Estrutura de Dados

Introdução

Extraído de : Estruturas de Dados – Homero L. Pícollo

Na resolução de um problema por meio de um programa, a primeira providência é conceber um algoritmo adequado.

A eficiência de um algoritmo qualquer está intimamente relacionada à disposição, na memória, dos dados que são tratados pelo programa. Por exemplo, se freqüentemente enfrentamos o problema de descobrir os números de telefones de nossos conhecidos, é conveniente dispor de uma relação de números, organizada em uma agenda. Se a organização for feita por ordem alfabética, a agenda de fato ajuda. Se, porém, organizássemos nossa agenda pela ordem de altura das pessoas, com raras exceções, a agenda se tornaria difícil de manusear.

As estruturas de dados são formas de distribuir e relacionar os dados disponíveis, de modo a tornar mais eficientes os algoritmos que manipulam esses dados. Vejamos alguns exemplos:

Problema
Manipular um conjunto de fichas de um fichário.

Solução
Organizar as fichas em ordem alfabética

Operações possíveis
Inserir ou retirar um ficha, procurar uma ficha, etc.

Estrutura de Dados Correspondente
LISTA – seqüência de elementos dispostos em ordem.

Problema
Organizar as pessoas que querem ser atendidas num guichê.

Solução
Colocar as pessoas em fila.

Operações possíveis
À medida que uma pessoa é atendida no guichê, outra entra no final da fila... Não é permitido “furar” a fila, ou seja, entrar uma pessoa entre outras que já estão presentes.

Estrutura de Dados Correspondente
FILA – seqüência de elementos dispostos em ordem com uma regra para a entrada e saída dos elementos (o primeiro que chega também é o primeiro que sai da estrutura).

Problema
Organizar um conjunto de pratos que estão sendo lavados, um a um, em um restaurante.

Solução
Colocar os pratos empilhados.

Operações possíveis
Colocar um prato limpo no alto da pilha, retirar um prato do alto da pilha, etc...

Estrutura de Dados Correspondente
PILHA – seqüência de elementos dispostos em ordem, mas com uma regra para entrada e saída dos elementos (o último que chega é o primeiro que sai da estrutura).

Problema
Conseguir um modo de visualizar o conjunto de pessoas que trabalham em uma empresa, tendo em conta sua função.

Solução
Construir um organograma da empresa.

Operações possíveis
Inserir ou retirar certas funções, localizar uma pessoa, etc...

Estrutura de Dados Correspondente
ÁRVORE – estrutura de dados que caracteriza uma relação de hierarquia entre os elementos (uma pessoa não pode pertencer a dois departamentos diferentes, cada diretoria tem os seus próprios departamentos, etc.).

Problema
Estabelecer um trajeto para percorrer todas as capitais de um país.

Solução
Utilizar um mapa que indique as rodovias existentes e estabelecer uma ordem possível para percorrer todas as cidades.

Operações possíveis
Encontrar um modo de percorrer todas as cidades, determinar o caminho mais curto para ir de uma cidade para outra, etc.

Estrutura de Dados Correspondente
GRAFO – estrutura bastante genérica que organiza vários elementos, estabelecendo relações entre eles, dois a dois.

Os exemplos vistos correspondem exatamente aos tipos básicos de estruturas de dados utilizadas em computação: **Listas**, **Filas**, **Pilhas**, **Árvores** e **Grafos**.

O estudo de estrutura de dados é parte fundamental para o desenvolvimento de programas e algoritmos. Assim como um número pode ser representado em vários sistemas diferentes, também um conjunto de dados relacionados entre si pode ser descrito através de várias estruturas de dados distintas.

Quando o programador cria um algoritmo para solucionar um problema, ele também cria uma estrutura de dados que é manipulada pelo algoritmo. A escolha de uma determinada estrutura pode afetar substancialmente a quantidade de área de armazenamento requerida para o processamento bem como o tempo deste processamento.

É, portanto, de grande importância o estudo de diferentes estruturas que possam ser utilizadas eventualmente na resolução de um problema, de forma a simplificar a sua implementação prática.

Um programador de pouca experiência utilizará em sua programação formas bastante simplificadas para a representação dos dados envolvidos (como matrizes e vetores), que possivelmente seriam adequadamente representados através de estruturas mais sofisticadas (filas encadeadas, árvores, etc.), tornando o processamento mais eficiente em termos de área de armazenamento e tempo.

Tipos de Dados

Os sistemas de computação têm por finalidade a manipulação de dados. Um algoritmo, basicamente, consiste na entrada de dados, seu processamento e a saída dos dados computados.

Existem dados considerados básicos. São eles:

- **Dados numéricos** : armazenam valores numéricos.
- **Dados alfanuméricos** : armazenam valores alfabéticos, numéricos, sinais, etc...
- **Dados Lógicos** : armazenam valores lógicos

Em muitos casos, a utilização dos dados básicos pode satisfazer o requisito de uma aplicação.

