

Algorithmes gloutons (2)

Exercice 1 : Le voyageur pressé

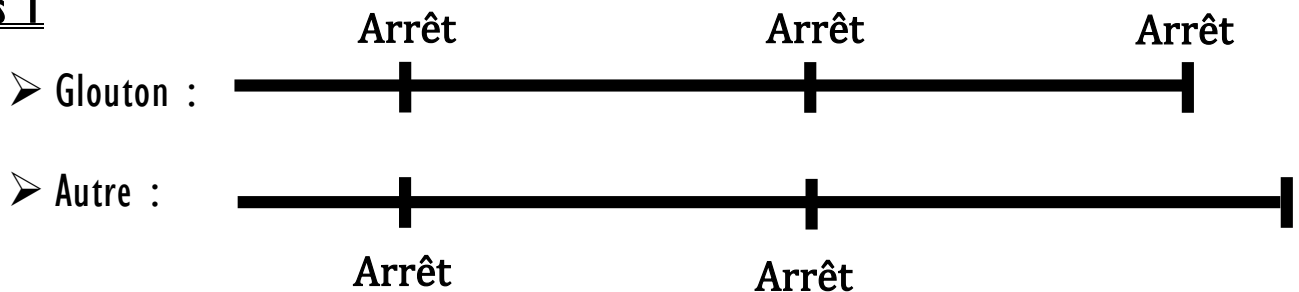
Un voyageur sur l'autoroute fait le plein d'essence seulement quand son réservoir ne contient plus assez d'essence pour atteindre la station-service suivante. Cette stratégie minimise-t-elle le nombre d'arrêtes de notre voyageur ?

Stratégie optimale.

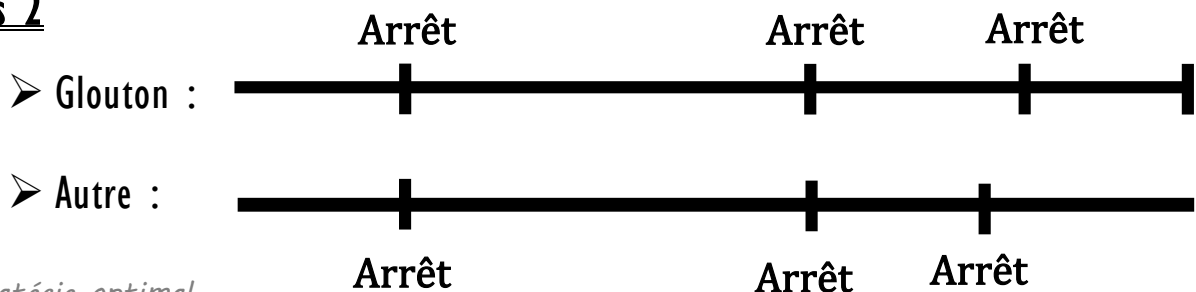
Supposons qu'en plus de notre stratégie il y a une autre solution qui est meilleur (avec moins d'arrêts), donc je prends cet autre stratégie et je la compare avec la stratégie gloutonne.

Il y a un endroit où les 2 diffèrent car la stratégie gloutonne dure autant qu'elle peut (donc dure plus que l'autre).

Cas 1



Cas 2



La stratégie optimale s'arrête avant le glouton



Exercice 2 : Le Flexit

Les ministres de diverses nations européennes doivent se voir le 31 décembre 2024 pour préparer le Flexit qui aura lieu à 23h59.

Or dans la grande effervescence (nervosité) engendrée par l'évènement, ils ne restent que quelques heures à Bruxelles, entre 2 avions (ou TGV).

On ne peut pas faire une réunion avec tout le monde. Mais si beaucoup de réunions ont lieu, les conclusions risquent d'être divergentes et cela va être dur de faire la synthèse avant minuit...

Il faut donc essayer de faire le moins de réunions possible (sachant qu'une réunion dure au moins une heure) et que tout ministre participe à au moins une réunion, pour exprimer la position de son pays.

Donc : vous avez n ministres ; le ministre M_i atterrit à l'heure a_i et repart à l'heure d_i ($d_i - a_i > 1h$ sinon le problème est impossible).

