Trabalho de Projeto e Análise de Algoritmos

Gabriel Luciano Gomes - 569631
glgomes@sga.pucminas.br

26 de maio de 2018

1 Abordagem Algoritmo Guloso

Para modelarmos o problema de forma Gulosa, devemos definir uma prioridade para que o prato seja escolhido de forma correta, ou seja, o melhor prato seja escolhido no dia corrente. Em razão disso, defini como melhor forma, o melhor custo benefício, onde a razão é definida por custo / lucro. Junto a isso, uma lista ordenada com os melhores custo-benefíc io é gerada e ordenada de forma crescente para que possa ser verificado qual melhor prato a ser escolhido no dia vigente.

Por outro lado, caso o prato seja escolhido mais de um dia em sequência, seu lucro é reduzido em 50%. Em função disso, a mesma estratégia anterior é aplicada aos pratos com 50% do lucro do primeiro dia, ou seja, calcular os novos lucros-benefícios com os lucros reduzidos e inseridos à lista dos melhores pratos a serem escolhidos que serão ordenados novamente. Por fim, concatenamos à lista os elementos pratos ordenados com o menor custo, visto que seu lucro é de 0 (por serem repetidos 2 ou mais vezes) e o respectivo lucro-benefício zerado, impossibilitando erro por divisão por 0.

Feito isso, devemos analisar a lista definida a vir de gerar a melhor sequência de pratos disponíveis para maximizar o lucro. Onde é feita por uma verificação de duas variáveis J e K, que controlarão o prato anterior e o iterador de pratos da lista. Inicialmente iremos caminhar com J e K para a primeira posição válida da lista, visto que elementos com um custo superior ao orçamento, porém com lucro muito elevado podem ocupar as primeiras posições desta. Em seguida iremos adicionar o primeiro prato válido a lista resposta e reduzir o seu custo do orçamento total. Após isso verificamos se o prato escolhido é igual a um prato escolhido anteriormente, se for, a variável que contém a posição anterior recebe o valor da variável J; se não, o J recebe o valor de K que representa a possibilidade voltar e utilizar um prato anterior do que o da posição da lista, visto que tem um melhor custo benefício. Após isso, verificamos se a variável J possui valor maior ao tamanho da lista, que representa a tentativa de todas as possibilidades da lista e não encontrou um prato que tinha um prato inferior ou igual ao orçamento restante e, caso isso ocorra, a lista de resposta é esvaziada.

Por fim, retornaremos a lista resposta que será tratada da seguinte maneira: Se for vazia, a resposta é impossível e terá o método de saída correspondente ao inválido; se possuir 1 ou mais elementos, terá o método de saída correspondente ao válido.

2 O Algoritmo guloso utilizado dá uma solução ótima? Por quê?

O algoritmo utilizado não dá umá solção ótima, pois se apresenta uma entrada com a configuração: 2 dias 3 pratos e orçamento 20, onde prato 1 : custo 20 e lucro 200; prato 2: custo 4 e lucro 4; prato 3: custo 5 e lucro 10, nos dá a resposta errado. Isso acontece pois o prato 1 é o melhor prato com custo benefício, mas ao selecioná-lo não teremos mais orçamento para atender outros pratos e consequentemente a isso, não nos dando uma solução. Nesse caso, deveríamos pegar os pratos de menor custo, pois teríamos uma solução que atenderia o problema, mas não atenderia as características de um Algoritmo Guloso.

3 Abordagem Algoritmo Dinâmico

Para modelar o problema em maneira dinâmica, utilizaremos n tabelas (quantidade de dias), que serão compostas pelo orçamento atual e referência dos pratos existentes para um determinado problema. A tabela será preenchida de forma que, iremos guardar o maior valor lucro obtido no dia, analisando os pratos existentes, semelhante ao problema da mochila, mas diferenciado em um quesito que, se só se escolhe um prato por dia, logo o maior orçamento irá repetir em todo o restante da tabela, até encontrar um outro maior, ou até o fim, caso contrário.

Juntamente a isso, teremos uma análise dos pratos utilizados, e verificaremos os pratos que serão utilizados se serão repetidos ou não, por meio de uma tabela auxiliar, que terá a mesma ideia do que a primeira, só que terá referência do item que fora utilizado no dia anterior, e caso não tenha nenhum no dia anterior, ou não puder ser pego nenhum deles, terá o valor de -1, contido em sua posição. Essa tabela também auxiliará na construção do caminho que levará a melhor solução. Vale ressaltar também a ideia utilizada foi baseada na aplicada na tabela LCS (*Longest Commom Subsequence*).

A saída terá a mesma forma que o algoritmo guloso. Entretanto, para a descoberta do mesmo, verificaremos as condições foram atendidas para a entrada (o orçamento, pratos foi suficiente para todos os dias). Caso as condições for verdadeira, iremos verificar a primeira tabela para verificar qual o maior obtido em todos os dias de preparamento, e analisaremos a tabela auxiliar para verificar quais foram os pratos utilizados para tal concluão, onde para isso, iremos pegar o valor atual da tabela 1, verificar a mesma posição, porém na tabela auxiliar e verificar o prato anterior e assim por diante. E, se as condições forem falsas, iremos mostrar a impossibilidade da solução (mostra na tela o valor 0.00).

4 Sub-estrutura ótima e sobreposição dos problemas

Para a solução desse problema, a melhor prática foi que utiliza a programaçâo dinâmica, pois ela fornece a resposta para todas a possibilidades possíveis para determinada entrada. Logo, cada coluna da tabela dinâmica produzida, é uma respota para um deternimado orçamento ou quantidade de dias desejado. Sendo assim, todos os subproblemas são armazenados e

o acesso a quaisquer um deles é definido pelo critério de seleção ao percorrer as tabelas produzidas.

