LIU Bin
22116863
IMPAIRES
happybin1989@gmail.com

Rapport de TP 2
Hash Linear Probing et Hash Join

1. Hash Linear Probing

Objectif du TP

1. Ne construire pas de collections Java car les tableaux doit être en taille de Hashmap ('M = 11') . Les clés sont stockés dans le tableaux[] de 'char' et [] 'int' pour les valeurs .
Le tableau qui sera hashé vers hashmap est définié par char Key [N] et int Values [N] (ici, N < M. je prends la taille N = 10).

2. Pour le programme principale, il faut uniquement appliquer des boucles "while". Généralement, ils sont appliqués sur les initialisations et les affichages des résultats.

3. Créer une interface 'HashProbing',qui contiens Put (key, value) ,Get (key) et Remove (key) pour la mise en oeuvre dans la class .

4. Créer une classe HashLinearProbing implémente des fonctions de l'interface 'HashProbing', les objectifs des fonctions internes sont comme ci_dessous:

- Hash(key) --> returner la valeur de '*hashcode()*' de 'key' modulo la taille 'M' ,qui sera utilisé sur le formule 'hi (x) = (h(x) + i) mod M' pour déterminer sa position dans hashmap (réalisés par les fonctions suivantes).
- Put (key, value) --> insérer les paires '(key, value)' dans le hashmap selon le retour de la valeur de 'Hash(key)'
- Get (key) --> retrouver les paires '(key, value)' dans le hashmap selon le 'key'
- Remove (key) --> enlever les paires '(key, value)' et rehacher des autres paires
- printHashTable (key, value) --> afficher les paires '(key, value)'

5. Créer une classe 'MainHashLinearProbing' qui fait un « main » client afin de réaliser toutes les fonctionnalités.

Algorithme:

Put (key, value):
M = # bucket entries //la taille de hashmap
index = hash (key) // généré par hashcode()
While Key [index) != empty

//parcourir le tableu de hashmap, si la position n'est pas vide , il veut donner un nouveu index pour poser '(key, value)'

index = (index + 1) % M

End while

Key [index] = key //dans cette position,il met les paires '(key, value)'

Values [index] = value

Get (key):

M = # buckets

index = hash (key)

valueToReturn = -1 // si le 'key' n'exite pas dans le hashmap ,il retourne -1 par défaut.

While Key [index] != key and Key [index] != empty

//comme dans 'Put (key, value)', sa règle de dépôts a bien suivi le formule 'index = (index + 1) % M', sa récupération de 'key' et 'values' doit aussi le respecter

index = (index + 1) % M

End while

If (Key [index] = key)

//quand je trouve la position où le key était, il me donne sa bonne 'value' correspondante

valueToReturn = Values [index]

Return valueToReturn

Remove (key):

M = # buckets

index = hash (key)

While Key [index) != key and Key [index) != empty

index = (index + 1) % M

End while

Key [index] = ' ', //si j'enleve un paire '(key, value)',

Value [index] = 0 // mais dans le tableau du type 'int', '0' par défaut. C'est pas possible d'y mettre un nil

// rehash : comme un paire est supprimé, les paires successives de cette paire '(key, value)' seront arrangés dans des autres positions selon le formule 'index = (index + 1) % M', il fait le processus comme ci_dessous

index = (index + 1) % M

While Key [index) != empty

savedKey = Key [index],

savedValue = Value [index]

Key [index] = ' '

