INSTITUTO DE COMPUTAÇÃO - UNICAMP

MC202EF - Estrutura de Dados

2º Semestre de 2017

Professor: Julio Cesar dos Reis
Monitores: Wellington Lucas Moura (PED)
Victor Luccas Soares Villas Boas Antunes (PAD)
José Carlos Vasques Moreira (PAD)

Laboratório 07: Painel de Controle de Programas

Prazo de entrega: 05/11/2017 23:59
Peso: 2
Em dupla

1 - Descrição

Deseja-se desenvolver um software para facilitar o acesso a um conjunto de programas instalados em um computador. Esse software deve ser capaz de efetuar acesso rápido aos programas para inicializá-los, desinstalá-los ou instalar um programa novo. Para organizar esses programas internamente, o software os divide em pastas e subpastas que imitam a estrutura de uma árvore binária de busca.

Nessa árvore, cada nó representa uma pasta, que possui um nome e armazena um programa (nome do programa). O nó pode possuir até duas subpastas referentes as subárvores esquerda e direita. Se o programa de um dado nó possui nome “programa_exemplo”, o nome das subpastas esquerda e direita devem ser respectivamente “programa_exemplo_esq” e “programa_exemplo_dir”, que representam as subárvores de programas cujo nome é lexicograficamente menor e maior que “programa_exemplo” (ou seja, levando em consideração a ordem alfabética do nome dos programas). Essas subpastas não existem caso “programa_exemplo” seja um nó folha, porém ambas ou apenas uma podem existir caso “programa_exemplo” seja um nó interno. A Figura 1 apresenta uma ilustração da árvore. Note que o nome de programa “B” é menor alfabeticamente que “D” e “F” é maior.

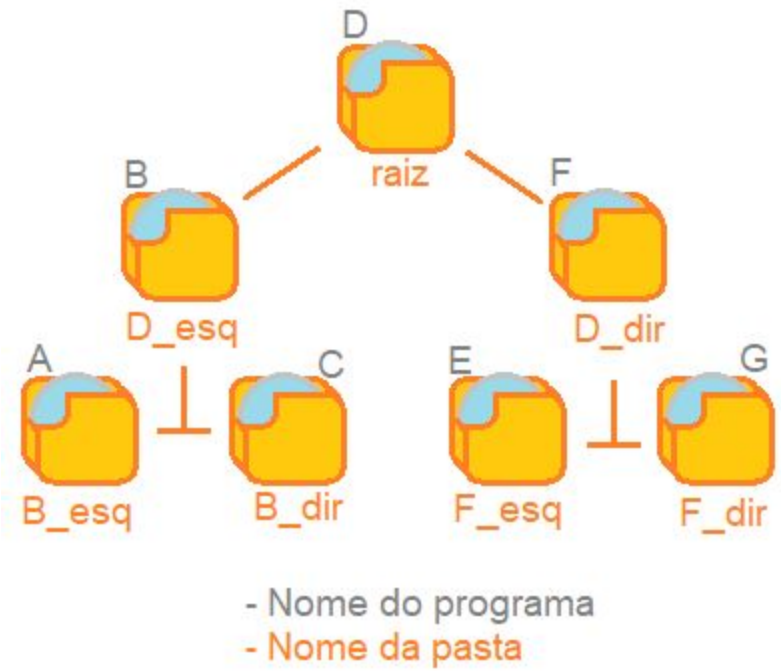


Figura 1: Ilustração da estrutura de árvore binária

Uma importante característica do software é que ele deve possibilitar efetuar um backup (cópia de segurança) da configuração atual de pastas, para que em caso de necessidade exista um ponto de restauração. Para esse fim, ele define duas operações, uma capaz de criar uma cópia de segurança, salvando a configuração atual da árvore em duas sementes geradoras[1], e outra capaz de recriar (restaurar) a configuração definida por essas mesmas sementes, sobrescrevendo a configuração original (ou seja, a árvore atual em memória).

Ao iniciar o software, já existem pastas criadas e programas instalados no sistema. Portanto, antes de efetuar qualquer operação, o software lê da entrada padrão qual o estado atual do sistema, ou seja, duas sementes geradoras. Esse estado além de inicializar a árvore, serve como ponto de restauração, se houver necessidade de recriar a configuração original (inicial) da árvore. Então deve-se efetuar uma cópia de segurança das semestes geradoras de entrada logo após a leitura da entrada das mesmas.

