Algoritmos e Estruturas de Dados

estruturas de dados pilhas e filas

2010-2011

Carlos Lisboa Bento

AED

avaliação da ficha nº2

Inscrições e avaliação nas fichas práticas

- **Os alunos devem inscrever-se nesta aula teórica (28 de Fevereiro) ou na secretaria do DEI *até 2ª feira dia 7 de Março 17:00***
- **A inscrição deve ser feita num slot livre na turma em que estão inscritos. *A não inscrição e defesa na turma em que estão inscritos leva a avaliação *nula* na respectiva ficha.**
- **Ausências à avaliação só serão consideradas através de apresentação de atestado médido ou comprovativo de participação em actos oficiais, ex. presenças em tribunal**

Estruturas de Dados
tipo de dados / estrutura de dados

Tipo de dados

- **Especifica como é que uma dada sequência de bits deve ser interpretada**
- **Determina o espaço (em memória) que deve ser reservado para armazenar o(s) objecto(s) declarado(s)**

Exemplo:

Type nome = array[1..60] of char;

Ocupa um espaço de 60 bytes

A informação armazenada deve ser interpretada como uma cadeia de caracteres

Estruturas de Dados
tipo de dados / estrutura de dados

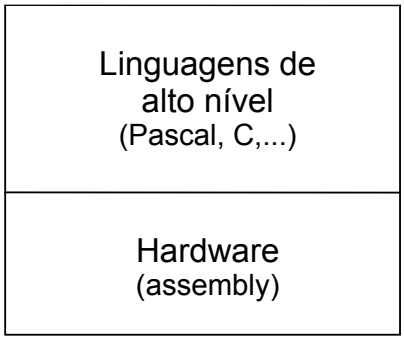
Tipo de dados

Hardware (assembly)

- **virgula flutuante**
- **inteiros**
- **cadeias de bits**
- **bits**

Estruturas de Dados
tipo de dados / estrutura de dados

Tipo de dados

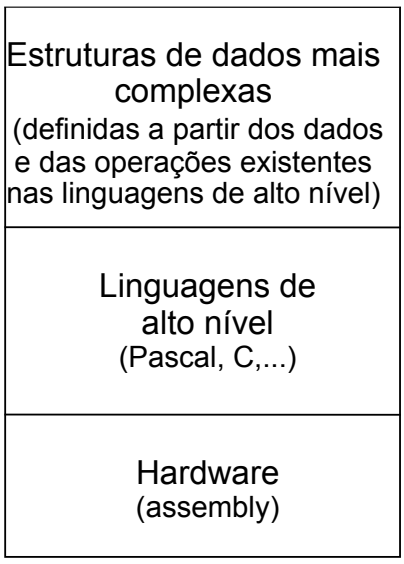


- **records**
- **arrays**
- **strings**
- **booleanos**

- **virgula flutuante**
- **inteiros**
- **cadeias de bits**
- **bits**

Estruturas de Dados
tipo de dados / estrutura de dados

Tipo de dados



- **árvores**
- **filas de espera**
- **pilhas**
- **listas**
- **grafos**

- **records**
- **arrays**
- **strings**
- **booleanos**

- **virgula flutuante**
- **inteiros**
- **cadeias de bits**
- **bits**

Estruturas de Dados
tipo de dados / estrutura de dados

Tipo de dados

Modelos de dados de mais alto nível (SGBD, ...)	- **modelo relacional** - **UML**
Estruturas de dados mais complexas (definidas a partir dos dados e das operações existentes nas linguagens de alto nível)	- **árvores** - **filas de espera** - **pilhas** - **listas** - **grafos**
Linguagens de alto nível (Pascal, C,...)	- **records** - **arrays** - **strings** - **boleanos**
Hardware (assembly)	- **virgula flutuante** - **inteiros** - **cadeias de bits** - **bits**

