Nom : MOSAAD
Prénom : Chehab
Numéro étudiant : 22106126

TP3 : Liste à raccourci - Skiplist.

1) **Description de l'archive logicielle fournie :**
2) **Description du principe de fonctionnement des SkipLists :**
3) **Travail à réaliser :**
    1. **Définition et construction d'une liste à raccourcis :**
        1) J'ai créé la SkipList en utilisant 2 structures qui sont les éléments de base de ma SkipList :
            1. La structure s_Node : J'ai créé cette structure en utilisant 4 champs qui sont :
                a. Level : le niveau du nœud dans la SkipList.
                b. Value : la valeur stockée dans le nœud.
                c. Next : un tableau de pointeurs vers les nœuds suivants dans chaque niveau.
                d. Prev : un pointeur vers le nœud précédent.
            2. La structure s_SkipList : J'ai créé cette structure en utilisant 4 champs qui sont :
                a. Size : le nombre d'éléments stockés dans la SkipList.
                b. Max_level : le niveau maximal de la SkipList.
                c. Sentinel : un pointeur vers le nœud sentinelle, qui marque le début de la SkipList.
                d. Prob : une fonction de distribution de probabilité qui est utilisée pour déterminer le niveau de chaque nœud dans la SkipList.

        2) Explication de la fonction SkipList skiplist_create(int nb_levels) : La fonction commence par allouer de la mémoire pour la sentinelle de la SkipList. Ensuite, elle alloue de la mémoire pour les pointeurs vers les éléments suivants de la sentinelle, selon le nombre de niveaux souhaité. Ensuite, elle initialise tous les pointeurs suivants de la sentinelle à la sentinelle elle-même, car il n'y a pas encore d'autres éléments dans la SkipList. Elle initialise également le pointeur précédent de la sentinelle à la sentinelle elle-même.
        Ensuite, la fonction alloue de la mémoire pour la nouvelle SkipList. Elle initialise le niveau de la sentinelle au nombre de niveaux spécifié, et initialise la sentinelle de la SkipList avec la sentinelle que nous avons créée.
        La fonction initialise également la taille de la SkipList à 0 car il n'y a pas encore d'éléments dans la SkipList, et initialise le niveau maximum de la SkipList au nombre de niveaux spécifié.
        Enfin, la fonction initialise le générateur de nombres aléatoires utilisé pour générer les niveaux de chaque élément de la SkipList en lui fournissant une graine de 0 (rng_initialize(0)) et retourne un pointeur vers la nouvelle SkipList créée.

        Explication de la fonction void skiplist_delete(SkipList d) : Pour chaque élément, la fonction libère d'abord la mémoire allouée pour les pointeurs next de l'élément, puis libère la mémoire allouée pour l'élément lui-même. Une fois que tous les éléments ont été supprimés, la fonction libère la mémoire allouée pour les pointeurs next de la sentinelle et pour la sentinelle elle-même. Enfin, la fonction libère la mémoire allouée pour la SkipList.

        3) Explication de la fonction unsigned int skiplist_size(SkipList d) : La fonction commence par vérifier si la SkipList passée en paramètre est valide en utilisant la fonction assert. Si la condition est fausse, le programme s'arrête et affiche un message d'erreur.
        Sinon la fonction retourne simplement la taille de la SkipList en accédant au champ size de la structure s_SkipList. Cette fonction possède une implémentation en O(1).

