POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

Rapport TP2

INF2010 : Structure de donnée et algorithme

Bakashov, Marsel - 2147174

Chowdhury, Rasel - 2143023

17 octobre 2022

Table des matières

Présentation de l’objectif du laboratoire

L’objectif de ce laboratoire en question était de tester l’efficacité de trois tables de hachage : la table à sondage linéaire, la table à sondage quadratique ainsi que la table à dispersement double. Pour ce faire, il fallait tout d’abord les concevoir. C’était apres tout, la base fondamentale du travail. Une fois cette étape réalisée, 3 listes d’items de type << Integer>>, *(<< **Array**>>)* ayant des grandeurs différentes soit de 15, de 60 et de 150 ont été teste sur les trois tables de hachage. Le nombre de collisions, le temps d’exécution de même que la taille du tableau étaient des facteurs prises en compte pour déterminer leur efficacité.

Résultat

a) Nombre de conflits

Linear Probing Hash Table

Array 1 :

Positions actives : 3 4 10 11 13 16 17 19 29 34 35 37 39 43 45

Array 2 :

Positions actives : 7 10 16 19 21 24 25 29 31 32 39 42 47 51 52 53 54 55 57 72 76 77 78 83 84 86 87 92 94 97 98 100 111 115 118 119 120 130 133 139 140 142 152 164 172 173 174 175 176 177 178 181 182 183 186 187 193 194 195 196

Array 3 :

Positions actives : 1 2 10 12 18 19 22 23 24 26 29 31 32 36 38 52 55 56 59 60 61 63 65 66 67 68 71 72 74 76 83 98 101 102 103 104 105 106 107 108 110 111 112 113 114 116 117 118 120 121 123 125 127 129 131 133 134 135 137 149 150 153 154 162 163 165 175 176 179 180 185 187 192 196 197 198 202 203 205 206 207 208 209 213 216 232 234 239 242 244 245 253 254 255 256 257 261 264 268 272 273 274 276 278 279 284 289 294 295 306 308 313 315 316 317 318 322 334 337 338 339 340 343 344 345 346 347 348 349 350 353 354 356 359 361 362 364 365 367 369 372 373 374 375 376 377 380 382 383 389

Table de hachage linéaire

Taille du tableau	Taille minimale (amas)	Taille maximale (amas)	Taille moyenne (amas)	Nombre de collisions	Temps d'exécution (ns)
15	2	2	2	7	61500
60	2	7	2.85	51	96800
150	2	8	3.44	151	371100

Quadratic Probing Hash Table

Array 1:

Positions actives: 3 4 10 11 13 16 17 19 29 34 35 37 39 43 45

Array 2:

Positions actives: 7 10 16 19 21 24 25 29 31 32 39 42 47 51 52 53 54 55 57 72 76 77 78 83 84 86 87 92 94 97 98 100 111 115 118 119 123 130 133 139 140 142 152 164 172 173 174 175 177 178 181 182 183 186 187 193 194 195 196

Array 3:

Positions actives : 1 2 10 12 18 19 22 23 24 26 29 31 32 36 38 52 55 56 59 60 61 63 65 66 67 68 71 72 74 76 83 87 98 101 102 103 104 105 106 107 110 111 112 113 114 115 116 117 118 119 120 121 122 127 129 131 133 134 135 137 149 150 153 154 162 163 165 175 176 179 180 185 187 192 196 197 198 202 203 205 206 207 208 209 213 216 232 234 239 242 244 245 253 254 255 256 259 261 264 268 272 273 274 276 278 279 284 289 294 295 306 308 313 315 316 317 318 322 334 337 338 339 340 343 344 345 346 347 348 349 350 353 354 356 358 361 362 364 365 367 369 372 373 374 376 377 381 382 383 389

Table de hachage quadratique

Taille du tableau	Taille minimale (amas)	Taille maximale (amas)	Taille moyenne (amas)	Nombre de collisions	Temps d'exécution (ns)
15	2	2	2	8	102700
60	2	5	2.69	44	116600
150	2	13	3.55	125	458100

Double Hash Table

Array 1 :

Positions actives : 3 4 10 13 16 17 19 29 34 35 37 39 43 45

Array 2 :

Positions actives : 7 10 16 19 21 24 29 31 32 39 42 47 51 52 53 54 55 57 72 76 77 78 83 84 86 87 92 94 97 99 100 105 111 115 118 119 121 129 130 133 134 136 139 140 142 147 150 152 164 172 173 175 177 181 182 183 186 193 194 196

Array 3 :

