IFT436 – Algorithmes et structures de données

Université de Sherbrooke

Examen périodique

Enseignant : Michael Blondin
Date : jeudi 19 octobre 2023
Durée : 110 minutes

Directives :

- Vous devez répondre aux questions dans le **cahier de réponses**, pas sur ce questionnaire ;
- **Une seule feuille** de notes au format 8½" × 11" est permise ;
- Les **fiches récapitulatives** des chapitres 1 à 4 se trouvent à la fin de ce questionnaire ;
- **Aucun matériel additionnel** n'est permis ;
- **Aucun appareil électronique** (calculatrice, téléphone, montre intelligente, etc.) n'est permis ;
- Vous devez donner **une seule réponse** par sous-question ;
- L'examen comporte **4 questions** sur **3 pages** valant un total de **50 points** ;
- La correction se base notamment sur la **clarté**, l'**exactitude** et la **concision** de vos réponses, ainsi que sur la **justification** pour les questions qui en requièrent une ;
- Si la **base d'un logarithme** n'est pas spécifiée, il s'agit de la base 2 : « log » dénote « $\log_2$ » ;
- Les indices d'une séquence **débutent à 1** ; autrement dit, $s = [s[1], s[2], \ldots, s[n]]$ où $n = |s|$.

Question 1 : analyse des algorithmes

(a) Soient les fonctions f_1, f_2, f_3, f_4 et f_5 ci-dessous. Donnez la complexité asymptotique de chaque f_i à l'aide d'une expression simple avec la notation $\mathcal{O}$. Ordonnez ensuite vos cinq expressions en ordre d'inclusion. 6 pts

$$\underbrace{(n^2 + 10n) \cdot \log(n^5)}_{f_1}, \quad \underbrace{\sum_{i=1}^{n} 100}_{f_2}, \quad \underbrace{\sum_{i=1}^{n} \sum_{j=i}^{n} n}_{f_3}, \quad \underbrace{\sum_{i=1}^{n} \log(2^i)}_{f_4}, \quad \underbrace{\log(n!) + n \log n}_{f_5}.$$

Clarification : une « expression simple » est de la forme $\mathcal{O}(n^c \log(n)^d)$, $\mathcal{O}(c^n)$ ou $\mathcal{O}(n!)$ avec $c, d \in \mathbb{R}_{\geq 0}$, par ex. $\mathcal{O}(1)$, $\mathcal{O}(n)$, $\mathcal{O}(n^3 \log n)$, $\mathcal{O}(2^n)$, $\mathcal{O}(n!)$, etc.

(b) Considérons un algorithme $\mathcal{A}$ dont la taille des entrées dépend des paramètres m et n. Après l'avoir analysé, nous avons obtenu $t_{\max} \in \mathcal{O}(100n + m \log n)$, où $t_{\max}$ dénote le temps d'exécution dans le pire cas de $\mathcal{A}$. Pouvons-nous simplifier et affirmer que $t_{\max} \in \mathcal{O}(m \log n)$? Justifiez. 2 pts

(c) Soit $f(n) \coloneqq n^3 + 25n - 42$. Montrez que $f \in \mathcal{O}(n^3)$ en identifiant explicitement une constante multiplicative c et un seuil n_0 de votre choix. Pour obtenir tous les points, vous devez obtenir $c \leq 4$. 2 pts

(d) Dites auxquels de ces ensembles appartient la fonction $f(n) \coloneqq 10 \cdot \log n$: 2 pts

$$\mathcal{O}(1.5^n), \quad \Theta(n), \quad \mathcal{O}((1/100) \cdot \log n), \quad \Omega(1), \quad \Omega(\log n), \quad \Omega(n^2).$$

Question 2 : tri

Considérons cet algorithme de tri :

Entrées : séquence s de $n \in \mathbb{N}$ éléments comparables
Résultat : trie s

```
1 i ← 1
2 tant que i ≤ n
3 |  si (i = 1) ∨ (s[i − 1] ≤ s[i]) alors
4 |  |  i ← i + 1
5 |  sinon
6 |  |  s[i − 1] ↔ s[i]                    // inverser le contenu de s[i − 1] et s[i]
7 |  |  i ← i − 1
8 retourner s
```

