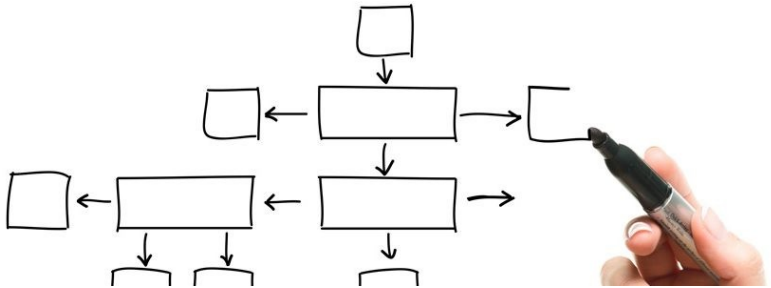
Estrutura de dados elementares (pilhas,filas e listas)

Neste capítulo

- Introdução
- Pilhas
- Filas
- Listas
- revisão

Introdução

Conjuntos são tão fundamentais para a Ciência da Computação quanto para a Matemática. Enquanto os conjuntos matemáticos são invariáveis, os conjuntos manipulados por algoritmos podem crescer, encolher ou sofrer outras mudanças ao longo do tempo. Estrutura de dados são formas de se representar esses conjuntos de modo que sejam usados em algoritmos.

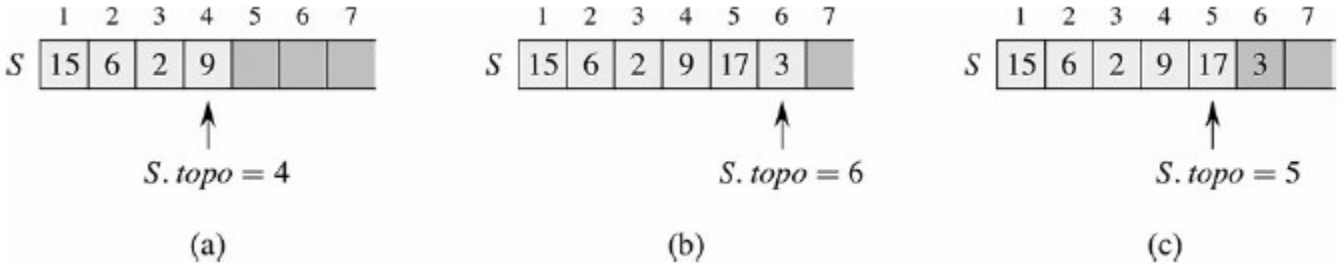


Pilha

é uma estrutura de dados que implementa a política LIFO (last in, first out), ou seja, ultimo a entrar é o primeiro a sair.

Operações realizadas por uma pilha:

- Push
- Pop
- isEmpty



```
STACK-EMPTY(S)
1   if S.topo == 0
2       return TRUE
3   else return FALSE

PUSH(S, x)
1   S.topo = S.topo + 1
2   S[S.topo] = x

POP(S)
1   if STACK-EMPTY(S)
2       error "underflow"
3   else S.topo = S.topo - 1
4       return S[S.topo + 1]
```