Une réunion à l'heure r_j peut réunir tous les ministres présents à ce moment-là pour une durée d'au moins une heure, donc tous ceux tels que

$$a_i \leq r_j < r_j + 1h \leq d_i$$

Il faut proposer des heures $r_i \dots r_k$ de début des différentes réunion sachant que l'on veut k minimal.

(I) Résoudre le problème sur l'exemple suivant (donner le nombre de réunions minimal, et l'heure des réunions).

Ministre	arrivée	départ	Ministre	arrivée	départ
BE	6h30	9h45	GR	9h01	16h30
DE	15h27	21h03	IT	15h42	19h17
DK	7h12	13h08	LU	13h33	18h22
ES	7h56	13h08	PL	13h33	20h07
FR	10h01	14h21	PT	19h07	23h02

6h	7h	8h	9h	10h	11h	12h	13h	14h	15h	16h	17h	18h	19h	20h	21h	22h	23h
	BE 6h30 — 9h45																
			GR 9h01 — 16h30														
										DE 15h27 — 21h03							
				FR 10h01 — 14h21													
							LU 13h33 — 18h22										
							PL 13h33 — 20h07										
		ES 7h56 — 13h08															
		DK 7h12 — 13h08															
														PT 19h07 — 23h02			
										IT 15h42 — 19h17							

6h	7h	8h	9h	10h	11h	12h	13h	14h	15h	16h	17h
	BE 6h30 — 9h45										
			GR 9h01 — 16h30								
				FR 10h01 — 14h21							
		ES 7h56 — 13h08									
		DK 7h12 — 13h08									

Première réunion : BE, DK, ES : 8h45

Réunion 2 : FR, GR : 13h21

13h	14h	15h	16h	17h	18h	19h	20h	21h	22h	23h
				DE 15h27 — 21h03						
	LU 13h33 — 18h22									
	PL 13h33 — 20h07									
								PT 19h07 — 23h02		
				IT 15h42 — 19h17						

Réunion 3 : LU, IT ,DE,

PL : 17h22

Réunion 4 : PT : 22h02

(2) Proposez un algorithme glouton qui résout le problème.

Le choix glouton doit être fait après un certain *pré-traitement* des données, lequel ?

Algorithme glouton

- Trier les ministres par ordre de départ.
- On prend le ministre qui peut en premier.
- On fait une réunion 1h avant son départ, tous les ministres présent y assiste.
- On fait la même chose avec les ministres qui restes.

(3) Montrez que votre algorithme minimise bien le nombre de réunions.

Pourquoi l'algorithme est optimal ?

Prenons une solution qui est meilleur et on regarde l'heure de la 1^{er} réunion de cette solution.

La 1^{er} réunion de l'autre solution peut commencer plus tard ? Non, car alors on rate un ministre.

Un autre cas figure est que ca commence a la même heure que le glouton, mais y'a certains ministres qui sont pas présent (par exemple dans la question (1) : une 3eme réunion avec LU, IT, DE a 17h22 et une réunion a 19h07 avec PT).

On a vu que la réunion peut pas commencer après l'autre solution. Si jamais la réunion commence plus tôt dans l'autre solution : (à compléter)..... et on se rapproche comme ca de la solution glouton.

Si il y a une réunion plus tôt dans l'autre solution : on peut toujours la retardé à l'heure du glouton.

Exercice 3

On considère le processus suivant :

A chaque instant on a un entier x initialement égal à 1.

A chaque étape on peut l'incrémenter avec 1 ou le doubler.

Le but est de produire une valeur cible n .

Par exemple,

A partir de 1 on peut obtenir 10 en quatre étapes :

$$\begin{array}{ccccccc} & +1 & * 2 & +1 & * 2 & & \\ 1 & \longrightarrow & 2 & \longrightarrow & 4 & \longrightarrow & 5 \longrightarrow 10 \end{array}$$

Evidemment, on peut toujours obtenir n uniquement en incrémentant par 1, mais ce n'est pas très efficace.

(I) Donnez un algorithme glouton pour obtenir le nombre *minimal* d'opérations permettant d'obtenir n à partir de 1.

J'ai un entier positif et je veux obtenir cet entier avec 2 opérations. A partir de 1, je veux le nombre minimal d'opérations pour faire ça.

