UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL REI

ALGORITMOS E ESTRUTURAS DE DADOS III

Aluno: Lucas Lagôa Nogueira
Curso: Ciência da Computação
Professor: Leonardo C. Rocha

São João del Rei, março de 2016

Sumário

1 – Introdução

O ser humano sempre procurou meios para facilitar sua vida, principalmente no ramo matemático e computacional, como podemos observar no trabalho prático que um simples jogo de cartas cujo objetivo é você prever o número máximo de pontos que seu adversário conseguirá se jogar contra você pode ser um objeto de estudo de paradigmas de programação e estruturas de dados.

Aprofundando no assunto de paradigmas de programação, que por definição é o conceito que pressupõe a forma que programador tem sobre a programação e execução de um programa. Resumidamente, como o programador vai estruturar e executar o problema. E especificamente nesse trabalho prático, será abordado dois exemplos dela, a heurística e a programação dinâmica.

2 – Soluções e propostas de implementação.

Para a resolução do problema proposto pelo trabalho prático, foram utilizados dois programas que visavam realizar que foi proposto. Um deles se baseava em uma heurística e o outro foi baseado em princípios de programação dinâmica.

2.1 – Heurística.

Por definição a palavra heurística para a computação significa que um algoritmo terá boas soluções na maioria das vezes, mas sem provas que ele será rápido para todas situações. Baseado nesse princípio o desenvolvimento da heurística para resolver o problema do super trunfo era basicamente uma lógica que fazia comparações de 4 em 4 cartas, garantindo que sempre seria selecionado a melhor opção para o adversário, cumprindo corretamente as exigências demandadas.

```
leia(vetor);

x = vetor(primeira posição) + vetor (penúltima posição)
y = vetor(última posição) + vetor (segunda posição)

se ( x > y ) então
    soma = vetor (primeira posição)
    acrescenta posição

    senão
        soma = vetor (última posição)
        diminui posição
    fim-se

fim-se
```

Figura 1: Pseudocódigo para a implementação por heurística

3	9	1	2
I			J

Figura 2: Representação do vetor junto com os índices para realizar as operações

Como pode-se observar na figura acima, um vetor era percorrido procurando sempre a maior possibilidade para achar a maior soma requerida, porém esse algoritmo apresentou alguns problemas durante a realização de testes com muitas cartas e quando a soma tinha valores na casa dos bilhões/trilhões. Mas ainda assim,

o resultado era aproximado ao esperado e o tempo para realizar as operações era bastante pequeno devido as operações não demandarem muitas operações, pois tudo era realizado um vetor.

2.1 – Método por programação dinâmica.

Visando resolver esse problema, procurei utilizar princípios de programação dinâmica que é um paradigma de programação e pela sua definição pode ser utilizado para a resolução de problemas computacionais nos quais a solução ótima pode ser computada a partir da solução ótima previamente calculada e memorizada. Em uma pesquisa para clarear as ideias, foi encontrado um vídeo que explicava um método de resolução de um jogo de cartas utilizando o princípio da programação dinâmica. O vídeo explicava como seria a realização do método de maximizar valores retirando de ambos os lados, de acordo com a figura demonstrada acima, e assim baseando na técnica explicada pelo autor do vídeo, foi realizada a estruturação e desenvolvimento do problema visando realizar o problema proposto. Esse método se baseia no preenchimento da matriz principal com os valores e numa segunda matriz com 0's, e com as iterações e somas, os valores de somas foram guardados em uma outra matriz, e o valor final ficaria na primeira linha e na última coluna, de acordo com o preenchimento na matriz superior.

```
se ( diagonal principal )
    primeiramatriz = vetor de cartas
    segundamatriz = 0
fim se
    senao
        se ( vetor de cartas (i) + segunda matriz (i+1)(j) < vetor de cartas (j) + segunda matriz (i) (j-1) )
            primeira matriz(i)(j) = segunda matriz (i)(j-1) + vetor de cartas (j)
            segunda matriz(i)(j) = primeira matriz (i)(j-1)
        fim se
        senao
            primeira matriz = segunda matriz (i+1) (j) + vetor de cartas (i)
            segunda matriz = primeira matriz (i+1)(j)
        fim senao
    fim senao
i++
j++
fim
```

Figura 3: Pseudocódigo para a implementação por programação dinâmica.

