Universidade Federal de Minas Gerais

Departamento de Ciência da Computação

Algoritmos 1

**Trabalho Prático 1**

**Jogo do Pulo**

Aluno: Lucas Augusto Araújo Aguiar

Professora: Olga Nikolaevna Goussevskaia

Belo Horizonte, 04 de Outubro de 2019

**1 - Introdução**

Neste trabalho, novamente recebi a tarefa de auxiliar o cientista Rick Sanchez em um novo problema. Tal problema consiste no seguinte: por ser muito famoso, Rick é constantemente convidado para visitar diversos planetas e adora fazê-lo. Contudo, ele é muito ocupado e possui um tempo limitado para visitar cada planeta. Tendo isso em vista, Rick me pediu para criar uma agenda para ele com quais planetas ele deve visitar por mês por ordem alfabética em função da quantidade de tempo que ele tem reservado mensalmente para essa tarefa. Como exemplo, vamos supor que o Rick me avisa que tem 700 horas por mês e que possui uma lista de 5 planetas para visitar, sendo o planeta A por 200 horas, B por 300 horas, C por 500 horas, D por 100 horas e E por 150 horas. Sendo assim, eu preciso retornar um agenda bem organizada, onde cada mês possui o máximo de proveito de tempo e em ordem crescente do tempo requisitado do planeta. Logo, eu retorno { 1 A 200, 1 D 100, 1 E 150, 2 B 300, 3 C 500 }, onde cada numerador que antecede o nome do planeta é o mês de visitação.

**2 - Implementação**

Classes, structs e métodos utilizados:

* Struct Planet: estrutura responsável por representar o planeta da minha lista sequencial. Tal struct possui como atributos: um inteiro month, outro inteiro timeStay e uma string namePlanet. O month guarda o mês que Rick irá visitar aquele planeta, o timeStay armazena o tempo que ele ficará naquele planeta e o namePlanet guarda o nome do planeta.
* Class PlanetList: classe responsável por armazenar os planetas. Tal classe possui como atributos: um vetor para armazenar todos os planetas(planets), o número total de planetas(numberOfPlanets), o tempo que Rick disponibiliza por mês para visitar os planetas(timeForMonth) e o tamanho do nome de cada planeta(sizeName).

Nessa classe, temos os seguintes métodos:

* PlanetList(): representa o construtor, responsável por inicializar a lista. Tal inicialização é feita definindo o número de planetas como 0;
* insertPlanet(timeStay, namePlanet): função responsável por inserir um planeta. Tal operação é feita criando um Planeta, atribuindo à ele o nome e o tempo que Rick precisa ficar nele e colocando-o no vetor.
* printList(): função responsável pelo output do meu programa, expondo o mês, o nome do planeta e o tempo que Rick precisa ficar no planeta. Tal operação é feita através de um loop, onde dou um ‘print’ de cada planeta.
* merge(left, middle, right): função responsável por juntar vetores ordenados no vetor principal. Tal tarefa é realizada utilizando vetores, auxiliares para os Planetas da esquerda até o do meio e o do meio até o da direita. Após isso, os vetores são comparados e alocados no vetor principal em ordem crescente, de modo a organizar todos os planetas dos dois vetores.
* mergeSort(left, right): função responsável por fazer as funções recursivas para organizar o vetor como um todo, fazendo chamadas da esquerda até o meio e do meio até a direita e, após isso, chama o merge para juntar os mergeSort chamados. Assim, os planetas são divididos em vetores cada vez menores, até que chegam no menor possível onde não podem mais ser divididos, onde convertem para o principal.
* organizeListForMonth(): função responsável por organizar a lista de acordo com os meses, chamada logo após o merge ter organizado por tempo necessário para cada mês. Tal operação é realizada com um loop que percorre todo o vetor, onde, passando por cada planeta, é verificado se o tempo desse planeta cabe na agenda para o mês. Caso não caiba, passamos para o próximo mês e o alocamos nesse novo mês.
* quicksortB(start, end): função responsável por chamar a countingSort para um determinado intervalo do vetor de planetas, chamando o countingSort para cada caractere do nome do planeta naquele intervalo.
* countingSort(start, end, planets, digit): função responsável por ordenar o nome do planeta de acordo com o dígito. Tal função é realizada da seguinte forma: criamos um vetor de planetas do tamanho do intervalo determinado pelo quicksortB, após isso, criamos um vetor c de inteiro e o zeramos. Após isso, os espaços nesse vetor referentes ao inteiro de cada dígito do nome do planeta é posto como 1. Depois disso, realizamos uma soma de modo que cada espaço de c seja somado com todos os anteriores à ele. Logo em seguida, utilizamos um vetor auxiliar para realizar a ordenação e passamos para o vetor principal(planets).
* getNumberOfPlanets(): função que retorna o número de planetas alocados no vetor.
* getTimeForMonth(): função que retorna o tempo mensal do Rick para visitar planetas.
* getMaxMonth(): retorna quantos meses são necessários para visitar todos os planetas.
* getLimits(limits[], month): função que aloca os limites de cada mês, colocando o índice do primeiro planeta para um mês específico e o último do mesmo mês. Função utilizada para organizar em ordem alfabética os planetas do mesmo mês.
* setSizeOfName(sizeOfName): função que determina o tamanho do nome de cada planeta.
* setTimeForMonth(timeForMonth): função responsável por setar o tempo mensal de Rick para visitação.
* Main: Na main, criei apenas o inputTreatment:
* inputTreatment(myPlanets): função responsável por cuidar do input do programa e orquestrar o output. Como o input possui um padrão de ‘horas mensais de Rick’ / ‘número de planetas’ / ‘tamanho do nome dos planetas’ e após isso é uma repetição determinada de ‘tempo para tal planeta’ / ‘nome do planeta’, o input é simples de ser feito para alocar os números nas variáveis. Após isso, tal função chama as funções citadas anteriormente para gerarmos o output requisitado.

