

Q 8.1 Faire un parcours (quelconque) du graphe suivant à partir du sommet 1, en indiquant la liste L obtenue et une forêt couvrante associée $F(L)$. En utilisant la règle de coloration présentée plus haut, préciser si le graphe est biparti.

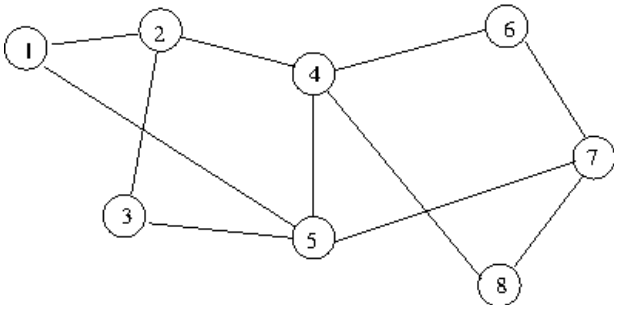


FIGURE 7 – Graphe non orienté.

On considère dorénavant un graphe $G = (S, A)$ non-orienté connexe quelconque. L'objet des questions 2 à 4 est de prouver que G est biparti si et seulement si il n'existe pas d'arête de A entre deux sommets de même couleur.

Q 8.2 Montrer que s'il n'existe pas d'arête entre deux sommets de même couleur dans $A \setminus F(L)$, alors le graphe G est biparti.

Q 8.3 Soit G un graphe biparti. Montrer que :

a) s'il existe une chaîne de longueur paire (en nombre d'arêtes) dans G entre deux sommets x et y alors ces deux sommets x et y appartiennent nécessairement au même sous-ensemble (S_1 ou S_2) dans la partition des sommets ;

b) s'il existe une chaîne de longueur impaire dans G entre deux sommets x et y alors l'un appartient à S_1 et l'autre à S_2.

Q 8.4 Soit L un parcours de G. Après coloration, on considère une arête $\{x, y\}$ entre deux sommets de même couleur. En considérant la chaîne qui relie x et y dans $F(L)$, montrer que le graphe n'est pas biparti.

Q 8.5 Compléter la procédure ci-dessous pour qu'elle permette de détecter si un graphe est biparti en $O(n + m)$. La variable globale *est_biparti* est initialisée à vrai. La valeur de cette variable à l'issue de la procédure doit être vrai si le graphe est biparti, et faux sinon. Le tableau C indexé sur les sommets indique la couleur des sommets. Il est initialisé de la façon suivante : pour tout x, $C[x]$ = incolore. Etant donné un sommet s_1 de départ, le premier appel de la procédure est $Parcours\_Rec(G, s_1, bleu)$.

Algorithme : Parcours_Rec(G : Graphe, s : Sommet, c : Couleur)
variable locale x : Sommet;
$C[s] \leftarrow c$;
pour tous les *voisins x de s* **faire**
  si **alors**
    est_biparti $\leftarrow$ faux;
  si $C[x] =$ *incolore* **alors**

Exercice 9 – Piratage informatique (exercice d'entraînement)

Ca y est ! Vous avez mis au point un nouveau virus informatique. Votre but est d'infecter l'ensemble des ordinateurs d'un réseau. Un exemple de réseau d'ordinateurs est représenté sur la figure 8 : les sommets sont les ordinateurs, et un arc (i, j) signifie que si l'ordinateur i est infecté, alors il va transmettre le virus à l'ordinateur j qui sera donc aussi infecté. Si par exemple vous introduisez le virus dans l'ordinateur 2, 2 est infecté, donc 1, 5 et 3 le seront aussi, donc 6 et 7 aussi, donc 8 aussi, et ainsi de suite ...

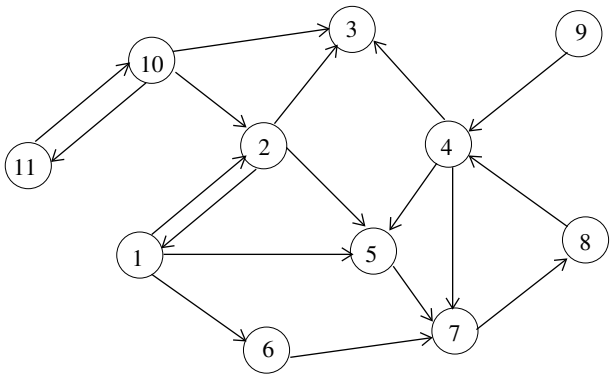


FIGURE 8 – graphe G'

Votre stratégie consiste à pirater certains ordinateurs du réseau pour y introduire le virus. Ensuite, vous laissez le virus se propager de lui-même. Vous devez choisir l'ensemble des ordinateurs piratés de manière à ce que tous les ordinateurs du réseau soient infectés (à la fin du processus de propagation). Sachant qu'il n'est pas évident de pirater un ordinateur, vous souhaitez parvenir à votre but en piratant un nombre minimum d'ordinateurs. Le but de cet exercice est de résoudre ce problème.

Nous notons $G = (S, A)$ le graphe orienté représentant le réseau d'ordinateurs. Nous dirons indifféremment qu'un sommet est infecté ou qu'un ordinateur est infecté.

Q 9.1 Soit $R \subset S$ le sous-ensemble de sommets (d'ordinateurs) que vous allez pirater. A quelle condition un sommet j du graphe sera-t-il infecté par le virus ? Sur l'exemple du graphe G' (figure 8), si l'on pirate $R = \{1, 7, 9\}$, est-ce que tous les sommets seront infectés ? De manière générale, que doit vérifier R (en termes de chemins dans le graphe) pour que la contrainte selon laquelle tous les sommets doivent être infectés soit vérifiée ?

Nous cherchons donc maintenant à trouver un ensemble R de taille minimale qui satisfait cette contrainte.

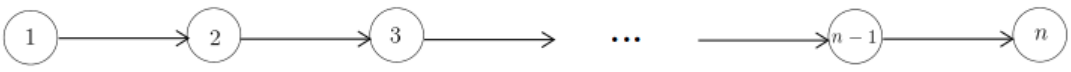


FIGURE 9 – graphe G''

Q 9.2 Votre complice vous propose la stratégie suivante : faire un parcours de G, et pirater l'ensemble des racines des arborescences composant la forêt couvrante associée au parcours.

1. Cette solution permet-elle à tous les ordinateurs d'être infectés à la fin du processus de propagation ? Justifiez votre réponse.
2. Considérons le graphe G'' qui est simplement un chemin $(s_1, s_2, \ldots, s_{n-1}, s_n)$ (voir figure 9). Quel est l'ensemble R^* optimal ? Dans le pire des cas, quel ensemble R la stratégie proposée par votre complice va-t-elle donner ? Est-ce une bonne stratégie ?

Q 9.3 On considère dans cette question le graphe G' *de la figure 8* (et non le graphe G'' de la figure 9).

1. Déterminez les composantes fortement connexes de G'.
2. Dessinez le graphe réduit G'_r de G'.
3. Déterminez un ensemble R' de G' qui vous semble optimal. Vous expliquerez brièvement votre choix (on ne demande pas de prouver ici l'optimalité de R').

Q 9.4 On considère dans cette question un graphe G orienté quelconque. Expliquez comment déterminer un ensemble R optimal (à partir du graphe G et de son graphe réduit G_r). Prouver l'optimalité de l'ensemble R construit (ainsi que le fait qu'à la fin du processus de propagation tous les ordinateurs soient infectés).

Exercice 10 – Détection de circuits dans un graphe (exercice de base)

Partie 1 : Application

La mise en œuvre d'une série de programmes $P_1, \cdots, P_n$ sur un réseau d'ordinateurs doit être planifiée. On donne les relations "P_j utilise les données calculées par le programme P_i" par les couples (P_i, P_j) suivants : (P_1, P_6), (P_2, P_1), (P_3, P_8), (P_4, P_1), (P_4, P_2), (P_5, P_{10}), (P_6, P_2), (P_6, P_{12}), (P_6, P_{13}), (P_8, P_{10}), (P_8, P_{12}), (P_9, P_1), (P_9, P_4), (P_{10}, P_7), (P_{11}, P_4), (P_{12}, P_3), (P_{12}, P_5), (P_{13}, P_3).

Q 10.1 Comment tester si cette mise en œuvre est possible ?

