UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Algoritmos e Estruturas de Dados I

Prof. Dr. Rodrigo Hübner

Lista Encadeada

Lista Encadeada

- Encadeamento de nós que armazenam dados
  - Acesso sequencial
  - Não permite acesso aleatório
  - **Bom** para inserções/remoções em qualquer parte da lista
  - Disponível em `C++` pela `STL` como `std::forward_list`
  - `std::List` é a Lista Duplamente Encadeada

```
head -> |3| -> |8| -> |2| -> |5| -> __
```

Lista Encadeada - Nodes

```
class Node {
public:
    int key;
    Node* next;
};

class LinkedList {
private:
    Node* head;
...}
```

```
head -> |3|   ->    |8| -> |2| -> |5| -> __
        Elem  Next
```

Lista Encadeada

```
LinkedList* list1 = new LinkedList();

list1->push_front(3);
list1->push_front(8);
list1->push_front(2);
list1->push_front(5);

list1->print();

list1->pop_front();

delete list1;
```

Lista Encadeada

Característica	Lista Sequencial	Lista Encadeada
Acesso sequencial	sim	sim
Acesso aleatório	sim	não
Redimensionamento	custoso	fácil
Inserção/remoção no início	custoso	fácil
Inserção/remoção no final	fácil	fácil (c/ endereço)
Inserção/remoção no meio	custoso	fácil (c/ endereço)

Lista Encadeada - Cabeçalhos

```
class Node {
public:
  int key;
  Node* next;
};
```

Lista Encadeada - Cabeçalhos

```
class LinkedList {
private:
  Node* head;
public:
  LinkedList();
  ~LinkedList();
  void push_front(int key);
  bool pop_front();
  int get(int pos);
  void print();
  int size();
  bool empty();
  void push_back(int key);
  bool pop_back();
};
```

Lista Encadeada - Operações

```cpp
#include "linked_list.h"
#include <cstdio>

LinkedList::LinkedList() {...}
LinkedList::~LinkedList() {...}

void LinkedList::push_front(int key) {...}
bool LinkedList::pop_front() {...}
int LinkedList::get(int pos) {...}
void LinkedList::print() {...}
void LinkedList::size() {...}
bool LinkedList::empty() {...}
void LinkedList::push_back(int key) {...}
bool LinkedList::pop_back() {...}
```

Lista Encadeada - Outras operações

```
Node* find(int key);
bool insert_after(int key, Node* pos);
bool remove_after(Node* pos);
bool insert(int pos);
bool remove(int pos);
bool remove(int key);
bool insert_sorted(int key);
```

Estratégias avançadas

- Adicionar **ponteiro para cauda** da lista
  - Facilita as inserções ao final da lista
  - Acarreta em mais código de controle (inserir e remover)

```
struct LinkedList {
    Node* head; // primeiro Node
    Node* tail; // último Node
};

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

Estratégias avançadas

- Emprego de **sentinela**
  - Elemento "coringa" como primeiro da lista

```
head -> |*| -> __
```

- Simplifica a implementação, pois "head" nunca precisa ser atualizado
  - Na implementação tradicional, as operações inserir e remover requerem verificações para atualizar o ponteiro "head".
  - Com a sentinela, não há necessidade de atualizar "head"...

Estratégias avançadas

- Lista com um Node

```
head -> |*| -> |8| -> __
```

- Lista habitual

```
head -> |*| -> |8| -> |2| -> |5| -> __
```

Próxima aula

- Listas Duplamente Encadeadas e Circulares