Positions actives : 1 10 12 18 19 22 23 26 29 31 32 36 38 52 55 59 60 63 65 67 68 71 74 76 83 98 101 102 103 105 107 110 111 113 116 117 118 120 121 127 129 131 133 135 137 149 150 153 154 162 165 175 176 179 180 185 187 192 196 197 198 199 201 202 205 207 208 209 213 216 219 221 222 232 234 237 238 239 242 243 244 245 248 249 253 254 255 259 261 264 265 268 272 274 275 276 278 279 284 289 292 294 295 296 298 300 303 305 306 308 311 313 315 316 317 318 320 322 334 335 337 338 340 343 344 345 347 348 353 354 356 361 364 365 367 369 372 373 374 375 376 381 382 383 384 386 389 392 395 396

Table de hachage à dispersement double

Taille du tableau	Taille minimale (amas)	Taille maximale (amas)	Taille moyenne (amas)	Nombre de collisions	Temps d'exécution (ns)
15	2	2	2	9	53200
60	2	5	2.7	40	208900
150	2	5	2.59	127	459700

De vos observations, est-ce que la complexité temporelle de l'insertion d'un nombre N d'éléments pour chacune des trois tables est conforme à la théorie ? Expliquez brièvement les résultats.

La conformité temporelle est dans le meilleur des cas égal à O (1) pour toutes fonctions qu'elles soient linéaires, quadratique ou encore à double dispersement. Dans le cas contraire, elle est dans son pire cas lorsqu'elle est dans le cas O(n). En se basant sur nos observations (*Table de hachage linéaire* et *Table de hachage quadratique*), il est possible de constater que le nombre de collision est assez proche du nombre total d'éléments qu'il peut comporter. En ce qui est de la *Table de hachage à dispersement double*, on constate que le nombre de collisions est un peu plus grand que la taille du tableau *(et significativement plus grande que celui de la Table de hachage linéaire et celui de la Table de hachage quadratique).* Cela dit, considérant la proximité proche du rapport du nombre de collisions et de la taille de la table, il peut être dit que la grande majorité des éléments ont une complexité temporelle de O (1). Cette observation est tout à fait conforme à celle de la théorie qui dit qu'en général plus elle se rapproche de O (1), mieux elle est.

b) Load Factor

Linear Probing Hash Table

Taille du tableau	Nombre de conflits avec un facteur de compression 0.25	Nombre de conflits avec un facteur de compression 0.50	Nombre de conflits avec un facteur de compression 0.75
15	3	7	24
60	17	51	107
150	50	151	402

Quadratic Probing Hash Table

Taille du tableau	Nombre de conflits avec un facteur de compression 0.25	Nombre de conflits avec un facteur de compression 0.50	Nombre de conflits avec un facteur de compression 0.75
15	3	8	17
60	17	44	89
150	47	125	280

Double Hash Table

Taille du tableau	Nombre de conflits avec un facteur de compression 0.25	Nombre de conflits avec un facteur de compression 0.50	Nombre de conflits avec un facteur de compression 0.75
15	3	9	18
60	21	40	88
150	53	127	262

Que constatez-vous au niveau de la relation du Load Factor et du nombre de conflits?

On observe le fait que plus le facteur de compression croit, plus le nombre de collisions est croit, et ce pour les 3 tables *(à la légère exception de la table de hachage à double dispersement qui retourne une erreur lorsque le facteur de compression 0.75 est testé).* Sinon, en règle générale, le nombre de collisions lorsque le facteur de compression est à 0.25 est plus faible que la normal de 0.50. De l'autre côté, le nombre de collisions lorsque le facteur de compression est égal à 0.75 est plus grande que la normal de 0.50. Mise à part ceci, il est à note que le nombre de collisions pour la table de hachage à double dispersement est significativement plus grande que les deux autres tables, ou on peut observer que le nombre de collisions est à peu prêt pareille. En termes d'efficacité et de performance, la table de dispersement double est certainement la mieux placé. En effet, contrairement aux deux autres tables, ce dernier présente 222 collisions pour une table de 150 éléments contre 129 et 130. Plus encore, la table à dispersement double émerge un message d'erreur lorsque ce dernier est testé avec un facteur de compression de 0.75, alors que les deux autres tables, soit linéaire et quadratique ont respectivement 319 et 279 collisions.

c)Rehash

Quelle est la complexité asymptotique de l'opération du Rehash?

La complexité algorithmique de Rehash est de **O (n* log n)**. En effet, le jumelage de l'opération de copie (ayant une complexité de O(n)) et de celle de findPos (ayant une complexité de O (log n)) donne : O(n) * O(log n) = **O(n* log n)**.

