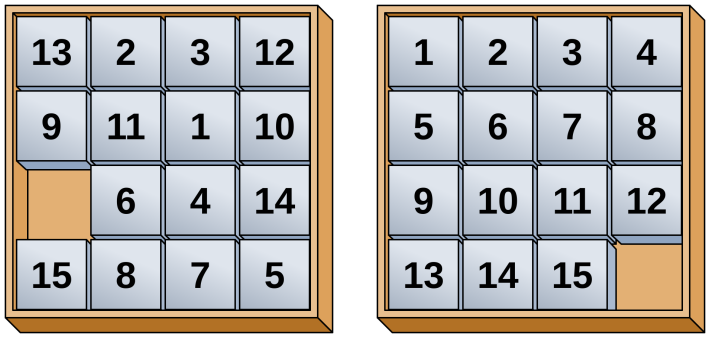
Jeu de taquin

Thèmes : suites récurrentes, matrices, graphes, parcours en largeur

Le *taquin* est un jeu se jouant seul dans lequel le joueur tente de replacer dans l'ordre 15 petites tuiles carrées numérotées de 1 à 15 qui peuvent glisser dans un cadre carré prévu pour 16 tuiles.



Taquin à résoudre Taquin résolu

Figure 1 Jeu de Taquin (sources: (1) et (2))

Dans ce sujet on s'intéressera à la résolution du **taquin à 8 tuiles** numérotées de 1 à 8 glissant dans un cadre de dimensions 3×3.

> Les programmes à écrire et les résultats demandés font intervenir une valeur `u0` fixée en début de sujet. On prendra la valeur `u0 = 42`

1 Mélange d'un taquin

Un taquin sera codé en `OCaml` par une matrice `t` de taille 3×3 contenant des entiers entre 0 et 8. L'absence de tuile sera noté par la valeur 0. La tuile de coordonnées $(i, j) = (0, 0)$ sera la tuile en haut à gauche. Une fonction `print_taquin : taquin -> int` est fournie pour vous permettre d'afficher joliment à l'écran un taquin. Cette partie se concentre sur la création d'un problème de taquin aléatoire, obtenu en mélangeant aléatoirement un taquin résolu.

Question 1

Écrire une fonction `taquin_résolu : unit -> taquin` qui retourne un nouveau taquin, dans la configuration du taquin résolu, comme représenté à dans la figure de droite ci-dessus, mais avec pour dimensions 3×3. Afficher le résultat à l'écran.

Question 2

Écrire une fonction `place_vide : taquin -> (int * int)` retournant le couple de coordonnées `(i, j)` de la position où il n'y a pas de tuile. Tester cette fonction sur le taquin résolu.

Dans une configuration donnée, il existe au plus 4 déplacements de tuiles possibles; on les notera `N` (nord), `E` (est), `S` (sud), `W` (ouest). **Cette direction est le sens de déplacement de la tuile qui se déplace**.

```
type direction = N | E | S | W;;
```

Question 3

Écrire une fonction `deplacement : taquin -> direction -> int` qui

- si le déplacement `direction` est possible, modifie `taquin` en conséquence puis retourne la valeur 1.
- si le déplacement n'est pas possible, retourne la valeur 0 sans changer `taquin`.

INDICATIONS

Commencer par utiliser la fonction `place_vide` pour déterminer la position de la case vide, puis étudier si le déplacement proposé est possible.

On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie par récurrence :

$$\begin{cases} u_0 & = \mathrm{u0} \\ u_{n+1} & = 1022 \times u_n \pmod{(2^{30}-3)} \end{cases}$$

Question 4

Construire un tableau `u` de longueur 100000 où chaque case `u.(i)` contient la valeur de u_i. Donner les valeurs de

a. u_{10},
b. u_{500},
c. u_{10000}.

On note $D(k,l)$ une liste de longueur l contenant des déplacements aléatoires. Cette liste est définie ainsi :

$$\forall i \in [\![0, l-1]\!], \quad D(k,l)[i] = \begin{cases} \texttt{N} \text{ si } u_{i+k} = 0 \pmod 4 \\ \texttt{E} \text{ si } u_{i+k} = 1 \pmod 4 \\ \texttt{S} \text{ si } u_{i+k} = 2 \pmod 4 \\ \texttt{W} \text{ si } u_{i+k} = 3 \pmod 4 \end{cases}$$

Question 5

Écrire une fonction `dep : int -> int -> direction list` prenant en arguments les entiers k et l et retournant cette liste de déplacements aléatoires. Donner les valeurs de

a. `D(0, 5)`,
b. `D(50, 5)`,
c. `D(140, 5)`.

