ALGORITMOS DE SELEÇÃO (Cormen, cap. 9)

Informalmente, o PROBLEMA DE SELEÇÃO é o de encontrar o k-ésimo menor elemento de um conjunto. Mais precisamente:

1. DEFINIÇÃO: o PROBLEMA DE SELEÇÃO é o de, dados um vetor V[1..n] de elementos ordenáveis e um índice k ∈ [1..n], encontrar um elemento x ∈ V que seja o k-ésimo elemento de alguma permutação ordenada V' de V.

   (Observe que, se os elementos de V forem todos distintos, então existe apenas uma permutação ordenada de V, e portanto uma única solução para o problema.)

   Uma MEDIANA INFERIOR de um vetor V[1..n] é uma saída do problema de seleção com k = ⌊(1+n)/2⌋, e uma MEDIANA SUPERIOR o análogo com k = ⌈(1+n)/2⌉. Quando "n" é ímpar, as duas coincidem e podemos nos referir simplesmente à "mediana" de V. Quando "n" é par, a expressão não-qualificada "mediana" pode se referir a qualquer uma das medianas (frequentemente a inferior).

Observe que o problema de seleção pode ser resolvido em tempo O(n*lg n): basta ordenar o vetor e então retornar o k-ésimo elemento do vetor ordenado. Agora, é possível resolver o problema em o(n*lg n)?

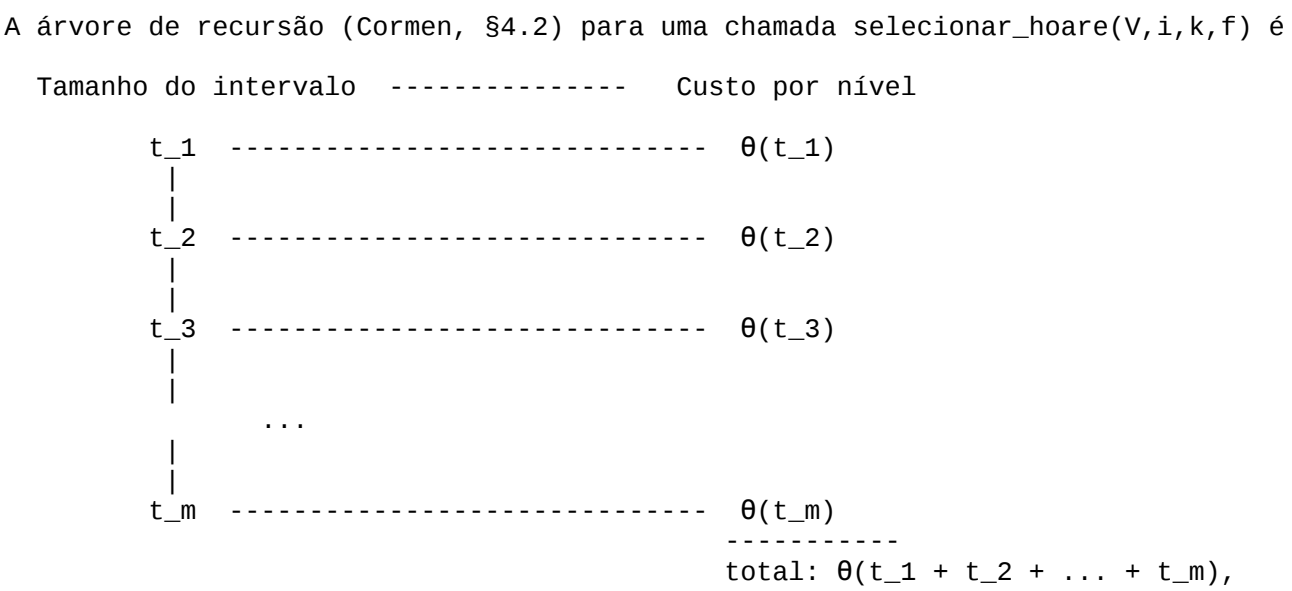
ALGORITMO DE SELEÇÃO DE HOARE

Os algoritmos de particionamento que vimos na aula sobre o Quicksort tem uma propriedade interessante para o problema de seleção: se uma chamada particionar(V,i,k,f) retorna um índice "p", então, denotando por V' o vetor particionado, o elemento V'[p] é uma solução para a entrada (V,p) do problema de seleção -- ou, em outras palavras, V'[p] é o/um p-ésimo menor elemento de V.