Partie 2 : Algorithme de parcours en profondeur

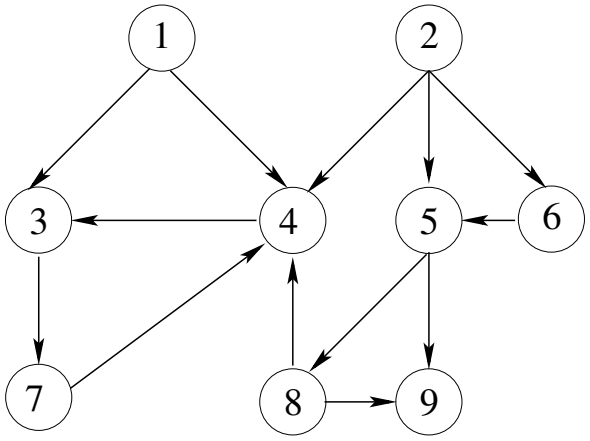


FIG. 10– Un graphe orienté.

Q 10.2 Est-ce que les listes $L_1 = (1, 3, 4, 7, 2, 6, 5, 8, 9)$ et $L_2 = (8, 4, 3, 7, 9, 1, 2, 6, 5)$ sont des parcours en profondeur du graphe de la figure 10 ? Si oui, fournir une forêt couvrante associée

et préciser si elle est unique.

Q 10.3 Ecrire un algorithme récursif de parcours en profondeur du graphe G.

Q 10.4 Effectuer un parcours en profondeur du graphe de la figure 10, à partir du sommet 1, en supposant que les successeurs d'un sommet sont visités dans l'ordre des indices croissants. On précisera quels sont les arcs de la forêt couvrante $\mathcal{F}(L, G)$, ainsi que les arcs avants, arrières, transverses et de liaison.

Partie 3 : Algorithme de détection de circuits

On souhaite adapter l'algorithme de parcours en profondeur pour qu'il puisse détecter la présence de circuits dans le graphe. À cette fin, on démontre tout d'abord qu'il y a équivalence entre l'existence d'un arc arrière et l'existence d'un circuit. Dans la suite, L est un parcours en profondeur du graphe G.

Q 10.5 Montrer que s'il y a un arc arrière dans le parcours L, alors G possède un circuit (condition nécessaire).

Q 10.6 Rappeler le lemme des sommets accessibles vu en cours.

Q 10.7 Montrer que tout circuit de G possède un arc arrière (condition suffisante).

Q 10.8 Comment peut-on adapter l'algorithme de parcours en profondeur pour qu'il détecte la présence de circuits dans le graphe ?

Exercice 11 – Plus courts chemins en nombre d'arcs (exercice de base)

Q 11.1 On considère une ville où l'on distingue les points d'intérêts suivants : a, b, c, d et e. Pour chaque point t, on fournit la liste des autres points d'intérêt qui permettent d'y accéder en voiture (les routes sont à sens unique). On obtient :

Point t	a	b	c	d	e
Prédécesseur	e	a	(b, d)	(a, b)	b

Peut-on, à partir du point d'intérêt a, atteindre tous les autres points d'intérêt de la ville ? Le problème pourra être modélisé à l'aide d'un graphe orienté G.

Q 11.2 On souhaite connaître le temps minimum nécessaire pour atteindre n'importe quel point d'intérêt à partir de a. On supposera que le temps de trajet entre l'origine et l'extrémité de tout arc de G est égal à 10 minutes.
Proposer un algorithme pour résoudre ce problème, et l'appliquer.

Exercice 12 – Parcours en largeur et algorithme de Dijkstra (exercice de base)

Q 12.1 Dérouler l'algorithme de Dijkstra pour déterminer l'arborescence des plus courts chemins depuis le sommet 1 dans le graphe $G = (S, A, c)$ orienté de la figure 12 (on représentera l'arborescence partielle des plus courts chemins à chaque étape de l'algorithme). En cas d'égalité, on examinera en priorité le sommet de plus petit indice.

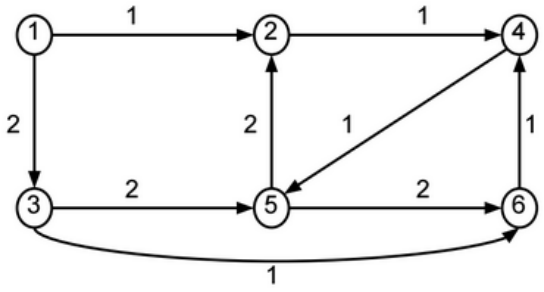


FIG. 12– Un graphe orienté valué.

On définit un nouveau graphe $G' = (S \cup S', A'_1 \cup A'_2, c')$ à partir de G, où :

- S' comporte un sommet x_a pour chaque arc $a = (x, y)$ de A pour lequel $c(a) = 2$,
- $A'_1 = \{a \in A : c(a) = 1\}$,
- A'_2 comporte deux arcs (x, x_a) et (x_a, y) pour chaque arc $a = (x, y)$ de A pour lequel $c(a) = 2$,
- c' est une fonction de coût unitaire (qui associe un coût de 1 à chaque arc de $A'_1 \cup A'_2$).

Q 12.2 Représenter G' ainsi que l'arborescence couvrante associée à un parcours en largeur de G' depuis le sommet 1 (qui visite en priorité le sommet de plus petit indice).
A quoi correspond cette arborescence si on la réinterprète dans G ? (on ne demande pas ici de justification)

Q 12.3 On suppose ici que le graphe est représenté sous forme de tableau de listes de successeurs.
Pour un graphe G avec des coûts 1 et 2, quelle est la complexité de l'algorithme consistant à construire un nouveau graphe G' à partir de G puis à réaliser un parcours en largeur de G' ? Rappeler la complexité de l'algorithme de Dijkstra. Conclusion ?

Exercice 13 – Algorithme de Dijkstra et analyse de sensibilité (exercice d'entraînement)

On considère le graphe valué G de la figure 13.

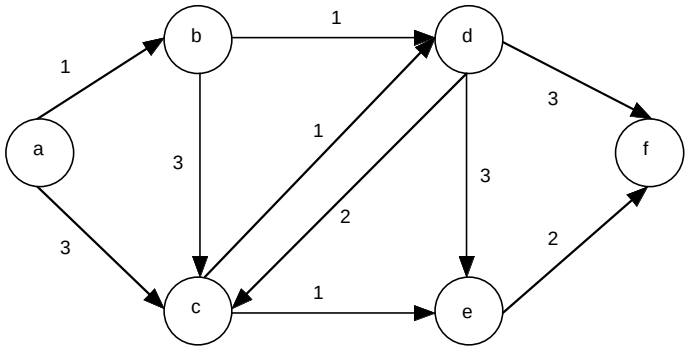


FIGURE 13 – Un graphe valué

Q 13.1 Appliquer l'algorithme de Dijkstra pour déterminer l'arborescence des chemins de coût minimum à partir du sommet a dans G.

Q 13.2 On suppose maintenant que le coût de l'arc (d, e) est égal à $3 - \epsilon$ où ϵ est un paramètre

réel positif. Pour quelles valeurs de ϵ l'arborescence des chemins de coût minimum est-elle inchangée ? Justifiez votre réponse.

Q 13.3 En supposant toujours que le coût de l'arc (d, e) est égal à $3 - \epsilon$, tracer la courbe donnant la valeur du plus court chemin de a à f en fonction de ϵ, pour $\epsilon \geq 0$.

Exercice 14 – Chemins de coût maximum (exercice d'entraînement)

Cet exercice propose l'analyse d'un algorithme qui détermine, s'ils existent, les chemins *de coût maximum* (d'origine fixée) d'un graphe valué.

Soit $G = (S, A)$ un graphe orienté à n sommets possédant une racine s et soit $c(a)$ (entier relatif) le coût de l'arc a. La figure 14 représente un tel graphe (le sommet racine est le sommet 1).

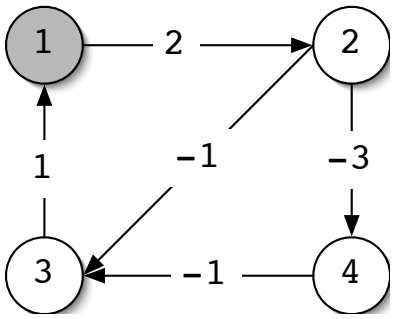


FIGURE 14 – Un graphe orienté valué

On note $C_k(x)$ l'ensemble des chemins de s à x possédant *au plus k arcs*.

Q 14.1 Démontrer que, s'il n'est pas vide, $C_k(x)$ contient un chemin de coût maximum de s à x d'au plus k arcs.

Q 14.2 Déterminer $C_0(s)$ et $C_0(x)$ pour tout sommet x distinct de s.

