11.- Exemples d'algorithmes d'approximation et d'algorithmes probabilistes.

Niveau : MPI

Prérequis : MP2I + Classes de complexité

Motivations : Certains problèmes difficiles, souvent NP-difficiles, ne peuvent pas être résolus de manière exacte par des algorithmes en temps polynomial. Il est possible de pallier ce problème au prix de l'exactitude des solutions, notamment grâce aux algorithmes d'approximation et/ou probabilistes.

I) Algorithmes d'approximation

1.) Introduction aux algorithmes d'approximation

Définition 1: On appelle problème d'optimisation un problème consistant à trouver, parmi un ensemble de solution $\mathcal{S}$ du problème, une solution $S \in \mathcal{S}$ telle que S maximise/minimise une fonction de coût $c : \mathcal{S} \longrightarrow \mathbb{R}$.

Exemples 2: Problème du sac à dos, du plus court chemin, ...

Définition 3: On appelle algorithme d'approximation d'un problème d'optimisation un algorithme retournant des solutions quasi-optimales du problème.

Remarque 4: L'intérêt d'un algorithme d'approximation est d'aller plus vite qu'un algorithme qui résout le problème de manière exacte. Ainsi, pour un problème NP-complet, on cherche un algorithme d'approximation en temps polynomial.

Définition 5 : On dit qu'un algorithme d'approximation a un rapport de performance $\rho(n)$ si pour toute entrée de taille n, le coût C de la solution produite par l'algorithme diffère d'un facteur au moins $\rho(n)$ du coût C^* d'une solution optimale : $\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n)$.
On dit qu'il s'agit d'un algorithme d'approximation $\rho(n)$.

2.) Problème de la couverture de sommets

Définition 6 : Soit $G = (S, A)$ un graphe non orienté.
On appelle couverture de sommets de G un ensemble $S' \subseteq S$ tel que $\forall \{u, v\} \in A$, $u \in S'$ ou $v \in S'$.

Problème 7 : Problème de la couverture de sommets
Entrée : Graphe non orienté $G = (S, A)$
Sortie : S' couverture des sommets de G de cardinal minimal

Propriété 8 : Le problème de décision associé à la couverture de sommets est NP-complet.

Algorithme 9 : Algorithme d'approximation du problème de couverture de sommets.

```
Fonction couverture-sommets(G = (S, A)) :
    C = ∅
    A' = A
    Tant que A' ≠ ∅ :
        Choisir {u, v} dans A'
        C = C ∪ {u, v}
        Supprimer de A toutes les arêtes incidentes à u ou v
    Retourner C
```

Propriété 10 : L'algorithme précédent renvoie une couverture de sommets de G en temps polynomial.

Théorème 11 : Cet algorithme est un algorithme d'approximation 2 du problème de couverture de sommets.

3.) Problème du voyageur de commerce

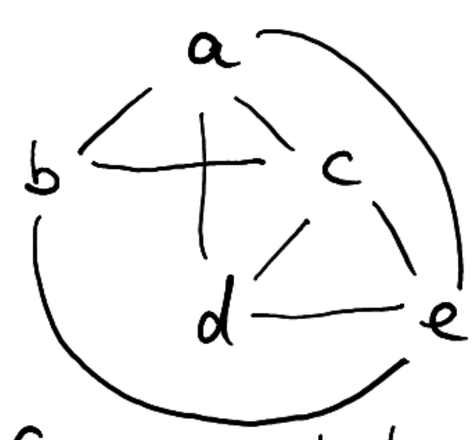
Problème 12 : Problème du voyageur de commerce
Entrées : $G = (S, A)$ un graphe non orienté complet, pondéré par c
Sortie : Chemin hamiltonien de G de coût minimal

Proposition 13 : Dans la définition précédente, si on ajoute la contrainte que c vérifie l'inégalité triangulaire, c'est-à-dire que : $\forall x, y, z,\ c(x, y) + c(y, z) \geq c(x, z)$, alors on parle du problème du voyageur de commerce avec inégalité triangulaire.