Exemple d'un cas PAS optimal (on peut faire ça avec 4 opérations).

$$\begin{array}{ccccccc} & +1 & * 2 & * 2 & +1 & +1 & \\ 1 & \longrightarrow & 2 & \longrightarrow & 4 & \longrightarrow & 8 \longrightarrow 9 \longrightarrow 10 \end{array}$$

Le méthode de multiplier par 2 tant que possible et après ajouter 1 ne marche pas.
Commencer par 1 et aller à la cible, ça marche pas.

Algorithme glouton

On calcule les opérations de la fin vers le début, pas du début vers la fin.

(C'est ça l'astuce dans cet exercice).

- On commence avec la cible n .
- Si n est pair : on divise par 2
 - Nouvelle cible : $\frac{n}{2}$
- Si n est impair : on décrémente par 1
 - Nouvelle cible : $n - 1$

Si la cible est impair, la dernière opération doit être **+**

(2) Prouvez qu'il est correcte.

Pourquoi c'est optimal ? • Preuve d'optimalité par contradiction

Supposons qu'on a une suite d'opérations optimale, différente de l'algorithme glouton, pour une cible n . « Différent de l'algo glouton », donc : a un moment donné n est pas pair et donc l'algorithme glouton fait « $\times 2$ » comme dernière opération, et l'autre suite fait $+1$ et forcément $+1$ avant, l'autre fait $\times 2$ avant. Donc on a un n .

$\times 2$

Glouton : $\rightarrow n$

$+1$

Autre : $\rightarrow n$

Donc, on a 2 cas : soit pas du tout de « *2 » avant (mais si y'a que des « +1 » ca sera pas optimal, sauf dans le cas ou $n = 2$).

$$\begin{array}{ccc} & * 2 & +1 \\ \text{Autre : } \rightarrow & \underbrace{\dots \dots \dots \dots} & \rightarrow n \\ & \text{il y a forcément "+1"} & \\ & \text{car le nb est impair} & \end{array}$$

Car si n est pair, « $n-1$ » forcément impair.

$$\begin{array}{ccc} & \text{nb de "+1":} & \\ & n-2k & \\ * 2 & \underbrace{\hspace{10em}}_{+1 +1} & \\ \text{Autre : } k \rightarrow & \underbrace{\dots \dots \dots \dots} & \rightarrow \rightarrow n \\ & \text{il y a que} & \\ & \text{des « +1 »} & \end{array} \quad \begin{array}{l} (n-2k+1) \text{ opérations} \end{array}$$

C'est optimal ?

Je peux faire mieux (faire plus court pour atteindre n a partir de k) ?

On obtient une autre séquence plus courte.

Je fais d'abord « +1 » au lieu de « *2 » et ensuite je fais « *2 » et ensuite je complète avec des « +1 » jusqu'à n .

Combien de « +1 » j'ai maintenant pour atteindre n ?

Je fais d'abord « +1 » au lieu de « *2 » et ensuite je fais « *2 » et ensuite je complète avec des « +1 » jusqu'à n.

$$\begin{array}{ccccccc} & +1 & & *2 & & +1 & +1 \\ k & \longrightarrow & (k+1) & \longrightarrow & (2k+2) & \longrightarrow & \dots \dots \longrightarrow n \end{array}$$

Nombre d'opérations : $n - 2k - 2 + 2 = n - 2k$ *operations*

Donc, une opération de moins que tout à l'heure !

Donc, **contradiction** car je prétendais que ce truc là est optimal.

J'ai supposé que c'est optimal → j'ai montrer que je peux faire plus petit → donc pas optimal → contradiction.

L'autre solution est censé être optimal, mais différente du glouton.

n est pair car c'est la seul façon d'avoir une opération différente que l'optimale.

Si j'ai 2 « +1 » qui se suivent, à la fin ça peut PAS être optimal, je peux économiser une opération.

Par exemple, $n = 12$

$$\begin{array}{ccccccc} & & & *2 & & +1 & +1 \\ 5 & \longrightarrow & 10 & \longrightarrow & 11 & \longrightarrow & 12 \end{array}$$

$$\begin{array}{ccccccc} & & & +1 & & *2 & \\ 5 & \longrightarrow & 6 & \longrightarrow & 12 & & \end{array}$$

Mais c'est mieux :

Une variante :

Remplacer le « +1 » par « -1 ». L'algorithme glouton sera pareil. La preuve : peut-être un peu différente..

