Trabalho 2, parte 1

→ LEMBREM-SE DE USAR PAPEL E CANETA COMO RASCUNHO ANTES DE IMPLEMENTAR <<--

Trabalho 2, parte 1: data limite para submissão: **14/06/2018 as 17:59**
(aviso: parte do assunto utilizado neste trabalho será ministrado após a segunda prova de INF213)

Considere o algoritmo de ordenacao *insertion sort* . Como todos sabem, a ordem de complexidade de tempo dele e' $O(n^2)$ no pior caso.

Dado um array de inteiros (que podem ser armazenados em 4 bytes), quantos movimentos do tipo "mover elemento" o algoritmo *insertion sort* faria para ordená-lo?

Uma forma trivial de se calcular isso consiste em ordenar o array e contar o numero de movimentos. Por exemplo, o array 5 1 8 4 9 exige um total de 6 movimentos para ser ordenado.

Explicação (o caractere "|" indica o final da porcao ja ordenada do array na iteração atual) :
5 1 8 4 1 (entrada)
5 | 1 8 4 1 (passo 1, 0 movimentos)
1 5 | 8 4 1 (passo 2, 1 movimento para inserir o numero 1)
1 5 8 | 4 1 (passo 3, 0 movimento para inserir o numero 8)
1 4 5 8 | 1 (passo 4, 2 movimentos para inserir o numero 4)
1 1 4 5 8 | (passo 5, 3 movimentos para inserir o numero 1)

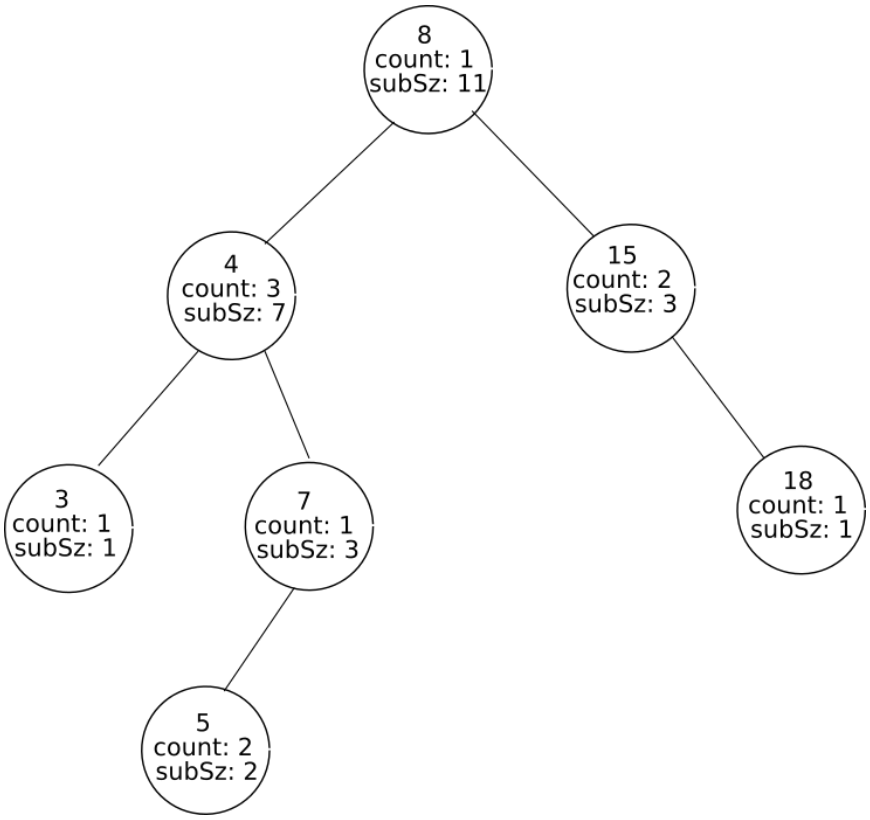
Porém, para calcular o total de movimentos não é necessário ordenar o array! Estude o algoritmo de inserção (faça vários exemplos) e descubra qual informação precisamos ter em cada passo para calcular o número de movimentos nesse passo.

Uma árvore binária de pesquisa pode ser utilizada para calcular essa informação de forma eficiente.

Modifique a classe MySet.h (disponivel em: https://drive.google.com/open?id=1nEjjn9gZHvCObO0fQs6DW8646AI00fnc) para que ela possa calcular essa informação. Para isso, você precisará modificar os nodos de modo que cada nodo armazene dois campos extras:

- Um contador *itemCount* para o numero de copia do elemento no nodo possuimos. Tradicionalmente em um conjunto não armazenamos elementos repetidos. Como nossa estrutura de dados precisará armazenar repetições, basta que cada nodo tenha um contador indicando quantas instâncias desse elemento há nela.
- Um inteiro *subTreeSize* indicando o número total de elementos na subárvore cuja raiz é o nodo em questão.

A figura abaixo ilustra uma árvore gerada após se inserir os elementos (nessa ordem): 8,4,15,3,7,5,5,18,4,4,15



Após adaptar nossa classe para armazenar tais valores (e para atualizá-los de forma correta a medida em que elementos são inseridos), crie uma função-membro que usa uma dessas informações para fazer uma consulta que será útil para resolver o problema proposto neste trabalho.

