Estruturas de Dados

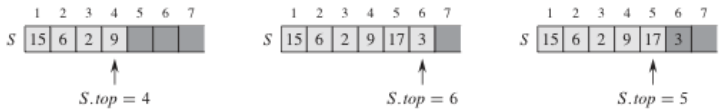
Estruturas básicas

1. Estruturas básicas (Cormen, capítulo 10. Seções 10.1, 10.2 e 10.4)

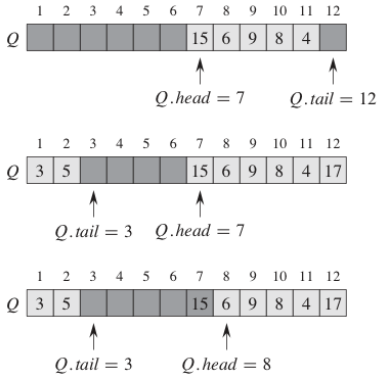
Estruturas de dados elementares: pilhas, filas, listas encadeadas e árvores enraizadas.

FIlas e Pilhas: são conjuntos dinâmicos usados para armazenar conjuntos de dados, e se diferenciam na forma como a operação DELETE é realizada. Existem várias formas eficientes de implementar filas e pilhas. Uma das formas mais simples e comuns é através de arranjos.

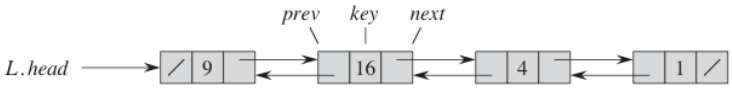
Pilhas: LIFO (last in, first out). O elemento eliminado é sempre o último a entrar. Geralmente usa operações com nome de PUSH e POP. O custo de remoção e adição de elementos é constante.
Exemplo de aplicação: backtracking.



FIlas: FIFO (first in, first out). O elemento eliminado é sempre o que está a mais tempo na fila. Geralmente usa operações com nome de ENQUEUE e DEQUEUE. O custo de remoção e adição de elementos é constante.



Listas encadeadas: estrutura de dados onde objetos estão organizados em uma ordem linear. A ordem dos objetos é determinada por um ponteiro entre cada objeto. Cada elemento da lista é um objeto com um atributo *key* e um ou dois ponteiros, *next* e *prev*.



Uma lista pode ter várias formas: simplesmente encadeadas ou duplamente encadeadas, ordenada ou não, circular ou não.

Se uma lista é **duplamente encadeada**, cada objeto da lista possui um ponteiro *prev* e um ponteiro *next*, apontando para o elemento anterior e para o próximo elemento da lista, respectivamente. Se uma lista é **simplesmente encadeada**, nós omitimos o ponteiro *prev* da estrutura de dados.

Se uma lista é **ordenada**, a ordem linear da lista corresponde à ordem linear das chaves armazenadas nos elementos da lista. O elemento de menor chave é, então, o primeiro elemento da lista, e o de maior chave é o último elemento da lista. Se a lista é **não ordenada** os elementos podem aparecer em qualquer ordem.

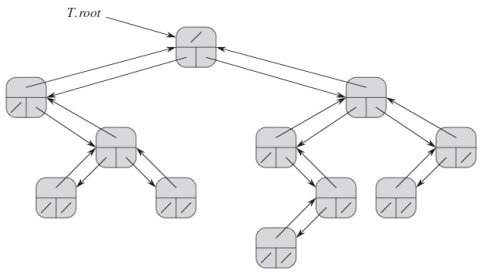
Se uma lista é **circular**, o ponteiro *prev* do primeiro elemento da lista aponta para o último elemento da lista, enquanto o ponteiro *next* do último elemento da lista aponta para o primeiro elemento da lista.

Operações em listas encadeadas:

- **Busca**: percorre toda a lista a partir do primeiro elemento, até que o elemento procurado seja encontrado ou a lista termine. Essa operação tem, no pior caso, complexidade O(n).
- **Inserção**: em uma lista não ordenada, basta inserir o elemento no início da lista. A operação tem complexidade constante (O(1)). Em uma lista ordenada é necessário, primeiro, percorrer a lista para encontrar o local de inserção e, em seguida, realizar a inserção. Nesse caso a complexidade será linear (O(n)).
- **Remoção**: se já temos um apontador para o elemento a ser removido, e a lista é duplamente encadeada, a operação tem complexidade O(1). Caso contrário, é necessário realizar uma pesquisa e depois uma remoção e a complexidade é da ordem de O(n).

Árvores enraizadas: como representar árvores? Usando estruturas de dados encadeadas, como listas.

Árvores binárias: a representação de árvores binárias é fácil. Cada nó da árvore possui um campo chave e 3 atributos, *p*, *left* e *right*, que apontam para o pai, filho da esquerda e filho da direita. Se um nó *x* temp *p* nulo, então *x* é a raiz da árvore.



O que fazer quando cada nó da árvore tem um número arbitrário de filhos, ou seja, quando temos uma **árvore com ramificação ilimitada**? Usamos a representação de **filho da esquerda, irmão da direita**.

Cada nó aponta para seu pai;
left-child aponta para o filho mais a esquerda;
right-sibling aponta para o irmão imediatamente à direita;

se *x* não possui filhos, *x.left-child* = NIL
se *x* é o filho mais à direita, *x.right-sibling* = NIL