Tabela de dispersão

Programação II – Engenharia de Telecomunicações

Prof. Emerson Ribeiro de Mello

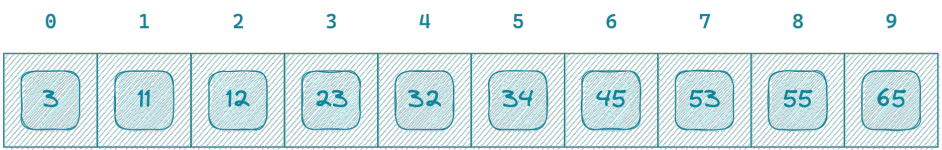
mello@ifsc.edu.br

INSTITUTO FEDERAL
Santa Catarina
Campus São José

Licenciamento

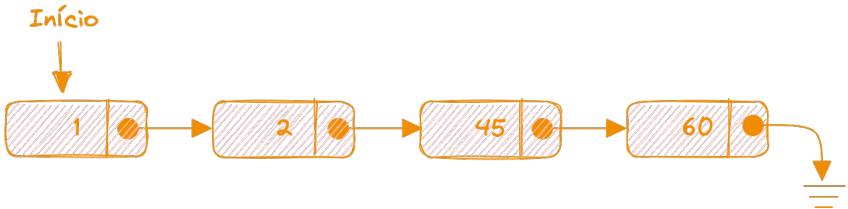
Lista linear em alocação sequencial

- Se pretende armazenar até 100 números inteiros, então deve-se reservar memória para os 100, mesmo que inicialmente tenha-se 0 elementos contidos
- Tem endereçamento direto a cada elemento e assim permite obter elemento em uma posição arbitrária no tempo $O(1)$
  - Exemplo: `int elemento = vetor[7];`



Lista encadeada

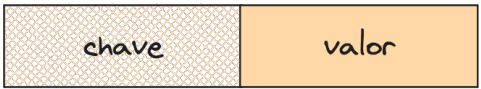
- Tem tamanho dinâmico e assim evita reservar previamente um grande espaço na memória
- Não tem endereçamento direto ao elementos e assim obter o valor do $n-$ésimo elemento tem complexidade $\Theta(n)$ no pior caso
  - Exemplo: Verificar se o valor 60 está armazenado na lista



Dicionários

Tabela de dispersão, Tabela de espalhamento ou ainda *Hash Table*

- Dicionário é uma estrutura de dados que associa chaves a valores



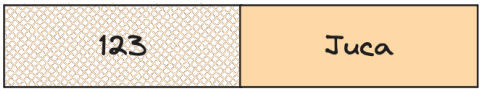
- **Combina as habilidades** de acesso arbitrário das **listas sequenciais** e da alocação dinâmica das **listas encadeadas**
- **Trata-se de uma generalização de arranjo**, sendo que qualquer posição pode ser acessada em tempo constante
  - Operações de inserção, busca[1] e remoção com complexidade $O(1)$
  - Não permite armazenar chaves repetidas

[1] A busca, quando houver **colisões**, tem complexidade $\Theta(n)$ no pior caso

Dicionários

Tabela de dispersão, Tabela de espalhamento ou ainda *Hash Table*

- Dicionário é uma estrutura de dados que associa chaves a valores



- **Combina as habilidades** de acesso arbitrário das **listas sequenciais** e da alocação dinâmica das **listas encadeadas**
- **Trata-se de uma generalização de arranjo**, sendo que qualquer posição pode ser acessada em tempo constante
  - Operações de inserção, busca[1] e remoção com complexidade $O(1)$
  - Não permite armazenar chaves repetidas

[1] A busca, quando houver **colisões**, tem complexidade $\Theta(n)$ no pior caso

Tabela de dispersão

Funcionamento

- Uma **função de espalhamento** (função *hash*) recebe uma chave como entrada e gera como saída o índice no arranjo de tamanho **m**

