Listas Lineares com Alocação Sequencial

Estrutura de Dados — QXD0010

Prof. Atílio Gomes Luiz

gomes.atilio@ufc.br

Universidade Federal do Ceará

2º semestre/2021

Introdução

Introdução

- Uma estrutura de dados armazena dados na memória do computador a fim de permitir o acesso eficiente dos mesmos.

- A maioria das estruturas de dados usam como recurso principal a memória primária (a chamada RAM) como pilhas, filas, árvores binárias de busca, árvores AVL e árvores rubro-negras.

- Outras são especialmente projetadas e adequadas para serem armazenadas em memórias secundárias como o disco rígido, como as árvores B.

- Uma estrutura de dados bem projetada permite a manipulação eficiente, em tempo e em espaço, dos dados armazenados através de operações específicas.

Lista linear — Definição

- Uma lista linear L é um conjunto de $n \geq 0$ nós (ou células) $L[0], L[1], \ldots, L[n-1]$ tais que suas propriedades estruturais decorrem, unicamente, da posição relativa dos nós dentro da sequência linear:
  - Se $n > 0$, $L[0]$ é o primeiro nó,
  - Para $0 < k \leq n-1$, o nó $L[k]$ é precedido por $L[k-1]$.

Lista linear — Definição

- Uma lista linear L é um conjunto de $n \geq 0$ nós (ou células) $L[0], L[1], \ldots, L[n-1]$ tais que suas propriedades estruturais decorrem, unicamente, da posição relativa dos nós dentro da sequência linear:
  - Se $n > 0$, $L[0]$ é o primeiro nó,
  - Para $0 < k \leq n-1$, o nó $L[k]$ é precedido por $L[k-1]$.
- Os nós de uma lista linear armazenam informações referentes a um conjunto de elementos que se relacionam entre si.
  - Informações sobre os funcionários de uma empresa.
  - Notas de alunos
  - Itens de estoque, etc.

Lista linear – Operações

Algumas operações que podemos querer realizar sobre listas lineares:

- Ter acesso a $L[k]$, $0 \leq k \leq n-1$ qualquer, a fim de examinar ou alterar o conteúdo de seus campos.

Lista linear – Operações

Algumas operações que podemos querer realizar sobre listas lineares:

- Ter acesso a $L[k]$, $0 \leq k \leq n-1$ qualquer, a fim de examinar ou alterar o conteúdo de seus campos.
  - Se fixamos $k = n-1$, temos uma ED chamada Pilha.

Lista linear – Operações

Algumas operações que podemos querer realizar sobre listas lineares:

- Ter acesso a $L[k]$, $0 \leq k \leq n-1$ qualquer, a fim de examinar ou alterar o conteúdo de seus campos.
  - Se fixamos $k = n-1$, temos uma ED chamada Pilha.
  - Se fixamos $k = 0$, temos uma ED chamada Fila.

Lista linear – Operações

Algumas operações que podemos querer realizar sobre listas lineares:

- Ter acesso a $L[k]$, $0 \leq k \leq n-1$ qualquer, a fim de examinar ou alterar o conteúdo de seus campos.
  - Se fixamos $k = n-1$, temos uma ED chamada Pilha.
  - Se fixamos $k = 0$, temos uma ED chamada Fila.
- Inserir um elemento novo antes ou depois de $L[k]$.

Lista linear – Operações

Algumas operações que podemos querer realizar sobre listas lineares:

- Ter acesso a $L[k]$, $0 \leq k \leq n-1$ qualquer, a fim de examinar ou alterar o conteúdo de seus campos.
  - Se fixamos $k = n-1$, temos uma ED chamada Pilha.
  - Se fixamos $k = 0$, temos uma ED chamada Fila.
- Inserir um elemento novo antes ou depois de $L[k]$.
- Remover $L[k]$.

