Corrigé feuille d'exercices n°6

(Option informatique-2^{ème} année)

```
include "carte.ml";;

(* empile: 'a list ref -> 'a -> unit = <fun> *)
let empile p x = p := (x:: !p);;

(* depile: 'a list ref -> 'a = <fun> *)
let depile p = match !p with
  [] -> failwith "erreur dans depile : pile vide"
| x::xs -> p := xs; x;;

La fonction contient vérifie si le sommet  $x$  appartient à la liste des sommets connus :
(* contient: 'a -> ('a * 'b * 'c) list -> bool = <fun> *)
```

```
let rec contient x = function
  [] -> false
| (y,d,l)::ys when x=y -> true
| _::ys -> contient x ys;;
```

La fonction **adjacent** empile les sommets adjacents au sommet v et qui n'appartiennent pas à la liste des sommets connus (on stocke en fait le sommet s' et sa distance à l'origine quand on passe par le sommet v ainsi que le chemin correspondant de s à s'

C = liste des sommets connus (les éléments de C sont de la forme (s_1, d, ch) où s_1 est un sommet, d la distance du départ s au sommet s_1 et ch est un chemin allant de s à s_1 ne passant que par des sommets connus sauf s_1)

p = pile des sommets en attente
 v = dernier sommet connu
 $d0$ = distance minimale de l'origine à v
 $ch0$ = chemin de longueur minimale de l'origine à v
 $arcs$ = listes des arcs du graphe

```
(* adjacent:
('a * 'b * 'c) list ->
('a * int * 'a list) list ref ->
'd -> int -> 'a list -> (('d * 'a) * int) list -> unit = <fun> *)
```

```
let adjacent C p v d0 ch0 arcs =
  let rec trouve p v = function
    [] -> ()
  | ((v1,v2),d)::suite when v = v1 && not contient v2 C
    -> empile p (v2,d+d0,ch0@[v2]); trouve p v suite
  | x::xs -> trouve p v xs in
  trouve p v arcs;;
```

cherche_min recherche le sommet le plus proche de l'origine parmi les sommets en attente :

```
(* cherche_min: ('a * 'b * 'c) list -> 'a * 'b * 'c = <fun> *)
```

```
let rec cherche_min = function
  [] -> failwith "Erreur dans cherche_min : liste vide"
| [x] -> x
| (x,dx,chs)::(y,dy,chy)::queue
  -> if dx <= dy then cherche_min ((x,dx,chs)::queue)
    else cherche_min ((y,dy,chy)::queue);;
```

```

(* supprime: 'a -> ('a * 'b * 'c) list -> ('a * 'b * 'c) list = <fun> *)
let rec supprime v = function
  [] -> []
| (x,dx,ch)::queue -> if x=v then supprime v queue
                      else (x,dx,ch)::(supprime v queue);;

  la fonction "chemin_min" trouve le plus court chemin dans un graphe orienté (représenté par
"arcs" la liste de ses arcs) entre le sommet de départ "s" et le sommet d'arrivée "t".
  elle retourne:

  - (-1,[]) s'il n'existe pas de chemin entre "s" et "t"

  - (d,[s;s2;...;t]) où d est la longueur minimale entre "s" et "t" et [s;s2;...;t] un chemin de longueur
  minimale entre "s" et "t"

(* chemin_min: (('a * 'a) * int) list -> 'a -> 'a -> int * 'a list = <fun> *)
let chemin_min arcs s t =
let C = ref [(s,0,[s])] and attente = ref [] and d = ref 0
and v = ref s and p = ref [] and chemin = ref [] in
adjacent (!C) p s 0 [s] arcs; (* on initialise la liste des sommets à explorer *)
while !p <> [] && !v <> t do
  let (v1,d1,ch1) = cherche_min !p (* on recherche le sommet le plus proche
  du départ parmi les sommets à explorer :
  - v1 est le sommet correspondant
  - d1 la distance minimale entre "s" et "v1"
  - ch1 un chemin de longueur minimale entre "s" et "v1" *)
  in C := (v1,d1,ch1)::(!C); (* on ajoute ce sommet à la liste des sommets connus *)
  p := supprime v1 (!p); (* on supprime v1 des sommets à traiter *)
  adjacent (!C) p v1 d1 ch1 arcs; (* on ajoute les sommets adjacents
  à v1 aux sommets à traiter *)

  v := v1; chemin := ch1; d := d1
done;
if !v = t then (!d,!chemin) (* On retourne (longueur minimale, chemin de longueur minimale *)
else (print_string "Il n'y a pas de chemin reliant les deux sommets";(-1,[])) ;;

# chemin_min A Lille Nice;;
- : int * ville list =
  1191, [Lille; Paris; Dijon; Macon; Lyon; Avignon; Aix; Nice]

# chemin_min A Nice Bayonne;;
- : int * ville list =
  955,
  [Nice; Aix; Avignon; Nîmes; Montpellier; Narbonne; Toulouse; Pau; Bayonne]

```