# Examen final

Durée : 1 heure 30.

## Avant de commencer

Votre copie doit être **anonyme**. Faites donc attention à bien cacheter vos copies en repliant le coin et en le collant, et à ne reporter que votre **numéro d'anonymat** fourni sur la première copie sur les intercallaires. Si vous cachetez votre copie avec de la colle, n'en mettez **que sur le pourtour du coin** et pas sur votre nom ou sur la portion du coin qui se repliera dessus, sinon il nous sera difficile de vous attribuer votre travail après la correction lors de la désanonymisation des copies.

Toutes vos réponses nécessitent une justification. Même lorsque ce n'est pas précisé, ne répondez jamais simplement « oui » ou « non » sans explication, car vous auriez alors très bien pu répondre au hasard. Vous ne recevrez donc pas les points des questions avec une telle réponse. Ce sujet comporte deux problèmes totalement indépendants, et au sein de chaque problème des questions peuvent être indépendantes. Prenez donc le temps de lire le sujet et de choisir les questions que vous abordez pour ne pas rester coincé.

## 1 Quickselect

L'algorithme *quickselect* est un algorithme qui a pour but de répondre au problème du $k^{\text{ième}}$ élément d'un tableau : étant donné un tableau d'éléments dans le désordre, quel serait l'élément d'indice $k$ de ce tableau s'il était trié ? Lorsque $k$ vaut 0, il s'agit de trouver le minimum. Si $n$ est la taille du tableau, pour $k = n - 1$, il s'agit de trouver le maximum et pour $k = \frac{n}{2}$ la médiane. Dans le tableau suivant, la $2^{\text{ième}}$ valeur est 5.

| 6 | 1 | 13 | 8 | 24 | 5 | 0 | 17 | 9 | 12 |
|---|---|---|---|---|---|---|---|---|---|

### 1.1 Algorithme naïf

**Question 1 –** Proposez un algorithme simple pour fournir le résultat. Cet algorithme prendra en paramètre le tableau et le $k$ souhaité, et renverra la $k^{\text{ième}}$ valeur du tableau. Il ne s'agit pas ici de chercher un algorithme performant mais simplement de trouver le résultat.

**Question 2 –** Donnez la complexité de *votre* algorithme dans le meilleur et le pire cas. Il s'agit ici de rester cohérent avec votre réponse précédente.

### 1.2 Pivoter

Comme pour le *quicksort*, l'algorithme *quickselect* utilise le principe du pivot dont l'algorithme est rappelé ci-dessous. Dans cet algorithme, tableau est le tableau qui subit le pivot, debut est l'indice de la première case de la zone du tableau qui subit le pivot et fin est l'indice de la case qui suit la dernière case pivotée du tableau.

```
Fonction pivoter(tableau, debut, fin) → entier
    pivot ← tableau[debut]
    inf ← debut
    sup ← fin − 1
    tant que inf ≤ sup faire
        tant que tableau[inf] ≤ pivot faire inf ← inf + 1
        tant que tableau[sup] > pivot faire sup ← sup − 1
        si inf < sup alors
            échanger tableau[inf] et tableau[sup]
            inf ← inf + 1
            sup ← sup − 1
    échanger tableau[debut] et tableau[sup]
    retourner sup
```

**Question 3 –** Appliquez l'algorithme pivoter sur le tableau fourni en introduction avec debut $= 0$ et fin $= 10$ (tout le tableau). Vous représenterez le tableau à chaque échange et annoterez les cases qui sont échangées. Vous indiquerez également la valeur de retour finale.

**Question 4 –** Si $n =$ fin $-$ debut est la taille de la zone pivotée du tableau, quelle est l'ordre de complexité de l'algorithme pivoter ? Justifiez.

## 1.3 Mise en œuvre

Le but est ici d'utiliser la fonction pivoter pour trouver le $k^{\text{ième}}$ élément du tableau. Le problème consiste donc à déterminer en fonction du résultat de la fonction pivoter la portion du tableau dans laquelle chercher le $k^{\text{ième}}$ élément.