A observação acima nos leva à seguinte estratégia para resolver uma entrada (V,k) do problema de seleção: particionar o vetor V com relação a um elemento qualquer, que passará então a estar numa posição "p"; se k = p, então esse elemento é uma solução para a entrada (V,k); se k < p, então basta procurar a solução à esquerda do pivô; se k > p, idem para a direita. O algoritmo abaixo materializa essa estratégia:

```
Algoritmo:    selecionar_hoare
Entrada:      um vetor ordenável V[1..n] e índices "i", "k", "f".
Pré-condição: 1 <= i <= k <= f <= n.
Saída:        um elemento "x" do vetor.
Pós-condição: V[i..f] estará particionado com relação a V[k], e V[k] = x.
---------------------------------------------------------------------------
1. SE i = f ENTÃO RETORNE V[i].
2. p := particionar_hoare(V,i,k,f)
3. SE k = p ENTÃO RETORNE V[k].
4. SE k < p ENTÃO RETORNE selecionar_hoare(V,i,k,p-1).
5.          SENÃO RETORNE selecionar_hoare(V,p+1,k,f).
```

A árvore de recursão (Cormen, §4.2) para uma chamada selecionar_hoare(V,i,k,f) é



sendo t_1 = f-i+1 o tamanho do intervalo V[i..f] e t_2 ... t_m os tamanhos dos intervalos de entrada das chamadas recursivas. No caso médio, o algoritmo executa em tempo linear (Cormen, §9.2). No pior caso, porém, temos

  t_{i+1} = t_i - 1

para todo i, e o tempo de execução do algoritmo é quadrático sobre t_1.

2. EXERCÍCIO: mostre que selecionar_hoare(V,i,k,f) executa em tempo O(t²), sendo
   --------- t = f-i+1 se i <= f.

 ALGORITMO LINEAR DE SELEÇÃO

Como vimos acima, o tempo de execução de uma chamada selecionar_hoare(V,i,k,f) é θ(t_1 + t_2 + ... + t_m), sendo t_1 o tamanho do intervalo V[i..f] e t_2 ... t_m os tamanhos dos intervalos das chamadas recursivas. Logo, se soubéssemos escolher o pivô de forma que, para todo i, t_{i+1} <= t_i * r, para alguma razão 0 < r < 1 fixa e independente da entrada, então o tempo de execução do algoritmo seria

  O(t_1 + t_1*r + t_1*r² + t_1*r³ + ... + t_1*r^{m-1})

  = O(t_1 * (1 + r + r² + r³ + ... + r^{m-1}))

  = O(t_1 * (1 / (1-r))) --> soma de P.G. infinita, razão < 1 (Cormen, §A.1)

  = O(t_1),

ou seja, linear sobre o tamanho do intervalo de entrada.

