UNIVERSIDADE FEDERAL DO CEARÁ - CAMPUS SOBRAL

CURSO DE ENGENHARIA DE COMPUTAÇÃO

DISCIPLINA: CONSTRUÇÃO E ANÁLISE DE ALGORITMOS

PROFESSOR: ANTÔNIO JOSEFRAN DE OLIVEIRA BASTOS

SELECT BFPRT: ESCOLHA DO K-ÉSIMO MAIOR ELEMENTO DE UMA LISTA EM TEMPO LINEAR

ALUNO: IZAIAS MACHADO PESSOA NETO

MATRÍCULA: 497372

SOBRAL

2023

1 INTRODUÇÃO

Dado um vetor v qualquer com n números, deseja-se saber qual o i-ésimo maior valor desse vetor. Nesse sentido, com $i = 1$, a saída para esse problema seria o maior elemento do vetor e assim por diante. De modo que quando $i = n$, a saída deve ser o menor elemento de v[1].

Para resolver esse problema, pode ser feita uma decrescente ordenação do vetor e procurado o i-ésimo maior valor na posição $i-1$ (com índice inicial sendo zero). Entretanto, os melhores algoritmos de ordenação tem complexidade $O(nlog(n))$, tornando impossível realizar a seleção desse problema por meio de ordenação em tempo linear (CORMEN *et al.*, 2001).

Para tanto, o algoritmo de seleção linear `Select-BFPRT` de Blum, Floyd, Pratt, Rivest e Tarjan (BLUM *et al.*, 1973) pode ser utilizado para escolher o i-ésimo maior valor de um vetor. Para tanto, esse algoritmo utiliza o `Partition-BFPRT`, que divide o vetor em sucessivos grupos de tamanho R, tira as medianas de cada um dos grupos e as posiciona no início do vetor. Feito isso, o `Partition-BFPRT` chama o `Select-BFPRT` para encontrar qual a mediana das medianas.

Tendo em vista a definição do problema, é necessário escolher um bom tamanho de partição R para o `Partition-BFPRT`, a fim de minimizar o tempo de execução do `Select-BFPRT`. Ao invés de realizar uma análise matemática complexa, os valores de R podem ser comparados experimentalmente, ao avaliar o comportamento do algoritmo `Select-BFPRT` para um série de instâncias aleatória.

Foram geradas uma série de instâncias aleatórias e para cada uma das instâncias foi executado o `Select-BFPRT`. Para cada uma das instâncias foi gerado um i aleatório, executado para $R = (3, 5, 7, 9, 11)$ e os tempos de execução coletados. Por fim, foi feita a média e o desvio padrão das execuções para cada R.

[1] A definição do problema para o i-ésimo menor valor é feita em (CORMEN *et al.*, 2001), pág. 183

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo do trabalho busca reunir o conhecimento utilizado para a implementação dos algoritmos `Partition` (Seção 2.1), `Partition-BFPRT` (Seção 2.2) e `Select-BFPRT` (Seção 2.3). Além disso, é feita uma prova do limite superior para a complexidade temporal do `Select-BFPRT` (Seção 2.4).

2.1 Partition

O algoritmo `Partition` escolhe o último elemento do vetor como pivô e a partir disso, faz com que todos os elementos menores que o pivô fiquem a sua esquerda e os elementos maiores que ele a sua direita. Ao fim da execução do `Partition`, é garantido que o pivô está na mesma posição q que estaria caso o vetor estivesse ordenado. Com isso, sabe-se que o pivô é o k-ésimo maior elemento do vetor, onde $k = r - q + 1$.

2.2 Partition BFPRT

Com essa ideia, é possível criar um algoritmo `Select` que dado a i-ésima posição buscada, chama o `Partition` e consegue saber se o valor i procurado está a esquerda ou a direita do pivô. Com isso, esse algoritmo se chama recursivamente passando um novo i e o lado escolhido do vetor em que o i-ésimo maior elemento se encontra. Entretanto, note que se a escolha de pivôs for ruim, a complexidade do algoritmo deve mudar drasticamente. Por isso, ao invés de escolher sempre o último pivô como no `Partition` ou escolher um pivô aleatório, pode ser usado o algoritmo `Partition-BFPRT`, que tem complexidade temporal $O(n)$.

