// 浮点二分

/\*\*\*\*浮点二分\*\*\*\*\*\*///////

const double eps = 1e-7;

double l, r;

while( r - l > eps)

{

double mid = (l+r)/2.0;

if(ok(mid))

{

l = mid;

ans = mid; /\*\*///\*/\*

}

else r = mid ;

}

#include<bits/stdc++.h>

using namespace std;

const double eps = 1e-8;

int n;

double p, a[100010], b[100010];

/\*

判断 n 台机器是否能坚持至少 t 秒：在 t 秒时间充电器能够充能量总值为 p\*t 。

枚举判断 n 台机器能否坚持 t 秒，对于能量不足的机器，

从 p\*t 的总值中扣除不足部分。

若最终 p\*t 仍有剩余，说明至少能坚持 t 秒，否则不能。

\*/

bool jug(double t)

{

double ext = p \* 1.0 \* t;

double sub;

for(int i=1;i<=n;i++)

{

sub = a[i] \* t - b[i];

if(sub > 0) // 消耗 大于 剩余

ext -= sub; //总产生能量得减去这些

if(ext < 0) return false; // 总能量小于0

}

return true;

}

int main()

{

double sum = 0;

scanf("%d %lf",&n,&p);

for(int i=1;i<=n;i++)

scanf("%lf %lf",&a[i],&b[i]), sum += a[i];

if((p - sum) >= -eps){ //当且仅当 Σai≤p 时，能够无限工作。

printf("-1\n");

return 0;

}

double l = 0, r = 1e12, mid, ans;

while((r-l) > eps)

//二分进行枚举能够同时工作 t 秒，再对 t 秒判断每台机器是否都能够坚持 t 秒钟。

{

mid = (l+r)/2;

if(jug(mid))

l = mid, ans = mid; //二分中重点

else

r = mid;

}

printf("%.8lf\n", ans);

}

/\*

针对这段代码可能出现在 32 位编译器中用下列样例将进入死循环的情况，可以考虑将 eps 改为 1e-6，或在 64 位编译器下运行。

原因： 32 位编译器下 double 的有效数字仅在 15 位左右。

Sample：

25000 49999

2 99999

2 99999

...

2 99999

正统做法：由于 double 的有效位数有限以及本题答案较大，在 1e-4 的相对精度下，限制二分次数 100 次左右可以保证达到精度要求且时间较短。