PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

Marcos Vinicius Araujo Almeida - 1910869
Breno Azevedo Marot - 1910423

Hashing

Tarefa 3

Rio de Janeiro, RJ
2021

Criando a função de hashing

Escolhemos utilizar o método da divisão, já que o tamanho do vetor a ser armazenado é um número primo, oque contribui bastante para evitar colisões nesse tipo de dispersão. Para esse tipo de dispersão, temos que pensar na string de entrada como um somatório, da seguinte maneira:

$$\sum a_i.b$$

Onde a^i é o i-nésimo caractere, multiplicado por b, uma base escolhida para ser maior do que o número de caracteres e dígitos. Para esse trabalho, escolhemos a base como sendo 256, e percebemos também que com esse resultado o número de colisões foi menor.

Implementando o código em C, temos:

```c
unsigned int hash(char *s)
{
    unsigned int h; // valor do hash
    unsigned const char *us;

    us = (unsigned const char *)s;

    h = 0;
    while (*us != '\0') //percorrendo a string
    {
        h = (h * BASE + *us) % MAX_PLACAS; //mod m
        us++;
    }

    return h;
}
```

- h representa o valor a ser retornado da função

Tratando as colisões

As colisões foram tratadas com o método do encadeamento interior, que consiste, após ocorrer uma colisão, encontrar a casa mais próxima vazia (livre para ser inserida), mantendo sempre o resultado dentro do limite de tamanho do vetor. (isso explica o uso do **mod** após cada interação)

```c
void insere(char placa[])
{
    unsigned int pos = hash(placa);

    if (strcmp(arr[pos], "xxxxxxx") != 0)
    {
        colisoes++;
        //printf("[%s] colidiu com [%s] (pos: %d)\n", placa, arr[pos], pos);

        pos_colididas[pos]++;

        for (int i = (pos + 1) % NUM_PLACAS;; i++)
        {

            if (i == NUM_PLACAS)
            {
                i = 0;
                continue;
            }
            //primeira casa vazia
            if (strcmp(arr[i], "xxxxxxx") == 0)
            {
                strcpy(arr[i], placa);
                break;
            }
        }
    }

    //sem colisões, livre para inserir
    else
    {
        strcpy(arr[pos], placa);
    }
}
```

O vetor *pos_colididas* existe para podermos mapear onde as colisões estão acontecendo. Por exemplo, se uma colisão ocorre na posição 0 do vetor *arr*, acessamos essa mesma posição no vetor *pos_colididas* e incrementamos o valor. É necessário também seja salvo o número de colisões totais. Como comentado anteriormente, após ocorrer uma colisão buscamos a posição mais próxima no vetor que esteja vazia, e inserimos o elemento colidido nesse local. É possível que a próxima posição livre esteja antes do local colidido, sendo assim temos que “dar a volta” no vetor (usando o operador **mod**) para podermos preencher.

Como resultado dessas funções (*hash* e *insere*), tivemos uma saída com algumas colisões.

```
Pos: 940  Colisões-locais: 2
Pos: 946  Colisões-locais: 1
Pos: 951  Colisões-locais: 1
Pos: 956  Colisões-locais: 1
Pos: 960  Colisões-locais: 2
Pos: 962  Colisões-locais: 1
Pos: 963  Colisões-locais: 2
Pos: 965  Colisões-locais: 1
Pos: 966  Colisões-locais: 1
Pos: 969  Colisões-locais: 1
Pos: 971  Colisões-locais: 1
Pos: 977  Colisões-locais: 1
Pos: 978  Colisões-locais: 2
Pos: 980  Colisões-locais: 1
Pos: 981  Colisões-locais: 1
Pos: 982  Colisões-locais: 1
Pos: 992  Colisões-locais: 1
Pos: 994  Colisões-locais: 1
Pos: 995  Colisões-locais: 1
Pos: 999  Colisões-locais: 1
Pos: 1001  Colisões-locais: 1
Pos: 1002  Colisões-locais: 1
Pos: 1003  Colisões-locais: 2
Pos: 1010  Colisões-locais: 3
Pos: 1012  Colisões-locais: 1
Pos: 1013  Colisões-locais: 1
Pos: 1017  Colisões-locais: 1
Pos: 1028  Colisões-locais: 2
Pos: 1030  Colisões-locais: 1
Pos: 1037  Colisões-locais: 1
Pos: 1040  Colisões-locais: 2
Pos: 1042  Colisões-locais: 1
Pos: 1046  Colisões-locais: 1
Pos: 1047  Colisões-locais: 1
Pos: 1049  Colisões-locais: 2
Pos: 1050  Colisões-locais: 1
Pos: 1058  Colisões-locais: 2
Pos: 1059  Colisões-locais: 1
Pos: 1065  Colisões-locais: 2
Pos: 1066  Colisões-locais: 1
Pos: 1073  Colisões-locais: 1
Pos: 1077  Colisões-locais: 1
Pos: 1079  Colisões-locais: 1
Pos: 1080  Colisões-locais: 3
Pos: 1081  Colisões-locais: 1
Pos: 1084  Colisões-locais: 2
Pos: 1088  Colisões-locais: 1
Pos: 1107  Colisões-locais: 2
Pos: 1108  Colisões-locais: 3
==========> TOTAL DE COLISÕES:: 447
```

O programa, a paritir do vetor *pos_colididas*, nos possibilitou visualizar melhor onde e quantas colisões ocorreram em determinado lugar. Ao inserir as placas no vetor, o programa gera uma mensagem se a inserção gerou alguma colisão.

Vale lembrar que o vetor *pos_colididas* não é necessário para a funcionamento do programa, inclusive na versão final do programa enviado para o EAD, ele não foi implementado. Ele representa apenas um jeito diferente de olharmos para as dispersões.

Programando os tempos de inserção

Para testar os tempos de inserção, foi utilizado a função da biblioteca <sys/time.h>, gettimeofday, que foi chamda para contar o tempo de cada “rajada” de inserções (N = 200, 300, 400 ...). Seu código ficou da seguinte maneira:

```
void preencheVetorTempos()
{
    arr = init();

    initVetorTempos();

    for (int i = 1; i <= 10; i++)
    {
        struct timeval start, end;

        colisoes = 0;

        reseta(arr);

        gettimeofday(&start, NULL);

        leArquivo(i * 100);

        gettimeofday(&end, NULL);
        printf("==========> TOTAL DE COLISÕES:: %d\n", colisoes);
        long seconds = (end.tv_sec - start.tv_sec);
        long micros = ((seconds * 1000000) + end.tv_usec) - (start.tv_usec);

        //printf("The elapsed time is %ld seconds and %ld micros\n", seconds, micros);

        tempos[i - 1] = micros;
    }
}
```

Nessa função lemos o arquivo 10 vezes, aumentando a quantidade de elementos inseridas em cada interação. Vale lembrar que o tamanho do vetor à ser inserido sempre possuiu 1109 posições, como foi pedido no enunciado. Após salvarmos esses tempos num vetor, exibimos esse resultados.

Arquivo de testes

O programa foi testado com 2 arquivos direntes. Um criado por nós e outro fornecido pelo professor da disciplina. Os testes que serão mostrados nesse relatório são em relação ao arquivo fornecido externamente.

Testando os tempos de inserção

Para testar os temos de inserção, foi necessário que fosse desconectado meu acesso à internet e finalizado todos os processos que estavam rodando em paralelo.

Configurações do computador utilizado para os testes:

- Sistema operacional: **Microsoft Windows 10 Pro**
- Placa mãe: **TUF H310M-PLUS GAMING/BR**
- Memória RAM instalada: **16,00 GB**
- Processador: **Intel® Core™ i7-9700KF CPU @ 3.600GHz, 8 núcleos**

Testes na próxima página →

Teste 1

```
marcos@DESKTOP-QAA5JC2:/mnt/d/Documentos/2021.2/Estruturas de Dados/Trabalhos/trab3$ ./teste
==========> TOTAL DE COLISÕES:: 3
==========> TOTAL DE COLISÕES:: 23
==========> TOTAL DE COLISÕES:: 44
==========> TOTAL DE COLISÕES:: 80
==========> TOTAL DE COLISÕES:: 122
==========> TOTAL DE COLISÕES:: 175
==========> TOTAL DE COLISÕES:: 230
==========> TOTAL DE COLISÕES:: 300
==========> TOTAL DE COLISÕES:: 373
==========> TOTAL DE COLISÕES:: 459
Com 100 elementos -> 547 mcirosegundos
Com 200 elementos -> 365 mcirosegundos
Com 300 elementos -> 366 mcirosegundos
Com 400 elementos -> 373 mcirosegundos
Com 500 elementos -> 470 mcirosegundos
Com 600 elementos -> 487 mcirosegundos
Com 700 elementos -> 507 mcirosegundos
Com 800 elementos -> 503 mcirosegundos
Com 900 elementos -> 532 mcirosegundos
Com 1000 elementos -> 805 mcirosegundos
```

Terminal 1.0 – Resultado 1

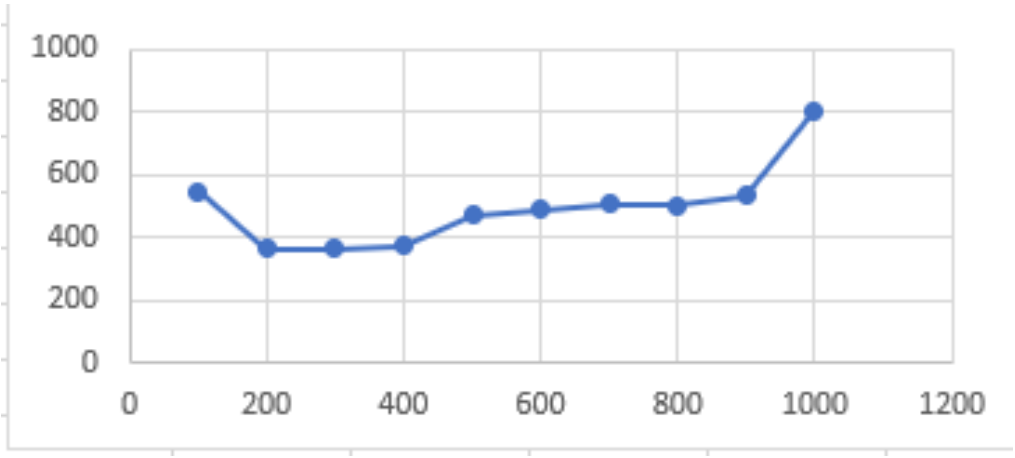

Gráfico 1.0

Teste 2

```
marcos@DESKTOP-QAA5JC2:/mnt/d/Documentos/2021.2/Estruturas de Dados/Trabalhos/trab3$ ./teste
==========> TOTAL DE COLISÕES:: 3
==========> TOTAL DE COLISÕES:: 23
==========> TOTAL DE COLISÕES:: 44
==========> TOTAL DE COLISÕES:: 80
==========> TOTAL DE COLISÕES:: 122
==========> TOTAL DE COLISÕES:: 175
==========> TOTAL DE COLISÕES:: 230
==========> TOTAL DE COLISÕES:: 300
==========> TOTAL DE COLISÕES:: 373
==========> TOTAL DE COLISÕES:: 459
Com 100 elementos -> 757 mcirosegundos
Com 200 elementos -> 494 mcirosegundos
Com 300 elementos -> 472 mcirosegundos
Com 400 elementos -> 487 mcirosegundos
Com 500 elementos -> 604 mcirosegundos
Com 600 elementos -> 659 mcirosegundos
Com 700 elementos -> 646 mcirosegundos
Com 800 elementos -> 657 mcirosegundos
Com 900 elementos -> 677 mcirosegundos
Com 1000 elementos -> 1004 mcirosegundos
```

Terminal 2.0

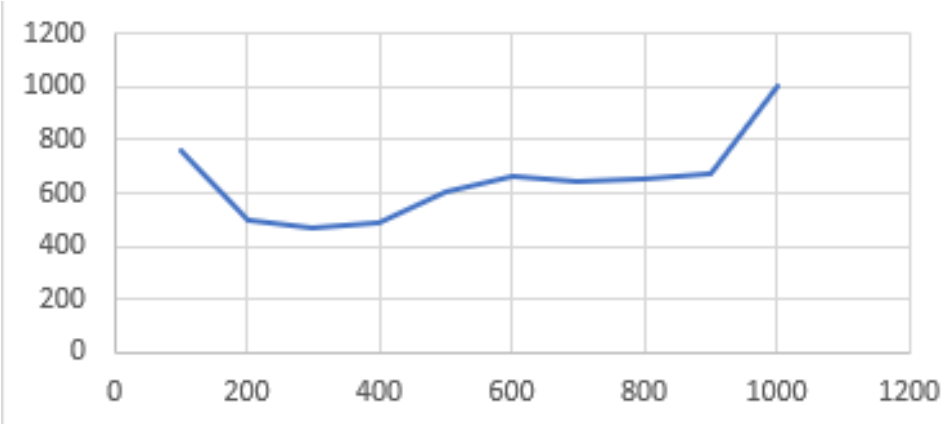

Gráfico 2.0

Teste 3

```
marcos@DESKTOP-QAA5JC2:/mnt/d/Documentos/2021.2/Estruturas de Dados/Trabalhos/trab3$ ./teste
==========> TOTAL DE COLISÕES:: 3
==========> TOTAL DE COLISÕES:: 23
==========> TOTAL DE COLISÕES:: 44
==========> TOTAL DE COLISÕES:: 80
==========> TOTAL DE COLISÕES:: 122
==========> TOTAL DE COLISÕES:: 175
==========> TOTAL DE COLISÕES:: 230
==========> TOTAL DE COLISÕES:: 300
==========> TOTAL DE COLISÕES:: 373
==========> TOTAL DE COLISÕES:: 459
Com 100 elementos -> 696 mcirosegundos
Com 200 elementos -> 468 mcirosegundos
Com 300 elementos -> 482 mcirosegundos
Com 400 elementos -> 485 mcirosegundos
Com 500 elementos -> 616 mcirosegundos
Com 600 elementos -> 618 mcirosegundos
Com 700 elementos -> 642 mcirosegundos
Com 800 elementos -> 640 mcirosegundos
Com 900 elementos -> 670 mcirosegundos
Com 1000 elementos -> 1007 mcirosegundos
```

Terminal 3.0

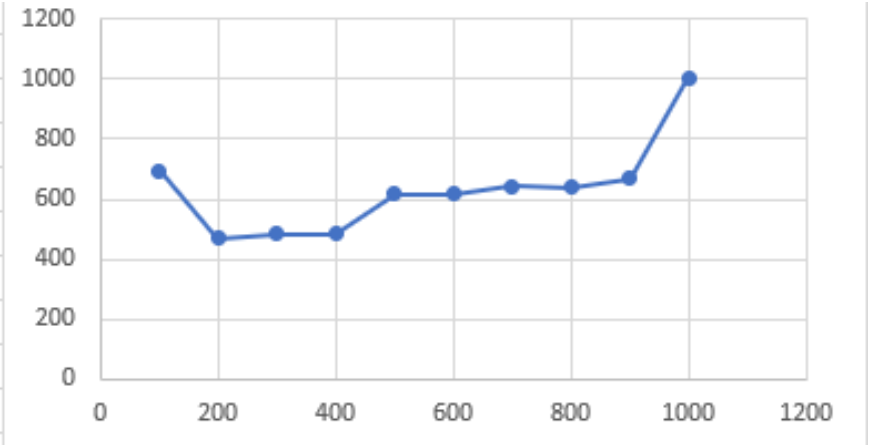

Gráfico 3.0

Teste 4

```
marcos@DESKTOP-QAA5JC2:/mnt/d/Documentos/2021.2/Estruturas de Dados/Trabalhos/trab3$ ./teste
==========> TOTAL DE COLISÕES:: 3
==========> TOTAL DE COLISÕES:: 23
==========> TOTAL DE COLISÕES:: 44
==========> TOTAL DE COLISÕES:: 80
==========> TOTAL DE COLISÕES:: 122
==========> TOTAL DE COLISÕES:: 175
==========> TOTAL DE COLISÕES:: 230
==========> TOTAL DE COLISÕES:: 300
==========> TOTAL DE COLISÕES:: 373
==========> TOTAL DE COLISÕES:: 459
Com 100 elementos -> 656 mcirosegundos
Com 200 elementos -> 447 mcirosegundos
Com 300 elementos -> 435 mcirosegundos
Com 400 elementos -> 509 mcirosegundos
Com 500 elementos -> 563 mcirosegundos
Com 600 elementos -> 582 mcirosegundos
Com 700 elementos -> 599 mcirosegundos
Com 800 elementos -> 604 mcirosegundos
Com 900 elementos -> 614 mcirosegundos
Com 1000 elementos -> 1049 mcirosegundos
```

Terminal 4.0

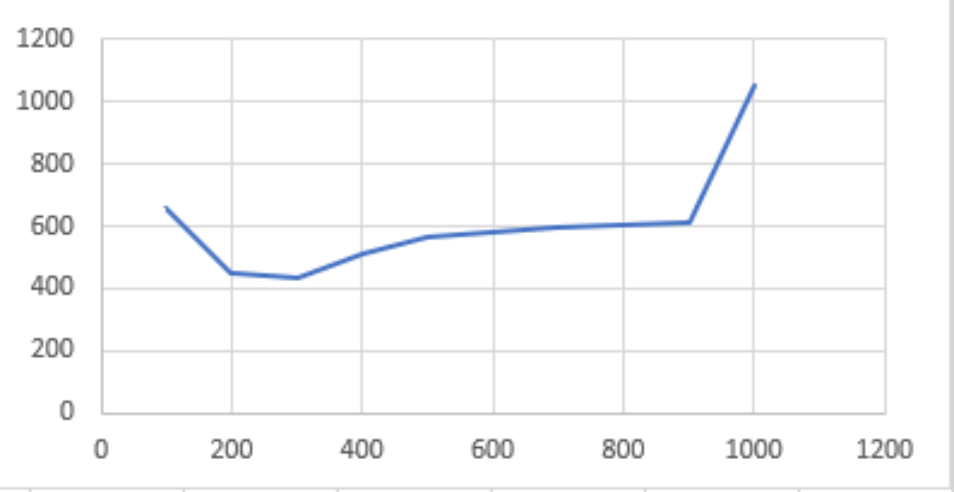

Gráfico 4.0

OBS: O **eixo X** dos gráficos representam o **tamanho da entrada** e o **eixo Y** representam os **tempos de execução**.

Conclusão

Em ambos os gráficos obtivemos resultados satisfatórios. As colisões em todos os testes não ultrapassaram 50% de colisões em relação ao tamanho do

vetor. A média de colisões, para o número máximo de placas lidas, foi em torno de 450 colisões.

Observamos que a primeira interação, que se refere a N=100, sempre apresenta um valor mais elevado do que o normal, porém nos próximos valores de N, percebemos que esse valor vai se “ajeitando”.

Os gráficos encontrados se assemelham bastante com um nível de complexidade **O(n)**, sendo n o número da amostra, que representa o pior caso. Isso pode ter se dado devido às placas aleatórias. Entre os valores de 400 até 900 na amostra o gráfico se aproxima da complexidade **O(1)**, que seria a complexidade ideal de uma tabela *Hash*.