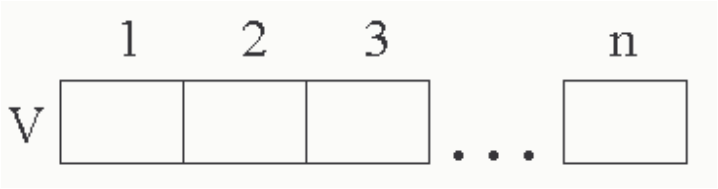
Em outros casos, pode-se utilizar um tipo de dados um pouco mais sofisticado, do tipo matrizes e vetores.

- **Vetores** : um arranjo unidimensional de dados do tipo básico, com um mesmo identificador (nome), mas diferenciado pela sua posição (índice) dentro do vetor.

  Pode-se definir um vetor como:

$$V = V(1), V(2), ..., V(n)$$

onde *V(i)* é o i-ésimo elemento do vetor *V* e $1 <= i <= n$.

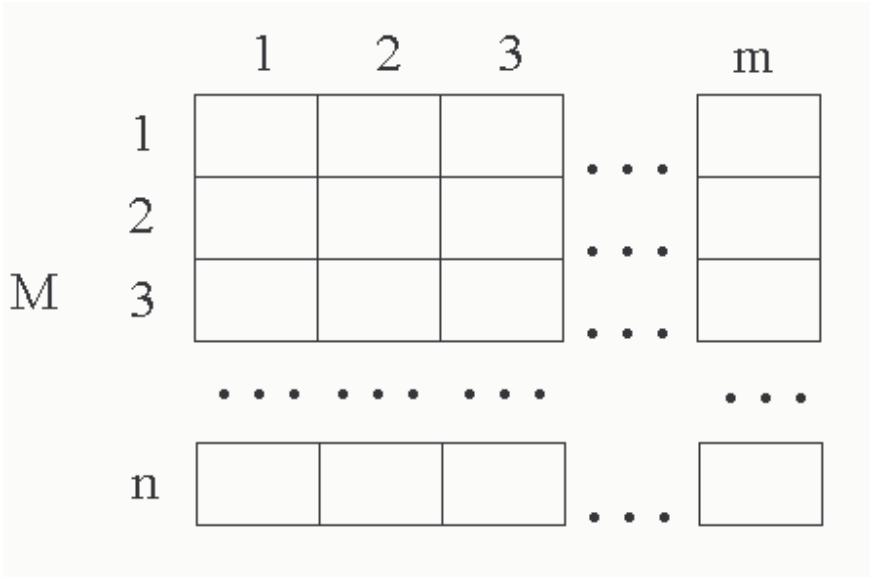


- **Matrizes** : um arranjo de várias dimensões de dados do tipo básico, com um mesmo identificador (nome), mas diferenciado pelas suas posições (índices) dentro da matriz.

  Uma matriz bidimensional pode ser definida como:

$$M = \begin{matrix} M(1,1) & M(1,2) & ,,, & M(1,m) \\ M(2,1) & M(2,2) & ... & M(2,m) \\ ... & ... & ... & ... \\ M(n,1) & M(n,2) & ... & M(n,m) \end{matrix}$$

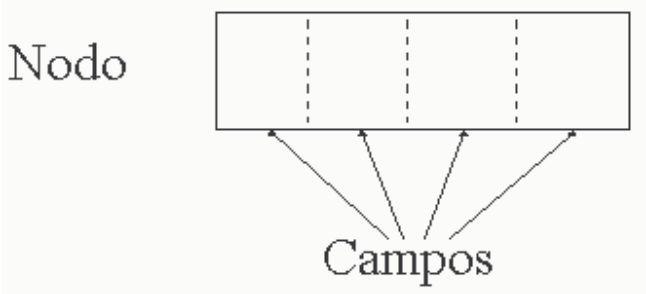
onde *M(i,j)* indica o elemento da matriz *M* posicionado na i-ésima linha e j-ésima coluna, sendo que 1 <= *i* <= *n* e 1 <= *j* <= *m*.



Nodo

O termo estrutura de dados é utilizado para referenciar diferentes formas de representação de dados. A escolha de uma determinada estrutura para representar um conjunto de dados relacionados deve-se, principalmente, ao tipo de operações que serão realizadas sobre o mesmo, usando uma manipulação otimizada em relação ao tempo necessário para efetuar as operações e á área de armazenamento requisitada para guardar estes dados.

A entidade elementar de uma estrutura de dados é o **nodo** (nó). Um nodo é diferenciado pelo seu endereço relativo dentro da estrutura e pode ser constituído de um ou vários campos.