Exercice 4

Supposons qu'on a n skieurs avec leur tailles données par t_1, \dots, t_n
et n paires de skis avec leur tailles données par s_1, \dots, s_n .

Donner un algorithme qui assigne une paire de skis à chaque skieur de sorte que la différence moyenne entre la taille du skieur et la taille du ski soit minimal.

Formellement, l'algorithme devrait donner une permutation (échange, intervention) σ (sigma)

De sorte que $\frac{1}{n} \sum_{i=1}^n |t_i - s_{\sigma(i)}|$ soit minimal.

Par exemple, pour les tailles de skieurs 1.7, 1.9, 1.8
et les skis 2.1, 1.8, 1.6

la solution est la permutation σ avec
 $\sigma(1) = 3$, $\sigma(2) = 1$ et $\sigma(3) = 2$.

Prouver que votre algorithme est correcte.

Analyse de l'exemple

On a n skieurs avec leur tailles données par t_1, \dots, t_n

$n = 3$: avec les tailles données par $t_1, t_2, t_3 = 1.7, 1.9, 1.8$

On a n paires de skis avec leur tailles données par s_1, \dots, s_n .

$n = 3$: avec les tailles données par $s_1, s_2, s_3 = 2.1, 1.8, 1.6$

Donner un algorithme qui assigne une paire de skis à chaque skieur de sorte que la différence moyenne entre la taille du skieur et la taille du ski soit minimal.

la solution est la permutation σ

Etant donné un skieur t_i on cherche une taille de paires de skis $s_{\sigma(i)}$

$$\sigma(1) = 3, \text{ car}$$

$$\min(|t_1 - s_j|) \Rightarrow |t_1 - s_3| \Rightarrow |1.7 - 1.6| = 0.1$$

$$\sigma(2) = 1 \text{ car}$$

$$\min(|t_2 - s_j|) \Rightarrow |t_2 - s_1| \Rightarrow |1.9 - 2.1| = 0.2$$

$$\sigma(3) = 2 \text{ car } s_2 = t_3$$

Exemple supplémentaire

■ Skieurs :

t_1	t_2	t_3	t_4
1.80	1.75	1.60	1.70

■ Skis :

s_1	s_2	s_3	s_4
1.73	1.52	1.90	1.80

Le but : Minimiser la moyenne des différences de tailles.

$$\underbrace{M_\sigma}_{\text{Mesure}} = \frac{1}{n} \sum_{i=1}^n |t_i - s_{\sigma(i)}|$$

Sigma (σ) va être une fonction, une permutation.

$$\sigma : \{1, 2, 3, 4\} \longrightarrow \{1, 2, 3, 4\}$$

Par exemple, si on le fait sans réfléchir :

$$\sigma(1) = 1, \quad \sigma(2) = 2, \quad \sigma(3) = 3, \quad \sigma(4) = 4$$

$$M = \frac{1}{4} (0.07 + 0.23 + 0.3 + 0.1) = \frac{1}{4} (0.7) = 0.15$$

Le but : Trouver le meilleur σ possible.

Algo (très) naïf

Essayer tous les permutations et prendre la meilleur.

Complexité : $n!$ (trop cher !)

Algo glouton

L'idée est : on trie les skieurs et les skis par taille, mais formulé ainsi c'est pas un algo glouton.

On prend le skieur le plus petit et on lui assigne la paire de ski la plus courte, et ainsi de suite.

Complexité : n^2 si on le fait naïvement, mais on peut faire $n \cdot \log n$ si je trie (je peut trié avec leurs indices).

Preuve que l'algorithme est correct

(C'est pas complètement trivial)

L'idée est de dire : admettant que c'est pas correct, et qu'il y a un sigma (σ) qui donne un meilleur M que la solution donnée par le glouton.

Donc, **preuve par contradiction.**

Premier chose, pour faciliter les choses, on peut supposer que les t_i et S_i sont déjà triée. Dans ce cas l'algorithme glouton donne juste l'identité pour la permutation.

Maintenant, on va supposer que il y a **une permutation σ avec M_σ optimale**. Cela veut dire qu'il y a un Sigma (σ) qui n'est pas l'identité (une permutation qui est pas l'identité).

