Módulo 3

Estruturas de Dados

Lição 3

Queue

Versão 1.0 - Mai/2007

Autor
Joyce Avestro

Equipe
Joyce Avestro
Florence Balagtas
Rommel Feria
Reginald Hutcherson
Rebecca Ong
John Paul Petines
Sang Shin
Raghavan Srinivas
Matthew Thompson

Necessidades para os Exercícios

Sistemas Operacionais Suportados

NetBeans IDE 5.5 para os seguintes sistemas operacionais:

- Microsoft Windows XP Profissional SP2 ou superior
- Mac OS X 10.4.5 ou superior
- Red Hat Fedora Core 3
- Solaris™ 10 Operating System (SPARC® e x86/x64 Platform Edition)

NetBeans Enterprise Pack, poderá ser executado nas seguintes plataformas:

- Microsoft Windows 2000 Profissional SP4
- Solaris™ 8 OS (SPARC e x86/x64 Platform Edition) e Solaris 9 OS (SPARC e x86/x64 Platform Edition)
- Várias outras distribuições Linux

Configuração Mínima de Hardware

Nota: IDE NetBeans com resolução de tela em 1024x768 pixel

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	500 MHz Intel Pentium III workstation ou equivalente	512 MB	850 MB
Linux	500 MHz Intel Pentium III workstation ou equivalente	512 MB	450 MB
Solaris OS (SPARC)	UltraSPARC II 450 MHz	512 MB	450 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Série 1.8 GHz	512 MB	450 MB
Mac OS X	PowerPC G4	512 MB	450 MB

Configuração Recomendada de Hardware

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	1 GB
Linux	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	850 MB
Solaris OS (SPARC)	UltraSPARC IIIi 1 GHz	1 GB	850 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Series 1.8 GHz	1 GB	850 MB
Mac OS X	PowerPC G5	1 GB	850 MB

Requerimentos de Software

NetBeans Enterprise Pack 5.5 executando sobre Java 2 Platform Standard Edition Development Kit 5.0 ou superior (JDK 5.0, versão 1.5.0_01 ou superior), contemplando a Java Runtime Environment, ferramentas de desenvolvimento para compilar, depurar, e executar aplicações escritas em linguagem Java. Sun Java System Application Server Platform Edition 9.

- Para **Solaris**, **Windows**, e **Linux**, os arquivos da JDK podem ser obtidos para sua plataforma em http://java.sun.com/j2se/1.5.0/download.html
- Para **Mac OS X**, Java 2 Plataform Standard Edition (J2SE) 5.0 Release 4, pode ser obtida diretamente da Apple's Developer Connection, no endereço: http://developer.apple.com/java (é necessário registrar o download da JDK).

Para mais informações: http://www.netbeans.org/community/releases/55/relnotes.html

Colaboradores que auxiliaram no processo de tradução e revisão

Alexandre Mori
Alexis da Rocha Silva
Aline Sabbatini da Silva Alves
Allan Wojcik da Silva
André Luiz Moreira
Anna Carolina Ferreira da Rocha
Antonio Jose R. Alves Ramos
Aurélio Soares Neto
Bárbara Angélica de Jesus Barbosa
Bruno da Silva Bonfim
Bruno dos Santos Miranda
Bruno Ferreira Rodrigues
Carlos Alexandre de Sene
Carlos Eduardo Veras Neves
Cleber Ferreira de Sousa
Everaldo de Souza Santos
Fabrício Ribeiro Brigagão
Fernando Antonio Mota Trinta
Frederico Dubiel
Givailson de Souza Neves
Jacqueline Susann Barbosa
João Paulo Cirino Silva de Novais
João Vianney Barrozo Costa
José Augusto Martins Nieviadonski
José Ricardo Carneiro
Kleberth Bezerra G. dos Santos
Kefreen Ryenz Batista Lacerda
Leonardo Leopoldo do Nascimento
Lucas Vinícius Bibiano Thomé
Luciana Rocha de Oliveira
Luís Carlos André
Luiz Fernandes de Oliveira Junior
Luiz Victor de Andrade Lima
Marco Aurélio Martins Bessa
Marcos Vinicius de Toledo
Marcus Borges de S. Ramos de Pádua
Maria Carolina Ferreira da Silva
Massimiliano Giroldi
Mauricio da Silva Marinho
Mauro Cardoso Mortoni
Mauro Regis de Sousa Lima
Namor de Sá e Silva
Nolyanne Peixoto Brasil Vieira
Paulo Afonso Corrêa
Paulo Oliveira Sampaio Reis
Pedro Antonio Pereira Miranda
Renato Alves Félix
Renê César Pereira
Reyderson Magela dos Reis
Ricardo Ulrich Bomfim
Robson de Oliveira Cunha
Rodrigo Fernandes Suguiura
Rodrigo Vaez
Ronie Dotzlaw
Rosely Moreira de Jesus
Seire Pareja
Silvio Sznifer
Tiago Gimenez Ribeiro
Vanderlei Carvalho Rodrigues Pinto
Vanessa dos Santos Almeida

