INFOF-203 – Syllabus d'exercices

Jean Cardinal, Justin Dallant & Robin Petit

Remerciements à Axel Abels, Gian Marco Paldino et Sarah Van Bogaert

2021-2022

Table des matières

Séance 1 — Échauffement

Rappel (Équivalence asymptotique)**.** *Pour deux fonctions $f, g : \mathbb{N} \to \mathbb{R}$ telles que :*

$$\exists N > 0 \text{ s.t. } \forall n \geq N : g(n) \neq 0,$$

on dit que f et g sont équivalents asymptotiquement lorsque :

$$\lim_{n \to +\infty} \frac{f(n)}{g(n)} = 1,$$

et que l'on note $f \sim g$.

Exercice 1.1. Écrire un programme qui, étant donné un tableau `a[]` de n entiers distincts, trouve un *minimum local* : un indice `i` compris entre 0 et $n-1$ tel que `a[i]<a[i+1]` et `a[i]<a[i-1]` (à condition que les bornes existent). Le programme ne devra effectuer que $\sim 2 \log n$ comparaisons au pire cas.

Exercice 1.2. Un tableau `a[]` est dit *bitonique* s'il existe un indice `i` tel que les éléments d'indices entre 0 et `i` sont triés en ordre strictement croissant, et ceux d'indices supérieurs à `i` sont triés en ordre strictement décroissant. Écrire un programme qui détermine si une valeur donnée appartient au tableau. Ce programme ne devra effectuer que $\sim 3 \log n$ comparaisons au pire cas.

Exercice 1.3 (Lancer d'oeufs)**.** On dispose d'un immeuble de N étages et de beaucoup d'oeufs. Un oeuf ne se casse que s'il est lancé d'un étage supérieur à F. Donner un algorithme pour déterminer F en lançant au plus $\sim \log N$ oeufs. Dans l'hypothèse où F est beaucoup plus petit que N, donner un algorithme qui lance au plus $\sim 2 \log F$ oeufs.

Exercice 1.4 (Majorité)**.** Donner un algorithme de complexité $O(n \log n)$ qui détermine si un tableau de n éléments deux à deux comparables en temps constant contient un élément majoritaire, c'est-à-dire un élément qui apparaît plus de $\lfloor n/2 \rfloor$ fois.

Exercice 1.5 (Majorité)**.** Donner un algorithme de complexité linéaire pour le problème précédent, qui n'utilise qu'un espace supplémentaire de taille constante, et qui ne compare les éléments que pour l'égalité.

Exercice 1.6. Décrire un algorithme de complexité $O(n)$ pour le problème suivant : étant donné une matrice `a` $n \times n$ dont les éléments apparaissent en ordre croissant dans chaque ligne et chaque colonne, déterminer si un élément `x` donné en entrée appartient à la matrice. On suppose que tous les éléments sont distincts.

Séance 2 — Union-Find

Exercice 2.1. Donner une implémentation récursive de la compression de chemin dans la méthode `find`.

Exercice 2.2. Trouver l'erreur dans cette écriture de la méthode `union` pour l'algorithme naïf :

```
public void union(int p, int q) {
  if (connected(p, q))
    return;
  for (int i = 0; i < id.length; i++)
    if (id[i] == id[p])
      id[i] = id[q];
  count--;
}
```

Exercice 2.3. Lesquels de ces tableaux `parent` ne peuvent pas être obtenus par la méthode d'union rapide pondérée avec compression de chemin ?

1. 0 1 2 3 4 5 6 7 8 9
2. 7 3 8 3 4 5 6 8 8 1
3. 6 3 8 0 4 5 6 9 8 1
4. 0 0 0 0 0 0 0 0 0 0
5. 9 6 2 6 1 4 5 8 8 9
6. 9 8 7 6 5 4 3 2 1 0

Exercice 2.4. Étant donnés n éléments, donner une suite de $\sim n$ opérations `union` (avec l'algorithme d'union rapide et la compression de chemin) telle que la hauteur d'au moins un des arbres construits soit $\Theta(\log n)$.

