Deuxième Tour Théorique

Tâches

Swiss Olympiad in Informatics

9 mars 2019

Indications

— Ouvre le livret seulement quand tu y es invité. L'épreuve commence en même temps pour tous les participants et dure cinq heures.
— Tous les appareils électroniques sont interdits, à l'exception des montres (sauf montres intelligentes). Éteins ton téléphone portable.
— Commence chaque exercice sur une nouvelle feuille et écris ton nom sur chaque feuille. Numérote les feuilles et trie les avant la reddition.
— N'utilise pas de crayon et n'écris pas en rouge.
— Écris lisiblement.
— Regarde bien chaque sous-tâche, même quand tu n'as pas pu résoudre une des précédentes. Quelques sous-tâches peuvent être résolues sans l'aide des précédentes.

Évaluation

Ta solution est évaluée en fonction de son exactitude, de sa complexité spatio-temporelle ainsi que de la justification de ces éléments. Nous attendons de toi que tu fournisses une preuve ou une esquisse de preuve de l'exactitude et de la complexité de ta solution.

Si un algorithme t'est demandé, tu devrais utiliser cette structure comme ligne directrice :

1. Décris l'idée derrière ton algorithme aussi clairement que possible.
2. Explique pourquoi ton approche résout correctement l'exercice.
3. Analyse la complexité spatio-temporelle asymptotique de ta solution.
4. Écris en pseudocode une solution. Tu peux omettre les parties faciles comme l'entrée et la sortie et utiliser des expressions mathématiques.

Si une partie de ta solution est utile pour plusieurs sous-tâches, il suffit de l'écrire une seule fois et de s'y référer à partir de là.

Bonne chance !

Recommendation pour fromage

La souris Stofl fait un apprentissage avec le Maître Fromage JK. Malheureusement, JK ne divulgue rien quand il s'agit de comment choisir deux types de fromages qui vont bien ensemble. Stofl sait que chaque type de fromage i a deux charactéristiques a_i, b_i que JK détermine avec son excellent sens de l'odorat. Stofl pense qu'il a trouvé la formule pour déterminer à quel point deux types de fromages vont bien ensemble :

$$\frac{\left|a_i - a_j\right| \cdot b_i \cdot b_j}{\max(|b_i|, \left|b_j\right|)} \tag{1}$$

Dans cette formule, $|x|$ dénote la valeur absolue de x (x si $x \geq 0$ et $-x$ sinon) et $\max(x, y)$ dénote le maximum de x et y (x si $x \geq y$ et y si $x < y$).

On appelle le résultat de cette formule la note de recommandation. Le plus haut la note de recommandation, le mieux les fromages i et j vont ensemble. Tu peux aussi combiner un type de fromage avec lui-même. Dans ce cas, la note de recommendation sera de 0 [1]. Il peut aussi y avoir des formages qui ne vont pas du tout ensemble et dans ce cas, le score sera négatif.

Souris Stofl voudrait impressionner JK et aimerait que tu écrives un programme pour lui qui détermine le plus rapidement possible quelle est la plus haute note de recommendation possible.

Description formelle Tu reçois comme entrée le nombre de types de fromages différents n ($2 \leq n$). Pour chaque type de fromage, il y a deux entiers a_i, b_i ($1 \leq i \leq n$). Ta tâche est de trouver la plus grande valeur possible pour l'expression (1).

Sous-tâche 1: Création d'une recommandation (10 points)

Aujourd'hui, il y a 9 types de fromages en stock. Quels sont les deux fromages que Stofl doit combiner pour que la note de recommendation soit aussi haute que possible ?

i	0	1	2	3	4	5	6	7	8
$\mathbf{a_i}$	1	-1	5	-2	3	5	-6	4	0
$\mathbf{b_i}$	5	1	2	4	7	-3	-2	3	4

Sous-tâche 2: Choisir deux types de fromages (70 points)

Le lendemain, il y a tant de différents types de fromages que Stofl ne peut plus faire les calculs à la main. Cependant, il remarque que la valeur b_i de chaque type de fromages est positive ($b_i \geq 0$).

Ecris un algorithme qui est asymptotiquement aussi efficace que possible pour résoudre ce problème. Tu peux supposer qu'un tableau $k[1 \cdots n]$ est donné, dans lequel chaque élément a deux attributs : a et b, les deux caractéristiques de ce type de fromage.

