

UFMG
UNIVERSIDADE FEDERAL
DE MINAS GERAIS

Programação e Desenvolvimento de Software 2

Listas encadeadas e Árvores binárias

Prof. Douglas G. Macharet
douglas.macharet@dcc.ufmg.br

DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

Introdução

- Tipos Abstratos de Dados (TADs)
 - Conjunto de valores
 - Conjunto de operações sobre esses valores
- O que o TAD representa e faz é mais importante do que como ele faz!
- Integridade, manutenção, reutilização, ...

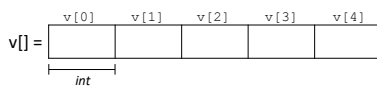
DCC

PDS 2 - Listas encadeadas e Árvores binárias

2

Introdução

- Como guardar uma coleção de elementos?
 - Arrays (vetores)
- Propriedades
 - Tamanho fixo
 - Acesso direto (índice)



DCC

PDS 2 - Listas encadeadas e Árvores binárias

3

Introdução

```
#include <iostream>

using namespace std;

int main() {
    int vetorA[5];
    vetorA[3] = 99;
    for(int i=0; i<5; i++)
        cout << vetorA[i] << "\t";
    cout << endl;

    int vetorB[5] = {};
    for(int i=0; i<5; i++)
        cout << vetorB[i] << "\t";
    cout << endl;

    double vetorC[] = {1.1, 2.2, 3.3};
    cout << vetorC[1] << endl;

    return 0;
}
```

DCC

PDS 2 - Listas encadeadas e Árvores binárias

4

Introdução

```
int main() {
    int stackMatrixB[3][4] = {
        {0, 1, 2, 3},
        {4, 5, 6, 7},
        {8, 9, 10, 11}
    };

    int rows = 3;
    int cols = 4;
    int** heapMatrix = new int*[rows];
    for (int i=0; i < rows; i++)
        heapMatrix[i] = new int[cols];

    for (int i=0; i < rows; i++)
        for (int j=0; j < cols; j++)
            heapMatrix[i][j] = (rows*i + j);

    for (int i=0; i < rows; i++)
        delete [] heapMatrix[i];
    delete [] heapMatrix;

    return 0;
}
```

<https://goo.gl/TAwL1l>
<http://www.cplusplus.com/reference/new/operator%20delete/>

DCC

PDS 2 - Listas encadeadas e Árvores binárias

5

Listas encadeadas

- Forma alternativa de guardar coleções
- Cada elemento guarda duas informações
 - O próprio valor do item
 - Referência para o elemento seguinte na cadeia



DCC

PDS 2 - Listas encadeadas e Árvores binárias

6

Listas encadeadas vs. Arrays

- Arrays
 - Acesso direto
 - Tamanho fixo e conhecido
 - $T = n \times \text{sizeof}(\text{elemento})$
- Listas encadeadas
 - Acesso sequencial
 - Tamanho variável (um elemento por vez)
 - $T = n \times \text{sizeof}(\text{elemento}) + n \times \text{sizeof}(\text{referência})$

Listas encadeadas

- A lista é constituída por células/nós
 - Conteúdo
 - Referência

```
struct Node {
    int data;
    Node* next;
};
```

Listas encadeadas

```
#include <iostream>
using namespace std;

int main() {
    Node a;
    Node b;

    a.data = 99;
    a.next = &b;

    b.data = 123;

    cout << a.data << endl;
    cout << a.next->data << endl;

    return 0;
}
```

<https://goo.gl/5oRFeR>

Listas encadeadas

- Operações
 - Criar uma nova lista (inicialização)
 - Inserir elementos
 - Retirar elementos
 - Localizar um elemento
 - Recuperar o valor de um elemento
 - Recuperar o elemento seguinte à um elemento

Listas encadeadas

- Todas as operações ficam dentro do Node?
 - Criar um TAD que efetivamente define a Lista
- Quais dados deve possuir?
 - Referência para o primeiro Node (Por quê?)
 - Head, Cabeça
 - Referência para o último Node (Por quê?)
 - Tail, Cauda

