Trabalho Prático 2
Algoritmos 1

Daniel Ferreira Abadi[1] **- 2018088062**

[1] Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG – Brasil

1. Introdução

O problema abordado consiste em simular o Jogo dos Diamantes utilizando programação dinâmica. O mesmo funciona da seguinte forma, recebemos um conjunto de pedras de diamantes e seus respectivos pesos, podemos então ”combinar”tais pedras chocando umas com as outras. Se os pesos das pedras são iguais, temos que ambos os diamantes são destruídos, e se os pesos são diferentes, a pedra com peso menor é destruída e o resto da maior é uma nova pedra. O objetivo do jogo é ficar sem pedras, ou com uma pedra com o menor valor possível.

2. Implementação

Para representar o conjunto de diamantes simplesmente iniciei um vetor com a quantidade dada inicialmente. Para realizar o trabalho tomei a decisão de usar o algoritmo KnapSack visto em aula. Para poder utilizá-lo, peguei o teto da divisão por 2 da soma de todos os pesos, tal resultado seria o peso total passado ao algoritmo, o motivo será explicado na subseção abaixo.

2.1. KnapSack

Aqui irei explicar o porquê de ter escolhido tal algoritmo, visto que a implementação do mesmo com programação dinâmica é idêntica ao do livro base da matéria.

Inicialmente fiquei tentado a soluções gulosas, tentei minimizando as subtrações, pegando o maior elemento e subtraindo o segundo maior ou menor elemento. Porém nada disso deu certo e me convenci de que possivelmente não haviam soluções gulosas para o mesmo. Então pensei em dividir o conjunto de diamantes em 2, tais que os conjuntos possuíssem a menor diferença de peso possível, e isso me pareceu extremamente parecido com o problema de KnapSack. Para tal divisão, peguei a metade da soma de todas as pedras e fixei como peso para o algoritmo. Como ele sempre me retorna o conjunto otimizado, apenas peguei este resultado, que é o conjunto 1, e subtraí do peso total, gerando o valor do conjunto 2. Logo a resposta seria a diferença entre os dois conjuntos.

3. Instruções de compilação e execução

Vá até a pasta Daniel_Ferreira_2018088062, pelo terminal, e digite “make” para compilar. Para executar o programa digite “./tp2”. Há também a opção “make test” que executa o programa utilizando os casos teste dados.

Há uma pasta chamada src2, que contém o arquivo com a implementação alternativa sem utilizar programação dinâmica, com seu próprio make file. Para compilá-la basta entrar na pasta src2 e digitar ”make”, e gerará o executável ”./alternativo”.

4. Análise de complexidade de tempo e espaço

Aqui irei focar apenas na complexidade do algoritmo KnapSack, visto que o main tem apenas um loop que recebe as entradas.

Em termos de tempo o algoritmo tem 2 loops aninhados, sendo o externo de 0 até o número de items e o interno de 0 até o peso passado, logo temos uma complexidade de O(n*peso), sendo n o número de pedras, visto que a função std::max dentro do loop tem complexidade O(1). Já em termos de complexidade espacial temos que o algoritmo cria uma matriz de n linhas por todos os pesos, ou seja, cada linha tem de 0 até o peso máximo em colunas. Portanto, temos que a complexidade espacial também é da ordem de O(n*peso).

Sobre o algoritmo sem programação dinâmica pedido, fiz uma versão recursiva do KnapSack e a complexidade temporal da mesma é algo em torno de O(2^n), e essa diferença será bem visível na próxima seção.

5. Análise experimental

O gráfico abaixo mostra a relação entre tempo e quantidade de pedras. Escolhi pular de 5 em 5 pedras para ficar melhor visualmente.