1. Parce que $|a_i - a_i| = 0$.

Sous-tâche 3: Caractéristiques négatives (20 points)

Le jour suivant, il y a aussi des types de fromages avec une caractéristique b_i négative. Si l'on suppose que nous avons trouvé une solution efficace pour la sous-tâche 2, comment doit-on modifier l'algorithme précédent pour qu'il marche dans le cas général ?

Note : Même si tu n'as pas pu résoudre la sous-tâche 2, tu peux supposer qu'un algorithme pour le problème avec b_i positif est donné.

Grenades

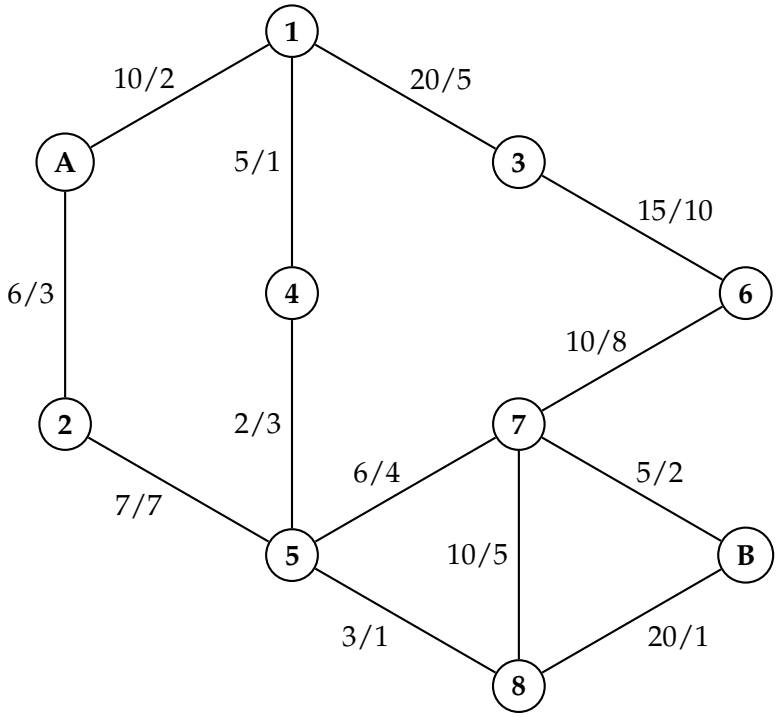
La souris Ada est l'organisatrice de l'OSI (Olympiade des Souris d'Informatique) à Bakou. Une de ses tâches consiste à transporter autant de grenades que possible depuis un hangar jusqu'au lieu de compétition pour fournir un goûter aux participants et participantes. Pour amener les grenades en un temps T d'un hangar au lieu de compétition, Ada utilise un camion dans lequel n'importe quelle quantité de grenades peut rentrer. Le problème est le suivant : chaque rue de Bakou a une limite de poids que le camion n'a pas le droit de dépasser pour y circuler, et les grenades pèsent évidemment plutôt lourd...

Aide Ada à trouver un chemin qui est plus court que T et par lequel elle peut faire passer un nombre maximal de grenades.

Description formelle Soit un graphe, dans lequel chaque arête est décrite par deux nombres entiers w et t. Trouve le poids maximal W, tel qu'il y a un chemin de A vers B, dont la longueur (la somme des t des arêtes qui le composent) est $\leq T$, et que pour toute arête dans le chemin, $w \geq W$.

Il est garanti que le graphe est connexe et qu'il y a toujours un chemin de A à B qui est égal ou inférieur à T.

Sous-tâche 1: Résoudre un exemple (10 points)



Quel est le poids maximal pour que le camion puisse circuler du hangar jusqu'au lieu

de compétition en $T = 18$ sans dépasser une limitation de poids ? Le hangar est marqué avec A, le lieu de compétition avec B. À côté de chaque arête sont écrits deux nombres, w/t, celui de gauche dénotant la limite de poids et celui de droite le temps que le camion prend à la parcourir. Explique rapidement pourquoi il n'y a pas de meilleure solution.

Sous-tâche 2: Temps infini (30 points)

Nous considérons pour l'instant que l'heure d'arrivée des grenades sur le lieu de compétition n'importe que peu, donc $T = \infty$. Les limites de poids pour les rues sont des nombres entiers positifs.

