# Travaux Dirigés

**Algorithmique**

28 septembre 2020

**Exercice 1** Calcul du plus grand commun diviseur (PGCD).

**Question 1** Donner une description du "problème du plus grand commun diviseur" en précisant la question et la donnée, et écrivez un algorithme pour le résoudre.

PGCD

**Donnée :** Deux entiers non négatifs $m$ et $n$ tels que $m \geq n$

**Question :** Quel est le plus grand commun diviseur de $m$ et $n$ ?

**Question 2** L'algorithme `PGCD_brute-force` calcule PGCD$(m, n)$, montrer qu'il est correct.

**Algorithme :** `PGCD_brute-force`
**Données :** Deux entiers $m$ et $n$ tels que $m \geq n$
**Résultat :** le plus grand commun diviseur de $m$ et $n$
**si** $n = 0$ **alors**
  **retourner** 0;
**sinon**
  $t \leftarrow n$;
  **tant que** $m \bmod t \neq 0$ *ou* $n \bmod t \neq 0$ **faire**
    $t \leftarrow t - 1$;
  **retourner** $t$;

**Question 3** Etant donné $n$, donner la valeur de $m$ qui correspond au *meilleur des cas* pour le temps de calcul. Combien d'opérations primitives sont effectuées par `PGCD_brute-force` dans ce cas ?

**Question 4** Etant donné $n$, donner une valeur de $m$ qui correspond à un *pire des cas* pour le temps de calcul. Combien d'opérations primitives sont effectuées par `PGCD_brute-force` dans ce cas ?

---

**Exercice 2** Algorithme d'Euclide.

L'algorithme `Euclide` calcule $\mathrm{PGCD}(m, n)$ :

**Algorithme :** `Euclide`
**Données :** Deux entiers $m$ et $n$ tels que $m \geq n$
**Résultat :** le plus grand commun diviseur de $m$ et $n$
**tant que** $n > 0$ **faire**
  $r \leftarrow m \mod n$;
  $m \leftarrow n$;
  $n \leftarrow r$;
**retourner** $m$;

**Question 5** On veut montrer que `Euclide` est correct, quel invariant de boucle doit-on utiliser ?

**Question 6** Que reste-t'il à démontrer ? (l'idée de la démonstration suffit)

**Question 7** Est-ce que `Euclide` peut faire plus d'itérations que `PGCD_brute-force` ?

**Question 8** Montrez que si $m \geq n$, alors $m \bmod n < m/2$

**Question 9** Utilisez le résultat précédent pour calculer combien d'opérations primitives sont effectuées par `Euclide` dans le pire des cas.

---

**Exercice 3** Utiliser la complexité algorithmique.

**Question 10** Soit deux algorithmes $A$ et $B$ de complexité respective $\mathcal{O}(n^2)$ et $\mathcal{O}(2^n)$.

(10.a) Lors de son exécution (sur un ordinateur donné), l'algorithme $A$ permet de traiter des données de taille $n_o$ en $1mn$. Quelle est la taille des données que l'on pourra traiter en $1mn$ avec un ordinateur 100 fois plus rapide ?

(10.b) Même question pour l'algorithme $B$ : il permet de traiter des données de taille $n_o$ en $1mn$ sur un ordinateur donné. Quelle est la taille des données que l'on pourra traiter en $1mn$ avec un ordinateur 100 fois plus rapide ?

Remarque : négligez les constantes pour répondre aux questions.

**Question 11** Soit deux algorithmes $A$ et $B$ pour résoudre un même problème.

La complexité de l'algorithme $A$ est en $\Theta(n^2)$ et son exécution pour des données de taille $n = 100$ dure $1s$.

La complexité de l'algorithme $B$ est en $\Theta(n \log n)$ et son exécution pour des données de taille $n = 100$ dure $20s$.

(11.a) Quel algorithme faut-il choisir pour une donnée de taille $n = 1000$ ?

(11.b) Même question si $n = 10000$.

Remarque : négligez les constantes additives pour répondre aux questions.

---

**Exercice 4** Notations asymptotiques.

**Question 12** Lesquelles des assertions suivantes sont vraies ?

1. $1000 \in \mathcal{O}(1)$
2. $n^2 \in \Omega(n^3)$
3. $n^3 \in \Omega(n^2)$
4. $2^{n+1} \in \mathcal{O}(2^n)$
5. $(n+1)^2 \in \Theta(n^2)$
6. $n^3 + 12 \in \mathcal{O}(2^n)$
7. $\sqrt{n} \in \Theta(n)$
8. $\pi^{32} \in \mathcal{O}(1)$
9. $(n-3)(n+1) \in \mathcal{O}(n^3)$
10. $n^3 m^2 \in \mathcal{O}(m^6)$
11. $2^{(n-12)} \in \Omega(2^n)$
12. $3^n \in \mathcal{O}(2^n)$
13. $n^{3.14} \in \Theta(n^3)$
14. $n^{3.14} \in \mathcal{O}(n^3)$
15. $n^{3.14} \in \Omega(n^3)$
16. $(n+1) * (\frac{n}{2} - 1) \in \Theta(n^2)$
17. $n^3 + 3n^2 + n + 2017 \in \mathcal{O}(n^3)$
18. $n^2 * n^3 \in \mathcal{O}(n^3)$
19. $\frac{1}{2}n^2 - 3n \in \Theta(n^2)$
20. $\frac{1}{4}n^2 * \Omega(n) \in \Omega(n^3)$
21. $\log_2(2n) \in \Theta(\log n)$
22. $\log_2(n^2) \in \Theta(\log n)$
23. $(\log_2(n))^2 \in \Theta(\log n)$
24. $3^n \in \mathcal{O}(2^n)$
25. $2^{n+3} \in \mathcal{O}(2^n)$
26. $2^{3n} \in \mathcal{O}(2^n)$
27. Si $g(n) \in \mathcal{O}(f(n))$ alors $2^{g(n)} \in \mathcal{O}(2^{f(n)})$
28. Si $g(n) \in \Theta(f(n))$ alors $|g(n) - f(n)| \in \Theta(1)$
29. Si $g(n) \in \Theta(f(n))$ alors $\frac{g(n)}{f(n)} \in \Theta(1)$
30. $\Theta(f(n)) \subseteq \mathcal{O}(f(n))$

**Question 13** Est-ce que les propositions suivantes sont vraies pour tout $f, g$ ? Pour chacune, prouvez-le ou trouvez un contre-exemple (si un contre-exemple est nécessaire, un schéma sans analyse numérique sera suffisant).