Cada campo de um nodo armazena uma informação, um dado, que pode ser do tipo numérico, alfanumérico, lógicos, imagens, sons, enfim, qualquer informação que possa ser manipulada.

Todos os nodos de uma mesma estrutura devem ter a mesma configuração.

Exemplo:

Listas Lineares

Uma lista é uma seqüência ordenada de elementos do mesmo tipo. Por exemplo, um conjunto de fichas de clientes de uma loja, organizadas pela ordem alfabética dos nomes dos clientes. Neste fichário é possível introduzir uma nova ficha ou retirar uma velha, alterar os dados de um cliente etc.

Do ponto de vista matemático, uma lista é uma seqüência de zero ou mais elementos de um determinado tipo. Geralmente se representa uma lista de elementos, separando-os por vírgulas.

$$a1 , a2 , ... na$$

onde n >= 0 e *a*i é um elemento da lista. O número n é dito comprimento da lista. Se n=0 temos uma lista vazia.

Definição dada por Knuth para uma lista linear : "Uma lista linear *X* é um conjunto de nodos *X*(1), *X*(2), ...,*X*(n), tais que:

a) *X*(1) é o primeiro nodo da lista;
b) *X*(n) é o último nodo da lista; e
c) Para 1<k<n, *X*(k) é precedido pelo nodo *X*(k-1) e sucedido por *X*(k+1)."

Ou, mais simplesmente, "Uma lista linear é a estrutura de dados que permite representar um conjunto de dados de forma a preservar a relação de ordem linear entre eles."

Basicamente, o manuseio de listas lineares envolve três tipos de operações:

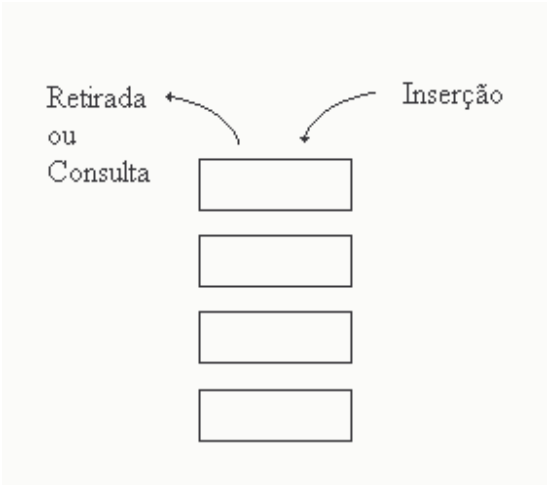
1) Inserção de novos nodos à lista;
2) Retirada de nodos da lista;
3) Consulta de conteúdo de algum nodo da lista.

Naturalmente existem outras operações que podem ser efetuadas sobre uma lista, que não são tão importantes para a caracterização dos diferentes tipos de listas, tais como:

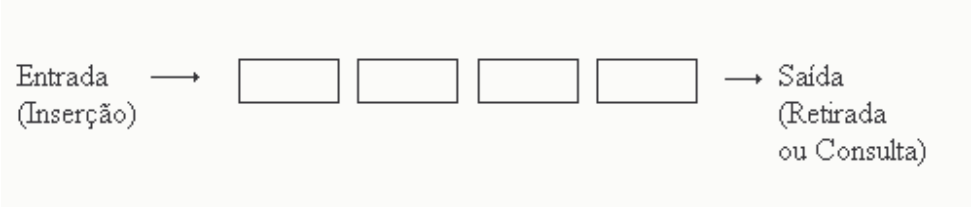
4) Concatenar duas listas;
5) Determinar o número de nodos de uma lista;
6) Localizar um nodo da lista com um determinado conteúdo;
7) Criação de uma lista;
8) Cópia de uma lista;
9) Ordenação de uma lista por algum critério;
10) Destruição de uma lista;
11) Modificação do conteúdo de algum nodo da lista.

As modalidades mais utilizadas de listas lineares são aquelas em que as três operações (1, 2 e 3) são realizadas nas extremidades da lista. Além disso, ao se manipular listas lineares, as operações freqüentemente se repetem. Devido à constante utilização de certas formas básicas de operações, foram definidos alguns tipos de listas lineares clássicas, de acordo com a forma que essas operações são realizadas. Estes tipos particulares de listas lineares são:

PILHA (*Stack*) : é uma lista linear em que todas as operações (inserção, retirada e consulta) são realizadas numa única extremidade da estrutura. Graficamente, uma pilha é representada por:

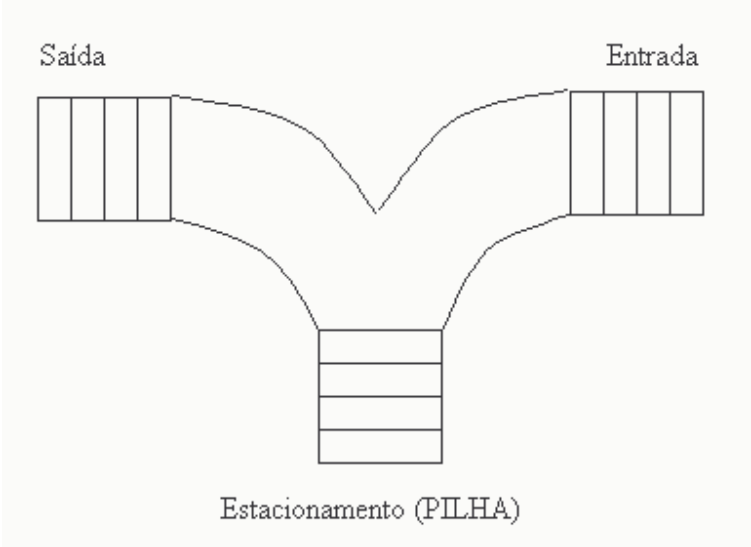


FILA (*Queue*) : é uma lista linear em que as inserções de novos nodos na estrutura são realizadas em uma das extremidades e as retiradas ou consultas aos nodos são realizadas na outra extremidade da lista. Graficamente tem-se:



Exercício:

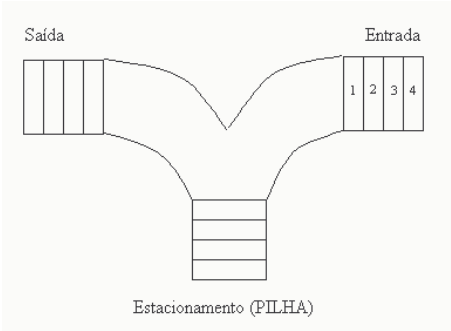
Imagine um terminal férreo com a seguinte configuração de trilhos:



Os vagões só podem entrar na área Estacionamento (que é estruturado como uma pilha) vindos da Entrada e só podem sair da pilha pela Saída. Denotando-se por I a entrada de um vagão no Estacionamento (inserção) e por R a saída de um vagão do Estacionamento (retirada) e considerando-se que na Entrada há quatro vagões numerados de 1 a 4, respectivamente (1 2 3 4) a execução da seqüência de operações de inserções e retiradas

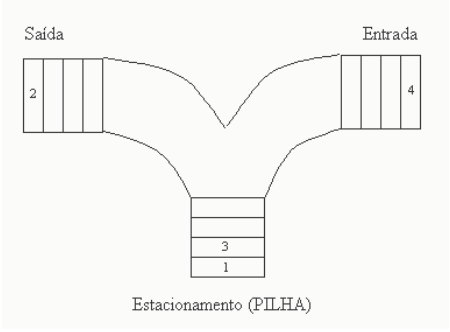
I I R I I R R R

Sobre os vagões da Entrada resultará na seguinte permutação dos vagões na Saída : 2 4 3 1.



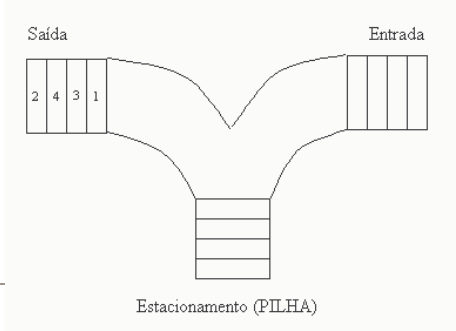
Isto é, considerando a configuração inicial (a), após a execução da seqüência I I R I, obter-se-á (b) :

(a)



(b)

e após toda a seqüência de operações ser efetuada, ter-se-á a seguinte configuração.



1) Se existirem seis vagões na Entrada (1 2 3 4 5 6), existe uma seqüência de operações que aplicada sobre a Entrada fornecerá na Saída a ordenação 3 2 5 6 4 1 dos vagões? Se sim, qual é a seqüência de operações? E uma seqüência que forneça a permutação 1 5 4 6 2 3? Se sim, qual é a seqüência?

Formas de representação de listas

Extraído de : Estrutura de Dados – Afonso Inácio Orth

Existem diferentes possibilidades de representação interna das listas lineares, mas duas destas formas são particularmente conhecidas e utilizadas, a saber:

- Alocação seqüencial (por contigüidade física)
- Alocação encadeada

O uso de uma ou outra destas formas depende, principalmente, do número de vezes que determinadas operações serão efetuadas sobre a lista

Alocação Seqüencial

A alocação seqüencial de uma lista significa que os nodos da estrutura são armazenados em áreas contíguas e seqüencialmente segundo a sua ordem lógica dentro da estrutura. Isto é, a ordem de alocação física e lógica dentro da estrutura coincide.

