Lista 1 - AEDs III 2021-2/UFSJ

Prof. Alexandre Bittencourt Pigozzo.
Valor da lista: 30 pontos.
Data de entrega: **26/10/2021**.
A lista 1 pode ser feita **individualmente ou em dupla**. Entregar através do portal didático todos os exercícios e códigos compactados em um arquivo com o nome do aluno ou da dupla.

Obs: Para os exercícios que tiverem implementação, favor implementar nas linguagens de programação mais tradicionais e conhecidas como C, C++, Python, Java, Javascript, etc. Caso você queira implementar em uma linguagem não tão conhecida, favor enviar todas as bibliotecas e instruções de instalação para executar o programa no Ubuntu 20.04.

Exercício 1 (1,0)

Para a BuscaSequencial, faça a análise de complexidade para o caso no qual a chave é encontrada na última posição do vetor. Explique a complexidade obtida e se ela muda com o tamanho da entrada.

Exercício 2 (1,0)

No algoritmo de busca sequencial, quais instruções dependem da entrada e quais não dependem? Ou seja, quais instruções são sempre executadas independente da entrada dada e quais instruções são executadas somente com determinadas entradas? Justifique.

Exercício 3 (1,0)

Quais as diferenças podemos perceber entre as complexidades dos algoritmos Máximo de um vetor e Busca sequencial? Explique.

Exercício 4 (1,0)

A complexidade do algoritmo máximo de um vetor depende de uma entrada específica de dados? Isto é, a complexidade depende do máximo estar na 1a, 2a, 3a posição? Ou na última posição?

Exercício 5 (1,0)

Refaça as análises dos algoritmos de BuscaSequencial e Máximo de um vetor utilizando somatórios e fazendo as análises separadas para o melhor caso e pior caso. Obs: Para cada algoritmo, você tem que definir uma ou mais operações mais significativas.

Exercício 6 (1,0)

Dada duas funções g(n) = $(n+1)^2$ e f(n) = n^2 mostre que as funções g(n) e f(n) dominam assintoticamente uma a outra.

Exercício 7 (1,0)

Mostre que g(n) = logb n é O(logc n) para quaisquer bases b e c inteiras com b > 1 e c > 1. Por exemplo, b pode ser 2 e c 10 ou b pode ser 10 e c 5 e, assim por diante.

Exercício 8 (1,5)

Calcule a complexidade do algoritmo SelectionSort dado abaixo. Considere como operação mais significativa a comparação < em arr[j] < arr[min_idx]. Faça a análise da complexidade para o melhor caso e para o pior caso. Utilize somatórios na sua resposta.

```
void swap(int *xp, int *yp) {
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void selectionSort(int arr[], int n) {
    int i, j, min_idx;

    // One by one move boundary of unsorted subarray
    for (i = 0; i < n-1; i++) {
        // Find the minimum element in unsorted array
        min_idx = i;
        for (j = i+1; j < n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;

        // Swap the found minimum element with the first element
        swap(&arr[min_idx], &arr[i]);
    }
}
```

Exercício 9 (1,5)

Calcule a complexidade do BubbleSort para o melhor e pior casos considerando a operação > em arr[j] > arr[j+1] como operação mais significativa. Utilize somatórios na sua resposta.

```c
void swap(int *xp, int *yp) {
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

// A function to implement bubble sort
void bubbleSort(int arr[], int n) {
    int i, j;
    for (i = 0; i < n-1; i++)
        // Last i elements are already in place
        for (j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1])
                swap(&arr[j], &arr[j+1]);
}
```

Exercício 10 (1,5)

Calcule a complexidade no melhor e pior caso para a função func2 dada abaixo. Considere como operação mais significativa a soma + em soma+= n. Utilize somatórios na sua resposta.

```c
int func2(int n) {
    int i, j, k, soma = 0;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            for (k = 0; k < n; k++)
                soma += n;

    return soma;
}
```

Exercício 11 (1,0)

Suponha que estamos comparando duas implementações de um algoritmo em um mesmo computador. Para entradas de tamanho n, a complexidade da implementação A é $fA(n) = 8.n^2$, enquanto a complexidade da implementação B é fB(n) = 64.n.log n. Para quais valores de n a implementação A é mais eficiente? E para quais valores de n a B é mais eficiente? Mostre que fB = O(fA).

Exercício 12 (1,0)

As funções log n e $\log n^2$ possuem a mesma ordem de complexidade? Justifique sua resposta.

Exercício 13 (2,0)

Resolva a equação de recorrência T(n) = 2.T(n/2) + 1, T(2) = 1.

Exercício 14 (2,0)

Encontre e resolva a equação de recorrência que calcula a complexidade da função recursiva a seguir. Considere as comparações escritas "comparações" como operações mais significativas.

```
void funcaoRecursiva(int n){
    if (n == 2)
        realiza duas comparações;
    else {
        funcaoRecursiva(n/2);
        funcaoRecursiva(n/2);
        realiza n-1 comparações;
    }
}
```

Exercício 15 (2,0)

Encontre e resolva a equação de recorrência que calcula a complexidade da função soma a seguir. Considere como operação mais significativa a soma +.

```
int soma(int n) {
    if (n != 0)
        return n + soma(n-1);
    else
        return n;
}
```

Exercício 16 (2,0)

Encontre e resolva a equação de recorrência que calcula a complexidade da função recursiva dada abaixo. Considere como operações mais significativas as comparações.

```
int buscaBinaria(int* V, int ch, int inicio, int fim){
    printf("Indices: %d,%d\n", inicio, fim);
    int meio = (inicio+fim)/2;
    printf("Comparando a chave com %d\n", V[meio]);
    if (ch == V[meio])
        return meio;
    else if (inicio >= fim)
        return -1;
    else if (ch < V[meio])
        return buscaBinaria(V, ch, inicio, meio-1);
    else
```

```
        return buscaBinaria(V, ch, meio+1, fim);
}
```

Exercício 17 (2,5)

Utilizando como base o código que gera as combinações, implemente um algoritmo para resolver o problema do Exemplo 1 do material de Projeto de Algoritmos.