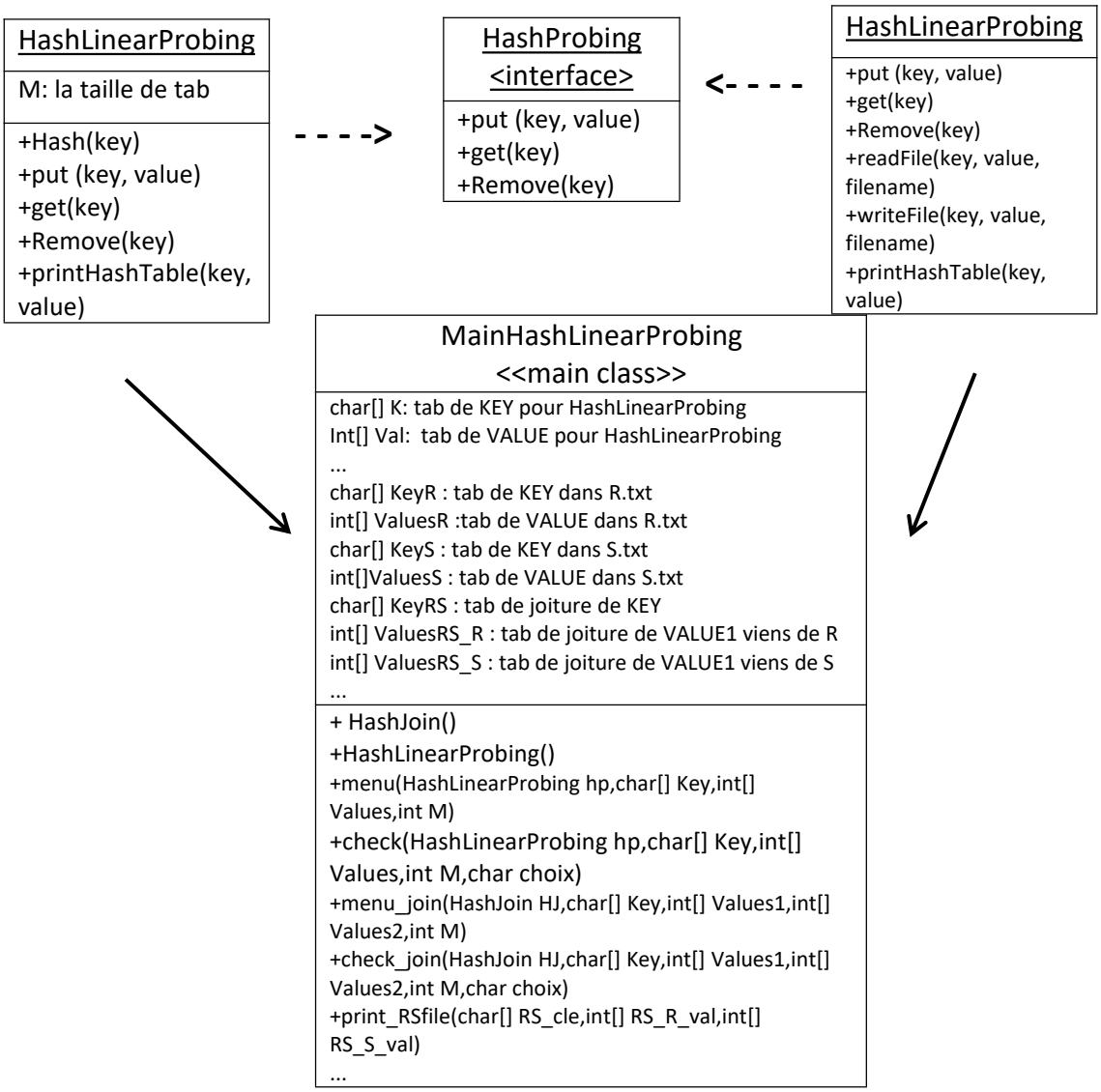
Value [index] = 0

Put (savedKey , savedValue)

index = (index + 1) % M

End while

UML (Hash Linear Probing et Hash Join)



Comme il y a encore des fonctions supplémentaires dans la 'main' class , la 'main' class ne peut pas déssiner la relation 'Dependency' directement vers l'interface 'HashProbing'. (ça veut dire que '*HashProbing hp = new HashLinearProbing()*' ne peut pas être accepté.Il faut '*HashLinearProbing hp = new HashLinearProbing()*')

Expérimentation

Key (char)	Value (int)
B	0
O	1
E	2
P	3
V	4
L	5
X	6
N	7
K	8
M	9

Hash(Key) (int)
0
2
3
3
9
10
0
1
9
0

	Key (char)	Value (int)
0	B	0
1	X	6
2	O	1
3	E	2
4	P	3
5	N	7
6	K	8
7	M	9
8		
9	V	4
10	L	5

Après faire le '*hashcode()*' modulo la taille de hashmap 'M', j'obtiens un relation de 'Hash(Key)'. Quand indiquer l'index de dépôts de (key, value) dans le hashmap.