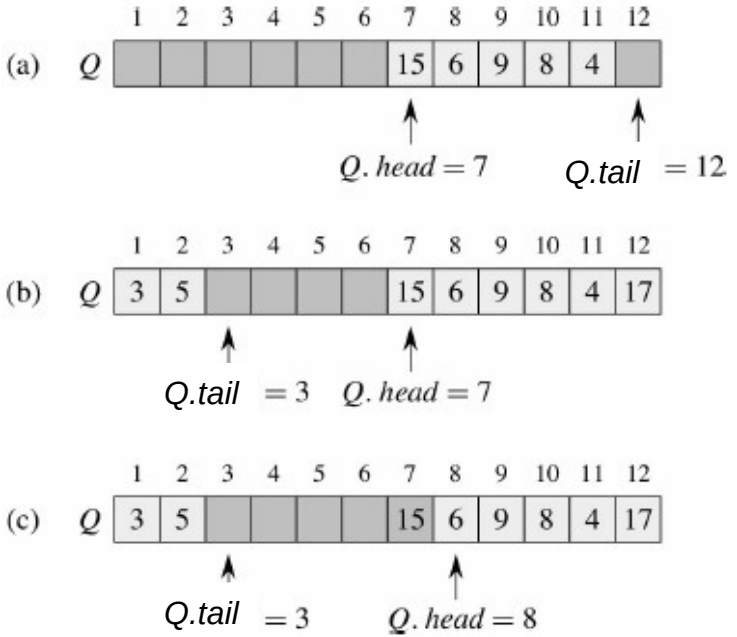
Pseudo código de uma pilha

Fila

É uma estrutura de dados que implementa uma política FIFO (First in, First out), ou seja, primeiro que entra é o primeiro que sai.

Operações realizadas por uma Fila:
- enqueue
- dequeue

A fila possui uma cabeça e uma calda.



```
ENQUEUE(Q, x)
1   Q[Q.fim] = x
2   if Q.fim = Q.comprimento
3       Q.fim = 1
4   else Q.fim = Q.fim + 1

DEQUEUE(Q)
1   x = Q[Q.início]
2   if Q.início == Q.comprimento
3       Q.início = 1
4   else Q.início = Q.início + 1
5   return x
```

Pseudo código de uma fila

Exercício

- Usando a figura acima como modelo, ilustre o resultado de cada operação na sequência PUSH(S, 4), PUSH(S, 1), PUSH (S, 3), POP (S), PUSH (S, 8) e POP (S) sobre uma pilha S inicialmente vazia armazenada no arranjo S[1 .. 6].
- Usando acima como modelo, ilustre o resultado de cada operação na sequência ENQUEUE(Q, 4), ENQUEUE (Q, 1), ENQUEUE (Q, 3), DEQUEUE (Q), ENQUEUE (Q, 8) e DEQUEUE (Q) em uma fila Q inicialmente vazia armazenada no arranjo Q[1 .. 6].
- Implemente o uma pilha usando orientação a objetos
- Implemente o uma fila usando orientação a objetos

Não precisamos reinventar a roda

- Inclua a biblioteca “<stack>” em c++
- .push() - adiciona elementos à pilha
- .top() - mostra o último elemento da pilha
- .pop() - retira o último elemento da pilha
- .empty() - retorna verdadeiro se a pilha estiver vazia
- .size() - retorna o tamanho da pilha

Não precisamos reinventar a roda

- Inclua a biblioteca “<queue>” em c++
- .push() - adiciona elementos à fial
- .front() - mostra a cabeça da fila
- .back() - mostra a cauda da fila
- .pop() - retira o último elemento da fila
- .empty() - retorna verdadeiro se a fila estiver vazia
- .size() - retorna o tamanho da pilha

Listas

Problemas de uma lista estática

- Quantidade fixa de elementos
- Memória alocada sem uso
- Impossibilidade de alocar mais memória

Lista ligada

- Cresce e diminui dinamicamente
- Objetos estão organizados em ordem linear
- Utiliza um ponteiro para cada item ao invés de índices
- Pode armazenar diferentes tipos de dados

Tipos de listas

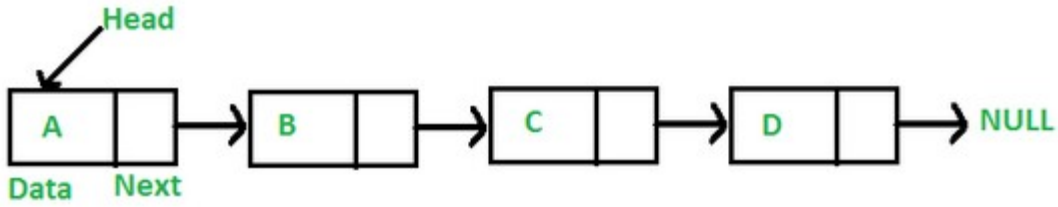
- Simplesmente ligada
  - Só possui ponteiro para o próximo elemento
- Duplamente ligada
  - Possui ponteiros para o próximo e para o anterior
- Lista ordenada
  - Possui os dados ordenados
- Lista circular
  - Ponteiro anterior do início aponta para o fim, e o ponteiro próximo do fim aponta para o início

