Alfredo's MAC0110 Journal

Alfredo Goldman

June 3, 2020

0.1 Aula 20 - Ainda ordenação

Nessa aula vamos continuar vendo a ordenação, mas antes disso, vamos finalmente ver o algoritmo de busca binária para encontrar um elemento em um vetor ordenado.

0.1.1 Busca binária

Ao invés de percorrer o vetor, desde o início, procurando o elemento como em:

```
function buscaLinear(x, v)
  for i in 1:length(v)
    if v[i] == x
      return i
    end
  end
  return 0
end
```

Podemos a cada procura, para ver se o elemento está no vetor, eliminar metado do vetor. Se o elemento for menor que o meio, olhamos do começo ao meio. se for maior que o meio, olhamos do meio ao fim. Se for igual, achou.

```
function buscaBinaria(x, v)
   inicio = 1
   fim = length(v)
   while in <= fim
      meio = div(inicio + fim, 2)
      if v[meio] == x
        return meio
      elseif x < v[meio]
        fim = meio - 1
      else
        inicio = meio + 1
      end
   end
   return 0
end
```

Há também a versão recursiva, para isso temos que ter o inĩcio e o fim como parâmetros.

```
function buscaBinariaRec(inicio, fim, x, v)
   if inicio > fim
     return 0
   end
   meio = div(inicio + fim, 2)
   if v[meio] == x
     return meio
   elseif x < v[meio]
     buscaBinariaRec(inicio, meio - 1, x, v)
   else
     buscaBinariaRec(meio + 1, fim, x, v)
   end
end
```

0.1.2 Métodos de busca mais elaborados

Uma forma mais eficiente de se ordenar um vetor é usando a divisão e conquista, isso é, dado um vetor, quebramos em duas partes, ordenamos as partes e depois fazemos o merge, no caso como podemos ter repetição, vamos usar a versão que permite duplicação.

```
function merge(u, v)
  pu = 1  # ponteiro em u
  pv = 1  # ponteiro em v
  resp = []
  while pu <= length(u) && pv <= length(v)
    if u[pu] < v[pv]
      push!(resp, u[pu])
      pu = pu + 1
```

```
    elseif v[pv] < u[pu]
      push!(resp, v[pv])
      pv = pv + 1
    else
      push!(resp, v[pv])
      pu = pu + 1
    end
  end
  while pu <= length(u)
    push!(resp, u[pu])
    pu = pu + 1
  end
  while pv <= length(v)
    push!(resp, v[pv])
    pv = pv + 1
    end
  return resp
end
```

Dado o merge, a ideia é:

- Divida o vetor no meio
- Ordene cada metade separadamente
- Devolva o merge

Para isso vamos ver mais uma possibilidade de Julia, dado um vetor v, v[1:meio] cria um vetor até o meio e v[meio + 1:length(v)] cria um vetor do meio + 1 ao final.

Com isso fica fácil fazer o mergeSort.

```
function mergeSort(v)
  if length(v) <= 1
    return v
  end
  meio = div(length(v), 2)
  v1 = v[1:meio]
  v2 = v[meio + 1:length(v)]
  v1ord = mergeSort(v1)
  v2ord = mergeSort(v2)
  return merge(v1ord, v2ord)
end
```

Vamos ao último algoritmo, que também fica melhor de forma recursiva, dado um vetor, o primeiro passo é escolher um elemento de forma a dividir o vetor em duas partes, quem for menor ou igual a esse elemento, e quem for maior. Isso deve ser feito de forma recursiva.

```
function quick!(i, j, v)
    if j > i
        pivo = v[div(i+j, 2)]
        left = i
        right = j
        while left <= right
            while v[left] < pivo
                left += 1
            end
            while v[right] > pivo
                right -= 1
            end
            if left <= right
                v[left], v[right] = v[right], v[left]
                left += 1
                right -= 1
            end
        end
        quick!(i, right, v)
        quick!(left, j, v)
    end
    return v
end
```