Soit $P(y)$ l'ensemble des prédécesseurs du sommet y. Etant donné $y \in S$ et $x \in P(y)$, si $C_k(y)$, $k \geq 1$ est non vide, on note $C_k^x(y)$ le sous-ensemble des chemins de $C_k(y)$ dont le dernier arc est l'arc (x, y).

Q 14.3 Montrer que les ensembles $C_k^x(y)$, avec $x \in P(y)$, sont disjoints et que leur union est égale à $C_k(y)$.

On notera $L_k(x)$ le coût maximum d'un chemin d'au plus k arcs de s à x dans G et on posera par convention $L_k(x) = -\infty$ si $C_k(x)$ est vide et $L_0(s) = 0$.

Q 14.4 Démontrer que le coût maximal d'un chemin de $C_k^x(y), x \in P(y)$ est égal à $L_{k-1}(x) + c(x, y)$.

Q 14.5 En déduire que pour tout sommet y de G et pour tout $k \geq 1$, on a :

$$L_k(y) = \max(L_{k-1}(y), \max_{(x,y)\in A}\{L_{k-1}(x) + c(x, y)\}).$$

Q 14.6 Démontrer que si G ne possède pas de circuit de coût strictement positif, $L_{n-1}(x)$ est le coût maximum d'un chemin de s à x dans G.

Q 14.7 On considère l'algorithme $CHMAX(G, c, s, K)$ suivant où *val* est une matrice de taille

$((K+1) \times n)$ telle que pour tout $(k,i) \in \{0,\ldots,K\} \times \{1,\ldots,n\}$, $val[k,i]$ est soit un entier relatif soit $-\infty$. La valeur initiale de chaque case de la matrice est $-\infty$.

Algorithme : CHMAX(G, c, s, K)
$val[0, s] \leftarrow 0$;
pour k *de* 1 *à* K **faire**
  pour *tout sommet* y *de* G **faire**
    $val[k, y] \leftarrow \max(val[k-1, y], \max_{x \in P(y)}\{val[k-1, x] + c(x, y)\})$;

Exécuter l'algorithme $CHMAX(G, c, s, K)$ sur le graphe $G = G_1$ de la figure 14, le sommet $s = 1$ et $K = 4$. Dessiner le graphe G_1 (pour chaque valeur de k) et écrire les valeurs de $val[k, x]$ à côté du sommet x sur chaque graphe.

Q 14.8 Montrer dans le cas général que, après l'exécution de $CHMAX(G, c, s, K)$, on a pour tout sommet x et tout $k \in \{0, \cdots, K\}$: $val[k, x] = L_k(x)$.

Q 14.9 Démontrer que si $L_n(x) > L_{n-1}(x)$, il existe un circuit de coût > 0 dans G.

Q 14.10 Donner et justifier la complexité de l'algorithme $CHMAX(G, c, s, K)$.

Exercice 15 – Algorithme de Prim (exercice de base)

Appliquer l'algorithme de Prim pour obtenir un arbre couvrant de poids minimum pour le graphe $G = (S, A, c)$ non orienté valué connexe représenté dans la figure 15.

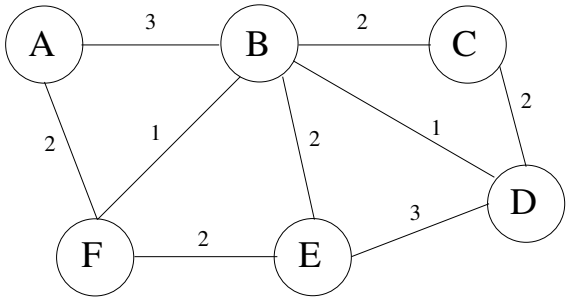


FIG. 15– Un graphe connexe valué.

Exercice 16 – Arbre couvrant et arborescence de chemins (exercice de base)

Soit $G = (S, A, v)$ un graphe non orienté valué connexe, tel que pour tout $e \in A, v(e) \geq 0$. On suppose que l'on a construit un arbre de poids minimum de G, ainsi qu'une arborescence des plus courts chemins d'un sommet $s \in S$ à tous les autres sommets du graphe. On suppose maintenant que le coût de chaque arête augmente de 1 : les nouveaux coûts sont $v'(e) = v(e)+1$.

Q 16.1 Est-ce que l'arbre couvrant de poids minimum change ? Donner un exemple dans lequel il change, ou bien prouver qu'il ne peut pas changer.

Q 16.2 Est-ce que l'arborescence des plus courts chemins change ? Donner un exemple ou elle

change, ou bien prouver que cela n'est jamais le cas.

Exercice 17 – Vrai ou faux ? (exercice de base)

Indiquer si chacune des propositions suivantes est vraie (dans ce cas là, le prouver), ou fausse (donner un contre-exemple). On supposera que $G = (S, A, v)$ est un graphe non-orienté connexe, avec $n = |S|$ et $m = |A|$.

a) Si le graphe G a plus de $n - 1$ arêtes, et s'il y a une seule arête de coût maximum, alors cette arête ne peut pas faire partie d'un arbre couvrant de poids minimum.

b) Si G contient un cycle avec une unique arête e de coût maximum, alors e ne peut pas faire partie d'un arbre couvrant de poids minimum.

c) Soit $e \in A$ une arête de coût minimum. Alors e appartient à un arbre couvrant de poids minimum de G.

d) Si l'arête de coût minimum de G est unique, alors elle fait nécessairement partie de tout arbre couvrant de poids minimum.

e) Si G contient un cycle avec une unique arête e de coût minimum, alors e fait partie de tout arbre couvrant de poids minimum.

f) La plus courte chaîne entre deux sommets fait nécessairement partie d'un arbre couvrant de poids minimum.

g) L'algorithme de Prim est valide même quand les coûts des arêtes sont négatifs.

Exercice 18 – Arbre de niveau de congestion minimal (exercice d'entraînement)

Soit $G = (S, A)$ un graphe non orienté connexe valué à n sommets possédant m arêtes et soit $c(a)$ le poids de l'arête a.
Etant donné $T = (S, A_T)$ un arbre couvrant de G, le *niveau de congestion* de T est défini par le poids de l'arête de poids maximal dans A_T.
On s'intéresse à la recherche d'un *arbre couvrant de niveau de congestion minimal* dans G.

Q 18.1 L'arbre couvrant défini par les arêtes en gras pour le graphe de la Figure 19 est-il un arbre couvrant de niveau de congestion minimal ?

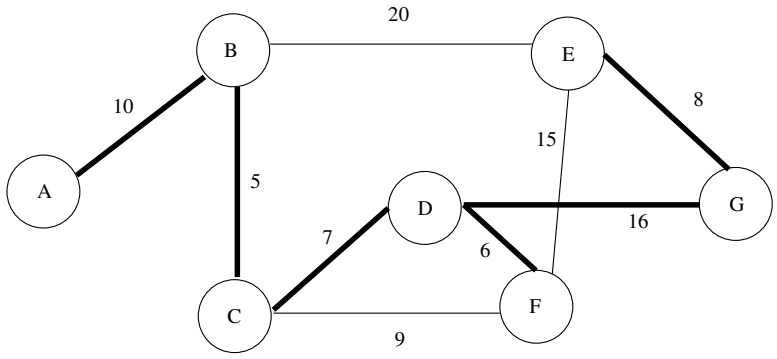


FIGURE 19 – Graphe non orienté valué et arbre couvrant

Q 18.2 Calculer un arbre couvrant de poids minimum du graphe représenté à la Figure 20. Vous préciserez le sous-graphe partiel obtenu à chaque itération.

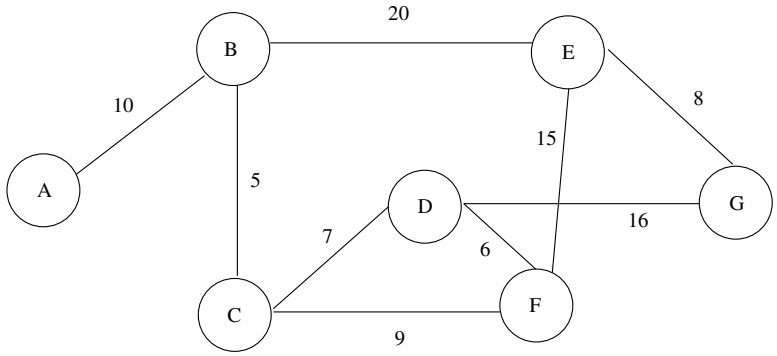


FIGURE 20 – Un graphe non orienté valué

Q 18.3 Dans le cas général, tout arbre couvrant de niveau de congestion minimal est-il un arbre couvrant de poids minimum ? Si oui, fournir une preuve. Si non, donner un contre-exemple.

