Comparativo entre estruturas de dados lineares estáticas, encadeadas e duplamente encadeadas

Fernando Concatto

Centro de Ciências Tecnológicas da Terra e do Mar
Universidade do Vale do Itajaí (UNIVALI)
Itajaí – SC – Brasil

fernandoconcatto@edu.univali.br

Resumo. *Estruturas de dados permitem ao programador organizar dados para facilitar sua manipulação, cada uma possuindo prós e contras. Este trabalho investigou as estruturas de dados lineares, chamadas de listas, identificando qual delas é mais efetiva nas operações de inserção, remoção e ordenação. Foi possível identificar a lista duplamente encadeada como superior às outras listas, obtendo melhores resultados nas operações de inserção e remoção e desempenho equivalente na operação de ordenação.*

1. Introdução

Ao desenvolver programas, é frequente a necessidade de manipular dados na memória do computador para transformá-los e refiná-los em saídas úteis. Estruturas de dados, no contexto da Ciência da Computação, são formas de organizar estes dados de forma a manipulá-los de maneira eficiente e conveniente. Diversas estruturas de dados bem definidas estão disponíveis aos programadores, e escolher corretamente a estrutura mais apropriada para o problema em questão pode ser um desafio considerável. Este trabalho busca realizar uma análise de uma categoria específica de estruturas de dados conhecida como lista, cuja principal característica é a organização linear dos dados, com o objetivo de identificar qual delas é a mais eficiente nas operações de inserção, remoção e ordenação de dados.

O trabalho foi dividido em mais três seções: a segunda define as operações para cada tipo de lista, a terceira expõe a análise comparativa realizada e a quarta seção apresenta as conclusões obtidas quanto ao trabalho.

2. Operações em Listas

Entre as operações mais comuns realizadas sobre estruturas de dados estão a inserção, remoção e ordenação. Esta seção busca apresentar as particularidades de cada operação nas listas investigadas neste trabalho: lista estática, lista simplesmente encadeada e lista duplamente encadeada.

2.1. Operações de Inserção

A operação de inserção é caracterizada pela adição de um novo elemento em uma estrutura vazia ou populada com elementos. Em listas estáticas, é necessário verificar se

há espaço suficiente para o novo elemento, pois é utilizado um vetor de tamanho limitado como estrutura auxiliar; esta limitação não está presente nas listas encadeadas.

A versão da operação de inserção implementada na lista estática envolve a necessidade de deslocar em um índice todos os elementos à frente da posição desejada, para que o dado antigo naquela posição não seja sobrescrito. Caso a inserção seja realizada no início, todos os elementos deverão ser deslocados; se for realizada na posição k, os elementos de k até o último elemento serão deslocados; por fim, se a operação for realizada no final da lista, nenhum elemento necessita ser deslocado, pois aquela posição já está vazia.

Já nas listas encadeadas, não é necessário realizar deslocamento de elementos, pois não são utilizados vetores. Para executar a inserção, é necessário percorrer a lista até encontrar o elemento antecessor à posição onde se deseja inserir o novo dado (na lista duplamente encadeada, é possível buscar o elemento na própria posição, pois há um ponteiro para o elemento anterior); então, os ponteiros nos elementos em torno daquela posição devem ser manipulados para manter o encadeamento da lista.

No caso da lista simplesmente encadeada, é necessário uma condição especial para inserção no início, pois o ponteiro para o primeiro elemento deve ser alterado. Para a lista duplamente encadeada, além do caso da inserção no início, é necessário tratar a inserção no final, pois este tipo de lista também possui um ponteiro para o último elemento.

2.2. Operações de Remoção

As operações de remoção são notavelmente similares às operações de inserção em todas as três listas, sendo necessárias apenas algumas adaptações.

Na lista estática, a direção do deslocamento dos elementos será invertida: ao invés de reservar espaço para o novo elemento, o elemento a ser removido será sobrescrito pelo elemento diretamente à sua frente, e assim sucessivamente até o penúltimo elemento da lista. Assim como na inserção, se o elemento a ser removido for o último, não é necessário realizar nenhum deslocamento, apenas o decremento da variável de quantidade.

Nas listas encadeadas, a necessidade de encontrar o elemento antecessor àquele que será removido ainda existe, e ainda requer tempo proporcional às posições percorridas. Então, os ponteiros dos elementos serão manipulados de forma a fazer com que nenhum deles aponte para o elemento a ser removido, e em seguida, a memória deverá ser liberada. Os casos especiais de início de final da operação de inserção também se aplicam nesta operação.

2.3. Operação de Ordenação

Para a implementação de ordenação nas listas, foram utilizados dois algoritmos bastante difundidos: bubble sort (ordenação por bolha) e merge sort (ordenação por união). Apesar das particularidades de cada lista, a implementação foi bastante similar em todas as três.