(13.a) $f(n) \in \Omega(g(n))$ alors $g(n) \in \mathcal{O}(f(n))$

(13.b) $f(n) \in \mathcal{O}(g(n))$ ou $g(n) \in \mathcal{O}(f(n))$

**Question 14** Prouvez que $\mathcal{O}(\max(f(n), g(n)) = \mathcal{O}(f(n) + g(n))$.

---

**Exercice 5** Analyse de complexité

**Question 15** Calculer la complexité dans le pire des cas et dans le meilleur des cas de l'algorithme suivant :

```
Algorithme : rechercheDicho(T)
Données : un tableau trié T de taille n et un entier x
Résultat : le premier indice i tel que T[i] = x, ou bien −1 si x ∉ T
i, j, m : entier;
début
    i := 1;
    j := n;
    tant que i < j faire
        m := ⌊(i+j)/2⌋;
        si x = T[m] alors
            retourner m;
        si x < T[m] alors
            j := m;
        sinon
            i := m + 1;
    retourner −1;
```

Préciser à quoi correspond chaque cas.

**Question 16** Que retourne les fonctions ci-dessous ? Evaluer l'ordre de grandeur de leur complexité.

```
Algorithme : f1(n)
Données : un entier n ≥ 0
Résultat : ???
r, i, j : entier;
début
    r := 0;
    pour i allant de 1 à n faire
        pour j allant de i + 1 à n faire
            r := r + 1;
    retourner r;
```

```
Algorithme : f2(n)
Données : un entier n ≥ 0
Résultat : ???
i, r : entier;
début
    i := n;
    r := 0;
    tant que i > 1 faire
        r := r + 1;
        i := i/2 (!!division entière!!);
    retourner r;
```

**Question 17** Quelle est la complexité de la boucle suivante (sachant que la somme des $n$ premiers entiers est égale à $\frac{1}{2}n(n+1)$ – appelé $n$-ème nombre triangulaire ; et la somme des $n$ premiers nombres triangulaires est égale à

$\frac{1}{6}n(n+1)(n+2))$ :

```
1 début
2   pour i allant de 1 à n − 1 faire
3     pour j allant de i + 1 à n faire
4       pour k allant de 1 à j faire
5         Instruction;
6   retourner r;
```

---

**Exercice 6** Algorithmes recursifs

**Question 18** Pour chacune des fonction suivantes, posez la forme récursive de la complexité :

$$T(x) = \begin{cases} \dots & \text{si } x \leq 1 \\ \dots & \text{si } x > 1 \end{cases}$$

et calculez l'ordre de grandeur ($\Theta$) de $T(x)$, en utilisant la méthode par substitution.

(18.a) rec1(x) = 1 si $x \leq 1$, 2×rec1(x-1) sinon.

(18.b) rec2(x) = 1 si $x \leq 1$, rec2(x-1) + rec2(x-1) sinon.

**Question 19** On veut caculer la complexité des différents algorithmes en nombre de multiplications, on suppose donc que multiplier deux entiers se fait en $\mathcal{O}(1)$.

Soient les algorithmes **`expo1`** et **`expo2`** ci-dessous :

```
1 Algorithme : expo1(y,x)
  Données : deux entiers y et x
  Résultat : y^x
2 res, i : entier;
3 début
4   res ← 1;
5   pour i allant de 1 à x faire
6     res ← res * y;
7   retourner res;
```

```
1 Algorithme : expo2(y,x)
  Données : deux entiers y et x
  Résultat : y^x
2 res : entier;
3 début
4   res ← 0;
5   si x = 1 alors
6     retourner y;
7   sinon
8     res ← expo2(y, x/2);
9     retourner res*res;
```

On considère tout d'abord que $x$ est nécessairement une puissance de 2.

(19.a) Quelle est la taille de la donnée (on suppose que $x$ et $y$ sont du même ordre de grandeur) ?

(19.b) Quelle est la complexité de l'algorithme **`expo1`** en fonction de $x$ ?

(19.c) Quelle est la complexité de l'algorithme **`expo1`** en fonction de la taille de la donnée, est-elle polynomiale ?

(19.d) Montrez par la méthode par substitution que l'algorithme **`expo2`** est en $\mathcal{O}(\log_2 x)$.

(19.e) Quelle est la complexité de l'algorithme **`expo2`** en fonction de la taille de la donnée ?

(19.f) Que concluez-vous pour ces deux algorithmes **`expo1`** et **`expo2`** ?

On va utiliser le théorême maître pour calculer la complexité d'algorithmes récursifs.

Soit l'algorithme **`expo3`** ci-dessous :

```
1 Algorithme : expo3(y,x)
  Données : deux entiers y et x
  Résultat : y^x
2 début
3     si x = 1 alors
4         retourner y;
5     sinon
6         retourner expo3(y, x/2) * expo3(y, x/2);
```

## Question 20

(20.a) Appliquez le théorème maître aux algorithmes **`expo2`** et **`expo3`**.

