S T Q Q S S D DATA / /

Capítulo 10 - Maps, Hash tables e Skip Lists

Exercícios

R-10.1) Pior caso de tempo de execução para inserir n pares chave-valor em um Map n implementado com a classe UnsortedTableMap.
A inserção em um Map levaria tempo constante O(1), mas devido a implementação utilizando UnsortedTableMap, cada um dos métodos fundamentais leva O(n) devido a necessidade de verificar toda a lista ao procurar uma entrada existente.

R-10.3) Implementação método containsKey() para a classe UnsortedTableMap;

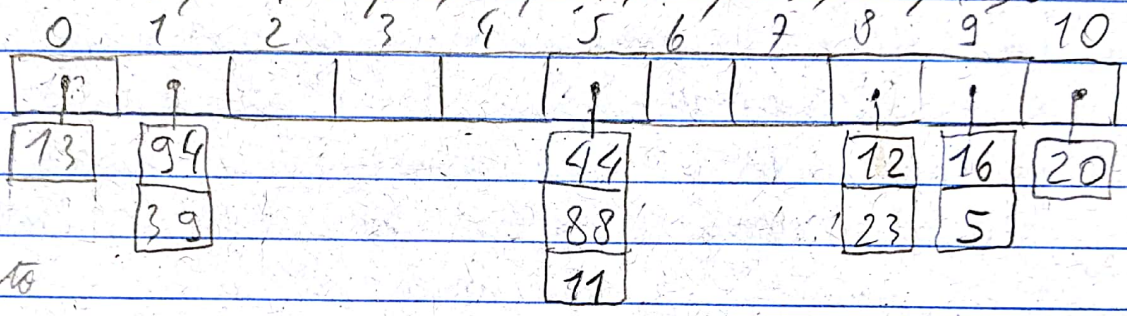
```
public boolean containsKey(K key) {
    Iterator<Entry<K,V>> i = entrySet.iterator();
    if (key == null) {
        while (i.hasNext()) {
            Entry<K,V> e = i.next();
            if (e.getKey == null)
                return true;
        }
    } else {
        while (i.hasNext()) {
            Entry<K,V> e = i.next();
            if (key.equals(e.getKey()))
                return true;
        }
    }
    return false;
}
```

S T Q Q S S D DATA / /

R-10.5 - Bom código Hash para número de identificação com uma sequência de números e letras 9X9XX99X9XX999999, onde '9' representa um dígito e 'X' uma letra.
Os códigos de hash polinomial ou os códigos de mudança cíclica seriam bons.

R-10.6 - Desenhe a tabela Hash de 11 entradas que resulta na utilização da função hash, $h(x) = (3x+5) \bmod 11$, para as seguintes entradas: 12, 44, 13, 88, 23, 94, 11, 39, 20, 16, 5

Chaining / colisões tratadas por encadeamento



R-10.7 - Implementação exercício anterior utilizando linear probing para tratamento de colisões.

0	1	2	3	4	5	6	7	8	9	10
13	94	39	16	5	44	88	11	12	23	20

R-10.8 - Implementação utilizando quadratic probing: $A[(h(k) + f(i)) \bmod N]$ onde, $f(i) = i^2$.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
					44	13		12								

O método quadratic probing falha em relação ao linear probing para o número 13, que inicialmente estava na posição 0 e agora está na posição 6.

STOOSSD 12, 44, 13, 88, 23, 94, 11, 39, 20, 16, 5 DATA / /

R-10.9 Exercício 6, colisões tratadas por hash duplo usando a função secundária $h'(K) = 7 - (K \bmod 7)$?

0	1	2	3	4	5	6	7	8	9	10
13	94	23	88	39	44	11	5	12	16	20

hash (3, 1, 5), se houver colisão utilize hash secundária.

R-10.10) Qual é o pior caso para colocar n entradas em uma tabela Hash inicialmente vazia, com colisões resolvidas por encadeamento? Qual é o melhor caso? Seria O(n) se todas as entradas fossem na mesma cadeia, o melhor caso seria O(1), o mesmo que um hash normal.

R-10.11) Tabela Hash Figura 10.6 com tamanho 19 e função Hash $h(K) = 3K \bmod 17$.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
entrada			12	18	41		36	25		54			38	10		90	28		

54
28
90
18
10
36
25
18
12
90

R-10.13 Alteração método bucketRemove da classe ChainHashMap: Qual o comportamento?

```
int oldSize = bucket.size();
V answer = bucket.remove(K);
(n -= (oldSize - bucket.size());
```

```
V answer = bucket.remove(K);
if (answer != null)
    n--;
```

S T Q Q S S D DATA / /

O comportamento do método será afetado pois com a alteração o n será decrementado apenas se answer for diferente de null. Na versão anterior o n é sempre decrementado pelo size.

R-10.17) Explique porque uma Hash table não é indicado para implementar um mapa ordenado.

Uma tabela hash usa uma função hash para definir em qual posição um elemento deve ser armazenado, tal função faz com que os elementos sejam armazenados de forma desordenada e também pode causar colisão, ou seja, um elemento pode ser armazenado na mesma posição de outro, o que requer funções específicas para tratamento de colisões.

R-10.18 - Pior caso de execução para deletar n entradas de um SortedTableMap que contem 2n entradas.

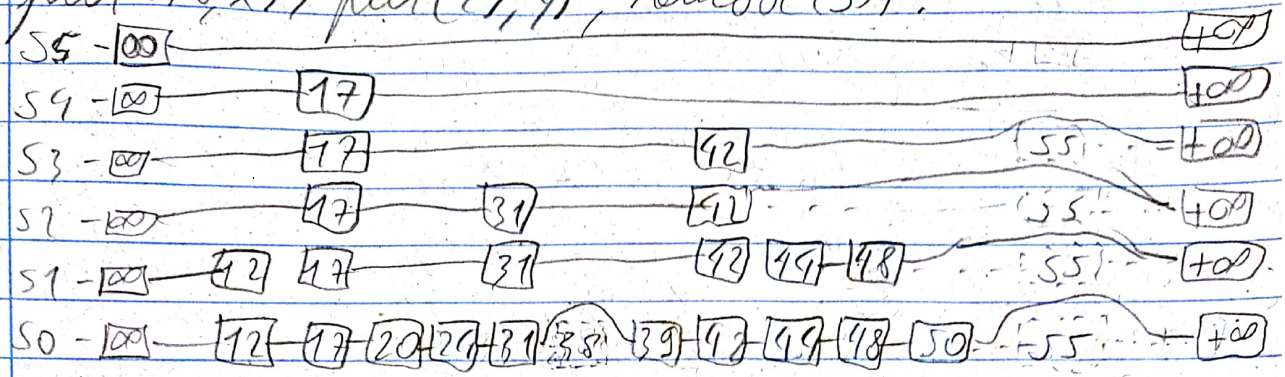
O pior caso de execução é O(n), pois a remoção percorre o Map inteiro.

R.10.19 - Semelhante ao R-10.3

R-10.21 - O método findIndex proposto pelo exercício não trará os resultados do método original e trará os resultados como nulo, pois não tem uma variável que salva o valor de comparação do índice.

S T Q Q S S D DATA / /

R-10.23 - Operações SkipList: remove(38), put(48,x), put(29,y), remove(55).



R-10.24 - Remoção de map em SkipList

```
Algoritmo SkipRemove(K)
    Entrada: uma chave K,
    Saída: o elemento p que contem a chave.
    p ← SkipSearch(K)

    Se p.key != K
        retorna null
    enquanto p != null
        p.left.right ← p.right;
        p.right.left ← p.left;
```