A inserção (instalação) de novos programas é equivalente ao algoritmo de inserção em uma árvore binária de busca. Nesse tipo de árvore binária, dada uma raiz, todos os elementos presentes em sua subárvore esquerda são menores ou iguais que ele por um determinado critério de ordenação, e todos os elementos de sua subárvore direita são maiores que ele. A inserção consiste em, partindo da raiz, buscar o lugar adequado para a inserção de forma a manter essa regra de ordenação. Durante essa busca, se estamos em um nó interno, comparamos o elemento que desejamos inserir com o valor desse nó (nome do programa), caso o valor a ser inserido é maior, inserimos recursivamente na subárvore direita, caso

[1] Veja enunciado do laboratório 06 para a definição e exemplo de sementes geradoras

contrário, na esquerda. Quando chegar a um nó folha, utiliza-se a mesma comparação e inseri-se o novo nó como filho esquerdo ou direito da folha (dependendo de seu valor). A Figura 2 apresenta um exemplo de inserção. Para o software que desejamos construir, cada nó representa uma pasta que ***sempre*** contém um programa (pastas não podem existir sem programas dentro delas, com exceção para a pasta “raiz"), e a ordenação é feita pela ordem lexicográfica desses programas. O nome desses programas e pastas contém apenas letras minúsculas, números e *underlines*. Considere ainda que todos os programas possuem nomes distintos.

