Trabalho 2
Árvores: Aplicação de Estrutura de Dados Avançada

Maio 2017

1 Prazo e formato

Este trabalho poderá ser realizado em **dupla ou individualmente** e sua entrega deverá ser feita até o dia **17/06/2017** às **23:00h**.

2 Introdução

ATENÇÃO: É importante que leia toda a especificação antes de começar, tome cuidado adicional com esse trabalho, já que ele não terá código inicial. Sugiro também que leia todas as notas de rodapé, para economizar esforço de implementação. Você não terá que implementar todas as funcionalidades das árvores e nem todas elas devem usar o algoritmo mais complexo, por isso atenção.

Árvore é um dos Tipos Abstratos de Dados (TAD) mais importantes, podendo ser utilizada de forma abrangente para representar computacionalmente uma miríade de conceitos. Por isso, neste trabalho você explorará melhor esse TAD e como ela pode ser aplicada na prática. Você ainda explorará melhor os *tradeoffs* envolvidos em duas estruturas de dados, uma árvore binária de busca (não necessariamente balanceada) e outra balanceada. Assim, poderá obter *insigts* práticos sobre cada uma das implementações, além de praticar e relembrar conceitos relacionados a elas.

Sua exploração deverá ser feita em duas etapas:

1. Implementação de dois tipos de árvores e resolução de um problema conhecido como *1D range search* ou busca por intervalo, seguindo as instruções descritas na Seção 3;
2. Análise mais profunda das suas implementações seguindo as instruções da Seção 4.

É importante que você siga a sequência acima, pois a segunda parte do trabalho depende da primeira. Por isso, antes de prosseguir para a segunda parte, tenha certeza sobre a corretude da primeira.

3 Implementação

Inicialmente, você deverá implementar duas árvores:

1. uma árvore binária de busca não necessariamente balanceada;
2. uma árvore binária de busca balanceada, podendo ela ser: Árvore AVL, Árvore Rubro-Negra ou qualquer outra que preferir.

Nessas duas implementações, você deverá obrigatoriamente implementar as funções listadas abaixo. Outras podem ser livremente adicionadas, desde que para cumprir os requerimentos aqui estabelecidos ou auxiliar no cumprimento deles.

- `cria_arvore` – inicializa a árvore com `NULL`;
- `insercao` – Função para inserção de nós em cada uma das árvores, ela deve ser feita respeitando as particularidades de cada estrutura de dados;
- `busca_por_intervalo` - Será discutidos na Seção 3.1;
- `verifica_se_eh_arvore_de_busca` - Será discutida na Seção 3.2;
- `percuso_em_ordem` - Função de exibição dos elementos da árvore seguindo o percurso em ordem;
- `libera` - Função para liberar a memória alocada pela árvore, já que seu programa não poderá ter vazamentos de memória e terá de liberar toda a memória alocada ao seu termino, ver Seção 4 para mais.

3.1 Busca Por Intervalo

A funcionalidade mais importante a ser implementada em ambas as árvores é a busca por intervalo, já que ela é o objetivo de seu trabalho. Contudo, considere primeiramente a seguinte estrutura que deverá estar contida em todos os nós:

```
1 typedef int elem_abb;
2
3 typedef struct no {
4   elem_abb indice;  // único membro obrigatório dessa estrutura
5   struct no *esq, *dir;
6 } No_arvore;
```

Esse elemento obrigatório (inteiro) terá a função de um índice e será por ele que a busca se dará. Essa busca por intervalo é uma versão simplificada do que acontece em bancos de dados, por exemplo em uma operação de busca para saber quais cidadãos cadastrados pagam imposto na faixa de R$ 1000 à R$ 3000 reais. Mais formalmente, seu programa receberá um conjunto inteiro P de pontos 1D, de tamanho n, onde, $P = \{p_1, p_2, \cdots, p_n\}$. Além disso ele também receberá como entrada um intervalo de busca, $[lo, hi]$[1]. Com base nisso, seu programa deverá retornar todos os pontos contidos em P que estejam entre lo e hi. Caso não existam números nesse intervalo de busca em sua árvore, sua busca deverá resultar em 0.

Para facilitar o entendimento considere o exemplo abaixo:

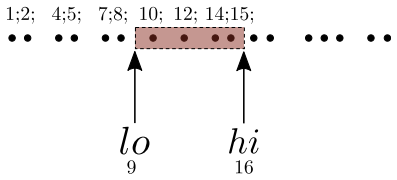

Figura 1: Intervalo de busca por pontos, sendo o destacado o objetivo da busca.

