Módulo I – Fundamentos Computacionais
Sessão 5 - Aula 5 – Teste de Software – Pairwise – Parte 4
Allpairs Algorithm

Prof. Dr. Luiz Alberto Vieira Dias
Prof. Dr. Lineu Mialaret

ITAú Analytics – Fundamento de Computação – CEDS 111
2º SEMESTRE 2018

Teste de Software
Pairwise Testing - Parte 3
Algoritmo *Allpairs*

CEDS -111 - Teste de Software - Semana 3 - Aula 5

Prof. Dr. Luiz Alberto **Vieira Dias** (ITA)
Prof. Dr. Adilson Marques da **Cunha** (ITA)
Prof. Dr. **Lineu** Mialaret (ITA/IFSP)
Prof. Dr. Paulo Marcelo **Tasinaffo** (ITA)

A preparação deste minicurso contou com o apoio dos Alunos do ITA

- Lucas Nadalete, MSc
  - D. Montini, DSc
  - G. Matuck, MSc
  - E. Mineda, MSc

As técnicas

- Duas técnicas principais são utilizadas para determinar os Casos de Teste em *Pairwise* Testing:
  - **Orthogonal Arrays (OA)** (matrizes ortogonais)
  - **Algoritmo Allpairs** (todos os pares)

A Técnica Allpairs (todos os pares)

- Já foi vista a técnica de Orthogonal Arrays (OA)

- Agora será apresentada a técnica de *Allpairs Algorithm* (algoritmo para todos os pares)

A Técnica Allpairs (1/2)

- Esta técnica calcula diretamente, por meio de um algoritmo matemático, todas as combinações possíveis de pares de funções a serem testadas

- **Só para matemáticos(as): ver programa em JAVA:**

<fit.c2.com/wiki.cgi?AllPairsAlgorithm>

Desafio1: Passar para Python e ver se funciona!

Desafio 2: Entender o algoritmo acima

A Técnica Allpairs (2/2)

- Para os mais práticos, James Bach (**http://www.satisfice.com/**) provê, na rede, um software que a partir de uma planilha Excel, calcula a cobertura para todos os pares de funções a serem testadas

- **Desafio 3:** Entre no site: **http://www.satisfice.com**

Na opção **Testing** clique em **Methodology** e depois em **Tools**. Há explicações (em inglês) e o software é livre

Programa em JAVA (Allpairs -1/4)

```
// Copyright (c) 2002 Cunningham & Cunningham, Inc.
// Released under the terms of the GNU General Public License version 2 or later.

package eg;

import fit.*;
import java.util.*;
import java.io.File;


public class AllPairs extends AllCombinations {

    public int rank;
    public int steps=0;
    public Map toItem = new HashMap();
    public List vars = new ArrayList();
    public SortedSet pairs = new TreeSet();

    protected void combinations() {
        populate();
        generate();
    }
```

Programa em JAVA (Allpairs - 2/4)

```
// Populate ////////////////////////////////

    protected void populate() {
        doAllVars();
        doAllVarPairs();
    }

    protected void doAllVars() {
        rank = 0;
        for (int i=0; i<lists.size(); i++) {
            List files = (List) lists.get(i);
            Var var = new Var(i, files);
            vars.add(var);
            doAllItems(var, files);
        }
    }

    protected void doAllItems(Var var, List files) {
        for (int i=0; i<files.size(); i++) {
            Item item = new Item (var, i, rank++);
            toItem.put(((File)files.get(i)).getName(), item);
            var.items.add(item);
        }
    }
```

Programa em JAVA (Allpairs - 3/4)

```
protected void doAllVarPairs() {
        for (int i=0; i<vars.size(); i++) {
            for (int j=i+1; j<vars.size(); j++) {
                doAllItemPairs((Var)vars.get(i), (Var)vars.get(j));
            }
        }
    }

    protected void doAllItemPairs(Var vl, Var vr) {
        for (int l=0; l<vl.size(); l++) {
            for (int r=0; r<vr.size(); r++) {
                pairs.add(new Pair(vl.get(l), vr.get(r)));
            }
        }
    }
```

Programa em JAVA (Allpairs - 4/4)

```
// Generate ////////////////////////////////

    protected void generate() {
        while(((Pair)pairs.first()).used == 0) {
            emit(nextCase());
        }
    }

    private Item[] nextCase() {
        Item slug[] = new Item[vars.size()];
        while (!isFull(slug)) {
            Pair p=nextFit(slug);
            fill(slug, p);
        }
        return slug;
    }

    protected void fill(Item[] slug, Pair pair) {
        slug[pair.left.var.index] = pair.left;
        slug[pair.right.var.index] = pair.right;
        pair.used++;
        pairs.add(pair);
    }

    protected boolean isFull (Item[]slug) {
        for (int i=0; i<slug.length; i++) {
            if (slug[i]==null) {
                return false;
            }
        }
        return true;
    }
```

(....) **(continua....)**

Satisfice Inc. - Test Methodology - www.satisfice.com

This is an unordered repository of a few of the test methodology documents that exemplify our approach to testing.

