Projet INF201

COMMENT VOTER ?

Groupe: INF2

Binôme: Shaghayagh HAJMOHAMMADKASHI

Hadil CHTIOUI

Mounia AREZZOUG

Question 1

- type candidat = string (* Le type candidat représente le nom d'un candidat qui est une chaine de caractères *).
- type bulletin = candidat (* Un bulletin est le nom d'un candidat qui est une chaine de caractères *).
- type urne = bulletin list (* le type urne est une liste qui contient des bulletins *).
- type score = int (* le type score est un entier qui représente le nombre de votes qu'a une personne *).
- type panel = candidat list (* Le type panel est une liste de bulletin qui contient les personnes qui se présentent a l'élection *).
- type résultat = candidat*score list (* le type résultat est une liste de couples tel que le premier élément du couple est le candidat et le deuxième élément du couple est le nombre de votes pour ce candidat *).

Question 2

Profil : Fonction compte : candidat -> urne -> int

Sémantique :

- Cette fonction prend en entrée un nom de candidat (une chaîne de caractères) et une urne (une liste de bulletins).
- Elle renvoie le nombre de votes (un entier) pour le candidat spécifié dans l'urne.

Exemple :

```
let u1 =
["Eric";"Kyle";"Stan";"Kyle";"Kyle";"Stan";"Eric";"Eric";"Kyle";"Eric";"Stan";"Eric";"Eric";"Eric";"Stan";"
Stan"];;

let u1 = ["Eric";"Kyle";"Stan";"Kyle";"Kyle";"Stan";"Eric";"Eric";"Kyle";"Eric";"Stan";"Eric";

"Eric";"Eric";"Stan";"Stan"];;

compte "Eric" u1;; (* Renvoie 7 *)
```

Réalisation :

```
Let rec compte (c:candidat)(u:urne):score=
  match u with
  |[]->0
  |x::suite->if x=c then1+(compte c suite) else (compte c suite);;
```

Question 3 :

Profil : panel -> urne -> résultat list

Sémantique :

Cette fonction prend en entrée le panel des candidats et l'urne des votes, et renvoie la liste des résultats de l'élection, qui est une liste de couples (candidat, score).

Exemple :

Let lc1= ["Keny";"Kyle";"Stan"]

let u1= ["Eric";"Kyle";"Stan";"Kyle";"Kyle";"Stan";"Eric";"Eric";"Kyle";"Eric";"Stan";"Eric";"Eric";"Eric";"Stan";"Stan"]

depouiller lc1 u1 → [("Keny", 0); ("Kyle", 4); ("Stan", 5)]

Réalisation :

```
Type resultat=(candidat*score) list;;
Let rec depouiller (lc:panel)(u:urne):resultat=
  match lc with
  |[]->[]
  |x::suite -> (x,(compte x u))::(depouiller suite u);;
```

Question 4 :

Profil : union: resultat -> resultat -> resultat

Sémantique :

La fonction **union** prend en entrée deux listes de type resultat et renvoie une liste de type resultat et fait la somme des deux listes d'entrée pour chaque candidat.

Exemples :

let r1 = [("Eric", 5); ("Kyle", 1); ("Stan", 5)];;

let r2 =[("Eric", 4); ("Kyle", 2); ("Stan", 6)];;

union r1 r2 → [("Eric", 9); ("Kyle", 3); ("Stan", 11)]

Réalisation :

```
let rec union (r1: resultat) (r2: resultat) : resultat =
  match r1, r2 with
  | [], [] -> []
  | (x1, s1) :: suite1, (x2, s2) :: suite2 -> (x1, s1 + s2) :: union
suite1 suite2
  | _, _ -> failwith "longueurs de listes sont différentes";;
```

Question 5 :

Profil : max_depouille = l : resultat -> resultat

Sémantique :

La fonction **max_depouille** est une fonction récursive qui prend en argument une liste de resultat et renvoie un resultat (candidat *score) autrement dit elle va renvoyer le candidat ayant le meilleur score et son score.

