Ciência da Computação

Projeto e Análise de Algoritmos III

Documentação Trabalho Prático tp-2

Vitor Ferreira França

2024

1-Introdução

O problema a ser solucionado é encontrar a máxima pontuação possível no jogo onde. Dado uma sequência de números inteiros, O jogador pode fazer várias jogadas. Em cada jogada ele pode escolher um elemento da sequência (vamos denotá-lo ak) e excluí-lo, sendo que os elementos ak+1 e ak−1 também devem ser excluídos da sequência. Essa jogada traz ak pontos para o jogador.

Exemplo:

Sequancia:“3 ,6 ,7 ,5 ,3”

-É escolhido o primeiro item “3” a1

Sequência após “7,5,3”

Pontuação =3

-É escolhido o último item “3” ak

Sequência após “7”

Pontuação =6

-É escolhido o item “7” ak

Sequencia após “ ”

Pontuação =13

Esse é a melhor pontuação possível

13

2- Construção e funcionamento do Algoritmo

2.1- Chegada de dados pelas linhas de comandos

É passado primeiramente a estratégia que será usada (“D” ou “A”), logo após, qual arquivo de input deverá ser utilizado, exemplo: `./tp2 d input0.txt` .

2.2- Variáveis de entrada

O arquivo “.txt” informara o tamanho do vetor e seus itens , desta forma

“A primeira linha contém o inteiro N (0 ≤ N ≤ 105) que mostra quantos números existem na sequência de Joâo. A segunda linha contém N inteiros a1, a2, ..., an.

Exemplo de entrada:

9

1 2 1 3 2 2 2 2 3 ”

2.3- Funções de abrir o arquivo para leitura e escrita no output

-A função para abertura de um arquivo , abre o mesmo utilizando o nome inserido pelo terminal ,por meio da construção de um caminho de onde estará localizado o input , após isso o caminho é aberto como uma “FILE” e retornada como um ponteiro para ser utilizada.

-**A função de escrita no output** recebe a resposta para que seja mandado para o “output.txt” ,após isto é construído o caminho para o arquivo para escrita e

Se caso ocorra um erro na abertura é printado o erro no terminal, após isso é fechado o arquivo.

2.3- Construindo a TAD que vai armazenar a sequência e suas funções

Foi utilizado um vetor dinâmico para tal, pela sua estrutura como vetor e sua dinamicidade para adicionar e retirar itens. Ela contem desta forma o array que armazena os itens ,seu tamanho e capacidade total do mesmo.

-Primeiramente é usado a função para criar o array dinâmico que aloca a memória inicial para o vetor com sua capacidade tamanho, e é retornado.

-A Função de adicionar primeiramente checa se é preciso redimensionar o vetor, logo após é adicionado o item na ultima posição do vetor .

-A função de remover um elemento primeiramente verifica se o índice está dentro dos limites válidos do array, após isto ela reposiciona todos os itens do vetor e e diminui seu tamanho.

-**A função de printar o array** feito para testes durante o desenvolvimento do software, recebe e percorre o vetor printando o que está nele “displayArray”.

-**A função para deletar** o item escolhido e os seus vizinhos do lado percorre faz uma pesquisa no vetor para achar o item desejado, e deletar ele os ao seu lado no vetor

2.4- Forma dinâmica da resolução do problema

-A função “biger” é uma função auxiliar que retorna o valor máximo entre dois números. Ela recebe dois números inteiros como entrada e retorna o maior deles.

-A função “Dinamica” é a função principal que calcula a pontuação máxima. Ela recebe um array de inteiros representando a sequência de números e o tamanho dessa sequência “size”.

Se o tamanho da sequência for zero (size == 0), isso significa que não há números na sequência, então a função retorna 0, pois não há pontuação a ser obtida.

A função declara duas variáveis inteiras in e ex, que representam a pontuação incluindo e excluindo o número atualmente considerado, respectivamente. Inicialmente, ambas são definidas como 0.

