Trabalho Prático
Laboratórios de Informática

Grupo 6

Guilherme Viveiros , Mateus Silva e
Angelo André
A80524 , A81952 , A80813

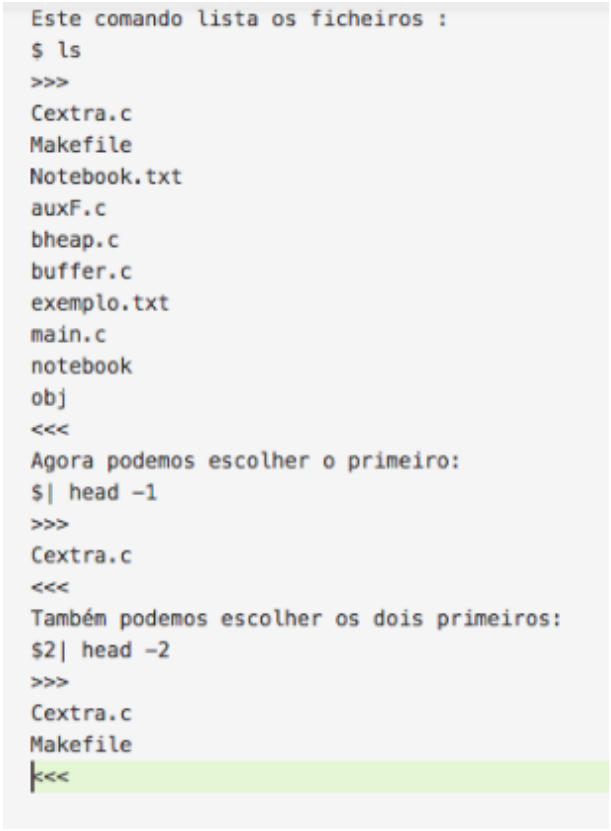
Departamento de Informática
Universidade do Minho

Contents

1 Motivação:

No âmbito da Unidade Curricular Sistemas Operativos, foi proposto pela equipa docente a implementação de um processador de notebooks.

Um processador de notebooks tem como objetivo receber um ficheiro , e retorná -lo com uma mistura de documentação e execução de fragmentos de código.



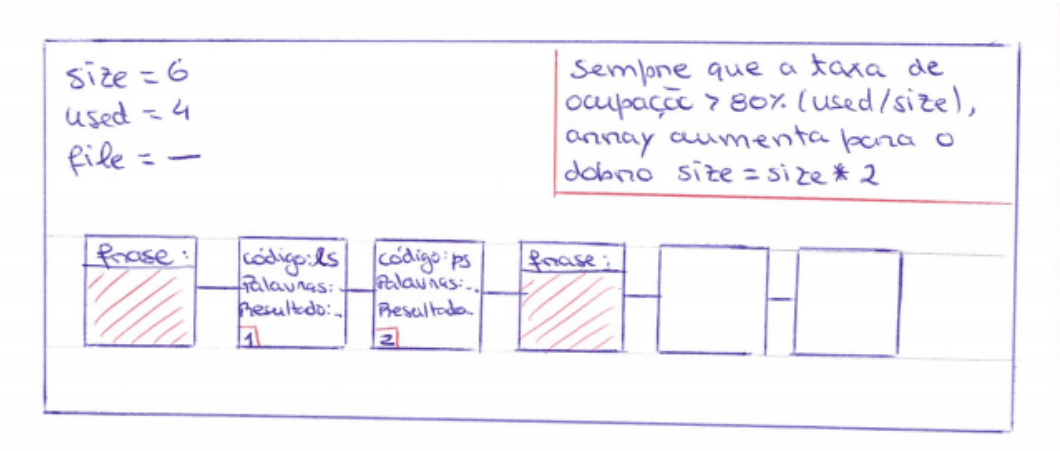
2 Carregamento de dados:

O carregamento dos dados será feito através de scripts de ficheiros de leitura, provenientes da parte docente ou dos alunos do grupo.

3 Estruturas principais(implementações):

Concebemos as nossas estruturas de maneira a que estas conseguissem estruturar todos os dados necessários e porventura acederem aos mesmos. Decidiu-se que seria uma boa implementação armazenar as informações num Buffer onde nele estivesse contido um growing array[1] denominado bloco. Como o respetivo trabalho se baseia numa adição sequencial de dados e numa procura dos mesmos estes tempos de execução são preciosos utilizando este bloco. Buffer também contém o respectivo tamanho do bloco, os elementos a ser utilizados pelo mesmo e um descritor no qual representa o ficheiro recebido como argumento.

Por sua vez , bloco será usado para armazenar todas as linhas do ficheiro recebido sucessivamente, ou seja ,cada bloco conterá uma frase , um índice que aplica uma distinção entre fragmentos de código e documentação , e porventura caso seja código , então o mesmo bloco contém as palavras associadas a este, para que seja efectuada a sua execução, e um índice (por ordem crescente) no qual representa a ordem do fragmento de código , visto que teríamos que aceder aos resultados posteriormente.



