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

Aluno: Guilherme Pereira
Matrícula: 2016023354

1. Introdução

O código foi feito com o intuito de resolver o problema da agenda de viagens de Rick Sanchez. É preciso saber em quais meses e em que ordem ele deverá visitar um certo grupo de planetas.

Um exemplo disso é o seguinte:
Ele precisa visitar 5 planetas e só pode ficar gastar 720 horas mensais em viagens interplanetárias. Dada uma lista de planetas, ele deve visitar primeiro os planetas com menor tipo de visitação. E em cada mês, a ordem de visitação será por ordem alfabética, considerando o nome do astro.

O objetivo deste trabalho é implementar diferentes algoritmos de ordenação, sendo um deles O(nlogn) e outro O(n x k). Para isso, foram utilizados os algoritmos Merge Sort, Counting Sort e Radix Sort.

2. Implementação

Estruturas de Dados:

Para a realização do trabalho foi criada uma classe planetas. Essa classe possui apenas dois parâmetros, name e visitTime. Ou seja, esses dois parâmetros armazenam o nome do planeta e o tempo de visitação, respectivamente.

Essa classe possui apenas dois métodos, getName e getTime. Elas basicamente retornam o nome do planeta e o tempo de visita, nessa ordem.

Funções e Procedimentos:

Planet():
Construtor do planeta, que no caso cria um planeta vazio, sem nome nem tempo de visitação.

Planet(std::string name, int visitTime):

Construtor do planeta, só que dessa vez o nome e o tempo de visitação com parâmetro. Esses valores acabam sendo os valores do planeta criado.

int getTime():
Método que retorna o valor do tempo de visitação do planeta

std::string getName():
Método que retorna o valor do nome do planeta.

void merge(Planet planets[], int left, int middle, int right):
Função que recebe como parâmetro um array de planetas, a posição da esquerda, a posição do meio e a posição da direita. Essa função junta dois arrays já ordenados em um único array também ordenado. É a base do merge sort e guarda esse novo array ordenado no próprio array passado como parâmetro.

void mergeSort(Planet planets[], int left, int right):
Função recursiva que recebe como parâmetro um array de planetas, a posição da esquerda e a posição da direita, ou seja, as posições do começo e final do array. Essa função divide um array em dois arrays, esquerda e direita, continuando nesse processo até os arrays possuírem apenas um elemento. Depois da divisão, ela usa a função merge para juntar todos os elementos. No final o array planets estará ordenado baseado no tempo de visitação.

void countingSort(Planet planets[], int sizeArr, int posChar):
Função iterativa que recebe como parâmetro um array de planetas, o tamanho do array e a posição de caractere considerada para fazer a ordenação. Essa função cria um array count para determinar quantas vezes cada letra do alfabeto ocorreu, considerando todos os algarismos da posição considerada de todos os planetas no array. Depois disso, ela adiciona a cada elemento do array count o elemento anterior. Assim, após isso a função ordena um vetor temporário, igual ao passado como parâmetro, considerando os novos valores do array count. Por fim, a função coloca no array parâmetro planets os valores armazenados no array temporário. Com tudo isso, essa função ordena um array de planetas considerando um caractere dos nomes dos planetas em ordem alfabética.

void radixSort(Planet planets[], int sizeArr, int numChar):
Função iterativa que recebe como parâmetro um array de planetas, o tamanho do array e o número de caracteres dos nomes dos planetas. Ela basicamente chama a função countingSort para cada caractere dos nomes dos planetas, sendo o último caractere o primeiro e o primeiro caractere o último a ser ordenado. Com tudo isso, essa função ordena um array de planetas em ordem alfabética, considerando o nome dos astros.

void printVisit(Planet planets[], int start, int end, int month, int numChar):
Função iterativa que recebe como parâmetro um array de planetas, um índice de começo, um índice de fim, o número do mês e o número de caracteres dos nomes dos planetas. Essa função cria um array temporário cujo tamanho depende dos índices do começo e fim. Logo após isso, esse array temporário recebe os planetas do array planets que então entre

o índice start e o índice end. Após isso, esse array temporário é ordenado em ordem alfabética por meio do radixSort. Finalmente, são impressos o mês, os planetas e o tempo de visitação em ordem alfabética. Essa função, então, imprime a agenda de visitas de um mês.

