IFT436 – Algorithmes et structures de données

Université de Sherbrooke

Examen périodique

Enseignant: Michael Blondin
Date: lundi 18 octobre 2021
Durée: 110 minutes

Directives:

— Vous devez répondre aux questions dans le **cahier de réponses**, *pas* sur ce questionnaire;
— **Une seule feuille** de notes manuscrites au format 8½" × 11" est permise;
— **Aucun matériel additionnel** (notes de cours, fiches récapitulatives, etc.) 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 **6 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, $s = [s[1], s[2], \ldots, s[n]]$ si $n = |s|$.

Question 1: analyse des algorithmes

(a) Ordonnez ces fonctions f_1, f_2, f_3, f_4 et f_5 selon la notation $\mathcal{O}$ (de la plus faible vers la plus haute complexité): 7 pts

$$\underbrace{4 \cdot \sum_{i=0}^{n} \left(\frac{2^i}{5} + 3 \right)}_{f_1}, \quad \underbrace{\log(n^{42}) + \frac{2}{9} n^4}_{f_2}, \quad \underbrace{\frac{n!}{(n-2)! \cdot 2!}}_{f_3}, \quad \underbrace{\frac{4}{3} \cdot \sum_{i=1}^{n} (6 + \log n)}_{f_4}, \quad \underbrace{\sum_{i=1}^{n} \sum_{j=1}^{n} (3i + 2j)}_{f_5}.$$

$$\underbrace{\mathcal{O}(f_4)}_{\mathcal{O}(n \log n)} \subset \underbrace{\mathcal{O}(f_3)}_{\mathcal{O}(n^2)} \subset \underbrace{\mathcal{O}(f_5)}_{\mathcal{O}(n^3)} \subset \underbrace{\mathcal{O}(f_2)}_{\mathcal{O}(n^4)} \subset \underbrace{\mathcal{O}(f_1)}_{\mathcal{O}(2^n)}$$

(b) Considérons un algorithme $\mathcal{A}$ qui reçoit en entrée un graphe non dirigé connexe de n sommets et m arêtes. Après l'avoir analysé, nous avons obtenu $t_{\max} \in \Omega(3m + 5 \log n - 2)$, où $t_{\max}$ dénote le temps d'exécution dans le pire cas de $\mathcal{A}$. Pouvons-nous affirmer que $t_{\max} \in \Omega(n)$? Justifiez. 2 pts

Oui. On a $t_{\max} \in \Omega(m + \log n)$. Par connexité, on a $m \geq n - 1$. Ainsi, $t_{\max} \in \Omega(n)$.

(c) Soit $f(n) \coloneqq 3n^2 + 5n - \log(n + 1)$. Montrez que $f \in \mathcal{O}(n^2)$ en identifiant explicitement une constante multiplicative c et un seuil n_0. Pour obtenir tous les points, vous devez choisir $c = 4$, sinon vous obtiendrez la moitié des points. 2 pts

On a $f \in \mathcal{O}(n^2)$ avec $c \coloneqq 4$ et $n_0 \coloneqq 5$ car:

$$\begin{aligned} f(n) &= 3n^2 + 5n - \log(n+1) \\ &\leq 3n^2 + 5n && \forall n \geq 0 \\ &\leq 3n^2 + n \cdot n && \forall n \geq 5 \\ &= 4n^2. \end{aligned}$$

(d) Dites auxquels de ces ensembles appartient la fonction $\sqrt{n}$: 2 pts

$$\mathcal{O}(2^n), \quad \mathcal{O}(n), \quad \Omega(3 \cdot \sqrt{n}), \quad \Theta(n), \quad \mathcal{O}(\log n), \quad \mathcal{O}(n^2), \quad \Omega(1).$$

$\mathcal{O}(2^n),\ \mathcal{O}(n),\ \Omega(3 \cdot \sqrt{n}),\ \mathcal{O}(n^2),\ \Omega(1)$

