IFT436 – Algorithmes et structures de données

Université de Sherbrooke

Devoir 2

Enseignant:	Michael Blondin
Date de remise:	lundi 7 octobre 2024 à 23h59
À réaliser:	en équipe de trois ou moins
Modalités:	remettre en ligne sur Turnin
Bonus:	les questions bonus sont indiquées par ★
Pointage:	max. 50 points + 5 points bonus

Question 1: Tri

Pour cette question, nous considérons (comme dans les notes de cours) que les éléments d'une séquence s de n éléments sont numérotés de 1 à n, c.-à-d. $s = [s[1], s[2], \ldots, s[n]]$. Considérons cet algorithme de tri:

Entrées : séquence T de $n \in \mathbb{N}_{>0}$ éléments comparables
Résultat : séquence T triée

```
1  trier(T):
2      corriger-inv(j, k):                          // sous-routine avec accès à T
3          si T[j] > T[k] alors
4              T[j] ↔ T[k]                          // inverser le contenu de T[j] et T[k]
5              retourner faux
6          sinon
7              retourner vrai
8      m ← n ÷ 2
9      faire
10         terminé ← vrai
11         pour i ∈ [1..m] faire
12             terminé ← corriger-inv(2i − 1, 2i) ∧ terminé
13         pour i ∈ [1..m] faire
14             si 2i < n alors
15                 terminé ← corriger-inv(2i, 2i + 1) ∧ terminé
16     tant que ¬terminé
17     retourner T
```

(a) Afin de vous familiariser avec l'algorithme, pour chacune des entrées suivantes, exécutez `trier`(T) et donnez le contenu de T à la fin de chaque tour de la boucle **faire ... tant que**: 2 pts

— $T = [66, 99, 100, 88, 77, 200]$;

— $T = [80, 20, 30, 40, 50, 60, 10]$.

(b) Expliquez brièvement, en mots, le fonctionnement de l'algorithme. 2 pts

(c) Expliquez pourquoi si une séquence s de n éléments n'est *pas* triée, alors elle contient une inversion (i, j) telle que $j = i + 1$. 3 pts

(d) Expliquez pourquoi l'algorithme termine et est correct. 3 pts

Indice: considérez (c) *et les propriétés des inversions vues en classe.*

(e) Analysez le temps d'exécution dans le pire cas afin de montrer qu'il appartient à $\mathcal{O}(n^3)$. 3 pts

Attention: établir $\mathcal{O}(n^2)$ est bien plus difficile.

(f) Argumentez que le temps d'exécution dans le pire cas appartient à $\Omega(n^2)$. 4 pts

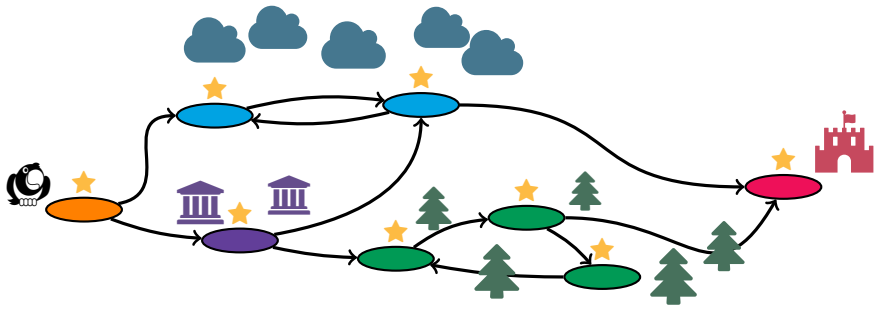
Indice: pensez à une famille de « mauvaises entrées ».

(g) Le temps d'exécution de l'algorithme dans le meilleur cas appartient-il à $\mathcal{O}(n)$? Justifiez votre réponse. 3 pts

★ Montrez que le temps d'exécution dans le pire cas appartient à $\Theta(n^2)$. ★ 2 pts

Question 2: Graphes

Considérons un jeu vidéo non linéaire où la complétion d'un niveau permet de passer à un niveau subséquent parmi certains choix. La première fois qu'on complète un niveau, on obtient une étoile. Il est possible de rejouer un niveau si on y revient, mais cela n'augmente pas le nombre d'étoiles obtenues. Par exemple, supposons que le jeu soit constitué de huit niveaux organisés de cette manière:



Le personnage, qui débute au premier niveau (à gauche), peut obtenir quatre étoiles en complétant les deux niveaux du ciel, puis en se déplaçant au château (le dernier niveau à droite). Toutefois, la solution qui maximise le nombre d'étoiles visite la cité, suivie d'un tour des trois niveaux de la forêt, complété d'une visite du château.

Cherchons à automatiser la recherche d'un tel score maximal. Nous supposons qu'un jeu est représenté par un graphe dirigé de n sommets et m arêtes, sous forme de liste d'adjacence. Pour chacune des questions suivantes, vous pourrez obtenir tous les points si votre algorithme fonctionne en temps $\mathcal{O}(n+m)$ dans le pire cas.