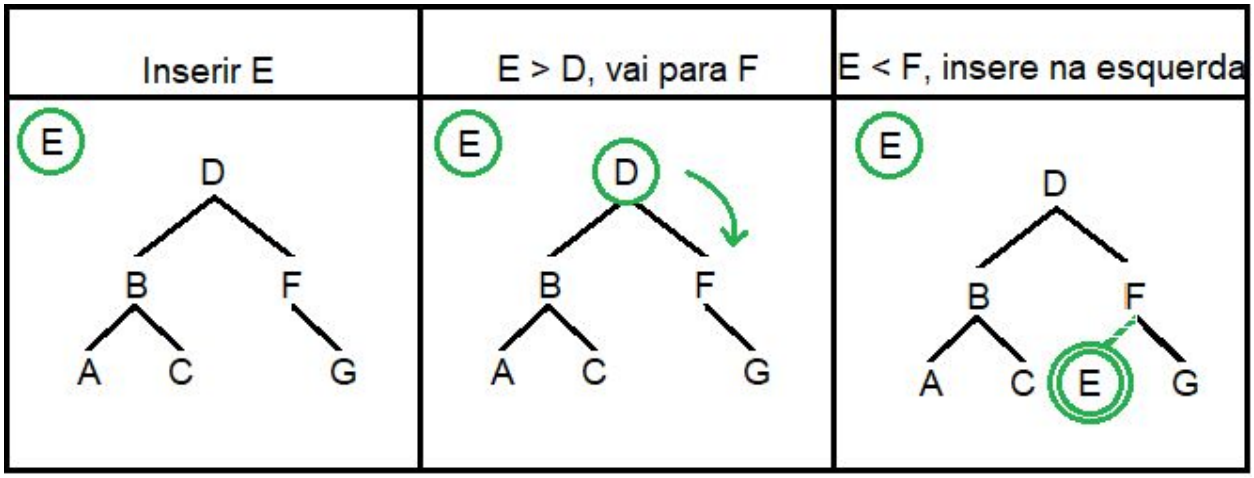


Figura 2: Exemplo de inserção

A remoção (desinstalação) também é feita como remoções em árvores binárias de busca. Buscamos na árvore o nó que tenha o valor que desejamos remover utilizando o critério de ordenação da árvore. Durante a busca, se chegar em um nó folha que não contêm o elemento que deseja-se remover (ou uma árvore vazia, dependendo da implementação), significa que esse elemento que deseja-se remover não existe. Por outro lado, caso ele exista, ele pode possuir dois filhos, um filho, ou nenhum filho (caso seja um nó folha), em cada uma das três situações, sua remoção é feita de uma maneira. Caso ele seja um nó folha, apenas o removemos e informamos seu pai da remoção. Caso ele só tenha um filho, o pai do nó atual recebe esse neto como filho, e o nó atual é removido. Caso ele tenha os dois filhos é necessário encontrar o primeiro elemento maior que ele na subárvore esquerda, ou o primeiro elemento menor que ele na subárvore direita, e trocar de posição com algum desses nós, chamados de descendentes. Esses descendentes são respectivamente o nó mais à esquerda da subárvore direita e o nó mais à direita da subárvore esquerda. Após trocado o nó que desejamos remover com algum de seus descendentes, o removemos recursivamente da subárvore para qual ele foi dirigido, nela temos certeza que ele será um nó de no máximo um filho, que pode ser removido sem problemas. A escolha do descendente influência no estado final da árvore após a remoção. Neste procedimento do software, utilize o descendente menor que o nó, ou seja, o elemento mais à direita da subárvore esquerda. Lembre-se que após

remover um programa, a pasta que o contém também deve ser removida. A Figura 3 apresenta exemplos de remoção.

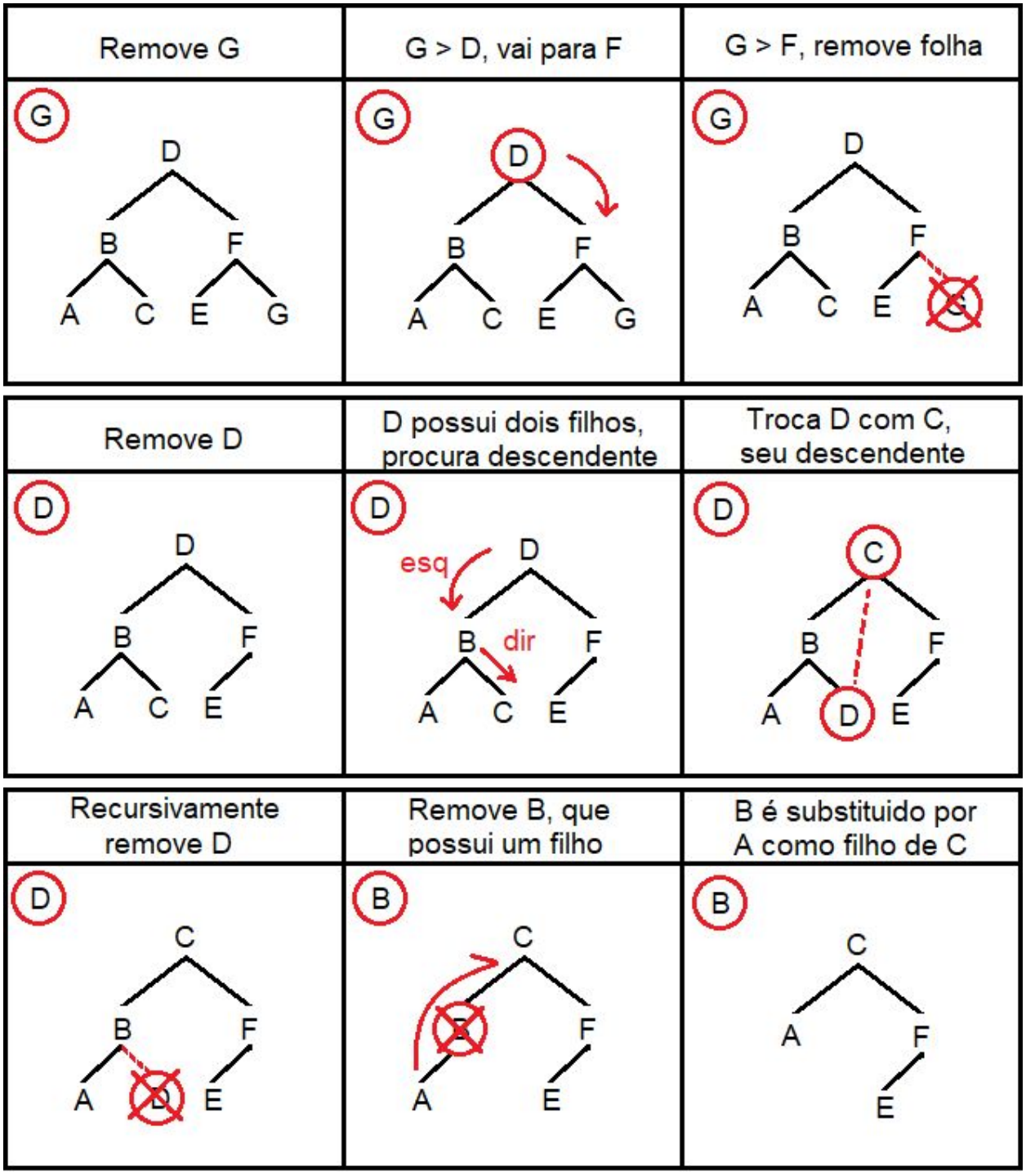
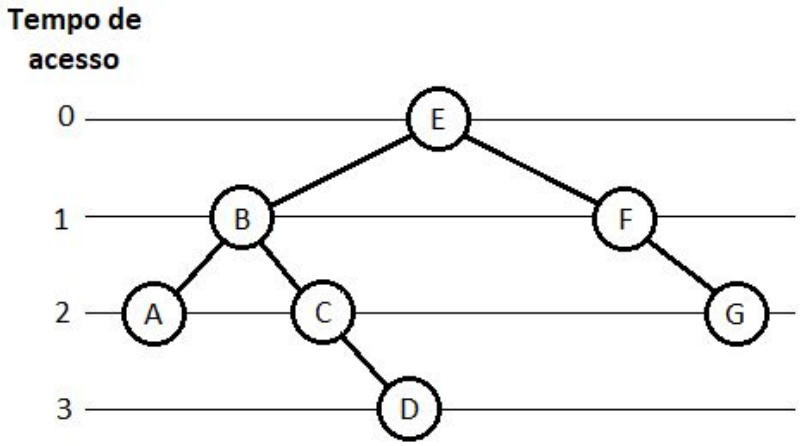


