Trabalho Prático 2

Avaliação do evento

Ítalo Dell'Areti

Universidade Federal de Minas Gerais (UFMG)

Belo Horizonte – MG – Brasil

italodellareti@ufmg.br

1. Introdução

O problema a ser resolvido é, dado um evento,o festival Rock In Rio, no qual haverá vários shows que aconteceram em sequência e existe apenas um palco, o que faz com que a apresentação dos grupos seja em sequência. Estes amigos, querem aproveitar ao máximo o festival sem perder tempo com shows que não valem a pena assistir e, dessa forma, criaram um aplicativo para dar nota para cada show e dando avaliações entre -5 e 5.

A finalidade deles é somar todas as notas de um mesmo show e descobrir qual será o intervalo de shows consecutivos que mais agradou o grupo. Logo, podendo prever e determinar o retorno no próximo festival.

Ao final, devemos conseguir determinar se é viável a realização dessa escolha, afirmando a possibilidade ou negando a existência de solução.Sendo assim, com base nas características e propriedades de divisão e conquista, podemos implementar este problema de forma a respeitar todas as nuances da problemática. Tendo em vista disso, este documento possui como objetivo explicitar o funcionamento desta implementação.

2. Implementação

O programa foi desenvolvido utilizando a linguagem de programação C++, compilada pelo G++ da GNU Compiler Collection. Ademais, o programa foi produzido e testado em um ambiente Linux distribuição Ubuntu 22.04 LTS utilizando Visual Studio Code.

2.1 - Modelagem

A priori, tendo em vista o problema, devemos discretizá-lo e expressar suas necessidades matematicamente.Como requerido na implementação, devemos utilizar algoritmos de divisão e conquista em nossa Implementação, modularizando e satisfazendo as exigências.

Como entrada do arquivo, na primeira linha temos dois números inteiros, A e S, que respectivamente representam o número de amigos que assistiram aos shows

e o número de shows que aconteceram no festival. Após isso, temos linhas que descrevem as avaliações dos integrantes do grupo. As avaliações são dadas por uma sequência de S números reais separados por espaço. Sendo assim, dados as informações fornecidas, podemos estruturar nossos elementos em um array, no qual devemos buscar o melhor subarranjo. Tendo em vista isso, iremos utilizar o problema do subarranjo máximo usando o algoritmo de divisão e conquista.Esta a ferramenta possui os seguintes passos:

1 - Dividir o array em 2 partes

2 - Retorna o máximo dos três seguintes

- Soma máxima do subarray na metade esquerda (Recursivamente);
- Soma máxima do subarray na metade direita (Recursivamente);
- Soma máxima do subarray, tal que o subarray intercepta o ponto médio.

A ideia é encontrar a soma máxima, começando no ponto médio e terminando em algum ponto à esquerda do meio, então encontre a soma máxima começando no meio + 1 e terminando em algum ponto à direita do meio + 1.

O problema do subarranjo máximo é interessante somente quando o arranjo contém alguns números negativos e é o que ocorre em nosso caso, pois temos notas entre -5 e 5. Em suma, o problema é encontrar a maior soma de um subarray contíguo dentro de um arranjo unidimensional, o que se encaixa perfeitamente em nosso problema, devido aos shows ocorrerem em seguida e possuirmos os valores das notas em arrays. Para uma melhor visualização do algoritmo, vamos exemplificar seu pseudo código.

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
1 left-sum = −∞
2 sum = 0
3 for i = mid downto low
4       sum = sum + A[i]
5       if sum > left-sum
6               left-sum = sum
7               max-left = i
8 right-sum = −∞
9 sum = 0
10 for j = mid + 1 to high
11      sum = sum + A[j]
12      if sum > right-sum
13              right-sum = sum
14              max-right = j
15 return (max-left, max-right, left-sum + right-sum)
```

Nesta primeira parte achamos o subarranjo máximo da metade esquerda, depois achamos o subarranjo máximo da metade direita e , ao final, retornamos os índices que limitam o subarranjo máximo que intercepta o ponto médio, junto com os elementos da soma left-sum e right-sum.

```
FIND-MAX-CROSSING-SUBARRAY(A; low; high)
1 if high == low
2       return (low; high;A[low])                    // caso base: só um elemento
3 else mid = ⌊(low + high)/2⌋
4       (left-low, left-high, left-sum) =
                FIND-MAXIMUM-SUBADRRAY(A, low, mid)
5       (right-low, right-high, right-sum) =
                FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
6       (cross-low, cross-high, cross-sum) =
                FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
7       if left-sum ≥ right-sum e left-sum ≥ cross-sum
8               return (left-low, left-high, left-sum)
9       elseif right-sum ≥ left-sum e right-sum ≥ cross-sum
10              return (right-low, right-high, right-sum)
11      else return (cross-low, cross-high, cross-sum)
```

A segunda parte começa com o caso-base de teste quando só se tem um elemento e, consequentemente, apenas um subarranjo.Após isso ele retorna a tupla com os índices do início e o fim do único elemento, juntamente com seu valor e realizando esta tarefa de modo recursivo.

2.2 - Organização e Análise

Algumas das funções do programa são:

max_crossing_sum() : Encontra a soma máxima possível no arranjo, possuindo complexidade de tempo O(nlog(n)) e complexidade de espaço O(1).Esta função é a implantação da parte 1 do pseudocódigo find-max-crossing-subarray do Thomas H. Cormen.

max_sub_array_sum() : Retorna a soma do subarray de soma máxima, possuindo complexidade de tempo O(nlog(n)) e complexidade de espaço O(1).Esta função é a implantação da parte 2 do pseudocódigo find-max-crossing-subarray do Thomas H. Cormen.

escolhe_melhores_shows(): Como o próprio nome indica, retorna os melhores shows, de for a usar a própria função **max_sub_array_sum()** para solucionar o problema.

2.3 - Estrutura de dados

Basicamente possui uma struct chamada resultado que armazena os índices do início e fim do array, além do valor da soma do subarray.

Instruções para compilação e execução

Para executar o programa basta dar um comando **make** para executar ou make clean para remover objetos e o executável.Além disso, o arquivo de texto de entrada **deve estar no diretório raiz da pasta do programa para o seu funcionamento. Sempre que mudar quaisquer valores de entrada**, você deverá executar o comando **./tp02 < arquivo_de_entrada.txt** para atualizar os valores da saída.

Para salvar os valor de saída em um arquivo de saida basta digitar o comando

./tp02 < arquivo_de_entrada.txt > arquivo_de_saida.txt

Vídeo com a implementação funcionando

Dropbox: https://www.dropbox.com/s/ag92gyau05ictc9/TP2%20ALG.mp4?dl=0

GoogleDrive:
https://drive.google.com/file/d/1NQ6ZWSxsFZ9A0bdlSWH88sPtbKXcP0Tm/view?usp=sharing

Bibliografia

Geeks for geeks: Maximum Subarray Sum using Divide and Conquer algorithm

Geeks for geeks: Divide and Conquer

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Algoritmos: Teoria e Prática. 3a edição.