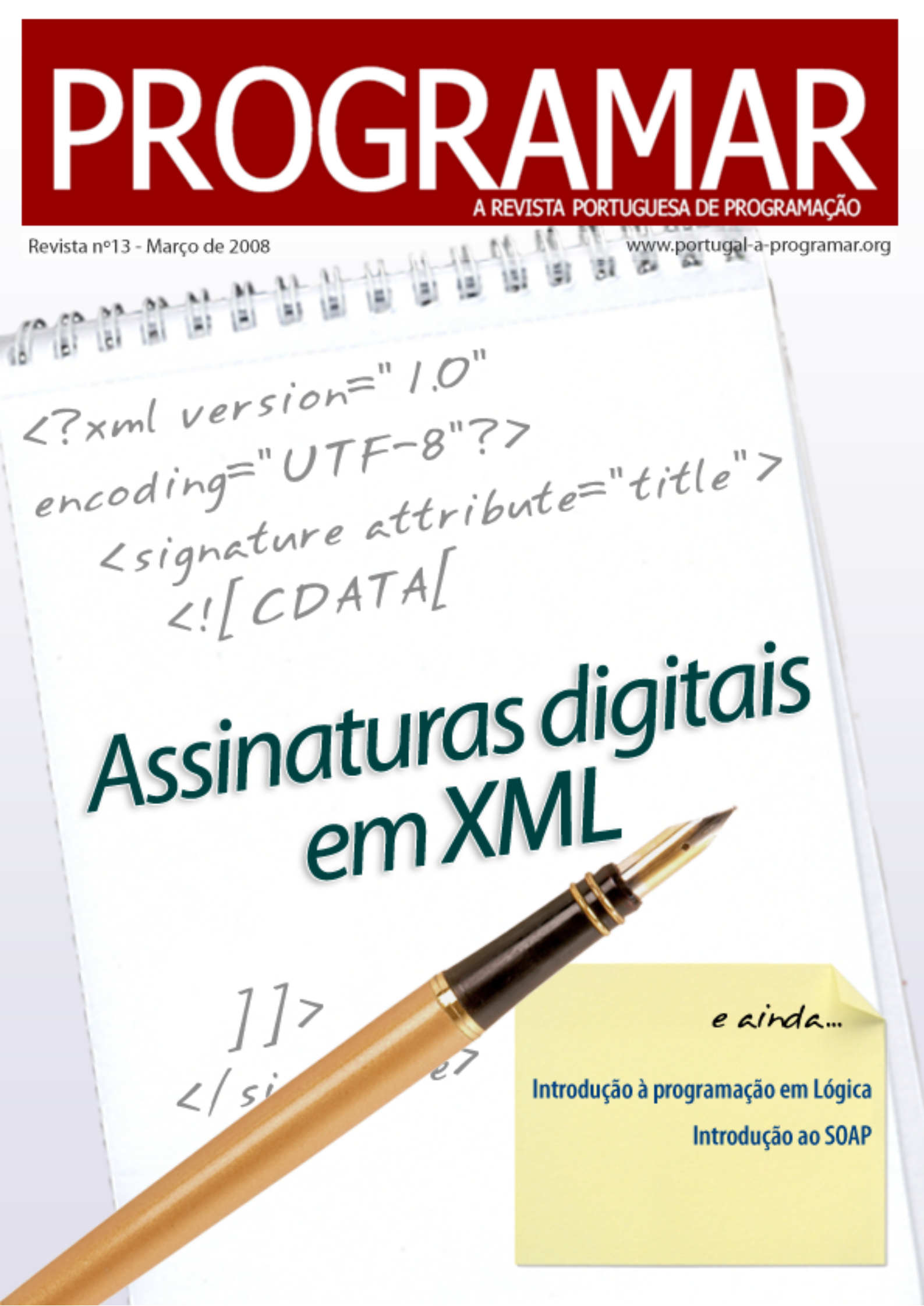
PROGRAMAR

A REVISTA PORTUGUESA DE PROGRAMAÇÃO

Revista nº13 - Março de 2008

www.portugal-a-programar.org



Assinaturas digitais em XML

e ainda...

Introdução à programação em Lógica

Introdução ao SOAP

índice

equipa PROGRAMAR

24 meses e a história de um projecto...

Era uma vez uma comunidade jovem de programação, bastante jovem. Os seus criadores e primeiros membros não eram gurus nem profissionais e muito menos *leets*. No entanto, no seio desta emanava o desejo de inovar e criar, duas palavras desde há muito apresentadas como a verdadeira matéria-prima para os alicerces do edifício do desenvolvimento deste país. Daí proveio a revista PROGRAMAR...

A ideia surgiu, finalmente, como todas surgem, brilhantemente, por David Pintassilgo, e o final de 2005 acabou agitado e não menos atribulado em longas discussões e questões existenciais. Será que é possível, que conseguimos, que queremos ou sequer podemos?

De facto queríamos, podíamos e conseguiríamos, só precisávamos de alguém que nos mostrasse tal, um pequeno condensador que na sua descarga poria toda a máquina a trabalhar. Sérgio Matias assim nos apresentou quase do nada a 1ª edição, uma compilação de alguns tutoriais e artigos espalhados pelo fórum, e todos nós admirámos e contemplámos: era de facto possível!

Torna-se interessante constatar que o nosso projecto, apesar de tudo o que de bom dele proveio, não é nada do outro mundo, mas apenas um pequeno esboço do que uma comunidade pode fazer quando motivada. Porque não há então mais projectos destes? Porque é que a revista PROGRAMAR a ninguém veio fazer concorrência quando foi lançada? Porque não ousam mais os anónimos da web o trabalho sem proveito à vista, mas consequentemente também o sentimento de gratificação pelo que foi feito, que acaba por ser aplaudido? A verdade é que é preciso coragem, a mesma que é necessária quando tudo parece escurecer depois da ideia brilhante ter surgido mas já se encontra nublada por incertezas e medos de trabalho sem proveito e inútil. Que prevaleça então a consciência de que grandes projectos não necessitam de *turbos* nem *nitros*, mas apenas de uma mente sonhadora e um pequeno condensador.

Rapidamente continuámos, a alegria do já feito levando a melhor organização, e a revista prosperou com Sérgio Santos a coordenar. 9 Edições passaram-se e a renovação trouxe-me a coordenador. Acompanhando a revista e colaborando desde a 1ª edição devo dizer que passámos por maus bocados que até agora soubemos superar, não por sermos os melhores, muito profissionais ou sabedores, mas sim porque os nossos utilizadores, redactores e colaboradores assim o permitiram. E enquanto o fizerem, continuaremos.

2 Anos de Revista Programar, parabéns!

15ª Semana Informática

Os estudantes da Licenciatura em Engenharia Informática e de Computdores (LEIC) do Insituto Superior Técnico, em colaboração com o Departamento de Engenharia Informática (DEI) e o Núcleo Estudantil de Informática do I.S.T. (NEIIST), têm vindo a organizar anualmente, desde 1993, a Semana Informática do Instituto Superior Técnico. Este ano, a 15ª Semana Informática terá lugar de 10 a 14 de Março.

O evento tem como principais objectivos a divulgação e debate relativamente às notícias no mercado das Tecnologias de Informação que marcam a realidade nacional e internacional e as próprias TI, em geral, que estão cada vez mais presentes no nosso dia-a-dia.

A Semana Informática terá um ciclo de apresentação e debate, onde irão ser apresentados diversos projectos de pesquisa, desenvolvimento e Ciência de Computadores em geral. Uma exposição de tecnologia de computadores, onde novos produtos e tecnologias na área das TI estarão expostos, e o LEIC Jovem, um evento dedicado a estudantes do Ensino Secundário que procura divulgar informação sobre Ciência de Computadores e Sistemas de Informação, serão ainda outras atracções de relevo nesta Semana Informática.

Quanto aos oradores convidados, o evento deste ano contará com presença do professor Professor Andrew Tanenbaum, Raymond Chen (Microsoft) e Shiki Okasaka (Google).

Resta acrescentar que o evento é gratuito e aberto a todos os interessados.

PJ alerta compradores online

Para evitar que os casos de burla online continuem a aumentar, a Polícia Judiciária lançou um alerta público onde indica quais os principais cuidados que o consumidor deve ter quando compra através da Internet.

Tendo em conta que os utilizadores não têm contacto pessoal com os vendedores dos produtos, nem conseguem ver os bens que querem adquirir ao vivo, a PJ frisa que os consumidores deverão optar sempre por uma loja segura, certificada, recomendada ou que represente electronicamente um espaço comercial que exista fisicamente. Há ainda que verificar em todos os casos se o site possui os dados essenciais da loja para que, caso seja necessário, o utilizador entre em contacto com a marca.

As autoridades recomendam ainda que o utilizador digite sempre o endereço para o site respectivo ao invés de clicar numa ligação fornecida, evitando assim o reencaminhamento para destinos falsos ou indesejáveis.

Por fim, os consumidores devem evitar divulgar dados pessoais e de contas bancárias e deverão confirmar se o produto adquirido, quando entregue, corresponde ao que foi encomendado e se os valores cobrados estão certos.

Microsoft reduz preço do Windows Vista

Desde 1 de Março, o Windows Vista pode ser adquirido a um preço cerca de 40% inferior ao preço anterior. De forma a fomentar o uso e promover uma maior adesão dos utilizadores a este sistema operativo, a Microsoft reduziu o seu preço a nível mundial.

Esta redução afectou toda a gama de versões do produto para particulares - as edições Windows Vista Home Basic, Windows Vista Home Premium e Windows Ultimate, assim como a actualização Windows Vista SP1 introduzida no início de Fevereiro.

Assinaturas Digitais XML

Desde a sua normalização pelo W3C, a linguagem XML tem vindo a ser adoptada por um número crescente de produtores de software como formato base para os documentos utilizados pelas aplicações que desenvolvem. O crescendo de utilização de documentos neste formato revelou o interesse em definir mecanismos que lhes permitissem aportar as características de segurança (origem, não-repúdio e integridade) adequadas a cenários de utilização mais exigentes. Para colmatar essa lacuna, o W3C definiu posteriormente a norma XMLDSIG. Neste artigo vamos-nos debruçar essencialmente no desenvolvimento em C# de uma pequena API para assinar digitalmente documentos XML.

Introdução e Enquadramento

De acordo com o RFC2828 uma assinatura digital define-se como sendo um valor calculado com um algoritmo criptográfico e anexado ao objecto de dados de tal forma que se possa usar essa assinatura para verificar a autenticidade e integridade dos dados.

A forma de funcionamento de uma assinatura digital XML é extremamente fiável. Respeitando escrupulosamente a terceira regra fundamental da criptografia, a integridade, a assinatura não só assegura que a pessoa que assinou o documento é de facto quem se espera, como também que o mesmo se manteve inalterado em todo o seu percurso digital até ao momento em que chegou ao destinatário. Relembrando noções básicas de criptografia assimétrica, "uma mensagem cifrada com uma chave pública apenas pode ser decifrada com a correspondente chave privada".

Percebendo o W3C XML Digital Signature

A especificação XML é responsável pela definição da informação que é usada na verificação de certificados digitais. Assinaturas digitais em XML são representadas pelo elemento XML Signature que contém uma estrutura com as seguintes regras:

- * representa zero ou mais ocorrências de "algo".
- + representa uma ou mais ocorrências de "algo".
- ? representa zero ou uma ocorrência de "algo".

O seguinte XML descreve uma assinatura segundo a especificação W3C, de acordo com as regras descritas.

```
<Signature ID?>
  <SignedInfo>
   <CanonicalizationMethod/>
    <SignatureMethod/>
      (<Reference URI?>
        (<Transforms>)?
       <DigestMethod>
       <DigestValue>
      </Reference>)+
  </SignedInfo>
  <SignatureValue>
   (<KeyInfo>)?
   (Object ID?)*
</Signature>
```

Dissecando a especificação XMLDSIG

O elemento Signature é o construtor primário da assinatura, de acordo com a especificação W3C. A assinatura pode ser envelop ou enveloped pela informação que está a ser assinada, ou por outro lado pode referenciar informação que não consta do ficheiro XML, e neste caso será uma assinatura detached. As assinaturas detached são muito úteis para assinar ficheiros externos ao XML, relembro que não existe qualquer limitação sobre o que se está a assinar.

- Etapas de uma assinatura

- SignedInfo

O elemento SignedInfo representa a informação que está actualmente assinada. Esta informação é processada sequencialmente por várias etapas no processo de assinatura.

- Canonicalization

O elemento CanonicalizationMethod contém o algoritmo que foi usado para canonizar/estruturar a informação, numa forma comum "reconhecida" por todos os intervenientes no processo. Este processo é extremamente importante. A canonização pode definir regras como "a definição de um standard para o EOF, remoção de comentários" ou outra qualquer manipulação de um documento assinado que possamos necessitar.

- Reference/Transform

O elemento Reference identifica o recurso que vai ser assinado e os algoritmos para processar a informação. Estes algoritmos estão enunciados no elemento Transform e podem incluir operações de Canonização, encoding/decoding, compressão, ou até XPATH ou transformações XSLT.

O elemento Reference pode conter vários elementos Transform. Nota: O elemento Reference contém um atributo opcional URI. A inclusão de um URI numa assinatura é opcional se a assinatura possuir apenas um elemento Reference.

• DigestMethod/DigestValue

O elemento DigestMethod representa o algoritmo aplicado à informação após passar pelo processo de transformação, gerando um valor único representado pelo elemento DigestValue. O DigestValue é uma aplicação ao resultado dos processos de Canonização e transformação, sem qualquer referência directa à informação a ser assinada.

• KeyInfo

O elemento KeyInfo é opcional e pode conter a assinatura pública do autor para permitir a verificação da autenticidade do documento "automática".

- Tipos de assinaturas XMLDSIG

As assinaturas XMLDSIG podem ser aplicadas em três formas básicas:

• Detached: O documento XML a ser assinado e a assinatura estão em dois ficheiros distintos, sendo que esta tem uma referência (URI) ao documento que se propõe a assinar.

```
<signature> ... </signature>
```

• Enveloped: O documento XML e a assinatura surgem no mesmo ficheiro de uma forma sequencial.

```
<document>
   <signature>...</signature>
</document>
```

• Enveloping: O documento e a assinatura estão contidos num envelope XML.

```
<signature>
   <document>...</document>
</signature>
```

Relembrar Conceitos

Antes de focar a fase de implementação, é necessário rever alguns aspectos importantes da criptografia, bem como certos aspectos organizacionais presentes no sistema operativo Windows. Comecemos pela Criptografia...

Certificados Digitais

Um certificado digital é um arquivo de computador que contém um conjunto de informações referentes à entidade para o qual o certificado foi emitido (seja uma empresa, pessoa física ou computador), mais a chave pública referente e a chave privada que acredita-se ser de posse unicamente da entidade especificada no certificado.

- A anatomia de um certificado X.509

Um certificado padrão X.509 contém os seguintes campos:

• Versão - Contem a versão do certificado X.509, actualmente versão 3

• Número de série - Todo certificado possui um, não é globalmente único, mas único no âmbito de uma AC, ac LCRs usam o número de serie para apontar quais certificados se encontram revogados.

• Tipo de algoritmo - Contem um identificador do algoritmo criptográfico usado pela AC para assinar o certificado juntamente com o tipo de função de hash criptográfica usada no certificado

• Nome do titular - Nome da entidade para o qual o certificado foi emitido

• Nome do emitente - Autoridade Certificadora que emitiu/assinou o certificado

• Período de validade - Mostra o período de validade do certificado no formato "Não antes" e "Não depois" (Ex. "Não antes de 05/03/2006 - 14:35:02" "Não depois de 05/03/2007 - 14:03:20")

• Informações de chave pública da entidade

• Algoritmo de chave pública

• Chave pública

• Assinatura da AC - A garantia que a AC provê sobre a veracidade das informações contidas neste certificado de acordo com as políticas da AC

• Identificador da chave do titular - É uma extensão do X.509 que possui um identificador numérico para a chave pública contida neste certificado, especialmente útil para que programas de computador possam se referir a ela

• Identificador da chave do emitente - A mesma ideia mencionada anteriormente, só que se referindo a chave pública da AC que emitiu o certificado

• Atributos ou extensões - A vasta maioria dos certificados X.509 possui campos chamados extensões (OID) que provêem algumas

• Informações extra, como registos adicionais do titular e do emitente, especificações de propósito do certificado, etc.