Exercice 2.5. Dans la méthode d'union rapide, on attache toujours l'arbre de hauteur plus petite (s'il y en a un) à celui de hauteur plus grande. Que se passe-t-il si on remplace dans cette méthode la hauteur par le nombre d'éléments ?

Séance 3 — Tris

Exercice 3.1 (Le drapeau tricolore)**.** Décrire un algorithme qui, en une passe dans un tableau de taille n avec v comme premier élément, partitionne les éléments en trois groupes successifs : les éléments strictement inférieurs à v, les éléments égaux à v, et ceux strictement supérieurs à v. On peut utiliser un opérateur de comparaison ternaire.

Exercice 3.2 (Boulons et écrous)**.** On se donne un ensemble de n écrous et n boulons. Chaque écrou peut être assemblé avec exactement un boulon, et chaque boulon ne convient qu'à un seul écrou. On peut comparer un écrou et un boulon, et déterminer lequel des deux est le plus grand. Mais on ne peut comparer deux boulons ou deux écrous. Donner un algorithme randomisé qui effectue $O(n \log n)$ comparaisons boulon-écrou en moyenne, et qui associe chaque écrou au bon boulon.

Exercice 3.3. Supposons que nous avons un tableau `a[]` de n éléments tel que chaque élément est à distance au plus k de sa position dans le tableau trié. Donner un algorithme qui permet de trier un tel tableau en $O(n \log k)$ comparaisons au pire cas.

Exercice 3.4. Décrire un algorithme qui étant donnés trois tableaux triés `a[]`, `b[]` et `c[]`, chacun de taille n, les fusionne dans un seul tableau trié `d[]` en effectuant au plus $\sim 6n$ comparaisons. Peut-on réduire le nombre de comparaisons à $\sim 5n$?

Exercice 3.5. Démontrer que le problème de fusion de trois tableaux de l'exercice précédent ne peut être résolu en moins de $4.754n$ comparaisons au pire cas.

Exercice 3.6. Étant donnés deux tableaux triés `a[]` et `b[]` de tailles respectives m et n, où $m \geq n$, donner un algorithme qui les fusionne dans un nouveau tableau `c[]` en utilisant $\sim n \log_2 m$ comparaisons au pire cas.

Séance 4 — Tri par tas

Exercice 4.1. Supposons qu'une application requiert un grand nombre d'insertions mais seulement très peu de suppressions du maximum. Laquelle de ces implémentations serait la plus efficace : un tas, un tableau trié, un tableau non trié ?

Exercice 4.2. Supposons qu'une application requiert un grand nombre de consultations du maximum mais seulement très peu de suppressions du maximum et d'insertions. Laquelle de ces implémentations serait la plus efficace : un tas, un tableau trié, un tableau non trié ?

Exercice 4.3. Donner un algorithme en temps linéaire qui vérifie si un tableau donné en entrée représente un tas.

Exercice 4.4 (Maintenir la médiane)**.** On définit la médiane d'un ensemble de n nombres distincts comme le nombre de l'ensemble qui a exactement $\lfloor (n-1)/2 \rfloor$ éléments plus petits, et $\lceil (n-1)/2 \rceil$ éléments plus grands. Décrire une structure de données qui contient un ensemble de nombres et permet les opérations suivantes : insertion en temps logarithmique, trouver la médiane en temps constant, supprimer la médiane en temps logarithmique.

Exercice 4.5. Donner un algorithme pour l'insertion dans un tas qui n'effectue que $O(\log \log n)$ *comparaisons* au pire cas.

Exercice 4.6. Prouver qu'il est impossible d'avoir une structure de données pour laquelle tant l'insertion que l'effacement du minimum n'utiliserait qu'au plus $O(\log \log n)$ comparaisons.