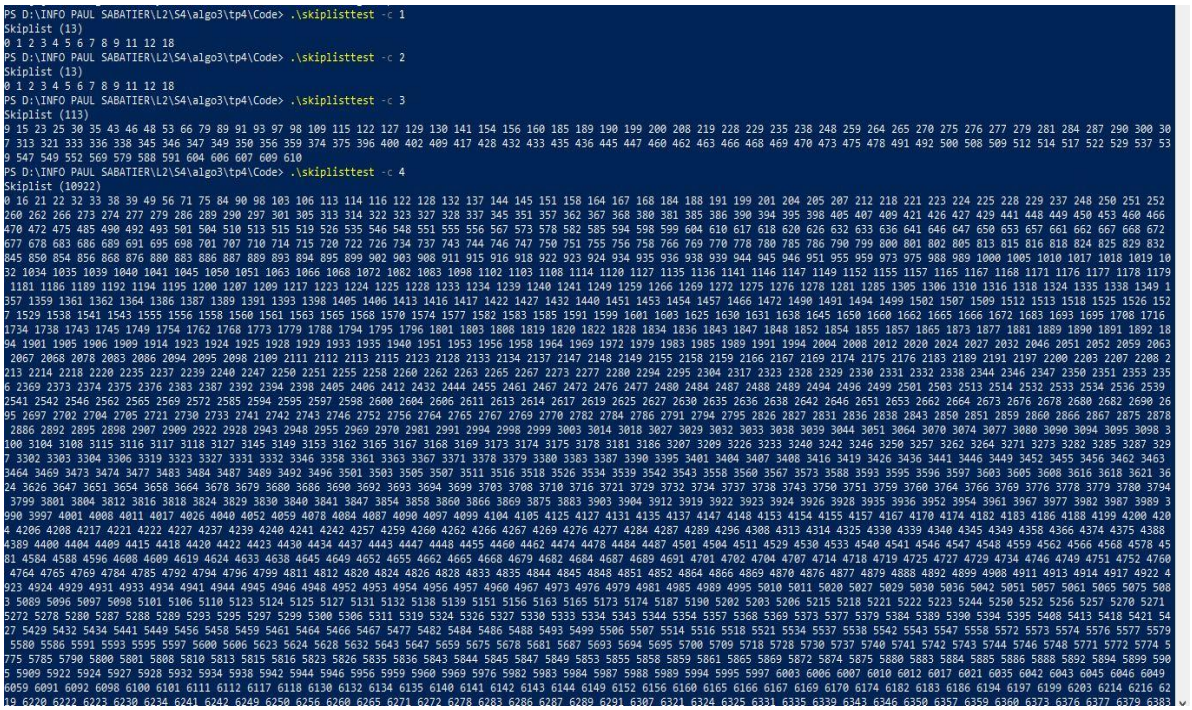
Explication de la fonction int skiplist_ith(SkipList d, unsigned int i) : La fonction commence par vérifier si l'indice spécifié est dans la plage de la SkipList à l'aide de la fonction assert. Si l'indice est valide, la fonction initialise un pointeur e à la sentinelle de la SkipList, puis parcourt chaque élément de la SkipList jusqu'à l'indice spécifié en déplaçant le pointeur e au suivant. Une fois que le pointeur e pointe sur l'élément à l'indice spécifié, la fonction retourne la valeur de cet élément. Cette fonction possède une implémentation en O(n).

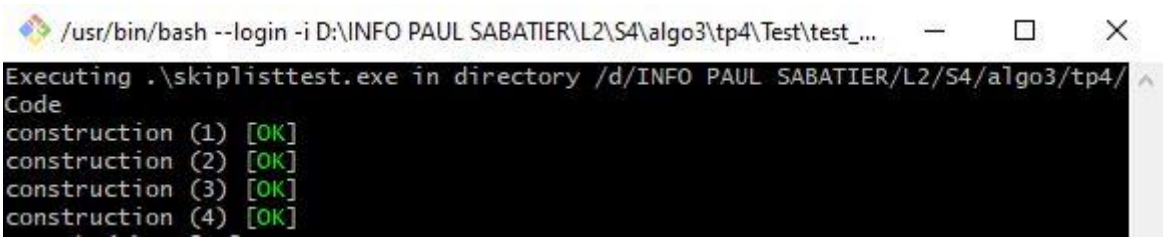
Explication de la fonction void skiplist_map(SkipList d, ScanOperator f, void *user_data) : La fonction commence par initialiser un pointeur e à la sentinelle de la SkipList. Ensuite, elle parcourt chaque élément de la SkipList en utilisant une boucle for ; à chaque itération, le pointeur e est déplacé vers le suivant. Également, la fonction f est ensuite appliquée à la valeur de l'élément actuel en utilisant l'argument. Cette fonction possède une implémentation en O(n).