[3,0]			
	[9,0]		
		[1,0]	
			[2,0]

[3,0]	[9,3]		
	[9,0]	[9,1]	
		[1,0]	[2,1]
			[2,0]

[3,0]	[9,3]	[4,9]	
	[9,0]	[9,1]	[10,2]
		[1,0]	[2,1]
			[2,0]

[3,0]	[9,3]	[4,9]	**[11,4]**
	[9,0]	[9,1]	[10,2]
		[1,0]	[2,1]
			[2,0]

Figura 4: Demonstração de como é preenchido as matrizes.

O método consiste em pegar a matriz que salva os 0's e somar com um valor do vetor, e onde achar a maior soma entre as posições, ela salvará na matriz resultado e a outra matriz receberá o índice da mesma posição da maior soma. Com a utilização dos métodos de programação dinâmica, os problemas encontrados anteriormente de valores aproximados, foram resolvidos, porém nesse método foi necessário a utilização de matrizes, o que seria um problema caso o valor da entrada tivesse valores muito grandes, assim alguns segundos seriam gastos para achar o resultado esperado, porém o resultado é mais correto do que o outro método aplicado.

3 – Organização do programa

Ambos programas foram implementados em 5 módulos. Sobre o algoritmo das heurísticas, ficou dividido em main.c, args.c, args.h, heuristica.c, heuristica.h e para o algoritmo da programação dinâmica, ficou dividido em main.c, args.c, args.h, heuristica.c e heuristica.h.

3.1 – main.c

No main.c de ambos algoritmos, foram computados basicamente as funções de tempo e os métodos de leitura das cartas de um arquivo “entrada.txt”.

3.2 – args.c

No args.c de ambos algoritmos, foi computado a função de leitura de parâmetro argc e argv.

3.3 – args.h

No args.h de ambos algoritmos, foram computados a struct do arg e o título da função de leitura de parâmetro.

3.4 – heuristica.c e tp1.c

Em ambos módulos foram computadas as funções utilizadas para realizar as operações, como a função heuristica e a função dinamico.

3.5 – heuristica.h e tp1.h

Nesses módulos, foram computados os títulos das funções que estão implementadas nos módulos de mesmo nome porém .c

E ambos possuem um MakeFile para compilar os programas.

4 – Análise de complexidade

Nessa seção será debatido uma análise da complexidade das funções utilizadas em ambos algoritmos e também a utilização e importância da função para realização do programa.

4.1 – Função para realizar os cálculos da heurística.

long long int heuristica

Essa função realiza um while de acordo com o tamanho do vetor, onde ele começa da última posição até a primeira que no caso sua complexidade é O(n), e mais internamente dele tem somente atribuições, if e elses que são O(1). Logo, a complexidade total da função é O(O(1) * O(n)), logo é O(n).

4. 2 – Função para alocar as matrizes da programação dinâmica.

long long int **alocarmatriz

Essa função realiza um for que depende do número de cartas para alocar as matrizes que serão utilizadas e como ele depende do número de cartas, a complexidade dessa função é O(n).

4.3 – Função para realizar as operações da programação dinâmica.

long long int dinamico

Essa função que realiza as operações da programação dinâmica inicialmente tem um for que depende do número total de cartas e um while aninhado que também roda dependendo do número de cartas para realizar as operações, logo a complexidade é $O(n^2)$. Porém essa função chama duas vezes a função de alocar as matrizes que é O(n) mas externamente ao for e while, logo a complexidade total é $\max(O(n^2),O(n),O(n)) = O(n^2)$.

4.4 – main.c de ambos programas.

4.4.1 – main.c do algoritmo de heurística.

Analisando a main.c desse algoritmo, destaca-se que tem um for para preencher o vetor com os valores de cartas retiradas do arquivo, cujo complexidade é O(n). Também podemos destacar que a função heuristica cuja complexidade é O(n) é chamada chama dentro da main.c e o restante das operações são de atribuição, cujo complexidade é O(1), logo a complexidade total da main.c é $\max(O(n),O(n),O(1))$.

4.4.2 – main.c do algoritmo de programação dinâmica.

Analisando a main.c desse algoritmo, destaca-se que também possui um for para preencher o vetor com os valores de cartas retiradas do arquivo, que tem a complexidade O(n). Porém a função dinamico cuja complexidade é $O(n^2)$ é chamada também na main.c e o restante de operações são atribuições, logo a complexidade total é $\max(O(n),O(n^2),O(1))$.