Em seguida, a função entra em um loop que itera sobre cada número na sequência, de 0 até size - 1.

Para cada número na sequência, a função atualiza as variáveis in e ex. A ideia aqui é simular duas situações: incluir o número atual e excluir o número atual.

Dentro do loop, a função armazena temporariamente o valor atual de inc em uma variável temporária temp.

Em seguida, atualiza o valor de in para a soma do valor atual de ex e o número atual da sequência (vet[i]). Isso simula a situação de incluir o número atual na pontuação.Depois, atualiza o valor de ex para o máximo entre o valor anterior de in (armazenado em temp) e o valor anterior de ex. Isso simula a situação de excluir o número atual da pontuação.

Após o término do loop, a função retorna o máximo entre in e ex, representando a pontuação máxima que João pode obter.

Pseudocódigo:

```
Função Dinamica(vetor, tamanho):
  Se tamanho for igual a 0:
    Retorne 0  // Não há números para calcular a pontuação

  pontuação_incluindo = 0
  pontuação_excluindo = 0

  Para cada elemento i no vetor:
    temp = pontuação_incluindo
    pontuação_incluindo = pontuação_excluindo + vetor[i]
    pontuação_excluindo = Maior(temp, pontuação_excluindo)

  Retorne Maior(pontuação_incluindo, pontuação_excluindo)
```

2.5- Forma alternativa da resolução do problema

-Estrutura de Dados Tupla: Esta estrutura é usada para armazenar uma sequência de inteiros e sua pontuação correspondente.

-Função Auxiliar auxiliar: Esta função é uma função auxiliar recursiva que calcula a pontuação máxima possível para uma dada sequência. Ela verifica três casos principais

Se a sequência estiver vazia (representada pelo índice inicio sendo maior ou igual ao tamanho n), a função retorna 0.

Se houver apenas um elemento na sequência, a função retorna o valor desse elemento.

Se a pontuação correspondente à sequência no índice inicio já estiver sido calculada e armazenada na memoização, a função retorna essa pontuação.

Caso contrário, a função calcula a pontuação incluindo e excluindo o elemento atual e armazena a pontuação máxima na memoização antes de retorná-la.

-Função alternativo: Esta função é a função principal que o usuário chama para calcular a pontuação máxima de uma sequência. Ela aloca memória para a memoização, chama a função auxiliar para calcular a pontuação máxima e, em seguida, libera a memória alocada antes de retornar o resultado.

