UNIKIN
FAC. POLYTECHNIQUE

**TRAVAIL DIRIGE 2 – COURS D'ALGORITHMIQUE ET PROGRAMMATION**

***(Promotion : Deuxième Graduat)***

1. Qu'est-ce un algorithme ?
   - *Un algorithme* est une séquence d'étapes de calcul qui transforme une entrée en une sortie afin de résoudre un problème bien spécifique. Il peut se résumer comme une séquence d'étapes qui prend à l'entrée une valeur ou un ensemble de valeurs et qui donne à la sortie une valeur ou un ensemble de valeurs.

2. Qu'est-ce un algorithme efficace ?
   - Un algorithme est dit *efficace* s'il résout un problème posé en utilisant un temps d'exécution minimum. Dans ce cas, on parle de temps d'exécution constant, et de manière idéale c'est l'unité.

3. Que pouvez-vous dire à propos de l'efficacité d'un algorithme ?
   - Le concept efficacité d'un algorithme est très utile en ce qui concerne le choix d'un algorithme et il sied quand même important de signaler que l'efficacité d'un algorithme ne dépend pas de la performance de la machine sur lequel il est exécuté. Pourquoi dit-on cela, c'est parce nous savons qu'avant de passer à l'implémentation, Il est conseillé d'utiliser l'approche théorique pour évaluer l'algorithme car cette dernière n'est dépendante des limites de la machine utilisée pour son exécution. Il faut aussi noter que pour une bonne entrée l'algorithme doit produire une bonne sortie. On parle de l'exactitude d'un algorithme.

4. Citez quelques-unes des techniques de conception d'un algorithme
   Nous pouvons citer :
   - La méthode de la force brute
   - La méthode gloutonne
   - La méthode du diviser pour régner

- La méthode probabiliste
- L'approche par la programmation dynamique

5. Commentons en quelques phrases :

   Quelques commentaires

   - *La force brute* : l'approche consiste à essayer toutes les solutions possibles.
   - *La méthode probabiliste* : elle est aléatoire et elle restreint des algorithmes. La méthode n'est pas trop utilisée compte tenu du fait que le domaine des aléatoires ne sait pas donner exactement le temps d'exécution.
   - *La méthode gloutonne* : On construit une solution de manière incrémentale en optimisant de manière aveugle un critère local. C'est un algorithme qui, étape par étape, fait le choix d'un optimum local. La méthode est mieux que celle de la force brute dans plusieurs contextes.
   - *La méthode de diviser pour régner* : le problème à résoudre est divisé en sous-problèmes semblables au problème initial, mais de taille moindre. Ensuite les problèmes sont résolus de manière récursive et enfin les solutions des sous-problèmes sont combinés pour avoir la solution du problème original.
   - *La méthode de programmation dynamique* : la solution optimale est trouvée en combinant des solutions optimales d'une série de sous-problèmes qui se chevauchent.

6. Qu'est-ce donc ?

   - *Le pseudocode* est appelé également Langage de description d'algorithmes (LDA) est une façon de décrire un algorithme sans référence à un langage de programmation particulier. Souvent elle permet de bien prendre toute la mesure de la difficulté de la mise en œuvre de l'algorithme et de développer une démarche structurée dans la conception de celui-ci. Il permet de donner l'algorithme dans un langage humain, expressif sans respecter la syntaxe d'un quelconque langage de programmation.

   - *Les ordinogrammes,* également appelés algorigrammes, logigrammes ou organigrammes est une représentation graphique normalisée des opérations et des décisions effectuées par un ordinateur.

Nous pouvons également présenter un algorithme en l'implémentant au moyen d'un langage de programmation bien spécifique (python, C++, java, …)

7. Pour quelles raisons une équipe de développeurs de logiciels choisit-elle de représenter les algorithmes par :
   - *Du pseudocode* : il permet à chaque sous-groupe de l'équipe de comprendre l'algorithme au cas où il ferait intervenir des implémentations dans plusieurs langages de programmation.
   - *Des ordinogrammes* aident à faciliter l'exécution et la compréhension de l'algorithme.
   - *Des bouts de code* : dans le cas où le développement par l'équipe se fait directement autour d'un seul langage de programmation.

8. En général pour un problème donné, on peut développer plusieurs algorithmes. Comment identifier le meilleur algorithme de cet ensemble ?
   - Il est vrai que pour un problème donné, on peut développer plusieurs algorithmes. De manière brève, pour identifier le meilleur algorithme nous serons appelés à calculer son temps d'exécution de chacun de nos algorithmes, Le meilleur sera celui qui présentera un temps d'exécution minimum possible lors de la production de la solution du problème. Notons aussi que cet algorithme doit solliciter moins la mémoire (un aspect qui n'est pas souvent pris en compte).

9. En quoi consiste l'analyse d'un algorithme ?
   - Analyser un algorithme revient à trouver son temps d'exécution et conclure en disant s'il est correct ou non. On vérifie si à la bonne entrée correspond la bonne sortie.

