Classe simple *IntervalSet*
Spécifications

Auteur 1 – Auteur 2

Introduction

La classe *IntervalSet* a pour rôle de gérer le stockage, la manipulation et l'affichage d'un nombre arbitraire d'intervalles disjoints ajoutés par l'utilisateur.

Première partie
Généralités

1 Définitions

Intervalle : Un intervalle est représenté par deux entiers a et b (les *bornes*) avec $a \leq b$, et symbolise l'ensemble des réels (appelés *éléments*) compris entre a et b inclus.

Intervalles disjoints : Des intervalles $[a, b]$ et $[c, d]$ sont dits *disjoints* s'ils ne partagent aucun élément (pas même les bornes), c'est-à-dire si $c > b$ ou $a > d$.

***IntervalSet* :** Objet permettant le stockage et la manipulation d'un ensemble trié d'intervalles tel que ces intervalles sont tous disjoints.

Union de deux intervalles : Si u et v sont deux intervalles *non disjoints*, nous appelons l'*union de u et v* le nouvel intervalle contenant à la fois les éléments de u et ceux de v.

Intersection de deux intervalles : Si u et v sont deux intervalles *non disjoints*, nous appelons l'*intersection de u et v* le nouvel intervalle contenant uniquement les éléments communs à u et à v.

Union de deux *IntervalSet* A et B : Nouvel *IntervalSet* contenant le regroupement des intervalles de A et B qui sont disjoints plus l'union de leurs intervalles non disjoints.

Intersection deux *IntervalSet* A et B : Nouvel *IntervalSet* ne contenant que les intersections des intervalles de A et B qui sont non disjoints.

2 Choix généraux

Tri des éléments : Les éléments contenus dans l'*IntervalSet* sont *triés*[1] en permanence, ce qui a de l'importance pour l'affichage et la récupération/suppression d'un intervalle.

Indexation des intervalles : Les méthodes de récupération et de suppression d'un intervalle se basent sur l'index interne de cet élément (les indexes allant de 1 à la taille *Count* de l'*IntervalSet*). Lors de l'ajout ou de la suppression d'un intervalle, les indexes des éléments suivant cet élément sont modifiés en conséquence.

Intervalles consécutifs : Deux intervalles $[a, b]$ et $[b+1, c]$ seront laissés tels quels et ne seront pas fusionnés lors des opérations sur les *IntervalSet* (notamment lors d'une *Union* ou d'une *Intersection*).

Ajout d'un *IntervalSet* à un autre *IntervalSet* : la fonction d'ajout (*Add*) qui prend en paramètre un *IntervalSet* « *is* » ajoute les éléments de *is* si et seulement si *tous* ses intervalles peuvent être ajoutés.

1. Dans l'ordre croissant

Deuxième partie
Spécification des méthodes

Construction d'un *IntervalSet* par copie d'un autre *IntervalSet*

Construit un nouvel *IntervalSet* à partir d'un *IntervalSet* passé en argument. Le nouvel *IntervalSet* contient des copies de tous les intervalles de l'*IntervalSet* original et n'est pas lié à celui-ci, i.e. il n'est pas affecté par les modifications ultérieures de celui-ci (et inversement).

Ajout d'un intervalle

Tente d'ajouter (à l'objet courant) un intervalle dont les bornes a et b sont passées en argument. Si l'intervalle passé en argument est valide (c'est-à-dire si $a \leq b$) et si cet intervalle est disjoint de tous les autres intervalles que contient déjà l'*IntervalSet*, alors l'ajout réussit et la méthode renvoie vrai. Sinon, l'ajout échoue et la méthode renvoie faux.

Construction d'un *IntervalSet* par union de deux *IntervalSet*

Renvoie un nouvel *IntervalSet* constituant l'union de l'*IntervalSet* sur lequel est appelée la méthode, et de l'*IntervalSet* passé en argument. Les objets originaux ne sont pas affectés par l'appel à cette méthode. Le nouvel *IntervalSet* contient ses propres intervalles et n'est pas affecté par les modifications ultérieures des objets originaux.