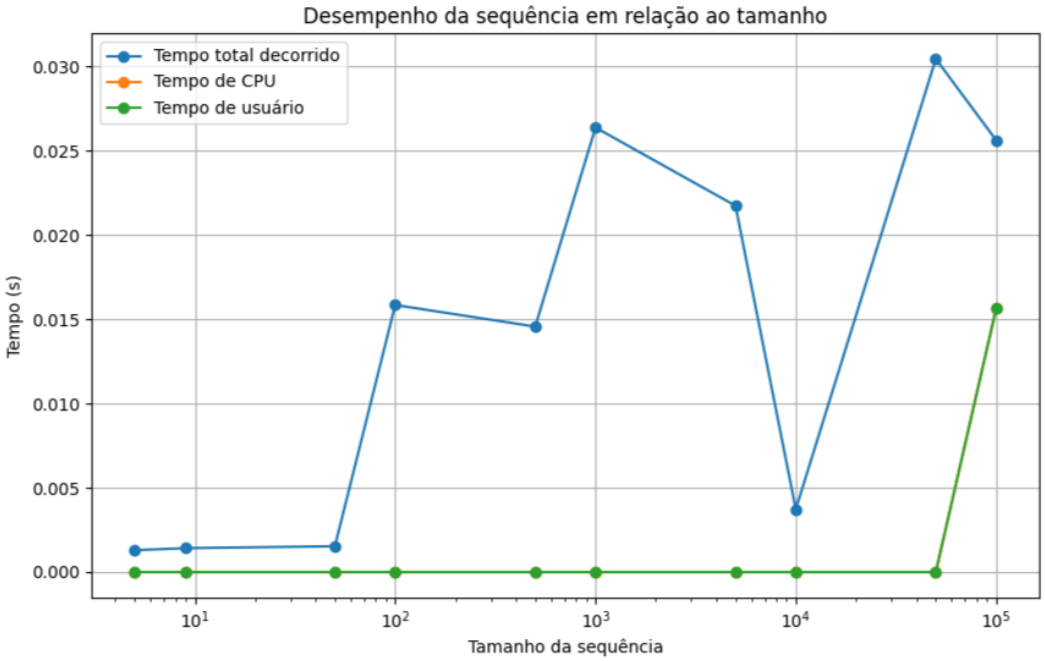
Pseudocódigo:

```
Estrutura Tupla:
  sequencia: vetor de inteiros
  pontuacao: inteiro

Função auxiliar calcularPontuacao(sequencia, inicio, tamanho, memo):
  Se inicio >= tamanho:
    Retorne 0
  Se inicio == tamanho - 1:
    Retorne sequencia[inicio]
  Se memo[inicio].sequencia != NULL:
    Retorne memo[inicio].pontuacao
  incluindo = sequencia[inicio] + calcularPontuacao(sequencia, inicio + 2,
tamanho, memo)
  excluindo = calcularPontuacao(sequencia, inicio + 1, tamanho, memo)
  memo[inicio].sequencia = sequencia
  memo[inicio].pontuacao = max(incluindo, excluindo)
  Retorne memo[inicio].pontuacao

Função calcularPontuacaoMaxima(seq, tamanho):
  memo = Vetor de Tupla de tamanho [tamanho]
  resultado = calcularPontuacao(seq, 0, tamanho, memo)
  Liberar memória alocada para memo
  Retorne resultado
```

3- Análise de tempo dos Algoritmo

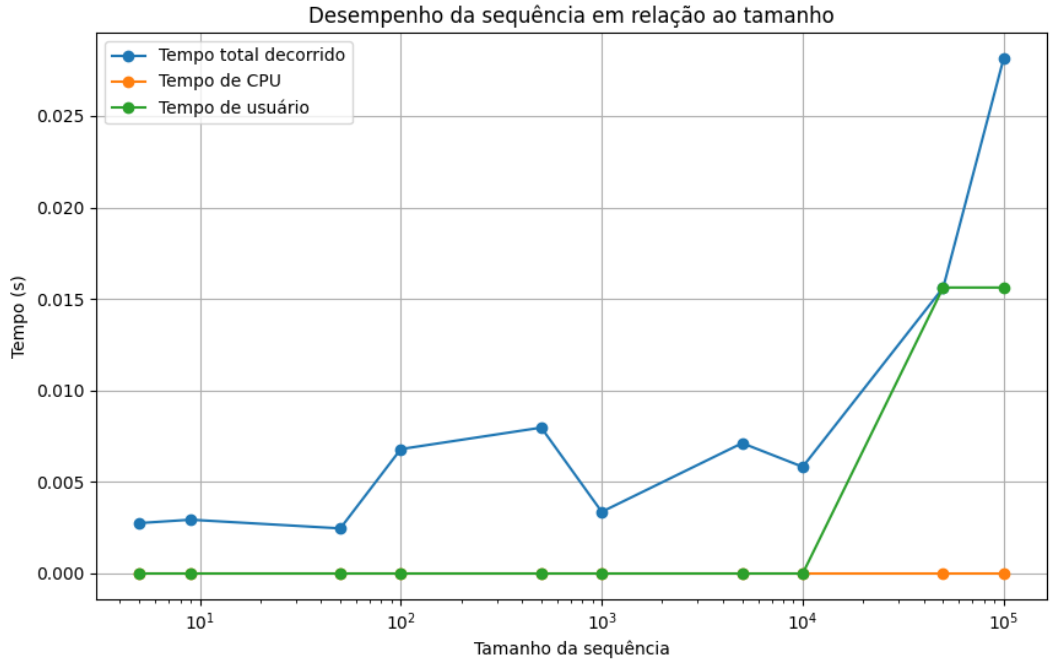
3.1- Tempos forma dinâmica da resolução do problema