**Question 5 –** Que pouvez-vous dire si $p$ est la valeur de retour de pivoter, et si $p = k$ ?

**Question 6 –** Que pouvez-vous dire si $p$ est la valeur de retour de pivoter, et si $p < k$ ?

**Question 7 –** Que pouvez-vous dire si $p$ est la valeur de retour de pivoter, et si $p > k$ ?

**Question 8 –** Que pouvez-vous dire lorsque la plage du tableau entre debut et fin est de taille 1 ?

**Question 9 –** Formulez un algorithme récursif de type diviser pour régner pour calculer le $k^{\text{ième}}$ élément. Cet algorithme prendra en paramètre le tableau, la valeur $k$ et les indices debut et fin de la plage du tableau dans laquelle le $k^{\text{ième}}$ est recherché. Notez que cet algorithme ne sera valable que si debut $\leq k <$ fin.

**Question 10 –** Appliquez votre algorithme sur le tableau fourni en introduction pour rechercher le $5^{\text{ième}}$ élément du tableau. Vous indiquerez tous les appels récursifs à votre fonction avec leurs paramètres, et l'état du tableau après chaque appel.

## 1.4 Complexité

On rappelle rapidement l'énoncé du Master théorème : soit $T$ une fonction de la forme $T(n) = aT\left(\frac{n}{b}\right) + f(n)$

1. si $f(n) = \mathrm{O}(n^c)$ avec $c < \log_b a$ alors $T(n) = \Theta(n^{\log_b a})$.
2. si $f(n) = \Theta(n^{\log_b a})$ alors $T(n) = \Theta(n^{\log_b a} \log n)$.
3. si $f(n) = \Omega(n^c)$ avec $c > \log_b a$ et si $\exists k < 1, \exists n_0 \geq 0, \forall n \geq n_0, af(n/b) \leq kf(n)$ alors $T(n) = \Theta(f(n))$.

**Question 11 –** Étudiez la complexité de votre algorithme en supposant que pivoter partitionne toujours la zone du tableau en deux zones de tailles égales, et que le cas $k = p$ n'arrive que pour des tableaux de taille 1.

**Question 12 –** En pratique le pivot ne partitionne pas toujours aussi bien les données. Quel est le pire cas de cet algorithme ? Quelle est sa complexité dans ce cas ?

**Question 13 –** Dans le cas d'un tableau réalisant ce pire cas, pouvez-vous donner une autre méthode pour fournir le $k^{\text{ième}}$ élément ? Quelle serait sa complexité ?

# 2 Bellman-Ford

En cours, nous avons vu l'algorithme de Dijkstra pour trouver les plus courts chemins d'un sommet vers tous les autres sommets. Dans le cas de graphes orientés munis d'arêtes orientées de poids négatifs, la complexité de l'algorithme peut devenir très mauvaise. L'algorithme de Bellman-Ford est un algorithme similaire qui est moins efficace que Dijkstra lorsque tous les poids sont positifs, mais permet de conserver une complexité acceptable dans le cas de poids négatifs et de détecter les cycles de poids négatifs qui font que la solution n'existe pas. Dans cet exercice, nous nous plaçons donc dans le cas de graphes *orientés* avec des poids sur les arêtes orientées qui peuvent être positifs ou négatifs.

## 2.1 Relâchement

L'algorithme de Bellman-Ford stocke comme Dijkstra pour chaque sommet $s$ une distance $\mathsf{distance}[s]$ et un sommet précédent $\mathsf{precedent}[s]$. L'algorithme fonctionne en utilisant des *relâchements* d'arêtes orientées, également utilisés par Dijkstra :

**Fonction** relacher($s_1 \to s_2$)
- $d \leftarrow \mathsf{distance}[s_1] + \mathsf{poids}(s_1 \to s_2)$
- **si** $\mathsf{distance}[s_2] > d$ **alors**
  - $\mathsf{distance}[s_2] \leftarrow d$
  - $\mathsf{precedent}[s_2] \leftarrow s_1$

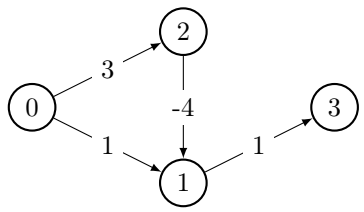
où $s_1$ et $s_2$ sont deux sommets du graphe reliés par une arête orientée $s_1 \to s_2$ de poids $\mathsf{poids}(s_1 \to s_2)$.