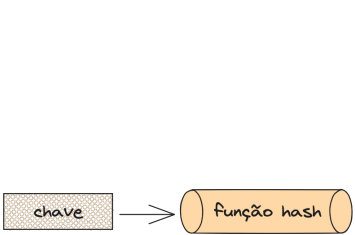


Tabela de dispersão

Funcionamento

- Uma **função de espalhamento** (função *hash*) recebe uma chave como entrada e gera como saída o índice no arranjo de tamanho **m**

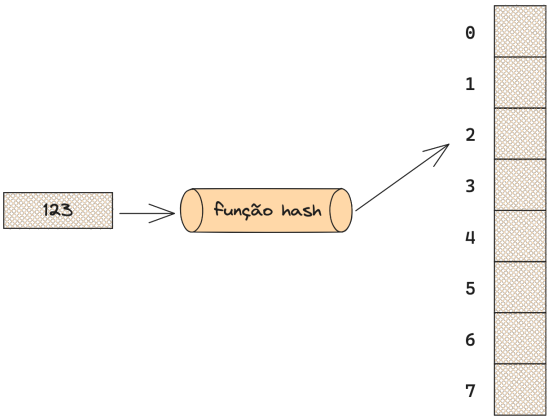


Tabela de dispersão

Funcionamento

- Uma **função de espalhamento** (função *hash*) recebe uma chave como entrada e gera como saída o índice no arranjo de tamanho **m**

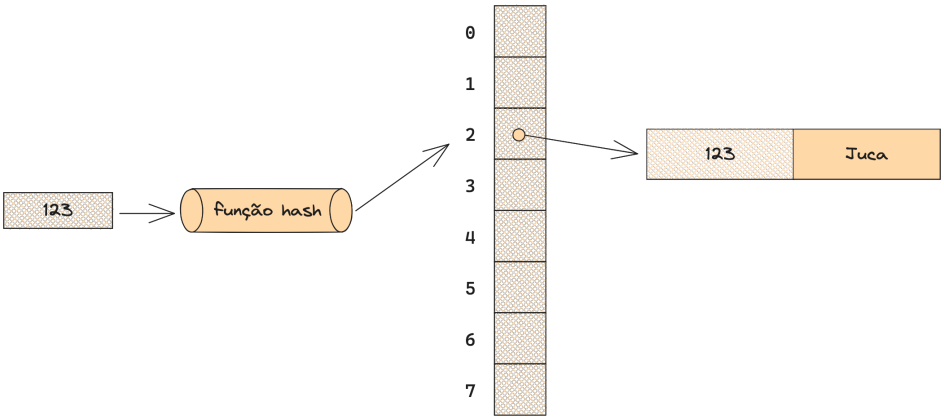


Tabela de dispersão

Funcionamento

- Uma **função de espalhamento** (função *hash*) recebe uma chave como entrada e gera como saída o índice no arranjo de tamanho **m**

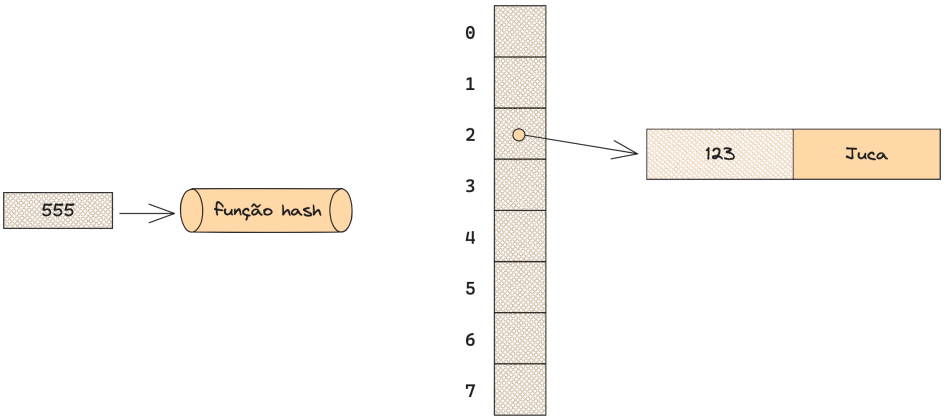


Tabela de dispersão

Funcionamento

- Uma **função de espalhamento** (função *hash*) recebe uma chave como entrada e gera como saída o índice no arranjo de tamanho **m**