5 Custo Assintótico Algoritmos

5.1 Algoritmo Guloso

- Estrutura for para ser passado a quantidade n de dias do problema
  - Estrutura while dentro deste, para verificar qual prato a ser escolhido no dia.
    - Comparação para escolher um prato disponível
      Com isso, temos:

$$O = \sum_{k=0}^{n} i = \frac{n * (n-1)}{2} = O(n^2)$$

5.2 Algoritmo Dinâmico

- Estrutura for para percorrer a quantidade n de dias do problema
  - Estrura for percorrer todos os m pratos registrados
    - Estrura for percorrer os$i$ valores de orçamento que vão de 0 até $k+1$
      - Estrura for para percorrer os m pratos e escolher qual o melhor prato para preencher a matriz no momento atual

$$O = k * n * \sum_{k=1}^{m+1} i = k * n * \frac{m * (m-1)}{2} \approx O(n^4)$$

6 Algoritmos clássicos adaptados e utilizados

Para solução do problema em programação dinâmica, foram utilizadas as ideias do Problema da Mochila e o do algoritmo LCS (Longest Commom Subsequence).

7 Máquina utilizada para testes

- **Processador:** Intel Core i5 - 4210U - 1.70Ghz, 2.40 Ghz
- **Memória Ram:** 8GB 1600Mhz DDR 3
- **Sistema Operacional** Windows 10 Home Single Language - 64bits
- **Disco Rígido:** SSD 240GB
- **Placa Gráfica:** GT- 740M - 2GB DDR3 128bits

8 Comportamento em relação a tempo de execução e memória alocada

Quanto o comportamento em relação a tempo, foram realizados testes onde a quantidade de dias, pratos e orçamentos foram variados para realização do mesmo. Abaixo seguem os resultados obtidos.

Figura 1: Variante - Número de dias

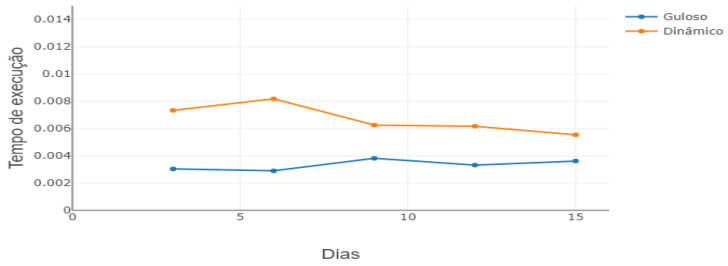


Figura 2: Variante - Número de pratos

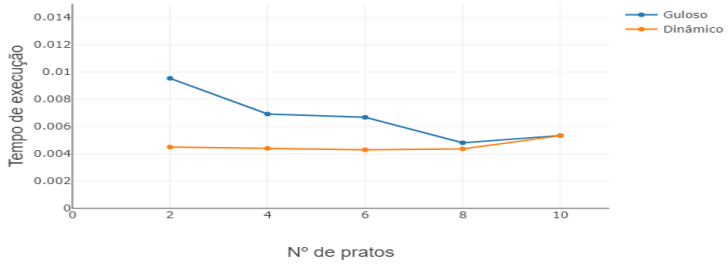
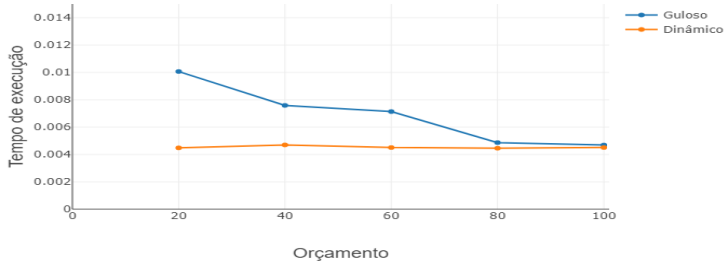


Figura 3: Variante - Orçamento máximo



Quanto a memória alocada, no algoritmo guloso utilizamos apenas um vetor de um tipo personalizado, chamado "dish". Neste, se encontram 3 variáveis do tipo double e 1 do tipo inteiro, com isso temos:

- Double - 8 Bytes
- Int - 4 Bytes

Logo,

$MemAlocada \approx (3 * 8 + 4) * n$ Bytes, tendo n igual ao número de pratos da entrada

Figura 4: Estrutura utilizada no Algoritmo Guloso

```
struct dish {
  int id;
  double cost;
  double profit;
  double benefitcost;
};
```

Já na abordagem dinâmica, utilizamos dois vetores do tipo Long, os quais definem a tabela dos melhores resultados e uma auxiliar para definir o caminho para escolha das melhores soluções. Nestes vetores se encontram as dimensões k, nem, que são as quantidade de dias, pratos e orçamento total para determinada entrada. Sendo assim, temos:

- Long - 4 Bytes

Logo,

$$MemAlocada \approx 2 * [4 * (k * n * m)] \ Bytes$$

Figura 5: Vetores utilizados na programação dinâmica

```
//Table used on DynamicProgramming, started with -1 value, which indicates the dish
// isn't available to be chosen.
vector<vector<vector<float>>> dp(k, vector<vector<float>>(n, vector<float>(m + 1, -1)));

//Matrix that stores the path of chosen dishes
vector<vector<vector<short>>> path(k, vector<vector<short>>(n, vector<short>(m + 1, -1)));
```

9 Bibliografia

CUNHA, Felipe. Projeto e análise de algoritmos. 01 feb. 2018, 14 jun. 2018. Notas de Aula.

Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. 2001. Introduction to Algorithms (2nd ed.). McGraw-Hill Higher Education.