Laboratoire 5 : programme de gestion d’un championnat

Les organisateurs d’un tournoi de mini-foot souhaitent un programme facilitant leur travail. Plus précisément, ils souhaitent disposer d’un programme qui :

* Encode les participants d’un tournoi;
* Affiche la liste des participants du tournoi ;
* Affiche les matchs de la prochaine journée ;
* Affiche le calendrier d’un participant ;
* Quitter le programme

# Encoder les participants d’un tournoi

Au début du tournoi, les organisateurs disposent de la liste des participants. Le programme demande aux organisateurs le nombre de participants avant de les encoder dans un tableau. Pour faciliter la suite du programme, ce dernier n’acceptera **que des nombres de participants pairs**.

Comment créer un tableau dont la longueur n’est connu qu’au moment de l’exécution ?

Afin de ne pas générer le même calendrier de matchs pour une même liste de participants, le programme mélange les éléments du tableau.

Idée : permuter N fois deux éléments du tableau pris au hasard.

# Afficher la liste des participants du tournoi

Pour chaque élément du tableau des participants, le programme affiche leur nom.

Conseil : la boucle for[each] est la plus adaptée à ce parcours. Utilisez-la !

# Afficher les matchs d’une journée

Si P est le nombre de participants, pour affronter chaque participant une seule et unique fois, un participant doit jouer P-1 matchs. Les organisateurs souhaitent que les participants jouent un seul match par journée : il y a P/2 matchs à planifier par journée[[1]](#footnote-2).

Pour générer les matchs d’une journée, le programme utilisera l’algorithme du tourniquet à point fixe dont le fonctionnement peut être décrit comme suit :

1. Décaler les éléments du tableau d’une position vers la droite, le dernier élément devenant le premier.
2. Permuter le premier élément et le deuxième, ce dernier revenant à sa position avant décalage.
3. Les p/2 matchs à jouer sont les p/2 couples d’indices (participants[i], participants[participants.length – i – ?]).

## Un exemple

Imaginons un tournoi dont les 6 participants mémorisés dans un tableau :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F |

Appliquons l’algorithme par étape.

1. Décalage

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| F | A | B | C | D | E |

1. Permutation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | F | B | C | D | E |

1. Identification des matchs

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | F | B | C | D | E |

Les matchs à jouer**[[2]](#footnote-3)** pour la première journée sont donc

|  |  |
| --- | --- |
| A | E |
| F | D |
| B | C |

Si le programme part de ce tableau pour générer la deuxième journée, cette dernière aura pour matchs :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | E | F | B | C | D |

Les matchs à jouer pour la deuxième journée sont donc

|  |  |
| --- | --- |
| A | D |
| E | C |
| F | B |

On peut montrer qu’au bout de P-1 itérations, chaque participant a rencontré une et une seule fois tous les autres participants.

# Afficher le calendrier d’un participant

Le programme doit afficher le calendrier de chaque participant qui correspond aux matchs qu’il doit jouer par journée.

Par exemple, les matchs joués par A seront :

Journée 1 : A contre E

Journée 2 : A contre D

Journée 3 : A contre C

Journée 4 : A contre B

Journée 5 : A contre F

Afficher le calendrier d’un participant revient à appliquer P-1 fois l’algorithme expliqué précédemment, **en supposant que le tableau des participants n’a pas été modifié entre-temps** (pas de décalage, ni de permutation).

Idée : pourquoi ne pas mémoriser chaque journée comme élément d’un tableau…à 3 dimensions !

# Quelques détails techniques

Votre programme doit comporter de petites méthodes.

Nous vous demandons au minimum de définir dans une classe **Tableaux** :

* public static void decalerADroite(String[] tableau) qui effectue le décalage à droite comme décrit par l’algorithme ;
* public static void permuter(String[] tableau, int i, int j) qui permute la valeur des élément d’indice i et j du tableau ;
* public static int[][] genererProchainMatch(String[ ] tableau) qui retourne la matrice décrivant les matchs de la prochaine journée.

Les méthodes decalerADroite et permuter devraient être testées. Cependant, elles ne retournent aucune valeur. Pour les tester, il est nécessaire de définir plusieurs tableaux : celui qui sera manipulé par la méthode et celui reflétant le résultat attendu. Les méthodes surchargées Arrays.equals(T[] tableau1, T[] tableau2) permettent de tester l’égalité entre deux tableaux d’une dimension.

Ces tests sont là pour vous aider ! Commencez par les cas de tests simples (pourquoi pas des tableaux de un ou deux éléments) avant de généraliser vos algorithmes.

## Quelques exemples de tests

### decalerADroite

|  |  |
| --- | --- |
| { « a »} | {« a »} |
| {« a », « b »} | {« b »,  «a »} |
| {« a », « b », « c »} | {« c », « a », « b »} |

### permuter

|  |  |  |  |
| --- | --- | --- | --- |
| { « a »} | 0 | 0 | {« a »} |
| {« a », « b »} | 0 | 1 | {« b »,  «a »} |
| {« a », « b », « c »} | 0 | 1 | {« b », « a », « c »} |
| {« a », « b », « c »} | 2 | 0 | {« c », « b », « a »} |

# Que rendre ?

Votre projet Netbeans pour le dimanche 23 novembre avant minuit.

# Améliorations possibles

Votre programme peut gérer les résultats des différents matchs et afficher le classement avant la prochaine journée. A cette fin, on peut prévoir deux nouveaux items :

* Encoder les résultats de la journée ;
* Afficher le classement.

1. On fera l’hypothèse que les organisateurs disposent de suffisamment de terrains pour que les P/2 matchs puissent être joués en une journée. [↑](#footnote-ref-2)
2. Remarquez qu’en cas d’un nombre de participants impairs, le participant au centre du tableau ne joue pas. [↑](#footnote-ref-3)