UNIVERSIDADE FEDERAL DE ITAJUBÁ
Campus Itabira

Engenharia da Computação
ECO027 – Projeto e Análise de Algoritmos

Projeto #1 R2D2

25037 – André Viana Sena de Souza

Prof. M.e Walter Aoiama Nagai
walternagai@unifei.edu.br

Monitor: Emmanuell Miqueleti
emmanuellnogmiq@hotmail.com

Itabira – MG

31 de agosto de 2014

Sumário

1 Introdução

O projeto consiste em executar uma tarefa simples e de maneira otimizada. Para tal, a linguagem de programação escolhida foi C++, se utilizando da IDE CodeBlocks para armazenamento do arquivo de projeto. Os cabeçalhos do mesmo foram organizados da seguinte maneira:

- algoritmo.h - Dispõe todos métodos necessários para execução plena do algoritmo principal.
- arquivo.h - Contém uma enumeração dos arquivos a serem lidos e a função LEIA.
- cronometro.h - Define os métodos utilizados para contagem de tempo das rotinas da aplicação.
- lista.h - Template de lista encadeada simples, a estrutura de dados base em todo o programa.

No módulo principal estão contidas a implementação da interface do usuário, e a lista contendo os resultados. Durante o desenvolvimento deste programa foi utilizado o compilador g++ 4.8 no sistema operacional Ubuntu 12.04.

2 Algoritmos e estruturas de dados

Diversos algoritmos foram utilizados para a implementação do projeto, no entanto, somente uma única estrutura foi necessária, o template de lista encadeada simples. Essa lista dispõe de um contador de elementos, métodos de iteração por elementos, e também uma função que retorna um vetor com os dados nela inseridos. O processo de inserção utiliza-se de um ponteiro extra, apontando para o último elemento da lista, com o intuito de acelerar o processo como um todo.

Os métodos iteradores foram implementados com este uso em mente:

```
for(lista->It_Inicio(); lista->Iterador()!=NULL; lista->It_Prox())
   foo(*lista->Iterador());
```

Sendo que lista->Iterador() sempre aponta para um dos membros da lista, porém quando a lista chega ao fim, retorna um ponteiro nulo.

Não foi necessário implementar métodos de remoção, pois a função Descarregar() exclui todos os membros da lista sempre quando possível.

2.1 Algoritmos

- AlgoritmoR2D2 - Lê os arquivos a partir da função LEIA e a enumeração ArquivosComputadorCentral. LEIA retorna uma lista com todos os números contidos no arquivo e é a partir deles que a contagem de pares é realizada. A função CONTE retorna esse valor, que é passado como parâmetro para a função MAIORNUMEROPRIMO como o limite superior para o cálculo de seu algoritmo. No final do processo, o Algoritmo de Invasão do R2D2 insere esse número primo em uma lista e procede com a leitura do próximo arquivo. Quando todos os arquivos terminam de ser lidos, o algoritmo retorna a lista de números primos encontrados.

- CONTE - Seguindo o príncipio de que somente pares {x, -x} resultam em zero quando somados, o algoritmo realiza a contagem. O primeiro passo é guardar os números em um vetor, para depois ordenar seus os dados, e percorrê-lo até que o primeiro número positivo apareça.

  A cada número negativo iterado, é efetuada uma pesquisa binária pelo seu oposto positivo no vetor. Se o par for completado, o contador é incrementado. Assim que o primeiro número positivo é percorrido, o algoritmo assume que todos os pares foram reconhecidos ({a, b} == {b, a}) e retorna o contador.

- BuscaBinária - Necessário para a função CONTE. Busca recursivamente por um elemento em um vetor ordenado. Retorna o index do elemento ou -1 caso ele não exista.

- QuickSort - Necessário para a função CONTE. Ordena recursivamente utilizando-se da técnica de divisão para conquista.

- MAIORNUMEROPRIMO - Em um for de limite superior a zero, busca por números primos, retornando o primeiro resultado encontrado.

- VerificaPrimo - Necessário em MAIORNUMEROPRIMO. Checa se o número é divisível por todos os números anteriores a ele. Esse processo pode ser otimizado utilizando-se de teoremas matemáticos para verificar se um número é primo ou não.

3 Apresentação e discussão dos resultados obtidos pelos experimentos

Para cada arquivo lido, a função CONTE apresentou diferentes tempos de execução:

Itens	Tempo de execução
1000	0 ms
2000	0 ms
4000	0 ms
8000	0 ms
16000	10 ms
32000	10 ms
64000	20 ms
128000	50 ms
256000	90 ms

Após o processo de otimização o programa se tornou muito mais rápido, ágil, e eficiente, devido ao uso da busca binária e do quicksort para ordenação. O ganho de performance é bastante evidente quando feita uma comparação entre o programa otimizado e o de teste, construído com o método de contagem implementando uma busca linear, que levou aproximadamente 1000 vezes mais tempo para ser executado ($\simeq$18,5 minutos x 1 segundo).