10. Quelles sont les deux méthodes d'analyse d'un algorithme ?
    Nous avons :
    - L'analyse théorique
    - L'analyse expérimentale

11. Quels sont les inconvénients de la méthode expérimentale ?
    - L'inconvénient principal de la méthode expérimentale est qu'elle mesure le temps d'exécution d'un algorithme en tenant compte des paramètres de la machine utilisée pour sa mesure. Or les machines présentent toujours des failles

12. En quoi consiste la méthode des opérations primitives ?
    - La méthode repose sur l'affectation des valeurs aux variables, on effectue des opérations arithmétiques, on compare des nombres, on procède à l'index d'un tableau, on suit la référence d'un objet et on finit par sortir une méthode.

13. Qu'est-ce la complexité d'un algorithme ?
    - Parler de la complexité d'un algorithme nous renvoie à nous intéresser à l'analyse des algorithmes en tenant compte de la relation qui existe entre le temps d'exécution et la taille de l'entrée comme paramètre principal d'analyse.

14. En quoi consiste la notation asymptotique ?
    - La notation asymptotique décrit le temps d'exécution des algorithmes par des notations sous forme des fonctions polynômes, linéaires, quadratiques, logarithmiques, … qui sont beaucoup plus claires et précises.

15. Quelles sont les fonctions qui apparaissent le plus souvent lors de l'analyse théorique des algorithmes ?

    Nous avons :

    - *Les fonctions constantes* $f(n) = c$
    - *Les fonctions linéaires* $f(n) = n$
    - *Les fonctions quadratiques* $f(n) = n^2$
    - *Les fonctions cubiques* $f(n) = n^3$
    - *Les fonctions exponentielles* $f(n) = a^n, a>0$
    - *Les fonctions logarithmes* $f(n) = \log_b n$
    - *Les fonctions* $f(n) = n\log_2 n$

*16.*

- L'algorithme le plus efficace parmi un ensemble d'algorithmes permettant de résoudre un problème est celui présentant un temps d'exécution minimum possible lors de la production de la solution du problème. Notons aussi que cet algorithme doit solliciter moins la mémoire (un aspect qui n'est pas souvent pris en compte).

17.

- La taille d'entrée est définie par le nombre d'éléments qui constituent une entrée. A titre illustratif nous pouvons prendre la taille n d'une liste à trier, c'est aussi le total de bits nécessaires à la représentation de l'entrée dans la notation binaire habituelle. En nous référant au cours d'informatique de Premier graduat, nous avons vu la notion des additionneurs, qui sont composés des demi-additionneurs. Pour additionner deux entiers décimaux, il faut au préalable les convertir en décimaux, et chaque décimal s'écrit avec un certain nombre des 0 et des 1. La somme des 0 et des 1 est la taille d'entrée.

18.

- Représentons-nous devant un cas où nous avons une liste composée de tous les territoires de la République Démocratique du Congo. Le problème que nous avons est celui de repérer le territoire de RUTSHURU.
  Le cas le plus favorable par rapport à notre exercice est celui où une fois lancée la première recherche, nous obtenons comme résultat le nom du territoire « RUTSHURU ». Le cas le plus défavorable est par rapport à notre exercice est celui où le résultat escompté est obtenu en dernier lieu de la recherche. Ce dernier cas est d'une
  importance particulière car il indique le temps maximal que prend notre algorithme.

19.

- *Une fonction récursive* est définie comme celle qui s'appelle elle-même au cours de son exécution jusqu'à revenir au cas de départ ou au cas de base. Partant de cette définition, on peut dire que la

récursivité consiste à avoir à la base un cas sur lequel on revient chaque fois au cours de l'exécution.

20.

- *La récursivité binaire* est utilisée pour rechercher un objet dans une séquence triée d'objets. C'est un des algorithmes les plus importants en usage.
- *La récursivité multiple* consiste à effectuer plusieurs appels récursifs à P dans le corps d'une procédure récursive P, à titre d'exemple, nous pouvons considérer la suite de Fibonacci qui est définie pour n entiers naturels : $\begin{cases} f_0 = 0,\ f_1 = 1 \\ f_n = f_{n-1} + f_{n-2} \end{cases}$

21.

Voici la manière selon laquelle un problème récursif doit s'énoncer

- Tout commence par avoir un cas de base ;
- Ensuite, une modification de son état pour se ramener au cas de base
- Enfin, s'appeler elle-même.

A titre illustratif, nous pouvons citer le cas de *la suite de Fibonacci*, *la fonction factorielle* et la *fonction exponentiation rapide* (la puissance nième), …

*Travail exécuté par les étudiants :*

- **IKINDA BALOMBO Chadrack (2GC)**
- **MBALA KABENGELA Mike (2GC)**
- **SEMIKENKE LWANGA Deogratias (2GC)**