IF2 – Algorithmes gloutons

Les algorithmes sont regroupés selon le type de problème qu'ils aident à résoudre ou selon les méthodes employées. Les algorithmes gloutons suivent une stratégie simple : lorsqu'à chaque étape, un choix doit être fait, c'est le choix optimal à ce moment qui est fait.

Objectifs :

- Distinguer un choix d'optimisation locale ou globale ;
- Reconnaître une stratégie gloutonne dans un programme ;
- Savoir utiliser un algorithme glouton pour résoudre un problème.

I – Problème du voyageur

I-1) Problème d'optimisation

On cherche à déterminer un itinéraire minimisant la distance totale parcourue pour aller de Nancy à Nancy en visitant les villes de Paris/Reims/Troyes et Metz. On peut visiter ces villes dans n'importe quel ordre.

Distances	Nancy	Metz	Paris	Reims	Troyes
Nancy	0	55	303	188	183
Metz	55	0	306	176	203
Paris	303	306	0	142	153
Reims	188	176	142	0	123
Troyes	183	203	153	123	0

Une solution d'étude est d'énumérer tous les ordres possibles, et calculer pour chacun la distance correspondante, pour sélectionner

ensuite la plus petite. Dans notre cas on a 24 trajets possibles mais on peut s'intéresser qu'à 12 d'entre eux, car les douze autres sont le même trajet mais dans un sens de rotation inversé. En notant N=Nancy, M=Metz...on obtient :

Itinéraire	Kilométrage/étape	Total
N-M-P-R-T-N	55+306+142+123+183	809
N-M-P-T-R-N	55+306+153+123+188	825
N-M-T-P-R-N	55+203+153+142+188	741
N-T-M-P-R-N	183+203+306+142+188	1022
N-T-M-R-P-N	183+203+176+142+303	1007
N-M-T-R-P-N	55+203+123+142+303	826
N-M-R-T-P-N	55+176+123+153+303	810
N-M-R-P-T-N	**55+176+142+153+183**	**709**
N-R-M-P-T-N	188+176+306+153+183	1006
N-R-M-T-P-N	188+176+203+153+303	1023
N-R-T-M-P-N	188+123+203+306+303	1123
N-T-R-M-P-N	183+123+176+306+303	1091

Le meilleur itinéraire sera donc le circuit : Nancy-Metz-Reims-Paris-Troyes-Nancy. Cependant la technique utilisée ne sera plus utilisable dès que le nombre d'étapes augmente. Par exemple on aura on 2 millions de circuits possibles si on a 10 étapes à suivre. Pour ce type de problème il n'existe pas d'algorithme donnant la solution optimale en un temps raisonnable lorsque le nombre de villes est grand.

Face à de tels problèmes d'optimisation impossibles à explorer exhaustivement, il est utile de connaître des algorithmes donnant rapidement la une réponse qui sans être optimale, resterait bonne : c'est la méthode gloutonne.

I-2) Algorithmes gloutons

Les algorithmes gloutons sont utilisés pour répondre à des problèmes d'optimisation, c'est-à-dire des problèmes d'algorithmiques dans lesquels l'objectif est de trouver « la meilleure solution » possible selon un critère. Le contexte général est le suivant :

- On considère un problème proposant un grand nombre de solutions. (24 dans notre cas)
- On dispose d'une fonction mathématique évaluant la qualité de chaque solution (la distance parcourue).
- On cherche une solution bonne, voire la meilleure.

Les algorithmes gloutons s'appliquent lorsque :

- La recherche d'une solution peut se ramener à une succession de choix qui précisent petit à petit une solution partielle.
- On dispose d'une fonction mathématique évaluant la qualité de chaque solution partielle.

L'approche gloutonne consiste à construire une solution complète par une succession de choix donnant la meilleure solution partielle.

I-3) Approche gloutonne

Il faut donc aller à la ville la plus proche à chaque choix d'étape parmi les villes non visitées.

Ainsi si on part de Nancy l'itinéraire glouton sera :

- Metz (55km)
- Reims (176km)
- Troyes (123km)
- Paris (153km)
- Et forcément retour à Nancy (303km).

