Universidade Estadual de Maringá
Departamento de Informática – Projeto e Análise de Algoritmos
Prof. Daniel Kikuti

Lista de exercícios: Programação Dinâmica e Gulosos

1. O Dia das Bruxas (*Halloween*) está próximo – entre o pôr do sol de 31 de outubro e 01 de novembro. Em alguns países que celebram a data, as crianças vestem fantasias e visitam as casas da vizinhança perguntando "doces ou travessuras?" (*trick or treat?*). Se as pessoas responderem "doce" essas mesmas dão doces às crianças, se disserem "travessura" as crianças assustam-nas com máscaras, ou fazendo travessuras como encher a frente da casa com papel higiênico e sprays de espuma colorida.

   No bairro onde Harry mora, as crianças são muito espertas. Elas possuem um mapa com a quantidade de doces que elas conseguirão em cada casa. Para não haver disputa entre os grupos de crianças, elas chegaram num acordo em que cada grupo deve escolher uma rua, e devem pedir doces em casas que estão no mesmo lado da rua. Também não podem pedir doces em casas que são vizinhas diretas.

   Por exemplo, suponha que na rua escolhida por Harry (Rua dos Alfeneiros) existem cinco casas em um dos lados da rua, com as seguintes quantias de doces $\{1, 3, 7, 3, 2\}$. Se ele escolher a segunda casa, ele não pode visitar a primeira, nem a terceira casa, mas poderia visitar também ou a quarta ou a quinta casa. Escolhendo a quarta casa, o total de doces obtido seria 6. O conjunto de casas que maximiza a quantidade de doces neste exemplo seria composto pelas casas de índice $\{1, 3, 5\}$, totalizando 10 doces.

   O problema. Dado um vetor $A[1..n]$ no qual cada posição i representa uma casa e $A[i]$ representa a quantidade de doces que podem ser obtidos naquela casa, maximize a quantia de doces respeitando a restrição de vizinhança.

   Considerando a descrição acima, pede-se:

   (a) Mostre que o problema possui subestrutura ótima.

   (b) Mostre que há sobreposição de problemas.

   (c) Apresente um algoritmo que faz uso da técnica de programação dinâmica.

   (d) Faça a análise da complexidade do seu algoritmo.

   (e) Considere o algoritmo guloso que iterativamente escolhe a casa com maior quantidade de doces e que não é vizinha direta de alguma das casas previamente visitadas. Esta escolha gulosa funciona? Justifique?

2. Considere o cálculo do número de combinações de n elementos k a k, isto é, o número de subconjuntos de k elementos dentre n. Sabe-se,

$$\binom{n}{k} = \begin{cases} 1 & \text{se } k = 0 \text{ ou } n = k \\ \binom{n-1}{k} + \binom{n-1}{k-1} & \text{se } n \geq 2 \text{ e } 0 < k < n. \end{cases}$$

   (a) Mostre que há sobreposição de problemas.

   (b) Apresente um algoritmo que usa programação dinâmica.

3. Dada uma matriz de custo $M_{m \times n}$, determine o caminho de custo mínimo partindo de $(1, 1)$ e chegando em (m, n). Cada célula da matriz M representa o custo de se passar pela célula. O custo total de um caminho é a soma de todos os custos das células usadas neste caminho (incluindo a fonte e o destino). De uma determinada célula só é possível ir para baixo, direita ou diagonal.

   Por exemplo, para a matriz M abaixo, o caminho de custo mínimo é composto pelas células $(1, 1) \to (2, 1) \to (3, 2) \to (3, 3) \to (3, 4)$, com custo total 8 $(1 + 4 + 1 + 1 + 1)$.

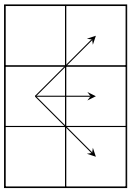
1	5	8	3
4	7	6	9
2	1	1	1

   (a) Apresente uma formulação recursiva para o problema (mostre que o problema possui subestrutura ótima).

   (b) Apresente um algoritmo guloso para o problema.

   (c) Mostre que seu algoritmo sempre devolve corretamente a resposta ou apresente uma instância em que ele falha.

4. Dada uma matriz $m \times n$ de inteiros, desejamos calcular um caminho de custo mínimo. Um caminho começa em qualquer lugar na coluna 1 (primeira coluna) e consiste de uma sequência de passos que terminam na coluna n (última coluna). Um passo consiste em viajar de coluna i para coluna $i + 1$ em uma linha (horizontal ou diagonal) adjacente. As primeiras e últimas linhas (linhas 1 e m) de uma matriz são consideradas adjacentes, ou seja, a matriz se "enrola" tal que ela representa um cilindro horizontal. Passos legais são ilustrados abaixo.