Q 18.4 Réciproquement, tout arbre couvrant de poids minimum est-il un arbre couvrant de niveau de congestion minimal ? Si oui, fournir une preuve. Si non, donner un contre-exemple.

Q 18.5 En déduire un algorithme pour calculer un arbre couvrant de niveau de congestion minimal. Préciser sa complexité temporelle.

On se propose dans la suite de l'exercice de s'intéresser à un autre algorithme de calcul d'un arbre couvrant de niveau de congestion minimal.
Soit x le nombre de valeurs distinctes de poids des arêtes de A. On note $p_1, \cdots, p_x$ l'ensemble de ces poids avec $p_1 \leq p_2 \cdots \leq p_x$.
Soit $G_i = (S, A_i)$ le graphe partiel contenant l'ensemble des arêtes de poids inférieur ou égal à p_i.

Q 18.6 Que peut-on dire du niveau de congestion de tout arbre couvrant de G si G_i n'est pas connexe ?

Q 18.7 On considère l'algorithme suivant :

$i \leftarrow 1$, $F \leftarrow G_1$;
tant que *graphe F n'est pas connexe* **faire**
  $i \leftarrow i + 1$;
  $F \leftarrow G_i$;
fin
retourner F;

1. Analyser sa complexité.
2. A partir du graphe F retourné par l'algorithme, proposer un algorithme en temps linéaire pour calculer un arbre couvrant de niveau de congestion minimal.
3. En déduire la complexité temporelle de calcul d'un arbre couvrant de niveau de congestion minimal.
4. Déroulez votre algorithme sur le graphe de la Figure 20. Vous fournirez notamment les graphes G_i obtenus à la fin de chaque itération de la boucle Tant que de l'algorithme.

Algorithmique – 3I003

Algorithmes gloutons

Conception et analyse d'algorithmes gloutons

Exercice 1 – Algorithme de Kruskal

Soit $G = (S, A)$ un graphe connexe. Soit T l'ensemble des arêtes retournées par l'algorithme de Kruskal ayant comme entrée le graphe G. Le but de cet exercice est de montrer que T est un arbre couvrant de coût minimum.

Q 1.1 Montrer que T est un arbre couvrant. On pourra pour cela montrer que l'ensemble des arêtes T est une forêt, est couvrant, et tel qu'il y ait une seule composante connexe.

Soit T^* un arbre couvrant de coût minimum. Si $T = T^*$ alors T est un arbre couvrant de poids minimum. Sinon, soit e l'arête de plus petit coût dans $T^* \backslash T$. Soit $c(e)$ le coût de e.

Q 1.2 Montrer que $T \cup \{e\}$ contient un cycle $\mathcal{C}$ tel que :

a) Chaque arête de $\mathcal{C}$ a un coût inférieur ou égal à $c(e)$.

b) Il existe dans $\mathcal{C}$ une arête f qui n'appartient pas à T^*.

Q 1.3 Soit T_1 l'arbre $T \backslash \{f\} \cup \{e\}$. Montrer que :

— T_1 est un arbre couvrant

— T_1 a plus d'arêtes en commun avec T^* que T n'en a.

— Le coût de T_1 est supérieur ou égal au coût de T.

Q 1.4 Conclure.

Exercice 2 – Deuxième meilleur arbre couvrant de poids minimum

Soient $G = (S, A)$ un graphe connexe non orienté avec $n = |S|$ et $m = |A|$. On suppose que $m \geq n$. On considère une fonction de pondération c définie sur A et on suppose que tous les poids des arêtes sont distincts.

Soit $\mathcal{T}$ l'ensemble de tous les arbres couvrants de G et soit T^* un arbre couvrant minimum de G. On appelle deuxième meilleur arbre couvrant minimum, un arbre couvrant T tel que $c(T) = \min_{T' \in \mathcal{T}, T' \neq T^*} \{c(T')\}$.

Q 2.1 Montrer que l'arbre couvrant de poids minimum est unique.

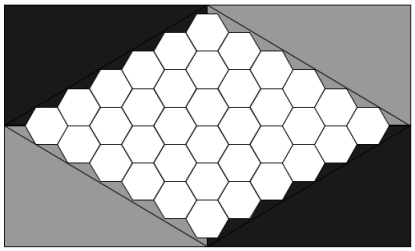
Q 2.2 Montrer que le deuxième meilleur arbre couvrant de poids minimum de G n'est pas forcément unique.

Exercice 3 – Jeu de Hex

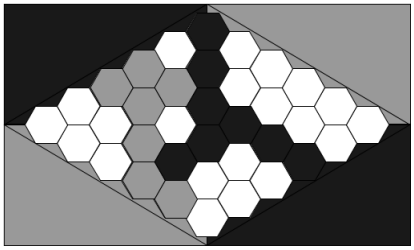
Le jeu de Hex est un jeu pour deux joueurs. Il se joue sur un plateau en forme de losange dont les cases sont hexagonales. Au début de la partie, le plateau est vide (voir figure de gauche). Chaque joueur est représenté par une couleur, noir et gris. Les joueurs possèdent des pions à leur couleur qu'ils disposent tour à tour sur une case de leur choix et un par un. Le plateau se remplit ainsi progressivement. L'objectif d'un joueur, par exemple Noir, est de relier les deux

côtés du losange symbolisés par sa couleur (la couleur noire pour Noir). Si la configuration des pions noirs permet la création d'une chaîne continue de pions noirs reliant un côté noir à l'autre (figure de droite), Noir a gagné et le jeu s'arrête (condition de victoire symétrique pour Gris).

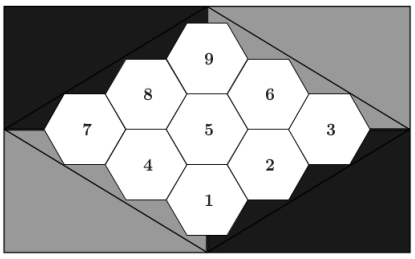
Un plateau vide au départ



Une configuration gagnante pour Noir



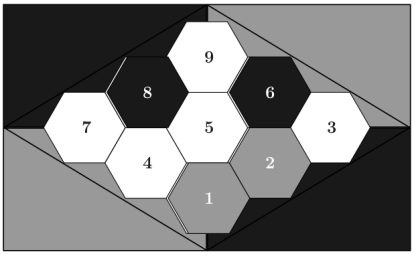
L'objet de cet exercice est de montrer comment la structure de données *union-find* peut être utilisée pour réaliser un programme de jeu de Hex. Plus précisément, on utilise cette structure de données pour détecter si un joueur a gagné à l'issue de son tour. Pour simplifier, on va considérer le jeu de Hex de format réduit 3×3 ci-dessous, dont les cases sont numérotées de 1 à 9. De plus, les quatre bords du plateau sont numérotés 10 (bord noir en haut à gauche), 11 (bord noir en bas à droite), 12 (bord gris en bas à gauche) et 13 (bord gris en haut à droite).



Q 3.1 Afin d'utiliser la structure de données union-find pour détecter si un joueur a gagné, indiquer comment définir l'ensemble des éléments, et à quoi correspondront les classes d'équivalence en cours de partie.

Q 3.2 Quelle est la partition initiale pour le jeu de format 3×3 donné plus haut ? (lorsque le plateau est vide)

Q 3.3 Si on place un pion noir sur la case 5 dans la configuration ci-dessous, quelle(s) opération(s) d'union(s) faut-il réaliser pour mettre à jour la structure de données ?



Q 3.4 Pour le plateau réduit 3×3 donné plus haut, quel test (exploitant la structure de données union-find) faut-il réaliser pour détecter si Noir a gagné ?

Exercice 4 – Le coût de la panne sèche

Le professeur Midas conduit une voiture entre Amsterdam et Lisbonne sur l'européenne E10. Son réservoir d'essence plein lui permet de parcourir n kilomètres, et sa carte lui donne les distances entre les stations-service sur la route. Le professeur souhaite faire le moins d'arrêts possible pendant le voyage.

Q 4.1 Donnez une méthode efficace permettant au professeur Midas de déterminer les stations-service où il peut s'arrêter.

Q 4.2 Montrez que votre stratégie aboutit à une solution optimale.

Exercice 5 – Coloration d'un graphe

Considérons un graphe $G = (S, A)$ non orienté, de degré maximal Δ. Le nombre chromatique de G, noté $\chi(G)$, est le nombre minimal de couleurs nécessaire pour colorier le graphe, c'est-à-dire pour colorier chaque sommet de telle façon que deux sommets distincts et adjacents aient toujours des couleurs différentes.

