DIOP Sokhna
LIETO Myke
BOSSO Gavril
DA SILVA SANTOS Lucas
GIRODENGO Camille

SAE 1.01 – *Analyse de système de vote*

I- Introduction

Dans le cadre de cette SAE, nous avons choisi de coder 4 systèmes de votes différents : par scrutin majoritaire, par scrutin de vote uninominal, par scrutin de vote instantané et par scrutin proportionnel. Chacun de ses systèmes est un algorithme à part. A noter que pour le rendu de ce projet, il est envoyé un lien vers le répertoire où se trouve le code source des projets ainsi que des dossiers ZIP contenant les projets pour faciliter la vérification.

II- Les algorithmes associés aux différents systèmes de vote

a. Le système de scrutin majoritaire

Les modes de scrutin majoritaires regroupent les modes de scrutin caractérisés par une victoire de la ou des personnes ayant obtenu davantage de voix que leurs concurrents.
C'est donc un système de vote dans lequel la personne ayant obtenu le plus de voix gagne.
Il s'agit du Système_1 qui affiche le ou les gagnant(s) du vote c'est à dire : le jeu et le nombre de voix qu'il a obtenu. Ou bien, s'il s'agit d'une égalité entre deux ou plusieurs jeu(x), le nombre de voix qu'ils ont obtenus et la liste de ces jeux. (Voir les exemples ci-dessous)

Dans le cas d'une victoire :
Résultat du scrutin :
Nb Votants : 60
Gagnant : jeu 2 avec 27 votes.

Dans le cas d'une égalité :
Résultat du scrutin :
Nb Votants : 38
Il y a une égalité de 13 votes entre :
Jeu 1
Jeu 2

Fonctionnement de l'algorithme :

On a tout d'abord créé deux tableaux : un premier tableau (*tabJeux*) de string avec les noms des différents jeux dans chaque case (que l'on a initialisé avec les 4 jeux : CounterStrike, Street Fighter II, Civilization VI et Mario Kart) et un deuxième tableau d'entier naturel (*tabVotes* qui a la même taille que *tabJeux*) qui va nous permettre de compter le nombre de voix de chaque jeu.

Nous allons tout d'abord rentrer dans une boucle simple et grâce à cin nous allons parcourir un fichier d'entrée (contenant la liste des votes sur chacune de ses lignes) dans lequel nous récupérerons chaque ligne (avec la méthode getline())tant que ce n'est pas la fin du fichier : c'est notre condition d'arrêt. Le nombre de ligne sera ensuite stocké dans une variable prédéfinie (nbVotants).

Lorsque dans une ligne de notre fichier d'entrée on récupère "1" par exemple, on incrémentera de 1 *tabVotes*[0] qui aura donc une nouvelle valeur. On va répéter la même opération (donc lorsque sur une ligne on lit "2", "3" ou encore "4") jusqu'à fin du fichier.

Une fois sorti de la boucle nous déclarons deux variables : *indiceVoteMax* (indice du jeu avec le plus de voies, initialisé à 0) et *egalite* (qui est un booléen initialisé à "faux").

Ces deux variables vont nous permettre par la suite de déterminer l'indice du jeu avec le plus de voix à l'aide d'une boucle ''pour" : si *tabVotes*[i] est supérieur à tabVotes[indiceVoteMax], *indiceVoteMax* aura pour valeur "i".

Dans le cas d'une égalité entre *tabVotes*[i] et *tabVotes*[*indiceVoteMax*], la variable *egalite* aura pour valeur "vrai".

La dernière partie de l'algorithme est consacré plus particulièrement à l'affichage avec :

- Dans le cas d'une victoire : l'affichage du nombre de votants, le nom du gagnant avec le nombre de voix qu'il a obtenu.
- Dans le cas d'une égalité : l'affichage du nombre de votants, la valeur de l'égalité et la liste des jeux qui sont à égalité (à l'aide d'une boucle "pour" qui va parcourir le tableau avec les votes : *tabVotes* et repérer les valeurs qui sont égales (lorsque *egalite* vaut vrai) pour nous donner leurs équivalents dans la liste des différents jeux à l'aide d'indice *indiceVoteMax*).

b. Le système de vote uninominal

Le système de vote uninominal est le système de vote utilisé lors des élections présidentielles, il y a potentiellement 2 tours, au premier tour les votants vont voter pour leur proposition favorite, si à l'issu de ce vote, une proposition à plus de 50% des voix, l'élection se termine, dans le cas où aucune proposition a plus de 50% des voix, un deuxième tour à lieu où les deux propositions avec le plus de voix lors du premier tour sont sélectionnés, les votants vont donc devoir choisir entre ces deux propositions et c'est celle avec le plus de voix qui sera élue.