Lista simplesmente ligada

- Composto por nós dinamicamente conectados
- Possui dois membros: dados e um ponteiro
- Head aponta para o nó inicial

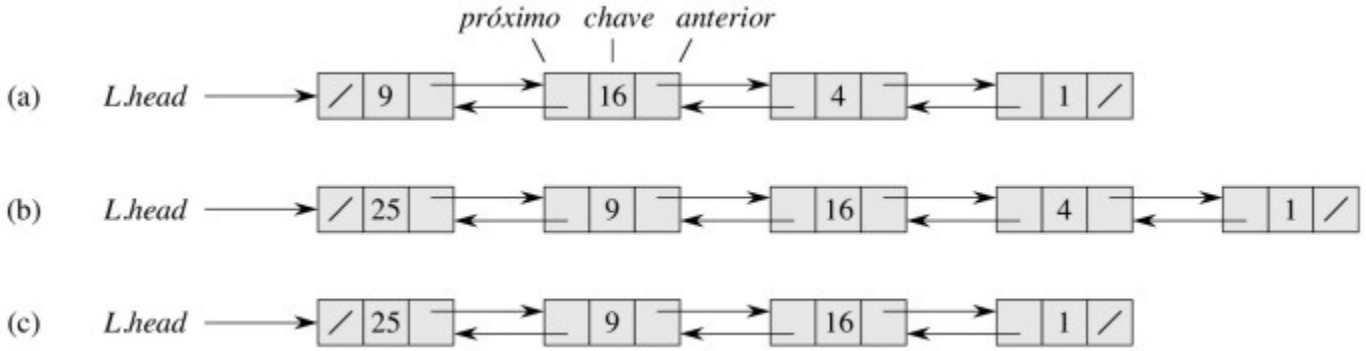
Operações realizadas por uma lista:
- Pesquisar
- Adicionar
- Remover
- Inserir

A lista possui uma cabeça mas não tem calda.



Lista duplamente ligada

- Possui dois ponteiros
- *próximo
- *anterior



Como fazer pesquisa?

```
LIST-SEARCH(L, k)
1   x = L.início
2   while x ≠ NIL e x.chave ≠ k
3       x = x.próximo
4   return x
```

- Busca linear simples
- Retorna o ponteiro para o elemento
- Retorna nulo se não achar
- O(n) no pior caso

Como inserir elemento?

- Corresponde a adicionar no início
- Também é possível inserir na posição desejada
- Como seria para inserir em uma lista simplesmente ligada?

```
LIST-INSERT(L, x)
1  x.próximo = L.início
2  if L.início ≠ NIL
3      L.início.anterior = x
4  L.início = x
5  x.anterior = NIL
```

Como remover um elemento?

- Recebe um ponteiro para x
- Desliga x da lista
- Para eliminar de acordo com a chave ou índice:
  - Pesquisar pelo elemento na lista
  - O(n) no pior caso

```
LIST-DELETE(L, x)
1   if x.anterior ≠ NIL
2       x.anterior.próximo = x.próximo
3   else L.início = x.próximo
4   if x.próximo ≠ NIL
5       x.próximo.anterior = x.anterior
```

