clement.rouvroy@ens.psl.eu

Arbrinator

Thèmes Arbres binaires, Algorithmique, Complexité algorxithmique

Type Centrale

Consignes Le candidat respectera le langage imposé (OCaml puis C) 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.

- (nouveau) Le barême a été retiré car il n'est pas donné dans les TPs concours.
- **Important**: Il semble que vous ayez quelques difficultés à bien rentrer dans un sujet (c'est normal en sup), exceptionellement ce sujet possède deux versions de la première partie: une formelle et une intuitioniste. Je vous demande lire d'abord la formelle et de comparer votre compréhension à l'intuition qui est donnée dans la v2J'. Si vous avez le moindre souci de compréhension (différence entre votre compréhension et l'intuition donnés) n'hésitez pas à me demander pourquoi votre compréhension était erronée ou si au contraire elle était toute aussi bonne. La compréhension d'un sujet est l'étape la plus importante (en info comme en philo), profitez de cet exercice.

Introduction En 2007, Arnaud Megret publie la première version d'Akinator, le génie du web auquel vous avez probablement déjà joué. Ce sujet a pour but de l'implémenter en OCaml.

I - Pendant que ça charge

A traiter impérativement.

Question 1 à l'écrit

1. Comment compiler du OCaml en ligne de commande sur Linux ?
2. Ecrire un algorithme qui renvoie le maximum des nombres inférieurs à la racine dans un arbre binaire de recherche.
3. Définir un type `C` pour un arbre binaire (tout le reste du sujet sera en OCaml)
4. Complétez cette phrase: "Dans un arbre ??, on a $n_i = ?$ avec n_i le nombre de noeuds internes."
5. (Voulu par votre prof d'info) Quel est l'algorithme pour passer d'un arbre n-aire à un arbre binaire ?

II - Fonctionnement

II.1 - Formelle

Soit Σ le jeu de caractère ASCII. On note Σ^* l'ensemble des mots dessus. On considère un jeu de questions-réponses $\mathcal{Q} = \{(X_i, Y_i)/X_i \in \Sigma^* \wedge Y_i \in \mathcal{P}(\Sigma^*)\}_{i\in\mathbb{I}}$.

On considère un jeu de données réponses-individus (qui map les réponses à une personne de la base de donnée) $\mathcal{D} = \{(Y_j, N_j)/Y_j \in \mathcal{P}(\Sigma^*) \wedge N_j \in \Sigma^*\}_{j\in\mathbb{J}}$.

On veut construire un arbre n-aire tel que les noeuds correspondent à l'ensemble des réponses d'un $D \in \mathcal{P}(\mathcal{D})$ une partie de notre base de donnée qu'on aura classifié par les questions des hauteurs inférieures.

Un noeud de hauteur k a autant de fils que la question X_k a de réponses possibles.

On a une feuille quand on a hauteur $= |\mathbb{I}|$, et uniquement à cette condition **si vous coupez dès qu'il n'y a plus qu'un seul élément vous ne pourrez plus modifier votre arbre car quand vous voudrez ajouter à cette feuille vous allez vouloir recréer un noeud, ce qui ne sera pas possible sans les réponses suivantes.**

Definition 1 Arbre n-aire

Un arbre n-aire est une généralisation des arbres binaires (vous pouvez remarquer que binaire = 2-aire dans la définition) telle que:

- Chaque noeud a une liste de fils (au plus n) et peut-être étiqueté (dans ce sujet il le sera)
- Les noeuds sans fils sont appelés feuilles.
- La hauteur se définie comme pour un arbre binaire modulo la taille de l'ensemble dont on prend le max.