A alocação seqüencial explora a seqüencialidade natural da memória do computador, armazenando os nodos da lista em posições vizinhas ou igualmente distantes uma da outra. Esquematicamente isto é mostrado a seguir:

A implementação deste tipo de lista é usualmente feita por meio do uso de Vetores.

Pilhas

Dois conceitos são importantes para o estudo de pilhas:

Topo da pilha : é o endereço (índice) do elemento mais recentemente alocado (inserido) na pilha. O topo de uma pilha será denotado pela variável **T**.

Base da pilha : é o endereço (índice) do nodo anterior ao nodo mais 'antigo' da pilha. Inicialmente, quando a pilha está vazia, a base tem valor zero e a pilha não estará vazia se a diferença entre o valor do topo e da base for maior que zero.

Exemplo:

PILHA CHEIA:

PILHA VAZIA:

***OVERFLOW* e *UNDERFLOW* em pilhas**

Diz-se que ocorre uma **situação de overflow** na pilha quando um novo nodo deve ser inserido , mas não há mais nodos disponíveis na mesma (um nodo está disponível quando não estiver vinculado à pilha, por exemplo, todos os nodos alocados entre o topo da pilha e a posição de maior índice do vetor que contém a pilha são nodos disponíveis).

Ocorre uma **situação de underflow** na pilha quando um nodo deve ser retirado da pilha, mas a pilha está vazia.

ALGORITMOS PARA MANIPULAÇÃO DE PILHAS

Algoritmo de CRIAÇÃO

Algoritmo de INCLUSÃO

Algoritmo de CONSULTA

Algoritmo de RETIRADA

Exercício: considere uma área de armazenamento de 5 nodos. Sobre esta área será montada uma pilha de nome PILHA. Qual o comportamento da pilha durante as seguintes operações:

- Inclusão de um nodo com valor VERMELHO
- Inclusão de um nodo com valor VERDE
- Inclusão de um nodo com valor AMARELO
- Inclusão de um nodo com valor BRANCO
- Inclusão de um nodo com valor PRETO
- Retirada de um nodo
- Retirada de um nodo
- Inclusão de um nodo com valor ROSA
- Retirada de um nodo
- Retirada de um nodo
- Retirada de um nodo
- Inclusão de um nodo com valor CINZA
- Retirada de um nodo
- Retirada de um nodo
- Retirada de um nodo

FILAS SIMPLES

Dois conceitos são importantes para o estudo de filas:

Ré da Fila : é o endereço (índice) do nodo do elemento mais recentemente alocado (inserido) na fila. A ré de uma fila será denotada pela variável **R**.

Frente da fila : é o endereço (índice) do nodo anterior ao nodo mais 'antigo' da fila. Inicialmente, quando a fila está vazia, a frente tem valor zero e a fila não estará vazia se o valor da frente for diferente do valor da ré. A frente será denotada pela variável **F**.

Exemplo:

FILA SIMPLES CHEIA :

FILSA SIMPLES VAZIA :

***OVERFLOW* e *UNDERFLOW* em filas**

Diz-se que ocorre uma **situação de overflow** na fila quando um novo nodo deve ser inserido, mas não há mais nodos disponíveis na mesma.

Ocorre uma **situação de underflow** na fila quando um nodo deve ser retirado da fila, mas a fila está vazia.

Falso *Overflow*

ALGORITMOS PARA MANIPULAÇÃO DE FILAS

Algoritmo de CRIAÇÃO

Algoritmo de INCLUSÃO

Algoritmo de CONSULTA

Algoritmo de RETIRADA

Exercício: considere uma área de armazenamento de 5 nodos. Sobre esta área será montada uma fila de nome FILA. Qual o comportamento da fila durante as seguintes operações:

- Inclusão de um nodo com valor VERMELHO
- Inclusão de um nodo com valor VERDE
- Inclusão de um nodo com valor AMARELO
- Inclusão de um nodo com valor BRANCO
- Retirada de um nodo
- Retirada de um nodo
- Inclusão de um nodo com valor ROSA
- Retirada de um nodo
- Retirada de um nodo
- Retirada de um nodo
- Inclusão de um nodo com valor CINZA
- Retirada de um nodo
- Retirada de um nodo
- Retirada de um nodo

FILAS CIRCULARES

Os mesmos conceitos de Frente e Ré das filas simples são utilizados em filas circulares. A diferença é que, na fila circular, após os ponteiros alcançarem o final da estrutura (vetor), tenta-se ocupar os nodos inicias da estrutura (vetor), fazendo-se um 'círculo'.

Exemplo:

FILA CIRCULAR CHEIA:

FILA CIRCULAR VAZIA:

***OVERFLOW* e *UNDERFLOW* em filas**

Diz-se que ocorre uma **situação de overflow** na fila quando um novo nodo deve ser inserido, mas não há mais nodos disponíveis na mesma.