Question 6

Écrire une fonction `deplacement_liste : taquin -> direction list -> int` prenant en entrée un taquin et une liste de déplacements et effectuant dans l'ordre les déplacements possibles en modifiant `taquin`. Cette fonction retournera le nombre de déplacements réellement effectués.
Donner le nombre de déplacements réellement effectués lorsqu'on part du taquin résolu et qu'on effectue les déplacements suivants :

a. `D(50, 1000)`,
b. `D(3500, 1000)`,
c. `D(7100, 1000)`.

INDICATIONS

Utiliser la fonction `deplacement`.

Soit k un entier naturel, on notera $T(k)$ le taquin obtenu en partant du taquin résolu et en appliquant la liste de déplacements $D(k, 1000)$.

Question 7

Écrire une fonction `taq : int -> taquin` retournant le taquin aléatoire $T(k)$.
a. Donner $T(700)$.
b. Quelle est la somme des valeurs sur la première ligne de $T(50)$?
c. Quelle est la position de la case vide dans $T(200)$?

2 Graphe du taquin

Les suites de déplacements possibles d'un taquin peuvent être modélisées à l'aide d'un **graphe orienté** dans lequel les sommets sont les configurations possibles du jeu et les arcs les déplacements possibles d'une configuration à une autre par déplacement d'une tuile. Dans une configuration donnée, le but du jeu est alors de trouver un chemin dans le graphe menant de cette configuration à la configuration de taquin résolu. Si possible, on cherchera à obtenir un chemin le plus court possible, c'est-à-dire nécessitant un minimum de déplacements de tuiles.

Afin d'obtenir une numérotation des sommets du graphe, on va associer à chaque taquin un unique entier appelé *configuration* du taquin et donné par la formule suivante :

$$c(t) = \sum_{i=0}^{2} \sum_{j=0}^{2} t[i][j] \times 10^{3i+j}$$

Question 8

Écrire la fonction `configuration : taquin -> int` retournant la configuration d'un taquin donné en entrée.
Donner la configuration du :
a. taquin résolu
b. taquin $T(50)$
c. taquin $T(200)$

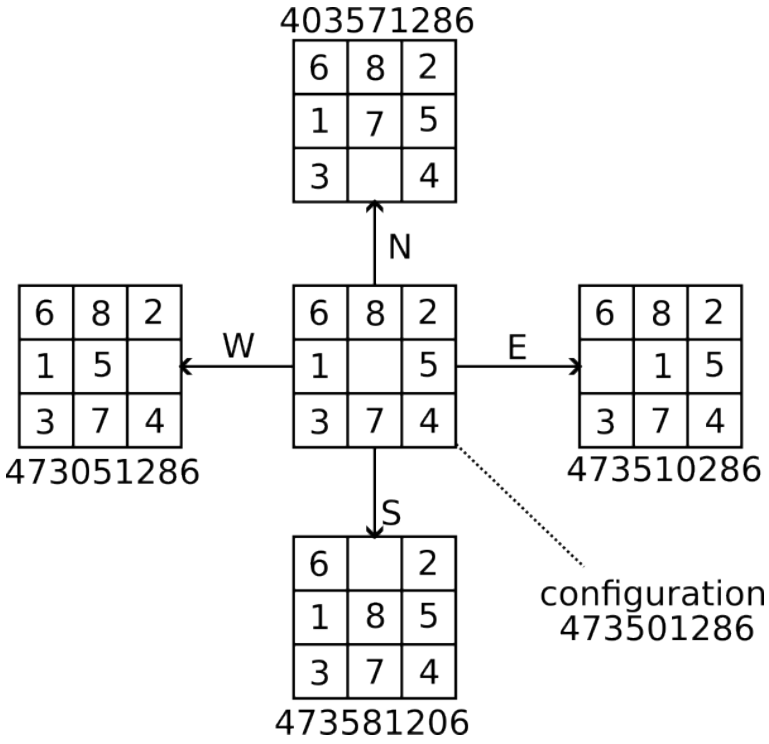
Réciproquement, on pourra toujours retrouver un taquin à l'aide de sa configuration en utilisant la fonction réciproque suivante :

```
(* Retourne l'unique taquin de configuration donnée en entrée *)
let decode c =
    let n = ref c in
    let t = Array.make_matrix 3 3 0 in
    for i = 0 to 2 do for j = 0 to 2 do
        t.(i).(j) <- n;
        n := !n / 10
    done done;
    t
;;
```

Le graphe du jeu de taquin est donc formellement un graphe orienté $G = (S, A)$ où

- l'ensemble des sommets S est l'ensemble des configurations possibles du taquin
- l'ensemble des arcs A est l'ensemble des couples (c_1, c_2) où c_1 est une configuration et c_2 une configuration obtenue par un déplacement possible (N, E, S, ou W) depuis c_1.