Listas encadeadas

```
List.hpp

#ifndef LIST_H
#define LIST_H

#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* next;
};

struct List {
    Node* head = nullptr;
    Node* tail = nullptr;

    void insertNode(int data);
    void removeNode(int data);
    void display();
};

#endif
```

Listas encadeadas

Inserção

- Como inserir um novo valor na lista (final)?
 - Criar o Node
 - Se a Lista estiver vazia
 - Novo Node será Cabeça e Cauda
 - Caso contrário
 - Novo Node será inserido após Cauda
 - Esse Node agora é a Cauda

DCC

PDS 2 - Listas encadeadas e Árvores binárias

13

Listas encadeadas

Inserção

```

List.cpp
#include "List.hpp"

void List::insertNode(int data) {

    Node* aux = new Node;
    aux->data = data;
    aux->next = nullptr;

    if (head == nullptr) {
        head = aux;
        tail = aux;
    } else {
        tail->next = aux;
        tail = aux;
    }
}

```

DCC

PDS 2 - Listas encadeadas e Árvores binárias

14

Listas encadeadas

Inserção

```

main.cpp
#include "List.hpp"

int main() {
    List lista;

    lista.insertNode(111);
    lista.insertNode(222);

    return 0;
}

```

DCC

PDS 2 - Listas encadeadas e Árvores binárias

15

Listas encadeadas

Inserção

```

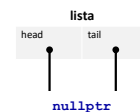
#include "List.hpp"

int main() {
    List lista;

    lista.insertNode(111);
    lista.insertNode(222);

    return 0;
}

```



DCC

PDS 2 - Listas encadeadas e Árvores binárias

16

Listas encadeadas

Inserção

```

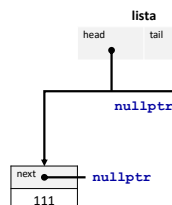
#include "List.hpp"

int main() {
    List lista;

    lista.insertNode(111);
    lista.insertNode(222);

    return 0;
}

```



DCC

PDS 2 - Listas encadeadas e Árvores binárias

17

Listas encadeadas

Inserção

```

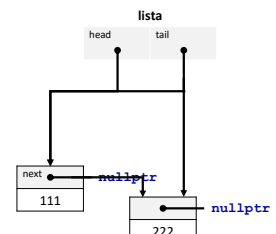
#include "List.hpp"

int main() {
    List lista;

    lista.insertNode(111);
    lista.insertNode(222);

    return 0;
}

```



DCC

PDS 2 - Listas encadeadas e Árvores binárias

18

Listas encadeadas

Remoção

- Como remover um determinado Node?
- Quantos casos?
 - Cabeça
 - Cabeça aponta para o próximo do Node removido
 - Cauda
 - Node anterior não aponta mais para ninguém
 - Node anterior vira Cauda
 - Outros
 - Node anterior aponta para o próximo do removido

DCC

PDS 2 - Listas encadeadas e Árvores binárias

19

Listas encadeadas

Remoção

```
void List::removeNode(int data) {
    Node *current = head;
    Node *previous = nullptr;

    while (current != nullptr) {
        if (current->data == data) {
            if (previous == nullptr) { // HEAD
                head = current->next;
            } else if (current->next == nullptr) { // TAIL
                previous->next = nullptr;
                tail = previous;
            } else {
                previous->next = current->next;
            }
            delete current;
            return;
        }
        previous = current;
        current = current->next;
    }
}
```

DCC

PDS 2 - Listas encadeadas e Árvores binárias

20

Listas encadeadas

Remoção

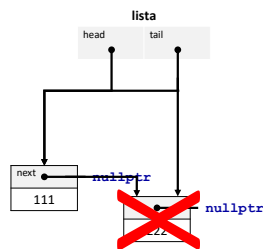
```
#include "List.hpp"

int main() {
    List lista;

    lista.insertNode(111);
    lista.insertNode(222);

    lista.removeNode(222);

    return 0;
}
```



