Rapport de projet :
Simuler des manifestations
ASD2

2022/2023

réalisé par:
Anh NGUYEN
Aymen EL OUAGOUTI

Nantes Université

I. La classe Group (Groupe) :

Choix portés :

Nous avons choisi d'utiliser une liste chaînée avec un enregistrement **Node** pour représenter les personnes d'un groupe. Cette structure de données nous permet d'insérer et de supprimer des personnes en temps constant sans avoir à décaler les autres éléments de la liste. Nous avons ajouté des pointeurs vers la première et la dernière personne de la liste chaînée, qui nous permettront de supprimer la première personne ou d'insérer une nouvelle personne en fin de liste en temps constant.

Nous avons également choisi d'utiliser une table de hachage pour stocker les personnes dans le groupe, la clé de la table est l'identifiant d'une personne et la valeur est un pointeur vers le maillon de la liste chaînée. Cette structure de données nous permet de rechercher une personne par son identifiant en temps constant.

SDA:

+ Création d'un groupe: **Group**(CdC name, CdC color, entier size) -> Group
  Pré : Aucune
+ Destruction d'un groupe: ~Group()
  Pré : Aucune
+ Suppression d'une personne du groupe par ID : removePerson(entier id)
  Pré: la personne portant l'ID donné doit être dans le groupe.
+ Obtention de la taille du groupe: getSize() -> entier
  Pré: Aucune
+ Accès au leader du groupe: getLeader() -> Person
  Pré: il doit y avoir au moins une personne dans le groupe.
+ Accès à la personne par ID : getPerson(entier id) -> Person
  Pré: la personne avec ID donné doit être dans le groupe.
+ Insertion d'une personne : insertPerson(pointeur vers Person person)
  Pré: La personne portant cet ID n'est pas présente dans le groupe auparavant.
+ Suppression d'un leader du groupe: removeLeader()
  Pré: il doit y avoir au moins une personne dans le groupe.
+ Itérabilité : fonctions iterator begin(), iterator end();
  Pré: Aucune

La représentation mémoire:

Exemple d'un maillon contenant 3 personnes:

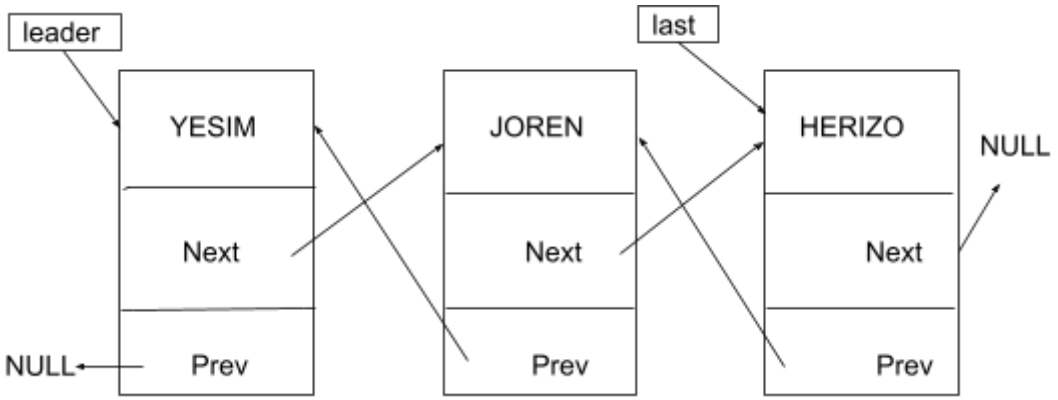
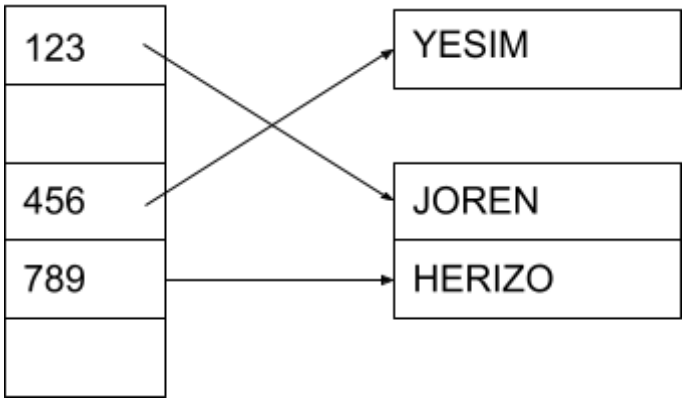


Table de hachage:



Les complexités des opérations:

