Trabalho Prático 2 – Algoritmos 1

Avaliação do Evento

1. Identificação

Aluno: Matheus Flávio Gonçalves Silva

Matrícula: 2020006850

2. Introdução

O problema proposto consiste na análise das opiniões de um grupo de amigos a respeito de uma versão fictícia do Rock In Rio em que sempre vão as mesmas bandas na mesma ordem. Assim sendo, cada amigo do grupo atribuiu uma nota de -5 a 5 para cada show e quer ir somente no intervalo de shows em que a soma das notas de todos os amigos é a maior, posto que é possível entrar a qualquer momento, mas não pode retornar caso saia.

3. Implementação

O programa, bibliotecas e cabeçalhos utilizados foram desenvolvidos na linguagem C++, sendo utilizado o compilador G++ (com padrão de execução conforme previamente ditado no arquivo Makefile disponível no arquivo TP2-Template-CPP).

Como o trabalho é entendido como um problema de encontrar o subvetor com a maior soma dentre os possíveis, a resolução do mesmo se dá mediante a utilização de algoritmos de divisão e conquista similares às ideias apresentadas em aula. Ao final a execução, é impressa uma linha de dados com dois inteiros representando os shows de entrada e saída de forma do grupo de amigos.

3.1. Estruturas de Dados

Para a implementação do trabalho foi utilizado um Tipo Abstrato de Dados nomeado **“Rock”** que possui dois inteiros **“startShow”** e **“endShow”** que indicam os shows de entrada e saída do grupo de amigos, além de um número em ponto flutuante **“sum”** que armazena o valor da soma da nota para os shows do intervalo armazenado em **“startShow”** e **“endShow”**. Além disso, são utilizadas várias variáveis básicas para auxílio e um **vector<float>rockRates** para armazenar a soma das notas dos shows que é utilizada em alguns dos procedimentos.

3.2. Funções e Procedimentos

Arquivo rock.cpp:

float getINT_MIN(): Retorna o valor **INT_MIN** que corresponde a um valor de soma de notas extremamente baixo para comparações e chamadas de procedimentos;

Rock::Rock (): Cria e inicializa o TAD com os valores de entrada e saída em 0 e a soma com um valor negativo consideravelmente baixo;

int Rock::getStartShow(): Retorna o valor de **startShow** do **TAD**;

void Rock::setStartShow (int value): Atribui o valor de value para **startShow**;

int Rock::getEndShow(): Retorna o valor de **endShow** do **TAD**;

void Rock::setEndShow (int value): Atribui o valor de value para **endShow**;

float Rock::getSum(): Retorna o valor de **sum** do **TAD**;

void Rock::setSum (float value): Atribui o valor de value para **sum**;

float maxOf3Floats(float num1, float num2, float num3): Compara e retorna o maior float dentre os 3 passados como argumento para a função.

float Rock::max3PartedSum(vector<float> &rockRates, int startShow, int middle, int endShow): Função que recebe o vetor particionado em 3 subvetores e analisa qual das partições possui a maior soma das notas por meio de dois laços for, um para partição da esquerda e o outro para a da direita, além de comparar para com a soma completa do vetor. Atualiza os valores de **startShow**, **endShow**, e **sum** quando a soma de uma das partições é maior que a soma atualmente armazenada no **TAD**. Retorna um float com o maior valor de **sum** dentre os argumentos da função.

float Rock::maxSubArraySum(vector<float> &rockRates, int startShow, int endShow): Função que particiona recursivamente em 3 subvetores a entrada do vetor de dados com as somas das notas dos shows para comparação de qual subvetor do subproblema atual possui a maior soma das notas. Faz uso da função **max3PartedSum** para realizar as comparações. É responsável também por identificar o caso de parada de divisão em subvetores mediante análise de **startShow** e **endShow**. Retorna o maior valor de **sum** dentre os 3 subvetores que a função cria e analisa.

3.3. Programa Principal

O programa principal cria uma série de variáveis de auxílio para serem passadas como parâmetros às outras funções, faz a leitura dos dados passados por meio dos arquivos txt, e imprime na tela os valores dos shows de entrada e saída dos amigos para o Evento em virtude das notas apresentadas por eles. É responsável por ler todo o arquivo de entrada para os possíveis vários casos nele presente e parar somente com entrada 0 0 para a quantidade de amigos e de show.

3.4. Análise de complexidade

3.4.1. Espaço

A variável que domina assintoticamente o programa é a variável **vector<float>rockRates**, dado que todas as outras variáveis e até mesmo o **TAD Rock** possuem tamanho constante. Como é definida como **vector<float>rockRates(rockAmount)**, é **O(rockAmount) = O(n)** para espaço, considerando que **n** a quantidade de shows passada como argumento.

O(n).

3.4.2. Tempo

Para a leitura do arquivo de entrada, que depende assintoticamente da quantidade de amigos e de shows, são utilizados dois laços for para essa leitura, o primeiro iterando sobre a quantidade de amigos **m** e o segundo sobre a quantidade de shows **n**. Assim, para a leitura dos dados é **O(m*n)**.

Já para os procedimentos de análise da maior soma, em que são realizadas tripartições do vetor de notas recursivamente e, chegando ao caso base, a junção dos subcasos para formar a solução completa do problema. É similar ao que é feito no mergesort, mas análise de 3 dados provenientes da comparação entre as execuções **T(n/2) > T(n/2) > somaAtual**, de modo que a complexidade não é alterada pelas comparações, mantendo o padrão da divisão e conquista. Desse modo, os procedimentos são **O(n $\log_2$ n)**.

Como a especificação determina **1 <= m <= 50** e **1 <= n <= 100000**, a execução de todo o programa é dominada assintoticamente pela execução dos algoritmos de comparação da soma.

Assim o programa é assintoticamente delimitado como:

O(n $\log_2$ n)

4. Conclusão

De um modo geral, a implementação do trabalho ocorreu por meio de 3 implementações de solução após a leitura da especificação do TP, dado que a primeira implementação havia sido realizada de forma 100% funcional por meio de programação dinâmica, que não era permitido, a segunda era de certa forma lenta e ocorria timeout no último dos testes extras e a implementação final ocorreu sem mais problemas, com a primeira versão dela sendo feita por meio de structs e posteriormente alteradas para POO a fim de manter um entendimento mais claro dos métodos e estruturas utilizados.

Referências

TP2 – Avaliação do evento. Disponível em: < https://virtual.ufmg.br/20222/pluginfile.php/458555/mod_assign/introattachment/0/TP2.pdf?forcedownload=1 >. Acesso em: 15 nov. 2022.

Algorithmic Strategies. Disponível em: < https://virtual.ufmg.br/20222/pluginfile.php/429731/mod_resource/content/1/05DivideAndConquer-editedbyJussara-v2.pdf >. Acesso em 15 nov. 2022.