TAD lista posicional

Rafael Alves da Costa

CIÊNCIA DE DADOS
FATEC Santana de Parnaíba

Aula 7 - Estruturas de Dados

09/2025

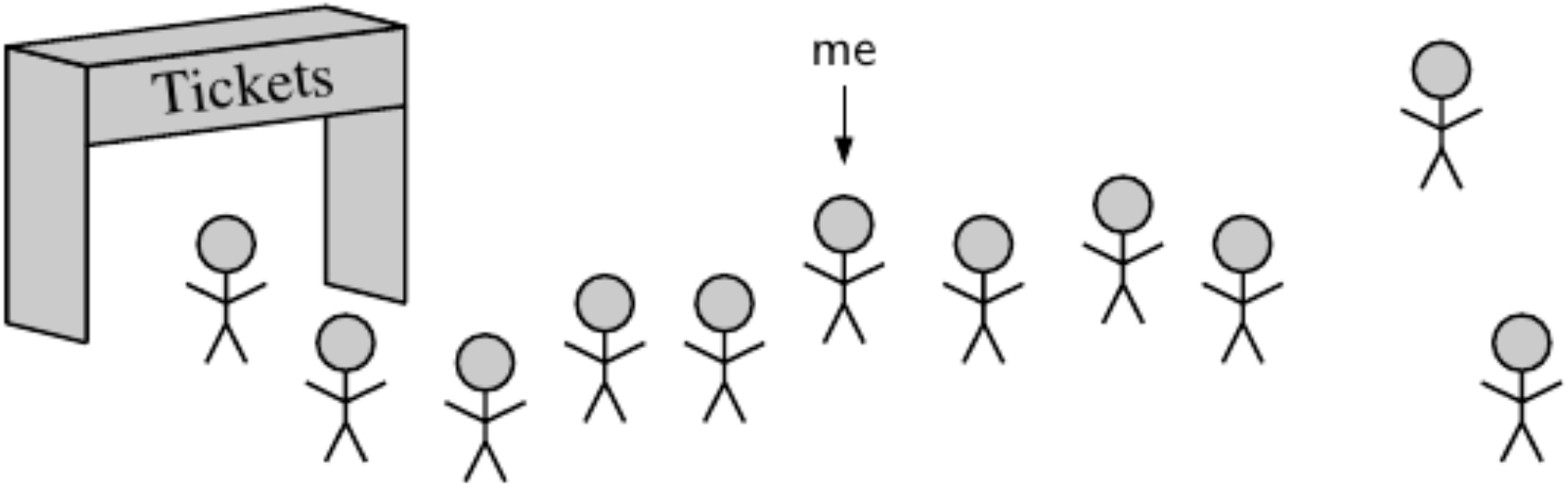
Sumário

1 TAD Lista Posicional

TAD Lista Posicional

O TAD Lista Posicional

Os tipos abstratos de dados (TADs) considerados até agora, como pilhas, filas, deques, permitem operações de atualização apenas em uma das extremidades de uma sequência. Porém, precisamos de uma abstração mais geral. Por exemplo, a fila é um bom modelo para clientes esperando atendimento ou para fãs comprando ingressos, mas é limitada em situações onde um cliente desiste antes de ser atendido ou alguém deixa um amigo entrar na fila.



Problemas com Índices Numéricos

Para sequências baseadas em array (como uma lista Python), índices inteiros são adequados para descrever a localização de um elemento. Porém, em listas encadeadas, índices numéricos não são eficientes porque requerem a travessia da lista de forma incremental. Além disso, índices não são uma boa abstração para descrever posições locais, pois mudam com inserções ou exclusões anteriores na sequência, dificultando a localização precisa de um elemento.

Abstração de Posições

Preferimos uma abstração onde há outro meio de descrever uma posição. Por exemplo, em um documento de texto, um processador de texto usa o cursor para descrever uma posição sem um índice inteiro, permitindo operações como “deletar o caractere no cursor” ou “inserir um novo caractere após o cursor”. Esta abstração é útil em aplicações como documentos, onde posições inerentes podem ser referenciadas sem depender de índices que mudam conforme o documento evolui.

Exemplo - Animação

Abstração de Posições

Preferimos uma abstração onde há outro meio de descrever uma posição. Por exemplo, em um documento de texto, um processador de texto usa o cursor para descrever uma posição sem um índice inteiro, permitindo operações como “deletar o caractere no cursor” ou “inserir um novo caractere após o cursor”. Esta abstração é útil em aplicações como documentos, onde posições inerentes podem ser referenciadas sem depender de índices que mudam conforme o documento evolui.

Uma Referência de Nó como uma Posição?

Um dos grandes benefícios de uma estrutura de lista encadeada é que é possível realizar inserções e exclusões em tempo O(1) em posições arbitrárias da lista, desde que tenhamos uma referência a um nó relevante da lista. É, portanto, muito tentador desenvolver uma TAD em que uma referência de nó sirva como o mecanismo para descrever uma posição.