Q 5.1 Notons $\{1, 2, \ldots, n\}$ les couleurs. Donnez un algorithme glouton, qui parcourt tous les sommets du graphe et attribue une couleur à chaque sommet.

Q 5.2 Montrez que cette approche permet de majorer le nombre chromatique par $\chi(G) \leq \Delta+1$.

Q 5.3 Donnez un exemple de graphe pour lequel l'algorithme précédent nécessite $\Delta+1$ couleurs, mais pour lequel 2 couleurs seulement sont nécessaires pour colorier le graphe.

Exercice 6 – Apprentisage musical

L'élève Chopin prépare un récital de piano. Il souhaite préparer un nombre maximum de morceaux tout en respectant un ensemble de contraintes. Il doit choisir parmi m morceaux. Chaque morceau i nécessite un temps de préparation de p_i heures. Chopin a accès au piano T heures au maximum pour répéter seul. Par ailleurs, il peut disposer de l'aide de n professeurs. Chaque professeur j peut consacrer au plus t_j heures à étudier avec Chopin en plus de ses répétitions en solitaire. Chaque professeur n'accepte de travailler qu'un seul morceau et à la seule condition que Chopin ne l'étudie pas avec un autre professeur.

Ainsi, pour apprendre un morceau i, Chopin peut :

- soit travailler seul en consacrant p_i heures de son temps (limité à T),
- soit choisir un unique professeur j (qui ne l'aide pas à préparer d'autres morceaux) pour travailler ensemble pendant $\min\{p_i, t_j\}$ heures. Si $p_i > t_j$, Chopin apprendra la fin du morceau en y consacrant $p_i - t_j$ heures de son temps de répétition solitaire (limité à T).

Il n'est pas utile de préparer qu'une partie d'un morceau i (autrement dit, y consacrer moins de p_i heures). Celui-ci ne pourra pas être joué.

Q 6.1 Supposons que Chopin décide d'apprendre exactement k morceaux.

a) Montrer que l'on peut ne s'intéresser qu'à l'ensemble P_k des k morceaux de plus petits p_i.

b) Montrer que l'on peut ne s'intéresser qu'à l'ensemble T_k des k professeurs de plus grands t_j.

Q 6.2 En vous appuyant sur la question précédente, décrire le principe d'un algorithme glouton

efficace pour déterminer si Chopin peut apprendre exactement k morceaux. Justifier la réponse. Quelle est la complexité de l'algorithme ?

Q 6.3 En vous aidant de la question précédente, décrire le principe d'un algorithme efficace qui calcule le nombre maximum de morceaux que Chopin peut étudier. Précisez sa complexité.

Exercice 7 – Playlist

Votre lecteur MP3 a une capacité de M megabits. Vous souhaitez télécharger de la musique sur votre lecteur. Vous sélectionnez n titres $t_1, \cdots, t_n$ de vos artistes préférés. Chaque titre t_i occupe m_i megabits.
On suppose que $\sum_{i=1}^{n} m_i > M$: la capacité de votre lecteur ne vous permet pas de télécharger l'ensemble des n titres.

Q 7.1 Vous souhaitez maximiser *le nombre de titres téléchargés* sur votre lecteur MP3. On considère l'algorithme glouton qui sélectionne les titres dans l'ordre de taille m_i croissante jusqu'à ce que la capacité de stockage maximale soit atteinte. Cet algorithme fournit-il une solution optimale ? Le prouver ou fournir un contre-exemple.

Q 7.2 Supposons maintenant que vous souhaitez maximiser *la mémoire occupée dans votre lecteur MP3 par les titres que vous aurez téléchargés*. On considère l'algorithme glouton qui sélectionne les titres dans l'ordre de taille m_i décroissante jusqu'à ce que la capacité de stockage maximale soit atteinte. Cet algorithme fournit-il une solution optimale ? Le prouver ou fournir un contre-exemple.

Exercice 8 – Ordonnancement de tâches sur une machine

On s'intéresse au problème d'ordonnancement de tâches sur une machine qui consiste à définir les dates de début d'exécution d'un ensemble de n tâches J_i, $i = 1, \cdots, n$ caractérisées par une durée opératoire p_i et une date d'échéance d_i à laquelle la tâche devrait idéalement être terminée. La machine ne peut exécuter qu'une tâche à la fois. Si l'on note s_i la date de début d'exécution de la tâche J_i, le retard de J_i sera défini par $l_i = \max\{0, s_i + p_i - d_i\}$. Si $l_i > 0$, la tâche J_i est en retard.

Le but de l'exercice est de déterminer un ordonnancement minimisant la valeur du retard maximal $L = \max_i l_i$ pour l'ensemble des tâches.

Q 8.1 Déterminer la valeur de L pour un ensemble de $n = 6$ tâches pour les données suivantes : $p = (3, 2, 1, 4, 3, 2)$, $d = (6, 8, 9, 9, 14, 15)$ et $s = (5, 1, 0, 11, 8, 3)$.

Q 8.2 Différents algorithmes gloutons peuvent être décrits selon le choix de l'ordre fixé sur les tâches pour leur exécution. Proposer un contre-exemple pour montrer que l'ordre des durées opératoires croissantes n'est pas optimal.

Q 8.3 Les dates d'échéance doivent donc être prises en compte pour définir un ordre sur les tâches. On définit l'urgence de la tâche J_i comme étant la valeur $d_i - p_i$ et on s'intéresse à l'algorithme glouton qui ordonnance les tâches selon l'ordre des urgences croissantes.
Cet algorithme est-il optimal ?

Q 8.4 Montrer que les ordonnancements ne contenant aucune période d'inactivité pour la machine sont dominants.

On considère l'algorithme glouton suivant que l'on appellera EDD :

Ré-indexer les tâches de façon à ce que $d_1 \leq d_2 \leq \cdots d_n$;
$t \leftarrow 0$;
pour *i variant de 1 à n* **faire**
  Affecter la tâche J_i à l'intervalle $[t, t + p_i]$;
  $s_i \leftarrow t$;
  $C_i \leftarrow s_i + p_i$;
  $t \leftarrow C_i$;
pour *i variant de 1 à n* **faire**
  Afficher les intervalles $[s_i, C_i]$;

Q 8.5 Dérouler l'algorithme EDD sur l'exemple de la question 1 et calculer la valeur du retard maximal pour la solution trouvée.

On suppose dans la suite que les tâches sont indexées dans l'ordre de leurs dates d'échéance croissantes. Etant donné un ordonnancement, une *inversion* est une paire de tâches J_i et J_j pour laquelle $d_i < d_j$ et J_j est ordonnancée avant J_i.

Q 8.6 Montrons que la permutation d'une paire de tâches inversées consécutives dans un ordonnancement n'entraîne pas l'augmentation de son retard maximal.

Q 8.7 Montrons qu'il existe un ordonnancement optimal ne possédant ni inversion, ni temps d'inactivité.

Q 8.8 Que peut-on dire de la valeur de retard maximal pour les ordonnancements ne possédant pas d'inversions ni de temps d'inactivité ?

Q 8.9 Montrer que l'ordonnancement renvoyé par l'algorithme EDD est optimal.

Exercice 9 – Codage de Huffman

Q 9.1 Le tableau ci-dessous indique la fréquence de chaque lettre dans la langue française (cette fréquence a été obtenue après analyse de plusieurs livres et encyclopédies). Construire l'arbre de Huffman correspondant à ces fréquences.

Lettre	Fréquence	Lettre	Fréquence
A	8.4	N	7.1
B	1.1	O	5.3
C	3.0	P	3.0
D	4.2	Q	1.0
E	17.3	R	6.6
F	1.1	S	8.1
G	1.3	T	7.1
H	0.9	U	5.7
I	7.3	V	1.3
J	0.3	W	0.1
K	0.1	X	0.4
L	6.0	Y	0.3
M	3.0	Z	0.1

Exercice 10 – Deviner la valeur d'une carte

On cherche à concevoir une stratégie pour minimiser l'espérance du nombre de questions posées dans le jeu suivant. On dispose d'un paquet de 45 cartes composé d'une carte de valeur 1, de deux cartes de valeur 2, de trois cartes 3, et ainsi de suite jusqu'à neuf 9. Quelqu'un tire une carte au hasard dans le paquet mélangé, et on cherche à identifier la valeur de la carte en posant des questions dont la réponse est oui ou non.

Q 10.1 Proposer une méthode optimale, et indiquer l'espérance du nombre de questions posées.

