Árvores 2-3 e Árvores Rubro-Negras

Prof. Dr. Juliano Henrique Foleis

Estude com atenção os vídeos e as leituras sugeridas abaixo. Os exercícios servem para ajudar na fixação do conteúdo e foram escolhidos para complementar o material básico apresentado nos vídeos e nas leituras. Quando o exercício pede que crie ou modifique algum algoritmo, sugiro que implemente-o em linguagem C para ver funcionando na prática. O único exercício que é necessário entregar está descrito na Seção "Atividade Para Entregar".

Vídeos

Árvores 2-3

Árvores Rubro-Negras

Leitura Sugerida

FEOFILOFF, Paulo. Estruturas de Dados. *Árvores 2-3* (Link)

FEOFILOFF, Paulo. Estruturas de Dados. *BSTs rubro-negras* (Link)

Exercícios dos materiais de leitura sugerida

Exercícios 1.1, 2.1, 2.2, 2.3, 3.1, 3.2 da página do Prof. Feofiloff (Árvores 2-3): (Link)

Exercícios 1.1, 1.2, 1.3, 1.4, 1.5, 1.7, 3.1, 3.2, 3.3, 3.4 da página do Prof. Feofiloff (Árvores Rubro-Negras): (Link)

Exercícios

1. Implemente as operações a seguir em uma árvore 2-3 em C:

- Inserção
- Busca
- Altura

As chaves e os valores correspondentes podem ser números inteiros.

DICA: A implementação da árvore 2-3 em C é bem trabalhosa. Faça se tiver tempo!

2. Existem duas medidas de altura para uma árvore rubro-negra. A primeira é a medida tradicional de altura, que é o número máximo de arestas percorridas a partir da raíz para atingir algum nó folha da árvore. Implemente essa função em *int ARN_Altura(ARN *A)* tanto de forma ansiosa quanto de forma preguiçosa.

3. A outra medida de altura de uma árvore rubro-negra é a altura negra. Neste caso, é o número de arestas negras percorridas a partir da raíz para atingir qualquer nó folha da árvore. Implemente essa função em *int ARN_AlturaNegra(ARN *A)* tanto de forma ansiosa quanto de forma preguiçosa.

4. O custo médio de uma busca bem-sucedida considerando que todos os nós são igualmente prováveis de serem buscados pode ser estimada pela expressão a seguir:

$$\left\lceil \frac{1}{N} \sum_{i=1}^{N} C[i] \right\rceil$$

tal que $C[i]$ é a profundidade do nó i na árvore, N é o número de nós da árvore e $\lceil\ \rceil$ é a função teto. Implemente a função *int ARN_CustoBuscaBemSucedida(ARN* A)*.

5. O custo médio de uma busca mal-sucedida considerando que todas as possíveis chaves tem a mesma probabilidade de serem buscadas pode ser estimada pela expressão a seguir:

$$\left\lceil \frac{1}{L} \sum_{i=1}^{L} (C[i] + 1) \right\rceil$$

tal que $C[i]$ é a profundidade do nó folha i na árvore, L é o número de nós folhas da árvore e $\lceil\ \rceil$ é a função teto. Implemente a função *int ARN_CustoBuscaMalSucedida(ARN* A)*.

6. Faça uma função *float ARN_Rubros(ARN* A)* que retorne a percentagem de *links* rubros de uma ARN. Para N = { 10000, 100000, 1000000 } insira N chaves aleatórias em uma árvore inicialmente vazia e anote a percentagem de links rubros da árvore resultante. Para cada N, repita o experimento 30 vezes. Calcule a média e o desvio padrão e anote em uma tabela.

Atividade para Entregar

A atividade a seguir é para ser feita individualmente e entregue via Moodle no tópico da Semana 3. A data-limite para entrega é dia 02/11/2021 às 23:55. Em caso de cópia as atividades dos participantes serão desconsideradas.

