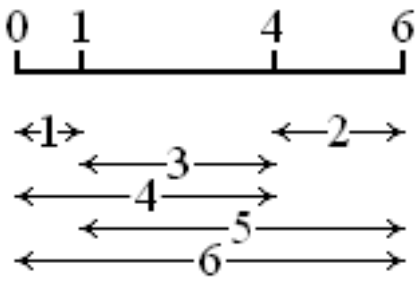
Programmation par contraintes en Oz

Exercice 1
Une règle de Golomb est une règle munie de marques à des positions entières, telle que deux paires de marques ne soient jamais à la même distance ; en d'autres termes, chaque couple de marques mesure une longueur différente des autres. Par définition, l'ordre d'une règle de Golomb est le nombre de marques qu'elle porte ; la longueur d'une règle de Golomb est la plus grande distance entre deux de ses marques. Voici par exemple une règle de Golomb d'ordre 4 et de longueur 6 :



Il n'est pas nécessaire qu'une règle de Golomb permette de mesurer toutes les distances entières entre 0 et la longueur de la règle mais si c'est le cas, on dit qu'il s'agit d'une règle de Golomb parfaite. La règle de l'exemple ci-dessus est parfaite.

1. Écrire une fonction Oz qui prend en argument deux entiers positifs N et L, et qui envoie une règle de Golomb d'ordre N et de longueur L. Est-ce qu'il y a des symétries qu'on peut exploiter ?

   Par exemple, lancée sur les arguments 4 et 6 la fonction pourrait renvoyer le résultat `[0 1 4 6]`.

   Indication : créer une variable à domaine fini pour toute marque à placer sur la règle (dont la valeur indique la position de cette marque sur la règle), et aussi une variable à domaine fini pour chaque paire de deux variables différentes (dont la valeur indique la distance entre ces deux marques).

2. Modifier votre solution pour chercher seulement des règles de Golomb parfaites. Dans ce cas votre fonction prendra seulement l'ordre N en argument car la longueur de la règle peut être calculée.

3. Modifier votre programme pour chercher des solutions optimales : votre fonction prendra maintenant en argument seulement l'ordre N, et envoiera une règle d'ordre N (pas nécessairement parfaite) et de longueur minimale.

Exercice 2 (Le retour de la colonie des vacances)
Dans cet exercice vous allez implémenter une variante plus avancée de l'exercice de la colonie des vacances de la feuille 2. Le but de cet exercice est de mettre en œuvre plusieurs techniques de la programmation par contraintes vues depuis :

- la génération d'un script à partir des données ;
- les combinaison booléennes de contraintes ;
- les contraintes souples.

Écrire une fonction `Vacances` qui prend les paramètres suivants :

1. le nombre de tentes disponibles ;
2. le nombre de places par tente ;
3. la liste des noms des vacanciers ;
4. la liste de leur contraintes et préférences.

Le liste des contraintes et préférences contient des éléments d'une de ces formes :

- `i(P N)` où `P` est le nom d'un vacancier et `N` un numéro de tente, indiquant une contrainte dure que `P` demande dormir dans la tente `N` ;
- `o(P N1 N2)` où `P` est le nom d'un vacancier et `N1` et `N2` des numéros de tentes, indiquant une contrainte dure que `P` demande dormir dans une des deux tentes indiquées ;
- `p(P1 P2)` où `P1` et `P2` sont des noms de vacanciers, indiquant une contrainte souple que ces deux personnes souhaitent dormir sous la même tente.

N'oubliez pas de prendre en compte le nombre de places disponible par tente.

Le script envoyé par la fonction doit permettre de trouver une solution qui respecte toutes les contraintes dures et un maximum des contraintes souples.

Par exemple, une solution possible de

```
{Browse {SearchOne {Vacances 3 2 [a b c d e f]
                                  [i(a 2) i(b 1) i(c 2) o(d 1 2) o(e 2 3)
                                   p(a b) p(c d) p(e f)]
                    }}}
```

est

```
[sol(a:2 b:1 c:2 d:1 e:3 f:3)]
```

Indications :

- Commencer avec un seul type de contrainte (par exemple `i`), et si cela fonctionne ajouter les autres ;
- Il y a une fonction `List.filter`, similaire à la fonction éponyme de OCaml, sauf qu'elle prend la liste en premier argument et le prédicat en second.