UFF - Análise e Projeto de Algoritmos - Prova 2 - 1/2015

1. Considere um tabuleiro $n \times n$ onde cada casa possui um custo associado $c(i, j)$ (inteiro positivo), onde o índice i indica a linha e o índice j a coluna. (A linha 1 é a primeira de baixo para cima, e a coluna 1 é a mais à esquerda.) Um caminho neste tabuleiro consiste de uma sequência de casas $(1, j_1), (2, j_2), \ldots, (r, j_r)$ tais que, para cada valor de $k \in \{1, \ldots, r-1\}$, vale que $j_{k+1} \in \{j_{k-1}, j_{k+1}\}$. Em outras palavras, os caminhos sempre partem da primeira linha e vão subindo linha a linha, sendo que a próxima casa está em uma mesma diagonal que a casa anterior. Exemplo: iniciando na casa $(1, 3)$, a próxima casa do caminho pode ser $(2, 2)$ ou $(2, 4)$.

   (a) (1.0) Seja $q(i, j)$ o custo mínimo de um caminho que parte da primeira linha e chega à casa (i, j). Escreva equações de recorrência para determinar $q(i, j)$. (Obs: o custo de um caminho é a soma dos custos das casas que compõem o caminho.)

   $q(i, j) = c(i, j) + \min(q(i-1, j-1,), q(i-1, j+1))\ , \qquad i > 1 \text{ e } 1 \leq j \leq n$
   $q(1, j) = c(1, j), \qquad 1 \leq j \leq n.$

   (b) (2.0) Escreva um algoritmo de programação dinâmica que determine o custo mínimo de um caminho que parte da primeira linha e chega à linha n. Determine sua complexidade.

```
Progdin(c, n)
For j=1 to n
    q(1, j) = c(1, j)
For i=2 to n
    For j=1 to n
        q(i, j) = ∞
        if (j > 1) then
            if (q(i − 1, j − 1) < q(i, j)) then
                q(i, j) = q(i − 1, j − 1)
        if (j < n) then
            if (q(i − 1, j + 1) < q(i, j)) then
                q(i, j) = q(i − 1, j + 1)
        q(i, j) = q(i, j) + c(i, j)
min = q(n, 1)
for j=2 to n
    if (q(n, j) < min) then
        min = q(n, j)
return min
```

   Complexidade é $O(n^2)$, já que a primeira parte (caso base) e última parte (custo mínimo da linha n) tem complexidade $\Theta(n)$, a segunda parte tem complexidade $\Theta(n^2)$.

   (c) (2.0) Escreva um algoritmo de backtracking que enumera todos os possíveis caminhos que partem da primeira linha e chegam à linha n. Determine sua complexidade.

```
Backtrack(c, n, i, j, x)
if (i == n) then
    print(x)
else
    for z = (max(1, j − 1)) to (min(n, j + 1)) com incremento z = z + 2
```

$x[i+1] = z$
Backtrack(c, n, i+1, z, x)

Chamada(c, n)
for j=1 to n
    x[1] = j
    Backtrack(c, n, 1, j, x)

A complexidade é n vezes a complexidade de Backtrack, já que chamamos n vezes na função chamada.

Se verificarmos uma chamada do Backtrack no nível um da árvore de busca temos uma opção, no nível dois temos 2 opções, no nível 3 temos 4 opções e no nível i temos $2^{(i-1)}$. Claro que eliminamos algumas desses opções já que quando $j < 1$ ou $j > n$ podemos eliminar essas soluções. Mas no pior caso podemos falar que a complexidade do algoritmo é exponencial $O(2^n)$.

2. Considere os dois problemas a seguir:

Caminho Longo em grafos :
Entrada: Um grafo G e um inteiro positivo k.
Questão: Existe em G um caminho simples com pelo menos k arestas?

Caminho Hamiltoniano :
Entrada: Um grafo G.
Questão: Existe em G um caminho simples que passa por todos os vértices?

(a) (1.0) Escreva a versão de Localização e de Otimização do problema Caminho Longo em grafos.
Localização:
Entrada: Um grafo G e um inteiro positivo k.
Questão: Localizar em G um caminho simples com pelo menos k arestas

Otimização:
Entrada: Um grafo G.
Questão: Localizar em G o caminho simples com o maior número de arestas?

(b) (1.0) Mostre que o problema Caminho Longo em grafos pertence à classe NP.
A justificativa da resposta SIM é feita em tempo polinomial, já que para isso basta percorrer o caminho e verificar se é simples e tem pelo menos k arestas.

(c) (1.5) Mostre que existe uma redução polinomial do problema Caminho Hamiltoniano para o problema Caminho Longo em grafos.
A transformação do problema caminho Hamiltoniano no de caminho longo em grafo é direta. O problema Hamiltoniano pode ser resolvido como o caminho longo em G com pelo menos $|V| - 1$ arestas.

(d) (1.0) Dê a definição de "problema NP-completo".

NP-completo é o subconjunto dos problemas de decisão em NP de tal modo que todo problema em NP se pode reduzir (em tempo polinomial), a um dos problemas NP-completo.

(e) (1.0) Sabe-se que o problema de Caminho Hamiltoniano é NP-completo. Pergunta-se: o problema Caminho Longo em grafos é NP-completo? Por que?

Sim. Para mostrar que um problema é NP-completo podemos mostrar que ele pertence a NP (feito numa questão acima) e que existe um problema NP-completo que é polinomialmente transformável nele (também feito numa outra questão acima).