Descrição da Atividade

Um percurso em-ordem de uma árvore de busca binária qualquer (ABB, AVL, ARN, etc) visita os nós da árvore em ordem crescente. Isto pode ser explorado para implementar um algoritmo de ordenação, conforme segue:

```
ENTRADA: vetor V com N inteiros
1. Crie uma ABB A
2. Insira todos os elementos de V em A
3. Faça um percurso em-ordem de A, inserindo os elementos de volta em V
4. Destrua a arvore A
```

a. Implemente a função *void ABB_Sort(int* v, int n)* conforme o pseudocódigo acima. Se precisar de mais parâmetros, crie uma função ABB_Sort_R que contenha os parametros adicionais, e use ABB_Sort apenas como uma *wrapper*. Use uma ABB (estudada na semana 2) como a estrutura auxiliar da ordenação.

b. Implemente a função *void ARN_Sort(int* v, int n)* conforme o pseudocódigo acima. Se precisar de mais parâmetros, crie uma função ABB_Sort_R que contenha os parametros adicionais, e use ABB_Sort apenas como uma *wrapper*. Use uma árvore rubro-negra como a estrutura auxiliar da ordenação.

c. Para N = {1000, 10000, 100000, 500000} ordene um vetor com **N inteiros em ordem decrescente** usando as funções implementadas nos itens **a** e **b**. Anote o tempo de execução na Tabela 1. Na Tabela 2 anote a altura da árvore logo após o passo 2 do algoritmo de ordenação descrito acima. Caso alguma execução não possa ser concluída, apenas anote na tabela o(s) caso(s) que isso aconteceu!

d. Para N = {1000, 10000, 100000, 500000} ordene um vetor com **N inteiros gerados aleatoriamente** usando as funções implementadas nos itens **a** e **b**. Para cada N e cada função execute a ordenação com 10 vetores aleatórios. Anote o tempo de execução médio entre todas as execuções e o desvio padrão na Tabela 3. Na Tabela 4 anote a média da altura das árvores logo após o passo 2 do algoritmo de ordenação descrito

acima para todas as execuções, juntamente com o desvio padrão. Use a função `int* random_vector(int n, int max, int seed)` implementada na semana 1 para fazer a geração dos vetores aleatórios: use *max=10*N* as sementes *10, 20, 30, 40, 50, 60, 70, 80, 90, 100* para gerar cada um dos 10 vetores aleatórios.

e. Houve diferença significativa entre o tempo de execução entre os dois métodos avaliados no caso da ordenação dos vetores inicialmente em ordem decrescente? Se sim, qual foi o melhor método? Qual foi o pior? Justifique suas respostas.

f. Houve diferença significativa entre o tempo de execução entre os dois métodos avaliados no caso da ordenação dos vetores aleatórios? Se sim, qual foi o melhor método? Qual foi o pior? Justifique suas respostas.

g. Em cada um dos métodos (usando ABB, ARN), o tempo de execução foi muito diferente entre a ordenação dos vetores inicialmente em ordem descrescente e dos vetores aleatórios? Em qual método houve a maior variação? E a menor variação?

h. No item anterior você percebeu que em um dos métodos a variação do tempo de execução entre a ordenação de vetores inicialmente em ordem descrescente e dos vetores aleatórios é grande. Como você pode diminuir essa diferença?

i. (OPCIONAL) Implemente a idéia descrita no item **h** e refaça os experimentos dos itens **c** e **d**. Compare com as demais abordagens.

Você deve Entregar

Entregue em formato .zip os arquivos a seguir:

- Os arquivos-fonte desenvolvidos nos itens **a**–**b**, bem como os arquivos-fonte criados para realizar os testes. Faça um *Makefile* para compilar o seu programa. Modularize conforme julgar necessário.
- As Tabelas preenchidas nos itens **c** e **d** e as respostas dos itens **e**–**h** em um único *pdf*.

Por favor entregue como especificado acima!

A data-limite para entrega é dia 02/11/2021 às 23:55.

BONS ESTUDOS!

	N			
	1000	10000	100000	500000
ABB				
ARN				

Figure 1: Tempo de Execução (em s) para Ordenar Vetores com N Elementos em Ordem Decrescente

	N			
	1000	10000	100000	500000
ABB				
ARN				

Figure 2: Altura das Árvores Antes do Percurso Em-Ordem

	N			
	1000	10000	100000	500000
ABB				
ARN				

Figure 3: Tempo de Execução (em s, média +- desvio) para Ordenar Vetores Aleatórios com N Elementos

	N			
	1000	10000	100000	500000
ABB				
ARN				

Figure 4: Altura das Árvores Antes do Percurso Em-Ordem