Opérations	Complexité
Création du groupe vide	$\Theta(1)$
Destruction du groupe	$\Theta(n^2)$
Taille du groupe	$\Theta(1)$
Accès une personne	$\Theta(1)$
Insertion d'une personne	$\Theta(1)$

Suppression d'une personne par ID	Θ(1)
Suppression d'une personne	Θ(1)
Accès au leader du groupe	Θ(1)
Accéder au groupe par iterator begin	Θ(1)
Accéder au groupe par iterator end	Θ(1)

II. La classe Procession (Cortège) :

Choix portés :

Un cortège qui est une liste de groupes avec les opérations mentionnées, nous pouvons créer une classe appelée Cortège qui contient une liste de pointeurs vers des groupes. Nous pouvons utiliser la classe std::list pour stocker les pointeurs de groupes.

Pour trier les groupes par ordre alphabétique des couleurs, on utilise une lambda expression qui compare les deux chaînes de caractères.

Pour trier les groupes par ordre décroissant des tailles, vous pouvez utiliser un tri par insertion (insertion sort).

SDA:

+ Création d'un cortège vide :
  Procession(CdC name) → Processon
  Pré: Aucune
+ Destruction d'un cortège :
  ~Procession();
  Pré: Aucune
+ Insertion d'un groupe : void
  insertGroup(Group* group);
  Pré: le groupe ne doit pas être déjà présent dans le cortège
+ Suppression d'un groupe à partir de son nom :
  removeGroup(CdC name);
  Pré: le nom du groupe doit correspondre à un groupe présent dans le cortège
+ Accès à une personne à partir de son ID :

getGroup(CdC name) ;

Pré: l'ID doit correspondre à une personne présente dans un des groupes du cortège

+ Suppression d'une personne à partir de son ID : void removePerson(entier id)

  Pré: l'ID doit correspondre à une personne présente dans un des groupes du cortège

+ Tri par couleurs des groupes :

  sortColor()

  Pré: Aucune

+ Tri par tailles des groupes :

  sortSize()

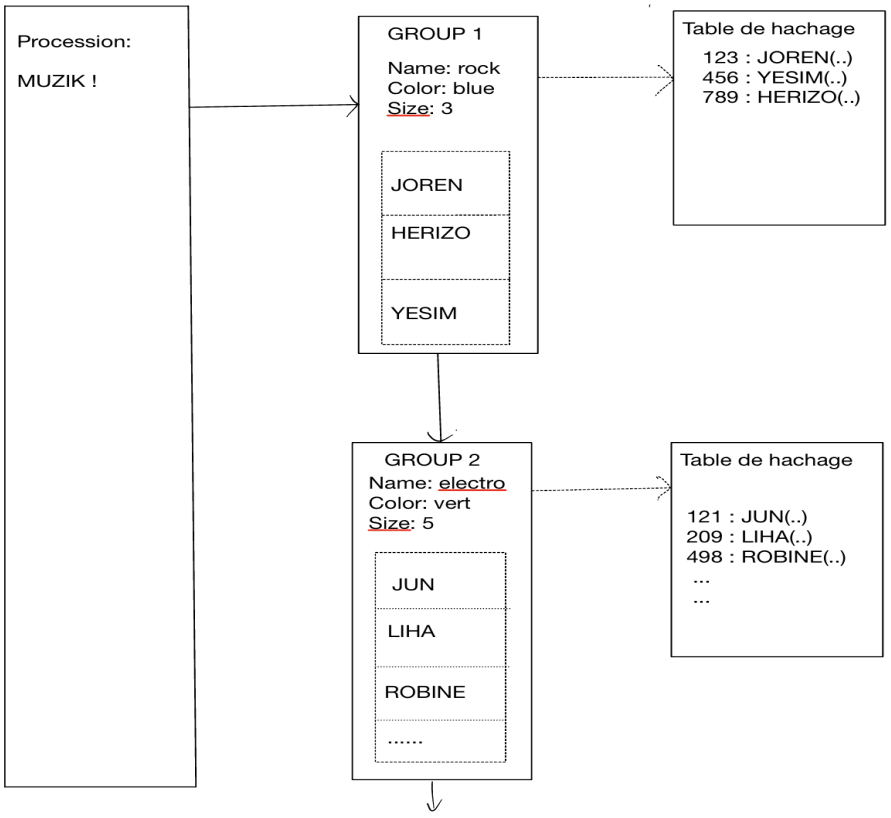
  Pré: Aucune

+ Itération sur les groupes du cortège :

  begin(), end();