Je fais 'Scanner scan = new Scanner(System.in)' pour les gérer plus simplement. Cela me permet de faire des choix à activer plusieurs de tests sur mes mêmes fonctions. Le programme s'excute et montre dans le console:

```
--------------------------------
Taper le num:
1 -> HashLinearProbing ; 2 -> HashJoin
Votre choix: ------>
1


--------------- Hash Linear Probing ---------------

Hash Table:
[ 'B',0 ]
[ 'X',6 ]
[ 'O',1 ]
[ 'E',2 ]
[ 'P',3 ]
[ 'N',7 ]
[ 'K',8 ]
[ 'M',9 ]
[ ' ',0 ]
[ 'V',4 ]
[ 'L',5 ]
```

```
Après tous les insertion dans hash map:
--------------------------------
Taper le num:
1 -> get ; 2 -> remove
Votre choix: ------>
1
Taper key à chercher selon Hash Table au_dessus: ------>
V
Get the value of 'V' : 4
Vous voulez faire des autres actions: taper y, sinon taper n'importe quoi !!!
Y
--------------------------------
Taper le num:
1 -> get ; 2 -> remove
Votre choix: ------>
2
Taper key à enlever selon Hash Table au_dessus: ------>
L

Hash Table:
[ 'B',0 ]
[ 'X',6 ]
[ 'O',1 ]
[ 'E',2 ]
[ 'P',3 ]
[ 'N',7 ]
[ 'M',9 ]
[ ' ',0 ]
[ ' ',0 ]
[ 'V',4 ]
[ 'K',8 ]

Vous voulez faire des autres actions: taper y, sinon taper n'importe quoi!!!
IQIO
------ SORTIE ------
```

Si je continue l'action et veux supprimer un autre élément, la fonction va procédze sur un tableau sans 'L'.

2. Hash Join

Objectif du TP

(1) La jointure des tableaux

1. Avant de la jointure, les deux tableaux d'entrée R et S doivent être triés pas la valeur de 'Key' tout d'abord.

2.La jointure des deux tableaux R et S sort le résultat dans RS.

3. Comme TP nous demande que la class 'HashJoin' doit réaliser une jointure par hachage en utilisant l'interface HashProbing, il faut implémenter des structures de *Put (key, value), Get (key)* et *Remove (key)*. Ses arguments doivent y entrer des tableaux[] en taille (pas de collections). Donc, il faut les définir en mêmes formes pour continuer cette partie du programme . (char Key [N] et int Values [N])

4. Après la sortie de la jointure et sa conjonction des 'Values', je définie un tableau 'char KeyRS [N]' et deux tableaux 'int ValuesRS_R[N]' qui viens de R.txt et 'int ValuesRS_S[N]' qui viens de S.txt. (voir la figure ci_dessous).

5. Finalement, la jointure sortie un seul fichier RS.txt .

6. Get (key) permet de lire le 'key' cherché et ses deux 'values' à partir du fichier RS.txt. Ils peuvent être des duplications.

7. Remove (key) permet de lire le 'key' avec ses deux 'values' à partir du fichier RS.txt et d'enlever quelques éléments (dans RS, un élement comprends un' key' et deux 'values')

Algorithme:

Put (key, value):

```
//cette fonction est justement pour implémenter 'hashjoin(key,value1,value2)'
KeyRS[index] = key ;
ValuesRS_R[index] = valueRS_R ;   //la 'value' viens de R.txt
ValuesRS_S[index] = valueRS_S ;   //la 'value' viens de S.txt
```

Get (key):

```
valueToReturn[][] = new int[10][2]; //Après la jointure de hash,il exite des duplication. Et chaque duplication contiens deux vaules .Afin de retourner tous les 'values', j'ai créé un tableau en dimension 2 à les stocker.
index = 0;
i = 0;
while((index < KeyRS.length) && KeyRS[index])!= empty)
 //parcourir le tableau du jointure
        if (KeyRS[index] == key)   // si trouver le key, je met les 2 'values' de ce 'key' dans 'valueToReturn[][]'
```

```
                valueToReturn[i][0] = ValuesRS_R[index];
                valueToReturn[i][1] = ValuesRS_S[index];
                i ++;
        index ++;
End while
valueToReturnLimit[][]  = Arrays.copyOf(valueToReturn, i) ; //pour diminuer la taille du tableau 'valueToReturnLimit[][]'
 return valueToReturnLimit;
```

