UNIVERSIDADE ESTADUAL DO OESTE DO PARANÁ
UNIOESTE - CAMPUS DE FOZ DO IGUAÇU
CENTRO DE ENGENHARIAS E CIÊNCIAS EXATAS
CURSO DE CIÊNCIA DA COMPUTAÇÃO

Implementação de Algoritmos para Multiplicação de Matrizes e Mochila Binária

Juliet Rigoti e Matheus Rezende
Professor: Rómulo Silva

5 de fevereiro de 2024

1 Introdução

No âmbito da disciplina de Projeto e Análise de Algoritmos, este trabalho apresenta a implementação de dois algoritmos para multiplicação de matrizes: o tradicional e o de Strassen. Além disso, foi implementado o algoritmo da Mochila Binária(*0-1 KnapSack problem*).

O problema da Mochila Binária busca determinar quais itens devem ser incluídos em uma mochila de capacidade limitada para maximizar o valor total dos itens transportados. Cada item pode ser colocado na mochila completamente ou não, o que caracteriza o problema como binário.

A implementação dos algoritmos de multiplicação de matrizes foi realizada utilizando matrizes quadradas, ou seja, com o mesmo número de linhas e colunas.

O algoritmo tradicional de multiplicação de matrizes consiste em multiplicar cada linha da primeira matriz por cada coluna da segunda matriz e somar os produtos resultantes.

O algoritmo de Strassen utiliza a técnica de divisão e conquista para dividir a multiplicação de matrizes em subproblemas menores, reduzindo a quantidade de operações matemáticas necessárias.

2 Algoritmos de Multiplicação de Matrizes

O algoritmo tradicional possui três laços alinhados, um laço para a linha, outro para coluna e o outro para o percorrimento da linha e coluna, logo é uma complexidade cúbica de $\Theta(n^3)$.

Nesse algoritmo foram feitos 3 testes, sendo eles os seguintes: teste de 16x16 dá 4 segundos, teste de 64x64 dá 4 minutos e 22 segundos, teste de 128x128 dá aproxidamente 35 minutos.

Figura 1: Os três laços do algoritmo tradicional

```
for(int linha=0; linha < ordem; linha++)
    for(int coluna=0; coluna < ordem; coluna++){
        somaprod=0;
        for(i=0; i<ordem; i++) somaprod+=mat[linha][i]*mat2[i][coluna];
        matRes[linha][coluna]=somaprod;
    }
```

Fonte: De autoria própria.

De acordo (KRAUSE; MORO; SCHNORR,) algoritmo de Strassen utiliza a abordagem de divisão e conquista para realizar a multiplicação de matrizes. São divididas em quatro submatrizes do mesmo tamanho e sete novas matrizes temporárias como visto na 2.

Figura 2: As setes submatrizes do algoritmo de Strassen

```
int** P1 = strassenMultiply(A11, subtract(B12, B22, k), k);
int** P2 = strassenMultiply(add(A11, A12, k), B22, k);
int** P3 = strassenMultiply(add(A21, A22, k), B11, k);
int** P4 = strassenMultiply(A22, subtract(B21, B11, k), k);
int** P5 = strassenMultiply(add(A11, A22, k), add(B11, B22, k), k);
int** P6 = strassenMultiply(subtract(A12, A22, k), add(B21, B22, k), k);
int** P7 = strassenMultiply(subtract(A11, A21, k), add(B11, B12, k), k);
```

Fonte: De autoria própria.

A partir disso essas matrizes temporárias são então somadas como mostra 2. O código então combina as submatrizes P1 a P7 para calcular o produto final de A11 e B12. Esta combinação envolve somar e subtrair os produtos das submatrizes de acordo com a fórmula do algoritmo de Strassen.

Figura 3: As quatro submatrizes do algoritmo de Strassen

```
int** C11 = subtract(add(add(P5, P4, k), P6, k), P2, k);
int** C12 = add(P1, P2, k);
int** C21 = add(P3, P4, k);
int** C22 = subtract(subtract(add(P5, P1, k), P3, k), P7, k);
```

Fonte: De autoria própria.

Criando novas matrizes, com a utilização da recursividade é gerado submatrizes para que sejam constituída de apenas um elemento. O código dentro das funções add e subtract calcula os elementos da submatriz resultante. Para cada elemento, o código realiza a soma ou subtração dos elementos correspondentes nas submatrizes P e suas variantes. Fazendo com que o algoritmo de Strassen tenha uma complexidade menor do que o método tradicional tendo complexidade de $O(\mathrm{n}^{2.807})$

