Algorithmique
TD n° 9 : Algorithmes gloutons

Les algorithmes gloutons sont assez naturels à concevoir. Leur principale difficulté consiste à savoir s'ils sont corrects, c'est-à-dire s'ils terminent et donnent toujours le résultat attendu. Pour montrer qu'un algorithme est incorrect, le plus simple est généralement de donner un contre-exemple. Pour montrer qu'un algorithme glouton est correct, une approche souvent efficace consiste à formuler un invariant et à montrer par récurrence sur le temps qu'il reste vrai pendant toute l'exécution.

Exercice 1 : un problème d'emploi du temps

Les cours de l'UFR *Chaos appliqué* ont tous lieu le mercredi, à des horaires déterminés sans la moindre concertation. Chaque UE offre le même nombre d'ECTS, et ne nécessite aucun travail personnel pour être validée. Les cours n'ont pas forcément la même durée. Pour maximiser les crédits obtenus, il suffit donc de s'inscrire au plus grand nombre possible d'UE... Malheureusement, le secrétariat refuse de procéder aux IP lorsque les créneaux horaires se chevauchent. Parmi les algorithmes gloutons suivants, lesquels fournissent une solution optimale ? Justifier.

1. Schéma général de l'algorithme par ajouts successifs : former la liste des cours auxquels on s'inscrit en les ajoutant à sa liste un par un. Tant qu'il reste des UE qu'on peut ajouter à sa liste, choisir une UE α parmi celles-ci, avec α...

- (a) se terminant le plus tard possible ;
- (b) de durée minimale ;
- (c) se terminant le plus tôt possible ;
- (d) commençant le plus tôt possible ;
- (e) commençant le plus tard possible ;
- (f) créant le moins possible de conflits avec les cours restants.

2. Schéma général de l'algorithme par éliminations successives : tant qu'il y a des conflits, éliminer de l'ensemble des cours une UE ω...

- (a) de durée maximale ;
- (b) engendrant le plus possible de conflits avec les autres cours.

3. Algorithme mixte : tant qu'il reste des UE disponibles, soit α et β les deux UE commençant le plus tôt (α avant β).

- si α et β sont disjointes, choisir α ;
- sinon, si α recouvre β, éliminer α ;
- sinon, éliminer β.

Exercice 2 : fractions égyptiennes

La façon égyptienne d'écrire des fractions $\frac{a}{b}$ avec $a, b \in \mathbb{N}$ et $a < b$ consiste à les écrire comme une somme de *fractions unitaires* (de la forme $\frac{1}{d}$) *distinctes*. Par exemple

$$\frac{2}{3} = \frac{1}{3} + \frac{1}{4} + \frac{1}{12}$$

Le but de cet exercice est de minimiser le nombre de fractions utilisées dans la somme.
Comment peut-on écrire $\frac{5}{6}$?
On considère les deux algorithmes suivants, dus tous les deux à Fibonacci (13ème siècle)

Algorithme 1

1. écrire $\frac{a}{b}$ comme la somme de a termes $\frac{1}{b}$;
2. tant qu'il reste des termes identiques dans la somme, en prendre un de plus petit dénominateur et le décomposer grâce à l'identité

$$\frac{1}{b} = \frac{1}{b+1} + \frac{1}{b(b+1)}$$

Ainsi pour l'exemple $\frac{2}{3}$ on commence par écrire $\frac{1}{3} + \frac{1}{3}$ puis l'on décompose le deuxième $\frac{1}{3}$ en $\frac{1}{3+1} + \frac{1}{3(3+1)}$ ce qui donne la décomposition précédente.

Algorithme 2 (glouton)

1. partir avec la somme d'une fraction $\frac{a}{b}$ avec $a < b$;
2. tant qu'elle n'est pas unitaire, appliquer à la dernière fraction de la somme l'identité

$$\frac{a}{b} = \frac{1}{\lceil b/a \rceil} + \frac{a - (b \bmod a)}{b\lceil b/a \rceil}$$

Par exemple $\frac{7}{15} = \frac{1}{3} + \frac{6}{45} = \frac{1}{3} + \frac{2}{15} = \frac{1}{3} + \frac{1}{8} + \frac{1}{120}$

Pour chacun de ces deux algorithmes,

1. Appliquer l'algorithme sur les exemples $\frac{2}{5}$ et $\frac{2}{11}$;
2. (Commencer à) appliquer l'algorithme sur l'exemple $\frac{5}{121}$;
3. montrer qu'il termine en produisant une décomposition correcte ;
4. si elle est optimale (en nombre de termes), le prouver ;
5. sinon, exhiber un contre-exemple ;
6. peut-on donner une borne de la longueur de la suite ainsi produite ?

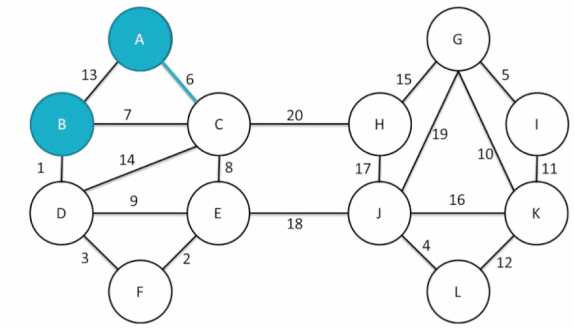
Exercice 3 : algorithme de Borůvka

Il s'agit d'un algorithme d'arbre couvrant de poids minimum écrit en 1926 par Otakar Borůvka pour concevoir un réseau de distribution électrique pour la Moravie du Sud. Il travaille sur des multi-graphes $G = (V, E)$: boucles autorisées, plusieurs arêtes possibles entre deux sommets donnés. Chaque arête a un poids strictement positif. On suppose que G est connexe.

Procédure `Borůvka(`G`)`

- $F \leftarrow \emptyset$
- **tant que** G *a plusieurs sommets* **faire**
  - supprimer les boucles de G
  - **pour** *toute paire de sommets* x *et* y **faire**
    - **si** *plusieurs arêtes entre* x *et* y **alors**
      - ne garder que la plus légère
  - **pour** *tout sommet* x **faire**
    - $e_x \leftarrow \{x, y\}$ l'arête ***minimale*** parmi les arêtes touchant x *(parmi les arêtes de poids minimal, celle avec* y *minimal)*
    - $F \leftarrow F \cup \{e_x\}$
  - **pour** *toute arête* e_x *choisie à l'étape précédente* **faire**
    - contracter e_x *(ses deux extrémités deviennent un seul sommet)*
- **retourner** F

1. Faire tourner l'algorithme sur l'exemple de Wikipedia ci-dessous.



2. Quelle est sa complexité ? En particulier, donner une borne pour le nombre de tours de la boucle externe, et réfléchir à l'implémentation des opérations des boucles *pour*.
3. À quoi la règle « y minimal » sert-elle ? Qu'est-ce qui se passe, si on a par exemple un graphe avec 3 sommets tous connectés avec une arête de même valeur ?
4. Montrer la correction de cet algorithme.
5. On dit qu'un algorithme est *parallélisable* si son temps d'exécution peut être divisé par k en utilisant k processeurs. Cet algorithme est-il parallélisable ?