Estruturas de Dados
tipo de dados / estrutura de dados

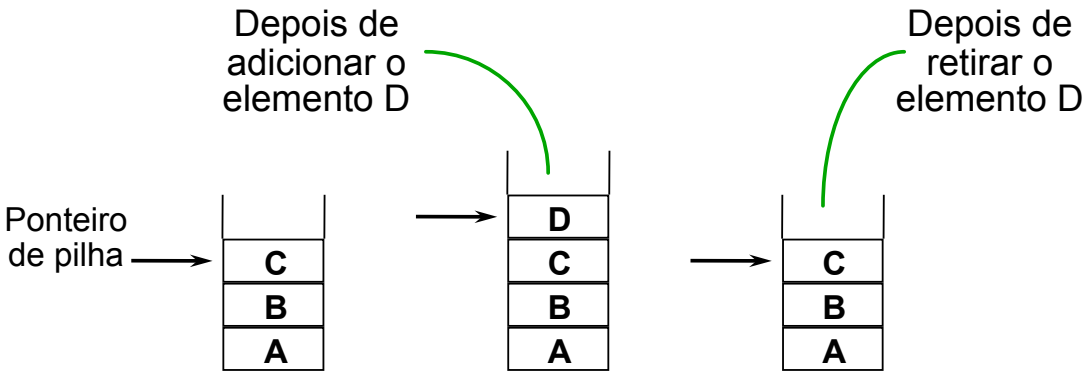
Estrutura de dados

Uma estrutura de dados compreende:
- uma representação dos dados (definição do tipo)
- um conjunto de operações sobre esses dados

Pilhas
conceitos

Colecção ordenada de elementos em que estes são adicionados ou retirados a partir de uma das suas extremidades designada por topo da pilha.

Limita o acesso ao elemento mais recentemente inserido na pilha.



Pilhas
conceitos

Operações sobre pilhas

void push(x) ➔ inserir x

Object pop( ) ➔ remover e devolver o elemento mais recentemente inserido

void makeEmpty( ) ➔ remover todos os elementos

boolean isEmpty( ) ➔ devolver true se vazia; senão false

boolean isFull() ➔ devolve true se a cheia; senão false

Pilhas
conceitos

INTERFACE

```
package DataStructures;

import Exceptions.*;

// Stack interface
//
// ******************ERRORS*****************
// pop on empty stack

/**
 * Protocol for stacks.
 * @author Mark Weiss
 */
public interface Stack
{
  Object    pop( )      throws Underflow;

  void    push( Object x );

  void    makeEmpty( );

  boolean isEmpty( );

  boolean isFull( );
}
```

APLICAÇÃO

```
import DataStructures.*;
import Exceptions.*;

// Simple test program for stacks

public final class TestStack
{
   public static void main( String [ ] args )
   {
      Stack s = new StackAr( );

      for( int i = 0; i < 5; i++ )
         s.push( new Integer( i ) );

      System.out.print( "Contents:" );
      try
      {
         for( ; ; )
            System.out.print( " " + s.pop( ) );
      }
      catch( Underflow e ) { }
      System.out.println( );
   }
}
```


Pilhas
aplicações

Apl – Exemplo #1

Análise sintactica de programas - verificação do balanceamento de símbolos
ex.: () {} [] /* */ //

```
delimiterMaching(file)
   read character ch from file;
   while not end of file
      if ch is '(', '[', or '{'
            push(ch);
      else if ch is '/'
            read the next character;
            if this character is '*'
                  push(ch);
            else ch = the character read in;
                  continue; // go to the beginning of the loop;
      else if ch is ')', ']', or '}'
            if ch and popped off delimiter do not match
               failure;
      else if ch is '*'
            read the next character;
            if this character is '/' and popped off delimiter is not '/'
                  failure;
            else ch = the character read in;
                  push back the popped off delimiter;
                  continue;
      // else ignore other characters;
         read next character ch from file;
      if stack is empty
         success;
      else failure;
```