O custo de um caminho é a soma dos números inteiros em cada uma das n células da matriz que são visitadas. Por exemplo, duas matrizes 5×6 ligeiramente diferentes são mostradas abaixo (a única diferença são os números da linha inferior).

3	4	1	2	8	6
6	1	8	2	7	4
5	9	3	9	9	5
8	4	1	3	2	6
3	7	2	8	6	4

3	4	1	2	8	6
6	1	8	2	7	4
5	9	3	9	9	5
8	4	1	3	2	6
3	7	2	1	2	3

O caminho mínimo está ilustrado em cada matriz. Note que o caminho para a matriz da direita tira proveito da propriedade adjacência das primeira e última fileira.

(a) Mostre que o problema possui subestrutura ótima.

(b) Mostre que há sobreposição de problemas e analise a complexidade da solução força bruta.

(c) Escreva um algoritmo que calcula o custo de um caminho mínimo e que faz uso da técnica de programação dinâmica (*bottom-up* ou *top-down*, a seu critério). Informe sua complexidade.

5. Uma determinada linguagem oferece uma operação primitiva que divide uma string em duas partes. Esta operação envolve a cópia da string original, e leva n unidades de tempo para uma string de tamanho n, independentemente da localização do corte. Suponha agora que você quer particionar a string em várias partes. A ordem em que as partições são feitas podem afetar o tempo total de execução. Por exemplo, dada uma string de 20 caracteres e cortes a serem feitos nas posições 3 e 10, se cortarmos primeiramente na posição 3 teremos um custo total de $20 + 17 = 37$, enquanto que cortando na posição 10 primeiro teremos um custo menor de $20 + 10 = 30$.

(a) Mostre que o problema possui subestrutura ótima.

(b) Escreva um algoritmo que faz uso da técnica de programação dinâmica (*bottom-up* ou *top-down*, a seu critério) para este problema.

(c) Considere um algoritmo guloso que sempre escolhe "o maior corte primeiro". Mostre que esta escolha nem sempre produz a solução ótima.

6. *Longest Increasing Subsequence (LIS).* Dada uma sequência de números naturais, determinar qual a subsequência crescente com mais elementos. Exemplos:

- $\{1, 2, 3, 4\}$ é a subsequência crescente máxima de $\{1, 2, 5, 3, 4\}$.
- $\{2, 4, 5, 7, 8, 9\}$ é a subsequência crescente máxima de $\{2, 4, 1, 5, 3, 7, 6, 8, 9\}$.
- $\{0, 2, 6, 9, 11, 15\}$ é uma subsequência crescente máxima de $\{0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15\}$ (note que $\{0, 4, 6, 9, 11, 15\}$ ou $\{0, 4, 6, 9, 13, 15\}$ também são).

(a) Mostre que o problema possui subestrutura ótima.

(b) Escreva um algoritmo que faz uso da técnica de programação dinâmica (*bottom-up* ou *top-down*, a seu critério). O algoritmo deve devolver o tamanho da subsequência crescente máxima.

(c) Analise a complexidade do seu algoritmo.

7. Considere o seguinte problema de seleção de atividades. "Dado um conjunto de n atividades $S = \{a_1, \ldots, a_n\}$ que requerem o uso de um recurso comum e, os tempos de início e término de cada atividade $[s_i, f_i)$, selecionar o maior conjunto possível de atividades mutuamente compatíveis, isto é, atividades a_i e a_j tais que $s_i \geq f_j$ ou $s_j \geq f_i$."

(a) Seja S_{ij} o conjunto de atividades que começam após o término da atividade a_i e terminam antes do início da atividade a_j. Denote por A_{ij} um conjunto máximo de atividades mutuamente compatíveis em S_{ij} que contém alguma atividade a_k. Mostre que esta formulação para o problema apresenta subestrutura ótima.

(b) Seja $|A_{ij}|$ o tamanho de uma solução ótima para o conjunto S_{ij}. Considerando a recorrência a seguir, apresente um algoritmo que faz uso da técnica de programação dinâmica para resolver este problema.

$$|A_{ij}| = \begin{cases} 0 & \text{se } S_{ij} = \emptyset \\ \max\limits_{a_k \in S_{ij}} (|A_{ik}| + |A_{kj}| + 1) & \text{se } S_{ij} \neq \emptyset \end{cases}$$

(c) Analise a complexidade do algoritmo desenvolvido no item anterior no pior caso.

