Projet individuel d'algorithmique-programmation IAP1 : groupe 4.1

octobre 2010

1 Informations générales

1.1 Travail à rendre

Le projet est à réaliser en OCaml **individuellement**. Il sera accompagné d'un **dossier** contenant impérativement la description des choix faits, la description des types et des fonctions. Pour chaque fonction, on donnera impérativement l'interface complète (dans le code en commentaire et dans le rapport pour les fonctions présentées).

Même si le sujet est décomposé en questions, il est possible qu'une question se résolve par l'écriture d'une ou plusieurs fonctions intermédiaires. Celles-ci doivent comporter une interface également.

Le dossier fournira également des cas de tests accompagnés des résultats attendus et retournés.

Sur le site du cours figure un petit document sur ce que l'on attend dans un rapport. Consultez-le !

1.2 Calendrier et procédure de remise

Le projet (dossier + copie du listing de code) est à rendre au secrétariat **le 22 novembre à 16h au plus tard**. Le numéro du groupe, ainsi que le nom du chargé de TD, devront figurer en gros, et en rouge sur la page de garde.

Le fichier .ml contenant votre code devra être déposé électroniquement au plus tard **le 22 novembre à minuit**.

Les soutenances seront organisées dés la semaine suivante. Il vous faudra consulter les panneaux d'affichage et votre courrier electronique pour obtenir l'ordre de passage.

Enfin n'attendez pas pour vous mettre au travail ! Un projet se travaille dès la remise du sujet afin d'avoir le temps de laisser murir la solution et de poser des questions au client (dans votre cas, votre chargé de TP).

1.3 Procédure de dépot

Pour déposer le code du projet, consultez les informations sur les machines de l'école, tout figure.

Ne vous inquiétez pas, l'interface fabrique un nom de projet déposé à partir de votre login. Votre projet ne sera pas confondu avec celui d'un autre.

Indiquez quand même en commentaire dans votre fichier de code Ocaml votre nom et votre groupe. Ce sera plus facile pour le correcteur.

Vous pouvez déposer successivement plusieurs versions, le dernier dépot écrase le précédent, seul le dernier dépot est pris en compte

Enfin tout cela peut se faire de l'extérieur.

Le code à déposer est un fichier.ml commenté **avec les interfaces des fonctions**. **Le fichier doit pouvoir être compilé (sans aucune intervention humaine - lecture du buffer complet) sur les machines Yaka de l'école**, n'oubliez pas de vérifier que tout fonctionne sur les machines de l'école avant de le dép oser - si vous l'avez développé sous un autre système d'exploitation.

2 Énoncé du projet : Hashiwo kakero

Vous incarnez l'aîné de la famille royale d'une petite île sans importance au milieu d'un grand archipel. En grandissant vous avez vu votre père, le roi, rentrer de conquête après conquête, étendant son influence sur toutes les îles voisines. Par sa force de caractère et sa puissance militaire, la fin de son règne a été paisible. Maintenant, à sa mort, vous devez assurer la continuité du gouvernement, une tache qui semble se compliquer lorsque vos conseillers vous apportent la nouvelle que certains de vos sujets sèment la révolte, revendiquent l'indépendance de leur île et menacent de diviser le royaume.

Pour promouvoir l'unité et le progrès économique et technologique, vous proposez la construction d'un réseau de ponts permettant à vos sujets de circulez librement aux quatre coins du territoire. Cette nouvelle incite une grande joie, mais pour tenir votre promesse vous devrez prendre en compte les besoins très particuliers de chaque île et respecter certaines contraintes techniques. Votre grande sagesse sera-t-elle à même de résoudre ce problème ?

2.1 Présentation du jeu

Le *Hashiwo kakero* (dont la traduction est « construire des ponts ») est un puzzle inventé et publié par le célèbre magazine japonais Nicoli, qui est aussi à l'origine notament du Sudoku et du Kakuro. Le but de ce projet est d'élaborer un programme capable de résoudre les puzzles de ce type.

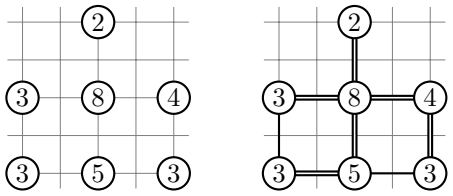
Chaque puzzle se présente sous la forme d'une grille d'une certaine taille et bien souvent de forme carrée. Sur certains croisements de cette grille, des ronds sont placés pour matérialiser les îles de votre royaume. L'importance de chaque île est notée à l'intereur du rond concerné.

Pour résoudre le puzzle vous devez former un réseau connexe (i.e. qui permet à un habitant de n'importe quelle île de rejoindre n'importe quelle autre île) en plaçant des ponts. Ceux-ci doivent respecter quelques contraintes.

- Un pont doit partir d'une île et arriver à une autre sans faire de virage.
- Chaque pont doit être horizontal ou vertical.
- Entre deux îles données, il peut y avoir deux, un ou aucun pont.
- Un pont ne peut croiser ni une île, ni un autre pont.

Chaque île doit être reliée au réseau par un nombre de ponts égal à son importance.

La solution de tout puzzle est unique, et vous pouvez considérer – car c'est le cas en général – que deux îles ne sont jamais placées à deux croisements consécutifs dans une direction horizontale ou verticale.



(a) le puzzle (b) sa solution

FIGURE 1 – Exemple de puzzle avec sa solution.

2.2 Méthodes de résolution

Il existe quelques astuces pour résoudre ce genre de puzzle. Il est à noter que ce sont les mêmes astuces qui sont utiles pour un humain, qui servent à constuire un algorithme de résolution par ordinateur.

