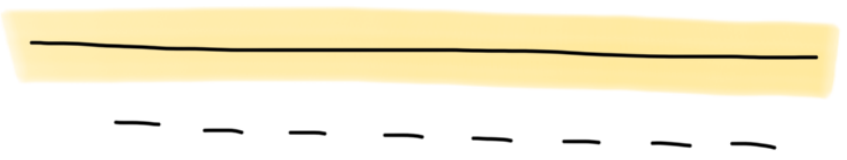
GLOUTON

-> Règle (1) :



Le glouton sélectionne une seule requête car elle commence plus tôt alors que la solution optimale contenait 8 requêtes.

-> Règle (2)

Le glouton sélectionne une seule requête car elle est plus courte alors que la solution optimale contenait 2 requêtes.

-> Règle (3)

Le glouton sélectionne 3 requêtes (celle du milieu en premier car elle n'a que 2 incompatibilités) alors que la solution optimale contenait 4 requêtes (les 4 en bas).

-> Règle (4)

Soit $G = (i_1, \dots, i_g)$ les requêtes choisies par le glouton avec la règle (4), et $O = (j_1, \dots, j_o)$ un ensemble de requêtes compatibles optimal, tous deux classés par ordre croissant des temps de fin, et avec $o \geq g$.

* Montrons par récurrence que $\forall r \in [\![1, g]\!], f(i_r) \leq f(j_r)$

Initialisation. Pour $r = 1$.

Le glouton sélectionne la requête se terminant en premier qui est compatible avec celles déjà choisies.

Comme $f(i_1) \leq f(i_k)$ pour $k \in [\![2, g]\!]$ (car G est classé par ordre croissant des temps de fin), i_1 est donc la première requête choisie par le glouton.

Ainsi i_1 termine avant toutes les requêtes de l'ensemble R_1, et en particulier avant j_1.

On a bien $f(i_1) \leq f(j_1)$

Hérédité

Supposons que $f(i_r) \leq f(j_r)$ pour un certain $r \geq 1$.

Montrons que $f(i_{r+1}) \leq f(j_{r+1})$.

Raisonnons par l'absurde : on suppose que $f(i_{r+1}) > f(j_{r+1})$

On sait que $d(j_{r+1}) \geq f(j_r)$ car O est un ensemble de requêtes compatibles.

Par H.R., on a $f(j_r) \geq f(i_r)$, donc

$d(j_{r+1}) \geq f(i_r)$: la requête j_{r+1} est donc compatible avec les r premières requêtes sélectionnées par le glouton.

Ainsi j_{r+1} est une requête compatible se terminant avant i_{r+1} : le glouton aurait sélectionné j_{r+1} et non i_{r+1} → contradiction.

Donc $f(i_{r+1}) \leq f(j_{r+1})$

* $\forall n \in [\![1, g]\!]$, $f(i_n) \leq f(j_n)$.

Raisonnons par l'absurde et supposons que $o > g$.

On sait que $f(i_g) \leq f(j_g)$.

Comme j_{g+1} (qui existe bien car $o > g$) est compatible avec j_g, elle est donc aussi compatible avec i_g.

Ainsi le glouton aurait sélectionné j_{g+1} également

→ contradiction : $o = g$.

* Le nombre de requêtes sélectionnées par le glouton (g) est égal au nombre de requêtes d'une solution optimale (o). Ainsi le glouton renvoie une solution optimale pour la règle (4).

→ Complexité

La complexité de l'algorithme glouton suivant la règle (4) est quasi-linéaire - dû au tri initial - des requêtes.

PROGRAMMATION DYNAMIQUE

Soit S_R le nombre maximal de requêtes compatibles de l'ensemble de requêtes R.

On note $comp(R, i)$ l'ensemble des requêtes de R compatibles avec la requête i (en particulier, $i \notin comp(R, i)$).

* On a la relation de récurrence suivante :

$$\begin{cases} S_{\emptyset} = 0 \quad \text{(aucune requête possible)} \\ S_{\mathcal{R}} = \max\left(S_{\mathcal{R}\setminus\{i\}},\ 1 + S_{comp(\mathcal{R},i)} \right) \end{cases}$$

- avec i une requête quelconque de $\mathcal{R}$

* la structure de données la plus adaptée pour la mémoïsation est le tableau associatif car il faut associer à des ensembles de requêtes $\mathcal{R}$, la valeur de $S_{\mathcal{R}}$.

* Le glouton n'explore qu'un seul sous-problème alors que la programmation dynamique les explore tous. La complexité temporelle (et même spatiale) est meilleure pour le glouton. Si on sait que le glouton fournit toujours la solution optimale (ce qui est le cas ici), il faut donc privilégier le glouton.