(d) O algoritmo a seguir resolve o problema corretamente? Justifique.

```
ALGORITHM(S)
1 while S ≠ ∅ do
2 |   Selecione a_k cujo intervalo sobrepõe menos as demais atividades
3 |   Adicione a_k no conjunto S'
4 |   Remova todos as atividades de S que sobrepõe a_k
5 return S'
```

8. Considere o problema de corte de hastes definido a seguir. "Dada uma haste de tamanho n e uma tabela de preços p_i, para $1 \leq i \leq n$, determinar a receita máxima r_n obtida após cortar a haste e vender os pedaços." O algoritmo a seguir faz uso da subestrutura ótima do problema (assumindo que $r_0 = 0$), isto é, $r_n = \max\limits_{1 \leq i \leq n} (p_i + r_{n-i})$.

```
CUT-ROD(p, n)
1 if n == 0 then
2 |   return 0
3 q ← −∞
4 for i ← 1 to n do
5 |   q ← max(q, p[i] + CUT-ROD(p, n − i))
6 return q
```

(a) Agora, considere uma modificação do problema de corte de hastes no qual, além de um preço p_i cada corte incorre em um custo fixo c. A receita associada à solução agora é a soma dos preços das peças menos os custos da execução dos cortes. Escreva um algoritmo que usa a técnica de programação dinâmica para este novo problema.

(b) Analise a complexidade do seu algoritmo.

(c) Escreva um algoritmo que imprime quais cortes fazem parte de uma solução ótima.

(d) Mostre que a seguinte estratégia gulosa nem sempre determina uma maneira ótima de se fazer cortes em hastes. Defina a *densidade* de uma haste de comprimento i como sendo p_i/i, isto é, seu valor por polegadas. A estratégia gulosa (para uma haste de tamanho n) corta um pedaço de tamanho i ($1 \leq i \leq n$), com a densidade máxima. Então, continua-se aplicando a estratégia gulosa no pedaço restante de tamanho $n - i$.

9. O problema da mochila é um problema de otimização combinatória, em que o objetivo é encher uma mochila com objetos de diferentes pesos e valores, de maneira a maximizar a soma dos valores dos objetos, não ultrapassando a capacidade de peso da mochila.

Existem duas variantes deste problema: **mochila binária**, na qual os objetos devem ser colocados de maneira integral (por exemplo, quadros, estátuas, ...); e **mochila fracionária**, na qual os objetos podem ser fracionados e então colocadas porções destes objetos na mochila (ouro em pó, pó de diamante, ...).

Sejam n a quantidade de itens, v_i o valor do item i, w_i o peso do item i e W capacidade total da mochila, pede-se:

(a) Mostre que o problema tem subestrutura ótima.

(b) Mostre que o algoritmo guloso que pega iterativamente o i-ésimo item com maior relação (v_i/w_i) sem ultrapassar a capacidade da mochila não garante a solução ótima para o problema da mochila binária.

(c) Mostre que usando o algoritmo do item anterior é possível obter o ótimo no problema da mochila fracionária. [Demonstração da propriedade de escolha gulosa.]

10. Considere o seguinte problema de seleção de atividades. "Dado um conjunto de n atividades $S = \{a_1, \ldots, a_n\}$ que requerem o uso de um recurso comum e, os tempos de início e término de cada atividade $[s_i, f_i)$, selecionar o maior conjunto possível de atividades mutuamente compatíveis, isto é, atividades a_i e a_j tais que $s_i \geq f_j$ ou $s_j \geq f_i$." Mostre que selecionar iterativamente as atividades de acordo com as seguintes escolhas gulosas não leva ao ótimo global.

(a) Selecione a atividade com menor tempo de início.

(b) Selecione a atividade que requer o menor intervalo de tempo.

(c) Selecione a atividade com o menor número de conflitos.

11. Considere o seguinte problema de escalonamento de tarefas: n tarefas disputam o uso exclusivo de um recurso. Cada tarefa i possui um tempo t_i necessário para executá-la e uma prioridade p_i. Definimos o tempo de término c_i para a tarefa i como sendo a soma de todos os tempos de término das tarefas antecedentes a i, incluindo t_i. O objetivo é minimizar a soma ponderada dos tempos de término, isto é, $\min \sum_{i=1}^{n} p_i c_i$.

    Mostre que a escolha de tarefas em ordem decrescente conforme a relação $p_i t_i$ (ou opcionalmente, $p_i^{t_i}$) sempre leva ao ótimo global, ou apresente um contra exemplo.