La première astuce. Tout d'abord, il y a un grand nombre de situations qui obligent la construction de ponts dans certaines directions. Lorsqu'une île est dans le coin ou sur le bord de la carte, les possibilités sont limitées. Aussi, une île très importante, même au beau milieu de l'archipel nécessitera forcément au moins un pont dans chacune des quatre directions possibles.

Au fur et à mesure que vous construisez des ponts, les situations se débloquent pour les autres îles. Un pont peut bloquer l'accès entre deux îles et ainsi conduire à une situation dont la solution est évidente. Une bonne partie des puzzles, et même des puzzles entiers, peuvent être résolus en appliquant cette seule stratégie en cascade.

La deuxième astuce. Par contre, pour les puzzles plus difficiles, il faut se servir du fait que les îles doivent toutes être reliées entre elles. Ainsi, s'il ne reste plus qu'un pont à construire à partir d'une certaine île, mais il y a deux destinations possibles, vous pouvez en éliminer une si vous savez que cela formera un sous réseau qui ne pourra jamais être relié au reste.

2.3 Programmation

A priori, votre solveur consiste à appliquer successivement la première astuce donnée ci-dessus à chaque île, puis à recommencer s'il y a eu des changements du réseau mais que le puzzle n'a pas encore été résolu. Si la première astuce ne produit aucun changement, vous pouvez tenter la deuxième pour débloquer la situation. Vous pouvez aussi y ajouter vos propres techniques de résolution.

2.3.1 Types de données

Le puzzle doit être représenté par une liste de couples, dont la première composante est une coordonnée et la deuxième est l'importance de l'île placée à cette coordonnée.

```
type coordinate = int * int;;
type importance = int;;
type puzzle = (coordinate * importance) list;;
```

La solution sera une matrice (représentée par une liste de listes) de ce qui se trouve à chaque croisement de la grille. Chaque sous liste représente une ligne de la solution avec le croisement le plus à gauche en premier. Les sous listes doivent être ordonnées de haut en bas.

```
type bridge = {
        isVertical: bool;
        isDoubled: bool
};;
type cell =
        | Nothing
        | Island of importance
        | Bridge of bridge;;
type solution = cell list list;;
```

Avec cette représentation, un croisement qui comporte deux ponts horizontaux est représenté par la valeur `Bridge {isVertical= false; isDoubled= true}`.

En plus de ces deux types, vous en aurez besoin d'un troisième qui permet de rassembler où vous en êtes dans la résolution du puzzle.

Question 1. Définir le type `data` qui correspond à ce troisième type.

Indication. La question précédente est certes la première, mais ce n'est sans doute pas celle à laquelle il faut réfléchir en premier.

Question 2. Définir la fonction `newData: puzzle -> data` qui initialise la résolution du puzzle.

2.3.2 Solveur

Question 3. Définir la fonction `strategy1: data -> data` qui applique la première astuce à chaque île du puzzle.

Question 4. Définir la fonction `strategy2: data -> data` qui applique la deuxième astuce à l'île qui est la plus adaptée.

Remarque. En effet, en regardant les exemples, l'application de la deuxième astuce donne plus ou moins de réussite selon l'île choisie.

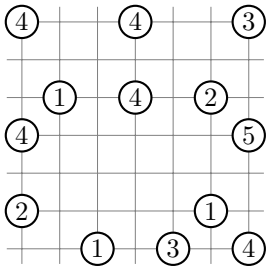
Question 5. A l'aide des fonctions précédentes et éventuellement d'autres fonctions qui représentent vos propres techniques de résolution, définir la fonction `solve: puzzle -> solution` qui résout un puzzle donné. Ne pas inclure les cas de tests de cette fonction dans l'interface.

Indication. Si vous n'avez pas réussi à implanter la fonction `strategy2` vous pouvez néanmoins écrire un solveur qui se contentera toujours d'appliquer la première astuce. Pour un grand nombre de puzzles, cela suffit, mais pas pour tous.

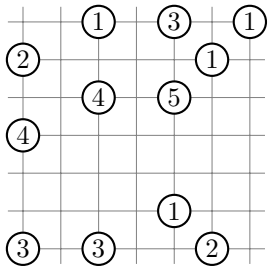
Question 6. Tester votre solveur sur les exemples donnés ci-après, des puzzles tirés d'autres sources ou construits par vous-même.

2.4 Exemples de puzzles

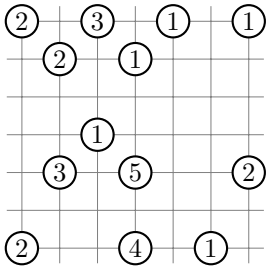
La difficulté de chaque puzzle est donnée par le nombre d'étoiles ($\star$) qui suivent son numéro.



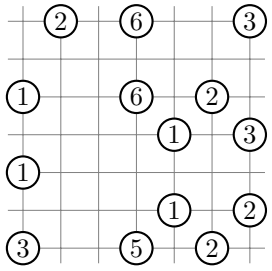
(a) puzzle 1 $\star$



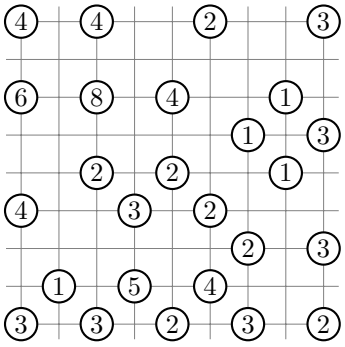
(b) puzzle 2 $\star$



(c) puzzle 3 $\star\star$



(d) puzzle 4 $\star\star\star$



(e) puzzle 5 $\star\star\star\star$