UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL
Faculdade de Computação

Índice baseado em Tabela de Dispersão

Trabalho 1

Estruturas de Dados

1 Descrição

Uma maneira muito comum de tratar colisões em tabelas de dispersão é manter uma lista linear encadeada com cabeça para cada subconjunto de colisões. Dessa forma, o tempo para inserção de um novo elemento na tabela será sempre $O(1)$.

As tabelas de dispersão mais simples são aquelas que armazenam apenas chaves inteiras. Contudo, frequentemente temos chaves não inteiras (p. ex. cadeias de caracteres) e dados satélite associados.

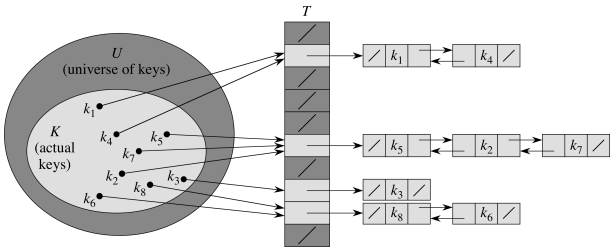

Figura 1: Tabela de dispersão com tratamento de colisões por encadeamento

Indexação de palavras

No contexto de buscas por palavras, a indexação de palavras (ou termos) em textos é uma operação muito comum. Seja para buscas em páginas web ou arquivos em um disco, a indexação está entre as tarefas pré-processamento de dados necessárias para consultas eficientes.

Contudo, ao buscar uma palavra, o armazenamento desse índice em uma estrutura linear pode resultar em um longo tempo de processamento, principalmente se tratando de textos com um grande volume de palavras. Portanto, precisamos de uma estrutura de dados eficiente para tal, de forma que, dada uma palavra, possamos consultar rapidamente as posições em que a palavra ocorre no texto.

Objetivo

Sua tarefa é escrever um programa que, dado um arquivo de texto qualquer, seja capaz de criar um índice de palavras, armazenando-o em uma tabela de dispersão. O **tratamento de colisões** deve ser feito utilizando **listas encadeadas**. Aqui, cada elemento inserido na tabela representa uma palavra, que é sua **chave**. Os elementos são inseridos sempre no final de uma das listas da tabela. Dessa forma, em cada lista encadeada da tabela, serão armazenadas primeiro as palavras que aparecem primeiro no texto.

Além disso, cada elemento terá **dados satélite** que armazenem em quais linhas a palavra aparece. Mais especificamente, cada elemento terá uma **lista encadeada** de inteiros mantidos em **ordem crescente** e **sem repetição**, onde cada inteiro representa uma linha onde a palavra apareceu. Veja o exemplo da Figura 2.

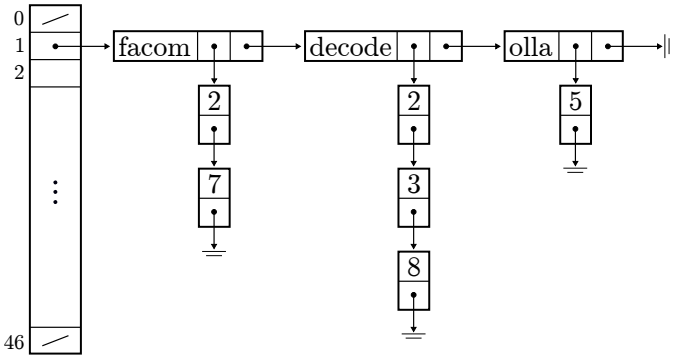

Figura 2: Exemplo de índice de ocorrências de palavras armazenado em uma tabela de dispersão. As palavras "facom", "decode" e "olla" têm todas o valor 1. "facom" ocorre nas linhas 2 e 7 do texto, "decode" ocorre nas linhas 2, 3 e 8 e "olla" apenas na linha 5. Pela ordem dos elementos, podemos deduzir que "facom" aparece primeiro que "decode" na linha 2 do texto.

Queremos também, a fim de verificar as propriedades da tabela de dispersão, imprimir a sequência de compartimentos com seus dados e o fator de carga. Dessa forma, ao final da execução do programa queremos ter uma visão geral da estrutura da tabela de dispersão.

Você deve implementar algum procedimento para, sistematicamente, remover todos os dados da tabela ao finalizar o programa, a fim de garantir que seu programa gerencia de forma eficiente seus recursos de memória.

