# Trabalho Prático 2

**DCC215 - Algoritmos 1**

**Igor Joaquim da Silva Costa**

## 1. Introdução

O problema proposto foi descobrir qual seria o intervalo de shows consecutivos que mais agradaria um grupo de amigos ao ir ao festival Rock In Rio. Mais precisamente, é apresentada uma sequência de shows e a nota de cada pessoa do grupo para cada um deles. O intervalo de shows que mais agrada um grupo é aquele onde a soma das notas dos shows no intervalo é a maior possível.

Para resolver o problema citado, cada show foi tratado como um número em um vetor, onde, a partir dele, devia-se calcular o subarray consecutivo com soma máxima, isso é, o par (x,y) onde x é pelo menos y e a soma do intervalo (x,y) é a máxima. Nesse sentido, o problema apresentado é reduzido à uma instância do Maximum Subarray Sum. Diante disso, foi implementado um algoritmo polinomial do tipo Dividir para Conquistar capaz de resolver o problema.

Diante do exposto, a documentação presente possui como objetivo detalhar como o sistema foi modelado (Seção 2), o quão eficiente ele pode ser (Seção 3) . Por fim, o projeto é sumarizado junto com os aprendizados gerados durante a produção do trabalho(Seção 4).

## 2. Modelagem

Esta seção tem como objetivo discutir as decisões que levaram à atual modelagem do programa.

### 2.1 Maximum Subarray Sum

Como elucidado na seção 1, o problema apresentado pode ser reduzido ao problema Maximum Subarray Sum, onde deseja-se encontrar qual é o par (x,y), com x pelo menos y, que possui a maior soma, dentre todos os pares (x,y) possíveis na instância. No contexto apresentado, cada posição x do array de entrada do problema Maximum Subarray Sum é a soma de todas as notas que cada pessoa do grupo deu para o show x. Dessa forma, o problema pode ser transformado em uma instância de Maximum Subarray Sum.

## 2.2 Resolver Maximum Subarray Sum

A fim de resolver o problema Maximum Subarray Sum (MSS), foi utilizado um algoritmo de dividir para conquistar. O problema apresentado pode ser quebrado em 3 partes, sendo N o tamanho do vetor da entrada.

O MSS do vetor que começa em 1 e termina em N pode ser de 3 formas, a primeira forma diz que MSS está entre 1 e $\frac{N}{2} - 1$. A segunda forma diz que o MSS está entre $\frac{N}{2} + 1$ e N. A terceira forma diz que, se o MSS não está nem completamente na metade esquerda, nem completamente na metade direita, ele obrigatoriamente deve possuir o elemento que parte o vetor ao meio.

Nesse sentido, o MSS de [1,N] é o Subarray máximo entre a metade da esquerda, a metade da direita e o maior array que o elemento que divide o vetor ao meio.

Além disso, devido à natureza do problema, sempre que dois Subarrays têm a mesma soma máxima, o desempate pelo critério maior número de elementos presentes no Subarray.

## 2.3 Encontrar Subarrays das metades e meio

O MSS das metades direita e esquerda é calculado recursivamente. Para calcular o Subarray que contém o meio, foi usada uma estratégia iterativa. O Subarray que contém o valor ao meio ou é o subarray máximo que termina no meio, ou é o subarray máximo que começa ao meio ou é a união dos dois últimos. Para calcular tais subarrays, basta iterar sobre (início,meio), para o subarray que termina no meio, e (meio,fim) para o subarray que começa ao meio, alterando a soma máxima sempre que uma nova soma máxima maior que a anterior tenha sido encontrada.

Dessa forma, é factível que o Maximum Subarray Sum possa ser resolvido em tempo polinomial, já que não existe sobreposição de problemas na hora de resolver as dependências de um intervalo em específico.

## 2.4 Estrutura de Dados

Para representar o array de entrada, foi usado o tipo padrão vector do c++, além disso, foi implementado um vetor de soma cumulativa, para facilitar o cálculo da soma de um intervalo.

## 3. Análise de complexidade

## 3.1 Espaço

Seja N o número de shows e M o número de amigos. Inicialmente, são criados dois vetores de N posições para representar os dados da entrada e a soma cumulativa. Assim, a complexidade de espaço se torna $O(N)$ na quantidade de shows.

## 3.2 Tempo

Para análise de tempo, considere N o número de shows e M o número de amigos.

### 3.2.1 Função de recorrência

Para se resolver o Maximum Subarray Sum, são feitas duas chamadas recursivas, cada uma com tamanho $\frac{N}{2}$, além de iterar sobre todos os elementos da instância passada. Tomando a equação de recorrência:

$$F(N) = 2 * F(\frac{N}{2}) + O(N)$$

O teorema mestre diz que tal função possui complexidade $O(Nlog_2(N))$ . Logo, o algoritmo para resolver o problema cresce linear-lognalmente no número de shows.

## 4. Conclusões

Com o intuito de encontrar o maior intervalo de shows que agrade a maior parte de um grupo de amigos, foi implementado um programa que utiliza algoritmos de dividir para conquistar para resolver o problema.

Durante o projeto do sistema foram levadas em consideração não só aspectos práticos da implementação de uma modelagem computacional, mas também como a linguagem de programação escolhida poderia ser uma ferramenta útil para chegar no objetivo esperado. Toda a questão de mapear um mini-mundo de interesse em um modelo computacional robusto se mostrou bastante produtiva, levando o aluno a pensar em formas criativas de se resolver e entender o problema, tendo como resultado um extenso aprendizado sobre como pensar, questionar e implementar um algoritmo de dividir para conquistar na prática. Por fim, o tempo

extra usado para projetar o sistema trouxe várias recompensas no sentido da implementação, sendo um aspecto a ser levado para trabalhos futuros.

Nesse sentido, todo o fluxo de trabalho foi essencial para a consolidação de conteúdos aprendidos em sala, além de apresentar, de forma prática, como softwares maiores, mais consistentes, robustos e inteligentes são projetados e implementados.

## 5. Instruções para compilação e execução:

### 5.1 Compilação

Existem partes do programa que são compatíveis apenas às versões mais recentes da linguagem c++, dito isso, deve-se seguir as seguintes configurações para a compilação:

Linguagem: C++
Compilador: Gnu g++
Flags de compilação: -std=c++11 -g
Versão da linguagem: standard C++11
Sistema operacional (preferência): distribuições baseadas no kernel Linux 5.15.

O comando para compilar o programa automaticamente está presente no arquivo **“Makefile”** e sua execução é chamada pelo comando **“make all”**. Deste modo, o executável “tp02” estará compilado e pronto para ser utilizado.

### 5.2 Execução

Seguem as instruções para a execução manual:

1. Certifique-se que o compilável foi gerado de maneira correta, se algum problema ocorrer, execute o comando “make all” presente no “Makefile”.

2. Uma vez que os passos anteriores foram cumpridos, execute o programa com o comando: ./tp02 < Teste01.txt )
3. A saída será impressa no terminal.