Técnicas e Análise de Algoritmos

Listas, Pilhas e Filas

Professor: **Jeremias Moreira Gomes**
E-mail: jeremias.gomes@idp.edu.br

Introdução

Listas Encadeadas

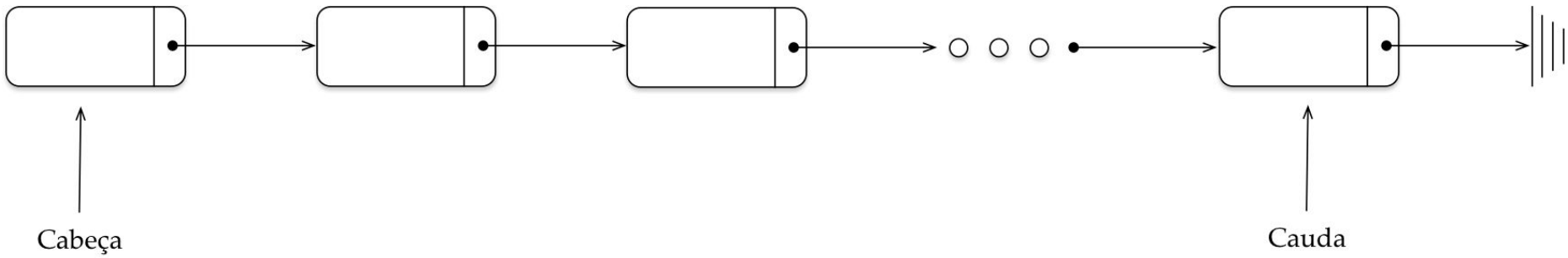
Lista Encadeada

- Uma lista encadeada, ou simplesmente lista, é uma estrutura composta por nós, onde cada nó armazenada uma informação e um ponteiro para o próximo nó da lista
- Conhecido o primeiro elemento da lista (head), é possível acessar todos os demais elementos
- Uma lista é uma estrutura de dados linear, devido a travessia sequencial e ordenada de seus elementos

Lista Encadeada

- As listas são uma alternativa aos vetores: a vantagem do acesso aleatório imediato dos vetores é substituída pela inserção e remoção eficientes
- Por conta da estrutura dos nós, o acesso aleatório em listas encadeadas tem complexidade O(N)

Lista Encadeada



Lista Encadeada - Inserção no Início

- A inserção no início (push_front()) de uma lista encadeada tem complexidade O(1)
- O primeiro passo da inserção é criar um novo nó
- Em seguida, deve ser preenchido o campo info
- O membro next deve apontar então para o primeiro elemento da lista (head)

Lista Encadeada - Inserção no Início

- Por fim, o membro head deve apontar para o novo elemento
- Caso atípico: caso o membro head esteja nulo no início da inserção, o membro tail também deve apontar para o novo elemento
- Caso a classe tenha o membro size, este deve ser incrementado na inserção

Lista Encadeada - Inserção no Final

- A inserção no final (push_back()) de uma lista encadeada tem complexidade O(1), desde que a classe tenha o membro tail
- O primeiro passo da inserção é criar um novo nó
- Em seguida, deve ser preenchido o campo info
- O membro next de tail deve apontar então para o novo elemento da lista

Lista Encadeada - Inserção no Final

- Por fim, o membro tail deve apontar para o novo elemento
- Caso de Borda:
  - Caso o membro tail esteja nulo no início da inserção, o membro head também deve apontar para o novo elemento
- Caso a classe tenha o membro size, este deve ser incrementado na inserção

Inserção em Posição Arbitrária

- A inserção em posição arbitrária tem complexidade O(N), onde N é o número de elementos da lista
- Primeiramente é necessário localizar a posição da inserção
- Além disso, é preciso identificar, se existir, o elemento que sucede o elemento que ocupa a posição de inserção
- O membro next do novo elemento deve apontar para o elemento que ocupa a posição de inserção

Inserção em Posição Arbitrária

- O membro next do nó que antecedia o elemento da posição de inserção deve apontar para o novo nó
- É preciso tomar cuidado com vários casos de borda:
  - Lista vazia
  - Apenas um elemento na lista
  - Inserção na primeira posição
  - Posição de inserção inválida

Remoção no Início

- A remoção de um elemento do início de uma lista (pop_front()) tem complexidade O(1)
- O primeiro passo da remoção é armazenar o membro head em uma variável temporária
- Em seguida, o membro head deve apontar para o próximo elemento da lista
- Por fim, o ponteiro armazenado na variável temporária é deletado
- O membro size deve ser decrementado, se existir

Remoção no Fim

- Mesmo com o membro tail, a remoção do último elemento de uma lista encadeada (pop_back()) tem complexidade O(N), onde N é o número de nós da lista
- Isto acontece porque é preciso localizar o elemento que antecede o último elemento (prev), processo que tem complexidade linear
- Localizado o elemento prev, a remoção é semelhante à remoção do início: o elemento tail é deletado e tail aponta para prev
- Por fim, o membro next de prev deve se tornar nulo
- O membro size deve ser decrementado, se existir

