NOME: Gabriel Ferreira de Souza Araujo **NUSP: 12718100**

RELATÓRIO EP2 - ALGORÍTMOS E ESTRUTURAS DE DADOS II

1. RESUMO

O código realizado é uma implementação de estruturas de dados para tabelas de símbolos, que são estruturas usadas para armazenar pares de chave-valor, semelhantes a dicionários. As estruturas de dados implementadas são:

1. Árvore de Busca Binária (ABB): A classe “NodeABB” representa um nó da ABB. A classe contém um ponteiro para a raiz da ABB (“rootABB”). A função `addABB` adiciona um novo nó à ABB, enquanto a função “findABB” procura um nó com a chave especificada.

2. Árvore Rubro-Negra (ARN): A classe “NodeARN” representa um nó da ARN. A classe “SymbolTable” contém um ponteiro para a raiz da ARN (“rootARN”). A ARN é usada para armazenar as chaves e valores da tabela de símbolos, garantindo que a árvore esteja balanceada. As funções “addARN” e “findARN” são usadas para adicionar e pesquisar nós na ARN, respectivamente.

3. Árvore 2-3: A classe “NodeA23” representa um nó da árvore 2-3. A classe contém um ponteiro para a raiz da árvore 2-3 (“rootA23”). As funções “addA23”, “findA23” e “splitA23” são usadas para adicionar, buscar e dividir nós na árvore 2-3, respectivamente.

OBS: Neste EP2, não consegui fazer funcionar árvore 2,3, pois tive problemas na função split(), mais especificamente no tratamento da raiz e a redistribuição recursiva de novos nós que estão abaixo da árvore. Em testes de poucas palavras, tive o resultado esperado, mas para testes a partir de 30 palavras, tive os problemas citados.

4. Árvore de Busca Aleatória (TR): A estrutura “NodeTR” representa um nó da TR. A classe contém um ponteiro para a raiz da TR (“rootTR”). A TR é usada para armazenar as chaves e valores da tabela de símbolos, onde cada nó tem uma prioridade aleatória. A função “addTR” adiciona um novo nó à TR e a “findTR” realiza a uma busca na árvore.

5. Vetor Ordenado (VO): A estrutura “nodeVO” representa um elemento do VO. A classe contém um ponteiro para o vetor “nodeVector”. A função “addVO” adiciona um novo elemento no vetor por meio de uma busca binária e a função “findVO” realiza a busca binária de uma chave.

Além disso, a classe `Item` é usada para representar os valores associados às chaves na tabela de símbolos. E por fim, na classe SymbolTable, temosas funções “add” e “value” que adicionam e atribuem os Itens necessários de acordo com a estrutura selecionada.

2. TESTES

Texto Desordenado com Repetição, N = 779125:

ABB -> Por ser desordenado, espera-se que a ABB seja em média balanceada e por isso, vai ter um bom desempenho no tempo. Tempo(s): 6.461000

ARN -> Essa estrutura é sempre balanceada, então sabemos que no pior caso temos O(log n). Tempo(s): 6.337000

TR -> Essa estrutura é balanceada com certa probabilidade, como o N é razoavelmente grande, podemos assumir que a estrutura é em média balanceada. Tempo(s): 8.259000

VO -> Essa estrutura é ineficiente em casos de N grande, por tanto é esperado um desempenho pior. Tempo(s): 18.251000

Texto Ordenado sem Repetição, N = 245366:

ABB -> O programa ficou rodando por mais de um minuto e depois tive erro de segmentação na mamória, esse comportamento foi esperado, pois, o caso de dados ordenados é o pior caso da ABB e para N grande é muito ineficiente. Tempo(s): inf

ARN -> Como é uma estrutura balanceada, foi obtido um bom desempenho nesse caso. Tempo(s): 4.292000

TR -> Neste caso, a Arvore Treap teve menor tempo de execução que a ARN, mas é esperado que em caso específicos, as duas estruturas obtêm desempenho próximo. Tempo(s): 3.9040000

VO -> Como neste caso, tempos palavras ordenadas, o vetor ordenado terá complexidade O(n^2) e temos um tempo bem mais longo. Tempo(s) -> 101.252000

Texto Desordenado sem Repetição, N = 245366:

ABB -> Por ser desordenado, espera-se que a ABB seja em média balanceada, além disso, como não tem repetições, espera-se que seja melhor que o caso com repetição, por isso, vai ter um bom desempenho no tempo. Tempo(s): 4.4060000

ARN -> Essa estrutura é sempre balanceada, então sabemos que no pior caso temos O(log n) e seu desempenho tende a ser melhor que ABB na maioria dos casos. Tempo(s): 4.0920000

TR -> Essa estrutura é balanceada com certa probabilidade, como o N é razoavelmente grande, podemos assumir que a estrutura é em média balanceada, além disso, como não tem repetição, espera-se que o desempenho melhore. Tempo(s): 4.4220000

VO -> Neste caso, era esperado que essa estrutura obtivesse um tempo mais longo dada sua complexidade, entretanto, neste EP, a estrutura ficou rodando em tempo infinito. Tempo(s) = inf.

Texto Desordenado com Repetição, N = 1000000:

ABB -> Por ser desordenado, espera-se que a ABB seja em média balanceada e por isso, vai ter um bom desempenho no tempo. Tempo(s): 9.2580000

ARN -> Essa estrutura é sempre balanceada, então sabemos que no pior caso temos O(log n), entretanto, neste caso, a ARN teve um tempo superior ao da ABB. Este comportamento não foi esperado, mas observando a organização do teste, pode-se notar que existem muitas palavras repetidas e que foram geradas de forma aleatória, diferentemente das anteriores, e assim, a ABB pode ter sido superior nesta situação. Tempo(s): 13.1950000

TR -> Essa estrutura é balanceada com certa probabilidade, como o N é razoavelmente grande, podemos assumir que a estrutura é em média balanceada. Tempo(s): 9.1810000

VO -> Essa estrutura é ineficiente em casos de N grande, por tanto é esperado um desempenho pior, entretendo, neste caso, o resultado do tempo foi razoável se comparado à estrura ARN, com uma diferencia média de 2 segundos, tal resultado não foi esperado mas esse fator pode estar relacionado com a organização do teste. Tempo(s): 15.832000

OBS: As consultas requisitadas no enunciado do EP foram todas testadas com os exemplos do enunciado e alguns outros, assim, foi possivel verificar a funcionalidade de cada consulta

3. Como Rodar

O código abre um arquivo “input” organizado da seguinte forma:

1. Estrutura

2. Número de Palavras

3. Texto

4. Número de Consultas

5. Consulta 1

6. Consulta n

Nos arquivos de teste anexados, temos um exemplo de como deve ser organizado o arquivo de entrada. A saída do código é impressa no terminal de acordo com os critérios do enunciado do EP. (importante que o arquivo de teste esteja no mesmo diretório do código e tenha o nome “input.txt”)

4. Considerações Finais

Durante os testes realizados, pudemos observar o desempenho e eficiência de cada estrutura em diferentes cenários. No caso de textos desordenados com repetição, a ABB demonstrou um bom desempenho, uma vez que, em média, a estrutura se mantém balanceada. A ARN também obteve resultados satisfatórios, pois é uma estrutura sempre balanceada. A TR mostrou-se, em média, balanceada e teve um desempenho aceitável. No entanto, o VO apresentou um desempenho significativamente inferior, especialmente em casos com um grande número de palavras.

Para textos ordenados sem repetição, a ABB apresentou um comportamento ineficiente, principalmente em cenários com um grande número de palavras. A ARN e a TR obtiveram bons resultados, sendo a TR ligeiramente mais rápida em alguns casos específicos. O VO, devido à sua complexidade, teve um tempo de execução bastante longo.

Já para textos desordenados sem repetição, a ABB mostrou-se eficiente, aproveitando o fato de a estrutura se manter balanceada e não possuir repetições. A ARN teve um desempenho semelhante ao da ABB, uma vez que também é uma estrutura sempre balanceada. A TR, assim como nos casos anteriores, obteve um tempo de execução razoável. Porém, novamente, o VO apresentou um desempenho insatisfatório e, neste caso específico, entrou em um loop infinito.

Em relação aos casos de teste com um número maior de palavras, observamos comportamentos não esperados. A ABB, que se mostrou eficiente em casos anteriores, teve um desempenho semelhante à ARN, provavelmente devido à organização aleatória das palavras repetidas no texto. A TR manteve-se balanceada e teve um bom desempenho, enquanto o VO continuou demonstrando ineficiência.