O algoritmo de seleção abaixo utiliza a ideia acima: particionando o intervalo de entrada em segmentos de 5 elementos (com a possível exceção do último), descobrindo a mediana de cada segmento e então utilizando como pivô a mediana dessas medianas, o algoritmo garante que pelo menos 30% do intervalo é composto de elementos maiores que o pivô, e o mesmo para elementos menores que o pivô; logo, a cada chamada recursiva, pelo menos 30% do intervalo é deixado de lado, obtendo-se a "razão fixa" desejada acima (o argumento não é simples como acima, porém, pois no algoritmo selecionar_hoare a escolha do pivô acontece em O(1), ao passo que no algoritmo abaixo ela exige uma chamada recursiva).

```
================================================================================
Algoritmo:    selecionar_bfprt // Blum, Floyd, Pratt, Rivest, Tarjan (1973).
Entrada:      um vetor ordenável V[1..n] e índices "i", "k", "f".
Pré-condição: 1 <= i <= k <= f <= n.
Saída:        um elemento "x" do vetor.
Pós-condição: V[i..f] estará particionado com relação a V[k], e V[k] = x.
--------------------------------------------------------------------------------
01. t := f-i+1, s := ⌈t/5⌉
02. Considere V[i..f] como dividido em "s" segmentos contíguos de tamanho 5,
    com a possível exceção do último deles, que deve ter tamanho "t mod 5" se
    tal valor não for zero.
03. Ordene cada um dos "s" segmentos.
04. SE s = 1 ENTÃO RETORNE V[k]. FIM_SE
05. PARA j DE 1 A s
06. | Troque V[i+j-1] de posição com a mediana do j-ésimo segmento de V.
07. ult_med := i+s-1, med_meio := ⌊(i+ult_med)/2⌋
08. selecionar_bfprt( V, i, med_meio, ult_med )
09. p := particionar_hoare( V, i, med_meio, f )
10. SE k = p ENTÃO RETORNE V[k]. FIM_SE
11. SE k < p ENTÃO RETORNE selecionar_bfprt( V, i,   k, p-1 ).
12.          SENÃO RETORNE selecionar_bfprt( V, p+1, k, f   ). FIM_SE
================================================================================
```

Observe que, exceto pelas linhas das chamadas recursivas do algoritmo acima, todas as outras linhas do algoritmo executam em O(t), sendo t = f-i+1. Em particular:

* A linha 2 é um recurso textual e na prática não é executada;

* Na linha 3, são O(t) segmentos, e ordenar cada segmento custa O(1), já que cada um tem tamanho <= 5, e que 5 é uma constante em relação à entrada.

Para obtermos uma equação de recorrência para o tempo de execução desse algoritmo, precisamos avaliar o tamanho das instâncias de entrada das chamadas recursivas por ele realizadas. É fácil verificar que, na linha 5, o intervalo V[i..ult_med] tem exatamente s = ⌈t/5⌉ elementos. Para as chamadas das linhas 11 e 12, a análise não é igualmente direta. Abaixo, uma análise informal é apresentada, cujo objetivo é explicitar as ideias que levam ao algoritmo. Entendido esse raciocínio essencial, uma argumentação precisa pode ser feita com base nas técnicas já apresentadas nesta disciplina (veja o último exercício).

ANÁLISE INFORMAL DO TEMPO DE EXECUÇÃO DE SELECIONAR_BFPRT

Nas linhas 11 e 12, o tamanho do intervalo da chamada recursiva é igual a t - D, sendo D o número de elementos "descartados" na chamada. Informalmente, podemos verificar que D é aproximadamente 3t/10, da seguinte maneira:

1. São aproximadamente t/5 medianas, uma para cada segmento de 5 elementos.

2. Como o pivô é a mediana das medianas, então aproximadamente a metade delas -- ou seja, t/10 medianas -- é maior que o pivô.

3. Além disso, cada mediana corresponde a um segmento de 5 elementos do vetor, e nesse segmento há dois elementos maiores que a mediana em questão. Logo, para cada uma das t/10 medianas maiores que o pivô, há ainda outros 2 elementos do vetor maiores que o pivô.

4. No total, portanto, sabemos que há t/10 + 2*t/10 = 3t/10 elementos do vetor que são necessariamente maiores que o pivô.

5. Logo, no caso da chamada recursiva da linha 11, os 3t/10 elementos do vetor que são maiores que o pivô são descartados, isto é, D = 3t/10;

6. O raciocínio é análogo no caso da chamada recursiva da linha 12.

Assim sendo, o tamanho do intervalo de entrada das chamadas recursivas das linhas 11 e 12 do algoritmo é no máximo t - 3t/10 = 7t/10, com o quê concluímos que uma função

f(t) = a*t + f(t/5) + f(7t/10), para todo "t" grande o suficiente, (1)

limita superiormente o tempo de execução do algoritmo. Resta, portanto, mostrar que f(t) = O(t).