O `Partition-BFPRT`, divide um vetor $A[p...r]$ em sucessivos grupos de tamanho um tamanho fixo R e os ordena de um por um. Após a ordenação, seleciona as medianas de cada um dos grupos e as posiciona no início do vetor. Com isso, é chamado o algoritmo `Select-BFPRT` para $A[1...R]$, com $i = \lceil r/2 \rceil$, para escolher a mediana das medianas.

2.3 Algoritmo Select BFPRT

A implementação realizada do `Select-BFPRT` (BLUM *et al.*, 1973) é apresentada na Listagem 2.3.1. O algoritmo tem como entrada um vetor, uma posição inicial p, posição final r e também um i, correspondente ao i-ésimo maior valor procurado. Inicialmente, como se procura o i-ésimo maior valor do vetor, e esse algoritmo realiza isso de forma recursiva, o caso base é quando $p = r$, onde o vetor tem somente um elemento, por isso nessa situação o índice dessa única posição do vetor é retornada.

Nesse algoritmo, q é o índice do pivô que divide o vetor, obtido por meio da chamada do `Partition-BFPRT`. Ao rodar o partition, garante-se que pelo menos o pivô vai estar na mesma posição que estaria caso a lista de valores estivesse ordenada. Nesse sentido, a diferença $k = r - q + 1$ entre o fim do vetor r e o índice do pivô q informa qual é a k-ésima maior posição que o valor do índice q ocupa. Caso k for igual ao i procurado, i-ésimo maior valor é o próprio pivô, por isso é retornado seu índice q.

Caso $i > k$, então o valor procurado é menor que o pivô. Com isso, pode ser feita a chamada recursiva do `Select-BFPRT` somente para o lado que é menor que o pivô. Entretanto, quando $i > k$, o i-ésimo maior valor do vetor $A[p...r]$ é diferente do i-ésimo maior valor de $A[p...q-1]$. Isso ocorre porque o valor i que está sendo procurado é o i-ésimo maior valor do vetor por completo, mas ao dividir o vetor em um novo $A[p...q-1]$, agora o valor procurado não pode ser mais o i-ésimo maior valor do vetor, os k maiores valores que são compostos pelo pivô e os elementos a sua direita foram removidos do vetor. Nesse sentido, o valor de i para a nova chamada recursiva é $i - k$. Já caso $i < k$, então o valor procurado é maior que o pivô, por isso, pode ser feita uma chamada recursiva buscando o i-ésimo maior valor a direita do pivô, em $A[q+1...r]$.

```python
def select_bfprt(self, array, p, r, i):
    n = r - p + 1

    if p == r:
        return p

    q = self.partition_bfprt(array, p, r)
    k = r - q + 1

    if i == k:
        return q

    if i > k:
        return self.select_bfprt(array, p, q - 1, i - k)

    return self.select_bfprt(array, q + 1, r, i)
```

Listagem 2.3.1 – Algoritmo Select-BFPRT

Para exemplificar o fato do i-ésimo maior valor de $A[p...r]$ ser diferente do i-ésimo maior valor de $A[p...q-1]$ quando $i > k$, ao buscar o quarto maior $i = 4$ no vetor $v = (1,2,3,7,9)$, que tem pivô igual a 3 e o pivô é o terceiro maior valor do vetor $k = 3$, é feita a chamada recursiva para o lado esquerdo do pivô, porque o procurado é menor que o pivô. Na chamada recursiva se tem $v' = (1,2)$ e não seria possível procurar o quarto maior valor do vetor, porque porém retirados os outros k maiores elementos desse vetor. Então, aa realidade, basta procurar o $i-k$ maior valor do vetor v', que seria $i-k = 4-3 = 1$ o maior elemento do vetor.