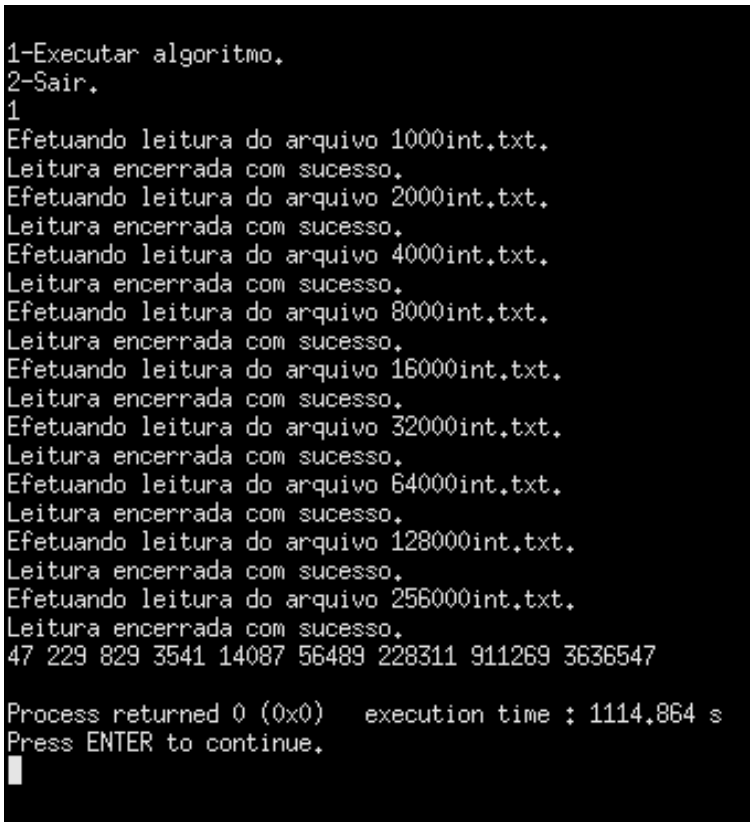

Figura 1: Tempo de execução do programa com busca linear.

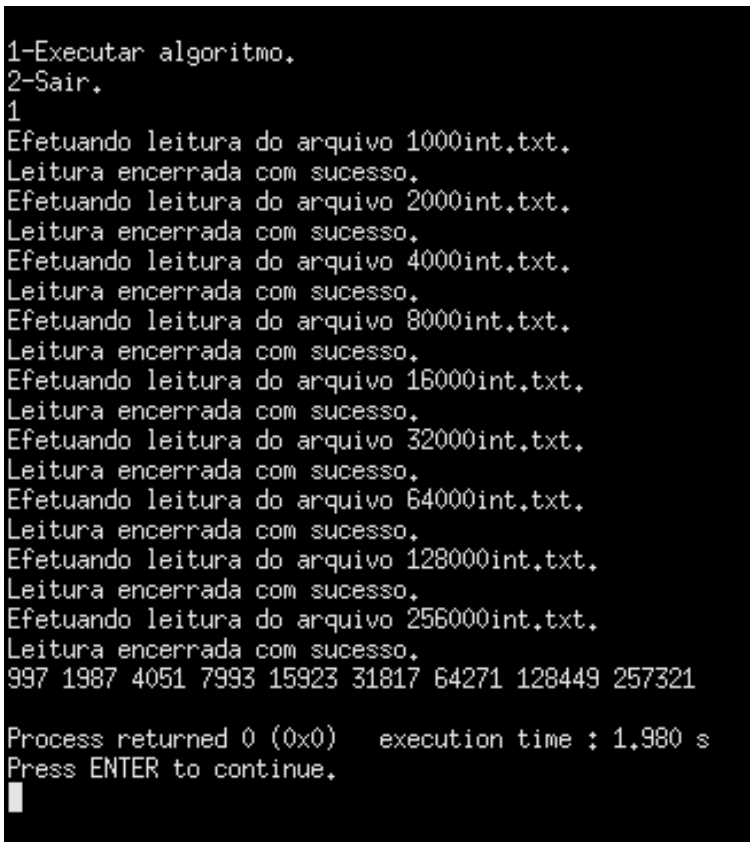

Figura 2: Tempo de execução do programa com busca binária.

4 Bibliografia

- http://stackoverflow.com
- http://rosettacode.org