Remoção em Posição Arbitrária

- A remoção em posição arbitrária também tem complexidade O(N)
- Esta é uma rotina de implementação complexa, dado o grande número de casos atípicos
- É preciso localizar o elemento que antecede o elemento a ser removido, como no caso da remoção no final
- Os membros head e tail deve ser devidamente tratados e atualizados, quando for o caso
- O membro next de prev também precisa ser atualizado corretamente

Lista Encadeada em C++

- A linguagem C++ oferece uma implementação de listas simplesmente encadeadas: o contêiner forward_list
- Por padrão a implementação é tão eficiente quanto a implementação de uma lista simplesmente encadeada em C
  - Por este motivo, é o único dentre os contêiners da STL que não tem um método size()
- As inserções e remoções constantes devem ser feitas através dos métodos push_front() e pop_front()

Lista Encadeada em C++

```cpp
int main()
{
    forward_list<int> L;
    cout << "L.empty() = " << L.empty() << endl;
    L.push_front(7);
    for (int i = 0; i < 5; i++) L.push_front(i);

    L.pop_front();
    int size = distance(L.begin(), L.end()); // O(n)
    cout << "L.size() = " << size << endl;

    int ultimo;
    for (auto v: L) ultimo = v;

    cout << "ultimo = " << ultimo << endl;
    return 0;
}
```

Lista Duplamente Encadeada

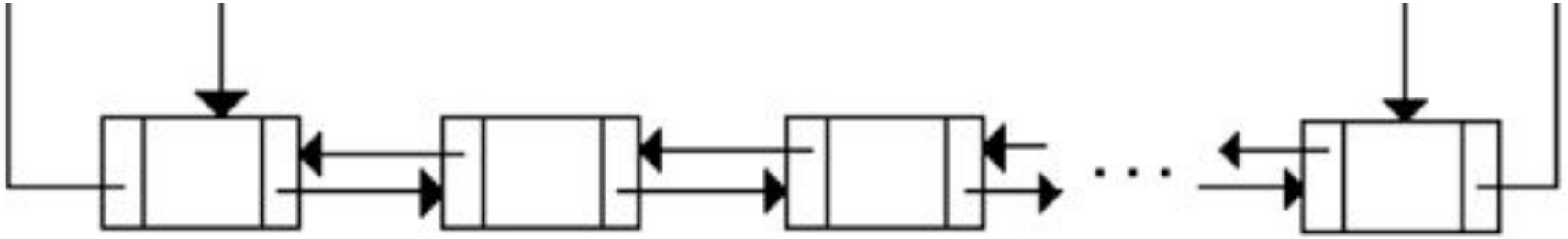
Lista Duplamente Encadeada

- Uma lista duplamente encadeada é uma estrutura composta por nós, onde cada nó armazenada uma informação, um ponteiro para o antecessor e um ponteiro para o próximo nó da lista
- A partir de qualquer elemento da lista é possível acessar todos os demais elementos
- Esta maior flexibilidade em relação às listas (simplesmente) encadeadas tem seu custo: cada nó precisa de um ponteiro extra, aumentando o uso de memória

Lista Duplamente Encadeada

- É uma estrutura de dados linear, devido a travessia sequencial e ordenada de seus elementos
- Por conta da estrutura dos nós, o acesso aleatório em listas duplamente encadeadas tem complexidade O(N)

Lista Duplamente Encadeada



Lista Duplamente Encadeada

- Pode ser implementada como uma struct em C ou uma classe em C++
- Deve ter um membro para o primeiro (head) e para o último (tail) elemento da lista
- Cada nó deve ter, no mínimo, três membros: um para armazenar as informações (info), um para representar o ponteito para o próximo nó (next) e um para representar o ponteiro para o nó anterior (prev)

Lista Duplamente Encadeada

- O primeiro elemento da lista tem membro prev nulo
- O último elemento da lista tem membro next nulo
- Esta configuração permite inserções e remoções, no íncio e no final, com complexidade O(1)

Lista Duplamente Encadeada - Inserção

- A inserção no início funciona de maneira análoga a lista encadeada (O(1))
  - Membro prev é nulo
- A inserção no final também funciona de maneira análoga (O(1))
  - Membro next é nulo
- A inserção em posição arbitrária depende de
  - Conhecendo-se a posição: O(1)
  - Não conhecendo: O(n)

Lista Duplamente Encadeada - Remoção

- A remoção no início (pop_front) funciona de maneira análoga a lista encadeada (O(1))
- A remoção no final (pop_back), diferente da lista encadeada, é constante (O(1))
- A remoção em posição arbitrária depende de
  - Conhecendo-se a posição: O(1)
  - Não conhecendo: O(n)

Lista Duplamente Encadeada em C++

- A linguagem C++ oferece uma implementação de listas duplamente encadeadas: o contêiner list
- Iteradores desse container não bidirecionais
- Possui o método size()
- As inserções e remoções constantes devem ser feitas através dos métodos push_front() e pop_front()
- Possui também o método insert(), que insere um elemento logo após o iterador indicado em complexidade constante (O(1))

