实验三 所有点之间的最短路径

实验目的

- 本次实验为验证型实验
- 熟悉所有点对最短路径的基本概念
- 理解 Seidel 算法的基本过程及其性能

> **Seidel算法**
>
> 输入：图G(V,E)的邻接矩阵 A
> 输出：图G(V,E)的 APD 矩阵 D
>
> 1. $Z \leftarrow A^2$
> 2. 计算A'（$i \neq j$且($A_{ij}=1$或$Z_{ij}>0$)时，$A'_{ij}=1$)
> 3. 如果对所有$i \neq j$，$A'_{ij}=1$，返回$D=2A'-A$
> 4. 递归计算图G′ 的APD矩阵D′
> 5. $S \leftarrow AD'$
> 6. 返回矩阵$D_{ij}=\begin{cases}2D'_{ij}, & \text{如果} S_{ij} \geq D'_{ij}Z_{ii} \\ 2D'_{ij}-1, & \text{如果} S_{ij} < D'_{ij}Z_{ii}\end{cases}$ $O(MM(n)logn)$

实验内容

- 1.下载Seidel.py代码，阅读该代码并参照课件理解其过程（30 分）•代码解释
- 2.生成不同规模的**无向无权**图，运行Seidel算法，记录并分析不同规模图的时间性能（40 分）
- 3.对于（2）的结果，将Seidel.py中的矩阵乘法替换为Strassen.py中的strassen函数，并做同样的性能分析，比较两种矩阵乘法对算法性能的影响（30 分）

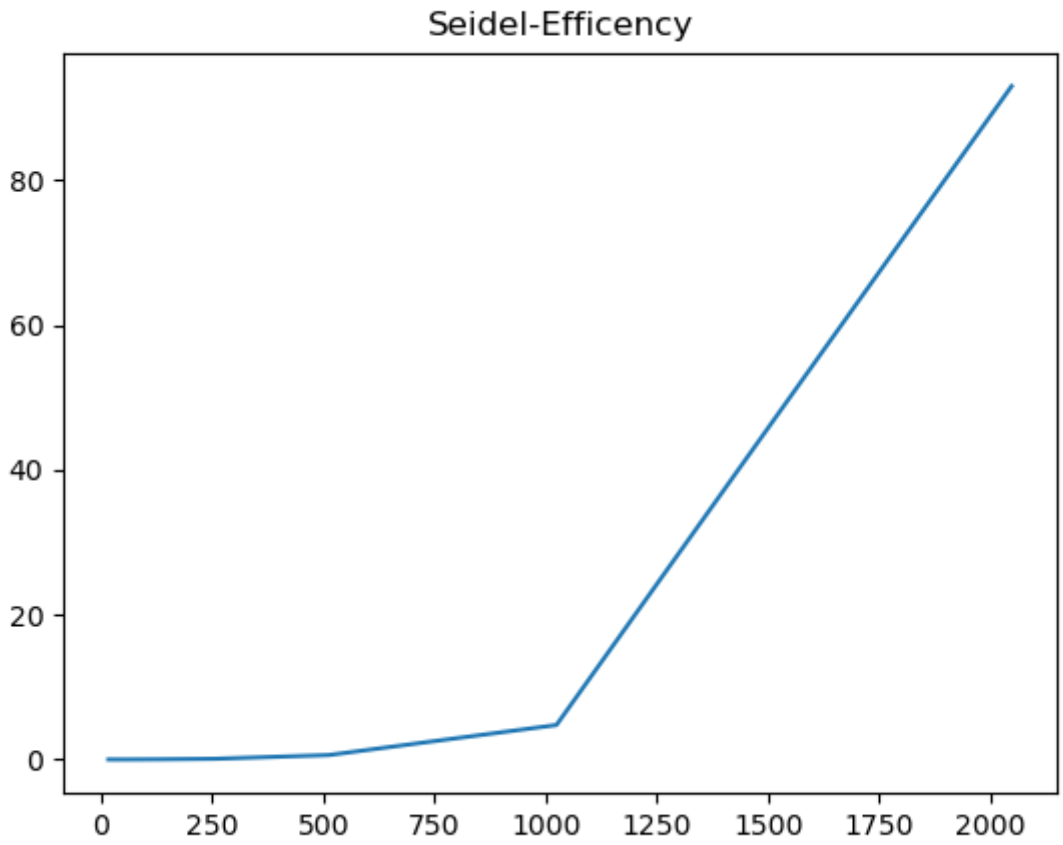
(1) Seidel算法代码

```
class Seidel:
    def get_mask(self, n):
        return np.ones(n, dtype=int) - np.eye(n, dtype=int)

    def get_distance_matrix(self, A):
        Z = np.matmul(A, A)
        B = np.zeros(A.shape, dtype=int)
        for i in range(B.shape[0]):
            for j in range(B.shape[1]):
                B[i][j] = 1 if ((i != j) and (A[i][j] == 1 or Z[i][j] > 0)) else 0

        if (np.array_equal(B - self.get_mask(B.shape[0]), np.zeros(B.shape))):
            D = 2*B - A
            return D
```

```
        T = self.get_distance_matrix(B)
        X = np.matmul(T, A)
        D = np.zeros(A.shape, dtype=int)
        for i in range(D.shape[0]):
            for j in range(D.shape[1]):
                D[i][j] = 2*T[i][j] if (X[i][j] >= T[i][j] * A[j].sum()) else 2*T[i][j] - 1
                # A[j].sum() - is degree of j verticle
        return D
```

(2)Seidel算法效率

通过模拟生成不同大小的图可以得到算法时间和输入规模的关系图：



(3)Strassen矩阵乘法优化

对于图规模比较小的时候我发现numpy自带的矩阵乘法出奇的快，而Strassen算法则效率一般，因为我又手动写了一个$O(n^3)$的矩阵乘法进行对比

```
请输入随机生成的图的大小:64
Seidel算法(手写矩阵乘法)耗时0.26871657371520996
优化矩阵乘法后的Seidel算法耗时:0.3563425540924072
Seidel算法(numpy自带矩阵乘法)耗时0.003908872604370117
```

在数据规模较小的时候，Strassen算法因为运行常数较大的原因并没有展示出其优势

```
请输入随机生成的图的大小:128
Seidel算法(手写矩阵乘法)耗时2.113999843597412
优化矩阵乘法后的Seidel算法耗时:2.6585118770599365
Seidel算法(numpy自带矩阵乘法)耗时0.019999980926513672
```

知道我把输入规模定为512时，此时我们可以发现Strassen算法的效率逐渐显示出了优势。

```
请输入随机生成的图的大小:512
Seidel算法(手写矩阵乘法)耗时150.62015438079834
优化矩阵乘法后的Seidel算法耗时:140.10071086883545
Seidel算法(numpy自带矩阵乘法)耗时0.7794747352600098
```

但是numpy自带的矩阵乘法仍然展示出了其惊人的效率，通过查阅资料发现原来是调用了C++的库...