(20.b) Ces deux algorithmes sont-il équivalents ? Que concluez-vous ?

On s'intéresse maintenant au cas général, où $x$ est un nombre pair ou impair. On veut résoudre le calcul d'exponentiation dans ce gas général mais plus efficacement qu'avec l'algorithme **`expo1`**.

Soit l'algorithme **expo4** ci-dessous :

**Algorithme :** expo4(y,x)
**Données :** deux entiers $y$ et $x$
**Résultat :** $y^x$
*res* : entier;
**début**
  **si** $x = 0$ **alors**
    **retourner** 1;
  **sinon si** $x\%2 = 0$ *(x est un multiple de 2)* **alors**
    $res \leftarrow expo4(y, x/2)$;
    **retourner** $res * res$;
  **sinon**
    **retourner** *?;*

### Question 21

(21.a) Remplissez la dernière ligne de l'aglorithme **expo4** de façon à ce qu'il fonctionne pour tout entier $x$.

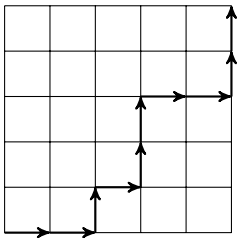
(21.b) Quelle est sa complexité ?

---

**Exercice 7** Nombre de Catalan.

L'algorithme suivant affiche une liste de chemins dans une grille d'ordre $n$.

**Algorithme :** `AfficherChemins`$(x, y, n, c)$
**Données :** Entiers : $x, y, n$, Chaine de caractère : $c$
**Résultat :** ?
**si** $y = n$ **alors**
  afficher $c$;
**sinon**
  **si** $y < x$ **alors**
    `AfficherChemins`$(x, y + 1, n, c + \text{'}\uparrow\text{'})$;
  **si** $x < n$ **alors**
    `AfficherChemins`$(x + 1, y, n, c + \text{'}\rightarrow\text{'})$;



**Question 22** De quels chemins s'agit-t'il si la donnée est $0, 0, 5,$"" ?

**Question 23** Un chemin "sous-diagonal" $c_n$ commence toujours par '$\rightarrow$'. De plus, il existe au moins un '$\uparrow$' précédé par $i + 1$ occurrences de '$\rightarrow$' et $i$ occurrences

de '↑' (le dernier '↑' est toujours dans ce cas). Tout chemin $c_n$ a donc la forme suivante où $i$ est le plus petit entier tel que cette proposition est vraie.

$$c_n = \rightarrow \underbrace{\cdots\cdots}_{c_i} \uparrow \underbrace{\cdots\cdots}_{c_{n-i-1}}$$

Montrer que $c_i$ et $c_{n-i-1}$ sont des chemins sous-diagonaux, pour des grilles d'ordre $i$ et $n-i-1$, respectivement.

**Question 24** On veut caculer $C_n$ le nombre de chemins sont affichés par cet algorithme. Donnez une forme récursive de ce nombre, en vous appuyant sur l'observation faite dans la question précédante. Le nombre de chemins dans une grille d'ordre 0 est :

$$C_0 = 1$$

Le nombre de chemins dans une grille d'ordre $n$ est (on donne la forme récursive, c'est à dire en fonction de $C_i$ avec $i < n$) :

$$C_n = \ldots$$

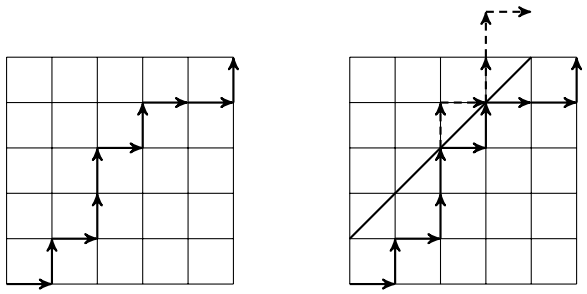


FIGURE 1 – A gauche : Un chemin $c$ monotone non sous-diagonal de $(0,0)$ vers $(n,n)$. A droite, le chemin $f(c)$ monotone de $(0,0)$ vers $(n-1,n+1)$ en pointillés

**Question 25** On veut prouver

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

(25.a) Donner le nombre total de chemins *monotones* de $(0,0)$ à $(n,n)$, c'est à dire tels que le successeur de $(x,y)$ est soit $(x+1,y)$ ou $(x,y+1)$

(*indice, il faut utiliser une de ces notions : permuation, arrangement ou combinaison*).

(25.b) Un chemin monotone de $(0,0)$ vers $(n,n)$ n'est pas sous-diagonal si et seulement si il touche la diagonale $y = x+1$, soit $f(c)$ l'opération qui transforme un chemin monotone non sous-diagonal en remplaçant, à partir de la position où la diagonale $y = x+1$ est touchée, toute occurence de $\rightarrow$ par $\uparrow$ et vice versa.

Montrer que si $c$ de $(0,0)$ vers $(n,n)$ est monotone et non sous-diagonal, alors $f(c)$ est monotone et s'arrête en $(n-1, n+1)$.

(25.c) Montrer que $f$ est une bijection, c'est à dire : si $c$ est monotone et part de $(0,0)$ et s'arrête en $(n-1, n+1)$, alors $f^{-1}(c)$ est monotone et non sous-diagonal.

(25.d) Donner le nombre total de chemins monotones qui partent de $(0,0)$ et s'arrêtent en $(n-1, n+1)$.

(25.e) En déduire la formule pour $C_n$.

**Question 26** Asymptotiquement, on a l'équivalence :

$$C_n \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}$$

C'est à dire

$$\lim_{n\to\infty} C_n \frac{n^{3/2}\sqrt{\pi}}{4^n} = 1$$

Montrer que $C_n \in \Theta\left(\frac{4^n}{n^{3/2}}\right)$

**Question 27** Soit le langage $\Sigma$ sur l'alphabet $\{`(', `)'\}$ qui ne contient que des expressions "bien-formées", c'est à dire telle que chaque parenthèse ouvrante peut être associée à exactement une parenthèse fermante postérieure dans la séquence.

