

数据结构第二次上机实验报告

林子开

2023 年 9 月 16 日

目录

1 Ordinary algorithm 的时间复杂度	1
1.1 理论分析	1
1.2 实验结果	2
1.3 小结	2
2 Strassen's algorithm 的时间复杂度	2
2.1 理论分析	2
2.2 实验结果	4
2.3 小结	5
3 附录 A: ordinary algorithm 完整源代码	5
4 附录 B: Strassen's algorithm 完整源代码	6

1 Ordinary algorithm 的时间复杂度

1.1 理论分析

Ordinary 算法的核心部分如下：

Listing 1: ordinary algorithm 核心部分

```
1 def ord_matrix_multiplication(A,B):
2     """只考虑两个矩阵均为方阵的情况"""
3     n = np.size(A,1)
4     C = np.zeros((n,n))
5     for i in range(n):
6         for j in range(n):
7             for k in range(n):
8                 C[i,j] = C[i,j] + A[i,k] * B[k,j] # 生成C中每个元素需要大约n次乘法和n次加法
9     return C
```

为便于讨论，只考虑矩阵 A 和矩阵 B 均为维度大小为 n 的方阵的情况。 $C = AB$ ，在计算 C 中的每个元素 C_{ij} 时，需要计算 n 次乘法和 n 次加法，复杂度为 $\Theta(n)$ ，由于 C 中一共有 n^2 个元素，则 Ordinary algorithm 算法的时间复杂度为 $\Theta(n^3)$ 。

1.2 实验结果

分别对矩阵大小 $n = 2^3, 2^4, 2^5, 2^6, 2^7, 2^8$ 的情况进行测试，实验结果如表1所示

表 1: Ordinary Algorithm 的实验结果

矩阵维度 n	2^3	2^4	2^5	2^6	2^7	2^8
time/s	0.000167	0.002	0.016504	0.126088	1.028406	7.999888
$\log_2(\text{time})$	-12.5502	-8.96548	-5.92106	-2.98749	0.04041	2.99998

做出“运行时间-矩阵大小”的双对数图如下



图 1: ordinary algorithm 的时间复杂度双对数

并对 $\log_2(\text{time})$ 与 $\log_2(n)$ 用 R 语言做线性回归（即最小二乘法），得到拟合直线的斜率，得到斜率 $k = 3.077$ 。

1.3 小结

可以看出，实验结果与理论分析较为一致，ordinary algorithm 算法的时间复杂度确实为 $\Theta(n^3)$ 。

2 Strassen's algorithm 的时间复杂度

2.1 理论分析

为便于讨论，在这一部分，对于 $C = AB$ ，仍假设 A, B 均为方阵，大小为 2 的整数幂次，以便进行矩阵分块。

Strassen's algorithm 分为以下几个步骤：

1. 递归基础情况，时间复杂度 $\Theta(1)$

Listing 2: Strassen's algorithm 递归基础情况

```
1 def Strassen_matrix_multiplication(A,B):
2     """只考虑方阵，且矩阵维度为2的整数幂次的ing情况"""
3     n = np.size(A,1)
4     # 若大小为1则直接返回，这是递归的基础情况
5     if n == 1:
6         return np.multiply(A,B)
```

2. 矩阵分块，时间复杂度 $\Theta(1)$

Listing 3: Strassen's algorithm 矩阵分块

```
1     temp = int(n/2)
2     A11 = A[0:temp, 0:temp]
3     A12 = A[0:temp, temp:]
4     A21 = A[temp:, 0:temp]
5     A22 = A[temp:, temp:]
6     B11 = A[0:temp, 0:temp]
7     B12 = A[0:temp, temp:]
8     B21 = A[temp:, 0:temp]
9     B22 = A[temp:, temp:]
```

3. 做 10 个矩阵加减法，生成 $S_1 \sim S_{10}$ ，时间复杂度 $\Theta(n^2)$

Listing 4: Strassen's algorithm 矩阵加减法生成辅助矩阵 S1 到 S10

```
1     S1 = B12 - B22
2     S2 = A11 + A12
3     S3 = A21 + A22
4     S4 = B21 - B11
5     S5 = A11 + A22
6     S6 = B11 + B22
7     S7 = A12 - A22
8     S8 = B21 + B22
9     S9 = A11 - A21
10    S10 = B11 + B12
```

4. 做 7 个矩阵乘法，使用递归方法，生成 $P_1 \sim P_7$ ，时间复杂度 $7T(\frac{n}{2})$

Listing 5: Strassen's algorithm 使用递归方式计算矩阵乘法，生成辅助矩阵 P1 到 P7

```
1     P1 = Strassen_matrix_multiplication(A11,S1)
2     P2 = Strassen_matrix_multiplication(S2,B22)
3     P3 = Strassen_matrix_multiplication(S3,B11)
4     P4 = Strassen_matrix_multiplication(A22,S4)
5     P5 = Strassen_matrix_multiplication(S5,S6)
6     P6 = Strassen_matrix_multiplication(S7,S8)
7     P7 = Strassen_matrix_multiplication(S9,S10)
```