O intervalo resultante dessa busca é aquele que está destacado e este deverá fazer parte do resultado do seu programa, para mais veja a Seção 3.4 e a Seção 3.5. **Importante:** você precisa ter acesso aos valores contidos no intervalo na função `main`, apenas percorrer a árvore e exibi-los na função NÃO é suficiente.

3.2 Verificação Árvore de Busca

Seu programa, imediatamente após a leitura dos dados, deverá **obrigatoriamente** chamar, para ambas árvores, a função `verifica_se_eh_arvore_de_busca` para verificar se elas são realmente árvores binárias de busca. A implementação dessa funcionalidade deverá também identificar o seguinte caso, o qual NÃO REPRESENTA uma árvore de busca binária:

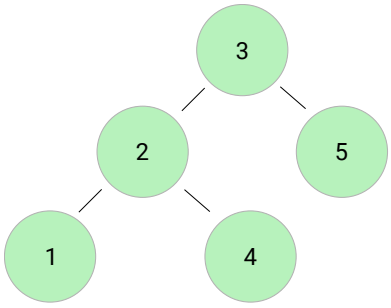

Figura 2: Caso que não representa uma árvore binária de busca e que deve ser identificado pela sua implementação.

Mesmo garantindo que na inserção dos dados na árvore esse caso não acontecerá, você ainda deverá fazer a implementação dessa funcionalidade. E caso essa função retorne indicando que a árvore não é uma árvore binária de busca, seu programa (`main`) deverá terminar imediatamente, retornando o valor -1. Contudo, tenha em

[1] Atenção na notação do intervalo, se lo e/ou hi estiverem presentes nas árvores, eles também deverão ser retornados.

mente que a intenção é que essa condição nunca aconteça durante os testes, por isso atente-se na implementação da inserção dos valores e na verificação da árvore.

3.3 Regras gerais de implementação

Além das já estabelecidas nas seções anteriores, sua implementação deve ainda:

- Estar totalmente contida nos seguintes arquivos, que deverão ser criados por você: `arvore_nao_balaceada.c`, `arvore_balaceada.c`, `arvore_nao_balaceada.h`, `arvore_balaceada.h` e `testa_arvores.c`. Caso seja necessário, poderá criar também arquivos adicionais para quaisquer outras TADs que utilizar, mas lembre-se de entregá-los, já que sem esses arquivos a compilação do seu programa não será possível;
- Podem ser definidas funções auxiliares, desde que respeitem as outras regras;
- E, por fim, embora seja possível utilizar trechos de implementação já prontos da internet (Desde que propriamente referenciada e NUNCA de um colega), recomenda-se fortemente que a implementação seja feita, para apreender/reforçar os conceitos utilizados. Notando que implementações copiadas da internet e não referenciadas serão anuladas ⇒ **nota zero na disciplina**.

Para aferir se sua implementação está correta e avaliar essa parte de seu trabalho, serão utilizados diversos casos de testes. O modelo deles está definido logo abaixo, é importante e requerido que seu trabalho faça exatamente o que é pedido.

3.4 Entrada

A entrada será composta por vários casos de teste. A primeira linha de um caso de teste contém dois números inteiros **N** e **M**, separados por um espaço em branco, indicando respectivamente quantos números (pontos) deverão ser lidos a seguir e o número de casos de busca ($1 \leq M \leq 200$). O caso de teste é então seguindo por uma linha, que contêm, os **N** inteiros separados por espaços[2], contendo os pontos. Os pontos devem ser inseridos nessa ordem na árvore. Cada inteiro deverá ser lido e armazenado nas duas árvores em questão, como um `indice`. Depois, o caso de teste tem ainda mais **M** linhas, que contêm, cada uma, dois inteiros separados por espaço (lo e hi), indicando um intervalo de busca a ser executado. Esse intervalo de busca, ainda deverá respeitar as seguintes condições: (i) ele nunca será maior que 200, ou seja, $hi - lo \leq 200$; (ii) o valor lo nunca será maior que hi, ou seja, ($lo \leq hi$). Essas verificações não são necessárias em seu programa. Caso ache necessário, você poderá aumentar o intervalo de busca para sua análise, mas saiba que para verificação do programa o limite será 200.

O último caso de teste é seguido por uma linha que contém apenas dois números zero, separados por um espaço em branco.