Un exemple de sommet et ses voisins sont donnés sur la figure suivante :



Question 9

Écrire une fonction `voisins : int -> int list` prenant en entrée une configuration c et retournant la liste des *voisins sortants* de c, c'est-à-dire la liste des configurations accessibles depuis c en 1 déplacement `N, E, S` ou `W` de tuile. **On retournera impérativement les voisins dans cet ordre**
Donner la liste des voisins du sommet
a. correspondant au taquin résolu;
b. correspondant au taquin $T(50)$.
c. Quel est le degré du sommet de configuration 789120345 ?

INDICATIONS

Utiliser la fonction `déplacement` pour tester si les déplacements sont possibles.

On sait que le parcours en largeur permet d'explorer les sommets d'un graphe par ordre de distance en nombre d'arcs depuis le sommet de départ x_0. Autrement dit, on peut se servir du parcours en largeur pour déterminer le chemin le plus rapide pour atteindre la configuration résolue depuis une configuration donnée.

Le parcours en largeur nécessite l'utilisation d'une **file**. On utilisera pour cela le module `Queue` de OCaml. La file servira à stocker les sommets découverts mais non encore explorés pendant le parcours. Pour se rappeler quels sont les sommets déjà visités on utilisera un **dictionnaire** dont les **clefs** sont les sommets déjà visités et la **valeur associée** est le sommet qui a permis sa découverte, c'est-à-dire le sommet qui le précède dans le parcours. On stoppe le parcours lorsqu'on explore le sommet correspondant au taquin résolu.

On utilisera donc l'algorithme de parcours en largeur suivant :

Entrées : une configuration de départ c_0
file ← nouvelle_file();
inserer(file, c_0);
précédent ← {};
Tant que file non vide **Faire**
  x ← extraire(file);
  Pour chaque $y \in$ voisins(x) **Faire**
    Si $y \notin$ précédent **Alors**
      inserer(file, y);
      précédent[y] ← x;
    Fin Si
  Fin Pour
  Si x est la configuration résolue **Alors**
    quitter la boucle Tant que
  Fin Si
Fin Tant que
Sorties : précédent

Question 10

Écrire une fonction `parcours_largeur : int -> (int, int) Hashtbl.t` retournant le dictionnaire `précédent` qui contient l'ensemble des sommets découverts pendant le parcours, associés à leur sommet précédent dans le parcours.

a. Combien de sommets on été explorés depuis le taquin initial $T(50)$?
b. Combien de sommets on été explorés depuis le taquin initial $T(120)$?
c. Quel sommet précède la configuration résolue dans l'exploration depuis $T(120)$?

Question 11

Écrire une fonction `reconstruction_chemin : int -> (int, int) Hashtbl.t -> int list` qui retourne un chemin optimal menant de c_0 à la configuration de taquin résolue, c'est-à-dire qu'elle retourne la liste des configurations [$c0$, ..., configuration résolu].

a. Afficher à l'écran les étapes de la résolution de $T(50)$ puis donner les 5 premiers déplacements à effectuer.
b. Afficher à l'écran les étapes de la résolution de $T(120)$ puis donner les 5 premiers déplacements à effectuer.
c. Combien faut-il de déplacements au minimum pour résoudre le taquin $T(100)$?
d. Combien faut-il de déplacements au minimum pour résoudre le taquin $T(200)$?

INDICATIONS

Se servir du dictionnaire `préc` pour reconstruire le chemin en partant de la fin, jusqu'à atteindre `c0`.

3 Sources

1. Par Booyabazooka — http://en.wikipedia.org/wiki/Image:15-puzzle.svg, Domaine public, https://commons.wikimedia.org/w/index.php?curid=1059593
2. Par Theon — own work, modified from the original file Image:15-puzzle.svg, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=3759824