Notions d'algorithmique 1 - quelques définitions

Un algorithme non déterministe est un algorithme qui peut présenter différents comportements sur différentes exécutions, par opposition à un algorithme **déterministe**. Plusieurs facteurs peuvent être à l'origine du fonctionnement non déterministe d'un algorithme :

- S'il utilise une entrée utilisateur, un minuteur matériel, une variable aléatoire ou des données externes.
- S'il opère d'une manière sensible au temps, par exemple si plusieurs processeurs écrivent dans la même variable au même moment. Dans ce cas, l'ordre d'écriture affecte le résultat.
- Si une erreur matérielle entraine un changement d'état imprévisible.

La programmation dynamique est une **méthode** d'optimisation procédant par énumération implicite des solutions. Elle fournit donc une **solution exacte.**

Un algorithme glouton (greedy **algorithm** en anglais, parfois appelé aussi algorithme gourmand) est un algorithme qui suit le principe de faire, étape par étape (et sans revenir sur une décision précédente), un choix optimum local. Dans certains cas cette approche permet d'arriver à un optimum global, mais dans le cas général c'est une heuristique.

En optimisation combinatoire, théorie des graphes et théorie de la complexité, une **heuristique** est un **algorithme** qui fournit rapidement une solution réalisable (exploitable, satisfaisante, approchée, …) , mais pas nécessairement optimale, pour un problème d'optimisation complexe.

Exemple d'algorithme glouton : Le problème du rendu de monnaie

Ce problème s'énonce de la façon suivante : étant donné un système de monnaie (pièces), comment rendre une somme donnée de façon optimale, c'est-à-dire avec le nombre minimal de pièces ?

L'approche "gloutonne" de notre algorithme revient à considérer les pièces une par une, en commençant par les pièces de plus grande valeur, et de diminuer le montant à rendre (en enlevant la valeur de la pièce que l'on considère à ce moment) jusqu'à ce que le montant soit inférieur à la valeur de la pièce que l'on regarde. On considère alors la plus grande valeur de pièce inférieure au montant, et on itère le raisonnement jusqu'à ce que le montant atteigne zéro.

Par exemple, la machine doit rendre 1 euro 22 centimes et nous avons à notre disposition des pièces de : 1 euro, 50 centimes, 10 centimes, 2 centimes, 1 centime.

Le principe de notre algorithme est le suivant :

1. On regarde la pièce de 1 €. Alors on peut enlever une pièce de 1 € au montant : on obtient 22 centimes à rendre. Or 22 centimes est inférieur à 1 euro et à 50 centimes, donc on passe à la pièce de 10 centimes.
2. On regarde la pièce de 10 centimes. Alors on peut enlever deux fois la pièce de 10 centimes pour obtenir 2 centimes à rendre (inférieur à 10 centimes et à 5 centimes).
3. On regarde la pièce de 2 centimes. Il suffit d'enlever une pièce de 2 centimes pour rendre le montant total.

   Au final, l'algorithme retourne "rendre 1 pièce de 1 euro, 2 pièces de 10 centimes, 1 pièce de 2 centimes".

Exercice 1:

Codez en langage C cet algorithme en précisant le nombre d'itérations nécessaires au rendu de monnaie. Aidez-vous du fichier *renduMonnaieEtudiant.cpp*

Testez votre programme avec différentes valeurs de monnaie à rendre et concluez si la solution proposée par l'algorithme est satisfaisante et/ou optimale.

Compter le nombre d'itération est utile pour évaluer les performances d'un algorithme et il est parfois utile de comptabiliser également les tests effectués au cours de son déroulement.

Exercice 2 :

Imaginez une solution pour modifier le code précédent permettant de compter le nombre de tests de la boucle while((MonnaieARendre>=tableauPieces[i])

Autre exemple d'algorithme glouton, on veut appliquer l'algorithme précédant au problème de l'affranchissement de courrier.

Problème : on dispose des timbres de valeurs 1,10,21,34,70 et 100 centimes et on souhaite affranchir un courrier pour 140 centimes.

Exercice 3

Modifiez le programme du rendu de monnaie pour répondre à ce nouveau problème et conclure sur l'efficacité de l'algorithme.