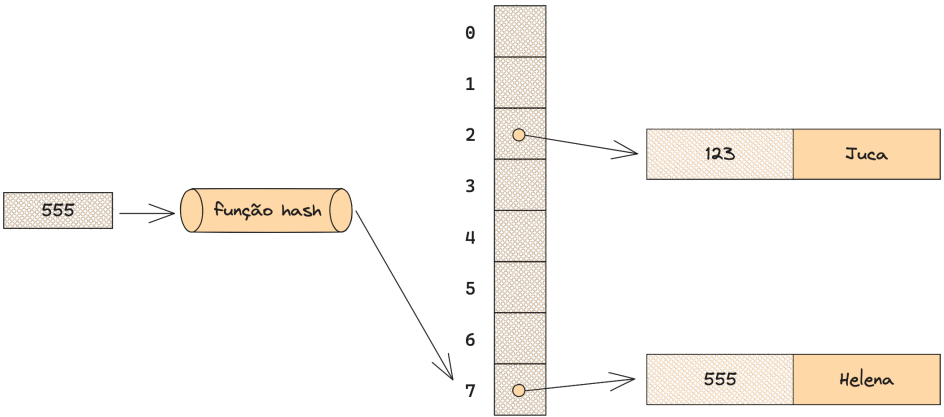


Tabela de dispersão

Funcionamento

- Uma **função de espalhamento** (função *hash*) recebe uma chave como entrada e gera como saída o índice no arranjo de tamanho **m**

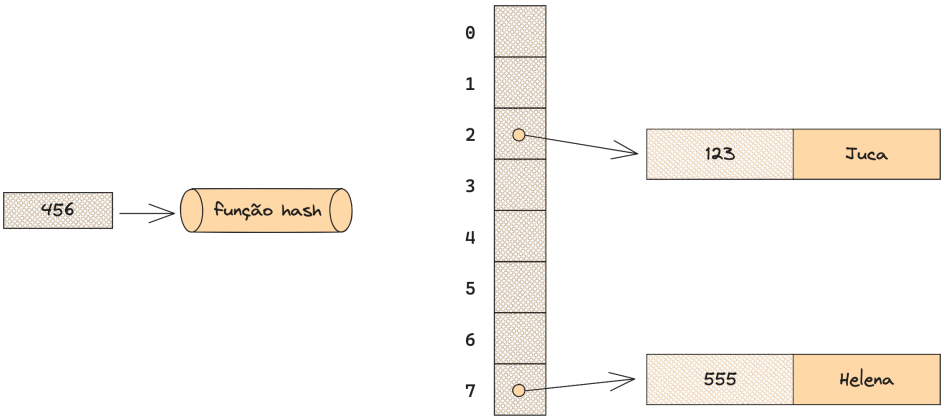


Tabela de dispersão

Funcionamento

- Uma **função de espalhamento** (função *hash*) recebe uma chave como entrada e gera como saída o índice no arranjo de tamanho **m**

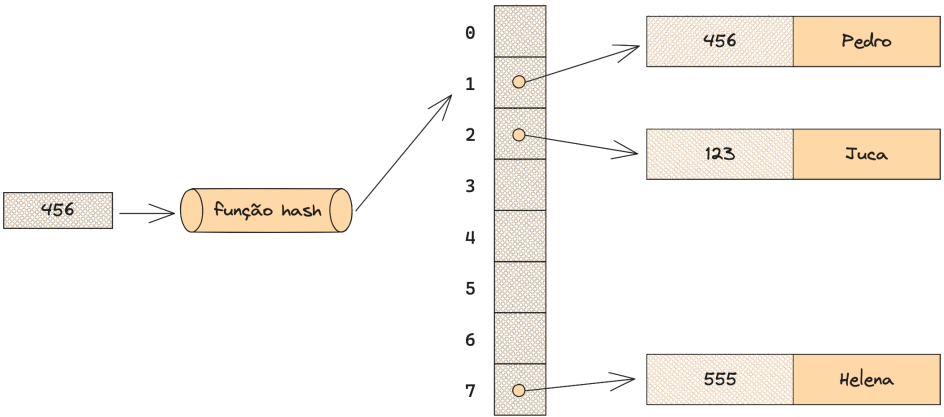
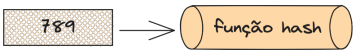