Tempos de execução:

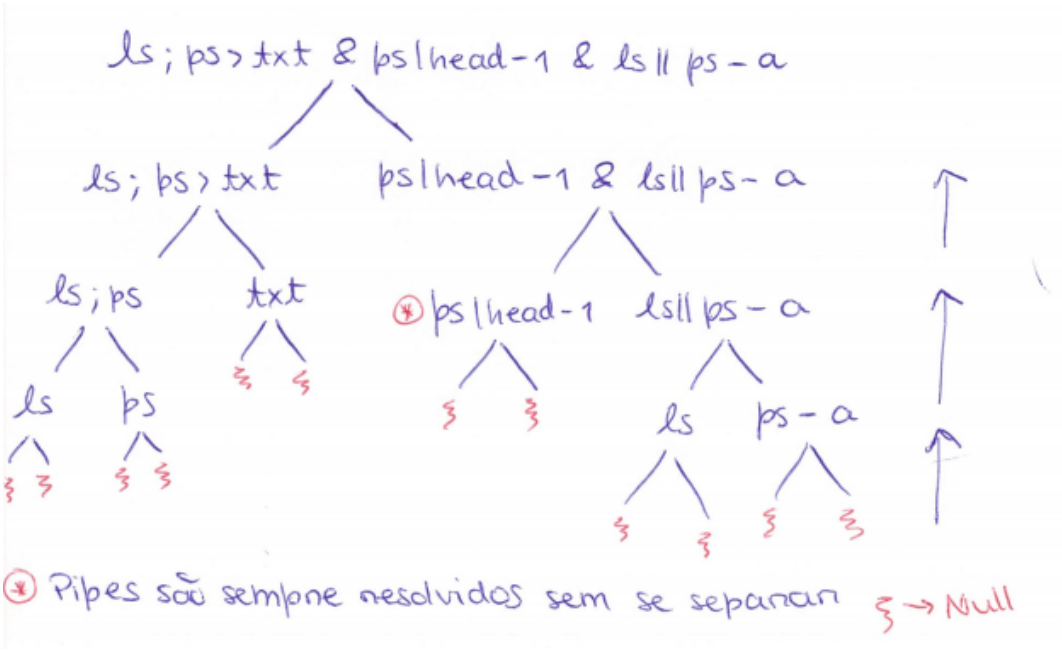
Criar	$\Theta(N)$
Adicionar	$O(1)$
Procurar	$O(1)$

[1] Lista de tamanho variável de dados que permite adicionador , procurar e remover elementos

4 Estruturas auxiliares(implementações):

Após termos a estrutura principal definida deparamo-nos com alguns problemas na execução de comandos complexos , isto é ,aqueles que contêm controladores e operadores de redirecionamento da Shell[2].

Como abdicar dos tempos constantes alcançados nas estruturas principais seria dispendioso decidimos então usar mais memória para a implementação de uma nova estrutura. Dito isto , foi implementada uma árvore binária[3] na qual já tinha sido estudada na cadeira Algoritmos e Complexidade, na qual divide o comando complexo em partes , e junta as respetivas sub-soluções para assim resolver o problema inicial , ou seja:



Neste exemplo concreto , os nodos finais são executados , e a partir destes os pais vão executando através do resultado dos filhos e assim recursivamente até ao comando inicial.

[2]https://unix.stackexchange.com/questions/159513/what-are-the-shells-control-and-redirection-operators

[3]Estrutura que representa uma árvore, por definição cada nó poderá ter até duas folhas

Existem algumas exceções na execução dos nodos finais , como no caso de redirecionamentos , ou seja ls ; ps > txt , neste caso não queremos executar txt só o filho esquerdo.

No caso em que o comando recebe stdin , a ideia implementada é percorrer a árvore até ao nodo inferior esquerdo , ou seja o primeiro comando, neste caso o ls ,e apenas este será executado com o respectivo stdin.

Esta divisão foi analisada pelos elementos do grupo, no qual foi abdicado algum tempo para analisar e compreender quais os operadores prioritários na divisão de um comando complexo.

5 Gestão de memória:

Para assegurar que não há perdas de memória nas diferentes partes do projeto usamos o software Valgrind[4], que indica os blocos de memória que foram reservados pela aplicação.

6 Modularização:

Todas as componentes do nosso trabalho foram criadas em ficheiros fonte separados e com atributos privados, para que ninguém consiga aceder a estes sem usar os métodos diretamente associados a cada uma das estruturas de dados criadas.

7 Convenções para entradas inválidas:

Dado que o utilizador pode usar comandos inválidos ou até parar o programa com CTRL C, as convenções para entradas inválidas foram simplesmente parar o programa e devolver o ficheiro recebido intacto.

[4] http://valgrind.org/

8 Referências:

[1] - J.W. J. Williams, Algorithm 232: Growing-Array, Communications of the ACM 7 (1964), 347-348.

[2] - J.W. J. Williams, Algorithm : Binary tree