2 Entrada e saída

Figura 3: Lista Encadeada

O nome do arquivo de entrada é dado pela linha de comando como o primeiro parâmetro, por exemplo `./programa texto.txt`. Você deve inserir na tabela de dispersão todas palavras com 3 ou mais caracteres, juntamente com a linha onde cada palavra ocorre. Cada palavra deve aparecer apenas uma vez na tabela, por isso cada elemento representa uma palavra e guarda, além da palavra em si, uma lista de inteiros que representam as linhas onde a palavra ocorre.

A leitura do arquivo deverá desprezar espaços em branco e sinais de pontuação, que serão considerados separadores de palavras. Além disso, a leitura deverá converter todas as letras maiúsculas em minúsculas, armazenando todas palavras em

letras minúsculas. Portanto versões maiúsculas ou minúsculas de palavras são consideradas idênticas. Você pode considerar que cada palavra contém no máximo 20 letras e que não haverá caracteres acentuados.

A saída referente à tabela de dispersão deve ser escrita da seguinte forma. Considerando que cada uma das 47 posições da tabela possui uma lista de elementos, escreva o conteúdo de cada uma das listas, da lista na posição 0 até a lista na posição 47. Para uma lista em específico, inicie a linha com o seu índice seguido do caractere "`:`", então na mesma linha os elementos na ordem em que aparecem, separados por espaço. Cada elemento por sua vez deve ser escrito no formato `palavra(`l_1`,`l_2`,...)` onde l_1,l_2,... são as linhas onde a palavra ocorre. Por exemplo, a saída referente à posição 1 (palavras com valor 1) da tabela da Figura 2 seria:

```
1:  facom(2,7) decode(2,3,8) olla(5)
```

Não deve haver espaço após o último elemento da lista. Considere que a primeira linha do arquivo é a linha 1. Observe que, caso uma palavra ocorra mais de uma vez na mesma linha, essa linha não deve constar mais de uma vez na lista ocorrências. Considerando novamente o exemplo da Figura 2, mesmo que a palavra "olla" apareça diversas vezes na linha 5, haverá um único valor 5 na lista de ocorrências da palavra.

A última linha do seu programa deve conter o texto "Fator de carga: " seguido do valor correspondente ao fator de carga, impresso com "%g".

3 Exemplo de entrada

```
Olla. Esse eh um arquivo de teste da Facom.
Esse eh um trabalho da disciplina de
estruturas de dados da FACOM.
Para trabalhar com metodos de codificacao
de URLs da facom, utilizamos as funcoes encode e decode.
Trabalho de estruturas de dados.
```

(continua com o exemplo de saída para esta entrada na próxima página...)

4 Exemplo de saída

```
0:
1: olla(1) facom(1,3,5) decode(5)
2:
3: funcoes(5)
4:
5: trabalhar(4)
6: dados(3,6)
7:
8:
9: esse(1,2)
10:
11: metodos(4) encode(5)
12:
13:
14:
15:
16:
17:
18:
19:
20:
21:
22: disciplina(2)
23: arquivo(1)
24: utilizamos(5)
25:
26:
27:
28:
29:
30:
31: urls(5)
32: teste(1)
33:
34:
35:
36:
37: com(4)
38:
39:
40:
41: estruturas(3,6)
42:
43:
44: para(4) codificacao(4)
45:
46: trabalho(2,6)
Fator de carga: 0.404255
```

5 Exigências

A seguir você encontrará uma lista de exigências para o desenvolvimento deste trabalho. Caso alguma delas não seja cumprida, a nota do trabalho será ZERO.

O programa DEVE ser feito em C++ utilizando as definições de classes fornecidas pelo professor. NÃO É PERMITIDO utilizar qualquer estrutura de dados já implementada em bibliotecas do C++ ou de terceiros (p. ex. vector, string), portanto você deve implementar todas as estruturas de dados necessárias para seu programa.

Você DEVE passar apenas uma vez pelo texto. À medida que as palavras são lidas do arquivo, o seu algoritmo deve pesquisar a tabela de dispersão para ver se a palavra já está presente. Se estiver, adicione o novo número de linha à lista dessa palavra. Se não estiver presente, crie um novo elemento na tabela e inicialize a lista de inteiros correspondente com a linha atual.

Você DEVE utilizar uma tabela com **47 posições** e, dada uma palavra p, usar a seguinte **função de dispersão**: $h(p) = valor(p)\ \%\ 47$. O *valor* de uma palavra é calculado de uma forma simples: some os valores ASCII de cada letra da palavra. Por exemplo, $valor(\text{teste}) = 549$, pois os valores de "t", "e" e "s" na tabela ASCII são 116, 101 e 115, respectivamente.

