Projeto de Algoritmo com Implementação nº 1

MC458 - 2s2020 - Tiago de Paula Alves - 187679

1 Problema da Seleção

Obtenção dos $1 \leq k \leq n$ menores elementos de um dado vetor V de tamanho n, em ordem crescente.

1.1 Método 1: Busca dos Menores Elementos

O primeiro método de seleção implementado consiste na busca do menor elemento de V, que é então removido do vetor, repetindo o processo k vezes para encontrar os k menores. Então, o tempo de execução do algoritmo fica em ordem $T_1(n,k) = \sum_{i=0}^{k-1}(n-i) + \Theta(1) = \Theta(kn)$. No pior caso, como $k \in O(n)$, a complexidade se torna $T_1(n) \in O\left(n^2\right)$.

1.2 Método 2: Quicksort

Para o segundo método foi utilizada a função dada que realiza a ordenação completa do vetor para que a seleção dos menores elementos seja em tempo constante. Logo, a ordenação, que no caso é feita com o Quicksort, domina o tempo de execução e a complexidade de pior caso se torna $T_2(n,k) = \Theta\left(n^2\right)$. Entretanto, no caso médio o Quicksort consegue uma complexidade $\Omega(n \lg n)$, que pode ser esperada nas implementações do algoritmo.

1.3 Método 3: Heap de Mínimo

O último método se baseia na construção de um heap de mínimo, onde as seleções de menor elemento podem ser feitas de forma eficiente. Assim, o heap pode ser montado a partir de V em tempo $\Theta(n)$ e as extrações do mínimo são feitas em $O(\lg n)$. Então, a complexidade desse método fica $T_3(n,k) = \Theta(n) + \sum_{i=0}^{k-1} O(\lg(n-i)) = O(n + k \lg n)$, que é o caso esperado já que os elementos de V são em sua maioria distintos. No pior caso, podemos dizer também que $T_3(n) \in O(n \lg n)$.

2 Eficiência dos Métodos

Para valores pequenos de k, o algoritmo mais eficiente foi o de busca linear, do método 1. Nessa faixa de entrada, podemos considerar k constante e, portanto, o tempo de execução desse algoritmo seria de ordem $O(n)$. O mesmo valor para o método 3, no entanto, tanto a construção do heap quanto a extração do mínimo requerem maior movimentação dos elementos, além de acessos em posições não-sequenciais, o que o torna menos eficiente que uma busca linear.

Na faixa de valores intermediários, o vencedor é heap de mínimo, do método 3. Nesse intervalo, a construção do heap passa a importar menos para o tempo do algoritmo como um todo, então podemos considerar o tempo como $O(k \lg n)$ amortizado, que é bem mais eficiente que o $O(kn)$ do método 1.

Para valores de k muito próximos de n, como os k menores elementos devem ser encontrados em ordem crescente, o algoritmo se torna basicamente uma ordenação. Nesse caso, o método 3 é comparável a um HeapSort *out-of-place*, com complexidade $O(n \lg n)$. No entanto, o QuickSort acaba sendo mais eficiente na prática, com o caso médio $O(n \lg n)$, mas sem necessidade de modificar o vetor a priori, como o HeapSort, e com acessos sequenciais, que auxilia na otimização pelo compilador e no uso eficiente do cache.

3 Resultados

Execução	k_1	k_2
1	130	495447
2	129	500474
3	131	508571
Média	130	501497

Os valores foram encontrados com o método 0, implementado pelo método da falsa posição, que é análogo a uma busca interpolada.