Verão 2019 - TÓPICOS DE PROGRAMAÇÃO

Prof. Dr. Leônidas O. Brandão (coordenador)

Profa. Dra. Patrícia Alves Pereira (ministrante)

Prof. Bernardo (Monitor)

Ao analisar algoritmos é importante verificar...

- **Finitude**: o algoritmo para?
- **Corretude**: o algoritmo faz o que promete?
- **Complexidade**: quantas instruções são necessárias no pior caso para resolver o problema tratado pelo algoritmo?

Problemas clássicos

Busca em vetor ordenado

- Um vetor está ordenado se for crescente ou decrescente.
- Um vetor $v[p..r]$ crescente se $v[p] \leq v[p+1] \leq \ldots \leq v[r]$.
- Um vetor $v[p..r]$ decrescente se $v[p] \geq v[p+1] \geq \ldots \geq v[r]$.

Busca sequencial

//recebe um número x e um vetor crescente v[p..r] com p <= r+1

// devolve j em p..r tal que x == v[j], caso j não existe, devolve -1

```
int buscaSequencial (int x, int v[], int p, int r)
{
    int j = p;
    while (j <= r && v[j] < x) ++j;
    if (j <= r && v[j] == x) return j;
    else return -1;
}
```

Busca Binária

//recebe um número x e um vetor crescente v[p..r] com p <= r+1

// devolve j em p..r tal que x == v[j], caso j não existe, devolve -1

```
int buscaBinaria (int x, int v[], int p, int r)
{
    while (p <= r) {
        int m = (p + r)/2;
        if (x == v[m]) return m;
        if (x < v[m]) r = m - 1;
        else p = m + 1;
    }
    return -1;
}
```

Agora é com vcs... Em duplas pensem...

Para cada algoritmo de busca mostre:

- **Finitude**: o algoritmo para? (o quê garante a parada?)
- **Corretude**: o algoritmo faz o que promete? (qual é o invariante?)
- **Complexidade de tempo**: melhor caso e no pior caso?
- Lembrando: a operação mais cara, que deve ser considerada, é a comparação entre elementos.

- Respondam de forma individual no sistema

Busca sequencial:

- **Finitude**: o algoritmo para?

o algoritmo será encerrado:

- **quando encontrar x no vetor ou quando j>r.**

- **Corretude**: o algoritmo faz o que promete?

Invariante: No começo de cada iteração temos v[j-1] < x

- **Complexidade**:
- Melhor caso: x é encontrado na primeira posição do vetor ➔ complexidade no melhor caso: O(1) - constante
- Pior caso: o x não é encontrado no vetor, nesse caso o numero de interações será igual ao número de elementos do vetor adicionado de 1: r-p+1 ➔ pior caso: O(n) – linear.

Busca binária:

- **Finitude**: o algoritmo para?

o algoritmo será encerrado:

- quando encontrar x no vetor ou quando p>r.

- **Corretude**: o algoritmo faz o que promete?

Partindo do pressuposto que o algoritmo será invocado apenas se v[p] < x < v[r], temos o seguinte invariante:

no começo de cada iteração é válido a seguinte afirmação: v[p-1] < x < v[r+1]

- **Complexidade**: Consumo de tempo da função no pior caso: O(lg n)

lg abreviatura para o logaritmo na base 2 e N = r-p+1.

Busca binária: complexidade

- **Melhor caso: O(1) - constante**

Na primeira iteração o elemento é encontrado: v[m]=x

- Pior caso: O(lg N)

lg abreviatura para o logaritmo na base 2 e N = r-p+1

x não está no vetor

Busca binária: nunca faz mais que lg n iterações

iterações	k	elementos	Comparações
1º	0	$\frac{n}{1} = \frac{n}{2^0}$	2
2º	1	$\frac{n}{2} = \frac{n}{2^1}$	2
3º	2	$\frac{n}{4} = \frac{n}{2^2}$	2
4º	3	$\frac{n}{8} = \frac{n}{2^3}$	2
…	…	…	..
…	k	$\frac{n}{2^k}$	2

$$numero\ máximo\ de\ iterações: \frac{n}{2^k} = 1 \quad \rightarrow \quad 2^k = n \quad \rightarrow \quad k = \lg n$$

$$numero\ de\ comparações: \quad 2(k+1) = 2\lg n + 2$$

Comparando as duas buscas no pior caso

buscas	**n=1000** = 10^3	**n=** 10^6	**n=** 10^9
sequencial	1.000	1.000.000	1000.000.000
binária	$2 \lg 10^3 + 2 \approx 22$	$2 \lg 10^6 + 2 \approx 44$	$2 \lg 10^9 + 2 \approx 62$