DCC

PDS 2 - Listas encadeadas e Árvores binárias

21

Listas encadeadas

Enumeração

- Como exibir o estado atual da Lista?
 - Percorrer a lista até chegar ao último elemento
 - Como determinar o último elemento?
 - Comparar com a Cauda
 - Aponta para nullptr

DCC

PDS 2 - Listas encadeadas e Árvores binárias

22

Listas encadeadas

Enumeração

```
void List::display() {
    Node *aux = head;
    while (aux != nullptr) {
        cout << aux->data << "\t";
        aux = aux->next;
    }
    cout << endl;
}
```

DCC

PDS 2 - Listas encadeadas e Árvores binárias

23

Listas encadeadas

Enumeração

```
#include "List.hpp"

int main() {
    List lista;

    lista.insertNode(111);
    lista.insertNode(222);
    lista.display();

    return 0;
}
```

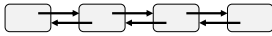
DCC

PDS 2 - Listas encadeadas e Árvores binárias

24

Listas duplamente encadeadas

- Referência também para o elemento anterior



```
struct Node {
    int data;
    Node* next;
    Node* previous;
};
```

Listas duplamente encadeadas

- Vantagens
 - Percorrida em ambas as direções
 - Inserção/Remoção diretas (fornecido ponteiro)
- Desvantagens
 - Espaço extra para a nova referência
- Qual devo utilizar?
 - Depende da sua aplicação!

Listas encadeadas

Exercício

- Listas Simples e Duplamente encadeadas
- Implemente as seguintes operações
 - Atributo que guarda o número de elementos
 - Verificar se um elemento está na lista
- Memória está sendo liberada corretamente?
 - Implementar um método que libera os Nodes

Árvores

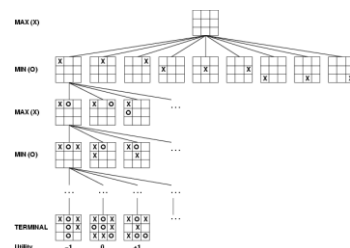
- Árvores
 - Estrutura eficiente para armazenar informação
- Adequada quando há necessidade de
 - Acesso direto e sequencial eficientes
 - Facilidade de inserção retirada de registros
 - Boa taxa de utilização de memória
 - Utilização de memória primária e secundária

Árvores

- Estrutura hierárquica
 - Muito utilizada para organização de elementos
- Cada elemento é chamado de nó
- Relação: Pai – Filhos
 - Raiz: primeiro nó da hierarquia
 - Folhas: nós que não possuem filhos
- Filho de um nó é a raiz de uma subárvore

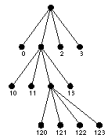
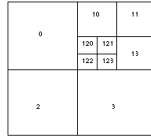
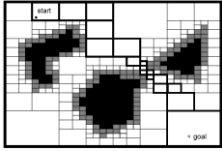
Árvores

Exemplo



Árvores

Exemplo



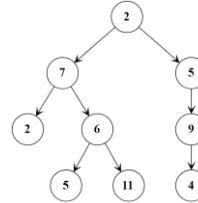
DCC

PDS 2 - Listas encadeadas e Árvores binárias

31

Árvores binárias

- Cada nó possui no máximo dois nós filhos



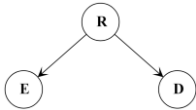
Obs: a árvore acima não impõe nenhuma ordenação em seus nós!

DCC

PDS 2 - Listas encadeadas e Árvores binárias

32

Árvores binárias



```
struct NodeT {
    int data;
    NodeT* esq;
    NodeT* dir;
};
```

DCC

PDS 2 - Listas encadeadas e Árvores binárias

33

Árvores binárias de pesquisa

- Até o momento consideramos árvores que representam os dados hierarquicamente
- Árvores binárias de pesquisa (organização)



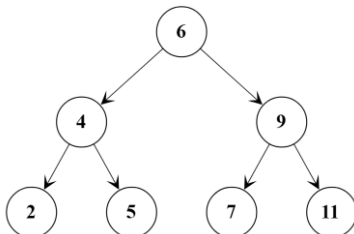
- Todos os registros com chaves menores estão dispostos na subárvore à esquerda
- Todos os registros com chaves maiores estão dispostos na subárvore à direita