Auxiliadores especiais

Revisão Geral do texto para os seguintes Países:

- **Brasil** – Tiago Flach
- **Guiné Bissau** – Alfredo Cá, Bunene Sisse e Buon Olossato Quebi – ONG Asas de Socorro

Coordenação do DFJUG

- **Daniel deOliveira** – JUGLeader responsável pelos acordos de parcerias
- **Luci Campos -** Idealizadora do DFJUG responsável pelo apoio social
- **Fernando Anselmo -** Coordenador responsável pelo processo de tradução e revisão, disponibilização dos materiais e inserção de novos módulos
- **Rodrigo Nunes -** Coordenador responsável pela parte multimídia
- **Sérgio Gomes Veloso -** Coordenador responsável pelo ambiente JEDI™ (Moodle)

Agradecimento Especial

John Paul Petines – Criador da Iniciativa JEDI™
Rommel Feria – Criador da Iniciativa JEDI™

1. Objetivos

Uma ***queue*** (fila) é um conjunto de elementos ordenado linearmente que têm as características ***First-In*** (Primeiro a entrar) e ***First-Out*** (Primeiro a sair). Conhecido também como lista **FIFO**.

Existem duas operações básicas para *queue*: (1) inserção no final, e (2) remoção no início.

Ao final desta lição, o estudante será capaz de:

- Definir os conceitos básicos e operações com ***queue* ADT**
- Implementar uma ***queue* ADT** usando representação **seqüencial** e **encadeada**
- Realizar operações em ***queues* circulares**
- Usar a ordenação topológica para produzir uma organização dos elementos que satisfaça a um padrão estabelecido

2. Representação de *Queue*

Para definir uma *queue* em Java, devemos usar a seguinte **interface**:

```
interface Queue{
  // Insere um item
  void enqueue(Object item) throws QueueException;

  // Remove um item
  Object dequeue() throws QueueException;
}
```

Assim como na ***stack***, devemos usar o seguinte código de Exception a fim de tratarmos as exceções:

```
class QueueException extends RuntimeException{
  public QueueException(String err){
    super(err);
  }
}
```

Como a ***stack***, a ***queue*** deve ser implementada utilizando a **representação seqüencial** ou **encadeada**.

2.1. Representação Seqüencial

Se a execução utiliza uma representação seqüencial, um *array* unidimensional é usado, portanto o tamanho é estático. Se a *queue* tem dados, ***front*** aponta para seu primeiro elemento, enquanto que ***rear*** aponta para a célula seguinte à última ocupada. A *queue* está vazia quando ***front*** é igual a ***rear*** e cheia quando ***front*** é igual a zero e ***rear*** é igual a **n**. Tentar remover um item de uma *queue* vazia causa um *underflow*, enquanto que tentar inserir em uma *queue* cheia causa um *overflow*. A figura a seguir mostra um exemplo de *queue*:

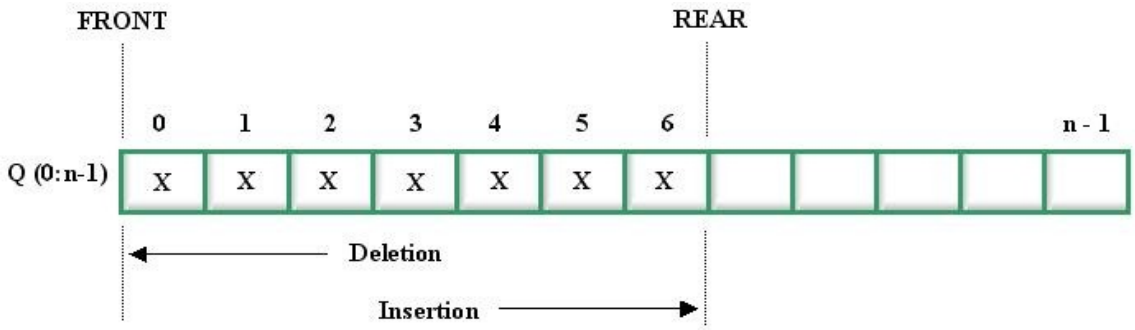


Figura 1: Operações em uma fila

Para iniciar, igualamos ***front*** e ***rear*** a 0:

```
front  = 0;
rear = 0;
```

Para inserir um item, digamos x, fazemos o seguinte:

```
Q [rear] = item;
Rear ++;
```

e para remover um item, fazemos o seguinte:

```
x = Q[front];
front ++;
```

Para implementar uma *queue* utilizando a representação seqüencial:

```
class SequentialQueue implements Queue{
  Object Q[];
  int n = 100 ;   // tamando da fila , padrão 100
  int front = 0;  // inicio e fim iniciar como 0
  int rear = 0;

  // Cria uma queue com tamanho definido de 100 elementos
  public SequentialQueue() {
    Q = new Object[n];
  }

  // Cria uma queue com tamanho a definir
  public SequentialQueue(int size) {
    n = size;
    Q = new Object[n];
  }

  // Insere um item na queue
  public void enqueue(Object item) throws QueueException {
    if (rear == n)
      throw new QueueException("Inserting into a full queue.");
    Q[rear] = item;
    rear++;
  }

  // Remove um item da queue
  public Object dequeue() throws QueueException {
    if (front == rear)
      throw new QueueException("Deleting from an empty queue.");
    Object x  = Q[front];
    front++;
    return x;
  }
}
```

Sempre que uma remoção é feita, um espaço vago é criado na frente da *queue*. Portanto, existe a necessidade de mover os itens de forma que o espaço vazio fique no fim da *queue* para futuras inserções. O método *moveQueue* executa esse procedimento. Esse método é chamado pelo código abaixo:

```
public void moveQueue() throws QueueException {
  if (front==0)
    throw new QueueException("Inserting into a full queue");
  for (int i=front; i<n; i++)
    Q[i-front] = Q[i];
  rear = rear - front;
```

```
    front = 0;
  }
```

É preciso modificar a execução do método ***enqueue*** para se utilizar do método ***moveQueue***:

```
  public void enqueue(Object item) {
    // se rear está no fim do array
    if (rear == n)
      moveQueue();
    Q[rear] = item;
    rear++;
  }
```

Podemos testar esta representação com a seguinte classe:

```
  public class TestQueue {
    public static void main(String [] args) {
      SequentialQueue queue = new SequentialQueue(4);
      queue.enqueue("1");
      queue.enqueue("2");
      queue.enqueue("3");
      queue.enqueue("4");
      System.out.println(queue.dequeue());
      System.out.println(queue.dequeue());
      System.out.println(queue.dequeue());
      System.out.println(queue.dequeue());
    }
  }
```

Como resultado, lembre-se que a *queue* armazena "Enfileirando" os dados, e retirá-os do primeiro ao último elemento armazenado.

