clement.rouvroy@ens.psl.eu

Somme de sous-array

Thèmes Arbres binaires, Algorithmique, Complexité algorithmique

Type Centrale

Consignes Le candidat respectera le langage imposé (OCaml) et ne devra pas utiliser de librairie hors-programme. Il est invité à **faire des schémas clairs et précis** de ses algorithmes lors des appels au correcteur, c'est à la fois un gain de temps pour les deux et une manière de montrer qu'il a compris ce qu'il manipule. Les preuves écrites doivent être formelles, sauf si la consigne précise que ce n'est pas nécessaire, la rigueur sera évaluée. L'examinateur ne débugera pas votre code, en revanche en cas de doute ("ai-je le droit à telle ou telle librairie ?", "je suis sortie de la VM, comment y revenir ?", "ai-je le droit à une indication pour cette question ?") n'hésitez pas à poser votre question. Elle ne vous dévalorisera pas, si la réponse peut vous retirer des points, l'examinateur vous demandera avant de vous donner la réponse si vous l'acceptez (*si vous n'avez pas de chance, le jour de l'oral il vous retirera les points rien que pour avoir posé la question, par exemple si vous demandez "comment trier une liste en $O(n\,log(n))$?" ou un autre résultat classique du programme, ça sera sûrement retenu contre vous*). Enfin, si l'examinateur n'est pas à votre portée quand vous avez besoin d'une aide / d'une question oral à donner, gardez le bras levé et lisez la suite, ne restez jamais passif.

Le barème n'est la qu'à titre indicatif, votre note finale n'est pas juste `points/total_possible`, il y a un redressement & des ajustements en cas de sujet trop long / trop difficile, donc pas de panique.

Introduction Tout ce sujet se focalise sur le problème de la somme de sous-array. Soit $\mathcal{A}$ un array, on cherche à donner un algorithme effiace pour les requêtes de la forme "Que vaut, pour i et j donnés, $\sum_{k=i}^{j} \mathcal{A}[i]$?".

I - Pendant que ça charge (3)

A traiter impérativement.

Question 1 à l'écrit

1. Encadrez le nombre de noeuds d'un arbre binaire en fonction de sa hauteur
2. Comment trouver le minimum d'un arbre binaire de recherche ? Le maximum ?
3. Quelle est la ligne pour compiler du OCaml sous Linux ? Pour exécuter le fichier produit ?

II - Les arbres segments (2)

Dans tout le sujet, on supposera quand on en aura besoin (et on l'indiquera) que la taille de $\mathcal{A}$ est une puissance de 2.

Un arbre segment est un arbre binaire dont les noeuds représentent des **segments d'un array**. Formellement, étant donné un array $\mathcal{A} := \{a_1, ..., a_n\}$, un arbre segment $\mathcal{S}$ est un arbre binaire dont les feuilles représentent des singletons (le segment $[a_i; a_i]$) et un noeud ayant pour fils l, r représente le segment $l@r$ (la concaténation, par exemple `[0;1]@[1;2] = [0;2]`). On note sur les noeuds les indices correspondent

au segment. La racine sera toujours $0, n-1$. Pour construire le noeud (i, j), on construit un noeud qui a pour fils (tant qu'ils existent) $\left(i, \left\lfloor \frac{j}{2} \right\rfloor\right)$ et $\left(\left\lfloor \frac{j}{2} \right\rfloor + 1, j\right)$.

Question 2 à l'écrit

Soit $\mathcal{A} = \{7; -4; 12; 0\}$, donner l'arbre segment pour l'array $\mathcal{A}$.

L'avantage des arbres segments est qu'ils permettent d'ajouter des labels sur les noeuds de notre arbre et donc de travailler en diviser pour régner.

III - Somme de segments d'array (23)

Etant donné un array $\mathcal{A}$, on veut résoudre des requêtes de types "Quelle est la somme des éléments de $\mathcal{A}$ entre i et j inclus ?".

III.1 - L'algorithme trivial

On considère l'algorithme suivant:

```
query(a,i,j):
  s = 0
  for k=i to j:
    s+= a[k]
  return s
```

Question 3 à l'écrit

Quelle est la complexité de query en espace ? En temps ? (gardez cette valeur dans un coin de votre tête pour la fin sujet)

III.2 - L'Algorithme sur arbre segment

Le but de ce sujet va être d'utiliser les arbres segments pour optimiser les étapes de calculs en faisant un pré-calcul acceptable et pouvoir faire plusieurs opérations en une. Pour cela on va coder un algortihme sur des arbres segments qui ont un label en plus: la **somme sur leur intervalle**.