Criptografia de chave pública

A criptografia de chave pública ou criptografia assimétrica é um método de criptografia que utiliza um par de chaves: uma chave pública e uma chave privada. A chave pública é distribuída livremente para todos os correspondentes via e-mail ou outras formas, enquanto a chave privada deve ser conhecida apenas pelo seu dono.

Num algoritmo de criptografia assimétrica, uma mensagem cifrada com a chave pública pode somente ser decifrada pela sua chave privada correspondente. Do mesmo modo, uma mensagem cifrada com a chave privada pode somente ser decifrada pela sua chave pública correspondente.

Os algoritmos de chave pública podem ser utilizados para autenticidade e confidencialidade. Para confidencialidade, a chave pública é usada para cifrar mensagens, com isso apenas o dono da chave privada pode decifrá-la. Para autenticidade, a chave privada é usada para cifrar mensagens, com isso garante-se que apenas o dono da chave privada poderia ter cifrado a mensagem que foi decifrada com a 'chave pública'.

RSA

RSA é um algoritmo de cifra de dados, que deve o seu nome a três professores do Instituto MIT (fundadores da actual empresa RSA Data Security, Inc.), Ron Rivest, Adi Shamir e Len Adleman, que inventaram este algoritmo — até à data (2005), a mais bem-sucedida implementação de sistemas de chaves assimétricas, e fundamenta-se em Teorias Clássicas dos Números. É considerado dos mais seguros. Foi também o primeiro algoritmo a possibilitar cifra e assinatura digital, é uma das grandes inovações em criptografia de chave pública.

KeyStore

Uma KeyStore é uma base de dados de chaves. As chaves privadas numa KeyStore possuem uma cadeia de certificados associada, que possibilita a autenticação à correspondente chave pública. Uma KeyStore também possui certificados de entidades confiáveis.

O Sistema operativo Windows possui uma KeyStore própria, onde são armazenados todos os certificados instalados. Existem também uma API embebida no Sistema Operativo que permite fazer a ponte entre SmarCards com certificados digitais, desde que o hardware possua um leitor de smartcards.

Programming, Programming, Program...

Requisitos

Para Prosseguir é necessário termos instalado no nosso computador uma licença do Windows XP ou superior, com a Framework .NET v2.0.

Ajuda possuírmos também uma licença do Visual Studio 2005, se bem que não é obrigatório pois existem alternativas OpenSource bastante completas na WWW.

Geração de um Certificado Digital

Vamos gerar um ficheiro PKCS#12. Um ficheiro PKCS12 é um formato que permite o armazenamento de chaves privadas juntamente com o certificado de chave pública, protegidos por uma palavra-chave. Este ficheiro servirá de base para os testes que vamos produzir ao longo do desenvolvimento. Não é obrigatória a geração deste ficheiro. Podemos criar em RunTime um certificado e exportar as respectivas chaves públicas e privadas. Este passo é apenas um exercício didáctico.

- Criação das chaves

```
makecert.exe -sv MyKey.pvk -n "CN=P@P"
MyKey.cer
```

Ser-vos-á pedida uma palavra-chave para a criação da chave privada.

- Criação do PKCS#12

pvk2pfx.exe -pvk MyKey.pvk -spc MyKey.cer -pfx cert.pfx -po pwd

Este aplicativo permite criar o ficheiro cert.pfx , protegido pela password pwd.

Classe KeyManager.cs

A classe KeyManager.cs servirá de ponte para a geração/carregamento de chaves publicas/privadas/Certificados, usando RSA. É bastante simples alterar a classe para podermos usar também o DSA, mas deixo isso à vossa curiosidade e "investigação".

É também possível interligar esta classe com a KeyStore do Windows ou simplesmente gerar uma chave pública/privada e exportá-las para XML.

```
using System;
using System.Collections.Generic;
using System.Text;

#region Crypt
using System.Security.Cryptography;
using
System.Security.Cryptography.Xml;
using
System.Security.Cryptography.X509Certif
icates;
#endregion

namespace AsynXmlCrypt
{
    public sealed class KeyManager
    {
        public static int KeySize =
1024;  // Tamanho da chave

        #region Private Properties
        private
RSACryptoServiceProvider m_rsa = null;
// Provider RSA
        #endregion

        #region Public Properties

        public
RSACryptoServiceProvider KeyProvider
// Propriedade que exporta o provider
        {
            get { return m_rsa; }
        }

        #endregion

        #region Constructors


        // Inicializa o Provider com
uma chave de tamanho igual Ãà KeySize
        public KeyManager(){
            m_rsa = new
RSACryptoServiceProvider(KeySize);
        }


        // Inicializa o Provider com
um ficheiro
        public KeyManager(string
filename)
        {
            m_rsa = new
RSACryptoServiceProvider(KeySize);
            _loadKey(m_rsa, filename);
        }

        // Inicializa o Provider com
um ficheiro XML de chave public e um
ficheiro XML de chave privada
        public KeyManager(string
publicKeyFilename, string
privateKeyFilename)
        {
            m_rsa = new
RSACryptoServiceProvider(KeySize);

LoadPublicKeyFromXmlFile(publicKeyFilen
ame);

LoadPrivateKeyFromXmlFile(privateKeyFil
ename);
        }

        // Inicializa o Provider com
um certificado
        public
KeyManager(X509Certificate2
certificate)
        {
            if (certificate == null)
                throw new
Exceptions.KeyManagerException("certifi
cate not initialized");
```

```
            if
(certificate.HasPrivateKey)
                m_rsa =
certificate.PrivateKey as
RSACryptoServiceProvider;
            else
                throw new
Exceptions.KeyManagerException("certifi
cate does not contains the PrivateKey
");
        }


        #endregion

        #region SAVE
        // Exporta a chave privada
para um ficheiro XML
        public void
SavePrivateKeyToXmlFile( string
filename, bool overwrite)
        {
            _saveKey(m_rsa,
filename,overwrite , true);
        }

        // Exporta a chave publica
para um ficheiro XML
        public void
SavePublicKeyToXmlFile(string
filename, bool overwrite)
        {
            _saveKey(m_rsa,
filename,overwrite , false);
        }

        private void
_saveKey(RSACryptoServiceProvider rsa,
string filename, bool overwrite, bool
privateKey){
            if (rsa == null)
                throw new
Exceptions.ServiceProviderException("Se
rvice Provider NULL Exception");

            if
(System.IO.File.Exists(filename) &&
!overwrite)
                return;

            string Key =
rsa.ToXmlString(privateKey);
            using
(System.IO.StreamWriter sw = new
System.IO.StreamWriter(filename,
false, Encoding.UTF8))
            {
                sw.WriteLine(Key);
                sw.Close();
            }
        }

        #endregion

        #region LOAD

        // Carrega uma chave privada de
um ficheiro XML
        public void
LoadPrivateKeyFromXmlFile(string
filename)
        {
            _loadKey(m_rsa, filename);
        }

        // Carrega uma chave publica de
um ficheiro XML
        public void
LoadPublicKeyFromXmlFile(string
filename)
        {
            _loadKey(m_rsa, filename);
        }

        private void
_loadKey(RSACryptoServiceProvider rsa,
string filename)
        {
            if (rsa == null)
                throw new
Exceptions.ServiceProviderException("Se
rvice Provider NULL Exception");

            if
(!System.IO.File.Exists(filename))
                throw new
System.IO.FileNotFoundException(filenam
e);

            using
(System.IO.StreamReader sr = new
System.IO.StreamReader(filename,
Encoding.UTF8))
            {
                string Key =
sr.ReadToEnd();
                sr.Close();

rsa.FromXmlString(Key);
            }
```

```
        }

        #endregion

    }
}
```

Utilizações da classe KeyManager.cs

- Gerando um par de chaves

O seguinte exemplo ilustra a forma de gerar um par de chaves (1024 bits) para posteriormente assinar documentos. As Chaves são posteriormente guardadas em ficheiros XML.

Normalmente não é muito usual guardar as chaves em ficheiros XML, mas nada impede a um utilizador, guardar e distribuir a sua chave pública, para assinatura e verificação.

```
using System;
using System.Collections.Generic;
using System.Text;

#region Crypt
using System.Security.Cryptography;
using
System.Security.Cryptography.Xml;
using
System.Security.Cryptography.X509Certif
icates;
using AsynXmlCrypt;

#endregion

static class Program{
        [STAThread]
        static void Main()
        {


AsynXmlCrypt.KeyManager.KeySize =
1024;
            AsynXmlCrypt.KeyManager km
= new AsynXmlCrypt.KeyManager();

km.SavePublicKeyToXmlFile(@"c:/Publica.
xml",true);

km.SavePrivateKeyToXmlFile(@"c:/Privada
.xml", true);
        }
}
```

Como se pode verificar pela análise do código fonte, auxiliados pela classe que construímos, a geração de um par de chaves passa a ser de codificação simples/trivial.

- Lendo um certificado PKCS#12

O seguinte exemplo ilustra a forma de carregar um certificado PKCS#12 para posteriormente assinar documentos. Podemos tentar carregar o certificado gerado previamente no ponto Geração de um Certificado Digital.

```
using System;
using System.Collections.Generic;
using System.Text;

#region Crypt
using System.Security.Cryptography;
using
System.Security.Cryptography.Xml;
using
System.Security.Cryptography.X509Certif
icates;
using AsynXmlCrypt;

#endregion

static class Program{
        [STAThread]
        static void Main()
        {
            AsynXmlCrypt.KeyManager km
= new AsynXmlCrypt.KeyManager(
                                        new
X509Certificate2 @"c:/cert.pfx",
"pwd")
            );
        }
}
```

- Lendo um certificado da KeyStore do Windows

O seguinte exemplo ilustra a forma de carregar um certificado da KeyStore do Windows para posteriormente assinar documentos. O Código é genérico e num caso real será necessário filtrar o certificado a extrair da KeyStore.

```
using System;
using System.Collections.Generic;
using System.Text;

#region Crypt
```

```
using System.Security.Cryptography;
using
System.Security.Cryptography.Xml;
using
System.Security.Cryptography.X509Certif
icates;
using AsynXmlCrypt;

#endregion

static class Program{
        [STAThread]
        static void Main()
        {

            // Store Current User
            X509Store store = new
X509Store(StoreName.My,
StoreLocation.CurrentUser);

store.Open(OpenFlags.ReadOnly |
OpenFlags.OpenExistingOnly);
            X509Certificate2 cc =
null;
            foreach (X509Certificate2
c in store.Certificates){
                // Vai buscar o
primeiro
                cc = c;
                break;
            }
            store.Close();

            //Carrega o par de chaves
do certificado.
            AsynXmlCrypt.KeyManager km
= new AsynXmlCrypt.KeyManager(cc);
        }
}
```

Numa aplicação "real", variantes deste código serão talvez aquelas que farão mais sentido de ser usadas já que normalmente pretendemos usar certificados que estejam instalados no nosso Sistema Operativo.

Assinar um documento XML

Vamos agora assinar um documento XML usando o certificado Cert.pxf, e recorrendo à classe KeyManager.cs .

```
using System;
using System.Collections.Generic;
using System.Text;

#region Crypt
using System.Security.Cryptography;
using
System.Security.Cryptography.Xml;
using
System.Security.Cryptography.X509Certif
icates;
using AsynXmlCrypt;

#endregion

static class Program{
        [STAThread]
        static void Main()
        {

            // Leitura do certificado
:)
            AsynXmlCrypt.KeyManager km
= new AsynXmlCrypt.KeyManager(
                                    new
X509Certificate2 @"c:/cert.pfx",
"pwd")
            );

            // Leitura de um documento
XML    (Test.XML)
            XmlDocument xmlDoc = new
XmlDocument();
            xmlDoc.PreserveWhitespace
= true;
            xmlDoc.Load("./test.xml");

            // Ok, vamos assinar o
documento
            SignXml(xmlDoc, km);

            // Gravar o documento.

xmlDoc.Save("./SignedTest.xml");
        }

       public static void
SignXml(XmlDocument Doc,
AsynXmlCrypt.KeyManager Key)
        {

            if (Doc == null)
                throw new
ArgumentException("Empty XML Document
Object ?");
```

```
            if (Key == null)
                throw new
ArgumentException("Empty KeyManager
Provider ?");

            SignedXml signedXml = new
SignedXml(Doc);

            // Passamos a chave :)
            signedXml.SigningKey =
Key.KeyProvider;

            // Criamos uma referencia
para a assinatura
            Reference reference = new
Reference();
            reference.Uri = "";

            // Adicionamos uma
transformação enveloped à referencia.

XmlDsigEnvelopedSignatureTransform env
= new
XmlDsigEnvelopedSignatureTransform();

reference.AddTransform(env);

            // Adicionamos a
referência ao objecto SignedXml.

signedXml.AddReference(reference);

            // Assinamos.

signedXml.ComputeSignature();

            // Extraimos a
representação da assinatura em XML
            XmlElement
xmlDigitalSignature =
signedXml.GetXml();

            // Juntamos a assinatura
XML ao documento.

Doc.DocumentElement.AppendChild(Doc.Imp
ortNode(xmlDigitalSignature, true));

            //Et Voilá :)

        }
}
```

Com este pedaço de código, criamos um documento SignedTest.xml com uma assinatura do tipo Enveloped. O resultado do ficheiro assinado será "algo" do género...

```
<Signature
xmlns="http://www.w3.org/2000/09/xmldsi
g#">
    <SignedInfo>
      <CanonicalizationMethod
Algorithm="http://www.w3.org/TR/2001/RE
C-xml-c14n-20010315" />
      <SignatureMethod
Algorithm="http://www.w3.org/2000/09/xm
ldsig#rsa-sha1" />
      <Reference URI="">
        <Transforms>
          <Transform
Algorithm="http://www.w3.org/2000/09/xm
ldsig#enveloped-signature" />
        </Transforms>
        <DigestMethod
Algorithm="http://www.w3.org/2000/09/xm
ldsig#sha1" />

<DigestValue>9H/rQr2Axe9hYTV2n/tCp+3UIQ
Q=</DigestValue>
      </Reference>
    </SignedInfo>

<SignatureValue>Mx4psIy9/UY+u8QBJRDrwQW
KRaCGz0WOVftyDzAe6WHAFSjMNr7qb2ojq9kdip
T8
Oub5q2OQ7mzdSLiiejkrO1VeqM/90yEIGI4En6K
EB6ArEzw+iq4N1wm6EptcyxXx
M9StAOOa9ilWYqR9Tfx3SW1urUIuKYgUitxsONi
UHBVaW6HeX51bsXoTF++4ZI+D
jiPBjN4HHmr0cbJ6BXk91S27ffZIfp1Qj5nL9on
FLUGbR6EFgu2luiRzQbPuM2tP
XxyI7GZ8AfHnRJK28ARvBC9oi+O1ej20S79CIV7
gdBxbLbFprozBHAwOEC57YgJc
x+YEjSjcO7SBIR1FiUA7pw==</SignatureValu
e>
</Signature>
```

A próxima etapa será a verificação/validação de um documento XML assinado.