**3 - Instruções de compilação e execução**

Para compilar o programa, basta acessar a pasta pelo terminal e digitar ‘make’. Após isso, para executar o programa, basta acessar a pasta build, ainda pelo terminal, e digitar ‘./tp2’. Ademais, para verificar os testes basta digitar ‘make test’ e para limpar os arquivos temporários criados basta digitar ‘make clean’.

**4 - Análise de complexidade**

Análise assintótica:

* insertPlanet apresenta complexidade O(1) para qualquer caso, uma vez que basicamente apenas insere o planeta no vetor.
* printList apresenta complexidade O(n), uma vez que precisa percorrer todo o vetor para mostrar cada planeta inserido nele.
* mergeSort apresenta complexidade O(n log(n)), sendo escolhido pois cumpria com os requisitos presentes na especificação do trabalho. A complexidade é tal pois, para n>1, sua fórmula é . Dessa forma, ao desenvolvermos essa fórmula teremos . Como para n = 1 a complexidade é O(1), . Substituindo, chegamos em o que é equivalente à O(n log(n)) em todos os casos.
* organizeListForMonth apresenta complexidade O(n), uma vez que percorre o vetor apenas uma vez, setando seus meses.
* quicksortB(Radix) apresenta complexidade O(nk), sendo escolhido pois cumpria com os requisitos presentes na especificação do trabalho, assim como o merge. Sua complexidade é tal pois percorremos o loop para passarmos por cada caractere do nome do planeta, ou seja, apresenta complexidade O(k), onde k é o tamanho do nome e, após isso, chamamos a função coutingSort que possui complexidade O(n). Dessa forma, a complexidade geral do radix é O(n k) em todos os casos.
* getLimits possui complexidade O(n) pois percorre todo o vetor procurando onde começa e onde termina cada mês.

Análise de custo de memória:

Analisaremos o custo de memória por lista e por cada função de ordenação.

* myPlanets: 150.000 ***x*** sizeof(Planet).
* myPlanets.quicksortB: k(j ***x*** sizeof(Planet) + 257 ***x*** sizeof(int)), onde k é o tamanho da palavra e j o número de planetas de mesmo mês em cada chamada.
* myPlanets.mergeSort: n ***x*** sizeof(Planet).

Tempo de execução para diferentes entradas:



x = [3, 10, 100, 5000, 10000, 50000]

y = [0.007413, 0.006908, 0.008714, 0.030016, 0.308036, 4.531570]

**5 - Conclusão**

O trabalho foi muito produtivo em questão de aprendizado, uma vez que consegui de fato aprender a implementar os métodos de ordenação. Contudo, devo admitir que o Radix foi extremamente complicado e recorri ao stackoverflow diversas vezes para entender sua aplicação e como funcionava. A implementação do merge foi mais simples, já que é um algoritmo simples. Ademais, em meu programa apareceu um problema que eu não soube resolver: os testes 12, 13 e 14 não passam dando segment fault e eu não consegui descobrir o porquê, uma vez que, caso eu troque o tamanho do vetor principal de 150.000 para 100.000, os casos 12 e 13, assim como todos os anteriores, passam. Contudo, o 14 não passa uma vez que ele não consegue alocar os 150.000 planetas.

**6 - Bibliografia**

Clean Code: A Handbook of Agile Software Craftsmanship, por Robert C. Martin, Michael C. Feathers e Timothy R. Ottinger.

StackOverFlow: <https://stackoverflow.com/questions/32543683/radix-sort-array-of-strings>, <https://stackoverflow.com/questions/23038622/radix-sort-on-an-array-of-strings>