Exemples :

[ ]-> erreur

[(a,b)]->(a,b)

Réalisation :

```
let r3= union r1 r2 ;;
let rec max_depouille (l: resultat): candidat*score =
  match l with
  | [] -> failwith "La liste ne peut pas être vide"
  | [x] -> x
  | (a1, b1) :: (a2, b2) :: suite ->
      let (candidat, score) = max_depouille ((if b1 > b2 then (a1, b1)
else (a2, b2)) :: suite) in
      if b1 > score then (a1, b1) else (candidat, score);;

max_depouille r3;;
```

Question 6 :

Profil : vainqueur_scrutin_uninominal = urne -> panel -> candidat

Sémantique :

la fonction **vainqueur_scrutin_uninominal** prend en argument un panel et une urne et renvoie candidat (string) qui est le vainqueur de cette élection.

```
let vainqueur_scrutin_uninominal (u: urne) (lc: panel) : candidat =
  let resultats = depouiller lc u in
  let (candidat_vainqueur, _) = max_depouille resultats in
  candidat_vainqueur;;
```

Question 7 :

Profil : suppr_elem = 'a list -> 'a -> 'a list

deux_premiers : urne -> panel -> (candidat * score) * (candidat * score)

Sémantique :

- La fonction **deux_premiers** permet d'extraire les deux premiers gagnants de l'élection à partir du dépouillage d'une urne .
- La fonction **suppr_eleme** qui prend deux arguments l : une liste et e : un élément supprime une occurrence de l'élément e dans la liste l.

Réalisation :

```
let rec suppr_elem (l:'a list)(e:'a):'a list=
  match l with
  |[]-> []
  |x::suite->if x=e then suppr_elem suite e
      else x::suppr_elem suite e;;

let deux_premiers (u:urne)(p:panel):(candidat*score)*(candidat*score)=
  let liste1=depouiller p u in
  let premier=max_depouille liste1 in
  let candidat_premier=vainqueur_scrutin_uninominal u p in
  let nouvelle_urne=suppr_elem u candidat_premier in
  let liste2=depouiller p nouvelle_urne in
  let second=max_depouille liste2 in
  premier, second;;
```

Question 8 :

- Présidentielles de 1988 : François Mitterrand gagne contre Jacques Chirac mais aurait perdu contre Raymond Barre qui n'arrive pas au second tour.

- Présidentielles de 1995 : Jacques Chirac gagne contre Lionel Jospin au second tour, mais si Philipe de Villiers n''etait pas candidat, Edouard Balladur aurait pu devancer Chirac au premier tour.

- Présidentielles US de 2000 : G.W. Bush est elu mais si Ralph Nader ne s''etait pas présenté, Al Gore aurait été élu.

- Présidentielles de 2017 : Sans la candidature de Benoit Hamon, Jean-Luc M´Mélenchon pouvait arriver au second tour.

- Présidentielles 2022:

Au premier tour, c'est Emmanuel Macron et Marine LePen qui remportent la victoire avec 27.85% et 23.15% chacun respectivement.

On pourrait se dire que dés sont lancés et les résultats pour la suite sont clairs, pourtant, selon Macron «rien est encore joué».

Il dit dans son discours après l'annonce des résultats du premier tout : «J'invite solennellement nos concitoyens, quelques soit leurs sensibilités, leurs choix au premier tour, à nous rejoindre. Certains le feront pour faire barrage à l'extrême droite. Je suis pleinement conscient que cela ne vaudra pas soutiens du projet que je porte, et je le respecte».

Jean-Luc Mélenchon, troisième avec 21.95% des voix, a martelé son rejet de Marine Le Pen mais n'appelle pas pour autant à voter pour Macron;

«Il ne faut pas donner une seule voix à Marine le Pen».