Séance 5 — Arbres de recherche

Exercice 5.1. Dessiner l'arbre rouge-noir résultant de l'insertion des clés de A à K dans l'ordre alphabétique. Décrivez de manière générale ce qu'il se produit lorsque des clés sont insérées en ordre croissant dans un arbre rouge-noir initialement vide.

Exercice 5.2. Donner une méthode `is23()` qui vérifie qu'aucun nœud n'est connecté à deux arêtes rouges et qu'il n'y a pas d'arête rouge vers la droite. Donner une méthode `isBalanced()` qui vérifie que tous les chemins de la racine à une feuille ont le même nombre d'arêtes noires. Combiner ces méthodes en une méthode `isRedBlackBST()` qui vérifie que l'arbre est un arbre binaire de recherche et qu'il satisfait ces deux conditions.

Exercice 5.3. Donner une méthode qui reçoit en entrée un tableau trié de n éléments et construit un arbre rouge-noir contenant les mêmes éléments en temps $O(n)$.

Exercice 5.4. Connexité du graphe des rotations. Démontrer que tout arbre binaire de recherche peut être transformé en n'importe quel autre arbre binaire de recherche sur les mêmes clés en n'utilisant que des rotations.

Exercice 5.5 (Treaps)**.** On considère une structure d'arbre binaire dans laquelle chaque nœud contient une clé et une priorité. L'arbre est ordonné comme un arbre binaire de recherche par rapport aux clés, et comme un tas par rapport aux priorités (la racine de chaque sous-arbre a la plus grande priorité du sous-arbre).

1. Montrer qu'une telle structure existe toujours et est unique, quel que soit le choix des paires clé, priorité.
2. Supposons qu'on assigne une valeur aléatoire uniforme aux priorités. Quelle est la forme de l'arbre binaire de recherche ? Que peut-on dire de la profondeur moyenne d'un nœud ?
3. Donner un algorithme d'insertion dans un treap.

Séance 6 — Parcours de graphes

Rappel (Graphes et chemins)**.** *Un* graphe *est une paire $G = (\mathcal{V}, \mathcal{E})$ de sommets et d'arêtes (i.e. $\mathcal{E} \subseteq \binom{\mathcal{V}}{2}$ si G est non dirigé, et $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ si G est dirigé). On note V la cardinalité de $\mathcal{V}$ et E la cardinalité de $\mathcal{E}$.*

Un chemin *P dans un graphe $G = (\mathcal{V}, \mathcal{E})$ est une suite ordonnée de sommets $P = (v_1, \ldots, v_n)$ tels que v_i et v_{i+1} sont adjacents pour tout $0 < i < n$. Si v_1 et v_n sont adjacents, alors P est un* cycle*. Si G n'admet pas de cycle, alors il est dit* acyclique*.*

Un chemin (ou cycle) est dit eulérien *s'il passe une unique fois par chaque arête de G, et est dit* hamiltonien *s'il passe une unique fois par chaque sommet.*

Un graphe $G = (\mathcal{V}, \mathcal{E})$ est biparti *si on peut partitionner son ensemble de sommets $\mathcal{V}$ en deux sous-ensembles disjoints $\mathcal{U}$ et $\mathcal{W}$ tels que pour toute paire de sommets adjacents (v_i, v_j), soit $v_i \in \mathcal{U}$ et $v_j \in \mathcal{W}$; soit $v_i \in \mathcal{W}$ et $v_j \in \mathcal{U}$.*

Exercice 6.1. Donner un algorithme de complexité linéaire pour les problèmes suivants, sur un graphe non-dirigé G donné en entrée :

1. Le graphe G est-il acyclique ?
2. Le graphe G est-il biparti ?
3. Le graphe G possède-t-il un cycle eulérien ?

Exercice 6.2. Que peut-on dire de la complexité des problèmes suivants ?

1. Étant donnés deux graphes G et H comportant le même nombre de sommets, est-il vrai que G et H sont identiques, à une permutation des noms des sommets près ?
2. Étant donné un graphe G, possède-t-il un cycle hamiltonien ?