Par exemple, "(()(()()))" $\in \Sigma$, "())())(" $\notin \Sigma$

Qelle est la taille d'une donnée, s'il s'agit d'un mot de $\Sigma$ de longueur $n$ ?

**Exercice 8** Programmation dynamique

**Question 28** La suite de Fibonacci $\mathcal{F}$ peut être définie de la façon suivante :

$$\mathcal{F}(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ \mathcal{F}(n-2) + \mathcal{F}(n-1) & \text{sinon} \end{cases} \tag{1}$$

(28.a) Justifiez l'utilisation de la programmation dynamique par rapport à un algorithme récursif pour calculer $\mathcal{F}(n)$.

(28.b) Proposez un programme dynamique pour calculer $\mathcal{F}(n)$

(28.c) Déroulez l'algorithme pour $n = 10$.

**Question 29** Une compagnie aérienne a décidé de changer sa politique concernant les bagages en cabine de la façon suivante :

- Chaque voyageur est autorisé à un seul sac à dos
- Il n'y a pas de restriction sur le volume
- Un sac à dos ne doit pas dépasser 10kg de poids

Un voyageur doit alors sélectionner des d'objets à mettre dans le sac à dos (parmi $n$ objets) qui ne dépassent pas le poids autorisé (10kg) tout en maximisant la valeur des objets sélectionnés.

Ce problème est très connu en optimisation combinatoire sous le nom "problème du sac à dos". Un problème de sac à dos $\mathcal{P}(n, W, w_1, \ldots, w_n, v_1, \ldots, v_n)$ est définit de la façon suivante :

- Il y a $n$ objets $o_1, o_2, \ldots o_n$ ($n \in \mathbb{N}^*$)
- $W$ est un entier qui représente le poids total autorisé
- $\forall i \in [1, n]$, le poids de $o_i$ est $w_i \in \mathbb{N}^*$
- $\forall i \in [1, n]$, la valeur de $o_i$ est $v_i \in \mathbb{N}^*$
- $\forall i \in [1, n]$, on note par $x_i$ la variable booléenne (binaire) de décision qui représente le fait que $o_i$ est sélectionné dans le sac à dos. C'est à dire :

  $x_i = 1$ si $o_i$ est sélectionné dans le sac à dos

  $x_i = 0$ si $o_i$ n'est pas sélectionné dans le sac à dos
- On cherche à
  - Maximiser $\sum_{i=1}^{i=n} v_i \times x_i$
  - Tel que $\sum_{i=1}^{i=n} w_i \times x_i \leq W$

(29.a) Soit $F(n, W)$ le coût d'une solution optimale de $\mathcal{P}(n, W, w_1, \ldots, w_n, v_1, \ldots, v_n)$. Trouvez une définition récursive de $F(n, W)$

(29.b) Justifiez l'utilisation d'un programme dynamique

(29.c) Écrire un programme dynamique pour calculer $F(n, W)$ :

(29.d) Faites l'exécution pour $\mathcal{P}(3, 10, 7, 2, 3, 5, 11, 6)$ en affichant la table dynamique (i.e., les $F(n, W)$ calculés par l'algorithme)

---

**Exercice 9** Algorithmes gloutons et matroïdes

**Question 30** Soit $E = \{x_1, x_2, \dots x_n\}$ un ensemble fini d'éléments et $d$ une fonction de distance sur les éléments de $E$. C'est à dire, pour tout $x_i \in E$ et $x_j \in E$, $d(x_i, x_j) \in \mathbb{R}^+$ représente la distance entre $x_i$ et $x_j$. Notez que $d(x_i, x_i) = 0$ pour tout $x_i \in E$.

Pour tout sous-ensemble $E_1 \subseteq E$, on définit le coût de $E_1$, noté par $c(E_1)$, comme la distance entre les deux éléments les plus éloignés de $E_1$. C'est à dire $c(E_1) = \max\{d(x_i, x_j)$ tel que $x_i \in E_1$ et $x_j \in E_1\}$.

Deux sous-ensembles $E_1 \subseteq E$ et $E_2 \subseteq E$ forment une 2-partition de $E$ si les deux conditions suivantes sont satisfaites :

— $E_1 \cap E_2 = \emptyset$ : c'est à dire l'intersection de $E_1$ et $E_2$ est vide

— $E_1 \cup E_2 = E$ : c'est à dire l'union de $E_1$ avec $E_2$ est égale à $E$.

Si $E_1$ et $E_2$ forment une 2-partition de $E$ alors le coût de cette partition est égale à la valeur maximale entre $c(E_1)$ et $c(E_2)$. On cherche à trouver une 2-partition de E de coût minimal.

(30.a) Trouver un exemple réel d'application (on peut considérer le cas général d'une $k$-partition au lieu de 2-partition)

(30.b) Proposer **informellement** une idée d'algorithme glouton pour ce problème (en quelques phrases).

(30.c) Donner l'algorithme glouton (sans forcément calculer le coût de la solution). Discuter sa complexité.

Vous pouvez utiliser les 4 fonctions suivantes : étant donnée une liste $L = \{y_1, \dots y_k\}$ de réels

— $max(L)$ renvoie un élément avec la valeur la plus grande de $L$

— $argmax(L)$ renvoie un indice $i$ tel $y_i = max(L)$

— $min(L)$ renvoie un élément avec la valeur la plus petite de $L$

— $argmin(L)$ renvoie un indice $i$ tel $y_i = min(L)$

**Question 31** Soit $E = \{a_1, a_2, \ldots, a_n\}$ un ensemble fini de tâches unitaires (qui nécessite une unité de temps pour l'exécution). Les tâches s'exécutent séquentiellement sur une même ressource. Par exemple, si $E = \{a_1, a_2, a_3, a_4\}$ et si on choisit d'exécuter les taches dans l'ordre $a_3, a_1, a_4, a_2$, alors $a_3$ commence à $t = 0$, $a_1$ commence à $t = 1$, $a_4$ commence à $t = 2$, et $a_2$ commence à $t = 3$.

