Archives Examens d'Algorithmique avancée
ENSIMAG Alternants 2A

1 Divide and Conquer (2018)

Un tableau possède un élément majoritaire si plus de la moitié (strictement) de ses entrées sont identiques. Etant donné un tableau A à n éléments, **on se propose de concevoir un algorithme efficace qui permet de déterminer s'il possède un élément majoritaire, et dans l'affirmative de l'identifier.** Les éléments du tableau ne sont pas forcément comparables comme des entiers ou des caractères alphabétiques et il n'est pas possible d'effectuer des comparaisons de la forme A[i] > A[j] (on peut penser par exemple à des images). Par contre, on peut répondre à des questions de la forme « A[i] = A[j]? » en temps constant.

Question 1. Proposez un algorithme simple pour trouver l'élément majoritaire dans un tableau de dimension n et évaluer son coût (on suppose que la mémoire est limitée).

Une autre approche consiste à utiliser le paradigme "diviser pour régner" : couper le tableau A en deux tableaux A1 et A2 de même taille (Dans un premier temps, on pourra supposer que la taille du tableau initial est une puissance de 2). On pourra analyser cet algorithme en distinguant les cas suivants :

1. A1 et A2 ont chacun un élément majoritaire
2. aucun des deux n'en possède
3. un seul des deux a un élément majoritaire

Question 2. Ecrire l'algorithme détaillé en pseudo code.
Question 3. Quel est son coût ? (On donnera l'équation de récurrence et on explicitera comment la résoudre).

On suppose toujours que la taille de A est une puissance de 2.

On se propose maintenant d'**améliorer la solution précédente** par un algorithme qui utilise toujours le paradigme *divide-and-conquer*. Soit cet algorithme garantit que le tableau, A ne contient pas d'élément majoritaire, soit il renvoie un élément a et un entier $c_a > n/2$ tel que a apparaisse au plus c_a fois dans le tableau et tout autre élément apparaisse moins de $n - c_a$ fois.

Question 4. Donner à partir de l'algorithme précédent un algorithme complet qui vérifie si un tableau A contient un élément majoritaire et calculer son coût (en pire cas).

2 Sac-à-dos (2018)

Un algorithme glouton raisonnable consiste à prendre un à un les objets dans l'ordre des densités décroissantes. Cependant, comme nous l'avons montré en cours, cette politique n'est pas bonne.
Question 1. Expliciter la construction d'une instance arbitrairement mauvaise.

Nous allons montrer maintenant comment l'adapter pour obtenir un algorithme avec une garantie de performance. On considère le premier objet qui ne rentre pas dans le sac en appliquant la politique *qualité-prix*. On note j cet objet.

Question 2. Expliciter l'argument pour prouver la propriété suivante :
$OPT \leq \Sigma_{i<j} b_i + b_j$

On en déduit l'algorithme suivant :

- Sélectionner les objets du sac-à-dos par la politique qualité-prix.
- Remplir le sac en considérant les objets dans cet ordre, prenant la plus grande valeur entre le bénéfice obtenu de cette façon et le bénéfice de l'objet suivant j.

Question 3. Montrer que cette politique est une $\frac{1}{2}$-approximation.

3 Variante de 2Partition (2018)

On se propose ici d'étudier la complexité de la variante suivante de 2Partition :

PP (Parfaite Partition)
Input : Etant donnés $2n$ entiers s_i.
Question : Existe-t-il une partition *parfaite* en deux parties égales A_1 et A_2, c'est-à-dire tels que $\Sigma_{i \in A_1} s_i = \Sigma_{i \in A_2} s_i$ où $|A_1| = |A_1| = n$?

Question. démontrer que ce problème est NP-complet.

4 Diviser pour Régner non-équilibré (2019)

On considère ici un problème de taille n qui se décompose en deux sous problèmes de tailles respectives $n_1 = a_1.n$ et $n_2 = a_2.n$ (avec $a_1 + a_2 = 1$ et $\frac{1}{4} \leq a_1, a_2 \leq \frac{3}{4}$).

Le coût de la recomposition des deux sous-problèmes est $max(a_1.n, a_2.n)$

Question 1. Ecrire l'équation qui calcule le coût à une étape.