Internet Explorer users: We've discovered that the IE Acrobat plug-in sometimes doesn't properly display the contents of PDF files. If you experience trouble, right click on the link and select "Save Target As..." Once you save the PDF file to your local drive, you can open it up without using IE and it should display properly.

Heuristics of Testability

This is a list of ideas for what makes a product more testable. It can help testers and developers improve the testability of a product so that testing goes faster and takes less effort.

Session-Based Test Management

Exploratory testing (sometimes referred to as "ad hoc" testing) is a creative, intuitive process. Everything testers do is optimized to find bugs fast, so plans often change as testers learn more about the product and its weaknesses. Session-based test management is one method to organize and direct exploratory testing. It allows us to provide meaningful reports to management while preserving the creativity that makes exploratory testing work. This page includes an explanation of the method as well as sample session reports, and a tool we developed that produces metrics from those reports.

SQA for New Technology Projects

When you're doing a 1.0 product, you can't rely much on experience to guide your SQA process. You also don't have regression test suites or any other specialized test materials that you can reuse. Meanwhile, the product itself is probably changing at a high rate. It's poorly documented and you may not be the first to know about major changes.

This document is a set of ideas for dealing with that situation. It begins with the idea that you have to change your thinking from a task orientation to a risk orientation.

General Functionality and Stability Test Procedure

(for Microsoft Windows 2000 Application Certification)

I produced this procedure for Microsoft to help them do a better job of assuring that applications that claim to be Windows 2000 compatible really are compatible. The procedure itself is documented in 6 pages. As far as I know it is the first published exploratory testing procedure. It's used along with a second non-exploratory procedure (which is 400 pages long!) to perform the certification test. What's interesting about that is the fact that my 6 pages represent about one third of the total test effort.

Satisfice Test Context Model

This is version 1.0 of a model I use to help me analyze software test projects. It depicts the major elements in the context of testing that should influence choices about test strategy, test logistics, and testing products.

Satisfice Heuristic Test Strategy Model

This is a model I've been using for years. It used to be called the Tripos model. I use this model to organize my thoughts about all the elements of test design. By memorizing the various checklists, I am able to rapidly generate lots of ideas for how to anything. This is a classic example of a set of testing heuristics.

Test Plan Evaluation Model

This is a model I use when I'm reviewing and critiquing a test plan. It lays out what are, in my opinion, all the interesting issues to consider concerning a test plan and associated documents. One of the interesting features of this model is a set of test project heuristics.

Test Plan Building Process

This is an experimental process for evolving a good test plan. I'm still experimenting with it. It's an example of a "forward-backward" process, where you proceed concurrently on all tasks, rather than linearly through each task in a predefined order. It's also yet another example of a heuristic approach to testing. This procedure doesn't tell you what to do, so much as suggest what to think about.

Navigation

Testing

- Methodology
- Tools

Como usar?

- A técnica Allpairs, será aplicada no mesmo Exemplo 1 dos Browsers das aulas anteriores:
  - 8 *Browsers*
  - 3 *Plug-ins*
  - 6 SO Clientes
  - 3 *servers*
  - 3 SO *servers*

Exemplo de uso de *Allpairs Algorithm*

- Um *website* deve operar corretamente com **8 *browsers*** diferentes:
  - Internet Explorer 6, 7, 8
  - Mozilla 1, 2, 3
  - Opera 10
  - Chrome 9
- Com **3 *plug-ins***: RealPlayer, Media Player ou nenhum
- Com **6 sistemas operacionais (SO) cliente:** Windows ME, NT, 2000, XP, Vista e 7
- Recebendo páginas de **3 *servers*** diferentes: IIS, Apache e Weblogic
- Rodando em **3 SO para *servers*** diferentes: Windows NT, 2003 e Linux

Como usar Allpairs (1/2)

- Depois de baixar e *unzippar* (descomprimir) o programa, crie uma tabela **Excel**, entre os dados, recorte-os e cole-os em um arquivo Wordpad do tipo txt: **input.txt**

Browser	Client OS	Plug-in	Server	Server OS
IE 6	Win 7	None	IIS	Win NT
IE 7	Win Vista	Real PL	Apache	Win 2000
IE 8	Win ME	Media PL	WebLogic	Linux
Chrome 9	Win NT			
Mozilla 1	Win 2000			
Mozilla 2	Win XP			
Mozilla 3				
Opera 10				

Como usar Allpairs (2/2)

- Para rodar o programa Allpairs, tecle em:

  All Programs > Accessories > Command Prompt > Run:

allpairs input.txt > output.txt

- A saída é o arquivo output.txt, que conterá a lista de todos casos de teste dos pares dos objetos/funções a serem testados