Sous-tâche 3: Deux poids (20 points)

Le temps est de nouveau compté. Cependant, chaque rue a maintenant comme limite de poids 1 ou 2, c'est-à-dire $1 \leq w \leq 2$.

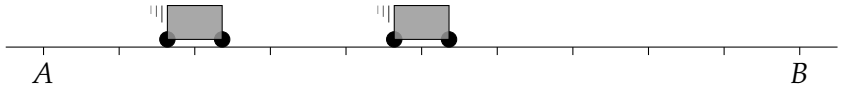
Sous-tâche 4: Cas général (40 points)

La limite de temps reste en place et les limites de temps sont à nouveau des nombres entiers positifs arbitraires.

Chariots de mine

Enfant, Benjamin adorait jouer dans son bac à sable. Récemment, il est devenu chef d'une mine de minerai en Azerbaïdjan. Une partie de son nouveau travail consiste à programmer les chariots de la mine le long d'une piste pour faire sortir les minerais de la mine. Complètement submergé par ce nouveau poste, il te demande de l'aide.

Il y a deux stations, une près de la zone minière, la station A, et une en dehors, la station B. Les deux stations sont reliées par une seule voie. La piste ne peut être utilisée que dans un seul sens à la fois, sinon les chariots pourraient entrer en collision. Un trajet sur la voie prend d secondes (quelle que soit la direction $A \to B$ or $B \to A$). Benjamin peut envoyer un nouveau chariot de mine le long de la voie depuis l'une ou l'autre des stations à chaque seconde.



Initialement, il y a n chariots de mine à la station A et 0 à la station B. L'équipe de minage mine des nouvelles unités de minerai aux temps $t_0, t_1, \ldots, t_{n-1}$, données en ordre croissant. Une fois minée, une unité est prête pour être envoyée de la station A à la station B sur un chariot. Un chariot peut porter au plus une unité par voyage. La journée de travail de Benjamin est terminée une fois que tous les minerais ont été envoyés à la station B et que tous les chariots sont de retour à la station A.

Le problème de Benjamin est de trouver le temps minimal T tel qu'il y a une manière de programmer le transport de tous les minerais vers la station B et de ramener les chariots à la station A.

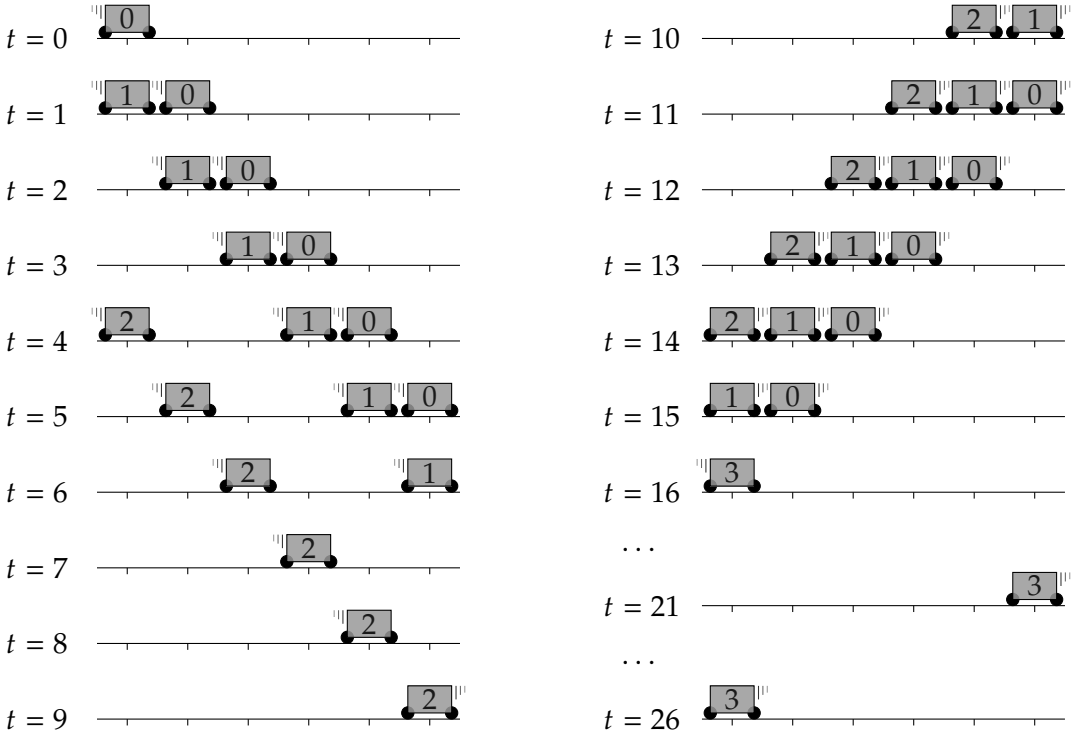
Description formelle Pour des entiers $t_0, t_1, \ldots, t_{n-1}$ donnés avec $t_0 < t_1 < \cdots < t_{n-1}$ et un entier $d > 0$, trouve le T minimal tel qu'il existe des nombres entiers $a_0, a_1, \ldots, a_{n-1}$ (les temps auxquels tu envoies le i-ème chariot de A vers B) et $b_0, b_1, \ldots, b_{n-1}$ (les temps auxquels tu envoies le i-ème chariot de B vers A) de manière à satisfaire les conditions suivantes :