Validar um documento assinado

A verificação de um documento previamente assinado é uma tarefa relativamente simples. O código fonte inicial (leitura do ficheiro, e chaves) é de alguma forma idêntico ao do ponto anterior.

```
using System;
using System.Collections.Generic;
using System.Text;

#region Crypt
using System.Security.Cryptography;
using
System.Security.Cryptography.Xml;
using
System.Security.Cryptography.X509Certif
icates;
using AsynXmlCrypt;

#endregion

static class Program{
        [STAThread]
        static void Main()
        {

            // Leitura do certificado
:)
            AsynXmlCrypt.KeyManager km
= new AsynXmlCrypt.KeyManager(
                                        new
X509Certificate2 @"c:/cert.pfx",
"pwd")
            );

            // Leitura de um documento
XML coma respectiva assinatura
(SignedTest.XML)
            XmlDocument xmlDoc = new
XmlDocument();
            xmlDoc.PreserveWhitespace
= true;

xmlDoc.Load("./SignedTest.xml");

            // Ok, vamos verificar a
validade do ficheiro XML
            bool
result=VerifyXml(xmlDoc, km);
            if (result)
             Console.Writeln("Ficheiro
Valido!");
            else
             Console.Writeln("Ficheiro
Invalido!");
        }

       public static void
VerifyXml(XmlDocument Doc,
AsynXmlCrypt.KeyManager Key)
        {
            if (Doc == null)
                throw new
ArgumentException("Empty XML Document
Object ?");
            if (Key == null)
                throw new
ArgumentException("Empty KeyManager
Provider ?");

            //
Instanciação/Inicialização
            SignedXml signedXml = new
SignedXml(Doc);

            //Neste ponto vamos
verificar quantos elementos
(ASSINATURAS) possui o ficheiro XML.
            XmlNodeList nodeList =
Doc.GetElementsByTagName("Signature");

            //O documento XML não
possui nenhuma assinatura!!
            //Decidimos lançar uma
excepção .
            if (nodeList.Count <= 0){
              throw new
CryptographicException("Failed: No Sig
was found in the document.");
            }

            //O documento XML possui
mais que uma assinatura!!
            //No contexto actual, não
possuimos KnowHow suficiente para
tratar esta variante,
            //pois queremos verificar
uma relação de 1 para 1, ou seja, um
documento assinado por uma chave.
            //Decidimos lançar uma
excepção .

            if (nodeList.Count >= 2){
               throw new
CryptographicException("Failed: More
that one sig was found for the
document.");
            }


            //Ok, se chegamos ate
aqui, então estamos prontos para
verificar a assinatura.
            //Carregamos o Elemento
XML que contém a assinatura para o
respectivo objecto
```

```
            //que faz a verificação.

signedXml.LoadXml((XmlElement)nodeList[
0]);

            //Retornamos o resultado
da verificação.
            return
signedXml.CheckSignature(Key.KeyProvide
r);
        }
}
```

O código fonte anterior é genérico e serve para verificar a validade de um documento XML, com uma assinatura (Detached, Enveloped ou Enveloping). Verificamos também que é necessário termos na nossa posse a chave pública (neste exemplo temos ambas, já que passamos o certificado), para verificar a validade do documento. Numa situação real, muitas das vezes não temos acesso à chave pública do autor e mesmo assim temos a necessidade de verificar a validade de um documento.

Fazendo um pequeno "upgrade" ao código de assinar, podemos incluir a nossa chave pública na assinatura, libertando o nosso código de verificação para uma versão mais "light" e eficiente.

- Upgrade da assinatura

```
public static void SignXml(XmlDocument
Doc, AsynXmlCrypt.KeyManager Key)
        {

            if (Doc == null)
                throw new
ArgumentException("Empty XML Document
Object ?");
            if (Key == null)
                throw new
ArgumentException("Empty KeyManager
Provider ?");

            SignedXml signedXml = new
SignedXml(Doc);

            // Passamos a chave :)
            signedXml.SigningKey =
Key.KeyProvider;

            // Criamos uma referencia
para a assinatura
            Reference reference = new
Reference();
            reference.Uri = "";

            // Adicionamos uma
transformação enveloped à referencia.

XmlDsigEnvelopedSignatureTransform env
= new
XmlDsigEnvelopedSignatureTransform();

reference.AddTransform(env);

            // Adicionamos a
referencia ao objecto SignedXml.

signedXml.AddReference(reference);

            //BEGIN UPGRADE

            //Inserimos a nossa chave
pública na própria assinatura para
posterior verificação.

            #region KeyInfo
            KeyInfo keyInfo = new
KeyInfo();
            keyInfo.AddClause(new
RSAKeyValue((RSA)Key.KeyProvider));
            signedXml.KeyInfo =
keyInfo;
            #endregion


            //END UPGRADE

            // Assinamos.

signedXml.ComputeSignature();

            // Extraimos a
representação da assinatura em XML
            XmlElement
xmlDigitalSignature =
signedXml.GetXml();

            // Juntamos a assinatura
XML ao documento.

Doc.DocumentElement.AppendChild(Doc.Imp
ortNode(xmlDigitalSignature, true));

            //Et Voilá :)

        }
```

- Upgrade da função de verificação

```
public static void
VerifyXml(XmlDocument Doc)
        {

            if (Doc == null)
                throw new
ArgumentException("Empty XML Document
Object ?");
            if (Key == null)
                throw new
ArgumentException("Empty KeyManager
Provider ?");

            //
Instanciação/Inicialização
            SignedXml signedXml = new
SignedXml(Doc);

            //Neste ponto vamos
verificar quantos elementos
(ASSINATURAS) possui o ficheiro XML.
            XmlNodeList nodeList =
Doc.GetElementsByTagName("Signature");

            //O documento XML não
possui nenhuma assinatura!!
            //Decidimos lançar uma
excepção .
            if (nodeList.Count <= 0){
              throw new
CryptographicException("Failed: No Sig
was found in the document.");
            }

            //O documento XML possui
mais que uma assinatura!!
            //No contexto actual, não
possuimos KnowHow suficiente para
tratar esta variante,
            //pois queremos verificar
uma relação de 1 para 1, ou seja, um
documento assinado por uma chave.
            //Decidimos lançar uma
excepção.

            if (nodeList.Count >= 2){
               throw new
CryptographicException("Failed: More
that one sig was found for the
document.");
            }

            //Ok, se chegamos ate
aqui, então estamos prontos para
verificar a assinatura.
            //Carregamos o Elemento
XML que contÃém a assinatura para o
respectivo objecto que
            //faz a verficação.

signedXml.LoadXml((XmlElement)nodeList[
0]);

            //BEGIN UPGRADE

            //Tentamos extrair a chave
publica da assinatura embebida.
            //Se não conseguirmos
então não podemos atestar a
            //validade do documento
retornando FALSO.

            KeyInfo keyInfo =
signedXml.KeyInfo;
            if (keyInfo.Count == 0)
                return false;

            //END UPGRADE

            //Retornamos o resultado
da verificação com a assinatura
embebida.
            return
signedXml.CheckSignature();
        }
```

Assim, e com base neste upgrade, não necessitamos de passar nenhuma chave pública para verificar a validade de um certo documento XML assinado, como podemos verificar no próximo pedaço de código:

```
static void Main()
        {
            // Leitura de um documento
XML coma respectiva assinatura
(SignedTest.XML)
            XmlDocument xmlDoc = new
XmlDocument();
            xmlDoc.PreserveWhitespace
= true;

xmlDoc.Load("./SignedTest.xml");

            // Ok, vamos verificar a
```

```
validade do ficheiro XML
            bool
result=VerifyXml(xmlDoc);
            if (result)
              Console.Writeln("Ficheiro
Valido!");
            else
              Console.Writeln("Ficheiro
Invalido!");
        }
```

Nota: Como este documento é um artigo didáctico, não estou a fazer o devido tratamento das excepções que podem ocorrer no funcionamento do software, o código aqui desenvolvido não está pronto para deploy em ambientes de produção como devem todos perceber.

Reflexões...

Tenham em atenção que a assinatura refere-se ao documento XML, assim se alterarmos o conteúdo do documento XML a verificação falhará. Mas se alterarmos o Elemento Signature; Incluirmos dados por exemplo sem alterar o conteúdo XML que assinamos, o processo de verificação e validação do documento funcionará correctamente.

Perguntam os leitores: Mas esta "falha" não invalida de certa forma a autenticidade do documento? Não, porque assinamos um documento (XML DATA). Além de que, não é uma falha, podemos "corrigir" esta "distracção" adicionando elementos ao objecto Signature que verifiquem a autenticidade da própria assinatura.

Este artigo, embora exiga conceitos mais avançados de criptografia, é de certa forma introdutório e fornece aos leitores boas bases para investigação e desenvolvimento de novas funcionalidades tais como : Múltiplas Assinaturas num documento, assinaturas com TimeStamp, Assinar apenas certos elementos XML, Cifrar/Decifrar Documentos/elementos XML baseado na nossa chave pública/privada, etc.

Conclusões

O XMLDSIG é de grande utilidade para a indústria, a sua aplicação passa por as mais variadas áreas. Podemos integrar esta tecnologia com as mais diversas áreas, tais como, WebServices para garantir a fiabilidade da transmissão de dados, podemos também usa-la para criar sistemas de protecção de software, aplica-la na troca de dados entre diferentes entidades que exijam garantias de autenticidade e não repudio da informação(O SAFT-PT por exemplo), Facturação electrónica, etc.

As aplicações são inúmeras, e tudo depende da imaginação e criatividade de quem desenvolve, integra e arquitecta sistemas.

Bibliografia

- http://www.w3.org/TR/xmldsig-core/
- http://msdn2.microsoft.com
- http://www.xml.com/pub/a/2001/08/08/xmldsig.html

SOBRE O AUTOR

Paulo Cabral é Engenheiro de Sistemas e Informática.
Tem 10 anos de experiência na área, estando actualmente inserido num projecto de I&D numa instituição de ensino superior.

Paulo Cabral

Introdução ao SOAP

O SOAP (Simple Object Access Protocol) é um protocolo concebido para trocar informação estruturada num ambiente descentralizado. Usa as tecnologias de XML para definir uma estrutura de mensagens extensíveis que podem ser trocadas sobre uma variedade de protocolos subjacentes. A estrutura foi projectada para ser independente de qualquer linguagem de programação, modelo ou outra implementação específica. O SOAP habilita aplicações-cliente a ligarem-se a serviços remotos e invocarem métodos desses serviços. Assim, uma aplicação--cliente pode adicionar um serviço disponibilizado na Web, ao seu feature set, localizar o serviço SOAP apropriado e invocar o método correcto.

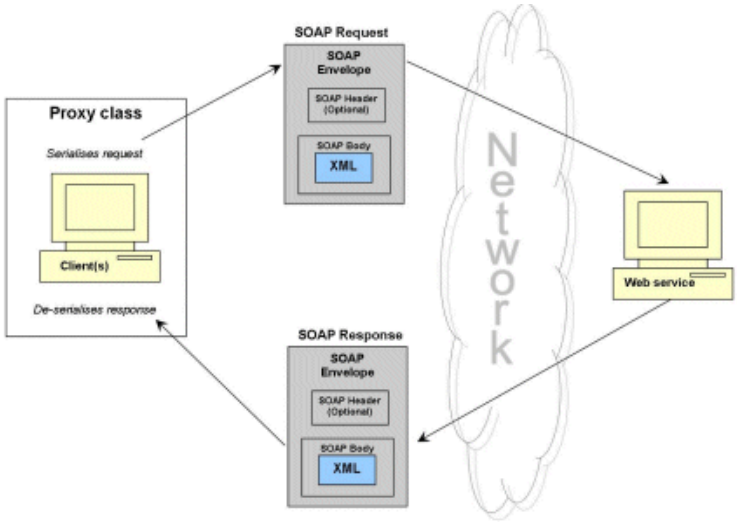
Ao contrário de outras arquitecturas distribuídas, como o COM/COM+ e o CORBA, o SOAP é meramente um protocolo de comunicação. Para conseguirmos comparar com essas arquitecturas temos que implementar o protocolo SOAP como uma arquitectura distribuída.

O SOAP oferece:

- Mecanismo para definir a unidade de comunicação,
- Mecanismo para lidar com erros,
- Mecanismo de extensão que permite evolução,
- Mecanismo entre as mensagens SOAP e o HTTP, representar tipos de dados em XML.

O SOAP pode ser usado numa variedade de sistemas de mensagens e protocolos. As mensagens SOAP são escritas inteiramente em XML e, portanto, são independentes das linguagens e plataformas. Desta forma, um cliente Java usando SOAP, correndo em Solaris, ou um cliente Perl usando SOAP correndo em Linux, podem ligar-se a um servidor Windows 2000 executando SOAP. O protocolo SOAP representa a essência da arquitectura de serviços na Web, habilitando diversas aplicações a trocarem dados e serviços.

Pode ver na figura o pedido SOAP. Este pedido é encapsulado dentro de um envelope SOAP (que por sua vez é encapsulado dentro do pedido HTTP, assumindo que estamos a usar HTTP como transporte). O envelope SOAP é dividido em duas partes, um Header (cabeçalho) e um Body (corpo). O SOAP Header aparece imediatamente a seguir à tag XML que abre o envelope, mas o seu uso é opcional.



É provável também, caso o Header exista, que se encontre um ou mais Headers que nos vão fornecendo meta-informação especifica da aplicação. Essa meta-informação pode conter qualquer coisa, embora a especificação do SOAP use o ID da transacção como exemplo.

O SOAP Header:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
<m:Trans
xmlns:m="http://www.mysite.com/transaction/"
soap:mustUnderstand="1">234</m:Trans>
</soap:Header>
...
...
</soap:Envelope>
```

O SOAP Body contém os argumentos e os nomes dos métodos remotos a serem chamados.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body>
   <m:GetPrice
```

```
xmlns:m="http://www.mysite.com/prices">

      <m:Item>Apples</m:Item>
   </m:GetPrice>
</soap:Body>
</soap:Envelope>
```

Toda a informação é serealizada usando XML.. Embora o SOAP use XML, que é formato de texto, podem ser usadas estruturas, unions e serem passados parâmetros por referência.

A sintaxe do SOAP

A sintaxe do SOAP é muito simples visto usar como base o XML. As regras mais importantes são:

•As mensagens SOAP têm que usar o XML
•Todas as mensagens têm que ter as tags de envelope e de enconding
•As mensagens não podem ter referências DTD
•As mensagens não podem ter instruções de processamento de XML

Abaixo podem ver o quão simples é a estrutura de uma mensagem SOAP:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
  ...
  ...
</soap:Header>
<soap:Body>
  ...
  ...
  <soap:Fault>
    ...
    ...
  </soap:Fault>
</soap:Body>
</soap:Envelope>
```

SOAP e a performance

Devido ao uso de XML, ao contrário de outros protocolos, a performance do SOAP é um pouco afectada pois existe a necessidade de extrair o envelope SOAP e efectuar o parsing do XML.

Exemplo de uma transacção SOAP

Neste exemplo temos uma aplicação que efectua um pedido ao servidor do preço de um artigo (GetStockPrice) passando o nome do artigo no parâmetro StockName. Na resposta é enviado o preço do artigo no parâmetro Price.

O pedido SOAP:

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml;
charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body
xmlns:m="http://www.example.org/stock">

    <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

A resposta SOAP:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml;
charset=utf-8
Content-Length: nnn
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body
xmlns:m="http://www.example.org/stock">

    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>
</soap:Envelope>
```

Conclusão

Com este artigo pretendeu-se que o leitor fique a perceber, de uma forma superficial, o funcionamento do SOAP. O SOAP não especifica necessidades da informação a ser trocada – isso cabe às aplicações que o usam.

O SOAP é assim uma plataforma sobre a qual informação específica de aplicações pode ser trocada. É um standard que permite grande expansibilidade e interoperabilidade entre sistemas. Esta interoperabilidade poderá naturalmente estar ameaçada se não forem utilizados standards puros SOAP. Nem todos os aspectos foram tratados neste artigo, pelo que se aconselha a pesquisar sobre este assunto.

Ver também...

Como forma de começar a explorar o mundo SOAP é aconselhada a visualização da palestra realizada no SAPO CodeBits, disponível no serviço SAPO Vídeos em http://videos.sapo.pt/vTPQgrAU7b7wcBDilQJf

Bibliografia

Kennard Scribner e Mark C. Stiver. Understanding SOAP. Sams Publishing.

SOBRE O AUTOR

Residente no Porto, João Brandão tem 9 anos de experiência em desenvolvimento de software e em montagem e elaboração de redes de computadores. Ocupa desde Dezembro de 2005 o cargo de Senior Engineer no Software Development Center da Qimonda AG. Tem como principais actividades e responsabilidades o desenvolvimento software e gestão de projectos numa grande variedade de aplicações na área web e não web.

João Brandão

Programação em Lógica com PROLOG

PROLOG é uma linguagem de programação baseada na lógica simbólica, servindo como ferramenta para o desenvolvimento de aplicações na área da Inteligência Artificial.

A linguagem PROLOG é utilizada em problemas que envolvam objectos e relações entre eles. É de fácil dedução que PROLOG vem de PROgramação em LÓGica. Em lógica define-se um Teorema e para verificar a sua validade, procura-se regras e factos. Comparativamente à lógica, em PROLOG exprime-se factos e relações entre factos, para inferir soluções para os problemas.

