Examen

CPES2: Algorithmique et Applications

Pierre SENELLART

17 janvier 2022

Les seuls documents autorisés sont deux feuilles A4, recto-verso (4 pages) avec le contenu de votre choix. Cet examen dure deux heures et contient un unique problème, formé de 4 parties et de 15 questions. Il est noté sur 20 points.

Tiers-temps : *Les étudiants et étudiantes disposant d'un tiers-temps doivent l'indiquer sur la copie et ne doivent traiter ni la partie C (questions 7 et 8) ni les questions 13 et 15. Ils et elles seront notés uniquement sur les questions restantes (1 à 6, 9 à 12, 14).*

On considère dans ce problème des structures de données permettant de représenter une *partition* d'un ensemble fini d'entiers {1,...,n}. On rappelle qu'une partition d'un ensemble X est un ensemble $\{X_1, \ldots, X_k\}$ de sous-ensembles de X tels que :

— $\bigcup_{i=1}^{k} X_i = X$;

— $X_i \neq \varnothing$ pour tout $1 \leqslant i \leqslant k$;

— $X_i \cap X_j = \varnothing$ pour tous $1 \leqslant i, j \leqslant k$ avec $i \neq j$.

Chaque X_i est appelé une *classe d'équivalence*.

Les stuctures de données que nous allons considérer devront supporter les opérations suivantes pour gérer une partition X d'un ensemble $\{1, \ldots, n\}$:

constructeur() Construire une partition vide $\mathcal{X} = \varnothing$ de l'ensemble vide.,

inserer_singleton() Ajoute un nouvel entier $n+1$ à l'ensemble $\{1, \ldots, n\}$ et ajouter le singleton $\{n+1\}$ à la partition $\mathcal{X}$: on obtient donc une partition $\mathcal{X} \cup \{\{n+1\}\}$ de l'ensemble $\{1, \ldots, n+1\}$.

representant(p) Étant donné un entier p en entrée avec $p \in \{1, \ldots, n\}$, renvoie un entier p' *représentatif* de la classe d'équivalence de p dans $\mathcal{X}$: en cas d'appel répété à cette fonction, p' doit être le même pour tous les entiers appartenant à cette même classe d'équivalence.

fusion(p,q) Étant donnés deux entiers p et q qui sont les représentants des classes d'équivalence X_i et X_j, fusionne les classes d'équivalence X_i et X_j : ainsi, $\mathcal{X}$ devient $(\mathcal{X} \setminus \{X_i, X_j\}) \cup \{X_i \cup X_j\}$.

Le but de cet examen est d'étudier plusieurs manières d'implémenter une telle structure de partition. Dans tout l'examen, on notera n le nombre d'éléments de l'ensemble dont on construit une partition et donc on parlera par exemple d'une opération ayant une complexité en $\Theta(n)$ quand cette complexité est linéaire en le nombre d'éléments de l'ensemble.

A. Préliminaires (3,5 points)

On suppose que l'on souhaite réaliser les opérations suivantes sur une partition :

— On construit une partition vide.

— On y insère le singleton $\{1\}$.

— On y insère le singleton $\{2\}$.

— On y insère le singleton $\{3\}$.

— On y insère le singleton $\{4\}$.

— On fusionne les classes d'équivalence des éléments 1 et 3.

— On fusionne les classes d'équivalence des éléments 1 et 4.

— On affiche un élément représentatif de chacun des éléments 1, 2, 3, 4.

1. (0,5 point) Quelle est la partition représentée par cette structure de données après toutes ces opérations ?
2. (1 point) Quels sont tous les affichages possibles de cette suite d'opérations ?
3. (1 point) Combien y a-t-il de partitions différentes sur l'ensemble $\{1, 2, 3\}$?
4. (1 point) Montrer que le nombre de partitions d'un ensemble à n éléments est en $\Omega(2^n)$.

B. Tableau dynamique (3 points)

On considère d'abord une implémentation dans laquelle on implémente une partition par un *tableau dynamique* T de n cases, où la case $T[p]$ est un représentant de la classe d'équivalence de p (choisi arbitrairement, mais unique pour l'ensemble des éléments de cette classe d'équivalence).