2.2. Representação Encadeada

Representação encadeada também pode ser utilizada para uma *queue*. Da mesma forma que para *stacks*, utilizará ***nodes*** com os campos ***INFO*** e ***LINK***. Na representação acoplada, um *node* com a estrutura definida a seguir será utilizado:

```
class Node {
  private Object info;
  private Node link;
  public Node(Object o, Node n) {
    info = o;
    link = n;
  }
}
```

A figura a seguir mostra uma *queue* implementada como uma *queue* encadeada:

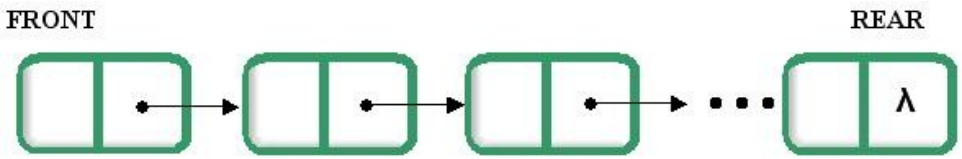


Figura 2: Representação encadeada de uma *Queue*

As definições já vistas sobre *node* serão utilizadas aqui.

A *queue* está vazia se ***front*** é igual a *null*. Na representação encadeada, desde que a *queue* cresça dinamicamente, o *overflow* acontecerá somente quando a classe ficar sem espaço para novas inserções e tratar disso está fora do escopo desse tópico.

O seguinte classe executa a representação encadeada de uma *queue* ADT:

```
class LinkedQueue implements Queue {
  private Node front;
  private Node rear;

  // Cria uma queue vazia
  public LinkedQueue() {
  }
  // Cria uma queue com n NODES inicialmente
  public LinkedQueue(Node n) {
    front = n;
    rear = n;
  }
  // Insere um item na queue
  public void enqueue(Object item) {
    Node n = new Node(item, null);
    if (front == null) {
      front = n;
      rear = n;
    } else {
      rear.link = n;
      rear = n;
    }
  }
  // Remove um item da queue
  public Object dequeue() throws QueueException {
    Object x;
    if (front == null)
      throw new QueueException("Deleting from an empty queue.");
    x = front.info;
    front = front.link;
    return x;
  }
}
```

Podemos testar esta representação com a seguinte classe:

```
public class TestQueue {
  public static void main(String [] args) {
    LinkedQueue queue = new LinkedQueue();
    queue.enqueue("1");
    queue.enqueue("2");
    queue.enqueue("3");
    queue.enqueue("4");
    System.out.println(queue.dequeue());
    System.out.println(queue.dequeue());
    System.out.println(queue.dequeue());
    System.out.println(queue.dequeue());
  }
}
```

3. *Queue* Circular

Uma desvantagem da implementação seqüencial anterior é a necessidade de se mover os elementos, no caso de *rear* ser igual a n e *front* ser maior que *0*, para abrir espaço a fim de inserir um novo elemento. Se a *queue* fosse vista como um círculo não haveria necessidade de executar esse movimento. Em uma *queue* circular, os elementos são considerados como se estivessem organizados dentro de um círculo. O *front* aponta para o elemento atual no início da *queue*, enquanto o *rear* aponta para o elemento à direita do último, no momento (sentido horário). A figura a seguir mostra uma *queue* circular:

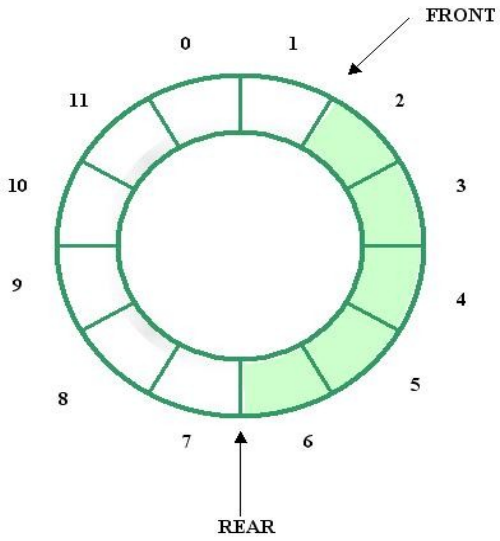


Figura 3: *Queue* Circular

Para iniciar uma *queue* circular:

```
front = 0;
rear = 0;
```

Para inserir um item, por exemplo, **x**:

```
Q[rear] = x;
rear  = (rear + 1) mod n;
```

Para apagar:

```
x = Q[front];
front = (front + 1) mod n;
```

Utilizamos a função MOD (módulo) para realizar um teste no início e final da *queue*. Quando inserções e remoções são feitas, a *queue* é movimentada em sentido horário. Se o início alcançar o final, isto é, se *front* é igual a *rear*, então teremos uma *queue* vazia. Se o final alcançar o início, uma condição também indicada por *front* igual a *rear*, então todos os elementos estão em uso e teremos uma *queue* cheia.

