▶ **MPSI – Option Informatique**
Année 2001, Quatrième TP Caml
Vincent Simonet (`http://cristal.inria.fr/~simonet/`)

Algorithmes gloutons

Les algorithmes servant à résoudre les problèmes d'optimisation parcourent en général une série d'étapes, au cours desquelles ils sont confrontés à un ensemble de possibilités. Un algorithme glouton fait toujours le choix qui semble le meilleur sur le moment, *dans l'espoir qu'il mènera à la solution optimale* globalement.

1 Le voleur intelligent

Un cambrioleur entre par effraction dans une maison et désire emporter quelques-uns des objets de valeur qui s'y trouvent. Il n'est capable de porter que x kilos : il lui faudra donc choisir entre les différents objets, suivant leur valeur (il veut bien entendu amasser le plus gros magot possible...).
On suppose dans un premier temps que les objets sont des matières fractionnelles (on peut en prendre n'importe quelle quantité, c'est le cas d'un liquide ou d'une poudre). Il y a n matières différentes, numérotées de 0 à $n-1$, la i-ème ayant un prix $p.(i)$ par kilo. La quantité disponible de cette matière est $q.(i)$. On suppose que tous les prix sont différents deux à deux.

▶ **Question 1** *Proposez un algorithme qui donne un choix optimal pour le voleur. Ce choix est-il unique ? Programmez une fonction* voleur *en Caml qui reprenne cet algorithme (vous pourrez supposer que le tableau p est trié).*

On suppose maintenant que les objets sont non fractionnables (c'est le cas d'une chaise ou d'un téléviseur). Le i-ème objet vaut un prix $p.(i)$ et pèse un poids $q.(i)$.

▶ **Question 2** *Proposez une méthode dérivée de la question 1 (sans la programmer en Caml). Donne-t-elle un choix optimal ?*

2 Réservation SNCF

On suppose que n personnes veulent voyager en train un jour donné. La personne i veut prendre le train $p.(i)$, où p est un tableau d'entiers. Les trains sont numérotés de 0 à $k-1$ et partent dans l'ordre de leur numéro. Chaque train peut contenir au plus c personnes.

▶ **Question 3** *Écrivez une fonction* possible *qui teste si tout le monde peut prendre le train de son choix. Vous veillerez à ce que votre fonction ne modifie pas le tableau passé en argument.*

On suppose maintenant que la personne i, si elle ne peut pas prendre le train $p.(i)$ parce qu'il est complet, accepte de prendre un des trains suivants (s'il y en a un).

▶ **Question 4** *Proposez et programmez un algorithme* sncf *pour affecter chaque personne à un train lorsque c'est possible.*

3 Remplissage d'un camion

On dispose de n marchandises, de poids respectifs $p.(0), p.(1), \ldots, p.(n-1)$, où p est un tableau d'entiers strictements positifs. On supposera ce tableau trié dans l'ordre croissant. On désire remplir un camion dont la charge maximale autorisée est x avec certaines des marchandises, de manière à obtenir le plus grand poids possible inférieur ou égal à x. De manière plus abstraite, on cherche en fait la plus grande valeur inférieure ou égale à x que l'on puisse obtenir en sommant les éléments d'un sous-ensemble de $\{p.(0), \ldots, p.(n-1)\}$ (on précise bien que cést la valeur en question que l'on cherche, et non pas le sous-ensemble correspondant).
On va concevoir un algorithme qui procède en n étapes : à l'étape i, on construira la liste de tous les poids inférieurs ou égaux à x que l'on peut obtenir en prenant des éléments dans $p.(0), \ldots, p.(i-1)$. On a besoin pour cela de quelques fonctions préliminaires.

▶ **Question 5** *Programmez une fonction* ajoute *qui prend pour argument un entier a ainsi qu'une liste d'entiers $m = [m_0; \ldots; m_{k-1}]$ et retourne la liste $[m_0 + a; \ldots; m_{k-1} + a]$.*

▶ **Question 6** *Écrivez une fonction* retire *qui prend l'entier x et une liste m d'entiers et retourne cette liste dans laquelle les éléments strictement supérieurs à x ont été supprimés.*

▶ **Question 7** *Définissez enfin une fonction* list_max *qui calcule le maximum de la liste passée en argument.*

▶ **Question 8** *Déduisez-en une fonction* camion *qui résout le problème. Que pensez-vous de la complexité de votre algorithme ?*

4 Réservation d'une salle

Un organisateur de tournois sportifs désire utiliser au mieux le gymnase local lors de la journée « portes ouvertes ». Il y a n événements numérotés de 0 à $n-1$. L'événement numéro i commence à l'heure d_i et finit à l'heure f_i. Autrement dit, l'événement i recquiert le gymnase durant l'intervalle de temps $[d_i; f_i[$. Le problème est de planifier le nombre maximal d'événements parmi les n dans le gymnase.

▶ **Question 9** *Indiquez comment modéliser la situation. Proposez un algorithme et écrivez une fonction Caml qui résout le problème. On pourra supposer que $f_0 \leq f_1 \leq \cdots \leq f_{n-1}$.*
L'idée gloutonne est sélectionner un à un des événements en prenant à chaque fois celui qui termine le plus tôt parmi ceux qui sont encore possibles. Le plus simple est probablement d'écrire une fonction récursive...

▶ **Question 10** *Pouvez-vous adapter votre algorithme de manière à optimiser la durée d'utilisation du gymnase ?*

5 L'angoisse de la panne sèche

Une route comporte n stations services, numérotées dans l'ordre du parcours, de 0 à $n-1$. La première est à une distance $d.(0)$ du départ, la deuxième est à une distance $d.(1)$ de la première, la troisième à une distance $d.(2)$ de la deuxième, etc., la fin de la route est à une distance $d.(n)$ de la n-ième et dernière station service.

