

## Objectifs

- À la fin de cette séance, vous devriez être capable de :
- reconnaître ou écrire un algorithme glouton ;
  - déterminer s'il est ou non optimal.

## Exercice 1 :

Un algorithme glouton pour colorier un graphe :

- prendre les sommets dans un ordre quelconque
- attribuer à chaque sommet la plus petite couleur non utilisée par ses voisins déjà coloriés

Combien de couleurs nécessite cet algorithme ?

Est-il optimal ?

## Correction de l'exercice 1

Le nombre de couleurs utilisées est le degré max d'un sommet + 1. Il suffit d'exhiber un graphe où un sommet de degré max est coloré en dernier et alors que tous ses voisins ont déjà une couleur distincte (c'est le cas pour une clique par exemple).

Exemple non optimal : une chaîne de 4 sommets, si on commence par les extrémités on les colore de la même façon et on est obligé d'utiliser 2 couleurs de plus pour les 2 sommets intermédiaires, alors que ce graphe est évidemment 2-colorable.

## Exercice 2 :

Un algorithme glouton un peu plus malin pour colorier un graphe :

- prendre les sommets par degré décroissant
- attribuer à chaque sommet la plus petite couleur non utilisée par ses voisins déjà coloriés

Que penser de cet algorithme par rapport au précédent ?

Est-il optimal ?

## Correction de l'exercice 2

Le principe est le même, mais on a l'espoir que le premier sommet  $x$  pour lequel on est amené à utiliser la couleur  $\deg(x) + 1$  soit un sommet de degré inférieur au degré max du graphe.

Évidemment pour les cliques ça ne change rien.

L'algo n'est toujours pas optimal : il suffit par exemple de prendre un cycle de longueur 6. L'ordre des sommets est alors indéterminé (ils sont tous de degré 2) et en commençant par 2 sommets diamétralement opposés on bute sur le même problème qu'à l'exercice précédent : 2 sommets intermédiaires doivent alors être colorés avec les couleurs 2 et 3 alors que le graphe est 2-colorable.

## Exercice 3 :

On se donne  $n$  réels  $\{x_1, x_2, \dots, x_n\}$ . On souhaite trouver le nombre minimal  $K$  tel que  $K$  intervalles  $I_1, I_2, \dots, I_K$  chacun de longueur 1 recouvrent tous les points, c'est-à-dire que tout point appartient à au moins l'un de ces intervalles. On considère ici des intervalles fermés (dont les bornes sont incluses).

1. Donner un exemple (avec  $n$  petit) montrant qu'il peut y avoir plusieurs recouvrements minimaux (plusieurs familles d'intervalles pour le même  $K$  minimal).  
Donner également un exemple où ce recouvrement minimal est unique.
2. Écrire un algorithme glouton qui détermine la valeur minimale de  $K$  en construisant les  $K$  intervalles.
3. Écrire la preuve de votre algorithme glouton.

### Correction de l'exercice 3

1.  $x_1 = 0, x_2 = 0.75, x_3 = 1.5$  peut être recouvert par exemple par  $[0; 1], [1; 2]$  mais aussi par  $[-0.5; 0.5], [0.75; 1.75]$ .  
Par contre  $x_1 = 0, x_2 = 1$  admet pour seul recouvrement minimal  $[0, 1]$ .
2. On trie les  $x_i$  par ordre croissant en temps  $\mathcal{O}(n \log n)$ , puis on construit les intervalles en temps  $\mathcal{O}(n)$  :  
 $i := 1$   
**while** *il reste des réels non recouverts* **do**  
   | Prendre l'intervalle  $[x_i; x_i + 1]$   
   | Incrémenter  $i$  tant que  $x_i$  est dans ce dernier intervalle
3. Notre algorithme produit une suite d'intervalles  $I$  (ordonnés de gauche à droite). Démontrons qu'à toute itération  $I$  est incluse dans une suite optimale d'intervalles  $I_{opt}$ .  
Initialement c'est évident ( $I = \emptyset$ ).  
Considérons la première itération à laquelle on ajoute à  $I$  l'intervalle  $[x_k; x_k + 1]$  ne faisant pas partie de  $I_{opt}$ . Par construction,  $x_k$  n'était pas déjà présent dans les intervalles de  $I$ . Donc il fait partie d'un des intervalles  $[x; y]$  de  $I_{opt} \setminus I$ , et en particulier  $x < x_k$ .  
Mais, toujours par construction, tout nombre situé entre  $x$  et  $x_k$  a déjà été pris dans un des intervalles de  $I$ , donc en remplaçant  $[x; y]$  par  $[x_k; x_{k+1}]$  dans  $I_{opt}$ , on ne « perd » aucun nombre. On a ainsi construit une autre solution optimale dans laquelle  $I \cup [x_k, x_k + 1]$  est incluse.