DCC

PDS 2 - Listas encadeadas e Árvores binárias

34

Árvores binárias de pesquisa



DCC

PDS 2 - Listas encadeadas e Árvores binárias

35

Árvores binárias de pesquisa

Propriedades

- O nível do nó raiz é 0
- Se um nó está no nível i então a raiz de suas subárvores estão no nível $i + 1$
- A altura de um nó é o comprimento do caminho mais longo deste nó até um nó folha
- A altura de uma árvore é a altura do nó raiz

DCC

PDS 2 - Listas encadeadas e Árvores binárias

36

Árvores binárias de pesquisa

- Operações
 - Criar uma nova árvore (inicialização)
 - Inserir elementos
 - Retirar elementos
 - Listar os elementos
 - Localizar um elemento
 - Recuperar o valor de um elemento

Árvores binárias de pesquisa

```

BST.hpp
#ifndef BST_H
#define BST_H

#include <iostream>
using namespace std;

struct NodeT {
    int data;
    NodeT* esq;
    NodeT* dir;
};

struct BST {
    NodeT* root = nullptr;

    void insertNodeHelper(NodeT* root, int data);
    void insertNode(int data);
    void removeNode(int data);
    void display();
};

#endif

```

Árvores binárias de pesquisa

Inserção

- Como inserir um novo valor na árvore?
 - Primeiramente, encontrar a posição. Como?
 - Se x é menor, vá para a subárvore da esquerda
 - Se x é maior, vá para a subárvore da direita.
 - Recursivamente, até o ponteiro nulo ser atingido
 - Ou o próprio elemento, já que não aceita duplicatas!
- Criar o Node

Árvores binárias de pesquisa

Inserção

```

BST.cpp
#include "BST.hpp"

NodeT* createNode(int data) {
    NodeT* aux = new NodeT;
    aux->data = data;
    aux->esq = nullptr;
    aux->dir = nullptr;
    return aux;
}

void BST::insertNode(int data) {
    if (root != nullptr) {
        insertNodeHelper(root, data);
    } else {
        root = createNode(data);
    }
}

void BST::insertNodeHelper(NodeT* n, int data) {
    if (data < n->data) {
        if (n->esq == nullptr) {
            n->esq = createNode(data);
        } else {
            insertNodeHelper(n->esq, data);
        }
    } else if (data > n->data) {
        if (n->dir == nullptr) {
            n->dir = createNode(data);
        } else {
            insertNodeHelper(n->dir, data);
        }
    }
}

```

Árvores binárias de pesquisa

Inserção

```

main.cpp
#include "BST.hpp"

int main() {
    BST bst;

    bst.insertNode(5);
    bst.insertNode(2);
    bst.insertNode(7);
    bst.insertNode(7);

    return 0;
}

```

Árvores binárias de pesquisa

Remoção

- Como remover um determinado Node?
 - Se possuir apenas um descendente
 - Substituir pelo filho
 - Se possuir dois descendentes
 - Nó mais à direita na subárvore da esquerda
 - Nó mais à esquerda na subárvore da direita
 - Não veremos isso → Estrutura de Dados!

Árvores binárias de pesquisa

Enumeração

- Como exibir o estado atual da Árvore?
 - Existem várias ordens de caminhamento, uma bem útil é o caminhamento central (*inorder*)
 - Nós visitados de forma ordenada
 - Caminha na subárvore da esquerda
 - Visita à raiz
 - Caminha na subárvore da direita
 - Como implementar isso recursivamente?

Árvores binárias de pesquisa

Enumeração

```
void inorder(NodeT* n) {
    if (n == nullptr)
        return;

    inorder(n->esq);
    cout << n->data << " ";
    inorder(n->dir);
}

void BST::display() {
    inorder(root);
    cout << endl;
}
```

Árvores binárias de pesquisa

Exercício

- Implemente as seguintes operações
 - Variações
 - $E \leq R < D$
 - $E < R \leq D$
 - Função que verifica se um elemento está na lista