Alguns conceitos

Um facto expressa uma verdade sobre uma relação. Um facto é uma relação entre objectos da forma:

```
relação(objecto1, ... ,objecton)
```

em que 'objecto1, ... , objecton', são os argumentos. O nome da relação é chamado predicado. Uma relação é um conjunto de n-tuplos de objectos.

Uma regra expressa uma relação entre factos. As regras são usadas quando se deseja dizer que um facto depende de uma conjunção de outros factos, que têm a forma:

```
cabeça :- corpo.
```

em que o conector ':-' significa "se".

O conjunto de regras e de factos designa-se por base de dados, ou base de conhecimento. Os objectos manipulados em PROLOG são chamados termos, que podem ser átomos ou estruturas. Os átomos poderão ser uma de duas coisas: constantes ou variáveis. As constantes podem ser: identificadores, que começam por letras minúsculas; Strings entre plicas; ou números inteiros.

Ao contrário de outras linguagens, uma variável em PROLOG, não representa uma posição de memória, mas sim uma associação a um objecto. Começam por letra maiúscula ou por "_", representando um objecto que não é possível de nomear.

As estruturas são representadas por:

```
simbolo_funcional(argumentos)
```

em que o 'simbolo_funcional' é um identificador, e os argumentos são uma lista de termos, que pode ser vazia.

Após concluída a base de conhecimento do programa em PROLOG, esta poderá ser consultada. Uma consulta é uma interrogação ao sistema, da forma:

```
?-predicado(argumentos).
```

A resposta dada a uma consulta, depende da forma como é feita a pergunta. Se esta for fechada, ou seja, se não contiver variáveis nos argumentos dos predicados, a resposta do sistema será da forma yes/no.

Se a pergunta for aberta, ou seja, se contiver variáveis nos argumentos dos predicados, serão listados todos os valores, que unificam com as variáveis, formando um facto verdadeiro. A primeira resposta do sistema, corresponde à primeira unificação das variáveis, que forma um facto verdadeiro. Para pedir ao sistema que mostre outras respostas possíveis, utiliza-se o simbolo ';' que significa "ou". O símbolo ';' pede ao PROLOG que procure na base de conhecimento, a partir da última solução retornada, a próxima unificação que satisfaça a consulta (processo designado de "back-tracking"). Quando não houver mais unificações verdadeiras para mostar, responde no.

A unificação corresponde a uma substituição que instância um termo, ou seja, é uma operação que atribui um valor a um termo. Para a pesquisa de uma resposta a uma consulta, o interpretador de PROLOG, utiliza uma árvore de prova, que "é uma entidade perfeitamente definida que constitui, por si só, uma prova, intuitivamente evidente, que um átomo é consequência lógica de um programa".

O interpretador de PROLOG percorre a árvore de prova com avanços e retrocessos ("back-tracking"), "cada avanço está associado a uma transformação (alvo e instanciação) e cada retrocesso está associado à anulação dessa transformação". Será apresentado um exemplo de uma árvore de prova, quando for abordado o tema das listas em PROLOG.

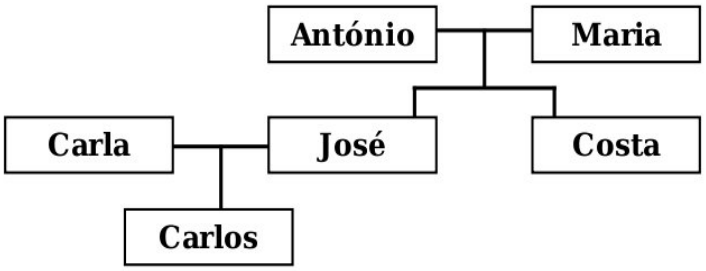
Um primeiro exemplo

Um dos primeiros exemplos e dos mais interessantes, para quem se inicia na Programação em Lógica é fazer a sua árvore genealógica.

```
% A base de conhecimento

pai(antonio,jose).
pai(antonio,costa).
pai(jose,carlos).
mae(maria,jose).
mae(maria,costa).
mae(carla,carlos).
casado(antonio,maria).
casado(jose,carla).
homem(antonio).
homem(jose).
homem(costa).
homem(carlos).
mulher(maria).
mulher(carla).

% Fim da base de conhecimento
```



O facto 'pai(antónio,josé).', lê-se: o indivíduo antónio é pai do indivíduo josé. Para manipular a base de conhecimento anterior, pode escrever-se, entre outros, os seguintes predicados (regras):

```
progenitor(P,F) :- pai(P,F).
progenitor(P,F) :- mae(P,F).

avo(A,N) :-
    progenitor(A,F),
    progenitor(F,N).

bisavo(BV,BN) :-
    avo(BV,N),
    progenitor(N,BN).

irmao(X,Y):-
    homem(X),
    progenitor(P,X),
    progenitor(P,Y),
    X \= Y.

irmaos(X,Y) :-
    progenitor(P,X),
    progenitor(P,Y),
    X \= Y.

tio(X,Y) :-
    irmao(X,I),
    progenitor(I,Y).

primo(A,B) :-
    progenitor(P1,A),
    progenitor(P2,B),
    irmao(P1,P2).

sogro(Q,W) :-
    pai(Q,A),
    casado(A,W).

cunhado(A,B) :-
    irmaos(A,I),
    casado(I,B).
```

Listas

Uma lista é uma estrutura recursiva binária, cujos elementos podem ser átomos ou termos estruturados, inclusive listas. Uma lista ou é vazia, não contém nenhum elemento, ou é uma estrutura com dois elementos: a cabeça e a cauda. Uma possível representação de uma lista será: [H|T], em que H representa a cabeça da lista e T a cauda da lista. As listas permitem agilizar bastante a resolução de diversos problemas e sobre elas podem ser escritos os mais diversos predicados:

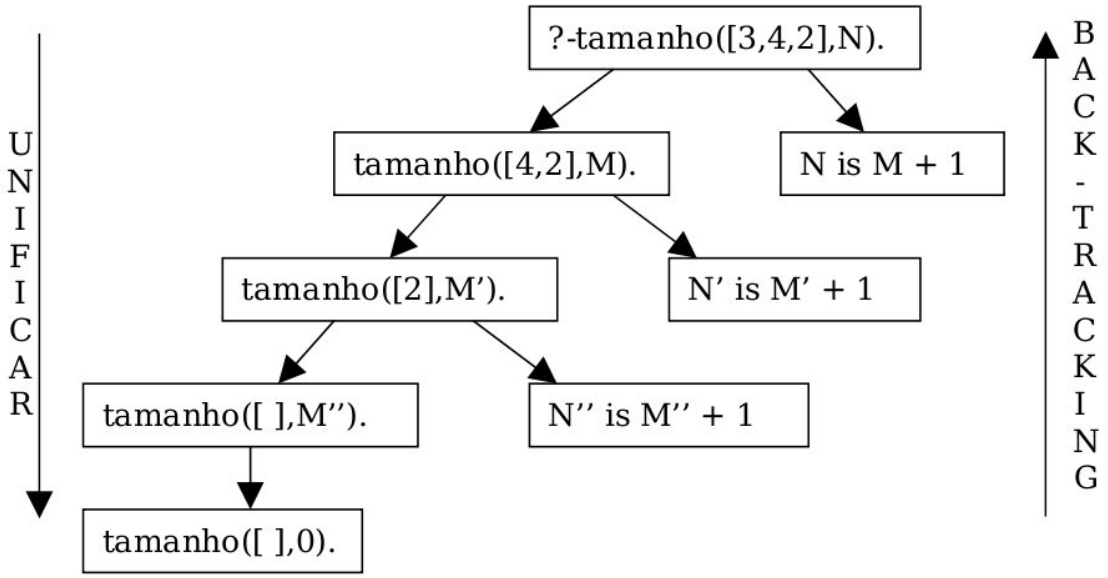
```
%Verificar se uma estrutura é uma lista
lista([]).
lista([_H|T]) :- lista(T).

%Calcular o tamanho de uma lista
tamanho([],0).
tamanho([_H|T], N) :-
    tamanho(T,M),
    N is M+1.

%Verificar se um elemento pertence a uma lista
membro(H,[H|_T]).
membro(X,[_H|T]) :-
    membro(X,T).

%Concatenar duas listas:
%[2][3,1] => [2,3,1]
concatena([],L,L).
concatena([H|T],L,[H|L1]) :-
    concatena(T,L,L1).

%Inverter uma lista:
% [1,2] => [2,1]
inverte([],[]).
inverte([H|T], L) :-
```




```
inverte(T,T1),
concatena(T1,[H],L).
```

Com as Listas, é possível definir outras estruturas de manipulação de dados, como por exemplo, uma pilha ("Stack"). Uma Stack poderá ser vista como uma estrutura do género:

Stack = [topo, ..., base]

```
%Empilhar um elemento – pop
pop(X,[],[X]).
pop(X,P,[X|P]).

%Desempilhar o elemento do Topo da Stack - push
push([],[]) :- write('Stack vazia!'), !.
push([_|T],T).

%Mostar o elemento do topo
mostra([]) :- write('Stack vazia!'), !.
mostra([X|_]) :- write('Elemento do topo: '), write(X).

%Tamanho da Stack
%Resume-se a calcular o tamanho de uma lista
```

Manipulação de Listas

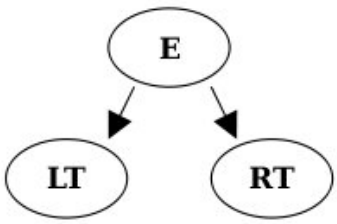
No caso de programas que têm de lidar com bases de dados consideravelmente grandes, torna-se útil a utilização de predicados que permitam pesquisar informação na base de dados, destacando-se o predicado findall(?Termos,?Predicados,?Lista), que retorna em Lista, a lista de todos os Termos que estão de acordo com os Predicados, e o predicado setof(?Termo,?Predicados,?Lista), que tem um

Árvores binárias

Outra estrutura também muito usada em PROLOG, são as árvores binárias. A árvore binária vazia poderá ser representada pela palavra void. Uma árvore binária não vazia será representada da seguinte forma:

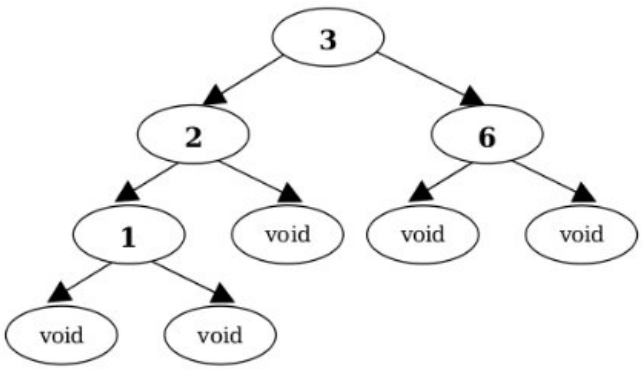
tree(E, LT, RT)

em que E representa o elemento do nó da árvore, LT a sub-árvore esquerda e RT a sub-árvore direita.



Exemplo de uma árvore binária:

tree(3, tree(2, tree(1, void, void), void), tree(6, void, void)).



Também sobre esta estrutura, podem ser escritos diversos predicados:

```
    %Verificar se uma estrutura é uma
árvore binária
    binaryTree(void).
    binaryTree(tree(_E, LT, RT)) :-
        binaryTree(LT),
        binaryTree(RT).

    %Verificar se um elemento pertence à
árvore
    treeMember(X,tree(X,_LT,_RT)).
    treeMember(X,tree(_,LT,_)) :-
        treeMember(X,LT).
    treeMember(X,tree(_,_,RT)) :-
        treeMember(X,RT).

    %Multiplicar todos os elementos da
árvore
    multiplica(void, 1).
    multiplica(tree(E, LT, RT),N) :-
        multiplica(LT, M),
        multiplica(RT, P),
        N is E * M * P.

    %Passar uma árvore para uma lista em
pré-ordem
    preOrdem(void,[]).
    preOrdem(tree(E, LT, RT), L) :-
        preOrdem(LT, L1),
        preOrdem(RT, L2),
        concatena([E|L1], L2, L).


    %Substituir um elemento de uma árvore
binária por outro elemento
    subs(_X,_Y,void,void).
    subs(X,Y,tree(X,LT,RT),tree(Y,A,B)):-
            subs(X,Y,LT,A),
            subs(X,Y,RT,B).
    subs(X,Y,tree(Z,LT,RT),tree(Z,A,B)):-
            subs(X,Y,LT,A),
            subs(X,Y,RT,B).
```

Cut

O operador "cut", que se denota por !, inÿuencia o percurso de pesquisa na árvore de procura. Quando utilizado, numa regra ou consulta, modiþca a árvore de procura, eliminando as uniþcações realizadas antes de ser activado. Ou seja, elimina todos os ramos da árvore de procura, que ainda não foram percorridos e que estão situados à direita do último nó uniþcado, inclusive, não sendo percorridos aquando do "back-tracking". O operador "cut", pode ser utilizado em diversas situações:

• Exclusão mútua incondicional, em que só é dada uma resposta a cada consulta,

```
    cabeça:-!,objectivo2,...,objectivon.
```

• Exclusão mútua condicional,

```
    cabeça:-objectivo1,...,!,..., objectivon.
```

• Resposta única,

```
    cabeça:-objectivo1,...,objectivon, !
```

Exemplos:

```
    %Implementação de um IF, à imagem de
C ou JAVA
    if_then_else(C,_T,E):- C, !, E.
    if_then_else(_C,_T,E):- E.

    membroComCut(H,[H|_T]):-!.
    membroComCut(X,[_H|T]):-
        membroComCut(X,T).


    %Exemplo de consultas, para mostrar a
diferença de resultados
    ?-membro(X,[2,4,6]).
        X=2;
        X=4;
        X=6;
        no

    ?-membroComCut(X,[2,4,6]).
        X=2
        yes
```

Ciclos em PROLOG

Embora a linguagem PROLOG seja totalmente diferente de linguagens como C ou JAVA, é igualmente possível construir ciclos em PROLOG, usando o predicado repeat, que já se encontra pré-definido no PROLOG, da seguinte forma:

```
repeat.
repeat:-repeat.
```

Associado ao repeat pode-se ter o predicado fail, que falha sempre. Um ciclo repeat-fail, executa um predicado P até que uma condição X seja verdadeira. O predicado fail permite forçar o "back-tracking". Como usar o repeat e o fail? Eis um exemplo:

```
%Um menu simples
menu:-
    repeat,
    write('1 - Opção A. '),nl,
    write('2 - Opção B. '),nl,
    write('3 - Opção C. '),nl,nl,
    write('Introduza a opção: '),
    read(Opcao),
    executa(Opcao),
    fail.
```

O predicado repeat permite chamadas iterativas ao write e ao read, até que a condição de saída executa(X), tenha êxito.

Ciclo FOR

```
for(X,[X,Z]) :- X =< Z.
    for(X,[Y,Z]) :-
        W is Y+1,
        W =< Z,
        for(X,[W,Z]).

    ?- for(X,[1,3]), write(X), write('
'), fail.
        1 2 3
        yes
```

Negação Lógica

Combinando o operador "cut" e o predicado fail, implementa-se a negação lógica,

```
not(A) :- A, !, fail.
not(A).
```

Se A tiver êxito, então not(A) falha; se A falha, então not(A) tem êxito.

Manipulação de Base de Dados

Os predicados seguintes permitem adicionar ou remover factos de uma base de dados PROLOG.

• asserta(facto) -> adiciona o facto no ínicio da lista de factos

• assertz(facto) -> adiciona o facto no fim da lista de factos

• retract(facto) -> remove o facto da base de dados

Manipulação de Termos

• var(X) -> Sucede se X é uma variável

• novar(X) -> Sucede se X não é uma variável

• atom(X) -> Sucede se X representa um átomo PROLOG

• integer(X) -> Sucede se X é um inteiro

• Estrutura=..Lista -> Converte um elemento composto numa lista, e vice-versa.

Exemplo:

```
f(a,b)=..L resulta em L=[f,a,b]
L=..[f,a,b] resulta em L=f(a,b)
```

• arg(Argumento,Termo,Valor)-> Retorna em Valor o Argumento do Termo.

• functor(Termo,Nome,Aridade)-> Retorna em Nome o nome do Termo, e em Aridade o número de argumentos do Termo.