Remove (key):

```
index = 0;
while(index < KeyRS.length)
        if (KeyRS[index] == key )
                KeyRS[index] = ' '; // Comme dans la fonction 'Remove (key):' de 'HashLinearProbing', ils sont enlevés et je met ' ' pour le type de 'char' et 0 pour 'int'
                ValuesRS_R[index] = 0;
                ValuesRS_S[index] = 0;
        index ++;
End while;
return KeyRS; // retourner un nouveau tableau après le supprimer
```

hashjoin(key,value1,value2):

```
i=0;
j=0;
k=0;
Vector<Character> kRS = new Vector<Character>();    //Comme je ne sais pas la taille de tableau , 'Vector' me permet d'éviter ce problème.
Vector<Integer> valRS_R = new Vector<Integer>();
Vector<Integer> valRS_S = new Vector<Integer>();
Arrays.sort(R); // trier des éléments du tableau avant la jointure
Arrays.sort(S);
while( i < R.length && j < S.length )
        if( R[i] == S[j] )    // if R(i) = S(j)
                k = j;
                while( (i < R.length) && (R[i] == S[j]) ) // while R(i) = S(j)
                        while(( j < S.length) && (R[i] == S[j])) // while R(i) = S(j) then
                                kRS.add(R[i]);
                                valRS_R.add(ValuesR[i]);
                                valRS_S.add(ValuesS[j]);
                                j++;
                        End while;
                i++;
                j = k;
        else if( R[i] > S[j] ) // if R(i) > S(j) then j=j+1
                j++;
        else if( R[i] < S[j] ) // else R(i) < S(j) then i=i+1
                i++;
```

```
End while;
t = 0;
while(t < kRS.size())
        put(KeyRS, ValuesRS_R, ValuesRS_S, kRS.elementAt(t) , valRS_R.elementAt(t) , valRS_S.elementAt(t), t); // je met tous les éléments de 'vector' dans des tableau en taille de la dernier valeur de 't'. car dans 'Get (key)' et 'Remove (key)',ils ont besoin des tableaux en taille exact.
        t++;
End while;
return kRS.size();
```

Expérimentation

KEYS of R: ['A' 'B' 'G' 'J' 'U' 'K' 'E' 'Z' 'V' 'B']
VALUES of R: [12 16 20 19 17 15 14 22 26 28]

KEYS of S: ['B' 'U' 'E' 'K' 'X' 'V' 'N' 'B' 'M' 'U']
VALUES of S: [2 1 8 9 11 3 6 7 13 5]

R.txt	
Key (char)	Value (int)
A	12
B	16
B	20
E	19
G	17
J	15
K	14
U	22
V	26
Z	28

S.txt	
Key (char)	Value (int)
B	2
B	1
E	8
K	9
M	11
N	3
U	6
U	7
V	13
X	5

RS.txt		
Key (char)	Value -R (int)	Value -S (int)
B	16	2
B	16	1
B	20	2
B	20	1
E	19	8
K	14	9
U	22	6
U	22	7
V	26	13