Un automobiliste prend le départ de la route avec une voiture dont le réservoir d'essence est plein. Sa voiture est capable de parcourir une distance r (mais pas plus !) avec un plein.

▶ **Question 11** *Donnez une condition nécessaire et suffisante pour que l'automobiliste puisse effectuer le parcours. On la supposera réalisée par la suite.*

▶ **Question 12** *L'automobiliste désire faire le plein le moins souvent possible. Écrivez une fonction Caml* rapide *qui détermine à quelles stations-service il doit s'arrêter.*
Une méthode consiste à toujours aller le plus loin possible sans faire le plein, c'est-à-dire à faire systématiquement le plein à la dernière station avant la panne sèche...

Maintenant, l'automobiliste part avec un réservoir vide. Il doit au départ acheter de l'essence (autant qu'il veut dans la limite de la contenance r de son réservoir) au prix de $e.(0)$ euros par litre. Par la suite, à la station numéro i, l'essence coûte $e.(i)$ euros par litre.

▶ **Question 13** *La voiture consomme exactement un litre de carburant par unité de distance. Écrivez une fonction* econome *qui indique à quelles stations l'automobiliste doit s'arrêter et combien de litres il doit prendre à chaque fois, afin que son trajet lui coûte le moins cher possible.*

▶ **MPSI – Option Informatique**
Année 2001, Quatrième TP Caml
Vincent Simonet (`http://cristal.inria.fr/~simonet/`)

Algorithmes gloutons

Un corrigé

▶ **Question 1** Pour amasser le plus gros butin, il suffit de considérer les différentes substances en commençant par la plus chère. À chaque fois, on en prend la quantité maximale (soit en finissant de remplir le sac, soit en prenant toute la quantité disponible).

```
let voleur p q x =
  let n = vect_length q in
  let reste = ref x in
  let reponse = make_vect n 0 in

  for i = (n - 1) downto 0 do
    reponse.(i) <-
      if q.(i) >= !reste then !reste else q.(i);
    reste := !reste - reponse.(i);
  done;
  reponse
;;
```

▶ **Question 2** L'algorithme proposé pour la question 1 ne peut pas s'étendre. Par exemple, si on peut porter 3 kilos et qu'il y a deux objets de 1 kilo de valeur 2 et un objet de 3 kilos de valeur 3, l'algorithme dit de prendre l'objet de 3 kilos alors qu'on peut amasser un meilleur butin en prenant les deux objets de 1 kilo.

▶ **Question 3** On se contente de calculer à un tableau train dont la case d'indice i contient le nombre de voyageurs désirant prendre le train numéro i. Si une des cases du tableau a un contenu strictement supérieur à c, on retourne false.

```
let possible c p k =
  let trains = make_vect k 0 in
  let n = vect_length p in
  let resultat = ref true in
  for i = 0 to n - 1 do
    trains.(p.(i)) <- trains.(p.(i)) + 1;
    if trains.(p.(i)) > c then
      resultat := false
  done;
  !resultat
;;
```

▶ **Question 4** On considère successivement tous les voyageurs à l'aide d'une boucle **for**. Pour chacun d'entre eux, on cherche le premier train qui n'est pas complet par une boucle **while**. Si ce n'est pas possible, on lève une exception.

```
let sncf c p k =
  let n = vect_length p in
  let trains = make_vect k 0 in
  let resultat = make_vect n 0 in
  let n = vect_length p in
  for i = 0 to n - 1 do
    let j = ref p.(i) in
    while !j < k && trains.(!j) >= c do
      j := !j + 1
    done;
    if !j = k then failwith "complet";
    resultat.(i) <- !j;
    trains.(!j) <- trains.(!j) + 1
  done;
  resultat
;;
```

▶ **Question 5**

```
let rec ajoute a = function
    [] -> []
  | t :: q -> (t + a) :: (ajoute a q)
;;
```

▶ **Question 6**

```
let rec retire x = function
    [] -> []
  | t :: q when t > x -> retire x q
  | t :: q -> t :: (retire x q)
;;
```

▶ **Question 7**

```
let rec list_max = function
    [] -> failwith "Liste vide"
  | t :: [] -> t
  | t :: u :: q ->
    if t > u then (list_max (t :: q))
    else (list_max (u :: q))
;;
```

▶ **Question 8**

```
let camion p x =
  let n = vect_length p in
  let liste = ref [0] in
  for i = 0 to n - 1 do
    liste :=
      (retire x (ajoute p.(i) !liste))
      @ (! liste)
  done;
  list_max !liste
;;
```

▶ **Question 9** On utilise une fonction récursive auxiliaire aux pour considérer successivement tous les événements. Elle prend deux arguments : h, l'heure de fin du dernier événement sélectionné et i le numéro de l'événement courant.

```
let salle d f =
  let n = vect_length f in
  let rec aux h i =
    if i >= n then []
    else if d.(i) < h then aux h (i + 1)
    else i :: (aux f.(i) (i + 1))
  in
  aux 0 0
;;
```

▶ **Question 11** L'automobiliste peut effectuer le parcours si et seulement si la distance entre deux stations service consécutive est toujours inférieure ou égale à r.

▶ **Question 12** On utilise à nouveau une sous-fonction récursive aux. Elle a deux arguments : i, le numéro de la station service au niveau de laquelle l'automobiliste « se trouve », et *reste*, la quantité d'essence encore présente dans son réservoir.

```
let rapide d r =
  let n = vect_length d in
  let rec aux i reste =
    if i = n then []
    else
      begin
        if d.(i) > reste then
          i :: (aux (i + 1) (r - d.(i)))
        else aux (i + 1) (reste - d.(i))
      end
  in
  aux 0 r
;;
```