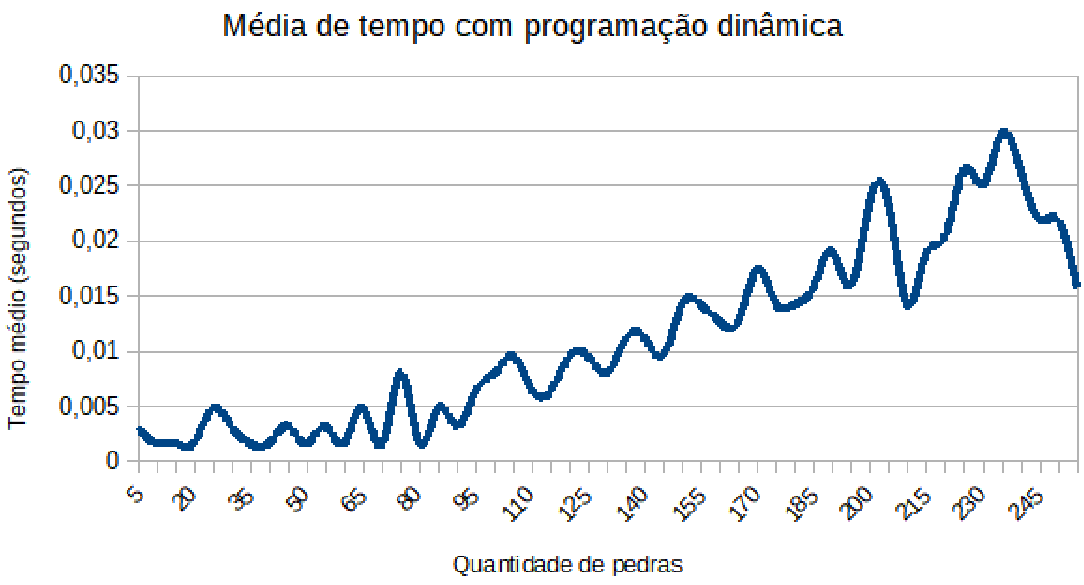

Figura 1. Média com programação dinâmica

Para cada quantidade de pedras foram feitas 10 amostras variando de 5 até 255, variando os pesos aleatoriamente até o máximo de 128. Como visto no gráfico acima, o algoritmo utilizando programação dinâmica é realmente muito rápido se comparado a força bruta ou ao próprio sem utilizar deste conceito. Visualmente me parece que é um tempo polinomial, o que corrobora com a complexidade dada na seção 4. Abaixo está o gráfico com os desvios padrões de cada medida feita.

O próximo gráfico mostra o tempo do KnapSack sem utilizar programação dinâmica. O mesmo também foi feito de 5 em 5 pedras, gerando valores aleatórios para

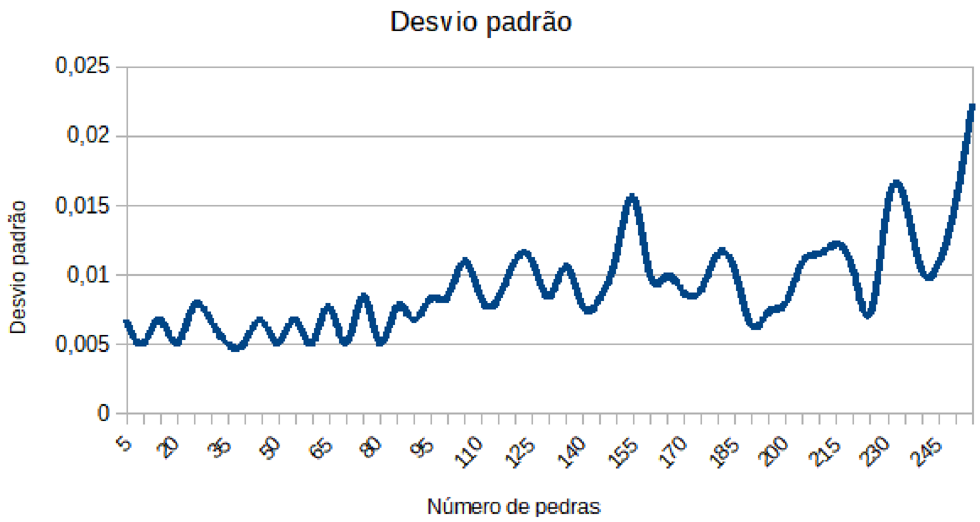

Figura 2. Desvio padrão

os pesos das mesmas. Note que só vai até 35 pedras, isso porque com 40 pedras 10 minutos não foram suficientes. Este grande salto nos tempos de execução representa bem a característica exponencial do algoritmo.

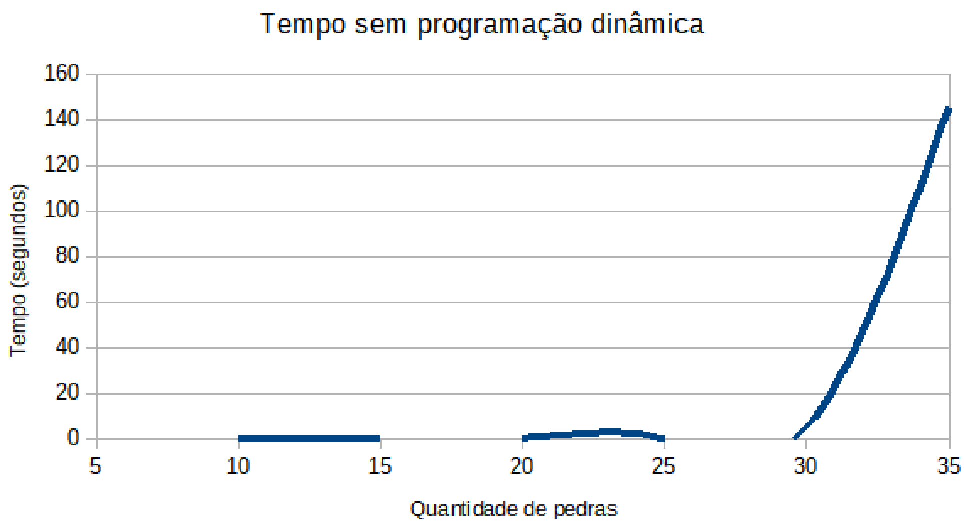

Figura 3. Gráfico média com DP

6. Conclusão

Tendo em comparação os tempos de execução das duas versões do KnapSack, é notável o poder que a programação dinâmica nos oferece, partindo de algo que poderia levar dias, ou anos, para tempos extremamente pequenos. Como dito anteriormente, tive bastante dificuldade de encaixar o problema dado com o conceito de programação dinâmica, o que mostra que mesmo sendo uma ferramenta excelente ela necessita de uma modelagem muito boa para ser corretamente utilizada. A única referência utilizada foi o livro texto da matéria [Kleinberg and Éva Tardos 2006].

Referências

Kleinberg, J. and Éva Tardos (2006). *Algorithm Design*. Pearson/Addison-Wesley, 1th edition.