TP2

Wilson Moreira

November 16, 2015

1 Introdução

O problema a ser resolvido é ilustrado com o seguinte cenário: Após a vitória da Aliança Rebelde, o lado negro da força tenta planejar uma nova revolução e reconquistar a galáxia. Só que em um momento de reestruturação o lado negro da força precisa economizar recursos ao máximo. Faz parte do plano estratégico do lado negro da força reconquistar determinados planetas primeiro. Através do uso do mapa da galáxia é possível saber quais são os planetas que estão na rota entre o planeta atual e o que será conquistado. Para tais investidas uma nova nave foi construída, a Estrela da Morte III, mas ela ainda está em fase de testes, devendo, portanto, se locomover a menor distância possível entre cada planeta, onde haverá a possibilidade de realizar reparos na nave.

Com o cenário acima definido, foi definida a tarefa de criar três algoritmos capazes de escolher entre N planetas no percurso um número K de planetas a serem reconquistados a fim de minimizar a maior sub-distância do percurso. Dos três algoritmos, cada um deve ser implementado com um paradigma diferente, um através de força bruta, um através de programação dinâmica e o outro através de algoritmo guloso.

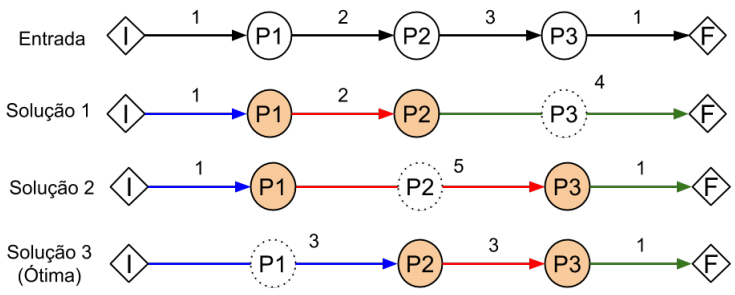


Figure 1: Exemplo de instância a ser resolvida, o objetivo é encontrar a solução ótima. Imagem retirada das especificações do trabalho.

2 Solução do problema

2.1 Força Bruta

A implementação do algoritmo por força bruta se deu através do uso de uma árvore de possibilidades.

A estrutura de dados utilizada foi essa, onde *conquistado* é uma variável que sinaliza se o planeta foi ou não conquistado, *nivel* que é relacionado a altura do nó, $numNaoConquistas$ que diz respeito ao número de planetas não conquistados no atual percurso, $trechoAtual$ é uma variável que guarda o trecho acumulado até o atual nó, é util para popular a subárvore desse nó, $maiorTrecho$ é o maior trecho presente no percurso até o nó.

Para calcular os itens dessa árvore de possibilidades, foram feitas duas funções recursivas. Uma delas aloca a árvore de acordo com os a entrada, contendo uma poda que elimina caminhos que não satisfazem os parâmetros da entrada no que diz respeito ao número de conquistas. A outra função calcula sub-trechos em todos os nós, guardando o maior subtrecho até cada nó e verificando qual o maior e que alcançou o fim até o momento. Após a recursão, o maior valor é impresso na tela.

No fim, os resultados que possuem as características de solução(nível e planetas não coquistados) correspondentes são comparados a fim de descobrir qual é a solução ótima.

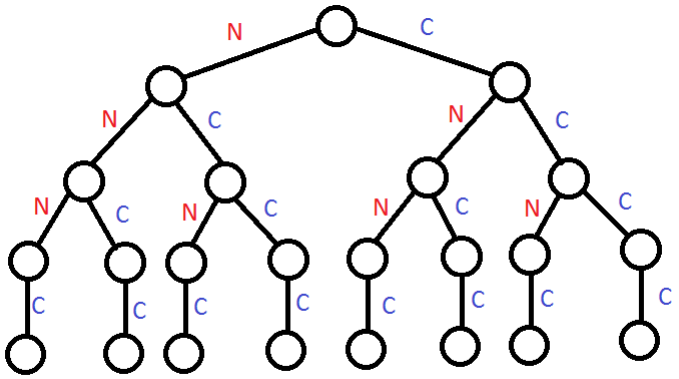


Figure 2: Exemplo genérico de árvore de possibilidades, o último planeta é sempre visitado, portando não há mais de uma opção no fim.