Pilhas
aplicações

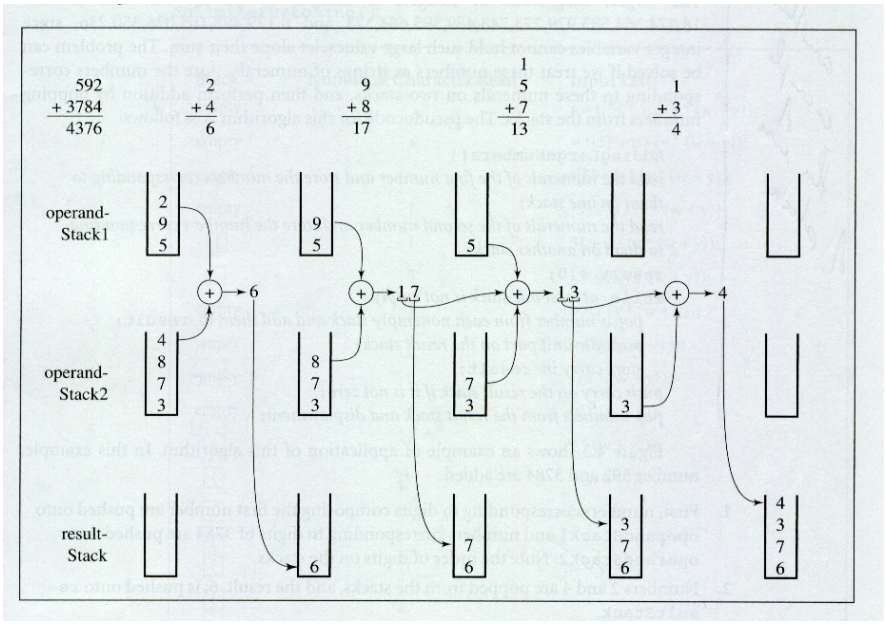
Apl – Exemplo #1

Análise sintactica de programas - verificação do balanceamento de símbolos
ex.: () {} []

Stack	Nonblank Character Read	Input Left
empty		s = t[5] + u / (v * (w + y));
empty	s	= t[5] + u / (v * (w + y));
empty	=	t[5] + u / (v * (w + y));
empty	t	[5] + u / (v * (w + y));
[	[	5] + u / (v * (w + y));
[	5	] + u / (v * (w + y));
empty	]	+ u / (v * (w + y));
empty	+	u / (v * (w + y));
empty	u	/ (v * (w + y));
empty	/	(v * (w + y));
(	(	v * (w + y));
(	v	* (w + y));
(	*	(w + y));
(		
(	(	w + y));
(		
(	w	+y));
(		
(	+	y));
(		
(	y	));
(	)	);
empty	)	;
empty	;	

Pilhas
aplicações

Apl – Exemplo #2 Adição de números inteiros muito longos



Data Structures and Algorithms in JAVA, Adam Drozdek

Pilhas
aplicações

Apl – Exemplo #2

Adição de números inteiros muito longos

```
AddingLargeNumbers()
read the numerals of the first number and store the numbers corresponding to
them on one stack;
read the numerals of the second number and store the numbers corresponding
to them on another stack;
result = 0;
while at least one stack is not empty
   pop a number from each nonempty stack and add them to result;
   push the unit part on the result stack;
   store carry in result;
push carry on the result stack if it is not zero;
pop numbers from the result stack and display them;
```


Pilhas
aplicações

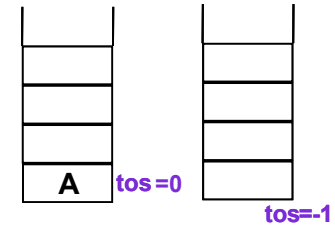
Apl – Exemplo #4

Implementação dos mecanismos de chamada de métodos em muitas linguagens de programação.

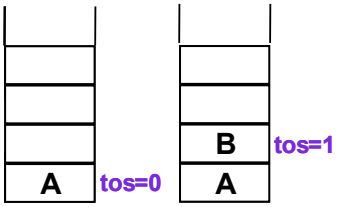
Apl – Exemplo #5

Avaliação de expressões em linguagens de programação - o valor de expressões intermédias guardado numa pilha.

