Estruturas de Dados - Mergesort

Aula passada

O que vimos:

- Insertion Sort

Complexidades:

Algoritmo	melhor caso	pior caso
Insertion Sort	$O(n)$ (vetor ordenado)	$O(n^2)$

Merge Sort

- Ordenação por união
- Algoritmo de divisão e conquista
- Utiliza a ideia da divisão e conquista e faz uma ordenação por partes
- Recordando: como funciona a ideia de divisão e conquista?

Merge Sort

- Passos de uma solução utilizando divisão e conquista:
  - dividir: dividir o problema em casos menores do mesmo problema (subproblemas)
  - conquistar: resolver os subproblemas quando eles forem simples o suficiente
  - combinar: unir a solução dos subproblemas para criar uma solução para um problema maior
- Deste modo, resolvemos o problema por completo
- Nosso problema é o de ordenação
- A partir disso, como aplicaremos esta ideia?

Merge Sort

- Ideia para ordenar:
  - suponha que tenhamos um vetor S de tamanho n
  - dividimos esse vetor de tamanho n em dois subvetores de tamanhos $\lfloor \frac{n}{2} \rfloor$ e $\lceil \frac{n}{2} \rceil$ e ordenamos cada um dos dois vetores individualmente
    - caso tenhamos um vetor unitário, o vetor está ordenado
  - combinamos os dois subvetores ordenados e montamos o vetor S ordenado por completo, com tamanho n
- Os passos acima são repetidos para a ordenação de cada subvetor
- Corretude: essa ideia funciona?

Merge Sort

- Caso base da recursão:
  - vetor unitário: caso trivial onde o vetor é ordenado
- Chamadas recursivas:
  - divisão e conquista: dividimos o problema maior em problemas menores do mesmo tipo
  - consideramos que as chamadas recursivas são eficazes para os subproblemas
    - em outras palavras, consideramos que as chamadas recursivas do nosso algoritmo ordenam os subproblemas com sucesso
- União dos subproblemas em um problema maior:
  - unir os resultados das chamadas recursivas para formar um resultado maior
  - este caso não pode ser resolvido com chamadas recursivas; se trata de um novo problema
  - caso seja resolvido corretamente, torna o algoritmo correto

Merge Sort: função Merge

- Função Merge:
  - recebe dois subvetores ordenados individualmente, e deve uni-los em um único vetor ordenado
    - obs: no caso do Merge Sort, os dois subvetores sempre estarão "um do lado do outro"
    - em vez de passar dois vetores, podemos passar índices
  - ideia: comparar os elementos de cada subvetor e escolher o menor elemento, guardando-o no vetor final ordenado
    - facilitador: os dois vetores estão ordenados!

Merge Sort: função Merge

Algoritmo: Merge(S, p, q, r)

Entrada: vetor S, índices p, q e r com $S[p..q]$ e $S[q+1..r-1]$ ordenados

Saída: vetor S ordenado de p a $r-1$

$n_1 = q - p + 1$
$n_2 = (r-1) - q$
criar novos vetores $L[1..n_1+1]$ e $R[1..n_2+1]$
para $i = 0$ **até** $n_1 - 1$ **faça**
    $L[i] = S[p+i]$
para $j = 0$ **até** $n_2 - 1$ **faça**
    $R[j] = S[(q+1)+j]$
$L[n_1+1] = \infty$
$R[n_2+1] = \infty$
$i = 0$
$j = 0$
para $k = p$ **até** $r - 1$ **faça**
    se $L[i] \leq R[j]$ **então**
        $S[k] = L[i]$
        $i = i + 1$
    senão
        $S[k] = R[j]$
        $j = j + 1$

Merge Sort: algoritmo

Algoritmo: MergeSort(S, p, r)

Entrada: vetor S, índices p e r

Saída: vetor S ordenado de p a $r-1$

```
se p < (r − 1) então
    q = ⌊(p + (r − 1)) / 2⌋
    MergeSort(S, p, q + 1)
    MergeSort(S, q + 1, r)
    Merge(S, p, q, r)
```

- **se** $p < (r-1)$ **então**
  - $q = \lfloor \frac{p+(r-1)}{2} \rfloor$
  - MergeSort$(S, p, q+1)$
  - MergeSort$(S, q+1, r)$
  - Merge(S, p, q, r)

Merge Sort: complexidade

- Complexidade do Merge Sort:
  - cada chamada para MergeSort para um vetor de tamanho n inclui:
    - duas chamadas para MergeSort passando vetores de tamanho $\frac{n}{2}$
    - uma chamada para Merge, que tem complexidade $O(n)$
  - seja $T(n)$ o tempo de execução de MergeSort para um vetor de tamanho n
  - podemos representar $T(n)$ através da seguinte recorrência

$$T(n) = \begin{cases} 1, & \text{se } n = 1 \\ 2T(n/2) + n, & \text{se } n > 1 \end{cases}$$

Merge Sort: complexidade

- Desenvolvendo a recorrência, temos:

$$
\begin{aligned}
T(n) &= 2T(n/2) + n \\
&= 2(2T(n/4) + n/2) + n = 4T(n/4) + 2n \\
&= 4(2T(n/8) + n/4) + 2n = 8T(n/8) + 3n \\
&= 8(2T(n/16) + n/8) + 3n = 16T(n/16) + 4n \\
&\vdots \\
&= 2^i T(n/2^i) + in
\end{aligned}
$$

- Para $T(1)$, temos $n/2^i = 1 \rightarrow i = \log_2 n$. Então, temos:

$$
\begin{aligned}
T(n) &= 2^i T(n/2^i) + in \\
&= 2^{\log_2 n} T(1) + n \log_2 n \qquad = n \cdot 1 + n \log_2 n = O(n \log n)
\end{aligned}
$$

Merge Sort

- Vantagens:
  - assim como vários algoritmos recursivos, tem uma implementação relativamente simples (a implementação mais complicada é a do Merge)
  - estável
  - complexidade $O(n \log n)$ constante
- Desvantagens:
  - função recursiva (funções recursivas têm um maior consumo de memória e processamento)
  - memória: o algoritmo utiliza vetores auxiliares em cada nível de chamada recursiva