  Pré: Aucune

La représentation mémoire:



Les complexités des opérations:

Opérations	Complexité
Création d'un cortège:	Θ(1)
Insertion d'un groupe:	Θ(1)
Suppression d'un groupe par son nom:	Θ(n)
Accès à un personne par son ID:	Θ(nm)
Suppression d'une personne par son ID:	Θ(nm)
Tri par couleur:	Θ(nlogn)
Tri par taille:	$\Theta(n^2)$
Destruction d'un cortège:	Θ(n)
Accéder au groupe par itérateur begin	Θ(1)
Accéder au groupe par itérateur end	Θ(1)

où n est le nombre de groupes dans le cortège et m est la taille maximale d'un groupe.

III. La classe Demonstration (Manif) :

Choix portés :

La structure de données choisie est une liste doublement chaînée circulaire, où chaque nœud représente un groupe et contient des informations sur le nom, la couleur et la taille du groupe, ainsi qu'un pointeur vers le leader du groupe et une liste chaînée simple qui contient les personnes du groupe. Chaque nœud dans la liste est connecté à son prédécesseur et à son successeur, formant ainsi une boucle.

Le choix a été fait de stocker les personnes dans une grille bidimensionnelle pour pouvoir les afficher visuellement. Chaque cellule de la grille est un pointeur vers une personne. Pour déplacer les personnes dans la grille, on actualise leur position à chaque étape de simulation.

SDA:

+ Création d'une manifestation vide :
  Demonstration(entier wid, entier len, Procession *proc) →Demonstration
  Pré: La largeur et la longueur de la manifestation doivent être positives.
+ Destruction d'une manifestation :
  ~Demonstration();
  Pré: Aucune
+ Simulation d'une étape:
  simStage();
  Pré: La démonstration doit être en cours.
+ Test de fin
  hasEnded() → boolean
  Pré: Aucune
+ Accès à une personne à partir de son ID :
  getPerson(entier id) → Person
  Pré: l''ID fourni doit correspondre à une personne existante dans la manifestation.
+ Suppression d'une personne à partir de son ID : void
  removePerson(int id);
  Pré: L'ID fourni doit correspondre à une personne existante dans la manifestation.
+ L'ensemble de leader en train de défiler: void
  std::vector<Person *> getLeaders()
  Pré:Aucune

La représentation mémoire est comme celle de Cortège, parce qu'une manifestation est un cortège qui se déplace.

Les complexités des opérations:

Opérations	Complexité
Création d'une manifestation vide	$\Theta(nm)$

Simulation d'une étape	Θ(nm)
Test de fin	Θ(m)
Accès à une personne à partir de son ID	Θ(n)
Suppression d'une personne à partir de son ID	Θ(n)
L'ensemble de leader en train de défiler:	Θ(nm)
Destruction d'une manifestation	Θ(1)

où n est le nombre total de personnes et m est la taille de la grille

IV.Résultat de simulation:

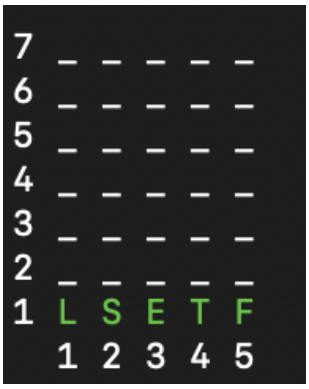
Sim1: simuler une manifestation du début à la fin (toutes les personnes ont fini de défiler)

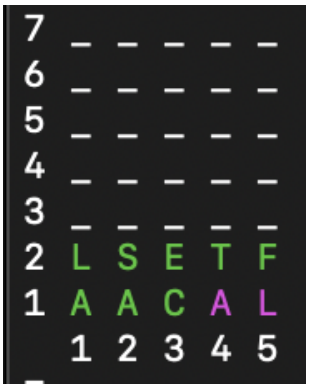
On crée une grille taille 5x7 pour simuler le déroulement d'une manifestation étape par étape jusqu'à ce que toutes les personnes aient fini de défiler.

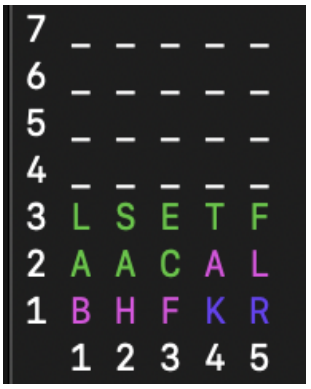
Pour simuler chaque étape de la manifestation, la fonction **simStage()** utilise une boucle pour parcourir les personnes concernées par le changement de position lors de l'étape actuelle. Elle met à jour leur position en appelant la fonction **updatePosition()** pour chaque personne. Ensuite, elle met à jour la grille en plaçant les pointeurs de personnes à leur nouvelle position sur la grille. Si toutes les personnes ont défilé jusqu'à la fin de la grille, la fonction **simStage()** vide la grille et affiche la grille mise à jour une dernière fois avant de se terminer.