Ocorre uma **situação de underflow** na fila quando um nodo deve ser retirado da fila, mas a fila está vazia.

Falso *Overflow*

ALGORITMOS PARA MANIPULAÇÃO DE FILAS

Algoritmo de CRIAÇÃO

Algoritmo de INCLUSÃO

Algoritmo de CONSULTA

Algoritmo de RETIRADA

Exercício: considere uma área de armazenamento de 5 nodos. Sobre esta área será montada uma fila circular de nome FILA_CIRC. Qual o comportamento da fila durante as seguintes operações:

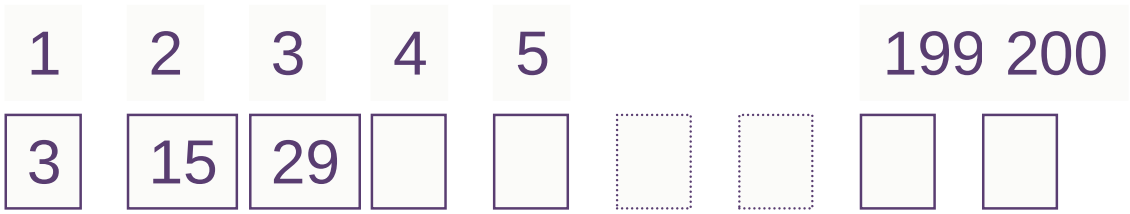
- Inclusão de um nodo com valor VERMELHO
- Inclusão de um nodo com valor VERDE
- Inclusão de um nodo com valor AMARELO
- Inclusão de um nodo com valor BRANCO
- Retirada de um nodo
- Retirada de um nodo
- Inclusão de um nodo com valor ROSA
- Retirada de um nodo
- Retirada de um nodo
- Inclusão de um nodo com valor CINZA
- Retirada de um nodo
- Inclusão de um nodo com valor MARROM
- Inclusão de um nodo com valor LARANJA

Alocação Encadeada

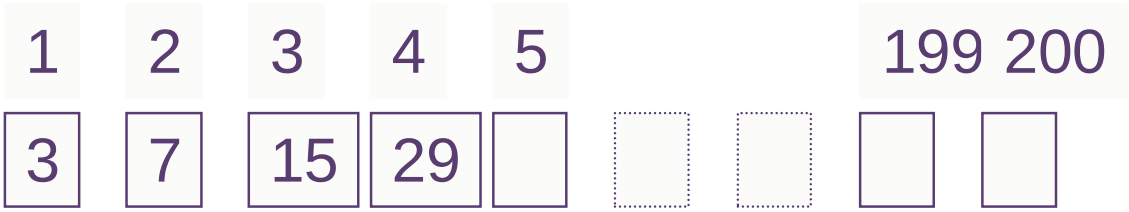
Como resolver-se-ia de forma eficiente o seguinte problema:

> "Gerar 200 números aleatórios. A cada geração de um número, este deve ser inserido numa lista linear de forma que os números componentes da mesma estejam constantemente organizados segundo a ordem crescente de seus valores".

Se num determinado instante a lista for :



e for gerado o número 7, após a sua inserção a lista ficará:



Na tentativa de inserir um novo elemento à lista, eventualmente é necessário deslocar ("empurrar") uma porção de lista a fim de que o novo nodo seja colocado em sua posição correspondente.

Esta solução ao problema é bastante onerosa, devido ao tempo necessário para ser realizado o deslocamento das informações contidas nos nodos e este torna-se crítico se for considerado um número bastante grande de elementos.

O uso de nodos compostos de vários campos, sendo que um deles tem como objetivo indicar o endereço do nodo que o segue logicamente dentro da estrutura, torna a resolução de certos problemas bastante mais eficiente.

Considerando-se um nodo com a configuração:

em que o campo INFO é formado de uma ou mais informações (campos) com as informações contidas no nodo ou uma indicação da localização destas informações e o campo ELO contém o endereço (índice) do nodo seguinte na estrutura.

Ao implementar-se estruturas encadeadas, utiliza-se uma matriz (ou vetor, dependendo das informações contidas nos nodos), ou ainda um registro, para alocar os campos INFO dos nodos e um vetor para os campos ELO. Segundo esta forma de representação, os vetores INFO e ELO:

	Info	Elo
1		
2	F	0
3	A	8
4		
5	B	2
6		
7		
8	C	11
9		
10		
11	K	5

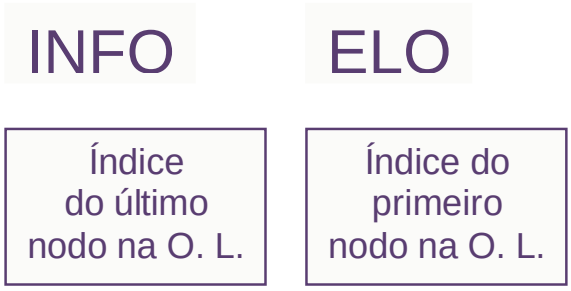
corresponde à seguinte lista encadeada:

Neste tipo de alocação, denominada *alocação encadeada*, a **Ordem Lógica** dos componentes da lista é determinada através do campo ELO, não importando a localização física dos nodos dentro dos vetores que alocam a lista.