Tamanho sequência:	Tempo total decorrido:	Tempo de CPU	Tempo de usuário:
5	0.001298	0.000000	0.000000
9	0.001418	0.000000	0.0000000
50	0.001531	0.000000	0.000000
100	0.015843	0.000000	0.000000
500	0.014562	0.000000	0.000000
1000	0. 026371	0.000000	0.000000
5000	0.021733	0.000000	0.000000
10000	0.003730	0.000000	0.000000
50000	0.030448	0.000000	0.000000
100000	0.025624	0.000000	0.015625



3.2- Tempos forma alternativa da resolução do problema

Tamanho sequência:	Tempo total decorrido:	Tempo de CPU	Tempo de usuário:
5	0.002756	0.000000	0.000000
9	0.002940	0.000000	0.000000
50	0.002465	0.000000	0.000000
100	0.006797	0.000000	0.000000
500	0.007972	0.000000	0.000000
1000	0.003370	0.000000	0.000000
5000	0.007124	0.000000	0.000000
10000	0.005828	0.000000	0.000000
50000	0.015618	0.000000	0.015625
100000	0.028146	0.000000	0.015625



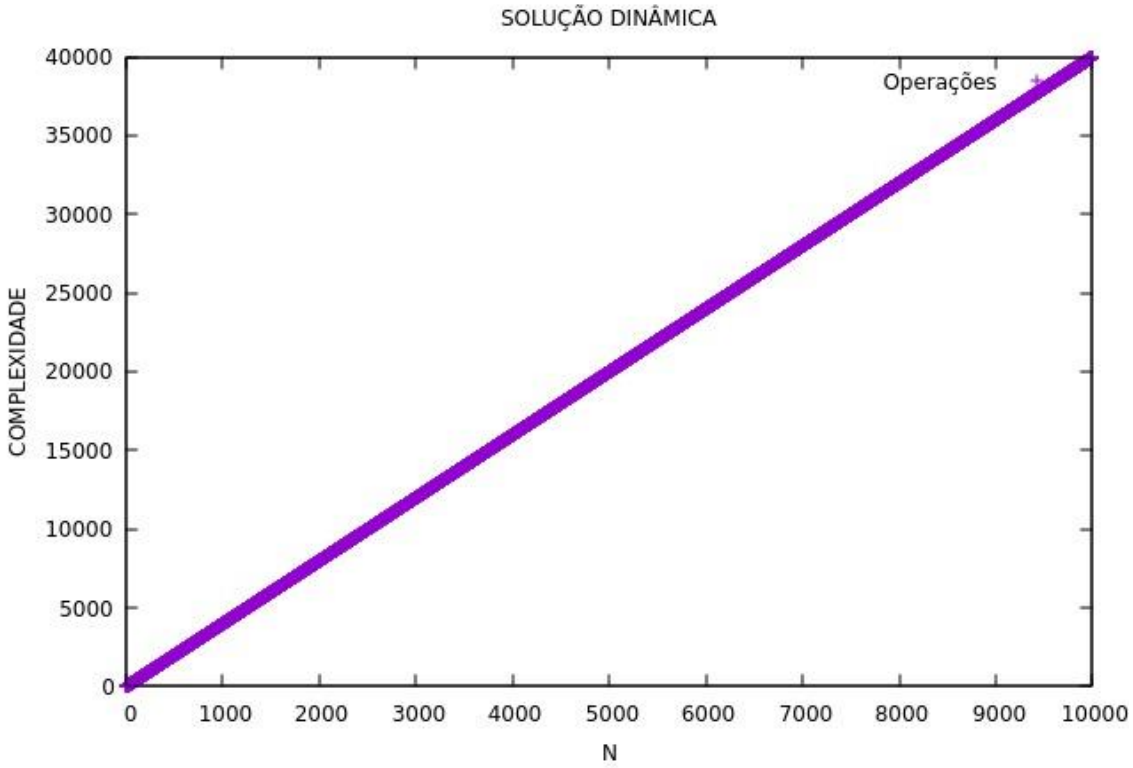
4- Análise de Complexidade dos Algoritmo