3.5 Saída

Para cada caso de teste, seu programa deverá imprimir duas linhas seguidas por **M** linhas. A primeira linha contendo o `percuso_em_ordem` da árvore balanceada[3] exibindo todos os valores contidos nela. A segunda contendo a quantidade de tempo médio (Tempo de relógio, não de CPU.) de busca por intervalo[4], de ambas as árvores separados por um espaço em branco (com, o tempo da árvore não balanceada primeiro) `em frações de segundo com 6 casas depois da vírgula`[5]. Depois dessas duas linhas, devem seguir **M** linhas, cada uma contendo os resultados de cada busca de uma árvore de sua escolha[6](os valores em si) **ordenados**[7].

3.6 Exemplo de Entrada e Saída

```
1 Entrada:
2 3 2
```

[2]Não necessariamente os inteiros lidos estarão em ordem e serão consecutivos.

[3]Embora a função `percuso_em_ordem` nunca seja chamada para a árvore que não necessariamente estará balanceada, ela ainda deve existir.

[4]Tenha certeza de estar medindo apenas o tempo de busca, ignorando qualquer outro processamento adicional, como por exemplo, a inserção de valores nas árvores.

[5]Para essa medição, sugiro que reaproveite o código do Trabalho 1.

[6]A busca deve ocorrer em ambas, mas apenas uma deverá ser exibida

[7]Aqui não é necessário gastar muito esforço, caso for implementar algum algoritmo de ordenação, como o número de valores contidos nesse intervalo nunca passará de 200, até o Bubble Sort será uma opção razoável. Ou ainda, caso quiser, poderá reaproveitar a solução de ordenação do Trabalho 1

```
1 2 3
1 5
1 1
6 2
16 23 42 4 8 15
1 8
32 37
2 1
1 8
2 4
0 0

Saída:
1 2 3
0.000001 0.000001
1 2 3
1
4 8 15 16 23 42
0.000001 0.000001
4 8
0
1 8
0.000001 0.000001
0
```

Para não deixar dúvidas, a saída deve ser composta por essas informações para cada caso de teste:

```
1 2 3             // Percurso em ordem da árvore balanceada
0.000001 0.000001 // Tempo médio de busca para cada árvore, começando com a árvore
    sem garantias de estar balanceada
1 2 3           // Primeiro caso de busca
1               // Segundo caso de busca
```

3.7 Arquivos de Entrada e Saída para teste

Juntamente com esse documento, você deve baixar dois arquivos: `teste.in` e `teste.out`. Esses arquivos possuem alguns casos de teste para uma auto-avaliação do correto funcionamento do seu programa. Para fazer o teste em uma distribuição Linux, levando em conta que o arquivo `teste.in` possui uma série de casos de teste e o arquivo `teste.out` contém os resultados esperados desses testes, execute os seguintes comandos:

Contudo, para realização dos testes abaixo você deverá comentar as linhas responsáveis por exibir o tempo gasto para busca dos intervalos — já que cada vez que executar o programa esses valores serão diferentes —, mas lembre-se que na a entrega final, essas linhas não poderão estar comentadas.

```
$ ./main < teste.in > minhasaida.out
```

Esse comando utilizará o arquivo `teste.in` como entrada para seu programa e salvará toda a saída dele no arquivo `minhasaida.out`. Com isso, você poderá comparar o `minhasaida.out` com a saída desejada, com o comando:

```
$ diff teste.out minhasaida.out
```

Se houver alguma diferença entre os arquivos, essa será impressa no terminal.

4 Análise da solução

A segunda parte do trabalho é uma discussão, que deve ser entregue em formato PDF, onde deverão ser analisadas as duas implementações de árvore. Essa análise deve seguir os seguintes tópicos[8].

[8]Não há nenhuma restrição em criar subtópicos, mas os aqui definidos devem ser seguidos e devem fazer parte de seu trabalho.

- Introdução
  - Descrever sobre os critérios ponderados na escolha da sua implementação de árvore balanceada. Justificar uso desse árvore e sua importância.
- Descrição das árvores
  - Breve descrição de cada uma das árvores;
  - Complexidade assintótica das operações de inserção e busca por intervalo em($Big\ O$).
- Detalhes de Implementação
  - Detalhes específicos de implementação. Por exemplo: usou função recursiva para busca de valores? como fez para retornar o conjunto de valores correspondentes ao intervalo que foi procurado? Fez alguma gambiarra para algo específico?
- Discussão e análise
  - Deve seguir a estrutura descrita na Seção 4.1.
- Conclusão
  - O que foi feito no seu trabalho?
  - Há oportunidades para melhora?
  - O que observou?