(a) Exécutez l'algorithme sur l'entrée $s = [30, 10, 20]$. Plus précisément, donnez le contenu de s et la position i après chaque tour de la boucle **tant que**. Utilisez cette notation : 3 pts

$$[\overset{\blacktriangledown}{3}0, 10, 20]$$

(b) Les affirmations suivantes sont toutes vraies. Choisissez-en *deux* (pas plus !) et montrez leur véracité. 8 pts

(i) L'algorithme termine.

(ii) L'algorithme est correct.

(iii) L'algorithme fonctionne en temps $\mathcal{O}(n^2)$ dans le pire cas.

(iv) L'algorithme fonctionne en temps $\Omega(n^2)$ dans le pire cas.

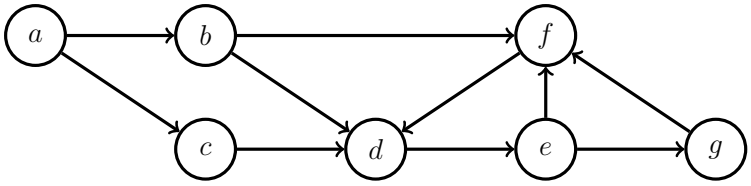
(v) L'algorithme fonctionne en temps $\mathcal{O}(n)$ dans le meilleur cas.

(vi) L'algorithme fonctionne en temps $\Omega(n)$ dans le meilleur cas.

(c) Dites si l'algorithme est stable et/ou sur place. Justifiez. 3 pts

Question 3 : graphes

(a) Considérons ce graphe dirigé $\mathcal{G}$:



(i) Donnez le contenu de $adj[b]$ et $adj[d]$, où adj est la liste d'adjacence de $\mathcal{G}$. 1 pt

(ii) Parcourez $\mathcal{G}$ en profondeur à partir du sommet a. Laissez une trace de l'ordre dans lequel les sommets sont marqués, et des arêtes qui contribuent à la découverte d'un sommet. Lors de l'exploration d'un sommet, considérez ses voisins en ordre alphabétique. 3 pts

(iii) Dites si $\mathcal{G}$ possède un ordre topologique. Si c'est le cas, donnez un tel ordre ; sinon, expliquez pourquoi il n'en existe aucun. 2 pts

(b) Soit $\mathcal{G} = (V, E)$ un graphe dirigé. Nous disons qu'un sommet $u \in V$ est *important* si, pour tout $v \in V$, nous avons $v \xrightarrow{*} u$ ou $u \xrightarrow{*} v$. Par exemple, dans le graphe de la page précédente, d est important car a, b et c peuvent atteindre d, et d peut atteindre e, f et g. Le sommet b n'est pas important car $b \not\xrightarrow{*} c$ et $c \not\xrightarrow{*} b$. 8 pts

Donnez un algorithme, sous forme de pseudocode, qui résout ce problème :

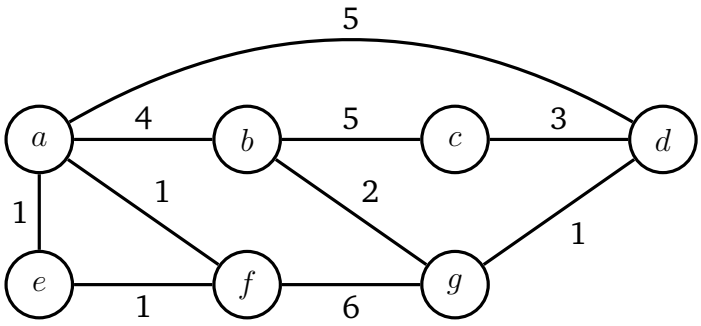
ENTRÉE : graphe dirigé $\mathcal{G} = (V, E)$ (sous forme de liste d'adjacence) et sommet $u \in V$
SORTIE : u est important ?