Maintenant, comme d'habitude,

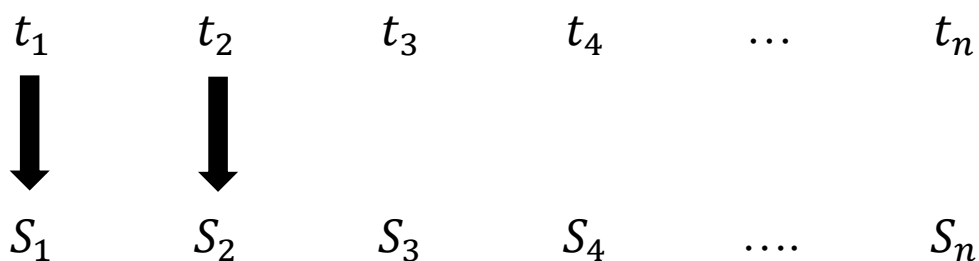
On va regarder le premier endroit ou ca diffère.

Il doit exister un k entre 1 et $n - 1$

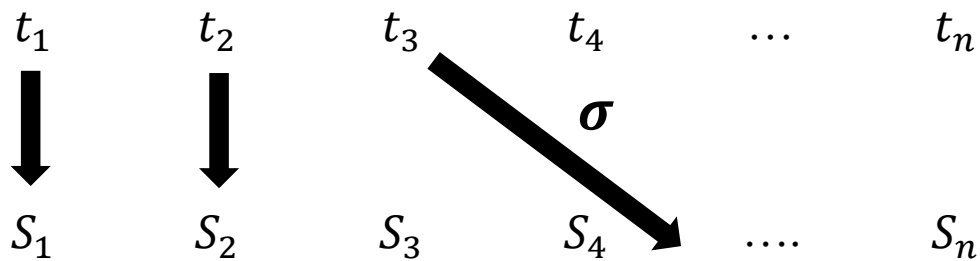
(Et pas j'jusqu'à n car si les $(n - 1)$ sont pareil, le n ième est forcément pareil aussi).

Alors il doit exister un $\sigma(k)$ t.q. $\sigma(k) = k' > k$.

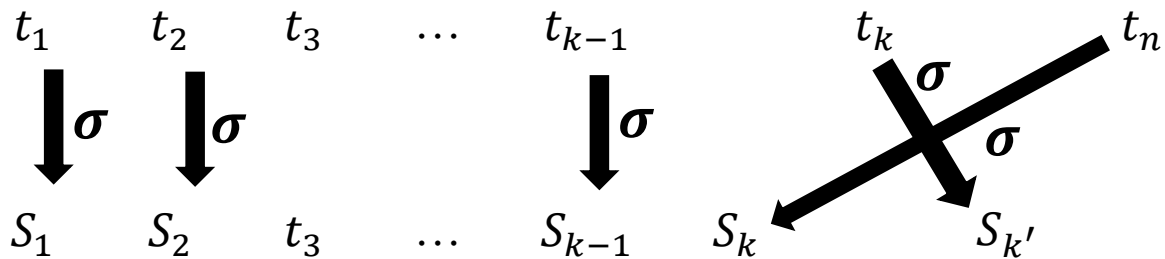
L'algo glouton donne :



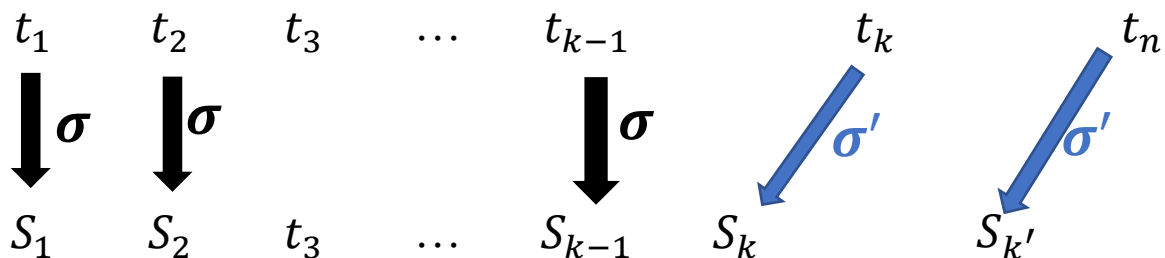
Dans l'autre solution il doit exister un σ qui va plus loin :



Pour l'instant, le Sigma (σ) fait ca :



