MC 202EF - Estruturas de Dados
Lab 02

(O conteúdo necessário para realizar este laboratório vai até a unidade:
Operações em listas e variações)

PED: Márcio de Carvalho Saraiva
PAD: Anderson, Mateus, Victor
Professor: Lehilton Lelis Chaves Pedrosa

29 de agosto de 2016

1. Problema

Imagine que temos um fichário contendo n pastas rotuladas mas sem nenhuma ordem especial (uma lista). Cada rótulo é uma chave (inteiro) que serve para identificar a pasta de modo único. O objetivo é atender uma sequência de requisições onde uma requisição é uma das seguintes operações: acesse(x), insere(x) e remove(x), em que x é um rótulo de um item (pasta). O custo de uma requisição para x é 1 mais o número de itens que precedem x ou 1 mais o tamanho da lista se o item não pertence à lista. Mais precisamente, acessar ou remover o i-ésimo item da lista custa i; inserir um novo item custa n+1, onde n é o tamanho da lista (o item é inserido no final da lista).

Você, como o responsável pelo fichário, gostaria de usar uma estratégia que permitisse minimizar o tempo total que você gasta para atender as requisições. Isso talvez fosse mais fácil se você soubesse de antemão quais são as requisições futuras, mas o problema acima é online, o que significa que as requisições vão chegando e devem ser atendidas imediatamente sem conhecimento de requisições futuras.

É fácil perceber que uma lista ligada é uma estrutura de dados adequada para implementar o modelo acima. O custo das buscas é proporcional ao número de itens percorridos.

Dois algoritmos para acesso à lista

Nos algoritmos de acesso à lista, costuma-se adotar a seguinte convenção: imediatamente após acessar ou inserir um item, podemos movê-lo (sem nenhum custo) para qualquer posição próxima do início da lista. As transposições (trocas) feitas para realizar este movimento são chamadas livres enquanto as demais transposições são chamadas pagas e tem custo 1 cada.

Muitos algoritmos foram propostos para gerenciar uma lista. Neste projeto consideramos dois algoritmos bem conhecidos que só usam transposições livres.

- MTF (Move-To-Front): após acessar um item, este é movido para o início da lista;
- TR (Transpose): após acessar ou inserir um item, faça uma transposição com o item anterior (sem custo). Se o item acessado for o primeiro da lista, mantenha-o na mesma posição;

Vejamos um exemplo concreto. Denote as operações de acesso, inserção e remoção por a(x), i(x) e r(x), respectivamente. Suponha que a lista inicial seja L=(1,2,3,4,5) e a sequência de requisições seja:

a(4),a(2),a(2),i(6),a(3),r(2),a(3),a(6),r(1).

Veja abaixo quais são os custos para cada algoritmo.

MTF

- Lista inicial L=(1,2,3,4,5)
- a(4). Custo = 4. Lista L=(4,1,2,3,5)
- a(2). Custo = 3. Lista L=(2,4,1,3,5)
- a(2). Custo = 1. Lista L=(2,4,1,3,5)
- i(6). Custo = 6. Lista L=(2,4,1,3,5,6)
- a(3). Custo = 4. Lista L=(3,2,4,1,5,6)
- r(2). Custo = 2. Lista L=(3,4,1,5,6)
- a(3). Custo = 1. Lista L=(3,4,1,5,6)
- a(6). Custo = 5. Lista L=(6,3,4,1,5)
- r(1). Custo = 4. Lista L=(6,3,4,5)

Custo total = 4+3+1+6+4+2+1+5+4 = 30.

TR

- Lista inicial L=(1,2,3,4,5)
- a(4). Custo = 4. Lista L=(1,2,4,3,5)
- a(2). Custo = 2. Lista L=(2,1,4,3,5)
- a(2). Custo = 1. Lista L=(2,1,4,3,5)
- i(6). Custo = 6. Lista L=(2,1,4,3,6,5)
- a(3). Custo = 4. Lista L=(2,1,3,4,6,5)
- r(2). Custo = 1. Lista L=(1,3,4,6,5)
- a(3). Custo = 2. Lista L=(3,1,4,6,5)
- a(6). Custo = 4. Lista L=(3,1,6,4,5)
- r(1). Custo = 2. Lista L=(3,6,4,5)

Custo total = 4+2+1+6+4+1+2+4+2 = 26.

Resultados conhecidos

Embora no exemplo acima o TR custe menos, ele é o pior dos dois algoritmos, no sentido de que, para a maioria das instâncias, o MFT comporta-se melhor do que o TR. Pode-se provar isso formalmente.

2. Entrada

Você pode supor que as chaves são inteiros entre 1 e 1000.

A primeira linha da entrada contém dois inteiros N e R, representando o tamanho inicial da lista e o número de requisições, com $1 \leq N \leq 100$ e $1 \leq R \leq 200$. A segunda linha contém N inteiros distintos representando os itens da lista. Seguem-se então R linhas, cada uma contendo uma das seguintes instruções:

- a <x> (acesse item x)
- i <x> (insira item x)
- r <x> (remova item x)

onde <x> é um inteiro.

Entrada:

```
5 9
1 2 3 4 5
a 4
a 2
a 2
i 6
a 3
r 2
a 3
a 6
r 1
```

3. Saída

A saída do programa consiste de 4 linhas contendo o custo de MTF, a lista final quando executado o MTF, o custo de TR e a lista final quando executado o TR.

Saída:

```
30
6 3 4 5
26
3 6 4 5
```

4. Dica

a. Como não sabemos o tamanho da lista de entrada a priori, use a função malloc para alocar o espaço necessário.

b. Há várias maneiras de fazer o projeto e algumas nem precisam usar listas ligadas. Entretanto, como o objetivo do projeto é se familiarizar com listas ligadas, você deve implementar os algoritmos MTF e TR sobre uma lista ligada. Você pode usar listas ligadas com nó cabeça, se quiser. Se quiser, pode usar listas duplamente ligadas.

c. Lembrem-se de liberar o espaço após o uso.

5. Avaliação

5.1 As notas dos laboratórios serão calculadas da seguinte maneira:

- 7 pontos proporcionais ao número de casos de teste (apenas os casos fechados presentes no sistema) que o código passe;
- 3 pontos referentes à qualidade de código (legibilidade, algoritmo, memory leak, etc...)

Atenção:

Na página deste exercício no run.codes, o aluno pode baixar um arquivo .zip com casos de testes adicionais abertos para testes locais dos alunos. Mas a nota do aluno na atividade depende dos casos de teste fechados do sistema.

5.2 Critérios avaliados neste laboratório:

Além de passar nos casos de teste o aluno deve buscar:

- Apresentar boas práticas de programação, como comentários no código, escolha do nomes para variáveis, reutilização de funções, etc; que possam melhorar a apresentação do código;

- Criar funções para contar operações, para inserir acessar e remover na lista;

- Criar listas ligadas simples ou duplas.

Atenção: Não serão aceitas cópias de código entre os alunos.

6. Entrega

A submissão de código deve ser feita no Run.codes em no **máximo 10 tentativas** até o dia 16/09/16 às 23:59:59.