Analysez son temps d'exécution dans le pire cas. Pour obtenir tous les points, il doit appartenir à $\mathcal{O}(|V|+|E|)$.

Précision : au besoin, vous pouvez invoquer des algorithmes couverts en classe sans les réimplémenter.

Question 4 : algorithmes gloutons

(a) Identifiez un arbre couvrant minimal du graphe pondéré ci-dessous avec l'algorithme de Prim–Jarník (celui qui fait croître un arbre) à partir de a. Laissez une trace de l'ordre dans lequel les arêtes de l'arbre sont sélectionnées. 3 pts



(b) Existe-t-il un graphe pondéré $\mathcal{G}$ sur lequel l'algorithme de Kruskal (celui qui fusionne des arbres dans une forêt) a des choix arbitraires à faire, mais où $\mathcal{G}$ ne possède qu'un seul arbre couvrant minimal ? Justifiez. 3 pts

(c) Nous disons qu'un graphe non dirigé $\mathcal{G} = (V, E)$ est *3-coloriable* s'il est possible de colorier ses sommets en utilisant trois couleurs distinctes de telle sorte que $\texttt{couleur}(u) \neq \texttt{couleur}(v)$ pour toute arête $\{u, v\} \in E$. Autrement dit, il est interdit de colorier deux sommets adjacents avec une même couleur.

(i) Montrez que le graphe ci-dessus est 3-coloriable en identifiant un coloriage. Si vous n'avez pas de surligneurs, utilisez les lettres `R`, `V` et `B` afin de représenter les couleurs rouge, vert et bleu respectivement. 1 pt

(ii) L'algorithme glouton suivant est-il correct ? Justifiez. 3 pts

Entrées : un graphe non dirigé $\mathcal{G} = (V, E)$
Résultat : $\mathcal{G}$ est 3-coloriable ?

```
1 tant que G possède un sommet v non colorié
2 |  si il existe une couleur c ∈ {R, V, B} qui ne colore aucun sommet adjacent à v alors
3 |  |  colorier v avec c    // si plusieurs choix possibles, choisir c arbitrairement
4 |  sinon
5 |  |  retourner faux
6 retourner vrai
```

Annexe :

Fiches récapitulatives

1. Analyse des algorithmes

Temps d'exécution

- *Opérations élémentaires*: dépend du contexte, souvent comparaisons, affectations, arithmétique, accès, etc.
- *Pire cas* $t_{\max}(n)$: nombre maximum d'opérations élémentaires exécutées parmi les entrées de taille n
- *Meilleur cas* $t_{\min}(n)$: même chose avec « minimum »
- $t_{\max}(m,n)$, $t_{\min}(m,n)$: même chose par rapport à m et n

Notation asymptotique

- *Déf.*: $f \in \mathcal{O}(g)$ si $n \geq n_0 \to f(n) \leq c \cdot g(n)$ pour certains c, n_0
- *Signifie*: f croît moins ou aussi rapid. que g pour $n \to \infty$
- *Transitivité*: $f \in \mathcal{O}(g)$ et $g \in \mathcal{O}(h) \to f \in \mathcal{O}(h)$
- *Règle des coeff.*: $f_1 + \ldots + f_k \in \mathcal{O}(c_1 \cdot f_1 + \ldots + c_k \cdot f_k)$
- *Règle du max.*: $f_1 + \ldots + f_k \in \mathcal{O}(\max(f_1, \ldots, f_k))$
- *Déf.*: $f \in \Omega(g) \leftrightarrow g \in \mathcal{O}(f)$; $f \in \Theta(g) \leftrightarrow f \in \mathcal{O}(g) \cap \Omega(g)$
- *Règle des poly.*: f polynôme de degré $d \to f \in \Theta(n^d)$

Notation asymptotique (suite)

- *Simplification*: lignes élem. comptées comme une seule opér.
- *Règle de la limite*:
$$\lim_{n\to+\infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & f \in \mathcal{O}(g) \text{ et } g \notin \mathcal{O}(f) \\ +\infty & f \notin \mathcal{O}(g) \text{ et } g \in \mathcal{O}(f) \\ \text{const.} & \Theta(f) = \Theta(g) \end{cases}$$
- *Multi-params.*: $\mathcal{O}, \Omega, \Theta$ étendues avec plusieurs seuils

