Projeto de Algoritmo com Implementação nº 2

Lucas Valente Viegas de Oliveira Paes - RA 220958

20 de dezembro de 2020

1 Prova de subestrutura ótima

Suponha que C é um caminho ótimo até determinado bloco b_n, sendo n o número blocos desde o chão até b_n, incluindo b_n. Considere $|C|$ o custo de um caminho e $|b|$ o custo de um bloco. Teremos, então, dois casos:

$n = 1$ Nesse caso, b_n está no chão, na primeira linha da matriz de escalada. Assim, simplesmente, $|C| = |b_n|$.

$n > 1$ Nesse caso, há pelo menos um bloco b_i, com $i < n$, além de b_n em C, de modo que este possa ser decomposto em pelo menos dois caminhos: C_1, do chão até b_i, e C_2, de b_i até b_n. Para representar essa situação, diremos que $C = C_1 + C_2$.

No primeiro caso, se C não fosse caminho ótimo, teríamos uma contradição trivial. Logo, C é caminho ótimo.

No segundo caso, suponha que C_1 não é caminho ótimo. Então, existe C_0 de forma que $|C_0| < |C_1|$. Assim, existe $C' = C_0 + C_2$ de forma que $|C'| = |C_0| + |C_2| < |C_1| + |C_2| = |C|$. Como isso é uma contradição, então C_1 é um caminho ótimo. Com raciocínio análogo, prova-se que C_2 também é caminho ótimo.

Ainda no segundo caso, vale a pena observar que estamos apenas subindo na parede de escalada, passando somente uma vez por cada um de seus níveis. Assim, é impossível que um bloco b_i se repita em um mesmo caminho C. Portanto, quaisquer caminhos C_1 e C_2 tal que $C = C_1 + C_2$ são independentes, ou seja, não compartilham blocos.

Eis a prova de subestrutura ótima do problema da escalada, além da independência de seus subproblemas.

2 Recorrência do problema

Seja $C(b_{i,j})$ termo que representa o menor custo para se ir do chão até um bloco $b_{i,j}$ localizado na i-ésima linha e j-ésima coluna da matriz de largura m. Além disso, seja $|b_{i,j}|$ o custo desse bloco. Então,

$$C(b_{i,j}) = \begin{cases} 0, & \text{se } i = 0 \\ \infty, & \text{se } j \notin [1, m] \\ min\{C(b_{i-1,j-1}), C(b_{i-1,j}), C(b_{i-1,j+1})\} + |b_{i,j}|, & \text{de outra forma} \end{cases}$$

No primeiro caso (base), considera-se zero o custo de permanecer no chão (o zerésimo nível da parede de escalada).

No segundo caso (base), considera-se infinito o custo de chegar num bloco lateralmente fora da parede de escalada. Isso facilita o uso de índices horizontais sem a introdução de condicionais complicados no terceiro caso, para situações em que $j = 0$ e $j = m + 1$.

No terceiro caso, queremos encontrar o menor custo até $b_{i,j}$, considerando que só conseguimos alcançá-lo a partir do bloco diretamente abaixo dele ou dos blocos nas suas diagonais inferior direita e esquerda, $b_{i-1,j}$, $b_{i-1,j-1}$ e $b_{i-1,j+1}$, respectivamente.

3 Projeto de algoritmo

Primeiramente, pensei em transformar o problema em uma busca por menor caminho em grafo, criando um vértice por bloco e arestas para cada passagem possível entre blocos. Descartei essa solução após perceber que transformar a entrada dada em formato de matriz em um grafo seria muito complicado, podendo aumentar bastante a complexidade de memória.

Ao analisar a recorrência do problema, percebi que, no i-ésimo nível, dependemos apenas dos custos do nível imediatamente abaixo para calcular os custos de se chegar em cada um de seus j blocos. Assim cheguei na solução abaixo:

> *Vamos inicializar um vetor de custos C de tamanho $m + 2$ com zeros, exceto na primeira e última posições, inicializadas com infinito. Então, para cada nível da matriz de escalada, de baixo para cima, vamos atualizar os valores de C de modo que sua $(j + 1)$-ésima posição contenha o custo de se chegar ao j-ésimo bloco daquele nível, incluindo o próprio bloco. Para calcular o custo total de um bloco, basta encontrar o mínimo custo entre os blocos*

a partir dos quais o alcançamos, ou seja, do bloco imediatamente abaixo e de suas diagonais inferiores, e somar com o custo do próprio bloco, armazenado na matriz de escalada. Ao final dos n níveis, basta retornar o menor valor armazenado em C.

O algoritmo desenvolvido usa programação dinâmica bottom-up, uma vez que soluciona-se os resultados dos subproblemas (blocos inferiores) antes de solucionar-se os problemas que deles dependem (blocos superiores).

4 Análise de complexidade

Quanto à complexidade de espaço, a única variável que depende das dimensões do problema é o vetor C, com $m + 2$ posições. Logo, o algoritmo tem complexidade espacial $O(m)$.

Quanto à complexidade de tempo, para cada um dos n níveis da matriz, percorremos o cada um de seus m elementos uma vez, fazendo três acessos aos custos de seus vizinhos inferiores e calculando seu mínimo. Por isso, o algoritmo tem complexidade temporal $O(nm)$.

Se considerarmos que $m \in O(n)$, então o algoritmo tem complexidade espacial $O(n)$ e temporal $O(n^2)$.

5 Notas sobre a implementação

A inicialização de C (caso base) foi alterada para conter os valores da primeira linha da matriz no lugar de zeros. Assim, evita-se sua atualização para a situação em que $min\{C(b_{i-1,j-1}), C(b_{i-1,j}), C(b_{i-1,j+1})\} = 0$, poupando-se uma iteração e começando de fato a partir da segunda linha. Isso faz com que se atualize C apenas $n - 1$ vezes ao invés de n, reduzindo o tempo de execução por uma constante.