sqtpm
[ggm]

ajuda
voltar

Trabalho: 23-auto

Data de abertura: 19/05/2025 12:00:00
Data limite para envio: 26/05/2025 23:59:00 (aberto)

Número de arquivos a enviar: entre 1 e 3

Número máximo de envios: 30
Número de envios feitos por ggm: 0

Casos-de-teste abertos: casos-de-teste.tgz

Recursos: cpu 1 s, memória 500000 kb, stack 8192 kb

Enviar:

Linguagem: C
Arquivos: Escolher arquivos Nenhum arquivo escolhido
Enviar

Lista auto-organizável

O problema de encontrar o registro que tem chave k em um conjunto de registros, conhecido como o problema da busca, precisa ser resolvido freqüentemente em sistemas de processamento de dados.

Quando há poucos registros ou quando a busca é feita com freqüência muito baixa, a busca seqüencial pode ser suficientemente eficiente. A busca seqüencial pode ser usada quando os registros estão em um vetor ou quando estão em uma lista encadeada.

Em muitas aplicações, as buscas apresentam localidade temporal (quando um registro é buscado a probabilidade de que ele seja buscado de novo em um espaço de tempo curto é alta) e espacial (quando um registro é buscado a probabilidade de que "vizinhos" sejam buscados é alta. Por exemplo, colegas de turma de um aluno, produtos do mesmo tipo mas de outro modelo e marca etc.)

Algumas estratégias de permutação foram propostas na literatura para melhorar o desempenho da busca seqüencial supondo localidade. Estruturas-de-dados que mudam sua organização a cada operação (inclusive operações de consulta) são chamadas de auto-organizáveis.

Este trabalho consiste em avaliar o desempenho da busca seqüencial em uma lista encadeada contra o desempenho da busca seqüencial com as seguintes estratégias de permutação:

- Move-to-front (MTF): quando um registro é recuperado ele é movido para o início da lista encadeada.
- Transpose: quando um registro é recuperado ele é trocado de posição com o registro que o precede.
- Count: cada registro tem um contador do número de vezes que ele foi buscado. Quando um registro é buscado, o contador é incrementado e ele é movido para uma posição anterior a todos os registros com contador menor ou igual ao dele.

Por exemplo, suponha que a lista inicial seja `L=(1,2,3,4,5,6,7,8,9)` e que a sequência de buscas seja `R=(7,8,7,8,8,7,3,5)`.

A busca seqüencial faz 7+8+7+8+8+7+3+5 = 53 comparações de chaves para realizar todas as buscas.

- **MTF**

sqtpm
[ggm]

ajuda
voltar

Chave buscada	Comparações de chaves	Lista
		L= (1,2,3,4,5,6,7,8,9)
7	7	L= (7,1,2,3,4,5,6,8,9)
8	8	L= (8,7,1,2,3,4,5,6,9)
7	2	L= (7,8,1,2,3,4,5,6,9)
8	2	L= (8,7,1,2,3,4,5,6,9)
8	1	L= (8,7,1,2,3,4,5,6,9)
7	2	L= (7,8,1,2,3,4,5,6,9)
3	5	L= (3,7,8,1,2,4,5,6,9)
5	7	L= (5,3,7,8,1,2,4,6,9)

Número de comparações total = 7+8+2+2+1+2+5+7 = 34.

- **Transpose**

Chave buscada	Comparações de chaves	Lista
		L= (1,2,3,4,5,6,7,8,9)
7	7	L= (1,2,3,4,5,7,6,8,9)
8	8	L= (1,2,3,4,5,7,8,6,9)
7	6	L= (1,2,3,4,7,5,8,6,9)
8	7	L= (1,2,3,4,7,8,5,6,9)
8	6	L= (1,2,3,4,8,7,5,6,9)
7	6	L= (1,2,3,4,7,8,5,6,9)
3	8	L= (1,3,2,4,7,8,5,6,9)
5	7	L= (1,3,2,4,7,5,8,6,9)

Número de comparações total = 7+8+6+7+6+6+8+7 = 50.

