# 1 Pendahuluan

# 2 Dekompisi secare rekursif

## 2.1 pendefinisian operasi dasar

# 3 Contoh DnC

## 3. Binary Search

## 3.1 karatsuba

# Perkalian Matriks denga Devide and Conquer

Misalkan kita punya dua matriks A dan B dengan ukuran n x n. proses pembagian dilakukan dengan menjadi 4 sub matriks dengan ukuran n/2 x n/2. Untuk kemudahan kita andaikan dengan k adalah bilangan bulat positif. Maka kita dapat menuliskan perkalian matriks A dan B sebagai berikut:

Dengan begitu kita punya matriks yang dapat dituliskan sebagai berikut:

Dengan:

Perhatikan bahwa kita perlu melakukan 8 kali perkalian matriks untuk mendapatkan matriks C. Dengan menggunakan algoritma DnC. Dengan begitu persamaan rekursif runnig time dari algoritma perkalian matriks adalah sebagai berikut:

Perhatikan juga bahwa kompleksitas dari melakukan conquer adalah karena algoritma ini bersifat inplace. Dengan mensubtitusi maka kita punya persamaan rekurensinya sebagai berikut:

Dan mudah di tunjukkan bahwa perkalian matris memiliki order of growth

def matmul\_dnc(  
 A: np.ndarray,  
 B: np.ndarray,  
 C: np.ndarray,  
 n: int = None,  
) -> np.ndarray:  
 """  
 matmul dnc on nxn matrix it's O(n^3)  
 """  
 if n == 1:  
 C += A \* B  
 return  
 n\_init = n  
 # if n not power of 2:  
 if not isPowerOfTwo(n):  
 n = max(A.shape[0], A.shape[1], B.shape[0], B.shape[1])  
 n = 2 \*\* (n - 1).bit\_length()  
 A = np.pad(A, ((0, n - A.shape[0]), (0, n - A.shape[1])))  
  
 B = np.pad(B, ((0, n - B.shape[0]), (0, n - B.shape[1])))  
 C = np.pad(C, ((0, n - C.shape[0]), (0, n - C.shape[1])))  
  
 # conquer  
 A\_11 = A[: n // 2, : n // 2]  
 A\_12 = A[: n // 2, n // 2 :]  
 A\_21 = A[n // 2 :, : n // 2]  
 A\_22 = A[n // 2 :, n // 2 :]  
 B\_11 = B[: n // 2, : n // 2]  
 B\_12 = B[: n // 2, n // 2 :]  
 B\_21 = B[n // 2 :, : n // 2]  
 B\_22 = B[n // 2 :, n // 2 :]  
 C\_11 = C[: n // 2, : n // 2]  
 C\_12 = C[: n // 2, n // 2 :]  
 C\_21 = C[n // 2 :, : n // 2]  
 C\_22 = C[n // 2 :, n // 2 :]  
  
 matmul\_dnc(A\_11, B\_11, C\_11, n // 2)  
 matmul\_dnc(A\_12, B\_21, C\_11, n // 2)  
 matmul\_dnc(A\_11, B\_12, C\_12, n // 2)  
 matmul\_dnc(A\_12, B\_22, C\_12, n // 2)  
 matmul\_dnc(A\_21, B\_11, C\_21, n // 2)  
 matmul\_dnc(A\_22, B\_21, C\_21, n // 2)  
 matmul\_dnc(A\_21, B\_12, C\_22, n // 2)  
 matmul\_dnc(A\_22, B\_22, C\_22, n // 2)  
  
 C = np.vstack((np.hstack((C\_11, C\_12)), np.hstack((C\_21, C\_22))))  
  
 if not isPowerOfTwo(n\_init):  
 C = C[:n\_init, :n\_init]  
 return C  
  
 return C

## 3.2 Strassen

Berdasarkan rangkuman materi brute-force, kita tahu bahwa perkalian matriks memiliki order of growth . Namun pada tahun 1969, Volker Strassen menemukan algoritma yang dapat mengurangi order of growth menjadi . Algoritma ini menggunakan 7 kali perkalian matriks untuk mendapatkan matriks C.

Strassen bisa mengurangi running time dari jumlah perkalian submatriks yang dibutuhkan menjadi 7, dengan sedikit trade-off pada saat melakukan conquer, yang menjadi .

Jadi, persamaan rekursif dari algoritma Strassen adalah sebagai berikut:

Dengan master therorem karena maka kita punya kompleksitas waktunya .

Ide dari penemuan algoritma strassen didasari pada fakta berikut:

Perhatikan bahwa kita hanya membutuhkan 1 kali perkalian dan 2 kali operasi penjumlahan atau pengurangan untuk mendapatkan pada form ruas kanan dibandingkan dengan 2 kali perkalian pada form ruas kiri.

Kita punya algoritma Devide and Conquer strassen sebagai berikut:

1. Jika , matriks masing-masing hanya berisi satu elemen. Lakukan perkalian skalar tunggal dan penjumlahan skalar tunggal, yang membutuhkan waktu . Jika tidak, partisi matriks input dan dan matriks output menjadi submatriks , seperti pada persamaan algoritma perkalian matriks dengan devide and conquer. Langkah ini membutuhkan waktu .
2. Buat matriks , masing-masing adalah jumlah atau selisih dari dua submatriks dari langkah 1. Buat dan nolkan entri dari tujuh matriks untuk menampung tujuh produk matriks . Semua 17 matriks dapat dibuat, dan diinisialisasi, dalam waktu .
3. Dengan menggunakan submatriks dari langkah 1 dan matriks yang dibuat pada langkah 2, secara rekursif hitung masing-masing dari tujuh produk matriks , yang membutuhkan waktu .
4. Perbarui empat submatriks dari matriks hasil dengan menambahkan atau mengurangi berbagai matriks , yang membutuhkan waktu .

Berikut adalah matriks antara yang perlu dihitung untuk menghitung matriks :

$$

$$ perhatikan bahwa Penjumlahan matriks di atas membutuhkan waktu. Dengan menggunakan matriks , kita dapat menghitung tujuh produk matriks yang dibutuhkan untuk menghitung matriks :

Dengan begitu kita punya