O problema da agenda de viagens de Rick Sanchez

Arthur de Lapertosa Lisboa[1]

[1]Estudante - Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brazil

arthurlapertosa@ufmg.br

Abstract. *Rick Sanchez's travel schedule problem boils down to a scientist who wants to automate his interplanetary travel schedule. Given a specification of planets and time spent on each planet, he wants the program to set up a visit schedule of the planets each month. For this, it was also considered a maximum monthly visitation time, defined by the user.*

Resumo. *O problema da agenda de viagens de Rick Sanchez se resume em um cientista que deseja automatizar sua agenda de viagens interplanetárias. Ele deseja que dado uma especificações de planetas e o tempo gasto em cada planeta, seja montada uma agenda de visitação dos planetas em cada mês. Para isso, foi considerada, também, um tempo máximo de visitação mensal, definido pelo usuário.*

1. Introdução

Dado o problema especificado acima, foram definidas as melhores estruturas de dados para o programa: vetores dinamicamente alocados para armazenamento dos nomes dos planetas e o tempo gasto naquele planeta e lista duplamente encadeadas, que armazenam vetores de planetas por mês de visitação.

Para os vetores alocados dinamicamente, a implementação não foi complicada. Apenas foi criada uma classe “vetor”, com todas os métodos necessários para adicionar planetas e seu tempo de visitação em cada elemento do vetor. Dentro desta classe, cada elemento do vetor é um tipo “struct elements”, que contém dois atributos: uma string que contém o nome do planeta e um int que contém o tempo de visitação para aquele planeta.

Para a lista, também foi criada uma classe “lista”. Ela foi implementada com os fundamentos de uma lista duplamente encadeada com uso de sentinela.

Para gerir estas duas classes, também foi criada a classe “Planets”, que é uma superclasse usada para gerenciar e interligar as duas classes auxiliares. Ela contém todos os método para criar o primeiro vetor de planetas de acordo com a entrada do usuário, também responsável por chamar os métodos de ordenação e montar a lista de visitação de cada planeta.

2. Implementação

2.1. Especificações técnicas

O programa foi escrito e testado numa máquina Windows 10 Home, 16GB de RAM, Processador Intel Core i7 8750H. Foi usada a IDE Microsoft Visual Studio 2019, e o compilador utilizado é o nativo do Visual Studio 2019, C++17. O programa também foi testado no terminal Ubuntu do Windows, usando o compilador G++.

2.2. Funcionamento das Classe "Lista"

A classe "Lista" foi implementada usando os conceitos de uma lista duplamente encadeada com uso de sentinela. Ela foi criada pensando no armazenamento de objetos do tipo vetor, que são efetivamente vetores. Cada elemento da lista corresponde a um mês de visitação do Rick Sanchez.

Os atributos desta classe são os seguintes: "int cardinalidade_", que mantém o tamanho da lista, "NoL* sentinela", que é uma variável do tipo NoL, que representa um ponteiro para a sentinela da lista duplamente encadeada. O NoL é um struct, definido na implementação, que possui os atributos: "vetor* planetas" que armazena o vetor de planetas daquele mês de visitação, "NoL* dir" e "NoL* esq", que são os ponteiros para o elemento que está a direita e para o elemento que está a esquerda, respectivamente.

Os métodos desta classe são os seguintes:

"lista()" que é o construtor da lista. Ele é responsável por inicializar a lista encadeada, alocando espaço para o sentinela e setando os valores do "dir" e "esq" do sentinela. Também é setado o valor 0 para o atributo "cardinalidade_".

"void addElement(vetor& planetsArray, int mesEnum)" que é o método responsável por adicionar um elemento com o vetor recebido pelo parâmetro por referência "planetsArray" à lista encadeada. Para cada elemento adicionado, também é adicionado um mês correspondente àquele vetor de planetas.

"void printL();" é um método auxiliar que printa a lista na saída padrão do sistema (stdout).

"~lista();" é o método destrutor da classe lista. Ele percorre por todos os elementos da lista, deletando um por um, para que seja liberado espaço na memória.

2.2. Funcionamento das Classe "vetor"

A classe vetor foi implementada como um vetor alocado dinamicamente. O vetor é inicialmente alocado com o tamanho passado para o seu construtor. Se eventualmente for feita a tentativa de inserir mais elementos do que foi definido inicialmente, a classe cria um novo vetor, com o dobro da capacidade, e copia todos os elementos para o novo vetor.