Yannick Jadot(4.63%), Fabien Roussel(2.28%), Anne Hidalgo(1.75%) et Valérie Pécresse(4.78%) appellent à faire barrière à la droite et empêcher l'arrivée au pouvoir de Le Pen en déposant dans l'urne un bulletin Emmanuel Macron.
De l'autre coté, les candidats Eric Zemmour(7.07%) et Nicolas Dupont-Aignan(2.06%) ont apporté leur soutien à Marine Le Pen et «supplient» les français à voter contre Macron.
Quant au reste des candidats, c'est à dire Philippe Poutou(0.77%), Jean Lassale(3.13%) et Nathalie Arthaud (0.56%) n'ont pas donné consigne de vote mis à part celle de «Votez blanc» pour rejeter et Marine Le Pen et Emmanuel Macron.

Au second tour, c'est Macron qui l'emporte. Comment se fait-il?

Tout s'est joué sur les consignes des «petits candidats», ceux qui n'ont pas gagné mais qui néanmoins ont su conquérir un certain nombre d'électeurs. On constate que le nombre d'électeurs qui ont voté en faveur de Jadot, Roussel, Hidalgo et Pécresse (et qui appellent à voter Macron) est bien supérieur à celui de Zemmour et de Dupont-Aignan (qui eux sont pour Le Pen).

Les électeurs se retrouvent obligés de voter stratégiquement, c'est-à-dire soit de voter blanc (ce qui ne sert à rien),soit de voter pour un candidat ou un parti qui n'est pas leur premier choix (et qui est généralement celui choisi par leur premier choix) afin de maximiser les chances de défaire un candidat ou un parti qu'il considèrent comme étant le pire choix. Macron l'avait assumé :
""Certains le feront pour faire barrage à l'extrême droite.""

En 2022, si on somme tous les taux, on pourra dire que (100-(27.85+23.15))=49% des électeurs qui ont voté n'ont pas été satisfaits du résultat de l'élection et c'est la exactement que se repose le problème du scrutin uninominal: il ne représente pas la volonté du peuple, seulement d'une partie... et parfois, une grosse partie (49% ce n'est pas rien!!!). *)

Question 9 :

- Le principal problème du scrutin uninominal est qu'il peut conduire à une représentation politique qui n'est pas proportionnelle à la volonté de l'électorat. En conséquence, le scrutin uninominal peut être considéré comme étant peu représentatif et peu équitable pour la représentation des minorités politiques ou des idées minoritaires. Il favorise souvent les grands partis politiques et peut donner lieu à des distorsions significatives dans la représentation politique.

Question 10:

- Dans le cadre d'une élection avec 12 candidats et un système de jugement majoritaire, le nombre de bulletins différents qu'un électeur peut mettre dans l'urne dépend du nombre de mentions possibles pour chaque candidat. Supposons qu'il y ait 6 mentions possibles pour chaque candidat (Très bien, Bien, Assez bien, Passable, Insuffisant et À rejeter), alors chaque électeur aurait 6 choix pour chaque candidat, soit 6^{12} bulletins différents possibles.
- Comparé aux 13 possibilités du scrutin uninominal (12 candidats + nul/blanc), le jugement majoritaire permet aux électeurs d'exprimer des préférences plus nuancées pour chaque candidat, ce qui peut conduire à des résultats d'élection plus représentatifs des opinions des électeurs.

