Documentação Trabalho Prático 02 de Algorítimos e Estruturas de Dados III

Paulo Cirino Ribeiro Neto
Engenharia de Sistemas
UFMG

17 de novembro de 2015

1 Introdução

Segundo a wikipedia "Um paradigma de programação fornece e determina a visão que o programador possui sobre a estruturação e execução do programa.". Esse trabalho têm o intuito exatamente de fomentar e exercitar as diferentes ópticas em que um programador pode planejar e formalizar a resolução de um problema algorítmico.

A questão desse trabalho é resolver o mesmo problema utilizando ferramentas e linhas de pensamento diferentes com o intuito de obter uma solução ideal ou próxima dela. Nesse caso em específico serão implementadas 3 soluções utilizando os paradigmas de **Força Bruta**,**Dinâmico**, e **Guloso**.

Iremos mostrar nesse trabalho as diferentes abordagens e as vantagens e desvantagens de cada solução.

2 O Problema

O problema a ser selecionado é dado um caminho que sai de um ponto inicial específico e chega a um ponto final diferente passando obrigatoriamente por **N** pontos e parando em **M** quaisquer um deles, de forma a minimizar a maior distância entre 2 pontos conectados.

3 A Solução

3.1 Força Bruta

O paradigma de programação de Força Bruta consiste em uma técnica de solução de problemas trivial, porém muito geral que trabalho no sentido de enumerar todos os possíveis candidatos da solução e checar cada candidato para saber se ele satisfaz a resolução do problema e se ele é uma solução ótima.

No caso desse problema em específico para gerar todas as possíveis combinações utilizamos 2 conceitos :

- Arranjo Básico
- Andar com Arranjo

O arranjo básico é a primeira combinação possível, para esse problema isso significa escolher **M** valores seguidos, que representam os planetas escolhidos para conquistar. No caso se $\mathbf{M} = 4$, o caso básico seria $\{1, 2, 3, 4\}$ que representa escolher os planetas **{ P1, P2, P3, P4}**.

E o conceito de andar que é alterar a escolha para próxima candidata a solução possível, no caso sitado seria **{ P1, P2, P3, P5}** . E no mesmo exemplo se o número de planetas for $\mathbf{N} = 5$ teríamos que se o mesmo andasse novamente seria **{ P1, P2, P4, P5}**, e isso continuaria até que se fosse atingida o ultimo arranjo possível que é **{ P2, P3, P4, P5}**.

Fica obvio que o número de possibilidades é um arranjo de **N** planetas e **M** escolhas, de forma matemática isso é :

$$\frac{\prod_{i=N-M}^{N} i}{M}$$

Além desses dois conceitos utilizamos também uma função que calcula o maior sub-caminho de um arranjo de planetas.

De forma resumida esse algorítimo funciona de forma a percorrer todas as distâncias de um sub-caminho desde o seu planeta inicial até o seu planeta final e soma-las de forma a comparar essa soma com a maior distância já previamente obtida.

De forma geral então podemos dizer que o força bruta funciona de modo a gerar todas as possibilidades com a função anda e calcular o maior sub-caminho do arranjo fornecido pela função calcMaiorDistCaminho e depois escolher o menor maior sub-caminho.

3.2 Pseudo-Código

O programa todo funciona como o pseudo-código abaixo:

```
Input: distVet, N, M
int Caso = [1:M];
int menorMaiorDist = ∞ ;
while Caso[1] ¡ M-N do
    int newDist = calcMaiorDistCaminho(distVet,Caso,N);
    if newDist < menorMaiorDist then
        menorMaiorDist = newDist;
    end
    anda(Caso);
end
Output: menorMaiorDist
```

Algorithm 1: Pseudo-Código Força Bruta

A função já mencionada que calcula o maior sub-caminho de um determinado arranjo de planetas segue abaixo:

```
Input: distVet[1:N+1],Caso[1:M], N
int maiorDist = −∞ ;
int i = 0 ;
int planeta = 0;
while i ¡= M do
    int distSubCaminho = 0;
    while planeta <= Caso[i] do
        distSubCaminho += distVet[planeta];
    end
    if distSubCaminho >= maiorDist then
        maiorDist = distSubCaminho;
    end
    i++;
end
Output: maiorDist
```

Algorithm 2: Pseudo-Código calcMaiorDistCaminho()