Question 2. Donner le principe de la résolution de cette équation et calculer le coût global en appliquant le *Master Theorem*, sachant que le coût est constant pour $n = 1$.

5 Une question d'ordre philosophique – Bonus (2019)

Rappelez en quelques lignes la finalité d'une étude de complexité face à l'étude un problème informatique.

En quoi de telles analyses peuvent être utiles à des utilisateurs potentiels ou à des développeurs d'algorithmes ?

6 Programmation Dynamique (2019)

Nous nous intéressons ici à la manipulation de chaines de caractères où l'on se propose ici d'étudier la distance entre deux chaines de caractères.

On définit la notion de *distance* entre les deux chaines de caractères C_1 et C_2 comme le **minimum de modifications nécessaires pour aller de C_1 à C_2**. Seules les trois types de modifications suivantes sont possibles :

1. Ajouter un caractère (σ) au début de la chaine C
2. Supprimer un caractère (σ) au début de la chaine C
3. Changer le premier caractère au début de la chaine par le caractère σ'

On associe un coût différent à chaque opération, les coûts de ces opérations sont respectivement $a(\sigma)$, $s(\sigma)$ et $c(\sigma, \sigma')$.

On définit alors une fonction de distance de la façon suivante :

$d(\sigma X, \sigma Y) = d(X, Y)$
$d(\sigma X, \sigma' Y) = \min(a(\sigma') + d(\sigma X, Y), s(\sigma) + d(X, \sigma' Y), c(\sigma, \sigma') + d(X, Y))$
$d(\epsilon, \epsilon) = 0$ où ϵ représente le caractère vide.

Question 1. Calculer la distance entre les deux chaines suivantes $C_1 = algorithme$ et $C_2 = gorilles$ en supposant que les opérations élémentaires a et s ont le même coût unitaire et c a un coût de $\frac{3}{2}$.

Détailler les différentes étapes.

Question 2. Ecrire une procédure récursive pour calculer la distance d entre deux chaines.

Evaluer le coût de cette procédure.

Question 3. On peut optimiser le calcul de cette fonction recursive en éliminant les calculs redondants. Pour cela, on peut stocker de façon temporaire certains résultats (en ajoutant une *mémoire* représentée par un tableau suffisamment grand).

Donner les grandes lignes de cette procédure optimisée.

Question 4. Ecrire une version itérative de la procédure de la question 2.

7 Autour du branch-and-Bound (2019)

On considère des problèmes de minimisation.
Question 1. Rappelez en quelques lignes le principe de résolution d'un Branch-and-Bound au regard d'un brute force.

Question 2. Sélectionner les choix positifs parmi les suivants concernant une *bonne* fonction d'évaluation des sous-problèmes (explicitez la réponse le cas échéant) :

1. être une borne inférieure
2. être une borne supérieure
3. être choisie en fonction de la politique de séparation
4. être une solution réalisable du sous-problème
5. être calculable facilement

Question 3. Même type de question concernant la politique de séparation. Sélectionner les choix positifs suivants :

1. le parcours n'a aucune importance
2. favoriser un parcours en largeur
3. favoriser un parcours en profondeur
4. prendre l'un ou l'autre parcours selon le problème cible

5. le parcours doit être choisi judicieusement

Question 4. Expliciter comment obtenir un algorithme $\frac{3}{2}$-approximation à partir d'un Branch-and-Bound.

8 Analyse du problème de Coloriage (2020)

8.1 Présentation du problème

On considère un graphe $G = (V, E)$ d'ordre n (nombre de sommets).

Un k-coloriage de G est une application qui associe à chaque sommet v_i de G une *couleur* (c'est-à-dire un indice de $\{1, ..., k\}$) tel que **tous les sommets adjacents sont assignés à une couleur différente**.

Ce problème peut également être vu comme une partition de l'ensemble des sommets V en k sous-ensembles disjoints V_1, ..., V_k correspondant aux sommets de même couleur.

Formellement, on s'intéresse à la résolution du problème de décision suivant :

k-COLOR
Instance. Un graphe $G = (V, E)$ (representé par exemple par sa matrice d'adjacence) et un entier k.
Question. Existe-t-il un k-coloriage de G ?