Question 2: tri

Considérons l'algorithme de tri suivant:

Entrée: séquence s de $n \in \mathbb{N}$ éléments comparables
Sortie: s triée

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

(a) Exécutez l'algorithme sur l'entrée $s = [22, 33, 11]$. Donnez une trace qui illustre clairement le contenu de la séquence s et du compteur i à la fin de chaque tour de la boucle **tant que**. 3 pts

s (i **en gras**)
$[22, \mathbf{33}, 11]$
$[22, 11, \mathbf{33}]$
$[22, \mathbf{11}, 33]$
$[\mathbf{22}, 11, 33]$
$[11, \mathbf{22}, 33]$
$[\mathbf{11}, 22, 33]$
$[11, 22, 33]$

(b) Les affirmations suivantes sont toutes vraies. Choisissez en *deux* (pas plus!) et montrez leur véracité. 7 pts

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

Considérons la famille d'entrées $[n, \ldots, 3, 2, 1]$. Après 2 itérations, on a $[n, \ldots, 3, 1, \mathbf{2}]$, après 4 autres itérations on a $[n, \ldots, 1, 2, \mathbf{3}]$, etc. En général, après $2i$ autres itérations on a $[n, \ldots, 1, 2, \ldots, \mathbf{i}]$. Ainsi, nous obtenons $\Omega(\sum_{i=1}^{n-1} 2i + n) = \Omega(n^2)$.

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

Cet algorithme est le tri par insertion, mais avec insertion à droite et avec les deux boucles combinées en une. La même logique s'applique: on maintient $s[k+1:n]$ triée, et on insère l'élément $s[k]$ au bon endroit vers la droite en au plus $n-k$ déplacements, et $n-k+1$ déplacements du compteur vers la gauche. Cela donne donc un temps d'exécution $\mathcal{O}(\sum_{k=1}^{n}(2(n-k)+1)) = \mathcal{O}(\sum_{k=0}^{n-1}(2k+1)) = \mathcal{O}(n^2)$.

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

On termine lorsque $i < 1$, et i décroît d'au plus un par itération. Il y a donc au moins n itérations.

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

Sur la famille $[1, 2, \ldots, n]$, on atteint toujours le **sinon** car il n'y a pas inversion. Ainsi, il y a exactement n itérations d'un bloc élémentaire.

(v) L'algorithme termine.

À chaque itération, on corrige une inversion, ou on décrémente i. Comme le nombre d'inversions est fini, à partir d'un point on décrémente forcément i, et on quitte donc éventuellement la boucle.

(vi) L'algorithme est correct.

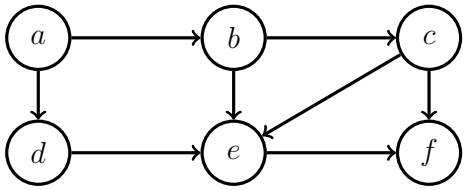
La boucle satisfait l'invariant « $s[i+1:n]$ est triée », car $[\,]$ est triée, et parce que le bloc **si** s'assure de préserver la propriété. Si on termine avec $i = 0$ (seule possibilité), on obtient « $s[1:n]$ est triée ».

(c) Dites si l'algorithme est stable et/ou s'il est sur place. Justifiez. 4 pts

Sur place: Oui, car aucune séquence auxiliaire n'est utilisée.
Stable: Oui, l'ordre relatif des éléments est préservé car on corrige des *inversions consécutives*.

Question 3: graphes

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



(i) Donnez la *matrice d'adjacence* de $\mathcal{G}$ et le degré entrant de chacun de ses sommets. 2 pts

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad \deg^- = [0, 1, 1, 1, 3, 2]$$

(ii) 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

Oui, par ex. $[a, b, c, d, e, f]$.

