Relatório – Tabela Hash

1. Descrição do algoritmo

A estrutura abordada neste tópico tem como principal característica a associação entre chaves e valores, em que estas primeiras são calculadas por meio da implementação de um método de hasheamento.

Um outro aspecto importante das Hash Tables é a possibilidade de colisões de valores que retornam a mesma chave quando 'hasheados', este podendo ter diversas soluções, como por exemplo a implementação de Linked Lists para o armazenamento dos valores

2. Pseudo-código

```
class HashTable
    initialize n as zero
    initialize max as 10
    initialize dlls as empty list

    for each bucket in HashTable
        add new DoublyLinkedList instance in dlls list
    endfor

method hash (parameters: number to value): number
    return value % max

method expand (): void
    initialize dllCopy as dlls list
    set max to 2 * max
    call clear() method

    for each i value from 0 to max / 2
        for each j value from 0 to dllCopy[i] size
            initialize value as dllCopy[i][j]

            call hash() method with value as param and use it's return
```

```
                    as dlls' index as the value is inserted by
                    DoublyLinkedList.insert() method

                    set n to n + 1
            endfor
        endfor

method insert(parameters: number to value): Boolean
        if n value is equal to max value
            call expand() method
        endif

        initialize key as hash() method with value as param return
        initialize ret as dlls[key].insert() with value as param return

        if ret is true
            set n to n + 1
        endif

        return ret


method remove(parameters: number to value): Boolean
        initialize key as hash() method with value as param return
        initialize ret as dlls[key].remove() with value as param return

        if ret is true
            set n to n - 1
        endif

        return ret


method search(parameters: number to value): list
```

```
        initialize key as hash() method with value as param return
        initialize pos as dlls[key].search() with value as param return
        initialize element as dlls[key][pos]
        if pos is false
                return false
        endif
        return [key, pos, element]

method clear(): void
        set dlls list to empty list
        for each bucket in HashTable
                add new DoublyLinkedList instance in dlls list
        endfor
        set n to 0
```

3. Vantagens e desvantagens

3.1 Vantagens

Uma das grandes vantagens de se utilizar tabelas hash é sua alta velocidade. O fato de conseguirmos acessar diretamente o valor que queremos através de uma regra e isso nos leva diretamente na chave que precisamos acessar mostra que a hash é um algoritmo muito rápido.

3.2 Desvantagens

As colisões em uma tabela hash são praticamente inevitáveis. Isso significa que em um cenário com muitas colisões, o hashing acaba ficando ineficiente pois ainda demanda de muito recurso de uma LDDE.

Para essa situação que pensamos em fazer um hashing dinâmico, quando a tabela iguala o número de elementos ao número de chaves hash, dobramos o tamanho da tabela e refazemos as regras para “dispersar” os valores dentro de uma mesma LDDE na posição da hash, otimizando o código.