Utilizando o mesmo v para procurar $i = 2$, é feita uma chamada recursiva para o lado direito do vetor, porque o valor procurado é maior que o pivô. Como $i < k$, o pivô utilizado na primeira chamada é o terceiro maior elemento do vetor $k = 3$, os outros $k-1$ maiores elementos do vetor estão na seleção $A[q+1...r]$, pois são maiores que o pivô. Ao contrário do que acontece no caso anterior, os $k-1$ maiores elementos não foram removidos, não havendo a necessidade de alterar o i. Logo, o segundo maior valor para $v' = (7,9)$ é também o segundo maior valor para v.

2.4 Complexidade do Select BFPRT

Considerando que a quantidade de valores n na lista é divisível por $R = 5$, desejamos encontrar um limite superior para a complexidade do algoritmo `Select-BFPRT`[1]. Para a ordenação de um grupo, como R é fixo, é gasto uma complexidade constante, logo para ordenar e mover as medinas para o início do vetor é gasto $O(n)$. Ao chamar o `Partition-BFPRT` é feita também uma chamada para `Select-BFRT`, que gasta $T(n/5)$, pois existem $\lceil n/5 \rceil$ grupos.

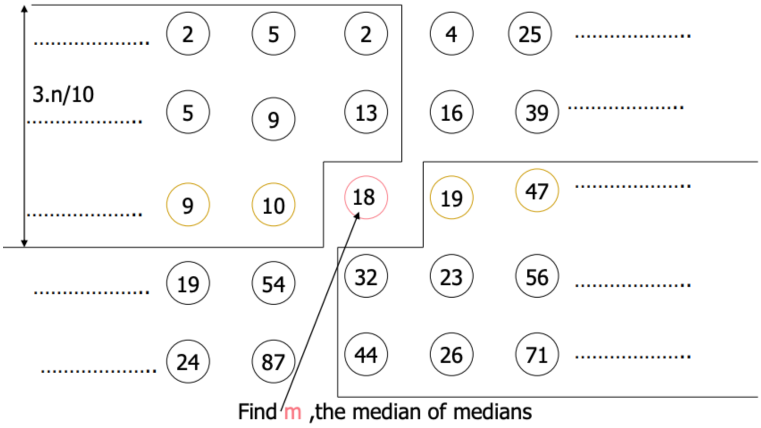
Considerando um $K = 5$, após realizar a média das medianas, no mínimo três elementos dos grupos em que cuja mediana é menor que o pivô são maiores que o pivô. Conforme destacano na ilustração da Figura 1, no mínimo 30% dos valores serão menores que o pivô. Nesse sentido, os outros 70% do vetor podem ser maiores que o pivô. Por isso, para a chamada recursiva, no pior caso, temos $T(7/10 \cdot n)$.

$$\begin{aligned} &T(n) \leq T(\tfrac{n}{5}) + O(n) + T(7 \cdot \tfrac{n}{10}) = \\ &= T(\tfrac{n}{5}) + T(7 \cdot \tfrac{n}{10}) + O(n) \end{aligned} \tag{2.1}$$

Desejamos encontrar um limite superior para a complexidade do `Select-BFRT`, para tanto, assumimos a hipótese que o algoritmo é limitado superiormente por uma função linear

[1] Ideia de prova retirada de (CORMEN *et al.*, 2001), pág. 189

Figura 1 – Ilustração do melhor caso na divisão usando `Partition-BFPRT`



Fonte: retirado de (LABER, 2020)

$T(n) \leq c \cdot n$. A Equação 2.2 descreve um outro limite superior, por meio do qual vamos encontrar um valor para a constante c. Entretanto, a constante a que limita a parte linear é diferente de c.

$$\begin{aligned} & T(n) \leq \tfrac{n}{5} \cdot c + \tfrac{7n}{10} \cdot c + an \\ & = \tfrac{n}{5} \cdot c + \tfrac{7n}{10} \cdot c + an \\ & = \tfrac{9}{10} cn + an \end{aligned} \tag{2.2}$$