Tabela de dispersão

Funcionamento

- Uma **função de espalhamento** (função *hash*) recebe uma chave como entrada e gera como saída o índice no arranjo de tamanho **m**



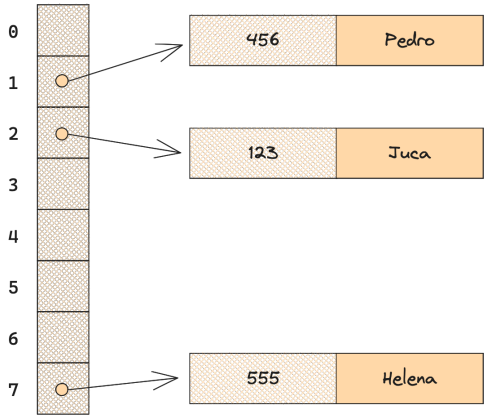
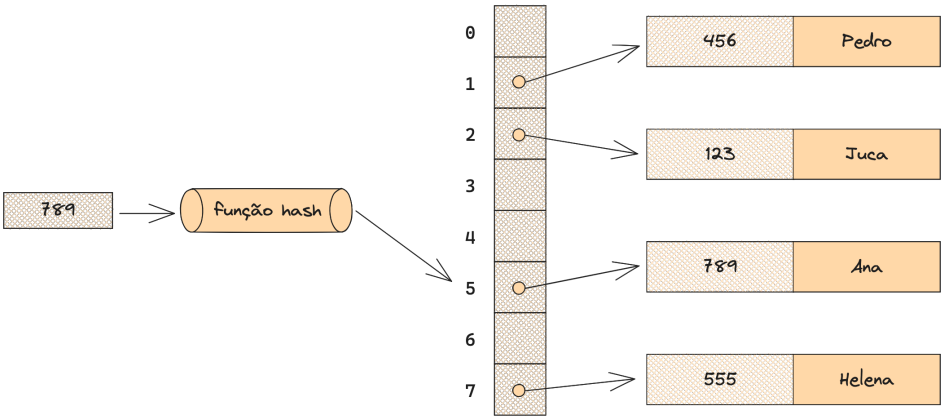


Tabela de dispersão

Funcionamento

- Uma **função de espalhamento** (função *hash*) recebe uma chave como entrada e gera como saída o índice no arranjo de tamanho **m**



Função de espalhamento

Função de resumo, função *hash* ou função de *hashing*

- Algoritmo determinístico que transforma uma entrada de **comprimento variável** (chave) em uma **saída de comprimento fixo** (índice no arranjo)
- Mapeia chaves distintas para índices distintos, fazendo o espalhamento das chaves pelo arranjo
- Na equação 1 tem-se uma função de espalhamento h que recebe uma chave **k** e retorna o resto da divisão de **k** por **m**, sendo **m** o tamanho do arranjo e **k** um número natural

$$h(k) = k \mod m, \quad m \in \mathbb{N} \tag{1}$$

- Com um espalhamento perfeito, toda operação de inserção ou remoção é feita em $O(1)$

Função de espalhamento

Colisões

Colisão

Quando chaves distintas, ao passarem por uma função de espalhamento, resultarem em um mesmo índice no arranjo

- Na prática o espalhamento perfeito não é possível, assim é necessário prover uma solução para quando isso ocorrer
  1. Combinar com outra estrutura de dados (*e.g.* lista encadeada)
  2. Usar endereçamento aberto com busca linear,
     - Insere elemento na próxima posição livre do vetor
     - Apresenta como limitações o agrupamento de valores e não garante um espalhamento uniforme
  3. Usar endereçamento aberto com busca quadrática

