UNIVERSIDADE FEDERAL DE VIÇOSA
DEPARTAMENTO DE INFORMÁTICA
INF332 – Projeto e Análise de Algoritmos

Exercícios

1. Dada uma sequência de n números ordenados tais que a diferença entre os termos consecutivos seja constante. Baseado na técnica Divisão e Conquista, escreva um algoritmo de complexidade $O(\log_2 n)$ para encontrar o elemento que falta. Suponha que o primeiro e o último elemento sejam sempre parte da sequência de entrada e o elemento ausente estará entre as posições 1 a n-1.
Por exemplo:
Sequência de entrada: [1, 7, 10, 13, 16]
Saída: O elemento ausente é 4.

Primeiro, Calcule a **diferença comum** (dc) da sequência:
Se a sequencia de entrada é Seq[1..n]: dc = (Seq[n] – Seq[1])/n

Supondo Seq[1..n] = Seq[ini..fim]

Determine a posição do meio: m = (ini + fim)/2
A ideia é verificar a diferença do **elemento do meio** com seu vizinho *esquerdo* e *direito*.

Verifique a diferença do elemento do meio com seu vizinho **esquerdo**:
Se Seq[m] – Seq[m-1] ≠ dc: o elemento ausente é Seq[m] - dc

Verifique a diferença do elemento do meio com seu vizinho **direito**:
Se Seq[m+1] – Seq[m] ≠ dc: o elemento ausente é Seq[m] + dc

Se o elemento ausente estiver no subarray esquerdo (Seq[m] - Seq[ini] ≠ (m - ini) * dc), recursivamente faça a busca no subarray esquerdo Seq[ini…m-1].

Caso contrário (se o elemento ausente estiver no subarray direito), recursivamente faça a busca no subarray direito Seq[m+1…fim]

O caso base acontece quando o subarray tem dois elementos e o elemento ausente deveria estar nesse subarray (Se fim – ini = =1 retornar Seq[fim] – dc)

Escreva o pseudocódigo do algoritmo.

2. Dado um array ordenado de tamanho n contendo elementos repetidos. Aplicando Divisão e Conquista escreva um algoritmo para encontrar a frequência de cada elemento sem percorrer todo o array.
Por exemplo:
Para o array de entrada: A = [2, 2, 2, 2, 4, 4, 5, 5, 8, 8, 8, 9]
Temos as seguintes frequências como saída: f(2) = 4, f(4) = 2, f(5) = 2, f(8) = 3 e f(9) = 1.

Suponha que o array é A[1..n] = A[ini..fim].

O caso base verifica se o último elemento do array é o mesmo que seu primeiro elemento.
Se A[ini] == A[fim], isso implica que todos os itens do subarray são iguais (já que o array é ordenado) e atualizamos a contagem de elementos pelo número total de elementos no subarray: freq[A[ini]] = freq[A[ini]] + (fim – ini + 1)
Senão, a contagem é feita recursivamente em A[ini, m] e A[m+1, fim], onde m = (ini+fim)/2.

Complexidade pior caso: O(nlog n). Quando todos os elementos do array são diferentes.
Melhor caso: O(1). Quando todos os elementos do array são iguais.

Escreva o pseudocódigo do algoritmo.

3. Dada uma string S = [c1, c2, ...,cn] de *n* caracteres. Alguns caracteres devem ser removidos para que a string se converta em um palíndromo. Encontre o número mínimo de remoções para convertê-la em um palíndromo.
Exemplo: S = [A C R A D R A M]
A C **R A** D **R A** M : Removendo C D e M obtemos [A R A R A] que é palíndromo.
A C **R** A **D R A** M : Ou removendo C A e M obtemos [A R D R A] que é palíndromo.
Escreva um algoritmo baseado em Programação Dinâmica para resolver o problema.

Solução:

Seja F[i, j] o número mínimo de remoções para transformar [ci,...,cj] em palíndromo.
Se i = j, F[i, i] = 0 (não há remoções)
Se i > j, F[i, j] = 0 (não há remoções)

Se i < j:
F[i, j] = F[i+1, j-1] se ci = cj. [c_i, $\mathbf{c_{i+1}}$ **,...,** $\mathbf{c_{j-1}}$, c_j] => [$\mathbf{c_{i+1}}$ **,...,** $\mathbf{c_{j-1}}$]
Se os extremos são iguais, considera o subproblema sem considerar os extremos.