Depuração

Inicialmente, para compreender como funciona o motor de inferência do PROLOG e para se ter acesso ao "raciocínio" do PROLOG, podem-se usar os seguintes predicados:

trace -> Activa a depuração
notrace -> Desactiva a depuração

Exemplo:

```
trace,
%Chama o predicado 'lista'
lista([1,2,3]),
notrace.
```

Acesso a ficheiros

A maioria das vezes, quando se faz um programa, é desejável guardar a informação gerada pela execução desse programa, para utilização futura. Em PROLOG, ler e guardar informação em ficheiros é algo bastante simples, mas que requer algum cuidado.

```
%see abre o ficheiro para leitura
see('nomeFicheiro.pl'),
read(X),
%seen fecha o ficheiro
seen.
```

Após aberto um ficheiro, para escrita, se ocorrer algum erro de execução enquanto o ficheiro não for fechado, perder-se-á todo o conteúdo do mesmo, pelo que é aconselhável manter sempre actualizado, um backup do ficheiro.

Caso se queira ler para a memória, uma base de dados contida num ficheiro, bastaria fazer:

```
:- initialization(iniciar).
iniciar:- [nomeFicheiro].
```

Para escrever dados num ficheiro, bastará fazer:

```
%tell abre o ficheiro para escrita
tell('nomeFicheiro.pl'),
write('qualque coisa'),nl,
%told fecha o ficheiro
told.
```

Penso que neste artigo foram abordados os tópicos essenciais ao início do desenvolvimento de programas em PROLOG. Embora seja uma linguagem com sintaxe relativamente fácil, requer muita prática e bastante raciocínio.

```
fim:-
        write('Muita Programação e muita Diversão!!!'),nl,
        write('FIM').
```

Bibliografia

• Sterling, Leon; Shapiro, Ehud; The Art of Prolog; Advanced Programming Techniques; 1994

SOBRE O AUTOR

Estudante na Universidade de Évora, no curso de Engenharia Informática, Gaspar Brogueira dedica o seu tempo ao aperfeiçoamento dos seus conhecimentos nas linguagens de programação C e JAVA. Tem especial interesse pelas áreas da Segurança e Criptograÿa.

Gaspar Brogueira

Algoritmia Clássica em C++

Breve Introdução Histórica

A linguagem C++ foi desenvolvida durante os anos 80 na Bell Labs, pelo cientista de computação dinamarquês Bjarne Stroustrup. Esta linguagem é muitas vezes retratada como uma evolução da linguagem C. De facto, esta linguagem foi a principal base de desenvolvimento de C++, tanto mais que a primeira versão da nova linguagem tinha o nome de C With Classes, evoluindo mais tarde para C++. Em português deve-se pronunciar "cê mais mais" (sêmâysmâys) sendo que em inglês esta linguagem é pronunciada como "cee plus plus" (siplâsplâs).

As vantagens introduzidas pelo C++ face ao C são indiscutíveis. No entanto, e muitas vezes devido ao facto de ser uma linguagem com facilidades de alto nível, ocorrem erros lógicos (os temidos bugs) difíceis de resolver e frequentemente com consequências trágicas para o computador que está a correr a aplicação. São por isso famosas as duas citações de Stroustrup acerca das "facilidades" da sua nova linguagem:

• "C faz com que dar um tiro no pé seja fácil; C++ torna isso mais difícil, mas quando nós o fazemos rebenta com a perna toda."

• "Sempre desejei que o meu computador fosse tão fácil de usar como o meu telefone. O meu desejo realizou-se. Já não sei usar o meu telefone."

Nem sempre é simples para iniciantes da linguagem ou programadores inexperientes evitar que estes erros aconteçam. É por esta razão que muitas vezes é recomendado um período de adaptação à sintaxe de C e desenvolvimento de aplicações nesta linguagem, antes de se dar o salto para o C++.

Algoritmos Clássicos - Ordenação

Existe um alargado conjunto de algoritmos informáticos que desempenham um papel quase vital no desenvolvimento de aplicações coerentes e sólidas. Estes algoritmos, conhecidos como algoritmos clássicos, são frequentemente usados como "sub-rotinas" em aplicações mais complexas.

Um grupo de algoritmos clássicos que apresenta um vasto domínio de aplicação é o dos algoritmos de ordenação. Este grupo de algoritmos apresenta variadas técnicas que possibilitam a ordenação dos elementos de um array segundo uma determinada lógica (pode-se proceder a uma ordenação de forma ascendente ou descendente).

Neste artigo serão apenas abordados dois dos muitos algoritmos de ordenação de arrays: o quicksort (por muitos considerado o melhor algoritmo de ordenação) e o bubblesort.

- BubbleSort

O algoritmo BubbleSort é talvez o algoritmo mais simples de se perceber dentro do género. A sintaxe que este algoritmo apresenta não necessita de explicações exaustivas, visto recorrer a aspectos básicos de programação (ciclos for/while, comparação de diferentes elementos do array, incrementação de uma variável, etc.). O próprio funcionamento do BubbleSort é muito simples: por cada iteração do ciclo, o maior elemento do array (para um determinado intervalo de elementos desse mesmo array) é colocado no índice final previamente estabelecido. Isto é conseguido à custa de sucessivas comparações de um elemento do array com o seu elemento seguinte.

O seguinte excerto de código ilustra a implementação do Bubblesort:

```
//função para trocar dois elementos do
array (utilização de referências para
afectação directa)
void swap (int& n1, int& n2)
{
      //guardar o valor de n2
      int aux = n2;
      //afectar n2 com o valor de n1
      n2 = n1;
      //afectar n1 com o valor de aux
(valor inicial de n2)
      n1 = aux;
}

//a função recebe como argumentos o
array a ordenar e o numero máximo de
elementos a ordenar
void bsort (int v[], int n)
{
      for (int i = n; i > 0; --i)
//ciclo responsável pelas
```

```
decrementações do índice máximo
      {
          for (int j = 0; j < i; ++j)
//ciclo responsável por incrementar os
índices a ordenar
          {
              if (v[j] > v[j+1]) //se
um elemento é maior que o seu elemento
seguinte
              {
                 swap (v[j], v[j+1]);
//chamar a função para trocar dois
elementos do array
              }
          }
      }
}
```

O primeiro ciclo for é o responsável por decrementar o índice máximo para as ordenações. Assim, se se quiser ordenar um array tomando em conta todos os seus elementos, terá de se iniciar a variável que irá percorrer todos os índices com o valor do tamanho do array. Depois de serem efectuadas todas as operações uma primeira vez, tem-se a certeza que o maior elemento do array se encontra no índice final. Na segunda vez que se irão realizar as operações de ordenação, não interessa que se tenha em conta todo o array (se assim fosse estariam a ser feitas comparações desnecessária, ocupando recursos da máquina injustificadamente). Logo, decrementa-se em 1 o índice máximo.

As operações terminarão quando o índice máximo for zero, ou seja, quando coincidir com o primeiro índice (índice zero). Isto acontece pois um array de um elemento está de certeza ordenado.

QuickSort

O algoritmo QuickSort (desenvolvido em 1960 pelo cientista Charles Antony Richard Hoare) é de longe o algoritmo de ordenação mais usado e considerado pelo maior parte dos programadores como o melhor algoritmo dentro do género.

Este algoritmo implementa uma solução para a ordenação de um array baseado no famoso lema da informática "dividir para conquistar". O que basicamente o QuickSort faz é ir dividindo o array, através da selecção de um pivot (elemento de referência), e ordenando depois cada parte.

Apresenta-se a seguir um exemplo de implementação do QuickSort:

```
void qsort (int v[], int inicio_v, int
fim_v)
{
     //afectar os indices que irão
percorrer o array com as posições
iniciais e finais do array
     int a = inicio_v, b = fim_v;

     if (a >= b)
        return;

     //pivot (elemento do meio do
array)
     int pivot = v[(a+b)/2];

     do {
        //ir pesquisando por números
que estejam à esquerda do pivot e que
sejam maiores que este
        while ( v[a] < pivot)
                 ++a;    //incrementar
"a" para se ir avançando no array

        //ir pesquisando por números
que estejam à direita do pivot e que
sejam menores que este
        while ( v[b] > pivot)
            --b;   //decrementar "b"
para se ir regredindo no array

        if (a <= b)  //trocar apenas
se "a" menor ou igual a "b"
        {
            if (v[a] != v[b])
//efectuar trocas apenas se os valores
forem diferentes
                    swap (v[a],
v[b]);
            //incrementar "a" e
decrementar "b" para se retomar as
pesquisas
            ++a; --b;
            }
     } while (a <= b);

     //quer-se ordenar um sub-array
que vai desde o principio do array
principal ate antes do pivot.
     //quando se terminam as
pesquisas, "b" representa a posição
antes do pivot.
     qsort (v, inicio_v, b);

     //quer-se ordenar um sub-array
que vai desde a posição seguinte ao
```

```
pivot até ao fim do array
     //principal quando se terminam as
pesquisas, "a" representa a posição
seguinte ao pivot.
     qsort (v, a, fim_v);
}
```

No fundo, este algoritmo pode-se resumir a sucessivas colocações de valores maiores que o pivot à sua direita e de valores menores à sua esquerda. No exemplo apresentado, o elemento escolhido como pivot é o elemento central de cada parte do array. Esta escolha tem tanto valor como qualquer outra. No entanto, se o array estiver parcialmente ordenado, ela é preferível.

Imagine-se o seguinte array (denomine-se o array por v): [1,3,1,2,5,4,3]. Vamos recorrer ao QuickSort para o ordenar:

• Seleccionar o pivot. Para este caso o pivot será o elemento contido no índice 3 (elemento central), que contém o valor 2;

• Colocar à esquerda do pivot só elementos menores que este e à direita só elementos maiores. Começando no princípio do array (pesquisar por elementos maiores ou iguais que o pivot), o elemento de índice 1 (valor 3) é maior que o pivot. Portanto, o elemento do índice 1 terá de ser trocado;

Nota: no exemplo apresentado, elementos iguais ao pivot tanto se podem encontrar à esquerda como à direita deste. No entanto, se durante a pesquisa de valores maiores ou menores que o pivot for encontrado algum elemento igual ao pivot, este será tomado para troca.

• A pesquisa por elementos maiores que o pivot (e que estivessem inicialmente à esquerda deste) termina, tomando-se o valor v[1] para troca;

• Para a pesquisa de valores menores que pivot e que estejam à sua direita (inicialmente), começa-se no índice final do array, com sucessivos decrementos. Como se pode observar, começando em v[6], o elemento v[3], que é o valor 2, é igual ao pivot (realce-se que o pivot é um valor). Ora, o ciclo while em que é feita a pesquisa por valores menores que o pivot, terminará para um valor igual ao pivot. Assim, será tomado para troca o valor contido em v[3];

• Procede-se à troca. O valor que se usou para ir percorrendo o array ascendentemente (no exemplo é a variável "a") é menor que o que se usou para percorrer o array de forma descendente (no exemplo é a variável "b"). Portanto, a condição a < = b retorna TRUE, entrando-se assim no processo de troca de elementos;

• Os elementos não são iguais (v[1] ! = v[3], retorna TRUE), logo todas as condições previstas para a troca estão satisfeitas;

• Invoca-se a função de troca (swap()), que se encarregará de trocar os elementos;

• Depois de trocados os elementos, fica-se com o vector v da seguinte forma:

v = [1,2,1,3,5,4,3]

• Os índices "a" e "b" são, respectivamente, incrementado e decrementado. Retoma-se, portanto, a pesquisa por valores maiores que o pivot em v[2] e a pesquisa por valores menores em v[2];

• Durante a pesquisa por valores maiores, conclui-se que o elemento v[3] (contém o valor 3) é maior que pivot. Logo, será tomado para troca;

• Já para as pesquisas por elementos menores que pivot, o elemento v[2] é igual que pivot, logo será tomado para troca;

• No entanto, a troca não será efectuada, já que o índice usado para percorrer o array ascendentemente é maior que o usado para percorrer o array descendentemente. Desta forma, não se processa a troca de elementos ("a" é maior que "b", logo a condição a ÿ b retorna FALSE);

• O ciclo do...while é também terminado, já que a condição de teste deste ciclo é a ÿ b;

• Procede-se então às duas chamadas recursivas à função qsort(), sendo que na primeira chamada será apenas tomado em conta o sub-array [1,2,1] e na segunda o sub-array [3,5,4,3] (no entanto, para questões de indexação, o array em causa continua a ser o próprio v, sendo que para o segundo sub-array, v[3] representa o primeiro valor 3);

• Para a primeira parte do array, o pivot será o valor 2 (v[1]), sendo que se troca v[1] (valor 2) e v[2] (valor 1);

• Para a segunda parte do array, o pivot será o valor 5 (v[4]), sendo que a única troca que se efectua é entre v[4] (valor 5) e v[6] (valor 3);

• Depois destas duas trocas, o array está totalmente ordenado.

O algoritmo QuickSort termina quando numa chamada recursiva a qsort(), o parâmetro "inicio_v" for maior ou igual a "fim_v". Foi imposta esta condição terminal pois um array de um elemento está de certeza ordenado (se o índice que

representa o início do array for o mesmo que representa o fim do array, então está-se na presença de um array de 1 elemento).

Algoritmos Clássicos – Pesquisa em Arrays

Tanto no mundo da programação, como no dia-a-dia, somos confrontados com diversas situações que requerem a pesquisa de algo num determinado conjunto de elementos, que em princípio terão características semelhantes ou iguais àquilo que se quer encontrar. Um exemplo muito simples, mas ilustrativo deste facto, é a situação em que se tem de encontrar um livro de uma disciplina dentro de uma mochila, na qual existem também outros materiais escolares (estojos, cadernos, etc.).

Em informática, o conceito e toda a logística em torno do pesquisar tem assumido um papel preponderante nos últimos anos. Exemplo disso é toda a agitação em torno dos diferentes motores de busca na Internet (Google, Yahoo!, etc.).

A própria programação não foge a esta regra. As pesquisas representam também uma operação com importância chave numa aplicação.

Uma maneira de encontrar algo no dia-a-dia poderá ser uma pesquisa sequencial aos elementos de um determinado conjunto. Ou seja, verificam-se, um a um, os elementos do conjunto até se encontrar o elemento pretendido. Este processo, apesar de se poder implementar em programação, não é de todo o melhor para se encontrar um elemento dentro de um conjunto de dados (por exemplo um array).

Neste contexto surge a pesquisa dicotómica, um algoritmo que facilita a procura de um certo elemento dentro de um array.

- Pesquisa Dicotómica em Arrays

A pesquisa dicotómica em arrays (também muitas vezes referida como pesquisa binária) requer, antes de mais, que o array esteja ordenado. Ou seja, este algoritmo requer que antes da sua execução seja posto em execução um algoritmo de ordenação, tal como o QuickSort ou o BubbleSort.

Este algoritmo é provavelmente um dos mais simples e rápidos de entender dentro do mundo da programação. Este algoritmo propõe que se vá verificando sucessivamente o elemento central do array (que vai sendo divido em sub-arrays), fazendo uma comparação entre este e o elemento que se quer encontrar. Se o elemento central for menor que o valor que se deseja encontrar, então deverão analisar-se os elementos que estão à direita do elemento central. Caso o elemento central seja maior, deverá proceder-se à pesquisa na parte esquerda do array.

Se o elemento nunca for encontrado, então a função retorna -1, indicando que no array em questão não existe o elemento que se procurava.

Exemplo de implementação de pesquisa dicotómica em arrays:

```
int search (int v[], int find, int
inicio_v, int fim_v)
{
    //se o delimitador à esquerda for
maior que o da direita
    if (inicio_v > fim_v)
    //o elemento não existe no array
em causa
        return -1;

    //elemento central (índice central
do array)
    int pivot = (inicio_v+fim_v)/2;

    if (v[pivot] == find)        //se
o elemento central for igual ao
pretendido
        return pivot;
//indicar o índice em que se encontrou

    //se o valor central for maior que
o valor a pesquisar
    if (v[pivot] > find)
        return search (v, find,
inicio_v, pivot-1);  //pesquisar na
parte esquerda do array
    //se o valor central for menor
    else
        return search (v, find,
pivot+1, fim_v);    //pesquisar na
parte direita do array
}
```

A função que implementa o algoritmo de pesquisa dicotómica em arrays (no exemplo é a função search), recebe como parâmetros o array a analisar (variável "v"), o valor a encontrar (variável "find"), o índice inicial da parte do array a ser analisada (variável "inicio_v") e o índice final da parte do array a analisar (variável "fim_v").

