UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
INF01124 - Classificação e Pesquisa de Dados – Semestre 2013/1

RELATÓRIO SOBRE O DESENVOLVIMENTO DO SOFTWARE VISUAL SORT

Desenvolvedores:

Cristiano M. D. Ruschel ()
João Paulo T. Ruschel (219435)

Porto Alegre, 2013

1 Descrição Geral do Problema e Solução

1.1 O Problema

Este trabalho consiste em desenvolver um aplicativo que faz a indexação e análise de currículos Lattes de pesquisadores, devendo ser capaz de organizar, classificar e consultar os elementos cadastrados. As funcionalidades mínimas são:

1. Listar todos os pesquisadores cadastrados
2. Listar os artigos de um dado pesquisador, organizando-os em:
   2.1. Tipo (em Periódico ou Conferência)
   2.2. Natureza (artigo Completo, Estendido ou Resumo)
   2.3. Quantidade de coautores
   2.4. QUALIS

Além dessas funções básicas, o programa também deve oferecer, no mínimo, mais duas funcionalidades extras quaisquer.

1.2 A Solução

1.2.1 Ideia Geral

A solução proposta é a de sair da monótona maneira trivial para a resolução de problemas parecidos e montar um programa que mostra para o usuário os elementos, com suas ligações e características, ao invés de simplesmente listar extensamente ditos elementos, com menus longos e layouts não intuitivos.

1.2.2 Linguagem de Programação, Frameworks e outros códigos.

1.2.2.1 Linguagem de Programação

A linguagem escolhida foi C#, devido ao conhecimento adquirido previamente por ambos integrantes do grupo, permitindo, assim, um tempo menor entre ideia e implementação. Além disso, os Frameworks para linguagem C# servem perfeitamente para a maneira com que o problema quer ser resolvido, principalmente para a parte gráfica e leituras dos arquivos XML.

1.2.2.2 Frameworks Utilizados

O único Framework utilizado foi o XNA, da Microsoft. Embora seja uma ferramenta desenvolvida e utilizada para Jogos, o XNA ofereceu uma ótima plataforma para construir o programa, sem precisar se preocupar com aspectos mais técnicos e de baixo nível, principalmente na parte gráfica.

1.2.2.3 Outros Códigos

Seguindo a lógica de códigos terceirizados somente para a parte gráfica, também foi utilizado um pequeno trecho que gerencia a Câmera 2D do programa. O código em si utiliza em grande parte o próprio XNA, usando matrizes de transformação e modos de desenho controlados pelo Framework. O desenvolvedor desse código foi David Amador, e seu site, onde o código pode ser consultador, é:

http://www.david-amador.com/

1.2.3 Funcionalidades do Programa

Além das funcionalidades básicas descritas previamente, o aplicativo VisualSort também, como funções extras:

1.2.3.1. Cataloga Conferências atendidas, Artigos, Capítulos de Livro e Periódicos publicados, considerando cada um como um elemento que pode ser examinado separadamente.

1.2.3.2. É baseado na visualização gráfica dos elementos pesquisados, tornando intuitivo diversos métodos de pesquisa que não seriam triviais em outros aplicativos;

1.2.3.3. Permite a pesquisa de elementos com dados adicionais, como palavras-chave, ano de publicação, idioma, nacionalidade, entre outros.

2 Implementando o VisualSort

2.1 Estruturas de Dados

2.1.1 Aspectos Gerais

O primeiro problema encontrado foi quanto à memória, já que o programa necessita memória adicional para guardar, além das informações de cada elemento (pesquisador, artigo, etc), informações sobre o mundo gráfico (posições, câmera, primitivas para ligações, etc). A solução para tal problema foi simplesmente dividir em Disco e em Memória Principal os dados, carregando e descarregando conforme necessário.

O segundo ponto foi como estruturar os elementos de forma que formem ligações entre eles. Esse problema foi tratado via listas de índices, que será explicado em 2.1.3.

Os XML são processados somente uma vez, e, a partir daí, as listas geradas são salvas em disco ou mantidas em memória para o processamento de pesquisa, visualizações, etc. Quando o programa fecha, tais listas são salvas para a próxima execução não ter que processar tudo de novo.

O programa possui um tipo específico para cada tipo de elemento: Pessoa, Artigo, Periódico, Conferência e CapítuloDeLivro, onde cada um desses se divide em Disco e Memória Principal.

2.1.2 Estruturas em Disco

Em disco, deixam-se somente dados específicos para cada tipo de dado e não são essenciais para o processamento rápido de pesquisas e informações instantâneas. Foram programadas com o prefixo *TF*.

TFPessoa
- NomeCompleto: string; o nome completo
- País: string; o país de nascimento do pesquisador
- TextoResumo: string; pequena biografia sobre o pesquisador

TFArtigo
- Título: string; o título do artigo
- ISSN: string; ISSN para a procura do QUALIS
- Periódico: string; periódico onde foi publicado
- MeioDivulgação: string; o meio de divulgação do artigo
- Idioma: string; Idioma
- AnoPublicação: int; Ano de publicação
- Natureza: string; natureza do artigo (Completo, Estendido ou Resumo)
- PalavrasChave: lista de strings; palavras-chave para fácil pesquisa

TFPeriódico
- Nome: string; nome do periódico
- ISSN: string; código de ISSN

Qualis: string; Qualis do periódico

TFCapítulo

Título: string; o título do capítulo
ISBN: string; o ISBN do livro
Idioma: string; idioma do livro
AnoPublicação: int; o ano de publicação do livro
Natureza: string; natureza do livro (Completo, Estendido ou Resumo)
Livro: string; nome do livro ao qual o capítulo pertence
MeioDivulgação: string; o meio de divulgação do livro

TFConferência

Nome: string; nome da conferência
Sigla: string; sigla referente à conferência
Caráter: string; internacional/nacional
Qualis: string; o qualis

2.1.3 Estruturas em Memória Principal

Na memória principal, não há a necessidade de tipos de dados diferentes para tipo de dado. Ao invés disso, foi-se criado um tipo genérico, chamado *TInfoNodo*, que tem a característica de poder ser qualquer tipo de dado (Pessoa, Artigo, Conferência, etc). Ele contém, obviamente, um parâmetro específico que contém a informação de quem ele é e que índice ele é, o *TPNodo*.