Exercício 18 (6,0) - Problema da mochila

Imagine que você irá realizar uma viagem de alguns dias, obviamente necessitará levar uma mochila com alguns itens dentro. Se você for uma pessoa organizada provavelmente fará uma pequena lista com todos os objetos que poderia levar (roupas, livros, aparelhos eletrônicos, etc.), mesmo que essa lista seja imaginária ela irá existir em algum momento. No entanto, uma mochila não é infinita e cada item tem um peso, por exemplo uma mochila que comporta até quatro livros não consegue levar cinco ou mais livros de tamanhos semelhantes na viagem. Logo, muitas vezes algo pode ser deixado para trás. Essa ideia gera uma pergunta bastante simples, mas com uma resposta nem tão simples as vezes: o que devo levar?

Estamos na seguinte situação: você possui apenas uma mochila para a viagem e vários itens para levar, mas não cabem todos na mochila. Uma maneira de resolver isso é atribuir valor de importância para cada objeto e levar na viagem apenas os mais importantes, coloco na mochila o item mais importante dentre todos e, em seguida, o segundo item mais importante que caiba no espaço restante e assim sucessivamente até completar a mala. Esse método se chama "Método Guloso", é simples e rápido, mas nem sempre trás bons resultados (imagine que você preencheu a mochila com um item muito grande e não sobrou espaço para nenhum outro item, por exemplo roupas).

Assim é definido o problema da mochila, temos um número limitado de itens para colocar em uma mochila levando em consideração o peso de cada item e a capacidade do recipiente, e queremos maximizar o valor total de utilidade, sendo que a cada item é associado o seu valor de utilidade. O problema da mochila é um problema de programação linear inteira e é classificado como NP-difícil (NP-hard), caso alguém deseje listar todas as possibilidades e tomar a maior delas, a complexidade do problema cresce na ordem de 2^n (complexidade exponencial), onde n é o número de itens.

Existem algumas variações do problema da mochila. Será cobrado, nesse exercício, o problema da mochila binária também conhecido como problema da mochila 0-1.

O problema da mochila 0-1, ou mochila binária, consiste em, dada uma mochila de capacidade W e m itens cujos peso pi e valor de utilidade vi são dados, escolher quais itens

serão alocados na mochila. Considere a variável de decisão xj de maneira que, se o item j é alocado na mochila, então xj = 1, caso contrário, então xj = 0. Assim, o modelo matemático para o problema da mochila 0-1 é dado por,

$$\max \sum_{i=1}^{m} v_i \cdot x_i$$
$$\text{sujeito a} \sum_{i=1}^{m} p_i \cdot x_i \leq W$$

A função a ser maximizada vem depois da palavra max. A expressão **sujeito a** está sendo utilizada para definir o conjunto de restrições do problema.

Esta variação é uma das mais estudadas dentre os problemas de mochila pois pode ser visto como um simples problema de programação inteira, ele aparece como subproblema em muitos problema mais complexos e pode representar muitas situações práticas do dia-a-dia. Para ilustrar, considere a formulação do seguinte problema com 5 itens e uma mochila de capacidade W = 10:

$$\max \; z = 4x_1 + 6x_2 + 5x_3 + 3x_4 + x_5$$
$$\text{sujeito a} \; 5x_1 + 4x_2 + 3x_3 + 2x_4 + x_5 \leq 10$$
$$x_i \in \{0, 1\}, \; i = 1, \ldots, 5$$

Com base na formulação acima, extraímos os seguintes valores para as utilidades e pesos:

	x_1	x_2	x_3	x_4	x_5
v_i	4	6	5	3	1
p_i	5	4	3	2	1

Para o problema da mochila binária, **não** existe algoritmo guloso que sempre encontra a resposta ótima. Mas alguns algoritmos gulosos conseguem obter resultados próximos ou iguais ao ótimo para várias entradas. O problema da mochila binária pode ser resolvido também por programação dinâmica (PD) e por força bruta. As soluções por PD e força bruta conseguem chegar no ótimo sempre.

A sua tarefa é **propor e implementar** um algoritmo guloso para resolver o problema da mochila binária.

Para entender melhor o problema, você deve resolvê-lo para instâncias menores (até 5 itens, considerando entradas com diferentes capacidades) fazendo um "força bruta" no papel com o objetivo de encontrar a solução ótima e comparar com a solução do algoritmo proposto.

Junto com o código, você deve entregar uma documentação com as seguintes informações:

- **Explicação de como é feita a escolha gulosa**;
- **O resultado ótimo para três entradas diferentes (considerando 5 itens e capacidade 10) mostrando um força bruta no papel (ou arquivo);**
- **As saídas de cinco testes diferentes considerando entradas com 5 itens e capacidades variadas (por exemplo, 10, 20, etc), explicando se o algoritmo guloso proposto se aproximou ou não do ótimo;**
- **E respondendo às seguintes questões:**
  - **Com base nos testes feitos, o desempenho do algoritmo guloso foi bom ou ruim? Justifique.**
  - **No geral, para o problema da mochila 0-1 por que você acha que um algoritmo guloso não consegue chegar no ótimo sempre?**