Observacoes:

- Neste trabalho você DEVERÁ utilizar uma árvore binária de pesquisa adaptada conforme explicado acima. Porem, ha outras formas de resolve-lo.
- Você não precisará implementar métodos para remover elementos da árvore.
- Para simplificar, suponha que a árvore gerada pelos casos de teste nunca ficará muito desbalanceada.

- Tente entender qual e' a "informação secreta" que você precisa obter da árvore sem a ajuda de terceiros (exercícios como esse são muito bons). Se após muito esforço você não encontrar a solução, peça ajuda ao professor e/ou a colegas.
- Na biblioteca padrão do C++ (e de outras linguagens) há uma implementação pronta de árvore binária de pesquisa. Porém, ela não fornece o tipo de informação que a árvore que você adaptara irá fornecer. Uma das aplicações de se conhecer bem estruturas de dados e' poder fazer estruturas customizáveis como essa quando as versões disponíveis nas bibliotecas não nos provê as funcionalidades que precisamos.

Formato da entrada:

Seu programa deverá ler da entrada padrão (stdin) uma string METODO, um número inteiro N e, após isso, os N números representando o array que deveria ser ordenado.

Se MÉTODO for "ORDENAR" (sem aspas), ordene o array utilizando o insertion sort e conte o numero de movimentos.

Se MÉTODO for "ARVORE" (sem aspas), faça a contagem utilizando a árvore binária descrita acima para acelerar os cálculos.

Espera-se que o método ARVORE será muito mais rápido do que o método ORDENAR (seu programa não será avaliado utilizando entradas grandes quando o método ORDENAR for utilizado).

Restrições da entrada:

METODO sera ou a string ORDENAR ou ARVORE
Se METODO for ORDENAR, 0 <= N <= 1000
Se MÉTODO for ARVORE, 0 <= N <= 2,000,000
Cada inteiro do array estara no intervalo [-2,000,000,000, 2,000,000,000]
Seu programa deverá ser capaz de processar a entrada em menos de 10 segundos em um computador razoável (ele será compilado com a flag O3 do g++).

Exemplo de entrada 1:

ARVORE
5
5 1 8 4 1

Saida esperada 1:

6

Exemplo de entrada 2:

ORDENAR
5
5 1 8 4 1

Saida esperada 2:
6

Exemplo de entrada 3:
ORDENAR
6
-1 0 8 20 30 100

Saida esperada 3:
0

Arquivo README

Seu trabalho deverá incluir um arquivo README.

Tal arquivo contera:

- Seu nome/matricula
- Informacoes sobre fontes de consulta utilizadas no trabalho
- Um pequeno relatório com as três informacoes descritas abaixo.

No seu arquivo README, indique a ordem de complexidade das duas versões do seu algoritmo (ARVORE e ORDENAR), supondo que a entrada terá tamanho N e que a árvore binária gerada estará sempre balanceada.

Além disso, faça experimentos com entradas contendo arrays (com números aleatórios) de tamanhos diferentes. Crie uma tabela no seu README com o tempo que cada versão do seu algoritmo gastou para processar cada entrada (mencione o tamanho das entradas na sua tabela) .

Finalmente, escreva um pequeno parágrafo descrevendo suas conclusões sobre esse experimento.

Submissao

Submeta seu trabalho utilizando o sistema Submitty ate a data limite. Seu programa sera avaliado de forma automatica (os resultados precisam estar corretos, o programa não pode ter erros de memoria, etc), passara por testes automaticos “escondidos” e a qualidade do seu codigo sera avaliada de forma manual.

Você devera enviar o arquivo README e todo o codigo fonte do seu trabalho (ele será compilado com o comando g++ *.cpp -O3 -Wall)

Duvidas

Dúvidas sobre este trabalho deverão ser postadas no sistema Piazza. Se esforce para implementá-lo e não hesite em postar suas dúvidas!

Avaliacao manual

Principais itens que serao avaliados (alem dos avaliados nos testes automaticos):
- Comentarios
- Indentacao
- Nomes adequados para variaveis
- Separacao do codigo em funcoes logicas
- Uso correto de const/referencia
- Uso de variavies globais apenas quando absolutamente necessario e justificavel (uso de variaveis globais, em geral, e' uma ma pratica de programacao)
- Etc

Regras sobre plagio e trabalho em equipe

- Este trabalho devera ser feito de forma individual.
- Porém, os alunos podem ler o roteiro e discutir ideias/algoritmos em alto nivel de forma colaborativa.
- As implementações (nem mesmo pequenos trechos de codigo) não deverão ser compartilhadas entre alunos. Um estudante não deve olhar para o código de outra pessoa.
- Crie um arquivo README (submeta-o com o trabalho) e inclua todas as suas fontes de consulta.
- Não poste seu codigo (nem parte dele) no Piazza (ou outros sites) de forma publica (cada aluno e' responsavel por evitar que outros plagiem seu codigo).
- Trechos de código não devem ser copiados de livros/internet. Se você consultar algum livro ou material na internet essa fonte deverá ser citada no README.
- Se for detectado plagio em algum trecho de codigo do trabalho a nota de TODOS estudantes envolvidos sera 0. Alem disso, os estudantes poderao ser denunciados aos orgaos responsaveis por plagio da UFV.