2ª Avaliação de Introdução à Análise de Algoritmos
Prof. Glauber Cintra

Você deve enviar essa avaliação pelo Google Classroom até o dia 14/set/2020 às 15:30h.

1) (**2 pontos**) Escreva uma função baseada em *divisão-e-conquista* que receba números os *a* e *b* (*b* natural) e devolva a^b. Determine a complexidade de tempo e de espaço da sua função e prove sua corretude.

Algoritmo Expo_DC
Entrada: a e b naturais
Saída: a^b

    Se b = 0
        Devolva 1 e pare
    aux = Expo_DC(a, ⌊b/2⌋)
    Se b%2 = 0
        Devolva aux*aux
    Se não
        Devolva aux*aux*a

Complexidade temporal e espacial

$T(b) = T(\lfloor b/2 \rfloor) + c$
$T(0) = 1$

$T(b) \in \Theta(\log n)$

Façamos por indução em b. Base n = 0, a 0 = 1. Suponha que n é um número natural. Para quaisquer números naturais a e b, o algoritmo calcula corretamente a n. (HI)

TEOREMA: O algoritmo Expo_DC é correto.
Prova: Se b = 0, o algoritmo Expo_DC retorna 1, o que é correto. Como b é natural, ele só poderá assumir valores positivos e isso ficará implicito na prova de corretude. Agora, vamos supor que b é par, o algoritmo retornará a^b, pois: $a^{b/2} * a^{b/2} = a^b$, o que é verdadeiro. Agora, se b é impar, o algoritmo retornará: $a^{\lfloor b/2 \rfloor} * a^{\lfloor b/2 \rfloor} * a = a^b$, o que também é verdadeiro.

2) (**2 pontos**) Usando o algoritmo de programação dinâmica vista em sala de aula, encontre uma subsequência crescente máxima da sequência (4, 1, 6, 2, 4, 3, 5, 2).

C = [1, 1, 2, 2, 3, 3, 4, 3] e o maior comprimento é 4.
Uma possível subsequência crescente máxima é [1, 2, 4, 5].

2) (**2 pontos**) O problema da *3-partição* consiste em, dado um conjunto C, determinar se é possível particionar C em 3 subconjuntos de tal forma que a soma dos elementos de cada subconjunto seja igual. Por exemplo, para o conjunto C = {1, 2, 3, 4, 5, 6} a resposta é *Sim*, pois podemos particionar C nos subconjuntos {1, 6}, {2, 5} e {3, 4} cuja soma dos elementos é 7. Já para o conjunto C = {1, 2, 3, 4, 5, 12} a resposta é *Não*, pois não é possível particionar C em 3 subconjuntos cuja soma dos elementos seja 9. Escreva um algoritmo baseado em *enumeração explícita* que resolva o problema da 3-partição. Determine a complexidade de tempo do seu algoritmo.

Entrada: sequência c, a partir do 0, e o tamanho da sequência
Saída: Sim, se c pode ser repartido em 3 subsequências com mesma soma.
Não, caso contrário.
Algoritmo três_partição(c, tam_c):
sum = 0

```
inteiros = verdadeiro

Para i de 0 até tam_c
        sum = sum+c[i]
        Se c[i] não for interior
                inteiros = falso

//verificando a existencia de solucoes de maneira trivial
Se sum % 3 != 0 && inteiros
        retorne Não e pare

//criando lista de vetores de bits para o armazenamento de subsequências de interesse e que
possuam soma igual a sum/3 para serem armazenadas na lista de vetores de bits

subseq_max = 2^tam_c

Para i de 0 até subseq_max
        sum_sub = 0

        Para j de 0 até tam_c
                Se (i&1(<<j) != 0
                        sum_sub = sum_sub + c[j]

        Se sum_sub = sum/3
                i++ em subseq

//verificando, entre as subsequências listadas, se existe 3 que não possuam termos em comum.
Caso exista, a união delas incluem todos os termos de c

Para i de 0 até subseq.size()
        Para j de i+1 até subseq.size()
                Para k de j+1 até subseq.size()
                        Se (subseq[i] & subseq[j]) = 0 &&
                          (subseq[j] & subseq[k]) = 0 &&
                          (subseq[i] & subseq[k]) = 0)
                                Retorne Sim e pare
Retorne Não
```

Complexidade temporal: Θ(nC * 2ˆnC) já que temos como região crítica a linha **Se (i&1(<<j) != 0**, pois é a região o qual se calcula a soma de cada subsequencia.