Linear Probing Hash Table

Array 1 :

Positions actives : 13 23 29 33 34 35 57 75 91 105 108 122 123 129 138

Array 2 :

Positions actives : 2 5 6 8 9 12 13 16 17 19 20 21 23 24 26 29 31 33 35 45 48 49 50 55 60 64 80 82 83 85 86 87 88 89 90 91 95 100 108 110 112 113 114 116 118 122 126 128 129 130 131 132 133 134 136 142 143 145 147 149

Array 3 :

Positions actives : 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150

Table de hachage linéaire

Taille du tableau	Taille minimale (amas)	Taille maximale (amas)	Taille moyenne (amas)	Nombre de collisions	Temps d'exécution (ns)
15	2	3	2.5	0	67300
60	2	8	3.18	31	138000
150	150	150	150	924	286700

Quadratic Probing Hash Table

Array 1 :

Positions actives : 13 23 29 33 34 35 57 75 91 105 108 122 123 129 138

Array 2 :

Positions actives : 2 5 6 8 9 12 13 16 17 19 20 21 23 24 26 29 31 33 35 45 48 49 50 55 60 64 80 82 83 85 86 87 88 89 90 99 100 108 110 112 113 114 115 116 118 122 126 128 129 130 131 133 134 136 138 142 143 145 147 149

Array 3 :

Positions actives : 0 1 2 3 4 5 6 7 8 9 10 11 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150

Table de hachage quadratique

Taille du tableau	Taille minimale (amas)	Taille maximale (amas)	Taille moyenne (amas)	Nombre de collisions	Temps d'exécution (ns)
15	2	3	2.5	0	39900
60	2	6	2.81	21	106400
150	150	150	150	471	275000

Double Hash Table

Array 1 :

Positions actives : 13 23 29 33 34 35 57 75 91 105 108 122 123 129 138

Array 2 :

Positions actives : 2 5 8 12 16 17 19 20 21 23 26 29 31 33 35 45 48 49 55 60 64 76 78 79 80 82 83 84 85 86 87 88 90 96 99 100 104 106 108 110 112 113 114 116 118 119 121 122 125 126 128 129 131 133 136 140 142 145 147 149

Array 3 :

Positions actives : 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150

Table de hachage à dispersement double

Taille du tableau	Taille minimale (amas)	Taille maximale (amas)	Taille moyenne (amas)	Nombre de collisions	Temps d'exécution (ns)
15	2	3	2.5	0	34300
60	2	7	2.73	17	95200
150	150	150	150	402	448700

Comment a varié le temps d'exécutions et le nombre de conflits par rapport au tableau de la partie a)? Pourquoi?

En suspendant le rehash du code, on remarque que le console n'affiche rien. La raison derrière ce comportement bizarre est simple : un élément n'arrive pas à intégrer la table de hachage en raison du suspens de rehash, qui ne permet pas d'alloué à nouveau une nouvelle taille. Ainsi, par rapport au tableau de la partie a), le temps d'exécution et le nombre de conflits est tellement grande qu'il bloque l'afficheur de console de toute sortie. On pourrait quasiment interpréter ceci comme si le code n'était plus en marche.

Annexe

1. Modification de la fonction insert afin d'obtenir un Load Factor de 0.25

```
1 usage
public void insert( AnyType x )
{
    // Insert x as active
    int currentPos = findPos( x );
    if( isActive( currentPos ) )
        return;

    array[ currentPos ] = new HashEntry<AnyType>( x,  i: true );

    // Rehash; see Section 5.5
    if( ++currentSize > array.length / 4 )
        rehash( );
}

/**
 * Expand the hash table.
 */
```