5 – Testes e análises de resultados

Para esse problema, foram utilizados dois métodos de resolução, como foi dito anteriormente, um de heurística e um de programação dinâmica.

5.1 – Método de resolução por heurística.

Para a resolução por heurística, houve um melhor desempenho em relação ao tempo porém os resultados nesse método não eram tão precisos quanto ao outro método. Segue abaixo, algumas figuras que demonstram os testes realizados por esse algoritmo. Discutindo sobre as vantagens desse algoritmo utilizado, podemos destacar a rapidez para a realizar as operações devido a pequena complexidade de suas operações envolvidas e também a fácil implementação do mesmo, pois ela segue os passos da heurística que são rápidos e não necessitam de muitas operações. E em relações as desvantagens observadas por esse algoritmo, destaca-se que para operações com muitas casas a serem utilizadas, na margem de bilhões ou trilhões, ele não é eficiente pois há uma diferença nos resultados encontrados, assim diminuindo sua soma final.

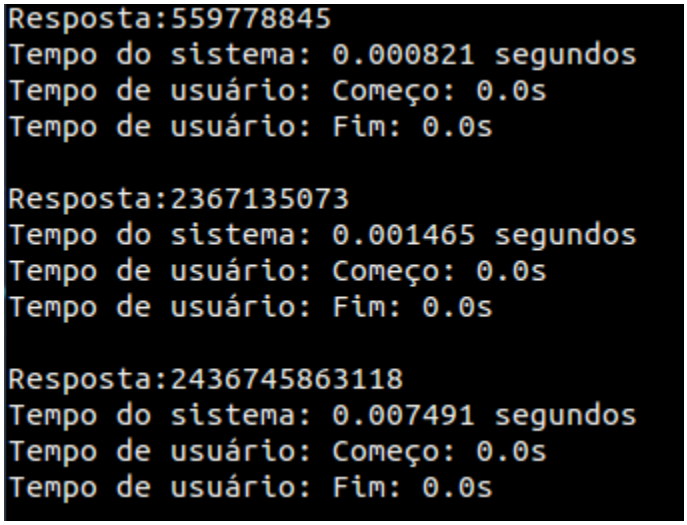

Figura 5: Testes realizados.

5.2 – Método de resolução por programação dinâmica.

Para a resolução por programação dinâmica, houve uma melhor precisão nos resultados, chegando ao resultado exato ao esperado. Segue abaixo, alguns testes realizados por esse algoritmo. Em relação as vantagens desse algoritmo, destaca-se a exatidão dos resultados esperados tanto para testes com números menores e também para números maiores. E por desvantagens, seria o tempo de execução e a memória para resultados com muitas casas, devido a alocação da matriz e suas operações realizadas, pois utiliza-se duas matrizes para obter o resultado final.

```
Resposta:559778845
Tempo do sistema: 0.000339 segundos
Tempo de usuário: Começo: 0.0s
Tempo de usuário: Fim: 0.0s

Resposta:2725177752
Tempo do sistema: 0.000533 segundos
Tempo de usuário: Começo: 0.0s
Tempo de usuário: Fim: 0.0s

Resposta:2515878770634
Tempo do sistema: 10.883712 segundos
Tempo de usuário: Começo: 0.0s
Tempo de usuário: Fim: 0.788000s
```

Figura 6: Testes realizados.