- On veut que σ' assigne t_k à S_k .
- On définit une nouvelle permutation σ' mais le problème c'est que dans la solution S_k est déjà assigner a quelqu'un d'autre.
- Donc, on va relier t_n à $S_{k'}$ avec σ' et t_k à S_k avec σ'



Il existe un $n > k$ t.q. $\sigma(n) = k$. On définit σ' comme σ sauf :

- $\sigma'(k) = k$
- $\sigma'(n) = k'$

On va calculer $M_\sigma - M_{\sigma'}$

Si c'est supérieur à 0 on a « gagner » car ça veut dire que $M_{\sigma'}$ est meilleur que M_σ . Si c'est égale à 0 c'est la même valeur et c'est bien pour nous car on se rapproche du glouton.

La formule de M est différente pour σ et σ' ? C'est quoi les endroits où ça change ? C'est k et n , pour σ' . Alors on aura :

$$M_\sigma - M_{\sigma'} = \frac{1}{n} \left(|t_k - s_{k'}| + |t_n - s_k| - |t_k - s_k| - |t_n - s_{k'}| \right)$$

Faut montrer que c'est supérieur ou égal à 0.

On sait que :

On sait que les S et les K sont triés. Donc : $S_k \leq S_{k'}$, $t_k \leq t_n$.

Il y a 6 cas :

1. $S_k \leq S_{k'} \leq t_k \leq t_n$ Tous les ski sont inférieurs à la taille des personnes.
2. $S_k \leq t_k \leq S_{k'} \leq t_n$
3. $S_k \leq t_k \leq t_n \leq S_{k'}$
4. $t_k \leq S_k \leq t_n \leq S_{k'}$
5. $t_k \leq S_k \leq S_{k'} \leq t_n$
6. $t_k \leq t_n \leq S_k \leq S_{k'}$

Maintenant faut faire le calcul pour chaque cas. On a que à insérer maintenant.

On va détailler 2 cas :

$$1. M_{\sigma} - M_{\sigma'} = \frac{1}{n} \begin{pmatrix} t_k - s_{k'} + t_n - s_k \\ -t_k + s_k - t_n + s_{k'} \end{pmatrix} = 0$$

C.-à-d. : la différence entre la permutation σ et σ' (qui est optimal) es 0.

Maintenant, on peut continuer inductivement avec σ' (σ est pas meilleur que σ') mais c'est pas une contradiction...

$$2. M_{\sigma} - M_{\sigma'} = \frac{1}{n} \begin{pmatrix} s_{k'} - t_k + t_n - s_k \\ -t_k + s_k - t_n + s_{k'} \end{pmatrix} = \frac{1}{n} (2s_{k'} - 2t_k) \geq 0$$

≥ 0 car dans le cas 2 : $t_k \leq s_{k'}$

Conclusion

Dans chaque cas :

- Soit $M_{\sigma} > M_{\sigma'}$: contradiction
- Soit $M_{\sigma} = M_{\sigma'}$

C'est que σ' est plus proche du glouton que σ . « Plus proche » dans le sens que σ' a moins de différences avec le glouton que σ . On continue inductivement.

On a définie

$$M_{\sigma} = \frac{1}{n} \sum_{i=1}^n |t_i - s_{\sigma(i)}|$$

J'aurai pu écrire :

$$M_{\sigma} = \frac{1}{n} \sum_{i=1}^n (t_i - s_{\sigma(i)})^2$$

Alors ça nous évite décrire tous les cas de valeurs absolus...

Encore une façon d'écrire :

$$M_{\sigma} = \max |t_i - s_{\sigma(i)}|$$

Mais la preuve marche que pour le cas initial.

Exercice 5

Un mot de parenthèses **w** est **équilibré**
si et seulement si une des conditions suivantes est remplie :

- **w** est le mot vide
- **w** = **(v)**, ou **v** est un mot équilibré
- **w** = **uv**, ou **u**, **v** sont des mots équilibrés.

Par exemple, le mot **w** = **((()())(())())** est équilibré.

Exemple d'un mot bien parenthéser :

(()) () () (())

(1) Proposez un algorithme pour déterminer si un mot de parenthèse est équilibré.

J'ai besoin d'un compteur qui commence à 0.

Si il y avait plusieurs types de parenthèses, on aurai utiliser une pile.