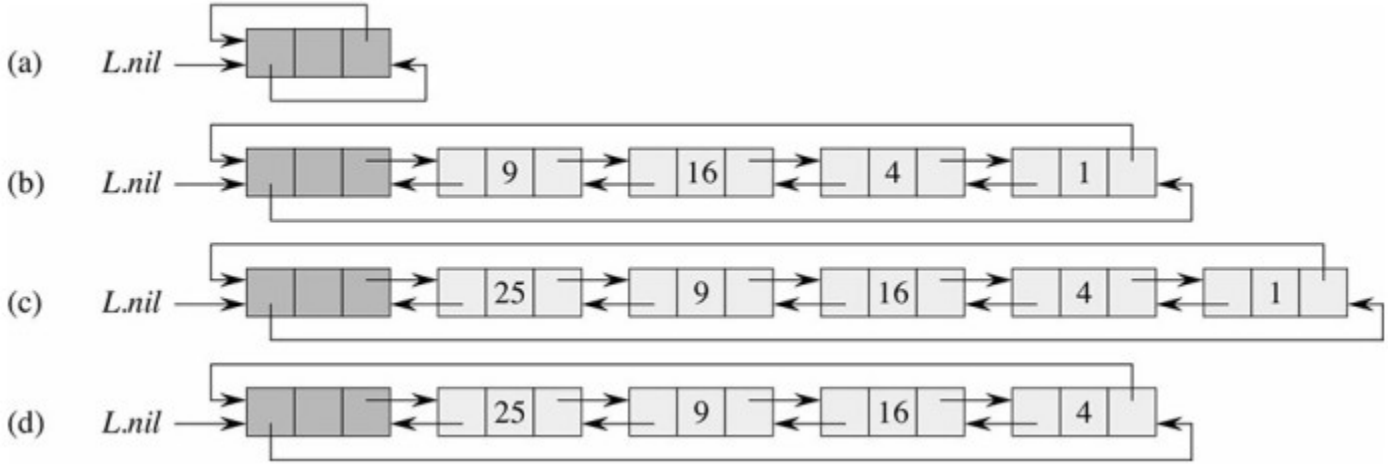
Removendo usando sentinela

- Se ignorar as condições de contorno
- No ínicio
- E no fim da lista

```
LIST-DELETE′(L, x)
1  x.anterior.próximo = x.próximo
2  x.próximo.anterior = x.anterior
```

Lista circular

- Lista duplamente ligada
- Com sentinela



Ajustes nas operações

```
LIST-SEARCH'(L, k)
1  x = L.nil.próximo
2  while x ≠ L.nil e x.chave ≠ k
3      x = x.próximo
4  return x
```

```
LIST-INSERT'(L, x)
1  x.próximo = L.nil.próximo
2  L.nil.próximo.anterior = x
3  L.nil.próximo = x
4  x.anterior = L.nil
```

Exercícios

- Implemente uma pilha usando uma lista simplesmente ligada L. As operações PUSH e POP ainda devem demorar o tempo O(1).
- Implemente uma fila por meio de uma lista simplesmente ligada L. As operações ENQUEUE e DEQUEUE ainda devem demorar o tempo O(1).

Não precisamos reinventar a roda

- .push_front() Insere termos por cima da lista
- .push_back() Insere termos por baixo da lista
- .pop_front() Retira termos por cima da lista
- .pop_back() Retira termos por baixo da lista
- .clear() Apaga todos os termos da lista

Métodos úteis

- Iterator – percorre a lista
- .sort() - ordena a lista
- .reverse() - inverte a lista

Iterator

- O iterator é um método que pode ser utilizado para inserir termos em uma determinada posição da lista.

```
list<int>::iterator it;
it = ex.begin();
advance(it,5);
ex.insert(it,0);
```

Passo 1 – Declarar o iterator

```
#include<iostream>
#include<list>

using namespace std;

int main(){
    int i;
    list<int> ex;

    list<int>::iterator it; //declarar o iterator
```

Passo 2 – definir o começo

- Do início ou pelo fim

```
for(i=1 ; i<11 ; i++){
    ex.push_front(i);  // definir os termos da lista
}

it = ex.begin();
//ou it= ex.end(); se for começar a contar por baixo
```

Passo 3 – avançar a até a posição desejada

Isso indica a posição da lista onde você quer executar uma operação

```
advance(it,5);
//o interator "it" avançará 5 casas
```

Passo 4 – Realizar a operação

- .insert() : inserir um valor na lista
- .erase() : apagar um valor na lista
- .merge() : adiciona o valor de uma lista em outra.

Desafio de programação

- Faça um programa em que o usuário põe matrículas em uma lista “matricula” e, com isso, possa interagir de diferentes maneiras. A interface deve ter as opções:
  - Inserir matrícula (sempre que uma nova matrícula for inserida, ela deve, primeiro, ser colocada em uma fila diferente para depois ser adicionada na lista utilizando o método .merge());
  - Apagar matrícula (a matrícula deve ser apagada usando o método .erase());
  - Limpar fila (a lista deve ser apagada com o método .clear());
  - Exibir lista (a lista deve ser inserida de maneira crescente).