Algorithmique – 3I003

Programmation dynamique

1. Programmation dynamique

Exercice 1 – Sous-séquence de plus grande somme (exercice de base)

On dispose d'un tableau d'entiers *relatifs* de taille n. On cherche à déterminer la suite d'entrées consécutives du tableau dont la somme est maximale. Par exemple, pour le tableau $T = [5, 15, -30, 10, -5, 40, 10]$, la somme maximale est 55 (somme des éléments $[10, -5, 40, 10]$).

Q 1.1 Dans un premier temps, on s'intéresse uniquement à la valeur de la somme de cette sous-séquence.

a. Quelle est la sous-structure optimale dont on a besoin pour résoudre ce problème ?

b. Caractériser (par une équation) cette sous-structure optimale.

c. En déduire la valeur de la somme maximale.

d. Ecrire un algorithme de programmation dynamique pour calculer cette somme.

e. Quelle est la complexité de cet algorithme ?

Q 1.2 Modifier l'algorithme pour qu'il renvoie également les indices de début et de fin de la somme. Sa complexité est-elle modifiée ?

Exercice 2 – Découpe d'une corde (exercice d'entraînement)

Le magasin "A la vieille campeuse" achète des cordes d'escalade de longueur n et les découpe (soigneusement) en cordes plus petites pour les vendre à ses clients. On souhaite déterminer un découpage optimal pour maximiser le revenu, sachant que les prix de ventes p_i d'une corde de i mètres sont donnés. Par exemple, supposons qu'on dispose d'une corde de $n = 10$ mètres, avec les prix de ventes indiqués dans le tableau suivant :

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	26

Q 2.1 Dans un premier temps, on suppose que chaque découpe est gratuite. On définit la *densité* d'une corde de longueur i comme étant le rapport p_i/i, c'est-à-dire son prix au mètre. Une stratégie gloutonne naturelle consisterait à découper une première corde de longueur i dont la densité soit maximale. On continuerait ensuite en appliquant la même stratégie sur la portion de corde restante de longueur $n - i$. Montrer que cette stratégie ne conduit pas toujours à un découpage optimal.

Q 2.2 On cherche à établir un algorithme de programmation dynamique pour déterminer le revenu de ventes maximal que l'on peut obtenir pour une corde, en supposant toujours que chaque découpe est gratuite.

a. Quelle est la sous-structure optimale dont on a besoin pour résoudre ce problème ?

b. Caractériser (par une équation) cette sous-structure optimale.

c. En déduire le revenu de ventes maximal qu'on peut tirer d'une corde.

d. Ecrire un algorithme de programmation dynamique pour calculer ce revenu.

e. Quelle est la complexité de cet algorithme ?

f. Appliquer l'algorithme à l'exemple.

Q 2.3 Modifier l'algorithme précédent afin qu'il renvoie également la découpe optimale. Appliquer à l'exemple.

Q 2.4 On suppose maintenant que chaque découpe a un coût c (le même pour chaque découpe). Le revenu associé à un découpage particulier est alors la somme des prix de ventes moins les coûts de découpe. Reprendre la question 2 dans ce nouveau cadre. Le découpage optimal est-il le même ?

Exercice 3 – Alignement de séquences (exercice d'entraînement)

Un brin d'ADN (acide désoxyribonucléique) est caractérisé par la suite de molécules, appelées *bases*, qui la composent. Il existe quatre bases différentes : deux purines (l'adénine A, la guanine G), ainsi que deux pyrimidines (la thymine T et la cytosine C). Une séquence d'ADN peut donc être modélisée par un mot sur un alphabet à quatre lettres ($AGTC$).

Pouvoir comparer l'ADN de deux organismes est un problème fondamental en biologie. L'*alignement* entre deux séquences ADN similaires permet d'observer leur degré d'apparentée et d'estimer si elles semblent ou non homologues, c'est-à-dire si elles descendent ou non d'une même séquence ancestrale commune ayant divergé au cours de l'évolution. Un alignement met en évidence les :

— identités entre les 2 séquences,

— insertions ou délétions (pertes de matériel génétique) survenues dans l'une ou l'autre des séquences.

Notons X et Y deux séquences ADN. Pour aligner ces deux séquences, on insère des espaces

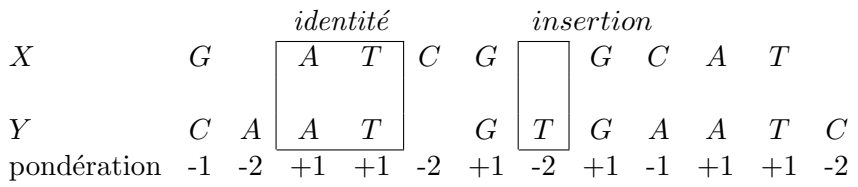


FIGURE 21 – Alignement de deux séquences ADN.

entre deux bases de chaque séquence, à des emplacements arbitraires (extrémités comprises), de sorte que les deux séquences résultantes (notées X' et Y') aient la même longueur (mais à un emplacement donné, il ne peut pas y avoir un espace pour chacune des deux séquences). Nous considérons le modèle d'évolution simplifié suivant : à chaque emplacement j on attribue une *pondération*

— $+1$ si $X'[j] = Y'[j]$ (alors aucun des deux n'est un espace),

— -1 si $X'[j] \neq Y'[j]$ et aucun des deux n'est un espace,

— -2 si $X'[j]$ ou $Y'[j]$ est un espace.

Le poids de l'alignement est la somme des poids des emplacements.

Q 3.1 Donner un algorithme de programmation dynamique déterminant la valeur de poids maximal de deux séquences ADN. Déterminez sa complexité dans le pire cas.

Q 3.2 Modifier l'algorithme précédent afin qu'il renvoie également l'alignement optimal.

Exercice 4 – Parenthèsage (exercice d'entraînement)

Votre journal préféré a commencé à publier des petites énigmes de la forme :

Parenthéser $7 + 1 \times 6$ de façon à maximiser le résultat.	Parenthéser $0.1 \times 0.1 + 0.1$ de façon à maximiser le résultat.
Mauvaise réponse : $7+(1\times6) = 7+6 = 15$ Bonne réponse : $(7 + 1) \times 6 = 8 \times 6 = 48$	Mauvaise réponse : $0.1 \times (0.1 + 0.1) = 0.1 \times 0.2 = 0.02$ Bonne réponse : $(0.1\times0.1)+0.1 = 0.01\times0.1 = 0.11$

Ces énigmes sont assez faciles à résoudre à la main, mais votre journal en publie des bien plus difficiles, dont les solutions peuvent bien sûr contenir des parenthèses imbriquées, comme par exemple : $((3 \times 2) \times (0.5 + 2)) \times (3 + 1)$. Pour ne pas vous ennuyer à résoudre ces énigmes, mais tout de même impressionner vos amis, vous décidez d'implémenter un algorithme pour résoudre ces énigmes. L'entrée de l'algorithme est une liste $x_1, o_1, x_2, o_2, \ldots, x_{n-1}, o_{n-1}, x_n$ de n nombres réels positifs $x_1, \ldots, x_n$ et de $n-1$ opérateurs $o_1, \ldots, o_{n-1}$. Chaque opérateur o_i est soit une addition ($+$) soit une multiplication ($\times$).

Le but de cet exercice est de concevoir un algorithme de programmation dynamique retournant une manière de placer des parenthèses dans l'expression donnée de façon à maximiser le résultat.
On notera $P[i, j]$ la valeur maximale, après parenthésage, de l'expression $x_i, o_i, \ldots, o_{j-1}, x_j$.

Q 4.1 Donnez les valeurs de i et j pour lesquelles $P[i, j]$ désigne la valeur maximale, après parenthésage, de l'expression $x_1, o_1, x_2, o_2, \ldots, x_{n-1}, o_{n-1}, x_n$?

Q 4.2 Un opérateur est dit *central* s'il ne se trouve pas entre deux parenthèses qui se correspondent. On admet qu'il existe pour toute expression un parenthésage la maximisant et ayant un seul opérateur central. Soit o_k l'opérateur central de l'expression $x_i, o_i, \ldots, o_{j-1}, x_j$ après un parenthésage optimal (i.e. on a alors des parenthèses entre x_i et x_k puis des parenthèses entre x_{k+1} et x_n). Expliquez pourquoi on a :

$$P[i, j] = P[i, k]o_kP[k + 1, j].$$

Q 4.3 Combien d'opérateurs faut-il examiner afin de trouver l'opérateur central du parenthésage optimal de l'expression $x_i, o_i, \ldots, o_{j-1}, x_j$? En déduire une relation de récurrence pour $P[i, j]$.

