Trabalho Prático 2

QuickSort

Leandro Freitas de Souza – 2021037902

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

leandrofreitas@dcc.ufmg.br

1. Introdução

A atividade proposta foi realizar, por meio de um QuickSort, a ordenação de um conjunto de strings de acordo com uma ordem determinada por um alfabeto, ou ordem lexicográfica.

2. Método

O programa foi desenvolvido na linguagem C, e compilado pelo compilador GCC, da GNU Compiller Collection. Além disso, a máquina utilizada no desenvolvimento do programa possui 8GB de memória RAM, e um processador Intel Core I5, de 11ª Geração.

2.1. Estruturas de Dados

A única estrutura de dados utilizada na implementação do programa foi uma array de strings, para realizar o quicksort. A alocação dessa array se deu de forma dinâmica, de forma a possibilitar que o problema receba entradas de tamanhos muito grandes.

2.2 Funções

O arquivo “ordenacao.h” fornece funções extremamente úteis, tanto para a leitura efetiva dos dados, formatação das strings e para a implementação do QuickSort.

Para o funcionamento da ordenação, a ordem lexicográfica foi armazenada em um vetor de inteiros de tamanho 26, sendo cada posição correspondente a uma letra em ordem alfabética. O valor de cada posição desse vetor é correspondente à posição da letra na ordem lexicográfica nova. Dessa forma, esse vetor se comporta de forma semelhante a um mapeamento, que associa cada letra com o seu valor. O cálculo dessa estrutura foi realizado em uma função chamada GetOrder.

Uma vez armazenada, a nova ordem lexicográfica foi usada nas funções strLcmp e strGcmp, que funcionam como os operadores "<" e ">", respectivamente. Tais funções nortearam os algorítmos de ordenação que foram utilizados.

Por fim, foram implementadas as funções de QuickSort e Insertion Sort, que realizam a ordenação do conjunto de strings de acordo com a ordem proposta.

Em adição às funções que auxiliam a ordenação das strings, uma função importantíssima presente no programa é a de leitura das strings, que, em tempo de leitura, ignora caracteres indesejados e transforma todas as letras da string em letras minúsculas.

- QuickSort

O funcionamento do QuickSort leva em consideração dois parâmetros fornecidos na execução do código: o tamanho máximo de uma array para se fazer um Insertion Sort, e a quantidade de elementos que vão ser utilizados para fazer uma mediana e realizar a partição.

Uma vez recebidos os parâmetros e escolhido o pivô x a partir de uma mediana de m elementos, o QuickSort funciona da seguinte maneira:

1- Executa o algorítmo de partição, que percorre o vetor a partir de dois ponteiros, um começando em sua direita e outro em sua esquerda, até que A[i] >= x e A[j] <=x, e trocar A[i] com A[j], e realize tal processo até que os ponteiros se cruzem.

2 – Chamar, de forma recursiva, a partição para tamanhos menores do vetor, até que ele esteja ordenado. (Obs: caso o tamanho do vetor seja menor ou igual ao tamanho máximo recebido na execução, realizar um insertion sort para esse trecho).

3. Análise de Complexidade

3.1. Tempo

Primeiramente, existem aspectos importantes para serem considerados no cálculo da complexidade do programa, tais como a complexidade para a leitura das strings, que é O(n), sendo n o número de caracteres da entrada, uma vez que o programa realiza uma operação para cada caractere lido, tornando esse tempo linear.

Outro aspecto a ser considerado é o tamanho médio de cada string, que impacta diretamente na complexidade do QuickSort. Essa definição será importante para a definição de um caso médio, uma vez que a complexidade do QuickSort será medida em função do número de palavras existentes na entrada, e independe do número de caracteres.

Dadas as considerações, temos três casos a serem considerados: um melhor caso, que conta com uma entrada que, independentemente de seu tamanho, possui um número desprezível de palavras em relação ao número de caracteres, um caso médio, que conta com strings de tamanho médio, e um pior caso, em que o número de strings é próximo ao número de caracteres.

No primeiro caso, temos que, uma vez que o número de strings é desprezível em relação ao número de caracteres, pode-se considerar a complexidade do QuickSort constante. Nesse sentido, a complexidade do programa será igual à complexidade de sua leitura, que é O(n).

No segundo caso, por sua vez, temos que, seja n o número de caracteres na entrada e x o tamanho médio das strings, temos que o programa receberá por volta de n/x strings. Como x é uma constante razoável,

temos que a complexidade assintótica do QuickSort será O(nlog(n/x)) = O(nlog(n)). Assim, a complexidade do programa como um todo será O(nlog(n)) + O(n) = O(nlog(n)).

Por fim, no terceiro caso, temos que, como o número de strings é próximo de n, a complexidade do QuickSort será O(nlog(n)), e, assim, o programa possuirá complexidade O(nlog(n)) + O(n) = O(nlog(n)).

Obs: Como foi utilizado o método da mediana de m elementos na implementação do QuickSort, seu pior caso ($O(n^2)$) foi desconsiderado.

3.2. Espaço

Uma vez que o QuickSort não utiliza muito espaço auxiliar de memória, temos que o programa não armazena muito mais que o espaço de um vetor de strings, ou seja, a complexidade de espaço do programa é O(n).

4. Estratégias de Robustez

Para garantir a robustez do programa, foram seguidas algumas estratégias.

Uma das estratégias utilizadas foi a implementação de uma função que, em tempo de leitura, ignore qualquer tipo de caractere indesejado, que possa interferir na execução da ordenação.

Além disso, a alocação dinâmica é de extrema importância para que o programa receba entrada de variados tamanhos. A alocação das strings de entrada se dava da seguinte forma. Um vetor de tamanho inicial 32 foi alocado. Caso houvesse a necessidade de aumentar o vetor, era realocado o dobro de seu tamanho.

5. Análise Experimental

A análise experimental foi feita por meio da execução dos casos testes disponibilizados no moodle, que não apresentaram nenhuma anomalia. Além disso, utilizei um gerador de strings aleatórias para gerar 10000 strings de tamanho 1, que representa algo próximo ao pior caso do programa. O comportamento também se deu de forma esperada.

6. Conclusão

A partir da realização desse programa, foi possível entender de forma mais clara o funcionamento do algorítmo do QuickSort. Além disso, foi possível aprofundar meus conhecimentos na linguagem C, que foi escolhida justamente para aprimorar os meus conhecimentos na manipulação de strings sem utilizar a biblioteca std::string.

A principal dificuldade encontrada na implementação do programa foi a capacidade de filtrar os acentos de strings. Não consegui achar uma forma efetiva de realizar esse procedimento, e, infelizmente, o programa não é capaz de formatar strings desse tipo.

No entanto, na minha opinião, o objetivo mais importante do trabalho foi atingido, que foi entender melhor o funcionamento e a adaptação de algoritmos de ordenação, e tratamento de strings.

7. Referências

https://virtual.ufmg.br/20221/course/view.php?id=11607

http://www.unit-conversion.info/texttools/random-string-generator/

Apêndice: Instruções para a execução do programa

Para a execução do programa, basta, em primeiro lugar, utilizar o terminal para acessar o diretório “TP” e executar o comando “make

all”. Uma vez executado, será gerado um executável no diretório “bin” com o nome “tp.exe”. Ao executar esse arquivo, devem ser fornecidos quatro parâmetros: o nome do arquivo com as informações de entrada, precedido por –i, o nome do arquivo de saída, precedido por –o, o valor do número de elementos a serem utilizados para se calcular a mediana, precedido por –m, e o tamanho máximo de uma array para que seja utilizado o InsertionSort, precedido por –s.