INTRODUÇÃO À OTIMIZAÇÃO POR DIVISÃO E CONQUISTA

Pedro V. Sousa da Silva, UFPR
André L. P. Guedes, UFPR

Resumo
Este artigo apresenta a otimização por divisão e conquista (Divide and Conquer Optimization), uma técnica geralmente usada para acelerar algoritmos de programação dinâmica. Sua aplicação é possível em problemas cujas funções de custo satisfazem propriedades de monotonicidade. O trabalho apresenta a ideia desta técnica, uma formalização, uma implementação generalizável e exemplos de aplicação. Este artigo é fortemente inspirado no texto Algoritmos em matrizes monótonas e Monge convexas, de Victor Sena Molero [7].

1. Problema inicial

Imagine o cenário onde temos uma matriz A de dimensões $N \times M$ e o objetivo seja determinar, para cada linha, a posição de seu menor elemento. Sem nenhuma condição a mais, não seria possível executar essa tarefa em tempo melhor que $\Omega(NM)$.

Suponha agora que a matriz não seja fornecida explicitamente, mas sim por meio de uma função

$$C(i, j) = A[i][j], \text{ para } 1 \leq i \leq N \text{ e } 1 \leq j \leq M$$

que pode ser avaliada em tempo O(1), assim sua leitura e armazenamento não impactam na complexidade do algoritmo. Além disso, suponha que C satisfaça uma propriedade de monotonicidade em um de seus índices. Nessas condições, é possível acelerar essa tarefa para $O((N + M) \log (N))$ por meio da técnica de otimização por divisão e conquista.

Uma descrição mais formal é dada a seguir.

Definição 1 (Vetor de índices de mínimos das linhas): Seja A uma matriz de dimensões $N \times M$. Definimos o vetor opt como,

$$\text{opt}[i] = \min \{j \mid A[i][j] \leq A[i][j'], 1 \leq j' \leq M\}$$

ou seja, $opt[i]$ guarda a posição da coluna que atinge o valor mínimo na linha i. Note que, em caso de empate, escolhe-se a posição mais à esquerda.

Definição 2 (Monotonicidade): Dizemos que o vetor opt é monotônico se $opt[i] \leq opt[i+1]$ para $1 \leq i < N$.

Desse modo, quando a monotonicidade é satisfeita para o vetor de índices de mínimos das linhas (opt) de uma matriz A e esta pode ser computada implicitamente, a otimização por divisão e conquista permite calcular todo o vetor opt em tempo $O((N + M) \log (N))$.

2. Ideia do algoritmo

Considere que desejamos computar o vetor opt para linhas com índices no intervalo $[l_i, r_i]$ e que as possíveis colunas ótimas atualmente estão no intervalo $[l_j, r_j]$.

Inicialmente, seja i a linha no meio do intervalo $[l_i, r_i]$, para esta linha encontramos o valor $opt[i]$ de maneira ingênua (busca linear). (Veja a Figura 1a)

Em seguida, o problema é dividido em dois subproblemas: um para linhas anteriores a i e outro para linhas posteriores a i. Sabemos pela monotonicidade de opt, que para todo $i' < i$ vale $opt[i'] \leq opt[i]$, ou seja, para todas as linhas entre $[l_i, i - 1]$ podemos limitar a busca da coluna ótima pela direita por $opt[i]$. De maneira similar, como para todo $i'' > i$ vale $opt[i''] \geq opt[i]$, podemos utilizar $opt[i]$ para restringir a busca pela esquerda para todas as linhas entre $[i + 1, r_i]$. Assim, esses dois subproblemas podem ser resolvidos de maneira recursiva até que todas as linhas tenham sido processadas. (Veja as Figuras 1b e 1c).

Em resumo, uma vez resolvida a linha do meio, podemos utilizar sua resposta para resolver recursivamente suas duas metades, com a vantagem de reduzir o intervalo de busca da coluna ótima. O Código 1 utiliza essa ideia.

```cpp
void solve(int li, int ri, int lj, int rj) {
    if (lj > rj) return;

    int i = (li + ri) / 2;
    opt[i] = lj;

    for (int j = lj; j <= rj; j++) {
        if (C(i, j) <= C(i, opt[i]))
            opt[i] = j;
    }

    solve(li, i - 1, lj, opt[i]);
    solve(i + 1, ri, opt[i], rj);
}
```

CÓDIGO 1: IMPLEMENTAÇÃO DA OTIMIZAÇÃO POR DIVISÃO E CONQUISTA.

A chamada inicial do algoritmo é **solve(0, n − 1, 0, m − 1)**, assumindo que a implementação utiliza índices iniciando em zero.