4) Explication de la fonction SkipList skiplist_insert(SkipList d, int value) : La fonction commence par allouer le nouveau nœud à insérer et l'initialiser avec la valeur donnée, un niveau aléatoire (en utilisant la fonction rng_get_value() avec le générateur associe à la liste et le niveau maximal de la SkipList), et des pointeurs suivants initialisés à NULL.
Si la SkipList est vide, le nouveau nœud est inséré à la position de la sentinelle, et les pointeurs suivants de la sentinelle sont mis à jour pour pointer vers le nouveau nœud.
Sinon, la SkipList est parcourue jusqu'à trouver l'endroit où le nouveau nœud doit être inséré, en fonction de la valeur de la clé du nouveau nœud.
Si un nœud avec la même valeur existe déjà, le nouveau nœud n'est pas inséré et la SkipList est renvoyée sans modification.
Une fois que l'emplacement d'insertion a été trouvé, les pointeurs précédents et suivants des nœuds concernés sont mis à jour pour insérer le nouveau nœud.
Si le niveau du nouveau nœud est inférieur ou égal au niveau du nœud courant, les pointeurs suivants du nouveau nœud sont mis à jour pour pointer vers les mêmes éléments que le nœud courant, et les pointeurs suivants du nœud courant sont mis à jour pour pointer vers le nouveau nœud.
Si le niveau du nouveau nœud est supérieur au niveau du nœud courant, les pointeurs suivants du nouveau nœud sont d'abord mis à jour pour pointer vers les mêmes éléments que le nœud courant pour les niveaux inférieurs au niveau du nœud courant.
Ensuite, la SkipList est parcourue à partir de l'élément précédent jusqu'à atteindre le niveau du nouveau nœud, et les pointeurs suivants de chaque élément dans la liste sont mis à jour pour pointer vers le nouveau nœud.
Enfin, la taille de la SkipList est incrémentée et la SkipList mise à jour est renvoyée.

5) Explication de la fonction void test_construction(int num) : la fonction crée une SkipList d en utilisant la fonction buildlist avec num comme argument. Ensuite, elle récupère la taille de la SkipList d à l'aide de la fonction skiplist_size et l'affiche à l'écran avec un message. Ensuite, la fonction boucle sur tous les éléments de la SkipList à l'aide de la fonction skiplist_ith et les affiche à l'écran. Enfin, la fonction supprime la SkipList à l'aide de la fonction skiplist_delete et affiche un retour à la ligne.

6)





2. Recherche d'une valeur dans une liste a raccourcis :

1) Explication de la fonction bool skiplist_search(SkipList d, int value, unsigned int *nb_operations): La fonction commence par initialiser un pointeur "current" à l'élément sentinelle de la SkipList, qui est le premier élément de chaque niveau. Ensuite, elle parcourt les niveaux de la SkipList, en partant du niveau le plus haut et en descendant vers le niveau 0.
   À chaque niveau, la fonction se déplace à l'élément suivant tant que sa valeur est inférieure à la valeur cherchée, tout en comptant le nombre d'opérations effectuées avec le pointeur "nb_operations". Si la valeur de l'élément courant est égale à la valeur cherchée, la fonction retourne "true" pour indiquer que la valeur a été trouvée.
   Si la fonction arrive au niveau 0 sans trouver la valeur cherchée, elle vérifie l'élément suivant de l'élément courant dans ce niveau. Si la valeur de cet élément est égale à la valeur cherchée, la fonction retourne "true", sinon elle retourne "false" pour indiquer que la valeur n'a pas été trouvée dans la SkipList.

   Il y a un erreur dans la fonction est qu'elle ne compte pas les opérations faites si la SkipList n'était pas triée dans le bon ordre mais elle trouve toujours les variables voulues dans le nombre d'opérations correctes.