Os atributos desta classe são os seguintes: "int size_", que mantém o tamanho atual do vetor, "int capacity_", que contém a capacidade máxima do vetor, "int stringSize_", que contém o tamanho da string que foi usado nos nomes dos planetas e o "elements*

elements_", que é um ponteiro para o primeiro elemento do vetor de "elements". "elements" é um struct definido como cada elemento do vetor. Ele contém uma std::string da STL do C++ (que foi usado apenas para armazenar o nome de uma maneira mais intuitiva, não foi usado nenhum método de da biblioteca), e um inteiro que corresponde ao tempo de visitação daquele planeta. Também foi criado dois construtores inline auxiliares para inicializar o struct de uma maneira mais fácil.

Os métodos mais importantes desta classe são os seguintes:

"vetor(int capacity, int stringSize)" que é o construtor da classe. Ele é responsável por inicializar o vetor. É criado um vetor de "elements" com a capacidade definida por "capacity". Também é inicializado todos os outros atributos da classe.

"elements& operator[](int index) const;" que é o uma sobrecarga do operador [], responsável por retornar o elemento com aquele índice.

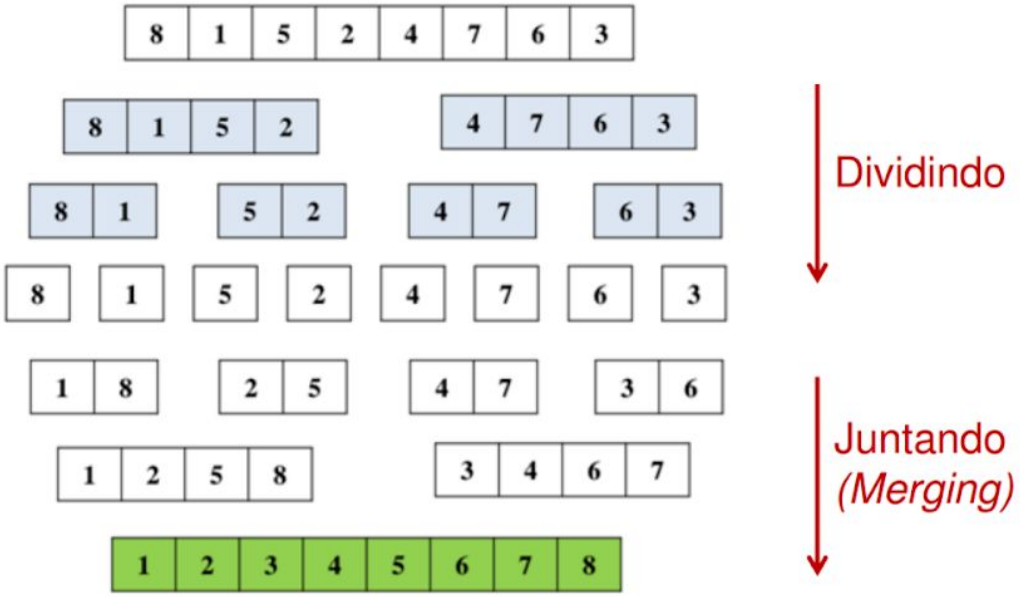
"void pushbackEl(std::string name, int integer);" é o método responsável por adicionar um novo elemento ao vetor. Ele chama também o método "void pushback(elements& element)", que efetivamente coloca o elemento dentro do vetor. Ele também testa toda vez se o vetor já atingiu sua capacidade máxima, e se isso acontecer, ele cria um novo vetor com o dobro da capacidade e copia elemento a elemento para o novo vetor.

"void sort();" é o método responsável por retornar organizar o vetor em ordem crescente de "stayTime". Ele é implementado usando o MergeSort, que é um método de ordenação de vetores de complexidade O(n*log(n)). Ele basicamente cria o vetor auxiliar usado pelo Merge Sort e chama o método "void mergeSort(elements aux[], int esq, int dir)", passando o vetor auxiliar criado como parâmetro, 0 como o "esq" e o "size_ - 1" como "dir".