En utilisant **substr(0,1)** qui nous permet de récupérer le premier caractère de cette chaîne, c'est-à-dire l'initiale du nom.

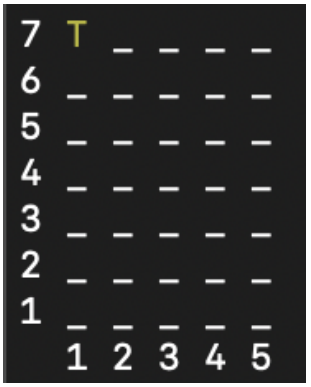
On simule la première étape → la seconde étape → la troisième étape.

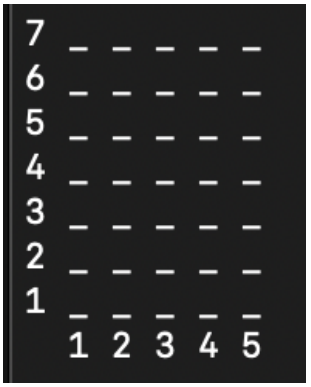






On continue jusqu'à ce que tout le monde est parti, et la dernière étape.





Il ne reste aucune personne dans la manifestation.

On a créé un nouveau group jaune et l'ajouté dans le cortège (et la manifestation) c'est pour ça la dernière personne dans la manifestation est en jaune.

<u>Résultats de simulation avec les temps de calcul:</u>

Le temps de calcul dépendra également de la taille de la manifestation, du nombre de groupes et de personnes dans chaque groupe. Plus la manifestation est grande, plus le temps de calcul sera long.

Exemple:

Pour un groupe de 23 personnes, alors la durée de la simulation : 0.004835 secondes.

Pour un groupe de 100 personnes, alors la durée de la simulation : 0.019226 secondes.
Pour un groupe de 5000 personnes, alors la durée de la simulation : 16.6049 secondes
Pour un groupe de 10000 personnes, alors la durée de la simulation : 61.0448 secondes.

Sim2: Problème :

Avec les tests qu'on a essayé de réaliser, le retrait d'une personne d'une manifestation la retire complètement du cortège et la détruit, on a donc une structure qui stocke l'id et la position d'une personne retirée ainsi qu'une liste qui contient ces personnes. Alors à la suppression, un drapeau est levé et l'exécution ne se produit pas comme dans une situation normale. On dérive vers un cas et on décale les positions de toutes les personnes se situant après la personne retirée.
Ici on retire la personne avec ID = 0, donc "G"
Le décalage est bien fait, mais un bug s'est incrusté et on n'est pas arrivé à le repérer, la case 1:5 reste toujours vide et donc chaque personne qui s'incruste dans la grille n'est pas visible à son apparition mais plutôt à l'étape d'après comme on peut le remarquer pour "M".
Ensuite en faisant une boucle pour itérer avec le retrait de chaque personne qui arrive à la première ligne (1), une exception concernant les IDs est levée et le programme s'arrête.

En bref, le problème est notre utilisation des IDs comme indicateurs pour les positions des personnes.

```
electro : G-S-J-G-N-M-E-I-
hip-hop : M-L-A-A-L-
rock : R-S-E-M-A-O-

Person removed
5 _ _ _ _ _
4 _ _ _ _ _
3 _ _ _ _ _
2 _ _ _ _ _
1 S J G N _
  1 2 3 4 5
5 _ _ _ _ _
4 _ _ _ _ _
3 _ _ _ _ _
2 S J G N M
1 E I M L _
  1 2 3 4 5
5 _ _ _ _ _
4 _ _ _ _ _
3 S J G N M
2 E I M L A
1 A L R S _
  1 2 3 4 5
5 _ _ _ _ _
4 S J G N M
3 E I M L A
2 A L R S E
1 M A O _ _
  1 2 3 4 5
5 S J G N M
4 E I M L A
3 A L R S E
2 M A O _ _
1 _ _ _ _ _
  1 2 3 4 5
5 E I M L A
4 A L R S E
3 M A O _ _
2 _ _ _ _ _
1 _ _ _ _ _
  1 2 3 4 5
```