3-COLOR est la variante du problème où $k = 3$.

L'objectif de la section suivante est de montrer que ce problème est NP-complet.

8.2 Complexité de 3-COLOR

Question 1. (1 pt)
Montrer que 3-COLOR est NP-complet

On détaillera l'algorithme en pseudo-code.

La réduction se fait à partir de la variante *NAE-3SAT* de 3SAT (Not All Equal-3SAT) définie comme les formules logiques qui possèdent une instantiation où toutes les clauses ont au moins un littéral VRAI et au moins un littéral FAUX.

Question 2. (Bonus : +2 pts) Cette question est facultative, on pourra admettre le résultat.
Montrer que le problème NAE-3SAT est NP-complet.

On construit maintenant une réduction de k-COLOR à partir de NAE-3SAT.

On considère une instance (positive) ϕ de NAE-3SAT à n variables x_i et m clauses et on construit un graphe G_ϕ d'ordre $2n + 3m + 1$ de la façon suivante (cette réduction est proche de celle vue en cours pour VC).

- On relie les sommets x_i à $\bar{x_i}$ $\forall i$, $1 \leq i \leq n$.
- On relie toutes les variables x_i à un sommet y.

- Pour chaque clause $C_j = l_{j,1} \vee l_{j,2} \vee l_{j,3}$, on ajoute au graphe un *triangle* (graphe complet de 3 sommets A_j, B_j et C_j) et on les relie respectivement aux trois littéraux l_j d'une même clause[1].

Question 3. (1 pt) Construire le graphe correspondant à la formule suivante (4 variables booléennes et 3 clauses) :

$\phi = (x_1 \vee x_2 \vee \bar{x_3}) \wedge (x_3 \vee \bar{x_1} \vee \bar{x_4}) \wedge (\bar{x_2} \vee x_4 \vee \bar{x_3})$.

Question 4. (2 pts) Montrer que si $\phi \in$ NAE-3SAT, on obtient un 3-coloriage du graphe G_ϕ à partir d'une instantiation valide de ϕ de la façon suivante :

- y est colorié en vert
- x_i (resp. $\bar{x_i}$) est colorié en rouge si la clause est vraie, en bleu sinon
- finalement, on complète chaque triangle de clause avec les autres couleurs (parmi vert, bleu et rouge)

On suppose maintenant que G_ϕ est 3-colorié et que y est en vert.

Question 5. (2 pts) Montrer que si G_ϕ est un 3-coloriage, alors ϕ est positive au sens de NAE-3SAT[2].

En déduire que 3-COLOR est NP-complet.

8.3 Indice chromatique

Le nombre (ou indice) chromatique d'un graphe G, noté $\mathcal{X}(G)$, est le **plus petit entier k pour lequel G admet un k-coloriage**.

Question 6. (1 pt) Déterminer l'indice chromatique du graphe de la figure 1. On justifiera la réponse.

8.4 Résolution exacte

Δ désigne le degré maximal du graphe G.

L'objectif ici est d'établir une borne supérieure de l'indice chromatique.

Question 7. (2 pts) Prouver que les graphes de degré maximum Δ sont $(\Delta + 1)$-coloriables en utilisant un argument de borne inférieure et en construisant un algorithme de coloriage polynomial.

On cherche maintenant à résoudre ce problème de façon exacte.

[1] On rappelle ici que les littéraux sont des variables ou leur complémentaire

[2] Montrer que chaque triangle de clause est relié à un littéral rouge et un littéral bleu

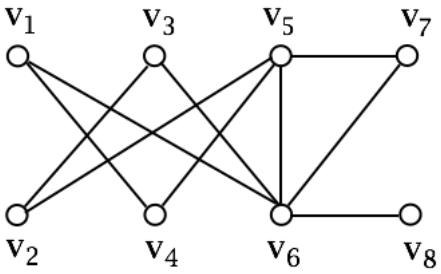


Figure 1: Exemple

Question 8. (2 pts) Proposer une politique de séparation en considérant pour chaque sommet une assignation de couleur pour ce sommet.

On pourra expliciter la politique en s'appuyant sur l'exemple de la figure précédente.