*** Exercice 6.3** (Graphes 2-arête-connexes)**.** Un pont dans un graphe connexe est une arête telle que si on l'enlève, le graphe est divisé en deux composantes connexes. Un graphe sans pont est dit *2-arête-connexe*. Donner un algorithme qui détermine si un graphe donné est 2-arête-connexe.

Exercice 6.4 (Un jeu de rôle)**.** Un monstre et un joueur se trouvent chacun sur un sommet distinct d'un graphe (simple, non dirigé). Le joueur et le monstre jouent à tour de rôle. A chaque tour, ils peuvent se déplacer sur un sommet adjacent, ou rester sur le même sommet. Donner un algorithme pour déterminer tous les sommets que le joueur peut atteindre avant le monstre. On suppose que le joueur joue en premier.

Séance 7 — Connexité forte

Exercice 7.1. Quelles sont les composantes fortement connexes d'un graphe dirigé acyclique ? Expliquer comment se comporte l'algorithme de Kosaraju-Sharir dans ce cas particulier.

Exercice 7.2. Vrai ou faux ? Justifier votre réponse par une démonstration (si vrai) ou un contre-exemple (si faux) :

1. Un postordre inverse du tranposé d'un graphe dirigé G est un postordre de G.
2. Si on inverse le rôle de G et $G^\top$ dans l'algorithme de Kosaraju-Sharir, le résultat reste correct.
3. Si l'on remplace le parcours en profondeur de la seconde phase de l'algorithme de Kosaraju-Sharir par un parcours en largeur, le résultat reste correct.

Exercice 7.3. Donner un algorithme de complexité linéaire pour calculer la composante fortement connexe d'un graphe G contenant un sommet donné v.

Exercice 7.4. Un *cycle impair* est un cycle de longueur impaire. Donner un algorithme de complexité linéaire pour décider si un graphe dirigé possède un cycle (dirigé) impair. *Indice : un graphe (non-dirigé) est biparti si et seulement s'il ne possède aucun cycle impair.*

Séance 8 — Arbres couvrants

Exercice 8.1. Donner un algorithme qui, étant donné un graphe pondéré sur les arêtes, construit un arbre couvrant dont le poids maximum d'une arête est le plus petit possible.

Exercice 8.2. Donner un algorithme qui, étant donné un graphe pondéré sur les arêtes, construit un arbre couvrant dont le poids médian d'une arête est le plus petit possible.

* **Exercice 8.3.** Donner un algorithme qui, étant donné un graphe pondéré sur les arêtes, détermine si l'arbre couvrant de poids minimum est unique.

Exercice 8.4. Coupe-cycles d'arêtes. Un ensemble d'arêtes est un *coupe-cycle* s'il contient au moins une arête de chacun des cycles du graphes. Si l'on enlève toutes les arêtes d'un coupe-cycle, le graphe résultant est acyclique (une forêt). Donner un algorithme efficace pour trouver, étant donné un graphe pondéré positivement sur les arêtes, un coupe-cycle d'arêtes de poids minimum.

Exercice 8.5. Olympiades américaines d'informatique. Dans une ville, il y a n maisons, chacune nécessitant une arrivée d'eau. Le coût de construire un puits pour la maison `i` est `w[i]`. Le coût de construction d'un tuyau d'eau entre la maison `i` et la maison `j` est `c[i][j]`. Donner un algorithme qui calcule le coût minimum d'une solution d'approvisionnement de la ville en eau.

Exercice 8.6. Étant donné un graphe dont les arêtes sont soit rouges, soit vertes, et un nombre k, donner un algorithme qui construit un arbre couvrant contenant exactement k arêtes rouges, ou certifie qu'il n'en existe pas.