2.2 Algoritmo Guloso

A implementação do algoritmo guloso não exigiu nenhuma estrutura de dados adicionais além do vetor contendo as distãncias entre os planetas. A estratégia do problema foi pegar a menor soma entre as distâncias contidas no vetor e unir essas duas distãncias, ao achar a menor de todo o vetor(e que

aparece primeiro), o próximo passo é encontrar a menor soma(que aparece por último) entre os planetas seguintes ao menor de todo o vetor. O segundo passo só não é executado em três ocasiões:

- Quando não há mais planetas para conquistar nas posições seguintes.
- Quando o número total de conquistas de planetas foi atingido.
- Quando só resta mais um planeta a ser conquistado.

A forma de atuação do algoritmo pode ser melhor observada através do pseudocódigo abaixo:

```
Algoritmo Guloso
-----------------------------------------------------------------------
Início
    numIteracoes = planetasNaRota - NumConquistas
    i=0

    Enquanto(1)
       Se i = numIteracoes
        Para
       Fim Se
       aux = identaVetor (vet distPlanetas, planetasNaRota, 0, final)

       Se i = numIteracoes
        Para
       Fim Se

       Se aux <= planetasNaRota-i e i < numIteracoes-1
        aux = identaVetor(vet distPlanetas, planetasNaRota, aux, final)
       Fim Se
    Fim enquanto

    Para i de 0 até planetasNaRota - numIteracoes faça
        Escolha maior valor
    Fim Para

    Imprima maior valor
Fim
-----------------------------------------------------------------------
```

Na função acima há a chamada da função *identaVetor* várias vezes, essa função basicamente percorre o vetor e junta as duas posições que possuem a menor soma. O motivo de chamar a função de formas diferentes se dá pela característica de que a função é chamada uma para o vetor inteiro e depois para a parcela depois daquela que foi escolhida. O pseudocódigo abaixo explica tal função, ajudando, inclusive a entender que a variável aux presente no código citado acima representa o momento do qual a segunda execução deve começar:

```
Função identaVetor
-----------------------------------------------------------------------
Parâmetros recebidos:
vet distPlanetas, planetasNaRota, inicial, final
-----------------------------------------------------------------------
Início
    Para i de inicial até final

        soma1 = distPlanetas[i] + distPlanetas[i+1]
        soma2 = distPlanetas[menor] + distPlanetas[menor+1]

        Se inicial = 0
            Se soma1 < soma2
                menor = i
            Fim Se
        Senão
            Se soma1 <= soma2
                menor = i
            Fim Se
        Fim Se
    Fim Para

    distPlanetas[menor] += distPlanetas[menor+1]
    Para i de menor até final
        distPlanetas[i+1] = distPlanetas[i+2]
    Fim Para

    Retorna menor+1
Fim
-----------------------------------------------------------------------
```

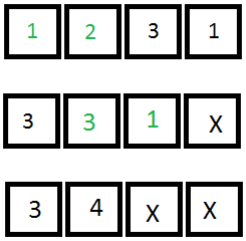


Figure 3: Exemplo de execução do Algoritmo Guloso, no caso existem três planetas no caminho e uma conquista é almejada.

Esse algoritmo guloso não é ótimo pois não funciona para todos os casos. Os testes em que o Algoritmo normalmente falha são os que possuem um número de conquistas próximos à metade de planetas. Ou seja, quanto maior o percentual de planetas conquistados, menores são as possibilidades de que esse algoritmo falhe.

3 Análise teórica de custo assintótico (espaço)

3.1 Força Bruta

Na execução por força bruta temos uma árvore binária de N níveis. Uma árvore binária de N nós possui a seguinte complexidade:

$$\sum_{x=0}^{N} 2^x$$

.

Temos também o nível $N+1$, como todos os nós do nível N só possuem um filho, adicionamos 2^N à soma, ficando:

$$2^N + \sum_{x=0}^{N} 2^x$$

.

3.2 Algoritmo Guloso

No caso do algoritmo guloso não há nada além do vetor que armazena os valores das distâncias entre os planetas, portanto, a complexidade em termos de espaço é $N+1$[1]

4 Análise teórica de custo assintótico (tempo)

4.1 Força Bruta

O custo da execução em termos de tempo se equipara ao custo de espaço em termos de notação, tendo em vista o número de nós, o custo de popular a árvore é de:

$$2^N + \sum_{x=0}^{N} 2^x$$

.

O custo para verificar qual é a solução não tem impacto relevante no tempo final de execução do algoritmo.

[1] Onde N é o número de planetas na rota.

4.2 Algoritmo Guloso

O guloso escolhe a menor soma do vetor e depois escolhe a segunda menor na parte do vetor subsequente à menor soma que foi previamente encontrada. Desta forma o custo em questão de tempo varia de acordo com a característica da entrada. Tendo a entrada como:

- N como número de planetas no caminho.
- K como o número de planetas a serem conquistados.