5. 做 8 个矩阵加减法，生成分块矩阵 $C_1 \sim C_4$ ，时间复杂度 $\Theta(n^2)$

Listing 6: Strassen's algorithm 矩阵加减法生成分块矩阵 C1 到 C4

```

1  C11 = P5 + P4 - P2 + P6
2  C12 = P1 + P2
3  C21 = P3 + P4
4  C22 = P5 + P1 - P3 - P7

```

6. 分块矩阵拼接, 时间复杂度 $\Theta(1)$

Listing 7: Strassen's algorithm 矩阵拼接

```

1  C1_ = np.hstack((C11, C12)) # 横向拼接
2  C2_ = np.hstack((C21, C22)) # 横向拼接
3  return np.vstack((C1_, C2_)) # 垂直拼接

```

由上述讨论可知, Strassen's algorithm 的时间复杂度满足递推公式

$$T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2)$$

由于 $n^2 = O(n^{\log_2 7 - \epsilon})$, $\epsilon = \log_2 7 - 2 \approx 0.807 > 0$, 根据 Master Method 可知,

$$T(n) = \Theta(n^{\log_2 7}) \approx \Theta(n^{2.80735})$$

2.2 实验结果

分别对矩阵大小 $n = 2^3, 2^4, 2^5, 2^6, 2^7, 2^8$ 的情况进行测试, 实验结果如表2所示

表 2: Strassen's Algorithm 的实验结果

矩阵维度 n	2^3	2^4	2^5	2^6	2^7	2^8
time/s	0.00183384	0.01380310	0.09567207	0.65380581	4.57863772	31.94628000
$\log_2(\text{time})$	-9.09091947	-6.17886381	-3.38575836	-0.61306589	2.19491842	4.99757604

做出“运行时间-矩阵大小”的双对数图如下

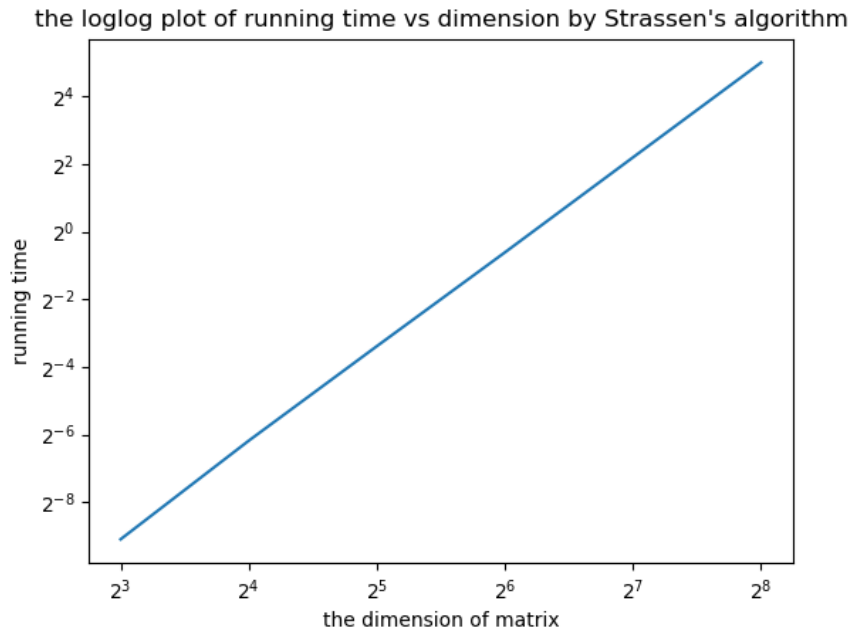


图 2: Strassen's algorithm 的时间复杂度双对数

并对 $\log_2(\text{time})$ 与 $\log_2(n)$ 用 R 语言做线性回归（即最小二乘法），得到拟合直线的斜率，得到斜率 $k = 2.80962$ 。

2.3 小结

可以看出，实验结果与理论分析较为一致，Strassen's algorithm 算法的时间复杂度确实为 $\Theta(n^{\log_2 7})$ 。

3 附录 A: ordinary algorithm 完整源代码

Listing 8: ordinary algorithm 完整 python 源码

```

1 import numpy as np
2 import time
3 import matplotlib.pyplot as plt
4
5
6
7 def ord_matrix_multiplication(A,B):
8     """只考虑两个矩阵均为方阵的情况"""
9     n = np.size(A,1)
10    C = np.zeros((n,n))
11    for i in range(n):
12        for j in range(n):
13            for k in range(n):
14                C[i,j] = C[i,j] + A[i,k] * B[k,j] # 生成C中每个元素需要大约n次乘法和n次加法
15    return C
16
17 def main():
18     time_store = []