Construction d'un *IntervalSet* par intersection de deux *IntervalSet*

Renvoie un nouvel *IntervalSet* constituant l'intersection de l'*IntervalSet* sur lequel est appelée la méthode et de l'*IntervalSet* passé en argument. Les objets originaux ne sont pas affectés par l'appel à cette méthode. Le nouvel *IntervalSet* contient ses propres intervalles et n'est pas affecté par les modifications ultérieures des objets originaux.

Troisième partie
Tests unitaires

Formalisme adopté

$\{\ [a_1,\ b_1],\ \ldots,\ [a_n,\ b_n]\ \}$ représente un objet IntervalSet qui contient effectivement les intervalles $[a_i,\ b_i]$ pour i de 1 à n, et avec a_i et b_i des entiers. Par définition, ces intervalles ne peuvent pas se chevaucher (dans le cas contraire, l'objet viole la spécification d'IntervalSet).

$<[a_1,\ b_i],\ \ldots,\ [a_n,\ b_n]>$ représente un objet IntervalSet, initialement vide, dont on aura appelé la méthode add n fois avec les intervalles $[a_i,\ b_i]$, pour i de 1 à n, et avec a_i, b_i des entiers. Ces intervalles peuvent se chevaucher, puisqu'ils ne concernent que l'appel à la procédure d'ajout (*add*), et non le résultat de celui-ci. Les différents appels à *add* devront toujours être effectués avec les intervalles dans l'ordre présenté (de gauche à droite, donc).

$|obj|$ représente le résultat de l'appel de *count* sur l'objet obj, obj pouvant être l'un des deux formalismes précédents.

$obj_1 \cup obj_2$ représente l'union des deux IntervalSets obj_1 et obj_2.

1 Nombre d'élements : méthode *Count*

T-1 : Nombre d'éléments d'un nouvel *IntervalSet* vide

Après la création d'un objet IntervalSet vide, auquel on n'ajoutera aucun intervalle, l'appel à Count devra renvoyer 0.

T-2 : Nombre d'éléments d'un *IntervalSet* vidé

On créera un objet IntervalSet vide, avant de lui ajoutera un certain nombre d'intervalles. On les supprimera ensuite tous sans exception. Un appel à *Count* devra alors renvoyer 0.

T-3 : Nombre d'éléments d'un *IntervalSet* non vide

Après la création d'un objet IntervalSet vide, on lui ajoutera un certain nombre d'intervalles. On vérifiera que la valeur renvoyée correspond bien en effet au nombre d'intervalles présents dans l'objet, en prêtant attention à la règle de non ajout en cas de conflit entre un intervalle présent et l'intervalle à ajouter.

Jeux de données

$$|<[0,1],[2,3],[4,7]>| = 3$$
$$|<[0,1],[1,3],[4,7]>| = 2$$
$$|<[0,0],[2467,16384],[9987,10]>| = 2$$
$$|<[0,0],[0,1],[-1,3],[3,4],[5,7],[22,57],[14,32],[64,7564],[32768,65536],[128493,5739928]>| = 7$$

On pourra également lancer un test composé de la façon suivante : on appelle *add* n fois (n étant grand, par exemple 10^5) avec des intervalles tous disjoints, et on vérifie que le retour de *count* est bien n.

T-4 : Nombre d'éléments d'un *IntervalSet* vide avec suppression

On créera un *IntervalSet* vide. On fera ensuite un (ou plusieurs) appel(s) à sa méthode *Remove*, puis on exécutera sa méthode *Count*. Quelque soit le nombre de fois que l'on aura appelé *Remove*, un appel à *Count* devra toujours renvoyer 0.

2 Réunion : méthode *Union*

T-5 : Union d'IntervalSet vides

On créera deux *IntervalSet*, qui seront laissés vides. L'union de ces deux objets devra renvoyer un *IntervalSet* vide également.

T-6 : Union avec un *IntervalSet* vide

On créera deux *IntervalSet*. Dans le premier, on ajoutera un jeu d'intervalles quelconques, tandis que le second sera laissé vide. On exécutera ensuite l'union de ces deux *IntervalSet* ; l'objet résultant devra être égal à l'*IntervalSet* non vide précédemment créée. Attention, il faudra réaliser deux unions pour ce test : celle où l'objet vide est passé en paramètre à la méthode *Union* de l'objet non vide, mais également celle où la situation est inversée (elles ne sont pas équivalentes).

