TRABALHO PRÁTICO 2:
O PROBLEMA DA AGENDA DE VIAGENS DE RICK SANCHEZ

Letícia da Silva Macedo Alves < leti.ufmg@gmail.com>

Matrícula: 2018054443

Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG) Belo Horizonte – MG – Brasil

1. Introdução

O problema especificado propõe o desenvolvimento de um programa para que o cientista Rick Sanchez possa organizar sua agenda de visitas interplanetárias. A partir da entrada fornecida, é alocado um vector com os número de planetas de interesse, onde tais planetas serão salvos com seus respectivos nomes e tempos de visitação. Para a organização da agenda de viagens, os planetas passarão por duas ordenações: a primeira, numérica, por seus tempos de visitação e a segunda, alfabética, por seus nomes. O resultado na saída serão as linhas de cada planeta, fornecendo, além dos tempos e nomes, o mês no qual serão visitados, de acordo com as especificações do problema, tal como a visita do máximo de planetas possíveis em cada mês. Na realização deste projeto, a linguagem utilizada foi C++11 e foram implementadas duas classes: Planeta e Ordenacao. A estrutura de dados escolhida para o armazenamento dos planetas a serem visitados foi o std::vector, facilitando a manipulação dos elementos (com base em indexação). O vector possui tamanho fixo, já que a quantidade de planetas é fornecida na entrada, logo, não é necessário utilizar funções específicas dessa estrutura como "push". Então, foi considerada adequada, atendendo perfeitamente o que se esperava da solução do problema, sem utilizar funções não permitidas. Para a verificação de alocação e desalocação correta de memória, foi utilizado o Valgrind.

2. Implementação

Os formatos de entrada e saída usados foram os padrões do sistema (stdin e stdout).

2.1. A classe Planeta

A classe possui os atributos: *std::string nomePlaneta* e *int tempoVisitacao*.
O único método que *Planeta* possui é seu construtor, que recebe seus dois atributos como parâmetros. Para armazenar os planetas é alocado, no método *void organizaAgenda* da classe *Ordenacao* um vector de ponteiros para *Planeta*, *std::vector<Planeta*>vetorPlanetas(numPlanetas)* onde o *numPlanetas* é fornecido na entrada, logo, o vector terá tamanho fixo. O endereço desse vector será parâmentro dos métodos utilizados na ordenação da agenda de visitas dos planetas (sendo os principais: *mergeSort*, *merge* e *radixSort*), que se encontram todos na classe *Ordenacao*.

2.2. A classe Ordenacao

Com base na natureza e especificação do problema, optou-se por utilizar como algoritmos de ordenação o MergeSort e o RadixSort, respectivamente para ordenação numérica e alfabética. Foi implementada então uma classe *Ordenacao*, na qual se encontram métodos aulixiares e os mais relevantes:

void organizaAgenda(int tempoMes, int numPlanetas, int tamanhoNome),

void mergeSort(std::vector<Planeta> &vetorPlanetas, int esquerda, int direita)*,

void merge(std::vector<Planeta> &vetorPlanetas, int esquerda, int meio, int direita)*,

void radixSort(std::vector<Planeta> &vetorPlanetas, int numPlanetas);*

2.2.1 O método organizaAgenda

O método será chamado na função *main*, onde os parâmetros são os 3 inteiros lidos (int t, p, c), e representam, respectivamente, o tempo em minutos para a visitação de planetas durante um mês, número de planetas a serem especificados e quantidade de caracteres para cada nome de planeta. É alocado o vector para os planetas e então, em um laço, são lidas as informações de cada planeta, para que sejam adicionados. Então, são chamados nos devidos momentos os métodos mergeSort e radixSort. Primeiramente o vetor todo de planetas passa pelo mergeSort, já o radixSort é responsável por ordenar alfabeticamente os planetas dentro de cada mês.Nessa ordenação desejada, em um laço serão definidos a que mês pertencem as visitas a cada um dos planetas, respeitando que não se deve ultrapassar o tempo total de visitação por mês e que deve ser visitado o máximo de planetas possível em cada mês e a agenda é mostrada por uma impressão.