Le circuit glouton représente donc 810 km. On remarque que c'est le quatrième choix optimal, mais il reste proche des meilleurs choix tout en étant bien meilleur que les pires.

II – Problème du rendu de monnaie

II-1) Présentation du problème

Pour simplifier nous supposons que nous n'avons que des pièces dont le système forme un n-uplet $S = (p_0, p_1, \dots p_i, \dots, p_{n-1})$, où les p_i représentent la valeur de la pièce d'indice i. Ces valeurs constituent une suite de nombres entiers strictement croissante avec $p_0 = 1$.

Si $p_0 > 1$, alors certaines sommes ne peuvent pas être rendues. Le problème du rendu de monnaie consiste à trouver une liste d'entiers positifs $[x_0, x_1, \dots, x_{n-1}]$ qui vérifie :

$$r = x_0 p_0 + \cdots + x_i p_i + \cdots x_{n-1} p_{n-1} : la\ contrainte$$

où r est la somme à rendre en minimisant la somme $x_0 + x_1 + x_2 \dots + x_{n-1}$, c'est-à-dire le nombre de pièces utilisées. Dans le système de la zone euro, par exemple, nous avons en centimes les pièces suivantes : $S = (1, 2, 5, 10, 20, 50, 100, 200)$. Il y a de nombreuses manières de rendre huit centimes. Les pièces qui peuvent être utilisées sont les pièces de 1, 2 et 5 centimes. Nous n'écrivons que les triplets possibles :

$$[8, 0, 0], [6, 1, 0], [4, 2, 0], [3, 0, 1], [2, 3, 0], [1, 1, 1]\ et\ [0, 4, 0].$$

Le triplet qui minimise le nombre de pièces est le triplet [1, 1, 1] avec 3 pièces.

II-2) Algorithme glouton

Avec le système donné en exemple, un algorithme glouton fournit la solution optimale. On cherche, par valeur décroissante en

partant de la pièce qui a la plus forte valeur, la première pièce qui a une valeur inférieure ou égale à la somme à rendre r.

- On prend cette pièce, on retranche sa valeur v à r.
- On recommence en partant de la pièce prise en cherchant celle qui a une valeur inférieure ou égale à la nouvelle somme à rendre r — v.
- On prend cette pièce, et ainsi de suite jusqu'à arriver à une somme nulle.

Les entrées sont p pour le système de pièces et r pour la somme à rendre.

```
euros=[1,2,5,10,20,50,100,200] #A mettre dans l'ordre croissant
def monnaie(p,r):
  #Quel n_uplet x faut-il pour obtenir la somme r
  n=len(p)
  i=n-1
  x=n*[0]
  while r>0:
    if r>=p[i]:
      r=r-p[i]
      x[i]=x[i]+1
    else:
      i=i-1
  return x
```

```
monnaie(euros,19)
```

```
[0, 2, 1, 1, 0, 0, 0, 0]
```

Dans le cas général, avec un système différent de celui montré en exemple, c'est un problème difficile à résoudre. Mais dans presque tous les systèmes de monnaie, l'algorithme glouton est optimal. Pour rendre la monnaie, on rend la pièce (ou le billet) de valeur maximale, et on continue tant qu'il reste quelque chose à rendre.

Le système monétaire utilisé en Angleterre jusque dans les années 1960 comportait une multitude de pièces :

{1 penny, 3 pence, 4 pence, 6 pence, 12 pence soit 1 shilling...}.

Pour rendre 8 pence par exemple, le choix glouton donne une pièce de 6 pence et deux pièces d'un penny, alors que le choix optimal est 2 pièces de 4 pence. L'algorithme glouton, dans ce cas n'est pas l'optimisation globale.

Un problème d'optimisation consiste à trouver parmi les différentes solutions d'un problème une solution qui soit la meilleur possible. La stratégie gloutonne est une heuristique visant à produire efficacement une solution **qu'on espère bonne, sans qu'elle soit nécessairement optimale**. Elle procède par une suite de choix en sélectionnant la solution qui paraît être la **meilleure locale**.