Para evitarmos ter a mesma relação significando duas condições diferentes, não permitiremos que o final alcance o início considerando que a *queue* está cheia quando existir apenas uma célula livre. Portanto a *queue* cheia é indicada por:

```
front == (rear + 1) mod n
```

Este é um exemplo completo para uma implementação de uma *queue* circular:

```
public class CircularQueue implements Queue{
  Object Q[];
  int n = 20;     // tamanho da queue, por padrão 20
  int front = 0;
  int rear = 0;


  // Cria a circular queue de tamanho padrão
  public CircularQueue(){
    Q = new Object[n];
  }

  // Cria a circular queue de tamanho n
  public CircularQueue(int size){
    n = size;
    Q = new Object[n];
  }

  public void enqueue(Object item) throws QueueException {
    if (front == (rear % n) + 1)
      throw new QueueException("Inserting into a full queue.");
    Q[rear] = item;
    rear    = (rear % n) + 1;
  }

  public Object dequeue() throws QueueException {
    Object x;
    if (front == rear)
      throw new QueueException("Deleting from an empty queue.");
    x = Q[front];
    front = (front % n) + 1;
    return x;
  }

  // Método principal para testar a queue
  public static void main(String args[]) {
    CircularQueue q = new CircularQueue(5);
    for (int i=1; i < 7; i++) {
      q.enqueue(new Integer(i));
      System.out.println(i +" inserted");
      System.out.println(q.dequeue() + " retrieved");
      System.out.println("front:"+q.front+" rear:"+q.rear);
    }
  }
}
```

4. Aplicação: Classificação Topológica

A Classificação Topológica é um problema característico de redes ativas. Utiliza ambas as técnicas de alocação, seqüencial e encadeada, na qual a *queue* encadeada está inserida em um array seqüencial.

É um processo aplicado em *elementos parcialmente ordenados*. A entrada é um conjunto de pares de *condicionamento parcial* e a saída é a lista de elementos, onde não existe nenhum elemento cujo antecessor já não esteja na saída.

4.1. Ordenação Parcial

É definida como uma relação entre os elementos de um conjunto S, caracterizada pelo símbolo $\preccurlyeq$ (que é lido como '*precede ou igual a*'). A seguir estão as propriedades da *condição* parcial $\preccurlyeq$:

- Transitividade: se $x \preccurlyeq y$ e $y \preccurlyeq z$, então $x \preccurlyeq z$
- Anti-simetria: se $x \preccurlyeq y$ e $y \preccurlyeq x$, então $x = y$
- Reflexividade: $x \preccurlyeq x$

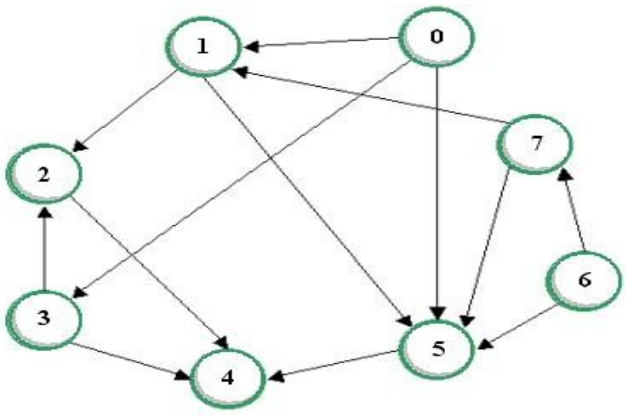
Resultado. Se $x \preccurlyeq y$ e $x \neq y$ então $x \prec y$.