E a função que anda funciona como o pseudo-código abaixo:

```
Input: Caso[1:M], N
int flag=1;
int cont=0;
int maxPos = N - 1;
int pos = M - 1;
while Caso[pos] < maxPos do
    maxPos−−;
    pos−−;
end
Caso[pos - 1] ++;
while pos < M do
    pos++;
    Caso[pos] = Caso[pos-1];
end
Output: Caso[1:M]
```

Algorithm 3: Pseudo-Código anda()

4 Guloso

A solução que utiliza a estratégia gulosa para esse problema funciona de forma a **minimizar o erro em relação a distância ideal**. A distância ideal é um

conceito criado que indica a distância que todos os sub-caminhos deveriam ter para o maior sub-caminho fosse mínimo, essa distância é obviamente a distância **D** total do caminho todo dividido pelos **M+1** sub-caminhos, ou seja **Id** = $D/(M+1)$.

Nem sempre será possível criar um sub-caminho com a exata distância ideal, assim devemos criar uma decisão de quanto queremos errar, para isso sempre mantemos 2 distâncias guardadas uma atual e a outra é a distância imediatamente anterior para que possamos escolher o sub-caminho que incorre menor erro quando comparamos com a distância ideal. Além do mais devemos lembrar de sempre de manter a distância ideal atualizada diminuindo a distância percorrida da distância total e fazendo **M = M-1**, de forma para que a nova distância ideal é sempre real ao seu novo caminho.

De forma resumida o algorítimo funciona de forma a encontrar um sub-vetor que inicia na primeira posição do vetor de distâncias que é o mais próximo da distância ideal possível, essa distância percorrida é removida do vetor inicial e temos então um novo problema que é o vetor de onde o sub-caminho passado parou até o final. Assim resolvemos esse novo problema da mesma forma só que atualizando a distância total e **M** e recalculando a distância ideal.

A implementação da estratégia gulosa escolhida para a solução desse problema mesmo funcionando para todos os casos testes fornecidos no moodle **não é ótima**.

4.1 Pseudo - Códigos

O método funciona como uma única função, que segue abaixo:

Algorithm 4: Pseudo-Código Guloso()

5 Programação Dinâmica

5.1 Pseudo - Códigos

Temos que o algorítimo de programação dinâmica assim como o de força bruta funciona com 3 partes, andar, calcular distância, e algorítimo principal.

No caso, a função de andar é praticamente igual a do método de força bruta, a única diferença é que essa retorna uma flag com a posição a partir de qual o vetor de planetas escolhidos foi alterado. Assim não irei mostrar esse pseudo-código aqui

Segue então o pseudo-código do metodo principal:

Algorithm 5: Pseudo-Código Guloso()

Segue o método que calcula o vetor de sub-caminho máximo até a última posição que ficou intacta:

Algorithm 6: Pseudo-Código Guloso()

6 Complexidade

6.1 Complexidade Temporal

A complexidade temporal é o problema mais serio das soluções de programação dinâmica e de força bruta, temos que para ambos a solução foi fatorial.

O problema maior para mim foi evitar de gerar todas as possíveis soluções, que são:

$$\frac{N!}{M!(N-M)!} = \binom{N}{M} \tag{1}$$

A diferença básica é a forma com que os métodos calculam a distância máxima de cada caminho, no caso do **PD** a complexidade é no pior caso **O(N)** e no caso médio $log(N)$ já no força bruta a mesma função é feita com complexidade $\mathbf{O(}N^2\mathbf{)}$.

Já o guloso têm complexidade de **O(N)**.

Segue abaixo as tabelas com complexidades das funções e métodos:

CaminhaFb	maxDistCalcFb	CaminhaDinamico	maxDistCalcDinamico
O(M)	O(M*N)	O(M)	O(N)

Força Bruta	Dinâmico	Guloso
$O(M * N * (\frac{N!}{M!(N-M)!}))$	$O(N * (\frac{N!}{M!(N-M)!}))$	O(N)

Temos que considerar também o fato de que mesmo a diferença no pior caso não ser muito diferente entre o algorítimo força bruta e o guloso, a verdade é que a complexidade θ é muito diferente, só não será demonstrada aqui por conta das dificuldades matemáticas.

6.2 Complexidade Espacial Teórica

Para nenhum dessas soluções a complexidade espacial é um problema, temos que para todos os casos ela é linear.