Figura 3: Exemplos de remoção

O objetivo do software ao organizar as pastas como uma árvore binária de busca é conseguir acesso eficiente a todos os programas. É sabido que caso a árvore gerada seja desbalanceada (a altura de uma sub-árvore seja muito maior que a outra), a eficiência desse acesso é comprometida. Portanto, o software deve oferecer operações para medir a velocidade de acesso à um programa e para consertar o balanceamento da árvore.

Em uma operação, o software testa a velocidade de acesso de um determinado programa (cujo nome é recebido do usuário). Nessa operação, o usuário insere o nome do programa a ser buscado e o tempo limite de teste, e o software inicia a busca na pasta “raiz”. Cada descida na árvore (ou seja, passar de um nó pai para um nó filho) significa que estamos abrindo uma nova subpasta, e esse processo leva uma unidade de tempo para ser realizado. Após encontrar o programa, o software responde ao usuário se a busca foi finalizada dentro do tempo estipulado. Por exemplo, se um programa se encontra em uma profundidade 3 (ou seja, para chegarmos nele partindo da raiz precisamos descer 3 vezes), isso significa que o software vai encontrá-lo em 3 unidades de tempo. Se o tempo máximo estipulado foi de 3 ou mais, o software informa que o programa passou no teste, caso contrário, o programa não passou no teste. A Figura 4 apresenta exemplos de teste de velocidade.



Nome	T max	T acesso	Resultado
A	3	2	O.K.
D	3	3	O.K.
E	0	0	O.K.
G	1	2	FALHA
C	0	2	FALHA

Figura 4: Exemplos de teste de velocidade em acesso aos programas

O software deve implementar um algoritmo de balanceamento. Na ordenação presente na árvore (que leva em conta a lexicografia do nome dos programas), é possível encontrar um programa cujo nome seja a mediana dentre todos os outros programas nessa ordenação. Esse programa divide todos os outros em “menores que ele” e “maiores que ele”. Esse programa é ideal para compor uma nova raiz e esses dois grupos de programas menores e maiores, seus filhos esquerdo e direito. Recursivamente também é possível definirmos quem seria a melhor raiz para as subárvores esquerda e direita que são formadas pelos programas restantes. Nem sempre existe um número ímpar de programas, portanto no caso de haver um número par, das duas medianas, deve-se escolher a maior. Durante esse processo o nome das pastas também

deve ser sempre atualizado de maneira a manter a regra descrita anteriormente[2]. A Figura 5 apresenta um exemplo de balanceamento esperado na operação.