- $b_i + d \leq T$ pour tout i (Benjamin a terminé au temps T),
- $a_i \geq t_i$ (on ne peut envoyer une unité de minerai qu'après qu'elle a été minée),
- $b_i \geq a_i$ (on ne peut renvoyer le chariot vers A seulement une fois qu'il est arrivé à B),
- $a_i \neq a_j$ et $b_i \neq b_j$ pour $i \neq j$ (on ne peut pas lancer deux chariots en même temps) et
- $|a_i - b_j| \geq d$ pour tout i et j (on ne peut pas utiliser la voie dans les deux directions en même temps).

Analyse Un algorithme pour ce problème reçoit en entrée les nombres entiers n et d, ainsi que la liste des temps de départ $t_0, \ldots, t_{n-1}$, avec $t_0 < \cdots < t_{n-1}$. Tu dois donner le temps d'exécution et l'utilisation de mémoire en fonction de n et d. Ces paramètres sont indépendants, donc $\Theta(n + d)$ est pire que $\Theta(n)$ et $\Theta(d)$. Similairement, $\Theta(\min(n, d))$ est

mieux que $\Theta(n)$ et $\Theta(d)$. Les valeurs $t_0, \ldots, t_{n-1}$ ne comptent pas comme utilisation de mémoire. Tu peux les lire sans utilisation additionnelle de mémoire, mais tu ne peux pas les modifier sur place.

Exemple Si $n = 4, d = 5$ et $t_0 = 0, t_1 = 1, t_2 = 4, t_3 = 16$, la réponse est 26 :



Les temps de départ des chariots sont $a = [0, 1, 4, 21]$ et $b = [11, 10, 9, 26]$.

Sous-tâche 1: Battre l'algorithme de Stofl (15 points)

Stofl a conçu l'algorithme présenté sur la page suivante pour résoudre le problème. Dans cette sous-tâche tu dois :

— Indiquer le temps d'exécution et l'utilisation de mémoire de l'algorithme de Stofl. Tu n'as *pas* besoin de justifier ta réponse (5 points).

— Trouve un contre-exemple pour lequel l'algorithme de Stofl ne donne pas la solution optimale. Dis quelle est la sortie de l'algorithme de Stofl (pas de justification demandée) et quelle serait la solution optimale (10 points).

Sous-tâche 2: Algorithme optimal (85 points)

Maintenant, à toi de jouer ! Trouve un algorithme qui résolve le problème optimalement.

Algorithme: Heuristique de Stofl

Entrée : n, d et le tableau $t[0], \ldots, t[n-1]$
Sortie : Une borne supérieure pour T, mais pas la minimale

```
1 w = 0                                  // le nombre de chariots attendants dans la station B
2 for i = 0 to n − 2 do
3 |   x = t[i + 1] − t[i] − 2 · d // temps supplémentaire pour un aller-retour
4 |   if x ≥ 0 then    // si nous avons assez de temps pour un aller-retour complet
5 |   |   w = max(0, w − x)
  |   |    // nous en faisons un et renvoyons les chariots
6 |   else             // sinon nous envoyons juste le minerai vers B
7 |   |   w = w + 1    // et ajoutons un chariot à ceux bloqués en B
8 return t[n − 1] + 2 · d + w
  // aller-retour pour le dernier chariot et retour des autres chariots
```