"void mergeSort(elements aux[], int esq, int dir)" é um método privado da classe, responsável por realizar efetivamente a ordenação do vetor "elements_". Este método é o MergeSort, que segue o paradigma de programação chamado divisão e conquista. Ele então divide o vetor em subparte menores e chama recursivamente a si mesmo, dividindo o vetor inicial em dois. Depois de ter divido o vetor em duas subpartes com metade do seu tamanho, ele chama o método "Merge", que junta novamente as duas subpartes, ordenadamente.

"void merge(elements aux[], int esq, int meio, int dir);" é o método responsável por juntar duas subpartes de vetores em um novo vetor, de forma já ordenada.

O funcionamento do MergeSort está exemplificado melhor na gravura:



O algoritmo é fundamentado em dividir o vetor e várias subpartes, até que elas sejam unitárias, e depois ir juntando tudo, de forma ordenada até chegar ao vetor ordenado. Importante destacar que este método é estável, algo imprescindível para este projeto.

“void sortName();” é o método responsável por ordenar, agora, por ordem crescente do nome do planeta. Ele é responsável por chamar o método “void radixSort();”, que faz esta ordenação.

“void radixSort();” é um privado método que chama, para cada caracter da string “planet”, o método “void countingSortChar(int index)”. Ele inicia pelo último caracter da string e vai chamando o método até o primeiro caracter. Desta forma, o vetor será organizado de forma alfabética.

“void countingSortChar(int index);” é um método privado que aplica a ordenação CountingSort no vetor, de acordo com o caracter do índice “index” da string “planet”, que contém o nome do planeta. Ele basicamente usa dois vetores auxiliares: um vetor para armazenar a quantidade de elementos com aquele índice que contém no vetor “elements_” e outro vetor para armazenar os elementos de “elements_”, já ordenados. Teoricamente precisaríamos saber a priori qual o elemento de valor máximo do vetor, e para isso, precisaríamos percorrer todo o vetor para isso. Porém, como a ordenação é por elementos do tipo “char”, e elementos deste tipo contém apenas 1 byte, pode ser criado um vetor auxiliar de contagem com tamanho máximo de 256, que é equivalente à um byte. Desta forma, será economizado uma passagem completa pelo vetor, sem que seja usado muito espaço de armazenamento do vetor auxiliar de contagem. O método funciona basicamente desta forma: é passado uma vez no vetor de “elements_” para que seja contado quantas vezes aquele elemento se repetiu no vetor. Feito isso, fazemos a soma acumulada dos elementos de vetorCount (vetor auxilixar de contagem), de maneira que vetorCount[i] = vetorCount[i-1] + vetorCount[i], para i > 0. O último passo

é percorrer "elements_" do ultimo elemento para o primeiro, de maneira que: novo [vetorCount [elements_[i]] - 1] = elements_[i]. É ainda necessário decrementar em 1 unidade o elemento vetorCount[elements_[i]]. Desta forma temos o vetor ordenado de forma crescente, de acordo com o caracter na posição “index” do atributo “string planet”, do vetor “elements_”.

“~vetor()” é o método destrutor da classe vetor. Ele deleta o vetor de “elements_” para que seja liberado espaço na memória.

2.2. Funcionamento das Classe “planets”

A classe planets é uma classe que interliga as classes “lista” e “vetor” para fazer os cálculos e ordenações necessárias para que seja criado a lista de visitação de planetas de forma adequada.

Os atributos desta classe são os seguintes: “vetor* planets_;”, que é o vetor de planetas, “int timePerMonth_;”, que contém o tempo máximo gasto por mês na visitação de planetas, o “int numberCharacteres_;”, que é o número de caracteres que o nome dos planetas possuem e o “lista planetsVisitation_;”, que é uma lista com os subvetores correspondentes aos meses e os respectivos planetas para serem visitados.

Os principais métodos desta classe são os seguintes:

“planets()” é o construtor padrão da classe planets. Ele é responsável chamar a função “void buildPlanets();”, que lê pela entrada padrão do sistema (STDIN) tudo o que é necessário para criação do vetor de planetas, que é: tamanho do vetor (número de planetas), o número de caracteres que o nome dos planetas possuem e o tempo por mês destinado à visitação de planetas.

“laboratorio(int volume);” é o construtor da classe laboratório responsável por criar um laboratório com um frasco com o volume passado como o parâmetro “volume”.

“void sort();” é o método responsável por organizar em ordem crescente de "StayTime" o vetor de planetas, através da chamada ao método sort do atributo objeto “planets_”.

