Universidade Estadual de Maringá
Departamento de Informática – Ciência da Computação
6889 – Projeto e Análise de Algoritmos / Prof. Daniel Kikuti

Aluno(a): _____

Segunda avaliação (Valor: 10,0)[1]

1. [Valor: 3,5] 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) [Valor: 0,7] Mostre que o problema possui subestrutura ótima.

   Solução:
   Começando com a n-ésima casa e pensando na solução ótima S, temos duas possibilidades:

   - $n \in S$: Então sabemos que a casa $n-1$ não faz parte da solução ótima S (restrição de vizinhança), portanto $S' = S - \{n\}$ é solução ótima para para o subproblema $A[1..n-2]$. Seja $|S|$ a quantidade de doces da solução ótima, S' tem que ser solução ótima para o subproblema $A[1..n-2]$, pois caso contrário, se existisse uma solução W' melhor que S' (isto é, $|W'| > |S'|$), poderíamos usar $W' \cup \{n\}$ e obter uma quantia de doces maior que $|S|$, o que contraria a otimalidade de S.
   - $n \notin S$: Então S é solução ótima para o subproblema $A[1..n-1]$. De maneira análoga ao caso anterior, S tem que ser solução ótima para o subproblema.

   Considerando a subestrutura ótima definida acima, podemos definir a seguinte recorrência para o problema:

$$Opt(n) = \begin{cases} 0 & n = 0 \\ A[1] & n = 1 \\ max(Opt(n-1), A[n] + Opt(n-2)) & n \geq 2 \end{cases}$$

   (b) [Valor: 0,7] Mostre que há sobreposição de problemas.

[1] A soma dos valores de todos os exercícios ultrapassa 10,0 pontos, mas seja X o valor total obtido por suas respostas corretas, então sua nota será dada por: $\min(X; 10, 0)$.

Solução:
Árvore parecida com Fibonacci.

(c) [Valor: 0,7] Apresente um algoritmo que faz uso da técnica de programação dinâmica.

Solução:

```
//Inicialize memo[0..n] com -1 em cada posição

HALLOWEEN(A, n)
1 if memo[n] != -1
2    return memo[n]
3 if n == 0
4    memo[n] = 0
5 else if n == 1
6    memo[n] = A[1]
7 else
8    memo[n] = max(HALLOWEEN(A, n-1), A[n] + HALLOWEEN(A, n-2))
9 return memo[n]
```

(d) [Valor: 0,7] Faça a análise da complexidade do seu algoritmo.

Solução:
A complexidade deste algoritmo depende do número de subproblemas distintos que precisam ser resolvidos e do tempo gasto em cada subproblema (desconsiderando recursões). Ao todo existem $n+1$ problemas distintos, e o tempo gasto em cada subproblema é $\Theta(1)$. Portanto, o custo total do algoritmo é $\Theta(n)$.

(e) [Valor: 0,7] 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?

Solução:
Não, considere o seguinte exemplo: $\{5, 8, 10, 8\}$. Escolhendo a casa que fornece 10 doces, seria obrigado a escolher em seguida a casa que fornece 5 doces (15 no total). Mas a solução ótima seria escolher as duas casas que dariam 8 doces cada (16 no total).

2. [Valor: 2,5] 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) [Valor: 0,75] Mostre que o problema tem subestrutura ótima.

   Solução:
   Começando com o n-ésimo item e pensando na solução ótima S, temos duas possibilidades:

   - $n \in S$: Então colocamos o item na mochila, somamos o valor do item ao valor da solução ótima para o subproblema consistindo dos itens $\{1, \ldots, n-1\}$ e capacidade da mochila $W = W - w_n$. $S' = S - \{n\}$ deve ser solução ótima para o subproblema, pois se houvesse uma solução A' cujo valor total fosse maior que o valor de S', poderíamos

substituir S' por A' de tal forma que o valor de $A' \cup \{n\}$ levaria a um valor maior que o de S, o que contradiz a otimalidade de S.

- $n \notin S$: Então S é solução ótima para o subproblema $\{1..n-1\}$ com peso W. De maneira análoga ao caso anterior, S tem que ser solução ótima para o subproblema.

Considerando a subestrutura ótima definida acima, podemos definir a seguinte recorrência para o problema:

$$Opt(i, W) = \begin{cases} 0 & W = 0 \text{ ou } i = 0 \\ Opt(i-1, W) & w_i > W \\ max(Opt(i-1, W), v_i + Opt(i-1, W - w_i)) & \text{caso contrário} \end{cases}$$

(b) [Valor: 0,75] 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.

Solução:
Suponha que temos 3 itens, $W = 50$. A tabela abaixo mostra os valores e pesos dos itens. Usando a escolha gulosa, o valor obtido seria de 160 (itens 1 e 2). A solução ótima para o problema seria pegar os itens 2 e 3, com valor total de 220.

	1	2	3
v_i	60	100	120
w_i	10	20	30

(c) [Valor: 1,0] Mostre que usando o algoritmo do item anterior é possível obter o ótimo no problema da mochila fracionária. [Demonstração da propriedade de escolha gulosa.]