Pilhas
implementação



pop()



push(B)

```
package DataStructures;

import Exceptions.*;

// StackAr class
// ******************ERRORS********************************
// pop on empty stack

/**
 * Array-based implementation of the stack.
 * @author Francisco Pereira
 */

public class StackAr implements Stack
{
   /**
    * Construct the stack.
    */
   public StackAr( )
   {
      theArray = new Object[ DEFAULT_CAPACITY ];
      topOfStack = -1;
   }
```


Pilhas
implementação

```
public boolean isEmpty( )
   {
      return topOfStack == -1;
   }

public void makeEmpty( )
   {
      topOfStack = -1;
   }


public Object pop( ) throws Underflow
   {
      if( isEmpty( ) )
         throw new Underflow( "Stack pop" );

      return theArray[ topOfStack-- ];
   }
```

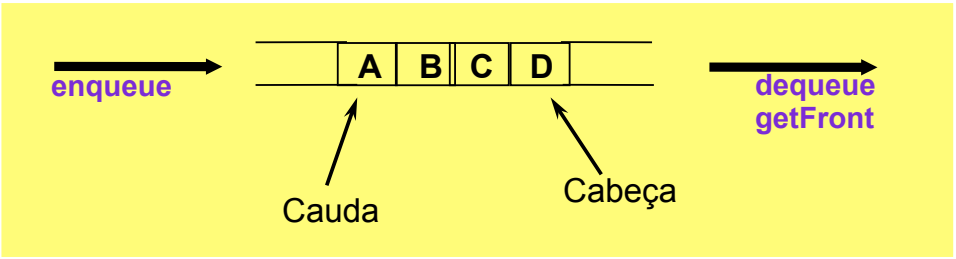
```
public void push( Object x )
   {
      if( topOfStack + 1 == theArray.length )
         doubleArray( );
      theArray[ ++topOfStack ] = x;
   }

private void doubleArray( )
   {
      Object [ ] newArray;

      newArray = new Object[ theArray.length * 2 ];
      for( int i = 0; i < theArray.length; i++ )
         newArray[ i ] = theArray[ i ];
      theArray = newArray;
   }

   private Object [ ] theArray;
   private int        topOfStack;

   static final int DEFAULT_CAPACITY = 10;
}
```

Filas de Espera
conceitos

Sequência ordenada de elementos, podendo ser adicionados novos elementos numa das extremidades (parte de trás ou cauda) e retirados na outra extremidade (parte da frente ou cabeça).

- **Só os elementos que estão à frente é que podem ser retirados**
- **Só se pode adicionar novos elementos na parte de trás**
- **A fila de espera comporta-se como um FIFO ("First-IN, First-Out")**

Filas de Espera
conceitos



void enqueue(Object x)	**--> insere x**
Object getFront( )	**--> devolve o primeiro elemento inserido**
Object dequeue( )	**--> devolve e remove o primeiro elemento inserido**
boolean isEmpty( )	**--> devolver true se vazia; senão false**
void makeEmpty( )	**--> remove todos os elementos**

Filas de Espera
tipo abstracto de dados

INTERFACE

```
package DataStructures;
import Exceptions.*;

// ******************ERRORS*****************
// getFront or dequeue on empty queue

// @author Mark Allen Weiss

public interface Queue
{
  boolean isEmpty( );
  Object getFront( ) throws Underflow;
  Object dequeue( ) throws Underflow;
  void  enqueue( Object X );
  void makeEmpty( );
}
```

APLICAÇÃO

```
import DataStructures.*;
import Exceptions.*;

// Simple test program for queues

public final class TestQueue
{
   public static void main( String [ ] args )
   {
      Queue q = new QueueAr( );

      for( int i = 0; i < 5; i++ )
         q.enqueue( new Integer( i ) );

      System.out.print( "Contents:" );
      try
      {
         for( ; ; )
            System.out.print( " " + q.dequeue( ) );
      }
      catch( Underflow e ) { }

      System.out.println( );
   }
}
```


Filas de Espera
aplicações

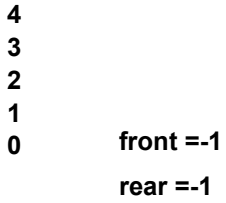
As filas de espera utilizam-se vulgarmente em três tipos de situações:

- Quando é necessário efectuar um tratamento sequencial de eventos ou de dados (o primeiro a chegar é o primeiro a ser tratado: FIFO);
- Como buffer (amortecedor) para comunicação entre processos assíncronos (permite atenuar as diferenças na cadências de produção/consumo da informação:
- p. ex. entre um computador e uma impressora
- Na simulação de mecanismos de espera em processos envolvendo o acesso de conjuntos de consumidores a recursos limitados:
- p. ex. filas espera nas caixas de um hipermercado

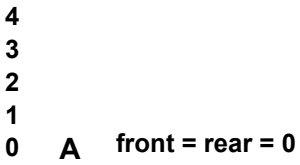
Filas de Espera
implementação

Deslocamento da fila ao longo da memória - Caso de uma fila de espera construída sobre um array de cinco posições

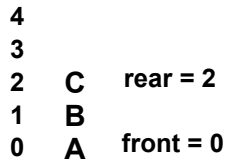
1) Início



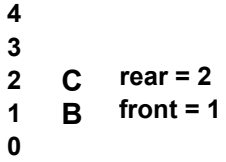
2) Inseriu A



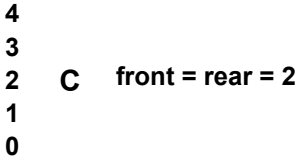
3) Inseriu B e C



4) Removeu A



5) Removeu B

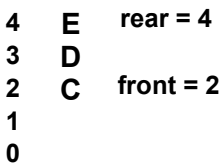


6) Inseriu D e E

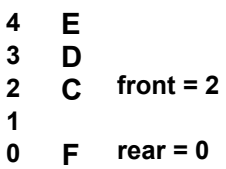
Filas de Espera
implementação

Filas de espera sobre um array circular - Solução

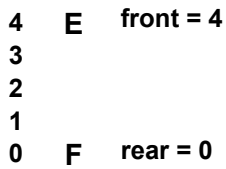
1) Após várias operações



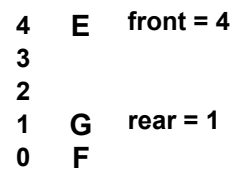
2) Inseriu F



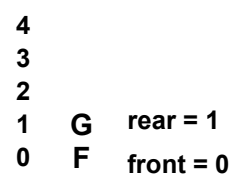
3) Removeu C e D



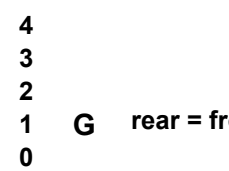
4) Inseriu G



5) Removeu E



6) Removeu F



Filas de Espera
implementação

Implementação (como apresentada no livro do Adam Drozdek)

void clear( ) ➔ remove todos os elementos

void enqueue(Object x) ➔ insere x

Object dequeue( ) ➔ devolve e remove o último elemento inserido

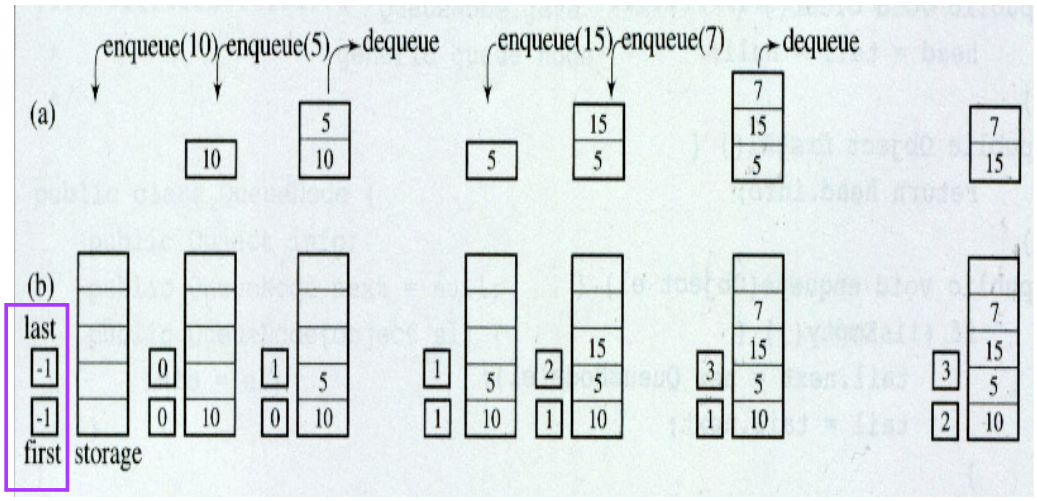
Object firstEl( ) ➔ devolve o último elemento inserido