3) (**2 pontos**) Seja $a_1, a_2, ..., a_n$ uma sequência de números. Dizemos que $c_1a_1 + c_2a_2 + ... + c_na_n$ é uma *combinação linear inteira* de $a_1, a_2, ..., a_n$ se $c_1, c_2, ..., c_n$ são números naturais. Escreva uma função baseada em *enumeração implícita* que receba uma sequência de números e um valor M e devolva *verdadeiro* se existe uma combinação linear inteira da sequência cujo valor seja exatamente M; *falso*, caso contrário. Por exemplo, para a sequência 5, 8, 3 e M = 11, a função deve devolver *verdadeiro*, pois a 1.5 + 0.8 + 2.3 tem valor 11. Para M = 7, a função deve devolver *falso*. (*sugestão*: adapte o algoritmo de *branch-and-bound* para o problema da mochila zero-um).

Algoritmo Comb_Linear_BB
Entrada: Uma sequência de números a $(a_1, a_2, \dots, a_n)$ e um valor M
Saída: Verdadeiro, se existir uma combinação linear inteira da sequência cujo valor seja igual M; Falso, caso contrário.

```
        sequência vazia b

        Para i de 1 até n
                adiciona a[i] em b,  ⌊ M/a[i ]⌋ vezes
```

mochila = mochila01_EI(M, b, b)
Se mochila = M
    retorne verdadeiro e pare
retorne falso

Algoritmo mochila01_EI
Entrada: A capacidade da mochila c, vetor pesos p e vetor valores v. Os dois vetores tem tamanho n e são indexados a partir de 1.
Saída: Soma da solução ótima

valor relativo dos itens em ordem decrescente
criar os vetores de bits X inicialmente zerado e X'

capacidade_restante = C, K = 0, M = 0

Para i = k + 1 até n
    Se P[i] <= capacidade_restante
        X[i] = 1
        capacidade_restante = capacidade_restantet - P[i]

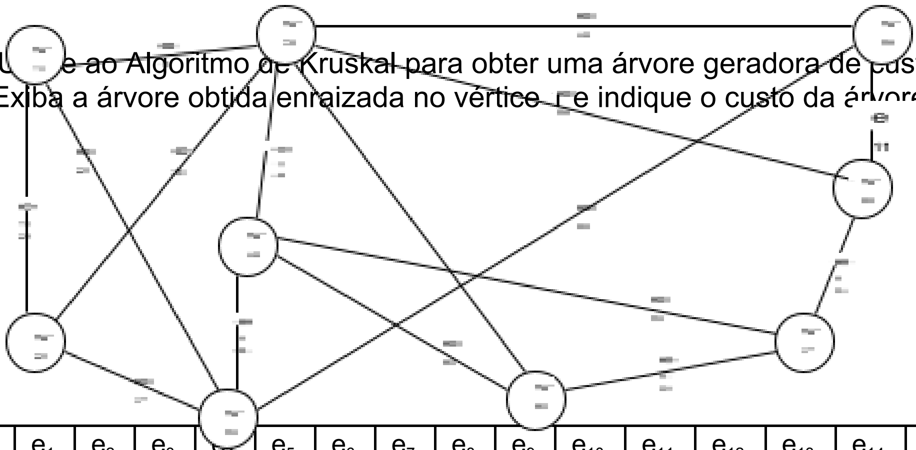
Se $\sum_{i=1}^{n} X[i] * V[i] > M$
    X'= X
    M= $\sum_{i=1}^{n} X[i] * V[i]$

K = max(-1, {i | i<n e X[i] = 1 e $\sum_{j=1}^{i-1} X[j] * V[j] + (C - \sum_{j=1}^{i-1} X[j] * P[j]) * V[i+1]/P[i+1] > M$})

Se K>0
    capacidade_restante = C - $\sum_{j=1}^{k-1} X[j] * P[j]$
    X[K] = 0
    Enquanto K>0

Retorne $\sum_{j=1}^{n} X'[i] * V[j]$

5) (**2 pontos**) U[illegible]e ao Algoritmo de Kruskal para obter uma árvore geradora de custo mínimo do grafo abaixo. Exiba a árvore obtida enraizada no vértice [illegible] e indique o custo da árvore.



Aresta	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}	e_{13}	e_{14}	e_{15}	e_{16}
Custo	11	7	12	10	1	14	8	4	3	15	9	2	6	5	16	13

Organizando o custo em ordem crescente.

Aresta	e5	e12	e9	e8	e14	e13	e2	e7	e11	e4	e1	e3	e16	e6	e10	e15
Custo	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Arestas: e5, e12, e9, e8, e14, e2, e11, e6 = 1+2+3+4+5+7+9+14 = 45