2) Explication de la fonction void test_search(int num) : la fonction crée une SkipList d en utilisant la fonction buildlist avec num comme argument. Ensuite, elle ouvre un fichier "search[num].txt" en lecture et récupère les valeurs qui y sont stockées. Pour chaque valeur, la fonction effectue une recherche dans la SkipList à l'aide de la fonction skiplist_search et enregistre le nombre d'opérations effectuées dans un tableau tabres. Elle affiche ensuite le résultat de la recherche (true si la valeur est trouvée, false sinon).
   Après avoir parcouru toutes les valeurs du fichier, la fonction calcule et affiche des statistiques sur les résultats de la recherche, telles que la taille de la SkipList, le nombre de valeurs trouvées

et non trouvées, ainsi que le nombre minimal, maximal et moyen d'opérations nécessaires pour effectuer les recherches.

3)

```
PS D:\INFO PAUL SABATIER\L2\S4\algo3\tp4\Code> .\skiplisttest -s 1
1 -> true
2 -> true
5 -> true
8 -> true
9 -> true
19 -> false
18 -> true
3 -> true
17 -> false
4 -> true
16 -> false
15 -> false
0 -> true
14 -> false
13 -> false
12 -> true
7 -> true
11 -> true
10 -> false
6 -> true
Statistics :
        Size of the list : 13
Search 20 values :
        Found 13
        Not found 7
        Min number of operations : 1
        Max number of operations : 7
        Mean number of operations : 5
```

```
PS D:\INFO PAUL SABATIER\L2\S4\algo3\tp4\Code> .\skiplisttest -s 2
1 -> true
2 -> true
5 -> true
8 -> true
9 -> true
19 -> false
18 -> true
3 -> true
17 -> false
4 -> true
16 -> false
15 -> false
0 -> true
14 -> false
13 -> false
12 -> true
7 -> true
11 -> true
10 -> false
6 -> true
Statistics :
        Size of the list : 13
Search 20 values :
        Found 13
        Not found 7
        Min number of operations : 1
        Max number of operations : 14
        Mean number of operations : 9
```

```
77 -> false
347 -> true
112 -> false
41 -> false
577 -> false
279 -> true
Statistics :
        Size of the list : 113
Search 369 values :
        Found 77
        Not found 292
        Min number of operations : 1
        Max number of operations : 99
        Mean number of operations : 37
```

```
7711 -> true
34778 -> false
31357 -> true
22548 -> false
1678 -> false
29991 -> false
33853 -> false
33851 -> false
42325 -> false
39097 -> false
15290 -> false
41995 -> false
Statistics :
        Size of the list : 10922
Search 37035 values :
        Found 8209
        Not found 28826
        Min number of operations : 2
        Max number of operations : 9674
        Mean number of operations : 1769
```

```
search (1)   [KO]
search (2)   [OK]
search (3)   [KO]
search (4)   [KO]
```

3. Itérateur et recherche linéaire d'une valeur dans une liste :

1) La structure s_SkipListIterator : J'ai créé cette structure en utilisant 4 champs qui sont :
    1. current : un pointeur vers le nœud actuel de l'itérateur.
    2. begin : un pointeur vers le nœud de début de la liste.
    3. next : un pointeur de fonction qui prend en entrée un objet de type SkipListIterator et renvoie un nouvel objet SkipListIterator correspondant au prochain élément de la liste.
    4. d : une référence à l'objet SkipList sur lequel l'itérateur est en train de travailler.

```
void iterate_on_skiplist (SkipList d) {
    SkipListIterator e = skiplist_iterator_create (d, DIRECTION_ITERATOR);
    for ( e = skiplist_iterator_begin (e); ! skiplist_iterator_end (e); e = skiplist_iterator_next (e) ) {
            Do_somehing_with ( skiplist_iterator_value (e));
    }
}
```

Afin d'exécuter ce code au-dessus, nous devons coder les fonctions skiplist_iterator_create (d, DIRECTION_ITERATOR), skiplist_iterator_begin (e), skiplist_iterator_end (e), skiplist_iterator_next (e), skiplist_iterator_value (e). Également, comme il y a la fonction skiplist_iterator_create, nous devons coder la fonction void skiplist_iterator_delete(SkipListIterator it) pour pouvoir supprimer l'itérateur et libère la mémoire. Mais aussi, DIRECTION_ ITERATOR peut prendre une des deux directions FORWARD, BACKWARD de parcours donc il y a deux autres fonctions qui sont SkipListIterator forward(SkipListIterator it) et SkipListIterator backward(SkipListIterator it).