Correction et terminaison

- *Correct*: sur *toute* entrée x qui satisfait la pré-condition, x et sa sortie y satisfont la post-condition
- *Termine*: atteint instruction **retourner** sur *toute* entrée
- *Invariant*: propriété qui demeure vraie à chaque fois qu'une ou certaines lignes de code sont atteintes

Exemples de complexité

$$\mathcal{O}(1) \subset \mathcal{O}(\log n) \subset \mathcal{O}(n) \subset \mathcal{O}(n \log n) \subset \mathcal{O}(n^2) \subset \mathcal{O}(n^2 \log n) \subset \mathcal{O}(n^3) \subset \mathcal{O}(n^d) \subset \mathcal{O}(2^n) \subset \mathcal{O}(3^n) \subset \mathcal{O}(b^n) \subset \mathcal{O}(n!)$$

2. Tri

Approche générique

- *Inversion*: indices (i,j) t.q. $i < j$ et $s[i] > s[j]$
- *Progrès*: corriger une inversion en diminue la quantité
- *Procédure*: sélectionner et corriger une inversion, jusqu'à ce qu'il n'en reste plus

Algorithmes (par comparaison)

- *Insertion*: considérer $s[1:i-1]$ triée et insérer $s[i]$ dans $s[1:i]$
- *Monceau*: transformer s en monceau et retirer ses éléments
- *Fusion*: découper s en deux, trier chaque côté et fusionner
- *Rapide*: réordonner autour d'un pivot et trier chaque côté

Propriétés

- *Sur place*: n'utilise pas de séquence auxiliaire
- *Stable*: l'ordre relatif des éléments égaux est préservé

Sommaire

Algorithme	Complexité (par cas) meilleur	moyen	pire	Sur place	Stable
insertion	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	✓	✓
monceau	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	✓	✗
fusion	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	✗	✓
rapide	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n^2)$	✓	✗

Usage

- *Petite taille*: tri par insertion
- *Grande taille*: tri par monceau ou tri rapide
- *Grande taille + stabilité*: tri par fusion

Tri sans comparaison

- *Par comparaison*: barrière théorique de $\Omega(n \log n)$
- *Sans comparaison*: possible de faire mieux pour certains cas
- *Représentation binaire*: trier (de façon stable) en ordonnant du bit de poids faible vers le bit de poids fort
- *Complexité*: $\Theta(mn)$ où $m =$ nombre de bits et $n = |s|$

3. Graphes

Graphes

- *Graphe*: $\mathcal{G} = (V, E)$ où $V =$ sommets et $E =$ arêtes
- *Dirigé vs. non dirigé*: $\{u, v\} \in E$ vs. $(u, v) \in E$
- *Degré (cas non dirigé)*: $\deg(u) = \#$ de voisins
- *Degré (cas dirigé)*: $\deg^-(u) = \#$ préd., $\deg^+(u) = \#$ succ.
- *Taille*: $|E| \in \Theta($somme des degrés$)$ et $|E| \in \mathcal{O}(|V|^2)$
- *Chemin*: séq. $u_0 \to \cdots \to u_k$ (taille $= k$, simple si sans rép.)
- *Cycle*: chemin de u vers u (simple si sans rép. sauf début/fin)
- *Sous-graphe*: obtenu en retirant sommets et/ou arêtes
- *Composante*: sous-graphe max. où sommets access. entre eux

Parcours

- *Profondeur*: explorer le plus loin possible, puis retour (pile)
- *Largeur*: explorer successeurs, puis leurs succ., etc. (file)
- *Temps d'exécution*: $\mathcal{O}(|V| + |E|)$

