Universidade Federal de Ouro Preto PCC104 - Projeto e An´alise de Algoritmos
Força Bruta e Busca Exaustiva
Prof. Rodrigo Silva
Neivaldo I. Matos Filho

1 - Algoritmo SelectSort

PAAselectsort.py > ...

```python
def selectsort(array):
    for i in range(0, len(array)):# faz a busca no array começando da posição 0 até o final, em busca no menor va
        min_i = i # atribui a variavel o menor valor
        for j in range(i + 1, len(array)):# nova busca comecando do proximo elemento(j) e faz a comparação até o
            if array[j] < array[min_i]: #comparacao  e depois faz a atribuicao
                min_i = j
        array[i], array[min_i] = array[min_i], array[i]#troca de posições no array

array = [9 ,4, 6, 3, 7, 5, 1, 11, 8, 8, 2]
selectsort(array)
print(array)
```

Expressão matematica: PA (a1, a2, a3, a4, a5,.. an-1, an)
Operação básica: IF(comparação)
Cálculo da função de custo:
C = (n -1 + 1)/2 = n(n-1)/2 = (n2 - n)/2 ϵ O(n)

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1] = \sum_{i=0}^{n-2} (n-1-i).$$

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} (n-1-i) = \frac{(n-1)n}{2}.$$

Indicação da classe de eficiência: algoritmo de ordenação por seleção Θn^2 nas entradas e na troca Θn (n-1).

2 - Algoritmo busca sequencial

PAABuscasequencial.py > sequencialBusca

```python
import time
import random
#Busca sequencial de elementos

def sequencialBusca(vetor, k):
    i = 0
    while i < len(vetor):
        if vetor[i] == k:
            return i
        i += 1
    return -1
```

Expressão matemática: função fatorial ("n! = n · (n-1) · (n-2) … 3 · 2 · 1")

$$C(n) = \frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \approx \frac{1}{2}n^2$$

$$\frac{T(2n)}{T(n)} \approx \frac{c_{op}C(2n)}{c_{op}C(n)} \approx \frac{\frac{1}{2}(2n)^2}{\frac{1}{2}n^2} = 4.$$

Operação básica = while(repetição até encontrar elemento)

Cálculo da função de custo:

- melhor caso C(n) = 1
- pior caso C(n) = n
- caso médio C(n) = (n+1)/2

Indicação da classe de eficiência: O(n)
LINEAR

3 - Algoritmo busca em largura

PAABFS.py > ...

```python
#BFS - Algoritmo para Busca em Largura
#importa a biblioteca para poder criar uma lista
from collections import defaultdict

class Graph:# Classe grafo, criação da lista adjacente
    def __init__(self):# funcao cria lista
        self.graph = defaultdict(list)# fornece uma funcao para criar valores
    def addEdge(self,u,v):# funcao adiciona vertices do grafo
        self.graph[u].append(v)
    def BFS(self, s):# funcao de busca em largura, recebe o nó do grafo para ser vizitado
        visited = [False] * (len(self.graph)) # marca todos os vertices como não visitados
        queue = [] # cria uma fila vazia para a funcao BFS
        queue.append(s)# pega o nó de origem
        visited[s] = True # marca o nó como visitado e insere na fila
        while queue: # funcao basica, enquanto a fila não for vazia
            s = queue.pop(0) # retira o ultimo vertice inserido
            print (s, " ") # imprime o vertice
            # Obtenha todos os vértices adjacentes dos vértices desenfileirados.
            # Se um adjacente não foi visitado, marque-o como visitado e coloque-o na fila
            for i in self.graph[s]:
                if visited[i] == False:
                    queue.append(i)
                    visited[i] = True
```

Expressão matematica: $$T(h,m) = \sum_{i=0}^{h-1} m^i = \frac{m^h - 1}{m - 1}$$

Cálculo da função de custo: $$T(h,m) = \frac{m^h - 1}{m - 1}$$

Indicação da classe de eficiência : $O(m^h)$

Para matriz de adjacência: $\Theta(|V^2|)$

Para Lista de adjacência: $\Theta(|V| + |E|)$

4 - Algoritmo Busca em profundidade

PAADFS.py > DFS

```python

from collections import deque

class Graph:# classe do grafo
    def __init__(self, edges, n):# funcao cria lista
        self.adjList = [[] for _ in range(n)]# Uma lista de listas para representar uma lista de adjacências
        for (src, dest) in edges:# adiciona arestas ao gráfico não direcionado
            self.adjList[src].append(dest)
            self.adjList[dest].append(src)
def DFS(graph, v, discovered):# Executa DFS no gráfico a partir do vértice `v`
    stack = deque()# cria uma Stack(Pilha) usada para fazer a busca em profundidade
    stack.append(v)# empurra o nó de origem para a Stack(Pilha)
    while stack:# enquanto(loop) até que a Stack(Pilha) esteja vazia
        v = stack.pop()# Retire um vértice da Stack(Pilha)
        if discovered[v]:# se o vértice já foi descoberto, ignore-o
            continue
        discovered[v] = True# aqui se o vértice que apareceu `v` ainda não foi descoberto;
        print(v, end=' ')# imprime `v` e processa seus nós adjacentes não descobertos na Stack(Pilha)
        adjList = graph.adjList[v]# faz para cada aresta (v, u)
        for i in reversed(range(len(adjList))):
            u = adjList[i]
            if not discovered[u]:
                stack.append(u)
```

Expressão matematica:

$$T(h, m) = \sum_{i=0}^{h-1} m^i = \frac{m^h - 1}{m - 1}$$

Cálculo da função de custo:

$$T(h, m) = \frac{m^h - 1}{m - 1}$$

Indicação da classe de eficiência : $O(m^h)$