Explication de la fonction SkipListIterator forward(SkipListIterator it) : la fonction déplace l'itérateur passé en argument vers le prochain élément en modifiant la valeur de son pointeur « current » pour qu'il pointe vers le next[0] de l'élément actuel. Enfin, elle retourne l'itérateur mis à jour pour pouvoir être utilisé par le programme appelant.

Explication de la fonction SkipListIterator backward(SkipListIterator it) : la fonction déplace l'itérateur passé en argument vers l'élément précédent dans l'ordre croissant de valeur. Pour cela, on modifie le pointeur « current » de l'itérateur en le faisant pointer sur le noeud précédent (celui dont la valeur est inférieure) dans la liste, en utilisant le pointeur « prev » de l'objet Node. Enfin, la fonction retourne l'itérateur mis à jour.

Explication de la fonction SkipListIterator skiplist_iterator_create(SkipList d, unsigned char w) : La fonction commence par allouer d'abord de la mémoire pour le nouvel itérateur, puis elle vérifie le sens de l'itérateur en utilisant la variable w. Si w est FORWARD_ITERATOR, l'itérateur est initialisé sur le premier élément de la SkipList, sinon, si w est BACKWARD_ITERATOR, l'itérateur est initialisé sur le dernier élément. La fonction stocke également le premier ou le dernier élément dans la variable begin de l'itérateur pour permettre la réinitialisation de l'itérateur plus tard. La fonction configure également la fonction de déplacement de l'itérateur en fonction du sens spécifié (avant ou arrière). Enfin, la fonction stocke la SkipList à laquelle l'itérateur est associé et renvoie le nouvel itérateur créé.

Explication de la fonction void skiplist_iterator_delete(SkipListIterator it) : la fonction commence par vérifier si le pointeur d'itérateur it est non nul avant d'exécuter la fonction free, pour éviter une erreur de segmentation. Si it est nul, la fonction ne fait rien.

Explication de la fonction SkipListIterator skiplist_iterator_begin(SkipListIterator it) : la fonction réinitialise l'itérateur en le positionnant sur le premier élément ou le dernier élément de la SkipList, selon le sens de l'itérateur. Elle met à jour le pointeur courant de l'itérateur avec le pointeur vers l'élément initial et retourne l'itérateur mis à jour.

Explication de la fonction bool skiplist_iterator_end(SkipListIterator it) : la fonction retourne une valeur booléenne (vrai ou faux) indiquant si l'itérateur a atteint la fin de la SkipList, c'est-à-dire si son pointeur courant pointe vers la sentinelle de la SkipList.

Explication de la fonction SkipListIterator skiplist_iterator_next(SkipListIterator it) : la fonction renvoie l'itérateur suivant en appelant la fonction "next" de l'itérateur sur l'itérateur courant. Le type de l'itérateur renvoyé dépend du sens de l'itération, avant ou arrière.

Explication de la fonction int skiplist_iterator_value(SkipListIterator it) : la fonction commence par obtenir la valeur de l'élément courant de l'itérateur. Si l'itérateur n'est pas arrivé à la fin de la SkipList (c'est-à-dire si le pointeur courant n'est pas égal à la sentinelle de la SkipList), alors la fonction retourne la valeur de l'élément courant. Sinon, si l'itérateur est à la fin de la SkipList, la fonction retourne 0.