Você DEVE, ao final do seu programa, liberar toda memória alocada dinamicamente (vetores, listas, e outras) e também garantir que seu programa não realiza acessos inválidos à memória. Para verificar essa questão, será utilizado o utilitário *Valgrind*, com o comando `valgrind --leak-check=full --show-reachable=yes --track-fds=yes ./programa texto.txt`. Embora execute mais lentamente, você pode **compilar** seu programa com a opção de depuração `-g`, permitindo que o valgrind detalhe mais a saída, incluindo os números das linhas de seu programa com eventuais problemas.

O uso das estruturas de dados conforme as especificações propostas é OBRIGATÓRIO. Para listas encadeadas, fica a seu critério implementá-las com ou sem cabeça. Observe que em nossas listas encadeadas um novo elemento será inserido sempre no final. Portanto, por questões de eficiência, você DEVE sempre ter armazenados os ponteiros para o inicio e para o fim de uma lista.

6 Entrega

Instruções para entrega do seu trabalho:

1. **Cabeçalho**

   Seu trabalho deve ter um cabeçalho com o seguinte formato:

```
/*************************************************
 *
 * Nome do(a) estudante
 * Trabalho 1
 * Professor(a): Nome do(a) professor(a)
 *
 */
```

2. **Compilador**

   Para a correção do trabalho, será utilizado o compilador da linguagem C++ da coleção de compiladores GNU `g++`, com as opções de compilação `-Wall -pedantic -std=c++11` para C++ ao corrigir os programas. Opcionalmente, você pode utilizar a flag `-g` para compilar seu programa e testá-lo pelo `valgrind`. Antes de entregar seu programa, verifique se ele tem extensão `.cpp`, compila sem mensagens de alerta e executa corretamente e não possui problemas de memória acusados pelo `valgrind`.

3. **Forma de entrega**

   A entrega será realizada diretamente na página da disciplina no AVA/UFMS. Um fórum de discussão deste trabalho já se encontra aberto. Após abrir uma sessão digitando seu *login* e sua senha, vá até o tópico "Trabalhos", e escolha "T1 - Entrega". Você pode entregar o trabalho quantas vezes quiser até às **06 horas e 00 minutos** do dia **13 de abril de 2020**. A última versão entregue é aquela que será corrigida. Encerrado o prazo, não serão mais aceitos trabalhos.

4. **Atrasos**

   Trabalhos atrasados não serão aceitos. Não deixe para entregar seu trabalho na última hora. Para prevenir imprevistos como queda de energia, problemas com o sistema, falha de conexão com a internet, sugerimos que a entrega do trabalho seja feita pelo menos um dia antes do prazo determinado.

5. **Erros**

   Trabalhos com erros de compilação receberão nota ZERO. Faça todos os testes necessários para garantir que seu programa está livre de erros de compilação.

6. **O que entregar?**

   Você deve entregar um único arquivo contendo APENAS o seu programa fonte com o mesmo nome de seu login no moodle, como por exemplo, `fulano_silva.cpp`. NÃO entregue qualquer outro arquivo, tal como o programa executável, já compilado.

7. **Verificação dos dados de entrada**

   Não se preocupe com a verificação dos dados de entrada do seu programa. Seu programa não precisa fazer consistência dos dados de entrada. Isto significa que se, por exemplo, o seu programa pede um número entre 1 e 10 e o usuário digita um número negativo, uma letra, um cifrão, etc, o seu programa pode fazer qualquer coisa, como travar o computador ou encerrar a sua execução abruptamente com respostas erradas.

8. **Arquivo com o programa fonte**

   Seu arquivo contendo o programa fonte na linguagem C++ deve estar bem organizado. Um programa na linguagem C++ tem de ser muito bem compreendido por uma pessoa. Verifique se seu programa tem a indentação adequada, se não tem linhas muito longas, se tem variáveis com nomes significativos, entre outros. Não esqueça que um programa bem descrito e bem organizado é a chave de seu sucesso. Não esqueça da documentação de seu programa e de suas funções.

   Dê o nome do seu usuário do moodle para seu programa e adicione a extensão `.cpp` a este arquivo. Por exemplo, `fulano_silva.cpp` é um nome válido.

9. **Conduta Ética**

   O trabalho deve ser feito INDIVIDUALMENTE. Cada estudante tem responsabilidade sobre cópias de seu trabalho, mesmo que parciais. Não faça o trabalho em grupo e não compartilhe seu programa ou trechos de seu programa. Você pode consultar seus colegas para esclarecer dúvidas e discutir idéias sobre o trabalho, ao vivo ou no fórum de discussão da disciplina, mas NÃO copie o programa!

   Trabalhos considerados plagiados terão nota ZERO.