No algorítimo de força bruta, temos que é utilizado 2 vetores, um de tamanho **M** e outro de tamanho **M+1** além é claro de algumas variáveis atômicas.

O PD é utiliza as mesma variáveis que o força bruta só que com a adição de 2 vetores de tamanho **N+1**.

Já no guloso é utilizado apenas um vetor, que é o vetor de distâncias com tamanho **N+1** e algumas variáveis atômicas.

Temos então o seguinte gráfico:

	Força Bruta	Programação Dinâmica	Guloso
1	O(N+M)	O(3N+M)	O(N)

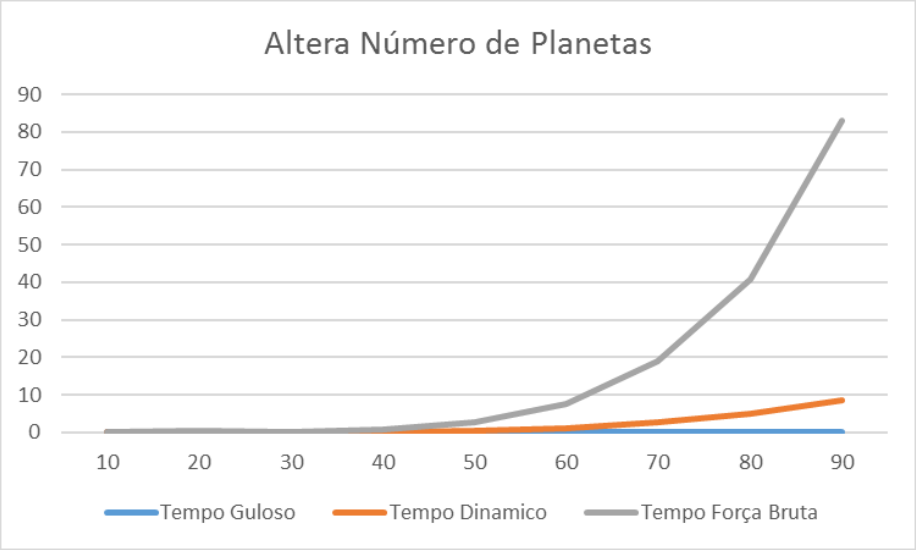

Figura 1: Alterando apenas número total de planetas.

7 Testes Práticos

Os testes forma conduzidos de forma a c compilar o tempo médio de 3 testes iguais, cada qual com 3 instâncias a serem testadas.

Quando observamos a figura 1 fica obvio o crescimento exponencial da complexidade do força bruta. podemos observar que a cada mudança no número total de planetas **N** temos que a inclinação da curva é inclinada em alguns graus.

Além disso fica um tanto quando obvio a diferença entre a complexidade dos métodos, percebemos que o força bruta é muitas vezes mais lento que os outros 2, o dinâmico é muito mais rápido que o força bruta, mas pela necessidade de analisar todas as possíveis combinações o mesmo é muito mais lento que o algorítimo guloso.

Além do mais percebemos que a complexidade do algorítimo de PD aumenta de forma muito mais suave com o número de planetas quando comparamos ao FB, enquanto ele aumenta o grau da inclinação de forma constante, o FB aumente o grau desse crescimento a cada incremento do total de planetas.

Agora quando observando a figura 2 podemos perceber uma característica interessante dos métodos dinâmico e força bruta, na qual eles se comportam de forma que a complexidade do problema aumenta de forma exponencial quando alteramos o número de planetas a serem escolhido até a metade do total de planetas, e depois essa complexidade decai na mesma proporção.