2) Explication de la fonction void test_search_iterator(int num) : la fonction lit un fichier contenant des valeurs à chercher dans la SkipList. Pour chaque valeur à chercher, la fonction positionne un itérateur au début de la SkipList et incrémente un compteur de recherche. Elle parcourt ensuite la SkipList à l'aide de l'itérateur, en comparant la valeur cherchée à chaque élément de la liste jusqu'à trouver la valeur cherchée ou atteindre la fin de la liste. Pour chaque recherche, la fonction stocke le résultat (true si la valeur a été trouvée, false sinon) et le nombre d'opérations effectuées pour trouver la valeur cherchée.
Une fois toutes les recherches effectuées, la fonction calcule des statistiques sur les résultats des recherches, notamment le nombre de fois où la valeur cherchée a été trouvée, le nombre de fois où elle n'a pas été trouvée, le nombre minimum, maximum et moyen d'opérations effectuées pour trouver chaque valeur cherchée. La fonction affiche ensuite les résultats des statistiques et libère la mémoire allouée.

3) La fonction void test_search_iterator(int num) est meilleure que la fonction void test_search(int num) parce qu'elle utilise un itérateur pour parcourir la SkipList, ce qui permet de réduire le nombre d'opérations nécessaires pour la recherche.
L'itérateur utilise une boucle while pour parcourir la liste, en vérifiant à chaque étape si la valeur cherchée est inférieure à la valeur du nœud suivant. En utilisant un itérateur pour parcourir la SkipList, la boucle while évite la répétition de certaines opérations inutiles de la boucle for dans

skiplist_search, ce qui réduit le temps d'exécution. En outre, la boucle for dans la fonction skiplist_search effectue des opérations de comparaison inutiles dans les niveaux supérieurs de la SkipList, car il est possible que la valeur cherchée soit inférieure à la valeur du nœud suivant dès le niveau supérieur. En utilisant un itérateur, on évite ces opérations inutiles et on réduit encore le temps d'exécution.
Le temps d'exécution est réduit car les itérateurs sont optimisés pour les boucles while, tandis que les boucles for sont moins optimisées. En outre, la boucle for doit effectuer plusieurs opérations à chaque itération, telles que l'accès à la valeur de d->max_level, la comparaison de i à zéro, et la soustraction de un à i. La boucle while dans l'itérateur n'a pas besoin d'effectuer ces opérations à chaque itération, car l'itérateur stocke les informations nécessaires pour parcourir la liste.

4)

```
PS D:\INFO PAUL SABATIER\L2\S4\algo3\tp4\Code> .\skiplisttest -i 1
1 -> true
2 -> true
5 -> true
8 -> true
9 -> true
19 -> false
18 -> true
3 -> true
17 -> false
4 -> true
16 -> false
15 -> false
0 -> true
14 -> false
13 -> false
12 -> true
7 -> true
11 -> true
10 -> false
6 -> true
Statistics :
        Size of the list : 13
Search 20 values :
        Found 13
        Not found 7
        Min number of operations : 1
        Max number of operations : 13
        Mean number of operations : 9
```

```
PS D:\INFO PAUL SABATIER\L2\S4\algo3\tp4\Code> .\skiplisttest -i 2
1 -> true
2 -> true
5 -> true
8 -> true
9 -> true
19 -> false
18 -> true
3 -> true
17 -> false
4 -> true
16 -> false
15 -> false
0 -> true
14 -> false
13 -> false
12 -> true
7 -> true
11 -> true
10 -> false
6 -> true
Statistics :
        Size of the list : 13
Search 20 values :
        Found 13
        Not found 7
        Min number of operations : 1
        Max number of operations : 13
        Mean number of operations : 9
```

```
77 -> false
347 -> true
112 -> false
41 -> false
577 -> false
279 -> true
Statistics :
        Size of the list : 113
Search 369 values :
        Found 77
        Not found 292
        Min number of operations : 1
        Max number of operations : 113
        Mean number of operations : 100
```

```
7711 -> true
34778 -> false
31357 -> true
22548 -> false
1678 -> false
29991 -> false
33853 -> false
33851 -> false
42325 -> false
39097 -> false
15290 -> false
41995 -> false
Statistics :
        Size of the list : 10922
Search 37035 values :
        Found 8209
        Not found 28826
        Min number of operations : 2
        Max number of operations : 10922
        Mean number of operations : 9707
```

```
iterator (1) [OK]
iterator (2) [OK]
iterator (3) [OK]
iterator (4) [OK]
```