II.2 - Intuition

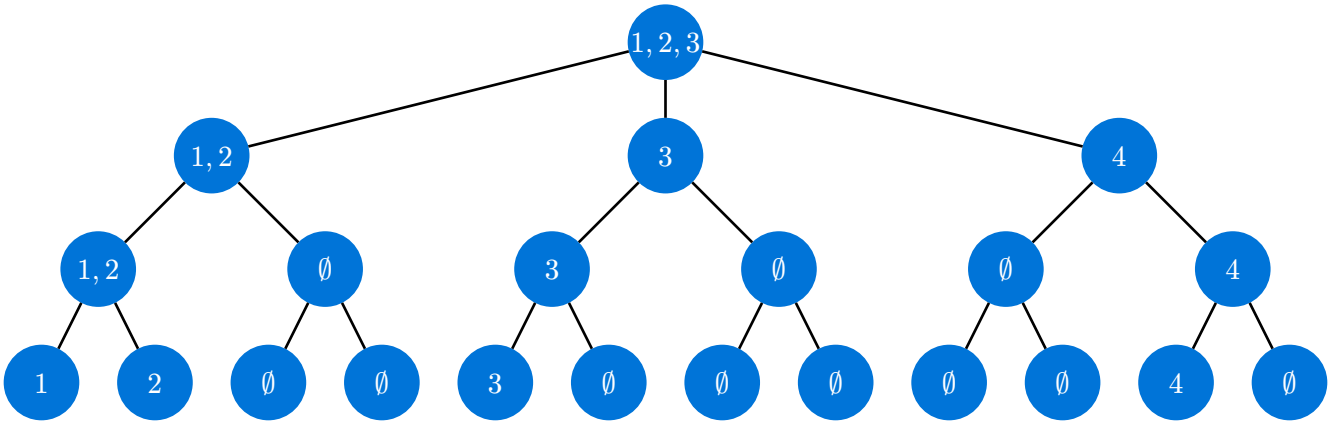
Pour fonctionner, notre **Arbrinator** (jeu de mots douteux) utilisera des arbres n-aires (la définition est donnée plus haut). L'idée est que chaque noeud représente une question (par exemple: Votre personnage est-il réel ?) et que ses fils représentent une réponse possible. Ainsi, si 33% de notre base de donnée est réel, que 33% ne l'est pas et que 33% est les deux à la fois et que c'est la deuxième question posée, tous les noeuds de la deuxième hauteur classifient les groupes déjà créés par la première question sur leur caractère réel ou non.

Voici un exemple concret: On a 3 personnes dans notre base de données, 2 chiens et 1 chat, la première question est "Êtes-vous: un chien ? un chat ? un ornythorinque ?", la deuxième est "Êtes-vous droitier ou gaucher ?", la troisième question est "Vous avez plus de 3 ans ?".

On peut apprendre de cet arbre que:

- On a 2 chiens, 1 chat et 1 ornythorinque dans la base de donnée
- Les deux chiens sont droitier ou gaucher (dans le sujet on devrait avoir des feuilles vides qui partent du premier 1, 2 mais pour des raisons techniques je ne les mets pas, on ne peut donc pas savoir si ils sont droitiers ou gauchers, on sait juste qu'ils ont la même patte principale)
- On sait que 1 est un chien de moins de 3 ans et que 2 est un chien de plus de 3 ans

Il est possible d'obtenir plus d'informations sur l'ornythorinque et sur le chat.



Question 2 à l'écrit

Quelle propriété immédiate ces arbres possèdent ? (De quel type ils sont)

III - Autour des arbres n-aires

III.1 - Déclaration des types

Avant de coder, il faut définir le type arbre, une question subsiste: faut-il utiliser une liste ou une array pour encoder un arbre n-aire ?

Question 3 à l'écrit

Expliquer quelle implémentantion entre les listes et les arrays est optimale en espace et laquelle est optimale en temps.

Dans la suite du sujet on va choisir l'implémentation par `array`.

On va utiliser ce type:

- Noeud: `etiquette_t` et une array de fils de taille n, les fils non remplies sont marqués par un type `Vide`.
- Feuille: `etiquette_t`
- Vide

Question 4 code

Le type `etiquette_t` est un alias pour le type `string list` (ce sera la partie de notre BDD qui est encore concernée par la suite de réponse). Définir un type `n_arbre` OCaml implémentant le type pour l'array.

