Corrigé du TP sur l'algorithme de Djikstra

I - Arithmétique dans $\mathbb{Z} \cup \{\infty\}$

Les procédures demandées ne posent aucun problème :

```
type entier = Infini | N of int ;;

let inferieur a b = match a,b with
     |Infini , _ -> false
     |_,Infini -> true
     |N(x),N(y) -> x<y;;

let add a b =  match a,b with
     |Infini , _ -> Infini
     |_,Infini -> Infini
     |N(x),N(y) -> N(x+y);;

let match_trois ordre a b c =
   if ordre b a then
      if ordre b c then
         2
      else
         3
   else
      if ordre c a then
         3
      else
         1;;
```

II - Une structure de file d'attente

Il suffit de traduire les méthodes vues en cours :

```
let echange tas i j=
   let s1 = tas.t.(i) and s2 = tas.t.(j) in
      tas.dico.(s1) <- j;
      tas.dico.(s2) <- i;
      tas.t.(i) <- s2;
      tas.t.(j) <- s1;;

(* les éléments du tas sont numérotés de 0 à k-1, donc le fils gauche du noeud i a
     pour numéro 2i+1 et son fils droits 2i+2 *)

let rec entasser tas i= match i with   (* on entasse depuis le noeud i *)
      | _ when 2*i+1 >= tas.k -> ()
      | _ when 2*i+1 = tas.k-1 -> if tas.ordre tas.t.(2*i+1) tas.t.(i) then
          echange tas i (2*i+1)
      | _ -> match match_trois tas.ordre tas.t.(i)  tas.t.(2*i+1) tas.t.(2*i+2)
          with
                | 1 -> ()
                | 2 -> echange tas i (2*i+1); entasser tas (2*i+1)
                | _ -> echange tas i (2*i+2); entasser tas (2*i+2);;
```

```
let rec remonter tas i= match i with   (* on remonte depuis le noeud i *)
      | 0 -> ()
      | _ -> let p = (i-1)/2 in
            if tas.ordre tas.t.(i) tas.t.(p) then
               begin
               echange tas i p;
               remonter tas p
               end;;

let extrait_min tas =
   match tas.k with
      | 0 -> failwith "le␣tas␣est␣vide"
      | k-> let s = tas.t.(0) in
               echange tas 0 (k-1);
               tas.k <- k-1;
               entasser tas 0;
               s;;

let faire_tas t =                (* on entasse de bas en haut, en commençant *)
   for i = (tas.k)/2-1 downto 0 do  (* par le premier noeud qui possède un fils *)
       entasser t i
   done;;
```

III - L'algorithme de Djikstra

Pas de problème non plus ... dès que l'on a bien compris les structures. g est défini par listes d'adjacente : $g.(i)$ est la liste des couples (j, p) où p est le poids (entier positif) de l'arc (i, j).

```
let initialiser g s =
   let n = Array.length g in
   let p = Array.make_matrix n n Infini in   (* p est la matrice des poids *)
   for i = 0 to n-1 do
      p.(i).(i) <- N 0;
      let l = ref g.(i) in‘
      while !l <> [] do
         let j,poids = List.hd !l in
         l := List.tl !l;
         p.(i).(j) <- N(poids)
      done;
   done;
      dico.(s)<- 0;
      t.(0) <- s;
      dico.(0) <- s;
      t.(s) <- 0;
      d, t, dico;;
```

```
let dijkstra g s =
   let n = Array.length g in
   let pi = Array.make n (-1) in
   let p, d, t, dico = initialiser g s in
   let ordre s t = inferieur d.(s) d.(t) in
   let tas = {k=n; t=t; dico = dico; ordre=ordre} in
      let rec met_a_jour a = function
         | [] -> ()
         | (b,_)::q -> if inferieur (add p.(a).(b) d.(a)) d.(b) then
                       begin
                       d.(b) <- add p.(a).(b) d.(a);
                       pi.(b) <- a;
                       remonter tas tas.dico.(b)
                       end;
                    met_a_jour a q in
      for i = 1 to n-1 do
         let a = extrait_min tas in
            met_a_jour a g.(a)
      done;
      let distance i = match d.(i) with
          | Infini -> failwith "pas de connexion"
          | N(x) -> x in
      let chemin i =
         let rec chemin_aux j l =
            if j=s then
               s::l
            else
               chemin_aux pi.(j) (j::l) in
         if i=s then
            [s]
         else if pi.(i) = (-1) then
            failwith "pas de connexion"
         else
            chemin_aux i [] in
   distance,chemin;;
```