Lista Duplamente Encadeada em C++

```
int main()
{
    list<int> L;
    L.push_back(7);
    L.push_front(12);

    for (int i = 0; i < 5; i++) L.push_back(i);

    auto it = L.begin();
    it++; it++;
    L.insert(it, 100);
    cout << "L.size() = " << L.size() << endl;

    for (auto v: L) cout << v << " ";
    return 0;
}
```

Pilhas

Pilhas

- Uma pilha é um tipo de dados abstrato cuja interface define que o último elemento inserido na pilha é o primeiro a ser removido
  - Esta estratégia de inserção e remoção é denominada LIFO – Last In, First Out
- De acordo com sua interface, uma pilha não permite acesso aleatório ao seus elementos
  - Apenas o elemento do topo da pilha pode ser acessado
- As operações de inserção e remoção tem complexidade O(1)

Pilhas

Método	Complexidade	Descrição
`clear(P)`	$O(N)$	Esvazia a pilha `P`, removendo todos os seus elementos
`empty(P)`	$O(1)$	Verifica se a pilha `P` está vazia ou não
`push(P, x)`	$O(1)$	Insere o elemento `x` no topo da pilha `P`
`pop(P)`	$O(1)$	Remove o elemento que está no topo da pilha `P`
`top(P)`	$O(1)$	Retorna o elemento que está no topo da pilha `P`
`size(P)`	$O(1)$	Retorna o número de elementos armazenados na pilha `P`

Pilhas em C++

- A biblioteca padrão de templates (STL) do C++ provê o contêiner stack, que implementa uma pilha
- Tanto o tipo de dado a ser armazenado quanto o contêiner que será usado na composição são parametrizáveis
- Por padrão, o contêiner utilizado é um deque (double-ended queue), mas os contêineres vector e list são igualmente válidos

Pilhas em C++

```cpp
int main()
{
    stack<pair<int, string>> pilha;

    pilha.push({1, "um"});
    pilha.push({2, "dois"});

    cout << pilha.top().first << endl;
    cout << pilha.top().second << endl;

    pilha.pop();

    cout << pilha.size() << "\n";
    return 0;
}
```

Aplicações para Pilhas

- Identificação de Delimitadores
  - Delimitadores são caracteres de marcação que delimitam um conjunto de informações
  - Em C++, os caracteres (), [], {} e os pares de caracteres /*, */ são delimitadores
  - Pilhas podem ser usadas para verificar se os delimitadores foram abertos e fechados corretamente
    - Exemplo 01: as expressões (()), [()()], ()[] são válidas
    - Exemplo 02: as expressões [),)(, ()], [[][] são inválidas

Filas

Filas

- Uma fila é um tipo de dados abstrato cuja interface define que o primeiro elemento inserido na pilha é o primeiro a ser removido
- Esta estratégia de inserção e remoção é denominada FIFO – First In, First Out
- De acordo com sua interface, uma fila não permite acesso aleatório ao seus elementos
  - Apenas os elementos dos extremos da fila podem ser acessados
- Operações de inserção e remoção devem ter complexidade O(1)

Filas

Método	Complexidade	Descrição
`clear(F)`	$O(N)$	Esvazia a fila `F`, removendo todos os seus elementos
`empty(F)`	$O(1)$	Verifica se a fila `F` está vazia ou não
`push(F, x)`	$O(1)$	Insere o elemento `x` no final da fila `F`
`pop(F)`	$O(1)$	Remove o elemento que está no início da fila `F`
`front(F)`	$O(1)$	Retorna o elemento que está no início da fila `F`
`size(F)`	$O(1)$	Retorna o número de elementos armazenados na fila `F`

Filas em C++

- A STL do C++ oferece uma implementação de fila: a classe queue
- Assim como no caso das pilhas, o contêiner usado na composição é, por padrão, a deque
  - Este contêiner pode ser substituído por qualquer contêiner que contenha os métodos pop_front(), push_back() e size(), dentre outros
  - Existe um método swap para trocar duas pilhas
    - Esse método também existe para pilhas

Filas em C++

```cpp
int main()
{
    queue<double> fila;

    fila.push(1.0);
    fila.push(2.0);
    fila.push(3.0);

    cout << fila.front() << endl;

    fila.pop();

    cout << fila.size() << "\n";

    return 0;
}
```

Outras Estruturas que Valem Mencionar

- **Vector**: já foi falado
  - Existe uma estrutura em C++ chamada **array**, que é similar a um vector, porém não redimensionável (tamanho fixo)
- **Deque**:
  - É uma fila com duas cabeças
  - Complexidade similar à classe vector, porém possui tempo constante (amortizado) na inserção e remoção dos extremos
- **Array:**
  - Vetor não redimensionável

Outras Estruturas que Valem Mencionar

Estrutura	Complexidade de Tempo								Complexidade de Espaço
	Caso Médio				Pior Caso				Pior Caso
	Acesso	Busca	Inserção	Remoção	Acesso	Busca	Inserção	Remoção	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)

Conclusão