Problemas com Referências de Nós

De fato, nossa classe `DoublyLinkedBase` (apresentada no notebook da aula 5 e 6) tem métodos `insert_between` e `delete_node` que aceitam referências de nós como parâmetros. No entanto, esse uso direto de nós violaria os princípios de design orientado a objetos de abstração e encapsulamento.

Simplicidade para os Usuários

Será mais simples para os usuários de nossa estrutura de dados se eles não forem incomodados com detalhes desnecessários de nossa implementação, como a manipulação de baixo nível de nós ou nossa dependência do uso de nós sentinelas. Note que, para usar o método `insert_between` da nossa classe `DoublyLinkedBase` para adicionar um nó no início de uma sequência, o nó sentinela `header` deve ser enviado como um parâmetro.

Robustez da Estrutura de Dados

Podemos fornecer uma estrutura de dados mais robusta se não permitirmos que os usuários acessem ou manipulem diretamente os nós. Dessa forma, garantimos que os usuários não possam invalidar a consistência de uma lista ao gerenciar incorretamente a ligação de nós.

Flexibilidade e Abstração

Ao encapsular melhor os detalhes internos de nossa implementação, temos maior flexibilidade para redesenhar a estrutura de dados e melhorar seu desempenho. De fato, com uma abstração bem projetada, podemos fornecer uma noção de uma posição não numérica, mesmo que usando uma sequência baseada em array.

TAD Lista Posicional - Implementação

Para possibilitar uma abstração geral de uma sequência de elementos com a capacidade de identificar a localização de um elemento, definimos uma TAD de lista posicional, bem como uma abstração de tipo de dado de posição mais simples para descrever uma localização dentro de uma lista.

Uma posição atua como um marcador ou token dentro da lista posicional mais ampla. Uma posição **p** não é afetada por mudanças em outras partes da lista; a única maneira de uma posição se tornar inválida é se um comando explícito for emitido para excluí-la.

TAD Lista Posicional - Implementação (Cont.)

- Uma instância de posição é um objeto simples, suportando apenas o seguinte método:
  - p.element(): Retorna o elemento armazenado na posição p.
  - No contexto da TAD de lista posicional, posições servem como parâmetros para alguns métodos e como valores de retorno de outros métodos. Ao descrever os comportamentos de uma lista posicional, começamos apresentando os métodos de acesso suportados por uma lista L:
    - L.first(): Retorna a posição do primeiro elemento de L, ou None se L estiver vazia.
    - L.last(): Retorna a posição do último elemento de L, ou None se L estiver vazia.
    - L.before(p): Retorna a posição de L imediatamente antes da posição p, ou None se p for a primeira posição.
    - L.after(p): Retorna a posição de L imediatamente após a posição p, ou None se p for a última posição.

TAD Lista Posicional - Implementação (Cont.)

- L.is_empty(): Retorna True se a lista L não contiver nenhum elemento.
- len(L): Retorna o número de elementos na lista.
- iter(L): Retorna um iterador para frente para os elementos da lista. Pesquise sobre iteradores em Python!!!.
  A TAD de lista posicional também inclui os seguintes métodos de atualização:
- L.add_first(e): Insere um novo elemento e na frente de L, retornando a posição do novo elemento.
- L.add_last(e): Insere um novo elemento e no final de L, retornando a posição do novo elemento.

TAD Lista Posicional - Implementação (Cont.)

- L.add_before(p, e): Insere um novo elemento e logo antes da posição p em L, retornando a posição do novo elemento.
- L.add_after(p, e): Insere um novo elemento e logo após a posição p em L, retornando a posição do novo elemento.
- L.replace(p, e): Substitui o elemento na posição p pelo elemento e, retornando o elemento anteriormente na posição p.
- L.delete(p): Remove e retorna o elemento na posição p em L, invalidando a posição.

TAD Lista Posicional - Implementação (Cont.)

- L.add_before(p, e): Insere um novo elemento e logo antes da posição p em L, retornando a posição do novo elemento.
- L.add_after(p, e): Insere um novo elemento e logo após a posição p em L, retornando a posição do novo elemento.
- L.replace(p, e): Substitui o elemento na posição p pelo elemento e, retornando o elemento anteriormente na posição p.
- L.delete(p): Remove e retorna o elemento na posição p em L, invalidando a posição.

- **Vamos verificar a implementação no notebook!!!**

Considerações

Considerações

- **Recordando!!!**
  - TDA Lista Posicional.
  - Exercícios.

Considerações

- **Recordando!!!**
  - TDA Lista Posicional.
  - Exercícios.
- **Referências utilizadas!!!**
  - Cormen, T. H. et al. Algoritmos: Teoria e prática, 3a edição. Elsevier, 2012.
  - Goodrich, M. T., Tamassia, R., Goldwasser, M. H. (2013). Data structures and algorithms in Python.