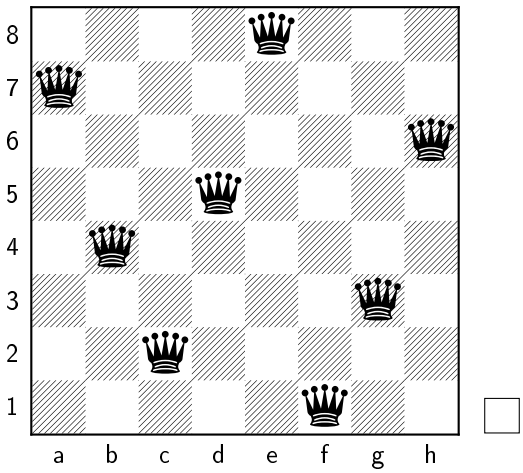
TD 1

Backtracking

1 Le problème des n dames

Le problème des n dames consiste à placer n dames sur un échiquier de taille $n \times n$ de telle sorte qu'aucune ne soit en prise : il ne faut donc pas plus d'une reine par ligne, par colonne et par diagonale.

Exemple avec un échiquier traditionnel :



1. Une petite découverte musical de la méthode pour $n = 4$: `https://youtu.be/R8bM6pxlrLY`
2. Proposer une solution pour $n = 4$.
3. Ecrire la fonction `possible : int -> int -> int list -> bool` qui prend en argument deux entiers représentant les coordonnées d'une case de l'échiquier, une liste d'entiers représentant la position des reines, et renvoie le booléen correspondant à la liberté de la case.
4. En utilisant la méthode du backtracking, écrire la fonction `recherche_sol : int -> int -> int list -> int list` qui prend en argument le nombre de lignes, la ligne courante et la liste des dames déjà placées et renvoie la liste des dames, solution du problème. L'idée est de construire cette fonction de façon récursive, dont l'arrêt se fait si la ligne courante dépasse le nombre de lignes. La fonction fera appel à une fonction auxiliaire récursive qui aura pour but de remplir la liste des reines.

2 Satisfiabilité

Pour éviter de faire la table de vérité d'une proposition logique, on peut utiliser la méthode du backtracking pour résoudre le problème SAT d'une proposition.

1. On considère, sous sa forme normale conjonctive, la proposition logique suivante :

$$F(a,b,c,d) = (b \vee c) \wedge (\neg a \vee \neg b \vee c) \wedge (\neg c \vee \neg d) \wedge (\neg a \vee \neg c \vee d) \wedge (a \vee d)$$

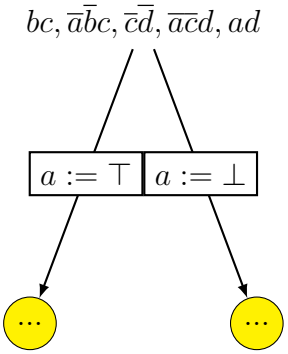
Pour simplifier la notation, nous choisirons de noter la formule de la façon suivante :

$$F(a,b,c,d) = bc, \overline{a}\overline{b}c, \overline{c}\overline{d}, \overline{a}\overline{c}d, ad$$

(a) On suppose que a prend la valeur vrai ($\top$), déterminer la forme normale conjonctive de $F(\top, b, c, d)$.

(b) On suppose que a prend la valeur faux ($\bot$), déterminer la forme normale conjonctive de $F(\bot, b, c, d)$.

