Algorithmique 2022 - TP 1: Diviser pour régner

Brian Pulfer
Brian.Pulfer@unige.ch

28 Septembre 2022

Remarques:

Veuillez suivre attentivement les spécifications de l'énoncé.

- **Réponses** - Essayez d'être exhaustif dans vos réponses. Les réponses comme "oui" et "non" ne permettent pas d'évaluer vos connaissances.
- **Tracés** - Mettez toujours des étiquettes d'axe et des titres pour les tracés.
- **Implémentation** - Veuillez utiliser exactement les noms fournis pour les fonctions. Vous pouvez utiliser *pytest* avec le script de test donné pour vérifier votre implémentation.
- **Soumission**
  Merci de téléchargé vos devoirs sur moodle:
  - **NomPrenom.pdf** avec votre rapport pour tout les exercices
  - **tp1.py** avec votre implementations
  - *Pas d'autres fichiers* (.pyc, .ipynb, DS_Store, __pycache__, ...)

  Délai - 4 Octobre 2022, 23:59 CET
- Pour toute questions et remarques, merci d'utiliser le forum moodle.

1 Élément majoritaire (3.5 Points)

Dans une liste d'éléments, il y a un élément majoritaire si **plus** de la moitié des éléments de la liste sont les mêmes.

1.1 Implémentation

Veuillez implémenter l'algorithme de **page 25** en implémentant et utilisant les fonctions suivantes:

- `is_majority(A, element)` - Indique si `element` est l'élément majoritaire de `A` (`True` ou `False`).
- `reduce(A)` - Convertit la liste en une liste plus courte en comparant les éléments par paires.
- `dandc(A)` - Utilize la technique *divide and conquer* pour retourner la valeur de son élément majoritaire s'il y en a un, `None` sinon.

Note: Dans `dandc(A)`, l'opération de réduction est effectuée autant de fois que possible (par exemple avec récursivité).

1.2 Analyse

1.2.1 Un élément majoritaire dans `A` est-il nécessairement aussi un élément majoritaire dans `reduce(A)`? Pourquoi? Donner un exemple.

1.2.2 Un élément majoritaire dans `reduce(A)` est-il nécessairement aussi un élément majoritaire dans `A`? Pourquoi? Donner un exemple.

1.2.3 Considérant le pire scénario pour les deux cas, en pratique, l'algorithme va-t-il s'exécuter plus rapidement avec 2^n éléments ou avec $2^n - 1$ éléments pour $n > 2$? Pourquoi? Donner un exemple.

1.3 Comparaison d'exécution

Implémentez la fonction `compare_naive_and_dandc`. La fonction doit afficher le tracé comparant les temps d'exécution de votre algorithme et de l'algorithme naïf fourni pour des tableaux de taille 1'000, 2'000, 3'000, ...10'000. Les tableaux doivent être générés de manière aléatoire, où chaque élément a un possibilité de 50% d'être 0 et n'importe quel nombre entre [1-9] sinon. N'oubliez pas d'ajouter des étiquettes d'axe, un titre et une légende. Quel est le résultat? Commentez.

2 Exponentiation (1.5 Points)

Nous nous intéressons dans cet exercice aux algorithmes capables de calculer des puissances entières quelconques de la forme $base^p$

2.1 Algorithme naïf

Écrire un algorithme naïf d'exponentiation `exp_naive(base, p)` qui multiplie le nombre `base` par lui-même `p` fois. Quelle est la complexité de cet algorithme?

2.2 Algorithme D&C

Tous les nombres réels positifs peuvent être exprimé comme une somme de puissances de 2. Par exemple:

$$53 = 32 + 16 + 4 + 1 = 2^5 + 2^4 + 2^2 + 2^0$$

Appliquer cette information à l'exposant pour créer un algorithme d'exponentiation diviser pour régner (`exp_dandc(base, p)`) avec complexitè logarithmique.

2.3 Comparaison

Implémenter la fonction `compare_exp` pour comparer les deux algorithmes. Tracez le temps d'exécution pour les deux algorithmes lors de l'exponentiation de 2 aux puissances suivantes : `[1000, 2000, 3000, 4000, 5000]`. N'oubliez pas d'ajouter des étiquettes d'axe, un titre et une légende. Quel est le résultat? Commentez.