T-7 : Union d'ensemble complètement disjoints

On créera deux *IntervalSet* auxquels on ajoutera n[2] intervalles tous complètement disjoints. L'appel de *Count* sur l'union de ces deux *IntervalSet* devra renvoyer n.

2. Au total, c'est à dire que si le premier (respectivement le second) *IntervalSet* possède n_1 intervalles (respectivement n_2), on a $n = n_1 + n_2$

Jeux de données

N'importe quel ensemble de n intervalles disjoints (y compris les bornes), qui seront ajoutés indifféremment à l'un ou à l'autre *IntervalSet* (ajouter tous les intervalles à un seul *IntervalSet* n'est pas un problème, mais ce cas a normalement déjà été testé). Pour faire le test, on pourra simplement ajouter des intervalles disjoints, puis comparer le résultat de *Count* sur l'objet union avec la somme des résultats de *Count* sur les 2 *IntervalSet* de base.
Jeu 1 :

$$\left|\Big\{[-64,-32],[0,1],[4,9],[102,172],[1025,5760],[272000,2004576]\Big\}\right.$$
$$\left.\cup\Big\{[2,2],[10,17],[27,54],[7902,25722],[3001757,4996125]\Big\}\right| = 11$$

Jeu 2 :

$$\left|\Big\{[10\cdot i, 10\cdot i],[10\cdot i+3, 10\cdot i+5]\Big\} \cup \Big\{[10\cdot i+1, 10\cdot i+2],[10\cdot i+6, 10\cdot i+9]\Big\}\right| = 4\cdot n$$

pour i variant de 1 à n (chaque objet possède ainsi $2\cdot n$ intervalles, construits selon les schémas donnés ci-dessus).

T-8 : Union d'un ensemble avec lui même

Après la création d'un *IntervalSet*, on lui ajoutera un grand nombre d'intervalles. Puis on appellera la méthode Union de cet *IntervalSet*, avec lui-même comme argument (c'est à dire qu'on fera l'union de cet *IntervalSet* avec lui même). On appellera alors count sur le résultat de cette union, et ce nombre d'éléments devra être égal à celui du premier *IntervalSet*. On pourra également appeler la méthode *Display* de ces deux *IntervalSet*, et vérifier que la sortie est bien rigoureusement identique.

T-9 : Union d'intervalles particuliers

On créera deux *IntervalSet* auxquels on ajoutera des intervalles calculés pour être, ou non, disjoints. On vérifiera ensuite que les intervalles obtenus dans l'union correspondent bien à ce qui doit résulter, par exemple avec l'affichage (*display*) de l'objet résultat, ou encore à des appels à *GetInterval*.

Jeux de données

Jeu 1 : On pourra ajouter un certain nombre d'intervalles construits sur le schéma suivant : $[i\cdot 4, i\cdot 4+3]$ dans le premier *IntervalSet* et $[i\cdot 4+1, i\cdot 4+4]$ dans le second, pour i variant de 1 à n. Ces intervalles se recoupent alternativement entre chaque objet, et devront former un seul grand intervalle dans l'union, avec pour bornes $i\cdot 4$ et $(n+1)\cdot 4$ (un appel à *Count* sur l'union devra donc renvoyer 1).

Jeu 2 : On pourra également construire 2 *IntervalSet* sur le même principe que l'exemple précédent, mais cette fois avec les intervalles qui ne se recoupent que deux à deux. On utilisera un schéma tel que celui-ci : $[i\cdot 3, i\cdot 3+1]$ dans le premier objet, et $[i\cdot 3+1, i\cdot 3+2]$ dans le second − là encore pour i variant de 1 à n. L'union de ces 2 objets doit contenir n intervalles.

Jeu 3 :

$$\left|\Big\{[-64,-32],[0,1],[4,9],[102,172],[1025,5760],[272000,3004576]\Big\}\right.$$
$$\left.\cup\Big\{[-32,2],[9,17],[109,125],[7902,25722],[3001757,4996125]\Big\}\right| = 6$$