TAD Fila de Prioridade

- Uma fila de prioridade armazena coleções de itens
- Um item é um par (chave, elemento)
- Principais métodos do TAD Fila De Prioridade
  - insert(k, o)
    Insere um item com chave k e elemento o
  - removeMin()
    Remove e retorna o item com a menor chave
- Métodos adicionais
  - min(k, o)
    Retorna, mas não remove, o item com a menor chave
  - size(), isEmpty()
- Aplicações:
  - Esperas de vôos
  - Leilões
  - Controle de estoque

Relação de ordem total

- Chaves em uma fila de prioridade podem ser objetos quaisquer sobre os quais uma ordem é definida
- Dois itens distintos podem ter a mesma chave

- Conceito matemático da relação de ordem total $\leq$
  - Propriedade reflexiva: $x \leq x$
  - Propriedade assimétrica: $x \leq y \wedge y \leq x \Rightarrow x = y$
  - Propriedade transitiva: $x \leq y \wedge y \leq z \Rightarrow x \leq z$

TAD Item

- Um **item** em uma fila de prioridade é simplesmente um par chave-valor
- Fila de prioridade armazenam itens para permitir inserção e remoção eficiente baseada em chaves
- Métodos:
  - key(): retorna a chave do item
  - value(): retorna o valor associado com o item

- Interface java:

```
/**
  * Interface para um item com
  * par chave-valor
**/
public interface  Entry  {
    public  Object key();
    public  Object value();
}
```

TAD Comparador

- Um comparador encapsula a ação de comparar dois objetos de acordo com uma relação de ordem total dada
- Uma fila de prioridade genérica usa um comparador
- O comprador é externo às chaves sendo comparadas
- Quando a fila de prioridade precisa comparar duas chaves, ela usa seu comparador

- Método primário do TAD Comparador:
  - compare(x, y): retorna um inteiro *i* tal que:
    - *i<0* se *a<b*,
    - *i=0* se *a=b*,
    - *i>0* se *a>b*,
    - um erro ocorre se *a* e *b* não podem ser comparados

Exemplo de Comparador

- Comparação de pontos 2D

```
/** Compara pontos 2D sobre o padrão de
    ordem lexicográfica */
public class Lexicographic implements
    Comparator {
  int xa, ya, xb, yb;
  public int compare(Object a, Object b)
   throws ClassCastException {
    xa = ((Point2D) a).getX();
    ya = ((Point2D) a).getY();
    xb = ((Point2D) b).getX();
    yb = ((Point2D) b).getY();
    if (xa != xb)
        return (xb - xa);
    else
        return (yb - ya);
  }
}
```

- objetos pontos:

```
/** Classe que representa pontos em um
    plano com coordenadas inteiras */
public class Point2D          {
  protected int xc, yc; // coordenadas
  public Point2D(int x, int y) {
    xc = x;
    yc = y;
  }
  public int getX() {
        return xc;
  }
  public int getY() {
        return yc;
  }
}
```

Ordenação com fila de prioridade

- Podemos usar uma fila de prioridade para ordenar um conjunto de elementos comparáveis
  1. Insira os elementos, um por um, através da operação insert(e, e)
  2. Remova os elementos ordenados com uma série de operações
  3. removeMin()
- O tempo de execução deste método depende da implementação da fila de prioridade

Algoritmo *PQ-Sort*(*S*, *C*)
  Entrada Sequencia *S*, Comparador *C* para os elementos de *S*
  Saída Sequencia *S* ordenada crescentemente de acordo com *C*
  P ← Fila de prioridade com comparador *C*
  enquanto ¬*S.isEmpty* ()
    e ← *S.remove* (*S. first* ())
    P.insert(*e*, *e*)
  enquanto ¬*P.isEmpty*()
    e ← *P.removeMin*().*key*()
    S.insertLast(*e*)

Implementação baseada em sequencia

- Implementação com uma sequência não ordenada

- Desempenho:
  - insert executa em tempo $O(1)$, uma vez que podemos inserir no início ou no fim da sequência
  - removeMin e min executam em tempo $O(n)$, uma vez que temos que percorrer toda a sequencia para encontrar a menor chave

- Implementação com uma sequência ordenada

- Desempenho:
  - insert executa em tempo $O(n)$, uma vez que temos que procurar seu lugar na sequência
  - removeMin e min executam em tempo $O(1)$ uma vez que a menor chave está no início da sequência

Método da seleção

- É uma variação do PQ-Sort, onde a fila de prioridade é implementada com uma sequência não ordenada
- Tempo de execução do método da Seleção:
  1. A inserção dos elementos na fila de prioridade com n operações insert executa em tempo $O(n)$
  2. Remover os elementos de forma ordenada da fila de prioridade com n operações removeMin executa em tempo proporcional a

$$1 + 2 + \ldots + n$$

- Método da seleção roda em tempo $O(n^2)$

Exemplo de seleção

	***Sequencia** S*	***Fila de prioridade** P*
Entrada:	(7,4,8,2,5,3,9)	()
Fase 1		
(a)	(4,8,2,5,3,9)	(7)
(b)	(8,2,5,3,9)	(7,4)
..	.. ..	
.	. .	
(g)	()	(7,4,8,2,5,3,9)
Fase 2		
(a)	(2)	(7,4,8,5,3,9)
(b)	(2,3)	(7,4,8,5,9)
(c)	(2,3,4)	(7,8,5,9)
(d)	(2,3,4,5)	(7,8,9)
(e)	(2,3,4,5,7)	(8,9)
(f)	(2,3,4,5,7,8)	(9)
(g)	(2,3,4,5,7,8,9)	()

Método da inserção

- É uma variação do PQ-Sort, onde a fila de prioridade é implementada com uma sequência ordenada
- Tempo de execução do método da Inserção:
  1. Inserir os elementos na fila de prioridade com n operações insert executa em tempo proporcional a
     $$1 + 2 + \ldots + n$$
  2. Remover os elementos de forma ordenada da fila de prioridade com uma série de n operações removeMin executa em tempo $O(n)$
- Método da inserção roda em tempo $O(n^2)$

Exemplo de inserção

	***Sequencia** S*	***Fila de prioridade** P*
Entrada:	(7,4,8,2,5,3,9)	()
Fase 1		
(a)	(4,8,2,5,3,9)	(7)
(b)	(8,2,5,3,9)	(4,7)
(c)	(2,5,3,9)	(4,7,8)
(d)	(5,3,9)	(2,4,7,8)
(e)	(3,9)	(2,4,5,7,8)
(f)	(9)	(2,3,4,5,7,8)
(g)	()	(2,3,4,5,7,8,9)
Fase 2		
(a)	(2)	(3,4,5,7,8,9)
(b)	(2,3)	(4,5,7,8,9)
..	..	..
.	.	.
(g)	(2,3,4,5,7,8,9)	()

Método da inserção "in-place"

- Ao invés de usar uma estrutura de dados externa, podemos implementar os métodos da seleção e inserção "in-place"
- Uma parte da própria entrada serve como fila de prioridade
- Para o método da inserção "in-place"
  - Mantemos ordenada a parte inicial da sequencia
  - Podemos usar swapElements ao invés de modificar a sequencia