Spectacle de lumières

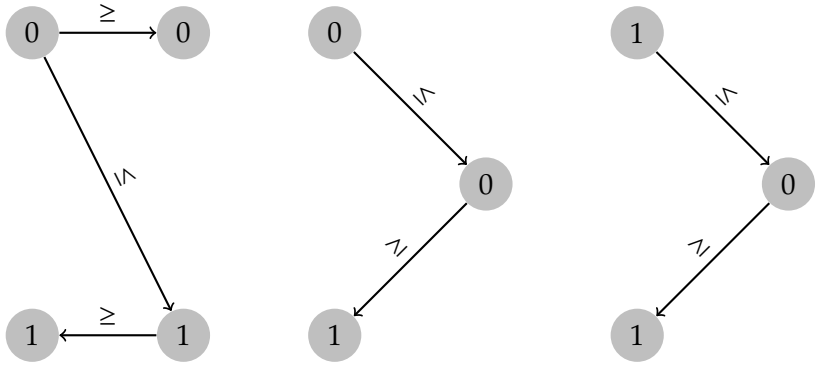
La souris Stofl aimerait mettre en place un grand spectacle de lumières sur la façade de sa maison. Il a acheté des guirlandes d'une forme étrange : elles ne forment pas juste un long fil mais une sorte d'arbre d'ampoules et de câbles.

Son idée est de montrer aux visiteurs toutes les possibilités de combinaisons d'ampoules allumées et éteintes pendant le spectacle, et de faire la transition entre ces différentes combinaisons en allumant ou éteignant une seule lampe à la fois.

Malheureusement, Stofl a acheté les guirlandes les moins chères dans un magasin douteux et elles ont toutes un gros problème de câblage : pour n'importe quelle paire d'ampoules directement connectées par un câble, il y en a toujours une des deux qui ne peut fonctionner que si l'autre est aussi allumée. Cette contrainte rend beaucoup de combinaisons impossibles. Cela déçoit beaucoup Stofl, mais il ne veut pas abandonner son idée et décide de simplement passer par toutes les combinaisons restantes. Cependant, cela rend plus ardue la tâche de trouver une suite convenable de combinaisons et c'est la raison pour laquelle Stofl t'a demandé de concevoir un algorithme pour l'aider.

Description formelle Soit un graphe G non orienté avec n sommets et m arêtes. Il est garanti que ce graphe est totalement acyclique. Le graphe n'est donc qu'une collection d'arbres. On attribue aux sommets de ce graphe une valeur soit de 0, soit de 1 (si l'ampoule correspondante est allumée ou non). Les arêtes sont étiquetées (directionnellement) avec soit '$\leq$', soit '$\geq$' : si une arête allant de a vers b est étiquetée avec '$\leq$', alors il est impossible de donner une valeur de 1 à a et de 0 à b (de même, si une arête est étiquetée avec '$\geq$', il est impossible d'attribuer une valeur de 0 à a et de 1 à b).

Toute attribution aux sommets de G à $\{0, 1\}$ est une combinaison valide si et seulement si toutes les conditions dépendant des arêtes sont satisfaites. Par exemple, les deux graphes de gauche ci-dessous sont dans une combinaison valide, mais pas celui tout à droite. Ton travail est de concevoir un algorithme pour déterminer une série de combinaisons telle que deux combinaisons successives ne diffèrent seulement par la valeur donnée pour exactement un sommet du graphe.



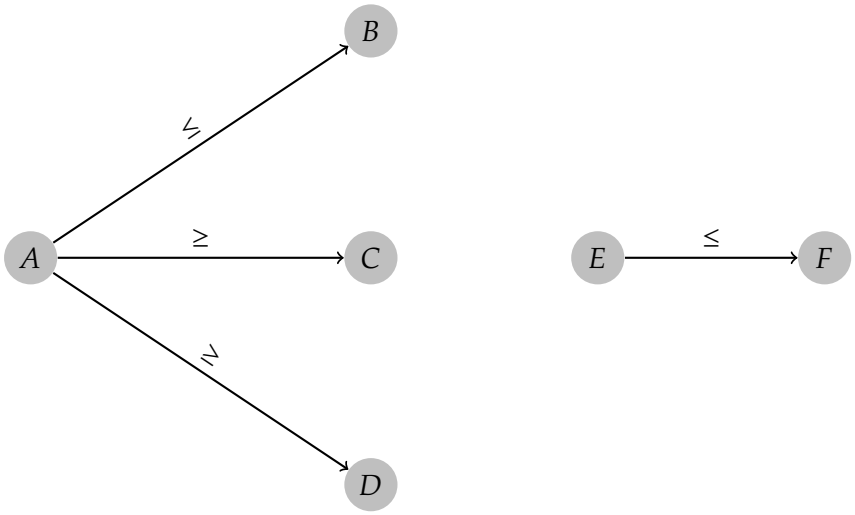
Pour ce faire, tu dois implémenter les deux fonctions suivantes :

