# Fila de Prioridade

Rafael Alves da Costa

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
FATEC Carapicuíba

Aula 10 - Estrutura de Dados

10/2025

# Sumário

# Fila de Prioridade

# Fila de Prioridade: História!

- **Apollo 11** "O trabalho de Margaret Hamilton evitou que o pouso na lua da Apollo 11 fosse abortado. Quando faltavam três minutos para a Apollo 11 pousar na lua, vários alarmes do módulo lunar começaram a tocar. O computador ficou sobrecarregado com atividades do radar de aproximação, desnecessárias para o pouso. No entanto, devido à arquitetura robusta do software, **o sistema continuou funcionando de maneira que as atividades prioritárias interrompessem as menos prioritárias**. Mas ela sabia, por ter escrito o código do computador, que ele seria capaz de realizar o pouso, pois foi programado para desconsiderar as tarefas desnecessárias no momento da alunissagem." - Referências

# Fila de Prioridade

- Na disciplina de **Estrutura de Dados** foi apresentado o TAD (Tipo Abstrato de Dados) **fila** como uma coleção de objetos que são adicionados e removidos de acordo com o **princípio** do primeiro a entrar, primeiro a sair (FIFO).
- Na prática, existem muitas aplicações em que uma estrutura semelhante a uma fila é usada para gerenciar objetos que precisam ser processados de alguma forma, mas para as quais a política **FIFO** não é **suficiente**. Considere, **por exemplo**, um **centro de controle de tráfego aéreo** que deve decidir qual voo autorizar para pouso entre muitos que estão se aproximando do aeroporto. Essa escolha pode ser influenciada por fatores como a distância de cada avião à pista, o tempo gasto esperando em um padrão de espera ou a quantidade de combustível restante. É improvável que as decisões de pouso sejam baseadas puramente em uma política FIFO.

# Fila de Prioridade

- Há **outras situações** em que uma **política** de **"primeiro a chegar, primeiro a ser atendido"** pode parecer razoável, mas para as quais **outras prioridades entram em jogo**. Usando outra analogia com companhias aéreas, suponha que um determinado voo esteja totalmente reservado uma hora antes da partida. Devido à possibilidade de cancelamentos, a companhia aérea mantém uma fila de passageiros em espera, na esperança de conseguir um assento. Embora a prioridade de um passageiro em espera seja influenciada pelo horário do check-in desse passageiro, outras considerações incluem a tarifa paga e o status de passageiro frequente. Portanto, pode ser que um assento disponível seja dado a um passageiro que chegou mais tarde do que outro, se tal passageiro tiver uma prioridade melhor atribuída pelo agente da companhia aérea.

# O TAD Fila de Prioridade

- Formalmente, modelamos um elemento e sua prioridade como um par chave-valor.
- Definimos a TAD fila de prioridade para suportar os seguintes métodos para uma fila de prioridade P:
- **P.add(k, v):** Insere um item com chave k e valor v na fila de prioridade P.
- **P.min():** Retorna uma tupla, (k, v), representando a chave e o valor de um item na fila de prioridade P com chave mínima (mas não remove o item); ocorre um erro se a fila de prioridade estiver vazia.
- **P.remove_min():** Remove um item com chave mínima da fila de prioridade P e retorna uma tupla, (k, v), representando a chave e o valor do item removido; ocorre um erro se a fila de prioridade estiver vazia.
- **P.is_empty():** Retorna True se a fila de prioridade P não contiver nenhum item.

# O TAD Fila de Prioridade

- Uma fila de prioridade pode ter múltiplas entradas com chaves equivalentes, caso em que os métodos **min** e **remove_min** podem relatar uma escolha arbitrária de item com chave mínima.
- Os valores podem ser de qualquer tipo de objeto.
- Em nosso modelo inicial para uma fila de prioridade, assumimos que a chave de um elemento permanece fixa uma vez que ele foi adicionado a uma fila de prioridade.