Pour chaque tâche $a_i$, il y a une date d'échéance $d(a_i)$. C'est à dire, la tâche $a_i$ doit finir avant la date $d(a_i)$ (sachant que $1 \leq d_i \leq n$). Cependant c'est souvent impossible de finir toutes les taches avant leurs dates d'échéance.

Une séquence de tâches $s_1, s_2, \ldots, s_c$ ($s_i \in E$) est dite **valide** si l'exécution dans l'ordre $s_1, s_2, \ldots, s_c$ respecte toutes les dates d'échéance. **Remarquons qu'une séquence de tâches $S = s_1, s_2, \ldots, s_c$ est valide *ssi* $d(s_i) \geq i$.** Un sous ensemble $A$ de $E$ est dit **réalisable** s'il existe une séquence valide de $A$. On s'intéresse à trouver un sous ensemble réalisable de tâches de cardinalité maximale. On va résoudre ce problème à travers le formalisme matroïde.

(31.a) Trouver un exemple réel d'application

(31.b) Soit $A$ un sous ensemble réalisable. Montrer que tout sous ensemble de A est réalisable.

(31.c) Soit $A$ un sous ensemble de tâches et soit $S$ la séquence des tâches de $A$ ordonnées par leurs dates d'échéance selon un ordre croissant ($\geq$). Montrer que $A$ est réalisable *ssi* $S$ est valide.

(31.d) Soit $A$ un ensemble de tâches. Pour tout $t \in [0, n]$, on note par $N_t(A)$ le nombre de tâches dans $A$ ayant des dates d'échéances $\leq t$. Montrer que $A$ est réalisable ssi $\forall t \in [0, n], N_t(A) \leq t$.

(31.e) Soit $A$ et $B$ deux sous ensembles réalisables tel que $|B| > |A|$.

- Montrer qu'il existe $k < n$ tel que $N_k(B) \leq N_k(A)$ et $\forall t > k, N_t(B) > N_t(A)$.
- À l'aide de la question 4, montrer qu'il existe $a \in B \setminus A$ tel que $A^* = A \cup \{a\}$ est réalisable

(31.f) Proposer un algorithme glouton optimal pour ce problème en utilisant le formalisme matroïde. Discuter sa complexité.

---

**Exercice 10** Structure de données

**Question 32** Les données $10, 9, ..., 2, 1$ sont insérées dans un tas initialement vide $T$ dans cet ordre. Que contient $T[1], ..., T[10]$ ?

**Question 33** L'agence de renseignement Syldave a placé sur écoute les $n$ habitants de la Bordurie référencés par leur numéro de téléphone.

Elle souhaite stocker les communications (émetteur $x$ et récepteur $y$) contenant le mot "sceptre", de façon à pouvoir exécuter les opérations de base suivantes en temps $\mathcal{O}(\log n)$ dans le cas moyen :

- ajouter un appel $(x, y)$ à l'ensemble
- déterminer si un numéro de téléphone $x$ a été émetteur de ce type de d'appels
- déterminer si un numéro de téléphone $y$ a été récepteur de ce type de d'appels
- déterminer pour une paire $(x, y)$ si $(x, y)$ appartient à l'ensemble

Quelle structure(s) de données proposez-vous ? Justifiez votre réponse en donnant (sans preuve) la complexité des quatre opérations de base. (Indice : vous pouvez utiliser plusieurs structures de données.)

---

## Exercice 11 **P** et **NP**

**Question 34** Les problèmes suivants ne sont pas connus pour être dans **P**, à quelle(s) classe(s) de complexité – parmi **NP** et **coNP**– appartiennent-il ? Pour chaque relation d'appartenance donner un certficat polynômial pour le (co-)problème et donner la complexité de l'algorithme de vérification.

*Sac-à-Dos*

**Données :** un ensemble de $n$ objets chacun ayant une valeur $v_i$ et un poids $w_i$, deux entiers $V$ et $W$ positifs

**Question :** Existe-t-il un remplissage du sac (sous-ensemble d'objets) tel que la valeur du chargement soit au moins $V$ et le volume inférieur ou égal à $W$ ?

*Tautologie*

**Données :** Une formule booléenne $\phi$

**Question :** Est-ce que $\phi$ est une tautologie ? (une tautologie est vraie quelle que soit son interprétation)

**Question 35** L'algorithme suivant résout le problème du *Sac-à-Dos*, quelle est sa complexité ? Pourquoi ne peut-on pas en déduire que *Sac-à-Dos* $\in \mathbf{P}$ ?

