**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 *ab*. Determine a complexidade de tempo e de espaço da sua função e prove sua corretude.

Algoritmo Expoente

**Entrada**: Um número a e um número b ∈ naturais

**Saída**: *ab*

Se b = 0

Devolva 1

Se não

aux = Expoente(a , )

Se b for par

Devolva aux \* aux

Se não

Devolva aux \* aux \* a

Corretude:

**Teorema**: o Algoritmo Pot5 é correto.

**Prova**: Por indução em *b*. Base: *x* = 0, qualquer número elevado a 0 é igual a 1 e o algoritmo devolve 1. Suponha agora que Expoente(a , ) devolva ab/2 (H.I.). Ao receber x > 0, se *x* for par o Expoente devolve:

aux \* aux = Expoente(a , ) \* Expoente(a , ) = \* = ab/2 \* ab/2  = ab

aux \* aux \* a = Expoente(a , ) \* Expoente(a , ) \* a = \* \* a = a(b-1)/2 \* a(b-1)/2 \* a= ab

Com isso podemos ver que o algoritmo é correto.

Complexidade temporal e espacial:

T(x) = T() + c

T(0) = c

Supondo x = 2^k, ∀ k ≠ 0 e ∈ N ⇒ **⎣**x/2**⎦** = **⎣**2(k-1)**⎦** = 2(k-1). Com isso podemos eliminar a função piso pois ela não será utilizada de acordo com a nossa suposição já que sempre resultará em um número natural.

Utilizando o método da soma:

Equação 0: T(x) = T(x/2) + c

Equação 1: T(x/2) = T(x/4) + c

Equação 2: T(x/4) = T(x/8) + c

...

Equação k – 1: T(x/2k-1) = T(x/2k) + c

Equação k : T(x/2k) = c

T(x) = k\*c ; k = => T(x) = . Ou seja o tempo e o espaço requerido pelo algoritmo Expoente é O().

1. (**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).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **i** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| **si** | 4 | 1 | 6 | 2 | 4 | 3 | 5 | 2 |
| **ci** | 1 | 1 | 2 | 2 | 3 | 3 | 4 | 3 |

SCM = 1, 2, 4, 5.

1. (**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.

Algoritmo 3-partição

Entrada: Um conjunto C com n números. C = {x1 , x2 , x3 , ... , xn }

Saída: 3 subconjuntos onde o resultado da soma dos números de cada subconjunto é a mesma.

Soma = 0

Para j = 0 até n – 1

Soma = Soma + Cj

Se Soma % 3 = 0 *E* n ≥ 3 / / Verifica se a soma dos elementos de C é divisível por 3

Para i = 0 até 2n – 1

S = Subconjunto( C , i ) / / Si receberá um subconjunto de C

Para f = 0 até (Tamanho de S) / / podendo chegar até n onde seria o próprio conjunto C

Soma\_S = Soma\_S + Sf

Se Soma\_S = (Soma/3) E S ≠ Resposta/ / Verifica se é um subconjunto valido e diferente dos outros já validos

Resposta= S / / Resposta recebe o subconjunto

Devolva Resposta

Se não

Devolva não é possível particionar C em 3 subconjuntos cujas somas dos elementos sejam iguais

**Procedimento Subconjunto**

Entrada: Um conjunto C com n números. C = {x1 , x2 , x3 , ... , xn }, um valor i.

Saída: Um subconjunto de C codificado pelos bits de i.

Para k = 1 até n

Se *i* % 2 = 1

subk = xk

i = i / 2

Devolva sub

A complexidade temporal do algoritmo é de O(n2n) por conta que a instrução critica leva tempo 2n e a função Subconjunto leva tempo linear n.

1. (**2 pontos**) Seja a1, a2, ..., an uma sequência de números. Dizemos que c1a1 + c2a2 + ... + cnan é uma *combinação linear inteira* de a1, a2, ..., an se c1, c2, ..., cn 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 Combinação\_Linear

Entrada: Uma sequência S = {a1 , a2 , a3 , ... , an } e um valor M

Saída: Devolva verdadeiro se existir uma combinação linear, se não devolva falso

k=0

aux = M

Faça

Para i = k+1 até n

c[i] = c / a[i]

aux = aux - a[i] \* c[i]

Se aux = 0

Devolva Verdadeiro e pare

Se não

k = max(0, {j | (j < n e c[j] >=1 e aux - a[j] > a[j+1]) || (j = n) e c[j] >= 1 e c[j] < a[1] \* c[1]})

Se k > 0

c[k] = c[k] - 1

aux = aux + a[k]

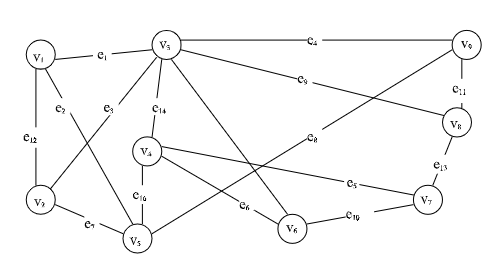
Se k = n

k = 0

Enquanto k > 0

Devolva Falso

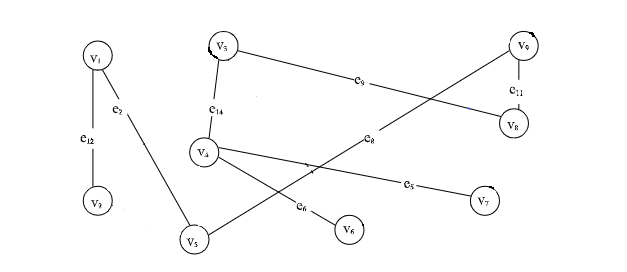
1. (**2 pontos**) Utilize ao Algoritmo de Kruskal para obter uma árvore geradora de custo mínimo do grafo abaixo. Exiba a árvore obtida enraizada no vértice 1 e indique o custo da árvore.



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

Organizando 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 |



Essa é a árvore geradora de custo mínimo e o custo dessa árvore é de 45.