# Algoritmo e Estrutura de Dados III

Trabalho Pratico 0

Isabela Meneguci de Souza Petrim

Agosto de 2017

## 1 Introdução

O trabalho prático 0 apresenta um garoto hipotético chamado Nubby que tem como um dos hobbies a programação. Nubby foi desafiado por seu amigo a resolver um problema onde dado um vetor "v"deve-se realizar uma busca pelo vetor, dado um intervalo ele deve retornar o máximo, mínimo e soma desse intervalo. Além disso, há duas operações adicionais que é adicionar ou subtrair um elemento em todo o intervalo passado.

Para resolver esse problema Nubby resolveu construir uma matriz quadrada guardando todos os intervalos com a soma, máx. e min. Caso a função adicionar ou subtrair seja chamada é preciso recalcular toda a matriz novamente. Seu amigo, lhe propôs uma outra maneira de resolver o problema, utilizando uma arvore de segmentos, que revolveria o problema de forma mais eficiente. Um dos motivos da maior eficiência desse método é que único elemento, ou intervalo, do vetor for submetido a uma operação de adição ou subtração não é necessário recriar toda a árvore.

O trabalho pode ser comparado a algumas situações "reais"e aplicáveis no dia a dia prático ou até mesmo dentro de uma empresa. Um exemplo prático para esse trabalho poderia ser aplicado a uma escola que deseja consultar qual é o máx. e min de notas de seus alunos. Além disso, há outras diversas situações práticas que podem ter sido relacionadas como motivação.Para realização do trabalho alguns conceitos são importantes como entender o funcionamento de uma matriz e como realizar sua implementação, entender uma árvore de busca binária e a árvore de segmentos e como realizar sua implementação na linguagem C.

# 2 Solução do Problema

Para resolver o problema apresentado são necessárias duas implementações uma matriz quadrada e uma árvore de segmentos, que devem retornar o mesmo valor para uma mesma consulta.

Para implementação do problema usando uma matriz fez-se necessário a criação de funções para calcular a soma, min, máx., adicionar e subtrair, além de uma função para construir a matriz, o "main". Nessa implementação em específico foi utilizada uma estrutura de dados (struct) para guardar o valor da soma, min e máximo em cada posição da matriz.

As funções min, máx. e soma implementados nesse código são relativamente triviais. Os três funcionam basicamente da mesma forma, eles recebem como parâmetro o intervalo que irá ser calculado e o vetor. A função soma através de um loop, recebe o valor inicial do vetor e soma com o próximo. A função min e máx atribuem o valor inicial como sendo o min/max e vão comparando posição a posição no vetor para saber se o valor atribuído inicialmente é menor/maior que o próximo se for, min/max recebe esse novo valor. Para melhor vizualização segue o pseudo código das funções soma e mínimo (máximo foi obtida por ser similar ao mínimo):

**Algoritmo 1** soma.c

```
function SOMA(int i , int j, int *v)
    for (count=i;count<=j;count++) do
        soma = soma +v[count];
    return soma;
```

**Algoritmo 2** min.c

```
function MIN(int i , int j, int *v)
    min=v[i];
    for (count=i;count<=j;count++) do
        if (v[count]<min) then
            min=v[count];
    return min;
```

As funções de adição e subtração, também funcionam de forma similar as de cima. Recebem um intervalo a ser percorrido e somam ou subtraem 1 a cada valor do intervalo passado. Após a implementação das funções acima é realizada a função Constrói Matriz, ela é um ponteiro do tipo "Operações"que foi a estrutura de dados definidos. Essa função tem como objetivo alocar

memória para as linhas e colunas da matriz e em cada intervalo da matriz guardar o resultado das funções min, máx e soma , passando para essas funções a linha, coluna e o vetor.

**Algoritmo 3** operacoes.c

```
function ConstroiMatriz((int tamanhov , int *v ))
    Operacoes **M = (Operacoes**) malloc((tamanhov+1)*sizeof(Operacoes*));
    for (a=1;a<=tamanhov;a++) do
        M[a] = (Operacoes*) malloc((tamanhov+1) * sizeof(Operacoes));
        for (b = 1; b <= tamanhov; b++) do
            M[a][b].min=min(a,b,v);
            M[a][b].max=max(a,b,v);
            M[a][b].soma=soma(a,b,v);
    return M;
```

Outra função é o "main", de nome matriz.c, a função primeiramente lê duas variáveis que serão passadas, a primeira delas é o tamanho do vetor e a segunda a quantidade de operações que serão feitas, logo em seguida a função aloca memória dinamicamente para o vetor de acordo com a quantidade lida na primeira variável. Após alocar a memória para o vetor, a função lê os elementos que são digitados do teclado para preencher o vetor. Em seguida, chama-se a função Constrói Matriz e é passado para ele o tamanho do vetor que foi lido e o vetor já preenchido. O loop que vem em sequência é o responsável por ler e executar as operações e o intervalo do vetor que o usuário digitar, as operações são a soma, máximo, mínimo, adição e subtração. Dependendo da operação que foi digitada a função chama a posição da matriz, referente a posição do vetor que o usuário solicitou, e realiza a consulta do resultado.