**Algorithme :** ProgDyn($\{n, v, w, V, W\}$)
**Données :** trois entiers $n, W, V$, deux tableaux de $n$ entiers $v$ et $w$
**Résultat :** $\bar{V}$ : matrice $(n+1) \times (W+1)$ d'entiers, où $\bar{V}[i,p]$ est la valeur maximum d'un sous-ensemble des $i$ premiers objets de poids total inférieur ou égal à $p$.

```
début
    pour p allant de 0 à W faire
        V̄[0,p] ← 0;
    pour i allant de 1 à n faire
        pour p allant de 0 à W faire
            si w[i] ≤ p alors
                V̄[i,p] ← max{V̄[i−1,p], v_i + V̄[i−1,p−w[i]]};
            sinon
                V̄[i,p] ← V̄[i−1,p];
    retourner V̄[n,W] ≥ V;
```

**Question 36** Démontrer que $\mathbf{P} \subseteq \mathbf{coNP}$

**Question 37** Démontrer que $\mathbf{NP} \neq \mathbf{coNP}$ implique $\mathbf{P} \neq \mathbf{NP}$

---

**Exercice 12** Sous-séquence maximale

Une généticienne souhaite trouver des gènes codants dans un brin d'ADN. Dans ce but elle décide d'abord de déterminer une région dans laquelle les bases nucléique "G" et "C" sont plus fréquentes que les bases "A" et "T".[1] Elle représente le brin d'ADN par un tableau $L$ contenant les éléments 1 (pour chaque G ou C) et $-1$ (pour chaque A et T) et veut calculer la *sous-séquence maximale*, c'est à dire telle que la somme de ses éléments est la plus grande possible. Autrement dit, elle veut résoudre le problème suivant (cf. Exemple 1) :

*Sous-séquence Maximale*

**Donnée :** Un tableau $L$ avec $n$ éléments dans $\{-1, 1\}$

**Question :** Quelles sont les valeurs de $s$ et $e$ dans $[1, n]$ qui maximisent $\sum_{i=s}^{e} L[i]$ ?

1. cette propriété est corrélé à celle de contenir un gène

**Exemple 1** *Soit* $L =$

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $-1$ | $-1$ | $1$ | $1$ | $-1$ | $1$ | $-1$ | $1$ | $1$ | $-1$ | $-1$ | $1$ | $1$ | $-1$ | $1$ |

*La sous-sequence* $1+1+-1+1+-1+1+1$ *correspondant à* $s=3, e=9$ *est maximale :* $1+1+-1+1+-1+1+1=3$ *et aucune autre sous-séquence n'a une somme strictement supérieure.*

**Question 38** Quelle est la taille de la donnée du problème de *Sous-séquence Maximale* ?

**Question 39** Ecrivez un algorithme ($\texttt{MaxSousSéquence}^q$) qui résout ce problème en $\Theta(n^2)$.

**Algorithme :** $\texttt{MaxSousSéquence}^q((L))$
**Données :** un tableau $L$ avec éléments dans $\{-1, 1\}$
**Résultat :** les entiers $s$ et $e$ tels que $\sum_{i=s}^{e} L[i]$ est maximal

L'algorithme $\texttt{MaxSousSéquence}^r$ (avec les procédures `MaxSS` et `MaxSScentre`) permet de résoudre le problème de la *Sous-séquence Maximale.*

**Algorithme :** $\texttt{MaxSousSéquence}^r(L)$
**Données :** un tableau $L$ avec éléments dans $\{-1, 1\}$
**Résultat :** les entiers $s$ et $e$ tels que $\sum_{i=s}^{e} L[i]$ est maximal
$(v, s, e) \leftarrow \texttt{MaxSS}(L, 1, |L|)$;
**retourner** $(s, e)$;

**Algorithme :** $\texttt{MaxSS}(L, i, j)$
**Données :** un tableau $L$ avec éléments dans $\{-1, 1\}$, deux entiers $1 \leq i \leq j \leq |L|$
**Résultat :** les entiers $v$, $s$ et $e$ tels que $v = \sum_{i=s}^{e} L[i]$ est maximal pour le tableau $L[i : j+1]$
**si** $i = j$ **alors**
  **retourner** $(L[i], i, i)$;
**sinon**
  $m \leftarrow \lfloor (i+j)/2 \rfloor$;
  $(v_1, i_1, j_1) \leftarrow \texttt{MaxSS}(L, i, p)$;
  $(v_2, i_2, j_2) \leftarrow \texttt{MaxSS}(L, p+1, j)$;
  $(v_3, i_3, j_3) \leftarrow \texttt{MaxSScentre}(L, m, i, j)$;
  **retourner** $\max((v_1, i_1, j_1), (v_2, i_2, j_2), (v_3, i_3, j_3))$

**Question 40** Soit $n$ la taille du sous-tableau $L[i : j+1]$ (i.e., $n = (j - i + 1)$), on considère que la procédure `MaxSScentre` est en $\Theta(n)$.

(40.a) Ecrivez la fonction récursive $T(n)$ qui représente la complexité de la procédure `MaxSS` en fonction de $n$

(40.b) Utilisez le théorème maître pour calculer la complexité de l'algorithme `MaxSS` en fonction de $n = (j - i + 1)$.

(40.c) Utilisez la méthode par substitution pour prouver cette complexité

**Question 41** Après une longue réflexion, la généticienne propose l'algorithme suivant :

```
1  Algorithme : MaxSousSéquence^l(L)
   Données : un tableau L avec éléments dans {−1, 1}
   Résultat : les entiers s et e tels que Σ_{i=s}^{e} L[i] est maximal
2  courant ← maximum ← L[0];
3  i ← s ← e ← 0;
4  pour chaque j allant de 2 à |L| faire
5      si L[j] > courant + L[j] alors
6          courant ← L[j];
7          i ← j;
8      sinon
9          courant ← courant + L[j];
10     si courant > maximum alors
11         s ← i;
12         e ← j;
13         maximum ← courant;
```

(41.a) Quelle est la complexité asymptotique de l'algorithme $\texttt{MaxSousSéquence}^l$ en fonction de la taille de la donnée ? Donnez l'ordre de grandeur en utilisant la notation $\Theta$.

(41.b) Est-ce qu'il résout le problème de la *Sous-séquence Maximale* ?

(41.c) En utilisant un processeur dont la fréquence est d'un million d'opérations par seconde, combien de temps (on demande un ordre de grandeur) prendraient les algorithmes $\texttt{MaxSousSéquence}^q$, $\texttt{MaxSousSéquence}^r$ et $\texttt{MaxSousSéquence}^l$ sur un brin d'ADN long de 500000 bases ?

