1 Drapeau de Dijkstra

1. On dispose de N boules noires et blanches alignées dans un ordre quelconque. On souhaite regrouper les boules de même couleur par échanges successifs. Il faut ne tester qu'une fois la couleur d'une boule.
   (a) Comment représenter les données du problème ?

On utilise un tableau de taille N dont les valeurs appartiennent à {Noir, Blanc}. Idée de l'algorithme : deux indices i_{inf} et i_{sup} indiquent la plage de "cases indéterminées". Si i_{inf} correspond à une case noire, i_{inf} est incrémenté : la plage des cases noires s'étend. Sinon, on échange les contenus des cases i_{inf} et i_{sup} et i_{sup} est décrémenté : la plage des cases blanches s'étend.

   (b) Donner un algorithme permettant de le résoudre.

Invariant : $\forall i, 1 \leq i < i_{inf}, t[i] = Noir$ et $\forall i, i_{sup} < i \leq N, t[i] = Blanc$.

```
Dijkstra_2couleurs(T) {
i_inf <- 1;
i_sup <- N;
Tant que (i_inf < i_sup) Faire {
   Si T[i_inf] = Noir Alors
      i_inf <- i_inf + 1;
   Sinon  {//T[i_inf] = Blanc
      Echanger(T, i_inf, i_sup)
      i_sup <- i_sup - 1}}
```

2. Même question avec trois couleurs (par exemple Noir, Gris et Blanc).

Trois couleurs (Noir, Gris, Blanc) Invariant : $\forall i, 1 \leq i \leq i_N, t[i] = Noir$, $\forall i, i_N < i < i_{inf}, t[i] = Gris$ et $\forall i, i_{sup} < i \leq N, t[i] = Blanc$.

```
Dijkstra_3couleurs (T) {
i_Noir <- 0;
i_inf <- 1;
i_sup <- N;
Tant que (i_inf <= i_sup) Faire {
   Selon que {
   T[i_inf] = Gris: i_inf <- i_inf + 1;

   T[i_inf] = Blanc: Echanger(T, i_inf, i_sup);
                      i_sup <- i_sup - 1;

   T[i_inf] = Noir: i_Noir <- i_Noir + 1;
                    Echanger(T, i_Noir, i_inf);
                    i_inf <- i_inf +1;
   }
}
```

2 Algorithmes à analyser

Que calculent les deux fonctions suivantes ? Quelle est leur complexité ?

1.
```
inconnue1 (a) {
     p <- 0;
     Tant que (a modulo 2 = 0) Faire {
        a <- a div 2;
        p <- p + 1;
     }
     Si (a = 1) Alors
        retourner p;
     Sinon
        retourner -1;
  }
```

Si $a = 2^p$, la fonction retourne p, sinon -1. Valeur de a entre 1 et 100 entrainant le plus de calculs : 2^6 (plus grande puissance de 2 inférieure à 100. Complexité de la fonction : $O(log_2 a)$

2.
```
inconnue2 (n, b) {
     p <- 1;
     r <- 0;
     Tant que (n > 0) Faire {
        r <- r + (n modulo 10) * p;
        n <- n div 10;
        p <- p * b;
     }
     retourner r;
  }
```

Calcul de la valeur décimale de n (donné en base b). Complexité : $O(log_{10}(n))$.