A resolução do problema utilizando uma árvore de segmentos, tem basicamente o mesmo princípio da implementação da matriz. O objetivo é armazenar os valore do máximo, mínimo e soma de todos os intervalos do vetor. Contudo, nessa árvore os nós não folhas armazenam essas operações para os intervalos de um único elemento (por exemplo o intervalo 1,1), os nós não folha armazenam o valor da união dos seus filhos, assim até a raiz. Uma das vantagens da busca pela árvore é o fato de não precisar consultar toda a árvore apenas um dos filhos ou o da direita ou o da esquerda. Além disso, quando é chamado a operação de adicionar ou subtrair não é preciso refazer toda a árvore (na matriz isso e necessário), apenas os ramos da direita ou esquerda.

Para construir a árvore primeiramente é alocado memoria dinamicamente para a arvore, e realizado a função que constrói a árvore com o valor do mínimo, máximo e soma de forma recursiva nesse processo também é alocado memória para os filhos da direita e esquerda da arvore. Nessa árvore, cada um de seus nós contém o valor das três operações para cada um dos intervalos da matriz. Uma validação inicial é realizada para reificar se "i==j", ou seja, se está construindo um nó folha. Se a condição for verdadeira basta fazer mínimo, máximo e soma receber o valor do intervalo (que será por exemplo 1 1, o máximo de 1 1 é o próprio número).

**Algoritmo 4** ConstroiTree.c

```
function CONSTROI((int *v, int i, int j, TipoApontador SegmentTree))
    if (i==j) then
        SegmentTree->soma = v[i];
        SegmentTree->max = v[i];
        SegmentTree->min = v[i];
        SegmentTree->i = SegmentTree->j = i;
        SegmentTree->soma
```

Caso contrário, para isso para construir o mínimo/ máximo / soma a função chama outras funções que recebem um intervalo e um vetor e calcula o mínimo/ máximo / soma desse vetor, da mesma forma que ocorre na matriz.

**Algoritmo 5** ConstroiTree.c

```
function CONSTROI((int *v, int i, int j, TipoApontador SegmentTree))
    SegmentTree->i = i;
    SegmentTree->j = j;
    Filho da esquerda = Aloca memoria para o filho da esquerda;
    Filho da Direita = Aloca memoria para o filho da direita;
    SegmentTree->min = Minimo(i,j,v);
    SegmentTree->max = Maximo(i,j,v);
  SegmentTree->soma = Constroi(v, i, meio, SegmentTree->filho_esq) +
Constroi(v, meio + 1, j, SegmentTree- > filho_dir); -
```

Nesse ponto, basicamente a árvore esta construída. Só são necessárias as funções que caminham pela árvore e retornam o valor do mim / máx. / soma para um determinado intervalo. Para fazer essa consulta, tanto para a mínimo/ máximo / soma foram feitas três funções recursivas para caminhar pela árvore e retornar o valor da operação. Na função main segue o mesmo padrão da matriz, recebendo o valor da do número de operações que será realizada e chamando dentro do loop as funções para cada operação digitada.

Na função add e subtração é realizada a adição ou subtração dos elementos do vetor e após esse processo chamado a função para reconstruir esses nos da árvore, sem precisar reconstruir a árvore inteira.

# 3 Análise de Complexidade

Nessa seção será apresentada a análise teórica de custo para os dois algoritmos a árvore e a matriz.

Complexidade da matriz

* Na função soma, min , máx , add e sub ao receberem o vetor e o intervalo realizam um loop do intervalo inicial que foi passado até o intervalo final, tendo complexidade O(i-j)

* A função que constrói a matriz possui dois loops, um que inicia do 1 indo até o tamanho do vetor e dentro desse loop está contido outro loop que percorre cada coluna da matriz , sendo assim a complexidade dessa função é $O(n^2)$

* Na função MAIN-Matriz possui um loop que vai de 1 até a quantidade de operações que serão realizadas, tendo complexidade O (m)