2. Modification de la fonction insert afin d'obtenir un Load Factor de 0.75

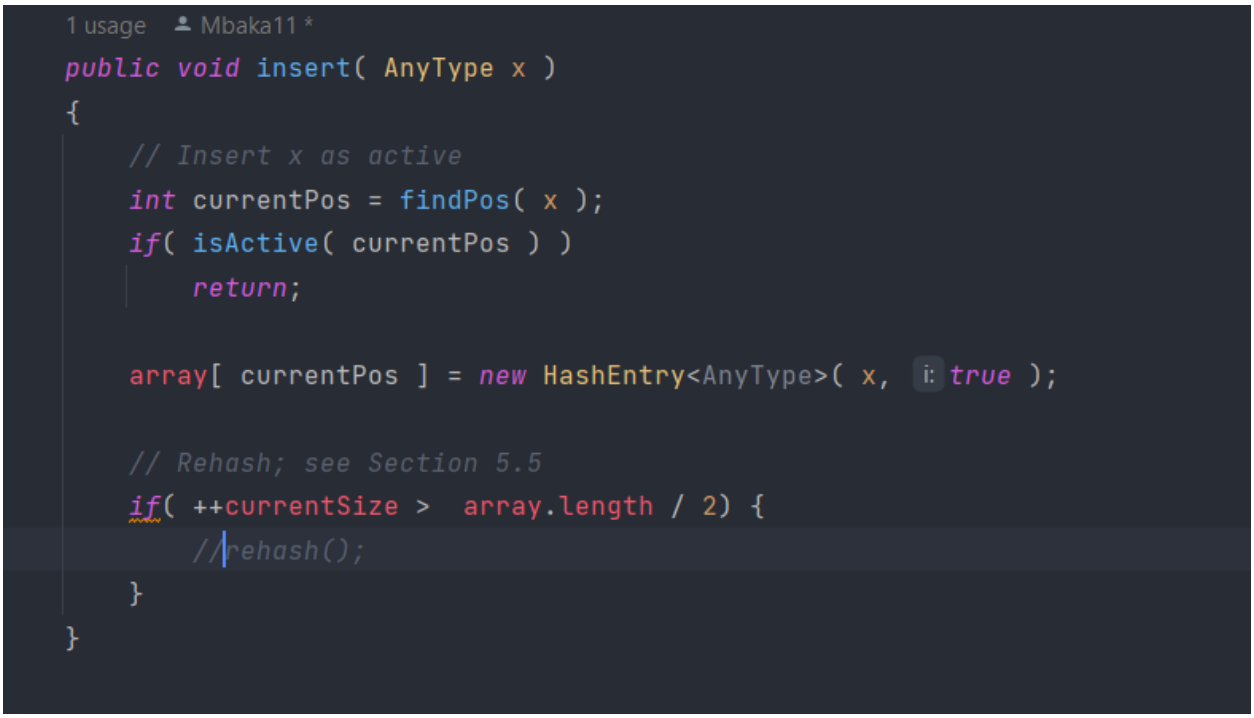
```
 */
1 usage
public void insert( AnyType x )
{
    // Insert x as active
    int currentPos = findPos( x );
    if( isActive( currentPos ) )
        return;

    array[ currentPos ] = new HashEntry<AnyType>( x,  i: true );

    // Rehash; see Section 5.5
    if( ++currentSize > 3 * array.length / 4 )
        rehash( );
}

/**
 * Expand the hash table.
 */
```

3. Suspension de l'appel de la fonction rehash (mise en commentaire)



```
1 usage   Mbaka11 *
public void insert( AnyType x )
{
    // Insert x as active
    int currentPos = findPos( x );
    if( isActive( currentPos ) )
        return;

    array[ currentPos ] = new HashEntry<AnyType>( x,  i: true );

    // Rehash; see Section 5.5
    if( ++currentSize >  array.length / 2) {
        //rehash();
    }
}
```

4. Ajout de constructeurs super afin de pouvoir initialiser les tables de hashage à 150

```
public class DoubleHashTable<AnyType> extends HashTable<AnyType> i
    1 usage   new *
    public DoubleHashTable() {
        super(DEFAULT_TABLE_SIZE);
    }
    new *
    public DoubleHashTable(int size) {
        super(size);
    }
    3 usages   Mbaka11 *
```

5. Initialisation de la taille des tables de hashage à 150

```
        5.841580961705714E7};

LinearProbingHashTable<Double> linearHashTable = new LinearProbingHashTable<Double>( size: 150);
QuadraticProbingHashTable<Double> quadraticHashTable = new QuadraticProbingHashTable<Double>( size: 150);
DoubleHashTable<Double> doubleHashTable = new DoubleHashTable<Double>( size: 150);

  //LINEAR

long startTime = System.nanoTime();
```

6. Ajout d'une variable nbrConflicts afin de compter le nombre de collisions / conflits

```
        int myhashValue2 = x.hashCode() % newValue;

        return  (newValue - myhashValue2);
    }
    4 usages
    public static final int DEFAULT_TABLE_SIZE = 11;

    24 usages
    public HashEntry<AnyType> [ ] array; // The array of eleme
    3 usages
    public int currentSize;              // The number of occu

    3 usages
    public int nbrConflicts;

    /**
     * Internal method to allocate array.
```

7. Ajout d’une fonction previousPrime pour myhash2

```
    return n;
}

1 usage   Mbaka11
public static int previousPrime( int n )
{
    n--;
    if( n == 2 )
        return n;

    if( n % 2 == 0 )
        n--;

    for( ; !isPrime( n ); n -= 2 )
        ;

    return n;
}
```

8. Ajout d’une fonction myhash2 pour la Double Hash Table

```
1 usage   Mbaka11
public int myhash2(AnyType x)
{
    int newValue = previousPrime(array.length);

    int myhashValue2 = x.hashCode() % newValue;

    return  (newValue - myhashValue2);
}
4 usages
public static final int DEFAULT_TABLE_SIZE = 11;

24 usages
```