4.1- Complexidade forma dinâmica da resolução do problema

-A função biger(int a, int b) compara dois números e retorna o maior. Esta função tem uma complexidade de tempo constante, ou seja, O(1), porque realiza um número fixo de operações.

-A verificação inicial if (size == 0) também tem uma complexidade de tempo constante, O(1).A inicialização das variáveis in e ex é outra operação de tempo

constante, O(1). O loop for (int i = 0; i < size; i++) itera sobre cada elemento do vetor. Dentro do loop, todas as operações (atribuição de variáveis e chamadas à função biger) são operações de tempo constante. Portanto, a complexidade de tempo do loop é proporcional ao tamanho do vetor, ou seja, O(n), onde n é o tamanho do vetor. Portanto, a complexidade total do tempo do algoritmo é dominada pelo loop for, resultando em uma complexidade de tempo linear, O(n).



4.2- Complexidade forma alternativa da resolução do problema

Função auxiliar:A função auxiliar é recursiva e é chamada para calcular a pontuação máxima. Ela recebe uma sequência de números, um índice de início, o tamanho da sequência e uma memória para memoização.

Em cada chamada recursiva, a função realiza operações constantes, como verificação de condições e atribuições de valores.

As operações principais da função são as chamadas recursivas, que percorrem a sequência de números.

Portanto, a complexidade temporal da função auxiliar depende do número de chamadas recursivas e do tamanho da sequência.

Como a função faz uso de memoização, cada elemento da sequência é processado apenas uma vez, resultando em uma complexidade de tempo de O(n), onde n é o tamanho da sequência.

Além disso, como a função utiliza memoização, a complexidade de espaço é O(n), onde n é o tamanho da sequência.

Função alternativo:

-A função alternativo é responsável por alocar memória para a memoização, chamar a função auxiliar e liberar a memória alocada.

As operações realizadas na função alternativo são todas operações de tempo constante, como alocação de memória, chamada de função e liberação de memória.

Portanto, a complexidade temporal da função alternativo é dominada pela complexidade temporal da função auxiliar, que é O(n), onde n é o tamanho da sequência.

A complexidade de espaço da função alternativo é O(n), onde n é o tamanho da sequência, devido à alocação de memória para a memoização.

Em resumo, a complexidade é O(n), onde n é o tamanho da sequência. Isso significa que o tempo de execução e a quantidade de memória necessária aumentam linearmente com o tamanho da sequência.

5- Conclusão

Para resolução deste problema foi pensado primeiramente uma resolução por foça bruta, checando todas as possibilidades existentes, porem foi exageradamente ineficiente, assim tendo de recorrer a outras formas. É possível notar que nas duas formas de resolução a complexidade em “big O” é igual porém o tempo da forma alternativa que foi desenvolvida de forma recursiva é basicamente o dobra da forma dinâmica que não é recursiva, assim provando um dos problemas de resoluções recursivas, demandado mais tempo.

6- Bibliografias:

-Programação Dinâmica - Algoritmo de Kadane. 2012. Disponível em https://marathoncode.blogspot.com/2012/09/algoritmo-de-kadane.html?m=1

-Ian Parberry, Problems on Algorithms, Prentice Hall, 1995.