Question 11:

- Type mention = Arejeter|Insuffisant|Passable|Assezbien|Bien|Tresbien
- Type bulletin_jm= mention list
- Type urne_jm = bulletin_jm list

Question 12 :

Profil : depouille_jm = u : urne_jm -> mention list list
Sémantique :

La fonction **depouille_jm** prend en argument une liste de type urne_jm et renvoie une list contenant les liste des mentions de chaque candidats.

```
Let u3: urne_jm = [[Tresbien;Assezbien;Arejeter;Passable];(* Premier bulletin *)
[Assezbien;Assezbien;Arejeter;Tresbien];(* Second bulletin *)
[Tresbien;Arejeter;Arejeter;Tresbien]] (* Troisième bulletin *)

Let rec depouille_jm (u:urne_jm) =
  match u with
  |[] ->[]
  |[]::_ ->[]
  | u -> (List.map List.hd u) :: depouille_jm (List.map List.tl u);;

Let ms = depouille_jm u3;;
```

Question 13 :

Profil : tri_mentions = mll: mention list list -> mention list list
Sémantique :

La fonction **tri_mentions** s'applique sur une liste de listes de mentions, et utilise la fonction de tri de liste prédéfinie "List.sort" pour trier les mentions par ordre croissant. Elle renvoie ensuite une nouvelle liste de listes de mentions triées.

```
Let tri (l:'a list):'a list = List.sort compare l;;

(*test*)
tri [Assezbien;Bien;Arejeter;Tresbien];;
let tri_mention (mll: mention list list):mention list list=
  List.map tri mll ;;

Let ms_triee = tri_mention ms;;
```

Question 14 :

Profil : mediane = liste :bulletin_jm -> mention
Sémantique :

La fonction **mediane** renvoie la médiane d'une liste triée.

```
Let mediane (liste:bulletin_jm):mention =
  Let n = List.length liste in
  List.nth liste (n / 2);;

(*test*)
List.map mediane ms_triee;;
```

Question 15 :

Profil : meilleure_mediane = mll: mention list list -> mention
Sémantique :
La fonction **meilleure_mediane** renvoie la meilleure mediane à partir d'une liste contenant les mentions de chaque candidats.

```
Let meilleure_mediane (mll:mention list list):mention =
  Let n=List.length mll in
  Let liste=tri(List.map mediane (tri_mention mll)) in
  List.nth liste (n-1) ;;

meilleure_mediane ms_triee;;
```

Question 16 :

Profil : supprime_perdants= mll: mention list list -> mention list list
Sémantique :
La fonction **supprime_perdants** supprime tous les candidats qui ont une médiane inférieur à la meilleure médiane et finit par remplacer la liste de mentions des candidats par une liste vide [ ].
Exemples :
supprime_perdants ms_triee =[[Assezbien; Tresbien; Tresbien]; []; []; [Passable; Tresbien; Tresbien]]

```
letsupprime_perdants (mll: mention list list): mention list list =
  letmeilleur_mediane = meilleure_mediane mll in
  List.map (fun bulletin ->
    if mediane bulletin >= meilleur_mediane then
      bulletin
    else
      []) mll;;
supprime_perdants ms_triee;;
```

Question 17 :

Profil : supprime_mention = liste : bulletin_jm -> m:mention -> bulletin_jm
supprime_meilleure_mediane = mll: mention list list -> mention list list
Sémantique :

- La fonction **supprime_mention** permet de supprimer une mention dans une liste en ne supprimant qu'une seule occurrence de cette mention.
- La fonction **supprime_meilleure_mediane** prend en argument une liste de listes de mentions (candidats) et supprime la meilleure médiane de chaque candidat en utilisant la fonction supprime_mention. Le processus est répété jusqu'à ce qu'il ne reste plus qu'un seul candidat ou qu'un candidat ait une médiane supérieure à toutes les autres.

```
let rec supprime_mention (m:mention) (l:bulletin_jm):bulletin_jm =
  match l with
  | [] -> []
  | hd::tl -> if hd = m then tl else hd :: supprime_mention m tl
;;

let supprime_meilleure_mediane (l:urne_jm):urne_jm =
  let non_empty_lists = List.filter (fun x -> x <> []) l in
  let best_median = meilleure_mediane non_empty_lists in
  List.map (fun lst -> if lst = [] then [] else supprime_mention best_median lst) l
;;
```

Question 18:

Profil : vainqueur_jm= mll : mention list list -> string
Sémantique :

- La fonction **vainqueur_jm** prend en argument une liste de listes de mentions (candidats) et renvoie indice du vainqueur.

