Exercício 1

Qual é o número esperado de comparações executadas na linha 6 do algoritmo?

Considerando que, na iteração i (para $i \in \{2, \dots, n\}$), a variável `maior` guarda

$$\max\{v[1], \dots, v[i-1]\} \tag{1}$$

Logo, `v[i] > maior` é verdadeira apenas quando $v[i]$ é o máximo de $v[1..i]$, o que ocorre com probabilidade $\frac{1}{i}$, pois a entrada é uma permutação uniforme. Assim, a comparação `v[i] < menor` (linha 6) é executada na iteração i se, e somente se, $v[i]$ *não* é o máximo de $v[1..i]$:

$$P(\text{executar linha 6 na iteração } i) = 1 - \frac{1}{i}. \tag{2}$$

Seja X_6 o número total de comparações na linha 6 e defina a série harmônica $H_n := \sum_{k=1}^{n} \frac{1}{k}$. Pela linearidade da esperança,

$$\begin{aligned}
E[X_6] &= \sum_{i=2}^{n} P(\text{executar linha 6 na iteração } i) && (3)\\
&= \sum_{i=2}^{n} \left(1 - \frac{1}{i}\right) && (4)\\
&= (n-1) - (H_n - 1) && (5)\\
&= n - (H_n) && (6)
\end{aligned}$$

Portanto,

$$\boxed{E[X_6] = n - H_n}. \tag{7}$$

Qual é o número esperado de atribuições efetuadas na linha 7 do algoritmo?

A atribuição da linha 7, `menor <- v[i]`, ocorre se, e somente se, `v[i] > maior` é falsa e `v[i] < menor` é verdadeira. Como `menor` guarda $\min\{v[1], \dots, v[i-1]\}$, a atribuição acontece exatamente quando $v[i]$ é o novo mínimo de $v[1..i]$.

Com entrada aleatória (permutação uniforme), a probabilidade de $v[i]$ ser o mínimo de $v[1..i]$ é $\frac{1}{i}$. Logo, se X_7 é o número total de atribuições/execuções na linha 7,

$$\begin{aligned}
E[X_7] &= \sum_{i=2}^{n} P(\text{executar linha 7 na iteração } i) && (8)\\
&= \sum_{i=2}^{n} \frac{1}{i} && (9)\\
&= H_n - 1. && (10)
\end{aligned}$$

Portanto,

$$\boxed{E[X_7] = H_n - 1}. \tag{11}$$

Exercício 3

Estratégia do Algoritmo

A estratégia ideal é usar a mediana como pivô em um algoritmo de *divide and conquer*, similar ao Quickselect, garantindo que o maior subproblema tenha tamanho no máximo metade do atual.

Algoritmo

Assumimos a existência de duas funções auxiliares:

- `Mediana(A, p, r)`: A função "caixa-preta" dada, que retorna a mediana do subvetor $A[p..r]$ em tempo linear.
- `Particiona(A, p, r, m)`: Uma função padrão de particionamento (como a do Quicksort) que rearranja o subvetor $A[p..r]$ em torno de um pivô m, colocando elementos menores que m à sua esquerda e maiores à sua direita. Ela retorna o índice final q onde o pivô m foi colocado. Esta operação também é linear.

Algoritmo 1: `k-esimoMenor`(A, p, r, k)

Data: Vetor A, índices p, r (início e fim), inteiro k
Result: k-ésimo menor elemento de $A[p..r]$

```
if p = r then
    return A[p];
end
m ← Mediana(A, p, r);
q ← Particiona(A, p, r, m);
i ← q − p + 1 ;                    // i é a ordem do pivô em A[p..r]
if k = i then
    return A[q];
end
else if k < i then
    return k-esimoMenor(A, p, q − 1, k);
end
else
    return k-esimoMenor(A, q + 1, r, k − i);
end
```

Funcionamento do Algoritmo

O algoritmo `k-esimoMenor` opera de forma recursiva, aplicando a estratégia de divide and conquer:

1. **Divisão:** A mediana m do subvetor $A[p..r]$ é encontrada usando a função caixa-preta.

2. **Conquista:** O subvetor é particionado em torno da mediana m. Após o particionamento, m está em sua posição correta q como se o vetor estivesse ordenado. Todos os elementos em $A[p..q-1]$ são menores que m, e todos em $A[q+1..r]$ são maiores.