III.2 - Algorithmes de base (prise en main)

Question 5 code

Ecrire une fonction `hauteur: n_arbre -> int` ainsi qu'une fonction `nombre_noeuds: n_arbre -> int`

Question 6 à l'écrit

Quelle est la complexité en temps et en espace de votre fonction `hauteur` ?

Encadrez la hauteur d'un arbre d'arité n en fonction de sa hauteur.

IV - Application à Arbrinator

IV.1 - Création de l'arbre

Les questions sont données dans le fichier `question_reponse.ml`. Vous devez ouvrir le fichier et copier son contenu dans votre code.

Pour coder l'algorithme de décision on a besoin d'un ordre sur les réponses des questions car on préfère manipuler `1,2,3` plutôt que `Droitier,Gaucher,Ambidextre`, en effet "aller au fils `Ambidextre`" est moins facile à coder que "aller au fils `3`", on subdivise notre problème en plusieurs petits problèmes simple à résoudre (programmer c'est faire du diviser pour régner en permanence).

Après avoir pris connaissance des types définies dans le fichier `question_reponse.ml`:

Question 7 code

Ecrire une fonction `numero: string -> (string list) -> int` qui prend en entrée une réponse, une liste de réponses, et renvoie le numéro de la réponse dans la lsite.

IV.1.a - Ajout d'un noeud)

Question 8 code

Ecrire une fonction `ajoute: n_arbre -> questions_reponses_t -> donnee_t -> n_arbre` qui ajoute une personne à la base de donnée.

Question 9 à l'écrit

Complexité en temps ?

IV.1.b - Suppression d'un noeud)

Question 10 code

Ecrire une fonction `suppression: n_arbre -> donnee_t -> n_arbre` qui retire l'individu `donnee_t` de l'arbre.

Question 11 à l'écrit

Complexité en temps ?

IV.1.c - Touche finale)

Désormais vous avez toutes les clés en mains pour faire votre arbre:

Question 12 code

Ecrire une fonction `creer_arbre: questions_reponses_t -> donnees_t -> n_arbre` qui implémente la construction donnée partie 1.

Question 13 à l'écrit

Complexité en temps?

IV.1.d - (Optionelle) Ajouter la possibilité de ne pas savoir)

Akinator vous laisse la possibilité de dire "je ne sais pas", notre code non car on a autant de branchement que de réponses possibles. On définit l'opération de `skip` comme suit:

Sauter la question numéro i avec les données D revient à créer l'arbre pour la question $i+1$ pour les données D

Question 14 code

Ecrire une fonction `creer_arbre_amelioree: questions_reponses_t -> donnees_t -> n_arbre` qui ajoute une branche à chaque noeud pour supporter le "je ne sais pas".

Question 15 à l'écrit

Quel est l'impact de cet ajout sur la taille de l'arbre ?

IV.2 - Coder une partie

Vous avez toutes les clés en mains pour jouer à Arbrinator contre votre PC !

Une partie se déroule comme suit:

- Tant que le joueur n'atteint pas une feuille on continue de jouer en posant la question suivante
- Si la feuille est vide, on demande au joueur à qui il pensait et on ajoute à l'arbre pour la partie suivante
- Si la feuille contient un seul élément on renvoie la réponse
- Si la feuille a plusieurs éléments on dit qu'on ne peut pas décider et on s'excuse **sans** demander à qui on pensait (c'est une extension trop longue car il faut poser une nouvelle question pour distinguer)

Question 16 code

Ecrire une fonction `joue: n_arbre -> questions_reponses_t -> n_arbre` qui effectue une partie et renvoie l'arbre modifiée après la partie.

clement.rouvroy@ens.psl.eu

V - Déjà fini ?

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 Borne inférieure

Montrez qu'un tri par comparaison a besoin d'au moins $n\log(n)$ opérations par un raisonnement sur les arbres.