

**sqtpm**

[202619]

[voltar](#)**Trabalho:** 02-auto-organizavel

Linguagens: C

Data de abertura: 2017/08/14 14:00:00

Data limite para envio: 2017/08/20 18:00:00 (encerrado)

Número máximo de envios: 25

Casos-de-teste abertos: [casos-de-teste.tgz](#)

---

## Lista auto-organizável

Em uma lista não há uma forma simples que facilite a recuperação dos registros armazenados nos nós. Para recuperar o nó na posição  $i$  de uma lista é preciso percorrê-la a partir da cabeça, fazendo  $i$  acessos a nós.

Em muitas aplicações, as frequências com que os registros são acessados não são uniformes e então faz sentido que os registros que são recuperados com maior frequência sejam colocados mais próximos da cabeça, mas tipicamente tais frequências não são conhecidas antecipadamente e mudam ao longo do tempo.

Estratégias de permutação têm sido aplicadas para reduzir o número de acessos à memória ao recuperar registros em uma lista. Tais estratégias movem o registro que acabou de ser buscado um certo número de posições em direção ao início da lista, sem modificar a ordem relativa dos demais registros. Listas acompanhadas de alguma estratégia desse tipo foram chamadas de *listas auto-organizáveis*.

Algumas estratégias de permutação foram propostas na literatura. As mais usadas incluem:

- Move-to-front (MTF): quando um registro é recuperado ele é movido para o início da lista, se ele ainda não estiver no início da lista.
- Transpose (TR): quando um registro é recuperado ele é trocado de posição com o registro que o precede, se ele ainda não estiver no início da lista.
- Count (C): cada registro tem um contador do número de acessos. Quando um registro é recuperado 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 seja  $L = (1, 2, 3, 4, 5)$  e a sequência de requisições seja  $R = (4, 2, 2, 4, 3, 1, 3)$ . Abaixo aparecem as modificações na lista e os custos para cada estratégia. O custo é calculado como o número de nós acessados para recuperar o nó requisitado.

### Move-to-front

- Lista inicial  $L = (1, 2, 3, 4, 5)$
- Requisição = 4. Custo = 4. Lista  $L = (4, 1, 2, 3, 5)$
- Requisição = 2. Custo = 3. Lista  $L = (2, 4, 1, 3, 5)$
- Requisição = 2. Custo = 1. Lista  $L = (2, 4, 1, 3, 5)$
- Requisição = 4. Custo = 2. Lista  $L = (4, 2, 1, 3, 5)$
- Requisição = 3. Custo = 4. Lista  $L = (3, 4, 2, 1, 5)$
- Requisição = 1. Custo = 4. Lista  $L = (1, 3, 4, 2, 5)$
- Requisição = 3. Custo = 2. Lista  $L = (3, 1, 4, 2, 5)$

Custo total =  $4+3+1+2+4+4+2 = 20$ .

**sqtpm**

[202619]

voltar**Transpose**

- Lista inicial  $L = (1, 2, 3, 4, 5)$
- Requisição = 4. Custo = 4. Lista  $L = (1, 2, 4, 3, 5)$
- Requisição = 2. Custo = 2. Lista  $L = (2, 1, 4, 3, 5)$
- Requisição = 2. Custo = 1. Lista  $L = (2, 1, 4, 3, 5)$
- Requisição = 4. Custo = 3. Lista  $L = (2, 4, 1, 3, 5)$
- Requisição = 3. Custo = 4. Lista  $L = (2, 4, 3, 1, 5)$
- Requisição = 1. Custo = 4. Lista  $L = (2, 4, 1, 3, 5)$
- Requisição = 3. Custo = 4. Lista  $L = (2, 4, 3, 1, 5)$

Custo total =  $4+2+1+3+4+4+4 = 22$ .

**Count**

- Lista inicial  $L = (1, 2, 3, 4, 5)$ . Contador  $C = (0, 0, 0, 0, 0)$
- Requisição = 4. Custo = 4. Lista  $L = (4, 1, 2, 3, 5)$ . Contador  $C = (1, 0, 0, 0, 0)$
- Requisição = 2. Custo = 3. Lista  $L = (2, 4, 1, 3, 5)$ . Contador  $C = (1, 1, 0, 0, 0)$
- Requisição = 2. Custo = 1. Lista  $L = (2, 4, 1, 3, 5)$ . Contador  $C = (2, 1, 0, 0, 0)$
- Requisição = 4. Custo = 2. Lista  $L = (4, 2, 1, 3, 5)$ . Contador  $C = (2, 2, 0, 0, 0)$
- Requisição = 3. Custo = 4. Lista  $L = (4, 2, 3, 1, 5)$ . Contador  $C = (2, 2, 1, 0, 0)$
- Requisição = 1. Custo = 4. Lista  $L = (4, 2, 1, 3, 5)$ . Contador  $C = (2, 2, 1, 1, 0)$
- Requisição = 3. Custo = 4. Lista  $L = (3, 4, 2, 1, 5)$ . Contador  $C = (2, 2, 2, 1, 0)$

Custo total =  $4+3+1+2+4+4+4 = 22$ .

Outra estratégia é a move-ahead-k, que move um registro k posições em direção à cabeça depois que ele é acessado. k pode ser definido como um número fixo, como um percentual da distância até a cabeça ou como outra função de distância. Há ainda outras estratégias que fazem combinações das que foram citadas neste texto.

Neste trabalho as três estratégias MTF, TR e C devem ser comparadas. Seu programa deve usar uma lista encadeada.

**Entrada**

A entrada para o programa são um inteiro N entre 1 e 1000, um inteiro R e uma seqüência de R inteiros no intervalo [1,10000]. Inicialmente a lista deve conter as chaves 1,2,...,N em ordem, como nos exemplos acima. Cada uma das R recuperações deve ser realizada na ordem em que são dadas, para MTF, TF e C, sempre a partir da lista inicial.

**Saída**

A saída são três inteiros indicando os custos das estratégias MTF, TF e C, respectivamente. O custo é medido como a soma do número de nós visitados durante o acesso, sem contar as operações realizadas na

reorganização da lista.

**sqtpm**

[202619]

[voltar](#)

## Exemplo

### Entrada:

5  
7  
4 2 2 4 3 1 3

### Saída:

20 22 22

## Observações

- A estratégia count vai fazer com que os registros fiquem em ordem não-crescente de contadores. Essa invariante permite que a movimentação seja implementada percorrendo a lista apenas uma vez, ao invés de duas como pode parecer necessário à primeira vista. Depois de fazer seu programa funcionar com duas passadas, tente implementar com apenas uma.
-