Solução:
Suponha que os objetos estão ordenados em ordem decrescente de valor por peso e, para simplificar a demonstração, assuma que todos os valores v_i/w_i são distintos:

$$\frac{v_1}{w_1} > \frac{v_2}{w_2} > \cdots > \frac{v_n}{w_n}.$$

Seja $\delta = \{x_1, \ldots, x_n\}$ a solução gulosa e $\delta^* = \{y_1, \ldots, y_n\}$ a solução ótima para o problema, onde $x_i, y_i \in [0..1]$ informam se o item i foi usado integralmente $(x_i, y_i = 1)$, ou parcialmente $(0 < x_i, y_i < 1)$ ou não foi usado $(x_i, y_i = 0)$. A solução gulosa consiste de uma sequência de zero ou mais 1s, seguida possivelmente de um número fracionário, seguida de uma sequência de zero ou mais 0s. Ambas as soluções fazem uso da capacidade máxima da mochila.
Se δ e δ^* são iguais, então não há o que demonstrar (a escolha gulosa gerou a solução ótima). Suponha que $\delta \neq \delta^*$. Seja k o índice do primeiro 0 que aparece em δ. Sabemos que se $\delta \neq \delta^*$, então existe pelo menos um $x_a > y_a$ para $a < k$ e um $x_b < y_b$ para $b \geq k$. Ou seja, a solução ótima inclui pelo menos um item (ou parte dele) e a solução gulosa não (e vice-versa).
Considere então a transformação de δ^* na tentativa de aproximar-se da solução δ, ou seja, remova da mochila uma parte do item b, com peso α; e insira na mochila a quantidade α do item a. Como $a < b \Rightarrow v_a/w_a > v_b/w_b$, isto aumenta o valor da mochila em:

$$\alpha\frac{v_a}{w_a} - \alpha\frac{v_b}{w_b} = \alpha\left(\frac{v_a}{w_a} - \frac{v_b}{w_b}\right) > 0.$$

O que contradiz a otimalidade da solução ótima.

3. [Valor: 1,5] 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) [Valor: 0,5] Selecione a atividade com menor tempo de início.

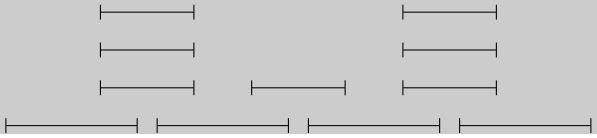
Solução:

(b) [Valor: 0,5] Selecione a atividade que requer o menor intervalo de tempo.

Solução:

(c) [Valor: 0,5] Selecione a atividade com o menor número de conflitos.

Solução:



4. [Valor: 1,0] 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.

   Solução:

   Considere duas tarefas com os seguintes tempos e prioridades: $t_1 = 1, p_1 = 2$, $t_2 = 2, p_2 = 2$. Se ordenarmos pela relação $p_i t_i$, escalonaríamos as tarefas na ordem $\{2, 1\}$, obtendo uma soma ponderada igual a $2 \cdot 2 + 3 \cdot 2 = 10$. O escalonamento ótimo no entanto é $\{1, 2\}$, levando a uma soma ponderada igual a $1 \cdot 2 + 3 \cdot 2 = 8$.

5. [Valor: 2,5] 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, $r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i})$.

```
Cut-Rod(p,n)
1 if n == 0
2    return 0
3 q = −∞
4 for i = 1 to n
5    q = max(q, p[i] + Cut-Rod(p,n-i))
6 return q
```

   (a) [Valor: 0,75] Escreva um algoritmo que usa a técnica de programação dinâmica para este problema.

   Solução:

```
//memo[0..n] foi inicializado com -1 em cada posição
//s[0..n] é um vetor que guardará a solução (os cortes)
```

```
   Cut-Rod-Memoizado(p,n)
 1 if memo[n] == -1
 2    if n == 0 memo[n] = 0
 3    else
 4       q = −∞
 5       for i = 1 to n
 6          aux = p[i] + Cut-Rod(p,n-i)
 7          if q < aux
 8             q = aux
 9             s[n] = i
10       memo[n] = q
11 return memo[n]
```

(b) [Valor: 0,75] Compare a complexidade de tempo dos dois algoritmos. Use notação assintótica e explique como você chegou à cada uma das complexidades.

Solução:

O algoritmo `Cut-Rod` tem custo $\Theta(2^n)$. Para chegar a esta conclusão, basta observar que o algoritmo tem a seguinte recorrência (resolução feita em aula):

$$T(n) = \sum_{j=0}^{n-1} T(j) + \Theta(1)$$

A análise do algoritmo `Cut-Rod-Memoizado` leva em conta o número de chamadas recursivas que são feitas e o tempo gasto em cada chamada. Note que existem $n + 1$ problemas distintos que serão memoizados. Uma vez que determinado valor já se encontra no memo, o custo total para este valor será de $\Theta(1)$. O tempo gasto em cada subproblema é $\Theta(n)$, considerando que os valores já se encontram disponíveis no memo (laço 5–9). Como existem $n + 1$ problemas distintos, o custo total de `Cut-Rod-Memoizado` será $\Theta(n^2)$.

(c) [Valor: 1,0] Escreva um algoritmo que imprime quais cortes fazem parte de uma solução ótima.

Solução:

```
Print-solution(s,n)
1 while n > 0
2    print s[n]
3    n = n - s[n]
```

Boa Prova!!!

Universidade Estadual de Maringá
Departamento de Informática – Ciência da Computação
6889 – Projeto e Análise de Algoritmos / Prof. Daniel Kikuti

Aluno(a): _____