Dado o limite superior de $T(n)$, que depende de c e a e também que $T(n) \leq nc$, pode-se definir c em função de a, o que é feito na Equação 2.3. Nessa equação é encontrado que c deve ser no mínimo dez vezes maior que a ($c \geq 10a$).

$$nc = \tfrac{9}{10} cn + an \rightarrow c \geq 10a \tag{2.3}$$

Substituindo com $a = 1$, então $c = 10$, isso faz com que $T(n) \leq 9n + n = 10n$. Portanto, a complexidade do algoritmo `Select-BFPRT` é $O(n)$, pois é limitada por uma função linear.

■

3 METODOLOGIA

Para comparar experimentalmente diferentes valores de K para tamanho das partições, foi elaborado um programa[1] que monta diversas instâncias de tamanho n e salva seus tempos de execução. Foram escolhidos os valores $R = (3, 5, 7, 9, 11)$ e para cada instância gerada é executado o algoritmo `Select-BFPRT`, a fim de comparar o comportamento do algoritmo com diferentes valores de K para uma mesma instância.

Cada instância é composta inicialmente por um vetor v com n valores. O vetor v é montado preenchendo sequencialmente o vetor com valores de 0 até $n-1$ e após isso os valores do vetor são permutados aleatoriamente. Além do vetor, o algoritmo `Select-BFPRT` também recebe um i para encontrar o i-ésimo maior elemento do vetor v, esse i é gerado aleatoriamente, podendo ser entre 1 e n.

O algoritmo tem dois comportamento, gerar uma quantidade de instâncias aleatórias de tamanho n e escrever os resultados em um arquivo *csv*, ou também digitar uma instância de um tamanho qualquer. Caso seja escolhido gerar várias instâncias aleatórias, é possível se aproveitar dos múltiplos núcleos do processador e executar mais de um processo simultaneamente. Por isso, quando se escolhe gerar várias instâncias é perguntado ao usuário a quantidade de processos simultâneos que devem ser instanciados.

Após as execuções dos experimentos, os arquivos *csv* podem ser processados calculando para cada uma das variações o tempo médio das execuções e desvio padrão desse tempo. Com isso, é possível fazer um comparativo que visa embasar a decisão de utilizar um tamanho específico de partição.

[1] O código está disponível em https://github.com/izaiasmachado/cana

4 RESULTADOS

A Tabela 1 apresenta o resultado da execução de 50000 instâncias, com tamanho de instância igual a $n = 10000$. Os resultados mostram que os Os tamanhos de partição 9 e 11 tem os tempos de execução muito baixos se comparado com os demais tamanhos de partição. Além disso, na Tabela 1, o destaque vai para o tamanho de partição igual a 11, pois obteve também o menor desvio padrão.

Espera-se com essa interpretação que os valores com menor desvio padrão se comportem com tempos médios mais uniformes. Dito isso, com o tamanho de partição 5 apesar de ter um tempo de execução maior que 7, 9 e 11, por ter um maior desvio padrão podem ser encontrados casos com tempo de execução bem menores que as demais partições, o mesmo ocorre para o maior tempo de execução, em que os demais tamanhos de partição (exceto o de 3) podem ter vantagem.

Tabela 1 – Resultado primeiro experimento

Tamanho de Partição (R)	Tempo Médio de Execução (s)	Desvio Padrão
3	0,342949	0,015350
5	0,247412	0,009311
7	0,229863	0,008160
9	0,224542	0,007553
11	0,222829	0,007362

Fonte: elaborado pelo autor (2023).

Já na Tabela 2, foram feitas 12000 instâncias, cada uma com $n = 100000$. Esse tamanho maior de instâncias permite compreender melhor o comportamento do algoritmo. Como a complexidade encontrada na Seção 2.4 deste trabalho é um limite superior, não se pode simplesmente comparar valores entre as duas tabelas de resultado. Apesar disso, o comportamento dos tempos de execução das instâncias de tamanho 9 e 11 são similares ao que foi encontrado no primeiro experimento.