Comme 'HashJoin' et 'HashLinearProbing' sont dans le même projet, je l'ai mis dans la main class 'MainHashLinearProbing' et ai appliqué aussi 'Scanner scan = new Scanner(System.in)' pour plusieurs de choix des actions.

```
--------------------------------
Taper le num:
1 -> HashLinearProbing ; 2 -> HashJoin
Votre choix: ------>
2


--------------- Hash join ---------------
KEYS of R: [ 'A' 'B' 'G' 'J' 'U' 'K' 'E' 'Z' 'V' 'B' ]
VALUES of R: [ 12 16 20 19 17 15 14 22 26 28 ]

KEYS of S: [ 'B' 'U' 'E' 'K' 'X' 'V' 'N' 'B' 'M' 'U' ]
VALUES of S: [ 2 1 8 9 11 3 6 7 13 5 ]

Vérifiez la jointure dans la fonction de 'hashjoin':
{ B 16 2 }
{ B 16 1 }
{ B 20 2 }
{ B 20 1 }
{ E 19 8 }
{ K 14 9 }
{ U 22 6 }
{ U 22 7 }
{ V 26 13 }
Après la jointure de hash, je sors des keys et values en format de tableau:
KEYS of RS dans le tableau : [ 'B' 'B' 'B' 'B' 'E' 'K' 'U' 'U' 'V' ]
VALUES_1 of RS relevant to R dans le tableau: [ 16 16 20 20 19 14 22 22 26 ]
VALUES_2 of RS relevant to S dans le tableau: [ 2 1 2 1 8 9 6 7 13 ]
--------------------------------
```

```
--------------------------------
Taper le num:
1 -> get ; 2 -> remove
Votre choix: ------>
1
Taper key à chercher selon Hash Table au_dessus: ------>
U

Retirer KEYS of RS dans RS.txt : [ 'B' 'B' 'B' 'B' 'E' 'K' 'U' 'U' 'V' ]
Retirer VALUES_1 of RS relevant to R dans RS.txt: [ 16 16 20 20 19 14 22 22 26 ]
Retirer VALUES_2 of RS relevant to S dans RS.txt: [ 2 1 2 1 8 9 6 7 13 ]

Trouver 2 elements de U:
U: 22 6
U: 22 7
Vous voulez faire des autres actions: taper y, sinon taper n'importe quoi !!!
Y
--------------------------------
Taper le num:
1 -> get ; 2 -> remove
Votre choix: ------>
2
```

```
Taper key à enlever selon le Table de jointure au_dessus: ------>
B
[ ' ' ' ' ' ' ' ' 'E' 'K' 'U' 'U' 'V' ]
[  0  0  0  0  19  14  22  22  26 ]
[  0  0  0  0  8  9  6  7  13 ]
Après enlever l'element de B :

Retirer KEYS of RS dans RS.txt : [ 'E' 'K' 'U' 'U' 'V' ]
Retirer VALUES_1 of RS relevant to R dans RS.txt: [ 19 14 22 22 26 ]
Retirer VALUES_2 of RS relevant to S dans RS.txt: [ 8 9 6 7 13 ]
Vous voulez faire des autres actions: taper y, sinon taper n'importe quoi!!!
SDF
------ SORTIE ------
```

3. Test Unitaire

1. Ajouter Junit4 dans le build path et teste chaque méthode avec le valeur souhaitée.
2. Ces tests unitaires doivent repecter des test indépendant (sans rappel de l'autre méthode de class)

5. Difficulités envisagés

Ce TP combine les réalisations des 2 opérations 'HashLinearProbing' et 'HashJoin' , tous les deux class implémentent la même interface. Car les définitions des tableaux[] sur 'hashLinearProbing' sont requis par la condition de fixer la taille des tableaux[] , 'Vector' n'est pas permis de stocker des données pour faire le Hash join dan sce TP. Car la jointure ne peut pas déterminer la taille des tableaux[] que je veut définis, c'est possible de rencontrer des problèmes comme le débordement du mémoire etc. Donc , je trouve un méthode '*Arrays.copyOf(tab.length) ;*'. Et je redéfinis des nouveaux tableaux[] avec ses tailles exacts et transforme 'Vector' en tableaux [] lorsque je compte le nombre de '*Vector.add(...)*' pour préciser la taille du tableau[].

Après manipuler des actions, je préfère de retirer plusieurs des tableaux[] à partir d'une fonction . Mais la fonction ne retourne qu'un seul tableaux[]. Je met donc des tableaux[] comme les arguments, qui permet les tableaux[] de transmettre ses changements dans la main class. J'ai donc amélioré la structure de l'interface 'HashProbing' et ai mis quelques arguments. Si comme par exemple 'HashLinearProbing' n'a pas besion de quelques arguments, je met null , ' ' ou 0 dans cette position (ça depends du type d'argument) lors de la rappel de cette fonction dans la main class.

Si je veut retourner plusieurs de valeurs, je propose un méthode de retourner un tableau[] qui conserve un , deux ou trois.. valeurs à retirer.

6. Conclusion

Ce TP me permet de bien comprendre des algorithmes sur HashLinearProbing et Hash Join. Et ce travail m'entreîne sur des divers applications et transfomations du tableu[] pour réaliser mon désir.