Trabalho Prático 2 - Estruturas de Dados

Alan Borges Martins Bispo

[1]Departamento de Ciência da Computação - Universidade Federal de Minas Gerais (UFMG)

Abstract. *In this work our goal is to solve another Rick Sanchez optimization problem, which consists of: given a family of planets and its respective time weights and names, we've to fit as most planets as possible in our monthly trip schedule, under a time limit upper bound. Therefore will be presented a solution that mostly consists of a implementation of different sorting algorithms, the merge and radix sort.*

Resumo. *Nesse trabalho o nosso objetivo consiste em resolver outro problema de otimização de Rick Sanchez, que é definido como: dada uma lista de planetas e seus respectivos pesos de tempo e nomes, temos que visitar o maior número de planetas possível por mês sob um limite de tempo determinado pela entrada. Assim, a solução apresentada abaixo, consiste basicamente na implementação de diferentes algoritmos de ordenação, o Merge Sort e o Radix Sort.*

1. Introdução

O problema consiste na otimização da agenda de viagens de Rick Sanchez, onde dada uma entrada definida por um limite de tempo mensal para visitas, número de planetas a serem visitados e o tamanho do nome dos planetas. Assim, iremos utilizar o Merge Sort para ordenar a lista de planetas em função do tempo necessário para visitar cada um deles, após isso iremos dividir a lista em subvetores de planetas que armazenam o maior número de planetas possível, representando as viagens de cada mês e por fim utilizando o radix resta ordenar os subvetores de planetas em função da ordem lexicográfica de caracteres do nome dos planetas, a partir do dígito de menor significância.

2. Implementação

O software se baseia no seguinte fluxo de dados: a entrada é recuperada via *standard input*, então um array de tamanho dinâmico (tamanho P) da classe *Planet* é criado, e seus elementos ajustados a partir dos valores da entrada; assim, esse *array* é primeiramente ordenado numa função de algoritmo *Merge Sort* com base no valor de cada tempo gasto em cada planeta. A seguir, com o vetor ordenado em função do tempo (de forma crescente), o programa separa o vetor em vários "baldes", que nada mais são do que nós numa lista ordenada que é a "agenda"; dessa forma, a class *Agenda* é aquela que guarda os nós *MonthBucket* que representam os meses que Morty deverá gastar para completar o planejamento da entrada. Durante a separação dos planetas nos meses ocorre também uma ordenação interna do balde naquele mês, pelo nome, através do Radix Sort. Assim, basta que imprima-se a lista ordenada dos planetas nos meses da agenda.

2.1. Merge Sort

Funcionamento: o *merge sort* funciona por divisão e conquista: isso significa que o algoritmo vai fazendo a divisão consecutiva do algoritmo em pedaços menores de forma a

reduzir seu ”esforço”; isto é, com pedeços menores é mais fácil de se resolver a ordenação. Dessa forma a divisão continua até o processo de *merge* onde acontece a montagem de subvetores através de troca e ordenação. O processo se repete recursivamente pelo vetor inteiro.

2.2. Radix Sort

Funcionamento: esse algoritmo funciona com base na ordenação das chaves via método de contagem, usando o valor do elemento para incrementar mais um na posição de um array auxiliar, e depois percorrendo esse array de forma crescente para obter a ordenação final. A ordenação é feita letra a letra para o nome, da menos significativa até a mais significativa.

3. Análise de Complexidade

Como a solução do nosso problema consiste basicamente da ordenação de vetores por meio do Merge Sort e Radix Sort, seja p o número de planetas e k o tamanho alfabeto o qual o nome dos planetas pertence.

A complexidade do Merge Sort é da ordem de $O(p \log_2 p)$ enquanto a do Radix Sort possui complexidade assintótica da ordem de $O(p \times k)$. Portanto, sendo $k = 26$

$$\begin{aligned} f(n) &= O(p \log_2 p) + O(26 \times p) \\ &= \max(O(p \log_2 p), O(26 \times p)) \\ &= O(p \log_2 p) \end{aligned}$$

4. Instruções de execução

O makefile preparado para a execução do código tem as seguintes opções:

1. make: Compila os arquivos de código fonte e armazena a saída na pasta atual, roda os testes.
2. make mem: Executa o arquivo gerado pela compilação no valgrind para verificação de vazamentos de memória.
3. make test: Executa os script de testes do trabalho.
4. make clean: Limpa a pasta contendo as saídas da compilação.

5. Conclusão

Concluí através da execução proposta do trabalho prático que para a elaboração de um algoritmo que fizesse a ordenação como exigido para a complexidade do trabalho O(nlogn) que além disso fosse estável, foi necessário usar o Merge Sort. Para o caso do algoritmo de complexidade O(k + n) foi necessário usar o Radix Sort, que tem complexidade de O(n).

Referências

1. https://www.geeksforgeeks.org/
2. Materiais didáticos providos em aula.