5. (2 points) Écrire sous la forme d'une classe Python une telle implémentation d'une partition (avec son constructeur, et ses méthodes `inserer_singleton`, `representant`, `fusion`). On pourra utiliser une liste Python comme tableau dynamique.
6. (1 point) Quelle est la complexité asymptotique amortie des opérations inserer_singleton, representant, fusion avec cette implémentation ? Justifier brièvement.

C. Listes chaînées (3 points)

On s'intéresse désormais à une implémentation dans laquelle chaque classe d'équivalence est stockée comme une liste doublement chaînée des entiers de la classe d'équivalence. On suppose de plus qu'on dispose également d'un tableau dynamique qui associe à chaque entier $1 \leqslant p \leqslant n$ le chaînon de la liste correspondant à cet entier.

7. (2 points) Écrire sous la forme d'une classe Python une telle implémentation d'une partition (avec son constructeur, et ses méthodes `inserer_singleton`, `representant`, `fusion`).
8. (1 point) Quelle est la complexité asymptotique amortie des opérations inserer_singleton, representant, fusion avec cette implémentation ? Justifier brièvement.

D. Forêt d'ensembles disjoints (10,5 points)

On considère maintenant une nouvelle implémentation dans laquelle la partition est représentée par un ensemble d'arbres (on appelle ça une *forêt*). Chaque arbre représente une classe d'équivalence et les nœuds de cet arbre les éléments de cette classe d'équivalence. Le représentant de chaque classe d'équivalence est l'entier à sa racine. On suppose de plus qu'on dispose également d'un tableau dynamique qui associe à chaque entier $1 \leqslant p \leqslant n$ le nœud de l'arbre correspondant à cet entier.

Quand deux classes d'équivalence doivent être fusionnées, on fait de l'un des arbres un fils de la racine de l'autre arbre.

De plus, on associe à chaque racine d'un arbre un *rang* : ce rang est initialement 0 mais est augmenté de 1 à chaque fois qu'une fusion cause l'ajout d'un nouveau fils à cette racine.

9. (1 point) Quelle est la complexité des opérations `inserer_singleton` et `fusion` ? Justifier.
10. (0,5 point) Quelle est la complexité de l'opération `representant` en terme de la hauteur maximale d'un arbre de la forêt ?
11. (1 point) Montrer que si on ne met pas de contrainte supplémentaire sur la manière dont la fusion est effectuée, la hauteur maximale d'un arbre de la forêt peut être un $\Theta(n)$.
12. (2 points) Proposer une approche pour la fusion (ne changeant pas la complexité asymptotique de l'opération `fusion`) permettant de garantir que la hauteur maximale d'un arbre de la forêt est en $O(\log n)$. Prouver que c'est bien le cas.
13. (0,5 point) Comparer les complexités obtenues avec cette approche à celles des parties B et C.
14. (4 points) On peut en fait optimiser encore cette structure, de la manière suivante : chaque fois que l'on applique l'opération `representant` à un nœud n, on fait du nœud n et de chacun de ses ancêtres, racine exclue, un enfant direct de la racine.

    Implémenter sous la forme d'une classe Python cette structure de données, avec ce raffinement à la méthode `representant` ainsi que le raffinement de la question 12 à la méthode `fusion`.
15. (1,5 point) On admettra que la complexité asymptotique amortie d'une séquence d'opérations avec ces optimisations est en $O(\alpha(n))$ où α est la *fonction Ackermann inverse*, une fonction qui croît extrêmement lentement (pour toutes les valeurs de n qu'on peut imaginer utiliser en pratique, $\alpha(n) \leqslant 4$).

    On considère maintenant le problème suivant : on se donne un *graphe*, c'est-à-dire un ensemble fini V de nœuds et un ensemble fini $E \subset V \times V$ d'arêtes, indiquant des connexions entre ces nœuds. On dit qu'il existe un chemin d'un nœud u à un nœud v s'il existe une suite finie d'arêtes $(u_0, u_1), (u_1, u_2), \ldots, (u_\ell - 1, u_\ell)$ dans E telles que $u_0 = u$ et $u_\ell = v$. Une *composante fortement connexe* est un ensemble maximal de nœuds C tel qu'il existe un chemin de tout nœud u à tout nœud v dans C.

    Montrer qu'on peut utiliser une forêt d'ensemble disjoints pour trouver l'ensemble des composantes fortement connexes d'un graphe en proposant un algorithme. Quelle est la complexité de ce calcul en terme de $|V|$ et $|E|$?