Trabalho 1 – Análise de Algoritmos

David Beyda e Luiz Carlos Rumbelsperger

# Tarefa 1: Implementação

Abaixo, a função de LinearSelection(A, k), que se utiliza do algorítmo de mediana das medianas, para fazer a seleção do k-ésimo menor elemento em tempo linear. Ela utiliza uma função simple\_median, que ordena uma lista com o sorted() do Python, e a mediana da lista.

1. **def** median\_of\_medians(ls, k):
2. n = len(ls) # O(1)
3. **if** k <= 0 **or** k > n:
4. **raise** Exception(f"""parameter k should be greater than 0 and less then the size of the input list.
5. Was = {k}.
6. Size of list was = {n}.
7. """
8. )
9. **if** n <= 5:
10. **return** sorted(ls)[k - 1]
12. group\_medians = [simple\_median(ls[i : i + 5]) **for** i **in** range(0, n, 5)] # O(n),
13. # python list slicing is O(i + 5 - i = 5) for a range of 5.
14. # simple\_median for a list of size 5 can be assumed O(1).
15. # if the list is not a multiple of 5, the last n % 5 elements will be treated just like the others.
16. n = ceil(len(group\_medians) / 2)
17. # for odd lengths this gets the middle point.
18. group\_mom = median\_of\_medians(group\_medians, n)
20. #left = [x for x in ls if x < group\_mom] # O(n)
21. #right = [x for x in ls if x > group\_mom] # O(n)
22. # unfortunately this is not optimized, even though it is readable.
24. left = []
25. right = []
26. **for** x **in** ls:
27. **if** x < group\_mom: left.append(x)
28. **elif** x > group\_mom: right.append(x)

31. left\_len = len(left)
32. right\_len = len(right)
33. middle\_len = len(ls) - right\_len - left\_len
35. **if** left\_len <= (k - 1) < (left\_len + middle\_len) : **return** group\_mom
36. **if** k - 1 < left\_len   : **return** median\_of\_medians(left, k)
37. **if** k - 1 >= (left\_len + middle\_len)  : **return** median\_of\_medians(right, k - left\_len - middle\_len)

É pedido que se justifique porque esse código satisfaz uma equação de recorrência da forma T(n) ≤ cst · n + T(n/5) + T(7n/10), em especial, explicar a justificativa para a parte em azul.

Analisando o algorítmo, ele começa pegando o tamanho da lista fornecida, que em python é feito em O(1), mas, no pior das hipóteses, seria feito em O(n), percorrendo toda a lista. Em seguida, caso n seja menor ou igual a 5, a é feita a seleção normal, por ordenação, O(nlogn). Entretanto, como isso só é feito para listas com tamanho menor ou igual a 5, conforme n aumenta, as iterações dessa parte do código não aumentam junto. Podemos considerar essa parte como O(1). Em seguida, é calculada a mediana com essa mesma função para toda a lista, dividida em grupos de 5. Cada cáclulo de mediana pode ser considerado feito em tempo constante O(1), e isso feito n/5 vezes, resulta em O(n). O próximo passo é calcular recursivamente a mediana desse grupo de medianas. Feito isso, deve-se percorrer novamente a lista original, separando seus elementos em duas listas novas, uma para os elementos maiores que a mediana e outra para os elementos menores que a mediana. Esse último passo é, novamente, O(n). Por fim, é aplicada a recursão.

**Como rodar o executável, e o formato de entrada esperado:** o arquivo referente à tarefa 1 é o arquivo “linear\_select.py”. Ele instancia um lista com números cuidadosamente escolhidos para refletirem diversos pontos críticos do algorítmo. Além disso, ele também define casos de teste na forma de tuplas (k, gabarito), onde k é o k a ser fornecido para o algorítmo, e gabarito é o número que o algoritmo deve retornar. A lista ls pode ser alterada conforme necessário, e os casos de teste também. Para rodar, basta digitar no terminal “python linear\_select.py”. Vale lembrar que o código foi feito usando **Python 3**. O algorítmo funciona para uma lista de inteiros ou de floats.

# Tarefa 2: Experimentos

Para a tarefa 2, fez-se um novo arquivo chamado “experiments.py”, que cuida da geração dos dados para o experimento requisitado. O programa gerou 10 médias para cada tamanho de vetor, utilizando ambos os algorítmos (linear e por ordenação). Em seguida, os dados foram importados para o Excel, onde se construiu um gráfico para comparação dos algorítmos:

|  |  |
| --- | --- |
| **Linear Selection** | **Sort Selection** |
| 0,000496411 | 0,049831295 |
| 0,000989938 | 0,207547235 |
| 0,001455808 | 0,473644161 |
| 0,00197289 | 0,846356726 |
| 0,002384901 | 1,325044584 |
| 0,002883172 | 1,915427113 |
| 0,003311586 | 2,613742447 |
| 0,003939819 | 3,417948008 |
| 0,004311872 | 4,324234319 |
| 0,004868054 | 5,35325036 |

Nesse gráfico, o tempo foi ajustado para que ficasse mais fácil de ser visualizado, logo, os valores de tempo das duas séries não possuem uma unidade exata, são apenas para fins de comparação entre elas próprias.

Percebe-se o rápido crescimento do Sort Selection. O Sort Selection foi implementado utilizando BubbleSort, conforme pedido no enunciado do trabalho, que é O(n²). O crescimento quadrático se mostra bem claro no gráfico. Além disso, o Linear Selection já começou com um tempo muito mais baixo do que o Sort Selection, e pode-se perceber que seu crescimento é linear. Isso faz com que, para valores grandes de n, a diferença no tempo de execução dois algorítmos seja grande.