Question 4 code

Définir un type `arbre` OCaml corerspondant. Vous êtes libre de l'implémentation, cependant vous devrez sûrement justifier votre choix oralement.

Question 5 code

Ecrire une fonction `array_vers_arbre: array -> arbre` qui prend en entrée un array et renvoie l'arbre segment associé.

Question 6 à l'écrit

Quelle est la complexité de votre construction ?

Preuve en 2 lignes possibles grace à une propriété du cours (point bonus)

Maintenant on aimerait bien pouvoir se servir de cette structure de données pour répondre à notre question initiale (la somme entre deux indices i et j).

En distinguant trois cas, répondez à la question suivante:

Question 7 code

Ecrire une fonction `somme: arbre -> int -> int -> int` qui prend en entrée un arbre segment, i et j qui renvoie la somme voulue grace à l'arbre.

Remarque: Vous pouvez, dans le cas compliqué, regarder l'intersection de `[i,j]` avec le fils gauche, puis avec le fils droit.

Question 8 à l'écrit

Quelle est la complexité de votre fonction `somme` (une preuve formelle n'est pas nécessaire tant que le résultat est correct)? Est-ce un gain significatif ?

III.3 - Gestion des modifications de l'array

Jusqu'à présent nous savons, au coup d'un pré-calcul acceptable, obtenir une structure de données permettant de répondre efficacement aux requêtes sur l'array initial. Mais si nous modifions **un seul élément de l'array**, il faut tout reconstruire, c'est affreuseuement non optimal. On va régler ce problème dans cette partie.

Question 9 code

Ecrire une fonction `modifie: arbre -> int -> arbre` qui prend en entrée l'arbre d'un array $\mathcal{A}$, l'indice modifié, la nouvelle valeur et qui renvoie un arbre qui prend en compte cette modification.

Question 10 à l'écrit

Comien de feuilles sont modifiées par votre fonction ? Si la réponse est plus que $O(\log(n))$, recommencer.

IV - Optimiser les requêtes préfixes (21)

Dans cette partie on va chercher à minimiser le nombre de noeuds de l'arbre segment produit pour un array $\mathcal{A}$ dans le cas des **requêtes préfixes** (de 0 à j). On va même le rendre égale à $|\mathcal{A}|$.

On va s'assurer que **dans le parcours préfixe de notre arbre, tout noeud qui est $i$ème dans le parcours en profondeur apporte une information qu'on ne pouvait pas obtenir par la simple connaissance des noeuds** $0, ..., i-1$..

IV.1 - Construction

Pour vérifier la propriété énoncée, on va devoir transformer notre arbre, on va passer de l'**arbre segment** à une **array** $\mathcal{F}$ (*de Fenwick*) selon l'algorithme suivant:

```
FENWICK(A):
1  rayer(A).
2  F ← array vide de taille A
3  pour i de 0 à |A| − 1:
4      F[i] ← premier ancêtre non rayé de f
5  renvoyer F
```

L'algorithme `rayer` raye tous les fils droits de l'arbre, comme sur l'exemple ci-dessous:

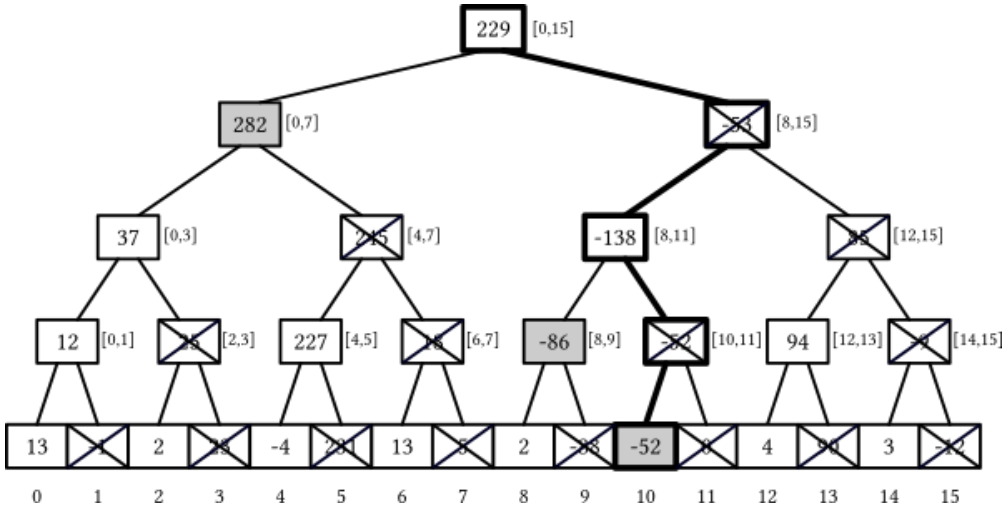

Figure 1: Exemple (algorithmica.org)

L'algorithme de rayage implique de rayer uniquement les fils droits, c'est un choix qui paraît un peu tombé de nul part, pour le comprendre vous pouvez traiter cette question:

QUESTION 11 À L'ÉCRIT

On rappelle qu'on encode le fait d'aller à droite par un 1 et à gauche par un 0.

On considère $i \in [\![0; |A| - 1]\!]$ et f la feuille associée au singleton $\{i\}$ dans l'arbre segment.

Si c est le chemin de la racine à f vu en binaire, comment déduire de c:
- Que f est rayé
- Si c est rayé, c' le chemin qui mène au noeud dont f stocke la somme.

QUESTION 12 CODE

Implémenter l'algorithme ci-dessus en une fonction `fenwick: arbre -> int array` qui prend en entrée un arbre segment et renvoie l'array de Fenwick associée.

IV.2 - UTILISATION DE L'ARRAY

On va maintenant voir comment ce servir de cette array $\mathcal{F}$ nouvellement créée.

Question 13 code

Comment obtenir, en maximum 2 opérations binaires, le premier bit à 1 dans un nombre binaire x de taille n ?

L'implémenter dans une fonction `premier_un: int -> int` qui prend en entrée un entier et renvoie la puissance de deux correspondant au nombre binaire composé de 0 partout sauf sur le premier bit valant 1 de l'entrée.

On va se servir de ce bit dans l'algorithme suivant:

SOMME_PREFIXE$(\mathcal{F}, i)$:

1 $r \leftarrow 0$.
2 **while** $i \neq 0$**:**
3 $\quad r \leftarrow r + \mathcal{F}[i]$
4 $\quad i -= premier\_un(i)$
5 renvoyer r

Question 14 à l'écrit, DIFFICILE

Montrer la correction de l'algorithme `somme_prefixe`.

Question 15 code

Implémenter l'algorithme ci-dessus.

En déduire une fonction `somme_quelconque : int array -> int -> int -> int` qui prend en entrée $\mathcal{F}$, i et j et renvoie la somme dans l'array initiale en utilisant l'array de Fenwick.

Question 16 à l'écrit

Qu'est-ce qui est le plus optimal entre l'algorithme `query` du début et l'algorithme `somme_quelconque` obtenu ? (on indiquera le temps de pré-calcul nécessaire à l'algorithme `somme_quelconque` et on le nuancera avec le coût dans le pire cas de l'algorithme)

V - Passe-temps

A ne traiter que si vous avez terminé le sujet. N'accordera **aucun** point si vous n'avez pas traité tous qui précède.

clement.rouvroy@ens.psl.eu

Exercice 1 Porté disparu

Un tableau $A[1...n]$ contient tous les entiers de 1 à n sauf 1. Il est facile de trouver cet entier manquant en utilisant un tableau auxiliaire de booléen. Cependant on va supposer ici qu'on est sur une architecture différente et que l'opération "accéder au i-ème élément d'un tableau" est $O(\log(n))$ avec n la taille de l'entier. On se donne à la place l'opération "Accéder au j-ème bit de l'entier numéro i" qui se fait elle en $O(1)$ (on se convaincra à la main que ça ne vient pas contredire l'hypothèse de $\log(n)$ pour l'opération usuelle).

Montrer qu'on peut tout de même trouver l'entier manquant en $O(n)$.

Exercice 2 Palindrome

On stocke des chaînes de caractères par liste chaînée. Par exemple `abc` devient `a->(b->(c->NULL))`. Développer un algorithme $O(n)$ pour dire si une chaîne est un plaindrome ou non.