Para análise de complexidade, seja $T(N, M)$ a função de complexidade de tempo desse algoritmo. No pior caso

$$T(N, M) = 2 \cdot T\left(\tfrac{N}{2}, \tfrac{M}{2}\right) + \mathcal{O}(M) \text{ que é } \mathcal{O}((N + M)\log(N)).$$

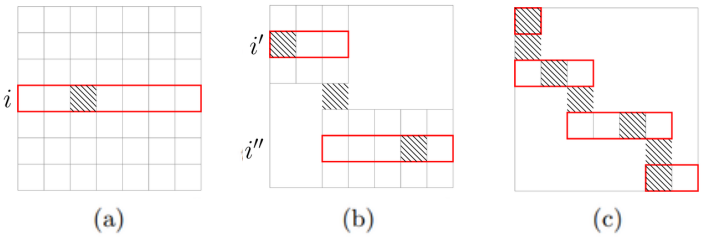

FIGURA 1: EXECUÇÃO DO ALGORITMO. A CADA NÍVEL DA CHAMADA DA FUNÇÃO, AS CÉLULAS EM VERMELHO SÃO PERCORRIDAS LINEARMENTE E OS MÍNIMOS SÃO HACHURADOS. FIGURA ADAPTADA DE MOLERO [7].

3. Problema de exemplo

Considere o seguinte problema de exemplo, adaptado do problema *Kimchi* (BOJ 11001) [4].

Problema 1: Dados dois vetores T e V, ambos de tamanho N, tais que $T_i \geq T_{i+1}$, e uma função $C(i, j) = (j - i) \cdot T_j + V_i$. Encontre o valor de $\min \{ C(i, j) \mid 1 \leq i, j \leq N \}$. Considere que $0 \leq T_i, V_i \leq 10^9$, $0 \leq N \leq 10^5$, e que todos os valores são inteiros.

Uma solução ingênua que verifica todos os pares (i, j), tem complexidade $O(N^2)$, o que é inviável para esses limites.

Mostraremos que é possível aplicar a técnica apresentada anteriormente, resultando numa solução $O(N \log(N))$.

Primeiramente definimos

$$\text{opt}[i] = \min \{j \mid C(i, j) \leq C(i, j') \text{ e } 1 \leq j' \leq N\}$$

Assim, a resposta do problema será

$$\min \{C(i, \text{opt}[i]) \mid 1 \leq i \leq N\}$$

Note que se provarmos que $opt[i] \leq opt[i+1]$ basta utilizar a função C definida neste problema no Código 1 e o problema estará resolvido.

4. Demonstração

Para realizar essa demonstração utilizaremos a *Propriedade de Monge* (também conhecida como *Desigualdade Quadrangular*), enunciada a seguir.

Teorema 1 (Propriedade de Monge): Seja C uma função de custo e opt o vetor de índices de mínimos das linhas como definido anteriormente. Se, para quaisquer inteiros $a \leq b \leq c \leq d$, vale $C(a, d) + C(b, c) \leq C(a, c) + C(b, d)$, então $opt[i] \leq opt[i+1]$.

A demonstração pode ser encontrada em Molero [7]. Note que a volta nem sempre é verdadeira e nesses casos é preciso utilizar argumentos mais criativos para demonstrar a monotonicidade.

No nosso problema, basta verificar que $C(i, j) = (j - i)T_j + V_i$ satisfaz a propriedade de Monge. De fato,

$$\begin{aligned}
C(a,d) + C(b,c) &\leq C(a,c) + C(b,d) \\
(d-a)T_d + V_a + (c-b)T_c + V_b &\leq (c-a)T_c + V_a \\
&\quad + (d-b)T_d + V_b \\
(d-a)T_d + (c-b)T_c &\leq (c-a)T_c \\
&\quad + (d-b)T_d \\
0 &\leq (c-a-c+b)T_c \\
&\quad + (d-b-d+a)T_d \\
0 &\leq (b-a)T_c + (a-b)T_d \\
0 &\leq (b-a)(T_c - T_d).
\end{aligned}$$

Como $b \geq a$, temos $b - a \geq 0$. Além disso, como $c \leq d$ e o enunciado garante que $T_i \geq T_{i+1}$, segue que $T_c \geq T_d$. Logo, $(b - a)(T_c - T_d) \geq 0$, concluindo que C satisfaz a propriedade de Monge.

Observe ainda que a desigualdade quadrangular depende apenas da parte da função que envolve simultaneamente os dois índices. Assim, decompondo

$$C(i, j) = x(i, j) + y(j) + z(i)$$

os termos $y(j)$ e $z(i)$ são cancelados na desigualdade de Monge, como acontece com V_i nesse exemplo. Nesse problema, poderíamos decompor

$$C(i,j) = -iT_j + jT_j + V_i$$

e portanto, bastaria verificar que a função $x(i, j) = -iT_j$ satisfaz a Propriedade de Monge.

