



Universidade Federal
de São João del-Rei

Algoritmos e

Estruturas de Dados III

Professor: Leonardo Rocha

Trabalho Prático 2

Felipe Caldas de Paiva

1 Introdução

O Problema deste trabalho prático foi caracterizado com base no filme Harry Potter, e descreve uma situação em que Harry está fugindo de Voldemort e deseja encontrar um certo artefato. Harry recebe um grid mágico mostrando a localização deste artefato(uma matriz $S(R \times C)$) e deve chegar até ao artefato sem que sua energia chegue a **ZERO**. O artefato sempre se localizará na posição $S[R][C]$ e Harry sempre começa da posição $S[0][0]$. Se sua energia chegar a **ZERO**, Voldemort o encontrará e vence a disputa. Logo, para isso, ele deve percorrer essa matriz $S(R \times C)$ e encontrar o caminho que vai de $S[0][0]$ até $S[R][C]$ que gasta menos energia possível, considerando que Harry pode se locomover apenas para baixo e para a direita.

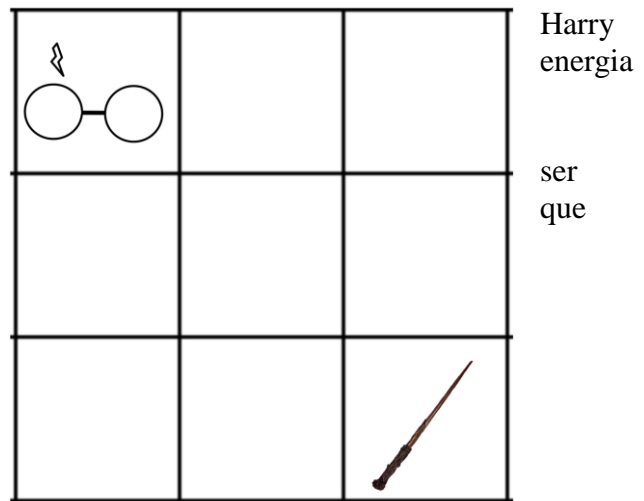
Sua energia se define pelos valores de cada posição da matriz, sendo sua energia inicial o valor de $S[0][0]$ e a energia gasta ou ganha em cada movimento o respectivo valor da posição que escolher ir. Para valores positivos, Harry ganha e para valores negativos ele perde energia.

Com isso, pode-se perceber que o problema a solucionado envolve encontrar um caminho para Harry chegue até o artefato de maneira segura.

Neste trabalho, Foram pedidas 2 estratégias diferentes para resolver esse problema.

2 Solução

Figura 1: Exemplo de grid mágico



De inicio, é necessário entender como o programa funciona, antes mesmo de começar as etapas de programação em si.



Figura 2: Fluxograma do funcionamento do programa

A divisão de etapas facilita a modularização do código em si, permitindo melhor visualização das tarefas e de cada funcionalidade do código.

2.1 Ler Dados do Arquivo

A execução do programa acontece através da linha de comando `./tp2 <estrategia> entrada.txt` que lê a estratégia escolhida pelo usuário e armazena os dados passados no arquivo de entrada através da função `fscanf()`, sendo possível ler o número de casos de teste que serão testados, as dimensões de cada matriz e ,através de 2 *laços for* alinhados, a respectiva matriz.

2.2 Estruturas implementada

Para guardar os dados lidos, usou-se de 2 estruturas de dados no programas.

```
typedef struct
{
    int R, C;
    int **matriz;
} Matriz;
```

A primeira estrutura é responsável por guardar as dimensões de uma matriz e um ponteiro para a matriz em si.

```
typedef struct
{
    int estrategia;
    Matriz *casos;
    int n_casos;
} File;
```

Essa estrutura por sua vez é responsável por armazenar a estratégia escolhida pelo usuário, o número de casos que serão testados e uma matriz para cada caso específico.

2.2 Executar a Estratégia escolhida

Após ler a estratégia passada por linha de comando e os dados de entrada, o programa realiza um *switch/case* para executar a respectiva estratégia escolhida e retorna a energia mínima necessária para chegar até a posição $S[R][C]$.

Obs.: Como foi solicitado a energia MÍNIMA, para valores de energia maiores do que zero, as funções retornam 1, já que qualquer valor acima disso seria energia de sobra.

3 Estratégias

3.1 Estratégia 1: Algoritmo Guloso

Para a primeira estratégia, realizamos um algoritmo guloso que recebe a matriz fornecida como parâmetro. Este algoritmo compara o valor da posição a direita e abaixo e retorna qual o maior, após essa comparação, incrementa energia com o maior valor e segue na respectiva direção.