O número de planetas na rota e o número de conquistas influenciam da seguinte forma: O número de operações exercidas sobre o vetor se dá pelo número de planetas menos o número de conquistas. Desta forma, o número de planetas influencia pela tamanho do vetor, e o número de conquistas influencia pelo número de vezes que o vetor será manipulado. Portanto, quanto menor o número de planetas e maior o número de conquistas, mais rápida será a execução.

Outra característica que altera o tempo de execução do algoritmo é a forma como os dados estão disponibilizados no vetor, se os menores valores sempre se encontram no início, então há um custo maior pra movimentar o vetor de forma a concatenar o mesmo. Outra situação é de que as menores somas estejam mais para o fim do vetor, essa condição é bem menos adversa e só apresenta um caso não tão bom quando os itens que formam a soma não permitem que a próxima seja calculada na sequência, sendo necessário voltar ao início, o custo é menor pois não agrega o ônus da movimentação de grande parte do vetor.

Sendo assim, o custo da função é da ordem $O((N-K)xA^2)$.

5 Execução e análise de experimentos

5.1 Força Bruta

Para os testes do algoritmo de Força Bruta, foram executados 24 testes, com o número crescente de 1 até 24 planetas na rota, sendo que o número de planetas a serem conquistados é sempre a metade do número de planetas na rota.

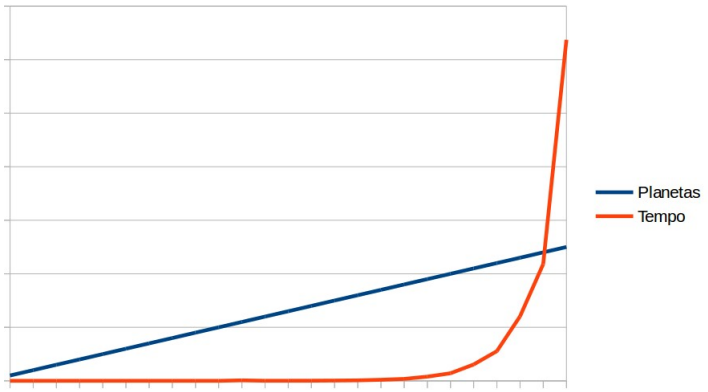
Os testes aplicados são:

- 1 planeta e nenhuma conquista
- 2 planetas e 1 conquista
- 3 planetas e 1 conquista
- 4 planetas e 2 conquistas
- 5 planetas e 2 conquistas
- 6 planetas e 3 conquistas

[2]Onde A é o tamanho médio das movimentações no vetor

- 7 planetas e 3 conquistas
- 8 planetas e 4 conquistas
- 9 planetas e 4 conquistas
- 10 planetas e 5 conquistas
- ...
- 24 planetas e 12 conquistas

Para as entradas especificadas acima, temos abaixo um gráfico com seus respectivos resultados em termos de tempo de execução:



5.2 Algoritmo Guloso

Para os testes do algoritmo guloso foram executados 50 testes, contendo de 10 até 500 planetas na rota, sendo que o crescimento da entrada no número de planetas se dá de 10 em 10.

Os testes aplicados são:

- 10 planetas e 5 conquistas
- 20 planetas e 10 conquistas
- 30 planetas e 15 conquistas
- 40 planetas e 20 conquistas
- 50 planetas e 25 conquistas
- ...
- 500 planetas e 250 conquistas

Para as entradas especificadas acima, temos abaixo um gráfico com seus respectivos resultados em termos de tempo de execução:

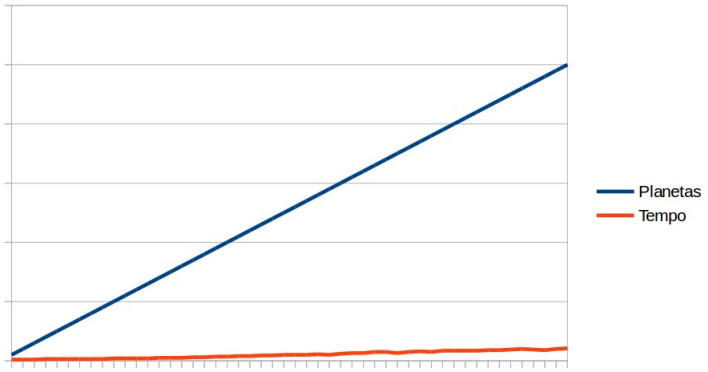


Figure 4:

6 Conclusão

A execução dos testes acompanhou o raciocínio abordado na parte de análise teórica de tempo. Onde o algoritmo guloso apresentou crescimento a níveis muito pequenos, enquanto o Força Bruta apresentou crescimento acima do padrão de crescimento das entradas.