Propriedades equivalentes são:

- Transitividade: se $x \prec y$ e $y \prec z$, então $x \prec z$
- Simetria: se $x \prec y$ então $y \prec x$
- Não-Reflexividade: $x \prec x$

Um exemplo familiar de condição parcial na matemática é a relação $\mathbf{u \subseteq v}$ entre os conjuntos u e v. A seguir temos outro exemplo onde a lista de condição parcial é mostrada à esquerda; o gráfico que ilustra a condição parcial é mostrado no centro, e a saída esperada é mostrada à direita.

0,1
0,3
0,5
1,2
1,5
2,4
3,2
3,4
5,4
6,5
6,7
7,1
7,5



Saída:
0 6 3 7 1 2 5 4

Exemplo de Classificação Topológica

4.2. Algoritmo

Juntamente com a execução do algoritmo, devemos considerar alguns componentes discutidos no capítulo 1- *input* (entrada), *output* (saída), e o *algoritmo apropriado*.

- *Input* (Entrada): um conjunto de pares com a forma (i, j) para cada relação i ≼ j que poderia representar o ordenamento parcial dos elementos. Os pares de entrada podem estar em qualquer ordem;
- *Output* (Saída): O algoritmo se torna uma seqüência linear de itens, de modo que nenhum item aparece na seqüência antes de seu antecessor direto;
- *Algoritmo apropriado*: Uma condição da Classificação Topológica é não visualizar/imprimir os itens cujos seus antecessores ainda não tenham executado esta tarefa. Para fazer isso, é necessário manter os números dos antecessores em cada item. Um *array* pode ser usado para esse propósito. Chamaremos esse *array* de COUNT (contador). Quando um item é enviado à saída o ***count*** de cada sucessor do item é decrementado. Se o ***count*** de um item é zero, ou se torna zero como resultado de todos os seus antecessores terem sido enviados à saída, esse seria o tempo em que os itens estão prontos para a visualização/impressão. Para manter-se a par dos sucessores, uma lista de ligações, chamada **SUC**, com a estrutura (INFO, LINK), será usada, onde INFO contém o rótulo do sucessor direto enquanto LINK aponta para o próximo sucessor, se existir.

Segue a definição de *node*:

```
class Node {
  int info;
  Node link;
}
```

O **COUNT** (*array* contador) é inicialmente definido como *0* e o *array* **SUC** como ***null*** para cada entrada do par (i, j),

```
COUNT[j]++;
```

e um **newNODE** (nó) é adicionado à **SUC**(i):

```
Node newNode = new Node();
newNode.info = j;
newNode.link = SUC[i];
SUC[i] = newNode;
```

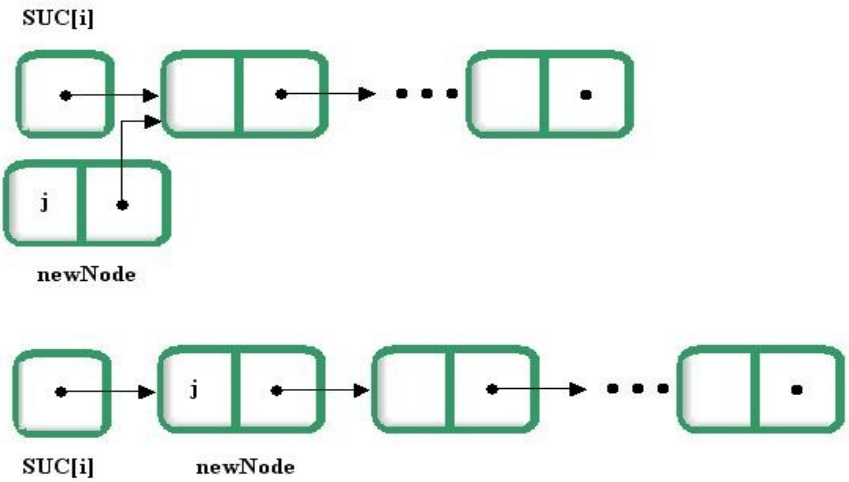


Figura 4: Adicionando um newNODE

Segue exemplo:

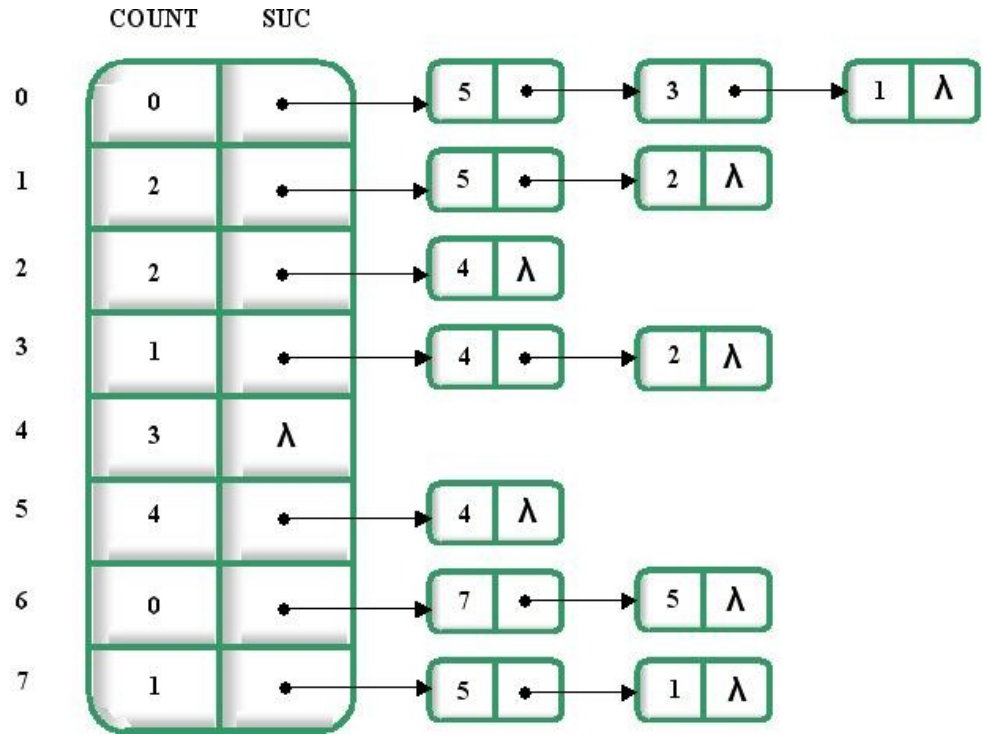


Figura 5: Representação de um exemplo de Classificação Topológica

Para gerar a saída, que é uma classificação linear de objetos de modo que nenhum objeto apareça na seqüência antes de seu antecessor direto, procedemos da seguinte maneira:

1. Procuramos por um item, digamos k, com o contador dos antecessores diretos igual a zero, ex., COUNT[k]==0. Coloque k na saída;
2. Buscamos a lista de sucessores diretos de k, e decrementamos 1 do contador de cada sucessor;
3. Repetir passos 1 e 2 até que todos os itens estejam na saída.

Para evitar percorrer todo o *array* **COUNT** repetidamente enquanto procuramos por objetos com o contador igual a zero, iremos constituir todos os objetos em uma ***queue* encadeada**. Inicialmente, a *queue* irá consistir de itens sem antecessor direto (sempre haverá ao menos um item). Subseqüentemente, cada vez que o contador de antecessores diretos de um item cair para zero, este será inserido na *queue*, pronto para a saída. Desde que para cada item, digamos **j**, com seu contador igual a *0*, podemos reutilizar COUNT [j] como um campo de ligação de modo que:

COUNT [j] = **k** se **k** é o próximo item na *queue*
= **0** se **j** for o último elemento na *queue*

Conseqüentemente temos uma *embedded linked queue* em um *array* seqüencial.

Se a entrada para o algoritmo estiver correta, isto é, se as relações de entrada satisfazem a condição parcial, o algoritmo termina quando a fila estiver vazia e com todos os objetos colocados na saída. Se, por outro lado, a condição parcial é violada de modo que existam objetos que constituem um ou mais laços de repetição (por exemplo, $1 \prec 2$; $2 \prec 3$; $3 \prec 4$; $4 \prec 1$), ainda assim este algoritmo termina, mas objetos incluídos nos laços de repetição não serão colocados na saída.