void results(Planet planets[], int numPlanets, int maxTime, int numChar):
Função iterativa que recebe como parâmetro um array de planetas, o número de planetas, o tempo máximo de visita para cada mês e o número de caracteres dos nomes dos planetas. O array de planetas que essa função recebe já está ordenado em ordem crescente considerando os tempos de visita. Essa função basicamente checa até que planeta pode ser feita as visitas, ou seja, até que planeta a soma de tempo de visitas não irá exceder o limite mensal de tempo de visitas. Após calcular o valor para cada mês é chamada a função printVisit, que irá imprimir o determinado mês calculado em ordem alfabética. Por fim, a função printVisit será chamada para os elementos do último mês que sobraram.

3. Instruções de compilação e execução

Para a compilação e execução foi usado um make file. O make file está localizado na pasta src, que está dentro da pasta guilherme_pereira. Para compila-lo é necessário um bash linux. Quando no bash, vá para o diretório src, em que o make file está, como já foi dito. Depois, basta executar o seguinte comando “make ./tp2” sem parênteses para realizar a compilação. Para executar o programa, após compilá-lo, basta executar o comando “./tp2” também sem parênteses.

Após esses dois comandos a execução do programa irá começar. No primeiro momento, é preciso inserir três inteiros, o tempo máximo de visitação em um mês, o número total de planetas e o número de caracteres nos nomes. Após isso, você deverá digitar n vezes, em que n é o número total de planetas, o tempo de visitação e o nome do planeta. Depois disso será impressa a ordem de visitação dos planetas. O programa finaliza automaticamente.

4. Análise de complexidade

Função void merge(Planet planets[], int left, int middle, int right):
Complexidade O(n) para tempo e espaço, para todos os casos.

Função void mergeSort(Planet planets[], int left, int right):
Complexidade O(nlogn) para tempo e O(n) para espaço, para todos os casos.

Função void countingSort(Planet planets[], int sizeArr, int posChar):
Complexidade O(n) para tempo e espaço, para todos os casos.

Função void radixSort(Planet planets[], int sizeArr, int numChar):
Complexidade O(n x k) para tempo e espaço, para todos os casos. Em que k representa o número de caracteres dos nomes dos planetas.

Função void printVisit(Planet planets[], int start, int end, int month, int numChar):
Complexidade O(n) para tempo e espaço, para todos os casos.

Função void results(Planet planets[], int numPlanets, int maxTime, int numChar):
Complexidade O(1) para espaço. Complexidade O(n) para tempo. Ambas as complexidades para todos os casos.

Programa principal:
Como o programa principal usa basicamente a função mergeSort e em seguida a função results, o programa como um todo é O(n) para espaço e O(nlogn) para tempo.

5. Conclusão

A implementação do trabalho mostra a importância da criação de algoritmos de ordenação eficientes. Foi possível perceber como cada algoritmo de ordenar possui vantagens e desvantagens e cada um deles é ideal para certas situações.

A complexidade não é a única coisa que precisa ser considerada ao usar um algoritmo desses, mas sim também a estabilidade. Para ordenar os planetas em ordem alfabética, por exemplo, é preciso garantir a estabilidade.

Com tudo isso, a principal dificuldade foi entender como esses diferentes algoritmos funcionam e como implementá-los. Suas implementações são diversas, alguns iterativos, outros recursivos. Essa diversidade gera certa dificuldade.

6. Bibliografia

https://virtual.ufmg.br/20192/pluginfile.php/429140/mod_resource/content/3/Counting%20e%20Radix%20sort.pdf

https://virtual.ufmg.br/20192/pluginfile.php/433337/mod_resource/content/0/Aula%2010%20-%20MergeSort.pdf