Análise de Algoritmos - 2020.1

Exame 2 - Gabarito

1 - (a)

O algoritmo é o mesmo dado em aula.

O caminho mais curto tem a seguinte ordem de visitação:

$1 \to 5 \to 3 \to 2 \to 6 \to 7 \to 4$. Peso total do caminho: 10

1 - (b)

Seja um grafo bipartido completo definido como $K_{n,m}$, onde n e m são os números de vértices dos dois conjuntos. Considera-se que a raiz de uma árvore possui altura 0 e $n, m \geq 2$.

Resposta para BFS: A altura é 2 para qualquer $n, m \geq 2$. (Lembramos que a altura de um arvore é o numero de arestas entre a raiz é a folha mais distante.)

Resposta para DFS: Considerando $n = m$, a altura da árvore será $2n - 1$.

(Bônus 0.5 pt) Se a resposta considera $m > n$ e a DFS começa pelo lado com maio numero de vertices, então a altura da árvore será $2n$. Caso a DFS inicie pelo outro lado, então a altura será $2n - 1$.

2 -

a) **Falso**. O problema de decisão associado ao problema da mochila integral é NP-completo.
b) **Falso**. A orientação da redução precisaria ser invertida para mostrar que $X \in$ NP-completo.
c) **Falso**. Uma vez que P $\subset$ NP, é impossível um problema em P não estar em NP.
d) **Falso**. Bellman-Ford também funciona com pesos negativos enquanto não existir ciclo de custo negativo. Neste ultimo caso, ele até poderia ser adaptado para detectar este tipo de situação.
e) **Verdadeiro**.
f) **Falso**. Algoritmos gulosos escolhem a **melhor decisão local** em relação a um critério de otimização. Estes algoritmos encontram a solução ótima para determinados problemas de otimimização.
g) **Verdadeiro**.
h) **Verdadeiro**.
i) **Verdadeiro**.
j) **Verdadeiro**.

3 - **Resposta trivial**: Usar diretamente o algoritmo de Prim, Kruskal ou Reverse-Delete. Pontuação: 0.5pts.

Resposta eficiente: Primeiramente, obtenha todas as componentes conectadas com arestas de peso 0. Isso pode ser resolvido por meio de uma BFS considerando so as arestas de peso 0 (com complexidade $O(k + |V|)$). Depois, seja k o número de componentes conectadas encontradas no passo anterior, basta adicionar $k - 1$ arestas de peso 1 de um vertice qualquer da primeira componente para um vertice qualquer de cada um das outras. Ao final, o grafo resultante será uma árvore geradora mínima de G. Complexidade do algoritmo: $O(k + |V|)$.

4 - a) (CANCELADA) Veja ao final do documento a redução para o problema de scheduling com lucros.

b)

Essa questão é similar ao problema do *weighted interval scheduling* visto em aula, com a adição da restrição de capacidade.

Nesse caso, $OPT(j, q)$ corresponde ao valor da solução ótima para o problema restrito às tarefas de $1, 2, \ldots, j$ sujeito a capacidade q. Considere $p(j)$ como a tarefa de maior indíce $i < j$ que é compatível com j (considerando as tarefas ordenadas por ordem de data de termino).

Caso 1: OPT seleciona a tarefa j:

- Acrescenta uma unidade no objetivo;
- Não pode usar tarefas incompatíveis $(p(j) + 1, p(j) + 2, \ldots, j - 1)$;
- Deve incluir a solução ótima do problema que inclui as tarefas compatíveis $1, 2, \ldots, p(j)$ com capacidade $q - q_j$.

Caso 2: OPT não seleciona a tarefa j:

- Deve incluir a solução ótima do problema que consiste nas tarefas restantes $1, 2, \ldots, j - 1$ com capacidade q.

Relação de recorrência:

$$OPT(j, q) = \begin{cases} 0 & \text{se } j = 0 \\ OPT(j-1, q) & \text{se } q_j > q \\ max\{OPT(j-1, q), 1 + OPT(p(j), q - q_j)\} & \text{caso contrário} \end{cases}$$

c)

Algorithm 1: Problema de *scheduling* com lucros

```
for q=0 até Q do
    OPT[0, q] = 0
Compute todos os p(1), p(2), ..., p(n);
for j=1 até n do
    for q=1 até Q do
        if q_j > q then
            OPT[j, q] = OPT[j − 1, q]
        else
            OPT[j, q] = max{OPT[j − 1, q], 1 + OPT[p(j), q − q_j]}
Return OPT[n, Q];
```

d) Complexidade: $O(nQ)$. Importante: esse algoritmo possui complexidade pseudopolinomial, sendo que a valor do parametro Q cresce de modo exponencial em relação ao tamanho dos dados de entrada.

5 -

a)

Algoritmo 1: $T(n) = 2T(n-1) + O(1)$. Complexidade: $O(2^n)$
Algoritmo 2: $T(n) = 4T(n/2) + O(1)$. Complexidade: $O(n^2)$
Algoritmo 3: $T(n) = T(n/2) + O(n^2)$. Complexidade: $O(n^2)$

b)
O Algoritmo 1.

Resposta para 4a) considerando o problema de *scheduling* (com lucros).

O problema da mochila integral (*integer knapsack problem*) considera uma lista de itens com lucro p_i e peso q_i e uma mochila com capacidade Q. O objetivo desse problema consiste em selecionar um subconjunto de itens tal que a soma de seus pesos é no máximo Q e a soma dos lucros dos itens é a máxima possível.

O problema de *scheduling* (com lucros) considera uma lista de tarefas com lucro p_i, peso q_i, data de início e_i e data de término l_i. O peso total das tarefas não pode exceder a capacidade Q. O objetivo é selecionar um subconjunto de tarefas **compatíveis** que respeita a capacidade máxima Q e que o lucro total seja o máximo possível.

A redução:

Podemos observar que o problema de *scheduling* é uma generalização do problema da mochila integral, com a adição da restrição de compatibilidade (como definido em e_i e l_i).

Partindo de uma instância do problema da mochila, podemos criar uma instância equivalente para o problema de *scheduling* da seguinte forma:

Problema da Mochila Integral	$\Leftrightarrow$	Problema de *Scheduling* com lucros
item i		tarefa i
lucro i		lucro i
peso i		peso i
		$e_i = i$
		$l_i = i + 0.0001$

Dessa forma, todas as tarefas no problema de *scheduling* são mutuamente compatíveis. Isto é, não existem intervalos que se sobrepõem. Por questão de simplicidade, usamos 0.0001 em l_i para representar um intervalo positivo ($l_i > e_i$) de i mas que não é maior que e_{i+1}.

Note que uma solução ótima do problema de *scheduling* para essa instância dá uma solução para o problema da mochila integral.