boolean isEmpty( ) ➔ devolver true se vazia; senão false

boolean isFull( ) ➔ devolver true se cheia; senão false

Filas de Espera
implementação

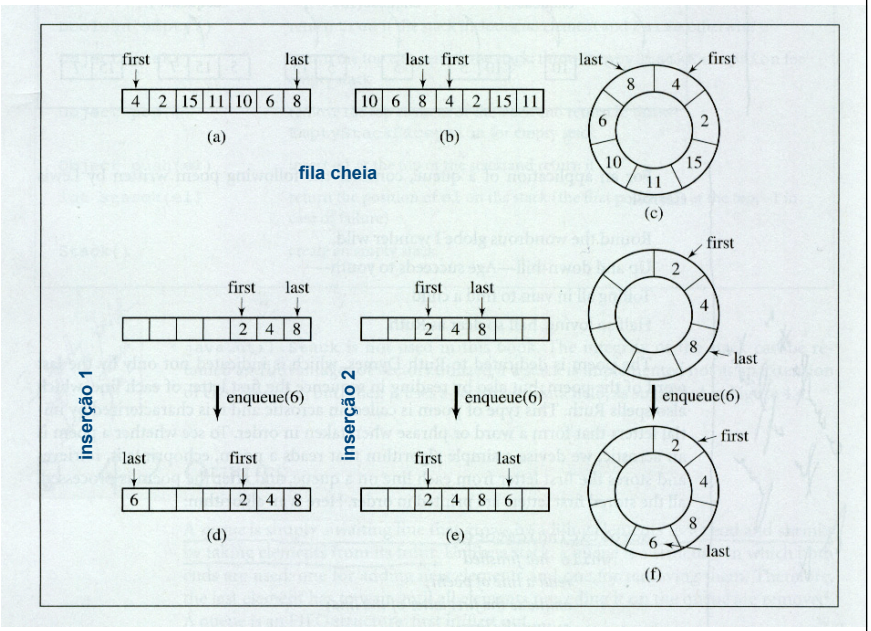
Implementação (como apresentada no livro do Adam Drozdek)



© DEI Carlos Bento ALGORITMOS E ESTRUTURAS DE DADOS 03 -

Filas de Espera
implementação

Condições de fila de espera cheia



Filas de Espera
implementação

Implementação (como apresentada no livro do Adam Drozdek)

```
// Adam Drozdek - queue implemented as an array
public class ArrayQueue {
   private int first, last, size;
   private Object[] storage;
   public ArrayQueue() {
      this(100);
   }
   public ArrayQueue(int n) {
      size = n;
      storage = new Object[size];
      first = last = -1;
   }
   public boolean isFull()  {
      return first == 0 && last == size-1 || first == last + 1;
   }
   public boolean isEmpty() {
      return first == -1;
   }
```

Filas de Espera
implementação

Implementação (como apresentada no livro do Adam Drozdek)

```
public void enqueue(Object el) {
    if (last == size-1 || last == -1) {
        storage[0] = el;
        last = 0;
        if (first == -1)
        first = 0;
    }
    else storage[++last] = el;
  }
  public Object dequeue() {
    Object tmp = storage[first];
    if (first == last)
        last = first = -1;
    else if (first == size-1)
        first = 0;
    else first++;
    return tmp;
  }
  public void printAll() {
    for (int i = 0; i < size; i++)
       System.out.print(storage[i] + " ");
  }
}
```