Ele também contém um tipo de ponteiro, chamado de *BPos* que liga esse TInfoNodo em específico para o resto de informação no disco. Quem faz essa ponte é uma estrutura chamada *TBlocoHandler*.

Finalmente, um *TInfoNodo* também possui um ponteiro para um tipo especial de dados, o *TDrawNodo*, que representa um Nodo que será desenhado na tela.

Para organizar os índices, cada tipo de dado possui uma chamada Lista-Mestre, que é simplesmente a lista onde cada Informação está realmente alocada. Todas as ligações cruciais (como ligações *TInfoNodo* - *TInfoNodo*) são feitas por índices; e ligações não-cruciais (como ligações *TInfoNodo* - *TDrawNodo*) são feitas por ponteiros simples, já que são criadas somente a tempo de execução e em nada importam.

TInfoNodo

Nome: string; o nome ou título (dependendo do tipo)
Data: BPos; o índice para conversar com os *TBlocoHandler*
Nodo: TPNodo; contém informação sobre o que o TInfoNodo em questão é
DrawNodo: TDrawNodo; ponteiro para o nodo de desenho (não salvo pois não é essencial)
Ligações: lista de TPNodo; uma lista com todas as ligações desse nodo com outros nodos

TPNodo

Índice: int; o índice na lista-mestre
Tipo: int; o tipo de elemento, seguindo uma sequência pré-definida ((0= Pessoa; 1= Artigo; 3= Periódico; 4= Capítulo; 5= Conferência))

TDrawNodo

Utiliza-se de tipos de dados provenientes do Framework XNA, como Vector2 e Color, para definir posições, velocidades e cores para cada nodo.

2.1.4 TBlocoHandler: a ponte Disco <-> Memória Principal

Os elementos presentes na memória principal a todo o tempo (os *TInfoNodo*s) não possuem informação profunda sobre si para não ocupar desnecessariamente a memória. Tais informações são salvas em disco e somente acessadas quando requeridas pelo usuário. A estrutura que faz tal ponte, ou seja, que carrega as informações de disco conforme é pedido, é o TBlocoHandler, que se organiza por blocos de tamanho fixo.

Cada tipo de dado (Pessoa, Artigo, etc) possui um TBlocoHandler específico, e cda TBlocoHandler se organiza independentemente.

TBlocoHandler

BlocoAtual: array de [Tipo]; onde Tipo é o tipo específico de dados. Esse array é o buffer, onde é carregado um bloco inteiro para poder acessar um elemento. Ele é liberado da memória logo depois.
blocoCount: int; o contador de Blocos com informações
uÍndiceNoBloco: int; o índice do último elemento no último bloco

Os blocos são salvos em formato binário somente uma vez, quando os dados estão sendo processados. Todas as execuções do programa seguintes que utilizam tais blocos simplesmente carregam os blocos pré-processados.

Para fazer o link entre TInfoNodo e um elemento num Bloco, é necessário um *BPos*, que é justamente um tipo de ponteiro para as informações.

BPos

Bloco: int; o índice do Bloco onde está o elemento
Offset: int; o offset, dentro do Bloco, onde está exatamente o elemento

2.2 Gráficos

O programa VisualSort foi construído em volta do conceito de ligação entre elementos, o que permite formação de grafos de ligação em real-time rapidamente, deixando a critério do usuário que nodo "viajar" agora.

2.2.1 A ajuda do Framework XNA

O XNA, utilizado amplamente para os gráficos do VisualSort, embora muito completo e com muitas funções interessantes, não criou tudo sozinho. O framework somente inicializa *drivers*, *canvas*, *devices* e disponibiliza uma gama de estruturas para processamento de vetores, cores (como o *Vector2* e *Color*), matrizes, entre muitos outros.

Também é inerente ao framework técnicas gráficas para processamento de texturas e anti-aliasing, usando BackBuffers opcionais e MipMapping, além de desenho de primitivas (linhas) por rajada. Todas essas técnicas são utilizadas no VisualSort.

A partir desse alicerce, foram desenvolvidas todas as funcionalidades gráficas do programa.

2.2.2 O Grafo

O grafo de ligações, definido pelas ligações do nodo central, é desenhado de forma a imitar as voltas de uma concha, de forma que os nodos são igualmente acessáveis a partir do central, e seu grafo é desenhado.

2.2.3 A interface

A interface (barras laterais) é desenhada a partir da GUI que os próprios desenvolvedores do VisualSort criaram, e se dispôs de *buttons*, *lists*, *memos*, *checkboxes*, *panels*, *editboxes* e *labels*.

2.2.4 A Câmera

A implementação da Câmera é de total autoria de *David Amador*. Ela se utiliza de matrizes de transformação para simular, no XNA, um 3D com Z constante, gerando, assim, um plano 2D com Zoom e Rotação.

3 Considerações Finais