Lista linear – Operações

Algumas operações que podemos querer realizar sobre listas lineares:

- Ter acesso a $L[k]$, $0 \leq k \leq n-1$ qualquer, a fim de examinar ou alterar o conteúdo de seus campos.
  - Se fixamos $k = n-1$, temos uma ED chamada Pilha.
  - Se fixamos $k = 0$, temos uma ED chamada Fila.
- Inserir um elemento novo antes ou depois de $L[k]$.
- Remover $L[k]$.
- Colocar todos os elementos da lista em ordem.

Lista linear – Operações

Algumas operações que podemos querer realizar sobre listas lineares:

- Ter acesso a $L[k]$, $0 \leq k \leq n-1$ qualquer, a fim de examinar ou alterar o conteúdo de seus campos.
  - Se fixamos $k = n-1$, temos uma ED chamada Pilha.
  - Se fixamos $k = 0$, temos uma ED chamada Fila.
- Inserir um elemento novo antes ou depois de $L[k]$.
- Remover $L[k]$.
- Colocar todos os elementos da lista em ordem.
  - Estudaremos algoritmos de ordenação no final do curso.

Lista linear – Operações

Algumas operações que podemos querer realizar sobre listas lineares:

- Ter acesso a $L[k]$, $0 \leq k \leq n-1$ qualquer, a fim de examinar ou alterar o conteúdo de seus campos.
  - Se fixamos $k = n-1$, temos uma ED chamada Pilha.
  - Se fixamos $k = 0$, temos uma ED chamada Fila.
- Inserir um elemento novo antes ou depois de $L[k]$.
- Remover $L[k]$.
- Colocar todos os elementos da lista em ordem.
  - Estudaremos algoritmos de ordenação no final do curso.
- Combinar duas ou mais listas lineares em uma só.

Lista linear – Operações

Algumas operações que podemos querer realizar sobre listas lineares:

- Ter acesso a $L[k]$, $0 \leq k \leq n-1$ qualquer, a fim de examinar ou alterar o conteúdo de seus campos.
  - Se fixamos $k = n-1$, temos uma ED chamada Pilha.
  - Se fixamos $k = 0$, temos uma ED chamada Fila.
- Inserir um elemento novo antes ou depois de $L[k]$.
- Remover $L[k]$.
- Colocar todos os elementos da lista em ordem.
  - Estudaremos algoritmos de ordenação no final do curso.
- Combinar duas ou mais listas lineares em uma só.
- Quebrar uma lista linear em duas ou mais.

Lista linear – Operações

Algumas operações que podemos querer realizar sobre listas lineares:

- Ter acesso a $L[k]$, $0 \leq k \leq n-1$ qualquer, a fim de examinar ou alterar o conteúdo de seus campos.
  - Se fixamos $k = n-1$, temos uma ED chamada Pilha.
  - Se fixamos $k = 0$, temos uma ED chamada Fila.
- Inserir um elemento novo antes ou depois de $L[k]$.
- Remover $L[k]$.
- Colocar todos os elementos da lista em ordem.
  - Estudaremos algoritmos de ordenação no final do curso.
- Combinar duas ou mais listas lineares em uma só.
- Quebrar uma lista linear em duas ou mais.
- Copiar uma lista linear em um outro espaço.

Implementação de uma lista linear

- O modo de implementar listas lineares depende da classe de operações mais frequentes. Não existe, em geral, uma única implementação para a qual todas as operações são eficientes.

Implementação de uma lista linear

- O modo de implementar listas lineares depende da classe de operações mais frequentes. Não existe, em geral, uma única implementação para a qual todas as operações são eficientes.

- Por exemplo, não existe uma implementação para atender às seguintes duas operações de maneira eficiente:

(1) ter acesso fácil ao elemento $L[k]$, para k qualquer.
(2) inserir ou remover elementos em qualquer posição da lista linear.

Implementação de uma lista linear

- O modo de implementar listas lineares depende da classe de operações mais frequentes. Não existe, em geral, uma única implementação para a qual todas as operações são eficientes.