Algorithme pour vérifier qu'un mot est bien parenthéser

- On utilise un compteur c
- On parcours le mot de gauche à droite
 - On incrémente c si il y a un « (»
 - On décrémente c si il y a un «) »
 - On vérifie qu'on se trouve jamais sous 0.
- A la fin, le compteur c doit être 0.

(2) Proposez un algorithme glouton en temps linéaire pour déterminer un des plus long sous-mots équilibrés d'un mot de parenthèses.

Prouvez-le et donnez la complexité.

Par exemple, pour le mot $) ((((((($

qui peut également être vu comme $) ((((((($

ça peut être

$((())$ ou $(((((((($.

Indication : Adaptez l'algorithme pour 1.

$) ((((((($

Ce mot est pas bien équilibré. Maintenant, je cherche un sous-mot équilibré. Autrement-dit : je peut effacer des parentaises ou je veut et ce que je cherche c'est le sous-mot le plus long que je peux obtenir en effaçant des caractères.

Par exemple, ici, je peut effacer tout sauf $((())$ qui me donne un sous-mot équilibré de taille 2. Je peux également obtenir des plus grands. Par exemple,

$) ((((((($

$) ((((((($

$) ((((((($

$) ((((((($

Les solutions optimales
(les plus long)

{



L'algo qu'on va écrire va trouver ça

On cherche donc :

- Un algo qui nous garantit de trouver un des sous-mot les plus long.
- Un algo glouton qui s'inspire de la question (1).

Algorithme glouton

- On utilise un compteur c qui marche comme dans l'algo 1.
- **Premier parcours** : Parcourir le mot de gauche à droite.
 - On efface tous les parenthèses fermantes si elle font descendre le compteur sous 0.
- **Second parcours** : Parcourir le mot de droite à gauche.
 - Cet fois on incrémente le compteur c pour «) » et on décrémente pour « (»
 - On efface tous les parenthèses ouvrantes si elle font descendre le compteur sous 0.

Dans le cas de notre exemple :

- Après le 1^{er} parcours : ~~)~~ (() () (() (
- Après le 2^{em} parcours : ~~)~~ ~~(~~ () () ~~(~~ () ~~(~~


Complexité : en temp linéaire car c'est juste parcourir le mot 2 fois.

Donc : $O(n)$

Prouver que effectivement on trouve une des meilleures sous-séquences

Il faut se poser la question : ou est-ce qu'on efface des caractères ?

- 1^{er} parcours : on efface des parenthèses fermentes
- 2^{em} parcours : on efface des parenthèses ouvrantes

Donc, une tel situation risque d'arriver ? 

Lemme

La situation suivante ne peut pas arriver : on efface un « (» et un «) » qui se trouve après dans le mot.

C.-à-d. : la situation suivante ne peut pas arriver : 

Indications pour la preuve (preuve pas compète)

Comment procéder ? Comme d'habitude :

- On a une solution autre que le glouton.
- La preuve seurai de dire : « J'ai une autre solution » (dans cet exemple, par exemple : $) (() () (() ()$ et il me faut un argument que cet autre solution est pas meilleur que le résultat du glouton.

Preuve par contradiction

Supposent qu'on a un sous-mot équilibré plus long que celui obtenu par l'algorithme glouton.

Glouton			
Même choix		Choix différent	
Autre			

Maintenant, faut regarder tous les cas possibles :

Cas 1

Glouton		(
Autre		€

Comment l'autre solution peut se rapprocher du glouton ? Comment ne pas effacer la parenthèse dans l'autre solution ? Si la solution est meilleur il doit y avoir une parenthèse ouvrante par la suite. On peut avancer une parenthèse ouvrante dans l'autre solution.

Cas 2

Glouton)
Autre		ƒ

Pareil.

Ca ce sont les 2 cas simples.

Cas 3

Le glouton a effacé une parenthèse fermante et l'autre ne la pas effacé.

Glouton)
---------	--	---

Autre)
-------	--	---

----- équilibré -----

Impossible.

Cas 4

Le glouton a effacé une parenthèse ouvrante et l'autre ne la pas effacé.

par exemple :) (() () (() (

Glouton		€
---------	--	---

Autre		(
-------	--	---

Dans le glouton, pour la suite du mot on a pas effacé des parenthèses fermentes.
Donc, autre ne peut pas être meilleur que le glouton.