Esta função termina quando a condição inicio_v > fim_v retornar TRUE. Pense-se no seguinte: após várias chamadas à função search() ocorre nova chamada à função, desta vez

com uma parte do array principal que só apresenta um elemento. Neste caso, o pivot que será escolhido será esse mesmo elemento. Se o pivot não for igual ao valor que se procura, então de certeza que o valor que se procurava não existe no array. No entanto, o algoritmo continua a ser executado. Só que para ambas as possibilidades que se apresentam (o pivot ser maior que o valor que se procura ou ser menor), o parâmetro "inicio_v" será maior que "fim_v".

Se o pivot for maior que o valor que se procura, então o parâmetro formal "inicio_v" terá o mesmo valor com que iniciou essa chamada à função, sendo que o parâmetro formal "fim_v" terá o valor de "pivot – 1" (como se trata de uma parte do array que tem apenas um elemento, tanto "inicio_v" como "fim_v" têm o mesmo valor que "pivot", logo "fim_v" será passado na nova chamada à função com um valor menor que "inicio_v").

O outro caso que pode ocorrer ("pivot" ter um valor menor que o valor que se procura), o parâmetro formal "inicio_v" será afectado com o valor de "pivot + 1" e o parâmetro formal "fim_v" terá o mesmo valor com que iniciou essa chamada à função (tal como no caso do "pivot" ser maior que o valor em procura, também neste caso "fim-v" e "inicio_v" têm o mesmo valor que é o valor de "pivot").

Portanto, faz todo o sentido que quando "fim_v" for passado com um valor menor que "inicio_v" o algoritmo cesse, indicando-se que o valor que se procura não estava contido no array ("return -1" indica a inexistência do valor dentro do array).

Algoritmos Clássicos – Gestão de Dados

Nos dias que correm, a gestão de dados através de aplicações informáticas é um domínio extremamente importante e em que se tem apostado fortemente em termos de desenvolvimento e pesquisa. Não é por isso de admirar que exista uma variedade tão grande de facilidades e algoritmos dentro deste ramo.

Desde sempre foi uma necessidade do ser humano ter a informação ordenada da melhor maneira possível, sendo que as técnicas usadas para a ordenação dessa informação foram evoluindo e transformando-se ao longo do tempo. E neste aspecto a informática não foi excepção. Desde que foi lançado o primeiro computador pessoal até à data actual, muita coisa se modificou na forma de gestão de dados. Os algoritmos responsáveis pela boa gestão de dados forem evoluindo, foram criadas plataformas específicas para gestão de dados (as tão famosas aplicações de gestão de base de dados, tais como Oracle, MySQL, SQL Server, ACCESS, etc.). Como este é um ramo em constante transformação prevê-se que as inovações não cessem para já. Mais, é dito que as condições para mais evoluções e descobertas de soluções muitíssimo eficientes a este nível estão agora reunidas. Os algoritmos implementados em linguagens tradicionais (nomeadamente o OOP, Object-Oriented Programming) estão a começar a ser implementados em sistemas específicos de gestão de dados.

É um ramo da informática sobre o qual recaem imensas expectativas.

- Listas Ligadas (apresentação de uma lista simplesmente ligada em anel)

Um dos algoritmos mais famosos para a gestão de dados é o que implementa uma estrutura de dados denominada Listas Ligadas. Esta estrutura de dados supera o armazenamento e gestão de dados em arrays na medida em que na altura de se inserir um novo elemento na lista não é necessário alterar a posição dos restantes elementos. Isto acontece pois todos os elementos da lista estão ligados ao seu seguinte (numa variação deste algoritmo é mesmo possível ligar cada elemento ao seu anterior), tal como num comboio, em que as diversas carruagens estão ligadas sequencialmente, construindo a estrutura total.

Também dentro deste algoritmo existem inúmeras variações, nomeadamente a Lista Simplesmente Ligada, a Lista Simplesmente Ligada em Anel e a Lista Duplamente Ligada. Neste artigo será apresentada a Lista Simplesmente Ligada em Anel pela sua simplicidade e eficiência nos objectivos a que se propõe o algoritmo.

Exemplo de implementação de uma simplesmente ligada:

```
class List
{
    struct Node //estrutura que
representa o nó/celula
    {
        int data;   //atributo em que
será contida a informaçao do nó
        Node* nextNode; //apontador
para o nó seguinte da lista
        //construtor que será invocado
para criar cada nó da lista
        Node (int val, Node* next)
        {
            data = val;
            nextNode = next;
        }
              //construtor default que
será invocado para construir o nó
sentinela
        Node()
```

```
        {
            nextNode = this;
        }
    };
    //criação do nó que servirá de
sentinela (invoca o construtor default
da estrutura)
    Node guard;
    //método auxiliar find() para
verificar se um dado valor existe ou
não na lista
    bool find (int val, Node*& prev,
Node*& next);

public:
    //método para inserir um novo
elemento numa lista
    bool insert (int val);
    //método para remover um
determinado valor da lista
    bool remove (int val);
    //destrutor; elimina todos os nós
da memória
    ~List();
};
```

Neste exemplo, o algoritmo é implementado através do paradigma OOP. Desta forma, consegue-se uma melhor abstracção e sobretudo prima-se em organização e eficiência.

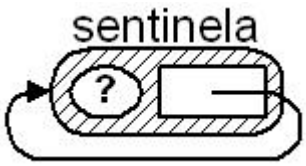
Como foi referido, as listas ligadas representam uma estrutura de dados constituída por células/nós, que contêm a informação a armazenar e a gerir, estando estes ligados numa forma sequencial. Em termos de programação, os nós da lista são implementados através uma estrutura (struct), contendo esta um ou mais atributos para conter informação e um outro atributo que a liga às restantes estruturas da lista. No exemplo, os nós são referenciados pela variável "Node".

No exemplo está implementada uma lista em que cada nó existe um atributo para conter informação (no caso um valor inteiro) e o atributo "nextNode", que é do tipo apontador para "Node" (irá apontador para a estrutura que representa o nó seguinte). "Node" tem também dois construtores, um com dois parâmetros (responsável pela criação dos nós normais da lista) e um construtor default (que será invocada aquando da criação do nó sentinela).

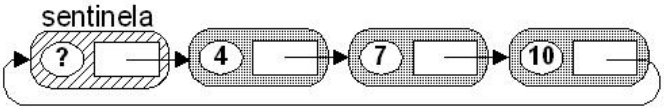
O construtor com parâmetros recebe um valor do tipo inteiro ("val"), que será o valor que o novo nó conterá, e um apontador para Node ("nextNode"), que representa o nó seguinte ao nó que será criado.

A particularidade que esta lista introduz é da ligação em anel. A organização da lista é feita desta forma a fim de facilitar os algoritmos de inserção e remoção dentro da lista. Este nó denominado sentinela (referenciado no exemplo como "guard") é sempre o primeiro nó da lista, sendo que quando a lista está vazia aponta para si mesmo (o nó seguinte ao nó sentinela é o próprio sentinela) e quando não está vazia, o último nó aponta para o sentinela (o nó seguinte ao último nó da lista é sempre o sentinela). Este nó especial é criado através da invocação do construtor default. Dentro deste construtor é usada a palavra-chave this para afectar o atributo "nextNode" do nó que invoca o construtor (neste caso, o sentinela). Ou seja, o nó que invocou o construtor passa a apontar para ele mesmo.

Se o código apresentado for posto em execução, a linha "Node guard" cria o sentinela, fazendo assim com que seja criado o primeiro nó da lista. Desta forma, quando a lista é criada tem este aspecto:



Quando forem inseridos valores na lista (os valores inseridos numa lista ficam sempre ordenados de uma forma ascendente), a lista apresenta o seguinte aspecto:



Repare-se na forma que a lista apresenta, com o último nó ligado ao sentinela.

No exemplo foram apresentados quatro métodos pertencentes à classe List: o método (auxiliar) find, o método insert, remove e o destrutor da classe.

O método find é usado pelos métodos de inserção e remoção da lista, find e remove respectivamente, a fim de se saber se um dado valor existe ou não dentro da lista (não faria qualquer sentido tentar inserir um valor que já estivesse presente na lista ou remover um valor que nem sequer existia dentro da lista). O método find poderia ser então programado da seguinte forma:

```
bool List::find (int val, Node*& prev,
Node*& next)
{
    guard.data = val;    //truque para
evitar a pesquisa infinita
```

```
    //afectar "next" como sendo o nó seguinte a sentinela e prev como sendo a própria sentinela
    for (next = guard.nextNode, prev = &guard;
    //cessar a pesquisa pelo elemento apenas quando a informação de "next" for maior ou igual a "val"
    next->data < val;
    //por cada iteração afectar "prev" com as características de "next" e "next" com as do nó seguinte
    prev = next, next = next->nextNode);

    //retornar TRUE se o valor existir ou FALSE se o valor não existir
    return next->data == val && next != &guard;
}
```

Este método realiza uma pesquisa por toda a lista para verificar a existência de um determinado valor. Recebe como parâmetros o valor a procurar e dois apontadores para Node passados por referência ("prev" e "next", que serão afectados directamente, já que são argumentos passados por referência). O atributo "val" de "guard" é afectado com a valor a procurar. Esta afectação é feita devido à forma como a pesquisa é realizada: o atributo para conter informação do apontador "next" vai sendo afectado com os sucessivos valores dos nós que representa, sendo que a pesquisa termina quando "nextÿdata" representar um valor maior que "val". Assim, imaginando-se que o valor que se procura não existia na lista e que todos os valores contidos na lista eram menores que o que se procurava, se "guard.data" não fosse afectado com o valor a pesquisar, a pesquisa nunca teria fim, já que nunca se contrariava a condição "nextÿdata < val", o que implicaria um ciclo for infinito.

Este método retorna TRUE ou FALSE consoante o valor que se procurava exista ou não, respectivamente. O valor de retorno é dado pelo valor lógico da expressão "nextÿdata == val && next != &guard". "Dissecando" esta expressão, como se trata de um a expressão em que se usa o operador lógico &&, apenas se todos os operandos forem condições verdadeiras, o valor lógico da expressão total é TRUE. Caso contrário (basta que um operando seja falso) a expressão terá o valor lógico FALSE.

O primeiro operando retorna TRUE se o valor do atributo "nextÿdata" for igual a "val". Ora, se o valor que se procurava estiver inserido na lista, então quando a pesquisa terminar o apontador "next" representa o nó em que está contido o valor em procura e "prev" o seu nó anterior. Assim, se este operando retornar TRUE, o valor encontra-se na lista.

No ínicio do método, foi afectado "guard.val" com o valor que se procura (para assim evitar pesquisa infinita). Desta forma, se o valor que se passou como parâmetro não existir na lista, e se todos os valores da lista forem menores que o que se procura, "nextÿdata == val" retorna TRUE. Ou seja, é indicado que o valor em procura está contido na lista, quando na realidade isso não se passa. Daí que se teste se o endereço de memória apontado por "next" é o endereço do nó sentinela. Em C++, a condição que traduz este caso é "next != &guard". Portanto, se esta condição retornar TRUE, já se sabe que o apontador "next" não está a apontar para o nó sentinela, ou seja, não percorreu toda a lista, não encontrando o valor pretendido.

O método "find" retorna TRUE (o valor existe de facto na lista) apenas quando "next" aponta para um nó que contém o valor pretendido e não está apontar para o sentinela.

Outro método apresentado no exemplo é o método de inserção. Exemplo de implementação do método de inserção de um novo valor:

```
bool List::insert (int val)
{
    //nós auxiliares para percorrerem a lista
    Node* next, *prev;

    if (find (val, prev, next)) //se o elemento já existir na lista
        return false;   //indicar o insucesso da operação de inserção

    //criar um novo elemento
    prev->nextNode = new Node (val, next);

    //indicar o sucesso da operação
    return true;
}
```

Os nós auxiliares criados dentro deste método ("next" e "prev") são os nós que serão passados por referência ao método auxiliar "find". Sendo passados por referência, não existe necessidade de retornar estes nós no fim do método "find".

A expressão dentro do if não é mais que uma chamada ao método auxiliar "find", recebendo este como parâmetros o valor a pesquisar ("val") e os nós auxiliares que irão percorrer toda a lista ("next" e "prev"). Se este método retornar TRUE,

então o valor já existe dentro da lista, logo não faz sentido tentar inseri-lo de novo. Daí que se faça "return false" caso isto se verifique.

Se a chamada "find (val, prev, next)" retornar FALSE, então o valor ainda não existe na lista, logo pode ser inserido. Portanto, se verificar este retorno por parte de "find", a execução da aplicação passa o controlo if (não atinge "return false") e por isso é necessário invocar o construtor para os nós normais da lista. Ora, este construtor recebe dois parâmetros: o valor que será contido pelo nó e o nó seguinte a este. Como no final da pesquisa o apontador "next" aponta para o nó em que está contido o primeiro valor maior que o que se procura, então o nó seguinte ao nó que se pretende criar será exactamente o nó "next". Portanto, os parâmetros a passar ao construtor estão já definidos. No entanto, será preciso ligar este nó ao seu nó antecessor. Daí se afectar o atributo "nextNode" de "prev" com o novo nó que será criado ("prev->nextNode = new Node (val, next)").

Outro método importante para a implementação de listas é o método de remoção. Exemplo de implementação do método de remoção:

```
bool List::remove (int val)
{
    Node* next, *prev;

    if (!find (val, prev, next)) //se o valor não existir na lista
        return false;   //indicar o insucesso da operação

    //desligar o nó que continha o valor pretendido da restante lista
    prev->nextNode = next->nextNode;
    //libertar o espaço de memória ocupado pelo nó a eliminar
    delete next;
    //indicar o sucesso da operação
    return true;
}
```

Também o método de remoção recorre ao método auxiliar "find" para reconhecer se o valor em questão já está ou não contido na lista. Neste caso, é indicado o insucesso da operação de remoção se o valor não estiver contido na lista (não faz qualquer sentido remover um valor que nem sequer está contido na lista). Daí que se faça "return false" caso a chamada "find (val, prev, next)" retorne FALSE (através do operador ! nega-se o valor lógico da expressão a avaliar pelo if, daqui que se possa escrever simplesmente "! find (val, prev, next)").

Se a chamada ao método auxiliar retornar TRUE, então é necessário desligar o nó da restante lista. Para isso liga-se o nó anterior àquele que se vai remover ao nó seguinte daquele que se vai remover. Em C++ esta operação é implementada pela linha "prev->nextNode = next->nextNode" (mais uma vez se salienta que no final da pesquisa, se o valor existir dentro da lista, o apontador "prev" aponta para o nó anterior ao que se pretende remover e "next" representa o próprio nó a remover).

Para eliminar o nó da memória usa-se o operador delete, para assim se libertar o espaço de memória ocupado pelo nó, sem levantar qualquer problema para a máquina.

Finalmente, é também implementado um destrutor da classe List. Exemplo de implementação do destrutor da classe:

```
List::~List()
{
    //nó que irá percorrer a lista
    Node* actual;

    while (guard.nextNode != &guard)
    {
        //afectar "actual" com as características do nó seguinte à sentinela
        actual = guard.nextNode;
        //desligar o nó da restante lista
        guard.nextNode = actual->nextNode;
        //eliminar o nó da lista
        delete actual;
    }
}
```

A função deste destrutor não é mais que a de eliminar da memória todos os nós constituintes da lista. Para isso, vão-se eliminando os nós seguintes ao sentinela ("guard.nextNode"). Só deverá ser feita esta eliminação, obviamente, enquanto a lista não estiver vazia. Ora, a lista estar vazia significa que apenas permanece em memória o nó sentinela. Portanto, a lista estará apenas vazia quando o nó sentinela tiver como nó seguinte ele próprio. Este facto testa-se através da condição "guard.nextNode != &guard" (realça-se o facto de se usar o operador referência, para que assim se teste o endereço de memória do nó seguinte a "guard", já que "nextNode" é do tipo apontador para Node). Enquanto a lista não estiver vazia, a operação realizada para eliminar o nó da memória é idêntica à realizada no método de remoção. Em primeiro lugar é necessário desligar o nó em questão da restante lista. O seu nó anterior (que será o

sentinela) passa a ter como nó seguinte o nó que era seguinte àquele que se irá remover. Depois disto, basta simplesmente usar o operador delete, libertando-se assim o espaço de memória ocupado pelo nó.