---

**Exercice 13** Plus courts chemins

Soit $V = \{1, \ldots, n\}$. L'algorithme suivant calcule tous les plus courts chemins depuis un sommet $s \in V$ vers chaque sommet $v \in V$ dans le graphe non-orienté

$G = (V, E)$ où $E$ est un ensemble de $m$ arêtes, c'est à dire de $m$ paires distinctes de sommets de $V$. Le graphe est représenté par des listes de voisins : la liste $N(v)$ contient tous les voisins du sommet $v$, c'est à dire l'ensemble des sommets $u$ tels que $\{u, v\}$ [2] $\in E$. Les arêtes sont valuées, et la "longueur" d'une arête $L[\{u, v\}]$ est accessible en temps constant.

Dans cet algorithme, toutes les opérations d'*ajout*, de *suppression*, d'*affectation* et les *additions* seront considérées en temps constant ($\mathcal{O}(1)$).

Les réponses des six premières questions sont à donner en **ordre de grandeur ($\Theta$) en fonction de $n$ et/ou de $m$.** Si vous avez un doute, une réponse moins précise utilisant $\Omega$ et/ou $\mathcal{O}$ rapportera plus de points qu'une réponse erronée en $\Theta$.

Notez que comprendre le fonctionnement de l'algorithme n'est pas absolument nécessaire pour répondre aux questions.

```
1  Algorithme : PlusCourtChemin
   Données : Un ensemble de sommets V, des listes de voisinage N, une table de distance
             L et un sommet s ∈ V
   Résultat : un tableau dist, où ∀v ∈ {1, ..., n}, dist[v] est la distance du plus court
             chemin entre s et v dans le graphe
2  Q ← ∅;
3  pour chaque v ∈ V faire
4      dist[v] ← ∞;
5      ajouter v à Q;
6  dist[s] ← 0;
7  tant que Q n'est pas vide faire
8      u ← sommet de Q tel que dist[u] est minimal;
9      retirer u de Q;
10     pour chaque v dans la liste N(u) faire
11         l ← dist[u] + L[{u, v}];
12         si l < dist[v] alors
13             dist[v] ← l;
14 retourner dist;
```

**Question 42** Quelle est la complexité des lignes 3 à 5 ?

**Question 43** Dans cet algorithme, combien de fois au total est exécutée la ligne 8 ?

2. ou $\{v, u\}$ puisque le graphe n'est pas orienté

**Question 44** Dans cet algorithme, étant donnée une arête $\{u, v\} \in E$, combien de fois cette arête est-elle visitée ? (les arêtes sont visitées lorsque leur longueur est lue, à la ligne 11)

**Question 45** Dans cet algorithme, combien de fois au total sont exécutées les lignes 11 à 13 ?

**Question 46** Dans une version de `PlusCourtChemin`, la ligne 8 s'exécute en $\Theta(n)$ et les lignes 11 à 13 en $\Theta(1)$. Quelle est la complexité de cette version de `PlusCourtChemin` ?

**Question 47** Dans une autre version de `PlusCourtChemin`, la ligne 8 s'exécute en $\Theta(1)$ et les lignes 11 à 13 en $\Theta(\log n)$. Quelle est la complexité de cette version de `PlusCourtChemin` ? [3]

**Question 48** La taille de la donnée est en $\Theta(n + m)$, est-ce que le problème de trouver le plus court chemin entre deux sommets d'un graphe est polynomial (dans $\mathbf{P}$) ?

**Question 49** Quel est le nombre maximum de chemins *unitaires* (ne passant pas plus d'une fois par le même sommet) dans un graphe avec $n$ sommets ?

---

**Exercice 14** Vote à la majorité absolue

Le résultat d'une élection avec $m$ candidats et $n$ votants peut se représenter par une table $S$ (un suffrage) de longueur $|S| = n$. Le $i$-ème vote est donné par $S[i]$ qui

3. Pour les curieux, cette version utilise un tas binaire, et donc mettre la valeur de $dist[v]$ à jour coûte $\Theta(\log n)$

est un entier dans $\{1, \ldots, m\}$ représentant un des $m$ candidats (les indices vont de 1 à la taille de la table dans tout le document).

Un candidat gagne l'élection s'il obtient strictement plus que $\left\lfloor \frac{|S|}{2} \right\rfloor$ voix, on dit qu'il est *majoritaire*. Par exemple, dans le scrutin suivant, le gagnant est le candidat '3', qui est majoritaire puisqu'il a obtenu $7 > \left\lfloor \frac{13}{2} \right\rfloor = 6$ voix :

$$1 \quad 1 \quad 1 \quad 3 \quad 3 \quad 2 \quad 2 \quad 3 \quad 3 \quad 3 \quad 2 \quad 3 \quad 3$$

*Majorité absolue*

**Donnée :** Une table $S$ de votes (dans $\{1, \ldots, m\}$) contenant un élément majoritaire

**Question :** Quel est l'élément majoritaire ?

Notez que le suffrage suivant n'est *pas une donnée valide* pour le problème puisqu'il n'y a pas d'élément majoritaire : $2 \quad 2 \quad 1 \quad 1 \quad 3 \quad 2 \quad 1 \quad 3 \quad 2 \quad 2$

**Question 50**

(50.a) Quelle est la taille de la donnée du problème *Majorité absolue*, si on fixe le nombre de candidats ($m = \Theta(1)$) ?