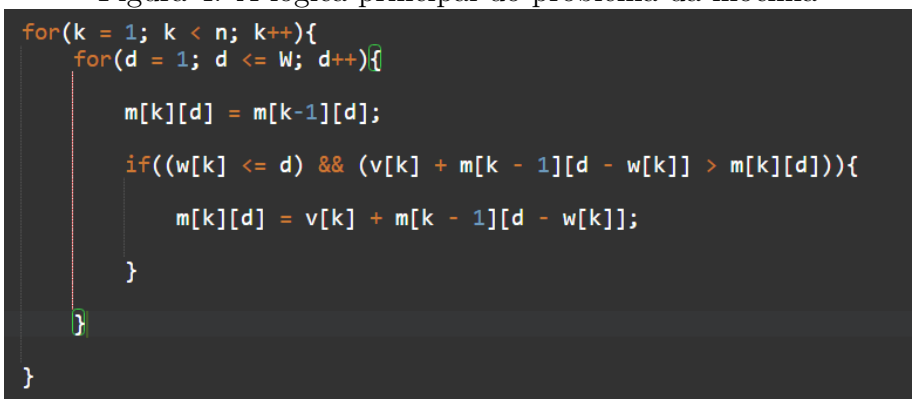
Nesse algoritmo foram feitos 4 testes: 16x16 com tempo de 2,4 segundos, 64x64 com 2 minutos, 128x128 com 13 minutos e 42 segundos e 256x256 levando 1 hora e 36 minutos.

3 Problema da Mochila Booleana

De acordo (MARQUES; ARENALES, 2002) o objetivo é escolher os objetos que maximizem o valor total sem ultrapassar a capacidade de peso da mochila.

Composto por dois vetores um sendo valor de cada item e outro o peso de cada item e um inteiro sendo a capacidade da mochila.

Figura 4: A lógica principal do problema da mochila

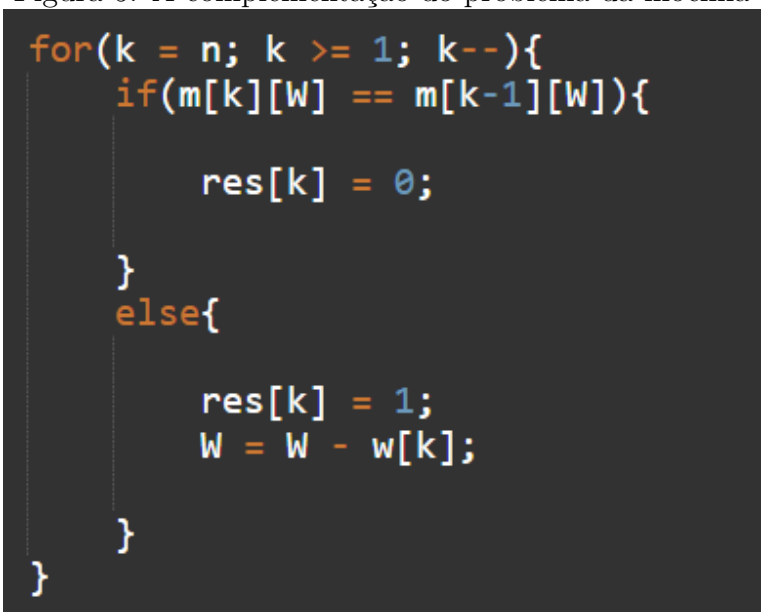


```
for(k = 1; k < n; k++){
    for(d = 1; d <= W; d++){

        m[k][d] = m[k-1][d];

        if((w[k] <= d) && (v[k] + m[k - 1][d - w[k]] > m[k][d])){

            m[k][d] = v[k] + m[k - 1][d - w[k]];

        }

    }

}
```

Fonte: De autoria própria.

A ideia é descobrir o valor total seja máximo, como mostrado 4. O código na imagem é uma implementação básica do algoritmo de programação dinâmica para o problema da mochila booleana que possui complexidade de $\Theta(nW)$.

Figura 5: A complementação do problema da mochila



```
for(k = n; k >= 1; k--){
    if(m[k][W] == m[k-1][W]){

        res[k] = 0;

    }
    else{

        res[k] = 1;
        W = W - w[k];

    }
}
```

Fonte: De autoria própria.

Para determinar os itens que maximizam o valor total, compara-se m[k, W] e m[k 1, W], caso o item k não esteja na solução, ele verifica se está no item k - 1. Quando diferentes, então o item k está na solução e então é confirmado para o vetor. No algoritmo em questão foram feitos 3 testes: capacidade 10 com 4 itens

levando 40ms, capacidade 900 com 200 itens resultando em 3 minutos e capacidade 1200 com 500 itens terminando em 10 minutos.

4 Referências

KRAUSE, A. M.; MORO, G. B.; SCHNORR, L. M. Análise de desempenho da multiplicação de matrizes por strassen contra o método tradicional. Citado na página 2.

MARQUES, F. d. P.; ARENALES, M. N. O problema da mochila compartimentada e aplicações. *Pesquisa Operacional*, SciELO Brasil, v. 22, p. 285–304, 2002. Citado na página 3.