Tabela de dispersão

Lidando com colisões usando lista encadeada

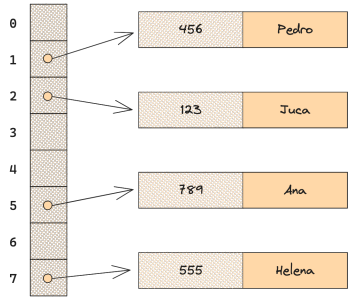


Tabela de dispersão

Lidando com colisões usando lista encadeada

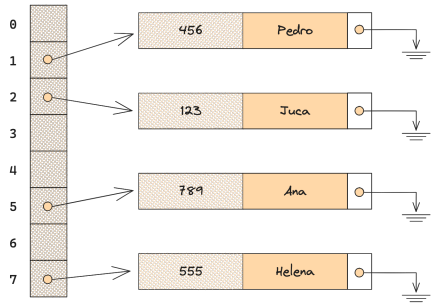


Tabela de dispersão

Lidando com colisões usando lista encadeada

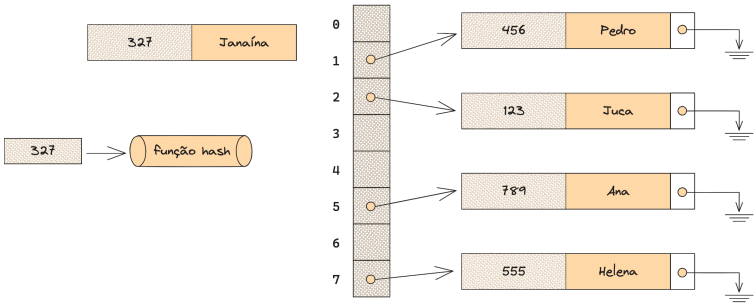


Tabela de dispersão

Lidando com colisões usando lista encadeada

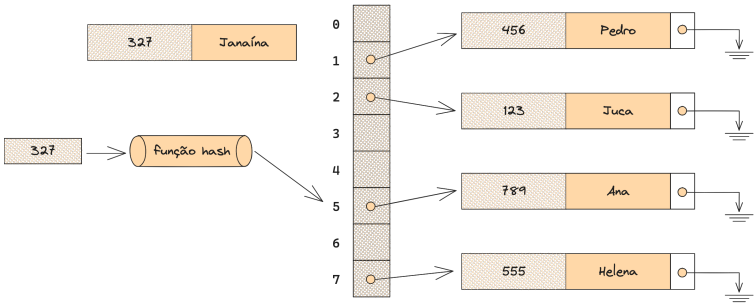
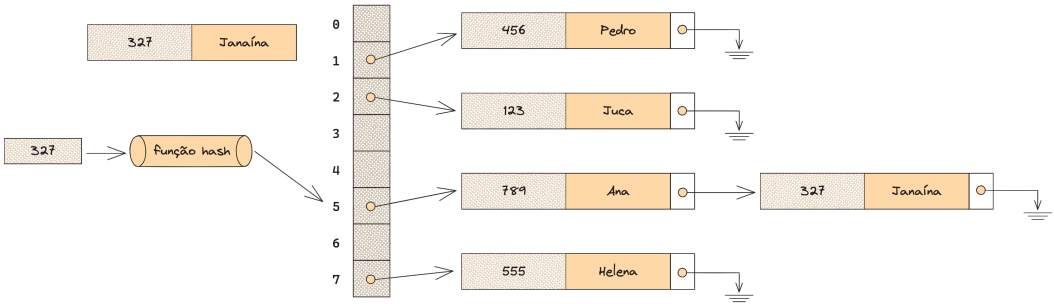


Tabela de dispersão

Lidando com colisões usando lista encadeada



Funções de espalhamento para tabelas de dispersão

Método da divisão (ou congruência linear) para chaves numéricas

- Divida a chave **k** pelo tamanho do arranjo **m** e use o resto da divisão como índice

$$h(k) = k \mod m, \quad m \in \mathbb{N}$$

- Para ter um espalhamento mais uniforme das chaves é necessário que **m** seja um número primo distante de pequenas potências de 2

```
pessoa_t juca = {20231113, "Juca", "juca@example.org"};
int chave = 20231113;
int m = 5013;

int posicao = chave % m; // 20231113 % 5013 = 3658

tabela[posicao] = juca;
```

Funções de espalhamento para tabelas de dispersão

Método da divisão para chaves alfanuméricas

- Converta uma chave alfanumérica em um número natural
- Faça a soma ponderada dos códigos ASCII de cada caractere que compõe a chave para evitar que permutações de uma chave gerem um mesmo *hash*
  - A posição do caractere multiplicado por seu código ASCII

```
int hash(char *chave, int m){
    int soma = 0;
    for(int i = 0; chave[i] != '\0'; i++){
        soma += (i+1) * chave[i];
    }
    return soma % m;
}

int posicao = hash("juca@example.org");
```

Implementação na linguagem C

Agenda de contatos usando dicionários

Tipos abstratos de dados

```c
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct {
    char *cpf;
    char *nome;
    char *email;
} pessoa_t;

typedef struct no {
    char *chave;
    pessoa_t  *valor;
    struct no *prox;
} no_t;

typedef struct dicionario {
    int  tamanho; // tamanho do vetor
    no_t **vetor;
} dicionario_t;
```

Agenda de contatos usando dicionários

Função para criar dicionário

```
dicionario_t *criar_dicionario(int m) {
    dicionario_t *d = NULL;
    if (m < 1) {
        return NULL;
    }
    if ((d = (dicionario_t *)malloc(sizeof(dicionario_t))) == NULL) {
        return NULL;
    }
    d->tamanho = m;
    if ((d->vetor = calloc(m, sizeof(no_t *))) == NULL) {
        return NULL;
    }
    // O calloc já deve colocar NULL, mas deixo aqui de forma explícita
    for (int i = 0; i < m; ++i) {
        d->vetor[i] = NULL;
    }
    return d;
}
```

Agenda de contatos usando dicionários

Funções para liberar memória reservada e evitar o vazamento de memória

```c
void destruir_pessoa(pessoa_t *pessoa) {
    if (pessoa->cpf != NULL) {
        free(pessoa->cpf);
    }
    if (pessoa->nome != NULL) {
        free(pessoa->nome);
    }
    if (pessoa->email != NULL) {
        free(pessoa->email);
    }
    free(pessoa);
}

void destruir_no(no_t *no) {
    if (no != NULL) {
        free(no->chave);
        destruir_pessoa(no->valor);
        free(no);
    }
}
```

Agenda de contatos usando dicionários

Funções para liberar memória reservada e função hash

```c
void destruir_dicionario(dicionario_t *d) {
    if (d != NULL) {
        for (int i = 0; i < d->tamanho; ++i) {
            // TODO destruir lista encadeada
            destruir_no(d->vetor[i]);
        }
        free(d->vetor);
        free(d);
    }
}

// método por divisão com chave alfanumérica
int hash(const char *chave, int m) {
    int soma = 0;
    for (int i = 0; chave[i] != '\0'; i++) {
        soma += (i + 1) * chave[i];
    }
    return soma % m;
}
```

Agenda de contatos usando dicionários

Função para inserir na tabela

```
bool inserir(dicionario_t *d, char *chave, pessoa_t *valor) {
    int indice = hash(chave, d->tamanho);
    no_t *no = malloc(sizeof(no_t));
    if (no == NULL) {
        return false;
    }
    // strdup reserva memória para fazer a cópia da string. Presente em string.h padrão
    // C23 https://en.cppreference.com/w/c/string/byte/strdup
    no->chave = strdup(chave);
    if (no->chave == NULL) {
        free(no);
        return false;
    }
    no->valor = valor;
    // TODO não está tratando colisões
    //  se houver colisão é necessário usar uma lista encadeada
    no->prox = NULL;
    // libera a memória se existir um nó anterior na posição
    destruir_no(d->vetor[indice]);
    d->vetor[indice] = no;
    return true;
}
```

Agenda de contatos usando dicionários

Funções para buscar e imprimir pessoa

```
pessoa_t *buscar(dicionario_t *d, char *chave) {
    int indice = hash(chave, d->tamanho);
    if (d->vetor[indice] != NULL) {
        // TODO Abaixo só pega o primeiro elemento da lista encadeada
        //  é necessário percorrer a lista encadeada e não apenas o primeiro
        //  elemento
        if (strcmp(d->vetor[indice]->chave, chave) == 0) {
        return d->vetor[indice]->valor;
        }
    }
    return NULL;
}

void imprimir_pessoa(dicionario_t *d, char *chave) {
    pessoa_t *p = buscar(d, chave);
    if (p != NULL) {
        printf("CPF: %s\tNome: %s\tEmail: %s\n", p->cpf, p->nome, p->email);
    } else {
        printf("Pessoa com CPF %s não encontrada\n", chave);
    }
}
```

Agenda de contatos usando dicionários

Função `main`

```c
int main(int argc, char **argv) {
    dicionario_t *dicionario = criar_dicionario(5);
    if (dicionario == NULL) {
      printf("Não foi possível reservar memória\n");
      exit(EXIT_FAILURE);
    }

    pessoa_t *p = malloc(sizeof(pessoa_t));
    // https://en.cppreference.com/w/c/string/byte/strdup
    // strdup reserva memória para fazer a cópia da string. Presente no padrão C23
    p->cpf = strdup("123");
    p->nome = strdup("Juca");
    p->email = strdup("juca@example.ogr");
    inserir(dicionario, p->cpf, p);

    imprimir_pessoa(dicionario, "123");

    destruir_dicionario(dicionario);

    return 0;
}
```

Caso seu compilador C não tenha suporte a função `strdup`

https://en.cppreference.com/w/c/string/byte/strdup

- A função abaixo pode ser usada para copiar cadeias de caracteres que são alocadas dinamicamente na memória

```c
char * copia_string(char *s) {
    char *copia = malloc(sizeof(char) * (strlen(s) + 1));
    if (copia == NULL) {
        return NULL;
    }
    strcpy(copia, s);
    return copia;
}

int main(){
    char *cpf = copia_string("123");
    free(cpf);

    return 0;
}
```

Exercícios

Exercício 1

Estudar o código, compilar e executar

1. Compile o código fornecido e use o valgrind para verificar se tem vazamento de memória

```
gcc main.c -o primeiro
valgrind --leak-check=yes ./primeiro
```

2. Compile o código fornecido com parâmetros adicionais do gcc para verificar se tem vazamento de memória

```
gcc -Wall -g -fsanitize=address main.c -o segundo
./segundo
```

Exercício 2

Corrigir o código fornecido para tratar colisões

As funções `destruir_dicionario`, `inserir` e `buscar` possuem um comentário `TODO` indicando que a implementação fornecida não trata colisões

1. Corrija o código fornecido para permitir tratar colisões por meio de uma lista encadeada
2. Use o valgrind e o gcc para garantir que sua solução não apresente vazamento de memória

Exercício 3

Implementar novas funcionalidades

1. Remoção de elemento no dicionário
   - Deve procurar pela chave e retornar `true` se conseguiu remover ou `false`, caso a chave não esteja na tabela
2. Exibir todas informações armazenadas no dicionário. No exemplo abaixo o dicionário tem tamanho 3 e contém 3 elementos, sendo que houve uma colisão na posição `1` e não tem elementos na posição `2`

```
0: [(123, Juca, juca@example.org)]
1: [(456, Ana, ana@example.org), (789, Pedro, pedro@example.org)]
2: []
```

3. Use o valgrind e o gcc para garantir que sua solução não apresente vazamento de memória

Referências

Aula baseada em

LAGO PEREIRA, Silvio do. **Estruturas de Dados em C - Uma Abordagem Didática**. Saraiva, 2016. ISBN 9788536517254. Disponível em: <https://app.minhabiblioteca.com.br/#/books/9788536517254>. Acesso em: 1 nov. 2023.