Para além dos métodos apresentados, numa implementação de uma lista simplesmente ligada em anel podem ser implementados outros métodos. Um método também bastante comum de uma lista é o método que imprime no console os valores de todos os nós; é igualmente frequente a implementação de um método que simplesmente procura por um determinado valor dentro da lista (útil em caso do utilizador querer procurar por um valor específico, sem o intuito de proceder a inserções ou remoções).

Conclusão

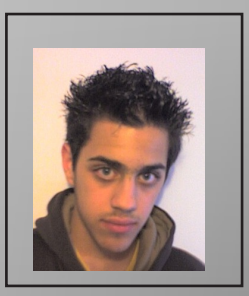
Neste artigo foram apresentados alguns dos algoritmos mais conhecidos dentro do mundo da programação, recorrendo para isso à linguagem C++. No entanto, e apesar da breve introdução e de cada exemplo ser acompanhado por descrições e explicações detalhadas, pressupõe-se que o leitor tinha já adquiridos conceitos dentro desta linguagem. Aconselha-se então aos leitores menos familiarizados com esta linguagem que realizem uma pesquisa e aprendizagem mais aprofundadas dos conceitos e temas mais básicos.

Através da redacção do presente artigo pretende-se desenvolver o gosto pela algoritmia junto dos leitores. É bastante importante contrariar a mentalidade da nova geração de programadores de que este mundo se cinge à aprendizagem de linguagens de programação. Assiste-se, cada dia que passa, a um crescente desprezo pela algoritmia, a qual deveria ser tomada como o elemento mais importante dentro da programação.

Bibliografia

• RODRIGUES, Pimenta; PEREIRA, Pedro; SOUSA, Manuela; Programação em C++ - Conceitos Básicos e Algoritmos; Lisboa; FCA; 7.ª edição; ISBN 972-722-038-X (1998);
• http://www.portugal-a-programar.org/forum/index.php/topic,2167.0.html
• http://www.portugal-a-programar.org/forum/index.php/topic,4739.0.html
• http://pt.wikipedia.org/wiki/C%2B%2B

SOBRE O AUTOR



Estudante de 17 anos, Mário Pereira frequenta o curso de Ciências e Tecnologia, sendo um grande amante do mundo informático. As suas linguagens preferidas são C++, C# e SQL. Aprendeu aquilo que sabe de programação maioritariamente como autodidacta, tendo um gosto especial por algoritmia e tecnologia das bases de dados.

Mário Pereira

Manipulação de Ficheiros com Ruby

Neste artigo vamos explorar a linguagem Ruby e os seus recursos para manipulação de ficheiros e directórios bem como as capacidades de Input/Ouput disponíveis. Para isso vão ser utilizadas as bibliotecas mais comuns como Dir, File e IO. Embora existam outras mais recentes e com mais recursos estão são as mais comuns, de simples utilização e que servem para base de bibliotecas mais recentes.

Directórios

Para trabalhar com directórios vamos utilizar a Classe Dir. Esta classe disponibiliza diversos métodos que permitem aceder e manipular directórios de forma rápida .

Vamos começar por ver um exemplo simples, um pequeno script que verifica se o directório actual é o definido pelo programa. Depois de passar para o referido directório todo o conteúdo é impresso um a um.

```
1 WORK_DIR = "/home/magician/"
2
3 if Dir.pwd != WORK_DIR
4   Dir.chdir( WORK_DIR )
5 end
6
7 Dir.foreach( WORK_DIR ) { |nome|
puts nome }
```

É possível ainda criar e eliminar directórios de forma muito simples, bastando usar os métodos mkdir e rmdir que a classe Dir dispõe.

```
1 Dir.mkdir( "/home/magician/Ruby",
755 )
2
3 Dir.rmdir( "/home/magician/Ruby" )
```

O exemplo acima não faz mais do que criar um directório no caminho dado e com as permissões dadas. Estas permissões são opcionais, podemos apenas dar o caminho. Em seguida eliminamos o directório criado com o método rmdir. Podíamos utilizar o método delete em detrimento ao método rmdir, pois ambos fazem exactamente o mesmo.

A classe dir também permite criar Streams a directórios, permitindo desta forma percorrer o conteúdo de directórios de forma mais flexível do que usando o método foreach mostrado anteriormente.

```
1  dir = Dir.open( "/home/magician/" )
2
3  dir.path #=> "/home/magician"
4
5  dir.tell #=> 0
6
7  dir.read #=> "Ficheiro 1"
8
9  dir.tell #=> 599320
10
11 dir.read #=> "Ficheiro 2"
12
13 dir.rewind
14
15 dir.close
```

Como podemos ver na linha 1 é aberta uma ligação ao directório. A partir dai podemos, entre outras coisas, percorrer todos os ficheiros e directórios contidos no directório aberto. O método tell (linha 5 e 9) retorna a localização do apontador sob forma de inteiro, o método read (linha 7 e 11) retorna sob forma de String a próxima entrada no directório, retornando nil quando não existirem mais entradas. Por fim o método rewind (linha 13) permite colocar o apontador da stream de novo no inicio e método close (linha 15) fecha a ligação ao directório.

Ficheiros

Criar um ficheiro não podia ser mais simples - basta utilizar a classe File:

```
1 file = File.new("exemplo.txt", "w")
```

O exemplo acima cria o ficheiro exemplo.txt e abre o ficheiro em modo de escrita. Vamos agora ver que outros modos de abertura e criação de ficheiros existem.

- "r" - Read-Only. Começa no inicio do ficheiro.
- "r+" - Read-Write. Começa no inicio do ficheiro.
- "w" - Write-Only. Passa o tamanho do ficheiro a zero ou cria um novo ficheiro para escrita.

• "w+" - Read-Write. Passa o tamanho do ficheiro a zero ou cria um novo ficheiro para escrita e leitura.
• "a" - Write-Only. Começa no fim do ficheiro caso este exista, caso contrário cria o ficheiro para escrita.
• "a+" - Read-Write. Começa no fim do ficheiro caso este exista, caso contrário cria o ficheiro para escrita e leitura.
• "b" - (DOS/WIN only) Binary.

Depois de criarmos um ficheiro, podemos precisar de apagar o ficheiro ou até mesmo mudar o nome desse ficheiro. Este processo é também muito simples, para isso a classe File disponibiliza os métodos rename e delete que permitem estas operações .

```
1 File.new( "exemplo.txt", "w" )
2 File.rename( "exemplo.txt",
"novoExemplo.txt" )
3 File.delete( "novoExemplo.txt" )
```

O exemplo acima cria o ficheiro "exemplo.txt" (linha 1), em seguida o nome do ficheiro é modificado de "exemplo.txt" para "novoExemplo.txt" (linha 2), por fim o ficheiro é eliminado (linha 3).

A classe File permite ainda realizar mais algumas operações úteis, como por exemplo verificar se um ficheiro existe, se é um directório, se é possível ler ou escrever no ficheiro, entre outras. Vamos ver alguns exemplos:

```
1  File.exist?( "ficheiro.txt" )
2
3  File.file?( "ficheiro.txt" )
4
5  File.directory?( "ficheiro.txt" )
6
7  File.readable?( "ficheiro.txt" )
8
9  File.writable?( "ficheiro.txt" )
10
11 File.executable?( "ficheiro.txt" )
12
13 File.zero?( "ficheiro.txt" )
14
15 File.size( "ficheiro.txt" )
16
17 File.size?( "ficheiro.txt" )
```

O exemplo mostra algumas das mais importantes operações sobre ficheiros. Na linha 1, o método exist? verifica se o ficheiro existe retornando true ou false, em seguida nas linhas 3 e 5 verificam se o ficheiro dado no path é realmente um ficheiro ou se é um directório. Nas linhas 7 e 9 os métodos readable? e writable? verificam se é possível ler ou escrever no ficheiro e na linha 11 o método executable verifica se o ficheiro é ou não executável. No final do exemplo podemos ver os métodos zero?, size e size?. O método zero? verifica se o ficheiro tem ou não tamanho igual a zero, os métodos size e size? fazem exactamente o mesmo que retornar o tamanho do ficheiro em bytes, com a diferença que caso este seja nulo, o primeiro retorna o e o segundo nil. Para além destes métodos é ainda possível obter mais algumas informações sobre os ficheiros como por exemplo data de criação, de edição e do ultimo acesso. Vamos por isso ver um exemplo desta funcionalidades.

```
1 File.ctime( "ficheiro.txt" )   #=>
Thu Jan 21 19:55:16 +0000 2008
2
3 File.mtime( "ficheiro.txt" )   #=>
Thu Jan 21 19:55:16 +0000 2008
4
5 File.atime( "ficheiro.txt" )   #=>
Thu Jan 25 19:55:16 +0000 2008
```

Como podemos ver, na linha 1 o método ctime retorna uma String como a apresentada acima com a data e hora a que o ficheiro foi criado, o mesmo se passando com os métodos mtime (linha 3) e atime (linha 5), mas neste caso estes métodos retornam a informação referente à ultima modificação e ao ultimo acesso do ficheiro respectivamente.

A classe File tem ainda mais um recurso bastante útil que é o chmod. Este método permite alterar as permissões de acesso ao ficheiro. As masks para as permissões são as seguintes.

r - read
w - write
x - execute

• 0700 - rwx para o dono.
• 0400 - r para o dono.
• 0200 - w para o dono.
• 0100 - x para o dono.
• 0070 - rwx para o grupo.
• 0040 - r para o grupo.
• 0020 - w para o grupo.
• 0010 - x para o grupo.
• 0007 - rwx para os outros.
• 0004 - r para os outros.
• 0002 - w para os outros.
• 0001 - x para os outros.
• 4000 - Altera o user ID na execução.
• 2000 - Altera o group ID na execução.
• 1000 - Salva texto em swap mesmo depois de usar.

A utilização do método chmod é muito simples, basta abrir o ficheiro como foi mostrado e executar o método chmod.

```
1 file = File.new( "ficheiro.txt", "r"
)
2 file.chmod( 0644 )
```

O exemplo acima dá permissão de de leitura de escrita (4 + 2) ao dono e de leitura ao grupo e todos os outros utilizadores.

Input/Ouput de Ficheiros

Ruby tem ainda mais uma classe muito útil, a classe IO. Esta pode ser usada isoladamente ou em conjunto com a classe File, e permite trabalhar com fluxos de input e output. Em seguida vamos ver alguns exemplos da utilização dos recursos desta classe através da classe File.

```
1 src = File.open("exemplo.txt","r")
2
3 puts src.readline   #=> Linha 1
4 puts src.readline   #=> Linha 2
5 puts src.readline   #=> Linha 3
6
7 src.flush
8 src.close
```

O exemplo acima abre o ficheiro em modo de leitura e lê as três primeiras linhas do ficheiro. Podíamos ainda utilizar o método gets que tem exactamente o mesmo funcionamento, com excepção de que o readline lança uma excepção quando o ficheiro chega ao fim. Para além de ler é também possível escrever para o ficheiro.

```
1 dst = File.open("exemplo.txt","w")
2
3 dst.puts("Linha 1")
4 dst.puts("Linha 2")
5 dst.puts("Linha 3")
6
7 dst.flush
8 dst.close
```

Este exemplo utiliza o método puts para escrever no ficheiro de destino, a cada execução do puts é automaticamente inserido um "\n", para escrever sem a inserção de nova linha podemos utilizar o método putc este método coloca um carácter e cada vez. Assim encerramos o artigo com toda a informação básica necessária para trabalhar com ficheiros.

API

- http://www.ruby-doc.org/core/classes/Dir.html
- http://www.ruby-doc.org/core/classes/File.html
- http://www.ruby-doc.org/core/classes/IO.html

SOBRE O AUTOR

Fábio Correia é estudante de Engenharia Informática na Universidade de Évora. Partilhando o estudo com a moderação do fórum Portugal-a-Programar e a participação na Revista Programar, como um dos redactores mais activos, ainda tem tempo para explorar algumas das suas linguagens preferidas: Java, PHP e a recente D.

Fábio Correia

Bioinformática
O lado do programador

Problemas e Soluções

No seguimento do artigo "Introdução à Bioinformática" (11ª Edição da PROGRAMAR), no qual abordei a existência desta àrea da informática (e da biologia), passo agora a exemplificar, em termos mais práticos, as ferramentas e técnicas mais populares, que são diariamente usadas por milhares de cientistas.

Relembrando, bioinformática define-se como o aplicar de técnicas de matemática aplicada, informática, estatística e inteligência artificial a problemas concretos de biologia e/ou bioquímica. Entre os problemas mais comuns, temos o alinhamento de sequências de DNA, a procura de genes (por associação e prévio conhecimento de outros), a assemblagem de genomas, a predição de estruturas de proteínas (secundárias, terciárias e quaternárias), bem como outras áreas menos "moleculares", como a evolução.

Para cada uma destas tarefas há determinadas estratégias a seguir, e, obviamente, uma boa estratégia aplicada a outro problema, resulta provavelmente num desastre em termos de eficácia. A razão mais simples por detrás desta pouca plasticidade de métodos, é o diferente conjunto de requisitos que cada problema assume. Exemplificando; um problema comum em biologia é o acesso a bases de dados de diferentes tipos e de diferentes especificações. Isto resulta numa difícil uniformização dos dados e, consequentemente, do tipo de métodos a usar para os obter e tratar. Além disso, é necessário pouco poder de computação "puro" para resolver estas tarefas. Interessa mais tirar partido de métodos que façam boa gestão do tráfego na web e que sejam eficientes a fazê-lo. Obviamente, uma linguagem como C é preterida em relação a outra como Perl. Já noutro problema, como a predição de estruturas de proteínas, onde milhões de complexos cálculos matemáticos são efectuados por segundo, já nos interessa tirar o máximo partido do poder de computação disponível, recorrendo então a linguagens que nos permitam desprezar operações supérfluas e canalizar recursos para onde de facto interessam. Não é de estranhar então que, ao procurar aplicações de bioinformática, se encontrem destas escritas nas mais variadas linguagens, sendo as mais usadas: Java, Python, Perl, C, C++ e C#[1]. Outras linguagens, como por exemplo Ruby, encontram também alguns adeptos entre os bioinformáticos. Porém, para além das diferentes escolhas no que toca a linguagens de programação, diferentes implementações de diferentes estratégias têm um efeito preponderante na eficácia final do programa. Ainda assim, alguns problemas podem partilhar soluções ou partilhar de ideias e conceitos que, à partida, seriam úteis num ou noutro caso isolado.

Reinventar a roda?

Em ciência, os resultados são, geralmente, publicados. Nas ciências biomédicas em geral, partilham-se métodos, ideias, conceitos e resultados, permitindo o avanço rápido da investigação, uma vez que, salvo raros casos, há sempre alguém que já tocou, nem que seja ao de leve, no problema que temos em mãos, e que para ele contribuiu. Esta filosofia de partilha estende-se também à bioinformática. Apesar de haver programas cujo código está fechado, há grupos de investigadores que se associaram para facilitar o trabalho na área, construindo verdadeiros repositórios de código, bem documentado e testado, que permite resolver problemas casuais de forma (quase) instantânea. Um bom exemplo é a Open Bioinformatics Foundation. Dela, surgiram vários projectos "filhos", entre os quais o projecto BioPython, o BioPerl e o BioJava. Fora estes, mais famosos e mais maduros, uma pesquisa no google revela várias comunidades, organizadas por linguagem, que servem de repositórios de código, disponibilizando facilmente, ao utilizador, bibliotecas e módulos que lhe simplificam o trabalho.

Segue-se uma breve lista das que uma pesquisa simples por "bio[nome da linguagem]" produziu (infelizmente, ou felizmente, parece que ainda não houve alma que se lembrasse de criar um bioassembly!):

- BioPython – www.biopython.org
- BioPerl – www.bioperl.org
- BioJava – www.biojava.org
- BioC++ - biocpp.sourceforge.net
- BioRuby – www.bioruby.org

Pegando numa delas, vamos analisar o potencial prático deste tipo de abordagem. Escolhi o BioPython porque é a com que trabalho e porque código em Python é, de facto, legível e perceptível mesmo a quem não programa na linguagem.