F[i, j] = 1 + min{F[i+1, j] , F[i, j-1]}, se ci ≠ cj. [c_i, $\mathbf{c_{i+1}}$ **,...,** $\mathbf{c_{j-1}}$, $\mathbf{c_j}$] ou [$\mathbf{c_i}$, $\mathbf{c_{i+1}}$ **,...,** $\mathbf{c_{j-1}}$, c_j]
Pode apagar o primeiro ou último caractere. A remoção é feita de tal maneira que a substring (subproblema) tenha o menor número de remoções

Para o exemplo: S = [A C R A D R A M]

F[1, 8]: **A** C **R A** D **R A** M (para encontrar a solução final, é **melhor** remover M)
F[1, 7]: **A** C **R A** D **R A** (como o primeiro e ultimo são iguais, a solução é obtida de C **R A** D **R**)
F[2, 6]: C **R A** D **R** (aqui é **melhor** remover C)
F[3, 6]: **R A** D **R** (como o primeiro e ultimo são iguais, a solução é obtida de **A** D**)**
F[4, 5]: **A** D (aqui pode ser removido **A** ou **D**. Remova D).

Para determinar as melhoras escolhas, os cálculos devem ser feito de forma ascendente, ou seja calcular F[1, 2], F[2, 3] .... F[1, 8] (ou seja montar toda a tabela).

4. Considere um conjunto de n atividades a serem realizadas. Para cada atividade i tem-se, um prazo para ser realizado (*deadline*) $d_i \geq 0$, e um lucro $p_i > 0$. O lucro pela realização de uma atividade será obtido somente se a atividade é concluída dentro do seu prazo. Suponha que, para realizar qualquer atividade gasta-se o mesmo tempo (uma unidade de tempo), e somente uma atividade deve ser realizado por vez. O problema consiste em selecionar um subconjunto de atividades que possam ser realizadas dentro dos seus prazos e que o lucro total (soma dos lucros das atividades selecionadas) seja o máximo. Também deve ser determinada a ordem de execução das atividades selecionadas. Uma solução será inviável se uma atividade escolhida finaliza após seu prazo. A execução das atividades deve ser inicializada num tempo zero (ou seja, a partir desse tempo inicial zero serão contabilizados os prazos para a execução das atividades).

Exemplo: $n = 6$ atividades:

Atividades	**1**	**2**	**3**	**4**	**5**	**6**
Prazos (d_i)	2	3	1	2	2	4
Lucros (p_i)	100	40	80	20	60	10

Uma solução viável é escolher as atividades 1, 5, 2 e 6 (executar nesta ordem). Suponha que a unidade de tempo seja hora. Note que a atividade 1 gastará 1h para ser executada, e como tem um prazo de 2, não estará atrasada. A atividade 5 finalizará uma hora depois da atividade 1, dentro do seu prazo (seu prazo é 2). A atividade 2 finalizará após 3h (dentro do seu prazo que é 3). A atividade 6 finalizará após 4h, dentro do seu prazo (seu prazo é 4). Como todas as atividades finalizam dentro dos seus respectivos prazos, o lucro total será: $p_1 + p_5 + p_2 + p_6 = 100 + 60 + 40 + 10 = 210$. É a solução ótima?

a) Escreva uma estratégia gulosa para resolver o problema (para selecionar as atividades, determinar a ordem de execução e maximizar o lucro total).
b) Aplique a estratégia gulosa para resolver o exemplo numérico mostrado acima. A estratégia sempre determinará a solução ótima?
c) Escreva o pseudocódigo do algoritmo guloso e calcule sua complexidade.

Estratégia gulosa:

d_{max} = Determinar o maior prazo (que será o número máximo de atividades a serem executadas).
Suponha que $d_{max} <= n$.
Será construída uma sequência/ordem de execução com d_{max} atividades (posições).

Ordene as atividades em ordem decrescente dos seus lucros p_i.
Escolha a primeira atividade da lista ordenada, e insira-a na posição correta de acordo a seu prazo de execução (ex. se seu prazo é 1, será inserida na primeira posição; se seu prazo é d_{max}, será inserida na última posição).
Iniciando da posição i = 2, da lista ordenada:
    Verifique se a atividade i é pode ser executada em alguma posição da sequência (respeitando seu prazo).
    Se não é possível, descartar esta atividade.
    Repita até que todas as d_{max} posições sejam preenchidas.
Somar o lucro das atividades escolhidas (inseridas na sequencia).

Para o exemplo acima:
$d_{max} = 4$ (quatro atividades serão escolhidas).
Atividades ordenadas: 1, 3, 5, 2, 4, 6

Sequência vazia: ___ ,____, ____, ____
Escolhe atividade 1 (prazo 2) e insere na segunda posição.
Sequência: ___ , 1 , ____, ___
Escolhe atividade 3 (prazo 1), insere na primeira posição.
Sequência: 3 , 1 , ____, ___
Escolhe atividade 5 (prazo 2). Descarta (posições 2 e 1 ocupados)
Escolhe atividade 2 (prazo 3). insere na terceira posição
Sequência: 3 , 1 , 2, __
Escolhe atividade 4 (prazo 2). Descarta (posições 2 e 1 ocupados)
Escolhe atividade 6 (prazo 43). insere na quarta posição
Sequência: **3 , 1 , 2 , 6**
Lucro total = 80 + 100 + 40 + 10 = 230

Escreva o pseudocódigo do algoritmo guloso.