- Por exemplo, não existe uma implementação para atender às seguintes duas operações de maneira eficiente:

(1) ter acesso fácil ao elemento $L[k]$, para k qualquer.
(2) inserir ou remover elementos em qualquer posição da lista linear.

A operação (1) fica eficiente se a lista é implementada em um vetor (array) em alocação sequencial na memória.

Implementação de uma lista linear

- O modo de implementar listas lineares depende da classe de operações mais frequentes. Não existe, em geral, uma única implementação para a qual todas as operações são eficientes.

- Por exemplo, não existe uma implementação para atender às seguintes duas operações de maneira eficiente:

(1) ter acesso fácil ao elemento $L[k]$, para k qualquer.
(2) inserir ou remover elementos em qualquer posição da lista linear.

A operação (1) fica eficiente se a lista é implementada em um vetor (array) em alocação sequencial na memória.

Para a operação (2) é mais adequada a alocação encadeada, com o uso de ponteiros.

Tipos de alocação

O tipo de armazenamento de uma lista linear pode ser classificado de acordo com a posição relativa na memória de dois nós consecutivos na lista.

- **Alocação sequencial:** dois nós consecutivos na lista estão em posições contíguas de memória.

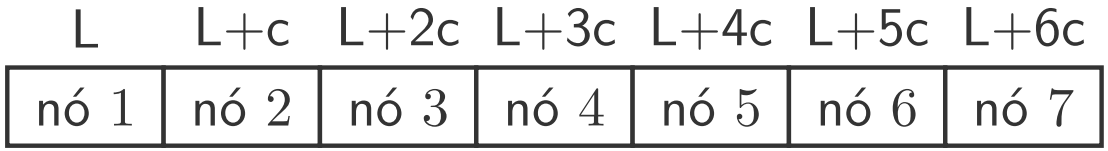
- **Alocação encadeada:** dois nós consecutivos na lista podem estar em posições não contíguas da memória.

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Listas Sequenciais

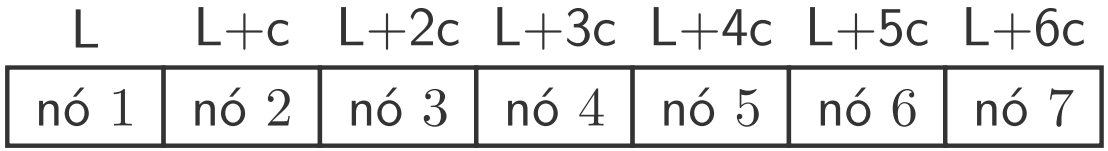
Listas sequenciais (Vetores)

- Nós em posições contíguas da memória.



Listas sequenciais (Vetores)

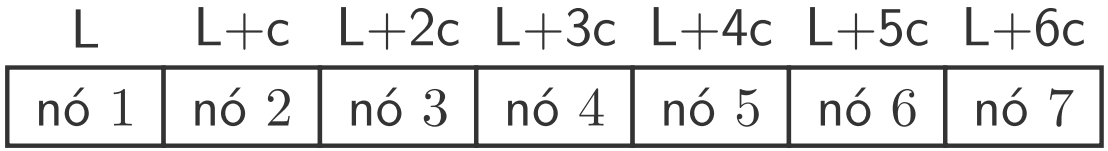
- Nós em posições contíguas da memória.



- Neste caso, o endereço real do $(j+1)$-ésimo nó da lista se encontra c unidades adiante daquele correspondente ao j-ésimo. A constante c é o número de bytes que cada nó ocupa.

Listas sequenciais (Vetores)

- Nós em posições contíguas da memória.



- Neste caso, o endereço real do $(j+1)$-ésimo nó da lista se encontra c unidades adiante daquele correspondente ao j-ésimo. A constante c é o número de bytes que cada nó ocupa.

- A correspondência entre o índice da lista e o endereço real é feita automaticamente pela linguagem de programação quando da compilação do programa.

TAD Lista Sequencial

- Lista sequencial pode ser modelada como um Tipo Abstrato de Dados.