Q 4.4 Écrire le programme dynamique associé à cette relation de récurrence. On s'intéressera uniquement à obtenir la valeur maximale de l'expression après parenthésage (et non le parenthésage). Vous pouvez dans votre algorithme écrire directement les nombres sous leur forme x_i et les opérateurs sous leur forme o_k (inutile de transformer un caractère en un nombre ou en un opérateur).
Quelle est la complexité de votre algorithme ?

Q 4.5 Comment peut-on modifier le programme écrit à la question précédente de façon à obtenir le parenthésage optimal (on ne demande pas de réécrire le programme, mais d'expliquer

en quelques phrases comment obtenir ce parenthésage optimal).

Q 4.6 Ce programme dynamique retourne-t-il toujours une solution optimale si les nombres x_i peuvent prendre une valeur négative ?

Exercice 5 – Jeu à deux joueurs (exercice d'approfondissement)

Considérons une rangée de n pièces de valeurs $v_1, \ldots, v_n$ et le jeu suivant à deux joueurs : à chaque tour, le joueur peut prendre soit la pièce la plus à gauche, soit la pièce la plus à droite. Le jeu s'arrête quand il n'y a plus de pièce. On cherche à calculer le gain optimal du joueur qui commence ; on considérera que les deux joueurs adoptent une stratégie qui maximise leur profit final. Par exemple, pour 5, 3, 7, 11 le gain optimal pour celui qui commence est 16(11 + 5) ; pour 8, 15, 3, 7 ce sera 22(7 + 15).

Q 5.1 Indiquer une rangée de pièces pour laquelle il n'est pas optimal pour le premier joueur de commencer par prendre la pièce de plus grande valeur. Autrement dit, cette stratégie gloutonne n'est pas optimale.

Q 5.2 Donner un algorithme en $O(n^2)$ pour calculer une stratégie optimale pour le premier joueur. Etant donnée la rangée initiale, l'algorithme calculera en $O(n^2)$ les informations nécessaires, et ensuite le premier joueur pourra déterminer en $O(1)$ le choix optimal à chaque tour en consultant les données précalculées.

Exercice 6 – Playoffs NBA (exercice d'approfondissement)

Les playoffs NBA (appelés séries éliminatoires au Canada) sont les séries éliminatoires de la National Basketball Association (NBA). Une série éliminatoire est une série de matchs deux équipes, au terme de laquelle l'une des deux équipes est éliminées. Chaque série se joue au meilleur d'une série de sept matchs, c'est-à-dire que la première équipe qui gagne 4 matchs remporte la série.

Plus généralement, considérons deux équipes A et B qui s'affrontent dans une série de matchs jusqu'à ce que l'équipe A remporte i victoires ou que l'équipe B remporte j victoires. Supposons que la probabilité de victoire de A sur un match (contre l'équipe B) est p, et que la probabilité de défaite de A est donc $q = 1 - p$ (il n'y a pas de match nul). Soit $P(i, j)$ la probabilité que A remporte la série (i.e. A a remporté i victoires et B moins de j victoires.

Q 6.1 Etablir une relation de récurrence pour $P(i, j)$ qui puisse être utilisée dans un algorithme de programmation dynamique.

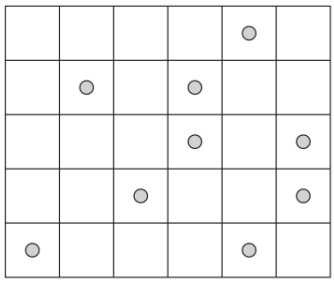
Q 6.2 Déterminer la probabilité que l'équipe A remporte une série au meilleur des sept matchs si $p = 0.4$ (ceci signifie que A gagne si elle est la première équipe à gagner 4 matchs).

Q 6.3 Ecrire un algorithme de programmation dynamique pour calculer la probabilité de gagner une série au meilleur des n matchs (n impair). Quel est sa complexité ?

Exercice 7 – Robot collecteur de pièces (exercice de base)

Des pièces sont placées sur les cases d'un échiquier $\ell \times c$ (ℓ lignes et c colonnes). Il y a au plus une pièce par case. Un robot, positionné sur la case en haut à gauche de l'échiquier, va collecter

le plus de pièces possibles et les placer sur la case en bas à droite. A chaque pas, le robot peut se déplacer d'une case vers la droite ou vers le bas. Quand le robot passe sur une case avec une pièce, il prend la pièce. Le but de cet exercice est de concevoir un algorithme pour collecter un nombre maximum de pièces.



Q 7.1 Dans un premier temps, on s'intéresse uniquement au nombre maximum de pièces qu'on peut collecter sans chercher à identifier la trajectoire correspondante.

a. Quelle est la sous-structure optimale dont on a besoin pour résoudre ce problème ?

b. Caractériser (par une équation) cette sous-structure optimale.

c. En déduire le nombre maximum de pièces qu'on peut collecter.

d. Ecrire un algorithme de programmation dynamique pour calculer ce nombre.

e. Quelle est la complexité de cet algorithme ?

Q 7.2 Modifier l'algorithme précédent afin qu'il renvoie également la trajectoire optimale.

Dans les questions qui suivent, on cherche à reformuler et résoudre le problème comme la recherche d'un plus long chemin dans un graphe.

Q 7.3 Pour l'échiquier ci-dessus, tracer un graphe valué avec un sommet source s et un sommet puits t, tel que le problème de collecte de pièces revient à déterminer un plus long chemin de s à t. Plus généralement, indiquer le nombre n de sommets et le nombre m d'arcs en fonction de ℓ et de c.

Q 7.4 En s'inspirant d'un algorithme vu en cours dont on rappelera le nom, donner le *principe* (sans entrer dans les détails) d'un algorithme de calcul d'un plus long chemin dans un graphe sans circuit. Appliquer cet algorithme sur le graphe de la question 1. Quel est le plus long chemin ?

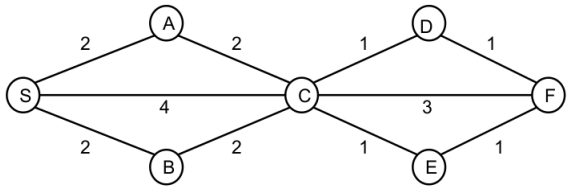
Q 7.5 Quelle est la complexité de l'algorithme précédent en fonction de n et de m ? En déduire la complexité de l'algorithme qui, partant d'un échiquier sur lequel des pièces sont positionnées, détermine un itinéraire optimal pour collecter un nombre maximum de pièces. Cette complexité sera formulée en fonction de ℓ et de c.

Exercice 8 – Nombre d'arêtes d'une plus courte chaîne (exercice d'entraînement)

Quand il y a plusieurs plus courtes chaînes entre deux sommets (avec des arêtes de différentes longueurs), la chaîne la plus adéquate parmi celles-ci est souvent *celle avec le plus petit nombre d'arêtes*. Par exemple, si les sommets représentent des villes et les arêtes représentent des liaisons aériennes entre les villes (les longueurs représentant les durées des vols), il peut y avoir plusieurs façons, optimales en terme de durée, de se rendre d'une ville s à une ville t. La meilleure alternative est alors celle qui comporte le moins de correspondance. On définit :

best[u] = nombre minimum d'arêtes dans une plus courte chaîne de s à u.

Dans l'exemple ci-dessous, les valeurs best[u] aux sommets $u = S, A, B, C, D, E, F$ sont $0, 1, 1, 1, 2, 2, 3$ respectivement.



Q 8.1 Donner le principe d'un algorithme efficace pour résoudre le problème suivant :

Instance : un graphe $G = (V, E)$; une longueur $l_e \geq 0$ pour tout $e \in E$; un sommet de départ $s \in V$.
Sortie : les valeurs best[u] pour *tous* les sommets $u \in V$.

Q 8.2 Appliquer cet algorithme à l'exemple ci-dessus.

Q 8.3 On suppose maintenant que non seulement les arêtes, mais également les sommets sont valués (dans l'exemple du tranport aérien, la valeur d'un sommet peut représenter la durée de la correspondance). La valeur d'une chaîne est la somme des valeurs des arêtes *et des sommets* qui la compose. Comment adapter l'algorithme précédent à ce nouveau cadre ? Que devient la complexité ?