12. Considere o problema de Soma de Subconjuntos descrito a seguir: "Dado uma coleção de números naturais $S = \{a_1, \ldots, a_n\}$ e um valor s, existe um subconjunto S' de S tal que $\sum_{x \in S'} x = s$?" Apresente um algoritmo guloso para este problema. Mostre que ele está correto ou apresente uma instância em que ele falha.

13. Considere o problema de escalonamento de intervalos ponderados, onde cada intervalo i está associada a um tempo de início s_i, a um tempo de término f_i e a um peso p_i. Neste problema o objetivo é encontrar um subconjunto de intervalos disjuntos (para i e j quaisquer deste subconjunto, $f_i \leq s_j$ ou $s_i \geq f_j$) cuja soma dos pesos máxima.

    (a) Mostre que o algoritmo guloso que escolhe sempre "o intervalo que termina primeiro", aplicado recursivamente ao subconjunto obtido pelo descarte de intervalos não disjuntos ao intervalo da escolha gulosa, nem sempre produz a solução ótima.

    (b) Ainda considerando a escolha gulosa do item anterior, e se os intervalos tivessem peso $p_i = 1$? O algoritmo funcionaria? Argumente.

14. Dadas duas palavras x e y, a distância de edição entre as duas é definida como sendo o número mínimo de caracteres que precisam ser substituídos, removidos ou adicionados para transformar uma em outra. Exemplos:

    - "*cat*" e "*bat*". As duas palavras diferem apenas na primeira letra. Basta substituirmos o '*c*' de *cat* por um '*b*' para chegar em *bat*. Portanto, a distância de edição é 1.
    - As palavras "fly" e "$flying$" são idênticas nos três primeiros caracteres, mas a segunda palavra possui três caracteres adicionais. Adicionando "ing" na primeira palavra produz a segunda palavra. A distância de edição neste caso é 3.
    - Considere as palavras "$grave$" e "$groovy$". Podemos efetuar as seguintes substituições na primeira palavra: (1) 'a' $\rightarrow$ 'o', (2) 'e' $\rightarrow$ 'y', depois (3) insira a letra 'o' na posição 4 (após o primeiro 'o'). Portanto, a distância de edição neste caso é 3.

    (a) Apresente uma formulação recursiva para o problema. Mostre que o problema possui subestrutura ótima.

    (b) Mostre que há sobreposição de problemas.

    (c) Apresente um algoritmo que faz uso da técnica de programação dinâmica.

    (d) Analise a complexidade do seu algoritmo.

    (e) Descreva um algoritmo que informa quais as operações devem ser feitas para transformar uma palavra em outra.

15. "Para quê serve Programação Dinâmica?" Certamente, para os alunos de Ciência da Computação, a resposta para esta questão é tão fundamental como a resposta para: "O que vai ter para o almoço no RU?" Para saciar sua ansiedade, irei responder (a primeira pergunta) de maneira bem simples.

    É um conteúdo que serve para formar um preguiçoso mais esperto e eficiente. Como assim? Nestas alturas do campeonato, provavelmente você já deve ter lido boa parte das questões e percebido que cada questão está associada a um valor e a uma dificuldade (estimada em alguma unidade de tempo). Talvez também já tenha passado por sua cabeça: "tenho que tirar x nesta prova". Assim, seu problema seria minimizar a quantidade de tempo gasto para alcançar ao menos a nota necessária.

    Fácil? Hum... não muito. Testar todas as possibilidades pode demorar mais tempo que o tempo total de prova (dependendo do número de questões). Moral da história: é bom aprender Programação Dinâmica se quiser ficar folgado.

    Exemplo:

    - Para $x = 6$ e a tabela de questões a seguir, o tempo mínimo seria 4.

questões	1	2	3	4
pontos	1	2	3	4
dificuldade	2	2	2	2

    Considerando o enunciado do problema acima, faça:

    (a) Apresente um algoritmo que faz uso da técnica de programação dinâmica e que resolve o problema (devolve o tempo mínimo necessário para se obter a pontuação).

