UFRR

UNIVERSIDADE FEDERAL DE RORAIMA

Seminários

Prof. Dr. Herbert Oliveira Rocha
herbert.rocha@ufrr.br

Análise de Algoritmos

SEMINARIO - Prazo de Entrega: 14/06/2022

ATENÇÃO: Descrever as soluções com o máximo de detalhes possível. Todos os artefatos (slides e outros) gerados para este trabalho devem ser adicionados em um repositório no site github.com, com o seguinte formato:

Aluno_ws_AA_RR_2022

OBSERVAÇÃO: Os temas abaixo já foram escolhidos em sala de aula. Verificar o seu tema nos próximos slides. Na apresentação do seminario será por meio de um slide. Cada equipe terá 10 minutos para efetuar sua apresentação.

Análise de Algoritmos

Objetivos:

- Função de custo e complexidade
- Código em C do algoritmo proposto
- Experimentação com a execução do algoritmo com diferentes entradas e coleta de tempo de execução
- Gráfico de linha com o tempo de execução em relação a cada entrada e análise da tendência de comportamente assintótico
- Apresentar um algoritmo que seja mais eficiente, em termos de complexidade, do que o algoritmo análisado

Análise de Algoritmos

1) Torre de Hanoi

2) Quicksort

3) Mergesort

4) Insertion sort

5) VerificaAlgo

6) FazAlgo

7) Inserção da Árvore Red and Black

8) Selection Sort

9) Busca Binária

Vamos Praticar! - Recursividade

(1) Torre de Hanoi

```
1. Hanoi(n, Origem, Destino, Auxiliar){
2.      se n > 0{
3.           Hanoi(n-1,Origem,Auxiliar,Destino)
4.           move o disco da Origem para o Destino
5.           Hanoi(n-1, Auxiliar,Destino, Origem)
6.       }
7.}
```

Vamos Praticar! - Recursividade

(2) Quicksort

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```

```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```

Vamos Praticar!

(3) Mergesort

```
MERGE3-SORT(A, p, r)
 1: if p < r then
 2:    d ← (p+r)/3; // inteiro
 3:    MERGE3-SORT(A,p,p+d);
 4:    MERGE3-SORT(A,P+d+1,r-d);
 5:    MERGE3-SORT(A,r-d+1,r);
 6:    MERGE3(A,p,d,r); // custo n
 7: end if
```

Vamos Praticar!

(4) Insertion sort

```
FUNÇÃO INSERTION_SORT (A[], tamanho)
        VARIÁVEIS
                i, j, eleito
        PARA i <- 1 ATÉ (tamanho-1) FAÇA
                eleito <- A[i];
                j <- i-1;
                ENQUANTO ((j>=0) E (eleito < A[j])) FAÇA
                        A[j+1]:= A[j];
# Elemento de lista numerada
                        j:=j-1;
                FIM_ENQUANTO
                A[j+1] <- eleito;
        FIM_PARA
FIM
```

Análise de Algoritmos

(5) VerificaAlgo

```
VerificaAlgo (n: int);
i, j, k, l: int;
para l := 1 TO 10.000 faça
      para i := 1 TO n-5 faça
         para j := i+2 TO n/2 faça
            para k := 1 TO n faça
               {Inspecione elemento}
```

Análise de Algoritmos

(6) FazAlgo

```
void FazAlgo (int n) {
   int i, j, k;
   FOR (i= 1; i<n - 1; i++) {
       FOR (j= i + 1; j<= n; j++) {
           FOR (k = 1; k<=j;k++) {
               Algum comando de custo O(1)
   } } }
}
```

(7) Inserção da Árvore Red and Black

```
VP-INSERT(T,z)
 1: TREE-INSERT(T,z);
 2: z.cor ← VERMELHO;
 3: VP-INSERT-FIX(T,z);
```

```
VP-INSERT-FIX(T,z)
 1: while (z.pai).cor = VERMELHO do
 2:   if z.pai = ((z.pai).pai).esq then
 3:     y ← ((z.pai).pai).dir;
 4:     if y.cor = VERMELHO then
 5:       (z.pai).cor ← PRETO;
 6:       y.cor ← PRETO;
 7:       ((x.pai).pai).cor ← VERMELHO;
 8:       z ← (z.pai).pai;
 9:     else if z = (z.pai).dir then
10:       z ← z.pai;
11:       LEFT-ROTATE(T,z);
12:       (z.pai).cor ← PRETO;
13:       ((z.pai).pai).cor ← VERMELHO;
14:       RIGHT-ROTATE(T,(z.pai).pai);
15:   else (igual ao "if" trocando "dir" e "esq")
16:   end if
17: (T.raiz).cor ← PRETO;
```

Análise de Algoritmos

(8) Selection Sort

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

Análise de Algoritmos

(9) Busca Binária

```
BUSCA-BINÁRIA(V[], início, fim, e)
    i recebe o índice do meio entre início e fim
    se (v[i] = e) entao
        devolva o índice i   # elemento e encontrado
    fimse
    se (inicio = fim) entao
        não encontrou o elemento procurado
    senão
       se (V[i] vem antes de e) então
          faça a BUSCA-BINÁRIA(V, i+1, fim, e)
       senão
          faça a BUSCA-BINÁRIA(V, inicio, i-1, e)
       fimse
    fimse
```