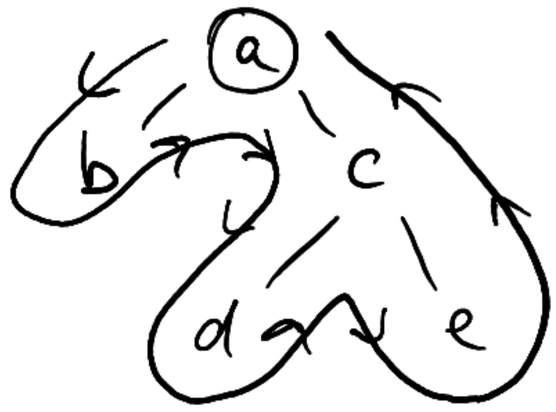
Algorithme 14 : Algorithme d'approximation du problème du voyageur de commerce avec inégalité triangulaire

```
Fonction voyageur-commerce(G = (S, A)) :
    Choisir r ∈ S.
    Calculer un arbre courant minimal T de G avec r pour racine.
    Calculer L un parcours préfixe de T.
    Renvoyer L
```

Exemple 15 : Exemple d'exécution de l'algorithme précédent



Graphe initial



Parcours complet :
a, b, a, c, d, c, e, c, a

Parcours renvoyé :
a, b, c, d, e

Théorème 16: Cet algorithme est un algorithme d'approximation 2 en temps polynomial du problème de voyageur de commerce avec inégalité triangulaire.

4.) Problème de la couverture d'ensemble

DEV: Problème de la couverture d'ensemble

5.) Schéma d'approximation

Definition 17: On appelle schéma d'approximation un algorithme d'approximation prenant en entrée un paramètre $\varepsilon > 0$ tel que, pour ε fixé, l'algorithme d'approximation a une garantie de performance d'au moins $1+\varepsilon$.

Definitions 18: On dit qu'un schéma d'approximation est polynomial si, pour ε fixé, l'algorithme s'exécute en temps polynomial. On dit qu'il est entièrement polynomial si le temps d'exécution est polynomial en n, la taille de l'entrée, et $\frac{1}{\varepsilon}$.

Activité 19: DM/Examen sur le schéma d'approximation entièrement polynomial pour le problème du sac à dos.

II) Algorithmes probabilistes

1.) Introduction aux algorithmes probabilistes

Definition 20: On dit qu'un algorithme est probabiliste s'il utilise une source de hasard, comme un générateur de nombre pseudo-aléatoires.

Exemple 21 : Tri rapide probabiliste

```
Fonction tri_rapide (t, g, d) :
    Si g < d :
        q = partition_rand(t, g, d)
        tri_rapide (t, g, q-1)
        tri_rapide (t, q+1, d)
```

```
Fonction partion_rand(t, g, d)
    i = random_int(g, d)
    Echanger t[i] et t[d]
    x = t[d]
    i = g - 1
    Pour j de g a d-1
        Si t[j] ≤ x
            i = i+1
            Echanger t[i] et t[j]
    Echanger t[i+1] et t[d]
    Retourner i+1
```

Théorème 22 : tri_rapide (t, g, d) trie le tableau t[g:d] et effectue en moyenne $O(n \log(n))$ comparaisons.

2.) Classification d'algorithmes probabilistes

Définition 23 : On appelle algorithme de Monte-Carlo un algorithme probabiliste qui donne un résultat correct avec une certaine probabilité, et termine toujours.

Définition 24 : On appelle algorithme de Las Vegas un algorithme probabiliste qui donne toujours un résultat correct mais dont le temps d'exécution n'est pas déterministe.

Remarque 25 : Un algorithme de Las Vegas peut ne pas terminer, même si souvent ils terminent presque sûrement.

Exemple 26 : L'algorithme du tri rapide probabiliste donne toujours un résultat correct mais le temps d'exécution peut varier entre $O(n \log(n))$ et $O(n^2)$ en fonction du pivot choisi aléatoirement : c'est donc un algorithme de Las Vegas.

3.) Test de primalité :

Problème 27 : Test de primalité
Entrée : $n \in \mathbb{N}$
Sortie : n est-il premier ?

Théorème 28 : Soit $n \in \mathbb{N}$. Alors, n est premier si et seulement si
$\forall a \in [\![2; n-1]\!]$, $a^{n-1} \equiv 1 [n]$.

Algorithme 29 : Test de primalité

```
Fonction test_premier(n):
    a = random_int(2, n-1)
    Renvoyer a^(n-1) ≡ 1 [n]
```

Propriété 30 : Si test_premier(n) renvoie Faux, alors n n'est pas premier, sinon n est "probablement" premier.

4.) Calcul de la médiane

DEV : Calcul de la médiane

5.) Génération de grands nombres premiers

Proposition 31 : La génération de grands nombre premiers est primordial pour la sécurité des systèmes (on parle de nombres premiers de taille cryptographique). Cependant, tester la primalité de manière exacte est trop coûteux : on utilise alors un algorithme probabiliste de Monte-Carlo.

Algorithme 32 : Génération de nombre premier.

```
Fonction genere_premier(k):
    Faire n = random_int(2^k, 2^(k+1)-1)
    Tant que non(test_premier(n))
    Retourner n
```