2.2.2 Os métodos mergeSort e merge

Ambos são chamados pelo método *organizaAgenda*. O algoritmo de ordenação MergeSort se baseia na comparação de valores e divisão sucessiva do problema em partes menores por meio da recursividade. O conjunto com todos os tempos de visitação dos planetas será dividido em grupos, até que cada grupo contenha apenas um elemento. Depois são ordenados dois dos tempos, dois grupos de dois elementos, dois grupos de quatro e assim sucessivamente até que se obtenham todos os elementos em ordem.

Vale comentar que o algoritmo é estável (isto é, não troca a posição relativa entre dois valores iguais no vetor de saída) e que conta com uma memória extra auxiliar, que é um novo vector de ponteiro para Planetas de mesma dimensão que o principal.

O resultado dessa ordenação serão os tempos de visitação em ordem numérica crescente.

2.2.3 O método radixSort

É responsável por ordenar alfabeticamente os nomes dos planetas. Funciona de forma semelhante ao QuickSort, mas utilizará o CountingSort como método auxiliar e comparará o valor numérico inteiro associado a cada letra (com base no código ASCII), ao invés de valores numéricos em si. Como o nome é uma cadeia de caracteres, serão contadas as letras em cada posição da string para ordenação. Ou seja, se *c* é o número de caracteres do nome, haverão *c* contagens, cada uma para uma posição na string. Dessa forma, a cada contagem, o conjunto de nomes passa por uma ordenação parcial, até que a ordenação alfabética seja completa após todas as posições da string serem contadas e ordenadas.

3. Instruções de compilação e execução

A compilação é realizada utilizando o compilador GCC. Como o projeto faz uso de Makefile, presente na parta “src”, o processo de compilação dos códigos-fonte é resumido a um único comando “make”. Ou seja, uma possível sequência de comandos para compilação e execução do programa em sistema Unix seria:

"$ cd src"

"$ make"

"$ ./tp2"

Através do comando "Ctrl + D", correspondente ao EOF do sistema (final da entrada), o loop do while da main termina e a execução é encerrada.

4. Análise de Complexidade

O algoritmo MergeSort possui uma complexidade temporal de $n*\log_2(n)$ para qualquer entrada (não varia para pior ou melhor caso), onde n é o tamanho do vetor de planetas. A complexidade em termo de uso de memória é O(n), porém vale lembrar que o algoritmo requer o uso de uma memória extra proporcial a n, para juntar as partes ordenadas. Assim O(n)+O(n) = O(2n) = O(n).

Já o radixSort (implementado usando o countingSort como algoritmo auxiliar) possui uma complexidade temporal para qualquer entrada de (n+k)*k, onde n é o tamanho do vetor de planetas e k a quantidade de caracteres para o nome de cada planeta. Ou seja, O(n*k)+O(k*k), que acaba sendo O(n*k), como especificado no problema. Para a memória, o algoritmo é O(c)+O(n), onde c é o maior inteiro que poderá representar uma letra e n o tamanho do vetor de planetas auxiliar que será alocado.

Vale ressaltar que, como as operações de combinação de resultados e as demais do programa que não envolvem diretamente as alocações e manipulações dos vetores de planetas podem ser ignoradas na análise de complexidade.

Assim, a complexidade final temporal do programa será $O(n*\log_2(n)) + O(n*k)$. Como $k < \log(n)$, como especificado pelo problema, $O(n*\log_2(n)) + O(n*k) = O(n*\log_2(n)) + O(n*\log(n)) = O(n*\log(n))$.

5. Conclusão

Os algoritmos de ordenação MergeSort e RadixSort mostraram-se adequados no processo de ordenação da agenda de planetas respectivamente em termos do tempo de visitação e nome dos planetas. Foram atendidas as especificações de complexidade temporal para ambos os tipos de ordenação. O projeto possibilitou a exploração dos conceitos diferentes visto durante o curso, como algoritmos de ordenação, modularização, alocação dinâmica, uso de memória, complexidade de algoritmo e estruturas de dados. Puderam então ser praticados muitos dos conceitos e técnicas vistos durante o curso.

Bibliografia ZIVIANI, Nívio. (1999) "Projeto de Algoritmos com implementações em Pascal e C", São Paulo, Pioneira.

• Gráfico de análise experimental do tempo de execução do programa