5. Programação dinâmica

Considere a seguinte recorrência de programação dinâmica.

Sejam N e K inteiros tais que $1 \le K \le N$ e seja $C(i, j)$, $1 \le i, j \le N$, uma função de custo. Definimos

$$E[k,i] = \begin{cases} \min\limits_{i \le j \le N} \{C(i,j) + E[k-1, j+1]\}, & \text{se } k > 1, \\ C(i,N), & \text{se } k = 1 \end{cases}$$

para $1 \le k \le K$ e $1 \le i \le N$.

Uma implementação direta dessa recorrência possui complexidade $O(KN^2)$, uma vez que, para cada estado (k, i), é necessário percorrer todos os possíveis j.

Suponha agora que a função $C(i, j)$ satisfaça a propriedade de Monge ou, mais genericamente, que o vetor de índices de mínimos das linhas seja monotônico. Nesse caso, a otimização por divisão e conquista pode ser aplicada independentemente para cada $k \ge 2$.

De fato, para um valor fixo de k, basta definir a função

$$C'(i,j) = C(i,j) + E[k-1, j+1]$$

e como o termo $E[k - 1, j + 1]$, para um k fixo, depende apenas de j, ele não interfere na propriedade de Monge nem na monotonicidade do vetor de índices de mínimos das linhas. Assim, para cada k ($2 \le k \le K$), podemos calcular

$$\min_{1 \le i,j \le N} C'(i,j)$$

em tempo $O(N \log(N))$ utilizando a otimização por divisão e conquista.

Consequentemente, toda a programação dinâmica é resolvida em $O(KN \log(N))$. Uma demonstração completa desse resultado também pode ser encontrada em Molero [7].

6. Problema de exemplo

Considere o seguinte problema de exemplo, adaptado do problema *Subarray Squares* (CSES 2086) [5].

Problema 2: Dado um vetor V de tamanho N e um inteiro K, particione V em exatamente K subvetores contíguos. O custo de um subvetor é definido como o quadrado da soma de seus elementos.

O objetivo é *minimizar* o custo total da partição. Considere que $1 \le V_i \le 10^5$, $1 \le K \le N \le 3000$, e que todos os valores são inteiros.

Seja $C(i, j) = (\sum_{x=i}^{j} V_x)^2$, a recorrência resolve o problema, isto porque podemos considerar $E[k, i]$:= menor custo de particionar o subvetor $V[i...N]$ em k partes. Assim, para solucionar o problema, basta verificar que $C(i, j)$ satisfaz a condição de monge e substituir no Código 1 com adição do termo da programação dinâmica.

7. Conclusão

Além dos exemplos apresentados neste artigo, alguns exemplos interessantes que utilizam essa otimização incluem *Money for Nothing (World Finals 2017)* [3] e sua variação *Analyzing Contracts* [1]. Também o problema *Yet Another Minimization Problem* [2] que combina essa técnica com o "Algoritmo de Mo", utilizado para calcular eficientemente a função de custo [6].

Referências

[1] Agustín Santiago Gutiérrez. Analyzing contracts. Disponível em: https://codeforces.com/gym/104736/problem/A, 2023. Acesso em: 06 jul. 2026.
[2] Codeforces. Yet another minimization problem. Disponível em: https://codeforces.com/contest/868/problem/F, 2017. Acesso em: 06 jul. 2026.
[3] ICPC World Finals 2017. Money for nothing. Disponível em: https://codeforces.com/gym/101471, 2017. Acesso em: 06 jul. 2026.
[4] Jaehyun Koo. Boj 11001. Disponível em: https://web.archive.org/web/20260419163141/https://www.acmicpc.net/problem/11001, 2015. Problema do Baekjoon Online Judge. Acesso em: 06 jul. 2026.
[5] Antti Laaksonen. Subarray squares. Disponível em: https://cses.fi/problemset/task/2086, 2026. Problema do CSES Problem Set. Acesso em: 06 jul. 2026.
[6] Mikhail Tikhomirov. Codeforces 438 analysis. Disponível em: https://codeforces.com/blog/entry/55046, 2017. Acesso em: 06 jul. 2026.
[7] Victor Sena Molero. Algoritmos em matrizes monótonas e monge convexas. Disponível em: https://linux.ime.usp.br/~victorsenam/mac0499/monografia/main.pdf, 2021. Acesso em: 06 jul. 2026.