“void fillPlanets();” é o método responsável por ler, a partir da entrada padrão do sistema (STDIN), todos os planetas, seus nomes e tempos de visitação e armazená-los no vetor.

“void buildVisitationList();” é o método que cria a lista de visitação mensal. Ele funciona da seguinte maneira: itera sobre os elementos já ordenados do vetor de planetas (é necessário que o método sort seja chamado anteriormente) e vai adicionando elementos ao vetor auxiliar “aux” até que o tempo máximo daquele mês seja esgotado. Quando o tempo se esgota, ou seja, já existem o número máximo de planetas naquele vetor para aquele mês, é chamado o método “sortName();”, que organiza os elementos naquele vetor em ordem alfabética. Logo em seguida, este vetor auxiliar é adicionado à lista de visitação, com os seu respectivo mês correspondente à visitação daqueles planetas.

2.3. Funcionamento da main

Na main foi criada uma estrutura básica, que cria um objeto do tipo planets e chama os métodos necessários para funcionamento do programa.

3. Instruções de compilação e execução

O makefile para compilação do programa já está pronto. Para compilar o programa e gerar o executável, basta digitar "make" no terminal do Linux, dentro da pasta raiz. Ele irá gerar um executável de nome "tp2". Para executá-lo, basta digitar "./tp2" no terminal.

4. Análise de complexidade

Para a função "sort", que ordena o vetor em ordem crescente de tempo de visitação, a complexidade é a de um algoritmo de Mergesort - que é $O(n \times log(n))$. A função "void mergeSort(elements aux[], int esq, int dir)" chama a si mesmo recursivamente. A altura da árvore de recursão é $log(n)$ e para cada nível da árvore de recursão, o algoritmo realiza n operações de comparação, totalizando a complexidade final de $O(n \times log(n))$. A equação de recorrência para esse algoritmo é $T(n) = 2T(\frac{n}{2}) + n$. Aplicando o caso 2 do teorema mestre, obtemos que $T(n) = \theta(n \times log(n)) = O(n \times log(n))$. Já para o espaço, a complexidade deste algoritmo é de $O(n)$. Isso ocorre pois precisamos de um vetor auxiliar, com o mesmo tamanho do vetor original para ordená-lo no método Mergesort.

Para a função "sortName()", que ordena os vetores antes de serem inseridos na lista de visitação, em ordem alfabética, o algoritmo utilizado é o Radix Sort. O Radix Sort implementado, chama, para cada um dos caracteres do string que armazena o nome do planta, o algoritmo Counting Sort. O algoritmo do Counting Sort, por sua vez, possui complexidade de $O(n + m)$, sendo m o tamanho do vetor auxiliar de contagem criado. Como a ordenação é feita a partir de caracteres, o tamanho de m é de 256. Isso ocorre pelo fato do "char" ocupar 1 byte de memória apenas. Considerando isso, o n dominará o elemento m, chegando a uma complexidade de tempo de $O(n)$. A complexidade de espaço do algoritmo Counting Sort é a criação de um vetor auxiliar do mesmo tamanho do original mais a criação do vetor de contagem auxiliar, de tamanho máximo 256 elementos. Isso totaliza uma complexidade de espaço de $O(n)$.

Tudo isso dissertado acima sobre o Counting Sort é apenas uma parte do Radix Sort, que vai efetivamente ordenar o vetor em ordem alfabética. Para cada caracter do nome dos planetas, o método Radix Sort irá chamar o Counting Sort. Isso totalizará uma complexidade final de $O(n \times k)$, sendo k o número de caracteres que possui nos nomes dos planetas. Já para a complexidade de espaço, é a mesma do Counting Sort: $O(n)$ - uma vez que o Radix Sort, chama o Counting Sort para cada caracter do nome dos planetas.

5. Conclusão

O projeto foi muito interessante para aplicar na prática os conhecimentos aprendidos na sala de aula. Foi muito bom poder fazer e aplicar as estruturas de dados do tipo lista e vetor alocado dinamicamente.. Foi muito interessante a utilização e aplicação dos método de ordenação Mergesort, Counting Sort e Radixsort. O programa foi desenvolvido usando C++ e orientação a objetos, o que agregou bastante conhecimento e experiência.

Bibliografia

http://stackoverflow.com/

Bruce Eckel (2002) “Thinking in C++, Volume One: Introduction to Standard C++”.

Slides apresentados em sala de aula.