3. **Combinação:** O algoritmo então decide em qual subproblema continuar a busca:
   - A posição (ordem) do pivô no subvetor atual é $i = q - p + 1$.
   - Se $k = i$, então o pivô é o elemento que procuramos, e ele é retornado.
   - Se $k < i$, o k-ésimo menor elemento deve estar no subvetor à esquerda, $A[p..q-1]$. O algoritmo é chamado recursivamente para esta parte, procurando ainda pelo k-ésimo elemento.
   - Se $k > i$, o elemento está no subvetor à direita, $A[q+1..r]$. Como já descartamos i elementos (o pivô e os que estão à sua esquerda), agora procuramos pelo $(k-i)$-ésimo menor elemento neste novo subvetor.

4. **Caso Base:** A recursão para quando o subvetor contém apenas um elemento ($p = r$), que é trivialmente o elemento procurado.

Prova de Corretude

Provamos a corretude por indução no tamanho do subvetor, $n = r - p + 1$.

Base: Se $n = 1$, então $p = r$ e k deve ser 1. O algoritmo entra na primeira condição e retorna $A[p]$, que é o primeiro (e único) menor elemento. A base é válida.

Hipótese de Indução (HI): Assuma que `k-esimoMenor` retorna o resultado correto para todos os subvetores de tamanho menor que n.

Passo Indutivo: Considere um subvetor de tamanho $n > 1$. O algoritmo encontra a mediana m e a posiciona no índice q através do particionamento. Por definição do particionamento, $A[q]$ é o $(q-p+1)$-ésimo menor elemento de $A[p..r]$. Seja $i = q-p+1$.

- **Caso 1:** $k = i$. O algoritmo retorna $A[q]$, que é o i-ésimo menor elemento. Como $k = i$, o resultado está correto.

- **Caso 2:** $k < i$. O k-ésimo menor elemento de $A[p..r]$ deve ser menor que o i-ésimo, $A[q]$. Portanto, ele deve estar no subvetor $A[p..q-1]$. O tamanho deste subvetor é $q - p = i - 1 < n$. Pela HI, a chamada recursiva `k-esimoMenor(A, p, q-1, k)` retornará o elemento correto.

- **Caso 3:** $k > i$. O k-ésimo menor elemento é maior que $A[q]$. Ele deve estar no subvetor $A[q+1..r]$. Já existem i elementos menores ou iguais a $A[q]$. Portanto, estamos procurando o $(k-i)$-ésimo menor elemento no subvetor da direita. O tamanho deste subvetor é $r - q < n$. Pela HI, a chamada `k-esimoMenor(A, q+1, r, k-i)` retornará o elemento correto.

Como todos os casos levam a um resultado correto, o algoritmo está correto por indução.

Análise de Complexidade

Seja $T(n)$ o tempo de execução do algoritmo para um vetor de tamanho n. O custo em cada chamada recursiva é a soma dos custos de suas operações:

1. `Mediana`: $O(n)$ (dado pelo enunciado).
2. `Particiona`: $O(n)$ (particionamento padrão é linear).
3. Chamada recursiva: Ocorre em um subproblema.

O pivô escolhido é a mediana. Assumindo que o tamanho n do vetor é uma potência de 2, a mediana divide o vetor em duas metades quase perfeitas. O maior subproblema no qual a recursão pode ser chamada terá tamanho no máximo $n/2$. Com isso, obtemos a seguinte recorrência para o tempo de execução:

$$T(n) = T\left(\frac{n}{2}\right) + cn.$$

Pelo Teorema Mestre (não lembro se foi ensinado em aula mas vi aqui, Moore, CS 2123: Divide-Conquer-Glue Algorithms [1]), com $a = 1$, $b = 2$, $k = 1$ e $a < b^k$, segue que

$$\Theta(n^k) = \Theta(n^1) \tag{12}$$

Portanto,

$$\boxed{T(n) = \Theta(n)}. \tag{13}$$

Logo, o algoritmo é linear.

Referências

[1] Tyler Moore. Quickselect algorithm — Divide-Conquer-Glue Algorithms (CS 2123). University of Tulsa. Disponível em: https://secon.utulsa.edu/cs2123/slides/dc2p.pdf.