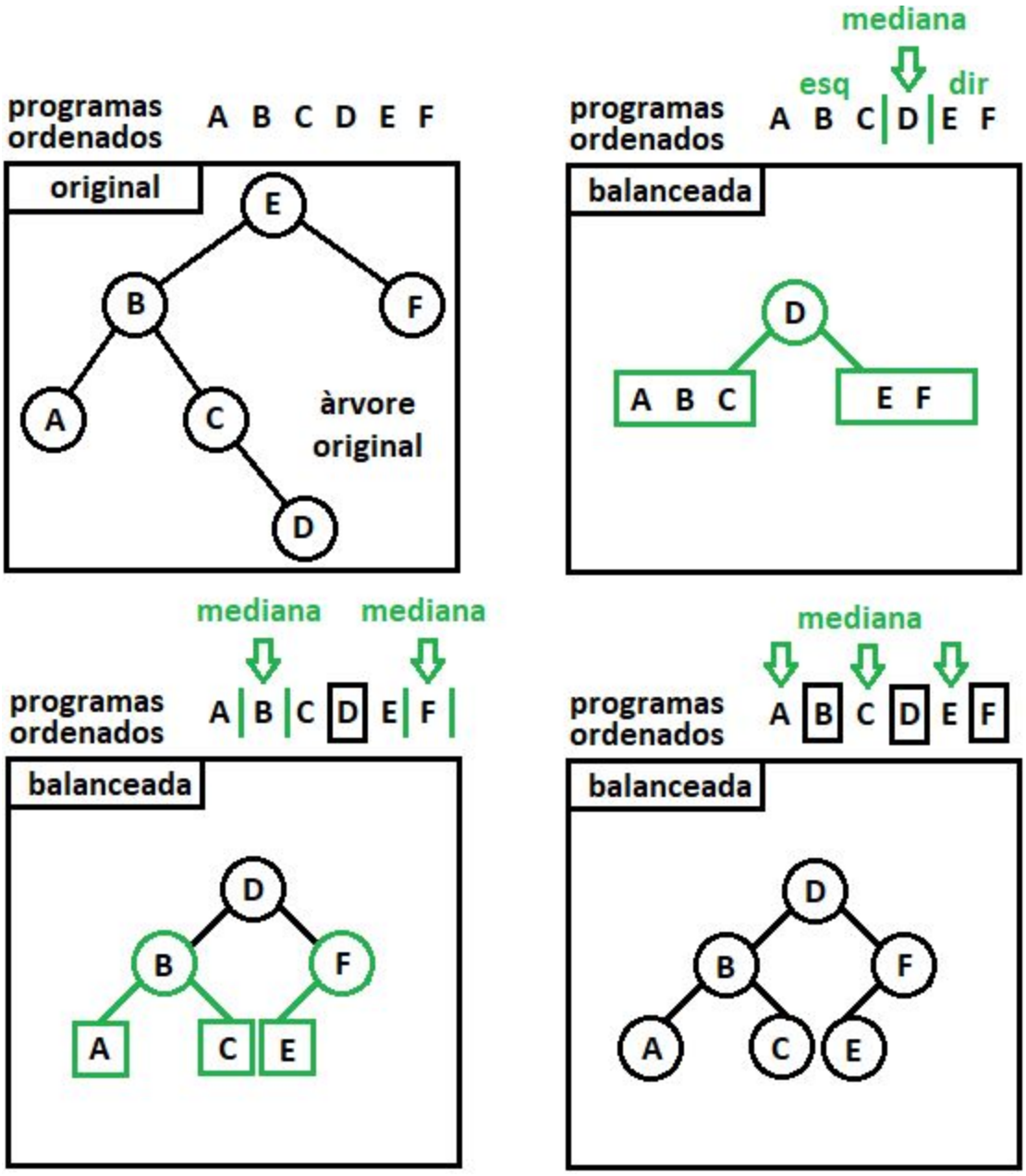


Figura 5: Exemplo de balanceamento (árvore original regenerada)

[2] A operação de balanceamento neste laboratório deve ser desenvolvida seguindo a idéia de regenerar a árvore binária de busca. Não confunda com balanceamentos baseados em rotação como das árvores do tipo AVL.