Essa abordagem da Classificação Topológica usa tanto técnicas seqüenciais quanto técnicas encadeamento de alocamento, e o uso de uma *queue* encadeada inserida em um *array* seqüencial.

5. Exercícios

a) *Ordenação Topológica.* Dado o ordenamento parcial de sete elementos, como eles podem ser arranjados de forma que nenhum elemento apareça na seqüência antes de seu antecessor direto?

```
1. (1,3), (2,4), (1,4), (3,5), (3,6), (3,7), (4,5), (4,7), (5,7), (6,7), (0,0)
2. (1,2), (2,3), (3,6), (4,5), (4,7), ( 5,6), (5,7), (6,2), (0,0)
```

5.1. Exercícios para Programar

1. Crie uma execução de múltiplas *queue* que coexista num *array* simples. Use o algoritmo de *Garwick's* para realocação de memória durante o *overflow.*

2. Escreva uma classe que execute o algoritmo de ordenação topológica.

3. Relação de matérias (escolares) utilizando Ordenação Topológica.

Execute uma relação de matérias utilizando o algoritmo de ordenação topológica. A classe deve solicitar um arquivo que contenha o conjunto de matérias e a ordenação parcial destas. As matérias devem estar na seguinte forma, no arquivo, **(número, matéria)** onde **número** é um inteiro atribuído a matéria e **matéria** é o identificador do curso [ex.: **(1, CS 1)**]. Cada par **(número, matéria)** deve estar em linhas separadas no arquivo. Para terminar a inclusão deve-se usar o par **(0, 0).** Os pré-requisitos das matérias devem ser obtidos a partir do mesmo arquivo. A definição de um pré-requisito deve estar na forma **(i, j)**, uma linha por par, onde **i** é um número atribuído ao pré-requisito da matéria de número **j**. O último pré-requisito deve ser o par **(0, 0)**.

A saída deve ser também em um arquivo, e seu nome deve ser solicitado ao usuário. A saída deve ser em uma tabela contendo o número do semestre (um número auto-incrementável de 1 a n) junto com as matérias do mesmo.

Para simplificar, consideraremos apenas matérias semestrais.

<table>
<tr><th>Exemplo de arquivo de entrada</th><th></th><th>Exemplo de arquivo de saída</th></tr>
<tr>
<td>(1, CS 1)
(2, CS 2)
.
.
.
(0, 0)
(1, 2)
(2, 3)
.
.
.
(0, 0)</td>
<td>← Início da definição de ordem parcial.</td>
<td>
<table>
<tr><td>Sem 1</td><td>Sem 2</td></tr>
<tr><td>CS 1</td><td>CS 12</td></tr>
<tr><td></td><td></td></tr>
<tr><td></td><td></td></tr>
<tr><td>Sem 2</td><td>Sem 4</td></tr>
<tr><td>CS 2</td><td>CS 135</td></tr>
<tr><td>CS 3</td><td>CS 140</td></tr>
<tr><td></td><td>CS 150</td></tr>
</table>
</td>
</tr>
</table>

Parceiros que tornaram JEDI™ possível

Instituto CTS
Patrocinador do DFJUG.

Sun Microsystems
Fornecimento de servidor de dados para o armazenamento dos vídeo-aulas.

Java Research and Development Center da Universidade das Filipinas
Criador da Iniciativa JEDI™.

DFJUG
Detentor dos direitos do JEDI™ nos países de língua portuguesa.

Banco do Brasil
Disponibilização de seus *telecentros* para abrigar e difundir a Iniciativa JEDI™.

Politec
Suporte e apoio financeiro e logístico a todo o processo.

Borland
Apoio internacional para que possamos alcançar os outros países de língua portuguesa.

Instituto Gaudium/CNBB
Fornecimento da sua infra-estrutura de hardware de seus servidores para que os milhares de alunos possam acessar o material do curso simultaneamente.