### Exercice 4 :

Ali Baba est enfin parvenu à entrer dans la caverne des 40 voleurs, mais il ne pourra pas emporter toutes leurs richesses : il n'est capable de porter que  $M$  kilogrammes sur son dos.

Devant lui se situent  $n$  sacs remplis d'épices rares et autres poudres d'or : chaque sac  $i$  pèse  $m_i$  kilogrammes et son contenu vaut  $v_i$  dirhams.

1. Quel mélange de poudres Ali doit-il emporter pour maximiser son profit ?
2. Démontrer que votre stratégie est optimale.
3. Un autre jour, Ali parvient également à entrer dans une caverne, mais cette fois le trésor est constitué d'objets indivisibles : une couronne en or, un rubis géant, un animal légendaire, etc.  
Votre stratégie est-elle encore valable ?

### Correction de l'exercice 4

1. Travailler par « prix au kg » décroissant : on prend un maximum de la poudre ayant la meilleure valeur  $v_i/m_i$  et on itère tant que le sac à dos n'est pas plein.
2. Il est facile de voir qu'un échange diminue la valeur totale du sac à dos (car le poids total est fixé et on diminuera le prix moyen au kg).
3. On construit un contre-exemple à 3 objets où les 2 objets les moins rentables forment l'optimal car ils maximisent le remplissage, par exemple :  $M = 5, m_1 = 1, m_2 = 2, m_3 = 3$  et  $v_1 = 6, v_2 = 10, v_3 = 12$ .  
Aucun algorithme glouton ne fonctionnera dans cette situation (c'est difficile à démontrer), il faut recourir à la programmation dynamique (pas abordée en Algo5).

### Exercice 5 :

Dans un petit restaurant, un serveur (seul) doit servir  $n$  clients ; il sait qu'il lui faudra  $t_i$  minutes pour servir le client numéro  $i$ .

Le mécontentement de chaque client se mesure au temps qu'il doit attendre avant d'être servi.

Afin de créer le moins de mécontentement possible, l'objectif est ici de minimiser le total des temps d'attente de tous les clients.

Comment faut-il procéder et pourquoi ?

## Correction de l'exercice 5

Supposons les clients numérotés par leur ordre de service : alors le client  $i$  patiente pendant  $\sum_{j < i} t_j$  minutes.

Le temps d'attente total est alors  $\sum_{i=1}^n \sum_{j < i} t_j$  et on peut réécrire cette somme en  $\sum_{i=1}^n (n-i)t_i$ .

Il est alors clair qu'on a intérêt à ce que  $t_i$  soit petit quand  $n-i$  est grand, donc quand  $i$  est petit. Il faut donc trier les clients par temps de service croissant.

On peut voir qu'en inversant l'ordre de service de 2 clients  $i < j$  tels que  $t_i < t_j$ , on ne peut qu'augmenter le temps d'attente total : en effet  $(i-j)(t_i - t_j) > 0$  donc  $it_i + jt_j > jt_i + it_j$  et donc  $(n-i)t_i + (n-j)t_j < (n-j)t_i + (n-i)t_j$ .