# O TAD Fila de Prioridade

- A tabela a seguir mostra uma série de operações e seus efeitos em uma fila de prioridade P inicialmente vazia. A coluna "Fila de Prioridade"é um tanto enganosa, pois mostra as entradas como tuplas e ordenadas por chave. Tal representação interna não é exigida de uma fila de prioridade.

| Operation | Return Value | Priority Queue |
|---|---|---|
| P.add(5,A) | | {(5,A)} |
| P.add(9,C) | | {(5,A), (9,C)} |
| P.add(3,B) | | {(3,B), (5,A), (9,C)} |
| P.add(7,D) | | {(3,B), (5,A), (7,D), (9,C)} |
| P.min( ) | (3,B) | {(3,B), (5,A), (7,D), (9,C)} |
| P.remove_min( ) | (3,B) | {(5,A), (7,D), (9,C)} |
| P.remove_min( ) | (5,A) | {(7,D), (9,C)} |
| len(P) | 2 | {(7,D), (9,C)} |
| P.remove_min( ) | (7,D) | {(9,C)} |
| P.remove_min( ) | (9,C) | { } |
| P.is_empty( ) | True | { } |
| P.remove_min( ) | "error" | { } |

# Implementação da Fila de Prioridade

- Para implementar uma fila de prioridade em Python, usa-se a composição para armazenar internamente os itens como pares consistindo de uma chave k e um valor v.
  - Classe PriorityQueueBase: Esta classe atua como uma classe base abstrata para a fila de prioridade. Ela define a estrutura básica e os métodos necessários para manipular itens na fila de prioridade.
  - Classe Item: Esta classe aninhada dentro de PriorityQueueBase armazena os itens da fila de prioridade como pares de chave-valor.
  - Métodos da Classe Item:
    - __init__(self, k, v): Inicializa um item com uma chave k e um valor v.
    - __lt__(self, other): Define a comparação entre itens baseada em suas chaves.
    - Método is_empty(self): Verifica se a fila de prioridade está vazia.

## Implementação com Lista Não Ordenada - Parte 1

- Nossa primeira implementação concreta de uma fila de prioridade armazena entradas em uma lista não ordenada.
- A classe `UnsortedPriorityQueue` herda de `PriorityQueueBase`.
- Pares chave-valor são representados como compostos, usando instâncias da classe `Item`.
- Esses itens são armazenados em uma `PositionalList`, implementada com uma lista duplamente ligada, onde todas as operações executam em tempo O(1).
- Começamos com uma lista vazia ao construir uma nova fila de prioridade.

## Implementação com Lista Não Ordenada - Parte 2

- O método `len` da fila de prioridade simplesmente retorna o comprimento da lista interna de dados.
- Cada vez que um par chave-valor é adicionado via o método `add`, criamos um novo `Item` e o adicionamos ao final da lista, em tempo O(1).
- Quando `min` ou `remove_min` é chamado, devemos localizar o item com a chave mínima.
- Definimos uma utilidade não pública `find_min` que retorna a posição de um item com a chave mínima.
- O método `remove_min` invoca o método `delete` na lista posicional.
- O método `min` usa a posição para recuperar o item ao preparar uma tupla chave-valor para retornar.
- Devido ao loop para encontrar a chave mínima, os métodos `min` e `remove_min` executam em tempo O(n).

## Implementação com Lista Não Ordenada - Parte 2

- O método `len` da fila de prioridade simplesmente retorna o comprimento da lista interna de dados.
- Cada vez que um par chave-valor é adicionado via o método `add`, criamos um novo `Item` e o adicionamos ao final da lista, em tempo O(1).
- Quando `min` ou `remove_min` é chamado, devemos localizar o item com a chave mínima.
- Definimos uma utilidade não pública `find_min` que retorna a posição de um item com a chave mínima.
- O método `remove_min` invoca o método `delete` na lista posicional.
- O método `min` usa a posição para recuperar o item ao preparar uma tupla chave-valor para retornar.
- Devido ao loop para encontrar a chave mínima, os métodos `min` e `remove_min` executam em tempo O(n).
- **Vamos ver a implementação no notebook!!!**

# Tempos de Execução no Pior Caso

- Tempos de execução no pior caso dos métodos de uma fila de prioridade de tamanho n, realizada por meio de uma lista duplamente ligada **não ordenada**.
- A exigência de espaço é O(n).