1. *start* : prend comme argument le graphe, initialise toute structure de données dont tu auras besoin et retourne une attribution initiale de valeurs aux sommets du graphe.
2. *next* : retourne l'index du prochain sommet dont la valeur devrait être changé ou -1 si tu as atteint la fin de la suite d'instructions.

Si e est une arête, tu peux utiliser $e.a$ et $e.b$ pour obtenir l'index des sommets reliés par e. $e.d$ est '$\leq$' si a doit être '$\leq$' b et '$\geq$' si a doit être '$\geq$' b.

Sous-tâche 1: Résoudre un exemple (8 points)

Résous le problème pour le graphe ci-dessous en donnant une liste avec l'état initial de chaque sommet ainsi qu'une suite de sommets dont il faut changer la valeur afin de passer par chaque combinaison valide exactement une fois.



Sous-tâche 2: Combiner des guirlandes (24 points)

Pour commencer, Stofl aimerait que tu l'aides à produire un spectacle avec quelques guirlandes pour lesquelles il a déjà trouvé une solution : il ne reste qu'à les combiner ! Dans cette sous-tâche, tu peux supposer qu'on te fournit des fonctions *start* et *next* pour tout arbre individuel dans le graphe. Le graphe t'est fourni comme une liste de ces arbres L. Tu peux utiliser des instructions comme *L[i].start()* pour utiliser les fonctions *start* and *next* du i-ème arbre. Tu dois combiner les guirlandes ; en d'autres termes, tu dois implémenter les fonctions *start* et *next* pour le graphe entier. Tu peux supposer que les fonctions pour les arbres ont un temps d'exécution de $O(x)$ pour *start* dans un arbre avec x sommets et $O(1)$ pour *next*.

Sous-tâche 3: Guirlandes individuelles (68 points)

Dans cette sous-tâche, tu travailles sur des guirlandes individuelles supplémentaires pour améliorer le spectacle. C'est tout ce dont tu as besoin, puisque tu sais déjà comment les combiner grâce à ton algorithme de la sous-tâche précédente ! Tu peux donc supposer que G est connecté, c'est-à-dire que tu travailles sur un simple arbre. Tu dois créer des fonctions *start* et *next* pour cet arbre. En échange d'une déduction de points, tu as le droit de supposer les propriétés suivantes pour cet arbre :

Sorte d'arbre	Points maximaux
Chemin (degré maximal 2) avec des arêtes de i à $i+1$ avec l'étiquette '$\geq$'	8
Arbre binaire (degré maximal 3) dont on te donne la racine r. Les arêtes sont étiquetées de sorte à ce que le parent soit toujours '$\geq$' son enfant.	24
Arbre dont on te donne la racine r. Les arêtes sont étiquetées de sorte à ce que le parent soit toujours '$\geq$' son enfant.	34
Arbre étiqueté arbitrairement	68

Barème Dans les deux dernières sous-tâches, le plus important est que ton algorithme soit correct. Il ne doit pas forcément être optimal pour marquer quelques points. Voici une liste de temps d'exécution que tu peux viser et le pourcentage de points maximaux que tu obtiendras pour chacun d'entre eux. Dans ce tableau, P est le nombre de combinaison valides et $\text{poly}(n)$ est un polynôme en n.

Temps d'exécution	Pourcentage de points
start en $O(P \cdot \text{poly}(n))$, *next* en temps constant	25%
start en $O(\text{poly}(n))$, *next* en $O(n)$	50%
start en $O(n)$, *next* en temps constant en moyenne	75%
start en $O(n)$, *next* en temps constant dans le pire des cas	100%