4.1 Análise teórica

Assim como pontuado na listagem da Seção 4, você terá que fazer uma análise do seu trabalho, mais especificamente uma comparação mais detalhada entre as implementações de árvore. Para fazer essa comparação, você deverá medir os fatores listados abaixo, realizando no **mínimo 3 e no máximo 5 experimentações**[9] com tamanhos diferentes para as árvores e observar o impacto causado por essa alteração (para cada experimentação, execute-a três vezes e use a mediana de cada valor para reportar seus resultados[10]):

- Tempo de inserção para cada uma das implementações (**NÃO** entregue o código medindo esse tempo, o propósito dele é apenas para a análise, para a entrega comente as chamadas de função para esse proposito);
- Tempo de busca dos valores para cada uma das implementações, assim como já discutido na Seção 3;
- A quantidade de rotações sofridas pela implementação de árvore balanceada, pois isso servirá de base para qualquer argumentação sobre o tempo de inserção dos valores.
- Uso total de memória[11]. Para essa medição você pode após a inicialização das árvores parar o programa, usando: *getchar();* e utilizar o próprio gerenciador de tarefas do seu sistema operacional para obter esse valor; ou usar o *valgrind*;
- Inclua também em sua análise um relatório do *valgrind* indicando que não há vazamentos de memória em sua implementação.

Para medição de tempo dos itens acima utilize tempo de relógio e tenha certeza de estar medindo apenas aquele item, por exemplo, ao medir a inserção não meça o tempo de procura dos valores (reaproveite o calculo de tempo do Trabalho 1). Lembre-se também que deverá medir o tempo em frações de segundo, usando seis casas decimais.

Reporte seus resultados utilizando qualquer meio que achar mais interessante ou conveniente (tabelas, gráficos..), mas garanta que associadas as essas observações esteja presente uma síntese desses dados, por exemplo: o tempo gasto para realizar a busca na árvore não balanceada cresce proporcionalmente ao número de objetos contidos nela.

Importante:

- Garanta que dois de seus 3 casos de teste sejam para o melhor e para o pior caso (ou pelo menos se aproximem deles e comente como isso foi feito);
- Lembre-se de reportar nos seus resultados qual o número de elementos nas árvores usou para cada caso.

[9] Caso veja necessidade em ultrapassar o mínimo de experimentações, as faça com o proposito de testar ou observar algum comportamento, evite criar experimentações sem objetivo.

[10] Isso é feito para evitar influência de outros fatores, como tarefas sendo executas no computador simultaneamente, normalmente esse número de execuções é bem maior para experimentações científicas, dependendo do que pretende.

[11] Aqui é desnecessário realizar três medições, já que se estiver inicializando as árvores da mesma forma, obterá o mesmo resultado.

5 Observações (LEIA COM ATENÇÃO)

- É necessária implementação de um algoritmo de busca por valores nas árvores, **NÃO** serão aceitos trabalhos onde a busca é feita copiando os valores da árvore para um vetor e depois realizando a busca.
- E lembre-se é necessário ter acesso aos valores contidos no intervalo de busca na função `main`.

6 Material a ser entregue

A entrega do trabalho será feita via Google Classroom. Você deverá entregar os dois arquivos que contém a implementação das árvores e suas funcionalidades: `arvore_nao_balaceada.c` e `arvore_balaceada.c`, inclusive os arquivos de cabeçalho: `arvore_nao_balaceada.h` e `arvore_balaceada.h`. Caso tenha definido TADs adicionais, também as entregue. Também deverá ser entregue seu arquivo de teste: `testa_arvores.c` e todos os arquivos que você utilizou de entrada e saída para teste (in e out). E, por fim, também deverá ser entregue seu relatório com as análises solicitadas na Seção 4, obrigatoriamente, em formato PDF. Junte todos esses arquivos em um arquivo .zip (ou .rar ou .tgz) e nomeie como: `nome1_nome2_trab1.zip`, onde `nome1` e `nome2` são os nomes dos integrantes que compõem a dupla.

7 Dúvidas

As dúvidas sobre este trabalho devem ser postadas no Google Classroom no mesmo local onde este trabalho está disponível.

Bom trabalho! Boa sorte!