Filas de Espera
aplicações

Apl – Exemplo # 1 – Simulação do balção de atendimento de um banco.
Tendo dados sobre a cadência de chegada de clientes ao banco e os tempos de atendimento determinar quantos caixas são necessários para ter um atendimento com tempos de espera aceitáveis (do livro do Adam Drozdek)

Number of customers per minute	Percentage of one-minute intervals	Range
0	15	1–15
1	20	16–35
2	25	36–60
3	10	61–70
4	30	71–100

(a)

The amount of of time needed for service in seconds	Percentage of customers	Range
0	0	—
10	0	—
20	0	—
30	10	1–10
40	5	11–15
50	10	16–25
60	10	26–35
70	0	—
80	15	36–50
90	25	51–75
100	10	76–85
110	15	86–100

(b)

Em 15% dos periodos de 1 mn entram 0 clientes
Em 20% dos periodos de 1 mn entram 1 clientes
Em 25% dos periodos de 1 mn entram 2 clientes
Em 10% dos periodos de 1 mn entram 3 clientes
Em 30% dos periodos de 1 mn entram 4 clientes

Filas de Espera
aplicações

Apl – Exemplo # 1 – Simulação do balcão de atendimento de um banco.

```
import java.util.*;

class BankSimulation {
   static Random rd = new Random();

   static int Option(int percents[]) {
      int i = 0, perc, choice = Math.abs(rd.nextInt()) % 100 + 1;
      for (perc = percents[0]; perc < choice; perc += percents[i+1], i++);
      //  System.out.println(" option2 "+choice+" "+ i +" "+perc+" "+percents[i]+" ");
      return i;
   }

   public static void main(String args[]) {
      int[] arrivals ={15,20,25,10,30};
      int[] service ={0,0,0,10,5,10,10,0,15,25,10,15};
      // NUMERO DE CAIXAS
      int[] clerks = {0,0,0,0};
      int clerksSize = clerks.length;
      int customers, t, i, numOfMinutes = 100, x;
      double thereIsLine = 0.0;
      Queue simulQ = new Queue();
```


Filas de Espera
aplicações

Apl – Exemplo # 1 – Simulação do balcão de atendimento de um banco.

```
for (t = 1; t <= numOfMinutes; t++) {
        System.out.print(" t = " + t);
        for (i = 0; i < clerksSize; i++)  // after each minute subtract
           if (clerks[i] < 60)              // at most 60 seconds from time
               clerks[i] = 0;               // after the 60 sec the clerk does not have work to do
           else clerks[i] -= 60;            // after the 60 sec the clerk has x-60 sec work to do

        customers = Option(arrivals);           // number of costumers arriving this minute
        for (i = 0; i < customers; i++) {       // enqueue all new customers
           x = Option(service)*10;              // time need for the costumer attendance (0-110 sec)
           simulQ.enqueue(new Integer(x));    // they require)
      }
```

Filas de Espera
aplicações

Apl – Exemplo # 1 – Simulação do balcão de atendimento de um banco.

```
// dequeue customers when clerks are available:
        for (i = 0; i < clerksSize && !simulQ.isEmpty(); )
            if (clerks[i] < 60) {
                x = ((Integer) simulQ.dequeue()).intValue();
                clerks[i] += x;                          // to a clerk if service time is still below 60 sec
            }
            else i++; // only increment clerk number if the current one is above 60 sec loaded
        if (!simulQ.isEmpty()) {
            thereIsLine++;
        }
        else System.out.println(" wait = 0;");
    }
    System.out.println("\nFor " + clerksSize + " clerks, there was a line "
        + thereIsLine/numOfMinutes*100.0 + "% of the time;\n");
  }
}
```


Análise de Complexidade
(leituras)

Sedgewick, Cap.4, pp127-153

Sedgewick, Cap.3, pp69-126

- Comprender o conceito de Tipo Abstracto de Dados (TAD)
- Conhecer e compreender as seguintes estruturas de dados:
  - matrizes;
  - pilhas; filas
- Saber como desenhar um programa simples de simulação de filas de espera

Análise de Complexidade

… bom trabalho, FIM!