4. Suppression d'une valeur dans une liste :

1) Explication de la fonction SkipList skiplist_remove(SkipList d, int value) : la fonction commence par initialiser un pointeur d'élément "current" à l'élément suivant de la sentinelle de la SkipList, ainsi qu'un pointeur d'élément de précedent "prev". Ensuite, elle parcourt la SkipList tant que le pointeur courant n'est pas la sentinelle de la SkipList et que la valeur de l'élément courant n'est pas égale à la valeur donnée. Lorsque l'élément avec la valeur donnée est trouvé, elle stocke le pointeur de l'élément précédent dans "prev". Ensuite, la fonction parcourt les niveaux de l'élément courant en partant de son niveau jusqu'au niveau 0. Pour chaque niveau, elle déplace le pointeur "prev" à l'élément précédent et met à jour les pointeurs suivants de l'élément précédent à l'indice i avec les pointeurs suivants de l'élément courant à l'indice i, pour chaque indice i supérieur ou égal au compteur et inférieur au niveau de l'élément précédent et courant. Après avoir mis à jour les pointeurs suivants des éléments précédents, la fonction met à jour le pointeur précédent de l'élément suivant de l'élément courant avec le pointeur précédent de l'élément courant, puis elle décrémente la taille de la SkipList. Enfin, elle libère la mémoire allouée pour les pointeurs suivants de l'élément courant et pour l'élément courant. Enfin, elle retourne la SkipList modifiée.

2) Explication de la fonction void test_remove(int num) : la fonction commence par construire une SkipList de taille 'num' en appelant la fonction 'buildlist'. Ensuite, elle ouvre un fichier de test pour les suppressions en appelant 'fopen' et en passant le nom de fichier généré par la fonction 'gettestfilename'. Si le fichier ne peut pas être ouvert, un message d'erreur est affiché et le

programme se termine. La fonction lit ensuite la première ligne du fichier de test, qui contient le nombre de valeurs à supprimer, et exécute une boucle pour supprimer chaque valeur lue dans le fichier de test en appelant la fonction 'skiplist_remove'. Elle ferme ensuite le fichier de test. Après les suppressions, la fonction obtient la taille de la SkipList en appelant 'skiplist_size', puis affiche les éléments de la SkipList, en partant du dernier, en appelant 'skiplist_ith' dans une boucle. Enfin, elle supprime la SkipList en appelant 'skiplist_delete'.

