Escola Superior de Tecnologia e Gestão | Politécnico do Porto

Curso de Licenciatura em Engenharia Informática

Ano letivo 2023/2024

Felgueiras, janeiro 2024

Conteúdo

[Resolucao de exame ED 3](#_Toc158143530)

[1 3](#_Toc158143531)

[2. 3](#_Toc158143532)

[3. 3](#_Toc158143533)

[4. 4](#_Toc158143534)

[5. 4](#_Toc158143535)

[a) 4](#_Toc158143536)

[b) 5](#_Toc158143537)

[6. 5](#_Toc158143538)

[a) 5](#_Toc158143539)

[b) 5](#_Toc158143540)

[7. 6](#_Toc158143541)

[a) 6](#_Toc158143542)

[b) 6](#_Toc158143543)

[c) 7](#_Toc158143544)

[8. 7](#_Toc158143545)

[a) 7](#_Toc158143546)

[b) 7](#_Toc158143547)

# Resolucao de exame ED

## 1

Uma *Stack,* é uma estrutura de dados linear na qual todas as inserções e remoções de

dados são feitas apenas numa extremidade da pilha, geralmente denominada de topo da pilha. Para implementar uma stack nós só precisamos de uma referência para o topo, ou seja, uma lista ligada simples já chega, mas usar uma lista duplamente ligada ou uma lista ligada simples, não tem diferença pois o que diferencia uma da outra é apenas que na lista duplamente ligada tem mais uma referencia para a cauda em que no caso do uso numa stack apenas iria ficar sempre nula, mas não afetaria a eficiência do programa.

## 2.

As listas baseadas em array oferecem acesso aleatório eficiente aos elementos, o que significa que você pode acessar qualquer elemento da lista diretamente usando seu índice. Isso torna a implementação do método next() do Iterator relativamente simples e eficiente. Os elementos são armazenados em uma área contígua de memória, o que pode melhorar a eficiência de cache. A inserção ou remoção de elementos no meio da lista pode ser cara, pois requer a realocação de elementos.

Listas ligadas são eficientes para inserção e remoção de elementos, especialmente no meio da lista, pois não requerem realocação de elementos. Isso é importante se você precisar modificar frequentemente a estrutura da lista. O acesso aleatório em uma lista ligada é menos eficiente, pois você precisa percorrer os elementos sequencialmente para chegar a um índice específico. Os elementos não estão armazenados em uma área contígua de memória, o que pode resultar em maior fragmentação de memória.

Se for para acesso rápido e eficiente aos elementos por índice, e a estrutura da lista não muda frequentemente, uma lista baseada em array pode ser mais eficiente.

Se a estrutura da lista muda frequentemente (inserções/remoções no meio) e o acesso sequencial é suficiente, uma lista ligada pode ser mais adequada.

## 3.

A afirmação é verdadeira se assumirmos que temos acesso aos últimos nós das duas listas ligadas, pois basta fazer o último nó da primeira lista apontar para o primeiro nó da segunda lista. Isso pode ser feito em tempo constante, ou seja, O(1). No entanto, se não tivermos acesso aos últimos nós das listas, teremos que percorrer uma das listas até encontrar o seu final, o que pode levar tempo linear, ou seja, O(n), onde n é o número de elementos da lista. Nesse caso, a afirmação é falsa.

## 4.

Lista original: [apple, tree, car, dog, yellow, frog, gun, harp]

Sublistas de quatro elementos: [apple, tree, car, dog], [yellow, frog, gun, harp]

Sublistas de dois elementos: [apple, tree], [car, dog], [yellow, frog], [gun, harp]

Sublistas de um elemento: [apple], [tree], [car], [dog], [yellow], [frog], [gun], [harp]

Sublistas de dois elementos: [apple, tree], [car, dog], [frog, yellow], [gun, harp]

Sublistas de quatro elementos: [apple, car, dog, tree], [frog, gun, harp, yellow]

Lista ordenada: [apple, car, dog, frog, gun, harp, tree, yellow]

## 5.

### a)



### b)

Remover o elemento mínimo(1):



Remover o elemento mínimo(1):



## 6.

### a)

Travessia em-ordem: Y, P, T, X, Z, K, Q

### b)

Travessia pre-ordem: X, Y, T, P, Q, Z, K

## 7.

### a)

Uma imagem com texto, papel, Produto em papel, notebook

Descrição gerada automaticamente

### b)

Uma imagem com texto, escrita à mão, Tipo de letra, documento

Descrição gerada automaticamente

### c)

Uma imagem com texto, escrita à mão, papel, tinta

Descrição gerada automaticamente

## 8.

### a)

**DFS**, pois ele explora um caminho até o fim antes de voltar atrás e tentar outro. Isso pode ser útil se quisermos encontrar um percurso que passe por cidades específicas ou que tenha alguma propriedade desejada.

### b)

**BFS**, pois ele explora todos os vizinhos de um nó antes de passar para o próximo nível. Isso pode ser útil se quisermos encontrar o caminho mais curto ou o menos congestionado entre dois nós.