2 - Entrada

A primeira linha deve conter um número inteiro **P**, esse número indica quantidade de programas já presentes no computador ao iniciar o software. Na segunda e na terceira linha são dadas, respectivamente, as sementes geradoras em ordem simétrica (*in-order*) e pré-ordem (*pre-order*) desses programas, para que o software construa a árvore inicial. Além de construir essa árvore recebida, as sementes também são salvas como cópia de segurança inicial (Operação #5 da Tabela 1). Armazenar uma cópia de segurança é necessário para atender a operação #6 da Tabela 1.

Em seguida, deve-se informar as operações de entrada. O software deve implementar todas as operações descritas na Tabela 1. Cada operação é indicada por um comando numérico nas linhas de entrada e exigem a entrada de parâmetros adicionais. Uma entrada deve ser fornecida com várias linhas. Cada linha deve conter vários números inteiros, sendo o primeiro deles o código da operação (de acordo com #Operação na Tabela 1) e os demais os parâmetros conforme a coluna Parâmetros de cada operação. O nome de programas e pastas sempre será composto apenas de caracteres alfabéticos minúsculos, números e *underlines*, e nunca ultrapassará 30 caracteres incluindo o terminador '\0'.

Tabela 1: Operações de Entrada

#Operação	Nome	Parâmetros	Descrição
1	Instalar novo programa	Char[]: nome	Instala na árvore um novo programa com o **nome** passado por parâmetro na posição adequada. A pasta na qual ele for instalado também deve receber duas subpastas de mesmo nome acrescidos dos sufixos **_esq** e **_dir** (cf. Figura 2). Obs.: O nome de um programa sempre será único.
2	Desinstalar programa existente	Char[]: nome	Procura na estrutura de pastas o programa cujo **nome** foi especificado e o desinstala. Ou seja, o nó da árvore é removido (cf. Figura 3). A pasta da qual ele for desinstalado deve conter subpastas vazias, que serão deletadas. Caso o programa não exista, imprime a mensagem correspondente.
3	Testar velocidade de resposta	Char[]: nome Int: tempo	Efetua um teste de velocidade ao acesso de um determinado programa. O teste consiste em buscar o programa nas subpastas de maneira que caso ele esteja na pasta "raiz" o

			tempo de acesso seja instantâneo (0). Caso contrário, ele custa mais tempo quanto mais profundo o programa se encontra nas subpastas. A resposta dessa operação consiste em verificar se o tempo de busca do programa foi inferior/igual ou superior ao tempo desejado (cf. Figura 4).
4	Otimizar capacidade de resposta	-	Efetua uma reorganização da estrutura de pastas visando balancear a árvore de busca de programas. Essa operação deve seguir o algoritmo descrito no enunciado e suas respectivas convenções (cf. Figura 5).
5	Cria sementes de cópia de segurança	-	A partir da configuração atual da estrutura de pastas, gera sementes em ordem simétrica (*in-ordem*) e pré ordem. A Tabela 2 apresenta a especificação necessária para a saída de dados.
6	Restaura cópia de segurança	-	Restaura a configuração das pastas para a árvore formada pelas sementes salvas como cópia de segurança. Ou seja, destrói a árvore atual e recria a versão salva em backup.
7	Imprime todos os programas	-	Imprime todos os caminhos para os programas instalados em ordem lexicográfica. Obs.: Entende-se por caminho uma cadeia de caracteres começada por “C:/”, contendo o nome de todas as pastas que precisam ser acessadas para chegar a um determinado programa, seguidas de “/”, e por fim, o nome do programa seguido de “.exe”. Exemplo na Tabela 2.

Exemplo de Entrada:

```
7
aaa bbb ccc ddd eee fff ggg
ddd bbb aaa ccc fff eee ggg
1 hhh
2 eee
2 fff
2 ggg
```

```
2 zzz
3 ddd 0
3 aaa 2
3 hhh 0
7
5
4
7
6
7
```

