Trabalho Prático 2 - Estruturas de Dados

Matheus Aquino Motta[1]

[1]Bacharelado em Matemática Computacional
DCC - Universidade Federal de Minas Gerais

matheusmotta@dcc.ufmg.br

Abstract. *The Rick Sanchez's schedule problem is another optimization modeling of a problematic scenario, where an intergalactic well known scientist, Rick Sanchez, wants to visit several planets in the universe. Where given a time month upper bound to visit different planets in the universe he wants to go to as most places as possible under the month time limit. Therefore, our problem is how to define the best combination and sequence of trips that he must take in order to solve the optimization problem, after this part our goal is to visit the monthly chosen planets in lexicographic order by its names. Hence, we need to treat the input planets list in order to get our answer, the algorithm that will be presented below shows a solution to this problem, using two different sorting algorithms, the Merge Sort and the Radix LSD sort, since at first we need to sort the planets by its time weight, then after by its name in lexicographic order.*

Resumo. *O problema da medição de Rick Sanchez consiste na modelagem e otimização de um cenário problemático, onde um cientista intergalaticamente conhecido, Rick Sanchez, quer visitar alguns planetas do universo. Onde dado um limite superior mensal de tempo para visitar diferentes planetas, ele deseja visitar o maior número possível de lugares sob o tempo limite mensal. Desse modo, o problema consiste em como definir a melhor sequência de combinações de viagens que ele deve escolher visando solucionar o problema de otimização, além disso a segunda parte do nosso objetivo consiste em visitar os planetas escolhidos para um determinado mês em ordem lexicográfica pelo seus nomes. Assim, precisamos tratar a lista de planetas de entrada, com objetivo de obter uma solução ótima, e o algoritmo apresentado abaixo mostra uma solução para esse problema, usando dois algoritmos de ordenação, o Merge Sort e o Radix Sort por digito menos significativo (LSD), haja vista que primeiramente precisamos ordenar a entrada de planetas pelo seu peso em tempo, e então depois pelo seu nome em ordem lexicográfica.*

1. Introdução

O problema da medição de Rick Sanchez consiste em um problema de decisão onde um cientista deseja visitar o maior número de planetas possível em um mês sob um limite superior de tempo, visitando-os em ordem alfabética mensalmente, a partir de seus respectivos nomes.

Nesse sentido o algoritmo desenvolvido apresenta uma simples abordagem de implememtação para a resolução do problema por meio de dois alogoritmos de ordenação, o Merge Sort e o Radix Sort LSD (Dígito menos significativo).

2. Implementação

Nessa seção iremos descrever a solução utilizada e suas e suas respectivas funções, assim como a implementação dos algoritmos de ordenação utilizados.

2.1. Merge Sort

O Merge Sort é um algoritmo estável de complexidade $\mathcal{O}(nlogn)$ que foi implementado de modo que a função *"SortMerge(Lista, Início, Fim)"* recebe como parâmetros o vetor de planetas recebido pela entrada, a posição de início do vetor e final do vetor, trivialmente definidos como 0 e o número total de planetas menos um, na primeira chamada da função. Essa que por sua vez "particiona" o vetor em dois subvetores *Left* e *Right* recursivamente de modo a dividir o vetor $log_2 n$ vezes alterando os parâmetros de *"Início"* e *"Fim"* da função, substituindo-os pelo elemento médio

$$Meio = \frac{Início + Fim}{2}$$

no *"Início"* e *"Fim"* para os subvetores *Left* e *Right* respectivamente de forma recursiva.

Assim, após a última divisão em subvetores, i.e, assim que os vetores tiverem tamanho igual a um, os subvetores serão "concatenados" em ordem crescente, por *Backtracking* a partir da função *"Merge(Lista, Início, Meio, Fim)"*, que basicamente pega os dois últimos subvetores e os concatena em ordem crescente.

Para implementação, todas as comparações foram feitas utilizando como parâmetro para a comparação a variável tempo respectiva a cada planeta, e sua ordenação foi feita em função do objeto planeta.