```

```

19     start_power = 3
20     end_power = 9
21     np.random.seed(123)
22     for power in range(start_power, end_power):
23         n = 2**power # n为矩阵维度
24         A = np.random.randn(n, n)
25         B = np.random.randn(n, n)
26         start_time = time.time()
27         for _ in range(9-power): # 多次计算求平均值, 更准确, 但维数太大时, 少算几轮
28             ord_matrix_multiplication(A, B)
29         end_time = time.time()
30         print("完成一轮矩阵乘法!", power)
31         time_store.append((end_time - start_time)/(9-power))
32     print("耗时: ", time_store)
33     print("log2(耗时) :", np.log2(time_store))
34     # fig, ax = plt.subplots(figsize=(8,8))
35     plt.loglog(np.exp2(list(range(start_power, end_power))), time_store)
36     plt.xscale("log", base=2)
37     plt.yscale("log", base=2)
38     plt.xlabel("the dimension of matrix")
39     plt.ylabel("running time")
40     plt.title("the loglog plot of running time vs dimension by ordinary algorithm")
41     # plt.show()
42     plt.savefig("ord.png")
43     return
44
45 if __name__ == '__main__':
46     main()

```

4 附录 B: Strassen's algorithm 完整源代码

Listing 9: Strassen's algorithm 完整 python 源码

```

1 import numpy as np
2 import time
3 import matplotlib.pyplot as plt
4
5 def Strassen_matrix_multiplication(A, B):
6     """只考虑方阵, 且矩阵维度为2的整数幂次的ing情况"""
7     n = np.size(A, 1)
8     # 若大小为1则直接返回, 这是递归的基础情况
9     if n == 1:
10         return np.multiply(A, B)
11
12     # 首先进行分块
13     temp = int(n/2)
14     A11 = A[0:temp, 0:temp]
15     A12 = A[0:temp, temp:]
16     A21 = A[temp:, 0:temp]
17     A22 = A[temp:, temp:]
18     B11 = A[0:temp, 0:temp]

```

```

19     B12 = A[0:temp, temp:]
20     B21 = A[temp:, 0:temp]
21     B22 = A[temp:, temp:]
22
23     # 然后做十个矩阵加法或减法，生成中间的辅助矩阵S1-S10
24     S1 = B12 - B22
25     S2 = A11 + A12
26     S3 = A21 + A22
27     S4 = B21 - B11
28     S5 = A11 + A22
29     S6 = B11 + B22
30     S7 = A12 - A22
31     S8 = B21 + B22
32     S9 = A11 - A21
33     S10 = B11 + B12
34
35     # 再做7个矩阵乘法，使用递归算法，生成中间的辅助矩阵P1-P7
36     P1 = Strassen_matrix_multiplication(A11,S1)
37     P2 = Strassen_matrix_multiplication(S2,B22)
38     P3 = Strassen_matrix_multiplication(S3,B11)
39     P4 = Strassen_matrix_multiplication(A22,S4)
40     P5 = Strassen_matrix_multiplication(S5,S6)
41     P6 = Strassen_matrix_multiplication(S7,S8)
42     P7 = Strassen_matrix_multiplication(S9,S10)
43
44     # 生成分块矩阵C1-C4
45     C11 = P5 + P4 - P2 + P6
46     C12 = P1 + P2
47     C21 = P3 + P4
48     C22 = P5 + P1 - P3 - P7
49
50     C1_ = np.hstack((C11, C12)) # 横向拼接
51     C2_ = np.hstack((C21, C22)) # 横向拼接
52     return np.vstack((C1_, C2_)) # 垂直拼接
53
54
55 def main():
56     time_store = []
57     start_power = 3
58     end_power = 9
59     np.random.seed(123)
60     for power in range(start_power,end_power):
61         n = 2**power # n为矩阵维度
62         A = np.random.randn(n,n)
63         B = np.random.randn(n,n)
64         start_time = time.time()
65         for _ in range(9-power): # 多次计算求平均值，更准确，但维数太大时，少算几轮
66             Strassen_matrix_multiplication(A,B)
67         end_time = time.time()
68         print("完成一轮矩阵乘法!",power)
69         time_store.append((end_time - start_time)/(9-power))
70     print("耗时: ", time_store)

```

```

71     print("log2(耗时) :", np.log2(time_store))
72     # fig, ax = plt.subplots(figsize=(8,8))
73     plt.loglog(np.exp2( list( range(start_power,end_power)))), time_store)
74     plt.xscale("log", base=2)
75     plt.yscale("log", base=2)
76     plt.xlabel("the dimension of matrix")
77     plt.ylabel("running time")
78     plt.title("the loglog plot of running time vs dimension by Strassen's algorithm")
79     # plt.show()
80     plt.savefig("Strassen.png")
81     return
82
83 if __name__ == '__main__':
84     main()

```