Qualquer nodo da lista pode ser acessado se a mesma for percorrida seguindo-se os endereçamentos fornecidos pelos campos ELO de cada nodo. O campo ELO do último nodo da lista (no exemplo, o nodo índice 2) contém o valor zero, indicando que não há mais nodos sucedendo-o. Basta, portanto, conhecer-se o índice do primeiro nodo da lista para que se tenha acesso a qualquer outro.

		Info	Elo
	1		
	2	F	0
Início	3	A	8
	4		
	5	B	2
	6		
	7		
	8	C	11
	9		
	10		
	11	K	5

Em geral, é utilizado um nodo especial, com função específica, cujos campos indicam os índices do primeiro e do último nodo da lista. Este nodo especial (e cada lista encadeada o tem) denomina-se **NODO-CABEÇA** e tem a seguinte forma:

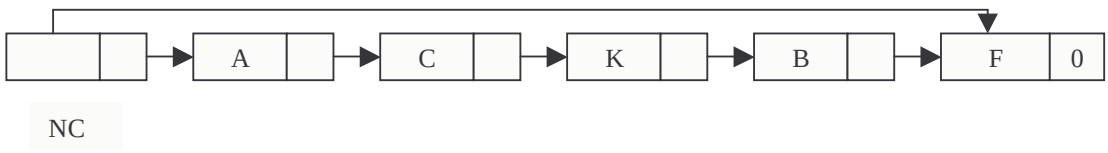


Cada lista, portanto, tem um nodo-cabeça associado. O mesmo também integrará a estrutura, bastando então ser conhecido o índice do nodo-cabeça para que toda a lista seja identificada.

Em geral, o índice do nodo-cabeça de uma lista será armazenado numa variável cujo nome seja significativo.

		Info	Elo
NC	1	**2**	**3**
	2	F	0
	3	A	8
	4		
	5	B	2
	6		
	7		
	8	C	11
	9		
	10		
	11	K	5

isto é, o nodo-cabeça da lista está no índice 1, o primeiro nodo no índice 3 e o último no índice 2 dos vetores. A representação gráfica desta mesma lista, incluindo o nodo cabeça é:



Continuamente são realizadas operações de inserção e retirada sobre a lista linear. Como saber, então, qual nodo está disponível para ser inserido ou como fazer para que um nodo, uma vez retirado da lista, possa ser novamente utilizado?

Todos os nodos que estão disponíveis (isto é, todas as posições dos vetores INFO e ELO que alocam a lista e não pertencem à mesma) formam uma estrutura a parte, do tipo pilha, e esta é denominada **PILHA DE NODOS DISPONÍVEIS** (**PND**).

O topo desta pilha tem o seu valor armazenado na variável **DISP** (disponível).

Inicialmente, todos os nodos da área de armazenamento que será utilizada para futura alocação das listas (supondo que estas não tenham sido criadas) são nodos disponíveis.

As seguintes são condições iniciais que a PND deve obedecer:

- todos os nodos da área estão encadeados entre si;
- a variável DISP contém a localização do primeiro destes nodos;
- o último nodo da pilha contém zero em seu campo ELO

Supondo que a variável TAM possui a quantidade de nodos na área a ser utilizada, um algoritmo para a criação da pilha de nodos disponíveis é o seguinte :

Algoritmo CRIA_PND

Na modificação dinâmica de listas encadeadas, sempre que for necessário inserir um nodo à mesma, este deve ser obtido na PND para então ser vinculado numa posição adequada.

O algoritmo para a obtenção de um nodo na PND é:

Algoritmo OBTEM

A função OBTEM deve retornar o índice do nodo disponível no momento, no qual será inserida a informação. Caso o valor retornado for 0 (zero), indica que não há mais nodos disponíveis. Ao mesmo tempo, a função deve atualizar a variável DISP para que ela aponte para o próximo nodo disponível.

O nodo-cabeça deve ser inicializado antes de fazer-se qualquer operação com a lista encadeada. Quando da sua inicialização, o valor para os campos Elo e Info do nodo-cabeça é 0 (zero)

	Info	Elo
NC	**0**	**0**

Portanto, a função de criação do nodo-cabeça é a seguinte:

Algoritmo CRIA_NC

Condição de Lista Cheia

OVERFLOW na lista

Condição de Lista Vazia

UNDERFLOW na lista

Listas Lineares Simplesmente Encadeada com ordenação pelo campo INFO

Na maioria das vezes a lista está ordenada pelo campo INFO (ou um de seus sub-campos), tanto em forma crescente como decrescente. Isto é, após cada inclusão ou retirada, ao percorrer-se a lista através do campo de encadeamento ELO, as informações deverão sempre estar ordenadas conforme o critério adotado. Os exemplos e algoritmos a seguir serão baseados numa lista linear simplesmente encadeada com ordenação pelo campo Info em ordem crescente (de A até Z, de 0 até 9).