Tendo como exemplo a situação representada na figura abaixo, o algoritmo guloso faria a comparação, inicialmente, entre 1 e -1, sendo 1 o valor maior. A energia então seria incrementada com esse valor e Harry seguiria para a posição que esse valor ocupa. Novamente o algoritmo compararia 2 e 2 e, como os valores são iguais, de forma arbitrária, o algoritmo prefere ir para baixo. Esse Processo é repetido até chegar na posição desejada, retornando a energia mínima necessária. Em caso do valor final ser negativo o algoritmo retorna o valor de energia com o sinal trocado incrementado de 1, pois se fosse o mesmo valor, ao final de tudo, a energia chegaria a 0 (ex: energia = -3, $3 - 3 = 0$, logo a energia mínima teria que ser 1). E em caso do valor final ser positivo ele retorna apenas 1, pois, como dito antes, essa seria a energia mínima já que qualquer valor acima disso seria 'desnecessário', não influenciando em nada.

grid

3.2 Estratégia 2:

Na segunda estratégia, dinâmica para encontrar basicamente em criar seus valores são a

em cada casa a partir da origem. Usando o grid acima como exemplo, o valor para se chegar na posição S[0][2] seria 3 pois 0+1+2. Porém o algoritmo leva em conta a comparação entre os adjacentes da posição que deseja chegar para definir seu valor. Para cada posição ele escolhe o adjacente de maior valor e soma este valor ao da posição atual. Em termos matemáticos a programação dinâmica deste algoritmo funciona basicamente assim:

$$M(i,j) = \begin{cases} \max(M(i,j-1), M(i,j+1), M(i-1,j), M(i+1,j)) + 1 & \text{se } i > 0 \text{ e } j > 0 \\ \max(M(i,j-1), M(i,j+1)) + 1 & \text{se } i = 0 \text{ e } j > 0 \\ \max(M(i-1,j), M(i+1,j)) + 1 & \text{se } i > 0 \text{ e } j = 0 \\ 0 & \text{se } i = 0 \text{ e } j = 0 \end{cases}$$

Figura 4: Definição da Programação Dinâmica

Após criar a nova tabela com os valores escolhido, o valor que estiver ocupando a posição T(RxC) será a solução, retornando assim, a energia necessária, baseando-se no mesmo conceito implantado na estratégia 1 para valores negativos e positivos.

3 Gravando resultado no arquivo de saída

Tendo o resultado obtido após a execução de umas das duas estratégias, o último passo é gravar esse resultado em um arquivo de saída “saida.txt”, usando a função **fprintf()**. O programa salva o resultado para cada um dos casos no arquivo e fecha ambos os arquivos de entrada e saída.

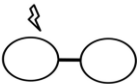

	1	2
-1	2	-3
1	0	 -1

Figura 3: Exemplo de

Programação dinâmica

4 Análise de Complexidade

4.1 Estratégia 1

Para a Estratégia 1 o algoritmo é feita com base no tamanho da matriz ($R \times C$). O algoritmo realiza um loop *while* que vai realiza comparações até chegar à posição $S[0][0]$ da matriz. Em cada posição faz 2 comparações até chegar ao final. Como se trata de um algoritmo guloso, fazendo com que não percorra todos as posições da matriz, sua complexidade pode ser definida levando em conta cada movimento na matriz que serão $R - 1$ movimentos para baixo e $C - 1$ movimentos para cima, levando a um numero total de $2*(R-1+C-1)$ movimentos, logo sua complexidade fica sendo como $O(n)$. Segue abaixo um gráfico (feito em python) do tempo gasto para cada execução do código referente ao número de casos pedidos.