Exemplos

Um problema simples, ideal para começar, seria partir de uma sequência de ADN e obter o respectivo ARN, a molécula que vai servir de molde à síntese de uma proteína. Lembrando conceitos básicos de biologia molecular, há quatro tipos de bases (comuns) no ADN, cada uma identificada por uma letra (a inicial): Adenina, Guanina, Timina, Citosina. Para o ARN, troca a Timina pelo Uracilo. Para além disso, temos que as bases são complementares entre si. Quando se dá a reacção de transcrição (ADN – ARN), cada base do ADN é "transcrita" numa exacta do ARN: A-U, T-A, C-G, G-C. Podemos então construir um programa que, dada uma sequência de ADN, nos devolva a correspondente de ARN. Porém, já não terão pensado nisso antes de nós? De facto, já, e como o processo de transcrição não é tão linear como o descrevi, já houve quem tenha escrito (e complementado) algoritmos que reproduzem in silico a transcrição (quase) tal como in vivo. Se estivermos a usar Python, basta-nos aceder ao módulo BioPython e chamar a classe Bio.Tools. O código fica tão simples como:

(dada uma sequência my_seq de ADN)

```
from Bio.Tools import Transcribe
transcriber =
Transcribe.unambiguous_transcriber
my_rna_seq =
transcriber.transcribe(my_seq)
print my_rna_seq
Seq('GAUCGAUGGGCCUAUAUAGGAUCGAAAAUCGC',
 IUPACUnambiguousRNA())
```

Uma vantagem notável de usar o módulo bioPython para uma simples operação de transcrição, quando comparada com a nossa alternativa de escrever o algoritmo à mão (que não passará das 5 linhas), é a fiabilidade dos resultados. Na verdade, a sequência my_seq não se trata de uma simples string. Trata-se de um objecto de uma outra classe, Bio.Seq, que é verificada, a nível de integridade biológica, e que define as operações que podem ser efectuadas sobre ela. Este simples mecanismo previne, por exemplo, o aplicar do método Transcribe numa sequência de ARN.
Entre as outras classes do BioPython, podemos encontrar código que funciona como parser de vários formatos. As diferentes bases de dados espalhadas pela web usam cada uma seu formato. Daí, são necessários diferentes parsers para podermos tratar esta informação. Em vez de os escrevemos nós (que dá muito trabalho e é complicado, experiência própria), podemos aceder facilmente a classes que o fazem na perfeição. O exemplo que se segue é bem ilustrativo da ilegibilidade de alguns formatos:

```
gi|6273290|gb|AF191664.1|AF191664
Opuntia clavata rpl16 gene;
chloroplast gene for (resto da
descrição)...
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGC
GAAAGAAAGAAAAAAATGAATCTAAATGATATAGGATTC
CACTATGTAAGGTCTTTGAATCATATCATAAAAGACAAT
GTAATAAA..(resto da sequência)..
```

Para obter informação de um ficheiro deste género (formato FASTA), usamos:

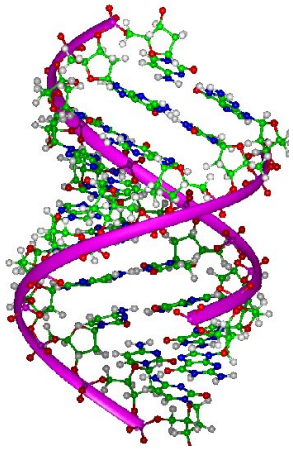
```
from Bio import Fasta
parser = Fasta.RecordParser()
file = open("ls_orchid.fasta")
iterator = Fasta.Iterator(file,
parser)
cur_record = iterator.next()
dir(cur_record)
['_colwidth', 'sequence', 'title']
```

Chamando os métodos title ou sequence, podemos aceder selectivamente a diferentes campos de informação. E assim, poupamos o trabalho de escrevermos nós um parser.

Como estes dois breves exemplos, há muitos outros. Há classes que importam duas estruturas proteicas de bases de dados e depois fazem a superimposição das duas, permitindo compará-las e discernir semelhanças e diferenças. Por exemplo, na comparação entre duas proteínas, uma funcional e outra mutante não-funcional, podemos ver quais os domínios (regiões) que podem estar na origem da perca de função. Outras classes permitem, com "one-liners", aceder a bases de dados e delas recuperar

informação estruturada (tendo previamente definido as variáveis search_command, search_database, search_term e return_format):

```
from Bio.WWW import NCBI
result_handle = NCBI.query(search,
base_de_dados, term =
termo_a_pesquisar, doptcmdl =
formato(ex. FASTA))
```



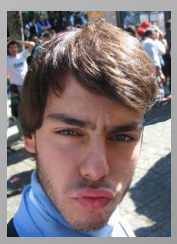
Conclusão

Concluindo, vemos que muita da contribuição de um bioinformático é facilitar o trabalho dos outros, seja construindo aplicações que facilitam o trabalho do cientista de wet-lab (quem realmente trabalha com material biológico), seja cedendo partes do seu código que aliviam o esforço do cientista de dry-lab (um bioinformático, portanto). Vemos também que, com o surgir diário de novas bases de dados, novas técnicas de análise, novas tecnologias disponíveis ao biólogo/bioquímico, há ainda muito trabalho a ser feito. Tal como disse no primeiro artigo sobre bioinformática, há investigação em informática para além dos bits e dos bytes, onde o contributo é muitas vezes relevante e visível. É uma questão de procurar, aventurar-se e de, no fundo, ter vontade de aprender.

Bibliografia

[1] Fourment, M., Gillings, M.R., "A comparison of common programming languages used in bioinformatics", BMC Bioinformatics (9:82), 2008

SOBRE O AUTOR



Estranhamente, João Rodrigues não vem das hostes dos informáticos tendo tido o primeiro contacto com a programação no 3º ano do curso. Um descontraído finalista da Licenciatura em Bioquímica em Coimbra, com particular interesse pelas áreas da Bioinformática e da Proteómica, entrou para a Revista PROGRAMAR e com o objectivo de mostrar que há mais na informática para além de bits e bytes.

João Rodrigues

Evento
Tecnologias Wireless

No passado dia 19 de Fevereiro, decorreu no Hotel Villa Rica em Lisboa, um evento dedicado às Tecnologias de Informação.

Este evento gratuito e organizado pela Rumos destinou-se a apresentar as últimas novidades relativas às tecnologias Wireless nomeadamente o tema da Voice Over Wireless e a Segurança das redes Wireless contando com o apoio da Cisco Systems Portugal e a colaboração da NextiraOne.

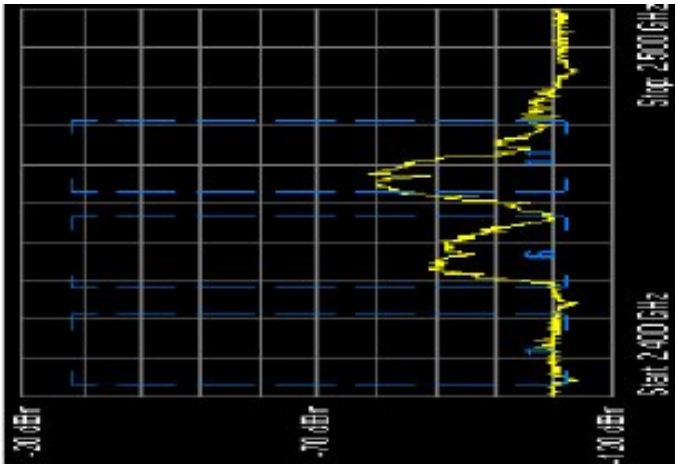
A primeira apresentação de título Cisco Wireless's Portfolio Development Manager ficou a cargo do orador Ricardo Santos da Cisco Systems de Portugal. Foi uma apresentação dedicada a três aspectos sendo eles: a apresentação da Solução WLAN Cisco, a Mobilidade e Redes Wireless e por fim o Standard 802.11n.

O orador focou o Cisco Unified Wireless Network, que é uma solução segura, escalável e que permite uma grande flexibilidade e liberdade em redes Wireless. E abordou diversos aspectos da mesma.

Também foram apresentas outras soluções bastante interessantes, como o Hybrid Remote Edge AP, que conforme mostrou o orador, "permitem terminar o tráfico localmente e/ou "tunneling back" para um controlador central" e "assegura operação em stand-alone quando o link da WAN está em baixo".

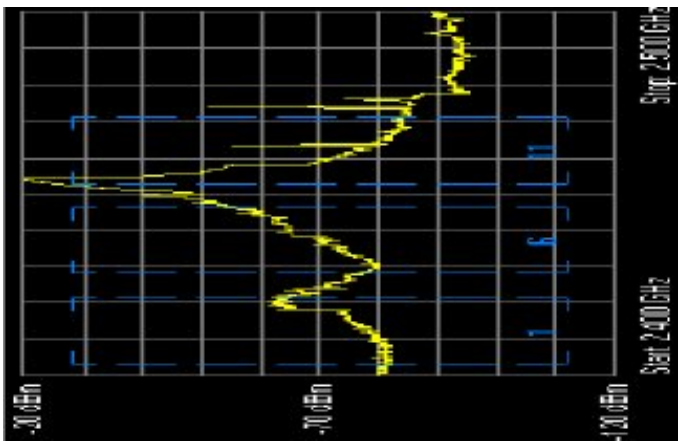
Outra solução para a gestão de RF em tempo real é o Radio Resource Management. Ao usarmos o RRM, o controlador que tem ao seu encargo um conjunto de Access Points, irá automaticamente ajustar cada AP de maneira a optimizar a cobertura de rede e a sua disponibilidade.

Além disso o RRM também permite fazer um balanceamento dinâmico dos clientes, por outras palavras divide o mal pelas aldeias. Ou seja se um AP estiver muito sobrecarregado, o controlador irá sugerir que o cliente se ligue a um AP mais livre. Outra solução apresentada e para mim foi a que gostei mais é a Cisco Spectrum Intelligence Technology. Esta imagem é um espectro de um AP com clientes, mostrado pelo orador Ricardo Santos.



Até aqui tudo bem, mas como sabemos existem muitas interferências prejudiciais para redes wireless e um sys adm, tem um trabalho complicado em descobrir o que está a causar o problema, onde está e também o que fazer.

É para isso que o Cisco Spectrum Intelligence Technology serve, para descobrir que aparelho está a causar a interferência, onde está e o que fazer. O CSIT analisa o espectro e sabe dizer que esta imagem em baixo é um espectro de um Micro-ondas.



Mais dois aspectos foram apresentados antes de passarmos para o 802.11n.

O primeiro é o Cisco Compatible Extensions (CCX) que permite uma interoperabilidade entre os clientes e a Cisco. Daí o CCX suportar o 802.11k, r, u, v e w. O segundo aspecto é o serviço de localização que conta com Chokepoints, CCX Tags, WiFi & CCX Devices, Wireless Location Appliance e Gestão WCS para saber exactamente onde determinado objecto está ou onde um cliente \ empregado se encontra.
Depois foram apresentados alguns Componentes Técnicos do 802.11n, como o MIMO (MRC, Beam forming, Spatial multiplexing), a utilização de Canais de 40Mhz, a Agregação de Pacotes (múltiplos pacotes agregados numa transmissão,

Block acknowledgements) e o Backward Compatibility (Clientes a/b/g/n no mesmo AP). A utilização destes aspectos permite melhorar em 5 vezes mais o envio de dados e diminuir em metade os Average Packet Retries.

Apesar de não ter sido possível a apresentação do Voice over Wireless, esta apresentação também feita no Porto foi uma abordagem do porquê usar VoIP mas neste caso em redes wireless e como isso é possível.

Depois de um coffe break passamos para uma apresentação que particularmente me interessava bastante – Segurança em Redes Wireless pelo orador Paulo Baptista da Rumos. Falou-se basicamente do IEEE 802.11: Standards, Ameaças e Soluções. Foram apresentados os avanços que tem sido feitos desde o passado em que se usava WEP com uma autenticação fraca e com senhas estáticas, passando por umas melhorias com o LEAP da Cisco em que já se usava senhas WEP dinâmicas e autenticação usando MS-CHAP modificado. Também se utilizou o WPA com encriptação mais forte usando PSK\TKIP e uma autenticação mais forte usando EAP-FAST e PEAP.

Actualmente usa-se o WPA2/802.11i em que se usa Wireless IDS, IEEE 802.11i com encriptação AES, autenticação 802.1x e uma gestão dinâmica de senhas e o WPA2 com AES. O orador Paulo Baptista mostrou como muitos desses protocolos funcionavam em especial, mostrou algumas considerações de como implementar o EAP.

Resumidamente podemos dizer que as duas opções que podemos usar são o PEAP e o EAP-TLS, sendo a diferença entre elas apenas a politica de autenticação, o PEAP usa Password/OTP e o EAP-TLS usa um certificado digital. Claro que esta diferença vai ter consequências no overhead do software do cliente e podemos dizer que é isso que irá levar a aplicação de um em detrimento de outro.

De seguida o orador Paulo Baptista abordou algumas das ameaças mais comuns a uma rede wireless. Mas também mostrou diversas soluções de segurança baseadas no CSWS – Cisco Secure Wireless Solution. Estas abordavam diversos desafios que uma rede enfrenta. Entre muitas a que mais me chamou á atenção foram: Como proteger a rede de utilização indevida? Como identificar os clientes e ao mesmo tempo proteger a rede de vírus?

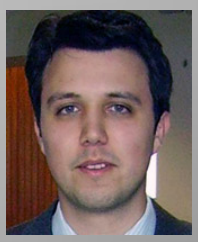
Se não estou em erro foi nesta parte em que se apresentou uma solução bastante interessante e que vou resumir brevemente. Um cliente novo chega à empresa, liga-se à rede, mas é feito um scan ao laptop, se existirem falhas de segurança no laptop, a rede apenas deixa que o cliente aceda às páginas web com as respectivas actualizações. No final o orador Paulo Baptista, mostrou algumas medidas a tomar para tornar a rede mais segura.

Por fim o orador Daniel Ferreira da NextiraOne, apresentou um Case Study da aplicação de muitas destas medidas de segurança e não só, a uma situação real.

Em resumo foi sem dúvida alguma, uma excelente iniciativa tanto da parte da Rumos como da Cisco Systems e a NextiraOne em mostrar e em divulgar estas inovações e também em mostrar o que se faz em Portugal nesta área.

Para mais informações sobre este evento, visitem o site da Rumos em http://formacao.rumos.pt/noticia.rumos?id=400 Na PTsec, existe um tópico aberto sobre este evento na secção Wireless: http://forums.ptsec.net/index.php?showtopic=1379.

SOBRE O AUTOR



Residente em Lisboa, Ciro Cardoso é um grande amante do vasto mundo que é a informática em especial a segurança. Está actualmente a trabalhar na próxima etapa do percurso Cisco com a certificação CCNP. Gosta bastante da área de redes, configuração de protocolos assim como a implementação.

Ciro Cardoso

Cprogramming

CProgramming é um site direccionado às linguagens C e C++. Nele é possivel encontrar de tudo um pouco, desde tutoriais, compiladores, blocos de código, até desafios de programação. É uma boa ferramenta para quem quiser começar em cada uma destas linguagens.

www.cprogramming.com

Haskell.org

O site oficial da linguagem Haskell. É possivel ver documentação de todo o tipo sobre esta linguagem, pesquisar blocos de código, e compiladores. Um site fundamental para quem queira conhecer um bocado mais desta linguagem e do paradigma funcional.

www.haskell.org

Services SAPO

Uma colecção de webservices disponibilizados pelo SAPO.pt, onde se podem destacar a geração de CAPTCHA, APIs do motor de busca, do sistema de mapas, ou mesmo verificar se há cobertura ADSL num determinado número de telefone.

services.sapo.pt

Revista Linux

A Verdadeira Revista Portuguesa de Linux

Edição Bimestral em formato PDF

Download Gratuito

www.revista-linux.com

Queres participar na Revista PROGRAMAR? Queres integrar este projecto, escrever artigos e ajudar a tornar esta revista num marco da programação nacional?

Vai a

www.revista-programar.info

para mais informações em como participar,
ou então contacta-nos por

revistaprogramar
@portugal-a-programar.org

Precisamos do apoio de todos para tornar este projecto ainda maior...

contamos com a tua ajuda!

Equipa PROGRAMAR

Um projecto Portugal-a-Programar.org