Projet IPF

Rapport du projet d'IPF 2019/2020

Réalisé par :

Imane EL MOUL

La phase 1 :

Dans cette phase on doit construire un graphe à partir d'un fichier « input.txt » passé en ligne de commande, calculer les plus courts entre la source et le puits mentionnés dans les deux premières lignes et écrire ces chemins sur un fichier output.txt.

(* Construction du graphe*)

Pour le faire j'ai essayé de découpé le problème en plusieurs fonction chacune fait une tâche précise et les regroupé enfin dans une seul fonction main. Et pour la construction des graphes j'ai utilisé les module graph et make_g pour la construction d'un graphe sous forme d'une map dont les clés sont de type String (les nœuds alphanumérique).

```
module Flux = Make_g(String);;
```

Au début j'ai pensé à extraire les nœuds présents dans le graphe j'ai choisis de lire le fichier ligne par ligne, enlever les trois première lignes pour pouvoir extraire les différents arcs du graphe et les mettre dans une liste. Chaque deux nœud successif représentent un arc dans le graphe.

```
let chemin_elem filename =
  let ic = open_in filename in
  let lines = read_lines ic in
  let nodes_1 = (remove_at 0 lines) in
  let nodes_2 = (remove_at 0 nodes_1) in
  let nodes = (remove_at 0 nodes_2) in
  close_in ic;
  (nodes);;
```

Ensuite construire le graphe en ajoutant les nœuds deux par deux en utilisant la fonction add_edge du module graph qui va assurer l'ajout des nœuds également.

```
let rec build_graph l g =
  match l with
  [] -> g
```

```
|t::t1::q ->  let g' = Flux.add_edge t t1 g in build_graph q g'
|[t] -> let g' = Flux.add_vertex t g in g';;
```

Enfin extraire la source et le puits du fichier (les deux premières lignes) pour pouvoir trouver les plus courts chemins.

(* Le plus court chemin*)

Pour la recherche des plus courts chemins j'ai choisit de trouver tous les chemins possible entre la source et le puits mais je n'ai pas pris en compte les chemins avec des boucles (les arcs
n ->n) parce que j'ai considéré qu'il ne peuvent pas être des plus court chemin puisqu'il représente un chemin déjà existant avec un nœud de plus

Après avoir extraire tous les informations nécessaire du fichier et construire le graphe on passe à la recherche des plus court chemins. Et comme d'habitude je préfère toujours découpé le problème en plusieurs fonctionnalités relativement simples.

J'ai commencé par trouver la liste des nœuds présents dans le graphe en construisant une liste constitué des clés de ma map, ensuite j'ai cherché à trouvé les prédécesseurs d'un nœud dans le graphe pour pouvoir revenir sur nos pas (dans un parcours en largeurs du graphe) afin de trouver tous les chemins possibles entre la source et le puits.

```
let rec neighbors g a nodes cond  =
match nodes with
 [] -> []
|t::q -> let succ_t = Flux.NodeSet.elements (Flux.succs t g) in
        if List.mem a succ_t && cond t then
              t:: neighbors g a q cond
       else neighbors g a q cond ;;
```

Trouver tous les chemins possibles (parcours en largeur)

```
let rec list_path g a to_b = match to_b with
  | [] -> assert false (* [to_b] contains the path to [b]. *)
  | a' :: _ ->
```

```
    if a' = a then [to_b]
    else
     let n = neighbors g a' (vertex_list (Flux.NodeMap.bindings g)) (fun c ->not(List.mem c to_b) )in
     List.concat(List.map (fun c -> list_path g a (c :: to_b)) n);;
```

Après avoir réussir à trouver tous les chemins possible il ne reste que trouver les plus court, chaque chemin est représenté par une liste alors il suffit de trouver les qui ont la longueur minimal.

```
let rec shortest_path  paths  min_l =
        match paths with
              [] -> []
             |t:: q -> if min_l = List.length t then
                              t::shortest_path  q min_l
                       else shortest_path  q min_l;;
```

(* Main*)

Au début il faut vérifier qu'on a passé un fichier en argument pour pouvoir construire le graphe sinon on aura une exception levé par main « no arguments ».
J'avais besoins des fonctions auxiliaires pour l'affiche la transformation des liste et les listes de listes à des chaine de caractère et une autre fonction pour l'écriture dans un fichier output.txt.
La fonction main fait appel à tous les fonctions précédentes pour construire le graphe et trouver les plus courts chemins.
S'il n'y a aucun chemin entre la source et le puits la fonction va écrire un message dans le fichier output.

```
let main g = if verify_args then
              let chemin_elmen= chemin_elem (Sys.argv.(1)) in
              let graphe =build_graph chemin_elmen g in
              let s,p = source_puit (Sys.argv.(1)) in
              let paths = list_path graphe s [p] in
              if List.length paths !=0 then
                 let minl_paths = l_min (lenght_paths paths) in
                 let short_paths = shortest_path  paths minl_paths in
```

```
            output_lines "output.txt" short_paths
        else
    let message = [[" Aucun chemin entre la source et la puit"]] in
    output_lines "output.txt" message
      else failwith "no arguments";;
```

Phase2 :

Le code initial de la phase 2 se trouve dans le fichier phase2.php, malheureusement cette phase n'est pas complète à cause des difficultés que j'avais rencontrés dans l'implémentation de l'algorithme de Dinic.
J'ai commencé par la création d'une fonction pour stocker les nœuds présents dans le fichier. Chaque arc et représenté par un couple de quatre éléments : les deux premiers représente l'ars et le troisième représente sa capacité initial (0) et le quatrième représente sa capacité maximale.
Ensuite il faut construire le graphe en utilisant les deux première éléments de chaque couple.

```
let rec build_graph2 nodes g =
  match nodes with
  [] -> g
  |(t,t1,_,_)::q ->  let g' = Flux.add_edge t t1 g in build_graph2 q g';;
```

Maintenant il faut que je puisse représenter les chemins entre la source et le puits par des chemins élémentaires et leur capacité (le format précédent)

```
let rec c_paths nodes paths = match paths with
                [] -> []
               |t::q -> c_path nodes nodes t 0::c_paths nodes q;;
```

Enfin il y a deux fonctions permettant de changer les capacités des chemins quand un flux bloquant est trouvé, pour le chemin qui passe par ce flux bloquant il doit mettre à jour la capacité initiale de tous ses arcs et les autres chemins doivent mettre à jour la capacité initiale des arcs en commun avec le chemin courant.
Et j'étais bloqué ici vu que les capacités initiales ne sont toujours mises à jour correctement.