O motivo das complexidades descritas na análise teórica, onde os algorítimos

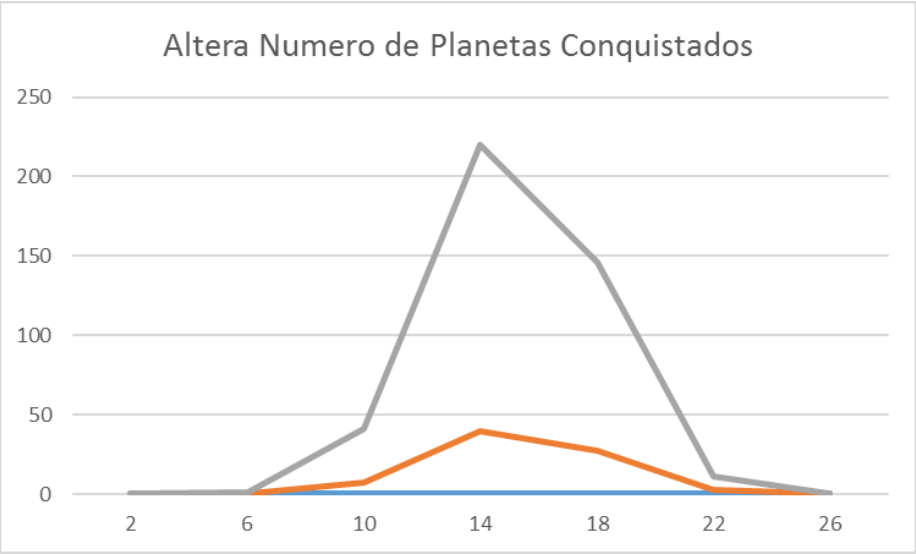

Figura 2: Alterando apenas Número de Planetas Conquistados.

não foram colocados exatamente como exponenciais, mas sim na forma de um equação que mostra que existe essa curva.

Como já foi descrito o pior caso para os algorítimos de Fb e Pd é quando o número de planetas conquistado é a metade do número total de planetas.

Assim fiz também uma compilação que mostra como a complexidade do algorítimo altera de acordo com o pior caso:

Percebemos pelo gráfico 3 e também pelos outros que existe uma discrepância muito grande entre o tempo de execução dos métodos.

O força bruta é absurdamente lento quando comparamos com programação dinâmica, mesmo para esses casos razoavelmente pequenos, mas isso não significa que o programação dinâmica teve um bom desempenho. Na realidade quando observamos realmente o Pd, ele é extremamente lento e têm crescimento exponencial da complexidade, apenas o guloso cresce de forma linear com a entrada.

Segue abaixo a tabela 1 com os dados que geraram os gráficos para melhor entendimento:

8 Conclusão

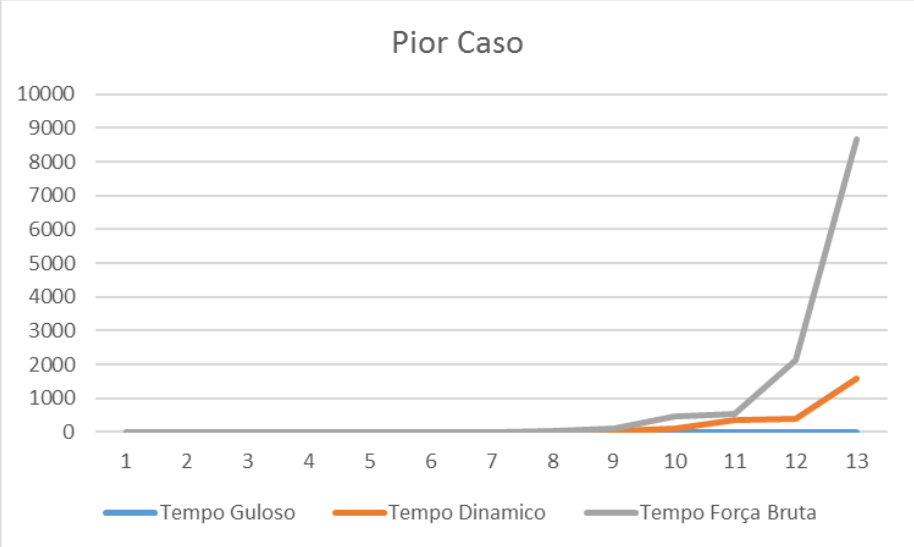

Figura 3: Pior Caso.

Num Planetas	NumEscolhas	Tempo Guloso	Tempo Dinamico	Tempo Força Bruta
4	2	0,172	0,156	0,151
8	4	0,172	0,156	0,17
12	6	0,203	0,156	0,16
16	8	0,172	0,156	0,196
20	10	0,213	0,281	0,57
22	11	0,222	0,563	1,698
24	12	0,156	0,656	6,126
26	13	0,156	6,101	24,407
28	14	0,141	22,439	120,217
30	15	0,156	103,276	473,178
32	16	0,141	372,3	524,319
34	17	0,156	402,438	2117,378
36	18	0,188	1593,838	8655,461

Tabela 1: Tabela de Dados