实验2

Algorithm StayPoint_Detection(*P*, *distThreh*, *timeThreh*)

Input: A GPS log *P*, a distance threshold *distThreh*
and time span threshold *timeThreh*
Output: A set of stay points *SP*={*S*}

```
1. i=0, pointNum = |P|; //the number of GPS points
2. while i < pointNum do,
3.     j:=i+1; Token:=0;
4.     while j < pointNum do,
5.         dist:=Distance(pi, pj);  //calculate the distance between points
6.         if dist > distThreh then
7.             ΔT:=pj.T-pi.T;   //calculate the time span between two points
8.             if ΔT>timeThreh then
9.                 S.coord:=ComputMeanCoord({pk | i<=k<=j})
10                S.arvT:= pi.T;  S.levT=pj.T ;
11.               SP.insert(S);
12.               i:=j; Token:=1;
13.            break;
14.        j:=j+1;
15.    if Token!=1  then i:=i+1;
16. return SP.
```

Figure 2. Algorithm for stay point detection

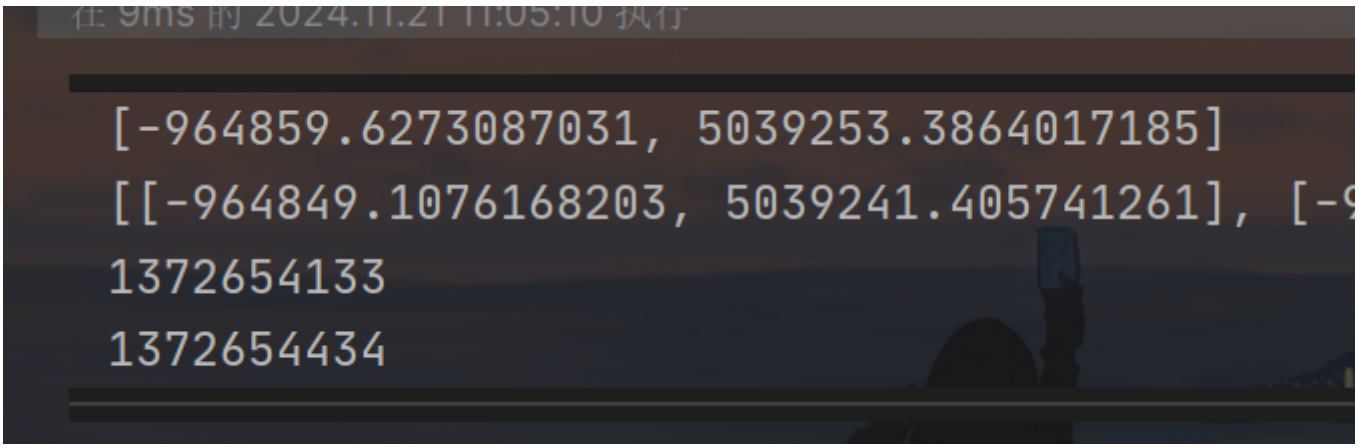
根据伪代码完成代码即可

调用util.Distance

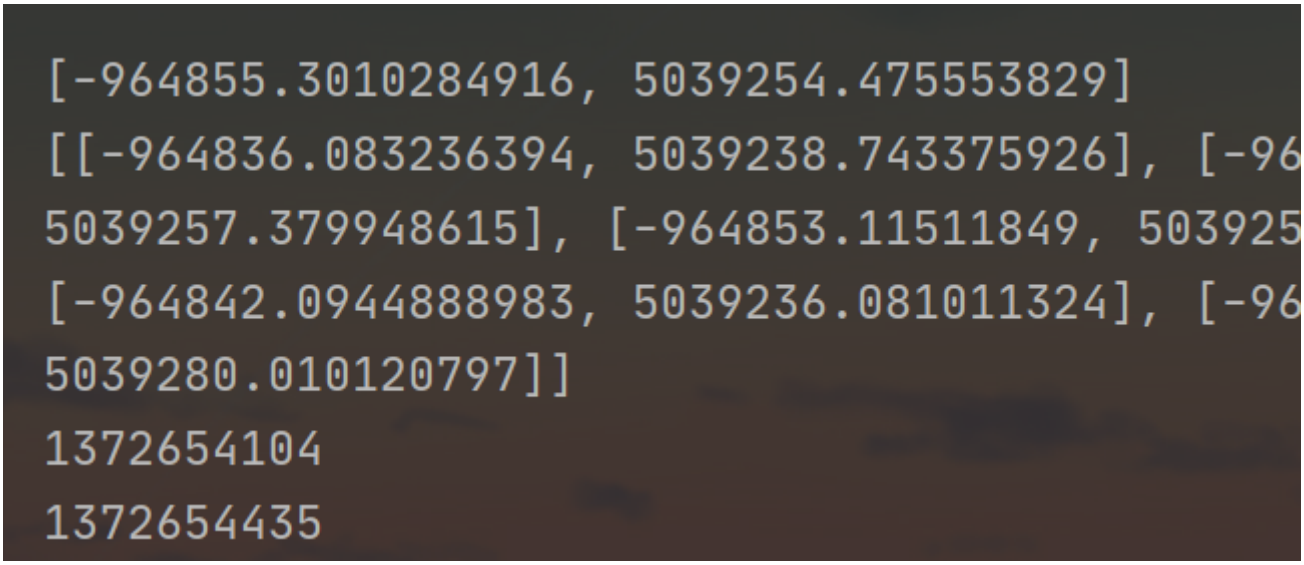
```python
def Distance(PointA, PointB, TypeC=0):
    # 精确的基于轨迹坐标的距离函数
    if TypeC == 0:
        latdis = fabs(PointA[1] - PointB[1]) * 111319.488
        lngdistemp = fabs(PointA[0] - PointB[0]) * 111319.488 * 0.5
        lngdis = lngdistemp * (cos(PointA[1] * pi/180.0) + cos(PointB[1] * pi/180.0))
        return sqrt(latdis ** 2 + lngdis ** 2)
    # haversine距离
    if TypeC == 1:
        return haversine(PointA, PointB, unit=Unit.METERS)
    # 墨卡托投影系下的欧式距离
    if TypeC == 2:
        pointa = wgs84_to_mercator(PointA[0], PointA[1])
        pointb = wgs84_to_mercator(PointB[0], PointB[1])
        return sqrt((pointa[0] - pointb[0])**2 + (pointa[1] - pointb[1])** 2)
```

这里如果选择typec=2，会导致已经转化过mercator坐标系再转化一次，选择typec=0，或者 `sqrt((traj[i][0] - traj[j][0])**2 + (traj[i][1] - traj[j][1])** 2)` 代码运行的效果是不一样的

typec=0



`sqrt((traj[i][0] - traj[j][0])**2 + (traj[i][1] - traj[j][1])** 2)`



最后选用了后一种，因为前一种的第一个停留点的停留点点数过少