Question 9. (1 pt) Quelle évaluation de borne des sous-arbres suggérez-vous pour couper des branches ?

8.5 Approximation de 3-COLOR

On considère l'algorithme glouton (*First-Fit*) suivant :

- colorier les sommets dans l'ordre $v_1, v_2, \ldots, v_n$
- on utilise la plus petite couleur possible pour colorier v_i (en d'autres termes, v_i a la plus petite couleur non encore utilisée par les voisins de v_i déjà coloriés)

Question 10. (2 pts) Construisez une instance où First-Fit utilise un nombre arbitrairement grand de couleurs par rapport à l'indice chromatique.

Question 11. (1 pt) Que se passe-t-il si les sommets de G étaient triés par degrés décroissants ?

On justifiera la réponse.

Question 12. (2 pts)

Construire un algorithme polynomial pour le 2-coloriage de graphes bi-partis (c'est-à-dire les graphes ne possédant pas de cycles impairs).

9 Une variante de 2Partition (2020)

Prouver que le problème suivant est NP-complet.

PP (Partition parfaite)
Instance: On considère $2n$ entiers s_i.
Question: Existe-t-il une *partition parfaite* en deux ensembles de même cardinaux A_1 et A_2, tels que $\Sigma_{i \in A_1} s_i = \Sigma_{i \in A_2} s_i$ où $|A_1| = |A_1| = n$?

Décrire une solution en programmation dynamique pour résoudre ce problème.

10 Gros sous-tableau (2021)

On se donne un tableau T de taille n (indicé de 1 à n), contenant des nombres (positifs ou négatifs). On cherche à **déterminer le sous-tableau contigu $T[i:j]$ de somme maximale**.

Autrement dit, on cherche les indices i et j qui maximisent $\sum_{k=i}^{j} T[k]$.

Question 1. Calculer (à la main), une solution pour les tableaux suivants (à chaque fois, donner les indices d'un sous-tableau de somme maximale et la somme).

$T_1 = [1, 4, -2, 3, 3, -4, -2, -2, 2, -1]$
$T_2 = [1, -2, 3, 3, 1, -5, -1, -3, 8, -5]$

Question 2. Évaluer le coût d'un algorithme brute force naïf qui calcule les sommes de tous les sous-tableaux.

Question 3. Proposer un algorithme en $\mathcal{O}(n^2)$ pour résoudre le problème (indication: on pourra se servir de la programmation dynamique pour calculer toutes les valeurs de $\sum_{k=i}^{j} T[k]$).

Question 4. Pour résoudre ce problème, on utilise une approche diviser pour régner. On découpe le tableau T en deux sous-tableaux T_L et T_H de taille moitié: T_L contient les éléments de 1 à $\frac{n}{2}$ et T_H contient les éléments de $\frac{n}{2}+1$ à n, on considèrera dans un premier temps que n est une puissance de 2. Le sous-tableau de somme maximale est un des trois tableaux suivants:

- T_L^{max} le sous-tableau de somme maximale de T_L;
- T_H^{max} le sous-tableau de somme maximale de T_H;
- T_{LH} la concaténation du sous-tableau de somme maximale de T_L terminant à $\frac{n}{2}$ et du sous-tableau de somme maximale de T_H commençant à $\frac{n}{2}+1$.

Les sous-tableaux T_L et T_H sont calculés en appelant la fonction récursivement.

- Faire un schéma explicatif de l'algorithme.
- Montrer que l'on peut calculer le tableau T_{LH} en temps $\mathcal{O}(n)$.
- Soit $C(n)$ la complexité de l'algorithme sur un tableau de taille n. Donner une formule de récurrence pour $C(n)$ et donner l'ordre de grandeur de $C(n)$.
- Ecrire l'algorithme récursif en utilisant le langage pseudo code que vous souhaitez.