(a) Nous disons qu'un jeu est *bien conçu* si: 5 pts

— il possède un (seul) niveau accessible à partir d'aucun autre (le *premier niveau*);
— il possède un (seul) niveau qui ne peut pas en atteindre d'autres (le *dernier niveau*);
— tous les niveaux sont accessibles à partir du premier niveau;
— tous les niveaux peuvent atteindre le dernier niveau.

Expliquez comment déterminer algorithmiquement si un jeu est bien conçu. Vous pouvez le faire en mots, sous forme de pseudocode, en invoquant des algorithmes vus en classes, etc.

(b) Nous disons que deux niveaux x et y appartiennent au même *monde* si x peut atteindre y et vice-versa, avec un nombre arbitraire de déplacements. Par exemple, il y a cinq mondes dans le jeu ci-dessus. En particulier, les mondes du ciel, de la cité et de la forêt contiennent respectivement deux, un et trois niveaux.

Supposons qu'on ait accès à une routine qui répond, en temps constant, aux questions de cette forme: 7 pts

le niveau x peut-il atteindre le niveau y avec un nombre arbitraire de déplacements?

Donnez un algorithme qui identifie les mondes d'un jeu bien conçu. Il doit numéroter chaque niveau avec un nombre de $[1..k]$, où k est le nombre de mondes. L'ordre dans lequel ils sont attribués n'importe pas, pourvu que deux niveaux aient le même nombre ssi ils font partie du même monde.

(c) **Donnez un algorithme qui détermine le nombre *maximal* d'étoiles qu'on peut obtenir dans un jeu bien conçu, où chaque monde ne contient qu'*un seul* niveau.** 8 pts

Indice: pensez à un ordre dans lequel considérer les niveaux, puis calculez le nombre maximal pour chaque niveau.

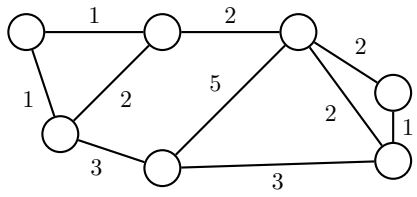
★ Donnez un algorithme qui effectue la même tâche, mais cette fois sans restriction sur la taille des mondes. ★ 2 pts

Question 3: Algorithmes gloutons

Une entrepreneure compte aménager un site de glamping écoresponsable de n gîtes dans les Cantons-de-l'Est, où n sera déterminé selon la subvention accordée par le Ministère du Tourisme. Afin d'accommoder le plus grand nombre de personnes, l'entrepreneure désire qu'on puisse se déplacer de chaque gîte vers chaque autre gîte à partir de chemins pavés. Elle vous embauche donc afin de l'épauler dans la conception du site.

(a) Étant donné des chemins pavés, on vous demande de vérifier rapidement s'il y a un chemin entre chaque paire de gîtes, mais pas de chemin superflu. Autrement dit, **donnez un algorithme qui détermine si un graphe non dirigé est un *arbre***. Expliquez brièvement pourquoi l'algorithme fonctionne, et analysez son temps d'exécution. Votre algorithme doit fonctionner en temps $\mathcal{O}(n)$ dans le pire cas, où n est le nombre de sommets. Considérez l'entrée comme un graphe représenté par une liste d'adjacence. 3 pts

(b) Le réseau de sentiers ci-dessous est finalement retenu. Quelle sera la longueur du sous-réseau pavé? Autrement dit, **quel est le poids d'un arbre couvrant minimal? Justifiez en exécutant graphiquement l'algorithme de Prim–Jarník, puis de Kruskal. Dites s'il y a plus d'une façon d'obtenir ce poids.** 4 pts



(c) Afin d'obtenir la subvention, le ministère requiert qu'il soit facile d'accéder à un téléphone d'urgence. Informellement, un téléphone doit nécessairement se trouver à au plus un sentier (pavé ou non) de chaque gîte. Formellement, chaque gîte u doit satisfaire cette condition: u est équipé d'un téléphone, ou tous les gîtes $\{v : u \to v\}$ possèdent un téléphone. **L'algorithme glouton suivant est-il correct? Justifiez.** 3 pts

---

Entrées : un graphe non dirigé $\mathcal{G} = (V, E)$
Sorties : la plus petite quantité de téléphones à installer pour obtenir une subvention

$T \leftarrow [v \mapsto \texttt{faux} : v \in V]$; $c \leftarrow 0$

tant que V *est non vide* **faire**
  retirer un sommet x de V
  si $\neg T[x]$ *et* ($\neg T[y]$ *pour au moins une certaine arête* $\{x, y\} \in E$) **alors** $T[x] \leftarrow \texttt{vrai}$; $c \leftarrow c + 1$

retourner c

★ Donnez un algorithme qui résout le problème suivant en temps $\mathcal{O}(k)$: ★ 1 pt

ENTRÉE: une forêt $\mathcal{F} = (V, E)$ sous forme de liste d'adjacence, et un entier $1 \leq k < |V|$, tels que $\deg(v) > 0$ pour tout $v \in V$
SORTIE: $\mathcal{F}$ contient-elle au moins $|V| - k$ arbres?

Considérez l'arithmétique, ainsi que ces opérations sur les séquences, comme élémentaires: l'ajout, l'accès, et l'obtention de la taille. Analysez le temps d'exécution de l'algorithme et expliquez pourquoi il est correct.