Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências da Computação
SCC0201 — Introdução a Ciências da Computação II

Trabalho 04: Tabela Hash

Professores: Fernando Pereira dos Santos e Moacir Antonelli Ponti

PAEs: Edresson Casanova (C) e Leo Sampaio Ferraz Ribeiro (D)

Monitores: Gabriel Alves Kuabara (B) e Guilherme Amaral Hiromoto (A)

Desenvolva o trabalho sem olhar o de colegas.
Se precisar de ajuda pergunte, a equipe de apoio está aqui por você.

1 Objetivo do Trabalho

Nesta tarefa, seu objetivo é implementar uma tabela hash utilizando hashing com encadeamento.

2 Tabela Hash

Uma tabela hash é uma estrutura de dados usada para armazenar pares de chaves / valor. Ela usa uma função hash para calcular um índice no qual um elemento será inserido ou pesquisado. Usando uma boa função de hash, que evite colisões (atribuir elementos diferentes ao mesmo índice), essa estrutura pode funcionar bem, pesquisando elementos em tempo médio $O(1)$.

Contudo, nenhuma função hash é perfeita e sempre há a possibilidade de ocorrerem colisões. Para tratar colisões existem diversas técnicas, porém uma das mais utilizadas é a resolução de colisão por encadeamento (separate channing). Neste método, cada elemento da tabela hash é na verdade uma lista ligada. Para armazenar um elemento na tabela hash, você deve inseri-lo em uma lista ligada específica. Desse modo, se houver qualquer colisão, os elementos devem ser inseridos na mesma lista ligada.

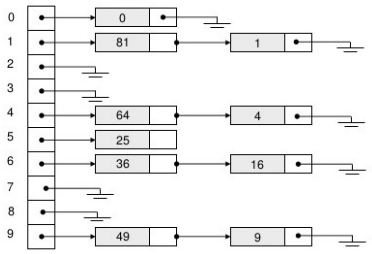

Figure 1: Exemplo de tabela hash com separate channing

3 Proposta

Seu programa deverá ser capaz de realizar inserções, remoções e buscas de palavras S em uma tabela hash com encadeamento de listas. A sua tabela hash deve possuir m listas (índices) e utilizar a seguinte função hash:

$$h(S) = \left(\sum_{i=0}^{|S|-1} S[i]x^i \mod p \right) \mod m$$

onde $|S|$ é o tamanho de cada palavra, $S[i]$ é o código ASCII do i-ésimo caractere de S, $p = 1000000007$ e $x = 263$. Para interagir com a estrutura de dados, o seu programa deve conter as seguintes funções.

- **add** palavra - insere a palavra na tabela hash. Se já existe na tabela, ignora a instrução.
- **del** palavra - remove a palavra na tabela hash. Se não existe na tabela, ignora a instrução.
- **get** i - retorna o conteúdo da i-ésima lista encadeada da tabela hash. Separa os elementos com espaços. Caso a i-ésima lista esteja vazia, imprime uma linha em branco.
- **check** palavra - imprime "sim" caso a palavra exista na tabela ou "não" caso contrário. A procura deve utilizar das propriedades da tabela hash para melhorar a eficiência.

OBS: Quando inserir uma nova palavra na lista encadeada, deve-se inserir no começo da lista.

3.1 Entrada

Há um único inteiro m na primeira linha - o número de listas que você deve ter. A próxima linha contém o número de instruções N. Por fim há N linhas, cada uma delas contém uma instrução no formato descrito acima.

3.2 Saída

Como saída deve-se imprimir o resultado de cada uma das instruções **get** e **check**, um resultado por linha, na mesma ordem conforme essas instruções são fornecidas na entrada.

3.3 Exemplo

Entrada:

5
12
add world
add HellO
get 4
check World
check world
del world
get 4
del HellO
add luck
add GooD
get 2
del good

Saída:

HellO world
nao
sim
HellO
GooD luck

Explicação:
O código ASCII de 'w' é 119, para 'o' é 111, para 'r' é 114, para 'l' é 108 e para 'd' é 100. Assim, h("world") = $(119 + 111 * 263 + 114 * 263^2 + 108 * 263^3 + 100 * 263^4 \mod 1000000007) \mod 5 = 4$. Acontece que o valor hash de "HellO" também é 4. Lembre-se de que sempre inserimos no início da cadeia, então, depois de adicionar "world" e, em seguida, "HellO" no mesmo índice de lista 4, primeiro aparece "HellO" e depois "world". A palavra "World" não é encontrada, e "world" é encontrado, pois o seu programa deve diferenciar as palavras maíusculas e minúsculas então 'W' e 'w' são diferentes. Depois de excluir "world", apenas "HellO" é encontrado na lista 4. Semelhante a "world" e "HellO", após adicionar "luck" e "GooD" a mesma lista 2, deve aparecer na lista em primeiro "GooD" e depois "luck".

4 Submissão

Envie seu código fonte para o run.codes (apenas o arquivo `.c`).

1. **Crie um header com identifiação**. Use um header com o nome, número USP, código do curso e o título do trabalho. Uma penalidade na nota será aplicada se seu código estiver faltando o header.

2. **Comente seu código**. O objetivo é que tenhamos um código claro e que facilite a correção. Exemplos: Se uma variável deixa sua função clara em nome, um comentário sobre ela não é necessário; Um loop triplo que acessa um vetor de matrizes e altera os valores a depender da posição provavelmente pede por um comentário explicando a ideia por trás.

3. **Organize seu código em funções**. Use funções para deixar cada passo da execução mais clara, mesmo que a função seja chamada apenas uma vez. Comente sua função, descrevendo suas entradas e saídas, sempre que o nome da função e entradas não deixar isso óbvio.

4. **Tire Dúvidas com a Equipe de Apoio**. Se não conseguiu chegar em uma solução, dê um tempo para descansar a cabeça e converse com a equipe de apoio sobre a dificuldade encontrada se precisar.