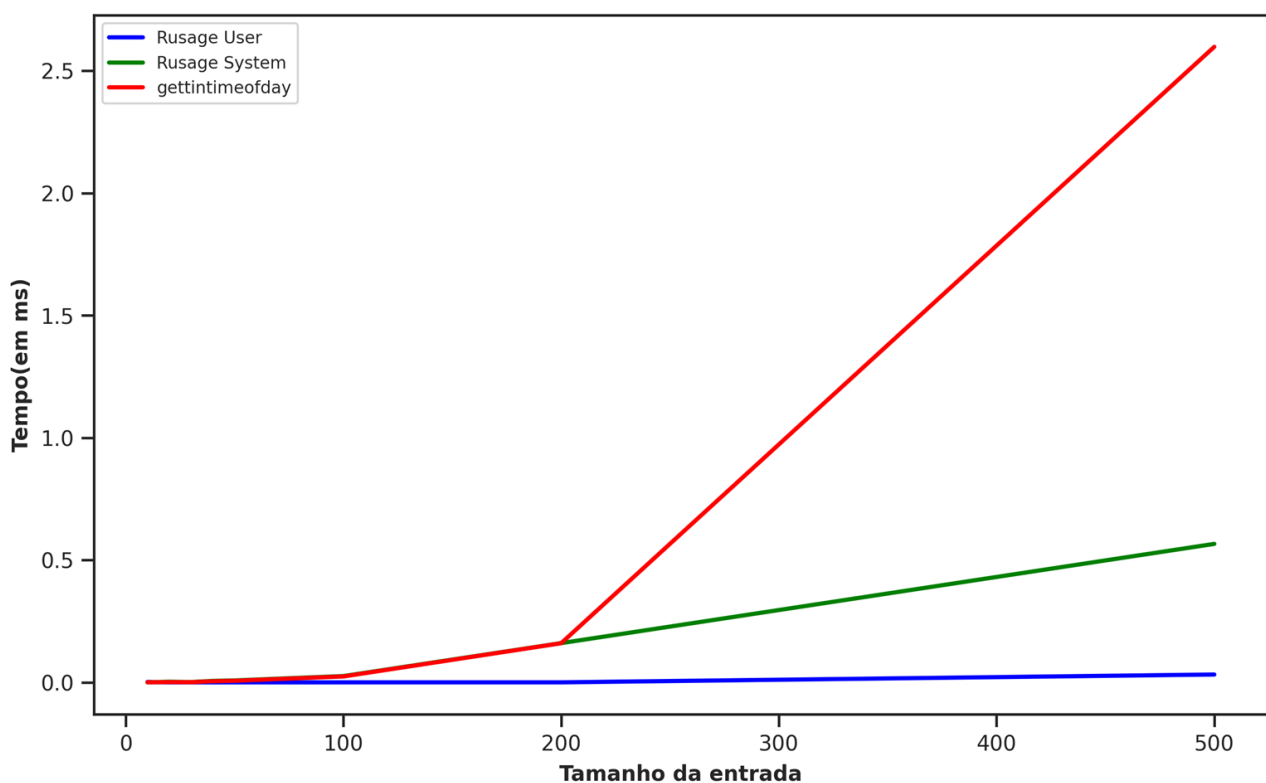
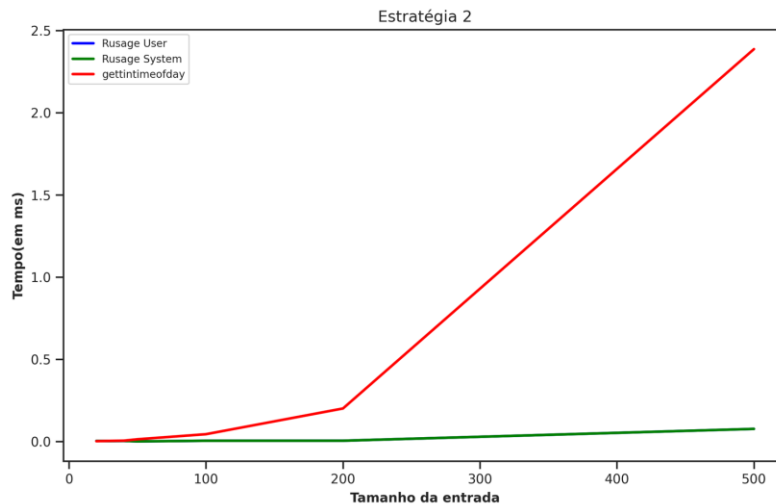


Figura 5: Gráfico referente à complexidade da Estratégia 1

4.2 Estratégia 2

Considerando a abordagem usada na estratégia 2, a função precisa percorrer toda a matriz e refazê-la com os novos valores, realizando comparações a cada posição para definir seu valor, para isso,

foram usados 2 laços *for* alinhados, fazendo com que um laço realize **R - 1** comparações e o outro **C - 1** comparações. Por fim, a função em sim acaba por realizar $(R - 1) * (C - 1)$ comparações, levando a uma complexidade $O(n^2)$.



5

Conclusão

Por fim, podemos perceber que a eficiência desse algoritmo está diretamente relacionada com o tamanho da entrada de número de casos passada a ele. Para entradas muito pequenas, a diferença de eficiência entre as 2 estratégias é irrelevante, porém, a solução entregue pelo algoritmo guloso nem sempre vai ser a ótima, diferente da solução entregue pela estratégia 2 que usa programação dinâmica para encontrar o resultado.

Dizendo em termos menos técnico e aproveitando da caracterização do trabalho que faz uso do livro “Harry Potter” como tema, se Harry estivesse querendo chegar ao artefato da maneira mais rápida possível seria recomendado o uso da Estratégia 1, permitindo a ele realizar menos movimentos e, ainda assim, chegar ao artefato. Todavia, caso ele estivesse buscando o caminho mais seguro e tempo não fosse um problema para ele, a estratégia 2 se mostra mais eficiente.