

Algoritmos e Complexidade

LEI (2º ano)

8ª Ficha Prática

Ano Lectivo de 2011/12

O objectivo desta ficha é o estudo de tabelas de hash.

1. Para implementar tabelas de hash usando o método de *open addressing* considere as seguintes declarações:

```
#define HASHSIZE    31        // número primo
#define EMPTY       ""
#define DELETED     "-"
```

```
typedef char KeyType[9];
typedef void *Info;
```

```
typedef struct entry {
    KeyType key;
    Info info;
} Entry;
```

```
typedef Entry HashTable[HASHSIZE];
```

- (a) Implemente as seguintes funções

```
int Hash(KeyType);           // função de hash
void InitializeTable(HashTable); // inicializa a tabela de hash
void ClearTable(HashTable);   // limpa a tabela de hash
```

- (b) Use o método *linear probing* na implementação das seguintes funções.

```
// insere uma nova associação entre uma chave nova e a restante informação
void InsertTable_LP(HashTable, KeyType, Info);
```

```
// apaga o elemento de chave k da tabela
void DeleteTable_LP(HashTable, KeyType);
```

```
// procura na tabela o elemento de chave k, e retorna o índice da tabela
// aonde a chave se encontra (ou -1 caso k não exista)
int RetrieveTable_LP(HashTable, KeyType);
```

- (c) Use agora o método *quadratic probing* na implementação das seguintes funções.

```
// função de hash
int Hash_QP(KeyType, int);
```

```
// insere uma nova associação entre uma chave nova e a restante informação
void InsertTable_QP(HashTable, KeyType, Info);
```

```
// apaga o elemento de chave k da tabela
void DeleteTable_QP(HashTable, KeyType);
```

```

// procura na tabela o elemento de chave k, e retorna o índice da tabela
// aonde a chave se encontra (ou -1 caso k não exista)
int RetrieveTable_QP(HashTable, KeyType);

```

(d) Efectue a análise assintótica do tempo de execução das funções que implementou.

2. Para implementar tabelas de hash usando o método de *chaining* considere as seguintes declarações:

```

#define HASHSIZE    31    // número primo

```

```

typedef char KeyType[9];
typedef void *Info;

```

```

typedef struct entry {
    KeyType key;
    Info info;
    struct entry *next;
} Entry;

```

```

typedef Entry *HashTable[HASHSIZE];

```

(a) Apresente uma implementação para as seguintes funções.

```

// função de hash
int Hash(KeyType);

// inicializa a tabela de hash
void InitializeTable(HashTable);

// limpa a tabela de hash
void ClearTable(HashTable);

// insere uma nova associação entre uma chave nova e a restante informação
void InsertTable(HashTable, KeyType, Info);

// apaga o elemento de chave k da tabela
void DeleteTable(HashTable, KeyType);

// procura na tabela o elemento de chave k, e retorna o apontador
// para a célula aonde a chave se encontra (ou NULL caso k não exista)
Entry *RetrieveTable(HashTable, KeyType);

```

(b) Efectue a análise assintótica do tempo de execução das funções que implementou.

3. Pretende-se agora que faça a implementação de tabelas de hash dinâmicas cujo tamanho do array alocado vai depender do *factor de carga* (n^o de entradas / tamanho da tabela)

(a) Adapte as declarações das estruturas de dados para este fim.

(b) Adapte as funções que definiu nas alíneas anteriores a esta nova implementação. Note que nas funções de inserção e de remoção

- quando o factor de carga é superior ou igual a 75% (50% no caso usar o método *quadratic probing*) o tamanho da tabela é aumentado para o dobro;
- quando o factor de carga é menor ou igual a 25% o tamanho da tabela é reduzido a metade.

(c) Faça a análise de custo amortizado para sequências de N inserções ou N remoções nestas tabelas de hash dinâmicas.