避免超时

1．减少循环

算法一般给的空间是足够的，所有能通过增加空间来实现循环次数减少的时候，尽量就少用循环，避免超时。

2．减少函数调用

如果需要多次调用某个函数，则最好不要把这段功能写成函数。

3．减少重复计算

对于一些计算，可能后一个操作需要依靠前一个操作，或者在计算时就需要连贯的列表。

> 这种情况下，采用 后减前/后除以前 的操作会很好。

C++

```
# 高效方案
for i in range(n):
    op,k = map(float,input().split())
    if op == 1:
        op1.append(op1[i]*k)
        op2.append(op2[i])
    elif op == 2:
        op1.append(op1[i])
        op2.append(op2[i]+k)

for ni in range(m):
    data = input().split()
    i,j,x,y = int(data[0]),int(data[1]),float(data[2]),float(data[3])
    k=op1[j]/op1[i-1]
    theta = op2[j]-op2[i-1]
    x,y = x*k,y*k
    x,y = x*cos(theta)-y*sin(theta),x*sin(theta)+y*cos(theta)
    print(x,y)


# 低效方案
for i in range(m):
    i,j,x,y = map(int,input().split())
    for op in op_lst[i-1:j]:
        if op[0] == 1:
            x,y = op1(x,y,op[1])
        elif op[0] == 2:
            x,y = op2(x,y,op[1])
    print(x,y)
```

4. 减少空节点

一般建树会采用 list 存放子节点（下标作为父节点的 id），但是这样循环遍历的时候需要判断空节点，增加循环时间，所以建议使用字典来存放 child，一旦一个节点不存在 child，就不会出现在 dict 中，

Python

```
# 下标代表父节点，比如下面这个就是0号节点（root）的子节点是1和2号节点
# 可以看到有很多空节点，一旦需要遍历，就要使用if进行判断
#  循环次数多（重点是增加了循环次数）
child_list = [[1,2],[5,0],[3,4],[0,0],[0,0],[0,0]]


# 而使用dict就不存在这种问题，不仅不需要增加if进行判断，而且循环次数极少
child_dict = {0:[1,2],1:[5],2:[3,4]}

```

5. 过程（中间）变量的考量

一些在算法过程中生成的中间变量可能会导致时间的额外增加。

比如 3 个矩阵相乘：假设 n>m，两个 n*m 的矩阵 A 和 C，以及一个 m*n 的矩阵 B，现在要 A x B x C.

如果先进行 A x B，则会生成一个 n*n 的矩阵，此时再继续与 C 相乘，时间复杂度为 O(n*m*n) 以及 O(n*n*m)，

但如果先进行B x C，则会生成一个 m*m 的矩阵,时间复杂度为 O(m*n*m) 以及 O(n*m*m)

显然，优先生成较小矩阵的方式要快很多。