3)

```
PS D:\INFO PAUL SABATIER\L2\S4\algo3\tp4\Code> .\skiplisttest -r 1
Skiplist (6)
18 9 6 4 2 0
PS D:\INFO PAUL SABATIER\L2\S4\algo3\tp4\Code> .\skiplisttest -r 2
Skiplist (6)
9 7 6 5 2 0
PS D:\INFO PAUL SABATIER\L2\S4\algo3\tp4\Code> .\skiplisttest -r 3
Skiplist (39)
610 606 591 588 522 517 475 466 463 460 447 445 428 402 356 338 336 333 313 287 284 276 265 264 259 229 200 199 190 156
154 129 115 93 91 46 43 35 9
PS D:\INFO PAUL SABATIER\L2\S4\algo3\tp4\Code> .\skiplisttest -r 4
Skiplist (561)
49369 49220 49164 49096 49041 48880 48851 48794 48762 48721 48680 48619 48609 48605 48473 48406 48403 48335 48295 48293
48204 48078 47928 47820 47762 47761 47586 47585 47542 47447 47435 47367 47335 47260 47258 47243 47197 47082 47073 47053
47005 46984 46983 46967 46961 46854 46632 46631 46517 46370 46176 46009 46007 45743 45738 45617 45515 45444 45434 45214
44914 44896 44639 44615 44506 44325 44279 44242 44050 43993 43974 43925 43866 43670 43625 43529 43496 43144 43052 42966
42837 42795 42766 42666 42653 42596 42546 42538 42479 42414 42311 42273 42244 42085 41868 41790 41680 41617 41602 41517
41499 41237 41112 41111 41028 41022 41005 40904 40800 40539 40502 40428 40427 40389 40280 40168 40124 40099 39996 39902
39730 39719 39413 38972 38928 38797 38772 38770 38724 38621 38601 38508 38485 38292 38288 38171 38165 37888 37818 37815
37758 37558 37490 37411 37336 37245 37181 37057 37011 36968 36653 36629 36528 36493 36436 36380 36343 36080 36024 35942
35941 35939 35850 35800 35688 35650 35567 35563 35504 35462 35434 35411 35387 35351 35129 35122 35086 34617 34525 34303
34272 34134 34085 34013 33990 33954 33857 33740 33582 33492 33313 33190 33135 33031 32998 32908 32866 32750 32733 32712
32594 32566 32532 32526 32347 32253 32121 32053 31709 31610 31605 31499 31187 31134 30755 30541 30462 30399 30387 30169
29836 29823 29585 29584 29491 29396 29360 29357 29315 29194 29169 29078 29053 29048 29040 28999 28755 28711 28670 28559
28438 28301 28241 28208 27934 27918 27772 27667 27654 27553 27463 27061 27053 27045 27036 27035 26993 26982 26822 26740
26736 26595 26520 26459 26442 26383 26347 26328 26304 26257 26232 26106 26043 25939 25887 25653 25626 25304 25269 25231
25217 25213 25063 25025 24987 24885 24847 24833 24653 24608 24553 24419 24347 24144 24053 23977 23976 23968 23938 23579
23540 23312 23207 22977 22877 22833 22691 22407 22272 22242 22233 22190 22032 21856 21809 21748 21611 21560 21525 21440
21151 21012 20438 20340 20292 20280 20252 20249 20182 20143 20009 19771 19507 19477 19463 19445 19311 19285 19030 18883
18808 18762 18523 18356 18297 18257 18224 18027 17934 17851 17676 17607 17597 17546 17457 17441 17361 17300 17279 17229
17001 16800 16783 16648 16457 16372 16359 16351 16206 16197 16090 16079 16054 15962 15855 15775 15580 15417 15405 15332
15322 15315 15250 15248 15217 15138 15129 14986 14851 14733 14720 14680 14677 14666 14552 14529 13925 13855 13642 13559
13553 13431 13186 13152 13065 13059 13048 12988 12947 12897 12773 12709 12628 12540 12431 12322 12231 11991 11982 11966
11945 11933 11891 11787 11736 11726 11611 11602 11598 11582 11570 11470 11331 11163 10990 10966 10917 10818 10646 10594
10582 10425 10384 10292 10284 10233 10156 10056 10055 10018 9819 9695 9638 9567 9474 9432 9417 9227 9224 9116 8926 8861
8798 8749 8697 8589 8506 8373 8333 8253 8167 7811 7683 7591 7476 7368 7279 7016 7015 6883 6810 6716 6620 6502 6457 6307
6256 6194 6140 6046 5956 5944 5883 5647 5632 5466 5418 5344 5327 5293 5156 5110 4960 4957 4922 4752 4562 4547 4511 4462
4259 4217 4127 3875 3801 3626 3560 3558 3496 3492 3462 3455 3378 3367 3363 3108 3018 2998 2741 2704 2646 2572 2477 2375
2338 2331 2317 2134 2133 2051 1762 1754 1457 1453 1451 1186 1114 1018 944 894 887 818 790 618 610 594 501 323 266 122 11
4
```

```
remove (1)    [OK]
remove (2)    [OK]
remove (3)    [OK]
```

```
/usr/bin/bash --login -i D:\INFO PAUL SABATIER\L2\S4\algo3\tp4\Test\test_...
Executing .\skiplisttest.exe in directory /d/INFO PAUL SABATIER/L2/S4/algo3/tp4/
Code
construction (1) [OK]
construction (2) [OK]
construction (3) [OK]
construction (4) [OK]
search (1)    [KO]
search (2)    [OK]
search (3)    [KO]
search (4)    [KO]
iterator (1) [OK]
iterator (2) [OK]
iterator (3) [OK]
iterator (4) [OK]
remove (1)    [OK]
remove (2)    [OK]
remove (3)    [OK]
```