TAD Lista Sequencial

- Lista sequencial pode ser modelada como um Tipo Abstrato de Dados.

- O TAD Lista Sequencial tem os seguintes atributos:
  - um vetor de inteiros.
  - a capacidade total do vetor.
  - a quantidade de elementos no vetor.
  - a posição atual do índice.

TAD Lista Sequencial

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Lista sequencial pode ser modelada como um Tipo Abstrato de Dados.

- O TAD Lista Sequencial tem os seguintes atributos:
  - um vetor de inteiros.
  - a capacidade total do vetor.
  - a quantidade de elementos no vetor.
  - a posição atual do índice.

- Operações possíveis são:
  - Criar lista.
  - Liberar lista.
  - Consultar o tamanho atual da lista.
  - Saber se lista está cheia.
  - Adicionar um elemento ao final da lista.
  - Remover um elemento da lista.

Implementação em C++

Programa cliente `main.cpp`

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  #include <iostream>
2  #include "SeqList.h"
3  using namespace std;
4
5  int main() {
6      SeqList list(30);
7      int i = 1;
8
9      while(!list.full()) {
10         list.push_back(i++);
11     }
12
13     cout << list << endl; // imprime lista na tela
14
15     return 0;
16 }
```

Arquivo SeqList.h

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  #ifndef SEQLIST_H
2  #define SEQLIST_H
3  #include <iostream>
4
5  class SeqList {
6  private:
7      int m_capacity; // capacidade total da lista
8      int m_size;     // numero de elementos na lista
9      int* m_array;   // ponteiro para o array de inteiros
10
11 public:
12     // Construtor
13     SeqList(int capacity);
14
15     // Destrutor
16     ~SeqList();
17
18     // Limpa a lista deixando-a vazia, com zero elementos
19     void clear();
```

Arquivo SeqList.h (continuação)

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1       // Retorna true se e somente se a lista estiver cheia
2       bool full() const;
3
4       // Adiciona um elemento ao final da lista
5       void push_back(int elemento);
6
7       // Adiciona o elemento na posição i da lista
8       void insert(int elemento, int i);
9
10      // Retorna o tamanho da lista
11      int size() const;
12
13      // Retorna a capacidade total da lista
14      int capacity() const;
15
16      // Retorna uma referência para o primeiro elemento
17      int& front();
18
19      // Retorna uma referência para o último elemento
20      int& back();
```

Arquivo SeqList.h


```
1       // Retorna o valor do elemento no índice i
2       int get(int i) const;
3
4       // Remove o elemento no índice i
5       void remove(int i);
6
7       // Remove o elemento na última posição da lista
8       void remove_back();
9
10      // Remove o elemento na primeira posição da lista
11      void remove_front();
12
13      // operador[] sobrecarregado como uma função-membro
14      int& operator[](int index);
15
16      // operador<< sobrecarregado como uma função global friend
17      friend std::ostream& operator<<(std::ostream& out,
18              const SeqList& list);
19  };
20
21  #endif
```

Arquivo `SeqList.cpp`

Exercício: Implementar as funções-membro da classe SeqList.

Exercícios

Exercícios

Implemente as seguintes operações adicionais na Lista Sequencial:

- `void replaceAt(int x, int k)`: Troca o elemento no índice k pelo elemento x (somente se $0 \leq k \leq size\_vec - 1$)

- `void removeAt(int k)`: Remove o elemento com índice k na lista. Deve-se ter $0 \leq$ `k` $\leq$ `size_vex-1`; caso contrario, a remoção não é realizada.

- `bool insertAt(int x, int k)`: Adiciona o elemento x no índice k (somente se $0 \leq k \leq size\_vec$ e $size\_vec < capacity\_vec$). Antes de fazer a inserção, todos os elementos do índice k em diante são deslocados uma posição para a direita.

- `void removeAll(int x)`: Remove todas as ocorrências do elemento x na lista.

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

FIM