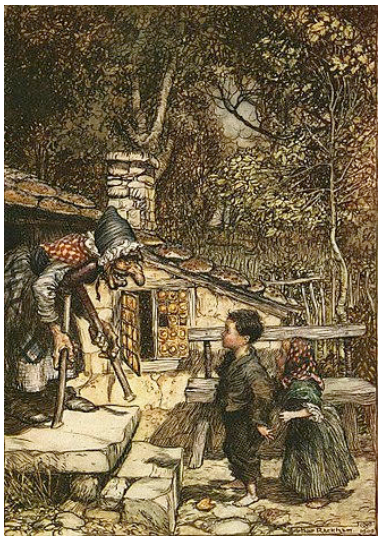
clement.rouvroy@ens.psl.eu

Hänsel und Gretel



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.

Introduction du sujet

Dans Hansel et Gretel, deux enfants se baladent en forêt et utilisent des graines pour marquer leur chemin, ce TP propose une modélisation informatique de la situation. Ce TP a deux parties:
1.Tout d'abord vous allez voir comment utiliser un marquage pour trouver tous les chemins les plus intéressants à suivre pour retrouver les enfants
2. Vous réfléchirez ensuite à comment optimiser le placement des graines pour assurer que l'on puisse vous retrouver.

clement.rouvroy@ens.psl.eu

I Question de cours

Question 1 à l'écrit

1. Définir les files de priorité.
2. Définir les tas binaires. Que peut-on dire de leur hauteur en fonction de n le nombre de noeuds ?
3. Proposer (en moins de 5 lignes et en français) une implémentation des files de priorité par tas binaires.

VRAI OU FAUX

a. Un type OCaml peut commencer par une majuscule.

b. Un graphe peut-être vu comme un arbre en toute généralité.

c. Un arbre peut-être vu comme un graphe en toute généralité.

d. Le parcours en largeur est plus optimisé (en complexité) que le parcours en profondeur.

e. Pour un arbre binaire on a toujours l'encadrement $h + 1 \leq n \leq 2^h - 1$.

II Sortir d'une forêt

On va vouloir modéliser les chemins possibles dans une forêt et des sommets marqués. Pour cela, on va utiliser un graphe orienté étiquetté dont les étiquettes sont des entiers. L'entier d'un graphe représente sa distance - noté d - au sommet marqué le plus proche, on prend la convention que pour tout sommet $v \in \mathcal{V}$:

$$\begin{aligned} d(v) = 0 &\Rightarrow v \quad \text{est marqué.} \\ d(v) = -1 &\Rightarrow v \quad \text{attend d'être calculé.} \\ d(v) > 0 &\Rightarrow v \quad \text{est à distance } \ d(v) \ \text{ du sommet marqué le plus proche.} \end{aligned}$$

Les sommets représentent des intersections et ils sont reliés aux autres intersections qu'ils peuvent atteindre. Le graphe est orienté pour éviter de revenir en arrière.

L'algorithme de sortie va consister à effectuer deux actions:

1. Générer tous les $d(v)$
2. Partir du sommet de départ et toujours prendre le voisin qui a le $d(v)$ le plus petit, si il y en a plusieurs on s'autorise de les parcourir 1 à 1.

Dans la suite on se donne les types (que vous prendrez soin de définir en OCaml) `etiquette: int` et `graphe: (etiquette array),(bool array) array`. Le graphe est donc encodé par matrice d'adjacence.

II.1 Génération des $d(v)$

Question 2 code

Ecrire une fonction `reset: graphe -> unit` qui marque tous les sommets comme attendant d'être calculés.

Question 3 code

Ecrire une fonction `marque: graphe -> int -> unit` qui marque le sommet passé en argument dans le graphe.

Question 4 code

En adaptant un parcours que vous connaissez (les deux marchent) et en utilisant la fonction `min` d'OCaml, proposer une fonction `genere: graphe -> unit` qui génère les $d(v)$.

Question 5 à l'écrit

Pourquoi a-t-on pu utiliser des fonctions qui renvoient toujours `unit` et quand même mettre à jour les graphes?

II.2 Parcours

On suppose que la sortie est toujours le sommet n et l'entrée le sommet 1.

Question 6 code

Ecrire une fonction `sommets_possibles: graphe -> int list` qui renvoie la liste des sommets atteignables en suivant toujours les sommets du voisinage qui ont le $d(v)$ le plus petit.

Définition 2.1 marquage parfait

Un marquage est dit parfait si le seul sommet accessible en respectant l'algorithme de parcours est le sommet de sortie.

Question 7 code

Ecrire une fonction `est_parfait: graphe -> bool` qui dit si un marquage donné par son graphe est parfait ou non.

III Marquer une forêt

Dans cette partie on se place dans le rôle des enfants, on dispose de k caillloux et on veut marquer les sommets de manière optimale, c'est-à-dire de manière à minimiser le degré d'erreur du marquage.

Définition 3.1 degré d'erreur du marquage

Si le marquage ne permet pas de tomber sur le sommet de sortie son erreur est de $+\infty$. Sinon, son erreur est définie comme le nombre de sommets accessibles par l'algorithme de parcours - 1 (le -1 est pour avoir un degré d'erreur 0 pour un graphe qui a uniquement la sortie d'accessible).

Question 8 à l'écrit

Que dire si $k \geq n$?

Question 9 à l'écrit

Combien de possibilités de marquages y a-t-il étant donné un graphe ?

C'est beaucoup, par conséquent on aimerait bien ne pas tout tester. On propose l'algorithme simple suivant:

1. On calcule $\rho = \left\lfloor \frac{n}{k} \right\rfloor$
2. On marque le chemin parcouru toutes les k cases, si on arrive à une impasse on revient en arrière, en récupérant le marquage si on en avait mis un.

Question 10 code

Implémenter l'algoirthme `marque_k: graphe -> int -> unit` qui modifie le graphe pour le marquer avec k cailloux.

Question 11 à l'écrit

Implémenter des tests simples (traitant un maximum de cas) & tester votre code.

Justifier les cas que vous traitez (montrez-moi que vous couvrez toutes les possibilités).