Exemples :
vainqueur_jm [[];[Passable]; []; []];; → « 1 »
vainqueur_jm [[Passable; Tresbien; Tresbien]; [Arejeter; Bien; Tresbien];
[Arejeter; Arejeter; Passable]; [Insuffisant; Passable; Tresbien]];; → « 0 »

```
let vainqueur_jm (mll:mention list list):string =
  let candidats = List.init (List.length mll) (fun i -> i) in
   let filtered_candidates = List.filter (fun i -> (List.nth mll i) != [])
candidats in
  match filtered_candidates with
  | [i] -> string_of_int i
  | _ ->
    let medians = List.map (fun i -> mediane (List.nth mll i)) filtered_candidates
in
    let max_median = List.fold_left max Arejeter medians in
    let max_count = List.fold_left (fun acc m -> if m = max_median then acc + 1
else acc) 0 medians in
    if max_count = 1
    then string_of_int (List.find (fun i -> mediane (List.nth mll i) = max_median)
filtered_candidates)
    else ""
;;
```

(Question 20)

Puisque le jugement majoritaire est un mode de vote consistant à attribuer des mentions à chaque candidat, ce qui fait que ce mode prend en considération les préférences des électeurs ; Alors le dilemme de vote utile se disparaît.

En revanche, le jugement majoritaire peut être une source problématique si les candidats appellent tous leurs électeurs à un vote très polarisé et si ces derniers sont nombreux à leur obéir, on risque alors d'aboutir à un rejet de tous les candidats, aucun ne réussissant à obtenir la médiane parfaite

Où même en traitant le cas précède on voit le paradoxe du mot 'majoritaire' qui ne reflète pas toujours les résultats (d'avoir des sauts entre les résultats d'un candidat par rapport aux autres ou bien d'avoir des différences très petites entre eux).

Question 21 :

Les définitions des types ville, zone et arbre.

- Type ville = string * resultat;;
- Type zone = Reg of string | Dpt of string;;
- Type arbre = Bv of ville | N of zone * arbre list;;

Question 22 :

Profil : trouve_bv = a: arbre -> nom_bv: ville -> resultat
Sémantique :

- La fonction **trouve_bv** extrait le résultat d'un bureau de vote à partir d'un arbre et du nom du bureau de vote(ville) .

```
let rec trouve_bv (a: arbre) (nom_bv: string) : resultat =
  match a with
  | Bv (v, res) -> if v = nom_bv then res else []
  | N (_, people) ->
    let results = List.map (fun pip -> trouve_bv pip nom_bv) people in
    try
      List.find (fun r -> r <> []) results
    with Not_found -> []
;;
```

Question 23 :