(iii) Si l'on retirait la direction des arêtes de $\mathcal{G}$, le graphe non dirigé résultant aurait-il un arbre couvrant? Justifiez. 2 pts

Oui, car le graphe serait connexe et qu'un graphe possède un arbre couvrant ssi il est connexe.

(b) Soit $\mathcal{G}$ un graphe dirigé acyclique. Si l'on renverse la direction des arêtes de $\mathcal{G}$, obtient-on forcément un graphe dirigé acyclique? Justifiez. 2 pts

Oui. Supposons qu'il existe un cycle $v_1 \to v_2 \to \cdots \to v_k = v_1$ dans le graphe renversé $\mathcal{G}'$. Nous obtenons un cycle $v_1 = v_k \to \cdots \to v_2 \to v_1$ dans $\mathcal{G}$, ce qui est une contradiction.

(c) Un *point d'articulation* d'un graphe non dirigé connexe $\mathcal{G}$ est un sommet v dont le retrait rend le graphe non connexe. Donnez un algorithme, sous forme de pseudocode, qui résout le problème suivant: 5 pts

ENTRÉE: graphe non dirigé connexe $\mathcal{G} = (V, E)$ (sous forme de liste d'adjacence) et sommet $v \in V$
SORTIE: v est-il un point d'articulation de $\mathcal{G}$?

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

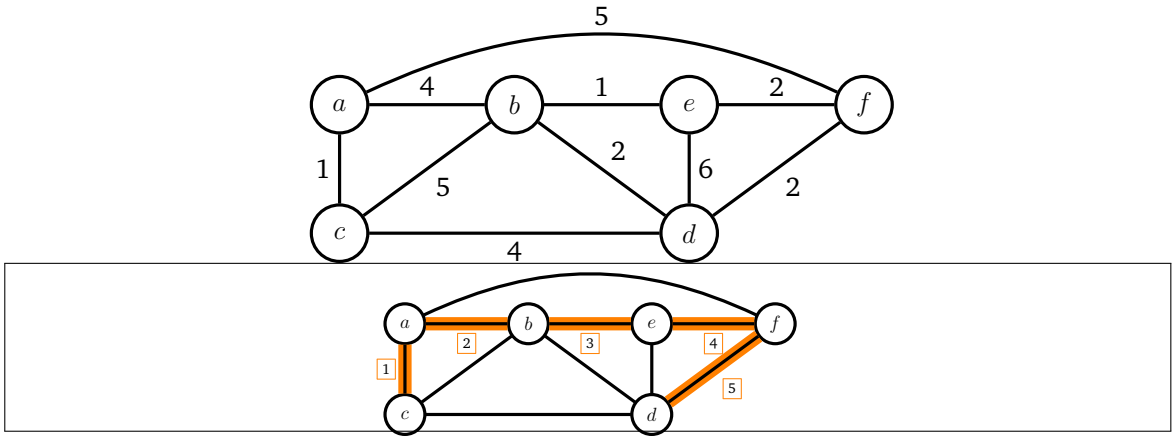
Précision: au besoin, vous pouvez invoquer des algorithmes couverts en classe sans les réimplémenter.

Entrée: graphe non dirigé $\mathcal{G} = (V, E)$ connexe représenté par la liste d'adjacence adj; $v \in V$
Sortie: v est un point d'articulation?

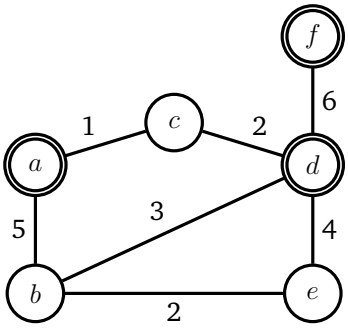
```
// Retirer arêtes adjacentes à v, et choisir sommet u ≠ v quelconque
adj' ← [x ↦ [] : x ∈ V]                                   // O(|V|)
pour x ∈ V                                                // O(Σ_{x∈V}(1 + deg(x))) = O(|V| + |E|)
    pour y ∈ adj[x]
        si x ≠ v et y ≠ v alors
            ajouter y à adj'[x]
            u ← x
// u peut atteindre tous les sommets (sauf v)?
X ← parcours-en-profondeur(adj', u)                       // O(|V| + |E|)
retourner |X| < |V| − 1                                   // O(1)
```