Na execução do algoritmo de bubble sort, a lista é percorrida do primeiro elemento até o penúltimo, e para cada índice é verificado se o elemento atual é maior que o elemento à sua frente: se verdadeiro, os dois elementos trocam de posição; se falso, nenhuma mudança é feita. O processo se repete até que nenhuma troca seja feita durante uma iteração.

Já o algoritmo de merge sort é consideravelmente mais intricado. Envolve duas operações principais: a separação da lista em duas partes, executada recursivamente, e a união das partes previamente separadas. O processo de separação divide a lista pela metade até que seja composta por apenas um elemento. Então, a lista é construída novamente a partir das listas divididas, combinando-as de forma com que a nova lista esteja ordenada. Juntamente com a divisão recursiva, esta propriedade faz com que a combinação possa ser realizada em tempo linear, pois as listas a serem combinadas estarão previamente ordenadas, fazendo com que apenas uma comparação seja necessária para cada elemento a ser inserido na lista combinada.

3. Comparativo

Para as operações de inserção, as três listas apresentam comportamentos distintos. A lista estática requer n instruções para inserções realizadas no início, devido à necessidade de deslocamento dos elementos, enquanto a inserção no final é constante, pois não requer deslocamentos. A inserção na posição requer $n - k$ instruções, onde k é a posição onde o elemento será inserido.

Para ambas as listas encadeadas, a operação de inserção propriamente dita é executada em tempo constante, pois requer simplesmente a manipulação dos ponteiros dos elementos em torno da posição escolhida. Entretanto, para realizar tal manipulação, é necessário *encontrar* os elementos que terão seus ponteiros alterados; esta operação requer tempo linear proporcional à distância do início da lista. Assim, a inserção no início é executada em tempo constante, mas a inserção na posição requer tempo linear. Na lista simplesmente encadeada, a inserção no final também requer tempo linear, pois todos os elementos serão percorridos; entretanto, a lista duplamente encadeada permite acessar o último elemento imediatamente, fazendo com que a inserção no final seja executada em tempo constante. Outra importante vantagem da lista duplamente encadeada é a possibilidade de iniciar a busca pelo elemento partindo tanto do início quanto do final, fazendo com que no pior caso sejam necessárias $n \div 2$ operações.

Na tabela a seguir, estão relatadas as complexidades pessimistas das funções de inserção no início, final e na posição. Para as listas encadeadas, foi considerada a complexidade de busca pelo elemento além da inserção propriamente dita. A lista duplamente encadeada se mostra superior às outras estruturas, com tempo constante na inserção no início e final, e maior eficiência na inserção na posição devido à capacidade de iniciar a busca a partir de ambas as extremidades. Na operação de inserção na posição, o símbolo k representa a posição onde o elemento será inserido.

Tabela 1. Comparativo entre operações de inserção

	Lista Estática	Lista Encadeada	Lista Duplamente Endadeada
InsereInício	f(n) = n	f(n) = 1	f(n) = 1
InsereFinal	f(n) = 1	f(n) = n	f(n) = 1
InserePosição	f(n) = n - k	f(n) = k	f(n) = k, com k ≤ n ÷ 2

As operações de remoção são consideravelmente similares às operações de inserção quanto aos passos necessários para executar a ação e, consequentemente, quanto à complexidade computacional. Na implementação na lista estática, a mesma quantidade de elementos será deslocada; nas listas encadeadas, a mesma quantidade de elementos deverá ser percorrida. Isso significa que a lista duplamente encadeada continua sendo superior às outras, pois permite que menos elementos sejam percorridos para completar a remoção. Para a remoção na posição, *k* indica a posição do elemento a ser removido.

Tabela 2. Comparativo entre as operações de remoção

	Lista Estática	Lista Encadeada	Lista Duplamente Endadeada
RemoveInício	f(n) = n	f(n) = 1	f(n) = 1
RemoveFinal	f(n) = 1	f(n) = n	f(n) = 1
RemovePosição	f(n) = n - k	f(n) = k	f(n) = k, com k ≤ n ÷ 2

As operações de ordenação apresentam a mesma complexidade para os três tipos de lista, pois não são realizadas operações de inserção ou remoção, apenas comparações, trocas e navegação sequencial.

Para o algoritmo de bubble sort, é necessário percorrer completamente a lista, realizando comparações e trocas até que os elementos estejam em ordem. No pior caso, onde todos os elementos estão fora de ordem, será necessário repetir o processo n vezes, onde n é a quantidade de elementos presentes na lista. Desta forma, no pior caso, o algoritmo demanda n^2 instruções.