A l'aide de la fonction **union** et de la fonction **trouve_bv** donner le résultat de la présidentielle 2022 si seulement les villes de Grenoble, Fontaine et Valence étaient prises en compte.

```
let resultat_grenoble = trouve_bv ara "Grenoble";;
let resultat_fontaine = trouve_bv ara "Fontaine";;
let resultat_valence = trouve_bv ara "Valence";;

let resultat_total = union (union resultat_grenoble resultat_fontaine)
resultat_valence;;
```

```
val ara : arbre =
  N (Reg "Auvergne-Rhône-Alpes",
   [N (Dpt "Drôme",
     [Bv
       ("Valence",
        [("ARTHAUD", 161); ("ROUSSEL", 595); ("MACRON", 7756);
         ("LASSALLE", 590); ("LE PEN", 4679); ("ZEMMOUR", 2080);
         ("MELENCHON", 8398); ("HIDALGO", 519); ("JADOT", 1701);
         ("PECRESSE", 1423); ("POUTOU", 186); ("DUPONT-AIGNAN", 573)]);
      Bv
       ("Romans-sur-Isère",
        [("ARTHAUD", 181); ("ROUSSEL", 371); ("MACRON", 4030);
         ("LASSALLE", 334); ("LE PEN", 3270); ("ZEMMOUR", 1072);
         ("MELENCHON", 4108); ("HIDALGO", 251); ("JADOT", 850);
         ("PECRESSE", 631); ("POUTOU", 111); ("DUPONT-AIGNAN", 341)])]);
    N (Dpt "Isère",
     [Bv
       ("Meylan",
        [("ARTHAUD", 28); ("ROUSSEL", 169); ("MACRON", 4457);
         ("LASSALLE", 164); ("LE PEN", 1288); ("ZEMMOUR", 928);
         ("MELENCHON", 2198); ("HIDALGO", 251); ("JADOT", 906);
         ("PECRESSE", 763); ("POUTOU", 64); ("DUPONT-AIGNAN", 162)]);
      Bv
       ("Echirolles",
        [("ARTHAUD", 104); ("ROUSSEL", 506); ("MACRON", 3276);
         ("LASSALLE", 259); ("LE PEN", 2737); ("ZEMMOUR", 779);
         ("MELENCHON", 5121); ("HIDALGO", 223); ("JADOT", 590);
         ("PECRESSE", 360); ("POUTOU", 92); ("DUPONT-AIGNAN", 202)]);
      Bv
       ("Fontaine",
        [("ARTHAUD", 55); ("ROUSSEL", 363); ("MACRON", 2111);
         ("LASSALLE", 146); ("LE PEN", 1835); ("ZEMMOUR", 541);
         ("MELENCHON", 3113); ("HIDALGO", 185); ("JADOT", 493);
         ("PECRESSE", 212); ("POUTOU", 83); ("DUPONT-AIGNAN", 121)]);
      Bv
       ("Saint-Martin-d"... (* string length 21; truncated *),
        [("ARTHAUD", 58); ("ROUSS"... (* string length 7; truncated *), 436);
         ("M"... (* string length 6; truncated *), 2769); ...]);
      ...]);
    ...])

val resultat_grenoble : resultat =
  [("ARTHAUD", 256); ("ROUSSEL", 1300); ("MACRON", 15968); ("LASSALLE", 845);
   ("LE PEN", 6444); ("ZEMMOUR", 3389); ("MELENCHON", 24568);
   ("HIDALGO", 1488); ("JADOT", 5644); ("PECRESSE", 2019);
   ("POUTOU", 508); ("DUPONT-AIGNAN", 661)]

val resultat_fontaine : resultat =
```

```
  [("ARTHAUD", 55); ("ROUSSEL", 363); ("MACRON", 2111); ("LASSALLE", 146);
   ("LE PEN", 1835); ("ZEMMOUR", 541); ("MELENCHON", 3113);
   ("HIDALGO", 185); ("JADOT", 493); ("PECRESSE", 212); ("POUTOU", 83);
   ("DUPONT-AIGNAN", 121)]

val resultat_valence : resultat =
  [("ARTHAUD", 161); ("ROUSSEL", 595); ("MACRON", 7756); ("LASSALLE", 590);
   ("LE PEN", 4679); ("ZEMMOUR", 2080); ("M´ ELENCHON", 8398);
   ("HIDALGO", 519); ("JADOT", 1701); ("P´ ECRESSE", 1423); ("POUTOU", 186);
   ("DUPONT-AIGNAN", 573)]

val resultat_total : resultat =
  [("ARTHAUD", 472); ("ROUSSEL", 2258); ("MACRON", 25835);
   ("LASSALLE", 1581); ("LE PEN", 12958); ("ZEMMOUR", 6010);
   ("MELENCHON", 36079); ("HIDALGO", 2192); ("JADOT", 7838);
   ("PECRESSE", 3654); ("POUTOU", 777); ("DUPONT-AIGNAN", 1355)]
```