Resultado para Allpairs

	Browser	*Client OS*	*Plug-in*	*Server*	*Server OS*
1	IE 6	Win Vista	None	IIS	Win NT
2	IE 6	Win XP	Real PL	Apache	Win 2000
3	IE 6	Win ME	Media PL	W	Linux
4	IE 6	Win NT	Real PL	WE	Win NT
5	IE 6	Win 2000	None	IIS	Linux
6	IE 6	Win ME	None	Apache	Win 2000
…	…	…	…	…	…
45	Mozilla 3	Win NT	~None	~Apache	~Linux
46	Mozilla 3	Win 2000	~Real PL	~WebLogic	~Win 2000
47	Opera 10	Win XP	~Media PL	~IIS	~Win NT
48	Opera 10	Win 2000	~None	~Apache	~Linux

Comentários

- O algoritmo de James Bach (2002)

  (www.satisfice.com) escolhe inicialmente o valor que foi colocado em par, o menor nº de vezes em relação aos outros
- Quando um resultado **não importa**, pois todos os pares já foram testados, isto é indicado na tabela com um '~' (til)
- Por isso, no presente exemplo, a partir da linha 25 (Caso de Teste 25) começam a aparecer tils ('~')

Ref: KARNER, Cem; BACH, James; PETTICHORD, Bret **Lessons learned in Software Testing: A Context-Driven Approach**. New York, NY: Jonh Wiley & Sons, 2002

Resultado para OA (p/ comparar)

	Browser	*Plug-in*	*Client OS*	*Server*	*Server OS*
1	IE 6	Real PL	Win ME	IIS	Win NT
2	IE 6	**Real PL**	Win 2000	**WebLogic**	**Linux**
3	IE 6	**Media PL**	Win NT	**WebLogic**	**Linux**
4	IE 6	Real PL	Win XP	IIS	Win NT
5	IE 6	nenhum	Win Vista	WebLogic	Linux
6	IE 6	Media PL	**Win NT**	Apache	Win 2003
…	…	…	…	…	…
61	Chrome 9	nenhum	Win Vista	WebLogic	Win 2003
62	Chrome 9	Media PL	**Win NT**	Apache	Linux
63	Chrome 9	Media PL	Win 7	Apache	Linux
64	Chrome 9	nenhum	**Win Vista**	WebLogic	Win 2003

Resultado para *Allpairs*

	Browser	*Client OS*	*Plug-in*	*Server*	*Server OS*
1	IE 6	Win Vista	None	IIS	Win NT
2	IE 6	Win XP	Real PL	Apache	Win 2000
3	IE 6	Win ME	Media PL	W	Linux
4	IE 6	Win NT	Real PL	WE	Win NT
5	IE 6	Win 2000	None	IIS	Linux
6	IE 6	Win ME	None	Apache	Win 2000
…	…	…	…	…	…
45	Mozilla 3	Win NT	~None	~Apache	~Linux
46	Mozilla 3	Win 2000	~Real PL	~WebLogic	~Win 2000
47	Opera 10	Win XP	~Media PL	~IIS	~Win NT
48	Opera 10	Win 2000	~None	~Apache	~Linux

Comparação *Allpairs & OA* (1/2)

- IMPORTANTE: Qualquer valor pode ser substituído, por outro da coluna, nas células marcadas com '~', que a cobertura de todos os pares será mantida
- Devido à natureza "balanceada" do OA, esta abordagem requereu 64 Casos de Teste, contra apenas 48 do Allpairs, que é "não-balanceado", mas ligeiramente mais eficiente

Comparação *Allpairs & OA* (2/2)

- Em alguns casos podem haver requisitos não-funcionais que incompatibilizam alguns Casos de Teste
- Por exemplo, **MacOS** e **IIS** são incompatíveis e não seria possível (na verdade, desnecessário) incluir simultaneamente estes dois pares no teste
- Algumas ferramentas permitem que estas *constraints* (condições) sejam consideradas, reduzindo-se ainda mais o número dos Casos de Teste

Considerações: Qual o melhor?

- Note que tanto ***Allpairs*** quanto **OA** determinam **todos** os pares de funções a serem testados
- A maioria dos defeitos ocorrem em pares de funções ou em funções isoladas, porém os defeitos que ocorrem em *triplets* e acima **não são testados!**
- **OA**, por ser mais balanceado, provê uma cobertura maior (excede o necessário: + caro), ao passo que ***Allpairs*** é mais eficaz (testa somente o necessário: + barato)

Cuidado

- Alguns casos podem ser muito importantes, como, por exemplo:
  - abortar lançamento do foguete, ou
  - desligar o reator nuclear
- Estas situações podem exigir o teste de *triples* ou acima
- Adicione sempre este tipo de **teste crucial** em seus Casos de Teste, se pertinente