Exercice 8.7. Algorithme de Boruvka. On se donne la méthode suivante pour construire un arbre couvrant de poids minimum : on ajoute des arêtes à une forêt, par phases. Initialement, la forêt est constituée des sommets isolés. À chaque phase, pour chaque arbre, on trouve une arête de poids minimum connectant cet arbre à un autre. On ajoute toutes ces arêtes dans la forêt, divisant donc le nombre d'arbres de la forêt (au moins) par deux. Donner les détails d'une implémentation efficace de cet algorithme et de sa complexité.

Séance 9 — Plus courts chemins

Exercice 9.1. Prouver que pour tout plus court chemin entre deux sommets s et t passant successivement par deux autres sommets v et w, la portion du chemin comprise entre v et w est un plus court chemin entre v et w.

Exercice 9.2. Est-il vrai que si l'on ajoute un nombre c à tous les poids d'un graphe pondéré sur les arêtes, la solution du problème des plus courts chemins à une source ne change pas ?

Exercice 9.3. On se donne un graphe *chemin*, dont tous les sommets sont de degré deux, sauf deux, qui sont de degré un. Les arêtes sont pondérées positivement. Donner un algorithme qui effectue un prétraitement du graphe en temps linéaire, et permet ensuite de retrouver les distances entre toute paire de sommets en temps constant.

Exercice 9.4. Donner un algorithme de complexité linéaire pour le problème des plus courts chemins à une source dans un graphe dirigé *acyclique*.

Exercice 9.5. Le triathlon des Flandres. On considère un graphe non dirigé pondéré sur les arêtes, et dont chaque arête est d'un des trois types suivant : voie d'eau, piste cyclable, ou piste de course. On souhaite connaître tous les plus courts chemins d'un sommet source s du graphe à tous les autres, avec la condition supplémentaire que les chemins doivent être constitués d'une suite d'arêtes de type voie d'eau, suivie d'une suite d'arêtes de type piste cyclable, suivies d'une suite d'arêtes de type piste de course. Chaque suite est constituée d'au moins une arête. Donner un algorithme efficace.

Exercice 9.6. Plus court chemin avec joker. Donner un algorithme de complexité proportionnelle à $E \log V$ qui, étant donné un graphe dirigé pondéré positivement sur les arêtes et deux sommets s et t, trouve le plus court chemin de s vers t, où on est autorisé à changer le poids d'une seule arête en 0.

Séance 10 — Programmation dynamique

Exercice 10.1. Modifier la procédure de recherche de la longueur maximum d'une sous-séquence commune de façon à :

1. renvoyer non seulement la longueur mais également une telle sous-séquence ;
2. n'utiliser qu'un espace supplémentaire de taille linéaire.

Exercice 10.2. Modifier la procédure de recherche du nombre minimum de multiplications dans un produit de matrices de façon à ce qu'elle renvoie un parenthésage optimal.

Exercice 10.3. Donner un algorithme efficace pour le calcul de la longueur d'une plus longue sous-séquence commune à trois chaînes de symboles. Peut-on résoudre en temps polynomial le problème de plus longue sous-séquence commune à un nombre arbitraire k de chaînes ?

Exercice 10.4 (Plus courte super-séquence commune.)**.** On veut calculer la longueur minimum d'une chaîne de symboles qui contient deux chaînes A et B données en entrée comme sous-séquences. Montrer que ce problème se réduit à une recherche de plus court chemin sur un graphe pondéré acyclique.

Exercice 10.5. Étant donné une matrice carrée A d'entiers positifs et de taille $n \times n$, on considère les suites serpentines de A : des suites d'éléments de A tels que chaque élément est soit à droite, soit en-dessous de l'élément précédent dans la suite, et est tel que la différence avec l'élément précédent est 1 ou -1. Donner un algorithme efficace pour trouver la longueur maximum d'une suite serpentine dans la matrice A donnée en entrée.

Exercice 10.6. Étant donné un nombre k et une collection de n nombres $x_1, x_2, \ldots, x_n$, donner un algorithme de complexité $O(nk)$ qui décide s'il existe un sous-ensemble des n nombres dont la somme est égale à k. Cet algorithme est-il polynomial ?