(c) Compléter l'arbre suivant :



2. Montrer, en utilisant l'arbre, l'insatisfiabilité de la formule suivante :

$$F_2(a,b,c) = abc, a\overline{b}, b\overline{c}, \overline{a}\overline{b}\overline{c}, \overline{a}c$$

3. Pour résoudre le problème en Ocaml, on définira une formule à l'aide de listes de listes.Une clause est donc représentée par une liste avec autant d'éléments que de variable. Pour chaque littéral positif, on associera la valeur 1, et -1 pour un littéral négatif. Si la variable n'apparait pas dans la clause, on associera la valeur 0.
Exemple : Pour la formule F, on aura alors :

```
let exemple_F
=[[0;1;1;0];[-1;-1;1;0];[0;0;-1;-1];[-1;0;-1;1];[1;0;0;1]];;
```

(a) Écrire la fonction `positive : int list list -> int list list` qui prend en argument une proposition et renvoie la proposition pour laquelle la première variable prend la valeur vraie. Exemple :

```
positive exemple_F;;
- : int list list = [[1; 1; 0]; [-1; 1; 0]; [0; -1; -1]; [0; -1; 1]]
```

(b)

(c) Écrire la fonction `negative : int list list -> int list list` qui prend en argument une proposition et renvoie la proposition pour laquelle la première variable prend la valeur faux. Exemple :

```
negative exemple_F;;
- : int list list = [[1; 1; 0]; [0; -1; -1]; [0; 0; 1]]
```

(d) Écrire la fonction `solution : int list list -> bool` qui prend une liste de listes, représentant un état terminal dans l'arbre, et renvoie le booléen.

- Soit la dernière listes est vide, donc toutes les clauses ont été éliminées. La proposition est donc vraie.
- Soit la liste contient les clauses constituées d'un seul littéral. Il faut alors qu'il soit toujours du même signe. (si les clauses contiennent que des zero, alors la proposition est fausse...)

(e) En déduire la fonction `recherche_sat : int list list -> bool` qui utilise la méthode du backtracking pour résoudre le problème SAT.

3 Résolution d'un sudoku

On considère la grille de sudoku suivante :

2	5			3		9		1
	1				4			
4		7				2		8
		5	2					
				9	8	1		
	4				3			
			3	6			7	2
	7							3
9		3				6		4

On utilisera les matrices, la valeur 0 remplaçant le case vide :

```
let grille_exemple =
[|[| 2;5;0;0;3;0;9;0;1|];
[| 0;1;0;0;0;4;0;0;0|];
[| 4;0;7;0;0;0;2;0;8|];
[| 0;0;5;2;0;0;0;0;0|];
[| 0;0;0;0;9;8;1;0;0|];
[| 0;4;0;0;0;3;0;0;0|];
[| 0;0;0;3;6;0;0;7;2|];
[| 0;7;0;0;0;0;0;0;3|];
[| 9;0;3;0;0;0;6;0;4|]|];;
```

1. Ecrire les fonctions suivantes, utiles à la résolution :
   - Une fonction d'affichage : `affiche : int array array -> unit`
   - Une fonction vérifiant qu'une valeur est absente d'une ligne donnée : `absent_ligne : 'a -> int -> 'a array array -> bool`
   - Une fonction vérifiant qu'une valeur est absente d'une colonne donnée : `absent_colonne : 'a -> int -> 'a array array -> bool`
   - Une fonction vérifiant qu'une valeur est absente d'un bloc donné par la case étudiée : `absent_block : 'a -> int * int -> 'a array array -> bool`
2. En utilisant la méthode du backtracking, écrire la fonction `est_valide : int -> int array array -> bool` qui renvoie un booléen indiquant si la grille peut être complétée. L'idée est de travailler récursivement sur la position courante (entre 0 et 80).Lors de l'inspection de la case à la position p deux cas peuvent se produire :

- la case est occupée, dans ce cas, l'algorithme passe à la case $p+1$.
- La case est libre. Dans ce cas, il cherche une valeur v qui n'est pas sur la ligne, la colonne ni le bloc de la case p. L'algorithme tente de résoudre le sudoku en mettant v dans la case p puis passe à la case $p+1$ (il fait donc un pas en avant). Si la valeur v ne permet pas de remplir la grille, l'algorithme libère la case p (il fait donc un pas en arrière) et tente une autre valeur si il en existe une. Si aucune des valeurs proposée pour remplir la case p ne conduit à une solution, l'algorithme renvoie faux.