Le programme sélectionne donc le fichier Entrée, si l'un des jeux à plus de 50% le programme affiche :
- Victoire par majorité absolue
- "le jeux"
- "son pourcentage de voix"

Dans le cas d'un deuxième tour suivi d'une victoire, le programme affiche :
- Victoire par deuxième tour
- "le jeux"
- "son nombre de votes"

Dans le cas d'un deuxième tour suivi d'une égalité, le programme affiche :
-Egalité entre
- "les deux jeux"
- pas de gagnant

Dans le cas d'une égalité entre les 4 propositions dès le premier tour le programme sélectionne les deux jeux situés en premier dans le tableau puis effectue un deuxième tour normalement.

Fonctionnement du programme :
- déclaration de la procédure "*triDoubleVectAssocies*"
- initialisation de deux tableaux :
  - tableau de chaines de caractères : *tabOption*
  - tableau d'entiers naturels : *tabCpt*
- déclaration d'un tableau de chaine de caractère pour déclarer les jeux
- les données rentrée dans le fichier d'entrée vont permettre de calculer le jeu qui va avoir le plus de voix
- le jeux avec le plus de votes va être placé dans la première case du tableau (*tabCpt*[0])
- on va calculer le pourcentage de voix de *tabCpt*[0] afin de déterminer s'il y aura un deuxième tour ou non
- s'il y a un deuxième tour les deux jeux avec le plus de voix sont sélectionnés et les compteurs sont réinitialisés à 0
- la même opération que pour le premier tour se produit
  - si un des deux jeux obtient plus de voix que l'autre victoire s'affiche
  - si une égalité survient un message annonçant une égalité est affiché

c. Le système de vote instantané

Le Système de vote instantanée est un système électoral où les électeurs classent les candidats par ordre de préférence. Lors de ce vote c'est uniquement le premier choix qui est comptabilisé, à la fin des votes si un candidat a plus de 50% des votes il a gagné, sinon dans tous les cas, le classement des préférences est donné.

```
En entrée:
2314
2314
2134
2423
2341
2413
3412
3124
4231
4231

En sortie:
lecture du fichier...
fin de lecture du fichier.
nombre de votes: 10
nombre de position 1 pour CounterStrike : 0
nombre de position 1 pour StreetFighter : 6
nombre de position 1 pour CivilizationVI : 2
nombre de position 1 pour MarioKart : 2
StreetFighter est le gagnant !
```

- Explication du système

1. Déclaration du struct "*jeux*" avec ses propriétés.
3. Déclaration des jeux avec les propriétés de "*jeux*"
4. Boucle infinie avec pour condition de sortie la fin de lecture du fichier.

A) On prend le fichier ligne par ligne

B) On considère chaque ligne comme une chaine de string, Pour chaque ligne nous prenons donc chaque caractère qui la compose : '1', '2', '3' ou '4'. Ces caractères signifient le vote d'un jeu et donc leur ordre va définir leur rang dans le vote. str[0] signifie par exemple le caractère d'indice 0 dans la chaine str, c'est le vote pour rang 1 (0+1). Si le caractère vaut '1', on va incrémenter position_1 dans le jeu concerné. Nous incrémentons donc les propriété "position" des jeux en conséquence.

D) Incrémentation du compteur total à chaque itération de la boucle, cela va nous permettre d'avoir le nombre exact de votes.

5. Affiche tous les votes et le nombre de votes pour la première position des jeux
6. Si la position d'un jeu est supérieure au compteur/2, il a gagné car il a plus de 50% des votes
7. Sinon il n'y a pas de gagnant mais tout de même un classement en fonction des rangs attribués.

Nous avons fait ce classement est créant un tableau des jeux et avec deux boucles 'for' imbriqués. Nous avons ensuite comparé chaque quantité de vote pour la position 1 de chaque jeu et avons incrémenté la propriété rang en conséquence de cette comparaison.
Par exemple, si le jeux 1 à 61 votes pour lui et le jeux 2 en a 48, le rang du deuxième jeu sera incrémenté car il a moins de vote que le premier jeu.

Nous avons utilisé les structs, car il était plus facile pour nous de voir les jeux comme des objet, nous avons pu, par exemple, utiliser les mêmes variables pour la position ou le nom de chaque jeu, cela nous permettait d'éviter de créer plusieurs variables ayant la même utilité.
Grâce à cela, nous avons pu faire des tableaux de jeux et n'en extraire seulement que le rang de chaque jeu.
Avant d'utilisé les structs, nous avons essayé d'envisager les jeux comme des tableaux avec chaque vote pour ce jeu dans le tableau, nous avons rencontré des difficultés notamment pour le système de rang, c'est pourquoi nous avons abandonné l'idée.
Nous n'avons malheureusement pas atteint notre objectif final qui était de faire supprimer le dernier du classement et de refaire un vote avec les trois jeux restants.
Nous avons essayé d'utiliser les matrices pour les systèmes de rangs, mais c'était trop difficile pour nous par manque de maîtrise.