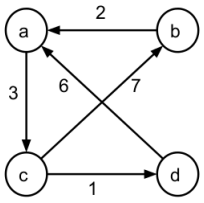
Exercice 9 – Plus courts chemins entre toute paire de sommets (exercice d'entraînement)

On vise dans cet exercice à concevoir un algorithme de programmation dynamique pour calculer les valeurs $d(i, j)$ des plus courts chemins pour toutes les paires de sommets i, j, dans un graphe orienté où les sommets sont indicés de 1 à n et où chaque arc (i, j) est muni d'une longueur ℓ_{ij}. Soit $d(i, j, k)$ la valeur d'un plus court chemin de i à j dont tous les sommets intermédiaires sont dans $\{1, \ldots, k\}$.

Q 9.1 On cherche à établir un algorithme programmation dynamique pour ce problème.

- **a.** Indiquer la valeur de $d(i, j, k)$ pour $k = 0$.
- **b.** Donner l'équation de récurrence caractérisant $d(i, j, k)$.
- **c.** En déduire la longueur $d(i, j)$ d'un plus court chemin entre i et j.
- **d.** Ecrire un algorithme de programmation dynamique pour calculer les longueurs des plus courts chemins pour tous les couples de sommets.
- **e.** Quelle est la complexité de cet algorithme ?

Q 9.2 Appliquer l'algorithme au graphe ci-dessous.



Exercice 10 – Résolution d'un système d'inéquations (exercice d'entraînement)

On considère un ensemble $\mathcal{X}$ de n variables réelles x_1, x_2, …, x_n et m contraintes où chaque contrainte $k \in \{1, \cdots, m\}$ est de la forme suivante :

$$x_{j_k} - x_{i_k} \leq b_k$$

où x_{j_k} et x_{i_k} sont des variables de $\mathcal{X}$ et b_k est un réel.
On cherche à déterminer si ce système de contraintes possède une solution admissible, c'est-à-dire si l'on peut donner des valeurs aux variables de manière à ce que toutes les contraintes soient vérifiées.
Pour résoudre ce problème, on construit un graphe contenant :

— n sommets numérotés de 1 à n (un sommet par variable) ;

— un sommet s ;

— n arcs (s, l) $(l \in \{1, \cdots, n\})$;

— un arc (i_k, j_k) pour chaque contrainte $x_{j_k} - x_{i_k} \leq b_k$ $(k \in \{1, \cdots, m\})$

Il y a donc $n + 1$ sommets et $n + m$ arcs. Les arcs sont valués de la manière suivante : les arcs (s, l) ont une valeur 0, l'arc (i_k, j_k) associé à la contrainte $x_{j_k} - x_{i_k} \leq b_k$ a la valeur b_k.

Q 10.1 Dans cette question, on s'intéresse au système de 4 variables et 5 contraintes suivant :

$$\begin{aligned} x_4 - x_3 &\leq 5 \\ x_3 - x_1 &\leq 8 \\ x_1 - x_4 &\leq -10 \\ x_1 - x_2 &\leq -11 \\ x_2 - x_3 &\leq 2 \end{aligned}$$

a. Représentez le graphe G associé au système ci-dessus.

b. Appliquez un algorithme de recherche de plus courts chemins pour déterminer des plus courts chemin d'origine s aux sommets 1, 2, 3 et 4 dans G.
On justifiera le choix de l'algorithme et on détaillera les étapes du déroulement de l'algorithme (on fournira en particulier les changements dans l'arborescence courante et l'arborescence finale obtenue). Que constatez-vous ?

c. Pouvez-vous en déduire que ce système de contraintes possède une solution admissible ? Si oui, donnez un exemple de valeurs aux variables. Si non, donnez un ensemble de contraintes incompatibles.

Q 10.2 On s'intéresse maintenant à un système quelconque à n variables et m contraintes et à son graphe associé (on ne considère donc plus le système donné dans la question 1).

a. Enoncez une condition nécessaire sur le graphe associé pour que le système possède une solution admissible. Prouvez que cette condition est bien une condition nécessaire.

b. Montrez que cette condition est suffisante.

Exercice 11 – Ensembles indépendants dans des arbres (exercice d'entraînement)

Un sous-ensemble de noeuds dans un graphe $G = (V, E)$ est dit *indépendant* s'il n'existe pas d'arête entre eux. Trouver le plus grand ensemble indépendant dans un graphe est un problème difficile. Par contre, si le graphe est un arbre, c'est un sous-problème plus facile. On cherche donc l'ensemble indépendant de plus grande taille dans un arbre A.
On note $r(A)$, la racine de A et $fils(A)$, la liste des fils de A. On pourra également noter $A(s)$, le sous-arbre de A dont la racine est s.

Dans un premier temps, on s'intéresse uniquement à la taille maximale d'un sous-ensemble.

Q 11.1 Montrer que si $v \in V$ est une feuille, il existe un ensemble indépendant maximum qui contient v.

Q 11.2 En vous appuyant sur la propriété de la question précédente, concevoir un algorithme glouton pour déterminer un ensemble indépendant maximum dans un arbre. Quelle est la complexité de cet algorithme ?

On suppose désormais que chaque sommet v de l'arbre a un poids w_v. On cherche à déterminer la valeur d'un ensemble indépendant *de poids maximum*.

Q 11.3 Dans cette question, on va concevoir un algorithme de programmation dynamique pour résoudre ce problème.

a. Quelle est la sous-structure optimale dont on a besoin pour résoudre ce problème ?

b. Caractériser (par une équation) cette sous-structure optimale.

c. En déduire le poids maximum d'un ensemble indépendant de A.

d. Ecrire un algorithme de programmation dynamique pour calculer cette valeur.

e. Quelle est la complexité de cet algorithme ?

Exercice 12 – Sac-à-dos, avec répétitions (exercice de base)

Lors du cambriolage d'une bijouterie, le voleur s'aperçoit qu'il ne peut pas tout emporter dans son sac-à-dos... Son sac peut supporter, au maximum, P kilos de marchandises (on supposera P entier). Or, dans cette bijouterie, se trouvent n objets différents, chacun en quantité illimitée. Chaque objet x_i a un poids p_i et une valeur v_i. Quelle combinaison d'objets, transportable dans le sac, sera la plus rentable pour le voleur ?
Exemple : pour $P = 10$ et les objets ci-dessous, le meilleur choix est de prendre un x_1 et deux x_4, ce qui fait un sac à 48 €.

objet	poids	valeur
x_1	6	30 €
x_2	3	14 €
x_3	4	16 €
x_4	2	9 €

Q 12.1 Dans un premier temps, on s'intéresse uniquement à la valeur maximale que pourra emporter le voleur.

a. Quelle est la sous-structure optimale dont on a besoin pour résoudre ce problème ?

b. Caractériser (par une équation) cette sous-structure optimale.

c. En déduire le montant maximal des objets volés.

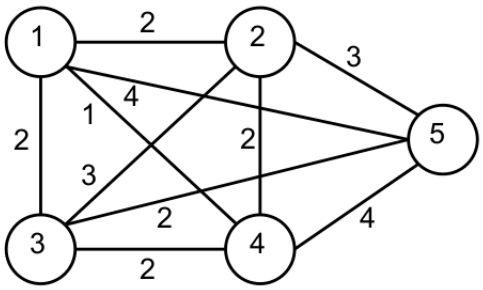
d. Ecrire un algorithme de programmation dynamique pour calculer cette valeur maximale.

e. Quelle est la complexité de cet algorithme ?

Q 12.2 Modifier l'algorithme pour qu'il renvoie également la combinaison d'objets volés. Sa complexité est-elle modifiée ?

Exercice 13 – Voyageur de commerce (exercice d'approfondissement)

Dans le problème du voyageur de commerce, on dispose d'une matrice (ici supposée symétrique) d des distances entre n villes indicées de 1 à n (d_{ij} et d_{ji} représentant la distance entre la ville i et la ville j), et le but est, en partant de 1, de visiter chaque ville exactement une fois et de revenir en 1 en minimisant la distance totale parcourue. Par exemple, dans le graphe ci-dessous, la distance totale parcourue pour la tournée $(1, 2, 3, 4, 5, 1)$ est 15, alors que la distance totale parcourue pour la tournée $(1, 3, 5, 2, 4, 1)$ est seulement 10.



Q 13.1 Donner la matrice d pour le graphe de l'exemple.

Q 13.2 Combien y a-t-il de tournées possibles pour n villes ?

Q 13.3 On cherche à établir un algorithme programmation dynamique pour déterminer la valeur d'une tournée optimale.

a. Quelle est la sous-structure optimale dont on a besoin pour résoudre ce problème ?

b. Caractériser (par une équation) cette sous-structure optimale.

c. En déduire la valeur d'une tournée optimale.

d. Ecrire un algorithme de programmation dynamique pour calculer cette valeur.

e. Quelle est la complexité de cet algorithme ?