Complexidade:
Ordenar atividades: O(n log n)
Determinar d_{max}: O(n)
Inserir as atividades na sequência de execução: O($n.d_{max}$).
Complexidade final: O(max{nlog n, nd_{max}}).

5. Considere uma matriz de tamanho com *n*x*n* números. Deseja-se escolher *n* números que estejam em linhas e colunas diferentes, cuja soma seja o máximo possível. Por exemplo, na tabela abaixo podem ser escolhidos 180 + 110 + 140 + 30 + 10 = 470 (é o valor máximo possível?).

	1	2	3	4	5
1	50	40	100	60	**180**
2	20	30	**110**	95	65
3	**140**	120	25	35	90
4	160	10	200	**30**	85
5	45	**10**	55	60	170

a) Escreva uma estratégia gulosa para resolver o problema.
b) Aplique sua estratégia para resolver a instância da tabela acima. A estratégia sempre determinará a solução ótima? Caso não, apresente um contraexemplo.

Estratégia gulosa:
1) Escolha o maior número da matriz (*mv*). Descarte os elementos que estão na linha e na coluna de *mv*.
2) Se ainda não foram escolhidos os *n* números, repita o passo anterior para escolher outro número (números descartado não podem ser escolhidos).

6. Seja uma sequência de *n* números inteiros S[1..n] = {s_1, s_2,..., s_n}. Deseja-se determinar uma subsequência {s_i...s_j} de S (não necessariamente consecutiva) formada pelo maior número de elementos ordenados de forma estritamente crescente, ou seja, $s_i <...< s_j$.

Por exemplo, para S = {5, 2, 8, 6, 3, 6, 9, 4}, algumas subsequências crescentes são: {5, 6}, {5, 8, 9}, {2, 8, 9}. Não são consideradas subsequências: {3, 5} e {3, 6, 8}. A melhor solução (solução ótima) é a subsequência {2, 3, 6, 9}, formada por 4 elementos. Resolver o problema utilizando PD, da seguinte maneira:

a) Defina a função de recorrência F[j] para calcular o número de elementos de uma subsequência crescente de S[1..j]. O maior valor contido em F deve ser o valor da solução ótima.
b) Escreva um algoritmo de PD que recebe a sequência S[1..n], e monta a tabela F[] utilizando a função de recorrência, e retorna o tamanho da maior subsequência crescente contida em S[1..n]. Calcule a complexidade do algoritmo.
c) Escreva um algoritmo para imprimir os elementos da maior subsequência crescente contida em S[1..n]. O algoritmo deve usar a tabela F[].
d) Aplique a relação de recorrência, de forma *bottom-up*, para encontrar a maior subsequência crescente de S = {50, 20, 80, 60, 30, 60, 90, 40}. Mostre, passo a passo, todos os cálculos realizados para a construção da tabela.

Seja F[j] o tamanho da maior subsequência crescente contida em S[1..j], para j≥1, sendo que a subsequência finaliza S[j].

$F[\,j\,] = 1 + \max\,\{F[\,i\,]: \forall i<j \text{ e } S_i < S_j\}, \qquad j = 1,...n$

$F[\,j\,] = 1$, se não existe $i < j$, tal que $S_i < S_j$.

Solucao ótima: Escolher o maior $F[j]$, $j = 1,...n$

Para o exemplo:

S	50	20	80	60	30	60	90	40
	1	2	3	4	5	6	7	8

F[1] = 1 + max{ } = 1
F[2] = 1+ max{ } = 1
F[3] = 1 + max{F[1], F[2]} = 1 + max{1, 1} = 2
F[4] = 1 + max{F[1], F[2]} = 1 + max{1, 1} = 2
F[5] = 1 + max{F[2] } = 1 + max{1} = 2
F[6] = 1 + max{F[1], F[2], F[5]} = 1 + max{1, 1, 2} = 3
F[7] = 1 + max{F[1], F[2], F[3], F[4], F[5], F[6] } = 1 + max{1, 1, 2, 2, 2, 3} = 4
F[8] = 1 + max{F[2], F[5]} = 1 + max{1, 2} = 3

Note que o maior valor encontrado é F[7] = 4. Ou seja, a subsequência crescente ótima tem 4 elementos finaliza em 90.
No calculo de F[7] o valor máximo foi F[6]. Ou seja a próxima subsequência finaliza em 60.
No calculo de F[6] o valor máximo foi F[5]. Ou seja a próxima subsequência finaliza em 30.
No calculo de F[5] o valor máximo foi F[2]. Ou seja a próxima subsequência finaliza em 20.
Solução: {20, 30, 60, 90}

Escreva o pseudocódigo do algoritmo de programação dinâmica.