Représentation

$$\begin{array}{c|ccc} & a & b & c \\ a & 0 & 1 & 1 \\ b & 1 & 0 & 0 \\ c & 0 & 1 & 0 \end{array} \quad \begin{array}{l} [a \mapsto [b,c], \\ b \mapsto [a], \\ c \mapsto [b]] \end{array}$$

	Mat.	Liste (non dirigé)	Liste (dir.)
$u \to v$?	$\Theta(1)$	$\mathcal{O}(\min(\deg(u), \deg(v)))$	$\mathcal{O}(\deg^+(u))$
$\{v : u \to v\}$	$\Theta(\lvert V\rvert)$	$\mathcal{O}(\deg(u))$	$\mathcal{O}(\deg^+(u))$
$\{u : u \to v\}$	$\Theta(\lvert V\rvert)$	$\mathcal{O}(\deg(v))$	$\mathcal{O}(\lvert V\rvert + \lvert E\rvert)$
Modif. $u \to v$	$\Theta(1)$	$\mathcal{O}(\deg(u) + \deg(v))$	$\mathcal{O}(\deg^+(u))$
Mémoire	$\Theta(\lvert V\rvert^2)$	$\Theta(\lvert V\rvert + \lvert E\rvert)$	

Propriétés et algorithmes

- *Plus court chemin*: parcours en largeur + stocker préd.
- *Ordre topologique*: $u_1 \preceq \cdots \preceq u_n$ où $i < j \implies (u_j, u_i) \notin E$
- *Tri topologique*: mettre sommets de degré 0 en file, retirer en mettant les degrés à jour, répéter tant que possible
- *Détec. de cycle*: tri topo. + vérifier si contient tous sommets
- *Temps d'exécution*: tous linéaires

Arbres

- *Arbre*: graphe connexe et acyclique (ou prop. équivalentes)
- *Forêt*: graphe constitué de plusieurs arbres
- *Arbre couv.*: arbre avec tous sommets d'un graphe $\mathcal{G}$ non dirigé; possible ssi $\mathcal{G}$ connexe; se trouve avec parcours en prof.

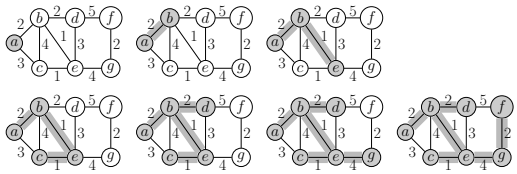
4. Algorithmes gloutons

Arbres couvrants minimaux

- *Graphe pondéré*: $\mathcal{G} = (V, E)$ où $p[e]$ est le poids de l'arête e
- *Poids d'un graphe*: $p(\mathcal{G}) = \sum_{e \in E} p[e]$
- *Arbre couv. min.*: arbre couvrant de $\mathcal{G}$ de poids minimal

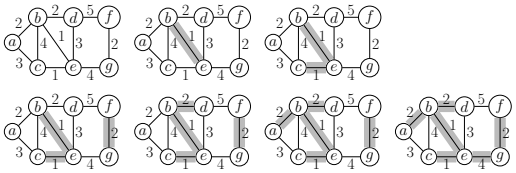
Algorithme de Prim–Jarník

- *Approche*: faire grandir un arbre en prenant l'arête min.
- *Complexité*: $\mathcal{O}(|E| \log |V|)$ avec monceau



Algorithme de Kruskal

- *Approche*: connecter forêt avec l'arête min. jusqu'à un arbre
- *Complexité*: $\mathcal{O}(|E| \log |V|)$ avec ensembles disjoints



Ensembles disjoints

- *But*: manipuler une partition d'un ensemble V
- *Représentation*: chaque ensemble sous une arborescence

Algorithme glouton

1) *Choisir* un candidat c itérativement (*sans reconsidérer*)
2) Ajouter c à solution partielle S si *admissible*
3) Retourner S si *solution (complète)*, « impossible » sinon

Problème du sac à dos

- *But*: choisir objets pour maxim. valeur sans excéder capacité
- *Algo. glouton*: trier en ordre décroissant par $val[i]/poids[i]$
- *Fonctionne* si on peut découper objets (sinon approx. à ½)