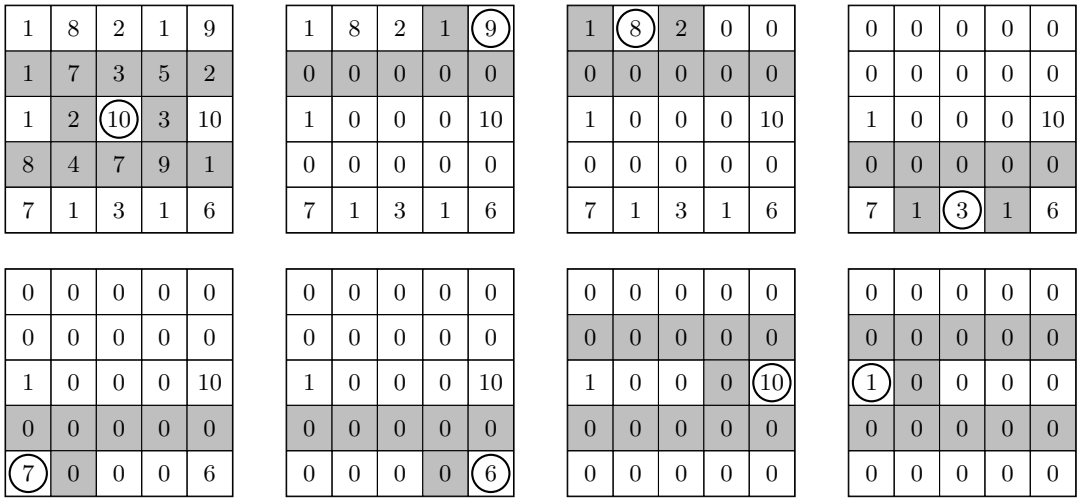
    (b) Mostre que o algoritmo guloso que seleciona dentre as questões restantes aquela com maior relação ponto/dificuldade (até obter um valor maior ou igual a x) nem sempre devolve a solução ótima.

16. Carlinhos é um garoto viciado em doces. Ele é assinante da *All Candies Magazine* (ACM) e foi selecionado para participar da *International Candy Picking Contest* (ICPC).

Nessa competição, um número aleatório de caixas contendo doces são dispostas em M linhas com N colunas cada (existe um total de $M \times N$ caixas). Cada caixa tem um número indicando quantos doces ela contém.

O competidor pode escolher uma caixa (qualquer uma) e pegar todos os doces dentro dela. Mas existe uma sacada (sempre existe uma sacada): quando uma caixa é escolhida, todas as caixas das linhas logo acima e logo abaixo são esvaziadas, assim como as caixas à direita e à esquerda da caixa escolhida. O competidor continua pegando uma caixa até que não hajam mais doces.

A figura abaixo ilustra isso, passo a passo. Cada célula representa uma caixa e o número de doces que ela contém. A cada passo, a caixa escolhida é circulada e as células sombreadas representam as caixas que serão esvaziadas. Após oito etapas o jogo acaba e Carlinhos pegou $10 + 9 + 8 + 3 + 7 + 6 + 10 + 1 = 54$ doces.



Considerando o enunciado do problema acima, faça:

(a) Apresente um algoritmo que faz uso da técnica de programação dinâmica e que resolve o problema de maximizar a quantidade de doces.

(b) Apresente um algoritmo guloso para este problema. Evidencie qual é a escolha gulosa. Argumente que seu algoritmo guloso está correto ou mostre uma situação em que ele não é capaz de devolver a resposta correta.

17. Dada uma sequência $X[1..n]$ de caracteres, $Y[1..m]$ é uma subsequência de X se existem índices $i_1 < i_2 < \cdots < i_m$ tais que $Y[j] = X[i_j]$ para $j = 1, \ldots, m$ (isto é, Y é uma sequência de caracteres, não necessariamente contíguos, retirados de X respeitando a ordem). Por exemplo, Y é uma subsequência de X com índices $2 < 3 < 5 < 7$.

$Y =$

1	2	3	4
C	T	A	C

$X =$

1	2	3	4	5	6	7
G	C	T	G	A	G	C

Uma subsequência é **palíndromo** se é a mesma ao ser lida da esquerda para a direita e vice-versa. Por exemplo, a sequência de caracteres X a seguir possui várias subsequências que são palíndromos, como $Y_1 =$ `ACGCA` e $Y_2 =$ `AAAA`.

$X =$

1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	C	G	T	G	T	C	A	A	A	A	T	C	G

Considere o problema de determinar o comprimento da maior subsequência que é um palíndromo de uma dada sequência X e responda as seguintes questões:

(a) Mostre que este problema possui subestrutura ótima.

(b) Escreva um algoritmo que recebe uma sequência $X[1..n]$, e devolve o maior comprimento de uma subsequência palíndromo de X. Seu algoritmo deve fazer uso da técnica de programação dinâmica e consumir tempo $O(n^2)$ (não esqueça de fazer a análise de complexidade de tempo).

(c) Escreva um algoritmo para obter a maior subsequência (não apenas o seu comprimento).