3 - Saída

Tabela 2: Especificação da Saída das Operações

#Operação	Saída
1	[INSTALL] Programa **nome**.exe instalado com sucesso na pasta **pasta_pai** Obs.: **pasta_pai** é o nome da pasta na qual o programa foi instalado. Por exemplo, quando o programa D da Figura 1 é instalado, **pasta_pai** é “raiz”. Quando o programa E é instalado, **pasta_pai** é “F_esq”. Obs2.: “.exe” é só uma formatação da saída, o programa em si não contém essa extensão em seu nome.
2	[UNINSTALL] Programa **nome**.exe desinstalado com sucesso Ou [UNINSTALL] Nao foi encontrado no sistema nenhum programa com nome **nome**
3	[DELAY][FAIL] O acesso ao programa **nome**.exe ultrapassou o limite de **t** segundo Ou [DELAY][OK] O acesso ao programa **nome**.exe foi concluido em **tx** segundos Obs 1.: **t** se refere ao tempo máximo (parâmetro) que o software tem para encontrar o programa. Obs 2.: **tx** se refere ao tempo que o software levou para encontrar o programa.
4	[OPTIMIZE] O sistema de acesso a programas foi otimizado

5	[BACKUP] Configuracao atual do sistema salva com sucesso
6	[RESTORE] Sistema restaurado para a versao do backup
7	[PATHS] C:/**raiz**/**D_esq**/**B_esq**/**A**.exe C:/**raiz**/**D_esq**/**B**.exe C:/**raiz**/**D_esq**/**B_dir**/**C**.exe C:/**raiz**/**D**.exe C:/**raiz**/**D_dir**/**F_esq**/**E**.exe C:/**raiz**/**D_dir**/**F**.exe C:/**raiz**/**D_dir**/**F_dir**/**G**.exe Obs.: **programaX** se refere ao nome de um programa. **raiz**, **programaX_dir** ou **programaX_esq** se referem ao nomes de pastas. “C:/”, “/” e “.exe” são detalhes adicionais da saída. Esse exemplo representa a árvore da Figura 1.

Exemplo de saída:

```
[INSTALL] Programa hhh.exe instalado com sucesso na pasta ggg_dir
[UNINSTALL] Programa eee.exe desinstalado com sucesso
[UNINSTALL] Programa fff.exe desinstalado com sucesso
[UNINSTALL] Programa ggg.exe desinstalado com sucesso
[UNINSTALL] Nao foi encontrado no sistema nenhum programa com nome zzz
[DELAY][OK] O acesso ao programa ddd.exe foi concluido em 0 segundos
[DELAY][OK] O acesso ao programa aaa.exe foi concluido em 2 segundos
[DELAY][FAIL] O acesso ao programa hhh.exe ultrapassou o limite de 0 segundo
[PATHS]
C:/raiz/ddd_esq/bbb_esq/aaa.exe
C:/raiz/ddd_esq/bbb.exe
C:/raiz/ddd_esq/bbb_dir/ccc.exe
C:/raiz/ddd.exe
C:/raiz/ddd_dir/hhh.exe
[BACKUP] Configuracao atual do sistema salva com sucesso
```

```
[OPTIMIZE] O sistema de acesso a programas foi otimizado
[PATHS]
C:/raiz/ccc_esq/aaa.exe
C:/raiz/ccc_esq/aaa_dir/bbb.exe
C:/raiz/ccc.exe
C:/raiz/ccc_dir/ddd.exe
C:/raiz/ccc_dir/ddd_dir/hhh.exe
[RESTORE] Sistema restaurado para a versão do backup
[PATHS]
C:/raiz/ddd_esq/bbb_esq/aaa.exe
C:/raiz/ddd_esq/bbb.exe
C:/raiz/ddd_esq/bbb_dir/ccc.exe
C:/raiz/ddd.exe
C:/raiz/ddd_dir/hhh.exe
```

4 - Informações Adicionais

- O laboratório é em dupla;
- Não há número máximo de submissões;
- O programa deve estar completamente contido em um único arquivo nomeado lab07.c;
- Apenas um dos integrantes da dupla deve submeter a solução;
- No início do arquivo inclua como comentário, o nome e o RA dos integrantes da dupla, além de uma breve descrição dos objetivos do programa, as entradas e as saídas esperadas;
- Documente a solução através de comentários ao longo do programa e indente corretamente o código para melhor legibilidade;

Submissões detectadas como plágio receberão conceito zero.

5 - Critérios de Avaliação

- Seu programa deve passar pelos casos de teste definidos para o laboratório. Se positivo, os critérios de avaliação em seguida serão analisados:
    - Respeitou o enunciado;
    - Usou a estrutura de dados mais indicada para a solução;
    - Alocou e liberou memória adequadamente;
    - Organizou e indentou bem o código;