A análise do algoritmo de merge sort se mostrou consideravelmente mais complicada. A primeira porção do algoritmo divide a lista pela metade até que a lista contenha apenas um elemento; este processo apresenta complexidade logarítmica de base 2, pois será necessária uma divisão adicional para cada vez que a quantidade de

elementos for dobrada. A segunda parte do algoritmo envolve a combinação de duas listas previamente ordenadas em uma nova lista, também ordenada. Devido à propriedade das listas estarem ordenadas, é possível realizar a combinação em tempo linear: mantendo referências inicializadas para o início de cada lista, é feita a comparação entre os dois elementos referenciados; o menor (ou maior, dependendo da ordem) é adicionado na nova lista, e a referência da lista cujo elemento foi adicionado é movida um índice à frente. O processo é repetido até que todos os elementos sejam adicionados na nova lista.

Sendo n a quantidade de elementos de uma lista, serão necessárias n operações para construir uma lista de n elementos parcialmente ordenada. Devido à característica logarítmica da divisão, essa construção será realizada $log_2\ n$ vezes para completar a ordenação da lista; assim, o algoritmo de mergesort requer $n\ log_2\ n$ operações para ser executado.

Figura 1. Representação gráfica do algoritmo merge sort

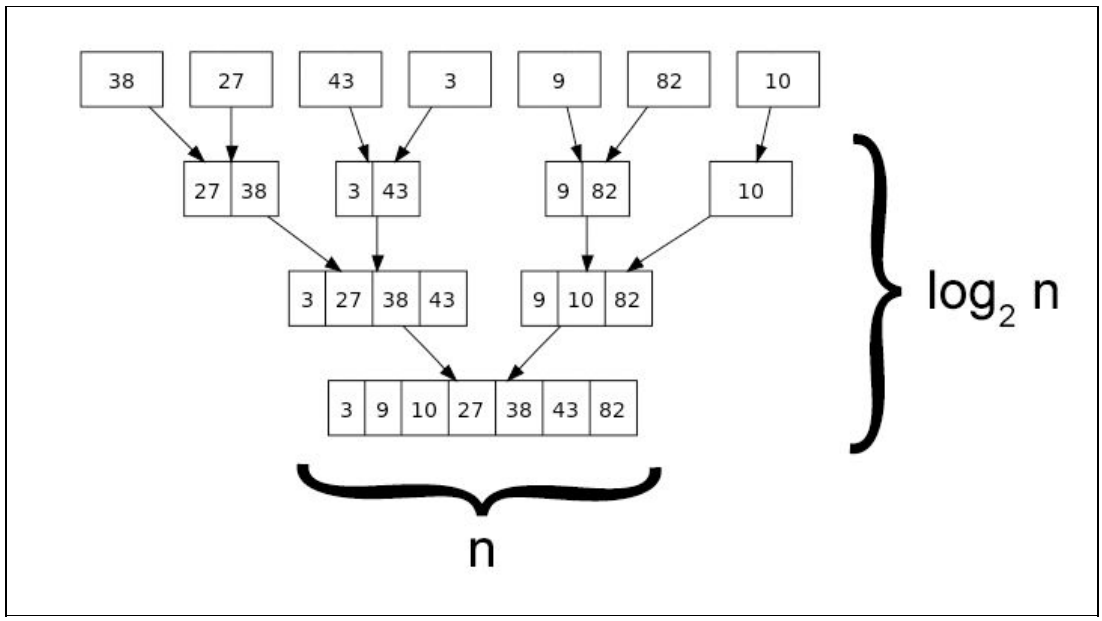

Tabela 3. Comparativo entre bubble sort e merge sort

	Lista Estática	Lista Encadeada	Lista Duplamente Encadeada
Bubble sort	$f(n) = n^2$	$f(n) = n^2$	$f(n) = n^2$
Merge sort	$f(n) = n \log_2 n$	$f(n) = n \log_2 n$	$f(n) = n \log_2 n$

4. Conclusões

Através da análise comparativa, é possível perceber que a lista duplamente encadeada se mostra superior à lista estática e à lista simplesmente encadeada tanto nas operações de inserção quanto nas operações de remoção. Em ambos os casos, as operações são executadas em tempo constante quando realizadas no início e no final, uma característica exclusiva à esta lista. Quando realizadas em uma posição específica, o desempenho da lista duplamente encadeada é ligeiramente superior às outras duas listas, devido à possibilidade de iniciar a busca a partir do início ou a partir do final. Quanto às operações de ordenação, não foi observado comportamento superior em nenhuma das listas; todas as três apresentaram a mesma complexidade computacional em ambos os algoritmos de ordenação.