Trabalho Prático 1 - Estrutura de dados

Victória Olívia Aráujo Vilas Boas - 2019431429

Belo Horizonte, 12 de Novembro de 2019

1 Introdução

O objetivo do trabalho proposto é ajudar o cientista Rick Sanchez a organizar sua agenda de viagens interplanetárias.

A agenda automatizada do Rick deve ser capaz de distribuir seus compromissos ao longo dos meses de modo que as viagens com menor duração ocorram primeiro e caso o tempo disponível em um mês não seja suficiente para todas as viagens, elas sejam agendadas nos meses seguintes seguindo o mesmo critério. Além disso, Rick deseja que as viagens dentro de um mesmo mês sejam organizadas de acordo com o nome dos planetas que irá visitar, em ordem alfabética.

A solução para esse novo desafio do Rick, contará com uma estruturas de dados implementadas em código em linguagem c++ que armazena o calendario de rick, e três metodos de ordenação para classificar a ordem de visita das cidades.

Esse código foi implementado em sistema MACOSX e testado em uma máquina linux Ubuntu.

2 Implementação

2.1 Estílo de Código

O estilo de código adotado na implementação do problema segue o estilo de código do projeto Unix, em sua maioria.

2.2 Arquitetura

A arquitetura adotada no problema conta com a seguinte estruturação de pastas:

main.cpp
Makefile
header

Dentro da pasta header são armazenados os seguintes arquivos: Um arquivo que contém uma classe descrevendo o compromisso de Rick, uma arquivo que contém funções auxiliares para a resolução do problema, e um arquivo que contém os algorítimos de ordenação utilizados na solução.

methods.cpp
helpers.hpp
Appointment.hpp

2.3 Classes

Para a implementação do problema, foi contruida a classe Appointment. Essa classe foi utilizada para armazenar a descrição do compromisso de rick: qual o mês em que a visita irá acontecer, o nome do país, e a duração da visita. Essa classe conta com dois construtores, uma com parâmetros, que é usado para marcar os compromissos, e um outro sem parâmetros, usados para popular uma estrutura de dados que será descrita mais a frente.

2.4 Funções e detalhes de implementação

Para a implementação da solução foram utilizados como base três métodos de ordenação: o radix sort, o counting sort e o merge sort.

Inicialmente o usuário entra com o tempo total de visita em um mês, o número de planetas e o número de letras que o nome do planeta terá. Em seguida o usuário entrará com o nome de cada planeta, e esses nomes serão armazenados em um array do tipo Appointment, descrito anteriormente.

Na sequência da função ***main*** será chamado o método de ordenação merge sort, que ordenará o vetor de de compromissos pelo tempo de visita. No fim, teremos os compromissos mais rápidos no começo do vetor, e assim, poderemos decidir os dias de cada comprimisso.

Ademais, a função ***months_counter*** será chamada, e ela irá decidir o mês de cada compromisso pelo tempo total que poderá ser dedicado em cada mês. Se a soma da duração dos compromissos ultrapassar o tempo total, as proximas visitas serão agendadas para o próximo mês, e assim sucessivamente. Além disso a cada vez que um novo mês passa a receber agendamentos, a posição do primeiro compromisso desse mês será armazenada no array months_init.

Em seguida, o ultimo mês de visita será armazenado, para que um loop seja criado e a função radix_sort será chamada para cada mês de visita. Além disso, dentro desse loop um outro loop é criado para que a ordenação seja executada para cada coluna do nome dos planetas, começando da última. Esse decisão de implementação foi tomada pois não há necessidade de saber qual o digito menos significativo do programa, dado que, por sabermos o número de caracteres da palavra, saberemos que o menos significativo será o ultimo caractere, de modo definido.

Dentro da função radix_sort o serão declaradas duas variáveis, que irão armazenar o índice de inicio e fim de cada mês de visita dentro do array de compromissos. Esses valores serão obtidos a partir do array months_init, acessando as posições referentes ao mês que se deseja ordenar e ao mês seguinte. Na sequência a função get_max será chamada para iterar pelos compromissos do mês e descobrir o maior código ASCII dentre os caracteres dos nomes dos planetas que está serão ordenados. Finalmente, a função counting_sort será chamada para ordenar cada coluna do nome dos planetas em cada mês.

3 Intruções de compilação e execução

Para compilar o código adequadamente é necessario seguir os seguintes passos:

1. Entrar na pasta ***src*** com o comando:

```
cd src
make tp2
```

2. Rodar o programa com o comando:

```
./tp2
```

4 Análise de complexidade

Não haverá melhor e pior caso, pois todos os métodos de ordenação agem de forma independente da entrada.

1. A complexidade de tempo em qualquer caso é de:

$$\mathcal{O}(k * n) \tag{1}$$

Sendo k o número de letras da palavra e n o número do maior código ASCII.

2. A complexidade de espaço em qualquer caso considera o tamanho da maior letra ascii e o numero m de entradas de planetas:

$$\mathcal{O}(k * m) \tag{2}$$

5 Conclusão

Assim, nota-se que o custo do