Question 5. Mathilde prétend que l'on peut résoudre le problème en utilisant l'algorithme suivant:

```
1  Entree  Un tableau T de taille n
2  Sortie  La somme des valeurs du sous-tableau de somme maximale
3  somme_courante ← 0;
4  meilleure_somme ← 0;
5  Pour k = 0 à n − 1 somme_courante ← somme_courante + T[k];
6  Si (somme_courante > meilleure_somme) alors
7  meilleure_somme ← somme_courante;
8  Si (somme_courante ≤ 0) alors  somme_courante ← 0
9  finPour
10 Retourner meilleure_somme
```

- Quelle est la complexité de cet algorithme?
- Cet algorithme rend-il la bonne somme ? Justifier votre réponse,

  c'est-à-dire: si oui, démontrer que l'algorithme est correct, si non donner un contre-exemple.
- Si l'algorithme donne la somme maximale, proposer une modification de l'algorithme pour qu'il retourne également les indices du sous-tableau de somme maximale.

Question 6. (Synthèse): Parmi les algorithmes étudiés, lequel recommandez-vous, et pourquoi ?

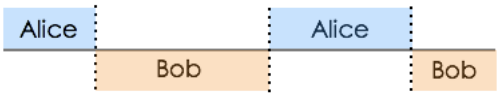
11 Autour du branch-and-Bound (2022)

Question 1. Deux voleurs Alice et Bob se rencontrent après un braquage pour partager leur butin. Ils ont volé un collier de pierres précieuses et ils veulent découper ce collier de la manière la plus équitable possible.

Définition formelle du problème

On considère un collier de n bijoux de k types (n_1 rubis, n_2 diamants, n_3 saphirs, etc.). Voilà un exemple pour $k = 3$.

Le problème et de déterminer une partition équitable en **coupant le collier en plusieurs parties contenant des pierres précieuses consécutives avec un nombre minimum de coupes**.
En d'autres termes, chaque voleur repartira avec une ou plusiers parties du collier avec le même nombre de pierres précieuses que l'autre voleur, comme le montre la figure suivante :



Préliminaire : Deux types de bijoux

Question 2.

- Prouvez que le problème avec $k = 2$ types de pierres précieuses peut être résolu avec seulement deux coupes.

  Détailler l'algorithme.

Passons maintenant au cas plus difficile de $k = 3$ types de pierres précieuses.

Echauffement : un résultat préliminaire négatif pout $k = 3$

- Montrez à partir de la construction d'un contre-exemple que deux coupes ne sont pas suffisantes pour partitioner équitablement un collier pour 3 types de pierres précieuses.

Quelques résultats positifs pour $k = 3$

Commençons par un algorithme simple.

Supposons que n_1, n_2 et n_3 sont des nombres entiers pairs. Alice et Bob doivent ainsi recevoir chacun la moitié de chaque n_i.

- Une stratégie possible pour répartir les pierres précieuses entre Alice et Bob est de commencer par la gauche du collier et de couper au premier endroit avec le maximum de pierres possibles et donner cette partie à Alice.

  Plus précisément, le collier est coupé juste avant d'atteindre $\frac{n_i}{2} + 1$ pour un indice i quelconque $1 \leq i \leq 3$.

  Ensuite, on continue la répartition pour Bob avec la partie la plus grande possible pour lui.

  Et ainsi de suite, repassant de Alice et Bob alternativement avec le même critère.

- **Question 3.** Faites tourner cet algorithme sur l'exemple précédent.

- **Question 4.** Formalisez le en pseudo-code.

- **Question 5.** Construisez un exemple pour montrer que cette stratégie peut conduire à plus de 3 coupes.

  Est-elle arbitrairement mauvaise ?

- **Question 6.** Créez votre propre heuristique et analysez la en terme du nombre de coupes et de complexité.

Résolution exacte : Branch-and-Bound

Le problème du partage en exactement 3 coupes est un problème difficile.

On cherche dans cette section à le résoudre exactement par Branch-and-Bound.

Question 7. Explicitez une stratégie de branchement (parcours de l'arbre des configurations).

Formalisez le problème.

Détaillez une procédure de calcul d'une borne inférieure pour les coupes.

En utilisant l'algorithme glouton ou votre heuristique, détaillez les premières étapes sur un exemple.

Cas particulier d'un collier avec k paires de pierres précieuses

Question 8. Si le nombre de pierres précieuses de chaque type est égal à 2, le problème de partitionement en nombre minimum de coupes peut être résolu facilement quel que soit k.

- Décrivez et analysez une solution optimale dans ce cas.