Análise dos algoritmos

Bárbara Letícia Rodrigues Milagres

April 12, 2023

1 Selection Sort

1.1 Código

```
void SelectionSort(int A[], int n)
{
    int min, aux;
    for(int i = 0; i < n-1; i++)
    {
        min = i;
        for(int j = i+1; j<n; j++)
        {
            if(A[j] < A[min])
            {
                min = j;
            }
        }
        if (i != min)
        {
            aux = A[i];
            A[i] = A[min];
            A[min] = aux;
        }
    }
}
```

1.2 Expressão matemática

$$\sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 \tag{1}$$

Ao resolver a expressão matemática, chega-se a:

$$\frac{n^2 - n}{2} = \frac{1}{2}n^2 - \frac{n}{2} \approx \frac{1}{2}n^2 \tag{2}$$

1.3 Eficiência

É possível provar que a equação é $\Theta(n^2)$ moatrando que existem duas constantes positivas c1 e c2 tais que:

$$c1 * n^2 <= \frac{1}{2} * n^2 <= c2 * n^2 \tag{3}$$

Podemos escolher c1 $= \frac{1}{4}$ e c2 $= 1$, então:

$$\frac{1}{4} * n^2 <= \frac{1}{2} * n^2 <= 1 * n^2 \tag{4}$$

Chega-se a:

$$\frac{n^2}{4} <= \frac{1}{2} * n^2 <= n^2 \tag{5}$$

2 Sequential Search

2.1 Código

```
int SequentialSearch2(int A[], int n, int K)
{
    int i = 0;
    while(A[i] != K)
    {
        i++;
        if (i >= n)
        {
            return -1;
        }
    }
    return i;

}
```

2.2 Expressão matemática

$$\sum_{i=0}^{n} 1 = n + 1 \tag{6}$$

2.3 Eficiência

É possível provar que a equação é $\Theta(n)$ moatrando que existem duas constantes positivas c1 e c2 tais que:

$$c1 * n^2 <= n + 1 <= c2 * n^2 \tag{7}$$

Podemos escolher c1 = 1 e c2 = 2, então:

$$1 * n <= n + 1 <= 2 * n \tag{8}$$

A equação é verdadeira para estas constantes.

3 BFS

3.1 Codigo

```
void BFS(int v, map<int, list<int>>& adj, map<int, bool>& visitados)
{
    queue<int> q;
    visitados[v] = true;
    q.push(v);

    while(!q.empty())
    {
        int no = q.front();
        q.pop();
        cout << no << " ";

```

```
        for(int vizinho:adj[no])
        {
            if(!visitados[vizinho])
            {
                visitados[vizinho] = true;
                q.push(vizinho);
            }
        }
    }
}
```

4 DFS

4.1 Código

```
void DFS(int v, map<int, list<int>>& adj, map<int, bool>& visitados)
{
    visitados[v] = true;
    cout << v << " ";

    list<int>::iterator it;
    for (it = adj[v].begin(); it != adj[v].end(); ++it)
        if (!visitados[*it])
            DFS(*it, adj, visitados);
}
```

Em ambos os algoritmos, sendo E o número de arestas e V o número de vértices, tem se:

$$\sum_{i=1}^{V} 1 \tag{9}$$

Que equivale à remoção de vértices explorados. E:

$$\sum_{j=1}^{E} 1 \tag{10}$$

Que equivale à adição vizinhos a serem explorados.

Com isso, conclui-se que a complexidade de ambos algoritmos é de $\Theta(V + E)$.