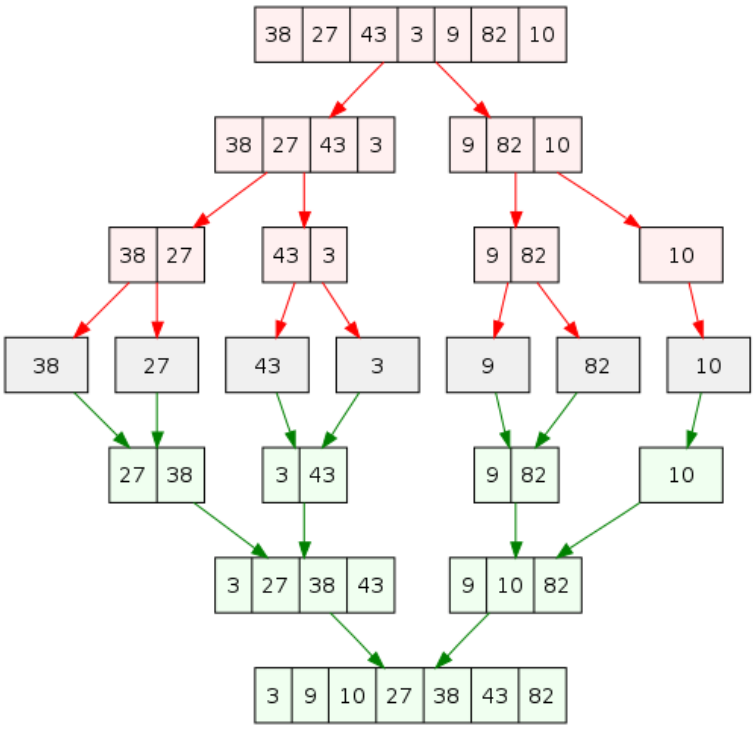


Figure 1. Merge Sort

Na imagem temos um exemplo da execução do *Merge Sort*, onde em vermelho temos a função *"SortMerge"* e em verde a função *"Merge"*.

2.2. Partição em meses e Radix Sort

Após a ordenação do vetor em função da váriavel tempo de cada planeta a próxima etapa do funcionamento do algoritmo consiste no particionamento da lista a partir da soma de pesos sob um limite mensal pré definido pela entrada.

Para isso, iremos percorrer a lista de planetas de modo encontrar subvetores cuja a soma dos seus respectivos x_i tempos seja menor ou igual ao limite estabelecido para um mês, de um valor l até um valor r definido pelos k planetas que couberem em um determinado mês. Onde l é trivialmente definido como 0 para o primeiro mês, representando o início do vetor de planetas e redefinido em função do valor k anterior para os potenciais cálculos seguintes.

$$\max_{k} \sum_{x=l}^{k} x_i \leq time$$

. Desse modo, iremos obter os limites dos subvetores que representam cada mês, que serão ordenados pelo *Radix Sort* em função do nome dos planetas a a partir da ordem lexicográfica dos caracteres da *string*, do digito de menor significância ao dígito de maior significância (LSD), por meio do *Counting Sort* que foi, um algoritmo estável de complexidade $\mathcal{O}(k)$ para strings, onde k é o tamanho do alfabeto.

O *Counting Sort* foi implementado de modo que, recebendo o index da *string* a ser avaliado de no nome dos planetas o vetor de planetas fosse percorrido e esses por sua vez salvos temporariamente em uma matriz de tamanho $26 \times (número\ de\ planetas)$, onde cada linha representa os planetas com a letra i, onde $0 \leq i \leq 25$, no index avaliado pela função. Destarte, após avaliar o subvetor para cada index da *string* do nome dos planetas os planetas são ordenados em ordem crescente e estável de acordo com a ordem lexicográfica, onde é garantida a estabilidade do algoritmo, que por sua vez possui uma complexidade $\mathcal{O}(n \times k)$, onde k é o tamanho do alfabeto e n o tamanho da *string* nome dos planetas, que por sua vez é igual para todos.

3. Análise de Complexidade

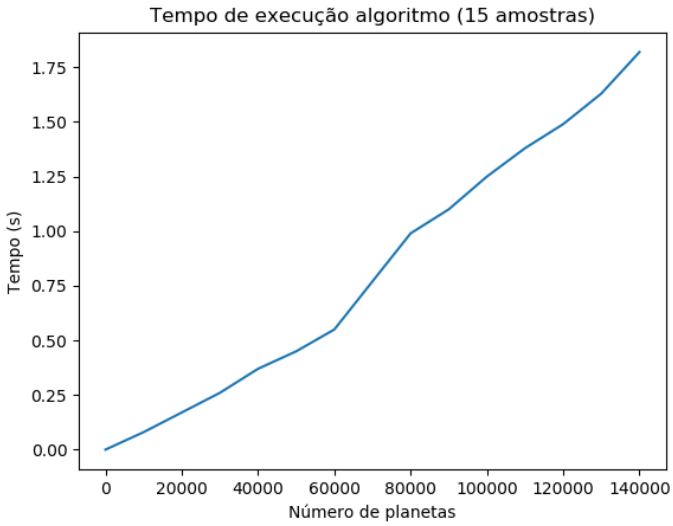
Sabemos que o algoritmo consiste basicamente em duas ordenações dos vetor de planeta em função de suas duas váriaveis, inteira numérica *"Tempo"* e *"Nome"* lexicográfica, por meio do *Merge Sort* e *Radix Sort* respectivamente, portanto, assumindo n como o número de planetas existentes na lista e k o tamanho do nome dos planetas, onde $k < \log_2 n$:

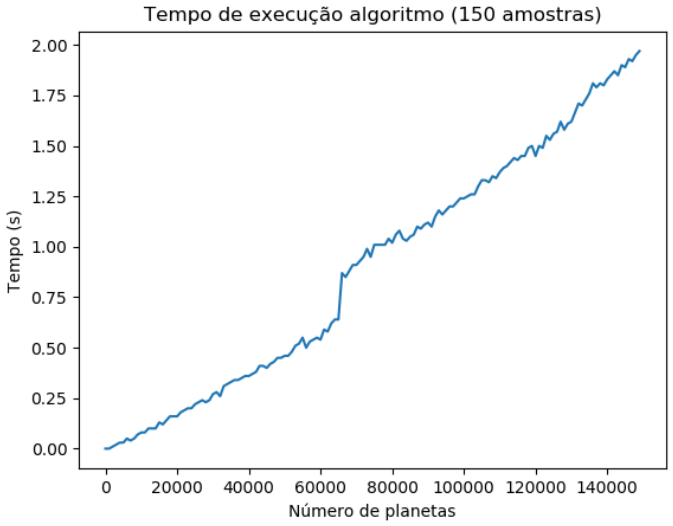
$$f(n,k) = \mathcal{O}(nlogn) + \mathcal{O}(n \times k)$$

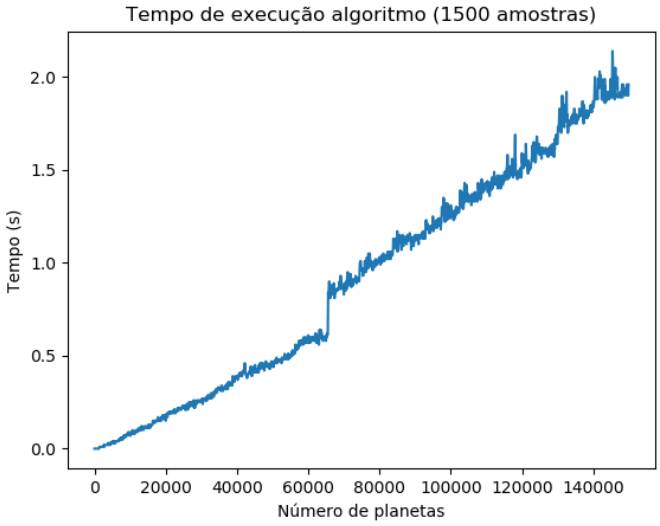
Onde k, tamanho do alfabeto é assumido como um valor constante 26.

$$\begin{aligned} f(n) &= \mathcal{O}(nlogn) + \mathcal{O}(n \times 26) \\ &= \mathcal{O}(nlogn) \end{aligned}$$

Isso de dá, devido ao fato de que a função *Merge Sort* domina assintoticamente a função *Radix Sort*, portanto a complexidade final é da ordem de $\mathcal{O}(nlogn)$.


Tempo de execução algoritmo (15 amostras)
Tempo (s)
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
0
20000
40000
60000
80000
100000
120000
140000
Número de planetas


Tempo de execução algoritmo (150 amostras)
Tempo (s)
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0
20000
40000
60000
80000
100000
120000
140000
Número de planetas


Tempo de execução algoritmo (1500 amostras)
Tempo (s)
0.0
0.5
1.0
1.5
2.0
0
20000
40000
60000
80000
100000
120000
140000
Número de planetas

O que podemos visualizar a partir dos gráficos de análise experimental gerados a partir de amostras geradas aleatoriamente. Os testes foram realizados em um Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz 8GB RAM.

4. Instruções de compilação e execução

Para a execução do programa basta a utilização do *Makefile* presente no arquivo e a execução do comando *make* no diretório, para limpar o diretório basta a execução do comando *make clean*.
O algoritmo foi implementado e testado em um ambiente linux Ubuntu

> Distributor ID: Ubuntu
> Description: Ubuntu 18.04.2 LTS
> Release: 18.04
> Codename: bionic

Utilizando o compilador *g++ -std=c++11*.

5. Conclusões

Esse trabalho possibilitou a aplicação de algoritmos de ordenação para solução de problemas. O uso do Merge Sort e Radix Sort foi dado pela necessidade de algoritmos estáveis e pelas restrições de complexidade do problema.

Além disso, o a análise experimental foi uma forma útil de visualizar o tempo de execução do problema e a visualização da complexidade assintótica de acordo com o tamanho da entrada. Entretanto, sendo a primeira análise experimental realizada para alguns projetos os dados utilizados podem não ter sido os ideais, porém geraram resultados como o esperado pela análise de complexidade teórica.

References

https://www.geeksforgeeks.org/
figure [1] http://www.differencebetween.net/technology/difference-between-quick-sort-and-merge-sort/