(50.b) Idem si on ne la fixe pas ?

```
1 Algorithme : BruteVote
  Données : Une table S de votes (dans {1, ..., m}) contenant un élément majoritaire
  Résultat : L'élément majoritaire
2 n ← |S|;
3 pour i allant de 1 à n faire
4     compte ← 0;
5     pour j allant de 1 à n faire
6         si S[i] = S[j] alors compte ← compte + 1;
7     si compte > ⌊n/2⌋ alors retourner S[i];
```

**Question 51** Quelle est la complexité en temps dans le pire des cas de l'algorithme `BruteVote` ?

L'algorithme suivant est celui couramment utilisé dans les bureaux de vote : on comptabilise le nombre de voix pour chaque candidat.

```
1  Algorithme : Dépouillement
   Données : Une table S de votes (dans {1, ..., m}) contenant un élément majoritaire
   Résultat : L'élément majoritaire
2  n ← |S|;
3  decompte ← tableau de taille m;
4  pour j allant de 1 à m faire  decompte[j] ← 0 ;
5  pour i allant de 1 à n faire
6  |  decompte[S[i]] ← decompte[S[i]] + 1;
7  |  si decompte[S[i]] > ⌊n/2⌋ alors retourner S[i];
```

## Question 52

(52.a) Quelle est la complexité dans le pire des cas, en temps puis en espace, de l'algorithme `Dépouillement` ?

(52.b) Dans quel cas (peu fréquent) `Dépouillement` est-il strictement pire en temps que `BruteVote` ?

L'algorithme récursif suivant détermine le gagnant si le premier appel est fait avec $i = 1$ et $j = |S|$ :

```
1  Algorithme : DiviserPourÉlire
   Données : Une table S de votes (dans {1, ..., m}) contenant un élément majoritaire,
             deux entiers 1 ≤ i ≤ j ≤ |S|
   Résultat : L'élément majoritaire parmi S[i], ..., S[j]
2  si i = j alors retourner S[i];
3  n ← j − i + 1;
4  mid ← ⌊(i+j)/2⌋;
5  l ← DiviserPourÉlire(Σ, S, i, mid);
6  r ← DiviserPourÉlire(Σ, S, mid + 1, j);
7  si r = l alors  retourner r ;
8  sinon
9  |  compte ← 0;
10 |  pour k allant de i à j faire
11 |  |  si S[k] = l alors  compte ← compte + 1 ;
12 |  si compte > ⌊n/2⌋ alors retourner l;
13 |  compte ← 0;
14 |  pour k allant de i à j faire
15 |  |  si S[k] = r alors  compte ← compte + 1 ;
16 |  si compte > ⌊n/2⌋ alors retourner r;
17 retourner S[i];
```

### Question 53

(53.a) Donner une justification (1 ou 2 phrases) que `DiviserPourÉlire` est correct.

(53.b) Ecrivez la fonction récursive $T(n)$ qui représente la complexité en temps de la l'algorithme `DiviserPourÉlire` en fonction de $n = j - i + 1$ :

(53.c) Quelle est la complexité en temps dans le pire des cas de l'algorithme `DiviserPourÉlire` en fonction de $n$ ?

Robert Boyer et J Moore ont montré comment déterminer le gagnant de manière plus efficace :

```
1  Algorithme : Boyer-Moore
   Données : Une table S de votes (dans {1, ..., m}) contenant un élément majoritaire
   Résultat : L'élément majoritaire
2  compte ← 0;
3  candidat ← ∅;
5  pour i allant de 1 à |S| faire
6      si compte = 0 alors candidat ← S[i];
7      si candidat = S[i] alors
8          compte ← compte + 1;
9      sinon
10         compte ← compte − 1;
11 retourner candidat;
```

### Question 54

(54.a) Quelle est la complexité dans le pire des cas, en temps puis en espace, de l'algorithme `Boyer-Moore` ?

(54.b) Parmis tous les algorithmes vus précédemment (en supposant qu'ils soient tous corrects), lequel ne constitue pas une preuve que le problème *Majorité absolue* appartient à la classe **P** ? Pourquoi ?

(54.c) Si la table ne contient pas d'élément majoritaire, alors `Boyer-Moore` renvoie un candidat indeterminé. Comment peut-on compléter cet algorithme pour renvoyer '0' dans ce cas (de suffrage inconclusif), *sans augmenter sa complexité* ?

Pour prouver que `Boyer-Moore` est correct, on utilise un invariant de boucle. On définit d'abord, pour un candidat $x$ et un indice $1 \leq i \leq n$, la valeur $marge(x, i)$

égale au nombre de voix pour le candidat $x$ dans $S[1], \ldots, S[i]$ *moins* le nombre de voix pour tout autre candidat dans le même intervale :

$$\begin{aligned} marge(x,i) &= |\{j : 1 \leq j \leq i \ \&\ S[j] = x\}| - |\{j : 1 \leq j \leq i \ \&\ S[j] \neq x\}| \\ &= 2 \times |\{j : 1 \leq j \leq i \ \&\ S[j] = x\}| - i \end{aligned}$$

On peut donc maintenant proposer les deux propriétés invariantes suivantes :

— Invariant $a(i)$ : à la fin de la $i$-ème itération, $marge(candidat, i) \leq compte$

— Invariant $b(i)$ : à la fin de la $i$-ème itération, $x \neq candidat \implies marge(x,i) \leq -compte$

**Question 55**

(55.a) Montrez que si les invariants sont vérifiés lorsqu'on sort de la boucle (c'est à dire, $a(n)$ et $b(n)$ sont toujours vrais), alors `Boyer-Moore` est correct.

(55.b) Ecrivez une trace de l'algorithme `Boyer-Moore` avec les valeurs de $i, compte, candidat$ et les valeurs correspondantes à la définition de $marge(x,i)$ plus haut. Est-ce que les invariants sont vérifiés à la fin de chaque itération (oui/-non) ?

(55.c) Que manque-t-il pour prouver que `Boyer-Moore` est correct ?

---