| Operation | Running Time |
|---|---|
| len | $O(1)$ |
| is_empty | $O(1)$ |
| add | $O(1)$ |
| min | $O(n)$ |
| remove_min | $O(n)$ |

# Implementação com Lista Ordenada - Parte 1

- Uma implementação alternativa de uma fila de prioridade usa uma lista posicional, mantendo as entradas ordenadas por chaves não decrescentes.
- Isso garante que o primeiro elemento da lista seja uma entrada com a menor chave.
- A classe `SortedPriorityQueue` facilita a implementação dos métodos `min` e `remove_min`.
- Utilizamos o método `first` da lista posicional para encontrar a posição do primeiro item, e o método `delete` para remover a entrada da lista.
- Supondo que a lista seja implementada com uma lista duplamente ligada, as operações `min` e `remove_min` levam tempo O(1).

# Implementação com Lista Ordenada - Parte 2

- O método add requer que percorramos a lista para encontrar a posição apropriada para inserir o novo item.
- Nossa implementação começa no final da lista e percorre para trás até que a nova chave seja menor que a de um item existente; no pior caso, avança até chegar ao início da lista.
- Portanto, o método add leva tempo O(n) no pior caso, onde n é o número de entradas na fila de prioridade no momento da execução do método.
- Em resumo, ao usar uma lista ordenada para implementar uma fila de prioridade, a inserção ocorre em tempo linear, enquanto encontrar e remover o mínimo pode ser feito em tempo constante.

# Implementação com Lista Ordenada - Parte 2

- O método add requer que percorramos a lista para encontrar a posição apropriada para inserir o novo item.
- Nossa implementação começa no final da lista e percorre para trás até que a nova chave seja menor que a de um item existente; no pior caso, avança até chegar ao início da lista.
- Portanto, o método add leva tempo O(n) no pior caso, onde n é o número de entradas na fila de prioridade no momento da execução do método.
- Em resumo, ao usar uma lista ordenada para implementar uma fila de prioridade, a inserção ocorre em tempo linear, enquanto encontrar e remover o mínimo pode ser feito em tempo constante.
- **Vamos ver a implementação no notebook!!!**

# Tempos de Execução no Pior Caso

| Operação | Lista Não Ordenada | Lista Ordenada |
|---|---|---|
| len | O(1) | O(1) |
| is empty | O(1) | O(1) |
| add | O(1) | O(n) |
| min | O(n) | O(1) |
| remove min | O(n) | O(1) |

Tempos de execução no pior caso dos métodos de uma fila de prioridade de tamanho n, realizada por meio de uma lista não ordenada ou ordenada. Suponhamos que a lista seja implementada por uma lista duplamente ligada. A exigência de espaço é O(n).

## Comparação: Lista Ordenada vs. Lista Não Ordenada

- A Tabela (do slide anterior) compara os tempos de execução dos métodos de uma fila de prioridade realizada por meio de uma lista ordenada e uma lista não ordenada, respectivamente.
- Vemos um interessante trade-off ao usar uma lista para implementar a TAD fila de prioridade.
- Uma lista não ordenada suporta inserções rápidas, mas consultas e remoções lentas.
- Uma lista ordenada permite consultas e remoções rápidas, mas inserções lentas.

# Considerações

# Considerações

- **Recordando!!!**
  - Fila de prioridade.
  - Fila de prioridade Ordenada.
  - Fila de prioridade Não Ordenada.
  - Exercícios.

# Considerações

- **Recordando!!!**
  - Fila de prioridade.
  - Fila de prioridade Ordenada.
  - Fila de prioridade Não Ordenada.
  - Exercícios.
- **Referências utilizadas!!!**

  - Cormen, T. H. et al. Algoritmos: Teoria e prática, 3a edição. Elsevier, 2012.
  - Levitin, A. Introduction to the Design and Analysis of Algorithms, 2007.
  - Goodrich, M. T., Tamassia, R., Goldwasser, M. H. (2013). Data structures and algorithms in Python.
  - Goodrich, M. T., Tamassia, R., Goldwasser, M. H. (2014). Data Structures and Algorithms in Java.