Após seres definidos os algoritmos CRIA_PND, OBTEM e CRIA_NC, as operações na lista podem ser executadas.

As principais operações que ocorrem em uma lista são:

- Inclusão
- Retirada
- Consulta

Algoritmo de INCLUSÃO

Algoritmo de RETIRADA

Neste algoritmo não deve ser esquecido que o nodo que antes estava ocupado, após a sua retirada passa a ser disponível, devendo, portanto, ser incluído na PND.

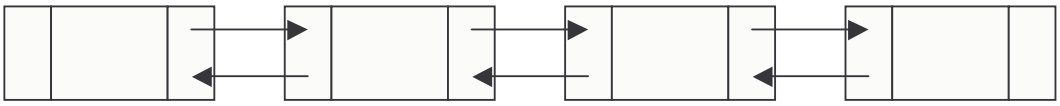
Algoritmo de CONSULTA

LISTAS DUPLAMENTE ENCADEADAS

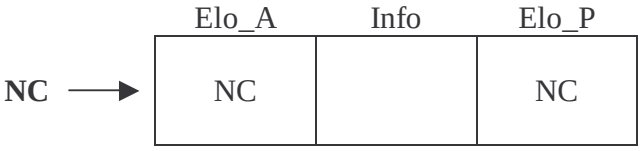
A estrutura encadeada considerada até aqui permite somente que a mesma seja percorrida em único sentido: qualquer pesquisa deve principiar no primeiro nodo da lista, conforme a Ordem Lógica, e continuar nos nodos seguintes até que seja encontrado o nodo procurado ou o último nodo da lista.

Entretanto, se a definição dos campos dos nodos for um pouco modificada, a lista poderá ser percorrida em ambos os sentidos: do início para o fim ou do fim para o início. Supõe-se, então, outro campo de elo em cada nodo, que passa a ter a seguinte configuração:

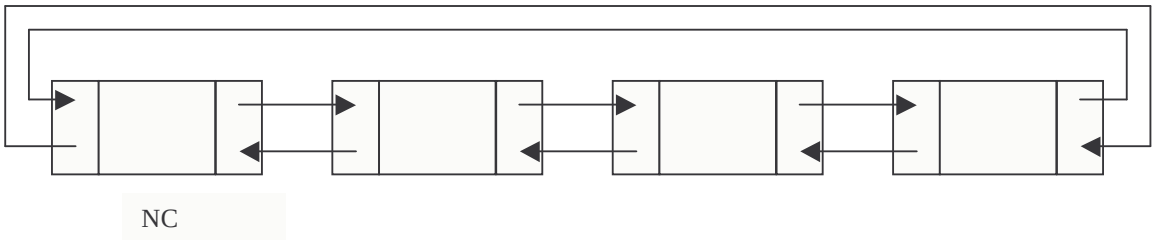
em que ELO_A(i) contém o índice do nodo anterior ao nodo i na lista e ELO_P(i) o índice do nodo posterior a i.



O nodo-cabeça nas listas duplamente encadeada possui uma inicialização diferente da lista simplesmente encadeada, pois os valores iniciais do ELO_A e do ELO_P do nodo-cabeça apontam para o índice do próprio nodo-cabeça.



Também diferente da lista simplesmente encadeada, o Elo_P do último nodo na ordem lógica, ao invés de apontar para 0 (zero), aponta para o nodo-cabeça. E o Elo_A do primeiro nodo na ordem lógica aponta também para o nodo-cabeça.



Condição de Lista Cheia

OVERFLOW na lista

Condição de Lista Vazia

UNDERFLOW na lista

Listas Lineares Duplamente Encadeada com ordenação pelo campo INFO

Muitas vezes a lista pode estar ordenada pelo campo INFO (ou um de seus sub-campos), tanto em forma ascendente como descendente. Isto é, após cada inclusão ou retirada, ao percorrer-se a lista através do campo de encadeamento ELO, as informações deverão sempre estar ordenadas conforme o critério adotado. Os exemplos e algoritmos a seguir serão baseados numa lista linear duplamente encadeada com ordenação pelo campo Info em ordem crescente (de A até Z, de 0 até 9).

Algoritmo CRIA_PND

Algoritmo OBTEM

Algoritmo CRIA_NC

Algoritmo INCLUSÃO

Algoritmo RETIRADA

Algoritmo CONSULTA