Trabalho Prático 2
O problema da agenda de viagens de Rick Sanchez

Daniel Ferreira Abadi[1] **- 2018088062**

[1]Universidade Federal de Minas Gerais

1. Introdução

O problema abordado consiste em uma forma de ordenar uma agenda de viagens pelo tempo necessário de cada uma e depois por ordem alfabética em cada mês. Inicialmente nos é dado o tempo máximo por mês, o número de planetas a serem visitados e o tamanho dos nomes dos planetas.

Para fazer a primeira ordenação, que é a de tempo, foi utilizado o algoritmo "Merge Sort" e para a segunda ordenação foi utilizado o algoritmo "Radix Sort", de complexidades O(n logn) e O(n*k), respectivamente.

2. Implementação

O problema exigia que tínhamos que utilizar um algoritmo de complexidade O(n logn) e estável para ordenar em relação ao tempo, e o único passado em sala de aula com tais características é o "Merge Sort". Já para a ordenação em relação ao nome, era exigido algum algoritmo com complexidade O(n*k), sendo k o tamanho da entrada, e o único com tal complexidade visto em sala foi o "Radix Sort".

Optei fazer um struct do tipo Planeta, que contém 3 atributos, um inteiro para guardar o tempo, uma string para guardar o nome e outro inteiro para armazenar o mês no qual Rick irá aquele planeta. Foi criado um vetor do tipo Planeta para armazenar todos os planetas e para serem realizadas as ordenações em cima do mesmo.

O compilador utilizado foi o G++, o padrão do Linux.

2.1. Merge Sort

O algoritmo recebe como parâmetros um vetor de planetas, a primeira posição e a última posição do vetor e não retorna nada. Ele divide o vetor até sobrar apenas um elemento em cada partição por meio da recursividade. Logo em seguida esses vetores menores são ordenados e juntados pela função Merge, até o ponto em que o vetor com duas divisões seja passado e ordenado.

2.2. Radix Sort

Este método recebe como parâmetros um vetor de planetas, a primeira posição do vetor, a última posição do vetor, o tamanho do vetor original e o tamanho do nome dos planetas. E não retorna nada.

Se faz necessário receber a primeira e a última posição do vetor e também o tamanho do vetor original pois o vetor é passado em pedaços para a função, devido aos meses.

Ele utiliza outro método chamado Counting Sort, que utiliza um vetor para auxiliar na ordenação, onde verifica o nome dos planetas de trás pra frente e assim ordena em ordem alfabética.

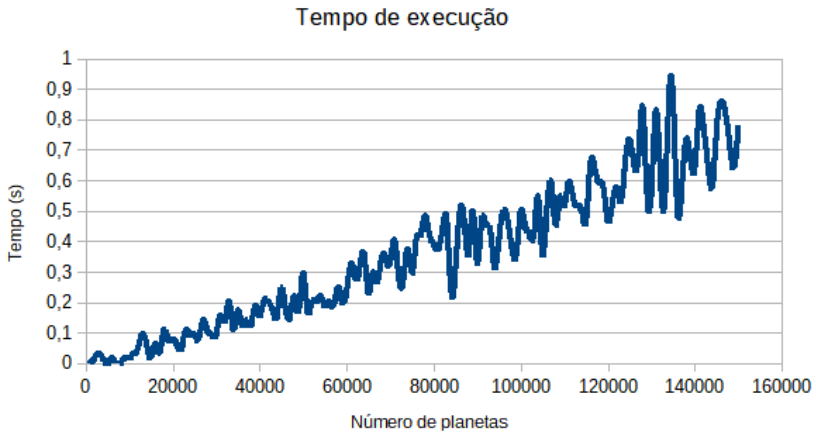
3. Instruções de compilação e execução

Para compilar digite ”make”, para executar o programa digite ”./tp2”. Já para executar e fazer os testes digite ”make test”. Para limpar os arquivos gerados pela compilação digite ”make clean”.

4. Análise de complexidade

Como foi dito anteriormente, a complexidade temporal dos algoritmos de ordenação são O(n logn) e O(n*k) no melhor caso, caso médio e pior caso. Já a complexidade espacial para o ”Merge Sort” é O(n logn), no pior caso, e para o ”Radix Sort” é O(n+s), também no pior caso, sendo ”s” o tamanho do alfabeto. Porém há ainda os laços dentro da função ”main” que servem para alocar os planetas, decidir em qual mês cada planeta fica e imprimir a agenda, sendo todos O(n).

Para analisarmos melhor a complexidade do programa, torna-se necessário a exposição de um gráfico alternando o tamanhos das entradas.



A curva se mostra bastante inconsistente, mas isso se deve ao fato de só definirmos o número de planetas. Porém, devido a sua complexidade quase linear, o comportamento do programa não se altera muito conforme a variação do tamanho da entrada, sendo de 1000 até 150000 pulando de 1000 em 1000.

5. Conclusão

Foram implementados os dois algoritmos de ordenação citados acima, utilizando um TAD do tipo planeta com 3 atributos, sendo eles tempo a ser gasto no planeta, nome e o mês que ele vai ser visitado.

O conjunto do trabalho é realizado de forma muito rápida e eficaz, mesmo para entradas muito grandes, devido a complexidade pequena dos algoritmos de ordenação. Tal fato me surpreendeu bastante pois os problemas ordenação são conhecidos por serem caros e demorados.

6. Referências

Foram utilizados os slides vistos em sala de aula para decidir qual algoritmo usar. O ”Merge Sort” foi adaptado do site ”geeksforgeeks” [1] . E o ”Radix Sort” foi implemen-

[1] https://www.geeksforgeeks.org/merge-sort/

tado baseado no código do site ”stackoverflow” [2] e nos slides do monitor Álisson Renan Svaigen.

[2] https://stackoverflow.com/questions/29978088/c-sorting-strings-using-lsd-radix-sor