Observe, então, que o início da árvore de recursão para a recorrência acima é

```
     Tamanho do intervalo  ------------  Custo por nível

          ____t____        -----------------  θ(t)
         /         \
       t/5        7t/10    -----------------  θ(9t/10)
      /   \       /    \
  t/25  7t/50  7t/50  49t/100  ------------  θ(81t/100) = θ(t*(9/10)²)
   ...
```

De fato, como

1) a*t é uma função linear em "t", e

2) t/5 + 7t/10 = 9t/10,

então o custo do 2º nível da recursão é a*(9/10)*t, o do 3º é a*(9/10)²*t, etc. Daí, temos

```
f(t) <= a*t + a*(9/10)*t + a*(9/10)²*t + a*(9/10)³*t + ... --> soma "infinita"
      = a*t*(1 + 9/10 + (9/10)² + (9/10)³ + ...)
      = a*t*(1/(1 - 9/10)) --> soma de P.G. infinita, razão < 1 (Cormen, §A.1)
      = a*t*10.
```

Logo, f(t) = O(t), CQD.

3. OBSERVAÇÃO IMPORTANTE: a análise acima usa a premissa de que, para cada segmento cuja mediana é "maior" que o pivô, há três elementos que, após o particionamento, estarão à direita do pivô: a mediana do segmento em questão e os dois elementos que, logo após a execução da linha 3 do algoritmo, estão à direita dessa mediana.

   PORÉM, A PREMISSA ACIMA SÓ É NECESSARIAMENTE VERDADE SE OS ELEMENTOS DO VETOR FOREM TODOS DISTINTOS.

   No caso em que podem haver elementos repetidos, é fácil ver que a premissa não necessariamente vale: considere o vetor

   [1 2 3 4 5 1 2 3 4 5 1 2 3 4 5];

   nesse vetor, todas as medianas são iguais a 3, e portanto o pivô também o será; além disso, é possível que, após o particionamento, o pivô fique à direita de todas as medianas, e portanto a premissa acima é falsa. O fato fica ainda mais evidente quando se considera um vetor cujos elementos são todos iguais.

É POSSÍVEL, PORÉM, modificar o algoritmo de forma que ele sempre execute em tempo linear, mesmo na possibilidade de entrada com elementos repetidos. Para tanto, basta utilizar, ao invés do algoritmo de bi-partição de Hoare, o algoritmo de tri-partição mencionado na nota de aula sobre o Quicksort. Nesse caso, o particionamento retorna os índices "a" e "b" que delimitam o trecho dos elementos do vetor que são iguais ao pivô, bastando então utilizar esses índices adequadamente nas últimas 3 linhas do algoritmo.

4. EXERCÍCIO (IMPORTANTE): escreva uma versão de selecionar_bfprt que usa um
   ---------- algoritmo de tri-partição ao invés de um algoritmo de bi-partição, e então mostre (de forma semelhante à argumentação original) que essa versão executa em O(t) mesmo se a entrada possuir elementos repetidos.

5. EXERCÍCIO (ESSENCIAL): considere a variação do algoritmo selecionar_bfprt na
   ---------- qual os segmentos têm não mais 5, mas sim 3 elementos. Por meio de análise análoga àquela acima, verifique se o tempo de execução do algoritmo continua linear. Faça o mesmo para segmentos de 7 elementos. E para segmentos de tamanho 2y + 1, com y >= 3, a resposta é sempre a mesma?

6. EXERCÍCIO (IMPORTANTE): a análise de tempo de execução acima foi informal;
   ---------- refaça então a argumentação, levando em consideração observações como:

   * O número preciso de medianas é ⌈t/5⌉, logo o número de medianas que não são o pivô é m = ⌈t/5⌉ - 1; se "m" for ímpar, então haverá ⌈m/2⌉ medianas maiores que o pivô e ⌊m/2⌋ menores, ou então o contrário.

   * Caso o último segmento não tenha exatamente 5 elementos, então ele não "contribui" com exatamente 2 elementos maiores/menores que o pivô.

   * O segmento cuja mediana é o pivô também contribui com 2 elementos maiores / menores que o pivô, possivelmente exceto se ele for o último segmento.

   Escreva então uma equação de recorrência precisa, utilizando os pisos e tetos que forem necessários, e prove por indução que a função resultante é O(t) (como de costume, basta provar o passo indutivo). Ao final, compare a sua resposta com o desenvolvimento da seção 9.3 do Cormen.