Question 4: algorithmes gloutons

(a) Identifiez un arbre couvrant minimal de ce graphe pondéré avec l'algorithme de Prim–Jarník. Laissez une trace de l'ordre dans lequel les arêtes de l'arbre sont sélectionnées. 3 pts



(b) Un *graphe terminal* est un graphe non dirigé $\mathcal{G} = (V, E)$ muni d'un ensemble de sommets $T \subseteq V$ dits *terminaux*. Un *arbre terminalisant* de $\mathcal{G}$ est un sous-graphe qui est un arbre et qui contient tous les sommets de T (et possiblement d'autres de V). Lorsque $\mathcal{G}$ est pondéré, on peut chercher à identifier un *arbre terminalisant minimal*. Par exemple, le graphe terminal pondéré ci-dessous est muni des sommets terminaux $T = \{a, d, f\}$. Ses arêtes $\{\{a, b\}, \{b, e\}, \{d, e\}, \{d, f\}\}$ forment un arbre terminalisant de poids 17.



Entrées: graphe non dirigé connexe $\mathcal{G} = (V, E)$ avec sommets terminaux $T \subseteq V$, et où E est pondéré par p
Résultat: un arbre terminalisant minimal de $\mathcal{G}$

$V' \leftarrow \emptyset$; $E' \leftarrow \emptyset$
trier E en ordre croissant selon r, puis selon p en bris d'égalité
considérer chaque sommet $v \in V$ comme un arbre
pour $\{u, v\} \in E$
  si (V', E') *forme un arbre et* $T \subseteq V'$ **alors**
    quitter la boucle
  sinon si u *et* v *n'appartiennent pas au même arbre* **alors**
    fusionner les arbres contenant u et v
    ajouter u et v **à** V'; **ajouter** $\{u, v\}$ **à** E'
retourner (V', E')

Considérons l'algorithme ci-dessus. Il associe à chaque arête e une *priorité* $r[e]$ définie comme suit: $r[e] := 0$ si les deux sommets de e sont terminaux, $r[e] := 1$ si un seul sommet de e est terminal, et $r[e] := 2$ sinon. Il procède à la manière de l'algorithme de Kruskal, mais en triant les arêtes selon leur priorité, puis leur poids.

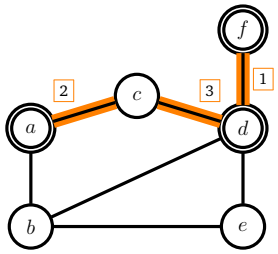
(i) Si l'on a accès à un algorithme de calcul d'arbre terminalisant minimal, comment peut-on l'utiliser afin de calculer un arbre couvrant minimal? 1 pt

Il suffit de l'appeler avec $T = V$ (ainsi tous les sommets seront couverts).

(ii) Exécutez l'algorithme ci-dessus sur le graphe terminal ci-dessus en laissant une trace de l'ordre dans 2,5 pts

lequel les arêtes de l'arbre sont sélectionnées. Dites si l'arbre terminalisant est minimal. Justifiez.

On obtient cet arbre terminalisant de poids 9. Il est minimal car: $\{d, f\}$ doit absolument en faire partie (seule arête qui touche à f), et $\{a, c\}$ aussi (car l'alternative $\{a, b\}$ est plus coûteuse).



(iii) Dites si l'algorithme ci-dessus est correct. Justifiez. 3,5 pts

Non, par exemple sur ce graphe on obtient un arbre terminalisant de poids 3, alors qu'il y en a un de poids 2: