Universidade Federal do Rio Grande do Sul
INF01124 – Classificação e Pequisa de Dados
Leonardo Barros Bilhalva e Santiago Gonzaga

Instituto de Informática
2021/1

Laboratório 04

21 de outubro de 2021

Explicação da função hash e resolução de conflitos

A função hash escolhida foi a polinomial. Devido ao fato de que a ordem importa, a função polinomial é fundamental para a interpretação de strings. O objetivo do programa é mapear strings de nomes para uma tabela, então essa função se mostra necessária. O tratamento de conflitos foi o com endereçamento fechado do tipo encadeamento. A resolução de conflitos desse tipo se mostra viável, pois é fácil de ser implementada e consegue propor uma uniformidade para a função hash. Essa uniformidade não conseguirá ser devidamente evidenciada no programa, pois a função que ”redistribui” os elementos encadeados na tabela não foi requisitada pelo professor. A escolha para a resolução de conflitos por encadeamento partiu do professor.

Função make_hash_table:

```
void make_hash_table(vector<string> vetor_de_nomes, vector<estrutura_tabela_hash *> &tabela_hash, int tamanho_tabela)  // Função que cria uma hash_table em tabela_hash a partir do vetor_de_nomes
```

A função é inicializada com vetor de nomes, que carrega os nomes acumulados no arquivo base, tabela hash, que é a tabela que será criada, e tamanho tabela, que é o tamanho ta tabela hash.

```
int numero = 0;                                 //numero será o caractere lido convertido para a tabela criada por mim onde | 1 == " " (espaço é 
long long hash = 0;                             //somatorio para hash de cada nome

for(int a = 0; a<vetor_de_nomes.size(); a++)        //percorre lista de nomes
{
    hash = 0;
    for(int b=0; b<vetor_de_nomes[a].size(); b++)   //percorre caracteres do nome atual
    {
        numero = 0;
        if(vetor_de_nomes[a][b] >= 'A' || vetor_de_nomes[a][b] <= 'Z')
        numero = vetor_de_nomes[a][b] - 40/*diminui para chegar no 1*/+ 26/*pula os primeiros 26 que sao as minusculas*/+ 1;/*espaço é o primeiro*/
        if(vetor_de_nomes[a][b] >= 'a' || vetor_de_nomes[a][b] <= 'z')
        numero = vetor_de_nomes[a][b] - 60/*diminui para chegar no 1*/+ 1;/*espaço é o primeiro*/
        if(vetor_de_nomes[a][b] == ' ')
        numero = 1;/*espaço é o primeiro*/

        hash = (59 * hash + numero) % tamanho_tabela;               // 59 é o número primo escolhido para potência da string, pois é o maior número 
    }
    // temos um hash pronto para ser alocado em -> long long hash
```

Começamos inicializando as variáveis número e hash que serão utilizadas posteriormente. Após, é feito um for com ”a” para percorrer o total de nomes em vetor de nomes. O segundo for em ”b” foi criado para percorrer a todos os caracteres da palavra atual. Dentro desse segundo for, número será a variável que guardará o tamanho dos caracteres para a função hash. A lógica que criamos para os números foi tentar diminuir o tamanho deles com relação aos originais da tabela ascii. O caractere ” ” será interpretado como 1, pois é o caractere que aparecerá em todas as palavras. As letras minúsculas aparecem em maior quantidade que as maiúsculas. Por isso, elas irão de ”a” a ”z” do número 2 ao 27. Por fim, as letras maiúsculas de 28 a 53. Após escolhido o número que representará o caractere atual, utilizaremos a variável hash para calcular a função hash atual com propriedades modulares. O valor 59 foi escolhido para ”p”, pois ele é o primeiro primo maior do que 53, número máximo representado na tabela criada. O módulo será feito pelo tamanho da tabela atual.

```
if(tabela_hash[hash] != NULL) // Se tabela hash apontar para diferente de nulo, significa que já foi p
{
    estrutura_tabela_hash *pointer = tabela_hash[hash];                    //cria um ponteiro par

    while(pointer->next != NULL)                                           //percorre até chegar
    {
        pointer = pointer->next;
    }

    estrutura_tabela_hash* temp1 = new estrutura_tabela_hash;              // cria um ponteiro te
    temp1->nome = vetor_de_nomes[a];
    temp1->ocupado = true;
    temp1->usado = true;
    temp1->next = NULL;

    pointer->next = temp1;                                                 // o ponteiro do últim
}
else    //caso aponta para nulo -> preencher!
{
    tabela_hash[hash] = new estrutura_tabela_hash;          // aloca uma nova estrutura e preenche
    tabela_hash[hash]->nome    = vetor_de_nomes[a];
    tabela_hash[hash]->ocupado = true;
    tabela_hash[hash]->usado   = true;
    tabela_hash[hash]->next    = NULL;
}
```

Se o ponteiro que o índice hash escolhido para a tabela hash apontar para nulo, significa que estamos tratando de um hash atualmente não instanciado. Então, criaremos a estrutura e alocaremos o conteúdo nela para ser o primeiro ”nodo” da lista. Se o ponteiro não for nulo, significa que já temos um nome alocado para esse hash. Então, precisaremos criar um ponteiro auxiliar para percorrer esse noto até o fim dele. Chegando ao fim, criaremos um novo nodo e alocaremos ele na última posição. Sendo assim, teremos um novo item adicionado a lista encadeada, ou seja, uma colisão bem tratada.