Mesmo que o comportamento dos tempos de execução tenha sido similar, o desvio padrão cresceu muito com na Tabela 2 se comparada com a Tabela 1. Isso pode ser explicado porque a quantidade de grupos cresce na ordem de 10 do primeiro experimento para o segundo. Isto é, há muitos casos com bons tempos de execução, mas da mesma forma, existem muitos casos com tempos ruins de execução. Dito isso, com tamanho de instâncias $n = 100000$ o destaque vai para um tamanho de partição maior $R = 11$, pois teve um tempo de execução e o menor desvio padrão.

Tabela 2 – Resultado segundo experimento

Tamanho de Partição (R)	Tempo Médio de Execução (s)	Desvio Padrão
3	5,093321	12,740226
5	3,700169	16,541186
7	3,265799	12,181696
9	3,162925	12,181599
11	3,330611	17,046008

Fonte: elaborado pelo autor (2023).

5 CONCLUSÕES

Os resultados dos experimentos, por utilizarem um tamanho fixo de instância não permitiu que fosse encontrado experimentalmente um caso médio para a complexidade temporal desse algoritmo, visto que seria necessário variar também o tamanho da instância. Mas nesse caso, foi possível observar um tempo de execução médio para os algoritmos dado um tamanho de entrada n e também um tamanho de partição R.

Nesse contexto, observou-se que para as instâncias com tamanho $n = 10000$, ordenar grupos de tamanho $R = 11$ pode ser mais vantajoso para encontrar um tempo médio de execução mais consistente. Ao utilizar valores com tempo igualmente baixo, mas desvio padrão maior, espera-se que o tempo de execução seja mais distante dos tempos médios, podendo haver tempos bons e também tempos ruins. Caso seja uma vantagem para uma aplicação o fato de ter tempos bons de execução esporadicamente, pode ser utilizado um $R = 7$ (para instâncias de tamanho $n = 10000$).

Outrossim, no segundo experimento foi utilizado um tamanho de instância $n = 100000$, em o tamanho de partição $R = 9$ obteve o menor desvio padrão e também menor tempo médio de execução. Caso a aplicação possa se beneficiar de tempos bons esporadicamente, pode se utilizado $K = 11$, que tem o maior desvio padrão e tempo médio de execução não diferente do obtido para $R = 9$.

Em resumo, a escolha tamanho de partição R deve ser baseada nas características específicas do problema e nos recursos disponíveis. O limite superior obtido na Seção 2.4 pode ser útil para realizar uma implementação do `Select-BFPRT` a fim de atender uma aplicação específica. Já as análises e resultados dispostos no Capítulo 4 devem ser considerados ao escolher um tamanho de partição. Além disso, o código desenvolvido no Capítulo 3 pode ser utilizado para experimentar em um ambiente próprio e o código modificado para avaliar como os diferentes tamanhos de partição se comportam para uma entrada com outras características.

REFERÊNCIAS

BLUM, M.; FLOYD, R. W.; PRATT, V. R.; RIVEST, R. L.; TARJAN, R. E. *et al.* Time bounds for selection. **J. Comput. Syst. Sci.**, v. 7, n. 4, p. 448–461, 1973. Disponível em: http://people.csail.mit.edu/rivest/pubs/BFPRT73.pdf.

CORMEN, T.; LEISERSON, C.; RIVEST, R.; STEIN, C. **Introduction To Algorithms**. MIT Press, 2001. (Mit Electrical Engineering and Computer Science). ISBN 9780262032933. Disponível em: https://books.google.com.br/books?id=NLngYyWFl_YC.

LABER, E. S. **Mediana em Tempo Linear**. 2020. Disponível em: https://www-di.inf.puc-rio.br/~laber/median-lineartime.pdf.