Lista 2 - Algoritmos 1

João Pedrosa
2019006752

Março de 2021

1 Questão 1

Iremos fazer uma busca binária adaptada. Usaremos i e j como os limites da área de interesse de nossa busca. Inicialmente $i = 0$ e $j = n$. Iremos sempre analisar o índice m que se encontra no meio da área de interesse $m = \frac{i+j}{2}$ e os dois elementos adjacentes à ele. Se os adjacentes forem ambos menores que $A[m]$, então $m = p$ e podemos parar o algoritmo. Caso os 3 elementos estejam ordenados de forma crescente, então $p < m$ e podemos fazer $j = m$. Caso os 3 elementos estejam ordenados de forma decrescente, $p > m$ e podemos fazer $i = m + 1$. Basta fazer isso até termos $m = p$Perceba que sempre dividimos a nossa área de interesse por 2 a cada passo do algoritmo. Logo, a equação de recorrência do algoritmo é $T(n) = T(\frac{n}{2}) + 1$, a mesma equação de recorrência de uma busca binária, portanto com complexidade $\in \mathcal{O}(\log n)$.

2 Questão 2

Iremos executar um guloso simples. Percorreremos o vetor inteiro, mantendo duas varíaveis:

- sp: o menor preço de compra até então. Inicializada como infinito.
- bd: o maior lucro obtido até então. Inicializada como 0.

Perceba que se formos vender na data i, sempre é ótimo comprarmos a ação em seu menor preço até i. Sendo assim, a melhor venda possível na data i será $p(i) - p(sp)$. Logo, a cada posição do vetor, sabemos que $bd = max(bd, p(i) - p(sp))$. Após percorrermos o vetor inteiro, bd guardará o maior lucro possível. Como percorremos o vetor uma única vez, temos complexidade $\in \mathcal{O}(n)$.

3 Questão 3

Iremos fazer uma busca binária adaptada. Usaremos i e j como os limites da área de interesse de nossa busca. Inicialmente $i = 0$ e $j = n$. Iremos sempre analisar o índice m que se encontra no meio da área de interesse $m = \frac{i+j}{2}$ e os dois elementos adjacentes à ele. Se os adjacentes forem ambos menores que $A[m]$, então $m = p$ e podemos parar o algoritmo. Caso os 3 elementos estejam ordenados de forma crescente, então $p < m$ e podemos fazer $j = m$. Caso os 3 elementos estejam ordenados de forma decrescente, $p > m$ e podemos fazer $i = m + 1$. Basta fazer isso até termos $m = p$Perceba que sempre dividimos a nossa área de interesse por 2 a cada passo do algoritmo. Logo, a equação de recorrência do algoritmo é $T(n) = T(\frac{n}{2}) + 1$, a mesma equação de recorrência de uma busca binária, portanto com complexidade $\in \mathcal{O}(\log n)$.

4 Questão 4

Perceba que, mesmo que não andemos o máximo de quilômetros possíveis todos os dias, começaremos a caminhada no dia seguinte podendo andar os mesmos d quilômetros, a única diferença é que estaremos mais longe do destino. Os quilômetros que economizamos em um dia não podem ser usados no dia seguinte, logo, nunca compensa pararmos antes de andarmos o máximo possível e qualquer algoritmo que não ande o máximo possível todos os dias fará pelo menos o mesmo número de paradas que o guloso.

5 Questão 5

Perceba que se $p > q$ então $p^n + q^{n+1} < q^n + p^{n+1}$. Isso significa que sempre compensa comprar a licença mais cara primeiro. Portanto, basta comprarmos as licenças por ordem decrescente de fator de multiplicação. A complexidade será a complexidade de ordenar um vetor: $\mathcal{O}(n \log n)$.

6 Questão 6

Primeiramente, determinaremos q como a diferença constante entre os dois números. Se $x_1 - x_0 = x_2 - x_1$, então $q = x_1 - x_0$. Caso contrário, o número que falta está entre $x_0 e x_2$ basta olharmos o valor de $x_3 - x_2$ e saberemos qual é o número que está faltando.

A partir daí, faremos uso de uma idéia parecida com a da questão 1. Usaremos i e j como os limites da área de interesse de nossa busca. Inicialmente $i = 0$ e $j = n$. Iremos sempre analisar o índice m que se encontra no meio da área de interesse $m = \frac{i+j}{2}$. Sabemos que $A[m]$ deveria ser igual à $x_0 + m * q$. Se essa condição for verdadeira, então o número faltante vem depois do índice m e fazemos $i = m + 1$. Se essa condição for falsa, o número faltante vem antes do índice m e fazemos $j = m$. Executamos esse algoritmo enquanto $i < j$. Ao fim do algoritmo, i irá guardar o índice em que deveria estar o número faltante na sequência. Novamente, dividimos nossa área de interesse por 2 a cada iteração. Equação de recorrência $T(n) = T(\frac{n}{2}) + 1$ e complexidade $\in \mathcal{O}(\log n)$.

7 Questão 7

Iremos resolver esse problema recursivamente fazendo uso de programação dinâmica. Nosso estado da PD será definido como $C(i, j)$, onde i é o pedágio que estamos analisando e j é o quilômetro da rodovia. Iremos começar analisando o primeiro pedágio e o primeiro quilômetro da rodovia. Daí podemos transicionar para dois estados:

- Colocar o pedágio i no quilômetro j e retornar $r_i + C(i + 1, j + 10)$.
- Não posicionar o pedágio i ainda e retornar $C(i + 1, j + 1)$.

Soluções com programação dinâmica são um pouco mais complexas de se explicar por extenso, então segue um pseudocódigo:

Algorithm 1 Questão 7

INPUT: Array R com os valores para o pedágio, inteiro N representando a quantidade de pedágios e inteiro M representando o tamanho da rodovia
OUTPUT: Maior valor possível que pode ser cobrado

```
G ← Matriz de tamanho N × M inicializada com -1
procedure DP(i, j)
    if i = N then
        return 0
    end if
    if j >= M then
        return −∞
    end if
    if G[i][j] ≠ −1 then
        return G[i][j]
    end if
    return G[i][j] = max(DP(i + 1, j + 10) + R[i], DP(i, j + 1))
end procedure
return DP(0, 0)
```

Perceba que cada estado será acessado no máximo uma vez, portanto temos complexidade de tempo igual à de espaço: $\mathcal{O}(nm)$.