Relatório EP1

Gabriel Henrique Pinheiro Rodrigues NUSP: 112.216-47

Abril 2020

I Implementação

As tabelas de símbolos foram implementadas seguindo o padrão <Chave, Valor> sendo Chave do tipo string e Valor do tipo int, portanto quando uma chave ou valor não é encontrado na tabela, é retornado respectivamente " " e -1;

1 Vetor Desordenado

1.1 Insere $O(n)$

É necessário verificar se o elemento a ser inserido pertence a tabela ou não, se não pertence insere no fim.

1.2 Devolve $O(n)$

Não há relação de ordem, logo é necessário checar todos os elementos da tabela até encontrar.

1.3 Remove $O(n)$

Não foi utilizado lazy deletion, logo é necessário buscar o elemento e redimensionar o array.

1.4 Rank $O(n)$

Percorre toda a tabela, comparando a chave dos elementos com a chave analisada.

1.5 Seleciona $O(n^2)$

Percorre toda a tabela, chamando a função rank para cada elemento até encontrar um com rank igual a k.

2 Vetor Ordenado

2.1 Insere $O(n)$

É necessário verificar se o elemento a ser inserido pertence a tabela ou não, se não pertence insere e desloca todos os elementos maiores do que ele.

2.2 Devolve $O(\log_2 n)$

Há relação de ordem, logo é possível fazer busca binária.

2.3 Remove $O(n)$

Não foi utilizado lazy deletion, logo é necessário buscar o elemento ($\log_2 n$) e redimensionar o array (Pior caso: O(n)).

2.4 Rank $O(\log_2 n)$

Como há relação de ordem entre os elementos, basta procurar o primeiro elemento maior do que o elemento da chave.

2.5 Seleciona $O(1)$

Como não foi utilizado lazy deletion o elemento de rank k é o elemento de índice k.

3 Lista Ligada Desordenada

3.1 Insere $O(n)$

É necessário verificar se o elemento a ser inserido pertence a tabela ou não, se não pertence insere no início.

3.2 Devolve $O(n)$

Não há relação de ordem, logo é necessário checar todos os elementos da tabela até encontrar.

3.3 Remove $O(n)$

A eliminação do elemento é $O(1)$, porém a busca do elemento é $O(n)$.

3.4 Rank $O(n)$

Percorre toda a tabela, comparando a chave dos elementos com a chave analisada.

3.5 Seleciona $O(n^2)$

Percorre toda a tabela, chamando a função rank para cada elemento até encontrar um com rank igual a k.

4 Lista Ligada Ordenada

4.1 Insere $O(n)$

É necessário verificar se o elemento a ser inserido pertence a tabela ou não, se não pertence insere de maneira ordenada.

4.2 Devolve $O(n)$

Há relação de ordem, mas não há acesso imediato a qualquer elemento da tabela, logo é necessário avançar até o elemento com a chave correspondente.

4.3 Remove $O(n)$

A eliminação do elemento é $O(1)$, porém a busca do elemento é $O(n)$.

4.4 Rank $O(n)$

Percorre a tabela enquanto as chaves são menores do que a chave analisada.

4.5 Seleciona $O(n)$

Percorre a tabela até chegar no elemento de índice k.

5 Árvore de Busca Binária

Abaixo será utilizado a nomenclatura **B para representar ABBs balanceadas** e **BN para as não balanceadas**, quando uma ABB não está balanceada ela se assemelha a uma lista ligada ordenada.

5.1 Insere B: $O(\log_2 n)$, NB: $O(n)$

Se o nó não faz parte da tabela, procura de maneira ordenada onde o nó deverá entrar.

5.2 Devolve B: $O(\log_2 n)$, NB: $O(n)$

Como há relação de ordem e cada nó possui no máximo dois filhos, a busca é logarítmica, caso a árvore não esteja balanceada ela se assemelha a uma lista ligada (busca $O(n)$).

5.3 Remove B: $O(\log_2 n)$, NB: $O(n)$

Busca o elemento ($O(\log_2 n)$), se é folha remove direto, se é raiz procura o menor elemento da sub árvore direita e substitui.

5.4 Rank B: $O(\log_2 n)$, NB: $O(n)$

Cada nó armazena a quantidade de filhos da esquerda e da direita, logo na busca se a chave procurada está a direita do nó analisado é somado 1 + a quantidade de filhos da esquerda desse nó ao rank, quando a chave é encontrada adiciona-se os filhos da esquerda desse nó.

5.5 Seleciona B: $O(\log_2 n \cdot \log_2 n)$, NB: $O(n^2)$

Realiza uma busca e para cada nó analisado chama a função rank.

6 Treaps

Como um treap junta o conceito de ABB com max Heap, a probabilidade dessa árvore ser balanceada é maior do que a ABB, um exemplo é o dicionário, que é o pior caso de uma ABB construída pela ordem de entrada.