## 2.2 Une première version

Une première version de l'algorithme de Bellman-Ford serait :

**Fonction** BellmanFord($s_0$)
- **pour chaque** sommet $s$ **faire**
  - $\mathsf{distance}[s] \leftarrow \infty$
  - $\mathsf{precedent}[s] \leftarrow \emptyset$
- $\mathsf{distance}[s_0] \leftarrow 0$
- $\mathsf{precedent}[s_0] \leftarrow s_0$
- **répéter**
  - **pour chaque** arête orientée $s_1 \to s_2$ du graphe **faire**
    - relacher($s_1 \to s_2$)
- **jusqu'à** ce qu'aucune distance n'ait été modifiée lors de l'itération

**Question 14 –** Appliquez l'algorithme sur le graphe suivant. Vous présenterez son déroulement sous la forme d'un tableau comme proposé ci-dessous avec un sommet par colonne, et une ligne par relâchement d'arête orientée. Chaque case comporte la distance et le prédécesseur du sommet de la colonne après le relâchement de l'arête orientée de la ligne.




| | | sommets | | | |
|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 |
| initialisation | × | $0, 0$ | $\infty, \emptyset$ | $\infty, \emptyset$ | $\infty, \emptyset$ |
| itération 1 | $0 \to 1$ | | | | |
| | $0 \to 2$ | | | | |
| | $1 \to 3$ | | | | |
| | $2 \to 1$ | | | | |
| itération 2 | $0 \to 1$ | | | | |
| | ⋮ | | | | |

**Question 15 –** Que se passe-t-il dans le cas où le graphe comporte un cycle de poids négatif, par exemple avec une arête orientée $3 \to 2$ de poids 1 dans le graphe précédent ?

## 2.3 Résister aux cycles négatifs

Pour résister aux cycles de poids négatifs, nous allons maintenant chercher une borne sur le nombre d'itérations de relâchement à réaliser. Il s'agit néanmoins de s'assurer que dans le cas d'un graphe sans cycle de poids négatifs, l'algorithme ait suffisamment d'itérations pour trouver les bons plus courts chemins.

**Question 16 –** En l'absence de cycles de poids négatif, un plus court chemin peut-il passer plusieurs fois par le même sommet du graphe ? Justifiez bien.

**Question 17 –** Pour un graphe contenant $n$ sommets, quel est donc le nombre maximal d'arêtes orientées le long d'un plus court chemin ? On parle du *nombre* d'arêtes orientées, et pas de la somme de leurs poids.

**Question 18 –** Montrez par récurrence que pour tout $k$, pour tout chemin $s_0 \to s_1 \to \cdots \to s_k$ permettant d'atteindre un sommet $s_k$ depuis $s_0$ en passant par $k$ arêtes orientées, alors au bout de $k$ itérations de l'algorithme $\mathsf{distance}[s_k]$ vaut au maximum la longueur de ce chemin :

$$\mathsf{distance}[s_k] \leq \sum_{i=0}^{k-1} \mathsf{poids}(s_i \to s_{i+1}).$$

**Question 19 –** En déduire une borne sur le nombre d'itérations à réaliser pour s'assurer qu'en l'absence de cycles de poids négatifs tous les plus courts chemins aient été trouvés.

**Question 20 –** Comment déterminer à l'issue de ces itérations si le graphe comportait ou non un cycle de poids négatif ?

## 2.4 Limiter les arêtes orientées à relâcher (plus difficile)

Pour l'instant chaque itération de l'algorithme relâche toutes les arêtes orientées du graphe, ce qui fait potentiellement beaucoup d'arêtes orientées à relâcher, alors que certaines sont inutiles.

**Question 21 –** Déterminez les arêtes orientées qui n'ont pas besoin d'être relâchées à chaque itération. En supposant que chaque sommet peut fournir en temps constant l'ensemble des arêtes dont il est la source, proposez une modification de l'algorithme pour réduire le nombre d'arêtes à relâcher à chaque itération.

**Question 22 –** Lors d'une itération, il est possible d'accélérer la convergence de l'algorithme selon l'ordre choisi pour relâcher les arêtes orientées. En vous inspirant de l'algorithme de Dijkstra, proposez un mécanisme pour essayer de traiter les arêtes orientées dans un ordre favorable.