* Em uma análise geral dentro do main já possui complexidade O(m) do seu loop, ao chamar as funções presentes nele a complexidade assintótica só irá ficar cada vez maior, sendo assim a complexidade total pode ser estima como a multiplicação da complexidade do loop interno , vezes a complexidade das funções sub, add, mas, min, soma e constrói matriz : O $(m*(n^2) * (n^2))$

Complexidade da árvore

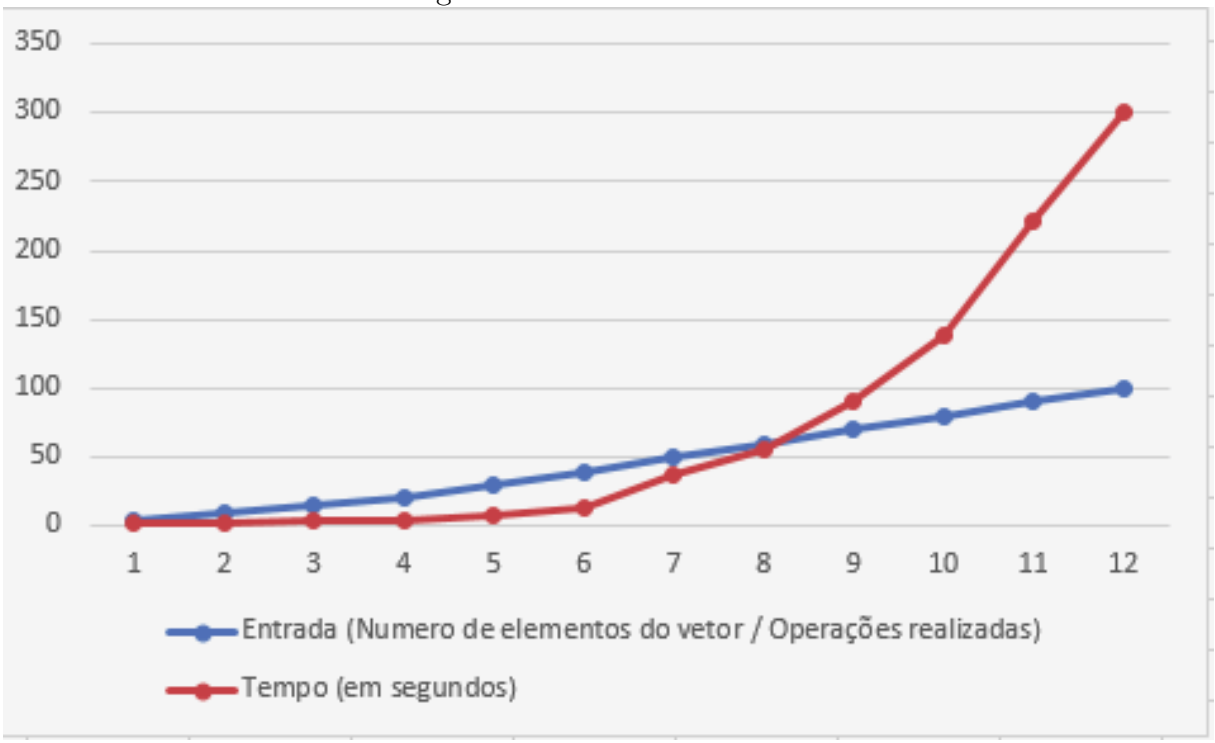
* A complexidade das funções soma, min , máx , add e sub são iguais a da matriz O(i-j) As funções Controi, ConsultaSoma, ConsultaMin, ConsultaMax , ConsultaAdd, ConsultaSub são recursivas tendo cada uma delas complexidade semelhante, sua complexidade é de O(log(n))

* A função MAIN-Tree possui um loop que vai de 1 até o valor das operações tendo complexidade O(m). Além disso, a complexidade total é a multiplicação de cada umas das funções que são chamadas dentro do main. Sendo O(m*log(n)*log(n)*log(n)*log(n)*log(n))

# 4 Análise Experimental

Na seção a seguir é feita uma análise experimental da complexidade dos algoritmos. Será inserido os dados observados em um gráfico para melhor visualização.

Figura 1: Matriz




# 5 Conclusão

Após realizar a implementação das duas maneiras nota-se que a árvore de segmentos é muito mais eficiente que uma matriz, por ter um custo bem menor e gastar bem menos tempo quando as entradas vão aumentando. Contudo, a implementação da árvore não é tão trivial quanto a matriz, sendo assim para valores pequenos de até 10 elementos no vetor e fazendo 10 operações a utilização da matriz é tão eficiente quanto a árvore, mas com a vantagem de possuir implementação mais fácil e de implementação mais rápida.

Figura 2: Tree