6.1 Insere $O(\log_2 n)$

Se o nó não faz parte da tabela, procura de maneira ordenada onde o nó deverá entrar, insere e realiza rotações para manter a propriedade max heap da árvore.

6.2 Devolve $O(\log_2 n)$

Como há relação de ordem e cada nó possui no máximo dois filhos, a busca é logarítmica.

6.3 Remove $O(\log_2 n)$

Busca o elemento ($O(\log_2 n)$), se é folha remove direto, se é raiz procura o menor elemento da sub árvore direita e substitui.

6.4 Rank $O(\log_2 n)$

Cada nó armazena a quantidade de filhos da esquerda e da direita, logo na busca se a chave procurada está a direita do nó analisado é somado 1 + a quantidade de filhos da esquerda desse nó ao rank, quando a chave é encontrada adiciona-se os filhos da esquerda desse nó.

6.5 Seleciona $O(\log_2 n \cdot \log_2 n)$

Realiza uma busca e para cada nó analisado chama a função rank.

7 Árvores 23

As árvores 23 são balanceadas.

7.1 Insere $O(\log_2 n)$

A inserção foi implementada de maneira recursiva.

7.2 Devolve $O(\log_2 n)$

A busca tem complexidade melhor caso: $O(\log_3 n)$.

7.3 Remove $O(\log_2 n)$

A remoção foi implementada de maneira iterativa.

7.4 Rank $O(n)$

Percorre toda a árvore e determina quantos elementos são menores do que a chave analisada.

7.5 Seleciona $O(n^2)$

Percorre toda a árvore e para cada nó chama a função rank.

8 Árvores Rubro-Negras

As árvores rubro-negras são balanceadas

8.1 Insere $O(\log_2 n)$

A inserção foi implementada de maneira iterativa.

8.2 Devolve $O(\log_2 n)$

Como há relação de ordem e cada nó possui no máximo dois filhos, a busca é logarítmica.

8.3 Remove $O(\log_2 n)$

Busca o elemento ($O(\log_2 n)$), se é folha remove direto, se é raiz procura o menor elemento da sub árvore direita e substitui.

8.4 Rank $O(\log_2 n)$

Cada nó armazena a quantidade de filhos da esquerda e da direita, logo na busca se a chave procurada está a direita do nó analisado é somado 1 + a quantidade de filhos da esquerda desse nó ao rank, quando a chave é encontrada adiciona-se os filhos da esquerda desse nó.

8.5 Seleciona $O(\log_2 n \cdot \log_2 n)$

Realiza uma busca e para cada nó analisado chama a função rank.

9 Tabelas de Hashing

A tabela de hashing foi implementada usando lista encadeada, o tamanho M escolhido foi 7000, pois foi realizado testes e para casos de 35000 palavra, a maior lista ligada possuia tamanho 8.

9.1 Função Hash

A função cria uma hash utilizando todos os caracteres da chave, o número primo utilizado foi 23.

9.2 Insere $O(1)$

Gera a hash do elemento e o insere.

9.3 Devolve $O(1)$

Gera a hash a partir da chave e acessa o elemento.

9.4 Remove $O(1)$

Gera a hash do elemento e o remove da sua posição.

9.5 Rank $O(M)$

Percorre toda a tabela e determina quantos elementos são menores que a chave analisada.

9.6 Seleciona $O(n * M)$

Para cada elemento da tabela chama a função rank.

II Testes

10 Tempo de construção da tabela

10.1 Livro: Metamorphosis - Size: 117Kb

O tempo abaixo representa uma média de uma amostra de 5.

ESTRUTURA	TEMPO MÉDIO(s)
VD	0.1867764
VO	0.0991724
LD	0.4201814
LO	0.4938202
AB	0.0234594
TR	0.0413610
A23	0.0253700
RN	0.0347956
HS	0.0182056

RANKING

ESTRUTURA	TEMPO MÉDIO(s)
HS	0.0182056
AB	0.0234594
A23	0.0253700
RN	0.0347956
TR	0.0413610
VO	0.0991724
VD	0.1867764
LD	0.4201814
LO	0.4938202

10.2 Livro: Holy Bible - Size: 4,6 Mb

O tempo abaixo representa uma média de uma amostra de 5.

ESTRUTURA	TEMPO MÉDIO(s)
VD	31.974740
VO	6.566988
LD	147.3511
LO	355.5532
AB	0.6862456
TR	0.7487126
A23	0.468637
RN	0.5458928
HS	0.2705492

RANKING

ESTRUTURA	TEMPO MÉDIO(s)
HS	0.2705492
A23	0.468637
RN	0.5458928
AB	0.6862456
TR	0.7487126
VO	6.566988
VD	31.974740
LD	147.3511
LO	355.5532