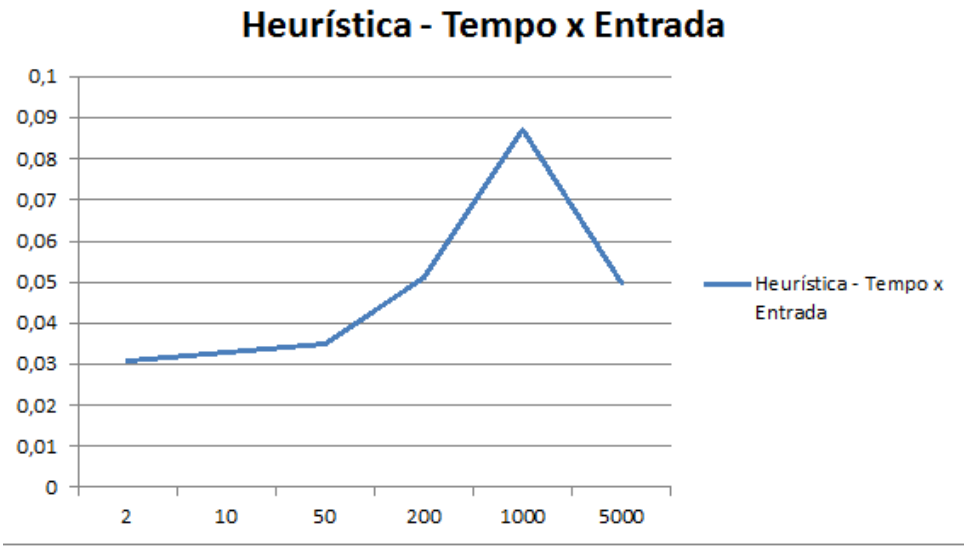

Figura 7: Gráfico para o algoritmo de heurísticas

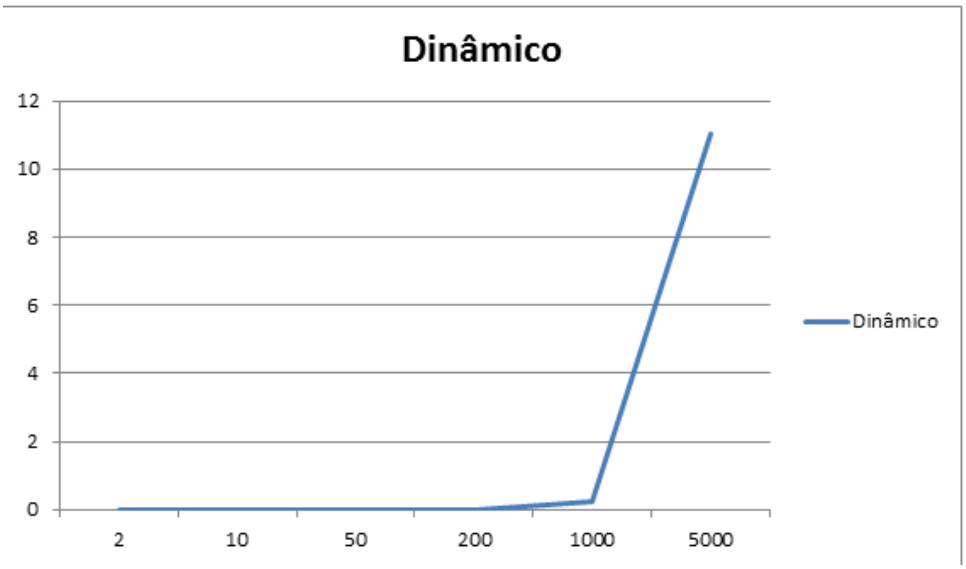

Figura 8: Gráfico para o algoritmo de programação dinâmica

Explicando um pouco os gráficos, para o primeiro teste podemos observar que com o aumento do número de cartas para ambos jogadores não faz muita diferença e o tempo não ultrapassa 0,1s para todos os testes. E observa-se uma variação no final da entrada, porém a mesma é justificada que no mais importa as operações e não o número de cartas. Mas já para o algoritmo de programação dinâmica observa-se que com o aumento do número de cartas (1000-5000) já observa-se uma diferença bem grande na questão de tempo de execução, beirando os 12 segundos.

Obs: testes realizados em um computador com processador Intel Celeron CPU 847 1.10GHz com 4 gigas de memória RAM.

Obs.1: testes realizados com uma mesma entrada, cuja entrada foi fornecida pelo monitor Sávyo Toledo.

6 – Considerações Finais

A implementação do trabalho ocorreu em partes sem muitos problemas na parte de implementação, mas alguns tópicos merecem atenção a ser debatidos. O algoritmo de heurísticas no geral é mais rápido que o de programação dinâmica, mas seus resultados não são tão precisos. Por isso o algoritmo que obteve melhor resultado foi o de programação dinâmica, que por um lado ele só tem problemas de tempo quando o número de cartas é muito grande.

7 – Referências bibliográficas

- Tushar Roy. ***Optimal Strategy Game Pick from Ends of array Dynamic Programming.*** Disponível em: <https://www.youtube.com/watch?v=WxpIHvsu1RI>. Acesso em: 12 mar. 2016.

- ROCHA, Leonardo C. D. ***Slides da unidade curricular de Algoritmos e Estruturas de Dados III***, 2016.