- **Count**

sqtpm
[ggm]

ajuda
voltar

Chave buscada	Comparações de chaves	Lista	Contadores
		L= (1,2,3,4,5,6,7,8,9)	C= (0,0,0,0,0,0,0,0,0)
7	7	L= (7,1,2,3,4,5,6,8,9)	C= (1,0,0,0,0,0,0,0,0)
8	8	L= (8,7,1,2,3,4,5,6,9)	C= (1,1,0,0,0,0,0,0,0)
7	2	L= (7,8,1,2,3,4,5,6,9)	C= (2,1,0,0,0,0,0,0,0)
8	2	L= (8,7,1,2,3,4,5,6,9)	C= (2,2,0,0,0,0,0,0,0)
8	1	L= (8,7,1,2,3,4,5,6,9)	C= (3,2,0,0,0,0,0,0,0)
7	2	L= (7,8,1,2,3,4,5,6,9)	C= (3,3,0,0,0,0,0,0,0)
3	5	L= (7,8,3,1,2,4,5,6,9)	C= (3,3,1,0,0,0,0,0,0)
5	7	L= (7,8,5,3,1,2,4,6,9)	C= (3,3,1,1,0,0,0,0,0)

Número de comparações total = 7+8+2+2+1+2+5+7 = 34.

Neste trabalho as busca seqüencial simples e com as três estratégias MTF, transpose e count devem ser comparadas.

Para cada uma das 4 formas de busca, a lista começa com as chaves inteiras `1,2,3,...,N`.

Depois, r buscas são feitas na lista. Para cada busca, registra-se o número de comparações de chaves.

Se uma chave buscada k não existir na lista então ela é adicionada, sendo que para busca seqüencial simples a inserção é feita no fim da lista, para MTF e transpose a inserção é feita no início da lista e para count a inserção é feita antes de todos os registros com contador igual a 1.

Entrada

A entrada para o programa começa com uma linha contendo o inteiro positivo n. Depois vem uma linha contendo o inteiro positivo r. Depois vêm r inteiros positivos, que são as chaves que devem ser buscadas na lista, na ordem dada.

Todos os inteiros são positivos com magnitude máxima igual a 2.147.483.647.

Saída

A saída são quatro linhas indicando o número de comparações de chaves das buscas seqüencial simples, seqüencial com MTF, seqüencial com transpose e seqüencial com Count, respectivamente. As buscas devem ser realizados na ordem dada na entrada, a partir da lista inicial contendo as chaves na ordem.

sqtpm
[ggm]

ajuda
voltar

Exemplo

Entrada
9 8 7 8 7 8 8 7 3 5

Saída
Sequencial: 53 MTF: 34 Transpose: 50 Count: 34

Entrada
5 12 1 2 3 4 5 10 1 12 2 3 4 5

Saída
Sequencial: 41 MTF: 60 Transpose: 51 Count: 60

Requisitos

- O programa deve usar uma lista encadeada para avaliar as estratégias.

Observações

- Essas estratégias também podem ser usadas em vetores, quando estamos fazendo buscas em dados não ordenados.
- A estratégia count vai fazer com que os registros fiquem em ordem não-crescente de contadores. Isso permite que a movimentação seja implementada fazendo apenas uma passada pela lista, ao invés de duas como pode parecer necessário à primeira vista. Depois de fazer seu programa funcionar com duas passadas, um exercício interessante é implementar com apenas uma.

Sobre organização do código e comentários

- Faça um programa organizado, bem indentado e que seja fácil de ler.
- Adicione comentários que vão ser úteis para entender o programa se você for relê-lo daqui a alguns anos: comentar cada linha vai ser redundante; documentar blocos de código e a estratégia usada na solução vai ser muito útil.