d. Le système de scrutin proportionnel

- Résumé du comportement

Le système de scrutin proportionnel est un système qui renvoie un classement allant de celui qui a eu le plus de vote à celui qui en a eu le moins. Le dossier associé à ce type de scrutin est « Système_4 ». Le programme affiche une liste numérotée contenant le nom du jeu suivit du nombre de vote en sa faveur selon le modèle suivant :
Resultat du scrutin par systeme proportionnel :

1. *Jeu1 : 8*
2. *Jeu3 : 5*
3. *Jeu2 : 1*

* Les nombres de vote sont donnés en tant qu'exemple pour monter le comportement du programme

- Explication

Pour regrouper les noms des jeu proposés et créer leurs compteurs, on a inclus la bibliothèque <vector> pour ensuite créer les vecteurs de string *vectorOptions* et d'entiers naturels *vectorCpt*, initialisé à la taille de *vectorOptions*.
Le programme va ensuite pour chaque ligne du fichier n entrée, récupérer la ligne en question, l'attribuer à une variable de type string *optVote* qui sera analysée dans l'équivalent d'un switch / case pour incrémenter le compteur correspondant de 1. Et ce jusqu'à la fin du fichier.
Une fois les compteurs finaux obtenus, il faut les trier du plus grand au plus petit mais utiliser la fonction swap ne permet pas à ce que l'indice des jeux options change en même temps que celui des compteurs de *vectorCpt* qui leur sont associés. Ainsi, on a dû créer la procédure de signature *« void triDoubleVectAssocies (vector<string> & vectorOptions, vector<unsigned> & vectorCpt) ; »* qui répond au problème motionné juste avant.
Le reste du programme est dédié à l'affichage du classement. La seule particularité est qu'avant d'entrer dans la boucle pour l'affichage, on déclare une variable entier naturel *rank* initialisée à 1 qui va représenter le classement des jeux. A noter qu'en prévision des égalités possibles entre les jeux, il n'était donc pas possible d'utiliser les indices : on incrémente (juste avant la fin de boucle) donc *rank*

de 1 seulement si la condition « i est inférieur à la taille de *vectorOptions* -1 ET_ALORS *vectorCpt*[i+1] vaut *vectorCpt*[i] » n'est pas respectée. Dans le cas contraire, on revient en haut de la boucle avec l'instruction « continue ; ».

III- Création des fichiers d'entrée et des fichiers oracles

Avant de réaliser des fichiers oracles contenant plus d'une centaine de vote, on a d'abord créé quelques fichiers d'entrée avec une dizaine de votes seulement pour vérifier la validité des programmes, c'est-à-dire si les fichiers d'entrée sont bien lus, si l'affichage correspond au modèle souhaité, si certains affichages ne se font pas, si les compteurs fonctionnent, si les procédures et fonctions de tri ou autres crées fonctionnent correctement, si les résultats sont bien implémentés dans le fichier de sortie, etc… La comparaison lors de ces tests se fait petit à petit et la comparaison est visuelle avant d'utiliser diff dans le terminal.

Cependant, les quatre systèmes que nous avons représentés ne fonctionnent pas de la même manière : certains requièrent plusieurs votes consécutifs pour un seul votant (Système_3) tandis que d'autres nécessitent un second tour (Système_2). Nous avons néanmoins établi un certain modèle à suivre quant au contenu de ces fichiers : chaque option est représentée par un nombre (1, 2, 3, ou 4), il n'y a qu'un seul vote par ligne et les noms des votants ne sont pas retenus dans le fichier d'entrée du fait que cette information n'est pas nécessaire. Si toutefois nous venions à en avoir besoin, il suffira de chercher les lignes i et i+1 telles que i = numéro_de_ligne_du_vote * 2.

Pour les systèmes 1 et 4, le modèle de fichier d'entrée suffit et ne nécessite pas de changement. Pour les systèmes 2 et 3 en revanche, il faut adapter le modèle.

Pour le Système_2, afin que les votes du premier et deuxième tour soient séparés et ainsi tous lu dès le premier tour, un caractère de séparation est placé sur la ligne entre le dernier vote du premier tour et le premier du deuxième. Ainsi, la string d'arrêt de lecture du 1er tour sera « - ». De plus les deux premières options du second tour se verront attribuer de nouveaux numéros (1 et 2) pour le scrutin du 2^{e} tour.

Pour le Système_3 nous avions deux possibilités : soit nous gardions le principe du premier modèle avec un seul vote par ligne et ainsi juste faire de sorte que le programme prenne en compte le fais qu'un votant correspond à quatre lignes consécutives, ou alors considérer les quatre votes sur une seule ligne. Finalement, le modèle retenu pour ce fichier est le modèle de 4 votes par ligne en utilisant des structs.

Pour la réalisation des fichiers oracles, les prédictions dépendent des fichiers d'entrée mais nous avons fait en sorte d'avoir au moins une situation et de victoire et une situation d'égalité pour chaque système. De plus les compteurs et résultats sont prévus avant les fichiers d'entrée ce qui nous évite d'écrire des valeurs aléatoires et d'avoir à les compter manuellement.