Estruturas de Dados - Busca Binária

Análise de complexidade

O que já vimos:

- Notação O
- Complexidade de tempo
- Níveis de complexidade

Algoritmos eficientes

- Considere o problema de busca em um vetor
  - dado um vetor S e um número x, retornar o índice que representa a posição de x em S
- Abordagem: busca linear
  - percorrer todo o vetor S procurando por x

Algoritmos eficientes: busca linear

Algoritmo: BuscaLinear(S, p, q, x)

Entrada: vetor S, índices p e q, natural x

Saída: índice de x em $S[p..q-1]$, se existir, e q caso contrário

```
i = p
enquanto S[i] ≠ x e i < q faça
    i = i + 1
retorne i
```

- Complexidade?
- Eficaz?
- Eficiente?

Algoritmos eficientes

- Agora considere o problema de busca em um vetor **ordenado**
  - dado um vetor S ordenado (vamos assumir ordem crescente) e um número x, retornar o índice que representa a posição de x em S
- Abordagem inicial: busca linear
  - percorrer todo o vetor S procurando por x

Algoritmos eficientes

- Como fazer melhor que BuscaLinear?
- Resposta: utilizar a particularidade do nosso problema
  - "busca em um vetor **ordenado**"
- Como utilizar essa particularidade para construir uma solução mais eficiente?

Algoritmos eficientes

- Ideia: comparar x ao elemento y que divide o vetor S ao meio
    - se $x = y$, então achamos o elemento
    - se $x < y$, então x só pode estar na primeira metade de S
    - se $x > y$, então x só pode estar na segunda metade de S
- Podemos repetir esse processo sucessivas vezes, trabalhando com vetores cada vez menores
    - obs: se chegarmos num ponto em que não conseguimos dividir o vetor, teremos a certeza de que x não está no vetor S

Algoritmos eficientes: busca binária

Algoritmo: BuscaBinaria(S, p, r, x)

Entrada: vetor S, índices p e r, natural x

Saída: índice de x em $S[p..r-1]$, se existir, e r caso contrário

$q = \lfloor \frac{p+r-1}{2} \rfloor$
$rr = r$
enquanto $p < r$ **e** $S[q] \neq x$ **faça**
    se $x < S[q]$ **então**
        $r = q$
    senão
        $p = q + 1$
    $q = \lfloor \frac{p+r-1}{2} \rfloor$
se $S[q] == x$ **então**
    retorne q
retorne rr

Algoritmos recorrentes

- Como calcular a complexidade de uma busca binaria?
- Lembrando o comentário feito anteriormente: "podemos repetir esse processo sucessivas vezes, trabalhando com vetores cada vez menores"
- Ideia de recorrência!
    - caso desejemos, podemos até escrever uma versão recursiva desse algoritmo

Complexidade de algoritmos recorrentes

- Como calcular a complexidade de um algoritmo recorrente?
- Temos de trabalhar com uma função $T(n)$ que represente a recorrência utilizada
- Para o problema da busca binária, considere $n = r - p$
  - n é o tamanho do vetor de entrada
- Busca binária: $T(n) = ?$

Complexidade: busca binária

- Seja $T(n)$ o número de comparações, no pior caso, para um vetor S de tamanho $n = r - p$

$$T(n) = \begin{cases} 1, & \text{se } n = 1 \\ 1 + T\left(\frac{n}{2}\right), & \text{caso contrário} \end{cases}$$

$$\begin{aligned} T(n) &= 1 + T\left(\frac{n}{2}\right) \\ T(n) &= 1 + 1 + T\left(\frac{n}{4}\right) \\ T(n) &= 1 + 1 + 1 + T\left(\frac{n}{8}\right) \\ &\vdots \\ T(n) &= i + T\left(\frac{n}{2^i}\right) \end{aligned}$$

Complexidade: busca binária

- $T(n) = i + T(\frac{n}{2^i})$
  - quando alcançarmos o nosso caso-base, paramos de substituir as recorrências
  - caso-base: $T(1) = 1$
  - para chegarmos no caso-base, teremos $T(\frac{n}{2^i}) = T(1)$
  - nesse caso, $T(n) = i + T(1)$
  - teremos $\frac{n}{2^i} = 1 \rightarrow i = \log_2 n$
  - logo, $T(n) = \log_2 n + 1$

Complexidades:

- Busca linear: $O(n)$
- Busca binária: $O(\log n)$

Complexidade de algoritmos recorrentes

Comentários:

- Cada comparação representa uma operação básica do algoritmo, por isso optamos por contar a quantidade de comparações efetuadas
- A complexidade para a leitura da entrada é $O(n)$, mas não é levada em conta na análise da busca binária
- As complexidades da busca linear e da busca binária nos permitem concluir que, para valores de n suficientemente grandes, o pior caso de busca binária é "mais rápido" do que o pior caso da busca linear