

Segunda Prova - Análise de Algoritmos

Prof. André Guedes
11 de dezembro de 2024

1. (15 pontos) O problema de encontrar o mínimo de um vetor pode ser resolvido com as técnicas de algoritmo guloso ou programação dinâmica? Se sim, escolha uma, apresente a solução e diga se isso é melhor que o algoritmo sequencial. Justifique.
2. (30 pontos) Proponha um algoritmo de programação dinâmica para o cálculo do coeficiente binomial $\binom{n}{k}$ baseado na recorrência abaixo (à esquerda). Discuta sua eficiência em relação ao algoritmo $CB(n, k)$ abaixo (à direita), e com a equação $\binom{n}{k} = \frac{n!}{k!(n-k)!}$, apresentando complexidades e argumentos que justifiquem a eficiência ou não do uso de cada um deles. Considere, para a equação, que o cálculo de $n!$ executa $n - 1$ multiplicações para todo $n \in \mathbb{N}$ e que devemos contar o número de multiplicações e divisões.

	$CB(n, k)$
$\binom{n}{k} = \begin{cases} 1, & \text{se } k = 0, \\ \binom{n-1}{k-1} + \binom{n-1}{k}, & \text{se } 1 \leq k \leq n, \\ 0, & \text{caso contrário.} \end{cases}$	Se $k = 0$ Devolva 1 Se $k \geq 1$ e $k \leq n$ $r \leftarrow CB(n-1, k-1) + CB(n-1, k)$ Devolva r Devolva 0

3. (15 pontos) Explique o funcionamento dos Algoritmos Gulosos. Esta técnica poderia ser usada para calcular o coeficiente binomial usando a recorrência da Questão 2? Dê um exemplo de um algoritmo guloso. Justifique.
4. No problema do troco (visto em aula) temos um conjunto de n moedas com valores $v_1, v_2, \dots, v_n$ de tal forma que $v_1 < v_2 < \dots < v_n$, e um valor N . Queremos formar o valor N usando o menor número de moedas. Podemos assumir que temos quantidade suficiente de moedas de cada valor e que é possível gerar o valor N . A seguinte recorrência resolve o problema de encontrar o número mínimo de moedas, onde $M(n, N)$ resolve o problema com moedas $v_1, v_2, \dots, v_n$ e valor N .

$$M(n, N) = \begin{cases} 0, & \text{se } N = 0 \\ \infty, & \text{se } N > 0 \text{ e } n = 0 \\ M(n-1, N), & \text{se } N, n > 0 \text{ e } v_n > N, \\ \min\{M(n-1, N), M(n, N - v_n) + 1\}, & \text{se } N, n > 0 \text{ e } v_n \leq N. \end{cases}$$

Considere uma solução por programação dinâmica que usa esta recorrência.

- (a) (10 pontos) Como é a estrutura de dados para armazenar os subproblemas? Como é a indexação?
- (b) (15 pontos) Qual o custo de execução deste algoritmo em função de n e N ?
- (c) (15 pontos) Que modificações são necessárias para que possamos recuperar o conjunto de moedas associado à resposta do problema?

1 Gabarito

- 1 O algoritmo sequencial pode ser visto como um algoritmo de Programação Dinâmica, usando a recorrência

$$M(i) = \begin{cases} v(i), & \text{se } i = 1, \\ \min\{M(i-1), v(i)\}, & \text{se } i > 1, \end{cases}$$

onde $M(i)$ nos dá o mínimo do vetor $v[1..i]$. Note que não precisamos guardar os subproblemas em um vetor, já que só usamos o último. Podemos usar só uma variável.

Com esta mesma recorrência, podemos pensar que é um Algoritmo Guloso, ou modificar para termos um Backtracking, mas não temos “escolhas” entre ramos.

Divisão e conquista - divide o vetor ao meio escolhe o mínimo de cada lado e pegue o menor deles. O custo é exatamente o mesmo do algoritmo sequencial.

2

CB-Din(n, k)

Se $k > n$

 Devolva 0

$P \leftarrow$ matriz indexada por $[0..n] \times [0..k]$

Para i de 0 até n

$P[i, 0] \leftarrow 1$

Para i de 0 até $k-1$

$P[i, i+1] \leftarrow 0$

Para i de 1 até n

 Para j de 1 até $\min\{i, k\}$

$P[i, j] \leftarrow P[i-1, j-1] + P[i-1, j]$

Devolva $P[n, k]$

O algoritmo CB-Din(n, k) tem custo $\Theta(nk)$, já que a matriz usada tem dimensões $(n+1) \times (k+1)$ e o custo de preencher cada posição é $\Theta(1)$.

O custo do algoritmos dado, CB(n, k) é $\mathcal{O}(2^{\max\{k, n-k\}})$, já que os ramos da árvore binária tem alturas $n-k$ e k . Mas podemos assumir que $\max\{k, n-k\} = \mathcal{O}(n)$ e então podemos dizer que o custo é $\mathcal{O}(2^n)$.

Como calcular $n!$ tem custo $n-1$, calcular $k!$ tem custo $k-1$, calcular $(n-k)!$ tem custo $n-k-1$ e ainda temos uma multiplicação e uma divisão, o custo de calcular o coeficiente binomial usando a equação dada é $(n-1) + (k-1) + (n-k-1) + 2$ que é igual a $2n+1$, que é $\Theta(n)$.

Portanto, o cálculo da equação é assintoticamente melhor que os demais. O algoritmo de Programação Dinâmica é assintoticamente melhor que o o algoritmo dado.

Apesar disso, calcular a equação pode gerar problemas de precisão numérica.

- 3 Algoritmos Gulosos funcionam quando temos “escolhas” para se fazer e estas escolhas são feitas baseadas em algum critério local (“heurística” ou “oráculo”).

Não podemos usar algoritmos gulosos na recorrência do coeficiente binomial, pois não temos “escolha” nesta recorrência.

4

- (a) Como os subproblemas são referenciados como $M(i, j)$, e i e j são naturais de 0 a n e N , respectivamente, e o valor de cada subproblema é um natural, a estrutura de dados é uma matriz M de naturais indexada de por $[0..n] \times [0..N]$.

- (b) Como o cálculo de cada posição da matriz é $\Theta(1)$ — condicionais e acesso a no máximo 2 posições da matriz — e o número de posições a serem preenchidas é $(n+1)(N+1)$, então o custo do algoritmo é $T_M(n, M) = (n+1)(N+1) \times \Theta(1) = \Theta(nN)$. Observe que este algoritmo é *pseudo-polinomial*.
- (c) É preciso guardar as decisões tomadas em cada passo. Podemos criar uma matriz auxiliar, P , de mesmo tamanho e indexação que a anterior, e na posição $P[i, j]$ guardamos $i-1$, caso $M[i-1, j]$ seja o ganhador do min, ou i , caso o ganhador seja $M[i, j-v_i] + 1$. Ou seja,

$$P[i, j] = \begin{cases} i-1, & \text{se } M[i, j] = M[i-1, j], \\ i, & \text{caso contrário.} \end{cases}$$

Para recuperar as moedas, basta seguir desde a posição n, N e indo para trás seguindo a seguinte regra. Se está na posição $[i, j]$ e $P[i, j] = i$, use uma moeda de valor v_i e vá para a posição $[i, j-v_i]$. Caso $P[i, j] = i-1$, vá para a posição $[i-1, j]$ sem usar moedas. Faça isso até que $i = 0$ ou $j = 0$.